Comparing Reduction Strategies in Resource-Conscious Lambda Calculi

John Maraist

September 12, 1997

Acknowledgments

THIS thesis was written in January–August 1996 at the Institute for Program Structures and Data Organisation (IPD) of the University of Karlsruhe, Germany, based on work begun in October 1993. The thesis readers were Profs. Martin Odersky and Philip Wadler, and the thesis defense (*Promotion*) occurred in October 1996. The readers accepted the thesis with minor corrections, which I completed and submitted to the University in September 1997.

First and most of all, I would like to thank my advisors, Martin and Phil, for their introduction to this field of study and their encouragement of my work.

Thanks also to my parents, Marcel and Mickey Maraist, for their patient and continued support, and to my family in St. Martinville and Lafayette.

Sincere thanks as well to my collaborators in research over the three years in Karlsruhe, again Martin and Phil, and Zena Ariola, Matthias Felleisen, Frank Silbermann and Dave N. Turner. Many of the ideas in this thesis were shaped during trips to Glasgow, Scotland, and I am grateful to the computing department at the university, in particular Phil and Dave, for welcoming me for a few weeks every now and again. Our research group in Karlsruhe was also very supportive in matters both academic and linguistic, especially Stefan Bresch, Christian Kemper, Hendrik Lock and Martin Wehr. I am grateful to the Department of Computer Science at Tulane University for the use of their computer facilities during my time in Louisiana. Corrections and final revisions to this thesis after the defense were completed primarily at the School of Computer and Information Science, University of South Australia.

But most of all I want to thank the friends who helped keep me sane and happy: James Hartman, Vince Barras, Carole Leslie, Bill Thomason, Jeff Bordelon, Mary Erhart. Thanks also to, associated with New Orleans, Swedlow, Jennifer, Whitney, Todd & Ginger, John K., Ray, Matt C., John V., Kerry & Doug, the C.S. and 'TUL people, the Ouders and Uptons; associated with Glasgow, Andrew, Brian, Collin; associated with (if no longer all local to) Acadiana, Matt & Brenda, Beth & Matt, the Bordelons and Beyts; associated with Karlsruhe and its environs: Martin & Sabine, Ernst & Marianne, Paul & Astrid, Waclaw, Christian & Claudia, James, Holger, Lutz, Stefan, Rose & Matthius, Olaf & Irena, Oliver G., Oliver S., the Tichy's, Ladds, and Landry's.

Adelaide, South Australia, September 1997.

Typographical Notes

THIS document was written in the LaTeX2ε typesetting language on the XEmacs editor, versions 19.13 and 19.14, and compiled with the Dante TeX/LaTeX distributions — marvelous programs all. In addition to the standard *latexsym* fonts, I have made use of the American Mathematical Society's packages *amsfonts* and *amssymb*, Herman Zapf's Euler script fonts as implemented in the *euscript* package, and the St. Mary's Road font package *stmaryrd* of Jeremy Gibbons and Alan Jeffrey. For the ubiquitous annotated arrows and large diagrams I used the *diagmac* and *catmac* packages written by John Reynolds and extended by Martin Odersky and myself. The *prooftree* package of Paul Taylor facilitated the typing proof trees, and his *dropcap* package affects the first letter of every chapter. I wrote the *citebackref* package, which allows each bibliography entry to refer to the pages on which it is cited, especially for this thesis, and the package is free for the asking. For the overhanging of figures into the marginal space I have used a package *widefig* based on code written by Alex Cowie and on suggestions from the *comp.text.tex* newsgroup. I have made frequent (but probably not frequent enough) use of the *ispell* program and its XEmacs interface, regular use of my father's paperback copy of *The Merriam-Webster Dictionary*, third edition, 1974, and occasional use of the electronic version of Bill Walsh's *The Curmudgeon's Stylebook* (*http://www.theslot.com/contents.html*) and of Springer-Verlag's style manual for their LNCS series.

Contents

List of Figures

Abstract

In this thesis we use ideas from linear logic to study different parameter-passing protocols for the lambda calculus. We identify a particular linear lambda calculus, and show how it may be taken as the common target of two different translations from traditional lambda terms such that one translation corresponds to call-by-name reduction, and the other to call-by-value reduction. Call-by-need may included in this framework by translation into an affine variant of the linear calculus. We refine the characterization of the three protocols through the identification of a separated-linear lambda calculus, which decomposes the notion of linearity. The original two translations are easily adapted for the separated system, which also admits a hybrid translation taking call-by-need reduction into the new common target.

Some parts of this work contain results which have been stated previously in other recent research; here we have developed a new proof technique which allows a more systematic proof of these claims. Specifically, we develop a new marking technique for redexes involving explicit bindings which allows claims of finite developments with unique completions for calculi with these terms. The technique allows us to neatly prove confluence and standardization results for these systems, and to demonstrate various conservative extensions and observational equivalences. The standardization results also allow extension of the translation results for the standard reduction strategies.

Chapter 1
Introduction

PERHAPS the oldest mechanisms still in use today for writing well-structured programs are functions and procedures. By abstracting certain pieces of code away from specific locations and values, programmers have achieved more flexible, readable and easily maintained code. The written body of a function contains references to *formal* parameters, which are really just names. The formals are placeholders for information to be supplied when the procedure is actually invoked. Every different invocation must supply this information, called the *actual* parameters, or *arguments*. In every language the meaning of programs, or the *semantics*, as well as the run-time efficiency of programs, depend crucially on the mechanism by which a language definition specifies that formal and actual parameters are associated with each other, or *passed* from the caller to the function.

To contemporary programmers, the best-known parameter-passing mechanism is *call-by-value*. This protocol states that only simple scalar expressions, called *values*, and sometimes groups of values, may be passed as parameters. Values are a proper subset of the possible expressions: "2+3" is an expression and non-value which is equal to the value "5". Thus, if a program in a call-by-value environment contains an expression $f(2 + 3)$, we would not be allowed to call the function f until we had reduced "2+3" to "5". Call-by-value corresponds well to the low-level machine image of programs, and to the expressions which fit nicely within the confines of registers and stack locations. Most languages currently in use in industry, being either low-level themselves or pursuing other forms of abstraction, implement a call-by-value semantics.

Call-by-value is distinct from the *call-by-name* protocol used by Church in his framework for algorithmic computation, the lambda calculus. Call-by-name places no limits on what one might pass as a parameter. Whereas in call-by-value we must reduce $f(2 + 3)$ to $f(5)$ before calling f, in call-by-name we can call f with either $2 + 3$ or 5. Call-by-name is a copying semantics, in that a copy of the actual parameter as it is at the time of function invocation is substituted for each occurrence of the formal parameter. If f is defined as

$$f(x) = x \cdot x$$

then an invocation of f with $(2 + 3)$ results in the expression $(2 + 3) \cdot (2 + 3)$. Call-by-name is of interest for a number of reasons even if its direct implementation is somewhat extravagant. It is a particularly easy framework to analyze and manipulate; it produces very similar results, if not operational behavior, to the more sophisticated, and more complicated, notion of call-by-need; and it is used in Algol 60.

Call-by-need is a protocol where only values may be copied, but where any expression may be taken as the argument to a function. In call-by-need one finds the implementation of sharing among different uses of an actual parameter in each function call: when one occurrence of a formal requires evaluation of the actual parameter, all future accesses of the actual will in fact access the result of its first evaluation. In other words, recomputation of passed parameters is avoided. Call-by-need also uses the notion of *garbage collection*. Since we wait to evaluate an argument until it is needed, it may turn out that we never need it, and thus that it is never evaluated. In such a case, when no part of the program refers to an argument, the system is free to discard it at any time.

Call-by-need more accurately models the operational behavior of "lazy" functional languages such as Haskell. These lazy functional languages have often been taken as implementing call-by-name, but with informal runtime optimization to the handling of parameters. Such a view, which does not consider the real differences between by-name and by-need semantics, can lead to false optimizations which actually worsen program behavior. Hughes has presented an exposition of the merits of functional programming in general, and lazy languages in particular [35]; Wadler surveys uses of strictness and laziness from a more recent point of view, including a number of industrial applications [79].

The difference between call-by-name and call-by-value has been heavily studied over the past two decades, much of the work descending from Plotkin's article "Call-by-name, call-by-value and the λ-calculus" [62]. At a high level, the idea behind Plotkin's approach is to map different parameter-passing mechanisms into a single system, with a different mapping for each protocol. By constructing sufficiently robust maps, the difference between the protocols becomes clear. Plotkin gave transformations from call-by-name and call-by-value into the calculus of the *continuation passing style*, or CPS calculus. Terms in CPS have roughly the same syntax as in the lambda calculus, but rather than returning their results, functions take an extra parameter called the *continuation* which specifies a second function to which the result of the first should be passed. Each of Plotkin's transformations makes the control operations of each particular mechanism explicit in the transformed term, so that reduction of the transformed term by either mechanism produces equivalent results.

In this thesis, we will compare the different calling mechanisms not by the control flow which each suggests, but instead through the notion of *resource consumption*. The notion of a resource underlies, for example, the action of copying a term: viewing an argument as a resource made available for invoking a function, copying is an action which our regime for managing resources may or may not allow. Similarly, the discarding of terms which a program no longer needs is another issue of what we consider a valid use of resources. While call-by-name obviously places no limits whatsoever on resource use, call-by-value does impose a blanket restriction on duplication of non-values. Still, call-by-need is the only system of the three which could fairly be described as *resource-conscious*, since it deals with resource management actively, at a separate level from computation. Call-by-value manages resources by default, as a consequence of its restricted computation (parameter-passing) mechanism.

As in Plotkin's work with the CPS transforms, we will present transformations from call-by-name, call-by-value and call-by-need into new systems which will make the differences in their behaviors manifest, but rather than focusing on an explicit flow of control as with the CPS transforms, we will focus on resource usage via properties of the calculi's type systems. All three of the calculi discussed so far have a common straightforward type system, and as usual for typed lambda calculi, the system exhibits a correspondence in the style of the Curry-Howard isomorphism [21, 34]: a relationship between the type systems and a formal logical frameworks. Here, the related logic is minimal intuitionistic logic. A function f in the lambda calculi will have a *function type*: if f takes arguments of type A and returns values of type B, we write its type as $A \to B$. Function types are Curry-Howard-style correspondent to *implication* in intuitionistic logic, which we also write with the $\to$ connective, and which is also often written as $\supset$. Our use of the Curry-Howard correspondence throughout this work is superficial: we do not actually explore the relationships between term reduction and proof normalization in the various pairs of reduction and deduction systems, which in any event are not all isomorphisms. We simply exploit the intuition allowed by the Curry and Howard's techniques into how we might construct the target calculi for our transformations.

In intuitionistic logic, structural inference rules play an important, if often overlooked, role: weakening allows assumptions to be discarded, while contraction allows duplication. Moreover, these rules are the *only* facility for duplicating and discarding assumptions. So at the level of the calculi, the corresponding typing rules are the mechanisms by which we introduce the copying or discarding of terms. A term is discarded when it is substituted for a variable which appears only on the left side of a typing judgment, which can occur only as the result of applying a weakening rule; a term is duplicated when it is substituted for a variable which appears more than once on the right side of a typing judgment, which can occur only by a contraction rule.

The transformations which we will consider will map the systems based on intuitionistic logic to systems based on Girard's *linear* logic [26]. In linear logic the authorization to weaken or contract is expressed in a formula by a new *modal* connective !, pronounced variously *of course*, *bang*, *shriek* and *pling*. Linear implication requires the precedent to be used exactly once in the proof of the consequent, and is written with a new connective $\multimap$, pronounced *lollipop* or *lolli*. The intuitionistic subset of linear logic is of particular interest to us here, so henceforward when we write "linear logic" we will mean this fragment, and will refer to the entire system of Girard as "full linear logic." The *linear lambda calculus* has a type system based on linear logic, and it is this system and variations on it which will serve as targets of our transformations.

Girard described two translations of intuitionistic logic into linear logic. The translations center around the question of the appropriate image in linear logic for $A \to B$: under his *standard* translation $\circ$, we map intuitionistic implication to $(!A^\circ) \multimap (B^\circ)$. That is, the image of a function may be used only once, but in being used may use its argument many times, or not at all. Moreover, the result of the function will not necessarily be marked with the modal connective, so it may be used only once. Under the *steadfast* translation $*$ we map $A \to B$ to $!(A^* \multimap B^*)$. Now the image of a function can be used many times, but for each individual use, the argument and result will be neither copyable nor discardable.

From Girard's introduction of linear logic in 1987 through around 1994, it seems to have been fairly widely known that the standard translation could be extended to terms with a call-by-name semantics, but the correspondence of the steadfast translation to call-by-value seems to have been less clear. We presented the extensions of the translations to terms and reduction sequences in work with Odersky, Turner and Wadler [51, 52]; similar results were independently presented by Mackie [47]. The treatment of call-by-need, however, is more difficult.

Call-by-need cannot be straightforwardly mapped into the linear lambda calculus. The standard transformation is clearly inappropriate for call-by-need, since $\circ$ allows non-values to be copied. The steadfast transformation is also inappropriate: call-by-need allows any term to be discarded once it is clear that the term will not be used, but in the steadfast transformation, only values are marked as discardable. Call-by-name and call-by-value calculi translate well into the linear calculus in part because the opportunities for discarding and duplication in these systems are coincident. In call-by-name, discarding and duplication is allowed for — but only for — any function argument; in call-by-value, for any value. Call-by-need, on the other hand, allows any function argument to be discarded, and any value to be duplicated.

For a first treatment of call-by-need, we use *affine* logic, rather then linear logic. Affine logic relaxes the linear restriction of the use of weakening, but retains the required modality for contraction. Since weakening is no longer restricted, the steadfast translation can now correspond to the behavior of call-by-need, and in fact the mapping by $*$ from call-by-name into the *affine lambda calculus* does preserve reduction sequences. However, the mapping does not meet all of the quality criteria for such transformations, and more importantly, does not allow a direct comparison with call-by-name and call-by-value, since it uses a different target.

The observation that call-by-need does not admit either Girard translation because its uses of weakening and contraction do not coincide suggests another method for dealing with call-by-need: rather than a linear lambda calculus where a single mode enables both weakening and contraction, we ought to allow each operation to be enabled separately. Jacobs has considered mathematical models for a variant of full linear logic with such separate control, and found that the idea is in fact well-grounded from a model-theoretic point of view [37]. Based on his earlier work on models for the substructural lambda calculi [36], Jacobs identified a category of models for a variant of linear logic with separate exponential connectives for each of weakening and contraction which generalizes affine and relevance logic as special cases [37]. Jacobs limited the scope of his study to the model theory, giving only a brief sketch of a possible syntax for the logic and not considering its relation to intuitionistic systems.

We present a *separated-linear* lambda calculus with separate connectives $!^w$ and $!^c$ for weakening and contraction, respectively. It would be fair to characterize our system as corresponding to (a minor variant of) the minimal intuitionistic fragment of Jacobs' system. It is easy to simply divide the $!$ operator into the composition of $!^w$ and $!^c$ in the definitions of $\circ$ and $*$, producing separated versions of the Girard translations, and it comes as no surprise to discover that these transformations have the same properties as their non-separated sources. What is more surprising about the separated system is that it allows a superior treatment of call-by-need. Since we can more accurately reflect duplication and discarding in call-by-need with the separate modes, we can build a *hybrid* translation from call-by-need into the separated-linear calculus. This hybrid satisfies all of the criteria we set for assessing the transformations, and finally allows a comparison of all three of the call-by-name, call-by-value and call-by-need systems in the same framework.

1.1 Background

Over the next few pages we review some of the ideas which give our work both foundation and motivation. From the notion of parameter-passing, our point of departure into more formal systems is the basic syntax of the lambda calculus, and some elementary operations on those terms, in Section 1.1.1. We consider the types which we give terms in Section 1.1.2. Corresponding to types are logical formulas, and we review at the same time the proof systems to which our calculi will correspond. Finally in Section 1.1.3 we focus on type inference, and moreover on the notion of inferring other properties of programs.

1.1.1 Terms of the core lambda calculus

Church's lambda calculus provides the syntax for abstracting parameter-passing away from other language features. The core of this system, which is common to all of the calculi we will consider, contains three forms for terms. The simplest term is just the occurrence of a variable. We let x, y, z range over variables in our presentation of terms. Functions are described using *abstractions*. A term $(\lambda x.\, M)$ is a function with formal parameter x and body M. Since functions are not given names, it is the functions themselves which are applied to arguments, and not a name assigned to represent some subroutine presented elsewhere. Functions access their arguments in *applications*, where we juxtapose a function M to the left of its argument N: $(M\ N)$. These variables, abstractions and applications form the core of our calculus, and most of our presentation centers around these three constructs. One can also consider two basic varieties of data type. We have first the *pair*, or *product* expression, of two other terms (M, N) and the related *projection* operators which return one or the other of the two components, $\pi_1\, L_1$ and $\pi_2\, L_2$. We also have a *coproduct*, or *sum* expression, like the algebraic data types of Haskell or ML. We consider two constructors inl and inr, each taking a single subterm. These tagged expressions are consumed by the case construct,

$$\mathrm{case}\ L\ \mathrm{in}\ \mathsf{inl}\, x : M,\ \mathsf{inr}\, y : N \quad .$$

We expect L to be or to become a sum expression. If it is of the form $\mathsf{inl}\, L_0$, then we bind x to L_0 and return M; if it is of the form $\mathsf{inr}\, L_1$, we bind y to L_1 and return N. In this thesis, our central presentation will not consider product and sum expressions for the sake of keeping the presentation simple, although we discuss the extension of our results from the core to the larger term language in the conclusion.

We distinguish between two kinds of variables, *free* and *bound*. A bound variable is just a formal parameter. The abstraction $(\lambda x.\, M)$ *binds* x. In any expression that contains $(\lambda x.\, M)$ as a subterm, we say that x *occurs bound*. By contrast, we say that any variable which is not enclosed under a lambda prefix for that variable *occurs free*. For example, in the term

$$(\lambda y.\, y\ z)\ (x) \quad ,$$

we have x and z occurring free, and y occurring bound. In many presentations of the lambda calculus, especially older ones, much is made of naming and renaming bound variables. We resolve these issues by taking all terms formed by renaming the formal parameter to be equivalent: so for example we make no distinction between the terms $(\lambda x.\, x)$, $(\lambda y.\, y)$ and $(\lambda z.\, z)$, which we write as

$$(\lambda x.\, x) \equiv (\lambda y.\, y) \quad .$$

It is possible that a particular name may be used as both a bound and a free variable in the same term, such as x in

$$M \equiv (\lambda x.\, \lambda y.\, x)\ y \quad ,$$

but such usage can lead to unintended confusion. Contracting this M in any classical sense, we have $M \to (\lambda y.\, y)$, which clearly runs against all other uses of the abstraction $(\lambda x.\, \lambda y.\, x)$:

$$\begin{array}{rcl}(\lambda x.\, \lambda y.\, x)\, 48 & \to & \lambda y.\, 48 \\ (\lambda x.\, \lambda y.\, x)\, ((\lambda z.\, z) w) & \to & \lambda y.\, ((\lambda z.\, z) w)\end{array}$$

and so forth. The function $(\lambda x.\, \lambda y.\, x)$ is clearly intended to take two arguments, returning the first and discarding the second, but the *name capture* effect has distorted the appropriate behavior of the term when the argument is the free variable y. In M we have that y is a free variable, but that in $(\lambda y.\, y)$ it only occurs free. We say that the prefix λx has *captured* the free use of y, and changed its intended role as a placeholder for some surrounding term. We remove this confusion by following the *hygiene* convention of Barendregt: in every subterm all bound variables must be renamed such that none have the same name as any free variable.

We use two mechanisms for manipulating the syntax of terms. *Substitution* allows us to replace free variable occurrences with another term. We define substitution to be a metalevel, implicit mechanism where a substitution applied to a term is indistinguishable from the result of a substitution, as opposed to a separate, formal operational manipulation. Specifically, we define substitution as follows:

$$\begin{array}{rcl} x[x := L] & \equiv & L \\ y[x := L] & \equiv & y,\ x \not\equiv y \\ (M\ N)[x := L] & \equiv & M[x := L]\, N[x := L] \\ (\lambda y.\, M)[x := L] & \equiv & \lambda y.\, (M[x := L]) \end{array}$$

The hygiene condition has allowed us to omit some technical cases from this definition. Since we are defining substitution for free variables, and since a bound and a free variable will have different names, in the clause for abstraction we need not specify that x and y be different. Note that by the hygiene condition we have that the free variables of N in an expression $M[x := N]$ are in fact free in $M[x := N]$: the substitution metanotation cannot capture variables.

A second mechanism which we use for dissecting the syntax of terms is the idea of a *context*. A context is simply a term with a *hole* instead of one subterm. We let C range over contexts. In classical lambda terms, the grammar for contexts is then:

$$C ::= [\,] \mid M\ C \mid C\ M \mid \lambda x.\, C \ \ .$$

We do not need multiple holes in contexts, and do not develop the additional overhead required for such a mechanism. We use contexts by *filling* the hole. Given a context C, the expression $C[M]$ is equivalent to the expression formed by replacing the hole in C with M. The crucial difference between substitution and hole filling is that hole filling is allowed to capture free variables in the substituted term. As an example, we consider

$$\begin{array}{rcl} N & \equiv & \lambda x.\, y \\ C & \equiv & \lambda x.\, [\,] \end{array}$$

and the result of installing x into the body of these terms by each of the two mechanisms. In writing the substitution $N[y := x]$, we are implicitly asserting that x is not bound in N, since we have taken substitution to be an operation on free variables. Thus, we must write N as (for example) $\lambda z.\, y$. So we have $N[y := x] \equiv \lambda z.\, x$. Hole filling may capture variables, so in $C[x]$, the occurrence of x in the hole is in fact bound, and not free, and we have $C[x] \equiv \lambda x.\, x$.

One term to which we will frequently refer is the *identity function* which immediately returns its argument. We refer to this function as I, and define

$$I \equiv \lambda x.\, x \ \ .$$

The convention of abbreviating the identity function by I is common, and will be both valid and useful in all of the systems we study.

The main technical results of this thesis will concern *reduction relations* on pairs of terms. We define reduction relations in terms of reduction *rules*

$$(\mathit{Name}) \qquad \mathcal{F} \to \mathcal{G}$$

where $\mathcal{F}, \mathcal{G}$ are term schema abstracted over (presumably common) subterms M_i. The above statement of a reduction rule is equivalent to the more traditional format for a relation

$$\mathit{Name} = \{(\mathcal{F}, \mathcal{G}) : \text{ the } M_i \text{ are terms}\} \ \ .$$

We will sometimes place side conditions on the rules, which are taken as additional conditions in the set abstraction. Given two rules (R_1) and (R_2), we take

(R_1, R_2) to be the union of the two relations. The pairs of terms related by the basic relation are called *top-level* contractions. *Single-step* reduction $\xrightarrow[(R)]{}$ is defined as the *compatible closure* of the relation (R):

$$\frac{(M,N) \in (R)}{M \xrightarrow[(R)]{} N} \qquad\qquad \frac{M \xrightarrow[(R)]{} N}{C[M] \xrightarrow[(R)]{} C[N]} \tag{1.1}$$

We will usually omit the rule name(s) from the arrows when the details are clear from the context. *Multistep reduction* $\twoheadrightarrow$ is the reflexive, transitive closure of the single-step reduction relation $\rightarrow$. We take σ, τ to range over reduction sequences. To indicate that we will refer to a reduction sequence $M \twoheadrightarrow N$ as σ, we write $\sigma : M \twoheadrightarrow N$. We write $M \not\xrightarrow[R]{}$ whenever there exists no N such that $M \xrightarrow[R]{} N$, or in other words, whenever M is in R-normal form.

1.1.2 Types, logics and type assignment

Terms have *types* by which we associate terms with similar behavior. The core system has only unspecified base types, and a function type. We let A, B range over types. For any two types A_1, A_2, we write $A_1 \rightarrow A_2$ for the type of functions which expect an argument of type A_1 and return an expression of type A_2. In a systems with products and sums, one has product types $A \times B$ and sum types $A + B$. To discuss the type of an expression, we must know the types of its free variables. A *typing environment* or *typing context* is a list of pairs of the form

$$\mathit{variable} \ : \ \mathit{type} \ .$$

We let the capital Greek letters $\Gamma, \Delta, \Phi, \Psi$ range over typing contexts. So given an environment, we can assert a *typing statement* that in an environment Γ, a term M will have type A:

$$\Gamma \vdash M : A \ .$$

We specify the types that a term may take by *typing judgments* or *rules*. Judgments are of the form

$$\frac{\mathit{Statement}_1 \quad \cdots \quad \mathit{Statement}_n \qquad \mathit{Possible\ side\ conditions}}{\mathit{Concluding\ statement}}\ \text{Rule name} \ .$$

A particular class of rules, the axioms, have no statements above the bar. A *typing proof tree* is a tree of judgments, each of whose statements above the bar are either axioms, or else the conclusion of another typing tree.

Curry and Howard noticed that typing proof trees of terms correspond to proof trees in various logics [21, 34]. Over the next few sections we informally discuss a number of these logics which are especially relevant to the type systems we will consider. Most computer scientists are at least informally familiar with classical logic, which does not play an active role by itself in the systems we discuss here, but does lend useful insights by way of contrast with the variations we do use. The logics which correspond to the type systems here are all *minimal*, meaning that they do not have a false-elimination rule, and are all *intuitionistic*. Many ideas characterize and underlie minimal intuitionistic logic, but two are particularly relevant to our presentation. First, these logics consider statements proving a single logical formula, rather than statements with multiple conclusions. This restriction corresponds to the fact that we type a single term at a time; multiple-conclusion logics seem more appropriate for process calculi [12, for example]. Second, intuitionistic logic is based on the idea of *constructive* proof, an idea we illustrate later, after establishing some details of classical logic. The final concept which we sketch in this section is the notion of a *linear* logic. Linear logics restrict the use of structural rules whereby assumptions can be duplicated or discarded, operations which are relevant to resource usage in the various lambda calculi. We have drawn extensively from a number of sources for this discussion, in particular from the work of Abramsky [2], Barber [7], Girard [26, 27], Girard, Lafont and Taylor [28], Ryan and Sadler [66] and Scedrov [69].

1.1.2.1 Classical logic

In this section we attempt to give the reader a feeling for the classical propositional logic, mostly based on the formulation of Gentzen. We begin with the ubiquitous truth tables: Classical logic is about truth, and assigning one of the value $\mathbf{t}$ for truth or $\mathbf{f}$ for falsehood to every well-formed formula in its language.

We assume a set of atomic statements (which we leave unspecified), ranged over by the metavariable Z. The set of well-formed formulas are defined as usual by an inductive grammar:

$$A, B ::= Z \mid \neg A \mid A \to B \mid A \times B \mid A + B \ .$$

We give *values* to formulas by the truth tables:

Value of:

A	B	$A \to B$	$A \times B$	$A + B$	$\neg A$
$\mathbf{t}$	$\mathbf{t}$	$\mathbf{t}$	$\mathbf{t}$	$\mathbf{t}$	$\mathbf{f}$
$\mathbf{f}$	$\mathbf{t}$	$\mathbf{t}$	$\mathbf{f}$	$\mathbf{t}$	$\mathbf{t}$
$\mathbf{t}$	$\mathbf{f}$	$\mathbf{f}$	$\mathbf{f}$	$\mathbf{t}$	
$\mathbf{f}$	$\mathbf{f}$	$\mathbf{t}$	$\mathbf{f}$	$\mathbf{f}$	

Atomic sentences are given values by an *assignment* of each to a truth value; we can consider different assignments for any formula. We write $\mathbf{v}$ to range over assignments, and $\mathbf{v}(A)$ to be the truth-value of the formula A under assignment $\mathbf{v}$. The specifics of assignments and the possibility of larger domains of values than just $\{\mathbf{t}, \mathbf{f}\}$, while in fact very rich subjects, are not especially relevant to the type systems which we discuss in this thesis, and we avoid any deeper treatment of those issues. Valuation of the connectives follows the familiar negation and distribution laws:

$$A = \neg\neg A$$

$$\neg(A \times B) = (\neg A) + (\neg B)$$
$$\neg(A + B) = (\neg A) \times (\neg B)$$

$$A_1 \times (A_2 + A_3) = (A_1 \times A_2) + (A_1 \times A_3)$$
$$A_1 + (A_2 \times A_2) = (A_1 + A_2) \times (A_1 + A_3)$$

From individual formulas with truth-values we move to relations on collections of formulas. By *collection* we specifically mean *multiset*: our collections may have duplicates, and the order in which we write them down is irrelevant. We consider two such relations, *entailment* $\vDash$ and *consequence* $\vdash$. Both relations connect left-hand and right-hand multisets of formulas, and the interpretation of both relations are similar: if all of the left-hand formulas are (in some sense) acceptable, then at least one of the right-hand formulas will be acceptable.

Entailment is closer to the truth-table notions of classical logic. We say that formulas Γ entail formulas Φ if for every assignment to atomic formulas which makes every member of Γ true, at least one member of Φ will be true. While entailment is based on considering all possible assignments for formulas, a more appropriate relation for our purposes is consequence. Consequence is not defined in terms of truth values, but rather in terms only of the formulas themselves. We define the consequence relation $\vdash$ in terms of judgment rules relating sequents, just as described for type systems, but without the variables and terms decorating the formulas. We have four classes of rule: the axiom rule, the structural rules, the cut rule and the logical rules, as shown in Figure 1.1.

The axiom rule introduces formulas, based on the tautology that under the assumption that a thing is true, it is true.

The structural rules allow redundant assumptions to be pared, and unused assumptions to be added. It is somewhat unusual to mention Contraction explicitly. In presentations of classical logic, it is more common first to assume that we are dealing with sets, rather than multisets, of formulas, and second, in rules where two or more sequents appear above the judgment bar, to use the same contexts for the respective sides of all sequents. For example, one might expect to see the rule

$$\frac{\Gamma \vdash \Phi, A \qquad \Gamma \vdash \Phi, B}{\Gamma \vdash \Phi, A \times B} \times\text{-R}$$

rather than the $\times$-R rule of the figure. Taken by itself, explicit Contraction is overly pedantic, but becomes relevant in relation to the linear logics below, where the operation is restricted. It can also seem unusual at first glance that one would allow weakening on the right-hand side of assumptions, but this arrangement is

Axiom

$$\frac{}{A \vdash A}\ \mathrm{Id}$$

Structural rules

$$\frac{\Gamma \vdash \Delta}{\Gamma, A \vdash \Delta}\ \text{Weakening-L} \qquad \frac{\Gamma \vdash \Delta}{\Gamma \vdash \Delta, A}\ \text{Weakening-R}$$

$$\frac{\Gamma, A, A \vdash \Delta}{\Gamma, A \vdash \Delta}\ \text{Contraction-L} \qquad \frac{\Gamma \vdash \Delta, A, A}{\Gamma \vdash \Delta, A}\ \text{Contraction-R}$$

Cut rule

$$\frac{\Gamma \vdash \Phi, A \qquad \Delta, A \vdash \Psi}{\Gamma, \Delta \vdash \Phi, \Psi}\ \text{Cut}$$

Logical rules (implication)

$$\frac{\Gamma, A \vdash \Delta, B}{\Gamma \vdash \Delta, A \to B}\ \to\text{-R} \qquad \frac{\Gamma \vdash \Phi, A_0 \qquad \Delta, A_1 \vdash \Psi}{\Gamma, \Delta, A_0 \to A_1 \vdash \Phi, \Psi}\ \to\text{-L}$$

Logical rules (negation)

$$\frac{\Gamma, A \vdash \Phi}{\Gamma \vdash \Phi, \neg A}\ \neg\text{-R} \qquad \frac{\Gamma \vdash \Phi, A}{\Gamma, \neg A \vdash \Phi}\ \neg\text{-L}$$

Logical rules (products, sums)

$$\frac{\Gamma \vdash \Phi, A \qquad \Delta \vdash \Psi, B}{\Gamma, \Delta \vdash \Phi, \Psi, A \times B}\ \times\text{-R} \qquad \frac{\Gamma, A, B \vdash \Phi}{\Gamma, A \times B \vdash \Phi}\ \times\text{-L}$$

$$\frac{\Gamma \vdash \Phi, A, B}{\Gamma \vdash \Phi, A + B}\ +\text{-R} \qquad \frac{\Gamma, A \vdash \Phi \qquad \Delta, B \vdash \Psi}{\Gamma, \Delta, A + B \vdash \Phi, \Psi}\ +\text{-L}$$

Figure 1.1: Classical logic in a sequent calculus style.

actually standard. One should first notice that weakening on the right does *not* allow us to “prove” any formula: we cannot deduce

$$- \vdash A \ ,$$

where $-$ indicates an empty context, since

$$- \vdash -$$

is not what the axiom or any proof tree allows. Weakening on the right is in accordance with the idea that sequents assert only that *at least one* of the right-hand formulas are valid, not necessarily all of them. Weakening on the left introduces formulas which we do not need to justify any on the right; weakening on the right introduces formulas we do not claim to individually follow from the collection to the left. This arrangement of multi-formula sequents on both sides of the turnstile and with structural rules applicable on both sides allows us to omit explicit mention of both the falsity constant $\perp$, indicating a deduced contradiction, and a double-negation rule. Again, we have chosen the present formulation because it better matches the systems which follow.

The logical rules introduce connectives on each side of the turnstile, and the Cut rule describes the communication between the two sides. It can be useful to view assumptions as resources: then the right-rules describe the production of resources, the left-rules describe consumption, and the Cut rule allows a transaction between a producer and a consumer. The Cut rule corresponds to computation: in the sequent calculus formulation, all of the computation in the system is concentrated into this rule.

1.1.2.2 Intuitionistic and minimal intuitionistic logic

If classical logic concerns truth, and neatly classifying formulas, then intuitionistic logic concerns building proofs. The principles of building proofs in intuitionistic logic were set forth by Heyting [31]:

1. A proof of $A \times B$ is a pair containing a proof of A and a proof of B.

2. A proof of $A + B$ may be either a proof of A or a proof of B.

3. A proof of $A \to B$ is a method for transforming a proof of A into a proof of B.

This notion of a *constructive* system originated with Brouwer [19]. More than classical logic, constructiveness lends itself toward typing terms of the lambda calculus. The constructive ideas are demonstrated by restricting the right-hand sides to be at most one formula, and with rules that exhibit the *subformula* property. The subformula property states that the sequents occuring above the bar in a judgment contain only formulas which are subformulas of those in the sequent below the bar. The main restriction is to the structural rules. The Contraction-R rule no longer makes any sense, and does not appear; the Weakening-R and $\neg$-R rules are only applicable when the right-hand side is empty. We will have *exactly* one formula to the right of the turnstile, since only the $\neg$ rules of the full intuitionistic system ever changed the population of the right-hand side. Thus, in the minimal system we completely omit the structural rules for the right-hand side along with the $\neg$ rules.

Beyond paring down the rules, we also move to a different presentation of the deduction rules. Decorating sequent calculus rules with terms, both the left-rules and the Cut rule require substitution; this structure of the typing trees would not reflect the structure of the terms themselves. Instead, we turn to the natural deduction style of rules, shown in Figure 1.2. Instead of left and right rules, we have introduction and elimination rules. The introduction rules of natural deduction are just the same as the right rules of the sequent calculus: we introduce a connective on the right side of the turnstile. Rather than rules wich introduce connectives on the left hand side of a sequent, we have elimination rules which describe how right-hand connectives may be deconstructed, and their components accessed. In fact, the Cut rule itself no longer appears; one instead proves Cut Theorems. Taken at the level of terms, the Cut Theorems state that substitution of a term for a variable of the same type will produce a well-typed result (in fact, one with the same type). This arrangement corresponds to the syntactic construction of terms: we take first an expression with a function type, and then another expression of the appropriate argument type, and write them juxtaposed as a function application.

1.1.2.3 Linear logic

In the analogy of classical logic discussing truth and intuitionistic logic discussing proof, linear logic discusses resources. When a formula is proven, rather than becoming one more fact which shall always be true, it is instead a coin with which we might buy another formula — sacrificing the coin in the transaction. This effect arrives from a restriction to the structural rules: if we neither duplicate nor discard assumptions, then every formula introduced in an axiom must be used, and used precisely once, to conclude some other formula toward the final result.

Of course, a system with only such a lockstep use of formulas would be trivial, and a new modal connective ! allows us to express the idea of truths which are in fact eternal. This mode enables resticted versions of the structural rules, indicating an inexhaustible source for the formula beneath it. The restriction on the structural rules also raises an important question for product and sum formulas. Intuitively, in the classical case one use pairs either *projectively* or *deconstructively*. In the former case we use one of a pair's components; in the latter we dismantle the pair and use both, *i.e.* by copying the pair and projecting one component from each. However, since nothing may be implicitly copied or discarded, in linear logic we could not have such a choice. The modal connective is not sufficient by itself to make this distinction; multiple use of a projective pair to enable deconstruction would also allow multiple use of the components. Thus, linear logic has two connectives for pairing, the multiplicative or tensor product $\otimes$ and the multiplicative sum $\&$. The full linear logic system with multiple conclusions also has two sum operators, the additive product $\parr$, pronounced *par*, and the additive sum $\oplus$. The additive product makes sense only in a system with multiple conclusions, and so does not appear in minimal, intuitionistic formulations.

Models of linear logic again satisfy distribution and negation laws. For distributivity, we have that the multiplicative tensor distributes over the additive ten-

Axiom

$$\frac{}{A \vdash A}\,\mathrm{Id}$$

Structural rules

$$\frac{\Gamma \vdash A}{\Gamma, B \vdash A}\,\text{Weakening} \qquad \frac{\Gamma, A, A \vdash B}{\Gamma, A \vdash B}\,\text{Contraction}$$

Logical rules (implication)

$$\frac{\Gamma, A \vdash B}{\Gamma \vdash A \to B} \to\text{-I} \qquad \frac{\Gamma \vdash A \to B \qquad \Delta \vdash A}{\Gamma, \Delta \vdash B} \to\text{-E}$$

Logical rules (products)

$$\frac{\Gamma \vdash A \qquad \Delta \vdash B}{\Gamma, \Delta \vdash A \times B} \times\text{-I}$$

$$\frac{\Gamma \vdash A_0 \times A_1}{\Gamma \vdash A_0} \times\text{-E} \qquad \frac{\Gamma \vdash A_0 \times A_1}{\Gamma \vdash A_1} \times\text{-E}$$

Logical rules (sums)

$$\frac{\Gamma \vdash A}{\Gamma, \Delta \vdash A + B} +\text{-I} \qquad \frac{\Gamma \vdash B}{\Gamma, \Delta \vdash A + B} +\text{-I}$$

$$\frac{\Gamma \vdash A_0 + A_1 \qquad \Delta, A_0 \vdash B \qquad \Delta, A_1 \vdash B}{\Gamma, \Delta \vdash B} +\text{-E}$$

Figure 1.2: Minimal intuitionistic logic in the natural deduction style.

---

sor, and similarly for $⅋$ and $\&$:

$$\begin{array}{rcl} A \otimes (B_1 \oplus B_2) & = & (A \otimes B_1) \oplus (A \otimes B_2) \\ A ⅋ (B_1 \,\&\, B_2) & = & (A ⅋ B_1) \,\&\, (A ⅋ B_2) \ . \end{array}$$

However the distribution of additives over multiplicatives is not generally true. We do not consider the laws which apply to negation in linear logic.

We move to a linear deduction system in a number of steps. We first extend the language of terms for the model connective as described above. The mode indicates formulas which may be used other than exactly once. It is very convenient to make a similar distinction in contexts, denoting assumptions which can be used as many times as needed. To this end we use a *dual* context: we separate the context into two zones, one of whose assumptions must be used exactly once, which use of assumptions in the other zone is unrestricted. Thus judgements take the form

$$\Phi;\ \Gamma \vdash M : A$$

where in the *linear* zone Γ each assumption must be used exactly once. The dual formulation leads to superior substitution results; Wadler [77] and Barber [7] discuss these issues in more detail.

Figure 1.3 shows the natural deduction formulation of dual intuitionistic linear logic, DILL. The new structural rules reflect the restrictions intended to each context: weakening and contraction apply only to assumptions in the nonlinear or *intuitionistic* zone. We also distinguish between the zone in the !-I rule: a conclusion may be used arbitrarily many times only if the assumptions which underlie it may also be used many times. The Dereliction rules is new. Dereliction allows us to relax the restrictions on linearity and apply structural rules to a particular assumption, so long as we no longer claim that that assumption is used linearly.

A variant of linear logic which we will also use is *affine* logic. In the dual, minimal, intuitionistic framework, we make one change to derive affine logic: in the Weakening rule, we add the unused assumption to the affine (formerly linear) zone:

$$\frac{\Phi;\ \Gamma \vdash B}{\Phi;\ \Gamma, A \vdash B}\,\text{Weakening} \ .$$

Axiom

$$\frac{}{-;\ A \vdash A}\ \text{Id}$$

Structural rules

$$\frac{\Phi;\ \Gamma \vdash A}{\Phi, B;\ \Gamma \vdash A}\ \text{Weakening} \qquad \frac{\Phi, A, A;\ \Gamma \vdash B}{\Phi, A;\ \Gamma \vdash B}\ \text{Contraction}$$

$$\frac{\Phi;\ \Gamma, A \vdash B}{\Phi, A;\ \Gamma \vdash B}\ \text{Dereliction}$$

Logical rules (exponential)

$$\frac{\Phi;\ - \vdash A}{\Phi;\ - \vdash !A}\ !\text{-I} \qquad \frac{\Phi;\ \Gamma \vdash !A \qquad \Psi, A;\ \Delta \vdash B}{\Phi, \Psi;\ \Gamma, \Delta \vdash B}\ !\text{-E}$$

Logical rules (multiplicative product)

$$\frac{\Phi;\ \Gamma \vdash A \qquad \Psi;\ \Delta \vdash B}{\Phi, \Psi;\ \Gamma, \Delta \vdash A \otimes B}\ \otimes\text{-I} \qquad \frac{\Phi;\ \Gamma \vdash A_1 \otimes A_2 \qquad \Psi;\ \Delta, A_1, A_2 \vdash B}{\Phi, \Psi;\ \Gamma, \Delta \vdash B}\ \otimes\text{-E}$$

Logical rules (multiplicative sum)

$$\frac{\Phi;\ \Gamma \vdash A \qquad \Phi;\ \Gamma \vdash B}{\Phi;\ \Gamma \vdash A \,\&\, B}\ \&\text{-I}$$

$$\frac{\Phi;\ \Gamma \vdash A_1 \,\&\, A_2 \qquad i \in \{1, 2\}}{\Phi;\ \Gamma \vdash A_i}\ \&\text{-E}$$

Logical rules (additive sum)

$$\frac{\Phi;\ \Gamma \vdash A_i \qquad i \in \{1, 2\}}{\Phi;\ \Gamma \vdash A_1 \oplus A_2}\ \oplus\text{-I}$$

$$\frac{\Phi;\ \Gamma \vdash A_1 \oplus A_2 \qquad \Psi;\ \Delta, A_1 \vdash B \qquad \Psi;\ \Delta, A_2 \vdash B}{\Phi, \Psi;\ \Gamma, \Delta \vdash B}\ \oplus\text{-E}$$

Figure 1.3: Natural deduction in dual intuitionistic linear logic, DILL.

In affine logic only contraction is restricted; affine assumptions may be used *at most* once.

1.1.3 Quantifiers, type inference and program analysis

From the point of view of language semantics, the motivation for our work is clear; we will provide a framework which casts greater light on the operational semantic distinction between three different calling conventions. Still, some distance separates the material we present from application to current programming systems.

Functional languages are usually *polymorphic*: in addition to value parameters, one also has type variables over which terms may be abstracted. Thus, the identity function may be written once for all types, rather than one version for integers, one for characters and so on. Polymorphism in the syntax of the calculus translates to *universal quantification* in the logic. Although we do not discuss quantification here, it seems simple enough to extend the translation results to quantifiers, perhaps restricting polymorphic typing to function abstractions.

One also usually considers type *inference* and its decidability. In type inference, an algorithm assigns a type to a term with no typing information except the types of free variables. It is well-known that type inference is decidable in the non-polymorphic calculi, but not in the full polymorphic versions. However, useful and decidable subsets of full polymorphism are frequently used, perhaps most notably the Hindley/Milner approach [56].

As well as types, one can infer other program properties, usually in frameworks similar to type systems. In particular, the various linear systems which we present seem particularly relevant to analysis of strictness [38, 42] and update [59] properties. Intuitively, a function is strict in an argument if it will necessarily evaluate and use the argument. One can see strictness in terms of linearity by an absence of weakening: if a function cannot discard an argument, it must be the case that it will use it. Update analysis concerns the rewriting of closures in memory. If an argument will not be used more than once, then it is not necessary to perform such an update. The ability to avoid an update can be viewed as an absence of contraction: if the formal parameter appears linearly, then by that linearity it would not be accessed twice. One potential application of the work we present here would be a more unified framework for these analyses, which might reveal improvements to known techniques.

1.2 Outline

We begin in Part I with a review of the two “classical” lambda calculi, the call-by-name and call-by-value systems, in Chapter 2. While not the main interest here, all of the systems we study later in this thesis will be derived from these two. This chapter is also useful for describing the basic properties which are well-known for these two systems, but which we shall need to establish for the calculi we introduce here, and so we spend some detail describing how their properties may be shown.

In Chapter 3 we consider the call-by-let calculus, an extension of the call-by-value system with explicit let-bindings. This calculus enjoys the usual properties of the classical systems, and the extension from call-by-value is in fact conservative. This chapter also introduces a useful new technique marking redexes in calculi with bindings, which allows more powerful proof techniques than had been previously possible in the calculi we consider.

The call-by-let system can be extended further to the call-by-need calculus, which we study in Chapter 4. We introduced this system in previous work with Ariola, Felleisen, Wadler and Odersky [5, 53], and present improved proofs of a number of results, relying crucially on the techniques introduced in the previous chapter.

Where the type systems for the calculi in Chapters 2-4 are all annotations of rules from minimal logic; in the second part of this thesis we consider systems based on linear and affine logic. As with call-by-need, we have previously considered these systems in work with Odersky, Turner and Wadler, but here present improved proofs of their basic properties. The linear lambda calculus which we discuss in Chapter 5 corresponds to an intuitionistic fragment of linear logic, and has two beta rules plus two rules for commuting conversions; we show that the system satisfies the subject reduction and confluence properties, and present a standard notion for linear reduction. The affine lambda calculus of Chapter 6 is based on affine logic, and like the call-by-need system is based on evaluating subterms only according to demand. The affine calculus has one rule in addition to the rules of the linear calculus, and again satisfies the usual properties. In Chapter 7 we relate the systems based on minimal logics to the systems based on linear and affine logic via the Girard translations.

In Part III we consider the result of decomposing the operator ! in the calculus of dual intuitionistic linear logic. We present the separated-linear lambda calculus in Chapter 8. We show that the calculus satisfies a number of the usual substitution properties as well as subject reduction and confluence. In Chapter 9 we adapt the Girard translations to the separated framework, and introduce the hybrid translation. The Girard translations both preserve substitutions, and keep the same soundness and completeness properties for typing judgments, reduction and standard evaluation as the original mappings. Moreover, the new hybrid translation is sound and complete for all three relationships.

Finally we conclude the main body of the thesis with an assessment of our results and a discussion of some open problems in Chapter 10. In an appendix, we consider extensions of the systems and unitary translations for recursion.

Part I

Unrestricted Weakening and Contraction

Chapter 2

Classical lambda reduction

But when we think of a street, something more substantial comes to mind. Something dryer, even for New Orleans. Here in this wilderness there was such an established and maintained public way, an ancient overland portage between the river and one of the bayous. At its junction with the river Jean Baptiste Le Moyne, Sieur de Bienville, set his city of New Orleans. Bienville named the bayou, St. John, for his patron saint, but the street he gave no name.

(Chase)

THIS chapter reviews the call-by-name and call-by-value lambda calculi. We discuss the relevant properties of these systems, and delve into some detail about the basic proof techniques which we will use throughout later chapters for the systems we introduce. This chapter is intended as background and as an introduction to the techniques which we will use over the next few chapters; none of the results in this chapter are actually original to this thesis. Some historical notes on the development of these systems have been presented by Klop [41], Curry and Feys [21] and Barendregt [9].

2.1 The call-by-name lambda calculus

Figure 2.1 reviews the call-by-name lambda calculus Name. Types, terms and values are standard: a type, corresponding to a formula of the logic, is the base type or a function type; we take Z to range over base types which we leave unspecified and take A, B to range over types. A term is a value or a function application, and a value is a variable or a function abstraction. We use V to range over values and L, M, N to range over terms. In addition to the metanotation indicated in the figure, here and in subsequent systems we will use Δ to range over top-level redexes, *i.e.* not under any compatible closure rules, with the corresponding contractum Δ'. This use of Δ is slightly ambiguous, since we use the same latter to range over typing environments, but the meanings should be clear since we will never combine the two notations. We will also make constant use of the variable naming convention of Barendregt [9], writing the bound and free variables of terms with separate names.

Typing contexts or enviroments are sets of assumptions of the form $x : A$, where each variable x is distinct. A typing judgment $\Gamma \vdash M : A$ indicates that in the context Γ, the term M has type A. The type rules are standard, and are just the functional core of Curry-Howard correspondent rules to minimal intuitionistic logic of Figure 1.2.

We use the same notation for call-by-name reduction as in the previous chapter. The *standard reduction* or *evaluation* relation $\mapsto$ is compatibly closed under *evaluation contexts* E rather than general contexts C, and is in fact a partial function on terms. In this chapter, the reduction rules for standard reduction will be the same as the rules for (ordinary) reduction, but this coincidence will not be true in general. We discuss standard reduction in more detail in Section 2.1.7.

In the next few sections, we discuss the properties of reduction in call-by-name, and some useful techniques for showing these properties. All of these results are well-known and can be found in various references. In particular, we tend to follow the style of Barendregt's reference for the non-type related results [9, Chapters 3 and 11].

2.1.1 Substitution

Basic results about substitution describe the interaction of that mechanism with typing and reduction. For typing systems, it will be simple to show that for a typing of a term M in an environment Γ in which x is taken to have type A, and

Syntactic domains

Types	A, B	$::=$	$Z \mid A \to B$
Values	V	$::=$	$x \mid \lambda x.\, M$
Terms	L, M, N	$::=$	$V \mid M\ N$
Evaluation Contexts	E	$::=$	$[\,] \mid E\ M$
Answers	A	$::=$	V

Typing judgments

$$\frac{}{x : A \vdash x : A}\ \text{Id} \qquad \frac{\Gamma \vdash M : B}{\Gamma,\, x : A \vdash M : B}\ \text{Weakening}$$

$$\frac{\Gamma,\, y : A,\, z : A \vdash M : B}{\Gamma,\, x : A \vdash M[y := x, z := x] : B}\ \text{Contraction}$$

$$\frac{\Gamma,\, x : A \vdash M : B}{\Gamma \vdash \lambda x.\, M : A \to B}\ \to\text{-I} \qquad \frac{\Gamma \vdash M : A \to B \qquad \Delta \vdash N : A}{\Gamma,\, \Delta \vdash M\ N : B}\ \to\text{-E}$$

Reduction relation

$$(\beta\to) \qquad (\lambda x.\, M)\ N \quad \to \quad M[x := N]$$

Standard reduction relation

$$(\beta\to) \qquad (\lambda x.\, M)\ N \quad \mapsto \quad M[x := N]$$

Figure 2.1: The call-by-name lambda calculus Name

---

a well-typed term N also of type A, then the substitution $M[x := N]$ is again well-typed, and can be given the same type as M.

Lemma 2.1.1. Let $\Gamma, x : B \vdash_{\mathsf{Name}} M : A$ and $\Delta \vdash_{\mathsf{Name}} N : B$. Then

$$\Gamma, \Delta \vdash_{\mathsf{Name}} M[x := N] : B \ \ .$$

Proof: In effect, this result says that we can compose typing proofs. The result follows by induction on the depth of the typing proof for M. The basic cases are both with the Id rule: if $M \equiv z \not\equiv x$, then we have a proof tree

$$\frac{}{z : A \vdash z : A}\ \text{Id}\ \ ,$$

and since $(z[x := N]) \equiv z$, the result is immediate. For $M \equiv x$, we have that the statement

$$\Gamma, \Delta \vdash_{\mathsf{Name}} M[x := N] : B$$

is just $\Delta \vdash_{\mathsf{Name}} N : B$, which is given. For the inductive step we consider the rule applied at the rootmost node of the typing tree; most of the cases fall into one of two general categories. When only one statement appears above the bar, the result is usually immediate. Taking for example the $\to$ -I rule, we have a tree of the form

$$\frac{\begin{array}{c}\vdots\ \mathcal{P} \\ \Gamma, x : B, z : A_0 \vdash M_0 : A\end{array}}{\Gamma, x : B \vdash \lambda z.\, M_0 : A_0 \to A}\ \to\text{-I}\ \ .$$

Then by the induction hypothesis we have $\Gamma, \Delta, z : A_0 \vdash M_0[x := N] : A$, from which we can deduce

$$\frac{\begin{array}{c}\vdots\ \mathcal{P}\text{ and the I.H.} \\ \Gamma, \Delta, z : A_0 \vdash M_0[x := N] : A\end{array}}{\Gamma, x : B \vdash \lambda z.\, (M_0[x := N]) : A_0 \to A}\ \to\text{-I}\ \ ,$$

and trivially $\lambda z.\, (M_0[x := N]) \equiv (\lambda z.\, M_0)[x := N]$ so we have the result.

The only rule with one conclusion above the bar which may give us some trouble in the Contraction rule, when the new variable which contraction introduces is in fact x, *viz.*

$$\dfrac{\Gamma, y : B, z : B \vdash M : A}{\Gamma, x : B \vdash M[y := x, z := x] : A}\ \text{Contraction}\ .$$

This case is tricky because we cannot simply apply the inductive hypothesis twice: after the first application, the depth of the proof tree would increase such that the hypothesis would no longer be applicable. The trick is to swap the Contraction rule with the rule above it whenever possible. So if for example we have at first

$$\dfrac{\dfrac{\begin{matrix}\vdots\ \mathcal{P}\\ \Gamma, y_0 : B, y_1 : B, z : A_0 \vdash M : A\end{matrix}}{\Gamma, y_0 : B, y_1 : B \vdash \lambda z.\, M : A_0 \to A}\ \to\text{-I}}{\Gamma, x : B \vdash \lambda z.\, M[y_0 := x, y_1 := x] : A_0 \to A}\ \text{Contraction}\ ,$$

we can also construct

$$\dfrac{\dfrac{\begin{matrix}\vdots\ \mathcal{P}\\ \Gamma, y_0 : B, y_1 : B, z : A_0 \vdash M : A\end{matrix}}{\Gamma, x : B, z : A_0 \vdash M[y_0 := x, y_1 := x] : A}\ \text{Contraction}}{\Gamma, x : B \vdash \lambda z.\, M[y_0 := x, y_1 := x] : A_0 \to A}\ \to\text{-I}\ .$$

We have not increased the depth of $\mathcal{P}$, and so we can apply the inductive hypothesis to the deduction of $\Gamma, x : B, z : A_0 \vdash M[y_0 := x, y_1 := x] : A$ as for the usual cases for $\to$ -I.

This manipulation will not be possible if the rule for the step above the Contraction step has two statements above it, with one of the two contracted variables in each, *viz.*

$$\dfrac{\dfrac{\begin{matrix}\vdots\ \mathcal{P}_0\\ \Gamma_0, y_0 : B \vdash M_0 : A_0 \to A\end{matrix}\quad \begin{matrix}\vdots\ \mathcal{P}_1\\ \Gamma_1, y_1 : B \vdash M_1 : A_0\end{matrix}}{\Gamma_0, \Gamma_1, y_0 : B, y_1 : B \vdash M_0\, M_1 : A}\ \to\text{-E}}{\Gamma_0, \Gamma_1, x : B \vdash (M_0\, M_1)[y_0 := x, y_1 := x] : A}\ \text{Contraction}\ .$$

In this case we can show the result directly. We can construct arbitrary renamings of the free variables of Δ within N; we let $\Delta \equiv x_1 : B_1, \cdots, x_n : B_n$ and consider renamings $\bar{\Delta} \equiv \bar{x}_1 : B_1, \cdots, \bar{x}_n : B_n$ and $\hat{\Delta} \equiv \hat{x}_1 : B_1, \cdots, \hat{x}_n : B_n$, typing terms $\bar{N}$ and $\hat{N}$ respectively. Then from the induction hypothesis on the $\mathcal{P}_i$ we have

$$\begin{matrix}\vdots\ \mathcal{P}_0 \text{ and the I.H.}\\ \Gamma_0, \bar{\Delta} \vdash M_0[y_0 := \bar{N}] : A_0 \to A\end{matrix}$$

and

$$\begin{matrix}\vdots\ \mathcal{P}_1 \text{ and the I.H.}\\ \Gamma_1, \hat{\Delta} \vdash M_1[y_1 := \hat{N}] : A_0\end{matrix}$$

which we can combine by $\to$ -E as usual, and apply the Contraction rule n times to replace each $\bar{x}_i$ and $\hat{x}_i$ by x_i:

$$\dfrac{\begin{matrix}\dfrac{\begin{matrix}\vdots\ \mathcal{P}_0\\ \Gamma_0, \bar{\Delta} \vdash M_0[y_0 := \bar{N}] : A_0 \to A\end{matrix}\quad \begin{matrix}\vdots\ \mathcal{P}_1\\ \Gamma_1, \hat{\Delta} \vdash M_1[y_1 := \hat{N}] : A_0\end{matrix}}{\Gamma_0, \Gamma_1, \bar{\Delta}, \hat{\Delta} \vdash M_0[y_0 := \bar{N}]\, M_1[y_1 := \hat{N}] : A}\ \to\text{-E}\\ \vdots\ \text{Contraction}\\ \Gamma_0, \Gamma_1, \Delta \vdash M_0[y_0 := N]\, M_1[y_1 := N] : A\end{matrix}}{\Gamma_0, \Gamma_1, \Delta \vdash (M_0\, M_1)[y_0 := N, y_1 := N] : A}\ \equiv\ .$$

Finally, if there are two statements above the bar, the substituted variable may appear in only one, so the inductive step is just as for the simpler cases of a single step above the bar. ♦

We will also use results on the interaction of substitution and reduction. It will always, in all of the systems we consider, be true that for all $N_1 \to N_2$, we will have $M[x := N_1] \twoheadrightarrow M[x := N_2]$. In fact, this observation is an immediate consequence of the way we defined reduction to be compatibly closed in Equations (1.1). It will often be true that $N_1[x := M] \to N_2[x := M]$, but we will sometimes need to restrict M, as we will first see when we discuss call-by-value reduction below.

Lemma 2.1.2. [9, Remark 3.1.7.(i) and Proposition 3.1.16] Let $L, M, N \in \mathsf{Name}$, where $M \xrightarrow[\mathsf{Name}]{} N$. Then:

1. $L[x := M] \xrightarrow[\mathsf{Name}]{}\!\!\!\!\!\rightarrow L[x := N]$.

2. $M[x := L] \xrightarrow[\mathsf{Name}]{} N[x := L]$.

Proof: The first clause is an discussed above. The second clause follows from an induction on M. The base case is a top-level $\beta{\to})$ step; we have

$$\begin{array}{rl} & M[x := L] \\ \equiv & ((\lambda y.\, M_0)\ M_1)[x := L] \\ \equiv & ((\lambda y.\, M_0[x := L])\ M_1[x := L]) \\ \to & (M_0[x := L])[y := (M_1[x := L])] \\ \equiv & (M_0[y := M_1])[x := L] \\ \equiv & N[x := L] \end{array}$$

since $y \notin \mathrm{fv}(L)$. The induction step is easy, as we can always just wrap the above sequence of terms in whatever context is relevant. ♦

2.1.2 Subject reduction

Subject reduction is a property of typed reduction which says that reduction steps preserve the types of the reduced expression; all of the systems which we consider will enjoy this property. Subject reduction follows rather easily from the substitution lemma for types described above, along with some juggling of type trees in the systems we will consider in later chapters.

Proposition 2.1.3, $\mathrm{SR}(\mathsf{Name})$**.** Name satisfies subject reduction.

Proof: Take $\Gamma \vdash M : A$ and $M \to N$. The proof is by induction on the typing proof tree of M. Only the base case of top-level reduction is at all interesting: we have a proof tree of the form

$$\dfrac{\dfrac{\begin{array}{c}\vdots\ \mathcal{P}_0 \\ \Gamma_0, x : A_0 \vdash M_0 : A\end{array}}{\Gamma_0 \vdash \lambda x.\, M_0 : A_0 : A} \to\text{-I} \qquad \begin{array}{c}\vdots\ \mathcal{P}_1 \\ \Gamma_1 : M_1 : A_0\end{array}}{\Gamma_0, \Gamma_1 \vdash (\lambda x.\, M_0)\ M_1 : A} \to\text{-E} \quad .$$

Then by Lemma 2.1.1 we have that $\Gamma_0, \Gamma_1 \vdash M_0[x := M_1] : A$, which is the result we need for the lemma. ♦

2.1.3 Normalization

Normalization properties limit certain classes of sequences in a reduction system to finite length; in particular, *strong normalization* guarantees the finiteness of *all* sequences in a system. Broadly speaking, we will not pursue normalization properties for the systems we present, even though many do in fact enjoy the property. As we hope that this work can be applied to analyses of actual programming languages, we should not preclude the addition of recursion operators, which do not usually enjoy strong normalization. Nonetheless, we will frequently prove strong normalization for subsets of our systems — subsets which should continue to be strongly normalizing even in non-strongly normalizing extensions — when it is useful for showing other properties.

2.1.4 Confluence

Confluence, *a.k.a.* the Church-Rosser property, states that terms derived by reduction from a common term have a common reduct: if $M \twoheadrightarrow M_1$ and $M \twoheadrightarrow M_2$, then there exists some N such that $M_1 \twoheadrightarrow N$ and $M_2 \twoheadrightarrow N$. Confluence is an important property, guaranteeing for example that final normal forms are unique:

Lemma 2.1.4. Let $\rightarrow$ be any confluent reduction relation; then $\rightarrow$-normal forms are unique.

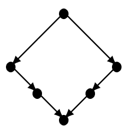
Church and Rosser demonstrated this result for the original lambda systems [20]; Barendregt's reference [9], which seems more readily available today, shows this result as Corollary 3.1.13.

Proof: Let N_1 and N_2 be $\rightarrow$-normal forms with $M \twoheadrightarrow N_i$ for both N_i. Then there exists some N such that for both N_i we have $N_i \twoheadrightarrow N$. But since both N_i are normal forms, we have that in fact it must be zero steps required to cenvert each N_i to N, and so $N_1 \equiv N \equiv N_2$. ♦

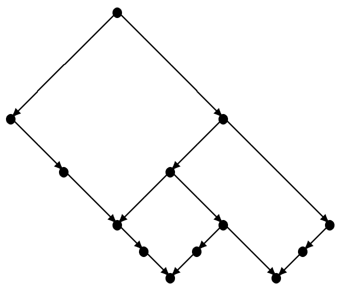
Unfortunately, confluence is difficult to show. It is important to note the difference between confluence and *weak* confluence: although the two are different, the latter can be useful. Weak confluence requires converging sequences only for divergence by a single step from a common term.

Definition. A reduction relation $\rightarrow$ is *weakly confluent* if for all M, M_1, M_2 where $M \rightarrow M_1$ and $M \rightarrow M_2$, there exists some N such that $M_1 \twoheadrightarrow N$ and $M_2 \twoheadrightarrow N$. ♢

Weak confluence is usually fairly easy to show given a substitution and reduction lemma in the style of Lemma 2.1.2. Unfortunately, it does not generally imply confluence. The problem is that without a stronger premise than what weak confluence offers, we cannot assume other than that both sequences $M_0 \twoheadrightarrow N$ and $M_1 \twoheadrightarrow N$ are strictly longer than one single step: graphically, and taking each of length two for simplicity, we have



When we try to use the relationships in the above diagram to join a two-step and a single step sequence from the same term, we reduce the problem only to itself again:



However, confluence does follow from weak confluence in a strongly normalizing system:

Lemma 2.1.5. Let $\rightarrow$ be a weakly confluent and strongly normalizing reduction relation. Then $\rightarrow$ is confluent.

This result originated with Newman [60], and is included in Barendregt's survey [9, Proposition 3.1.25].

Proof: Since $\rightarrow$ is strongly normalizing, $\rightarrow$-normal forms exist for all terms. It is sufficient to show that the normal form of M is unique: then this unique normal form is the common reduct for all terms derived from M; weak confluence suffices here since a reduction sequence of maximal langth for M can be identified. ♦

As stated above, we will use this technique for showing confluence only when considering subsets of the systems which should continue to be strongly normalizing under an overall extension for recursion. We will therefore also need a more general technique for showing confluence for non-normalizing systems. We return to this topic in Section 2.1.6, where we use the notion of developments.

Syntactic domains As in Name, plus the following:

$$\text{Terms} \qquad L', M', N' \quad ::= \quad \cdots \mid {}^{[\beta\to]}(\lambda x.\, M')\, N'$$

Reduction relation

$$\begin{array}{lllll} (\beta{\to}_0) & {}^{[\beta\to]}(\lambda x.\, M')\, N' & \to & M'[x := N'] \\ (\beta{\to}_1) & (\lambda x.\, M')\, N' & \to & M'[x := N'] \end{array}$$

Figure 2.2: The marked call-by-name calculus Name'.

2.1.5 Developments and residuals

Developments and residuals are technical notions based on the idea of tracking redexes through a term. For example, in the term

$$(\lambda xy.\, y\; x\; x)\; (I\; I)\; (\lambda xy.\, y)$$

there are two redexes which we can immediately contract: the application

$$(\lambda xy.\, y\; x\; x)\; (I\; I) \tag{2.1}$$

and the application

$$I\; I \;\; . \tag{2.2}$$

We can also *mark* these redexes, and observe how they are duplicated, shifted and discarded by other contractions. Along with each calculus we study, we will also consider a *marked* variant of that calculus. The marked call-by-name calculus Name', which we present in Figure 2.2, includes an extra term form for a marked $(\beta{\to})$ redex:

$${}^{[\beta\to]}(\lambda x.\, M')\, N' \;\; .$$

We let M', N' and so forth range over marked terms. Generally speaking, we will have a new term in the marked calculus for every rule in a particular system, although we will sometimes not mark certain redexes, and we will sometimes need more than one rule to describe how a single marked term may be reduced. Returning to the running example, to mark the redex in Line 2.1 we use the term

$${}^{[\beta\to]}(\lambda xy.\, y\; x\; x)\; (I\; I)\; (\lambda xy.\, y)$$

while for the redex in Line 2.2 we have

$$(\lambda xy.\, y\; x\; x)\; ({}^{[\beta\to]}I\; I)\; (\lambda xy.\, y) \;\; .$$

We will write Name'_0 to refer to reduction of marked terms only, that is by the rule $(\beta{\to}_0)$ only. A substitutivity result on marked term reduction will be useful.

Lemma 2.1.6. Let $L', M', N' \in \mathsf{Name}'$, where $M' \xrightarrow[\mathsf{Name}'_0]{} N'$. Then:

1. $L'[x := M'] \xrightarrow[\mathsf{Name}'_0]{} \!\!\!\!\rightarrow L'[x := N']$.
2. $M'[x := L'] \xrightarrow[\mathsf{Name}'_0]{} N'[x := L']$.

Proof: Both follow by structural induction and compatible closure arguments exactly as in Lemma 2.1.2. ♦

Residuals, intuitively, are the redexes which other contractions leave marked. For example, if we mark Redex (2.2) and contract Redex (2.1), we produce a term with two residuals of the marked redex:

$$(\lambda y.\, y\; ({}^{[\beta\to]}I\; I)\; ({}^{[\beta\to]}I\; I))\; (\lambda xy.\, y) \;\; .$$

A concrete definition of residuals requires a bit more notation. We need to consider the location of marked redexes in a term without discussing the term itself, or to discuss marking as separate entities from the marked term itself. We do so in terms of strings of symbols which index subterms. In Name we have three such symbols or *steps*, namely $\underline{\lambda 1}$, $\underline{@1}$ and $\underline{@2}$, read respectively "under the abstraction," "on the left side of the application" and "on the right side of the application." A *path* is a string of steps which is taken as a roadmap from the top-level of a term to a particular subterm. In the above example, we find Redex (2.1) indexed by the path $\underline{@1}$, since application associates to the left. Redex (2.2) would then be indexed by $\underline{@1}\,\underline{@2}$. We let γ, δ range over paths, and $\bar{\gamma}, \bar{\delta}$ range over sets of paths. If a term M has a redex at the subterm indexed by a path γ, we can refer to the contraction of that redex as $M \xrightarrow{\gamma} N$.

We use sets of paths to relate marked and unmarked terms. It is clear that we can project a marked term M' to an unmarked term simply by erasing its marks, and we indicate this operation by $|M'|$. To move in other direction, deriving a marked term from an unmarked one, we specify a set of paths to redexes in the unmarked term, taking exactly those redexes to be marked. If we take M' to be the marking of redexes $\bar{\gamma}$ in M, we write $M' \equiv (M, \bar{\gamma})$, and we refer to M' as a *lifting* of M.

We can also project and lift reduction sequences. For projection we simply erase all markings from each term in the sequence. For liftings, it is clear that given some marked term $M' \equiv (M, \bar{\gamma})$ where $\sigma : M \to N$, we have some N' and $\bar{\delta}$ such that $N' \equiv (N, \bar{\delta})$. Taking $\sigma' : N' \equiv (N, \bar{\delta})$, we write $|\sigma'| \equiv \sigma$. We formally define residuals in terms of lifting: the set of residuals of this $\bar{\gamma}$ with respect to σ is defined to be $\bar{\delta}$, which we abbreviate as $\bar{\gamma}/\sigma = \bar{\delta}$. We will use paths in a number of analyses in the various proofs in addition to the definition of residuals.

A *development* is a reduction sequence where we contract only marked redexes, either marked in some original marking, or their residuals. For a development which is a single-step reduction sequence from M' to N' we write $M' \xrightarrow[\text{dev}]{} N'$, and for multistep sequences, $M' \xrightarrow[\text{dev}]{}\!\!\!\!\!\rightarrow N'$. It is clear that reduction of marked terms by Name_0' and developments of markings coincide.

Lemma 2.1.7. Name_0' reduction corresponds to Name-developments. ♦

In all of the systems we will study, we will show that all developments are finite. We prove the equivalent result that marked reduction is strongly normalizing. Our technique is simple, if tedious. We derive a calculus of *weighted* marked terms, identical to the marked calculus except that every non-binding variable occurrence is accompanied by a *weight*, some positive integer. In other words, the grammar of weighted marked terms is just like the grammar of marked terms except for the production of variables:

$$\dot{L}, \dot{M}, \dot{N} ::= x^i \mid \cdots \ .$$

We evaluate weighted terms with a norm, written $\|\dot{M}\|$. For call-by-name, the norm is simple [9, Definition 11.2.15]: the norm of a term is simply the sum of the weights occurring in that term, which is always positive since all weights are positive, and since all terms have at least one variable occurrence. We need one further notion, that of a *decreasing weighting*. A weighted call-by-name term $\dot{M}$ has a decreasing weighting if for every subterm ${}^{[\beta\to]}(\lambda x.\, \dot{M}_0)\, \dot{M}_1$ within it, the norm of $\dot{M}_1$ is less than every weight associated with x in $\dot{M}_0$ [9, Definition 11.2.16]:

$$(\forall x^i \text{ occurring in } \dot{M}_0)\ i > \|\dot{M}_1\| \ .$$

Straightforward, albeit lengthy, analysis can show that if $\dot{M}$ has a decreasing weighting and $\dot{M} \to \dot{N}$, then both $\|\dot{M}\| > \|\dot{N}\|$ and $\dot{N}$ has a decreasing weighting; it is also possible to construct a decreasing weighting for every term [9, Lemma 11.2.17]. Since all norms are positive integers, strong normalization is immediate.

Proposition 2.1.8, $\mathsf{FD}(\mathsf{Name})$**.** All Name-developments are finite.

Proof: As outlined above. ♦

A *complete* development is one in which the final term is unmarked; we write $M' \xrightarrow[\text{cpl}]{} N$ to denote such a reduction sequence. Since developments and marked reduction coincides, a complete development is one which ends in a $(\beta{\to}_0)$-normal form. It will be useful to state conditions under which all complete developments end in the same term; such a statement is called a *strengthened* finiteness of developments result. For classical beta reduction this latter statement is universally true; all call-by-name developments are finite and can be extended to complete developments, all of which end in the same term. The strengthened result is implied by the confluence of marked reduction. If marked reduction is confluent, then $(\beta{\to}_0)$-normal forms are unique, and then so are all complete developments. Since we already know marked reduction to be strongly normalizing, by Lemma 2.1.5 we need only show that it is weakly confluent, which follows from an analysis of the possible redexes with Lemma 2.1.2.

Proposition 2.1.9, FD!(Name)**.** All Name-developments are finite, all can be extended to complete developments, and all complete Name-developments end in the same term. ♦

Strengthened finiteness of developments will hold for almost all markings in the systems we will consider in later chapters, although the additional term structures in the let-binding and linear systems will add some technical hurdles from what Hindley referred to in a different context as redexes being "too close together" [32]. In some systems, it will suffice simply to identify such aberrant markings, and exclude them from the theory; in others, we will explore more general notions of reduction which work around the difficulties. We will use properties of developments of redexes — especially on these more general notions of reduction — to achieve more elegant proofs of confluence and standard evaluation strategies than in previous work in much the same way as in the proof techniques surveyed by Barendregt [9] are clearer and more flexible than many earlier results on the classical systems.

We will need one more technical result about Name developments for a proof in Chapter 4. The proof is by Barendregt [9, Lemma 11.3.3].

Lemma 2.1.10. Let $M, M_0, M_1 \in \mathsf{Name}$ and $(M, \bar{\gamma}) \in \mathsf{Name}'$, where we have both $\sigma : M \xrightarrow[\mathsf{Name}]{} \!\!\!\!\!\rightarrow M_0$ and $(M, \bar{\gamma}) \xrightarrow[\text{cpl}]{} M_1$. Then there exists some N such that $(M_0, \bar{\gamma}/\sigma) \xrightarrow[\text{cpl}]{} N$ and $M_1 \twoheadrightarrow N$.

Barendregt's lemma actually goes further, and proves a stronger result about a subset of Name terms, namely that over Λ_I terms we will have as well that the term N is not the same as M_1, *i.e.* that the mandated sequence $M_1 \twoheadrightarrow N$ has length at least one. The restriction to Λ_I terms is used only for the additional condition, and the condition we state here in fact holds for Name reduction in general.

Proof: By induction on the length of σ. For the single-step case, if σ is a development of $(M, \bar{\gamma})$ then $N \equiv M_1$. Otherwise we have σ contracting some redex indexed by the path δ, and we can consider the marking $(\bar{\gamma} \cup \{\delta\})$ for M. Then both σ and all complete developments of $(M, \bar{\gamma})$ are partial developments of $(M, \bar{\gamma} \cup \{\delta\})$, which by FD!(Name) has a unique completion. For multistep σ the result is simply the inductive consideration of one step at a time. ♦

The finiteness of developments is, in apparent contradiction to the discussion a few paragraphs above about the sort of result we seek, a form of normalization result. In effect, the property says that a certain subset of the reduction relation is strongly normalizing. However, we are not compromising the possibility of non-normalizing extensions, since developments of the calculi extended with recursive constructs will continue to be finite.

2.1.6 Confluence revisited

With the material of the previous section, we can devise an alternate proof technique for confluence. The inference relies on two observations: first, every single-step reduction can be seen as the complete development of the marking of a single redex. The second point is that, taking as in the premise of weak confluence that $M \to M_1$ and $M \to M_2$, the two steps contract (in the non-degenerate case) two different redexes in M, both of which can be marked, and from this marking both single steps can be seen as partial developments. Then by strengthened finiteness of developments, the completions of these partial developments end in the same term: the common reduct. The difference between this reasoning on developments and the earlier argument with weak confluence is that with complete developments we reach the common reduct in a single step — the completion. With a single-step completion relation, as opposed to the multiple steps required by simple reduction, we can construct a simple inductive argument from which confluence follows.

The preceding paragraph makes a significant implicit assumption: that given any two redexes in a term M, we can have a marked term lifted from M in which

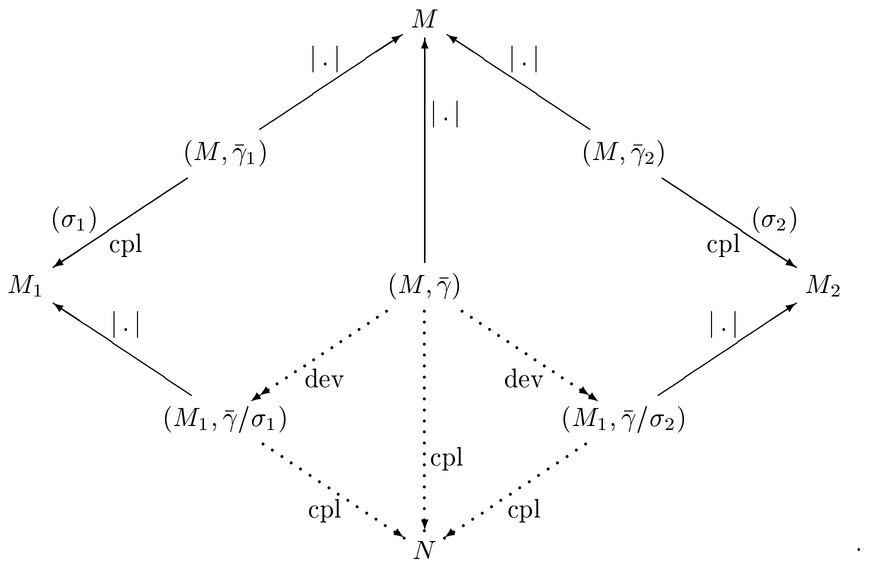


Figure 2.3: The relationship necessary for strengthened finiteness of developments to imply confluence.

both of those redexes are marked. Of course, in call-by-name this notion is trivial, since the usual set union of the paths to marked redexes will clearly suffice. However, in later systems such a construction will not necessarily be possible, so we formulate the main lemma here in terms of a slightly more general notion, that of *joinable markings*.

Definition 2.1.11. Let R' be a marked reduction relation which can mark all redexes of a system R. We say that R' has *joinable markings* if for some *joining operator* $+$ mapping pairs of paths sets to another path set, the following statement holds:

> Let $M \in R$ and $(M, \bar{\gamma}_0), (M, \bar{\gamma}_1) \in R'$ where
>
> $$\begin{aligned} \sigma_0' : (M, \bar{\gamma}_0) &\xrightarrow[\text{cpl}]{} N_0 \text{ and} \\ \sigma_1' : (M, \bar{\gamma}_1) &\xrightarrow[\text{cpl}]{} N_1 \ . \end{aligned}$$
>
> Then $(M, \bar{\gamma}_0 + \bar{\gamma}_1) \in R'$, and each $|\sigma_i'|$ can also be taken as a development of $(M, \bar{\gamma}_0 + \bar{\gamma}_1)$.

We refer to such an $(M, \bar{\gamma}_0 + \bar{\gamma}_1)$ as a *joining* of the $(M, \bar{\gamma}_i)$. ◊

For call-by-name the joining operator is just set union, a rather trivial observation, and joinability is rarely (if ever) mentioned explicitly in literature discussing call-by-name developments. However, in the systems we present later the joining operator is not trivial, and thus it is important to explicitly recognize the property.

Now we can describe the criteria for deducing confluence from strengthened finiteness of developments:

Lemma 2.1.12. Let (R) be a notion of reduction with a notion of development where all markings are joinable, all R-developments are finite and extendible to complete developments, and all complete R-developments end in the same term. Then R-reduction is confluent.

Proof: Given two complete developments of R-markings $\sigma_0 : (M, \bar{\gamma}_0) \xrightarrow[\text{cpl}]{} N_0$ and $\sigma_1 : (M, \bar{\gamma}_1) \xrightarrow[\text{cpl}]{} N_1$, both $\bar{\gamma}_0$ and $\bar{\gamma}_1$ can be seen as partial developments of M with some joining $\bar{\gamma}$ of $\bar{\gamma}_0$ and $\bar{\gamma}_1$. The partial developments have a unique completion N, and from each N_i we can reach N by a single complete development of the residuals of $\bar{\gamma}$ with respect to N_{1-i}. This reasoning is illustrated in Figure 2.3. Then since each single step in a reduction sequence can be seen as the complete development of a single redex, confluence follows by the obvious inductions. ♦

Proposition 2.1.13, CR(Name)**.** Name is confluent.

Proof: With Lemma 2.1.12, whose conditions are satisfied by the above results and discussion. ♦

The following simple lemma will also be helpful in the various confluence proofs.

Lemma 2.1.14. We take $\rightarrow^{=}$ to indicate the reflexive closure of a reduction relation $\rightarrow$ on terms X, and let $(R_1), (R_2)$ be two notions of reduction on X.

1. If for all M, M_1, M_2 such that $M \xrightarrow[(R_1)]{} M_1$ and $M \xrightarrow[(R_2)]{} M_2$ we have some M_0 such that $M_1 \xrightarrow[(R_2)]{}\!\!\!\!\rightarrow M_0$ and $M_2 \xrightarrow[(R_1)]{}^{=} M_0$, then $\xrightarrow[(R_1)]{}\!\!\!\!\rightarrow$ and $\xrightarrow[(R_2)]{}\!\!\!\!\rightarrow$ commute: for all N, N_1, N_2 such that $N \xrightarrow[(R_1)]{}\!\!\!\!\rightarrow N_1$ and $N \xrightarrow[(R_2)]{}\!\!\!\!\rightarrow N_2$, there exists some N_0 such that $N_1 \xrightarrow[(R_2)]{}\!\!\!\!\rightarrow N_0$ and $N_2 \xrightarrow[(R_1)]{}\!\!\!\!\rightarrow N_0$.
2. If (R_1) and (R_2) reduction are each individually confluent, and if Condition (1) holds, then (R_1, R_2) reduction is confluent.

Proof: As in Barendregt [9, Lemmas 3.3.5 and 3.3.6], for example. ♦

2.1.7 Standard reduction

The notion of standardization which we pursue, rather than the formulation of Curry and Feys [21], is the notion of evaluation functions of Plotkin [62] expressed as a reduction relation in the manner of Felleisen and Hieb [23]: In the former, standardization is a process by which we may transform any reduction sequence — from and to arbitrary terms — into a canonical form. Here, for a given reduction-closed, syntactic notion of *answer*, we consider a strategy for choosing redexes in a term which leads to and stops at an answer whenever such a sequence exists. Formally, we define:

Definition 2.1.15. For a reduction relation $\rightarrow$ with a subset $\mapsto$ and a notion of answer A, we say that $\mapsto$ is a *standard reduction strategy* for $\rightarrow$ if all of the following are true:

1. If $M \mapsto N_0$ and $M \mapsto N_1$ then $N_0 \equiv N_1$.
2. For all answers A, $A \not\mapsto$.
3. If $M \twoheadrightarrow A$ then there exists some answer A_0 such that $M \mapsto\!\!\!\!\rightarrow A_0$.

If $\sigma : M \rightarrow N$ but $M \not\mapsto N$, then we say that σ is an *internal* step, written $\xrightarrow[\mathrm{i}]{}$. We adapt the same notations for ordinary reduction $\rightarrow$ to standard and internal steps, such as standard reduction of a specific subterm $\overset{\gamma}{\mapsto}$, the reflexive, transitive closure of internal reduction $\xrightarrow[\mathrm{i}]{}\!\!\!\!\rightarrow$, and so forth. We will also combine the notations for internal and standard reduction with the notation for developments, *e.g.* writing $\xrightarrow[\texttt{i-dev}]{}$ for a development consisting only of internal steps. ◊

The first two clauses of the definition are usually easy, if not trivial, to show; the final clause tends to be more difficult.

We define the standard reduction relation in terms of evaluation contexts and standard reduction rules. Evaluation contexts indicate where redexes should be found; standard reduction rules can restrict redexes to specific terms. In the call-by-name calculus, we have two forms of evaluation context E. First, every top-level standard redex should be contracted, so a hole $[\,]$ by itself is an evaluation context (it is hard to imagine a system where the hole by itself would not be an evaluation context). Evaluation contexts can also take the form $E\ M$. Since evaluation contexts do not form abstractions when a top-level redex Δ fills the hole, a term matching the form $E[\Delta]$ is not itself a top-level $\beta\rightarrow$ redex. This form means, then, that the evaluation strategy is to evaluate on the left-hand side of the application until we do have a top-level redex to contract.

For call-by-name standard reduction rules, we just use the general reduction rules again. In some later systems, we will restrict the internal form of standard redexes to remove possible ambiguities in evaluation order.

Finally, for call-by-name evaluation we take answers to be abstractions. Alternately, one can consider a larger language with constants and primitive functions, or with various combinations of datatypes; here we favor the brevity of the minimal formulations, and address extensions in the closing comments.

It is fairly straightforward to show all three clauses of Definition 2.1.15 for the call-by-name reduction relations. The first two follow from what is often called a *unique evaluation contexts lemma*:

Lemma 2.1.16. Let $M \in \mathsf{Name}$. Then exactly one of the following is true:

1. M is an answer.
2. There exists some evaluation context E and variable $x \in \mathrm{fv}(M)$ such that $M \equiv E[x]$.
3. There exists some evaluation context E and top-level redex Δ such that $M \equiv E[\Delta]$.

Proof: By structural induction on M. We consider its top-level configuration.

- $M \equiv x$. Clause 2 applies with $E \equiv [\,]$, and no other clause applies since a free variable is neither an answer nor a redex.
- $M \equiv \lambda x.\, M_0$. Clause 1 applies with $E \equiv [\,]$. No other clauses apply since $\lambda x.\, [\,]$ is not an evaluation context.
- $M \equiv (M_0\ M_1)$. We consider M_0 by the induction hypothesis.

  If Clause 1 applies to M_0 and so it is an abstraction, then M is a top-level redex. That is, we have Clause 3 with $E \equiv [\,]$. Clause 1 does not apply to M since no application is an answer, and Clause 2 does not apply since we cannot form any application context to look within M_0, under the abstraction.

  If Clause 2 applies to M_0, with $M_0 \equiv E_0[x]$, then clearly Clause 2 also applies to M, with $E \equiv E_0\ M_1$. Again, no application is an answer, and so Clause 1 does not apply; by the induction hypothesis no other clause can apply to M_0, and so none could apply to M either.

  If Clause 3 applies to M_0, then it also applies to M; the construction is the same as in the preceding paragraph.

♦

For the final clause of the standardization definition, we will use the general technique outlined presently, which relies in an essential way on joinability, marking redexes and the strengthened finiteness of development results. We can assume by Lemma 2.1.20 a reduction relation R with notation as usual and satisfying the following properties, along with a reduction-closed syntactic class of answers A:

P0. All R-developments are finite and can be extended to a complete development, all complete R-developments of the same term and marking end in the same term, and R has joinable markings with joining operator $+$.

P1. If $M \underset{\mathrm{i}}{\rightarrow} A$ then M is also an answer.

P2. If $\sigma : M \twoheadrightarrow N_0$ and $M \xrightarrow{\gamma} N_1$, both by internal steps, then every redex in γ/σ is also internal.

P3. If $M \xrightarrow{\gamma} N$ is internal and $N \mapsto N_0$, then M has a standard redex.

P4. If $\sigma : M \twoheadrightarrow N_0$ is internal and $M \overset{\gamma}{\mapsto} N_1$, then γ/σ contains a single element which is also the standard redex of N_0.

Although the techniques we will use for demonstrating these properties for the various systems we will show will vary from system to system, the properties are sufficient to show the stanrdiztion result we seek.

First, we can divide every complete development into a standard sequence followed by an internal sequence.

Lemma 2.1.17. Let R satisfy **P0** and **P2**, and let $(M, \bar{\gamma}) \underset{\mathrm{cpl}}{\longrightarrow} N$ be a complete R-development. Then there exists some $(M_0, \bar{\gamma}_0)$ such that:

- $\sigma : (M, \bar{\gamma}) \underset{\mathrm{dev}}{\twoheadrightarrow} (M_0, \bar{\gamma}_0)$ is an R-development.
- $|\sigma| : M \mapsto\!\!\!\!\twoheadrightarrow M_0$ is a standard R-reduction sequence.
- Every complete R-development of $(M_0, \bar{\gamma}_0)$ contracts only internal R-redexes.

Proof: We can contract standard redexes while they exist; this sequence must be finite by **P0**. When we have run out of standard steps, all completing sequences will be internal by **P2**. ⧫

We now show that an internal complete development followed by a standard step can be followed by a standard sequence followed by another internal complete development.

Lemma 2.1.18. Let R satisfy **P0**, **P2**, **P3** and **P4**, and let $\sigma : (M, \bar{\gamma}) \xrightarrow[\text{i-cpl}]{} M_0$ and $M_0 \overset{\delta}{\mapsto} N$. Then there exists some $(N_0, \bar{\delta}_0)$ such that:

- $\tau : M \mapsto\!\!\!\rightarrow N_0$.
- $\bar{\gamma}/\tau = \bar{\delta}_0$.
- $(N_0, \bar{\delta}_0) \xrightarrow[\text{i-cpl}]{} N$.

Proof: By inductive application of **P3** and **P4** we have a redex δ_1 such that $\delta_1/\sigma = \delta$. By **P0** we can consider the marking $\bar{\gamma} + \{\delta_1\}$, from which we can construct the desired sequence by Lemma 2.1.17 since R also satisfies **P2**. ⧫

Lemma 2.1.19. Let R satisfy **P0**, **P2**, **P3** and **P4**, and let $M \underset{\text{i}}{\to} M_0 \mapsto\!\!\!\rightarrow N$. Then there exists some N_0 such that $M \mapsto\!\!\!\rightarrow N_0 \underset{\text{i}}{\twoheadrightarrow} N$.

Proof: The single step $M \underset{\text{i}}{\to} M_0$ is a complete development of one redex, so the result follows by induction on the length of the sequence $M_0 \mapsto\!\!\!\rightarrow N$ by applying Lemma 2.1.18. ⧫

Now we can perform induction on the number of internal steps in a sequence to deduce standardization from the five properties.

Lemma 2.1.20. Let R satisfy **P0** through **P4**, and let $M \twoheadrightarrow A$. Then there exists some A_0 such that $M \mapsto\!\!\!\rightarrow A_0$ and $A_0 \twoheadrightarrow A$.

Proof: We can consider the number of internal redexes in $M \twoheadrightarrow A$. Our base case is zero internal steps, at which point the result is immediate. Otherwise we consider the last of the n internal steps,

$$M \xrightarrow[\mathsf{Name}]{} \!\!\!\!\rightarrow M_0 \xrightarrow[\mathsf{i\text{-}Name}]{} M_1 \overset{}{\underset{\mathsf{Name}}{\mapsto\!\!\!\rightarrow}} A \ .$$

By Lemma 2.1.19, we have $M_0 \mapsto\!\!\!\rightarrow N_1 \underset{\text{i}}{\twoheadrightarrow} V$. By **P1**, N_1 is also some answer A_0. Then the reduction sequence $M \twoheadrightarrow M_0 \mapsto\!\!\!\rightarrow A_0$ has fewer than n internal steps, and by the induction hypothesis the result follows. ⧫

Although the deduction of these properties do vary from system to system, the basic approach to each is largely the same. The results for Name are representative, if simpler than in later systems. We already have **P0** from FD!(Name) and the discussion above. We will use Barendregt's style of proof for the remaining properties, which for Name are in his Lemma 11.4.3.

Lemma 2.1.21. *Call-by-name reduction satisfies **P1**:* Let $M \xrightarrow[\mathsf{i\text{-}Name}]{} N$. Then M is an abstraction if and only if N is an abstraction.

Proof: Internal redexes cannot be top-level, so M and N will have the same top-level structure, *i.e.* both will be abstractions or both will be applications. ⧫

Lemma 2.1.22. *Call-by-name reduction satisfies **P2**:* If $\sigma : M \xrightarrow[\mathsf{i\text{-}Name}]{} N_0$ and $M \xrightarrow[\mathsf{Name}]{\gamma} N_1$, both by internal steps, then every redex in γ/σ is also internal.

Proof: Here it is useful to consider the grammar of *speculative* contexts:

$$H ::= \lambda x.\, C \mid H\ M \mid M\ C \ .$$

Clearly, evaluation and speculative contexts partition the space of all contexts, so internal redexes are just those under a speculative context. It is moreover clear that reduction of redexes within a speculative context will produce a result again within a speculative context, and so the result is clear. ⧫

Lemma 2.1.23. *Call-by-name reduction satisfies **P3**:* If $M \xrightarrow{\gamma} N$ is internal and $N \mapsto N_0$, then M has a standard redex.

Syntactic domains As for Name, except:

$$\text{Evaluation Contexts} \qquad E \quad ::= \quad [\,] \mid E\,M \mid (\lambda x.M)\,E$$

Typing judgments As for Name.

Reduction relation

$$(\beta{\to}_v) \qquad (\lambda x.\,M)\,V \quad \to \quad M[x := V]$$

Standard reduction relation

$$(\beta{\to}_v) \qquad (\lambda x.\,M)\,V \quad \mapsto \quad M[x := V]$$

Figure 2.4: The call-by-value lambda calculus Val.

Proof: So we have $M \equiv H[\Delta] \underset{\mathrm{i}}{\to} H[\Delta'] \equiv N$, and we proceed by structural induction on H.

- $H \equiv H_0\,M_1$. If $H_0 \equiv \lambda y.\,C_0$ then the standard step for both N and M is top-level. Otherwise we have the result by the induction hypothesis on H_0.

- $H \equiv M_0\,C_1$. Again if M_0 is an abstraction, then both standard steps are top-level; otherwise both standard steps are the same and within M_0.

We cannot have $H \equiv \lambda x.\,C_0$ since then neither M nor N could have a standard redex, and the proof is complete. ♦

Lemma 2.1.24. *Call-by-name reduction satisfies **P4**:* If $\sigma : M \to N_0$ is internal and $M \overset{\gamma}{\mapsto} N_1$, then γ/σ contains a single element which is also the standard redex of N_0.

Proof: So we have

$$\begin{array}{rcl} \sigma : M \equiv H_0[\Delta_0] & \xrightarrow[\mathsf{i\text{-}Name}]{} & H_0[\Delta_0'] \equiv N_0 \\ M \equiv E_1[\Delta_1] & \overset{\gamma}{\underset{\mathsf{Name}}{\longmapsto}} & E_1[\Delta_1'] \equiv N_1 \end{array}$$

and we must construct the appropriate E, Δ such that $N_0 \equiv E[\Delta]$.

If τ is top-level and $E_1 \equiv [\,]$, then either $H_0 \equiv (\lambda x.\,C)\,L$ or $H_0 \equiv (\lambda x.\,L)\,C$, and in either case the unique residual of γ is again the top-level redex N_0 itself.

Otherwise we have $E_1 \equiv (E_2\,M_1)$. If $H_0 \equiv (H_1\,M_1)$ then the result follows by the induction hypothesis on $E_2[\Delta_1] \equiv H_1[\Delta_0]$. Otherwise it must be the case that $H_0 \equiv (E_2[\Delta_1]\,C)$, and contraction of Δ_1 within C clearly does not tamper with the standard step. ♦

Now the standardization result for call-by-name is immediate:

Lemma 2.1.25. Let $M \xrightarrow[\mathsf{Name}]{}\!\!\!\!\to V$; then $M \underset{\mathsf{Name}}{\longmapsto}\!\!\!\!\to V_0$ and $V_0 \xrightarrow[\mathsf{Name}]{}\!\!\!\!\to V$.

Proof: By Lemma 2.1.20, with the results above. ♦

Proposition 2.1.26, $\mathrm{Std}(\mathsf{Name})$**.** The relation $\underset{\mathsf{Name}}{\longmapsto}$ is a standard strategy for reduction by $\xrightarrow[\mathsf{Name}]{}$.

Proof: By Lemma 2.1.16 and Lemma 2.1.25. ♦

2.2 The call-by-value lambda calculus

Figure 2.4 reviews the call-by-value lambda calculus Val. Types, terms and values are as in the call-by-name calculus. The syntactic properties of Val are well-known, and their proofs are essentially the same as in the call-by-name system above, so we do not repeat the details of those results.

Syntactic domains As in Val, plus the following:

$$\text{Terms} \qquad L', M', N' \quad ::= \quad \cdots \mid {}^{[\beta\to]}(\lambda x.\, M')\, V'$$

Reduction relation

$$\begin{array}{llll} (\beta{\to}_{v0}) & {}^{[\beta\to]}(\lambda x.\, M')\, V' & \to & M'[x := V'] \\ (\beta{\to}_{v1}) & (\lambda x.\, M')\, V' & \to & M'[x := V'] \end{array}$$

Figure 2.5: The marked call-by-value calculus Val'.

Lemma 2.2.1. Let $\Gamma, x : B \vdash_{\mathsf{Val}} M : A$ and $\Delta \vdash_{\mathsf{Val}} N : B$. Then

$$\Gamma, \Delta \vdash_{\mathsf{Val}} M[x := N] : B \ .$$

♦

Lemma 2.2.2. Let $L, M, N, V \in \mathsf{Val}$, where $M \xrightarrow[\mathsf{Val}]{} N$. Then:

1. $L[x := M] \twoheadrightarrow_{\mathsf{Val}} L[x := N]$.
2. $M[x := V] \xrightarrow[\mathsf{Val}]{} N[x := V]$.

The restriction to values in the second clause reflects the possible dependence of the reduction rules on a free variable: if we have

$$(\lambda x.\, M)\, y \to M[x := y] \ ,$$

then reduction would no longer hold if we substituted a non-value for y. ♦

The subject reduction result is exactly as in call-by-name, since the terms and typing rules remain unchanged.

Proposition 2.2.3, $\mathrm{SR}(\mathsf{Val})$**.** Val satisfies subject reduction. ♦

One can construct the marked call-by-value reduction relations as for call-by-name; we show the syntax and rules in Figure 2.5 and assume notation analogous to that for Name'. Results on Val' follow almost exactly as in Name' since $(\beta{\to}_{v0})$ is just a restricted form of $(\beta{\to}_0)$. Strong normalization of $(\beta{\to}_{v0})$ is immediate since $(\beta{\to}_{v0})$ is a strict subset of $(\beta{\to}_0)$. For confluence we can first show a substitution result *à la* Lemma 2.1.6, with weak confluence the following and with strong normalization implying full confluence.

Proposition 2.2.4, $\mathrm{FD!}(\mathsf{Val})$**.** All Val-developments are finite and can be extended to complete developments, and all complete Val-developments of the same terms and marking end in the same term. ♦

Once again, the joining operator for Val-markings is just set union.

Proposition 2.2.5, $\mathrm{CR}(\mathsf{Val})$**.** Val is confluent. ♦

With $\mathrm{FD!}(\mathsf{Val})$ and the joining operator we have **P0**. As for call-by-name, we take call-by-value answers to be abstractions; **P1** is clear since only a top-level standard step can take a non-answer to an answer, and no $(\beta{\to}_v)$ step takes an answer to a non-answer. For the remaining three properties we use the same structural induction techniques as for call-by-name, with speculative contexts

$$H \ ::= \ \lambda x.\, C \mid H\ M \mid (\lambda x.\, M)\ H \mid M\ N\ C \mid x\ C \ .$$

Standardization is then clear.

Proposition 2.2.6, $\mathrm{Std}(\mathsf{Val})$**.** $\xmapsto[\mathsf{Val}]{}$ is a standard strategy for $\xrightarrow[\mathsf{Val}]{}$ reduction to answers. ♦

Plotkin has presented proofs of similar call-by-value confluence and standardization results using other techniques [62].

Chapter 3

Generalizing call-by-value

CALL-BY-VALUE provides a simple contrast of parameter-passing mechanism to call-by-name, but it is difficult to reason with call-by-value by itself. For transformations into the continuation-passing style, for example, one must include a number of administrative reduction steps to achieve satisfactory results [62, 64, 67, for example]. We generalize call-by-value into the *call-by-let* calculus by adding explicit let-bindings which model sharing of common subterms. We have previously studied this system in work with Odersky, Turner and Wadler [51, 52]; call-by-let is a subset of the call-by-need calculus which we study in the next chapter. Here we give new proofs for a number of the previous results, and extend the system with a notion of standard reduction.

This chapter might seem rather detailed for what is arguably a relatively obscure calculus. The techniques we use for proving properties in subsequent systems first emerge here, and we spend some detail making these techniques clear. We begin with the basic properties of the system: substitutions in Section 3.1, subject reduction in Section 3.2. We present a finiteness of developments result in Section 3.3. The statement of strengthened finiteness in this section requires a restriction on marking for unique completions, but is sufficient to show confluence (Section 3.4) and establish a standard evaluation strategy (Section 3.5). In addition to marking redexes, it is also useful to mark let-bindings which are not necessarily redexes: we study such a scheme in Section 3.6, and apply it to show that the extension of call-by-value to call-by-let is in fact conservative. Finally in Section 3.8 we present a method for removing the restriction on markings which we impose in Section 3.3 while at the same time preserving the uniqueness of completions. While not necessary immediately, the technique will be useful in later chapters, particularly for the call-by-need and affine systems.

Figure 3.1 defines the call-by-let lambda calculus Let. The terms are as in the previous call-by-name and call-by-value systems with the addition of a let-binding construct, and the notion of values as abstractions remain unchanged. The types and typing rules are also the same, with the addition of the obvious rule for let-bindings. Reduction for Let is defined by the four rules (I, V, C, A), which abbreviate *introduce*, *value*, *commute* and *associate*.

3.1 Substitutions

We will make use of two results on substitution in Let. Lemma 3.1.1, the first result, is the observation that substitution preserves types; it is effectively just cut elimination for minimal logic. The second result, Lemma 3.1.2, is a similar statement for reduction.

Lemma 3.1.1. Let $\Gamma \vdash_{\mathsf{Let}} N : A$ and $\Delta, x : A \vdash_{\mathsf{Let}} M : B$. Then

$$\Gamma, \Delta \vdash_{\mathsf{Let}} M[x := N] : B \ .$$

Proof: By a simple induction on the typing proof of N. ♦

Lemma 3.1.2. Let $M \to N$. Then:

1. $M[x := V] \to N[x := V]$.
2. $L[x := M] \twoheadrightarrow L[x := N]$.

Proof: Both of these clauses follow by a straightforward induction over the structure of terms, just as for Name, although in the first clause it is important to note the restriction to variables as $M[x := L] \to N[x := L]$ does not hold in general, for example when $L \equiv (I\ I)$, $M \equiv \text{let } y = x \text{ in } (y\ M_0)$ and $N \equiv x\ M_0$. ♦

Syntactic domains As for Val, plus the following:

$$\text{Terms} \qquad L, M, N \quad ::= \quad \cdots \mid \text{let } x = M \text{ in } N$$

Typing judgments As for Val, plus the following:

$$\dfrac{\Gamma \vdash M : A \qquad \Delta,\, x : A \vdash N : B}{\Gamma,\, \Delta \vdash \text{let } x = M \text{ in } N : B}\,\text{Let}$$

Reduction relation

(I)	$(\lambda x.\, M)\ N$	$\to$	$\text{let } x = N \text{ in } M$
(V)	$\text{let } x = V \text{ in } M$	$\to$	$M[x := V]$
(C)	$(\text{let } x = L \text{ in } M)\ N$	$\to$	$\text{let } x = L \text{ in } (M\ N)$
(A)	$\text{let } y = (\text{let } x = L \text{ in } M) \text{ in } N$	$\to$	$\text{let } x = L \text{ in } (\text{let } y = M \text{ in } N)$

Figure 3.1: The call-by-let calculus Let.

3.2 Subject reduction

The subject reduction result follows directly from the substitution results.

Proposition 3.2.1. Let satisfies subject reduction.

Proof: Subject reduction can be verified by structural induction on the contractum. We consider two representative cases.

- *Reduction by* (V) *steps.* We take $\text{let } x = V \text{ in } M \to M[x := V]$, where

$$\dfrac{\begin{array}{c}\vdots\, \mathcal{P}_1 \\ \Gamma \vdash V : A\end{array} \quad \begin{array}{c}\vdots\, \mathcal{P}_2 \\ \Delta, x : A \vdash M : B\end{array}}{\text{let } x = V \text{ in } M : B}\,\text{Let} \quad .$$

  Then by Lemma 3.1.1 with $\mathcal{P}_1$ and $\mathcal{P}_2$ we have $M[x := V] : B$ as desired.

- *Reduction by* (C) *steps.* We take

$$(\text{let } x = L \text{ in } M)\ N \to \text{let } x = L \text{ in } (M\ N) \ ,$$

  where

$$\dfrac{\dfrac{\begin{array}{c}\vdots\, \mathcal{P}_1 \\ \Gamma \vdash L : A_0\end{array} \quad \begin{array}{c}\vdots\, \mathcal{P}_2 \\ \Delta, x : A_0 \vdash M : A \to B\end{array}}{\Gamma, \Delta \vdash \text{let } x = L \text{ in } M : A \to B}\,\text{Let} \qquad \begin{array}{c}\vdots\, \mathcal{P}_3 \\ \Phi \vdash N : A\end{array}}{\Gamma, \Delta, \Phi \vdash (\text{let } x = L \text{ in } M)\ N : B}\,\to\text{-E}$$

  where the $\mathcal{P}_i$ hold by the inductive hypothesis. Then we can use the $\mathcal{P}_i$ for:

$$\dfrac{\begin{array}{c}\vdots\, \mathcal{P}_1 \\ \Gamma \vdash L : A_0\end{array} \quad \dfrac{\begin{array}{c}\vdots\, \mathcal{P}_2 \\ \Delta, x : A_0 \vdash M : A \to B\end{array} \quad \begin{array}{c}\vdots\, \mathcal{P}_3 \\ \Phi \vdash N : A\end{array}}{\Delta, \Phi, x : A_0 \vdash M\ N : B}\,\text{Let}}{\Gamma, \Delta, \Phi \vdash \text{let } x = L \text{ in } (M\ N) : B}\,\to\text{-E} \quad .$$

  Irritating complications can arise from inconveniently placed Weakening and Contraction rules. For example we might have the situation in the first tree in Figure 3.2. However it is clear that we can obtain an equivalent typing by moving the interfering steps towards the bottom of the tree, as the lower tree of that figure shows. The manipulation then follows as before.

The reasoning for (I, A) steps is as for (C) steps. ♦

Before

$$
\dfrac{
  \dfrac{
    \dfrac{\begin{matrix}\vdots\,\mathcal{P}_1\\ \Gamma, z_1 : B_1 \vdash L : A_0\end{matrix} \quad \begin{matrix}\vdots\,\mathcal{P}_2\\ \Delta, z_2 : B_1, x : A_0 \vdash M : A \to B\end{matrix}}{\Gamma, \Delta, z_1 : B_1, z_2 : B_1 \vdash \text{let } x = L \text{ in } M : A \to B}\,\text{Let}
  }{\Gamma, \Delta, z_0 : B_1 \vdash \text{let } x = L[z_1 := z_0] \text{ in } M[z_2 := z_0] : A \to B}\,\text{C}
  \quad
  \dfrac{\begin{matrix}\vdots\,\mathcal{P}_3\\ \Phi \vdash N : A\end{matrix}}{\Phi, z_3 : B_2 \vdash N : A}\,\text{W}
}{\Gamma, \Delta, \Phi, z_0 : B_1, z_3 : B_2 \vdash (\text{let } x = L[z_1 := z_0] \text{ in } M[z_2 := z_0])\; N : B}\,\to\text{-E}
$$

After

$$
\dfrac{
  \dfrac{
    \dfrac{
      \dfrac{\begin{matrix}\vdots\,\mathcal{P}_1\\ \Gamma, z_1 : B_1 \vdash L : A_0\end{matrix} \quad \begin{matrix}\vdots\,\mathcal{P}_2\\ \Delta, z_2 : B_1, x : A_0 \vdash M : A \to B\end{matrix}}{\Gamma, \Delta, z_1 : B_1, z_2 : B_1 \vdash \text{let } x = L \text{ in } M : A \to B}\,\text{Let}
      \quad
      \begin{matrix}\vdots\,\mathcal{P}_3\\ \Phi \vdash N : A\end{matrix}
    }{\Gamma, \Delta, \Phi, z_1 : B_1, z_2 : B_1 \vdash (\text{let } x = L \text{ in } M)\; N : B}\,\to\text{-E}
  }{\Gamma, \Delta, \Phi, z_1 : B_1, z_2 : B_1, z_3 : B_2 \vdash (\text{let } x = L \text{ in } M)\; N : B}\,\text{Weakening}
}{\Gamma, \Delta, \Phi, z_0 : B_1, z_3 : B_2 \vdash (\text{let } x = L[z_1 := z_0] \text{ in } M[z_2 := z_0])\; N : B}\,\text{Contraction}
$$

Figure 3.2: Propagation of Weakening and Contraction steps for the subject reduction result for (C) steps.

3.3 Finiteness of developments

We will present a number of marked variants of Let. With the first, Let', we follow the example of classical marked calculi and mark individual Let redexes. Developments based on this system are finite, but we have unique completions only for markings which satisfy a certain restriction called *well-separation*, which we introduce in Section 3.3.3. While we can establish confluence and standardization under this restriction with only slight extra effort, it will be very useful in the next chapter to omit the restriction, and so we will consider a generalization of call-by-let in Section 3.8

3.3.1 Marked call-by-let reduction

Figure 3.3 presents the terms and basic reduction rules, and Figure 3.4 the compatible closure rules, of the marked Let calculus Let'. We use the same notational conventions for Let' as in Let. Moreover, following Barendregt's convention we take Let'_0 to be the restriction to marked reduction only, and Let'_1 to be the restriction to unmarked reduction only. We define the unsubscripted rule to be the union of the subscripted rules, *e.g.* $(I) = (I_0, I_1)$. We let σ', τ' range over Let' reduction sequences. The remarking operator $\urcorner$ manipulates contexts used in compatible closure to make sure that the term under every mark is in fact a redex: (C, A) contractions rely on having a binding at certain points within the redex, while (V) steps remove bindings without necessarily replacing them with another. Without remarking, we might have sequences like

$$ {}^{[C]}({}^{[V]}\text{let } x = V \text{ in } \lambda y.\, x)\; M \xrightarrow[(V_0)]{} {}^{[C]}(\lambda y.\, V)\; M $$

or

$$ {}^{[C]}(\text{let } x = V \text{ in } \lambda y.\, x)\; M \xrightarrow[(V_1)]{} {}^{[C]}(\lambda y.\, V)\; M \quad , $$

which, aside from not being part of the term language, makes no sense. Moreover, application of these commuting rules simply seeks to remove bindings from a position which blocks other contractions: with no more binding to move, it is appropriate that the tag disappear. Since it is no longer trivial that reduction preserves membership in the grammar of terms, we must explicitly prove the result.

Lemma 3.3.1. Terms $M' \in \mathsf{Let}'$ are closed under reduction $\xrightarrow[\mathsf{Let}']{}$.

Proof: The case of top-level reduction is clear by inspection of the reduction rules. For compatible closure, the result follows by a straightforward structural induction which transparently relies on remarking. ♦

Syntactic domains As in Let, plus with the following:

$$
\begin{array}{llrl}
\text{Values} & V', W' & ::= & x \mid \lambda x.\, M' \\
\text{Terms} & L', M', N' & ::= & V' \mid M'\, N' \mid P\, M' \\
 & & \mid & {}^{[I]}(\lambda x.\, M')N' \\
 & & \mid & {}^{[C]}(P\, M')\, N' \\
\text{Let bindings} & P & ::= & \text{let } x = M' \text{ in} \\
 & & \mid & {}^{[V]}\text{let } x = V' \text{ in} \\
 & & \mid & {}^{[A]}\text{let } x = P\, M' \text{ in} \\
\text{Contexts or terms} & Q' & ::= & C' \mid M'
\end{array}
$$

Reduction rules

$$
\begin{array}{lllll}
(I_0) & {}^{[I]}(\lambda x.\, M')\, N' & \to & \text{let } x = N' \text{ in } M' \\
(I_1) & (\lambda x.\, M')\, N' & \to & \text{let } x = N' \text{ in } M' \\
(V_0) & {}^{[V]}\text{let } x = V' \text{ in } M' & \to & M'[x := V'] \\
(V_1) & \text{let } x = V' \text{ in } M' & \to & M'[x := V'] \\
(C_0) & {}^{[C]}(P\, M')\, N' & \to & P\,(M'\, N') \\
(C_1) & (P\, M')\, N' & \to & P\,(M'\, N') \\
(A_0) & {}^{[A]}\text{let } x = (P\, M') \text{ in } N' & \to & P\,(\text{let } x = M' \text{ in } N') \\
(A_1) & \text{let } x = (P\, M') \text{ in } N' & \to & P\,(\text{let } x = M' \text{ in } N')
\end{array}
$$

Figure 3.3: Syntax and basic rules of the marked call-by-let calculus Let'.

Remarking operator

$$
\begin{array}{rcl}
[\,]^{\urcorner} & \equiv & [\,] \\
M'^{\urcorner} & \equiv & M' \\
(\lambda x.\, C')^{\urcorner} & \equiv & \lambda x.\, C'^{\urcorner} \\
(Q'_0\, Q'_1)^{\urcorner} & \equiv & (Q_0'^{\urcorner})\,(Q_1'^{\urcorner}) \\
(\text{let } x = Q'_0 \text{ in } Q'_1)^{\urcorner} & \equiv & \text{let } x = Q_0'^{\urcorner} \text{ in } Q_1'^{\urcorner} \\
({}^{[I]}(\lambda x.\, Q'_0)\, Q'_1)^{\urcorner} & \equiv & {}^{[I]}(\lambda x.\, Q_0'^{\urcorner})\,(Q_1'^{\urcorner}) \\
({}^{[V]}\text{let } x = Q'_0 \text{ in } Q'_1)^{\urcorner} & \equiv & {}^{[V]}\text{let } x = Q_0'^{\urcorner} \text{ in } Q_1'^{\urcorner} \\
({}^{[C]}[\,]\, M')^{\urcorner} & \equiv & [\,]\, M' \\
({}^{[C]}Q'_0\, Q'_1)^{\urcorner} & \equiv & {}^{[C]}Q_0'^{\urcorner}\, Q_1'^{\urcorner}, \quad Q'_0 \not\equiv [\,] \\
({}^{[A]}\text{let } x = [\,] \text{ in } M')^{\urcorner} & \equiv & \text{let } x = [\,] \text{ in } M' \\
({}^{[A]}\text{let } x = Q'_0 \text{ in } Q'_1)^{\urcorner} & \equiv & {}^{[A]}\text{let } x = Q_0'^{\urcorner} \text{ in } Q_1'^{\urcorner}, \quad Q'_0 \not\equiv [\,]
\end{array}
$$

Compatible closure

$$
\frac{\Delta_0 \xrightarrow[(V)]{} \Delta_1 \text{ (top-level)}}{C[\Delta_0] \xrightarrow[(V)]{} C^{\urcorner}[\Delta_1]}
\qquad\qquad
\frac{\Delta_0 \xrightarrow[(I,C,A)]{} \Delta_1}{C[\Delta_0] \xrightarrow[(I,C,A)]{} C[\Delta_1]}
$$

Figure 3.4: Compatible closure in the marked call-by-let calculus Let'.

The usual substitution results continue to hold.

Lemma 3.3.2. Let $M' \xrightarrow[\mathsf{Let}']{} N'$. Then:

1. $M'[x := V'] \xrightarrow[\mathsf{Let}']{} N'[x := V']$.
2. $L'[x := M'] \twoheadrightarrow_{\mathsf{Let}'} L'[x := N']$.

Proof: Trivial, again noting the restriction to values in the first clause. ♦

We can make a slightly stronger statement as well regarding marked redexes.

Lemma 3.3.3. Let $M' \xrightarrow[\mathsf{Let}'_0]{} N'$. Then:

1. $M'[x := V'] \xrightarrow[\mathsf{Let}'_0]{} N'[x := V']$.
2. $L'[x := M'] \twoheadrightarrow_{\mathsf{Let}'_0} L'[x := N']$.

Proof: Trivial, since the substitution will not disrupt markings and the steps they require. ♦

We can relate Let' terms to Let terms by simply erasing the marks.

Definition. Let $M' \in \mathsf{Let}'$. Then $|M'|$ is the term formed by removing all markings from M'. Moreover, if we take

$$\sigma' : M' \to M'_0 \to M'_1 \cdots \quad ,$$

then $|\sigma'|$ is the reduction sequence $|M'| \to |M'_0| \to |M'_1| \cdots$. ♢

Lemma 3.3.4. Let $M' \in \mathsf{Let}'$, then $|M'| \in \mathsf{Let}$. Let σ be a Let' reduction sequence; then $|\sigma|$ is a Let reduction sequence.

Proof: Trivial. ♦

Lemma 3.3.5. Let $M' \in \mathsf{Let}'$. If $|M'| \to N$, then there exists some $N' \in \mathsf{Let}'$ such that $M' \to N'$ and $N \equiv |N'|$; moreover this N' is unique.

Proof: For top-level redexes the result is obvious from the statements of the reduction rules: we first observe that Let'_1 reduction is the same as in Let but for marked subterms, and Let'_0 differs only by the outermost mark; then depending on whether the clause is marked, we contract M' by either the marked or unmarked Let' rule. Marks within terms are not an issue, in particular since the statement of rules with inner bindings — namely the (C_0, C_1, A_0, A_1) rules — are quantified over all prefixes P, marked and unmarked. With the remarking operator we can wrap the initial redex in arbitrary contexts whose projections under $|\,.\,|$ are equivalent, and so the result follows. Uniqueness is clear since for any contraction in the unmarked term, only either the 0- or 1-subscribed rule will correspond to the marked redex. ♦

We again use marked reduction to trace specific redexes as they are duplicated and moved by other contractions. We extend the alphabet of steps with new symbols $\underline{\ell 1}$ and $\underline{\ell 2}$ for respectively the bound and free subterms of a let-binding. We use the same metanotation as before: steps s, t; paths $\gamma, \delta, \varphi, \psi, \rho$; sets of paths $\bar{\gamma}$ and so forth. We have the usual identification between marked terms M' and pairs $(M, \bar{\gamma})$, as well as notions of residual, development and finite and complete developments, and keep the usual notation for all of the above.

The following two lemmas are trivial. The first formalizes the obvious link between developments and marked reduction, and the second lets us identify the reduction rule by which residuals are contracted.

Lemma 3.3.6. σ is a development of (M, γ) if and only if the lifting of σ to σ' as per Lemma 3.3.5 is a Let'_0 sequence. ♦

Lemma 3.3.7. Let $M \xrightarrow[\mathsf{Let}]{\gamma} N$ where γ is an R-redex for some Let-reduction rule R, and let σ be a Let-reduction sequence starting from M. Then every redex in γ/σ is an R-redex. ♦

3.3.2 Simple finiteness of developments

We consider two statements of finite developments for Let, one applicable to all terms, the other stronger but applicable only to a subset of terms. The weaker statement is just strong normalization of marked reduction, which we prove in this section. The stronger statement, which we prove in the next section, requires moreover that the result of all complete developments of the same term be unique.

To show Let'_0 to be strongly normalizing we consider a *weighting* of the variables in a term. We weigh variables in Let' through the system $\mathsf{Let}^\bullet$ by adding natural numbers as weights to non-binding occurrences of variables, with $\dot{L}, \dot{M}, \dot{N}$ ranging over $\mathsf{Let}^\bullet$ terms and $\dot{V}, \dot{W}$ over $\mathsf{Let}^\bullet$ values. We define the norm of a $\mathsf{Let}^\bullet$ term as follows, disregarding any markings on redexes:

$$\begin{array}{rcl} \|x^i\| & = & i \\ \|\lambda x.\, \dot{M}\| & = & \|\dot{M}\| \\ \|\text{let } x = \dot{M} \text{ in } \dot{N}\| & = & 2\|\dot{M}\| + \|\dot{N}\| \\ \|\dot{M}\ \dot{N}\| & = & 2\|\dot{M}\| + 2\|\dot{N}\| \end{array}$$

The following lemma is central: if we substitute a term N for a variable in M, where all occurrences of the variable have greater weight than the norm of N, the norm of M will be less after the substitution than before.

Lemma 3.3.8. Let $\dot{M}, \dot{N} \in \mathsf{Let}^\bullet$ such that for all occurrences of x^i in $\dot{M}$ we have $\|\dot{N}\| < i$. Then $\|\dot{M}[x := \dot{N}]\| \leq \|\dot{M}\|$.

We include equality in the statement for the case where x does not oocur in $\dot{M}$.

Proof: Clear from the definitions of decreasing weighting and the norm. ♦

We are interested in a particular class of weightings, *decreasing* weightings.

Definition. A term $\dot{M} \in \mathsf{Let}^\bullet$ is said to have a decreasing weighting if both of the following conditions are true:

1. For every subterm ${}^{[I]}(\lambda x.\, \dot{M}_0)\, \dot{M}_1$ of $\dot{M}$ and for all occurrences of x^i within $\dot{M}_0$, $\|\dot{M}_1\| < i$.
2. For all subterms $\text{let } x = \dot{M}_0 \text{ in } \dot{M}_1$ or ${}^{[V]}\text{let } x = \dot{M}_0 \text{ in } \dot{M}_1$ — the latter where $\dot{M}_0$ is of course a value — of $\dot{M}$ and for all occurrences of x^i within $\dot{M}_1$, $\|\dot{M}_0\| < i$.

We refer to terms of the three forms ${}^{[I]}(\lambda x.\, \dot{M}_0)\ \dot{M}_1$, $(\text{let } x = \dot{M}_0 \text{ in } \dot{M}_1)$ and ${}^{[V]}(\text{let } x = \dot{V}_0 \text{ in } \dot{M}_1)$ as *weight-relevant* terms. ◊

Strictly speaking, these conditions are much stronger than what we really need to show finiteness of developments, but the stronger conditions will be useful later. Specifically, it is not necessary to require that unmarked let-bindings adhere to Clause 2 in the definition for the results of this section, but we do so anyway to more easily reuse these results for system in Section 3.6.

As suggested by the name, the norm is decreased by $\mathsf{Let}^\bullet_0$ reduction of terms with a decreasing weighting. Moreover, the result of $\mathsf{Let}^\bullet_0$ reducing a term will again have a decreasing weighting.

Lemma 3.3.9. Let $\dot{M} \xrightarrow[\mathsf{Let}'_0]{} \dot{N}$, where $\dot{M}$ has a decreasing weighting. Then $\|\dot{M}\| > \|\dot{N}\|$.

Proof: By a structural induction down to the four possible contractions. We show the four base cases, and a representative of the trivial induction steps.

1. *Top level marked (I) contraction.* We have terms $\dot{M} \equiv {}^{[I]}(\lambda x.\, \dot{M}_0)\ \dot{M}_1$ and $\dot{N} \equiv \text{let } x = \dot{M}_1 \text{ in } \dot{M}_0$, and the decrease is trivial since every term has weight at least 1:

$$\begin{array}{cl} & \|{}^{[I]}(\lambda x.\, \dot{M}_0)\ \dot{M}_1\| \\ = & \|(\lambda x.\, \dot{M}_0)\ \dot{M}_1\| \\ = & 2\|\dot{M}_0\| + 2\|\dot{M}_1\| \\ > & \|\dot{M}_0\| + 2\|\dot{M}_1\| \\ = & \|\text{let } x = \dot{M}_1 \text{ in } \dot{M}_0\|\ . \end{array}$$

2. *Top level marked* (V) *contraction.* We have $\dot{M} \equiv {}^{[V]}\text{let } x = \dot{V} \text{ in } \dot{M}$ and $\dot{N} \equiv \dot{M}[x := \dot{V}]$. We can use the definition of a decreasing weighting and Lemma 3.3.8 to show that $||\dot{M}[x := \dot{V}]|| \leq ||\dot{M}||$. Then we have the result with

$$\begin{array}{rl} & ||{}^{[V]}\text{let } x = \dot{V} \text{ in } \dot{M}|| \\ = & ||\text{let } x = \dot{V} \text{ in } \dot{M}|| \\ = & 2||\dot{V}|| + ||\dot{M}|| \\ > & ||\dot{M}|| \\ \geq & ||\dot{M}[x := \dot{V}]|| \ . \end{array} \tag{3.1}$$

3. *Top level marked* (C) *contraction.* We have simply

$$\begin{array}{rl} & ||{}^{[C]}(\text{let } x = \dot{L} \text{ in } \dot{M})\ \dot{N}|| \\ = & ||(\text{let } x = \dot{L} \text{ in } \dot{M})\ \dot{N}|| \\ = & 4||\dot{L}|| + 2||\dot{M}|| + 2||\dot{N}|| \\ > & 2||\dot{L}|| + 2||\dot{M}|| + 2||\dot{N}|| \\ = & ||\text{let } x = \dot{L} \text{ in } (\dot{M}\ \dot{N})| \ . \end{array} \tag{3.2}$$

4. *Top level marked* (A) *contraction.* Again, this case is straightforward:

$$\begin{array}{rl} & ||{}^{[A]}\text{let } x = \text{let } y = \dot{L} \text{ in } \dot{M} \text{ in } \dot{N}|| \\ = & ||\text{let } x = \text{let } y = \dot{L} \text{ in } \dot{M} \text{ in } \dot{N}|| \\ = & 4||\dot{L}|| + 2||\dot{M}|| + ||\dot{N}|| \\ > & 2||\dot{L}|| + 2||\dot{M}|| + ||\dot{N}|| \\ = & ||\text{let } y = \dot{L} \text{ in let } x = \dot{M} \text{ in } \dot{N}|| \ . \end{array} \tag{3.3}$$

For non-top level reduction, the inductive step is simple since the norm of a term decreases directly with the norm of its subcomponents. Considering application as an example, let $\dot{L}\ \dot{M} \to \dot{L}\ \dot{N}$ with $||\dot{M}|| > ||\dot{N}||$, and we have

$$\begin{array}{rl} & ||\dot{L}\ \dot{M}|| \\ = & 2||\dot{L}|| + 2||\dot{M}|| \\ > & 2||\dot{L}|| + 2||\dot{N}|| \\ = & ||\dot{L}\ \dot{N}|| \ . \end{array}$$

It is clear from the definition of the norm that a similar analysis holds for all term forms. ♦

The following argument occurs twice in Lemma 3.3.11, so we present it just once right here.

Lemma 3.3.10. Let $\dot{M}, \dot{N} \in \mathsf{Let}^\bullet$ both have decreasing weighting where for all occurrences of x^i in $\dot{M}$ we have $i > ||\dot{N}||$. Then $\dot{M}[x := \dot{N}]$ has a decreasing weighting.

Proof: Let $\dot{L}$ be a weight-relevant term in $\dot{M}[x := \dot{N}]$, where either

$$\begin{array}{rcl} \dot{L} & \equiv & {}^{[I]}(\lambda y.\, \dot{L}_0)\ \dot{L}_1 \\ \text{or } \dot{L} & \equiv & {}^{[V]}\text{let } y = \dot{V} \text{ in } \dot{L}_0 \\ \text{or } \dot{L} & \equiv & \text{let } y = \dot{V} \text{ in } \dot{L}_0 \ . \end{array}$$

If the considered occurrence of $\dot{L}$ comes from $\dot{M}$, then we have the result by Lemma 3.3.8: since $||\dot{L}_1[x := \dot{N}]|| < ||\dot{L}_1||$ by the hypothesis, the condition for decreasing weighting involving y continues to hold. If the considered occurrence of $\dot{L}$ comes from $\dot{N}$, then the result is trivial since it is unchanged. ♦

Lemma 3.3.11. Let $\dot{M} \xrightarrow[\mathsf{Let}'_0]{} \dot{N}$, and let $\dot{M}$ have a decreasing weighting. Then $\dot{N}$ has a decreasing weighting.

Proof: We first consider top-level reduction, taking each of the four possible reduction rules separately, and then show validity under compatible closure. In each of the cases for the former analysis, we show that every weight-relevant subterm of $\dot{N}$ again satisfies the criteria for a decreasing weighting. This lemma corresponds to Barendregt's proof of the similar result for call-by-name [9, Lemma 11.2.18.(ii)].

1. *Marked top-level* (I) *reduction.* So $\dot{M} \equiv {}^{[I]}(\lambda y.\, \dot{M}_0)\ \dot{M}_1$ and let $\dot{L}$ be a weight-relevant subterm of $\dot{N}$.

   - If $\dot{L}$ is a marked redex, then $\dot{L}$ is necessarily the residual of a weight-relevant subterm $\dot{L}_0$ with decreasing weighting in $\dot{M}$, and moreover since (I) reduction duplicates nothing, but only moves around the two maximal, non-identical subchildren $\dot{M}_0$ and $\dot{M}_1$ of $\dot{M}$, $\dot{L} \equiv \dot{L}_0$ and the result is trivial.
   - If $\dot{L}$ is an umarked let-binding, let γ be the path indexing $\dot{L}$ in $\dot{N}$.
     - If $\gamma \equiv \underline{\ell 1}\gamma_1$ or $\gamma \equiv \underline{\ell 2}\gamma_2$, then $\dot{L}$ also occurs as a subterm of $\dot{M}$ at position respectively $\underline{@2}\gamma_1$ or $\underline{@1}\,\underline{\lambda 1}\gamma_2$, so adherence to the decreasing weighting criteria follows from $\dot{M}$ having a decreasing weighting.
     - If $\gamma \equiv \epsilon$, the the result is immediate from $\dot{M}$ having a decreasing weighting.

2. *Marked top-level* (V) *reduction.* So $\dot{M} \equiv {}^{[V]}\mathrm{let}\ x = \dot{V}\ \mathrm{in}\ \dot{M}_1$, and again let $\dot{L}$ be a weight-relevant subterm of $\dot{N} \equiv \dot{M}_1[x := \dot{V}]$.

   - If $\dot{L}$ is a marked redex, then it is necessarily the residual of a weight-relevant subterm $\dot{L}_0$ with decreasing weighting in $\dot{M}$. If $\dot{L}_0$ is a subterm of $\dot{V}$, then the result is trivial since first of all $\dot{L} \equiv \dot{L}_0$, and so itself meets the requirements for a decreasing weighting, and moreover because its norm must be less than the weighting of any variable it replaces. Otherwise, for $\dot{L}_0$ a subterm of $\dot{M}_1$, the result follows by Lemma 3.3.10.
   - If otherwise $\dot{L}$ is an unmarked let-binding, let γ be the path indexing $\dot{L}$ in $\dot{N}$. We examine γ to determine the term to which it corresponds in $\dot{M}$; in effect we are manually extending the notion of residual to cover certain non-redexes.
     - If $\gamma \equiv \gamma_0\gamma_1$ where γ_0 indexes an occurrence of x in $\dot{M}_1$, then it is clear that γ_1 indexes an occurrence of $\dot{L}$ in $\dot{V}$. Since $\dot{V}$ is a subterm of $\dot{M}$, $\dot{V}$ also has a decreasing weighting, and thus $\dot{L}$ does as well.
     - Otherwise we have that γ indexes an unmarked let-binding $\dot{L}_0$ in $\dot{M}_1$, where $\dot{L} \equiv \dot{L}_0[x := \dot{V}]$. Since $\dot{M}$ has decreasing weighting, we know that all occurrences of x within $\dot{L}_0$ have weight not less than $\|\dot{V}\|$, the result follows by Lemma 3.3.10.

3. *Marked top-level* (C) *reduction.* So $\dot{M} \equiv {}^{[C]}(P\,\dot{M}_0)\ \dot{M}_1$, and we let $\dot{L}$ be a weight-relevant subterm of $\dot{N}$. If $\dot{L}_0$ is a subterm of P, $\dot{M}_0$ or $\dot{M}_1$ then the result follows as in the analysis of the exact form of $\dot{L}$ in the case for (I_0) steps above. If on the other hand $\dot{L}_0 \equiv (P\,\dot{M}_0)$, then we have $\dot{L} \equiv P\,(\dot{M}_0\ \dot{M}_1)$; since the variable bound by P does not appear free in $\dot{M}_1$, the decreasing weighting condition on $\dot{L}$ is again satisfied.

4. *Marked top-level* (A) *reduction.* By the same reasoning as for marked top-level (C) reduction.

For a redex within $\dot{M}$ by compatible closure, we proceed by structural induction on $\dot{M}$. The inductive steps correspond to the compatible closure of the redex within non-contracted contexts. For non-weight-relevant terms the result is immediate.

- *Marked top-level* (I) *redexes.* So we have ${}^{[I]}(\lambda x.\, \dot{M}_0)\ \dot{M}_1$, with two further cases depending on which subterm is contracted.

  1. *Reduction* $\dot{M}_0 \to \dot{N}_0$. So $\dot{N} \equiv {}^{[I]}(\lambda x.\, \dot{N}_0)\ \dot{M}_1$. Let x^i occur in $\dot{N}_0$; then we must have x^i occurring in $\dot{M}_0$ as well since reduction will not create new weightings of a free variable, and by the initial conditions this $i > \|\dot{M}_1\|$.
  2. *Reduction* $\dot{M}_1 \to \dot{N}_1$. So $\dot{N} \equiv {}^{[I]}(\lambda x.\, \dot{M}_0)\ \dot{N}_1$. Let x^i occur in $\dot{M}_0$; then by the initial decreasing weighting condition we have $i > \|\dot{M}_1\|$. Moreover by Lemma 3.3.9 we have that $\|\dot{M}_1\| > \|\dot{N}_1\|$, so $i > \|\dot{N}_1\|$ and the weighting is again decreasing.

For marked top-level (V) redexes and unmarked let-bindings we can use similar reasoning as for marked top-level (I) redexes; for (C, A) steps the result is trivial, and the lemma follows. ♦

At this point we have a useful corollary about the (C, A) rules:

Corollary 3.3.12. (C, A) reduction in Let is strongly normalizing.

Proof: By the above arguments, which in the restricted case of (C, A) reduction apply to all terms with any weighting. ♦

For simple finiteness of Let-developments it remains only to show that every term does in fact have a decreasing weighting.

Lemma 3.3.13. Let $M' \in \mathsf{Let}'$. Then there exists a term $\dot{M} \in \mathsf{Let}^\bullet$, $|\dot{M}| \equiv M'$, such that $\dot{M}$ has a decreasing weighting.

Proof: Considering a term with n free variables all weighted at most m, it is clear from the definition of the norm that the maximum norm of such a term occurs in a left-nested (or equivalently, right-nested) series of applications of one such variable (or abstractions and/or let-enclosures thereof) to another, that is,

$$((\cdots((x_1^{a_1}\ x_2^{a_2})\ x_3^{a_3})\cdots)\ x_{n-1}^{a_{n-1}})\ x_n^{a_n} \quad ,$$

with each $a_i \le m$ and norm

$$\begin{aligned}
& \|((\cdots((x_1^{a_1}\ x_2^{a_2})\ x_3^{a_3})\cdots)\ x_{n-1}^{a_{n-1}})\ x_n^{a_n}\| \\
\le\ & \|((\cdots((x_1^{m}\ x_2^{m})\ x_3^{m})\cdots)\ x_{n-1}^{m})\ x_n^{m}\| \\
=\ & 2\|(\cdots((x_1^{m}\ x_2^{m})\ x_3^{m})\cdots)\ x_{n-1}^{m}\| + 2\|x_n^m\| \\
=\ & 2^2\|(\cdots((x_1^{m}\ x_2^{m})\ x_3^{m})\cdots)\ x_{n-2}^{m}\| + 2^2\|x_{n-1}^m\| + 2\|x_n^m\| \\
=\ & 2^{n-2}\|x_1^m\ x_2^m\| + 2^{n-2}\|x_3^m\| + \cdots + 2^2\|x_{n-1}^m\| + 2\|x_n^m\| \\
=\ & 2^{n-1}\|x_1^m\| + 2^{n-1}\|x_2^m\| + 2^{n-2}\|x_3^m\| + \cdots + 2^2\|x_{n-1}^m\| + 2\|x_n^m\| \\
=\ & 2^{n-1}m + 2^{n-1}m + 2^{n-2}m + \cdots + 2^2 m + 2^1 m \\
=\ & 2^{n-1}m + \sum_{n=1}^{i-1} 2^i m \\
<\ & \sum_{n=1}^{i} 2^i m
\end{aligned}$$

So to give a variable weight strictly greater than a term with no more than n variable occurrences each of weight no more than m, $(\sum_{i=0}^{n} 2^i)m$ will clearly suffice.

Correspondingly, to construct a decreasing weighting for a marked term M', we number all variable occurrences in M' beginning with 1 from right to left, except in a let-binding where we number the bound term first. We then give a term numbered i the weight f_i, where f is the recurrence relation defined by the following two equations:

$$\begin{aligned}
f_1 &= 1 \\
f_n &= \left(\sum_{i=0}^{n} 2^i\right) \cdot f_{n-1} \ , \qquad n > 1
\end{aligned} \tag{3.4}$$

By the above arguments this weighting is clearly decreasing. ♦

Proposition 3.3.14. Reduction in Let_0' is strongly normalizing.

Proof: By Lemma 3.3.13 every term $M' \in \mathsf{Let}'$ has a decreasing weighting $\dot{M}$, and so by Lemma 3.3.9 and Lemma 3.3.11 all Let_0'-sequences from M' are of length at most $\|\dot{M}\|$. ♦

Now the finiteness of developments is immediate.

Corollary 3.3.15, $\mathrm{FD}(\mathsf{Let})$**.** All Let-developments are finite.

Proof: By Lemma 3.3.6 and Proposition 3.3.14. ♦

The following result is a sort of combination of Corollary 3.3.12 and Proposition 3.3.14, and will be useful later.

Corollary 3.3.16. (I_0, V_0, C, A) reduction in Let is strongly normalizing.

Proof: Again by the above arguments which in the restricted case of (C, A) reduction apply to all terms regardless of weighting. ♦

3.3.3 Uniqueness of complete developments

We have two typical counterexamples to the strengthened finiteness of developments for this marking scheme. We first have a containing (C) step, with reductions to two distinct Let'_0-normal forms

$$\begin{array}{ll} & {}^{[C]}({}^{[A]}\text{let } x = P\,L' \text{ in } M')\,N' \\ \xrightarrow[(C_0)]{} & {}^{[A]}\text{let } x = P\,L' \text{ in } (M'\,N') \\ \xrightarrow[(A_0)]{} & P\,(\text{let } x = L' \text{ in } M'\,N') \end{array} \tag{3.5}$$

and

$$\begin{array}{ll} & {}^{[C]}({}^{[A]}\text{let } x = P\,L' \text{ in } M')\,N' \\ \xrightarrow[(A_0)]{} & {}^{[C]}(P\,(\text{let } x = L' \text{ in } M'))\,N' \\ \xrightarrow[(C_0)]{} & P\,((\text{let } x = L' \text{ in } M')\,N') \ . \end{array}$$

We also have non-confluence arise where the enclosing redex is another (A) redex:

$$\begin{array}{ll} & {}^{[A]}\text{let } x = ({}^{[A]}\text{let } y = (P\,L') \text{ in } M') \text{ in } N' \\ \xrightarrow[(A_0)]{} & {}^{[A]}\text{let } y = (P\,L') \text{ in let } x = M' \text{ in } N' \\ \xrightarrow[(A_0)]{} & P\,(\text{let } y = L' \text{ in let } x = M' \text{ in } N') \end{array} \tag{3.6}$$

and

$$\begin{array}{ll} & {}^{[A]}\text{let } x = ({}^{[A]}\text{let } y = (P\,L') \text{ in } M') \text{ in } N' \\ \xrightarrow[(A_0)]{} & {}^{[A]}\text{let } x = P\,(\text{let } y = L' \text{ in } M') \text{ in } N' \\ \xrightarrow[(A_0)]{} & P\,(\text{let } x = (\text{let } y = L' \text{ in } M') \text{ in } N') \ . \end{array}$$

To avoid this problem we consider a subset of Let' terms which do enjoy confluence of Let'_0 reduction.

Definition 3.3.17. A marked call-by-let term M' is *well-separated* if it contains no subterms of either form

$${}^{[C]}({}^{[A]}\text{let } x = P\,L' \text{ in } M')\,N'$$

or

$${}^{[A]}\text{let } x = ({}^{[A]}\text{let } y = P\,L' \text{ in } M') \text{ in } N' \ .$$

A set of redex occurrences γ in M is well-separated if (M, γ) is a well-separated Let' term. ◊

Remark: Well-separation does cause some complications in later proofs, and in Section 3.8 we will generalize the notion of marked reduction in a way which lets us do without the restriction at the cost of a more complex system. For the present system, the uniqueness result will be useful even with the restriction. We consider both systems rather than proceeding with the more general, more powerful one alone to make clearer what can be achieved with the simpler system, and what sort of results would require the more complicated marking. To be specific, the complications caused by well-separation are precisely due to the fact that it seems impossible to construct a joining operator for well-separated marking sets. The best illustrations of the difficulty are just the same examples motivating well-separation. Consider the two redexes shown marked in Equation 3.5 or 3.6: let γ_0 and γ_1 be singleton sets which mark the two redexes between them. It is clearly impossible to mark the original term such that all of the requirements for the joining — namely, both inclusion of the complete development of each γ_i as partial developments of the joining, and a unique completion of the joining — would be satisifed.

Despite the above remarks, the next lemma is clearly trivial.

Lemma 3.3.18. All (I, V)-markings are joinable, with set union as the join operator. ♦

Well-separation is clearly preserved by most reduction steps, but not by some unmarked (A) contractions. As a counterexample we have

$$\begin{array}{rl} & {}^{[C]}(\text{let } x = ({}^{[A]}\text{let } y = M_1 \text{ in } M_2) \text{ in } M_3)\ N \\ \xrightarrow[(A_1)]{} & {}^{[C]}({}^{[A]}\text{let } y = M_1 \text{ in } (\text{let } x = M_2 \text{ in } M_3))\ N \quad , \end{array}$$

which clearly does not terminate with a well-separated term.

Lemma 3.3.19. Let $M', N' \in \mathsf{Let}'$, $M' \xrightarrow[(I,V,C,A_0)]{} N'$ with M' well-separated. Then N' is well-separated.

Proof: Well-separation is based on two cases, outermost (C) and outermost (A) markings. We consider each separately.

- *Outermost (C) markings.* Let ${}^{[C]}((P\, M_1')\ M_2')$ occur in M'; by the initial well-separation we can exclude the possibility that $(P\, M_1')$ be marked as a (A) redex. There are two ways in which a disallowed marking may occur:
  1. If we had $P \equiv {}^{[V]}\text{let } x = V' \text{ in}$ then we would have a top-level (V) redex whose result (based on M_1') could be a marked (A) redex. But the contraction of ${}^{[V]}\text{let } x = V' \text{ in } M_1'$ as a top-level (V) redex in the context ${}^{[C]}([]\ M_2')$ would remove the ${}^{[C]}$ tag from the context via the remarking operator $\urcorner$, and the subterm would not be of a form which invalidates well-separation. The reasoning is the same for an unmarked (V) step.
  2. If we had $P \equiv \text{let } x = ({}^{[A]}\text{let } y = N_1' \text{ in } N_2') \text{ in}$ then as in the discussion above an (A_1) step could break well-separation, which is why we have disallowed (A_1) steps in this lemma.

  Since neither of the above are possible, N' must be well-separated.
- *Outermost (A) markings.* We have three analogous cases of how a marked (A) step could lose well-separation. We take ${}^{[A]}\text{let } x = (P\, M_1') \text{ in } M_2'$ to occur in M'. If $P\, M_1'$ were a top-level (V) step, the outermost ${}^{[A]}$ marking would be removed were it contracted; an (A_1) step could again cause well-separation to be violated, but again we have disallowed well-separation.

So only (A_1) steps can disrupt well-separation, and thus the remaining rules preserve the property. ♦

The remainder of the uniqueness of complete developments result follows from the confluence of marked redex contraction from well-separated terms, and given the strong normalization results of the previous section, we need only show weak confluence.

Lemma 3.3.20. Reduction of well-separated Let' terms by Let_0' is weakly confluent.

Proof: We take $M', N_1', N_2' \in \mathsf{Let}'$, all well-separated with both $M' \xrightarrow[\mathsf{Let}_0']{} N_i'$, and proceed by structural induction on M' to construct some N' such that for both i we have $N_i' \twoheadrightarrow_{\mathsf{Let}_0'} N'$. We first consider the case where at least one of the marked redexes in question is top-level, and assume without loss of generality that $M' \xrightarrow[\mathsf{Let}_0']{} N_1'$ is the top-level one:

1. *Top-level (V_0) reduction.* We have $M' \equiv {}^{[V]}\text{let } x = V_1' \text{ in } M_1'$ and two further cases based on the location of the second redex.
   (a) $V_1' \xrightarrow[\mathsf{Let}_0']{} V_2'$ *with* $N_2' \equiv {}^{[V]}\text{let } x = V_2' \text{ in } M_1'$. So we have

   $$\begin{array}{rcl} N_1' & \equiv & M_1'[x := V_1'] \\ & \twoheadrightarrow_{\mathsf{Let}_0'} & M_1'[x := V_2'] \end{array}$$

   by Lemma 3.3.3.(1), with $N_2' \xrightarrow[(V)]{} M_1'[x := V_2']$ clear.

   (b) $M_1' \xrightarrow[\mathsf{Let}_0']{} M_2'$ *with* $N_2' \equiv {}^{[V]}\text{let } x = V_1' \text{ in } M_2'$. So here we have

   $$\begin{array}{rcl} N_1' & \equiv & M_1'[x := V_1'] \\ & \twoheadrightarrow & M_2'[x := V_1'] \end{array}$$

   by Lemma 3.3.3.(2), with $N_2' \xrightarrow[(V)]{} M_2'[x := V_1']$ clear.

2. *Top-level* (I_0, C_0, A_0) *reduction.* First we consider $M \equiv {}^{[C]}(P\, M_1')\, M_2'$. If the second contraction is contained wholly within P, M_1' or M_2' then the result is trivial as it is simply moved around; otherwise since M' is well-separated we can have only $M \equiv {}^{[C]}({}^{[V]}\text{let } x = V' \text{ in } M_1')\, M_2'$ with $N_2' \equiv (M_1'[x := V'])\, M_2'$ since $\urcorner$ will delete the ${}^{[C]}$ tag, and then clearly

$$N_1' \equiv \text{let } x = V' \text{ in } (M_1'\, M_2') \xrightarrow[(V)]{} N_2'$$

since $x \notin \text{fv}(M_2')$. The analysis of a top-level (A_0) redex and the effect of $\urcorner$ upon it are similar, as is top-level (I_0) reduction, with the added felicity in the latter that we need consider only redexes properly contained within the subterms, since (say) a lambda abstraction itself cannot be a top-level redex.

Compatible closure is trivial, and the proof is complete. ♦

The following corollary will be useful in the next section:

Corollary 3.3.21. Marked (C, A) reduction of well-separated Let' terms is weakly confluent.

Proof: By the last item in the proof above; clearly the inner redex is contracted by the same rule regardless of whether the outer redex is contracted. ♦

We can now conclude that Let_0' reduction is confluent, and proceed to the strengthening of $\text{FD}(\mathsf{Let})$.

Proposition 3.3.22. Reduction of well-separated Let' terms by Let_0' is confluent.

Proof: By Lemma 2.1.5 with Proposition 3.3.14 and Lemma 3.3.20. ♦

Proposition 3.3.23, $\text{FD!}(\text{WS-}\mathsf{Let})$**.** Let $M \in \mathsf{Let}$ be well-separated, and let $\bar{\gamma}$ index redexes in M. Then all Let-developments of $(M, \bar{\gamma})$ are finite and can be extended to complete developments, and all complete developments of $(M, \bar{\gamma})$ end in the same term.

Proof: We have finiteness and extension by $\text{FD}(\mathsf{Let})$. Since every development is a Let_0' sequence and vice-versa, then with $\text{FD}(\mathsf{Let})$ and Proposition 3.3.22 we have by Lemma 2.1.4 that the Let_0'-normal form of $(M, \bar{\gamma})$ — and hence the result of all its complete developments — is a unique, and by Lemma 3.3.19 a well-separated term as well. ♦

The next two lemmas are easy applications of $\text{FD!}(\text{WS-}\mathsf{Let})$.

Lemma 3.3.24. Let $M, M_1, M_2 \in \mathsf{Let}$ and $(M, \bar{\gamma}) \in \mathsf{Let}'$ such that $\bar{\gamma}$ is well-separated, $(M, \bar{\gamma}) \xrightarrow[\text{cpl}]{} M_1$ and $\sigma : M \xrightarrow[(I,V)]{\delta} M_2$. Then there exists some N such that $(M_2, \bar{\gamma}/\sigma) \xrightarrow[\text{cpl}]{} N$ and $M_1 \xrightarrow[(I,V)]{} \!\!\!\!\!\rightarrow N$.

Proof: We note first that since arbitrary (I, V) redexes can be marked in a well-separated marking, $\bar{\gamma} \cup \{\delta\}$ is again well-separated. So then we have

$$(M, \bar{\gamma} \cup \{\delta\}) \xrightarrow[(I,V)]{\delta} (M_2, (\bar{\gamma} \cup \{\delta\})/\sigma) \equiv (M_2, \bar{\gamma}/\sigma) \ ,$$

and take N as the complete reduct of $(M_2, \bar{\gamma}/\sigma)$. Then since the complete development of $(M, \bar{\gamma})$ is a partial development of $(M, \bar{\gamma} \cup \{\delta\})$, we can extend the latter to a complete development, which by $\text{FD!}(\text{WS-}\mathsf{Let})$ must also end in N. Then by Lemma 3.3.7, the extension is again a (I, V) sequence. ♦

Corollary 3.3.25. Let $M, M_1, M_2 \in \mathsf{Let}$ and $(M, \bar{\gamma}) \in \mathsf{Let}'$ such that $\bar{\gamma}$ is well-separated, $(M, \bar{\gamma}) \xrightarrow[\text{cpl}]{} M_1$ and $\sigma : M \xrightarrow[(I,V)]{} \!\!\!\!\!\rightarrow M_2$. Then there exists some N such that $(M_2, \bar{\gamma}/\sigma) \xrightarrow[\text{cpl}]{} N$ and $M_1 \xrightarrow[(I,V)]{} \!\!\!\!\!\rightarrow N$.

Proof: Follows from the preceding result by induction on the length of σ. ♦

Syntactic domains As for Let, plus the following:

Evaluation Contexts	E	$::=$	$[\,] \mid E\,M \mid \text{let } x = E \text{ in } M$
Answers	A	$::=$	V

Standard reduction relation

(I)	$(\lambda x.\, M)\, N$	$\mapsto$	$\text{let } x = N \text{ in } M$
(V)	$\text{let } x = V \text{ in } M$	$\mapsto$	$M[x := V]$

Figure 3.5: Standard reduction for call-by-let.

3.4 Confluence

One would hope that finiteness of developments results would lead immediately to the confluence result for the full calculus as for $(\beta\!\rightarrow)$ and $(\beta\!\rightarrow_v)$, but since Let' markings are not joinable, such is not the case. Nonetheless, we do have that a term with arbitrary marked (I, V) redexes is well-separated, and that all (I, V)-markings are joinable (Lemma 3.3.18), so with FD!(WS-Let) we can use Lemma 3.3.18 to assert the confluence of this subset, which would in fact be the most difficult aspect of the proof otherwise.

Lemma 3.4.1. Reduction by (I, V) in Let is confluent. ♦

It is clear that the (C, A) rules are themselves confluent, and that they commute with (I, V) steps as specified below.

Lemma 3.4.2. Reduction of Let terms by (C, A) steps is confluent.

Proof: By Lemma 2.1.5 with Corollary 3.3.12 and Corollary 3.3.21. ♦

Lemma 3.4.3. Let $M, M_1, M_2 \in \mathsf{Let}$ with $M \xrightarrow[(I,V)]{} M_1$ and $M \xrightarrow[(C,A)]{} M_2$. Then there exists some N such that $M_1 \xrightarrow[(I,V)]{} N$ and $M_2 \xrightarrow[(C,A)]{} \!\!\!\!\rightarrow N$:

$$\begin{tikzcd}
M \arrow[r, "(C,A)"'] \arrow[d, "(I,V)"] & M_2 \arrow[d, dotted, "(I,V)"'] \\
M_1 \arrow[r, dotted, two heads, "(C,A)"] & N
\end{tikzcd} .$$

Proof: The result follows from FD!(WS-Let), with the observation that the (C, A) step would never duplicate or discard the (I, V) redex, hence guaranteeing that the complete development of its (singleton) set of residuals will require exactly one step. ♦

Now with the above results and a general property of reduction, we have confluence of Let.

Proposition 3.4.4, CR(Let). Let is confluent.

Proof: Follows from Lemma 3.4.1 through Lemma 3.4.3 by Lemma 2.1.14. ♦

3.5 Standard call-by-let reduction

Figure 3.5 presents the relation we will show to be a standard strategy for Let reduction. In Let the set of answers is equivalent to the set of values, as in Name and Val. Given a notion of reduction $\mapsto$ which is (or which we assert to be) a standard strategy for $\rightarrow$, we refer to steps of $\rightarrow$ which are not steps of $\mapsto$ as *internal* steps, written $\xrightarrow[\text{i}]{}$ for single steps and $\twoheadrightarrow_{\text{i}}$ for multiple steps.

Notation. We will make extensive use of arguments about the relative positions of redexes in a term, and we develop our notation for discussing steps and paths within a term presently. For a term M, each indexing path γ for M gives rise to a context, whose hole is exactly at the position given by γ. We refer to that context as $\mathcal{C}^M_\gamma$, and to the indexed term as $M|_\gamma$, so then $M \equiv \mathcal{C}^M_\gamma[M|_\gamma]$. Similarly, we will sometimes need to consider the path which gives rise in terms to a context, and define $[\,]/C$ to be the path γ such that $C|_\gamma \equiv [\,]$. For example, taking $M \equiv (\text{let } x = M_1 \text{ in } M_2)\ M_3$ and $\gamma \equiv \underline{@1}\,\underline{\ell 2}$, we have the following:

$$\begin{array}{rcl} M|_\gamma & \equiv & M_2 \\ \mathcal{C}^M_\gamma & \equiv & (\text{let } x = M_1 \text{ in } [\,])\ M_3 \\ [\,]/((\text{let } x = M_1 \text{ in } [\,])\ M_3) & \equiv & \gamma\ . \end{array}$$

The result of $[\,]/C$ does exist and is always unique since we take contexts to have a single hole.

We will use the following basic lemmas for the proof of the standardization result.

Lemma 3.5.1. 1. For a Let context C and a Let evaluation context E, $C[E]$ is a Let evaluation context if and only if $E[C]$ is a Let evaluation context if and only if C is a Let evaluation context.

2. Let γ index subterms of both M and N. Then $\mathcal{C}^M_\gamma$ is a Let evaluation context if and only if $\mathcal{C}^N_\gamma$ is a Let evaluation context.

Proof: The first clause is straightforward. If C is an evaluation context then both of the other two hold by the definition of evaluation context, since given two evaluation contexts E_1, E_2 it is clear that $E_1[E_2]$ is trivially an evaluation context again. If $E[C]$ is an evaluation context then by structural induction on E, C is an evaluation context as well. Finally if $C[E]$ is an evaluation context, then trivially C must match the production of an evaluation context as well.

The second clause is clear from the fact that whether a context is an evaluation context depends only on the path from the top-level to the hole. ♦

As usual we have a unique evaluation contexts lemma to show that the standard reduction relation is in fact not just a relation, but also a partial function, defined in particular on all closed non-answers, and undefined on all answers.

Lemma 3.5.2. Let $M, V \in \mathsf{Let}$. Then:

1. There exists no E, Δ, with Δ a top-level (I, V)-redex, such that $V \equiv E[\Delta]$.

2. There exists at most one E, Δ, with Δ a top-level (I, V)-redex, such that $M \equiv E[\Delta]$.

3. If M is closed then either M is a value or the E, Δ of the previous clause exist and are unique.

Proof: Clause (1) is trivial by inspection of the grammars for E and V. Clause (2) follows by structural induction on M:

- $M \equiv x$. Then E, Δ cannot be formed.
- $M \equiv \lambda x.\,N$. Follows from Clause (1).
- $M \equiv M_0\ M_1$. We consider M_0:
  - $M_0 \equiv x$. Then no such E, Δ can be formed.
  - $M_0 \equiv \lambda x.\,N$. Then $E \equiv [\,]$ and $\Delta \equiv M$.
  - Other M_0. Then by induction on M_0, if we have E_0, Δ such that $M_0 \equiv E_0[\Delta]$, then $M \equiv (E_0[\Delta]\ M_1)$ and $E \equiv E_0\ M_1$; otherwise again none can exist.
- $M \equiv \text{let } y = M_0 \text{ in } M_1$. We consider M_0:
  - $M_0 \equiv V$. Then $E \equiv [\,]$, $\Delta \equiv M$ and no other E can be formed.
  - Other M_0. Then by induction on M_0 as for top-level application.

Clause (3) is similar to the previous clause: we can note that the only cases where E, Δ do not exist are when M is a value or when a free variable appears; for M closed the latter case will not arise. ♦

The next lemma simply states that internal steps will never convert a non-answer to an answer, nor vice-versa. Taken with the previous lemma, these two results are sufficient to show that $\mapsto$ is an answer-finding strategy on closed terms *in a strongly normalizing system*. As discussed in the introduction, we have chosen not to rely overly on strong normalization in order to more easily adapt our results to non-normalizing extensions; and therefore mention this fact only as an aside.

Properties **P1** through **P4** (p. 27) hold for call-by-let reduction as well.

Lemma 3.5.3. *Call-by-let reduction satisfies **P1**:* Let $\sigma : M \underset{\mathrm{i}}{\rightarrow} N$. Then M is a value if and only if N is a value.

Proof: If M is a value then the result is trivial by inspection of the reduction rules. If N is a value, then assume that M is a non-value; we must have $M \xrightarrow[(I,V)]{\epsilon} N$, but that step is standard, not internal, and so M must be a value. ♦

The complement to evaluation contexts are *speculative* contexts. Speculative contexts are given by the following grammar:

$$\begin{aligned} H \ ::= \lambda x.\, C \mid M\ C \mid H\ M \\ \mid \text{let } x = H \text{ in } M \\ \mid \text{let } x = M \text{ in } C \end{aligned}$$

With the following lemma, speculative contexts will be useful in the proofs of this section. Since the grammar above is just a more constructive definition of "all contexts which are not evaluation contexts," speculative contexts will appear in inductive arguments regarding non-standard (I, V) steps.

Lemma 3.5.4. Speculative and evaluation contexts partition the set of all contexts: every context C is either an evaluation or a speculative context, but for no evaluation context E and speculative context H do we have $E \equiv H$.

Proof: Clear from the definition. ♦

Lemma 3.5.5. *Call-by-name reduction satisfies **P2**:* Let

$$\begin{aligned} \sigma_0 : M &\ \xrightarrow[\mathrm{i}]{\gamma_0}\ N_0 \\ \text{and } \sigma_1 : M &\ \xrightarrow[\mathrm{i}]{\gamma_1}\ N_1 \ . \end{aligned}$$

Then every element of γ_1/σ_0 is internal.

Proof: If γ_1 indexes a (C, A) redex, then the result is trivial; all of its residuals would again be (C, A) redexes, and hence internal. So we can assume henceforward that γ_1 is an (I, V) step, and since it is internal it must be within a speculative context H; we will show that the contexts formed by all members of γ_1/σ_0 are all speculative. So we have

$$\begin{aligned} \sigma_0 : M \equiv C_0[\Delta_0] &\ \xrightarrow[\mathrm{i}]{\gamma_0}\ C_0[\Delta_0'] \equiv N_0 \\ \sigma_1 : M \equiv H_1[\Delta_1] &\ \xrightarrow[\mathrm{i}]{\gamma_1}\ H_1[\Delta_1'] \equiv N_1 \end{aligned}$$

and we proceed by structural induction on M, examining C_0:

- $C_0 \equiv [\,]$. Then γ_0 is a (C, A) step, and we consider each of the two redexes separately.
  - *γ_0 is a (C) step.* So we have

    $$M \equiv (\text{let } x = M_0 \text{ in } M_1)\ M_2 \xrightarrow[(C)]{\epsilon} \text{let } x = M_0 \text{ in } (M_1\ M_2) \equiv N_0$$

    and we consider H_1. We know from the possible forms of a speculative context that $H_1 \not\equiv [\,]$, $[\,]\ M_2$ nor $(\text{let } x = [\,] \text{ in } M_1)\ M_2$, and we consider the remaining valid combinations.
    * $H_1 \equiv (\text{let } x = M_0 \text{ in } M_1)\, C$. So $\gamma_1 = \underline{@2}\gamma_1'$ and $\gamma_1/\sigma_0 = \{\underline{\ell 2}\underline{@2}\gamma_1'\}$, which clearly gives rise to a speculative context.
    * $H_1 \equiv (\text{let } x = M_0 \text{ in } C)\ M_2$. As in the previous case: we have $\gamma_1 \equiv\!= \underline{@1}\underline{\ell 2}\gamma_1'$ and $\gamma_1/\sigma_0 = \{\underline{\ell 2}\underline{@1}\gamma_1'\}$, which again gives rise to a speculative context.

        * $H_1 \equiv (\text{let } x = H \text{ in } M_1)\ M_2$. So $\gamma_1 \equiv \underline{@1}\,\underline{\ell 1}\gamma_1'$, where $\mathcal{C}^{M_0}_{\gamma_1'}$ is speculative. Then $\gamma_1/\sigma_0 = \{\underline{\ell 1}\gamma_1'\}$, and since $\mathcal{C}^{M_0}_{\gamma_1'}$ is speculative, so is

$$\mathcal{C}^{\text{let } x = M_0 \text{ in } (M_1\ M_2)}_{\underline{\ell 1}\gamma_0'} \equiv \text{let } x = (\mathcal{C}^{M_0}_{\gamma_1'}) \text{ in } (M_1\ M_2) \ .$$

    - γ_0 *is a* (A) *step.* As in the analysis for the (C) step. All but one case simply moves a redex in a clearly internal position to another clearly internal position; otherwise we rely on the fact that when a subcontext of a speculative context is in evaluation position, it must again be speculative.

- $C_0 \equiv \lambda x.\, C$. In this case the result is trivial; *all* contractions under an abstraction — as well as their residuals with respect to other steps under the same abstraction — are internal.

- $C_0 \equiv C_a\ M_1$. So we have $M \equiv M_0\ M_1$ with $M_0 \equiv C_a[\Delta_0]$. Again we notice that $H_1 \not\equiv [\,]$, and consider the side of the application where γ_1 lies:

    - $H_1 \equiv H_a\ M_1$. By the structural induction hypothesis on M_0, C_a and H_a.
    - $H_1 \equiv (C_a[\Delta_0])\ C$. Then the set γ_1/σ_0 is a singleton, and its contents form the context $(C_a[\Delta_0'])\ C$, which is again speculative.

- $C_0 \equiv M_0\ C_a$. This case is simple: once again $H_1 \not\equiv [\,]$. Then if $H_1 \equiv H\ C_a[\Delta_0]$, we have a single residual generating the context $H_1 \equiv H\ C_a[\Delta_0']$ which is clearly speculative. If otherwise $H_0 \equiv M_0\ H_a$, then by compatible closure all residuals generate contexts of the form $M_0\ H$, which are again speculative.

- $M \equiv \text{let } x = M_0 \text{ in } M_1$ *and* $C \not\equiv [\,]$. As for the cases where M is an application and γ_0 is not top-level: since γ_1 also cannot be top-level lest it be standard, the analysis is precisely the same as above.

♦

Lemma 3.5.6. *Call-by-let reduction satisfies **P3**:* Let $\sigma : M_0 \xrightarrow[\mathrm{i}]{\gamma} N_0$ and $N_0 \mapsto N_1$. Then M_0 has a standard redex: there exists some M_1 such that $M_0 \mapsto M_1$.

Proof: We take

$$M_0 \equiv C[\Delta_0] \xrightarrow[\mathrm{i}]{} C[\Delta_0'] \equiv N_0 \ .$$

and

$$N_0 \equiv E_1[\Delta_1] \mapsto E_1[\Delta_1'] \equiv N_1$$

and proceed by induction on C to find the evaluation context E and top-level standard redex Δ for which $M \equiv E[\Delta]$.

- $C \equiv [\,]$. Then $M_0 \xrightarrow[(C,A)]{\epsilon} N_0$, since top-level (I, V) steps are standard, not internal. We consider each rule separately:

    - $M_0 \xrightarrow[(C)]{\epsilon} N_0$. Then we have

$$\begin{aligned} M_0 &\equiv (\text{let } x = L_0 \text{ in } L_1)\ L_2 \\ \text{and } N_0 &\equiv \text{let } x = L_0 \text{ in } L_1\ L_2 \ . \end{aligned}$$

      We distinguish two possibilities regarding E_1.

        * If $E_1 \equiv [\,]$, then L_0 must be a value. We take $E \equiv ([\,]\ L_2)$, with $(\text{let } x = L_0 \text{ in } L_1)$ as the top-level standard (V) redex.
        * $E_1 \equiv \text{let } x = E_2 \text{ in } L_1\ L_2$ where $E_2[\Delta_1] \equiv L_0$, and so

$$E_0 \equiv (\text{let } x = E_2 \text{ in } L_1)\ L_2$$

          and $\Delta \equiv \Delta_1$.

    - $M \xrightarrow[(A)]{\epsilon} N$. This case is similar to the previous case; the point is that although these commuting conversions do move terms around, the manipulations do not introduce standard redexes.

- $C \equiv L\,C_0$ or let $x = L$ in C_0. Then trivially the standard redex of M and its position are exactly the same as in N.
- $C \equiv C_0\,L$ or let $x = C_0$ in L. We consider E_1:
  - $E_1 \equiv [\,]$. Then again $E \equiv [\,]$.
  - Otherwise, we rely on the induction hypothesis over the subterm with both redexes. Note that within each $C_0[\Delta_0]$ we do have as the hypothesis requires that the internal contraction must again be internal; if it were standard in the subterm then it would be standard again in M due to the position of the subterms in this particular case.

No other construction of C is possible as otherwise E_1 could not be an evaluation context. ♦

Lemma 3.5.7. *Call-by-name reduction satisfies **P4**:* Let $\sigma : M_0 \xrightarrow[\mathrm{i}]{\gamma} N_0$ and $M_0 \overset{\delta}{\longmapsto} M_1$. Then δ/σ contains a single element δ_1 which is the standard redex of N_0: $N_0 \overset{\delta_1}{\longmapsto} N_1$.

Proof: We take
$$M_0 \equiv C[\Delta_0] \xrightarrow[\mathrm{i}]{} C[\Delta_0'] \equiv N_0$$
and
$$M_0 \equiv E_0[\Delta_1] \xrightarrow[\mathrm{i}]{} E_0[\Delta_1'] \equiv M_1 \ \ .$$
We proceed by induction on E_0, constructing in each case the evaluation context E and top-level standard redex Δ such that $N_0 \equiv E[\Delta]$, and showing this contraction to be the residual of the redex at δ.

- $E_0 \equiv [\,]$. Then the result is clear, since Δ_1 must properly contain Δ_0. This fact is easily verified: if M_0 is a top-level (I) redex, then either Δ_0 is a different (I, V), which would clearly not move the top-level redex, or it is a (C, A) step, which must be properly contained in either the body of the function or within its argument since an (I) redex cannot also be a top-level (C, A) step: hence it will also not move the standard step from top-level. The reasoning for a (V) step is similar; the term could not also be a top-level (A) step which would move the outer binding inward. So trivially, $E \equiv [\,]$ and $\Delta \equiv N_0$, whose standard contraction is the residual of the redex at δ.
- $E_0 \equiv E_a\,L$. Three classes for C are possible:
  - $C \equiv [\,]$. In this case we must have a (C) step for the internal reduction. Then
  $$M_0 \equiv \Delta_0 \equiv (\text{let } x = L_1 \text{ in } L_2)\,L$$
  and $N_0 \equiv$ let $x = L_1$ in $L_2\,L$. The result follows in one of two ways depending on E_a:
    - If $E_a \equiv [\,]$ then have that L_1 is a value, $L_1 \equiv V$, and the standard redex of N_0 is top-level: overlining the standard redex in each case, we have
    $$\begin{array}{rcl} M_0 & \equiv & \overline{(\text{let } x = V \text{ in } L_2)}\,L \\ N_0 & \equiv & \overline{\text{let } x = V \text{ in } (L_2\,L)} \ \ . \end{array}$$
    So clearly we have $E \equiv [\,]$ and $\Delta \equiv N_0$; by the definition of the (C_1) rule (Figure 3.3, p. 34) we also have this step as the appropriate residual.
    - Otherwise we have $E_a \equiv$ let $x = E_b$ in L_2 and $L_1 \equiv E_b[\Delta_1]$, and again overlining the standard redex in each case the result is again clear from compatible closure over Let' reduction rules:
    $$\begin{array}{rcl} M_0 & \equiv & (\text{let } x = E_b[\overline{\Delta_1}] \text{ in } L_2)\,L \\ N_0 & \equiv & \text{let } x = E_b[\overline{\Delta_1}] \text{ in } (L_2\,L) \ \ . \end{array}$$
  - $C \equiv C_a\,L$. Then the result follows from the induction hypothesis.
  - $C \equiv E_a[\Delta_1]\,C_a$. Then we have $\Delta_1 \equiv \Delta_0$ and $E_1 \equiv E_a\,C_a[\Delta']$; the residual property is again clear from compatible closure.

- $E_0 \equiv \text{let } x = E_a \text{ in } L$. As for application above.

♦

Unfortunately, Lemma 2.1.20 does not apply, since **P0** does not hold. However, we do have a weaker version of **P0** which will lead to standardization. We present the general technique below, and as before assume a reduction relation R, plus a subset W of the markings of R-terms.

P0′. All R-developments are finite and can be extended to a complete development, all complete R-developments of the same term and marking $\bar{\gamma}$ end in the same term whenever $\mathsf{W}(\bar{\gamma})$. If $\mathsf{W}(\bar{\gamma})$, and δ indexes a standard redex, then $\bar{\gamma}$ and $\{\delta\}$ have a joining $\bar{\gamma} + \{\delta\}$, and moreover $\mathsf{W}(\bar{\gamma} + \{\delta\})$. If $\bar{\gamma}_0$ marks M_0 with $\mathsf{W}(\bar{\gamma}_0)$ and

$$(M_0, \bar{\gamma}_0) \xrightarrow[\text{dev}]{} (M_1, \bar{\gamma}_1) \ ,$$

then $\mathsf{W}(\bar{\gamma}_1)$; if $(M_0, \bar{\gamma}_0) \mapsto (N_0, \bar{\delta}_0)$ then $\mathsf{W}(\bar{\delta}_0)$ as well. Finally, for all γ, we have $\mathsf{W}(\{\gamma\})$.

We proceed as in Lemma 2.1.17 through Lemma 2.1.20.

Lemma 3.5.8. Let R satisfy **P0′** and **P2**, and let $(M, \bar{\gamma}) \xrightarrow[\text{cpl}]{} N$ be a complete R-development where $\mathsf{W}(\bar{\gamma})$. Then there exists some $(M_0, \bar{\gamma}_0)$ such that:

- $\mathsf{W}(\bar{\gamma}_0)$.
- $\sigma : (M, \bar{\gamma}) \xrightarrow[\text{dev}]{} \!\!\!\!\rightarrow (M_0, \bar{\gamma}_0)$ is an R-development.
- $|\sigma| : M \mapsto\!\!\!\!\rightarrow M_0$ is a standard R-reduction sequence.
- Every complete R-development of $(M_0, \bar{\gamma}_0)$ contracts only internal R redexes.

Proof: We can contract standard redexes while they exist; this sequence must be finite by **P0′**, and by **P0′** we also have $\mathsf{W}(\bar{\gamma}_0)$. When we have run out of standard steps, all completing sequences will be internal by **P2**. ♦

Lemma 3.5.9. Let R satisfy **P0′**, **P2**, **P3** and **P4**, and let $\sigma : (M, \bar{\gamma}) \xrightarrow[\text{i-cpl}]{} M_0$ and $M_0 \overset{\delta}{\mapsto} N$ where $\mathsf{W}(\bar{\gamma})$. Then there exists some $(N_0, \bar{\delta}_0)$ such that:

- $\mathsf{W}(\bar{\delta}_0)$.
- $\tau : M \mapsto\!\!\!\!\rightarrow N_0$.
- $\bar{\gamma}/\tau = \bar{\delta}_0$.
- $(N_0, \bar{\delta}_0) \xrightarrow[\text{i-cpl}]{} N$.

Proof: As in Lemma 2.1.18, noticing that **P0′** also gives $\mathsf{W}(\bar{\delta}_0)$. ♦

Lemma 3.5.10. Let R satisfy **P0′**, **P2**, **P3** and **P4**, and let $M \xrightarrow[\text{i}]{} M_0 \mapsto\!\!\!\!\rightarrow N$. Then there exists some N_0 such that $M \mapsto\!\!\!\!\rightarrow N_0 \xrightarrow[\text{i}]{}\!\!\!\!\rightarrow N$.

Proof: Just as in Lemma 2.1.19, since since for a siungleton marking set $\{\gamma\}$, **P0′** gives us $\mathsf{W}(\{\gamma\})$. ♦

Lemma 3.5.11. Let R satisfy **P0′** and **P1** through **P4**, and let $M \twoheadrightarrow A$. Then there exists some A_0 such that $M \mapsto\!\!\!\!\rightarrow A_0$ and $A_0 \twoheadrightarrow A$.

Proof: As in Lemma 2.1.20. ♦

Now to apply Lemma 3.5.11 to call-by-let, we need only the following lemma:

Lemma 3.5.12. Call-by-let satisfies **P0′**, with well-separation as W and set union as the operator $+$.

Proof: With FD!(WS-Let) for the conditions on developments; for the various joinings we need observe only that arbitrary (I, V) steps may be included in a well-separated marking; conservation of W by developments and standard reduction rules is immediate from Lemma 3.3.19; the singleton marking condition is immediate from the definition of well-separation. ♦

Syntactic domains

Values	$V', W' ::$	$=$	$x \mid \lambda x.\, M'$
Terms	$L', M', N' ::$	$=$	$V' \mid M'\, N' \mid P\, M'$
Let bindings	$P ::$	$=$	$\text{let } x = M' \text{ in} \mid {}^{[\ell]}\text{let } x = M' \text{ in}$

Reduction rules

(I)	$(\lambda x.\, M')\, N'$	$\to$	$\text{let } x = N' \text{ in } M'$
(V)	$\text{let } x = V' \text{ in } M'$	$\to$	$M'[x := V']$
	${}^{[\ell]}\text{let } x = V' \text{ in } M'$	$\to$	$M'[x := V']$
(C)	$(P\, M')\, N'$	$\to$	$P\,(M'\, N')$
(A)	$\text{let } x = (P\, M') \text{ in } N'$	$\to$	$P\,(\text{let } x = M' \text{ in } N')$
	${}^{[\ell]}\text{let } x = (P\, M') \text{ in } N'$	$\to$	$P\,({}^{[\ell]}\text{let } x = M' \text{ in } N')$

Figure 3.6: The binding marking variant of Let', Let^{ℓ}.

Corollary 3.5.13. If $\sigma : M \xrightarrow[\mathsf{Let}]{} \!\!\!\!\rightarrow A$ then there exists some A_0 such that $M \underset{\mathsf{Let}}{\longmapsto\!\!\!\!\rightarrow} A_0$ and $A_0 \xrightarrow[\mathsf{Let}]{} \!\!\!\!\rightarrow A$.

Proof: By Lemma 3.5.11, with Corollary 3.5.13, Lemma 3.5.3, Lemma 3.5.5, Lemma 3.5.6 and Lemma 3.5.7. ♦

Proposition 3.5.14, $\mathtt{Std}(\mathsf{Let})$**.** Reduction by $\underset{\mathsf{Let}}{\longmapsto}$ is a standard strategy for $\xrightarrow[\mathsf{Let}]{}$ reduction to values.

Proof: By Lemma 3.5.2 and Corollary 3.5.13. ♦

3.6 An alternate marking theory for Let terms

The notion of marking in Let' allows us to keep track of certain Let redexes while other redexes are contracted. We will also need to track the positions of certain let-bindings as various redexes are contracted to show strong normalization of (V, C, A)-reduction and for the conservative extension of Val to Let. To this end, we consider a slightly different marking of Let terms, marking bindings rather than redexes. Figure 3.6 presents the syntax and reduction rules for this Let^{ℓ} system. Again, marked terms are interchangeable with pairs of unmarked terms and sets of paths to bindings, and we assume the usual notion of residual and projection from Let^{ℓ} to Let. We say that a term with no marked let-bindings is *binding-unmarked*.

Lemma 3.6.1. Let $\sigma' : M' \to N'$. Then there exists some N such that $\sigma : |M'| \to N$; moreover $|N'| \equiv N$.

Proof: Trivial, just by erasing marks. ♦

In the above we can take $|\sigma'| \equiv \sigma$.

Lemma 3.6.2. Let $|M'| \equiv M$ and $\sigma : M \to N$. Then there is a unique marked term N' such that $M' \to N'$ and $|N'| \equiv N$.

Proof: Again since exactly one of the marked or unmarked versions of the rule used by σ will apply to the corresponding redex of M'. ♦

Since marked let-bindings might be irreducible at any given point in a reduction path, the notion of developments makes little sense in this context. Instead, we present Let^{ℓ} with two very specific goals in mind. First, we use the system to show that arbitrary (V, C, A) reduction in Let is strongly normalizing. The weighting of Let' is insufficient here because in general, given a reduction sequence $M' \xrightarrow[(V,C,A)]{} \!\!\!\!\rightarrow N'$, steps later in the sequence are not necessarily expressible as the residual of a redex in M'. In such a case they would not be markable by first marking M' and considering residual markings, complicating the reasoning. We will also use Let^{ℓ} to investigate the notions of *righteous* and *wicked* reduction steps, which will be of significant use in the proof of the conservative extension of Val to Let.

3.6.1 Strong normalization of (V, C, A) reduction

General (V, C, A)-reduction has the property of strong normalization. To show this property, we can reuse the norm from the previous marking of Let to weigh Let^ℓ terms as well, again disregarding markings when assessing weight.

We adapt the notion of a decreasing weighting from that of Let'. Since we do not need to consider (I) redexes in any detail, and since moreover we no longer mark applications, we can safely omit the requirement regarding marked (I) redexes. Since we want to consider steps at any point in a reduction sequence, not just those which occur as residuals of some redex at the beginning, and since (C, A) steps can "uncover" previously blocked (V, C, A) steps, we must strengthen the second condition to apply to all let-bindings. We achieve this extension by simply marking all of them: (V, C, A) steps preserve the markedness of bindings and never introduce new, unmarked ones.

Definition. A weighted Let^ℓ term $\dot{M}$ is said to have a decreasing weighting if the following condition is true: For every subterm $({}^{[\ell]}\mathrm{let}\ x = \dot{M}_0 \text{ in } \dot{M}_1)$ of $\dot{M}$ and for all occurrences of x^i within $\dot{M}_1$, we have that $\|\dot{M}_0\| < i$. ◊

Lemma 3.6.3. Let $\dot{M}$ have a decreasing weighting, and let $\dot{M} \xrightarrow[(V,C,A)]{} \dot{N}$. Then $\|\dot{M}\| > \|\dot{N}\|$.

Proof: As indicated in Equations (3.1), (3.2) and (3.3) respectively in Clauses (2), (3) and (4) of the proof of Lemma 3.3.9, the marking of the redex is not relevant to the result. ⧫

Lemma 3.6.4. Let $\dot{M} \xrightarrow[(V,C,A)]{} \dot{N}$ by a top-level contraction, and let $\dot{M}$ have a decreasing weighting but no marked redexes. Then $\dot{N}$ has a decreasing weighting.

Proof: This proof is a modification of the proof of Lemma 3.3.11. We consider each of the three possible reduction rules separately, and in each case show that every let-binding in $\dot{N}$ again satisfies the criteria for a decreasing weighting.

1. *Top level (V) contraction.* So $\dot{M} \equiv \mathrm{let}\ x = \dot{V} \text{ in } \dot{M}_1$, and $\dot{L} \equiv \mathrm{let}\ y = \dot{L}_0 \text{ in } \dot{L}_1$ occur in $\dot{N} \equiv \dot{M}_1[x := \dot{V}]$. We can associate $\dot{L}$ with a let-binding in $\dot{M}$ as follows by considering the path γ in $\dot{N}$ to $\dot{L}$, and the term $(\dot{N}, \{\gamma\})$. Since (V, C, A) reduction does not create new let-bindings — only (I) steps do so — there is some redex at δ in $\dot{M}$ such that γ is one of its residuals. We let $\dot{L}_0$ be the redex indexed by δ in $\dot{M}$.

   If $\dot{L}_0$ is a subterm of $\dot{V}$, then the result is trivial since first of all $\dot{L} \equiv \dot{L}_0$, and itself meets the requirements for a decreasing weighting, and moreover because its norm must be less than the weighting of any variable it replaces.

2. *Top level (C, A) contraction.* Trivial; terms brought into the scope of an identifier will not be relevant to the conditions for a decreasing weighting.

⧫

Lemma 3.6.5. Let $\dot{M} \xrightarrow[(V,C,A)]{} \dot{N}$, and let $\dot{M}$ have a decreasing weighting. Then $\dot{N}$ has a decreasing weighting.

Proof: As in the proof of Lemma 3.3.11, except that we need not consider the (I_0) steps. ⧫

Lemma 3.6.6. All Let^ℓ terms can be given a decreasing weighting.

Proof: Just as in Lemma 3.3.13. ⧫

Proposition 3.6.7. (V, C, A)-reduction in Let is strongly normalizing.

Proof: By Lemma 3.6.3, Lemma 3.6.5 and Lemma 3.6.6. ⧫

3.6.2 Casting wicked redexes from paths of righteousness

The reduction path of the righteous man is beset on all sides by the inequities of the commuting conversions and the tyranny of evil redexes [72, loosely].

In relating Let-reduction sequences to Val-reduction sequences, certain orderings of contractions will obscure the correspondence of terms of the former to the latter. In this section, we categorize Let^ℓ redexes whose contractions, when projected into Let, would yield such confusion as *wicked*. Our main result here is that given any Let^ℓ sequence from M' to N' where N' has no wicked redexes, it is possible to construct a Let^ℓ sequence from M' to N' which contracts no wicked redexes. This statement is useful because, as we will see in the definitions below, call-by-value terms have no wicked redexes, so the result applies particularly well to sequences ending in call-by-value terms — exactly what we need for the conservative extension result.

Let $\sigma : M' \xrightarrow[\mathsf{Let}^\ell]{} N'$. We can divide all such σ into two classes, the *righteous* and the *wicked*, as follows.

Definition 3.6.8. A reduction step $\sigma : M' \xrightarrow[\mathsf{Let}^\ell]{} N'$ is *wicked* if M is of the form $C'[{}^{[\ell]}\text{let } x = M'_1 \text{ in } M'_2]$ where $M'_2 \xrightarrow[\mathsf{Let}^\ell]{} N'_2$ and $N' \equiv C'[{}^{[\ell]}\text{let } x = M'_1 \text{ in } N'_2]$. If a wicked σ contracts the top-level redex at position γ in M', we say that γ indexes a wicked redex. Reduction steps and redexes which are not wicked are said to be *righteous*; if a term contains only righteous redexes, we will refer to that term as righteous. ♢

Lemma 3.6.9. Let γ be wicked in M', and let $\sigma : M' \xrightarrow[\mathsf{Let}^\ell]{\delta} N'$ for some wicked redex δ. Then all redexes in γ/σ are also wicked.

Proof: If γ and δ are down different branches of a term, or if δ indexes a subterm of γ, then the result is trivial as the contraction would not affect γ. If γ is a subterm of δ, we can notice that the path δ will itself contain a step into the right-hand side of a marked let-binding, and so will all residuals of γ, since contracting δ will not alter its context. ♦

The preceding lemma allow us to make some statement about the residuals of wicked and righteous redexes. Since wicked redexes remain so during the contraction of our wicked redexes, it will always be possible to develop righteously first, and then wickedly.

Definition 3.6.10. Let $\bar{\gamma}$ mark wicked (respectively, righteous) redexes in M'. When we speak of a development of $(M', \bar{\gamma})$, we mean the projection of a development $(|M'|, \bar{\gamma}) \xrightarrow[\text{dev}]{} (N, \bar{\delta})$ in Let', into Let, subsequently lifted into Let^ℓ. By the preceding lemmas, $\bar{\delta}$ again indexes wicked (righteous) redexes in the lifting of N derived from lifting the development to that beginning with M'. ♢

One fine point is worth noting here: wicked redexes can be redeemed by following a righteous path. For example, consider the term ${}^{[\ell]}\text{let } x = V \text{ in } M$ where V and M are unmarked Let terms. We let σ be the path contracting top-level (V) redex, and let δ mark some redex in M. Clearly σ is righteous and δ is wicked, but since the result of σ will be entirely unmarked, δ/σ must be righteous. Terms with no marked let-bindings clearly have no wicked redexes, and have a useful property.

Lemma 3.6.11. Let M be binding-unmarked; clearly M is righteous. For all σ such that either $\sigma : M \to N$ or $\sigma : L \to M$, σ is righteous. In other words, all entirely wicked reduction sequences leading to or from a binding-unmarked term are zero-length.

Proof: Obvious, since an unmarked term can contain no wicked redexes, and since wicked redexes do not remove the marking from which their wickedness stems. ♦

Lemma 3.6.12. Let $\bar{\gamma}$ be well-separated and wicked for L' with $(L', \bar{\gamma}) \xrightarrow[\text{cpl}]{} M'$, and let δ be righteous for M' with $M' \xrightarrow{\delta} N'$. Then there exists some righteous redex δ_0 in L' such that $\sigma : L' \xrightarrow{\delta_0} M'_0$ and $(M'_0, \bar{\gamma}/\sigma) \xrightarrow[\text{cpl}]{} N'$. Moreover, we also have reduction sequences τ_1, τ_2 and a term M'_1 where

$$\tau_1 : (M'_0, \bar{\gamma}/\sigma) \xrightarrow[\text{dev}]{} (M'_1, \bar{\gamma}/\sigma/\tau_1)$$

is righteous and $\tau_2 : (M'_1, \bar{\gamma}/\sigma/\tau_1) \xrightarrow[\text{cpl}]{} N'$ is wicked.

Remark: The decomposition into τ_1 and τ_2 is not necessarily unique since the contraction of righteous redexes does not always preserve the righteousness of others. For example, the term $\text{let } x = \lambda z.\, I\, I \text{ in } {}^{[\ell]}\text{let } y = M \text{ in } x$ has two righteous redexes, under the abstraction and at top level. If the former is contracted first, the latter remains righteous, but the reverse is not true, as so there are two possible decompositions. What is important is that it is possible first to contract righteous redexes until no more are marked — which is mandated by FD!(WS-Let) — and then to complete the development by contracting only wicked redexes (guaranteed by Lemma 3.6.9).

Proof: The first claim is clear since righteous redexes are never subterms of wicked redexes, and so we will always have $\delta_0 \equiv \delta$. Moreover, since any wicked redexes which are subterms of the term indexed by δ in L' are contained within the right-hand side of a let-binding contained again within L', and since $\bar{\gamma}$ is well-separated, we know that $\bar{\gamma} \cup \{\delta\}$ is also well-separated. Since σ is a marked contraction, we know from Lemma 3.3.19 that $(\bar{\gamma} \cup \{\delta\})/\sigma \equiv \bar{\gamma}/\sigma$ is again well-separated, and so all complete developments terminate in N'. Finally, we can decompose the complete development into τ_1 and τ_2 by Lemma 3.6.9 and FD!(WS-Let). ◆

Lemma 3.6.13. Let $(L', \bar{\gamma}) \xrightarrow[\text{cpl}]{} M'$ be wicked, and let $\sigma : M' \twoheadrightarrow N'$ be righteous. Then there exists some M'_0 such that $L \twoheadrightarrow M'_0$ is righteous and $M'_0 \twoheadrightarrow N'$ is wicked.

Proof: By induction on the length of σ, applying Lemma 3.6.12 at every step. ◆

Lemma 3.6.14. Let $\sigma : L' \twoheadrightarrow M'$ be wicked, and let $M' \twoheadrightarrow N'$ be righteous. Then there exists some M'_0 such that $L \twoheadrightarrow M'_0$ is righteous and $M'_0 \twoheadrightarrow N'$ is wicked.

Proof: By induction on the length of σ, using Lemma 3.6.13 at step and treating each single wicked step as a complete development of a single redex. ◆

Corollary 3.6.15. Let $L' \twoheadrightarrow M'$ be wicked, let $M' \twoheadrightarrow N'$ be righteous, and let N' be binding-unmarked. Then there exists some σ such that σ is righteous and $\sigma : L' \twoheadrightarrow N'$.

Proof: By the above result and Lemma 3.6.11. ◆

Lemma 3.6.16. Let $\sigma : M \twoheadrightarrow N$ where N is binding-unmarked. Then there exists some righteous τ such that $\tau : M \twoheadrightarrow N$.

Proof: By induction on the number of wicked steps in σ. ◆

The notions of righteousness and wickedness are useful when attempting to reorder certain reduction steps before others. We consider two *corrupting* rules which add markings to a system:

$$
\begin{array}{lll}
(C_+) & (\text{let } x = L' \text{ in } M')\; N' & \rightarrow \;\; {}^{[\ell]}\text{let } x = L' \text{ in } M'\; N' \\
(A_+) & \text{let } x = (\text{let } y = L' \text{ in } M') \text{ in } N' & \rightarrow \;\; {}^{[\ell]}\text{let } x = L' \\
 & & \qquad \text{in } {}^{[\ell]}\text{let } y = M' \text{ in } N'
\end{array}
$$

Lemma 3.6.17. Let $\bar{\gamma}$ mark well-separated (C, A) redexes in $M \in \mathsf{Let}$, and let σ be a development of $(M, \bar{\gamma})$. Then σ can be lifted to a (C_+, A_+) sequence.

Proof: Follows from the well-separation of $\bar{\gamma}$; the only difficulty would arise in lifting a (C, A) step when a previous lifting would mark a let-binding, and the marking prevent the (C_+, A_+) step, which assumes unmarked let bindings at the start. ◆

Lemma 3.6.18. Let $\sigma : L' \xrightarrow[(C_+, A_+)]{\gamma} M'$ and $\tau : M' \xrightarrow[(I, V)]{\delta} N'$ be righteous. Then there exists some redex φ in L' such that $\varphi/\sigma = \{\delta\}$.

Proof: We perform a straightforward structural induction on L', considering its structure and looking for the top-level redex indexed by γ.

- $L' \equiv x$. This case is not possible.
- $L' \equiv \lambda x.\, L'_0$. By the structural induction hypothesis on L'_0.

- $L' \equiv L'_0\ L'_1$. By analysis of the possibilities for γ and δ.
  - $\delta \equiv \epsilon$. Then $\varphi \equiv \underline{@1}$.
  - $\gamma \equiv \underline{@1}\gamma_0$, $\delta \equiv \underline{@2}\delta_0$. Then $\varphi \equiv \delta$.
  - $\gamma \equiv \underline{@2}\gamma_0$, $\delta \equiv \underline{@1}\delta_0$. Then again $\varphi \equiv \delta$.
  - $\gamma \equiv s\gamma_0$, $\delta \equiv s\delta_0$. By the induction hypothesis on the appropriate subterm L'_i.
  - $\gamma \equiv \epsilon$. So we have $L'_1 \equiv \text{let } x = L'_2 \text{ in } L'_3$. We cannot have $\delta \equiv \underline{\ell 2}\delta_0$ since then δ would be wicked, and we have already covered the case where $\delta \equiv \epsilon$ above. So we take $\delta \equiv \underline{\ell 1}\delta_0$, and then clearly $\varphi \equiv \underline{@1}\,\underline{\ell 1}\delta_0$.
- $L' \equiv \text{let } x = L'_0 \text{ in } L'_1$. We again consider the various possibilities for γ and δ; the first four cases are almost exactly like the first four of the preceding case.
  - $\delta \equiv \epsilon$. Then $\varphi \equiv \underline{\ell 1}$.
  - $\gamma \equiv \underline{\ell 1}\gamma_0$, $\delta \equiv \underline{\ell 2}\delta_0$. Then $\varphi \equiv \delta$.
  - $\gamma \equiv \underline{\ell 2}\gamma_0$, $\delta \equiv \underline{\ell 1}\delta_0$. Then again $\varphi \equiv \delta$.
  - $\gamma \equiv s\gamma_0$, $\delta \equiv s\delta_0$. By the induction hypothesis on the appropriate subterm L'_i.
  - $\gamma \equiv \epsilon$. So we have $L'_1 \equiv \text{let } x = L'_2 \text{ in } L'_3$. We cannot have $\delta \equiv \underline{\ell 2}\delta_0$ since then δ would be wicked, and we have already covered the case where $\delta \equiv \epsilon$ above. So we take $\delta \equiv \underline{\ell 1}\delta_0$, and then clearly $\varphi \equiv \underline{\ell 1}\,\underline{\ell 1}\delta_0$.

♦

Corollary 3.6.19. Let $\bar\gamma$ be well-separated and mark (C_+, A_+) redexes in L' such that

$$(L', \bar\gamma) \xrightarrow[\text{cpl}]{} M' \;\; ,$$

and let $M' \xrightarrow[(I,V)]{} \!\!\!\!\rightarrow N'$ be a series of righteous reductions. Then there exists some righteous reduction sequence $\sigma : L' \xrightarrow[(I,V)]{}\!\!\!\!\rightarrow M'_0$ such that $(M'_0, \bar\gamma/\sigma) \xrightarrow[\text{cpl}]{} N'$. ♦

The details of the application of this material is suggested by the title of this section. Rearranging a reduction sequence so that wicked steps follow righteous steps, if the final result of the sequence is binding-unmarked then we can assert the existence of a reduction sequence between the same terms with only righteous steps.

3.7 Conservative extension of Val to Let

What is the relationship between Val and Let? Clearly, every $\beta{\rightarrow_v}$-reduction in Val can be simulated in Let by a pair of (I) and (V) reductions. That is, Let-reduction extends Val-reduction, and in fact this extension is conservative.

3.7.1 A Val-mimicking standard strategy for (I, V)

The next two definitions and lemma illuminate how Val reductions can mimic Let reductions. Since Let contains terms not in Val, we introduce the let-expansion relation $\cdot^{\overline{\ell\lambda}}$ to associate a Val term with each such "extra" Let term.

Definition 3.7.1. The operator $\cdot^{\overline{\ell\lambda}}$: Let $\rightarrow$ Val is given by

$$\begin{array}{rcl} x^{\overline{\ell\lambda}} & \equiv & x \\ \lambda x.\, M^{\overline{\ell\lambda}} & \equiv & \lambda x.\, M^{\overline{\ell\lambda}} \\ (M\ N)^{\overline{\ell\lambda}} & \equiv & M^{\overline{\ell\lambda}}\ N^{\overline{\ell\lambda}} \\ (\text{let } x = M \text{ in } N)^{\overline{\ell\lambda}} & \equiv & (\lambda x.\, N^{\overline{\ell\lambda}})\ M^{\overline{\ell\lambda}} \end{array}$$

◊

At the level of reduction sequences, we already know that each Val step corresponds directly to an (I) step followed by a (V) step. From Let, we may or may not need to contract an (I) redex to create the (V) step, so we will relate the following notion $(I^?V)$ to Val steps.

Definition 3.7.2. We write $M \xrightarrow[(I^? V)]{\gamma} N$ whenever either

1. $M \xrightarrow[(V)]{\gamma} N$ or
2. $M \xrightarrow[(I)]{\gamma} M_0 \xrightarrow[(V)]{\gamma} N$.

◇

We will omit the γ as usual when we can do so unambiguously.

Lemma 3.7.3. Let $M, N, N_0 \in \mathsf{Let}$.

1. $M \xrightarrow[(I^? V)]{} N$ if and only if $M^{\overline{\ell\lambda}} \xrightarrow[\mathsf{Val}]{} N^{\overline{\ell\lambda}}$.
2. Let $N_0 \xrightarrow[(I)]{} N$. Then $N_0^{\overline{\ell\lambda}} \equiv N^{\overline{\ell\lambda}}$.
3. Let $M \in \mathsf{Val}$. Then $M^{\overline{\ell\lambda}} \equiv M$.

Proof: Trivial. ⧫

In the remainder of this section, we show that arbitrary (I, V) sequences can be reordered as a series of $(I^? V)$ steps followed by a series of (I) steps.

Lemma 3.7.4. Let $M, M_i, N \in \mathsf{Let}$.

(i) Let $\sigma : M_0 \xrightarrow[(I)]{\gamma} M_1$, $M_1 \xrightarrow{\delta} M_2$ and $\gamma \not\equiv \delta$. Then there exists some φ such that $\varphi/\sigma = \{\delta\}$.

(ii) Let $\sigma : M_0 \xrightarrow[(I)]{\gamma} M_1$, $M_1 \xrightarrow[(V)]{\delta} M_2$, $\gamma \not\equiv \delta$. Then $M_0 \xrightarrow[(V)]{} M_1' \xrightarrow[(I)]{} \!\!\!\!\rightarrow M_2$.

(iii) Let $\sigma : M_0 \xrightarrow[(I)]{\gamma} M_1$, $M_1 \xrightarrow[(I)]{\delta/\sigma} M_2$. Then we have $\tau : M_0 \xrightarrow[(I)]{\delta} M_1'$ and $M_1 \xrightarrow[(I)]{\sigma/\tau} M_2$.

(iv) Let $\sigma : M \xrightarrow[(I,V)]{} \!\!\!\!\rightarrow N$. Then there exists some M_0 such that

$$M \xrightarrow[(I^? V)]{} \!\!\!\!\rightarrow M_0 \xrightarrow[(I)]{} \!\!\!\!\rightarrow N \ \ .$$

Proof: Clause (i) is obvious. Clause (ii) follows from Clause (i); we have some δ_0 such that $\delta_0/\sigma = \{\delta\}$, and trivially $\{\gamma, \delta_0\}$ is well-separated, so $M_0 \xrightarrow[(V)]{\delta_0} M_1'$ is a partial development, whose completion is by (I) contractions and which must end in M_2. Clause (iii) is similar to Clause (ii).

Clause (iv) follows by induction on the number of (V) steps in σ, $\|\sigma\|_V$. The simple case of $\|\sigma\|_V = 0$ is already an (I)-sequence. For $\|\sigma\|_V = 1 + n$ we decompose $\sigma = \sigma_0\sigma_1$, $\|\sigma_0\|_V = n$ and

$$\sigma_0 : M \xrightarrow[(I,V)]{} \!\!\!\!\rightarrow N_0 \quad \sigma_1 : N_0 \xrightarrow[(V)]{} N_1 \xrightarrow[(I)]{} \!\!\!\!\rightarrow N \ \ .$$

Then by the induction hypothesis we have

$$\tau_0 : M \xrightarrow[(I^? V)]{} \!\!\!\!\rightarrow M_0 \xrightarrow[(I)]{} \!\!\!\!\rightarrow N_0 \ \ .$$

But then we apply clauses (ii) and (iii) to σ_2,

$$\sigma_2 : M_0 \xrightarrow[(I)]{} \!\!\!\!\rightarrow N_0 \xrightarrow[(V)]{} N_1 \xrightarrow[(I)]{} \!\!\!\!\rightarrow N \ \ ,$$

and get some τ_1,

$$\tau_1 : M_0 \xrightarrow[(I^? V)]{} M_1 \xrightarrow[(I)]{} \!\!\!\!\rightarrow N \ \ ,$$

and then combine the first part of τ_0 with τ_1 to get

$$M \xrightarrow[(I^? V)]{} \!\!\!\!\rightarrow M_1 \xrightarrow[(I)]{} \!\!\!\!\rightarrow N \ \ .$$

⧫

Corollary 3.7.5. Let $M \in \mathsf{Let}$ and $N \in \mathsf{Val}$ such that $M \xrightarrow[(I,V)]{} \!\!\!\!\rightarrow N$. Then $M \xrightarrow[(I^? V)]{} \!\!\!\!\rightarrow N$.

Proof: By Lemma 3.7.4.iv we have M_0 such that $M \xrightarrow[(I^? V)]{} \!\!\!\!\rightarrow M_0 \xrightarrow[(I)]{} \!\!\!\!\rightarrow N$. Since $N \in \mathsf{Val}$, we must have $M_0 \equiv N$, since the result of an (I)-step always has a let-binding, and hence could not be a Val-term. ⧫

3.7.2 Conservative extension

For the final conservative extension result we will recall the results on the binding-marking Let calculus Let^{ℓ} (Section 3.6.2). The proof of Proposition 3.7.6 follows immediately from those ideas and the results on $(I^{?}V)$ steps.

Proposition 3.7.6. Let conservatively extends Val: For all terms M and N in Val, $M \xrightarrow[\mathsf{Val}]{} \!\!\!\!\rightarrow N$ if and only if $M \xrightarrow[\mathsf{Let}]{} \!\!\!\!\rightarrow N$.

Proof: We take $M, N \in \mathsf{Val}$ where $\sigma : M \xrightarrow[\mathsf{Let}]{} \!\!\!\!\rightarrow N$, and proceed by induction on the number of (C, A) steps in σ. Where we have zero such steps the result follows by Lemma 3.7.3.(1) and .(3), and Lemma 3.7.4.(iv).

Otherwise for the inductive step we consider the final (C, A) step in the sequence,

$$M_0 \xrightarrow[(C,A)]{} M_1 \xrightarrow[(I,V)]{} \!\!\!\!\rightarrow N \ .$$

Since single redexes trivially form well-separated markings, Lemma 3.6.17 allows us to lift this sequence into Let^{ℓ},

$$M_0 \xrightarrow[(C_+,A_+)]{} M_1' \xrightarrow[(I,V)]{} \!\!\!\!\rightarrow N' \ .$$

Since $N \in \mathsf{Val}$, any lifting of it into Let^{ℓ} will trivially be binding-unmarked, so $N \equiv N'$ and by Lemma 3.6.16 we have

$$M_0 \xrightarrow[(C_+,A_+)]{} M_1' \xrightarrow[(I,V)]{} \!\!\!\!\rightarrow^{\mathcal{R}} N \ ,$$

where $\mathcal{R}$ denotes righteousness. Then by Corollary 3.6.19 we can rearrange that sequence as

$$M_0 \xrightarrow[(I,V)]{} \!\!\!\!\rightarrow^{\mathcal{R}} N_0' \xrightarrow[(C_+,A_+)]{} \!\!\!\!\rightarrow N \ .$$

But since the result of (C_+, A_+) steps must contain let-bindings and so could not be in Val, we have $N \equiv N_0'$, and in fact

$$M_0 \xrightarrow[(I,V)]{} \!\!\!\!\rightarrow^{\mathcal{R}} N \ ,$$

completing the inductive step. ♦

3.8 Avoiding the well-separation restriction

In this section we develop a strengthened finiteness of developments theory for Let which is not subject to a well-separation restriction. Although such a theory is not uninteresting in itself, it is not especially more useful than the results of Section 3.3 for the present chapter. However, it will be quite useful in the extension of Let to Need, where the (C, A) rules play a more central rôle.

We move from Let to the *generalized* call-by-let calculus $\mathsf{Let}^{\star}$. Figure 3.7 shows the syntax and basic reduction rules, with compatible closure as usual. Terms in $\mathsf{Let}^{\star}$ are just the same as Let, but we allow certain multistep reduction sequences to be condensed into single steps. In a $(\star)$ contraction, we define the *rank* of the step to be the number of prefixes moved; for example the step $(P^n\, M)\, N \to P^n\, (M\, N)$ has rank n. Reduction in $\mathsf{Let}^{\star}$ clearly corresponds to reduction in Let.

Proposition 3.8.1. Let $M, N \in \mathsf{Let}$. Then $M \xrightarrow[\mathsf{Let}]{} \!\!\!\!\rightarrow N$ if and only if $M \xrightarrow[\mathsf{Let}^{\star}]{} \!\!\!\!\rightarrow N$.

Proof: In the forward direction, we simple replace each (C, A) step with a rank 1 $(\star C, \star A)$ step; in the backward direction, we replace a $(\star C, \star A)$ of rank n with n consecutive (C, A) steps. ♦

That proposition allows us to transfer many properties back and forth from Let to $\mathsf{Let}^{\star}$ with little effort. For example, since Let and $\mathsf{Let}^{\star}$ share a term language and typing rules, subject reduction is immediate:

Corollary 3.8.2. Reduction in $\mathsf{Let}^{\star}$ satisfies the subject reduction property: let

$$\Gamma \vdash_{\mathsf{Let}^{\star}} M : A$$

and $M \xrightarrow[\mathsf{Let}^{\star}]{} N$. Then $\Gamma \vdash_{\mathsf{Let}^{\star}} N : A$.

Syntactic domains

Terms	L, M, N	$::=$	$V \mid M\,N \mid \text{let } x = M \text{ in } N$
Values	V, W	$::=$	$x \mid \lambda x.\, M$
Prefixes	P	$::=$	$\text{let } x = M \text{ in}$

Typing Judgments As in Let.

Reduction Rules

(I)	$(\lambda x.\, M)\, N$	$\to$	$\text{let } x = N \text{ in } M$
(V)	$\text{let } !x = V \text{ in } M$	$\to$	$M[x := V]$
$(\star C)$	$(P^n\, M)\, N$	$\to$	$P^n\,(M\, N)$
$(\star A)$	$\text{let } !x = (P^n\, M) \text{ in } N$	$\to$	$P^n\,(\text{let } !x = M \text{ in } N)$

Figure 3.7: The generalized call-by-let calculus, Let*.

$$\begin{array}{rcl}
\mathbf{d}(M \to N, 0) & = & 0 \\
\mathbf{d}((\lambda x.\, M)\, N \to \text{let } x = N \text{ in } M, n) & = & 0 \\
\mathbf{d}(\text{let } x = V \text{ in } M \to M[x := V], n) & = & -1 \ \ , \ n \geq 0 \\
\mathbf{d}(\text{let } x = (P^m\, M) \text{ in } N \to P^n\,(\text{let } x = M \text{ in } N), n) & = & m \ \ , \ n \geq 0 \\
\mathbf{d}((P^m\, M)\, N \to P^m\,(M\, N), n) & = & 0 \\
\mathbf{d}(P^m\, M \to M, n) & = & -m \ \ , \ n \geq 0 \\
\mathbf{d}(L\, M \to L\, N, n) & = & 0 \\
\mathbf{d}(M\, L \to N\, L, n) & = & 0 \\
\mathbf{d}(\text{let } x = L \text{ in } M \to \text{let } x = L \text{ in } N, n) & = & \mathbf{d}(M \to N, n-1) \ \ , \\
 & & \qquad\qquad n \geq 0 \\
\mathbf{d}(\text{let } x = M \text{ in } L \to \text{let } x = N \text{ in } L, n) & = & 0 \\
\mathbf{d}(\lambda x.\, M \to \lambda x.\, N, n) & = & 0
\end{array}$$

Figure 3.8: The damage function for Let*.

Proof: From SR(Let) by the above. ♦

The interaction of substitution and reduction in this system is as in the other systems:

Lemma 3.8.3. Let $M' \xrightarrow[\mathsf{Let}^{\star\prime}]{} N'$. Then:

1. $M'[x := V'] \xrightarrow[\mathsf{Let}^{\star\prime}]{} N'[x := V']$.
2. $L'[x := M'] \twoheadrightarrow_{\mathsf{Let}^{\star\prime}} L'[x := N']$.

Proof: Trivial, again noting the restriction to values in the first clause. ♦

Generalizing the commuting conversion reductions in the manner of this system allows us to state an FD! theorem with no well-separation restriction, which we set forth in the next section. We will need the notion of a *damage* function to keep track of bindings inserted into or removed from a nested series. The damage function subsumes the notion of the rank of a top-level $(*)$ step, but discards the rank of any internal $(*)$ steps when they are not within the bodies of a specified number of bindings, and includes any bindings removed by non-$(*)$ steps in that range. This function $\mathbf{d}$ for single Let* reduction steps is presented in Figure 3.8.

The first argument to $\mathbf{d}$ is the reduction sequence which we will consider, and the second argument is the number of top-level right-nested let-bindings we consider. So if we take $n = 2$ for a sequence beginning with the term

$$\text{let } x_0 = M_0 \text{ in let } x_1 = M_1 \text{ in } M_2 \ \ ,$$

then we are interested in the number of let-bindings the system introduces or removes between the top level and the residual of M_2. If (say) M_0 is a value

Syntactic domains

$$
\begin{array}{llcl}
\text{Terms} & L', M', N' & ::= & V' \mid M'\,N' \mid \text{let } x = M' \text{ in } N' \\
 & & \mid & {}^{[I]}(\lambda x.\, M')\, N' \\
 & & \mid & {}^{[V]}\text{let } x = V' \text{ in } N' \\
 & & \mid & {}^{[\star C_n]}(\mathcal{P}^n\, M')\, N' \\
 & & \mid & {}^{[\star A_n]}\text{let } x = \mathcal{P}^n\, M' \text{ in } N' \\
\text{Values} & V', W' & ::= & x \mid \lambda x.\, M' \\
\text{Markable prefixes} & \mathcal{P} & ::= & \text{let } x = M' \text{ in} \\
 & & \mid & {}^{[V]}\text{let } x = V' \text{ in} \\
 & & \mid & {}^{[\star A_n]}\text{let } x = \mathcal{P}^n\, M' \text{ in}
\end{array}
$$

Degenerate Equivalences

$$
\begin{array}{rcl}
{}^{[\star A_0]}\text{let } x = M' \text{ in } N' & \equiv & \text{let } x = M' \text{ in } N' \\
{}^{[\star C_0]}M'\, N' & \equiv & M'\, N'
\end{array}
$$

Reduction rules

$$
\begin{array}{lllll}
(I_0) & {}^{[I]}(\lambda x.\, M')\, N' & \to & \text{let } x = N' \text{ in } M' \\
(I_1) & (\lambda x.\, M')\, N' & \to & \text{let } x = N' \text{ in } M' \\
(V_0) & {}^{[V]}\text{let } x = V' \text{ in } M' & \to & M'[x := V'] \\
(V_1) & \text{let } x = V' \text{ in } N' & \to & M'[x := V'] \\
(\star C_0) & {}^{[\star C_n]}(\mathcal{P}^n\, M')\, N' & \to & \mathcal{P}^n\,(M'\, N') \\
(\star C_0^<) & {}^{[\star C_n]}(\mathcal{P}^{n-m}\, \mathcal{P}^m\, M')\, N' & \to & \mathcal{P}^{n-m}\,({}^{[\star C_m]}(\mathcal{P}^m\, M')\, N') \\
 & & & \text{ where } m, n > 0 \\
(\star C_1) & (\mathcal{P}^n\, M')\, N' & \to & \mathcal{P}^n\,(M'\, N') \\
(\star A_0) & {}^{[\star A_n]}\text{let } x = (\mathcal{P}^n\, M') \text{ in } N' & \to & \mathcal{P}^n\,(\text{let } x = M' \text{ in } N') \\
(\star A_0^<) & {}^{[\star A_n]}\text{let } x = (\mathcal{P}^{n-m}\, \mathcal{P}^m\, M') & \to & \mathcal{P}^{n-m}\,({}^{[\star A_m]}(\text{let } x = \mathcal{P}^m\, M' \\
 & \quad \text{in } N' & & \qquad \text{in } N')) \\
 & & & \text{where } m, n > 0 \\
(\star A_1) & \text{let } x = (\mathcal{P}^n\, M') \text{ in } N' & \to & \mathcal{P}^n\,(\text{let } x = M' \text{ in } N')
\end{array}
$$

Figure 3.9: The generalized marked call-by-let calculus $\mathsf{Let}^{\star\prime}$.

and the sequence contracts that binding, then $\mathbf{d}$ would return a value of -1. If $M+1 \equiv (P^m\, N$, and the contraction is a $(\star A)$ step on that binding, then $\mathbf{d}$ would return m.

3.8.1 Generalized developments

Figure 3.9 presents the syntax and basic reduction relation of the general marked call-by-let calculus $\mathsf{Let}^{\star\prime}$; Figure 3.10 presents the rules for compatible closure of the reduction relation, which rely on the damage function $\mathbf{d}$ of Figure 3.8. As usual, we take $\mathsf{Let}^{\star\prime}_0$ reduction to mean any step or sequence by the $(I_0, V_0, \star C_0, \star C_0^<, \star A_0, \star A_0^<)$ rules, and $\mathsf{Let}^{\star\prime}_1$ reduction to refer to reduction by any other rules.

We define the projection $|\,.\,|$ of $\mathsf{Let}^{\star\prime}$ terms and sequences into $\mathsf{Let}^{\star}$ as usual, and the projection lemma is trivial.

Lemma 3.8.4. Let $M' \xrightarrow[\mathsf{Let}^{\star\prime}]{} N'$; then $|M'| \xrightarrow[\mathsf{Let}^{\star}]{} |N'|$.

Proof: It is clear from inspection of the various corresponding rules that one can simply erase the markings from $\mathsf{Let}^{\star\prime}$ to get $\mathsf{Let}^{\star}$ steps. ♦

Lifting redexes from $\mathsf{Let}^{\star}$ to $\mathsf{Let}^{\star\prime}$ from an initial marking is also straightforward.

Lemma 3.8.5. Let $M \xrightarrow[\mathsf{Let}^{\star}]{} N$, and let $M' \in \mathsf{Let}^{\star\prime}$ such that $|M'| \equiv M$. Then there exists some $N' \in \mathsf{Let}^{\star\prime}$ such that $M' \xrightarrow[\mathsf{Let}^{\star\prime}]{} N'$ and $|N'| \equiv N$.

Proof: As usual, by giving the appropriate indexes to N'. ♦

We have the usual results on substitution and reduction:

1. Let $M' \to N'$. Then:

   (a) $L'\, M' \to L'\, N'$.

   (b) $M'\, L' \to N'\, L'$.

   (c) let $x = L'$ in $M' \to$ let $x = L'$ in N'.

   (d) let $x = M'$ in $L' \to$ let $x = N'$ in L'.

   (e) $\lambda x.\, M' \to \lambda x.\, N'$.

   (f) ${}^{[\star C_n]}\mathcal{P}^n\, L' M' \to {}^{[\star C_n]}\mathcal{P}^n\, L' N'$.

   (g) ${}^{[\star A_n]}\mathcal{P}^n\, L' M' \to {}^{[\star A_n]}\mathcal{P}^n\, L' N'$.

2. Let $\sigma : \mathcal{P}^n\, M' \to N'$, where $n > 0$. Then:

   (a) ${}^{[\star C_n]}(\mathcal{P}^n\, M')\, L' \to {}^{[\star C_{n+\mathrm{d}(\sigma)}]}N'\, L'$.

   (b) ${}^{[\star A_n]}$let $!x = \mathcal{P}^n\, M'$ in $L' \to {}^{[\star A_{n+\mathrm{d}(\sigma)}]}$let $!x = N'$ in L'.

Figure 3.10: Compatible closure in $\mathsf{Let}^{\star\prime}$.

Lemma 3.8.6. Let $M' \xrightarrow[\mathsf{Let}^{\star\prime}_0]{} N'$. Then:

1. $M'[x := V'] \xrightarrow[\mathsf{Let}^{\star\prime}_0]{} N'[x := V']$.
2. $L'[x := M'] \twoheadrightarrow_{\mathsf{Let}^{\star\prime}_0} L'[x := N']$.

Proof: Trivial, as in the previous systems, since the substitution will not disrupt markings. ♦

The notion of residual and development are similar to previous systems, although the separation of redex annotation from the representation of the term itself is more tricky since marks now contain an integer to indicate the (largest possible) rank of the step; we likewise extend the metanotation γ to range over pairs of paths and natural numbers: $\gamma \equiv (\delta, n)$, for example, and $\bar{\gamma}$ to range over sets containing both paths and pairs. We will sometimes abuse this notation by referring to both a pair and the path it contains by the same name: $\gamma \equiv (\gamma, m)$. Let $\partial(M)$ denote the set of all valid paths into an Let term M. Then a marking $\bar{\gamma}$ for a term M is a subset of $\partial(M) \cup (\partial(M) \times \mathbb{N})$ where:

1. For all paths $\gamma \in \bar{\gamma}$ not in a pair, the subterm $M|_\gamma$ is a valid (I, V) redex.
2. For all pairs $(\gamma, n) \in \bar{\gamma}$, the subterm $M|_\gamma$ is a valid $(\star C, \star A)$ redex of rank n.
3. For all pairs $(\gamma, n_1), (\gamma, n_2) \in \bar{\gamma}$, we must have $n_1 = n_2$.

The last condition ensures that a single term is not multiply marked.

As usual, we have designed the notion of a valid marking to correspond with marked terms, and we can refer to a marked term or to the corresponding unmarked term and valid marking interchangeably.

Lemma 3.8.7. Let $M \in \mathsf{Let}^{\star}$ and $N' \in \mathsf{Let}^{\star\prime}$. Then $M \equiv |N'|$ if and only if there exists a valid marking $\bar{\gamma}$ for M such that $(M, \bar{\gamma}) \equiv N'$.

Proof: Clear from the definition of valid markings, whose last condition ensures that no redex is doubly-marked. ♦

This complication settled, we define residuals and developments as usual.

Definition 3.8.8. Let $\bar{\gamma}$ be a valid marking for $M \in \mathsf{Let}^{\star}$, and let $\sigma : M \twoheadrightarrow_{\mathsf{Let}^{\star}} N$. By Lemma 3.8.5 we have some term $N' \in \mathsf{Let}^{\star\prime}$ such that $|N'| \equiv N$ and $M' \twoheadrightarrow_{\mathsf{Let}^{\star\prime}} N'$, and by Lemma 3.8.7 we can separate N' as $N' \equiv (N, \bar{\delta})$. Then the residual of $\bar{\gamma}$ with respect to σ, written $\bar{\gamma}/\sigma$, is defined to be $\bar{\delta}$. ◊

Definition 3.8.9. Let $M' \equiv (M, \bar{\gamma}) \in \mathsf{Let}^{\star\prime}$, and let $\sigma : M' \xrightarrow[\delta]{} N'$. Then σ is a *development* of M' whenever δ is the path underlying some member of $\bar{\gamma}$. A multistep reduction sequence is a development if each individual step is a development of the residual of the marking to the beginning of that step. A development is *complete* if the final term in the sequence is unmarked. ◊

Lemma 3.8.10. $\mathsf{Let}^\star$ developments and ${\mathsf{Let}^\star}'_0$ sequences coincide.

Proof: By Lemma 3.8.7, since ${\mathsf{Let}^\star}'_0$ sequences contract only marked redexes. ♦

3.8.2 Finiteness of generalized developments

Much of the basic finiteness of developments result follows from this next assertion in the same way that many other results are facilitated by Proposition 3.8.1:

Lemma 3.8.11. Let $M' \xrightarrow[{\mathsf{Let}^\star}'_0]{} N'$. Then $M' \xrightarrow[(I_0,V_0,A,C)]{} N'$.

Proof: Trivial, by inspection of the reduction rules. ♦

Corollary 3.8.12. ${\mathsf{Let}^\star}'_0$ reduction is strongly normalizing.

Proof: From Corollary 3.3.16 by Lemma 3.8.11. ♦

Corollary 3.8.13, $\mathrm{FD}(\mathsf{Let}^\star)$**.** All $\mathsf{Let}^\star$ developments are finite. ♦

3.8.3 Uniqueness of complete generalized developments

We begin this section with two examples: since our aim for this more complicated notion of marking is to improve the flow of proofs, we show how the bumpy examples at the beginning of Section 3.3.3 of problems for Let' are in fact not problematic here.

Example 3.8.1. We consider a marking of the outermost (C) and (A) steps in the term

$$(\text{let } x = P\, L' \text{ in } M')\, N' \ \ .$$

Taking the markings to have rank 1, we have

$$^{[\star C_1]}(^{[\star A_1]}\text{let } x = P\, L' \text{ in } M')\, N' \ \ .$$

Contracting the (C) step first, we complete the development as:

$$\begin{array}{ll} & ^{[\star C_1]}(^{[\star A_1]}\text{let } x = P\, L' \text{ in } M')\, N' \\ \xrightarrow[(C_0)]{} & ^{[\star A_1]}\text{let } x = P\, L' \text{ in } (M'\, N') \\ \xrightarrow[(A_0)]{} & P\,(\text{let } x = L' \text{ in } M'\, N') \ \ . \end{array}$$

Contracting the (A) step first, we have the following completion:

$$\begin{array}{ll} & ^{[\star C_1]}(^{[\star A_1]}\text{let } x = P\, L' \text{ in } M')\, N' \\ \xrightarrow[(A_0)]{} & ^{[\star C_2]}(P\,(\text{let } x = L' \text{ in } M'))\, N' \\ \xrightarrow[(C_0^<)]{} & P\, ^{[\star C_1]}((\text{let } x = L' \text{ in } M')\, N') \\ \xrightarrow[(C_0)]{} & P\,(\text{let } x = L' \text{ in } (M'\, N')) \ \ . \end{array}$$

Clearly the completion is unique. ◊

Example 3.8.2. For the interfering (A) steps in the term

$$\text{let } x = (\text{let } y = (P\, L') \text{ in } M') \text{ in } N' \ \ ,$$

we again assign the 1-rank markings

$$^{[\star A_1]}\text{let } x = (^{[\star A_1]}\text{let } y = (P\, L') \text{ in } M') \text{ in } N' \ \ .$$

The completions have a unique final term: taking the inner redex first, we have

$$\begin{array}{ll} & ^{[\star A_1]}\text{let } x = (^{[\star A_1]}\text{let } y = (P\, L') \text{ in } M') \text{ in } N' \\ \xrightarrow[(A_0)]{} & ^{[\star A_2]}\text{let } x = P\,(\text{let } y = L' \text{ in } M') \text{ in } N' \\ \xrightarrow[(A_0^<)]{} & P\,(^{[\star A_1]}\text{let } x = (\text{let } y = L' \text{ in } M') \text{ in } N') \\ \xrightarrow[(A_0)]{} & P\,(\text{let } x = (\text{let } y = L' \text{ in } M') \text{ in } N') \ \ . \end{array}$$

Contracting the outermost redex first, we have:

$$\begin{array}{ll} & {}^{[\star A_1]}\text{let } x = ({}^{[\star A_1]}\text{let } y = (P\,L') \text{ in } M') \text{ in } N' \\ \xrightarrow[(A_0)]{} & {}^{[\star A_1]}\text{let } x = P\,(\text{let } y = L' \text{ in } M') \text{ in } N' \\ \xrightarrow[(A_0)]{} & P\,(\text{let } x = (\text{let } y = L' \text{ in } M') \text{ in } N') \ . \end{array}$$

◊

We will show that $\mathsf{Let}^{\star\prime}{}_0$ reduction is weakly confluent, from which full confluence is immediate (by Lemma 2.1.5 with Corollary 3.8.12); the strengthened finiteness of developments result then follows.

Lemma 3.8.14. $\mathsf{Let}^{\star\prime}{}_0$ reduction is weakly confluent.

Proof: We consider the case where one top-level redex contains a second distinct contraction:

- *Top-level (I_0) reduction.* This case is straightforward. Let

  $$M' \equiv {}^{[I_0]}(\lambda x.\, N_0')\, N_1'$$

  and $M_1' \equiv \text{let } x = N_1' \text{ in } N_0'$. By inspection of the rules we must have the second redex within either N_0' or N_1'; in either case the result is immediate.

- *Top-level (V_0) reduction.* In this case the result follows immediately from Lemma 3.8.6.

- *Top-level $(\star C_0)$ reduction.* Let

  $$\begin{array}{rcl} M & \equiv & {}^{[\star C_n]}(\mathcal{P}^n\, N_0')\, N_1' \text{ and} \\ M_1 & \equiv & \mathcal{P}^n\,(N_0'\, N_1') \ . \end{array}$$

  The following cases are straightforward if not trivial:

  – $M_2' \equiv {}^{[\star C_n]}(\mathcal{P}^n\, N_2')\, N_1'$ *where* $N_0' \to N_2'$. Then $N' \equiv \mathcal{P}^n\,(N_2'\, N_1')$.
  – $M_2' \equiv {}^{[\star C_n]}(\mathcal{P}^n\, N_0')\, N_2'$ *where* $N_1' \to N_2'$. Then $N' \equiv \mathcal{P}^n\,(N_0'\, N_2')$.
  – $M_2' \equiv {}^{[\star C_n]}(\mathcal{P}_0^{m_0}\,(\text{let } x = N_3' \text{ in } \mathcal{P}_1^{m_1}\, N_0'))\, N_1'$ *where*

  $$\begin{array}{rcl} m_0 + m_1 & = & n - 1 \\ \mathcal{P}^n & \equiv & \mathcal{P}_0^{m_0}\,(\text{let } x = N_2' \text{ in } \mathcal{P}_1^{m_1}) \ \textit{and} \\ N_2' & \to & N_3' \ . \end{array}$$

  Then simply $N \equiv \mathcal{P}_0^{m_0}\,(\text{let } x = N_3' \text{ in } \mathcal{P}_1^{m_1}\,(N_0'\, N_1')))$.
  – *Where the second redex is a top-level $(\star C_0^{<})$ step with*

  $$\begin{array}{rcl} M & \equiv & {}^{[\star C_n]}(\mathcal{P}_0^{n-m}\, \mathcal{P}_1^m\, N_0')\, N_1' \ \text{, and} \\ M_2' & \equiv & \mathcal{P}_0^{n-m}\, {}^{[\star C_m]}(\mathcal{P}_1^m\, N_0')\, N_1' \ . \end{array}$$

  Then $N \equiv M_1'$.

  For the remaining cases the damage function and its effect on compatible closure become relevant. We have that the contraction to M_2' must be a $(V_0, \star A_0)$ step since only these rules are applicable at the top-level to let-bindings, and consider each separately. In both cases we take $m_0 + m_1 = n - 1$ and $\mathcal{P}^n \equiv \mathcal{P}_0^{m_0}\, \mathcal{P}_1\, \mathcal{P}_2^{m_1}\, N_2'$; we will take $\mathcal{P}_1$ to be marked respectively $[V]$ and $[\star A]$ in each case and construct the common reduct N'.

  – $\mathcal{P}_1 \equiv {}^{[V]}\text{let } x = V' \text{ in}$. Considering M without the topmost mark, we have

  $$\begin{array}{rl} \sigma : & (\mathcal{P}_0^{m_0}\,({}^{[V]}\text{let } x = V' \text{ in } (\mathcal{P}_2^{m_1}\, N_2')\, N_0'))\, N_1' \\ \to & (\mathcal{P}_0^{m_0}\,((\mathcal{P}_2^{m_1}\, N_2')[x := V']\, N_0'))\, N_1' \ , \end{array}$$

  and since $n > m_0$ we have $\mathbf{d}(\sigma, n) = -1$. So the mark on M_2 has rank $n - 1$:

  $$M_2 \equiv {}^{[\star C_{n-1}]}(\mathcal{P}_0^{m_0}\,((\mathcal{P}_2^{m_1}\, N_2')[x := V']\, N_0'))\, N_1' \ .$$

  Then moreover since the substitution of V' for x will not affect the number of prefixes in $\mathcal{P}_2^{m_1}$, we can take

  $$N' \equiv \mathcal{P}_0^{m_0}\,((\mathcal{P}_2^{m_1}\, N_2')[x := V']\,(N_0'\, N_1'))$$

  and the case is complete.

- $\mathcal{P}_1 \equiv {}^{[\star A_l]}\text{let } x = \mathcal{R}^l\, N_2 \text{ in}$. As for the preceding case of a (V_0) step, except that the damage function returns some positive integer.

- *Top-level* $(\star C_0^<, \star A_0, \star A_0^<)$ *reduction.* As in top-level $(\star C_0)$ reduction.

Then since the cases of compatible closure and of redexes in disjoint subterms are trivial, the proof is complete. ♦

Corollary 3.8.15. $\mathsf{Let}^{\star\prime}{}_0$ reduction is confluent.

Proof: By Lemma 2.1.5 with Corollary 3.8.12 and Lemma 3.8.14. ♦

Proposition 3.8.16, $\mathrm{FD!}(\mathsf{Let}^\star)$**.** All $\mathsf{Let}^\star$ developments are finite and can be extended to complete developments, and all complete $\mathsf{Let}^\star$ developments of a given terms and marking end in the same term.

Proof: By Lemma 2.1.4 with $\mathrm{FD}(\mathsf{Let}^\star)$ and Corollary 3.8.15. ♦

3.8.4 Joinability of markings

One final hurdle remains for a proof of confluence based on $\mathrm{FD!}(\mathsf{Let}^\star)$: we must demonstrate that all markings are joinable. The result is not trivial. Under the usual definition of set union $\cup$, unions of valid markings are not necessarily valid. For example, both

$$^{[\star A_1]}\text{let } x = \mathcal{P}\,\mathcal{R}\, M' \text{ in } N'$$

and

$$^{[\star A_2]}\text{let } x = \mathcal{P}\,\mathcal{R}\, M' \text{ in } N'$$

are valid, but we have no way to write the term marked with the union of the sets corresponding to these two terms, and the union is not valid. So instead of the usual notion $\cup$ of set union, we will use a slightly different notion $\Cup$, which resolves this problem by simply taking the larger of the two ranks.

Algorithm 3.8.17. Let $\bar{\gamma}, \bar{\delta}$ be valid marking sets of a term $M' \in \mathsf{Let}^{\star\prime}$. We inductively construct $\bar{\gamma} \Cup \bar{\delta}$ as follows:

1. $\{\} \Cup \bar{\gamma} = \bar{\gamma}$.
2. If $\bar{\gamma} = \{\gamma_0\} \cup \bar{\gamma}_1$ where γ_0 is not the underlying path of any element of $\bar{\gamma}_1$
   Then $\bar{\gamma} \Cup \bar{\delta} = \{\gamma_0\} \cup (\bar{\gamma}_1 \Cup \bar{\delta})$.
3. If $\bar{\gamma} = \{(\bar{\gamma}_0, n)\} \cup \bar{\gamma}_1$ where γ_0 is not the underlying path of any element of $\bar{\gamma}_1$
   Then:
   (a) If there exists some $m > 1$ such that $\bar{\delta} = \{(\gamma_0, m)\} \cup \bar{\delta}_1$ where γ_0 is not the underlying path of any element of $\bar{\delta}_1$
   Then $\bar{\gamma} \Cup \bar{\delta} = \{(\gamma_0, \max\{m, n\})\} \cup (\bar{\gamma}_1 \Cup \bar{\delta}_1)$.
   (b) Otherwise $\bar{\gamma} \Cup \bar{\delta} = \{(\gamma_0, n)\} \cup (\bar{\gamma}_1 \Cup \bar{\delta})$.

Lemma 3.8.18. For all valid marking sets $\bar{\gamma}, \bar{\delta}$ of M', the algorithm for constructing $\bar{\gamma} \Cup \bar{\delta}$ always returns a finite set which is also valid.

Proof: Termination is ensured since all valid sets are finite, and since each recursive call uses a strictly smaller set. Validity is clear since in the same term we could not have coincident redexes for two different rules in each set, for example

$$\begin{array}{rcl} \gamma_0 & \in & \bar{\gamma} \\ (\gamma_0, n) & \in & \bar{\delta}\ , \end{array}$$

which would indicate that$\bar{\gamma}$ indexes a (I) or (V) step at γ, while $\bar{\delta}$ indexes a $(\star C)$ or $(\star A)$ redex at γ. ♦

It is also clear that $\Cup$ is idempotent, symmetric and transitive. Along with the new notion of union, we have an alternate notion of a subset:

Definition 3.8.19. We write $\bar{\gamma} \Subset \bar{\delta}$ if both of the following are true:

1. For all $\gamma \in \bar{\gamma}$, $\gamma \in \bar{\delta}$.

2. For all $(\gamma, n) \in \bar{\gamma}$, there exists some $m \geq n$ such that $(\gamma, m) \in \bar{\delta}$.

$\Diamond$

The relation $\Subset$ has two key properties: by Lemma 3.8.20, supersets preserve developments, and by Lemma 3.8.21, the result of an application of $\Cup$ has its components as $\Subset$-subsets.

Lemma 3.8.20. Let $\sigma : (M', \bar{\gamma}_0) \xrightarrow[\text{dev}]{} (N', \bar{\gamma}_1)$, and let $\bar{\gamma}_0 \Subset \bar{\delta}_0$. Then there exists some $\bar{\delta}_1$ such that $\delta : (M', \bar{\delta}_0) \xrightarrow[\text{dev}]{} (N', \bar{\delta}_1)$.

Proof: Either the step is just the same, or a $(\star C_0, \star A_0)$ step in σ becomes a $(\star C_0^{<}, \star A_0^{<})$ step in δ. ♦

Lemma 3.8.21. For all valid $\bar{\gamma}, \bar{\delta}$ we have $\bar{\gamma} \Subset (\bar{\gamma} \Cup \bar{\delta})$.

Proof: Clear from the definitions of the operator and relation. ♦

We can now use $\Cup$-union for validity in combining marking sets for the confluence proof.

Lemma 3.8.22. Let $\bar{\gamma}, \bar{\delta}$ be valid marking sets of M' with $(M', \bar{\gamma}) \xrightarrow[\text{dev}]{} \!\!\!\!\!\rightarrow (N', \bar{\gamma}_1)$. Then there exists some $\bar{\delta}_1$ such that $(M', \bar{\gamma} \Cup \bar{\delta}) \xrightarrow[\text{dev}]{} \!\!\!\!\!\rightarrow (N', \bar{\delta}_1)$.

Proof: Follows from Lemma 3.8.20 and Lemma 3.8.21. ♦

Corollary 3.8.23. Let $\bar{\gamma}, \bar{\delta}$ be valid marking sets of M' with $\sigma' : (M', \bar{\gamma}) \xrightarrow[\text{cpl}]{} N'$. Then there exists some $\bar{\delta}_0$ such that $\tau' : (M', \bar{\gamma} \Cup \bar{\delta}) \xrightarrow[\text{dev}]{} \!\!\!\!\!\rightarrow (N', \bar{\delta}_0)$ and $|\sigma'| \equiv |\tau'|$.

Proof: Immediate since the complete development is just a particular partial development. ♦

Corollary 3.8.24. All $\mathsf{Let}^\star$-markings are joinable.

Proof: By the previous result we clearly satisfy the definition of joinability. ♦

Confluence of $\mathsf{Let}^\star$ is now straightforward:

Proposition 3.8.25, $\mathrm{CR}(\mathsf{Let}^\star)$. Reduction in $\mathsf{Let}^\star$ is confluent.

Proof: By Lemma 2.1.12 with $\mathrm{FD!}(\mathsf{Let}^\star)$ and Corollary 3.8.24. ♦

The proof of confluence for Let also follows from this result.

Alternate proof of $\mathrm{CR}(\mathsf{Let})$*, Proposition 3.4.4:*
Follows from $\mathrm{CR}(\mathsf{Let}^\star)$ by Proposition 3.8.1 and Corollary 3.8.24. ♦

The marking scheme makes no difference to the proof of standardization is this chapter: since our standard reduction relation contains only (I, V) steps, we can formulate the proof in the usual manner and not cross the well-separation restriction. However in the next chapter, the standard evaluation rules for call-by-need will include the (A) rule, and the technique of this section will become quite useful.

Chapter 4
Call-by-need

ASIDE from call-by-name and call-by-value/let, we have a third flavor of parameter-passing mechanism: call-by-need. Essentially, call-by-need combines the observational properties of call-by-name with the copying behavior of call-by-value. Figure 4.1 defines the call-by-need calculus Need as a slight modification of the call-by-let calculus Let. The only change is the addition of a new reduction rule (G), which allows unneeded computations to be discarded. The label (G) abbreviates *garbage collection*. We first explored this system in an untyped framework with Odersky and Wadler [53, 54]. Here, as with call-by-let in the previous chapter, we give new proofs of the previous results, which are systematic and easier to follow than the earlier arguments since the more generalized notion of marking introduced in Section 3.8 allows the earlier proof outlines to be reused again, especially for the proof of the standard evaluation strategy.

In the following three sections of this chapter we review the basic syntactic properties of call-by-need: substitutions, subject reduction and confluence. In Section 4.4 we develop generalized reduction and finite developments theories in the manner of Let*. We have two applications of these FD! results: the equivalence of call-by-name and call-by-need in Section 4.5, and the standardization results of Section 4.6. The latter fulfills the motivation of generalized reduction: with only the weaker notion of generalized developments requiring well-separation, it is not clear how to avoid a purely brute-force approach to the standardization result. Finally, in Section 4.7 we conclude with a comparison of our system with other approaches to call-by-need.

4.1 Substitution properties

We have the usual substitution results for Need. As with Val and Let, the interaction of substitution and reduction in valid in general only for the substitution of a variable (a value) for another value.

Lemma 4.1.1. Let $\Gamma, x : B \vdash_{\mathsf{Need}} M : A$ and $\Delta \vdash_{\mathsf{Need}} N : B$. Then $\Gamma, \Delta \vdash_{\mathsf{Need}} M[x := N] : B$.

Lemma 4.1.2. Let $L, M, N, V \in \mathsf{Need}$, where $M \xrightarrow[\mathsf{Need}]{} N$. Then:

1. $M[x := V] \xrightarrow[\mathsf{Need}]{} N[x := V]$.

2. $L[x := M] \xrightarrow[\mathsf{Need}]{} \!\!\!\!\rightarrow L[x := N]$.

Proof: Both lemmas are trivial, as in the call-by-let versions. ♦

4.2 Subject reduction

Proposition 4.2.1. Need satisfies subject reduction.

Proof: As in Proposition 3.2.1 for Let, with one more case for the (G) rule: if we have

$$\dfrac{\begin{array}{c}\vdots\, \mathcal{P}_1 \\ \Gamma \vdash_{\mathsf{Need}} M : A\end{array} \quad \begin{array}{c}\vdots\, \mathcal{P}_2 \\ \Delta, x : A \vdash_{\mathsf{Need}} N : B\end{array}}{\Gamma, \Delta \vdash_{\mathsf{Need}} \mathrm{let}\ x = M\ \mathrm{in}\ N : B}\ \mathrm{Let}$$

where $x \notin \mathrm{fv}(N)$, then by a simple induction on the depth of $\mathcal{P}_2$ we can construct some $\mathcal{P}_2'$ which implies $\Delta \vdash_{\mathsf{Need}} N : B$ and which is just the same as $\mathcal{P}_2$ without

Call-by-need

Syntactic domains As for Let.

Typing judgments As for Let.

Reduction relation As for Let, plus the following:

$$(G) \qquad \text{let } x = M \text{ in } N \quad \to \quad N, \qquad \text{if } x \text{ is not free in } N$$

Figure 4.1: The call-by-need lambda calculus Need.

the introduction of $x : A$ by Weakening; then

$$\cfrac{\cfrac{\begin{array}{c}\vdots\ \mathcal{P}_2'\\ \Delta \vdash_{\mathsf{Need}} N : B\end{array}}{\vdots}\ \text{Weakening}}{\Gamma, \Delta \vdash_{\mathsf{Need}} N : B}\ \text{Weakening}$$

completes the case. ♦

4.3 Confluence

We can take advantage of the similarity of Need to Let to give a quick proof of confluence for Need. We need only show that (G) steps by themselves are confluent, which is trivial, and that they commute with the remaining steps.

Lemma 4.3.1. Reduction by (G) steps is confluent.

Proof: Trivial: we can show by inspection that if $M \xrightarrow[(G)]{} M_1$ and $M \xrightarrow[(G)]{} M_2$, then we have some N such that for both i, $M_i \xrightarrow[(G)]{}^{=} N$. The full result follows by induction. ♦

Lemma 4.3.2. Let $M, M_1, M_2 \in \mathsf{Need}$ such that $M \xrightarrow[(G)]{} \!\!\!\!\to M_1$ and $M \xrightarrow[(I,V,C,A)]{} M_2$. Then there exists some N such that $M_1 \xrightarrow[(I,V,C,A)]{}^{=} N$ and $M_2 \xrightarrow[(G)]{} \!\!\!\!\to N$. Graphically:

$$\begin{tikzcd} M \arrow[r, "(I,V,C,A)"'] \arrow[d, two heads, "(G)"] & M_2 \arrow[d, dotted, two heads, "(G)"] \\ M_1 \arrow[r, dotted, "="', "(I,V,C,A)"'] & \exists N \end{tikzcd}$$

Proof: By induction on the structure of M, searching for the top-level (I, V, C, A) step. We have that $M_1 \equiv N$ exactly when the let-binding mentioned in a top-level (V, C, A) step is also one of the let-bindings contracted in a (G) step. ♦

Proposition 4.3.3, $\mathrm{CR}(\mathsf{Need})$**.** Need is confluent.

Proof: Follows from $\mathrm{CR}(\mathsf{Let})$, Lemma 4.3.1 and Lemma 4.3.2 by Lemma 2.1.14. ♦

A second, more limited confluence result will be useful in some of the later proofs, by allowing us to assert that all (V, C, A, G)-normal form are unique.

Lemma 4.3.4. (V, C, A, G) reduction on Need terms is confluent.

Proof: By a slight modification of Lemma 3.4.3 and $\mathrm{CR}(\mathsf{Need})$; we can straightforwardly omit (I) steps without any complications. ♦

Syntactic domains As in $\mathsf{Let}^\star$.
Typing Judgments As in Let.
Reduction Rules As in $\mathsf{Let}^\star$, plus:

$$(G) \qquad \text{let } x = M \text{ in } N \quad \to \quad N \qquad \text{if } x \notin \text{fv}(N)$$

Damage function As in $\mathsf{Let}^\star$.

Figure 4.2: The generalized call-by-need calculus, $\mathsf{Need}^\star$.

Syntactic domains As in $\mathsf{Let}^{\star\prime}$.
Reduction Rules As in $\mathsf{Let}^{\star\prime}$, plus:

$$\begin{array}{llcll}
(G_{1a}) & \text{let } x = M \text{ in } N & \to & N & \text{if } x \notin \text{fv}(N) \\
(G_{1b}) & {}^{[V]}\text{let } x = V \text{ in } M & \to & M & \text{if } x \notin \text{fv}(M) \\
(G_{1c}) & {}^{[\star A_n]}\text{let } x = P^n\, M \text{ in } N & \to & N & \text{if } x \notin \text{fv}(N)
\end{array}$$

Compatible Closure As in $\mathsf{Let}^{\star\prime}$.

Figure 4.3: The marked generalized call-by-need calculus, $\mathsf{Need}^{\star\prime}$.

4.4 Generalized reduction and developments

We will develop a theory of marked reduction in the style of Section 3.8. The well-separation restriction required by the simpler of the two marking theories of Let caused no major complications for the subsequent proofs of that system's properties, but the analogous system for Need would not be especially useful.

Figure 4.2 describes the generalized call-by-need calculus. It is noteworthy that we have not generalized the (G) rule: that is, we do *not* have a rule

$$(\star G) \quad P^n\, M \to N \quad \text{ if for all } (P_i \equiv \text{let } x_i = N_i \text{ in}) \text{ in the } P^n,\, x_i \notin \text{fv}(M).$$

We have chosen between one of two sources of complexity. Under the formulation we have chosen, it will be necessary to present separate reasoning for (I, V, C, A) steps and for (G) steps, since our generalized development results will not apply to the latter (we will point out in the text the specific instances of the extra cases, although they should be clear). The present system will also make the strengthened finiteness of developments result rather trivial, since marked $\mathsf{Need}^\star$ reduction will be just the same marked $\mathsf{Let}^\star$ reduction. Had we chosen to generalize and mark (G) steps as well, the notion of marked reduction would be much more complex, since (G) and (V, A) steps can overlap in rather complicated combinations. We believe that the present formulation is much simpler than the alternative, and do not pursue the issue further.

We adapt the same notation and terminology for discussing $\mathsf{Need}^\star$ terms and reduction as in $\mathsf{Let}^\star$. Although single-step $\mathsf{Need}^\star$ reduction relates more terms than single-step Need reduction, their multistep reduction relations are again equal:

Lemma 4.4.1. $M \xrightarrow[\mathsf{Need}]{} \!\!\!\!\to N$ if and only if $M \xrightarrow[\mathsf{Need}^\star]{} \!\!\!\!\to N$.

Proof: As in comparing Let and $\mathsf{Let}^\star$, a single $(\star C, \star A)$ can correspond to many (C, A) steps. ♦

Figure 4.3 describes the marked generalized call-by-need calculus $\mathsf{Need}^{\star\prime}$. We refer to any combination of the (G_i) steps as simply (G). We take the definition of residuals and developments in $\mathsf{Need}^{\star\prime}$, as well as other notations, to be the same as in $\mathsf{Let}^{\star\prime}$. In particular, we abbreviate $(I_0, V_0, C_0, C_0^<, A_0, A_0^<)$ reduction by $\mathsf{Need}^{\star\prime}{}_0$, and reduction by other $\mathsf{Need}^{\star\prime}$ rules as $\mathsf{Need}^{\star\prime}{}_1$.

Lemma 4.4.2. The reduction relations $\xrightarrow[\mathsf{Let}^{\star\prime}_0]{}$ and $\xrightarrow[\mathsf{Need}^{\star\prime}_0]{}$ are equal.

Proof: By inspection, as they have the same syntax and are based on identical rules. ♦

Lemma 4.4.3. Reduction in $\mathsf{Need}^{\star\prime}{}_0$ is strongly normalizing and confluent.

Proof: Follows from the corresponding results on $\mathsf{Let}^{\star\prime}{}_0$ (Corollary 3.8.12 and Corollary 3.8.15, respectively) by Lemma 4.4.2. ⧫

Proposition 4.4.4, FD!($\mathsf{Need}^\star$)**.** All $\mathsf{Need}^\star$ developments are finite and can be extended to complete developments, and all complete developments from a given term and marking end in the same term.

Proof: From Lemma 2.1.4 with FD(Need) and Lemma 4.4.3 as usual. ⧫

Lemma 4.4.5. All $\mathsf{Need}^\star$-markings are joinable.

Proof: Immediate, since we use the same marking system as with $\mathsf{Let}^\star$. ⧫

We can define complete developments in Need indirectly, in terms of $\mathsf{Need}^\star$ developments. The notation is not as flexible as in $\mathsf{Need}^\star$, but it is a useful shorthand.

Definition 4.4.6. Let $\bar{\gamma}$ mark (I, V, C, A) redexes in $M \in \mathsf{Need}$. We define the complete development of $(M, \bar{\gamma})$ to be N whenever $(M, \bar{\gamma}) \xrightarrow[\text{cpl}]{} N$ holds in $\mathsf{Need}^\star$, taking all ranks initially equal to 1. By FD!($\mathsf{Need}^\star$), this N is unique. ◊

However, it does not make sense to define developments of markings of Need terms in general, since some residuals of $\mathsf{Need}^\star$ markings are not definable in Need. Still, it would be reasonable to define and discuss partial developments of marking of (I, V) redexes, since the behave just the same in Need as in $\mathsf{Need}^\star$.

4.5 Call-by-need and call-by-name

In this section we study a relationship between Need terms and reductions and Name terms and reductions. The relation between terms is just the duplication of let-bound terms in the bodies of the bindings in the style of the $(\beta \to)$ rule. In the first part of this section we define the operator $\pitchfork$ which formalizes this mapping, and show some basic results about the operator, including the soundness of $\pitchfork$ for mapping multistep Need reduction sequences to multistep Name reduction sequences.

Completeness is more tricky for two reasons: first, reduction in Need may "overshoot" reduction in Name. For example, we can consider the term $M \equiv (\lambda x.\, f\; x\; x)\; (I\; I)$, where $I \equiv \lambda x.\, x$. In Name we have

$$\begin{array}{cl} & (\lambda x.\, f\; x\; x)\; (I\; I) \\ \to & f\; (I\; I)\; (I\; I) \\ \to & f\; I\; (I\; I) \\ \equiv & N\ . \end{array}$$

But for all L where $M \xrightarrow[\mathsf{Need}]{} \!\!\!\!\to L$, we do not necessarily have $L^{\pitchfork} \equiv N$; the strongest statement we can make is that we will have $L^{\pitchfork} \underset{\mathsf{Name}}{=} N$:

L	$L^{\pitchfork}$
$\text{let } x = I\; I \text{ in } f\; x\; x$	$f\; (I\; I)\; (I\; I)$
$(\lambda x.\, f\; x\; x)\; (\text{let } y = I \text{ in } y)$	$(\lambda x.\, f\; x\; x)\; I$
$\text{let } x = (\text{let } y = I \text{ in } y) \text{ in } f\; x\; x$	$f\; I\; I$

and so forth: we show only (I) contractions, since as we shall see, others preserve $\pitchfork$-images. This difficulty is easily overcome: we simple relax the statement of the completeness result to allow such overshooting; such behavior is perfectly reasonable for a call-by-need calculus.

The second complication arises in finding redexes in a Need term M which correspond to each redex in the $\pitchfork$-image of a M: in some cases, there may be no corresponding redex in the original term. For example, in the term $M \equiv \text{let } x = I \text{ in } x\; y$, there is no readily markable redex corresponding to the one contracted in $M^{\pitchfork} \equiv I\; y \xrightarrow[\mathsf{Name}]{} y$. The only redex in M is a (V) redex, which again does not vary the $\pitchfork$ image. Moreover, other sorts of redex in Need terms can interfere similarly. Our solution to this problem is to normalize terms with respect to the (V, C, A, G) rules: then we can always associate redexes in a $\pitchfork$-image with a redex — obviously a (I) redex — in the original term. After establishing these preliminary results, the completeness result follows naturally.

Definition. We define the *let contraction* map $\pitchfork$ from (marked) call-by-need terms to (marked) call-by-name terms as follows:

$$\begin{array}{rcl} x^{\pitchfork} & \equiv & x \\ (\lambda x.\, M)^{\pitchfork} & \equiv & \lambda x.\, M^{\pitchfork} \\ (M\ N)^{\pitchfork} & \equiv & M^{\pitchfork}\ N^{\pitchfork} \\ (\text{let } x = M \text{ in } N)^{\pitchfork} & \equiv & N^{\pitchfork}[x := M^{\pitchfork}] \\ ({}^{[I]}(\lambda x.\, M)\ N)^{\pitchfork} & \equiv & {}^{[\beta\to]}(\lambda x.\, M^{\pitchfork})\ N^{\pitchfork} \ . \end{array}$$

For terms decorated with other redex markers, we simply drop the marker and translate according to the above rules. ◊

Lemma 4.5.1. Let $M, N \in \mathsf{Need}$. Then:

1. $(M[x := N])^{\pitchfork} \equiv M^{\pitchfork}[x := N^{\pitchfork}]$
2. If $M \underset{(V,C,A,G)}{=} N$ then $M^{\pitchfork} \equiv N^{\pitchfork}$.

Proof: The first clause follows by a trivial induction. For the second, we consider top-level reduction by each of the four rules in turn, compatible, reflexive and transitive closure being trivial.

1. *Top-level* (V) *steps.* So we have

$$\begin{array}{rl} M \equiv & \text{let } x = V \text{ in } M_1 \\ \xrightarrow[(V)]{} & M_1[x := V] \equiv N \ , \end{array}$$

   and the result in trivial as $M^{\pitchfork} \equiv \text{let } x = V^{\pitchfork} \text{ in } M_1^{\pitchfork} \equiv N^{\pitchfork}$.

2. *Top-level* (C) *steps.* So we have

$$\begin{array}{rl} M \equiv & (\text{let } x = M_1 \text{ in } M_2)\ M_3 \\ \xrightarrow[(C)]{} & \text{let } x = M_1 \text{ in } M_2\ M_3 \equiv N \ . \end{array}$$

   The result is again trivial, as $x \notin \text{fv}(M_3)$ and so

$$M^{\pitchfork} \equiv N^{\pitchfork} \equiv (M_2^{\pitchfork}\ M_3^{\pitchfork})[x := M_1^{\pitchfork}] \ .$$

3. *Top-level* (A) *steps.* So we have

$$\begin{array}{rl} M \equiv & \text{let } x = (\text{let } y = M_1 \text{ in } M_2) \text{ in } M_3 \\ \xrightarrow[(A)]{} & \text{let } y = M_1 \text{ in let } x = M_2 \text{ in } M_3 \equiv N \ . \end{array}$$

   As in the previous case, substitutions can be extended to include terms in the surrounding context where the substituted variable does not occur free.

4. *Top-level* (G) *steps.* So we have

$$\begin{array}{rcl} M & \equiv & \text{let } x = M_1 \text{ in } M_2 \\ & \xrightarrow[(G)]{} & M_2 \equiv N \ , \end{array}$$

   where since $x \notin \text{fv}(M_2)$, $M^{\pitchfork} \equiv M_2^{\pitchfork}$ already.

♦

Corollary 4.5.2. If $M \underset{(V,\star C,\star A,G)}{=} N$ then $M^{\pitchfork} \equiv N^{\pitchfork}$.

Proof: By Lemma 4.4.1 and Lemma 4.5.1.(2) ♦

We will write $\gamma\bar{\gamma}$ as a shorthand for $\{\gamma\gamma_0 : \gamma_0 \in \bar{\gamma}\}$. We now extend the map $\pitchfork$ to paths with respect to the term which the path indexes.

Definition. Let $\bar{\gamma}$ index redexes in $M \in \mathsf{Need}$, where $(M, \bar{\gamma})^{\pitchfork} \equiv (M^{\pitchfork}, \bar{\delta})$. Then we define the $\pitchfork$-image of $\bar{\gamma}$ with respect to M to be $\bar{\delta}$, in symbols $\bar{\gamma}^{\pitchfork}_{[M]} \equiv \bar{\delta}$. We will write the $\pitchfork$-image of a single path to mean simply the $\pitchfork$-image of the singleton set containing just that path, *i.e.*, $\gamma^{\pitchfork}_{[M]} \equiv \{\gamma\}^{\pitchfork}_{[M]}$. ◊

The following lemma explicitly justifies what might otherwise appear to be an abuse of the notation. Since $\pitchfork$-images of $(V, \star A, \star C, G)$-equal terms are identical, we can associate the $\pitchfork$-image of a path from either term with the $\pitchfork$-image of either term to produce the same valid member of Name'.

Lemma 4.5.3. Let $M \underset{(V,\star C,\star A,G)}{=} N$, let $\bar{\gamma}$ index redexes in N and let $\bar{\delta} = \bar{\gamma}^{\pitchfork}_{[N]}$. Then $(M^{\pitchfork}, \bar{\delta})$ is a marked call-by-name term, and $(M^{\pitchfork}, \bar{\delta}) \equiv (N^{\pitchfork}, \bar{\delta})$

Proof: Trivial, since $M^{\pitchfork} \equiv N^{\pitchfork}$. ♦

Lemma 4.5.4. Let $M \xrightarrow{\gamma} N$ be a $\mathsf{Need}^{\star}$ step. Then $(M, \{\gamma\})^{\pitchfork} \xrightarrow[\text{cpl}]{} N^{\pitchfork}$.

Proof: For $(V, \star C, \star A, G)$-steps the result is trivial, as then $M^{\pitchfork} \equiv N^{\pitchfork}$ and $\gamma^{\pitchfork}_{[N]} \equiv \{\}$. For (I) steps we proceed by structural induction on M.

1. $M \equiv \lambda x.\, M_0$. Then $N \equiv \lambda x.\, N_0$, and we have a simple inductive case. We take $\sigma \equiv \underline{\lambda 1}\delta$; then by the induction hypothesis we have $(M_0^{\pitchfork}, \delta^{\pitchfork}_{[M_0]}) \xrightarrow[\text{cpl}]{} N_0$, and so $(\lambda x.\, M_0^{\pitchfork}, \underline{\lambda 1}\delta^{\pitchfork}_{[\lambda x.\, M_0]}) \xrightarrow[\text{cpl}]{} \lambda x.\, N_0$.
2. $M \equiv M_0\, M_1$, *where* $\gamma \not\equiv \epsilon$. Again, by a simple induction as for the previous case.
3. $M \equiv (\lambda x.\, M_0)\ M_1$ *and* $\gamma \equiv \epsilon$. So $\delta^{\pitchfork}_{[M]} \equiv \{\epsilon\}$, and then both $N^{\pitchfork} \equiv M_1^{\pitchfork}[x := M_0^{\pitchfork}]$ and
$$(M, \{\epsilon\})^{\pitchfork} \equiv ((\lambda x.\, M_0^{\pitchfork})\ M_1^{\pitchfork}, \{\epsilon\}) \xrightarrow[\text{cpl}]{} M_1^{\pitchfork}[x := M_0^{\pitchfork}]\ .$$
4. $M \equiv \text{let } x = M_0 \text{ in } M_1$. We have two cases, depending on the location of the redex.
   (a) $\gamma \equiv \underline{\ell 1}\delta$. Then $M_0 \xrightarrow{\delta} N_0$, $N \equiv \text{let } x = N_0 \text{ in } M_1$ and $N^{\pitchfork} \equiv M_1^{\pitchfork}[x := N_0^{\pitchfork}]$. By the induction hypothesis we have
   $$(M_0, \{\delta\})^{\pitchfork} \xrightarrow[\text{cpl}]{} N_0^{\pitchfork}\ ,$$
   or in other words
   $$(M_0, \{\delta\})^{\pitchfork} \xrightarrow[\mathsf{Name}'_0]{} \!\!\!\!\rightarrow N_0^{\pitchfork} \in \mathsf{Name}\ .$$
   So by Lemma 2.1.6 we have σ such that
   $$\begin{array}{rcl} \sigma : (M, \{\gamma\})^{\pitchfork} & \equiv & M_1^{\pitchfork}[x := (M_0, \{\delta\})^{\pitchfork}] \\ & \xrightarrow[\mathsf{Name}'_0]{}\!\!\!\!\rightarrow & M_1^{\pitchfork}[x := N_0^{\pitchfork}] \\ & \equiv & N\ , \end{array}$$
   and since $M_1^{\pitchfork}, N_0^{\pitchfork} \in \mathsf{Name}$ and unmarked, so is N, and σ is in fact a complete development.

   (b) $\gamma \equiv \underline{\ell 2}\delta$. Then $M_1 \xrightarrow{\delta} N_1$, $N \equiv \text{let } x = M_0 \text{ in } N_1$ and $N^{\pitchfork} \equiv N_1^{\pitchfork}[x := M_0^{\pitchfork}]$. The proof proceeds largely as in the previous subcase. By the induction hypothesis we have
   $$(M_1, \{\delta\})^{\pitchfork} \xrightarrow[\text{cpl}]{} N_1^{\pitchfork}\ ,$$
   or in other words
   $$(M_1, \{\delta\})^{\pitchfork} \xrightarrow[\mathsf{Name}'_0]{}\!\!\!\!\rightarrow N_1^{\pitchfork} \in \mathsf{Name}\ .$$
   So by Lemma 2.1.6 we have σ such that
   $$\begin{array}{rcl} \sigma : (M, \{\gamma\})^{\pitchfork} & \equiv & M_1^{\pitchfork}[x := (M_0, \{\delta\})^{\pitchfork}] \\ & \xrightarrow[\mathsf{Name}'_0]{} & N_1^{\pitchfork}[x := M_0^{\pitchfork}] \\ & \equiv & N\ , \end{array}$$
   and once again since $M_0^{\pitchfork}, N_1^{\pitchfork} \in \mathsf{Name}$ and unmarked, so is N, and we have that σ is a complete development.

Syntactic Domains

$$\begin{array}{lrcl} \text{Appliable Terms} & \mathfrak{H} & ::= & \mathfrak{V} \mid \mathfrak{H}\,\mathfrak{M} \\ \text{Values} & \mathfrak{V} & ::= & \lambda x.\,\mathfrak{M} \mid x \\ \text{Terms} & \mathfrak{L}, \mathfrak{M}, \mathfrak{N} & ::= & \mathfrak{V} \mid \text{let } x = \mathfrak{H}\,\mathfrak{M} \text{ in } \mathfrak{M} \mid \mathfrak{H}\,\mathfrak{M} \end{array}$$

where in a term $(\text{let } x = \mathfrak{H}\,\mathfrak{M}_0 \text{ in } \mathfrak{M}_1)$, $x \in \text{fv}(\mathfrak{M}_1)$

Figure 4.4: (V, C, A, G)-normal Need terms.

♦

Corollary 4.5.5. Let $(M, \bar{\gamma}) \xrightarrow[\mathsf{Need}^\star\text{-cpl}]{} N$. Then $(M, \bar{\gamma})^{\pitchfork} \xrightarrow[\mathsf{Name}\text{-cpl}]{} N^{\pitchfork}$.

Proof: From Lemma 4.5.4 it is clear that $(M, \bar{\gamma})^{\pitchfork} \xrightarrow[\mathsf{Name}\text{-dev}]{} N^{\pitchfork}$; completeness follows from the observation that if $N \in \mathsf{Need}^\star$ — *viz.* if it is unmarked as opposed to a marked term which must be in $\mathsf{Need}^{\star\prime}$ — then $N^{\pitchfork} \in \mathsf{Name}$, also unmarked. ♦

Corollary 4.5.6. Let $M, N \in \mathsf{Need}^\star$, $M \xrightarrow[\mathsf{Need}^\star]{} \!\!\!\!\!\rightarrow N$. Then $M^{\pitchfork} \xrightarrow[\mathsf{Name}]{} \!\!\!\!\!\rightarrow N^{\pitchfork}$.

Proof: By induction on the length of the reduction sequence, since every single step can be regarded as a complete development. ♦

Figure 4.4 describes a subset of the grammar of Need terms; in fact we will show now that this subset identifies the Need terms which are in (V, C, A, G)-normal form. We let $\mathfrak{L}, \mathfrak{M}, \mathfrak{N}$ range over these terms. Again, Lemma 4.4.1 allows us to move between discussions of Need and $\mathsf{Need}^\star$ reduction; the lemma is easier to establish with the former.

Lemma 4.5.7. Let $\mathfrak{M}$ be as described in the figure.

1. $\mathfrak{M}$ is a Need term.
2. Moreover, a term $M \in \mathsf{Need}$ is in (V, C, A, G)-normal form if and only if it can be expressed as some term $\mathfrak{M}$.

Proof: The first clause is trivial. For the second clause, it is clear that every term $\mathfrak{M}$ has no (V, C, A, G)-redexes. For the converse, we consider structural induction on (V, C, A, G)-normal forms, and show that they are indeed equivalent to some term $\mathfrak{M}$. Only applications and let-bindings are interesting; other term-forms are trivial.

1. Let $M \equiv M_0\ M_1$ be a (V, C, A, G)-normal form. By the induction hypothesis, each M_i is equivalent to some $\mathfrak{M}_i$. Moreover, for M not to be a (C)-normal form, M_0 cannot be a let-binding, only a (V, C, A, G)-normal value or application: in other words, it must be some $\mathfrak{H}$, and so we have $M \equiv \mathfrak{H}\,\mathfrak{M}_1$.
2. Let $M \equiv \text{let } x = M_0 \text{ in } M_1$ be a (V, C, A, G)-normal form. By the induction hypothesis, again each M_i is equivalent to some $\mathfrak{M}_i$, and we clearly have $x \in \text{fv}(M_1)$. For M not to be a top-level (V, A)-redex, we must have that M_0 is neither a value nor another let-binding, only a (V, C, A, G)-normal application: in other words, it must be some $(\mathfrak{H}\,\mathfrak{N})$, and so we have $M \equiv \text{let } x = (\mathfrak{H}\,\mathfrak{N}) \text{ in } \mathfrak{M}_1$.

♦

Definition 4.5.8. For terms $M, N \in \mathsf{Need}$, we write $M \xrightarrow[(V,C,A,G)\text{-nf}]{} N$ whenever $M \xrightarrow[(V,C,A,G)]{} \!\!\!\!\!\rightarrow N$ and N is a (V, C, A, G)-normal form. ♢

Lemma 4.5.9. For all $M \in \mathsf{Need}$ there is a unique N such that $M \xrightarrow[(V,C,A,G)\text{-nf}]{} N$.

Proof: As for Proposition 3.6.7: the difference in argument from those of Section 3.6.1 and for this lemma is trivial since (G) easily fits those arguments for strong normalization. Uniqueness follows by Lemma 4.3.4. ♦

Again, it is clear by Lemma 4.4.1 that a (V, C, A, G)-normal form is also a $(V, \star C, \star A, G)$-normal form, and we will move between the two corresponding notations freely.

Lemma 4.5.10. Let $\sigma : L_0 \xrightarrow[(V,\star C,\star A,G)]{} L_1$, $(L_0, \bar{\gamma}) \xrightarrow[\text{cpl}]{} M$ where γ marks (I) steps, and take N where $(L_1, \bar{\gamma}/\sigma) \xrightarrow[\text{cpl}]{} N$. Then $M \xrightarrow[(V,\star C,\star A,G)]{} N$.

Proof: For σ a $(V, \star C, \star A)$ step the result is immediate from FD!($\mathsf{Need}^\star$). Otherwise for a (G) step we have the result by induction on the size of $\bar{\gamma}$ with a simple comparison of the relative positions of the redexes at each step. ♦

Lemma 4.5.11. Let $L \xrightarrow[(V,\star C,\star A,G)\text{-nf}]{} \mathfrak{L}$, $M \xrightarrow[(V,\star C,\star A,G)\text{-nf}]{} \mathfrak{M}$ and let $\bar{\gamma}$ index (I)-redexes in L such that $\sigma : (L, \bar{\gamma}) \xrightarrow[\text{cpl}]{} M$. Then $\mathfrak{L} \xrightarrow[\mathsf{Need}^\star]{} \!\!\!\!\!\rightarrow \mathfrak{M}$.

Proof: The point is to strengthen the obvious fact that $\mathfrak{L} \underset{\mathsf{Need}^\star}{=} \mathfrak{M}$. We fix τ as some $(V, \star C, \star A, G)$-reduction sequence from L to $\mathfrak{L}$, and the result follows by induction on the length of τ, applying Lemma 4.5.10 at each step. ♦

Corollary 4.5.12. Let $L \xrightarrow[(V,\star C,\star A,G)\text{-nf}]{} \mathfrak{L}$, $M \xrightarrow[(V,\star C,\star A,G)\text{-nf}]{} \mathfrak{M}$ and $L \xrightarrow[\mathsf{Need}^\star]{} M$. Then $\mathfrak{L} \xrightarrow[\mathsf{Need}^\star]{} \!\!\!\!\!\rightarrow \mathfrak{M}$.

This condition is stronger than just confluence for (I) steps: confluence tells us only that there exists some N such that both $\mathfrak{L}$ and $\mathfrak{M}$ reduce to N. This lemma asserts that in fact this N is equivalent to $\mathfrak{M}$.

Proof: If $L \xrightarrow[\mathsf{Need}^\star]{} M$ is a $(V, \star C, \star A, G)$-step, then the result is trivial by confluence. Otherwise, the result follows from the above lemma since the single step can be viewed as the complete development of a single (I)-redex. ♦

Lemma 4.5.13. Let $\mathfrak{M}$ be a $(V, \star C, \star A, G)$-normal $\mathsf{Need}^\star$ term where $\mathfrak{M}^{\pitchfork} \equiv M \in \mathsf{Name}$, and let γ index a $(\beta \rightarrow)$-redex in M. Then there exists some δ indexing an (I)-redex in $\mathfrak{M}$ such that $\gamma \in \{\delta\}^{\pitchfork}_{[\mathfrak{M}]}$.

Proof: By induction on the structure of the term $\mathfrak{M}$.

- $\mathfrak{M} \equiv \lambda x.\,\mathfrak{M}_0$. Then we must have $\gamma \equiv \gamma_0$ and the result follows by the induction hypothesis on $\mathfrak{M}_0$ and γ_0.
- $\mathfrak{M} \equiv \mathfrak{H}\,\mathfrak{N}$. If $\gamma \equiv \epsilon$ then $\delta \equiv \epsilon$ as well. Otherwise the result follows by the induction hypothesis on the appropriate subterm (a valid argument for $\mathfrak{H}$ since every appliable term $\mathfrak{H}$ is also a general term $\mathfrak{M}$).
- $\mathfrak{M} \equiv \text{let } x = \mathfrak{H}\,\mathfrak{M}_1 \text{ in } \mathfrak{M}_2$. We distinguish between two possibilities: whether the marked redex in M originates in $\mathfrak{M}_1$ or in $\mathfrak{M}_2$. The distinction is made by the following technical criterion: do there exist γ_1 and γ_2 such that $\gamma \equiv \gamma_1\gamma_2$ and $\mathfrak{M}_2^{\pitchfork}|_\gamma \equiv x$? In other words, in indexing $\mathfrak{M}_2^{\pitchfork}$ by γ, do we run into a reference to the let-bound variable somewhere along the indexing path?
  - *Such γ_0, γ_1 exist.* Then the result follows by induction on $(\mathfrak{H}\,\mathfrak{M}_1)$ and γ_0.
  - *No such γ_0, γ_1 exist.* Then clearly we have some δ such that $\mathfrak{M}_2|_\delta$ is an application $\mathfrak{V}\,\mathfrak{N}$ and $\delta^{\pitchfork}_{\mathfrak{M}_2} \equiv \gamma$. Moreover, $\mathfrak{V}$ must be an abstraction and not a variable: if it were a variable, and for M to have a redex at γ, then it would be necessary that the variable to be let-bound to an abstraction, which is impossible by (V)-normalization. We also have that $\mathfrak{V}$ is not a let-binding, since $\mathfrak{M}$ is $(\star C)$-normal. So δ does in fact mark a redex in $\mathfrak{M}$.

♦

Corollary 4.5.14. Let $\mathfrak{M}$ be a $(V, \star C, \star A, G)$-normal $\mathsf{Need}^\star$ term where $\mathfrak{M}^{\pitchfork} \equiv M \in \mathsf{Name}$, and let $\bar{\gamma}$ index $(\beta \rightarrow)$-redexes in M. Then there exists some $\bar{\delta}$ indexing (I)-redexes in $\mathfrak{M}$ such that $\bar{\gamma} \subseteq \{\bar{\delta}\}^{\pitchfork}_{[\mathfrak{M}]}$.

Proof: $\bar{\delta}$ is just the union of the individual corresponding redexes for every member of $\bar{\gamma}$ predicted by Lemma 4.5.13 above; here we need not bother with the $\Cup$-style union of Section 3.8.4 since all marking sets of (I) redexes and their unions are always valid. ♦

The main lemma of this section follows:

Lemma 4.5.15. Let $\mathfrak{M}$ be a $(V, \star C, \star A, G)$-normal Need term, and let $M, N \in \mathsf{Name}$ where $\sigma_0 : M \xrightarrow[\mathsf{Name}]{} \!\!\!\!\twoheadrightarrow \mathfrak{M}^{\pitchfork}$ and $\sigma_1 : M \xrightarrow[\mathsf{Name}]{\gamma} N$. Then there exists some $(V, \star C, \star A, G)$-normal Need term $\mathfrak{N}$ such that $\mathfrak{M} \xrightarrow[\mathsf{Need}^\star]{} \!\!\!\!\twoheadrightarrow \mathfrak{N}$ and $N \xrightarrow[\mathsf{Name}]{} \!\!\!\!\twoheadrightarrow \mathfrak{N}^{\pitchfork}$:

$$
\begin{tikzcd}
\mathfrak{M} \arrow[r, dotted, "\mathsf{Need}^\star"'] & \exists\mathfrak{N} \\
\mathfrak{M}^{\pitchfork} & \mathfrak{N}^{\pitchfork} \\
M \arrow[u, "\mathsf{Name}"'] \arrow[r, "\gamma", "\mathsf{Name}"'] & N \arrow[u, dotted, "\mathsf{Name}"']
\end{tikzcd}
.
$$

Proof: Since $M \xrightarrow[\mathsf{Name}]{} N$ in a single step, σ_0 is a complete development. Then by Lemma 2.1.10, we have some $M_0 \in \mathsf{Name}$ such that $\tau : (\mathfrak{M}^{\pitchfork}, \gamma/\sigma_1) \xrightarrow[\text{cpl}]{} M_0$ and $N \twoheadrightarrow M_0$. By Corollary 4.5.14 we have some marking $\bar{\delta}$ of $\mathfrak{M}$ such that $\gamma/\sigma_1 \subseteq \bar{\delta}^{\pitchfork}_{[\mathfrak{M}]}$. We take N_0 to be the result of the complete development of $\mathfrak{M}$ by the redexes marked in $\bar{\delta}$, which is unique by FD!($\mathsf{Need}^\star$), and which has $(V, \star A, \star C, G)$-normal form $\mathfrak{N}$. By Corollary 4.5.5, we have $(\mathfrak{M}, \bar{\delta})^{\pitchfork} \xrightarrow[\mathsf{Name}\text{-cpl}]{} \mathfrak{N}^{\pitchfork}$ as well. Since $\gamma/\sigma_1 \subseteq \bar{\delta}^{\pitchfork}_{[\mathfrak{M}]}$, τ is a partial development of $(\mathfrak{M}, \bar{\delta})^{\pitchfork}$ which can be extended by FD!(Name) to a complete development ending in $\mathfrak{N}^{\pitchfork}$. So since developments can be projected to sequences in the unmarked calculi, we have that both $\mathfrak{M} \xrightarrow[\mathsf{Need}^\star]{} \!\!\!\!\twoheadrightarrow N_0 \xrightarrow[\mathsf{Need}^\star]{} \!\!\!\!\twoheadrightarrow \mathfrak{N}$ and $N \xrightarrow[\mathsf{Name}]{} \!\!\!\!\twoheadrightarrow M_0 \xrightarrow[\mathsf{Name}]{} \!\!\!\!\twoheadrightarrow \mathfrak{N}^{\pitchfork}$. The reasoning is summarized in Figure 4.5. ♦

Corollary 4.5.16. Let $\mathfrak{M}$ be a $(V, \star C, \star A, G)$-normal Need term, and let $M, N \in \mathsf{Name}$ where $M \xrightarrow[\mathsf{Name}]{} \!\!\!\!\twoheadrightarrow \mathfrak{M}^{\pitchfork}$ and $M \xrightarrow[\mathsf{Name}]{} \!\!\!\!\twoheadrightarrow N$. Then there exists some $(V, \star C, \star A, G)$-normal Need term $\mathfrak{N}$ such that $\mathfrak{M} \xrightarrow[\mathsf{Need}^\star]{} \!\!\!\!\twoheadrightarrow \mathfrak{N}$ and $N \xrightarrow[\mathsf{Name}]{} \!\!\!\!\twoheadrightarrow \mathfrak{N}^{\pitchfork}$:

$$
\begin{tikzcd}
\mathfrak{M} \arrow[r, dotted, two heads, "\mathsf{Need}^\star"'] & \exists\mathfrak{N} \\
\mathfrak{M}^{\pitchfork} & \mathfrak{N}^{\pitchfork} \\
M \arrow[u, two heads, "\mathsf{Name}"'] \arrow[r, two heads, "\mathsf{Name}"'] & N \arrow[u, dotted, two heads, "\mathsf{Name}"']
\end{tikzcd}
.
$$

Proof: By the obvious induction using Lemma 4.5.15 at each step. ♦

Lemma 4.5.17. Let $M \in \mathsf{Need}$ and $N_0 \in \mathsf{Name}$ where $M^{\pitchfork} \xrightarrow[\mathsf{Name}]{} \!\!\!\!\twoheadrightarrow N_0$. Then there exists some $N \in \mathsf{Need}$ such that $M \xrightarrow[\mathsf{Need}^\star]{} \!\!\!\!\twoheadrightarrow N$ and $N_0 \xrightarrow[\mathsf{Name}]{} \!\!\!\!\twoheadrightarrow N^{\pitchfork}$.

Proof: By Proposition 3.6.7, Lemma 4.4.1 and Corollary 4.5.16. ♦

We can now tie everything together, using the relationship between Need- and $\mathsf{Need}^\star$-reduction to move the results above from the latter system into the former.

Proposition 4.5.18, Equivalence of Name and Need. The mapping $\pitchfork$ is sound and complete for mappings of Name reduction sequences to Need reduction sequences, where Need sequences are allowed to "overshoot" Name results:

$$
\begin{tikzcd}
M \arrow[r, "\mathsf{Need}"] & N \\
M^{\pitchfork} \arrow[r, dotted, "\mathsf{Name}"] & N^{\pitchfork}
\end{tikzcd}
\qquad
\begin{tikzcd}
M \arrow[rr, dotted, "\mathsf{Need}"] & & \exists N \\
M^{\pitchfork} \arrow[r, "\mathsf{Name}"] & N_0 \arrow[r, dotted, "\mathsf{Name}"] & N^{\pitchfork}
\end{tikzcd}
$$

Proof: By Lemma 4.4.1, Corollary 4.5.6 and Lemma 4.5.17. ♦

For this proof we have concentrated on decribing the relationship between the reduction relations of call-by-name and call-by-need. Other proofs of a similar result by Ariola and Felleisen [3, 4, 5] and in our earlier work with Odersky and Wadler [53] focused on the observational equivalence relations. In fact, the result here imply equivalence of the observational equivalence relations [54], but the converse is not true since the reduction relations are a more specific result.

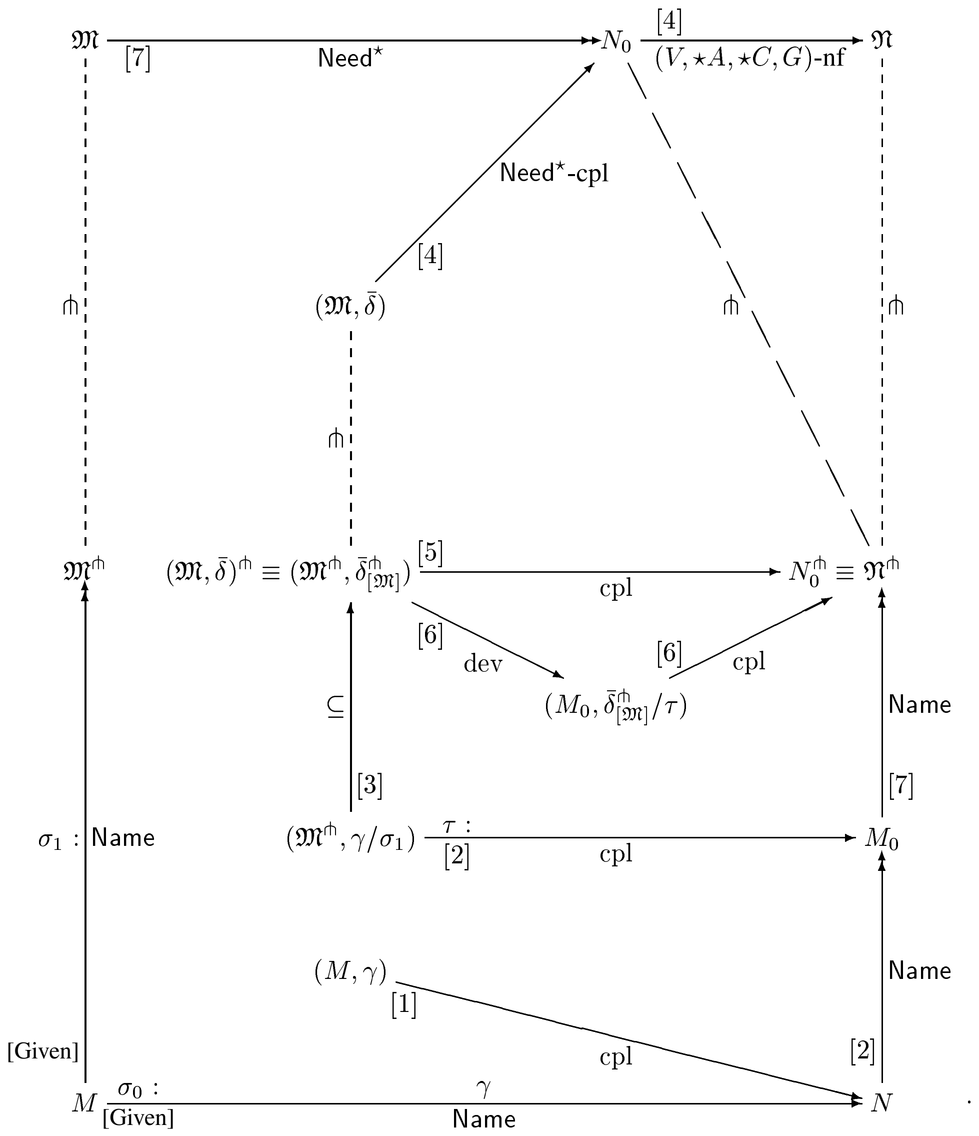


Figure 4.5: Reasoning for the proof of Lemma 4.5.15. The numbers in square brackets refer to the sentence in the proof where the particular link is established. Except where explicitly indicated, all (complete) developments are Name sequences.

4.6 Standard call-by-need reduction

Figure 4.6 shows the evaluation strategy $\mapsto$ for Need [3, 4, 5, 53, 54] which we will show in this section to be standard. Although this proof has exactly the same general approach as the corresponding proof of the previous chapter, the present exposition is far more complicated, simply because call-by-need evaluation is a more complicated notion than evaluation in the preceding systems.

The *need* in call-by need is manifest in the evaluation context forms

$$E ::= \cdots \mid \text{let } x = M \text{ in } E \mid \text{let } x = E \text{ in } E[x] \ .$$

By the first, we delay evaluation of bound terms; by the second, we return to evaluate a bound term only when the bound variable is demanded. The syntactic restrictions to the (C, A) rules reflect the structure of evaluation contexts as well. We have a form

$$E ::= \cdots \mid E\, M$$

which requires us to reduce the left-hand side of an application to an answer; the standard (C) rule therefore requires its own left-hand side to be an answer before the rule becomes applicable. Similarly, reflecting the evaluation context form

$$E ::= \cdots \mid \text{let } x = E \text{ in } E[x] \ ,$$

the standard (A) rule requires that its bound term also be reduced to an answer. This arrangement is a somewhat arbitrary design decision: it would be perfectly reasonable — reasonable in the sense that we would clearly have the same evaluation behavior — to contract (C, A) redexes upon their arrival in evaluation position, and evaluate within the bodies only of top-level (C, A)-normal terms. The advantage of the present formulation is that it is very much simpler than such an alternative, and we do not second-guess this design point further.

As usual, the standardization proof consist of a unique evaluation contexts lemma followed by a statement of rearranging sequences to allow standard steps to proceed internal steps; as usual, the former is far simpler than the latter. We begin with the unique evaluation context lemma, Lemma 4.6.1, and a handful of simple substitution invariants on the new syntactic domains. The remainder of the proof is based largely on marking and developments, so we introduce the notion of standard evaluation for Need* in 4.7, and complete the main series of lemmas primarily in Need* rather than Need. Unlike with the confluence result, or with the standardization result for Let, we cannot avoid problems caused by the well-separation restriction here. The problem cases which first motivated well-separation (Section 3.3.3) disallow a simple proof. We would like to make the usual argument that a complete development of internal redexes followed by a single standard redex can be replaced by a standard sequence followed by a complete internal development:

If $(L, \bar{\gamma}) \xrightarrow[\text{cpl}]{} M \mapsto N$ then $\sigma : L \mapsto\!\!\!\!\rightarrow M_0$ and $(M_0, \bar{\gamma}/\sigma) \xrightarrow[\text{cpl}]{} N$.

Unfortunately, this statement is not true. In standard Let reduction, where standard steps are always (I, V), the standard redex δ of L can always be added to $\bar{\gamma}$, and $\bar{\gamma} \cup \{\delta\}$ will again be well-separated since arbitrary (I, V) redexes can be marked in a well-separated set. On the other hand, in Need we can have standard (C, A) steps which may not be well-separated from internal redexes.

Example 4.6.1. In a non-well-separated pair of (A) redexes, the inner redex may be standard. Take

$$L \equiv \text{let } x = (\text{let } y = (\text{let } z = L_0 \text{ in } A) \text{ in } E_0[y]) \text{ in } E_1[x]$$

and $\bar{\gamma} \equiv \{\epsilon\}$, *i.e.* only the top-level redex marked. We have $(L, \bar{\gamma}) \xrightarrow[\text{cpl}]{} M \mapsto N$ where

$$\begin{aligned} M &\equiv \text{let } y = (\text{let } z = L_0 \text{ in } A) \text{ in let } x = E_0[y] \text{ in } E_1[x] \\ N &\equiv \text{let } z = L_0 \text{ in let } y = A \text{ in let } x = E_0[y] \text{ in } E_1[x] \end{aligned}$$

We also have that the residual in M of the standard redex δ_0 of L is that top-level standard contraction from M to N, namely $\delta_0 \equiv \underline{\ell_1}$, but that $\bar{\gamma} \cup \{\delta_0\} \equiv \{\epsilon, \underline{\ell_1}\}$ is not well-separated, since

$$(L, \{\epsilon, \underline{\ell_1}\}) \equiv {}^{[A]}\text{let } x = ({}^{[A]}\text{let } y = (\text{let } z = L_0 \text{ in } A) \text{ in } E_0[y]) \text{ in } E_1[x] \ .$$

Then contracting the standard step $\tau : L \overset{\delta_0}{\longmapsto} M_0$ we have

$$(M_0, \bar{\gamma}/\tau) \equiv {}^{[A]}\text{let } x = (\text{let } z = L_0 \text{ in let } y = A \text{ in } E_0[y]) \text{ in } E_1[x]$$

and $(M_0, \bar{\gamma}/\tau) \xrightarrow[\text{cpl}]{} N_0$ where

$$N_0 \equiv \text{let } z = L_0 \text{ in let } x = (\text{let } y = A \text{ in } E_0[y]) \text{ in } E_1[x] \ \ .$$

Clearly $N_0 \not\equiv N$; rather $N_0 \xrightarrow[\text{i}]{} N$. $\Diamond$

The point, as in the failure of weak confluence to prove confluence in general, is that we have no basis for a crucial inductive argument to lift our results from single to multiple steps: in general we will not have only one step in the standard development of the initial term and marking $(L, \bar{\gamma} \cup \{\delta_0\})$; the example counters the claim that we will have a single complete development. Moreover, a similar problem plagues one of the other two instances of non-well-separation.

Example 4.6.2. In a non-well-separated pair of a (C) redex too close to a (A) redex, the inner redex may be again be standard. Take

$$L \equiv (\text{let } x = (\text{let } y = L_0 \text{ in } A) \text{ in } E_0[x]) \ L_1$$

and $\bar{\gamma} \equiv \{\epsilon\}$. We have $(L, \bar{\gamma}) \xrightarrow[\text{cpl}]{} M \mapsto N$ where

$$\begin{array}{rcl} M & \equiv & \text{let } x = (\text{let } y = L_0 \text{ in } A) \text{ in } (E_0[x]) \ L_1 \\ N & \equiv & \text{let } y = L_0 \text{ in let } x = A \text{ in } (E_0[x]) \ L_1 \end{array}$$

We also have that the residual in M of the standard redex δ_0 of L is that top-level standard contraction from M to N, namely $\delta_0 \equiv \underline{@1}$, but that $\bar{\gamma} \cup \{\delta_0\} \equiv \{\epsilon, \underline{@1}\}$ is not well-separated, since

$$(L, \{\epsilon, \underline{\ell 1}\}) \equiv {}^{[C]}({}^{[A]}\text{let } y = (\text{let } z = L_0 \text{ in } A) \text{ in } E_0[y]) \ L_1 \ \ .$$

Then contracting the standard step $\tau : L \overset{\delta_0}{\longmapsto} M_0$ we have

$$(M_0, \bar{\gamma}/\tau) \equiv {}^{[C]}(\text{let } z = L_0 \text{ in let } y = A \text{ in } E_0[y]) \ L_1$$

and $(M_0, \bar{\gamma}/\tau) \xrightarrow[\text{cpl}]{} N_0$ where

$$N_0 \equiv \text{let } z = L_0 \text{ in } (\text{let } y = A \text{ in } E_0[y]) \ L_1 \ \ .$$

Again we have only $N_0 \xrightarrow[\text{i}]{} N$, and not equivalence. $\Diamond$

Happily, coincident (A, G) steps do not arise to cause problems, since the (A) step could be standard. In a term $(\text{let } x = (\text{let } y = L \text{ in } M) \text{ in } N)$, we cannot have both that $N \equiv E[x]$ to allow the (A) step to be standard, and that $x \notin \text{fv}(N)$ to allow the whole term to be a (G) step.

Following some substitution results for $\mathsf{Need}^\star$, we give a statement about the uniqueness of evaluation positions in Lemma 4.6.7. The key lemmas of the substitution result then follow: corresponding to the five points of the proof strategy on p. 27, we have the five results beginning with Lemma 4.6.9. We extend these results in the usual way from developments to general sequences, incorporating (G) steps in Lemma 4.6.15. We return to Need from $\mathsf{Need}^\star$ in the final result, Proposition 4.6.17.

Lemma 4.6.1. Let $M \in \mathsf{Need}$. Then exactly one of the following is true:

1. M is an answer in Need.

2. There exist some unique evaluation context E and variable $x \in \text{fv}(M)$ such that $M \equiv E[x]$.

3. There exist some unique evaluation context E and top-level standard Need redex Δ such that $M \equiv E[\Delta]$.

Proof: We proceed by structural induction on M.

- $M \equiv x$. Trivially this is Clause 2 with $E \equiv [\,]$.

- $M \equiv \lambda x.\, M_0$. Abstractions are answers, so we have Clause 1.

Syntactic domains As for Need, plus the following:

Standard values	V	$::=$	$\lambda x.\, M$
Abstraction closures	A	$::=$	$\lambda x.\, M \mid \text{let } x = M \text{ in } A$
Evaluation contexts	E	$::=$	$[\,] \mid E\ M \mid \text{let } x = M \text{ in } E \mid \text{let } x = E \text{ in } E[x]$

Standard reduction relation

(I)	$(\lambda x.\, M)\ N$	$\mapsto$	$\text{let } x = N \text{ in } M$
(V)	$\text{let } x = V \text{ in } E[x]$	$\mapsto$	$(E[x])[x := V]$
(C)	$(\text{let } x = M \text{ in } A)\ N$	$\mapsto$	$\text{let } x = M \text{ in } (A\ N)$
(A)	$\text{let } y = (\text{let } x = M \text{ in } A) \text{ in } E[y]$	$\mapsto$	$\text{let } x = M \text{ in } (\text{let } y = A \text{ in } E[y])$

Figure 4.6: Standard reduction for call-by-need.

- $M \equiv M_0 M_1$. By the induction hypothesis we have one of three possibilities for M_0.
  - M_0 *is an answer.* We have two possibilities for the two forms of answers:
    - $M_0 \equiv \lambda x.\, N_0$. Then M is a top-level (I) step.
    - $M_0 \equiv (\text{let } x = N_0 \text{ in } N_1)$. Here M is a top-level (C) step.
  - $M_0 \equiv E_0[x]$ *where* $x \in \text{fv}(M_0)$. Then $x \in \text{fv}(M)$ as well, and $E \equiv E_0\ M_1$ for Clause 2 again.
  - $M_0 \equiv E_0[\Delta]$. Then Clause 3 holds for M as well with $E \equiv E_0\ M_1$.
- $M \equiv (\text{let } x = M_0 \text{ in } M_1)$. We have three possibilities again, here based on M_1.
  - M_1 *is an answer.* Then M is an answer as well, with Clause 1 applying.
  - $M_1 \equiv E_1[y]$ *where* $y \in \text{fv}(M_1)$. Here we have two possibilities based on the equivalence of x and y.
    - $x \not\equiv y$. Then we also have $y \in \text{fv}(M)$, so Clause 2 applies to M with $E \equiv (\text{let } x = M_0 \text{ in } E_1)$.
    - $x \equiv y$. In this case we must also inductively consider M_0.
      - M_0 *is an answer.* We have Clause 3 with $E \equiv [\,]$ *i.e.* a top-level redex, and we distinguish the two forms of answer to determine a particular reduction rule.
        - $M_0 \equiv \lambda x.\, N_0$. Then we have a top-level (V) redex.
        - $M_0 \equiv (\text{let } x = N_0 \text{ in } N_1)$. Here we have a top-level (A) redex.
      - $M_0 \equiv E_0[x]$ *where* $x \in \text{fv}(M_0)$. Then for M we also have Clause 2 applicable, with $E \equiv (\text{let } x = E_0 \text{ in } E_1[x])$.
      - $M_0 \equiv E_0[\Delta]$. Then for M we also have Clause 3 applicable, with $E \equiv (\text{let } x = E_0 \text{ in } E_1[x])$.
  - $M_1 \equiv E_1[\Delta]$. Then Clause 3 applies to M, with $E \equiv (\text{let } x = M_0 \text{ in } E_1)$.

We have exhausted the possibilities for M, and the proof is complete. ♦

Lemma 4.6.2. Evaluation contexts are closed under substitution of values for variables: E is an evaluation context if and only if $E[x := V]$ is an evaluation context (*i.e.* taking $[\,][x := V] \equiv [\,]$).

Proof: Trivial: where the structure of an evaluation context relies on a variable, the variable is bound within the evaluation context as a whole, and so is not affected by the substitution. ♦

Lemma 4.6.3. Answers are closed under substitution of values for variables: if A is an answer then $A[x := V]$ is also an answer.

Note that the converse is not true in this case: for example, we have that $(\text{let } x = M \text{ in } y)$ is not an answer, but that $(\text{let } x = M \text{ in } y)[y := \lambda z.\, N]$ is.

Syntactic domains As for Need and $\mathsf{Need}^\star$.

Prestandard reduction relation

$$\begin{array}{lllll}
(I) & (\lambda x.\, M)\, N & \mapsto & \text{let } x = N \text{ in } M \\
(V) & \text{let } x = V \text{ in } E[x] & \mapsto & (E[x])[x := V] \\
(\star C) & (P^n\, A)\, N & \mapsto & P^n\, (A\, N) \\
(\star A) & \text{let } x = (P^n\, A) \text{ in } E[x] & \mapsto & P^n\, (\text{let } x = A \text{ in } E[x])
\end{array}$$

Figure 4.7: Prestandard reduction for generalized call-by-need.

Proof: Again trivial since free variables do not arise in the definition of an answer. ♦

Our previous proof of standardization for this system [5, 53] employed a rather brute-force approach; here we give a more elegant proof with the generalized marking mechanism.

Note that the result of Lemma 3.5.1 for call-by-let, which allowed a proof of standardization using a well-separation criterion wothout joinable redexes, does not hold here. In the context $E_0[\text{let } x = E_1 \text{ in } E_2[x]]$, the fact that the path to E_0 is that of a term in evaluation position does not imply that the subterm on the right in the binding is of the required form.

Figure 4.7 presents an evaluation strategy $\xmapsto[\mathsf{Need}^\star]{}$ for $\mathsf{Need}^\star$. In fact, $\xmapsto[\mathsf{Need}^\star]{}$ gives only part of what we have defined a standard reduction strategy to be. We do not require that only one single such step be possible from a given term; the $(\star C, \star A)$ rules we have chosen clearly allow several possible redexes at the same evaluation position. However, we do require that all possible decompositions of a term into an evaluation context and a top-level restricted step share the same evaluation context. This relaxation will still suffice for our overall goal of showing $\xmapsto[\mathsf{Need}]{}$ to be (fully) standard, since exactly one of any possible $\xmapsto[\mathsf{Need}^\star]{}$ steps from a particular term will also be a $\xmapsto[\mathsf{Need}]{}$ step. We therefore refer to $\xmapsto[\mathsf{Need}^\star]{}$ as a *prestandard*, rather than standard, relation.

The relationship between standard Need reduction and prestandard $\mathsf{Need}^\star$ reduction is as is the general systems.

Lemma 4.6.4. For all $M, N \in \mathsf{Need}$, $M \xmapsto[\mathsf{Need}]{} \!\!\!\rightarrow N$ if and only if $M \xmapsto[\mathsf{Need}^\star]{}\!\!\!\rightarrow N$.

Proof: Clear by inspection of the rules. ♦

Generalized prestandard reductions are preserved by substitution of values for variables:

Lemma 4.6.5. Let $\sigma : M \xmapsto[\mathsf{Need}^\star]{} N$. Then for all x, V we have

$$M[x := V] \xmapsto[\mathsf{Need}^\star]{} N[x := V] \ .$$

Moreover if we have γ such that $\sigma : M \overset{\gamma}{\mapsto} N$, then $M[x := V] \overset{\gamma}{\mapsto} N[x := V]$ as well.

Proof: For top-level prestandard contractions we have the result by Lemma 4.6.2 and Lemma 4.6.3; for closure the result is implied by Lemma 4.6.2. ♦

The converse of the above lemma is not true in general: we must have the additional condition that the variable must not be demanded before substitution. The notion of a demanded variable is a useful shorthand, which we formalize now.

Definition. We say that M *demands* $x \in \mathrm{fv}(M)$ if there is some evaluation context E such that $M \equiv E[x]$. If no such evaluation context exists, then we say that M *does not demand* x. ♢

Lemma 4.6.6. Let $M[x := V] \xmapsto[\mathsf{Need}^\star]{} N$, where M does not demand x. Then M has a prestandard redex.

Proof: For top-level redexes the result is clear from the definition of $\mapsto$; for closure in evaluation contexts we have a straightforward structural induction. ♦

Lemma 4.6.7. Let $M \in \mathsf{Need}/\mathsf{Need}^\star$. Then exactly one of the following is true:

1. M is an answer in Need/Need*.
2. M demands some $x \in \mathrm{fv}(M)$.
3. There exists at least one top-level prestandard Need* redex Δ, plus some unique evaluation context E such that $M \equiv E[\Delta]$.

Proof: Just as in Lemma 4.6.1 except that many possible ranks for $(\star C, \star A)$ redexes may arise. ♦

The next two lemmas are related. Answers A and terms of the form $E[x]$ are important in call-by-need since recognition of evaluation contexts and prestandard steps depend on these syntactic constructs. In fact, both structures are preserved by internal redexes.

Lemma 4.6.8. Let M demand some $x \in \mathrm{fv}(M)$, and $M \underset{\mathrm{i}}{\to} N$ in Need*. Then N demands x as well.

Proof: By induction on the position in M of the internal redex. We take $M \equiv E[x]$ and construct the E_1 such that $N \equiv E_1[x]$.

- *Top-level (I) reduction.* This case is not possible; a top-level (I) redex would necessarily be prestandard.
- *Top-level (V) reduction.* We have a number of subcases, reflecting how a top-level (V) step could be internal.
  - $M \equiv \mathrm{let}\ z = y\ \mathrm{in}\ M_0 \to M_0[z := y] \equiv N$.
    - $x \equiv y$. Then one of the following hold:
      - $M_0 \equiv E_0[y] \equiv E_0[x]$. Then we have $E_1 \equiv E_0[y][z := y] \equiv E_0[x][z := x]$ which clearly continues to demand x.
      - $M_0 \equiv E_0[z]$. Trivially, by Lemma 4.6.2.
    - $x \not\equiv y$. Then we must have $M_0 \equiv E_0[x]$, and the substitution makes no difference.
  - $M \equiv \mathrm{let}\ y = \lambda x.\, M_0\ \mathrm{in}\ M_1$ *where* $M_1 \not\equiv E_0[z]$. Then we must have some E_2 such that $M_1 \equiv E_2[x]$, and $E_1 \equiv E_2[z := \lambda y.\, M_0]$ with demand for x by $x \not\equiv z$ and Lemma 4.6.2.
- *Top-level $(\star A)$ reduction.* We have one of the following configurations.
  - $M \equiv (\mathrm{let}\ z = (P^n\, M)\ \mathrm{in}\ N)$ *where* $N \not\equiv E_0[z]$. This case is trivial: we will have E_1 of the form $E_1 \equiv P^n\, E_2$. Then if for example $E \equiv \mathrm{let}\ z = (P^n\, M)\ \mathrm{in}\ E_0$ we have $E_2 \equiv \mathrm{let}\ z = M\ \mathrm{in}\ E_0$ and so on.
  - $M \equiv (\mathrm{let}\ z = (P^n\, M)\ \mathrm{in}\ E_0[z])$ *where M is not an answer.* Again, this case is trivial, with an analysis as in the previous case.
- *Top-level $(\star C)$ reduction.* As in top-level $(\star A)$ reduction.
- *Top-level (G) reduction.* Trivial; a bound, demanded variable could not be discarded.

We have compatible closure as usual by induction on the size of the evaluation context. ♦

In evaluating properties **P0** through **P4**, we transparently read *prestandard* for *standard* in the definition of the properties; since the properties deal with completeness of the strategy rather than uniqueness of the redex this matter is inconsequential.

Lemma 4.6.9. *Generalized $(I, V, \star C, \star A)$-reduction satisfies **P0**:* All Need* developments are finite and can be extended to a complete development, all complete Need*-developments of the same term and marking end in the same term, and Need* has joinable markings with the joining operator $\Cup$.

Proof: Just as for Let*, since the markings and marked rules are identical. ♦

Lemma 4.6.10. *Generalized call-by-need satisfies **P1**:* Let $\sigma : M \underset{\mathrm{i}}{\twoheadrightarrow} N$ in Need*. Then M is an answer if and only if N is an answer.

Note that this lemma applies to all internal Need* steps, including (G) steps.

Proof: By structural induction on M. We first consider top-level $(V, \star C, \star A, G)$ steps, with top-level internal (I) reduction not possible.

- *Top-level* (V) *steps.* The forward direction is immediate from Lemma 4.6.3. For the converse, we must use the fact that σ is internal. We take $M \equiv (\text{let } x = V \text{ in } M_0)$ where M_0 does not demand x. If N is an answer but M is not, then M_0 must demand x. But then σ would be internal, which it is not, so M must be an answer as well.
- *Top-level* $(\star C)$ *steps.* This case is not possible, since a top-level $(\star C)$ step must also be a top-level application, which cannot be an answer.
- *Top-level* $(\star A)$ *steps.* Trivially, for a $(\star A)$ step

  $$M \equiv \text{let } x = (P^n\, M_0) \text{ in } M_1 \to P^n\, (\text{let } x = M_0 \text{ in } M_1) \equiv N \ \ ,$$

  M is an answer if and only if M_1 is an answer, which is true if and only if N is an answer.
- *Top-level* (G) *steps.* By an analysis of the rule, as for the $(\star A)$ step.

Compatible closure is trivial by the induction hypothesis, and the proof is complete. ♦

Lemma 4.6.11. *Generalized* $(I, V, \star C, \star A)$*-reduction satisfies* ***P2****:* Let γ, δ mark distinct internal $\mathsf{Need}^\star$ redexes in M, $\sigma : M \xrightarrow[(\gamma)]{} N$. Then every member of δ/σ is internal.

Proof: By induction on the structure of M. We assume without loss of generality that γ and δ are different, and perform an analysis of the relative positions of γ and δ. We begin with the case where one of the two redexes is top-level, and afterward where the two are in distinct subterms.

- *The redex at* γ *is top-level.* We consider each of the markable redexes separately, ignoring the impossible top-level (I) step.
  - *The redex at* γ *is a* (V) *step.* So we have $M \equiv (\text{let } x = V_0 \text{ in } M_1)$. If δ is within V, then all residuals will be internal since they will still be under an abstraction. Otherwise if δ is within M_1 we have the result by Lemma 4.6.2.
  - *The redex at* γ *is a* $(\star C)$ *step.* The rank of the contraction here is irrelevant, and we take the simplest case of rank one. We have $M \equiv (\text{let } x = M_0 \text{ in } M_1)\ M_2$ where M_1 is not an answer. If δ is within an M_i the result is trivial; in the former case we observe that x will be demanded in M_1 after the contraction of γ only if it is also demanded before. If δ refers to the let-binding $(\text{let } x = M_0 \text{ in } M_1)$ itself, and we have M_0 appropriate for a top-level (V, A) redex, then it is clear that (*e.g.* for a (V) step) the demand on x does not change by contracting the redex at γ.
  - *The redex at* γ *is a* $(\star A)$ *step.* As in the previous case for $(\star C)$ steps.
- *The redex at* δ *is top-level.* So δ cannot be an (I) step; then to guarantee that subterms remain in the appropriate forms for an internal $(V, \star C, \star A)$ step we rely on Lemma 4.6.8 and Lemma 4.6.10.

For γ, δ in distinct subterms it again follows from Lemma 4.6.8 and Lemma 4.6.10 that where δ had been in a non-evaluation context we will not create an evaluation context around the residual of δ by contracting γ, and so the result does follow. ♦

Lemma 4.6.12. *Generalized* $(I, V, \star C, \star A)$*-reduction satisfies* ***P3****:* Let $\sigma : M_0 \xrightarrow[\mathrm{i}]{} N_0$ and let $N_0 \mapsto N_1$ in $\mathsf{Need}^\star$. Then M_0 has a prestandard redex: there exists some M_1 such that $M_0 \mapsto M_1$.

Proof: So we have

$$\begin{aligned} \sigma : M_0 \equiv C_0[\Delta_0] &\xrightarrow[\mathrm{i}]{} C_0[\Delta_0'] \equiv N_0 \\ N_0 \equiv E_1[\Delta_1] &\mapsto E_1[\Delta_1'] \equiv N_1 \ \ , \end{aligned}$$

and we seek to construct the E, Δ such that $M_0 \equiv E[\Delta]$ and Δ is a prestandard step. We proceed by structural induction on M_0, and perform an analysis on C_0 and E_1:

- $C_0 \equiv [\,]$. We construct E based on the specific rule which contracts Δ_0. As usual, we cannot have a (I) step as both top-level and internal.
  - *By the* (V) *rule, with* $M_0 \equiv \Delta_0 \equiv (\text{let } x = V \text{ in } M_1)$ *and* M_1 *not demanding* x. We have $N_0 \equiv M_1[x := V]$, and the case by Lemma 4.6.6.
  - *By the* $(\star C)$ *rule, with* $M_0 \equiv \Delta_0 \equiv (P^n\, L_1)\, L_2$*, where* L_1 *is not an answer.* So we have $N_0 \equiv \Delta_0 \equiv P^n\, (L_1\, L_2)$. One case is not possible:
    - $E_1 \equiv P^n\,[\,]$*, with* $\Delta_1 \equiv L_1\, L_2 \equiv (\lambda x.\, L)\, L_2$. Then we would have σ as prestandard rather than internal.

    One simple possibility is:
    - $E_1 \equiv P^n\,(E_2\, L_2)$. Then $E \equiv (P^n\, E_2)\, L_2$ and $\Delta \equiv \Delta_1$.

    For the remaining cases we must look into the structure of the P^n, and so we take

    $$P^n\,(L_2\, L_2) \equiv \text{let } x_1 = M_1 \text{ in} \cdots \text{in let } x_n = M_n \text{ in } (L_2\, L_2) \ .$$

    We also have some E_0 and one of these x_i such that $L_1 \equiv E_0[x_i]$.
    - $E_1 \equiv (\text{let } x_1 = M_1 \text{ in} \cdots \text{in let } x_{i-1} = M_{i-1} \text{ in } [\,])$ *and*

      $$\Delta_1 \equiv (\text{let } x_i = A \text{ in} \cdots \text{in let } x_n = M_n \text{ in } (E_0[x_i]\, L_2)) \ .$$

      So depending on the structure of A we have either a (V) or a $(\star A)$ prestandard step. Either way, we have $E \equiv E_1\, L_2$ and $\Delta \equiv (\text{let } x_i = A \text{ in} \cdots \text{in let } x_n = M_n \text{ in } E_0[x_i])$.
    - *For*

      $$\begin{aligned} E_1 \equiv (&\text{let } x_1 = M_1 \\ &\text{in} \cdots \text{in let } x_i = E_2 \\ &\qquad \text{in} \cdots \text{in let } x_n = M_n \\ &\qquad\qquad \text{in } (E_0[x_i]\, L_2)) \end{aligned} .$$

      Then we have

      $$\begin{aligned} E \equiv (&\text{let } x_1 = M_1 \\ &\text{in} \cdots \text{in let } x_i = E_2 \\ &\qquad \text{in} \cdots \text{in let } x_n = M_n \\ &\qquad\qquad \text{in } (E_0[x_i])\, L_2) \end{aligned}$$

      and $\Delta \equiv \Delta_1$.

    Clearly we have reviewed all of the possibilities for the evaluation context/redex decomposition, so the case is complete.
  - *By the* $(\star A)$ *rule.* By an analysis as the one above for the $(\star C)$ rule.
  - *By the* (G) *rule.* Taking $M_0 \equiv (\text{let } x = M_1 \text{ in } M_2)$, we clearly have $E \equiv (\text{let } x = M_1 \text{ in } E_1)$.
- *For other* C_0*, we begin with the special case where* $E_1 \equiv [\,]$. This result is immediate: if σ is in fact internal, then it is clear that we must have $E \equiv [\,]$ as well.

Otherwise the result is straightforward. If the topmost configuration of C_0 is that of an evaluation context — that is, if C_0 takes one of the forms

$$\begin{aligned} C_0 &\equiv C_1\, L \\ C_0 &\equiv \text{let } x = L \text{ in } C_1 \\ C_0 &\equiv \text{let } x = C_1 \text{ in } E_0[x] \end{aligned} \tag{4.1}$$

where in Equation (4.1), $C_1[\Delta_0]$ does not demand x — then we have the result by the induction hypothesis. Finally, when the internal and prestandard contractions are in different subterms, we have the result immediately by Lemma 4.6.8 and Lemma 4.6.10. ♦

Lemma 4.6.13. *Generalized* $(I, V, \star C, \star A)$*-reduction satisfies* ***P4****:* Let $\sigma : M_0 \underset{\mathrm{i}}{\rightarrow} N_0$ and $M_0 \overset{\delta}{\mapsto} N_1$ in $\mathsf{Need}^\star$. Then δ/σ contains a single element δ_1 which is a prestandard redex of N_0: $N_0 \overset{\delta_1}{\longmapsto} N$.

Proof: The analysis is similar to that of the previous proof. We proceed by structural induction on M_0. The inductive case where both contractions are in the same subterm is trivial, and we detail only the three base cases: where one of the two redexes is top-level, and where each is in a distinct subterm of the top-level expression. We take

$$\begin{aligned} \sigma : M_0 \equiv \quad & C_0[\Delta_0] \underset{\mathrm{i}}{\rightarrow} C_0[\Delta_0'] & \equiv N_0 \\ \sigma : M_0 \equiv \quad & E_1[\Delta_1] \mapsto E_1[\Delta_1'] & \equiv N_1 \ , \end{aligned}$$

and construct in each case the required E and Δ.

- $E_0 \equiv [\,]$. This case is the simplest; the result follows by a consideration of the structure of the terms following from each possible redex by Lemma 4.6.8 and Lemma 4.6.10.
- $C_0 \equiv [\,]$ *and* $E_0 \not\equiv [\,]$. So we have $\Delta_0 \equiv M_0$ in the form of an evaluation context, and moreover again σ is not an (I) step. We consider the different applicable rules for σ_0 separately; all of these cases are quite similar to the corresponding steps of the the proof of the previous Lemma 4.6.12.
  - $\sigma : M_0 \equiv (\text{let } x = V \text{ in } M_1) \xrightarrow[\text{i-}(V)]{} M_1[x := V]$, *where x is not demanded in M_1*. So clearly the prestandard step in within M_1, and by Lemma 4.6.5 is the residual again; singularity of the residual is clear since within M_1 it could be neither duplicated nor discarded.
  - $M_0 \equiv (P^n\, M_1)\, M_2 \xrightarrow[\text{i-}(\star C)]{} P^n\, (M_1\, M_2)$, *where M_1 is not an answer.* Again, singularity of the residual is clear since no terms are duplicated or discarded. We proceed with an analysis of E_1.
    - *The prestandard redex is within M_1:* $E_1 \equiv (P^n\, E_2)\, M_2$. Then the result is trivial, with $E \equiv P^n\, (E_2\ M_2)$.

    For the remaining cases we have some variable x_i, bound within the P^n, demanded by M_1, and we take

    $$\begin{aligned} (P^n\, M_1)\, M_2 \quad & \equiv \quad (\text{let } x_1 = L_1 \text{ in} \cdots \text{in let } x_n = L_n \text{ in } M_1)\, M_2 \\ M_1 \quad & \equiv \quad E_2[x_i] \ . \end{aligned}$$

    - $E_1 \equiv (\text{let } x_1 = L_1 \text{ in} \cdots \text{in let } x_{i-1} = L_{i-1} \text{ in } [\,])\, M_2$, *where*

      $$\Delta_1 \equiv \text{let } x_i = A \text{ in} \cdots \text{in let } x_n = L_n \text{ in } E_2[x_i] \ .$$

      Here we have simply

      $$\begin{aligned} E \quad & \equiv \quad \text{let } x_1 = L_1 \text{ in} \cdots \text{in let } x_{i-1} = L_{i-1} \text{ in } [\,]) \\ \Delta \quad & \equiv \quad \text{let } x_i = A \text{ in} \cdots \text{in let } x_n = L_n \text{ in } E_2[x_i]\, M_2 \end{aligned}$$

    - *For*

      $$\begin{aligned} E_1 \equiv (&\text{let } x_1 = L_1 \\ &\text{in} \cdots \text{in let } x_{i-1} = L_{i-1} \\ &\qquad \text{in} \cdots \text{in let } x_n = L_n \\ &\qquad\qquad \text{in } E_2[x_i])\, M_2 \ . \end{aligned}$$

      Then we have just

      $$\begin{aligned} E \quad \equiv \quad & \text{let } x_1 = L_1 \\ & \text{in} \cdots \text{in let } x_{i-1} = L_{i-1} \\ & \qquad \text{in} \cdots \text{in let } x_n = L_n \\ & \qquad\qquad \text{in } (E_2[x_i]\, M_2) \\ \Delta \quad \equiv \quad & \Delta_1 \ . \end{aligned}$$

  - $M_0 \equiv (\text{let } x = (P^n\, M_1) \text{ in } M_2) \xrightarrow[\text{i-}(\star A)]{} P^n\, (\text{let } x = M_1 \text{ in } M_2)$. By an analysis of where the prestandard redex is, as above for $(\star C)$ steps.
  - $M_0 \equiv (\text{let } x = M_1 \text{ in } M_2) \xrightarrow[\text{i-}(G)]{} M_2$. So we must have $E_1 \equiv \text{let } x = M_1 \text{ in } E_2$, and clearly we have a unique $E \equiv E_2$ and $\Delta \equiv \Delta_1$.
- *The two redexes are in different subterms.* Trivially here the prestandard step — being outside of the effect of the internal step — could not be duplicated or discarded, and so the residual is unique; by Lemma 4.6.8 and Lemma 4.6.10 it is again a prestandard step.

The inductive step, closure of both redexes within the same evaluation context, is trivial, and the proof is complete. ♦

Lemma 4.6.14. Let $M \xrightarrow[(I,V,\star C,\star A)]{} \!\!\!\!\rightarrow A$; then there exists some A_0 such that

$$M \mathrel{\mapsto\!\!\!\!\rightarrow}_{\mathsf{Need}^\star} A_0 \xrightarrow[(I,V,\star C,\star A)]{} \!\!\!\!\rightarrow A \ .$$

Proof: By the general technique of Lemma 2.1.20, here with Lemma 4.6.9, Lemma 4.6.10, Lemma 4.6.11, Lemma 4.6.12 and Lemma 4.6.13. ♦

Lemma 4.6.15. Let $M \xrightarrow[(G)]{} M_0 \mathrel{\mapsto}_{\mathsf{Need}^\star} N$. Then there exists some N_0 such that

$$M \mathrel{\mapsto}^{=}_{\mathsf{Need}^\star} N_0 \xrightarrow[(G)]{} \!\!\!\!\rightarrow N \ .$$

Proof: The result is a straightforward analysis of the relative positions of the redexes. We have the (G) step duplicated or discarded when it is within the substituted value of a (V) step, and the prestandard step removed when the (G) rule removes a binding from a $(\star C, \star A)$ redex of rank one. ♦

Lemma 4.6.16. Let $\sigma : M \xrightarrow[\mathsf{Need}^\star]{} \!\!\!\!\rightarrow A$; then there exists some A_0 such that

$$M \mathrel{\mapsto\!\!\!\!\rightarrow}_{\mathsf{Need}^\star} A_0 \xrightarrow[\mathsf{Need}^\star]{} \!\!\!\!\rightarrow A \ .$$

Proof: By induction on the number of non-standard $\mathsf{Need}^\star$ steps in σ. If the last such step in σ is a (G) step, then we reduce the problem to the inductive hypothesis by Lemma 4.6.15 and Lemma 4.6.10. If the last non-standard step is an internal $(I, V, \star C, \star A)$ step, then the result follows from Lemma 4.6.14. ♦

We can now deduce the main standardization result for call-by-need.

Proposition 4.6.17, $\mathtt{Std}(\mathsf{Need})$**.** The relation $\mapsto_{\mathsf{Need}}$ is a standard strategy for reduction by $\xrightarrow[\mathsf{Need}]{}$ to answers.

Proof: We have that $\mapsto_{\mathsf{Need}}$ selects a unique redex for every (closed) non-answer and no redex for every answer by Lemma 4.6.1, and that it always finds an answer when one is available by Lemma 4.6.16 with Lemma 4.4.1. ♦

4.7 Other approaches to call-by-need

Independently and concurrently, Ariola and Felleisen proposed a similar system, which corresponds to the standard reductions of our calculus [3, 4]. The rules which we use for our standard evaluation relation $\mapsto$, they take as the basis for the general reduction relation $\rightarrow$ as well. The restriction makes a number of proofs easier, since the problem which we solve with the generalized (C, A) reduction rules simply does not arise; their system can also be seen as closer to the literal behavior of implementations of sharing in the superficial sense that their system allows fewer nonstandard steps. On the other hand, fewer program transformations can be expressed as equalities, even though the computational properties of both calculi are equivalent. In their later work [4], they also consider the relationship of the calculus with the stronger notion of *fully lazy* graph reduction [80], and with a speculative parallel implementation of non-strictness. Finally, their work includes the *recursive letrec* calculus, an extension for mutually recursive bindings, based on work of Ariola and Klop [6]. Interestingly, reduction in the recursive letrec calculus is confluent only with the restricted form of general reduction rules that correspond to our rules for the $\mapsto$ relation, closed under arbitrary contexts [6].

Several other methods for modeling call-by-need have been studied. Okasaki, Lee and Tarditi [61] adapted Danvy and Hatcliff's [22, 30] treatment of the call-by-name continuation-passing style (CPS) transform for call-by-need. Danvy and Hatcliff decomposed the call-by-name CPS mapping into two components: first, a translation from call-by-name into a call-by-value calculus extended with *thunks*, and second, the standard call-by-value CPS transform, extended to translate the new thunking constructs. Okasaki, Lee and Tarditi develop a call-by-need CPS transform in two steps. First, the thunk extension of call-by-value is extended further, with a distinct updating thunk. Then, the target of the call-by-value CPS transform is taken to be the CPS-calculus extended with store operations (allocate, update, dereference). They show this transformation to be correct with respect to a graph-reduction implementation. This technique should be useful in studying implementation models, but is considerably more complicated than the call-by-need calculi, and would seem a difficult tool for reasoning about programs. Josephs [39] gives a continuation and store-based denotational semantics of lazy evaluation, but no consideration of operational properties.

Purushothaman and Seaman [63] and Launchbury [44] give operational semantics for lazy evaluation using a natural semantics with an explicit, global store. Purushothaman and Seaman give a structured operational semantics of call-by-name PCF with explicit environments which is then shown to be equivalent to a standard denotational semantics for PCF. Their system is rather complicated, which is due only in part to their incorporation of all of PCF, rather than just the lambda calculus. Launchbury [44] presents a simpler system with a simpler operational semantics and gives in addition rules for recursive Let-bindings that capture call-by-need sharing behavior. The key point about both approaches is that they provide an operational model of call-by-need, but do not provide anything like a calculus or a reduction system. With a single, global store, it is difficult to reason about parts of programs independent of a specific context.

The issues we tackle with the call-by-need calculus are distinct from both the study of explicit substitutions and the idea of optimal reduction. Explicit substitution [1] treats the act of substitution within the calculus, rather than at the metalevel we have used here. The explicit substitutions are certainly a form of resource-consciousness, but nonetheless a different form, the key difference lying in the substitution propagating rule

$$(App) \qquad (M\ N)[x := L] \to (M[x := L])\,(N[x := L]) \ .$$

which of course we would want to apply only when L is a value.

Explicit substitutions are frequently associated with optimal reduction, which requires a much stronger notion of sharing than the style of implementation which we model with call-by-need. A number of researchers [1, 24, 55] have studied reductions that preserve sharing in calculi with explicit substitutions, especially in relation to optimal reduction strategies. Having different aims, the resulting calculi are considerably more complex than those presented here. Yoshida [81] presents a weak lambda calculus with explicit environments similar to Let constructs, and gives an optimal reduction strategy. Her calculus subsumes several of our reduction rules as structural equivalences. However, due to a different notion of observation, reduction in this calculus is not equivalent to reduction to weak head-normal form.

Kamareddine and Nederpelt introduced an alternate *item notation* for terms on the Barendregt cube [40] which admits a notion of *generalized reduction* [17] that is very succinct, and that achieves a part of what our system does. Basically, it marks arguments with a δ, writes these marked arguments to the left of functions, and writes abstractions as separate prefixes before their bodies. So where we write

$$\begin{aligned} M_0 &\equiv (\lambda x.\, \lambda y.\, M_1)\, M_2\, V \\ N_0 &\equiv (\lambda x.\, N_1)\, ((\lambda y.\, \lambda z.\, V)\, N_2) \ , \end{aligned}$$

in their notation we would have

$$\begin{aligned} M_0 &\equiv (V\delta)(M_2\delta)(\lambda x)(\lambda y)(M_1) \\ N_0 &\equiv (((N_2\delta)(\lambda y)(\lambda z)V)\delta)(\lambda x)(N_1) \ . \end{aligned}$$

Their notion of generalized reduction allows beta reduction to occur around balanced argument/abstraction pairs. One can view $(M\delta)$ elements as left bracketing, (λx) elements as right bracketing, and give a context-free definition of what we mean by a balanced prefix:

$$S ::= \cdot \mid (M\delta)S(\lambda x) \mid S_1\, S_2 \ .$$

Then the generalized reduction relation $\hookrightarrow$ has a very simple definition,

$$(N\delta)S(\lambda x)M \hookrightarrow S(M[x := N]) \ .$$

Their system is easiest to compare with Need_ℓ, the let-less formulation of call-by-need [5, 53, 54], which we summarize in Figure 4.8. Under these rules it is clear

Syntax and typing rules As in Val.

Reduction rules

$$
\begin{array}{llll}
(V_\ell) & (\lambda x.\, M)\ V & \to & M[x := V] \\
(C_\ell) & (\lambda x.\, L)\ M\ N & \to & (\lambda x.\, L\ N)\ M \\
(A_\ell) & (\lambda x.\, L)\ ((\lambda y.\, M)\ N) & \to & (\lambda y.\, (\lambda x.\, L)\ M)\ N \\
(G_\ell) & (\lambda x.\, M)\ N & \to & M, \qquad x \notin \mathrm{fv}(M)
\end{array}
$$

Figure 4.8: The let-less formulation of call-by-need Need_ℓ.

that M_0 is a (C_ℓ) redex, and that N_0 is an (A_ℓ) redex, and moreover that they will each expose a (V_ℓ) step not contractable at first.

Under the item notation, M_0 is already a $\hookrightarrow$ redex, where the result is equivalent to the result of contracting first the (C_ℓ) step, and then the (V_ℓ) step. In Need_ℓ we have

$$
\begin{array}{ll}
 & M_0 \\
\equiv & (\lambda x.\, \lambda y.\, M_1)\ M_2\ V \\
\xrightarrow[(C_\ell)]{} & (\lambda x.\, (\lambda y.\, M_1)\ V)\ M_2 \\
\xrightarrow[(V_\ell)]{} & (\lambda x.\, M_1[y := V])\ M_2
\end{array}
$$

while under generalized reduction we have

$$
\begin{array}{ll}
 & M_0 \\
\equiv & (V\delta)(M_2\delta)(\lambda x)(\lambda y)(M_1) \\
\hookrightarrow & (M_2\delta)(\lambda x)(M_1[y := V])
\end{array}
$$

whose final terms are clearly equivalent.

In N_0 we also have a $\hookrightarrow$ step, but too much is copied for a by-need regime: $\hookrightarrow$ cannot simulate (A_ℓ) steps. In Need_ℓ we have

$$
\begin{array}{ll}
 & N_0 \\
\equiv & (\lambda x.\, N_1)\ ((\lambda y.\, \lambda z.\, V)\ N_2) \\
\xrightarrow[(A_\ell)]{} & (\lambda y.\, (\lambda x.\, N_1)\ (\lambda z.\, V))\ N_2 \\
\xrightarrow[(V_\ell)]{} & (\lambda y.\, N_1[x := \lambda z.\, V])\ N_2
\end{array}
$$

but under generalized reduction,

$$
\begin{array}{ll}
 & N_0 \\
\equiv & (((N_2\delta)(\lambda y)(\lambda z)V)\delta)(\lambda x)(N_1) \\
\hookrightarrow & N_1[x := ((N_2\delta)(\lambda y)(\lambda z)V)]\ ,
\end{array}
$$

which is clearly not equivalent.

It is straightforward to extend $\hookrightarrow$ to a system which expresses call-by-need. Recalling the item notation rendering for the definition of a value,

$$V ::= x \mid (\lambda x)(M)\ ,$$

we have a very simple item notation description of call-by-need answers:

$$A ::= S(\lambda x)(M)\ .$$

Just as the generalized rule already accounts for (C_ℓ) steps, We can account for (A_ℓ) steps there as well. We extend the rule for $\hookrightarrow$ to a call-by-need generalized rule $\xhookrightarrow[\mathsf{Need}]{}$,

$$((S_0\, V)\delta)S_1(\lambda x)M \ \xhookrightarrow[\mathsf{Need}]{}\ S_0\, S_1\, (M[x := V])\ ,$$

plus a garbage collection rule

$$(M\delta)S(\lambda x)N \ \xhookrightarrow[\mathsf{Need}]{}\ S\, N, \text{ if } x \notin \mathrm{fv}(N)\ .$$

It is clear that $\xhookrightarrow[\mathsf{Need}]{}$ reduces terms to answers exactly when Need_ℓ does so as well.

Bloo, Kamareddine and Nederpelt [17] study the behavior of this approach in the various systems on the Barendregt cube [10]. A number of interesting problems arise for general reduction under non-simple types; we discuss the relationship of our work in general with the Barendregt cube in Section 10.1 of the conclusion.

Part II

A Mode for Weakening and Contraction

Chapter 5

The linear lambda calculus

So, it begins.
(Straczynski)

UP to this point, we have considered traditional calculi with intuitionistic logic underlying their type systems. We now turn our attention to a pair of calculi based on an intuitionistic fragment of Girard's linear logic and Logic of Unity [26, 27]. The key point of these systems is that the structural typing rules of Weakening and Contraction are no longer freely applicable anywhere in a typing proof, but rather only in conjunction with the new ! connective.

Figure 5.1 presents the details of the linear lambda calculus Lin. Types now correspond to formulas of the minimal intuitionistic fragment of linear logic which we discussed in Section 1.1.2.3: a base type, a modal type or a linear function. Terms are either variables, introducers or eliminators for the new connective, or function abstractions or applications. We use the same metanotation as in previous systems. Again, type environments are sets containing assumptions of the form $x : A$. In the context of dual calculi, we let Φ and Ψ range over intuitionistic assumptions, and Γ and Δ range over linear assumptions. In a judgment

$$\Phi;\ \Gamma \vdash M : A$$

type environments Φ and Γ should bind disjoint sets of identifiers. If Γ and Δ are any two type environments with distinct variables, then Γ, Δ denotes their union. We write '$-$' for an empty type environment. We take type environments to be sets rather than lists, so the order of bindings is irrelevant, and we may moreover safely omit Exchange rules.

A typing judgment $\Phi;\ \Gamma \vdash M : A$ indicates that in the intuitionistic type environment Φ plus linear type environment Γ, term M has type A. If the judgment

$$x_1 : A_1, \ldots, x_m : A_m;\ y_1 : B_1, \ldots, y_n : B_n \vdash M : C \tag{5.1}$$

holds then the free variables of M will be drawn from $x_1, \ldots, x_m$, each of which occurs any number of times, and $y_1, \ldots, y_n$, each of which occurs exactly once.

There are five rules concerned with the modal connective: Dereliction, Contraction and Weakening are structural rules, while !-I introduces and !-E eliminates the ! connective. An intuitionistic assumption can only be introduced by the !-E rule. The only thing that we may do with such an assumption is to duplicate it via Contraction, discard it via Weakening or convert it to a linear assumption via Dereliction. Furthermore, it is only intuitionistic assumptions that can appear in the !-I rule, as the empty linear type environment indicates. Analogous to the conclusion $-;\ x : A \vdash x : A$ of rule Id, we can derive the conclusion $x : A;\ - \vdash x : A$ by combining Id with Dereliction, and then $x : A;\ - \vdash\ !x :!A$ with !-I.

The reduction relation is specified by two beta rules, $(\beta\multimap)$ and $(\beta!)$, and two commuting rules, $(!\multimap)$ and $(!!)$, with compatible closure as usual. We use the same conventions and notations for reduction in Lin as in previous systems.

Each of the reduction rules has a logical basis.

- Rule $(\beta\multimap)$ arises when a $\multimap$-I rule meets a $\multimap$-E rule and the two rules annihilate.
- Rule $(\beta!)$ arises when a !-I rule meets a !-E rule and the two rules annihilate.
- Rule $(!\multimap)$ arises when a !-E rule meets a $\multimap$-E rule, commuting one through the other.
- Rule $(!!)$ arises when two !-E rules meet, commuting one through the other.

The rules are compatible with an operational interpretation where one evaluates ($\text{let } !x = M \text{ in } N$) by first evaluating M to the form $!M_0$ and then evaluating $N[x := M_0]$. Thus we may view the term associated with !-E as forcing evaluation and the term associated with !-I as suspending evaluation.

Syntactic domains

Types	A, B	$::=$	$Z \mid\ !A \mid A \multimap B$
Terms	L, M, N	$::=$	$x \mid\ !M \mid \text{let } !x = M \text{ in } N \mid \lambda x.\, M \mid M\ N$

Typing judgments

$$\frac{}{-;\ x : A \vdash x : A}\ \text{Id} \qquad \frac{\Phi,\ y : A,\ z : A;\ \Gamma \vdash M : B}{\Phi,\ x : A;\ \Gamma \vdash M[y := x, z := x] : B}\ \text{Contraction}$$

$$\frac{\Phi;\ \Gamma \vdash M : B}{\Phi,\ x : A;\ \Gamma \vdash M : B}\ \text{Weakening} \qquad \frac{\Phi;\ x : A,\ \Gamma \vdash M : B}{\Phi,\ x : A;\ \Gamma \vdash M : B}\ \text{Dereliction}$$

$$\frac{\Phi;\ - \vdash M : A}{\Phi;\ - \vdash\ !M :!A}\ \text{!-I} \qquad \frac{\Phi;\ \Gamma \vdash M : !A \qquad \Psi,\ x : A;\ \Delta \vdash N : B}{\Phi,\ \Psi;\ \Gamma,\ \Delta \vdash \text{let } !x = M \text{ in } N : B}\ \text{!-E}$$

$$\frac{\Phi;\ \Gamma,\ x : A \vdash M : B}{\Phi;\ \Gamma \vdash \lambda x.\, M : A \multimap B}\ \multimap\text{-I} \qquad \frac{\Phi;\ \Gamma \vdash M : A \multimap B \qquad \Psi;\ \Delta \vdash N : A}{\Phi,\ \Psi;\ \Gamma,\ \Delta \vdash M\ N : B}\ \multimap\text{-E}$$

Reduction relation

$(\beta\multimap)$	$(\lambda x.\, M)\ N$	$\to$	$M[x := N]$
$(\beta!)$	let $!x =\ !M$ in N	$\to$	$N[x := M]$
$(!\multimap)$	(let $!x = L$ in M) N	$\to$	let $!x = L$ in $(M\ N)$
$(!!)$	let $!y =$ (let $!x = L$ in M) in N	$\to$	let $!x = L$ in (let $!y = M$ in N)

Figure 5.1: The linear lambda calculus Lin.

5.1 Substitutions

When we consider systems where the typing assumptions are divided into zones, we will have one result for each zone where we might find the typing assumption for x, with restrictions on the typing of N appropriate to the zone from which that assumption on x is drawn. In rule $(\beta\multimap)$ the variable x appears linearly in M, so any free variable that appears linearly in N will still appear linearly in $M[x := N]$. Hence the substitution is well formed. In rule $(\beta!)$ the variable x may appear any number of times in N, so a free variable of M may be copied arbitrarily many times in $N[x := M]$. It is here that distinguishing the two sorts of assumptions is particularly helpful: the constraint on the !-I rule guarantees that the term $!M$ may only contain free variables that can appear any number of times. Hence this substitution is also well formed. Lemma 5.1.1 below makes these ideas concrete. Since we express well-formedness in terms of the type system, the lemma says that the term which results from substituting of a well-typed term for a free variable in other well-typed terms is again well-typed. This result corresponds to Cut theorems in the logic.

Lemma 5.1.1. Let $M, N \in$ Lin.

(i) Let $\Phi;\ \Gamma, x : A \vdash M : B$ and $\Psi;\ \Delta \vdash N : A$. Then $\Phi, \Psi;\ \Gamma, \Delta \vdash M[x := N] : B$.

(ii) Let $\Phi, x : A;\ \Gamma \vdash M : B$ and $\Psi;\ - \vdash N : A$. Then $\Phi, \Psi;\ \Gamma \vdash M[x := N] : B$.

Proof: By induction on the typing trees. ♦

5.2 Subject reduction

The proof of the subject reduction result for Lin is essentially the same as in the result for Let.

Proposition 5.2.1. Lin satisfies subject reduction.

Syntactic domains As in Lin, plus:

$$\text{Prefixes} \quad P \ ::= \text{let } !x = M \text{ in}$$

Typing Judgments As in Lin.

Reduction Rules

$$\begin{array}{lllll}
(\beta\text{–}\!\circ) & (\lambda x.\, M)\, N & \to & M[x := N] \\
(\beta!) & \text{let } !x = !M \text{ in } N & \to & N[x := M] \\
(\star\text{–}\!\circ) & (P^n\, M)\, N & \to & P^n\,(M\, N) \\
(\star!) & \text{let } !x = (P^n\, M) \text{ in } N & \to & P^n\,(\text{let } !x = M \text{ in } N)
\end{array}$$

Figure 5.2: The generalized linear lambda calculus, $\mathsf{Lin}^\star$.

Proof: The proof of Proposition 5.2.1 is straightforward, its essence being the logical content of the reduction rules listed above. We must make some technical accounting for moving typing rules not reflected in terms' structure to a position "above" other rules which are so reflected: briefly, all uses of Dereliction and Weakening may be moved toward the leaves of the type tree, uses of Contraction on pairs of variables from different subterms may be pushed toward the root, and uses of Contraction of pairs of variables from the same subterm may be moved toward the leaves. These commutings are easily shown by simple structural inductions.

The result for each of the four rules is similar. For $(\beta\text{–}\!\circ)$ applied to a well-typed term, we have $\Phi;\ \Gamma \vdash (\lambda x.\, M)N : B$, and since we can clearly commute all but possibly some Contraction steps to above the $\multimap$-E step, we have an application of the $\multimap$-E rule leading to this first judgment by contractions, and similarly we can commute all other steps to above the $\multimap$-I step, and have a $\multimap$-I immediately above the $\multimap$-E:

$$\dfrac{\dfrac{\dfrac{\dfrac{\vdots\ \mathcal{F}}{\Phi_0;\ \Gamma_0, x : A \vdash M_0 : B}}{\Phi_0;\ \Gamma_0 \vdash \lambda x.\, M_0 : A \multimap B}\multimap\text{-I} \qquad \dfrac{\vdots\ \mathcal{G}}{\Psi_0;\ \Delta_0 \vdash N_0 : A}}{\Phi_0, \Psi_0;\ \Gamma_0, \Delta_0 \vdash (\lambda x.\, M_0)\, N_0 : B}\multimap\text{-E}}{\dfrac{\vdots}{\Phi;\ \Gamma \vdash (\lambda x.\, M)\, N : B}}$$

Then with $\mathcal{F}$, $\mathcal{G}$ and Lemma 5.1.1.(i) we have subject reduction. For rule $(\beta!)$ we follow the same reasoning, building a type tree for a let-binding to a !-prefixed term rather than an application of an abstraction. For the $(!\text{–}\!\circ, !!)$ rules we need only observe that structural rules can be commuted above the relevant elimination steps, and do not need Lemma 5.1.1. ♦

5.3 Developments

One can develop a theory of simple markings, developments and well-separation for the linear lambda calculus more or less exactly as for call-by-let, with the similar tradeoffs of system complexity versus proof complexity. Here we consider only developments of reduction sequences according to the generalized notion of marked redex.

5.3.1 Generalized Lin reduction

Figure 5.2 shows the syntax and reduction rules of the generalized linear lambda calculus. We again define the *rank* of $(\star)$ steps to be the number of prefixes moved. We have the usual correspondence between generalized and ordinary steps. Reduction in $\mathsf{Lin}^\star$ clearly corresponds to reduction in Lin.

Lemma 5.3.1. Let $M, N \in \mathsf{Lin}$. Then $M \xrightarrow[\mathsf{Lin}]{}\!\!\!\!\!\!\rightarrow N$ if and only if $M \xrightarrow[\mathsf{Lin}^\star]{}\!\!\!\!\!\!\rightarrow N$.

$$\begin{array}{rcl}
\mathbf{d}(M \to N, 0) &=& 0 \\
\mathbf{d}((\lambda x.\, M)\, N \to M[x := N], n) &=& 0 \\
\mathbf{d}(\text{let } !x = !N \text{ in } M \to M[x := N], n) &=& -1 \ \ , \ n \geq 0 \\
\mathbf{d}(\text{let } !x = (P^m\, M) \text{ in } N \to P^n\, (\text{let } !x = M \text{ in } N), n) &=& m \ \ , \ n \geq 0 \\
\mathbf{d}((P^m\, M)\, N \to P^m\, (M\, N), n) &=& 0 \\
\mathbf{d}(P^m\, M \to M, n) &=& -m \ \ , \ n \geq 0 \\
\mathbf{d}(L\, M \to L\, N, n) &=& 0 \\
\mathbf{d}(M\, L \to N\, L, n) &=& 0 \\
\mathbf{d}(\text{let } !x = L \text{ in } M \to \text{let } !x = L \text{ in } N, n) &=& \mathbf{d}(M \to N, n-1) \ \ , \ n \geq 0 \\
\mathbf{d}(\text{let } !x = M \text{ in } L \to \text{let } !x = N \text{ in } L, n) &=& 0 \\
\mathbf{d}(!M \to !N, n) &=& 0 \\
\mathbf{d}(\lambda x.\, M \to \lambda x.\, N, n) &=& 0
\end{array}$$

Figure 5.3: The damage function for $\mathsf{Lin}^\star$.

Syntactic domains

$$\begin{array}{llcl}
\text{Terms} & L', M', N' & ::= & x \mid \lambda x.\, M' \mid !M' \mid M'\, N' \\
& & \mid & \text{let } !x = M' \text{ in } N' \\
& & \mid & {}^{[\beta\multimap]}(\lambda x.\, M')\, N' \\
& & \mid & {}^{[\beta !]}\text{let } !x = !M' \text{ in } N' \\
& & \mid & {}^{[\star !_n]}\text{let } !x = \mathcal{P}^n\, M' \text{ in } N' \\
& & \mid & {}^{[\star\multimap_n]}(\mathcal{P}^n\, M')\, N' \\
\text{Markable prefixes} & \mathcal{P} & ::= & \text{let } !x = M' \text{ in} \\
& & \mid & {}^{[\beta !]}\text{let } !x = M' \text{ in} \\
& & \mid & {}^{[\star !_n]}\text{let } !x = \mathcal{P}^n\, M' \text{ in}
\end{array}$$

Degenerate Equivalences

$$\begin{array}{rcl}
{}^{[\star !_0]}\text{let } !x = M' \text{ in } N' &\equiv& \text{let } !x = M' \text{ in } N' \\
{}^{[\star\multimap_0]}M'\, N' &\equiv& M'\, N'
\end{array}$$

Figure 5.4: Syntax of the generalized marked linear lambda calculus $\mathsf{Lin}^{\star\prime}$.

Proof: In the forward direction, we simple replace each $(!\multimap, !!)$ step with a $(\star\multimap, \star!)$ step of rank 1; in the backward direction, we replace a $(\star\multimap, \star!)$ of rank n with n consecutive $(!\multimap, !!)$ steps. ♦

That proposition allows us to transfer many properties back and forth from Lin to $\mathsf{Lin}^\star$ with little effort. For example, since Lin and $\mathsf{Lin}^\star$ share a term language and typing rules, subject reduction is immediate:

Corollary 5.3.2. Reduction in $\mathsf{Lin}^\star$ satisfies the subject reduction property: let $\Gamma \vdash_{\mathsf{Lin}^\star} M : A$ and $M \xrightarrow[\mathsf{Lin}^\star]{} N$. Then $\Gamma \vdash_{\mathsf{Lin}^\star} N : A$.

Proof: From SR(Lin) by the above. ♦

The damage function $\mathbf{d}$ for generalized linear lambda, shown in Figure 5.3, plays the same role in compatible closure of the marked terms of the next section as in $\mathsf{Let}^\star$.

5.3.2 Marked reduction and developments

Figure 5.4 presents the syntax of the general marked linear lambda calculus $\mathsf{Lin}^{\star\prime}$, and Figure 5.5 describes reduction in that system. As usual, we take $\mathsf{Lin}^{\star\prime}_0$ reduction to mean any step or sequence by the $(\beta\multimap_0, \beta!_0, \star\multimap_0, \star\multimap_0^<, \star!_0, \star!_0^<)$ rules, and $\mathsf{Lin}^{\star\prime}_1$ reduction to refer to reduction by any other rules.

The usual substitution results will be helpful:

Reduction rules

$(\beta\multimap_0)$	${}^{[\beta\multimap]}(\lambda x.\, M')\, N'$	$\to$	$M'[x := N']$
$(\beta\multimap_1)$	$(\lambda x.\, M')\, N'$	$\to$	$M'[x := N']$
$(\beta!_0)$	${}^{[\beta!]}\text{let } !x = !M' \text{ in } N'$	$\to$	$N'[x := M']$
$(\beta!_1)$	$\text{let } !x = !M' \text{ in } N'$	$\to$	$N'[x := M']$
$(\star\multimap_0)$	${}^{[\star\multimap_n]}(\mathcal{P}^n\, M')\, N'$	$\to$	$\mathcal{P}^n\,(M'\, N')$
$(\star\multimap_0^<)$	${}^{[\star\multimap_n]}(\mathcal{P}^{n-m}\,\mathcal{P}^m\, M')\, N'$	$\to$	$\mathcal{P}^{n-m}\,(\mathcal{P}^m\, M'\, N')$, $m, n > 0$
$(\star\multimap_1)$	$(\mathcal{P}^n\, M')\, N'$	$\to$	$\mathcal{P}^n\,(M'\, N')$
$(\star!_0)$	${}^{[\star!_n]}\text{let } !x = (\mathcal{P}^n\, M') \text{ in } N'$	$\to$	$\mathcal{P}^n\,(\text{let } !x = M' \text{ in } N')$
$(\star!_0^<)$	${}^{[\star!_n]}\text{let } !x = (\mathcal{P}^{n-m}\,\mathcal{P}^m\, M') \text{ in } N'$	$\to$	$\mathcal{P}^{n-m}\,(\text{let } !x = \mathcal{P}^m\, M' \text{ in } N')$, $m, n > 0$
$(\star!_1)$	$\text{let } !x = (\mathcal{P}^n\, M') \text{ in } N'$	$\to$	$\mathcal{P}^n\,(\text{let } !x = M' \text{ in } N')$

Compatible Closure

1. Let $M' \to N'$. Then:
   (a) $L'\, M' \to L'\, N'$.
   (b) $M'\, L' \to N'\, L'$.
   (c) $\text{let } !x = L' \text{ in } M' \to \text{let } !x = L' \text{ in } N'$.
   (d) $\text{let } !x = M' \text{ in } L' \to \text{let } !x = N' \text{ in } L'$.
   (e) $\lambda x.\, M' \to \lambda x.\, N'$.
   (f) $!M' \to !N'$.
   (g) ${}^{[\star\multimap_n]}\mathcal{P}^n\, L' M' \to {}^{[\star\multimap_n]}\mathcal{P}^n\, L' N'$.
   (h) ${}^{[\star!_n]}\mathcal{P}^n\, L' M' \to {}^{[\star!_n]}\mathcal{P}^n\, L' N'$.
2. Let $\sigma : \mathcal{P}^n\, M' \to N'$, where $n > 0$. Then:
   (a) ${}^{[\star\multimap_n]}(\mathcal{P}^n\, M')\, L' \to {}^{[\star\multimap_{n+\mathbf{d}(\sigma)}]}N'\, L'$.
   (b) ${}^{[\star!_n]}\text{let } !x = \mathcal{P}^n\, M' \text{ in } L' \to {}^{[\star!_{n+\mathbf{d}(\sigma)}]}\text{let } !x = N' \text{ in } L'$.

Figure 5.5: Reduction in the generalized marked linear lambda calculus $\mathsf{Lin}^{\star\prime}$.

Lemma 5.3.3. Let $L', M', N' \in \mathsf{Lin}^{\star\prime}$ and $M' \xrightarrow[\mathsf{Lin}^{\star\prime}]{} N'$. Then:

1. $M'[x := L'] \xrightarrow[\mathsf{Lin}^{\star\prime}]{} N'[x := L']$, and
2. $L'[x := M'] \xrightarrow[\mathsf{Lin}^{\star\prime}]{} \!\!\!\!\!\!\rightarrow L'[x := N']$.

Proof: As usual and trivial. ♦

We define the projection $|\,.\,|$ of $\mathsf{Lin}^{\star\prime}$ terms and sequences into $\mathsf{Lin}^{\star}$ as usual, and the projection lemma is trivial.

Lemma 5.3.4. Let $M' \xrightarrow[\mathsf{Lin}^{\star\prime}]{} N'$; then $|M'| \xrightarrow[\mathsf{Lin}^{\star}]{} |N'|$.

Proof: It is clear from inspection of the various corresponding rules that one can simply erase the markings from $\mathsf{Lin}^{\star\prime}$ to get $\mathsf{Lin}^{\star}$ steps. ♦

Lifting redexes from $\mathsf{Lin}^{\star}$ to $\mathsf{Lin}^{\star\prime}$ from an initial marking is also straightforward.

Lemma 5.3.5. Let $M \xrightarrow[\mathsf{Lin}^{\star}]{} N$, and let $M' \in \mathsf{Lin}^{\star\prime}$ such that $|M'| \equiv M$. Then there exists some $N' \in \mathsf{Lin}^{\star\prime}$ such that $M' \xrightarrow[\mathsf{Lin}^{\star\prime}]{} N'$ and $|N'| \equiv N$.

Proof: As usual, by giving the appropriate indexes to N'. ♦

The notion of residual and development are similar to previous systems, although the separation of redex annotation from the representation of the term itself is more tricky since marks now contain an integer to indicate the rank of the step; we likewise extend the metanotation γ to range over pairs of paths and natural numbers: $\gamma \equiv (\delta, n)$, for example, and $\bar{\gamma}$ to range over sets containing both paths and pairs. We will sometimes abuse this notation by referring to both a pair and the path it contains by the same name: $\gamma \equiv (\gamma, m)$. Let $\partial(M)$ denote the set of all valid paths into an Lin term M. Then a marking $\bar{\gamma}$ for a term M is a subset of $\partial(M) \cup (\partial(M) \times \mathbb{N})$ where:

1. For all paths $\gamma \in \bar{\gamma}$ not in a pair, the subterm $M|_{\gamma}$ is a valid $(\beta{\multimap}, \beta!)$ redex.
2. For all pairs $(\gamma, n) \in \bar{\gamma}$, the subterm $M|_{\gamma}$ is a valid $(\star{\multimap}, \star!)$ redex of rank n.
3. For all pairs $(\gamma, n_1), (\gamma, n_2) \in \bar{\gamma}$, we must have $n_1 = n_2$.

The last condition ensures that a single term is not multiply marked.

As usual, we have designed the notion of a valid marking to correspond with marked terms, and we can make refer to a marked term or to the corresponding unmarked term and valid marking interchangeably.

Lemma 5.3.6. Let $M \in \mathsf{Lin}^{\star}$ and $N' \in \mathsf{Lin}^{\star\prime}$. Then $M \equiv |N'|$ if and only if there exists a valid marking $\bar{\gamma}$ for M such that $(M, \bar{\gamma}) \equiv N'$.

Proof: Clear from the definition of valid markings, whose last condition ensures that no redex is doubly-marked. ♦

This complication settled, we define residuals and developments as usual.

Definition 5.3.7. Let $\bar{\gamma}$ be a valid marking for $M \in \mathsf{Lin}^{\star}$, and let

$$\sigma : M \xrightarrow[\mathsf{Lin}^{\star}]{} \!\!\!\!\!\!\rightarrow N \ .$$

By Lemma 5.3.5 we have some term $N' \in \mathsf{Lin}^{\star\prime}$ such that $|N'| \equiv N$ and $M' \xrightarrow[\mathsf{Lin}^{\star\prime}]{} \rightarrow N'$, and by Lemma 5.3.6 we can separate N' as $N' \equiv (N, \bar{\delta})$. Then the residual of $\bar{\gamma}$ with respect to σ, written $\bar{\gamma}/\sigma$, is defined to be $\bar{\delta}$. ♢

Definition 5.3.8. Let $M' \equiv (M, \bar{\gamma}) \in \mathsf{Lin}^{\star\prime}$, and let $\sigma : M' \xrightarrow[\delta]{} N'$. Then σ is a *development* of M' whenever $\delta \in \bar{\gamma}$. A multistep reduction sequence is a development if each individual step is a development of the residual of the marking to the beginning of that step. A development is *complete* if the final term in the sequence is unmarked. ♢

Lemma 5.3.9. $\mathsf{Lin}^{\star}$ developments and ${\mathsf{Lin}^{\star\prime}}_0$ sequences coincide.

Proof: By Lemma 5.3.6, since ${\mathsf{Lin}^{\star\prime}}_0$ sequences contract only marked redexes. ♦

5.3.3 Finiteness of generalized developments

The proofs of the claims in this section are very similar to the proofs in Sections 2.1.5 and 3.3.2. Constructing a $\mathsf{Lin}^\star$ calculus as usual for weighing marked generalized terms, we can adapt the norm for $\mathsf{Let}^\star$ terms just by changing evaluations of let-bindings to !-eliminators, and similarly for the definition of a decreasing weighting. The various analyses proceed largely as in the sequence of $\mathsf{Let}^\star$ proofs, except for reasoning about the $(\beta{\multimap})$ rule, which operates as the earlier study for the $(\beta{\to})$ steps for weighted call-by-name.

Lemma 5.3.10. Let $M' \xrightarrow[\mathsf{Lin}^{\star\prime}_0]{} N'$, and let M' have a decreasing weighting. Then:

1. $\|M'\| > \|N'\|$.
2. N' has a decreasing weighting. ♦

Lemma 5.3.11. Every $\mathsf{Lin}^{\star\prime}$ term has a decreasing weighting. ♦

These two lemmas give us the expected formalization and finiteness results.

Corollary 5.3.12. $\mathsf{Lin}^{\star\prime}_0$ reduction is strongly normalizing. ♦

Corollary 5.3.13, $\mathrm{FD}(\mathsf{Lin}^\star)$**.** All $\mathsf{Lin}^\star$ developments are finite. ♦

5.3.4 Uniqueness of complete generalized developments

This section is very similar to Section 3.8.3, where we presented the corresponding results for Let; in particular the arguments for $(\star{\multimap}, \star!)$-based rules are just the same as for $(\star C, \star A)$-based rules. We first show $\mathsf{Lin}^{\star\prime}_0$ reduction to be weakly confluent; full confluence follows by Lemma 2.1.5 and Corollary 5.3.12, and then the strengthened finiteness of developments result.

Lemma 5.3.14. $\mathsf{Lin}^{\star\prime}_0$ reduction is weakly confluent.

Proof: We consider the case where one top-level redex contains a second distinct contraction:

1. *Top-level* $(\beta{\multimap}_0)$ *reduction.* This case is straightforward. Let
$$\begin{array}{rcl} M' & \equiv & {}^{[\beta\multimap_0]}(\lambda x.\, N_0')\; N_1' \text{ and} \\ M_1' & \equiv & N_0'[x := N_1']\ . \end{array}$$
By inspection of the rules we must have the second redex within either N_0' or N_1'; in either case the result follows by Lemma 5.3.3.

2. *Top-level* $(\beta!_0)$ *reduction.* Again, the result is immediate from Lemma 5.3.3.

3. *Top-level* $(\star{\multimap}_0)$ *reduction.* Let
$$\begin{array}{rcl} M & \equiv & {}^{[\star\multimap_n]}(\mathcal{P}^n\, N_0')\; N_1' \text{ and} \\ M_1 & \equiv & \mathcal{P}^n\,(N_0'\, N_1')\ . \end{array}$$
The following cases are straightforward if not trivial:

   - $M_2' \equiv {}^{[\star\multimap_n]}(\mathcal{P}^n\, N_2')\; N_1'$ *where* $N_0' \to N_2'$. Then $N' \equiv \mathcal{P}^n\,(N_2'\, N_1')$.
   - $M_2' \equiv {}^{[\star\multimap_n]}(\mathcal{P}^n\, N_0')\; N_2'$ *where* $N_1' \to N_2'$. Then $N' \equiv \mathcal{P}^n\,(N_0'\, N_2')$.
   - $M_2' \equiv {}^{[\star\multimap_n]}(\mathcal{P}_0^{m_0}\,(\text{let }!x = N_3' \text{ in } \mathcal{P}_1^{m_1}\, N_0'))\; N_1'$ *where*
   $$\begin{array}{rcl} m_0 + m_1 & = & n-1 \\ \mathcal{P}^n & \equiv & \mathcal{P}_0^{m_0}\,(\text{let }!x = N_2' \text{ in } \mathcal{P}_1^{m_1}) \ \textit{ and} \\ N_2' & \to & N_3'\ . \end{array}$$
   Then simply $N' \equiv \mathcal{P}_0^{m_0}\,(\text{let }!x = N_3' \text{ in } \mathcal{P}_1^{m_1}\,(N_0'\, N_1'))$.
   - *Where the second redex is a top-level* $(\star{\multimap}_0^<)$ *step with*
   $$\begin{array}{rcl} M & \equiv & {}^{[\star\multimap_n]}(\mathcal{P}_0^{n-m}\, \mathcal{P}_1^m\, N_0')\; N_1' \textit{ and} \\ M_2' & \equiv & \mathcal{P}_0^{n-m\,[\star\multimap_m]}(\mathcal{P}_1^m\, N_0')\; N_1'\ . \end{array}$$
   Then $N \equiv M_1'$.

For the remaining cases the damage function and its effect on compatible closure become relevant. We have that the contraction to M_2' must be a $(\beta!_0, \star!_0)$ step since only these rules are applicable at the top-level to let-bindings, and consider each separately. In both cases we take $m_0 + m_1 = n-1$ and $\mathcal{P}^n \equiv \mathcal{P}_0^{m_0}\,\mathcal{P}_1\,\mathcal{P}_2^{m_1}\,N_2'$; we will take $\mathcal{P}_1$ to be marked respectively $[\beta!]$ and $[\star!]$ in each case and construct the common reduct N'.

- $\mathcal{P}_1 \equiv {}^{[\beta!]}\mathrm{let}\ !x = !L_0'\ \mathrm{in}$. Considering M' without the topmost mark, we have

$$\begin{aligned}\sigma : (\mathcal{P}_0^{m_0}\,({}^{[\beta!]}\mathrm{let}\ x = !L_0'\ \mathrm{in}\ (\mathcal{P}_2^{m_1}\,N_2')\,N_0'))\,N_1' \\ \to \quad (\mathcal{P}_0^{m_0}\,((\mathcal{P}_2^{m_1}\,N_2')[x := L_0']\,N_0'))\,N_1' \ \ ,\end{aligned}$$

  and since $n > m_0$ we have $\mathbf{d}(\sigma, n) = -1$. So the mark on M_2' has rank $n-1$:

$$M_2' \equiv {}^{[\star\multimap_{n-1}]}(\mathcal{P}_0^{m_0}\,((\mathcal{P}_2^{m_1}\,N_2')[x := L_0']\,N_0'))\,N_1' \ \ .$$

  Then moreover since the substitution of L_0' for x will not affect the number of prefixes in $\mathcal{P}_2^{m_1}$, we can take

$$N' \equiv \mathcal{P}_0^{m_0}\,((\mathcal{P}_2^{m_1}\,N_2')[x := L_0']\,(N_0'\,N_1'))$$

  and the case is complete.
- $\mathcal{P}_1 \equiv {}^{[\star!_l]}\mathrm{let}\ !x = \mathcal{R}^l\,N_2\ \mathrm{in}$. As for the preceding case of a $(\beta!_0)$ step, except that the damage function returns some positive integer.

4. *Top-level* $(\star\multimap_0^<, \star!_0, \star!_0^<)$ *reduction.* As in top-level $(\star\multimap_0)$ reduction.

Then since the cases of compatible closure and of redexes in disjoint subterms are trivial, the proof is complete. ♦

Corollary 5.3.15. $\mathsf{Lin}^{\star\prime}_0$ reduction is confluent.

Proof: By Lemma 2.1.5 with Corollary 5.3.12 and Lemma 5.3.14. ♦

Proposition 5.3.16, FD!($\mathsf{Lin}^\star$)**.** All $\mathsf{Lin}^\star$ developments are finite and can be extended to complete developments, and all complete $\mathsf{Lin}^\star$ developments of a given terms and marking end in the same term.

Proof: By Lemma 2.1.4 with FD($\mathsf{Lin}^\star$) and Corollary 5.3.15. ♦

We again omit the fine details of the following proof since they are nearly identical the result for the $\mathsf{Let}^\star$ markings.

Lemma 5.3.17. All $\mathsf{Lin}^\star$-markings are joinable.

Proof: As for $\mathsf{Let}^\star$-markings. ♦

5.4 Confluence

With the last two results of the previous section, confluence is now painless.

Proposition 5.4.1, CR(Lin)**.** Lin is confluent.

Proof: By Lemma 5.3.1 and Lemma 2.1.12 with FD!($\mathsf{Lin}^\star$) and Lemma 5.3.17. ♦

5.5 Standard linear lambda reduction

Figure 5.6 presents our notion of standard reduction for Lin. *Values* are a subset of the terms which are freely copyable, namely, variables, computation suspended under a !, and abstractions. In Lin the set of answers is equivalent to the set of values; in some later systems the two will not be equivalent. Given a notion of reduction $\mapsto$ which is (or which we assert to be) a standard strategy for $\to$, we refer to steps of $\to$ which are not steps of $\mapsto$ as *internal* steps, written $\underset{\mathrm{i}}{\to}$ for single steps and $\underset{\mathrm{i}}{\twoheadrightarrow}$ for multiple steps. We use the same notation for discussing paths γ and the subterms they index as in Let. In fact, this proof of standard reduction is very similar to the earlier result.

We begin with the unique evaluation contexts result for Lin.

Syntactic domains As for Lin, plus the following:

$$\begin{array}{lrcl} \text{Values (and answers)} & V, W & ::= & x \mid \lambda x.\, M \mid\, !M \\ \text{Evaluation Contexts} & E & ::= & [\,] \mid E\, M \mid \text{let } !x = E \text{ in } M \end{array}$$

Standard reduction relation

$$\begin{array}{llcl} (\beta\multimap) & (\lambda x.\, M)\, N & \mapsto & M[x := N] \\ (\beta!) & \text{let } !x = !M \text{ in } N & \mapsto & N[x := M] \end{array}$$

Figure 5.6: Standard reduction $\underset{\mathsf{Lin}}{\longmapsto}$ in the linear lambda calculus.

Lemma 5.5.1. Let $M, V \in \mathsf{Lin}$. Then:

1. There exists no E, Δ, with Δ a top-level $(\beta\multimap, \beta!)$-redex, such that $V \equiv E[\Delta]$.
2. There exists at most one E, Δ, with Δ a top-level $(\beta\multimap, \beta!)$-redex, such that $M \equiv E[\Delta]$.
3. If M is closed then either M is a value or the E, Δ of the previous clause exist and are unique.

Proof: Clause (1) is trivial by inspection of the grammars for E and V. Clause (2) follows by structural induction on M:

- $M \equiv x$. Then E, Δ cannot be formed.
- $M \equiv \lambda x.\, N$ or $!N$. Follows from Clause (1).
- $M \equiv M_0\, M_1$. We consider M_0:
  - $M_0 \equiv x$. Then E, Δ do not exist, as no other evaluation contexts could be formed.
  - $M_0 \equiv \lambda x.\, N$. Then $E \equiv [\,]$ and $\Delta \equiv M$. Note that M_0 could not be any other value due to the type rules.
  - Other M_0. Then by induction on M_0, if we have E_0, Δ such that $M_0 \equiv E_0[\Delta]$, then $M \equiv E[E_0[\Delta]]$; otherwise again none can exist.
- $M \equiv \text{let } !y = M_0 \text{ in } M_1$. We consider M_0:
  - $M_0 \equiv x$. Then again E, Δ cannot be formed.
  - $M_0 \equiv !N$. Then $E \equiv [\,]$ and $\Delta \equiv M$. Note that M_0 could not be any other value due to the type rules.
  - Other M_0. Then by induction on M_0 as for top-level application.

Clause (3) is similar to the previous clause: we can note that the only cases where E, Δ do not exist are when M is a value or when a free variable appears; for M closed the latter case will not arise. ♦

Since we do not have standard $(!\multimap, !!)$ rules, the standard $\mathsf{Lin}^\star$ rules are just the same as for Lin; we thus do not have only a prestandard notion as for $\mathsf{Need}^\star$. We now show the usual properties **P0** through **P4** for $\mathsf{Lin}^\star$.

Lemma 5.5.2. *Generalized linear lambda reduction satisfies **P0**:* All $\mathsf{Lin}^\star$ developments are finite and can be extended to a complete development, all complete $\mathsf{Lin}^\star$-developments of the same term and marking end in the same term, and $\mathsf{Lin}^\star$ has joinable markings with the joining operator $\Cup$ of Section 3.8.4.

Proof: The first three requirements are just FD!($\mathsf{Lin}^\star$), and the arguments for the joining operator are just as for $\mathsf{Let}^\star$. ♦

Lemma 5.5.3. *Generalized linear lambda reduction satisfies **P1**:* Let $\sigma : M \underset{\mathrm{i}}{\rightarrow} N$. Then M is a value if and only if N is a value.

Proof: If M is a value then the result is trivial by inspection of the reduction rules. If N is a value, then assume that M is a non-value; we must have $M \xrightarrow[(\beta\multimap,\beta!)]{\epsilon} N$, but that step is standard, not internal, and so M must be a value. ♦

Again we have *speculative* contexts, given by the following grammar, as the complement of evaluation contexts:

$$H ::= \lambda x.\, C \mid\, !C \mid M\ C \mid H\ M \mid \text{let } !x = H \text{ in } M \ \ .$$

Lemma 5.5.4. Speculative and evaluation contexts partition the set of all contexts: every context C is either an evaluation or a speculative context, but for no evaluation context E and speculative context H do we have $E \equiv H$.

Proof: Clear from the definition. ♦

Lemma 5.5.5. *Generalized linear lambda reduction satisfies **P2**:* Let $\sigma_0 : M \xrightarrow[\text{i}]{\gamma_0} N_0$ and $\sigma_1 : M \xrightarrow[\text{i}]{\gamma_1} N_1$. Then every element of γ_1/σ_0 is internal.

Proof: If γ_1 indexes a $(\star\!\multimap, \star!)$ redex, then the result is trivial; all of its residuals would again be $(\star\!\multimap, \star!)$ redexes, and hence internal. So we can assume henceforward that γ_1 is a $(\beta\!\multimap, \beta!)$ step, and since it is internal it must be within a speculative context H; we will show that the contexts formed by all members of γ_1/σ_0 are all speculative. So we have

$$\begin{array}{rcl} \sigma_0 : M \equiv C_0[\Delta_0] & \xrightarrow[\text{i}]{\gamma_0} & C_0[\Delta_0'] \equiv N_0 \\ \sigma_1 : M \equiv H_1[\Delta_1] & \xrightarrow[\text{i}]{\gamma_1} & H_1[\Delta_1'] \equiv N_1 \end{array}$$

and we proceed by structural induction on M, examining C_0:

- $C_0 \equiv [\,]$. Then γ_0 is a $(\star\!\multimap, \star!)$ step, and we consider each of the two redexes separately. Without loss of generality we need consider only steps of rank 1, since steps of higher rank can be expressed just as several rank 1 steps.
  - γ_0 *is a* $(\star\!\multimap)$ *step.* So we have

    $$M \equiv (\text{let } !x = M_0 \text{ in } M_1)\, M_2 \xrightarrow[(!\multimap)]{\epsilon} \text{let } !x = M_0 \text{ in } (M_1\ M_2) \equiv N_0$$

    and we consider H_1. We know from the possible forms of a speculative context that $H_1 \not\equiv [\,]$, $[\,]\ M_2$ nor $(\text{let } !x = [\,] \text{ in } M_1)\ M_2$, and we consider the remaining valid combinations.
    - $H_1 \equiv (\text{let } !x = M_0 \text{ in } M_1)\, C$. So $\gamma_1 = \underline{@2}\gamma_1'$ and $\gamma_1/\sigma_0 = \{\underline{\ell 2}\,\underline{@2}\gamma_1'\}$, which clearly gives rise to a speculative context.
    - $H_1 \equiv (\text{let } !x = M_0 \text{ in } C)\ M_2$. As in the previous case: we have $\gamma_1 \equiv = \underline{@1}\,\underline{\ell 2}\gamma_1'$ and $\gamma_1/\sigma_0 = \{\underline{\ell 2}\,\underline{@1}\gamma_1'\}$, which again gives rise to a speculative context.
    - $H_1 \equiv (\text{let } !x = H \text{ in } M_1)\ M_2$. So $\gamma_1 \equiv \underline{@1}\,\underline{\ell 1}\gamma_1'$, where $\mathfrak{C}^{M_0}_{\gamma_1'}$ is speculative. Then $\gamma_1/\sigma_0 = \{\underline{\ell 1}\gamma_1'\}$, and since $\mathfrak{C}^{M_0}_{\gamma_1'}$ is speculative, so is

      $$\mathfrak{C}^{\text{let } !x = M_0 \text{ in } (M_1\ M_2)}_{\underline{\ell 1}\gamma_0'} \equiv \text{let } !x = (\mathfrak{C}^{M_0}_{\gamma_1'}) \text{ in } (M_1\ M_2) \ \ .$$

  - γ_0 *is a* $(\star!)$ *step.* As in the analysis for the $(\star\!\multimap)$ step. All but one case simply moves a redex in a clearly internal position to another clearly internal position; otherwise we rely on the fact that when a subcontext of a speculative context is in evaluation position, it must again be speculative.
- $C_0 \equiv \lambda x.\, C$ *or* $C_0 \equiv\, !C$. In this case the result is trivial; *all* contractions under an abstraction or suspension — as well as their residuals with respect to other steps under the same abstraction or suspension — are internal.
- $C_0 \equiv C_a\ M_1$. So we have $M \equiv M_0\ M_1$ with $M_0 \equiv C_a[\Delta_0]$. Again we notice that $H_1 \not\equiv [\,]$, and consider the side of the application where γ_1 lies:
  - $H_1 \equiv H_a\ M_1$. By the structural induction hypothesis on M_0, C_a and H_a.
  - $H_1 \equiv (C_a[\Delta_0])\ C$. Then the set γ_1/σ_0 is a singleton, and its contents form the context $(C_a[\Delta_0'])\ C$, which is again speculative.

- $C_0 \equiv M_0\, C_a$. This case is simple: once again $H_1 \not\equiv [\,]$. Then if $H_1 \equiv H\, C_a[\Delta_0]$, we have a single residual generating the context $H_1 \equiv H\, C_a[\Delta_0']$ which is clearly speculative. If otherwise $H_0 \equiv M_0\, H_a$, then by compatible closure all residuals generate contexts of the form $M_0\, H$, which are again speculative.

- $M \equiv \text{let } !x = M_0 \text{ in } M_1$ *and* $C \not\equiv [\,]$. As for the cases where M is an application and γ_0 is not top-level: since γ_1 also cannot be top-level lest it be standard, the analysis is precisely the same as above.

♦

Lemma 5.5.6. *Generalized linear lambda reduction satisfies **P3**:* Let $\sigma : M_0 \xrightarrow[\mathrm{i}]{\gamma} N_0$ and let $N_0 \mapsto N_1$. Then M_0 has a standard redex: there exists some M_1 such that $M_0 \mapsto M_1$.

Proof: We take

$$M_0 \equiv C[\Delta_0] \underset{\mathrm{i}}{\rightarrow} C[\Delta_0'] \equiv N_0 \ \ ,$$

and

$$N_0 \equiv E_1[\Delta_1] \mapsto E_1[\Delta_1'] \equiv N_1$$

and proceed by induction on C to find the evaluation context E and top-level standard redex Δ for which $M \equiv E[\Delta]$.

- $C \equiv [\,]$. Then $M_0 \xrightarrow[(\star\multimap,\star!)]{\epsilon} N_0$, since top-level $(\beta\multimap, \beta!)$ steps are not internal. We consider each rule separately, again needing to consider only rank 1 steps:

  – $M_0 \xrightarrow[(\star\multimap)]{\epsilon} N_0$. Then $M_0 \equiv (\text{let } !x = L_0 \text{ in } L_1) L_2$ and $N_0 \equiv \text{let } !x = L_0 \text{ in } (L_1\, L_2)$. We distinguish two possibilities regarding E_1.

    ∗ If $E_1 \equiv [\,]$, then L_0 must be a value. We take $E \equiv ([\,]\, L_2)$, with $(\text{let } !x = L_0 \text{ in } L_1)$ as the top-level standard $(\beta!)$ redex.

    ∗ $E_1 \equiv \text{let } !x = E_2 \text{ in } L_1\, L_2$ where $E_2[\Delta_1] \equiv L_0$, and so $E_0 \equiv (\text{let } !x = E_2 \text{ in } L_1)\, L_2$ and $\Delta \equiv \Delta_1$.

  – $M \xrightarrow[(\star!)]{\epsilon} N$. This case is similar to the previous case; the larger point is that although these commuting conversions do move terms around, the manipulations do not introduce standard redexes.

- $C \equiv L\, C_0$ or $\text{let } !x = L \text{ in } C_0$. Then trivially the standard redex of M and its position are exactly the same as in N.

- $C \equiv C_0\, L$ or $\text{let } !x = C_0 \text{ in } L$. We consider E_1:

  – $E_1 \equiv [\,]$. Then again $E \equiv [\,]$.

  – Otherwise, we rely on the induction hypothesis over the subterm with both redexes. Note that within each $C_0[\Delta_0]$ we do have as the hypothesis requires that the internal contraction must again be internal; if it were standard in the subterm then it would be standard again in M due to the position of the subterms in this particular case.

No other construction of C is possible as otherwise E_1 could not be an evaluation context. ♦

Lemma 5.5.7. *Generalized linear lambda reduction satisfies **P4**:* Let $\sigma : M_0 \xrightarrow[\mathrm{i}]{\gamma} N_0$ and $M_0 \overset{\delta}{\mapsto} M_1$. Then δ/σ contains a single element δ_1 which is the standard redex of N_0: $N_0 \overset{\delta_1}{\longmapsto} N_1$.

Proof: We take

$$M_0 \equiv C[\Delta_0] \underset{\mathrm{i}}{\rightarrow} C[\Delta_0'] \equiv N_0$$

and

$$M_0 \equiv E_0[\Delta_1] \underset{\mathrm{i}}{\rightarrow} E_0[\Delta_1'] \equiv M_1 \ \ .$$

We proceed by induction on E_0, constructing in each case the evaluation context E and top-level standard redex Δ such that $N_0 \equiv E[\Delta]$, and showing this contraction to be the residual of the redex at δ.

- $E_0 \equiv [\,]$. Then the result is clear, since Δ_1 must properly contain Δ_0. This fact is easily verified: if M_0 is a top-level $(\beta\text{-}\!\circ)$ redex, then either Δ_0 is a different $(\beta\text{-}\!\circ, \beta!)$, which would clearly not move the top-level redex, or it is a $(\star\text{-}\!\circ, \star!)$ step, which must be properly contained in either the body of the function or within its argument since a $(\beta\text{-}\!\circ)$ redex cannot also be a top-level $(\star\text{-}\!\circ, \star!)$ step. Hence it will also not move the standard step from top-level. The reasoning for a $(\beta!)$ step is similar; the term could not also be a top-level $(\star!)$ step which would move the outer binding inward. So trivially, $E \equiv [\,]$ and $\Delta \equiv N_0$, whose standard contraction is the residual of the redex at δ.

- $E_0 \equiv E_a\ L$. Three classes for C are possible:

  - $C \equiv [\,]$. In this case we must have a $(\star\text{-}\!\circ)$ step for the internal reduction. Then
  $$M_0 \equiv \Delta_0 \equiv (\text{let } !x = L_1 \text{ in } L_2)\ L$$
  and $N_0 \equiv \text{let } !x = L_1 \text{ in } L_2\ L$. The result follows in one of two ways depending on E_a:

    * If $E_a \equiv [\,]$ then have that L_1 is !-prefixed, and the standard redex of N_0 is top-level: overlining the standard redex in each case, we have
    $$\begin{array}{rcl} M_0 & \equiv & \overline{(\text{let } !x = !L_0 \text{ in } L_2)\ L} \\ N_0 & \equiv & \overline{\text{let } !x = !L_0 \text{ in } (L_2\ L)}\ . \end{array}$$
    So clearly we have $E \equiv [\,]$ and $\Delta \equiv N_0$; by the definition of the $(\star\text{-}\!\circ_1)$ rule (Figure 5.5, p. 91) we also have this step as the appropriate residual.
    * Otherwise we have $E_a \equiv \text{let } !x = E_b \text{ in } L_2$ and $L_1 \equiv E_b[\Delta_1]$, and again overlining the standard redex in each case the result is again clear from compatible closure over Lin' reduction rules:
    $$\begin{array}{rcl} M_0 & \equiv & (\text{let } !x = E_b[\overline{\Delta_1}] \text{ in } L_2)\ L \\ N_0 & \equiv & \text{let } !x = E_b[\overline{\Delta_1}] \text{ in } (L_2\ L)\ . \end{array}$$

  - $C \equiv C_a\ L$. Then the result follows from the induction hypothesis.
  - $C \equiv E_a[\Delta_1]\ C_a$. Then we have $\Delta_1 \equiv \Delta_0$ and $E_1 \equiv E_a\ C_a[\Delta']$; the residual property is again clear from compatible closure.

- $E_0 \equiv \text{let } !x = E_a \text{ in } L$. As for application above.

♦

Lemma 5.5.8. Let $M \xrightarrow[\mathsf{Lin}^\star]{} \!\!\!\!\!\rightarrow A$; then $M \underset{\mathsf{Lin}^\star}{\mapsto}\!\!\!\!\rightarrow A_0$ and $V_0 \xrightarrow[\mathsf{Lin}^\star]{}\!\!\!\!\!\rightarrow A$.

Proof: By Lemma 2.1.20 with Lemma 5.5.2, Lemma 5.5.3, Lemma 5.5.5, Lemma 5.5.6 and Lemma 5.5.7. ♦

Proposition 5.5.9, $\mathrm{Std}(\mathsf{Lin})$**.** Reduction by $\underset{\mathsf{Lin}}{\longmapsto}$ is a standard strategy for $\xrightarrow[\mathsf{Lin}]{}$ reduction to values.

Proof: By Lemma 5.5.1 we have that $\mapsto$ is indeed deterministic and undefined on values. Completeness of the strategy follows from Lemma 5.5.8 with Lemma 5.3.1. ♦

5.6 Other formulations of linear logic and lambda calculi

The linear lambda calculus which we have presented in this chapter was first put forward in our work with Odersky, Turner and Wadler [51, 52], and is a minor refinement of a syntax of Wadler [77, 78]. The main motivation of all of these systems is Girard's Logic of Unity [27]. The Logic of Unity is a considerably larger system than any of those derivatives: it includes all of the connectives of linear logic [26] in a sequence calculus, with two multisets of assumptions on each side of the turnstile, and a system of polarities to regulate migration of assumptions between zones. Nonetheless, barring a few small tweaks which simplify our reduction rule ($\beta!$), our typing system corresponds to a fragment of that logic. A similar calculus has been devised by Plotkin and Barber [7]. As with the call-by-let and call-by-need systems, we believe that the proof techniques involving generalized reduction are new.

A number of different lambda calculi based on linear logic have been proposed, including work by Lafont [43], Holmström [33], Abramsky [2], Mackie [46], Lincoln and Mitchell [45], Troelstra [73], Benton, Bierman, de Paiva and Hyland [13, 14, 15, 16], della Rocca and Roversi [65] and earlier work of Wadler [75, 76]. Like the Logic of Unity, but unlike most other linear lambda calculi, we use two forms of assumption, which enables the subject reduction property to be established in a simple way. The other systems listed in the introduction either lack this property altogether, or satisfy only a restricted version, or else possess full subject reduction but have a more complex syntax for ! introduction. Most other systems treat Weakening and Contraction as logical rules with associated term forms. Our system treats Weakening and Contraction as structural rules, without the clutter of term forms. The result is more compact and arguably more suitable as the basis of a programming language. Some elaboration of these points can be found in Wadler's work [77, 78]. Every term of our language is also a term of the language in the first of those papers [77], from which it is easy to see how to give a semantics to this language in a categorical model in the style of Seely [71] as amended by Bierman [15, 16].

Chapter 6

The affine lambda calculus

AFFINE logic relaxes the Weakening rule of linear logic. Although it remains true that only modal assumptions may be duplicated, linear (or rather, affine) assumptions may be discarded; then with Dereliction any assumption may be discarded. The affine lambda calculus reflects this allowance in its typing system. Figure 6.1 presents the details of the resulting affine lambda calculus Aff. The types, terms and contexts are the same as for the linear lambda calculus Lin. The type rules are also identical to those for Lin, with the exception that the rule Weakening of Lin, which allows weakening only on intuitionistic assumptions (to the left of the semicolon), is replaced by a rule which allows weakening on linear assumptions (to the right of the semicolon). This new rule is strictly stronger than the previous rule, which can be derived by combining the new Weakening rule with Dereliction. Reduction is defined by the rules $(\beta\multimap, \beta!, !\multimap, !!)$ together with a new rule (!W). Again, this rule has a logical basis, arising when a !-E rule meets a Weakening rule and a Dereliction rule, cancelling both. The new reduction rule for weakening cannot be valid in Lin, as it does not satisfy subject reduction. For instance,

$$-;\ y : !A, z : B \vdash (\text{let } !x = y \text{ in } z) : B$$

is a valid judgment in both Lin and Aff, but for the result of a $(!\mathrm{W})$ step we have

$$-;\ y : !A, z : B \vdash z : B \ \ ,$$

which is still valid in Aff but not in Lin.

We begin as usual with basic substitution and subject reduction results in the first two sections. In Section 6.3 we discuss developments; once again we pursue only the more general $\star$-notion of marked reduction. We apply the developments theory in each of the subsequent two sections. First, in Section 6.4 we derive a confluence result for Aff. This result follows in the usual manner from the strengthened finiteness of developments result. In Section 6.5 we consider a notion of standard evaluation for affine lambda. The operational impact of the switch to an affine calculus is largely the same as the switch between Let and Need, as in both cases we see the arrival of what amounts to a garbage collection rule. However in Aff, some interesting questions arise over the role of the ! prefix in evaluation.

6.1 Substitutions

Aff satisfies the usual substitution properties.

Lemma 6.1.1. Let $M, N \in \mathsf{Aff}$.

1. Let $\Gamma_0;\ \Delta_0 \vdash_{\mathsf{Aff}} N : B$ and $\Gamma_1;\ \Delta_1, x : B \vdash_{\mathsf{Aff}} M : A$. Then $\Gamma_0, \Gamma_1;\ \Delta_0, \Delta_1 \vdash_{\mathsf{Aff}} M[x := N] : A$.

2. Let $\Gamma_0;\ - \vdash_{\mathsf{Aff}} N : B$ and $\Gamma_1, x : B;\ \Delta \vdash_{\mathsf{Aff}} M : A$. Then $\Gamma_0, \Gamma_1;\ \Delta \vdash_{\mathsf{Aff}} M[x := N] : A$.

Proof: By the usual induction on typing trees. ♦

Lemma 6.1.2. Let $L, M, N \in \mathsf{Aff}$, where $M \xrightarrow[\mathsf{Aff}]{} N$. Then:

1. $L[x := M] \twoheadrightarrow_{\mathsf{Aff}} L[x := N]$.

2. $M[x := L] \xrightarrow[\mathsf{Aff}]{} N[x := L]$.

Proof: The first clause is by induction on L; the second is just compatible closure. ♦

Syntactic domains As for Lin.

Typing judgments As for Lin, but changing Weakening as follows:

$$\frac{\Phi;\ \Gamma \vdash M : B}{\Phi;\ \Gamma,\, x : A \vdash M : B}\ \text{Weakening}$$

Reduction relation As for Lin, plus the following:

$$(!\mathrm{W}) \qquad \text{let } !x = M \text{ in } N \quad \to \quad N, \qquad \text{if } x \text{ is not free in } N$$

Figure 6.1: The affine lambda calculus Aff.

Syntactic domains As in $\mathsf{Lin}^\star$.
Typing Judgements As in Aff.
Reduction Rules As in $\mathsf{Lin}^\star$, plus:

$$(!W) \qquad \text{let } !x = M \text{ in } N \quad \to \quad N\ , \qquad x \notin \mathrm{fv}(N)$$

Damage function As in $\mathsf{Lin}^\star$, plus:

$$\mathbf{d}(\text{let } !x = M \text{ in } N \to N, n) \quad = \quad -1, \qquad x \notin \mathrm{fv}(N)$$

Figure 6.2: The generalized affine lambda calculus, $\mathsf{Aff}^\star$.

6.2 Subject reduction

As before, the logical origin of the rules ensures the subject reduction property.

Proposition 6.2.1. Aff satisfies subject reduction.

Proof: As usual, with Lemma 6.1.1 for the β rules. ♦

6.3 Generalized reductions and developments

We extend $\mathsf{Lin}^\star$ to the corresponding generalization of Aff in the same manner as the extension of $\mathsf{Let}^\star$ to $\mathsf{Need}^\star$; Figure 6.2 presents the details. Reduction in $\mathsf{Aff}^\star$ clearly corresponds to reduction in Aff.

Proposition 6.3.1. Let $M, N \in \mathsf{Aff}$. Then $M \underset{\mathsf{Aff}}{\twoheadrightarrow} N$ if and only if $M \underset{\mathsf{Aff}^\star}{\twoheadrightarrow} N$.

Proof: In the forward direction, we simple replace each $(!\multimap, !!)$ step with a $(\star\multimap, \star!)$ step of rank 1; in the backward direction, we replace a $(\star\multimap, \star!)$ of rank n with n consecutive $(!\multimap, !!)$ steps. ♦

That proposition allows us to transfer many properties back and forth from Aff to $\mathsf{Aff}^\star$ with little effort. For example, since Aff and $\mathsf{Aff}^\star$ share a term language and typing rules, subject reduction is immediate:

Corollary 6.3.2. Reduction in $\mathsf{Aff}^\star$ staisfies the subject reduction property: let $\Gamma \vdash_{\mathsf{Aff}^\star} M : A$ and $M \xrightarrow[\mathsf{Aff}^\star]{} N$. Then $\Gamma \vdash_{\mathsf{Aff}^\star} N : A$.

Proof: From $\mathrm{SR}(\mathsf{Aff})$ by the above. ♦

For the marked variant $\mathsf{Aff}^{\star\prime}$ of $\mathsf{Aff}^\star$ we use the same syntax as in $\mathsf{Lin}^{\star\prime}$, and extend the reduction relation as shown in Figure 6.3, again paralleling the derivation of $\mathsf{Need}^{\star\prime}$ from $\mathsf{Need}^\star$. As usual, we take $\mathsf{Aff}^{\star\prime}_0$ reduction to mean any step or sequence by the $(\beta\multimap_0, \beta!_0, \star\multimap_0, \star\multimap_0^<, \star!_0, \star!_0^<)$ rules, and $\mathsf{Aff}^{\star\prime}_1$ reduction to refer to reduction by any other rules.

Lemma 6.3.3. The reduction relations $\xrightarrow[\mathsf{Lin}^{\star\prime}_0]{}$ and $\xrightarrow[\mathsf{Aff}^{\star\prime}_0]{}$ are equal.

Proof: By inspection, as they have the same syntax and are based on identical rules. ♦

Syntactic domains As in $\mathsf{Lin}^{\star\prime}$.
Degenerate Equivalences As in $\mathsf{Lin}^{\star\prime}$.
Reduction rules As in $\mathsf{Lin}^{\star\prime}$, plus

$$\begin{array}{llcll}
(!\mathrm{W}_{1a}) & \mathrm{let}\ !x = N' \ \mathrm{in}\ M' & \to & N' \ , & x \notin \mathrm{fv}(N') \\
(!\mathrm{W}_{1b}) & {}^{[\beta !]}\mathrm{let}\ !x = !N' \ \mathrm{in}\ M' & \to & N' \ , & x \notin \mathrm{fv}(N') \\
(!\mathrm{W}_{1c}) & {}^{[\star !_n]}\mathrm{let}\ !x = \mathcal{P}\, N' \ \mathrm{in}\ M' & \to & N' \ , & x \notin \mathrm{fv}(N')
\end{array}$$

Compatible Closure As in $\mathsf{Lin}^{\star\prime}$.

Figure 6.3: The generalized marked affine lambda calculus $\mathsf{Aff}^{\star\prime}$.

Lemma 6.3.4. Reduction in ${\mathsf{Aff}^{\star}}'_0$ is strongly normalizing and confluent.

Proof: Follows from the corresponding results on ${\mathsf{Lin}^{\star}}'_0$ (Corollary 5.3.12 and Corollary 5.3.15, respectively) by Lemma 6.3.3. ♦

Proposition 6.3.5, $\mathrm{FD!}(\mathsf{Aff}^{\star})$**.** All $\mathsf{Aff}^{\star}$ developments are finite and can be extended to complete developments, and all complete developments from a given term and marking end in the same term.

Proof: From Lemma 6.3.4 by Lemma 2.1.4. ♦

Lemma 6.3.6. All $\mathsf{Aff}^{\star}$-markings are joinable.

Proof: Immediate, since we use the same marking system as with $\mathsf{Lin}^{\star}$. ♦

6.4 Confluence

We show confluence for Aff using $\mathrm{FD!}(\mathsf{Aff}^{\star})$ for most rules, with separate reasoning for $(!\mathrm{W})$ steps.

Lemma 6.4.1. Reduction by any of the $(\beta\multimap, \beta!, !\multimap, !!)$ rules in $\mathsf{Aff}^{\star}$ is confluent.

Proof: By Lemma 2.1.12 with $\mathrm{FD!}(\mathsf{Aff}^{\star})$ and Lemma 6.3.6. ♦

Corollary 6.4.2. Reduction by the the $(\beta\multimap, \beta!, !\multimap, !!)$ rules in Aff is confluent.

Proof: By Proposition 6.3.1 and Lemma 6.4.1. ♦

Lemma 6.4.3. Reduction by $(!W)$ steps is confluent.

Proof: Trivial: as for (G) steps in Need, we can show by inspection that if

$$\begin{array}{ll} M & \xrightarrow[(!\mathrm{W})]{} M_1 \text{ and} \\ M & \xrightarrow[(!\mathrm{W})]{} M_2 \ , \end{array}$$

then we have some N such that for both i, $M_i \xrightarrow[(!\mathrm{W})]{}{}^{=} N$. The full result follows by induction. ♦

Lemma 6.4.4. Let $M, M_1, M_2 \in \mathsf{Aff}$ such that

$$\begin{array}{ll} M & \xrightarrow[(!\mathrm{W})]{} \!\!\!\!\rightarrow M_1 \text{ and} \\ M & \xrightarrow[(\beta\multimap,\beta!,!\multimap,!!)]{} M_2 \ . \end{array}$$

Then there exists some N such that $M_1 \xrightarrow[(\beta\multimap,\beta!,!\multimap,!!)]{}{}^{=} N$ and $M_2 \xrightarrow[(!\mathrm{W})]{}\!\!\!\!\rightarrow N$. Graphically:

$$\begin{tikzcd}
M \arrow[r, "(\beta\multimap,\beta!,!\multimap,!!)"'] \arrow[d, twoheadrightarrow, "(!\mathrm{W})"] & M_2 \arrow[d, dotted, twoheadrightarrow, "(!\mathrm{W})"] \\
M_1 \arrow[r, dotted, "="', "(\beta\multimap,\beta!,!\multimap,!!)"'] & \exists N
\end{tikzcd}$$

Proof: As in Lemma 4.3.2, by induction on the structure of M, searching for the top-level $(\beta\multimap, \beta!, !\multimap, !!)$ step. We have that $M_1 \equiv N$ exactly when the let-binding mentioned in a top-level $(\beta!, !\multimap, !!)$ step is also one of the let-bindings contracted in a $(!\mathrm{W})$ step. ♦

Proposition 6.4.5, $\mathrm{CR}(\mathsf{Aff})$**.** Reduction in Aff is confluent.

Proof: By Lemma 2.1.14 with Corollary 6.4.2, Lemma 6.4.3 and Lemma 6.4.4. ♦

Syntactic domains As for Aff, plus the following:

Answers	A	$::=$	$\lambda x.\, M \mid\ !A \mid \text{let}\ !x = M \text{ in } A$
Evaluation Contexts	E	$::=$	$[\,] \mid\ !E \mid E\, M \mid \text{let}\ !x = M \text{ in } E \mid \text{let}\ !x = E \text{ in } E[x]$

Standard reduction relation

$(\beta\multimap)$	$(\lambda x.\, M)\, N$	$\mapsto$	$M[x := N]$
$(\beta!_s)$	$\text{let}\ !x = !A \text{ in } E[x]$	$\mapsto$	$(E[x])[x := A]$
$(!\multimap_s)$	$(\text{let}\ !x = M \text{ in } A)\, N$	$\mapsto$	$\text{let}\ !x = M \text{ in } (A\, N)$
$(!!_s)$	$\text{let}\ !x = (\text{let}\ !y = M \text{ in } A) \text{ in } E[x]$	$\mapsto$	$\text{let}\ !y = M \text{ in } (\text{let}\ !x = A \text{ in } E[x])$

Figure 6.4: Standard reduction for affine lambda.

6.5 Standard affine reduction

What constitutes a standard strategy for the affine lambda calculus? In particular, what role should ! play in evaluation? Since ! is often intuitively characterized as corresponding to (among many other notions) the suspension of a computation, at first glance it would seem immediate that we should base our standard strategy on a definition of answer that includes arbitrary suspensions:

$$A ::= \lambda x.\, M \mid\ !M \mid \text{let}\ !x = M \text{ in } A\ \ .$$

Despite the intuitive appeal of this formulation, we would have little connection with standard call-by-need evaluation, at least according to the formulations and translations proposed to date [3, 4, 5, 53, 54], which would not even preserve answerhood. We return to this point in Section 7.2.3, after we study the relevant translations.

An alternate interpretation of ! is possible. Since in the affine system the rôle of ! is simply to prefix terms which are copyable, we can view ! as an update point: intuitively then, standard evaluation of a term $!M$ be thought of as forcing a closure containing M, and updating it with the result. "Forcing" of course refers just to standard evaluation again, so in such a notion of answer, we should have ! prefixed only to other answers:

$$A ::= \lambda x.\, M \mid\ !A \mid \text{let}\ !x = M \text{ in } A\ \ .$$

This definition of answer suggests the standard strategy shown in Figure 6.4, which we will pursue in this section.

One could carry the intuition of !-as-update point further. In the answer

$$!(\text{let}\ !x = M \text{ in } A)\ \ ,$$

M is not necessarily an answer, and may if demanded require arbitrarily many standard steps to become an answer. Should our notion of answer be more stringent, and disallow suspensions of !-eliminators? The question is compelling, but there are sound philosophical and pragmatic arguments for a negative reply.

Bad programs exist; needlessly inefficient programs number among the bad programs. A program such as $!(\text{let}\ !x = M \text{ in } A)$ which suspends a modal eliminator is unquestionably less efficient than one of the form $\text{let}\ !x = M \text{ in } !A$. Still, a programmer is free to write code which gives rise to the inefficient form. More importantly, such forms need not necessarily ever arise; in the translation of call-by-need into Aff which we consider in the next chapter, for example, we do not need such terms. Standard strategies might be able to avoid some inefficiencies, but they are not optimal strategies, and cannot be expected to perform well on deliberately poorly written code: on a philosophical level, we view the notion of copyable non-values as falling into this unfortunate group.

The pragmatic objection to the strictest notion of answers is simple: they destroy the close connection to Need gained by the less severe restriction. The term $!(\text{let}\ !x = M \text{ in } \lambda y.\, y)$, not such an *über*answer, is a good example. Were we simply to move the prefix inward to produce the valid *über*answer $(\text{let}\ !x = M \text{ in } !\lambda y.\, y)$, we would of course be leaving Aff, which has no such rule. Otherwise there are two clear candidates for driving the term to *über*answerhood: either one could always force and cut !-eliminated terms in standard sequences, or else devise a second standard strategy which is to be applied under, as distinct from over, the ! construct. The first option is clearly more correspondent to a strict system such as Let, and would not be appropriate for a demand-driven system such as Need. The second approach may in fact be possible: but in view of what we feel would be the severe complexity of the system, we do not consider it further.

The standard reduction result here is rather complicated, which just as in the standardization proof for Need arises from the complexity of the definition of standard reduction. Fortunately, much of the notions of standard evaluation in Need and Aff are similar, which will matters more familiar. We begin with the usual unique evaluation contexts lemma.

Lemma 6.5.1. Let $M \in \mathsf{Aff}$. Then exactly one of the following is true:

1. M is an answer.
2. There exist some unique evaluation context E and $x \in \mathrm{fv}(M)$ such that $M \equiv E[x]$.
3. There exist some unique evaluation context E and top-level standard Aff redex Δ such that $M \equiv E[\Delta]$.

Proof: By structural induction on M.

- $M \equiv x$. Then Clause 2 is true, with $E \equiv [\,]$.
- $M \equiv\, !M_0$. By the induction hypothesis on M_0: the same case holds for M as for M_0, wrapping the answer (for Clause 1) or evaluation context (for Clauses 2 and 3) in $![\,]$.
- $M \equiv \lambda x.\, M_0$. Trivially, Clause 1 is true.
- $M \equiv M_0\ M_1$. We consider M_0 inductively.
  - *Clause 1 holds for M_0: M_0 is an answer.* By the typing rules we have some Γ such that $\Gamma \vdash M_0 : A_0 \to B_0$, and so we cannot have $M_0 \equiv\, !A$. We will have Clause 3 for M, with $\Delta \equiv M$ and $E \equiv [\,]$: the specific redex will depend on the structure of M_0, which we consider presently:
    - $M_0 \equiv \text{let } !x = N_0 \text{ in } A$. This case is standard $(!\text{–}\!\circ)$ reduction. Moreover, by the induction hypothesis on A, since it and M_0 are both answers no other evaluation contexts can be formed from M_0, and this configuration is in fact unique.
    - $M_0 \equiv \lambda x.\, N_0$. This case is standard $(\beta\text{–}\!\circ)$ reduction. Again, no evaluation contexts can be formed within $\lambda x.\, N_0$, so this configuration is unique.
  - *Clause 2 holds for M_0: $M_0 \equiv E_0[x]$.* Then Clause 2 holds for M as well, with $E \equiv E_0\ M_1$.
  - *Clause 3 holds for M_0: $M_0 \equiv E_0[\Delta]$.* Then Clause 3 holds for M as well, with $E \equiv E_0\ M_1$ as in the previous case.
- $M \equiv \text{let } !x = M_0 \text{ in } M_1$. Here we consider the inductive analysis of M_1.
  - *Clause 1 holds for M_1: M_1 is an answer.* Then trivially M is an answer as well, and by the induction hypothesis on M_1 no evaluation contexts can be formed from the term.
  - *Clause 2 holds for M_1: $M_1 \equiv E_1[y]$ for $y \in \mathrm{fv}(M_1)$ but $x \not\equiv y$.* Then Clause 2 holds for M as well, with $E \equiv \text{let } !x = M_0 \text{ in } E_1$, with uniqueness as usual.
  - *Clause 2 holds for M_1: $M_1 \equiv E_1[x]$.* Now we must consider M_0 inductively as well.
    - *Clause 1 holds for M_0: M_0 is an answer.* By the typing rules we have that for some Γ we have $\Gamma \vdash M_0 :!A_0$, so we cannot have $M_0 \equiv \lambda z.\, N_0$. We have Clause 3 for M, with $\Delta \equiv M$, $E \equiv [\,]$ and uniqueness as before; the specific rule again depends on the form of M_0:
      - $M_0 \equiv \text{let } !x = N_0 \text{ in } A$. Then we have a standard $(\beta!)$ step.
      - $M_0 \equiv !N_0$. Then we have a standard $(!!)$ step.

Syntactic domains As in Aff and $\mathsf{Aff}^\star$.

Standard reduction relation

$$
\begin{array}{llcl}
(\beta\text{-}\!\!\circ) & (\lambda x.\, M)\, N & \mapsto & M[x := N] \\
(\beta !) & \text{let } !x = !A \text{ in } E[x] & \mapsto & (E[x])[x := A] \\
(\star\text{-}\!\!\circ) & (P^n\, A)\, N & \mapsto & P^n\, (A\, N) \\
(\star !) & \text{let } !x = (P^n\, A) \text{ in } E[y] & \mapsto & P^n\, (\text{let } !x = A \text{ in } E[y])
\end{array}
$$

Figure 6.5: Evaluation of $\mathsf{Aff}^\star$ terms.

- * *Clause 2 holds for M_0: $M_0 \equiv E_0[z]$, $z \notin \text{fv}(M_0)$.* Then Clause 2 holds for M as well, with $E \equiv \text{let } !x = E_0 \text{ in } E_1[x]$ and uniqueness as before.
  * *Clause 3 holds for M_0: $M_0 \equiv E_0[\Delta]$.* Then Clause 3 again holds for M as well, with $E \equiv \text{let } !x = E_0 \text{ in } E_1[x]$ and uniqueness as before.
- *Clause 3 holds for M_1: $M_1 \equiv E_1[\Delta]$.* Then Clause 3 and uniqueness hold for M once more, with $E \equiv \text{let } !x = M_0 \text{ in } E_1$.

♦

The notion of a demanded variable is again useful.

Definition. As in Need and $\mathsf{Need}^\star$ we say that M *demands* $x \in \text{fv}(M)$ if there is some evaluation context E such that $M \equiv E[x]$. If no such evaluation context exists, then we say that M *does not demand* x. ◊

We also have a number of substitution results on evaluation contexts and answers.

Lemma 6.5.2. Evaluation contexts are closed under substitution of terms for variables: C is an evaluation context if and only if $C[x := M]$ is an evaluation context (*i.e.* taking $[\,][x := M] \equiv [\,]$).

Proof: Trivial: where the structure of an evaluation context relies on a variable, the variable is bound within the evaluation context as a whole, and so is not affected by the substitution. ♦

Lemma 6.5.3. Answers are closed under substitution of terms for variables: if A is an answer then $A[x := M]$ is also an answer.

Proof: Again trivial since free variables do not arise in the definition of an answer. ♦

Note that again the converse is not true in this case: for example, we have that $(\text{let } !x = M \text{ in } y)$ is not an answer, but that $(\text{let } !x = M \text{ in } y)[y := \lambda z.\, N]$ is. However, we can prove the converse for substitution of all non-answers, and for substitution of values for non-demanded variables.

Lemma 6.5.4. Let $M[x := N]$ be an answer, and at least one of the following two conditions hold:

- M does not demand x.
- N is not an answer.

Then M is an answer as well.

Proof: Clear since the only way we can have $M[x := N]$ an answer but M not an answer is in the example in the above paragraph. ♦

We again use the generalized form of the commuting conversion rule in $\mathsf{Aff}^\star$ to take advantage of the strengthened finiteness of developments result there. Figure 6.5 shows the standard evaluation strategy $\mapsto$ for $\mathsf{Aff}^\star$.

Lemma 6.5.5. For all $M, N \in \mathsf{Aff}$, $M \underset{\mathsf{Aff}}{\mapsto\!\!\!\twoheadrightarrow} N$ if and only if $M \underset{\mathsf{Aff}^\star}{\mapsto\!\!\!\twoheadrightarrow} N$.

Proof: Clear by inspection of the rules. ♦

Generalized standard reductions are preserved by substitution of values for variables:

Lemma 6.5.6. Let $\sigma : M \underset{\mathsf{Aff}^\star}{\longmapsto} N$ and $L \in \mathsf{Aff}$. Then for all x we have

$$M[x := L] \underset{\mathsf{Aff}^\star}{\longmapsto} N[x := L] \ .$$

Moreover if we have γ such that $\sigma : M \overset{\gamma}{\longmapsto} N$, then $M[x := L] \overset{\gamma}{\longmapsto} N[x := L]$ as well.

Proof: For top-level standard contractions we have the result by Lemma 6.5.2 and Lemma 6.5.3; for closure the result is implied by Lemma 6.5.2. ♦

The converse of the above lemma is not true in general: we must have the additional condition that the variable is not demanded before substitution.

Lemma 6.5.7. Let $M[x := L] \underset{\mathsf{Aff}^\star}{\longmapsto} N$, where M does not demand x. Then M has a standard redex.

Proof: For top-level redexes the result is clear from the definition of $\mapsto$; for closure in evaluation contexts we have a straightforward structural induction. ♦

Again, $\underset{\mathsf{Aff}^\star}{\longmapsto}$ gives only part of what we have defined a standard evaluation strategy to be, since the $(\star\!\multimap, \star!)$ rules allow several possible redexes at the same evaluation position. However, we do require that all possible decompositions of a term into an evaluation context and a top-level standard step share the same evaluation context. Again, this relaxation will suffice for our overall goal of showing $\underset{\mathsf{Aff}}{\longmapsto}$ to be (fully) standard.

Lemma 6.5.8. Let $M \in \mathsf{Aff}/\mathsf{Aff}^\star$. Then exactly one of the following is true:

1. M is an answer in $\mathsf{Aff}/\mathsf{Aff}^\star$.
2. M demands some $x \in \mathrm{fv}(M)$.
3. There exists at least one top-level standard $\mathsf{Aff}^\star$ redex Δ, plus some unique evaluation context E such that $M \equiv E[\Delta]$.

Proof: Just as in Lemma 6.5.1 except that many possible ranks for $(\star\!\multimap, \star!)$ redexes may arise. ♦

The next two lemmas are related. Answers A and terms of the form $E[x]$ are important in affine lambda evaluation since recognition of evaluation contexts and standard steps depend on these syntactic constructs. In fact, both structures are preserved by internal redexes.

Lemma 6.5.9. Let M demand some $x \in \mathrm{fv}(M)$, and $M \underset{\mathrm{i}}{\rightarrow} N$ in $\mathsf{Aff}^\star$. Then N demands x as well.

Proof: By induction on the position in M of the internal redex. We take $M \equiv E[x]$ and construct the E_1 such that $N \equiv E_1[x]$.

- *Top-level* $(\beta\multimap)$ *reduction.* This case is not possible; a top-level $(\beta\multimap)$ redex would necessarily be standard.
- *Top-level* $(\beta!)$ *reduction.* We have a number of subcases, reflecting how a top-level $(\beta!)$ step could be internal.
  - $M \equiv \mathrm{let}\ !z = !M_0\ \mathrm{in}\ M_1$*, where* M_0 *is not an answer.* If M_1 demands x, then the reasoning is just as in the corresponding case of the proof of Lemma 4.6.8. Otherwise we have M_1 demanding z and M_0 demanding x, and the result follows since the composition of two evaluation contexts is clearly by their definition again an evaluation context.
  - $M \equiv \mathrm{let}\ !z = !A\ \mathrm{in}\ M_1$*, where* M_1 *does not demand* x*.* The result follows by Lemma 6.5.2 since no answer can demand a variable.
- *Top-level* $(\star\!\multimap, \star!, !\mathrm{W})$ *reduction.* Exactly as in the corresponding cases for Lemma 4.6.8.

We have compatible closure as usual by induction on the size of the evaluation context. ♦

Lemma 6.5.10. *Generalized affine lambda reduction satisfies* ***P0****:*

Lemma 6.5.11. *Generalized affine lambda reduction satisfies* ***P1****:* Let $\sigma : M \underset{\mathrm{i}}{\rightarrow} N$ in $\mathsf{Aff}^{\star}$. Then M is an answer if and only if N is an answer.

Proof: By structural induction on M. We first consider top-level $(\beta!, \star\!\multimap, \star!, !\mathrm{W})$ steps, with top-level internal $(\beta\!\multimap)$ reduction not possible.

- *Top-level* $(\beta!)$ *steps.* Immediate from Lemma 6.5.3 and Lemma 6.5.4.
- *Top-level* $(\star\!\multimap, \star!, !\mathrm{W})$ *steps.* Exactly as in Lemma 4.6.10, the corresponding result for $\mathsf{Need}^{\star}$.

Compatible closure follows trivially by the induction hypothesis, and the proof is complete. ♦

Lemma 6.5.12. *Generalized affine lambda reduction satisfies* ***P2****:* Let γ, δ mark distinct internal $\mathsf{Aff}^{\star}$ redexes in M, $\sigma : M \xrightarrow[(\gamma)]{} N$. Then every member of δ/σ is internal.

Proof: By induction on the structure of M. We assume WLOG that γ and δ are different, and perform an analysis of the relative positions of γ and δ. We begin with the case where one of the two redexes is top-level, and afterward where the two are in distinct subterms.

- γ *is top-level.* We consider each of the three possible markable redexes separately, again disregarding the impossible case of a top-level internal $(\beta\!\multimap)$ step.
  - γ *is a* $(\beta!)$ *step.* So we have $M \equiv (\mathrm{let}\ !x = !M_0\ \mathrm{in}\ M_1)$, and again one of two paths to internalness of the redex.
    - M_1 *does not demand* x. If δ is within M_0, then all residuals will be internal: since M_1 did not demand x, it would not demand the replacement for x. Otherwise if δ is within M_1 we have the result by Lemma 6.5.2.
    - M_0 *is not an answer.* Again by the substitution lemmas.
  - γ *is a* $(\star\!\multimap, \star!)$ *step.* Exactly as in Lemma 4.6.11 for $\mathsf{Need}^{\star}$.
- δ *is top-level.* So again δ cannot be an $(\beta\!\multimap)$ step; then to guarantee that subterms remain in the appropriate forms for an internal $(\beta!, \star\!\multimap, \star!)$ step we rely on Lemma 6.5.9 and Lemma 6.5.11.

For γ, δ in distinct subterms it again follows from Lemma 6.5.9 and Lemma 6.5.11 that where δ had been in a non-evaluation context we will not create an evaluation context around the residual of δ by contracting γ, and so the result does follow. ♦

Lemma 6.5.13. *Generalized affine lambda reduction satisfies* ***P3****:* Let $\sigma : M_0 \underset{\mathrm{i}}{\rightarrow} N_0$ and let $N_0 \mapsto N_1$, both in $\mathsf{Aff}^{\star}$. Then M_0 has a standard redex: there exists some M_1 such that $M_0 \mapsto M_1$.

Proof: So we have

$$\begin{aligned} \sigma : M_0 \equiv C_0[\Delta_0] &\underset{\mathrm{i}}{\rightarrow} C_0[\Delta_0'] \equiv N_0 \\ N_0 \equiv E_1[\Delta_1] &\mapsto E_1[\Delta_1'] \equiv N_1 \quad , \end{aligned}$$

and we seek to construct the E, Δ such that $M_0 \equiv E[\Delta]$ and Δ is a standard step. We proceed by structural induction on M_0, and perform an analysis on C_0 and E_1:

- $C_0 \equiv [\,]$. We construct E based on the specific rule which contracts Δ_0. As usual, we cannot have a $(\beta\!\multimap)$ step as both top-level and internal.
  - *By the* $(\beta!)$ *rule, with* $M_0 \equiv \Delta_0 \equiv (\mathrm{let}\ !x = !M_1\ \mathrm{in}\ M_2)$ *and* M_2 *not demanding* x. We have $N_0 \equiv M_2[x := M_1]$, and the case by Lemma 6.5.7.
  - *By the* $(\star\!\multimap, \star!, !\mathrm{G})$ *rules.* Exactly as in the reasoning for the same cases in Lemma 4.6.12.

- *For other C_0, we begin with the special case where $E_1 \equiv [\,]$.* This result is immediate: if σ is in fact internal, then it is clear that we must have $E \equiv [\,]$ as well, since σ is not itself topmost.

- Otherwise the result is straightforward. If the topmost configuration of C_0 is that of an evaluation context — that is, if C_0 takes one of the forms

$$\begin{aligned} C_0 &\equiv\ !C_1 && (6.1)\\ C_0 &\equiv\ C_1\,L \\ C_0 &\equiv\ \text{let } x = L \text{ in } C_1 && (6.2)\\ C_0 &\equiv\ \text{let } x = C_1 \text{ in } E_0[x] \end{aligned}$$

  where in (6.2) E does not demand x — then we have the result by the induction hypothesis. Finally, when the internal and standard contractions are in different subterms, we have the result immediately by Lemma 6.5.9 and Lemma 6.5.11.

♦

Lemma 6.5.14. *Generalized affine lambda reduction satisfies **P4**:* Let $\sigma : M_0 \underset{\mathrm{i}}{\rightarrow} N_0$ and $M_0 \overset{\delta}{\mapsto} N_1$ in $\mathsf{Aff}^\star$. Then δ/σ contains a single element δ_1 which is a standard redex of N_0: $N_0 \overset{\delta_1}{\longmapsto} N$.

Proof: The analysis is similar to that of the previous proof. We proceed by structural induction on M. The inductive case where both contractions are in the same subterm is trivial, and we detail only the three base cases: where one of the two redexes is top-level, and where each is in a distinct subterm of the top-level expression. We take

$$\begin{aligned} \sigma : M_0 \equiv\ & C_0[\Delta_0] \underset{\mathrm{i}}{\rightarrow} C_0[\Delta_0'] & \equiv N_0 \\ \sigma : M_0 \equiv\ & E_1[\Delta_1] \mapsto E_1[\Delta_1'] & \equiv N_1 \ \ , \end{aligned}$$

and construct in each case the required E and Δ.

- $E_0 \equiv [\,]$. This case is the simplest; the result follows with Lemma 6.5.9 and Lemma 6.5.11 by a consideration of the structure of the terms following from each possible redex.

- $C_0 \equiv [\,]$ *and* $E_0 \not\equiv [\,]$. So we have $\Delta_0 \equiv M_0$ in the form of an evaluation context, and moreover again σ is not an $(\beta\multimap)$ step. We consider the different applicable rules for σ_0 separately; these cases are quite similar to the corresponding steps of the the proof of the previous Lemma 6.5.13, and, except in the $(\beta!)$ case, identical to the result for $\mathsf{Need}^\star$, Lemma 4.6.13.

  - $\sigma : M_0 \equiv (\text{let } !x = !M_1 \text{ in } M_2) \xrightarrow[\mathsf{i}\text{-}(\beta!)]{} M_2[x := M_1]$, *where x is not demanded in M_2.* So clearly the standard step in within M_2, and by Lemma 6.5.6 is the residual again; singularity of the residual is clear, since within M_2 it could be neither duplicated nor discarded.
  - *M_0 is a top=level internal $(\star\multimap, \star!, !\mathrm{W})$ step.* Exactly as in the corresponding cases for Lemma 4.6.13.

- *The two redexes are in different subterms.* Trivially here the standard step — being outside of the effect of the internal step — could not be duplicated or discarded, and so the residual is unique; by Lemma 6.5.9 and Lemma 6.5.11 it is again a standard step.

The inductive step, closure of both redexes within the same evaluation context, is trivial, and the proof is complete. ♦

Lemma 6.5.15. Let $M \xrightarrow[(\beta\multimap,\beta!,\star\multimap,\star!)]{} \!\!\!\!\rightarrow A$; then there exists some answer A_0 such that

$$M \overset{}{\underset{\mathsf{Aff}^\star}{\longmapsto\!\!\!\!\rightarrow}} A_0 \xrightarrow[\mathsf{i}\text{-}(\beta\multimap,\beta!,\star\multimap,\star!)]{} \!\!\!\!\rightarrow A \ \ .$$

Proof: By Lemma 2.1.20 with Lemma 6.5.10, Lemma 6.5.11, Lemma 6.5.12, Lemma 6.5.13 and Lemma 6.5.14. ♦

Lemma 6.5.16. Let $M \xrightarrow[(!\mathrm{W})]{} M_0 \underset{\mathsf{Aff}^\star}{\longmapsto} N$. Then there exists some N_0 such that $M \underset{\mathsf{Aff}^\star}{\longmapsto}^{=} N_0 \xrightarrow[(!\mathrm{W})]{}\!\!\!\!\rightarrow N$.

Proof: As in Lemma 4.6.15, we perform the straightforward analysis of the relative positions of the redexes. We have the $(!\mathrm{W})$ step duplicated or discarded when it is within the substituted value of a $(\beta\multimap, \beta!)$ step, and the prestandard step removed when the $(!\mathrm{W})$ rule removes a binding from a $(\star\multimap, \star!)$ redex of rank one. ♦

We can now proceed with the main standardization results. We first have a "pseudo-standardization" result for the non-unique sequence produced by $\mathsf{Aff}^\star$, and then a full standardization result for Aff.

Proposition 6.5.17. Let $M \xrightarrow[\mathsf{Aff}^\star]{}\!\!\!\!\rightarrow A$; then there exists some A_0 such that

$$\begin{array}{lcl} M & \underset{\mathsf{Aff}^\star}{\longmapsto}\!\!\!\!\rightarrow & A_0 \quad \text{and} \\ A_0 & \xrightarrow[\mathsf{Aff}^\star]{}\!\!\!\!\rightarrow & A \ . \end{array}$$

Proof: As in the proof of Lemma 4.6.16 for $\mathsf{Need}^\star$, by induction on the number of non-standard $\mathsf{Aff}^\star$ steps in σ. If the last such step in σ is a $(!W)$ step, then we reduce the problem to the inductive hypothesis by Lemma 6.5.16 and Lemma 6.5.11; if the last non-standard step is an internal $(\beta\multimap, \beta!, \star\multimap, \star!)$ step, then we have the result from Lemma 6.5.15. ♦

Proposition 6.5.18, $\mathrm{Std}(\mathsf{Aff})$**.** The relation $\underset{\mathsf{Aff}}{\longmapsto}$ is a standard strategy for $\xrightarrow[\mathsf{Aff}]{}$ reduction to answers: let $M \xrightarrow[\mathsf{Aff}]{}\!\!\!\!\rightarrow A$; then there exists some A_0 such that $M \mapsto\!\!\!\!\rightarrow A_0$ and $A_0 \xrightarrow[\mathsf{Aff}]{}\!\!\!\!\rightarrow A$.

Proof: We have that $\underset{\mathsf{Aff}}{\longmapsto}$ selects a unique redex for every (closed) non-answer by Lemma 6.5.1, and that it always finds an answer when one is available by Proposition 6.5.17 with Lemma 6.5.5. ♦

Chapter 7

The Girard translations

SO far, we have set forth several calculi in two families — intuitionistic and linear — with no immediate connection between the two families. In this chapter we establish a connection. In his presentation of linear logic [26], Girard gave two translations from intuitionistic logic into linear logic. In previous work with Odersky, Turner and Wadler [51, 52] we extended these translations to the terms of the calculi in Curry-Howard correspondence to intuitionistic logic, and considered their behavior on reduction sequences. Girard's *standard* translation $\circ$ corresponds to call-by-name reduction, and his *steadfast* translation $*$ to the call-by-value/let semantics. Moreover, taking the target of $*$ to be affine lambda, we can treat call-by-need reduction as well. We extend these results here for standard reduction.

Mackie has also presented an extension of the standard translation to the level of terms [47]. Mackie's work is complementary to the results presented here. Our translations are into a linear lambda calculus corresponding to (dual) intuitionistic linear logic, while Mackie's translation is into proof nets, corresponding to classical (full) linear logic. Mackie showed that the translations on proof nets were sound for beta and eta reduction. In our earlier work we showed that the respective translations were sound and complete for call-by-name and sound for call-by-value reduction. We extend these translation results here for eta steps as well. Previously, we showed that the steadfast translation of call-by-need terms into an affine system, where only contraction is restricted and weakening is freely allowed, is also sound. Troelstra [73] and Schellinx [70] have presented other embeddings of intuitionistic logic into linear logic. Gonthier, Abadi and Levy [29] have a translation based on taking $A \to B$ into $!(A \multimap B)$, but it is significantly more complex than our translation at the term level because it deals with the rather more complex notion of optimal reduction.

An application of these results appears in a companion paper by Mossin, Turner and Wadler [59]. They have devised a type system that can infer information about which variables are used linearly in a call-by-need or call-by-value lambda calculus. Such types are useful for program transformation: the reduction

$$((\lambda x.\, M)\, N) \to M[x := N]$$

does not in general hold for a call-by-value or call-by-need calculus, but it does hold if x is used linearly. It may also be useful for implementing call-by-need: normally a closure needs to be overwritten on evaluation, but this step may be saved if the closure is bound to a variable that is used linearly.

We consider the two translations in the following two sections of this chapter. The latter section is in three parts, corresponding to each of the three system to which we apply the steadfast translation.

7.1 The standard translation

Figure 7.1 describes the standard Girard translation $\circ$. This translation takes types, terms and contexts A, M and Γ of the call-by-name lambda calculus Name to types, terms and contexts A°, M° and Γ° of the linear lambda calculus Lin.

The idea behind this translation is that all assumptions are taken as intuitionistic, and that ! is added to the left of $\vdash$ and $\multimap$, but not to the right. Every function argument is surrounded by !, which can be thought of as suspending evaluation, corresponding to the call-by-name discipline. Prefixing ! to arguments also indicates that without further evaluating the argument, a function may discard or duplicate them freely.

Standard Translation
Of types

$$\begin{array}{rcl} Z^\circ & \equiv & Z \\ (A \to B)^\circ & \equiv & (!A^\circ) \multimap B^\circ \end{array}$$

Of terms

$$\begin{array}{rcll} x^\circ & \equiv & x & \\ (\lambda x.\, M)^\circ & \equiv & \lambda y.\, \text{let } !x = y \text{ in } M^\circ & \qquad y \text{ fresh} \\ (M\ N)^\circ & \equiv & M^\circ\, !N^\circ & \end{array}$$

Of typing contexts

$$(x_1 : A_1, \ldots, x_n : A_n)^\circ \quad \equiv \quad x_1 : A_1^\circ, \ldots, x_n : A_n^\circ$$

Figure 7.1: The standard Girard translation $\circ$.

In particular, corresponding to $(\beta \to)$ we have

$$\begin{array}{cl} & ((\lambda x.\, M)\ N)^\circ \\ \equiv & (\lambda y.\, \text{let } !x = y \text{ in } M^\circ)\ !N^\circ \\ \xrightarrow[(\beta\multimap)]{} & \text{let } !x = !N^\circ \text{ in } M^\circ \\ \xrightarrow[(\beta!)]{} & M^\circ[x := N^\circ] \\ \equiv & (M[x := N])^\circ \ . \end{array}$$

Then with the following lemma, we have soundness of the translation for reduction. We extend the translation to contexts by defining $[\,]^\circ \equiv [\,]$.

Lemma 7.1.1. Let $M \in \mathsf{Name}$ and let C be a Name-context. Then $(C[M])^\circ \equiv C^\circ[M^\circ]$.

Proof: Induction on the structure of C. ♦

To show that the translation is complete, we will use an auxiliary grammar of a subset of Lin terms and a second relation †. We define the grammar as follows:

$$S, T ::= x \mid \lambda y.\, \text{let } !x = y \text{ in } S \mid S\ !T \mid \text{let } !x = !S \text{ in } T \ , \tag{7.1}$$

restricted to terms well-typed according to the usual rules for Lin. Terms S are either the $\circ$-image of a Name term, or else are Lin-reducible from one, as the following lemma makes clear.

Lemma 7.1.2. The set of terms described by Grammar (7.1) contains the image of $\circ$ on Name terms and is closed under Lin-reduction.

Proof: The first claim is clear; since the grammar for S can produce x, the set is closed under substitution for variables, and the second follows. ♦

We define the mapping † by:

$$\begin{array}{rcl} (\lambda y.\, \text{let } !x = y \text{ in } S)^\dagger & \equiv & \lambda x.\, S^\dagger \\ (S\ !T)^\dagger & \equiv & S^\dagger\ T^\dagger \\ (\text{let } !x = !S \text{ in } T)^\dagger & \equiv & T^\dagger[x := S^\dagger] \ . \end{array} \tag{7.2}$$

The type system for Lin guarantees that $y \notin \text{fv}(S)$ in Equation 7.2, and so † is in fact meaningful for such S in that for a term $\lambda y.\, \text{let } !x = y \text{ in } S$ it will not introduce a new free variable from an occurrence of y in S. Again, we extend † to term contexts by defining $[\,]^\dagger \equiv [\,]$. The following lemmas establish the relevant properties of †.

Lemma 7.1.3. The translation † from the subset of Lin ranged over by Grammar (7.1) to Name preserves substitution, is a right-inverse to $\circ$ and is sound with respect to types and reduction:

(i) $(S[x := T])^\dagger \equiv S^\dagger[x := T^\dagger]$.

(ii) Let $M \in \mathsf{Name}$. Then $M \equiv M^{\circ\dagger}$.

(iii) If $\Phi;\ \Gamma \vdash_{\mathsf{Lin}} S : A$ then $\Phi^\dagger, \Gamma^\dagger \vdash_{\mathsf{Name}} S^\dagger : A^\dagger$.

(iv) If $S \xrightarrow[\mathsf{Lin}]{} \!\!\!\!\!\rightarrow T$ then $S^\dagger \xrightarrow[\mathsf{Name}]{} \!\!\!\!\!\rightarrow T^\dagger$.

(v) If $S \mapsto_{\mathsf{Lin}}\!\!\!\!\!\rightarrow T$ then $S^\dagger \mapsto_{\mathsf{Name}}\!\!\!\!\!\rightarrow T^\dagger$.

Proof: The first clause is true because † does not alter variables. The second clause is clear from a simple structural induction and case analysis on Name terms; and third and fourth clauses, on terms S, with Lemma 2.1.2 for compatible closure in the fourth clause.

For Clause (v) we consider the standard reduction-closed set of ∘-images of Name terms, which is in fact the same as the general reduction closure S. Of the two possible top-level standard steps, we have first

$$(\lambda y.\, \text{let } !x = y \text{ in } S)\ !T \mapsto \text{let } !x = !T \text{ in } S \ ,$$

and

$$\begin{array}{rl} & ((\lambda y.\, \text{let } !x = y \text{ in } S)\ !T)^\dagger \\ \equiv & (\lambda x.\, S^\dagger)\, T^\dagger \\ \mapsto & S^\dagger[x := T^\dagger] \\ \equiv & (S[x := T])^\dagger \\ \equiv & (\text{let } !x = !T \text{ in } S)^\dagger \end{array} \tag{7.3}$$

with Equivalence (7.3) by Lemma 7.1.3.(i). For standard $(\beta!)$ steps the result is trivial since $(\text{let } !x = !S \text{ in } T)^\dagger \equiv T^\dagger[x := S^\dagger]$ already. The restriction of Lin evaluation contexts to those which we can form from terms S is given by $E ::= [\,] \mid E\ !M$, which † clearly maps to Name evaluation contexts. ♦

We can now state and prove the main result for the standard translation of Name.

Proposition 7.1.4, Standard translation of Name. The translation ∘ from Name to Lin preserves substitution, types, reductions and standard reductions:

(i) $(M[x := N])^\circ \equiv M^\circ[x := N^\circ]$.

(ii) $\Gamma \vdash_{\mathsf{Name}} M : A$ if and only if $\Gamma^\circ;\ - \vdash_{\mathsf{Lin}} M^\circ : A^\circ$.

(iii) $M \xrightarrow[\mathsf{Name}]{} \!\!\!\!\!\rightarrow N$ if and only if $M^\circ \xrightarrow[\mathsf{Lin}]{} \!\!\!\!\!\rightarrow N^\circ$.

(iv) $M \mapsto_{\mathsf{Name}}\!\!\!\!\!\rightarrow N$ if and only if $M^\circ \mapsto_{\mathsf{Lin}}\!\!\!\!\!\rightarrow N^\circ$.

Proof: We prove (i) by an easy structural induction over terms of Name. The forward direction of Clause (ii) follows by an easy structural induction over type derivations in Name, and the reverse direction from Lemma 7.1.2 and Clause (iii) of Lemma 7.1.3. The proof of (iii) in the forward direction is given above, with compatible closure following from Lemma 2.1.2. The backward direction follows from Lemma 7.1.2 and Clause (iv) of Lemma 7.1.3.

For Clause iv we have $(\lambda x.\, M)\ N \mapsto M[x := N]$, and

$$\begin{array}{rl} & ((\lambda x.\, M)\ N)^\circ \\ \equiv & (\lambda y.\, \text{let } !x = y \text{ in } M^\circ)\ N^\circ \\ \mapsto & \text{let } !x = !N^\circ \text{ in } M^\circ \\ \mapsto & M^\circ[x := N^\circ] \\ \equiv & (M[x := N])^\circ \end{array} \tag{7.4}$$

with Equivalence 7.4 by Proposition 7.1.4.(i). Then for evaluation contexts we take $[\,]^\circ \equiv [\,]$ as usual, plus $(E\ M)^\circ \equiv E^\circ\ !M^\circ$ is again an evaluation context in Lin. So for E a Name evaluation context and M a top-level standard Name redex contracting to N, we have $E[M] \mapsto_{\mathsf{Name}} E[N]$, and by Lemma 7.1.1 and the above we have

$$(E[M])^\circ \equiv E^\circ[M^\circ] \mapsto_{\mathsf{Lin}} E^\circ[N^\circ] \equiv (E[N])^\circ$$

as well. ♦

Translation
Of types

$$\begin{array}{rcl} A^* & \equiv & !A^+ \\ Z^+ & \equiv & Z \\ (A \to B)^+ & \equiv & (A^* \multimap B^*) \end{array}$$

Of terms

$$\begin{array}{rcl} V^* & \equiv & !V^+ \\ (M\ N)^* & \equiv & (\text{let } !z = M^* \text{ in } z)\ N^* \\ (\text{let } x = M \text{ in } N)^* & \equiv & \text{let } !x = M^* \text{ in } N^* \\ x^+ & \equiv & x \\ (\lambda x.\, M)^+ & \equiv & \lambda y.\, \text{let } !x = y \text{ in } M^* \end{array}$$

Of typing contexts

$$(x_1 : A_1, \ldots, x_n : A_n)^* \quad \equiv \quad x_1 : A_1^+, \ldots, x_n : A_n^+$$

Figure 7.2: The steadfast Girard translation $*$.

7.2 The steadfast translation

Figure 7.2 describes the steadfast Girard translation $*$. We use the steadfast translation for call-by-value, call-by-let and call-by-need, which we consider separately below.

7.2.1 Of call-by-value

Translation $*$ takes types, terms and contexts A, M and Γ of Val to types, terms and contexts A^*, M^* and Γ^* of Lin. There is also an auxiliary mapping $+$ from types and values A and V of Val to types and terms A^+ and V^+ of Lin; this mapping omits the outermost '!'. As before, in the translation of applications and abstractions y and z are fresh variables not appearing in M.

The idea behind this translation is that ! is added on both the left and right of $\vdash$ and $\multimap$. Function arguments are no longer surrounded by !, so the argument will be reduced until a ! is encountered before evaluation of the function body proceeds. Under this translation, the only terms beginning with ! are those of the form $V^* \equiv !V^+$, so function arguments are forced to reduce to values.

In particular, corresponding to the call-by-name $(\beta \to)$ rule we have

$$\begin{array}{rl} & ((\lambda x.\, M)\ N)^* \\ \equiv & (\text{let } !z = !(\lambda y.\, \text{let } !x = y \text{ in } M^*) \text{ in } z)\ N^* \\ \xrightarrow[(\beta !)]{} & (\lambda y.\, \text{let } !x = y \text{ in } M^*)\ N^* \\ \xrightarrow[(\beta \multimap)]{} & \text{let } !x = N^* \text{ in } M^* \ , \end{array}$$

and the reduction cannot necessarily proceed further. But if we replace the argument term N by a value V, then corresponding to the call-by-value $(\beta \to_v)$ rule we have

$$\begin{array}{rl} & ((\lambda x.\, M)\ V)^* \\ \xrightarrow[\mathsf{Lin}]{} & \text{let } !x = !V^+ \text{ in } M^* \\ \xrightarrow[(\beta !)]{} & M^*[x := V^+] \\ \equiv & (M[x := V])^* \end{array}$$

which shows that the translation is sound.

Unfortunately, the translation is not complete for reduction in general. Taking P to be any non-value, along with any term M and the identity function I, we clearly have

$$I\ P\ M \not\twoheadrightarrow_{\mathsf{Val}} P\ M \ ,$$

However we do have that the translation of the former reduces to the latter:

$$
\begin{array}{cll}
 & (I\ P\ M)^* & \\
\equiv & (\text{let } !x = ((\text{let } !y = !(\lambda z.\, \text{let } !w = z \text{ in } !w) \text{ in } y)\ P^*) \text{ in } x)\ M^* & \\
\xrightarrow[(\beta !)]{} & (\text{let } !x = ((\lambda z.\, \text{let } !w = z \text{ in } !w)\ P^*) \text{ in } x)\ M^* & \\
\xrightarrow[(\beta \multimap)]{} & (\text{let } !x = (\text{let } !w = P^* \text{ in } !w) \text{ in } x)\ M^* & \\
\xrightarrow[(!!)]{} & (\text{let } !w = P^* \text{ in let } !x = !w \text{ in } x)\ M^* & (7.5) \\
\xrightarrow[(\beta !)]{} & (\text{let } !w = P^* \text{ in } w)\ M^* & \\
\equiv & (P\ M)^*\ . &
\end{array}
$$

Interestingly, the translation is complete for standard reduction. The reason is simple enough: Step (7.5) is internal, not standard, and so would not arise as a counterexample.

For the completeness result we will use the following grammar:

$$
\begin{array}{rl}
S, T & ::= \ !U \mid U\ S \mid (\text{let } !x = S \text{ in } x)\ T \mid \text{let } !x = S \text{ in } T \\
U & ::= \ x \mid \lambda y.\, \text{let } !x = y \text{ in } S
\end{array}
$$

This grammar is clearly equivalent to the set of standard Lin-reduction closed $*$-images of Val terms.

Lemma 7.2.1. Let $M \in \mathsf{Val}$ and $N \in \mathsf{Lin}$ such that $M^* \underset{\mathsf{Lin}}{\mapsto\!\!\!\twoheadrightarrow} N$. Then N matches some term S of the grammar above.

Proof: As usual, by inspection of the grammar productions and reduction rules. ♦

We relate terms of this grammar to Val terms by the following variant of the usual inverse steadfast translation from Lin into Let:

$$
\begin{array}{rcl}
(!U)^\ddagger & \equiv & U^\# \\
((\text{let } !x = S \text{ in } x)\ T)^\ddagger & \equiv & S^\ddagger\, T^\ddagger \\
(U\ S)^\ddagger & \equiv & U^\#\, S^\ddagger \\
(\text{let } !x = S \text{ in } T)^\ddagger & \equiv & (\lambda x.\, T^\ddagger)\ S^\ddagger \\
x^\# & \equiv & x \\
(\lambda y.\text{let } !x = y \text{ in } S)^\# & \equiv & \lambda x. S^\ddagger
\end{array}
$$

As usual, $\ddagger$ can be exchanged with substitution of values, and is a right-inverse to $*$ on Val terms:

Lemma 7.2.2. Let $S, U \in \mathsf{Lin}$ according to the grammar above. Then $(S[x := U])^\ddagger \equiv S^\ddagger[x := U^\#]$.

Proof: Trivial. ♦

Lemma 7.2.3. Let $M \in \mathsf{Val}$. Then $(M^*)^\ddagger \equiv M$.

Proof: As in Lemma 7.2.7; the definition of $\ddagger$ there is coincident on $*$-images to the one here. ♦

Completeness of $*$ for standard reduction sequences will then follow from the soundness of $\ddagger$ on the same:

Lemma 7.2.4. Let $S, T \in \mathsf{Lin}$ according to the above grammar.

1. If $\Phi; \Gamma \vdash_{\mathsf{Lin}} S : A$ then $\Phi^\ddagger, \Gamma^\ddagger \vdash_{\mathsf{Val}} S^\ddagger : A^\ddagger$.
2. If $S \underset{\mathsf{Lin}}{\mapsto\!\!\!\twoheadrightarrow} T$ then $S^\ddagger \underset{\mathsf{Val}}{\mapsto\!\!\!\twoheadrightarrow} T^\ddagger$.

Proof: The typing result follows by the straightforward structural induction. For evaluation, we consider first single top-level steps, and then compatible closure in evaluation contexts; multistep standard reduction follows by the usual induction.

If $S \equiv (\lambda y.\, \text{let } !x = y \text{ in } S)\ T \underset{\mathsf{Lin}}{\mapsto} \text{let } !x = T \text{ in } S$, then translating we have simply

$$
\begin{array}{cl}
 & ((\lambda y.\, \text{let } !x = y \text{ in } S)\ T)^\ddagger \\
\equiv & (\lambda x.\, S^\ddagger)\, T^\ddagger \\
\equiv & (\text{let } !x = T \text{ in } S)^\ddagger
\end{array}
$$

If otherwise we have $S \equiv \text{let } !x = !U \text{ in } S \underset{\mathsf{Lin}}{\longmapsto} S[x := U]$ then we have

$$\begin{array}{ll} & (\text{let } !x = !U \text{ in } S)^{\ddagger} \\ \equiv & (\lambda x.\, S^{\ddagger})\, U^{\#} \\ \underset{\mathsf{Val}}{\longmapsto} & S^{\ddagger}[x := U^{\#}] \\ \equiv & (S[x := U])^{\ddagger} \end{array} \qquad (7.6)$$

with Equivalence (7.6) by Lemma 7.2.2.

Compatible closure in evaluation contexts is straightforward: the subset of Lin-evaluation contexts which one can form from terms S clearly match the following grammar:

$$E ::= [\,] \mid (\text{let } !x = E \text{ in } x)\; S \mid \text{let } !x = E \text{ in } S \ .$$

Terms of the form $E[\Delta]$ clearly translate again to top-level redexes in Val evaluation contexts. ♦

Proposition 7.2.5, Steadfast translation of Val. The translation $*$ from Val to Lin preserves substitution of values, preserves types and preserves reductions.

(i) $(M[x := V])^* \equiv M^*[x := V^+]$.

(ii) $\Gamma \vdash_{\mathsf{Val}} M : A$ if and only if $\Gamma^*;\ - \vdash_{\mathsf{Lin}} M^* : A^*$, and
$\Gamma \vdash_{\mathsf{Val}} V : A$ if and only if $\Gamma^*;\ - \vdash_{\mathsf{Lin}} V^+ : A^+$.

(iii) If $M \underset{\mathsf{Val}}{\twoheadrightarrow} N$ then $M^* \underset{\mathsf{Lin}}{\twoheadrightarrow} N^*$.

(iv) $M \underset{\mathsf{Val}}{\longmapsto\!\!\!\!\rightarrow} N$ if and only if $M^* \underset{\mathsf{Lin}}{\longmapsto\!\!\!\!\rightarrow} N^*$.

We should point out here that Clause (iii) contrasts with earlier work with Odersky, Turner and Wadler [51], where we incorrectly claimed the translation of reduction sequences also to be complete; the same holds for the translations of call-by-let and call-by-need below.

Proof: The proof of Clause (i) and Clause (ii) are analogous to the corresponding proofs for Name, and the proof of Clause (iii) is given above. In Clause (iv), soundness is straightforward: we have $(\lambda x.\, M_0)\; V \underset{\mathsf{Val}}{\longmapsto} M_0[x := V]$, and

$$\begin{array}{ll} & ((\lambda x.\, M_0)\; V)^* \\ \underset{\mathsf{Lin}}{\longrightarrow} & \text{let } !x = !V^+ \text{ in } M_0^* \\ \underset{(\beta !)}{\longrightarrow} & M_0^*[x := V^+] \\ \equiv & (M_0[x := V])^* \end{array}$$

Completeness, where claimed, follows as usual from Lemma 7.2.1, Lemma 7.2.3 and Lemma 7.2.4. ♦

Why is the translation of standard reduction complete while the translation of arbitrary reduction is not? When we allow $(!\multimap, !!)$ conversion among the images, we relate Val-term images which are not related in Val.

Example 7.2.1. It is clear that in general $I\ P\ M \underset{\mathsf{Val}}{\not\twoheadrightarrow} P\ M$, but $(I\ P\ M)^* \underset{\mathsf{Lin}}{\not\twoheadrightarrow} (P\ M)^*$. Taking

$$J \equiv I^* \equiv \lambda y.\, \text{let } !x = y \text{ in } !x \ \ ,$$

we have:

$$\begin{array}{ll} & (I\ P\ M)^* \\ \equiv & (\text{let } !z = ((\text{let } !w = !J \text{ in } w)\ P^*) \text{ in } z)\ M^* \\ \to & (\text{let } !z = (J\ P^*) \text{ in } z)\ M^* \\ \to & (\text{let } !z = (\text{let } !x = P^* \text{ in } !x) \text{ in } z)\ M^* \\ \to & (\text{let } !x = P^* \text{ in let } !z = !x \text{ in } z)\ M^* \\ \to & (\text{let } !x = P^* \text{ in } x)\ M^* \\ \equiv & (P\ M)^* \end{array}$$

◊

Finding a precise categorization of such examples and a set of additional rules for call-by-value which will give rise to a complete translation remain open problems.

7.2.2 Of call-by-let

The clause in the definition of $*$ for let-bindings becomes relevant for translating Let and Need terms.

We consider the following grammar of Lin terms:

$$\begin{array}{lcll}
\text{Term image} & S, T & ::= & !U \mid P\ S \mid \text{let } !x = S \text{ in } T \\
\text{Value image} & U & ::= & x \mid \lambda y.\text{let } !x = y \text{ in } S \\
\text{Prefix} & P & ::= & U \mid \text{let } !x = S \text{ in } x
\end{array}$$

It is clear that terms matching S in the above grammar are the images of terms in Let under $*$ closed under $(\beta\multimap, \beta!)$ reduction.

Lemma 7.2.6. Let $M \in$ Let, $M^* \xrightarrow[(\beta\multimap,\beta!)]{} \!\!\!\!\rightarrow N$. Then N can be produced by the above grammar for S.

Proof: Trivial. ♦

As an inverse to $*$ on terms S we take the following maps $\ddagger$:

$$\begin{array}{rcl}
(x)^{\#} & \equiv & x \\
(\lambda y.\text{let } !x = y \text{ in } S)^{\#} & \equiv & \lambda x.\, S^{\ddagger} \\
(!U)^{\ddagger} & \equiv & U^{\#} \\
(U\ T)^{\ddagger} & \equiv & U^{\#}\ T^{\ddagger} \\
((\text{let } !x = S \text{ in } x)\ T)^{\ddagger} & \equiv & S^{\ddagger}\ T^{\ddagger} \\
(\text{let } !x = S \text{ in } T)^{\ddagger} & \equiv & \text{let } !x = S^{\ddagger} \text{ in } T^{\ddagger}
\end{array}$$

Lemma 7.2.7. For all $M \in$ Let, $(M^*)^{\ddagger} \equiv M$.

Proof: By a straightforward structural induction on M:

- $M \equiv x$. Then $((x)^*)^{\ddagger} \equiv (!x)^{\ddagger} \equiv x$.
- $M \equiv \lambda x.\, N$. Then

$$\begin{array}{rcl}
((\lambda x.\, N)^*)^{\ddagger} & \equiv & (\lambda y.\,\text{let } !x = y \text{ in } N^*)^{\ddagger} \\
& \equiv & \lambda x.\,(N^*)^{\ddagger} \\
& \equiv & \lambda x.\, N \ \ .
\end{array}$$

- $M \equiv N_0\ N_1$. Then

$$\begin{array}{rcl}
((N_0\ N_1)^*)^{\ddagger} & \equiv & ((\text{let } !x = N_0^* \text{ in } x)\ N_1^*)^{\ddagger} \\
& \equiv & (N_0^*)^{\ddagger}\ (N_1^*)^{\ddagger} \\
& \equiv & N_0\ N_1 \ \ .
\end{array}$$

- $M \equiv \text{let } x = N_0 \text{ in } N_1$. Then

$$\begin{array}{rcl}
((\text{let } x = N_0 \text{ in } N_1)^*)^{\ddagger} & \equiv & (\text{let } !x = N_0^* \text{ in } N_1^*)^{\ddagger} \\
& \equiv & \text{let } x = (N_0^*)^{\ddagger} \text{ in } (N_1^*)^{\ddagger} \\
& \equiv & \text{let } x = N_0 \text{ in } N_1 \ \ .
\end{array}$$

♦

Lemma 7.2.8. Let $S, U \in$ Lin according to the above grammar. Then $(S[x := U])^{\ddagger} \equiv S^{\ddagger}[x := U^{\#}]$.

Proof: Trivial. ♦

Lemma 7.2.9. Let $U \in$ Lin according to the above grammar. Then $U^{\#} \in$ Let is a value.

Proof: Clear from the definition of $\#$. ♦

Lemma 7.2.10. Let $S, T \in$ Lin according to the above grammar.

1. If $S \xrightarrow[(\beta\multimap,\beta!)]{} \!\!\!\!\rightarrow T$, then $S^{\ddagger} \xrightarrow[(I,V)]{} \!\!\!\!\rightarrow T^{\ddagger}$.

2. If $S \mapsto\!\!\!\twoheadrightarrow T$, then $S^\ddagger \mapsto\!\!\!\twoheadrightarrow T^\ddagger$.

Proof: We have the first clause by an analysis of the transformation of each single reduction step. There is one possible $(\beta{\multimap})$ step; we have

$$(\lambda y.\text{let } !x = y \text{ in } S)\, T \xrightarrow[(\beta\multimap)]{} \text{let } !x = T \text{ in } S$$

and

$$\begin{array}{rl} & ((\lambda y.\,\text{let } !x = y \text{ in } S)\, T)^\ddagger \\ \equiv & (\lambda x.\, S^\ddagger)\, T^\ddagger \\ \xrightarrow[(I)]{} & \text{let } x = T^\ddagger \text{ in } S^\ddagger \\ \equiv & (\text{let } !x = T \text{ in } S)^\ddagger \ . \end{array}$$

We have two different $(\beta!)$ steps: first, we take let $!x = !U$ in $S \xrightarrow[(\beta!)]{} S[x := U]$.

$$\begin{array}{rll} & (\text{let } !x = !U \text{ in } S)^\ddagger & \\ \equiv & \text{let } x = U^\# \text{ in } S^\ddagger & \\ \xrightarrow[(V)]{} & (S^\ddagger)[x := U^\#] & (7.7) \\ \equiv & (S[x := U])^\ddagger \ , & (7.8) \end{array}$$

with Reduction (7.7) by Lemma 7.2.9, and Equivalence (7.8) by Lemma 7.2.8. We can also have (let $!z = !U$ in z) $S \xrightarrow[(\beta!)]{} U\ S$. Then we have

$$\begin{array}{rl} & ((\text{let } !z = !U \text{ in } z)\ S)^\ddagger \\ \equiv & (\text{let } !z = U^\# \text{ in } z)\ S^\ddagger \\ \xrightarrow[(V)]{} & U^\#\ S^\ddagger \\ \equiv & (U\ S)^\ddagger \ . \end{array}$$

The second clause for standard sequences follows from two observations. First, all of the preceding involves top-level steps, which are standard, and so we have the result for top-level standard steps. Second, it is clear that the translations take evaluation contexts to evaluation contexts, and so the second clause follows. ♦

Proposition 7.2.11, Steadfast translation of Let**.** The translation $*$ from Let to Lin preserves substitution of values, is sound and complete for typing judgments and is sound for reduction sequences.

(i) $(M[x := V])^* \equiv M^*[x := V^+]$.

(ii) $\Gamma \vdash_{\mathsf{Let}} M : A$ if and only if $\Gamma^*;\ - \vdash_{\mathsf{Lin}} M^* : A^*$, and
$\Gamma \vdash_{\mathsf{Let}} V : A$ if and only if $\Gamma^*;\ - \vdash_{\mathsf{Lin}} V^+ : A^+$.

(iii) If $M \xrightarrow[\mathsf{Let}]{}\!\!\!\!\!\twoheadrightarrow N$ then $M^* \xrightarrow[\mathsf{Lin}]{}\!\!\!\!\!\twoheadrightarrow N^*$.

(iv) $M \underset{\mathsf{Let}}{\mapsto\!\!\!\twoheadrightarrow} N$ if and only if $M^* \underset{\mathsf{Lin}}{\mapsto\!\!\!\twoheadrightarrow} N^*$.

Proof: The proof of (i) and (ii) is similar to that for Val. To prove (iii) in the forward direction, consider each possible reduction in Let. Reduction by rule (I) translates to

$$\begin{array}{rl} & (\text{let } z = !(\lambda y.\,\text{let } !x = y \text{ in } M^*) \text{ in } z)\ N^* \\ \xrightarrow[(\beta!)]{} & (\lambda y.\,\text{let } !x = y \text{ in } M^*)\ N^* \\ \xrightarrow[(\beta\multimap)]{} & \text{let } !x = N^* \text{ in } M^* \ . \end{array}$$

Reduction by rule (V) translates to

$$\begin{array}{rl} & \text{let } !x = !V^+ \text{ in } M^* \\ \xrightarrow[(\beta!)]{} & M^*[x := V^+] \ . \end{array}$$

Reduction by rule (C) translates to

$$\begin{array}{rl} & (\text{let } !z = (\text{let } !x = L^* \text{ in } M^*) \text{ in } z)\ N^* \\ \xrightarrow[(!!)]{} & (\text{let } !x = L^* \text{ in } (\text{let } !z = M^* \text{ in } z))\ N^* \\ \xrightarrow[(!\multimap)]{} & \text{let } !x = L^* \text{ in } ((\text{let } !z = M^* \text{ in } z)\ N^*) \ . \end{array}$$

And reduction by rule (A) translates to (!!) in Lin.

For Clause (iv) we have soundness for top-level steps by the equivalent reasoning for general (I, V) steps above, and for compatibility by the observation that $*$ preserves evaluation contexts; completeness follows by Lemma 7.2.10.(2). ♦

Again, $*$ is not complete for general Let-reduction sequences: we have

$$\begin{array}{cl} & ((\text{let } x = I\, I \text{ in } x)\; N)^* \\ \equiv & (\text{let } !z = (\text{let } !x = (I\, I)^* \text{ in } !x) \text{ in } z)\; N^* \\ \xrightarrow[(!!)]{} & (\text{let } !x = (I\, I)^* \text{ in let } !z = (!x) \text{ in } z)\; N^* \\ \xrightarrow[(\beta!)]{} & (\text{let } !x = (I\, I)^* \text{ in } x)\; N^* \\ \equiv & (I\, I\, N)^* \;\; , \end{array}$$

but $(\text{let } x = I\, I \text{ in } x)\, N \twoheadrightarrow (I\, I)\, N$ does not hold in Let. Similar to completeness for standard sequences, the translation can also be shown both sound and complete if we restrict reduction in Let to (I, V) steps, and in Lin to $(\beta{\multimap}, \beta!)$ steps.

Proposition 7.2.12. Let $M, N \in$ Let. Then $M \xrightarrow[(I,V)]{} \!\!\!\!\twoheadrightarrow N$ if and only if

$$M^* \xrightarrow[(\beta\multimap,\beta!)]{}\!\!\!\!\twoheadrightarrow N^* \;\; .$$

Proof: The forward direction follows as in the proof of the forward direction of the last case in Proposition 7.2.11; the backward direction follows from Lemma 7.2.6, Lemma 7.2.7 and Lemma 7.2.10. ♦

7.2.3 Of call-by-need

By design, if $*$ is now viewed as taking the call-by-need calculus Need into the affine calculus Aff, then the transformation result still holds as in Let. We begin with a characterization of the steadfast images of answers and of evaluation contexts. We take $\bar{A}, \bar{E}$ according to the above grammar:

Need-answer images	$\bar{A}$	$::= \;!\lambda y.\, \text{let } !x = y \text{ in } S$
		$\mid\; \text{let } !x = S \text{ in } \bar{A}$
Need-evaluation context images	$\bar{E}$	$::= [\,] \mid (\text{let } !x = \bar{E} \text{ in } x)\; S \mid \bar{E}\; S$
		$\mid\; \text{let } !x = M \text{ in } \bar{E}$
		$\mid\; \text{let } !x = \bar{E} \text{ in } \bar{E}[!x]$

The characterization of terms $\bar{A}$ is straightforward; both of these results have a straightforward inductive proof.

Lemma 7.2.13. Let $A \in$ Need; then there exists some $\bar{A}_0$ such that $A^* \equiv \bar{A}_0$. ♦

Lemma 7.2.14. For all $\bar{A}$, $\bar{A}^\ddagger$ is a Need-answer. ♦

The correspondences of evaluation contexts is a bit more complicated because the steadfast translation adds prefixes to variable occurrences; however the proofs are still just straightforward

Lemma 7.2.15. Let E be a Need-evaluation context. Then there exists some term $\bar{E}_0$ such that for all $x \in \text{fv}(E)$ we have $(E[x])^* \equiv \bar{E}_0[!x]$. ♦

Lemma 7.2.16. Let $\bar{E}$ match the above grammar. Then there is some E_0 such that for all $x \notin \text{fv}(\bar{E})$ we have some $(\bar{E}[!x])^\ddagger \equiv E_0[x]$. ♦

The arguments of completeness for value substitutions and for typing judgements are the same as in the previous sections for Let and Val. For the completeness of the translation of standard reduction sequences, we have the following lemma.

Lemma 7.2.17. Let $S \underset{\mathsf{Aff}}{\longmapsto} T$; then $S^\ddagger \underset{\mathsf{Need}}{\longmapsto} T^\ddagger$.

Proof: For the four top-level reduction steps we have simple case analyses, using Lemma 7.2.14 and Lemma 7.2.16 to ensure that terms have the required form for the restricted redex bodies; compatible closure is straightforward. ♦

Proposition 7.2.18, Steadfast translation of Need. The translation $*$ from Need to Aff preserves substitution of values, is sound and complete for typing judgments and is sound for reduction sequences.

(i) $(M[x := V])^* \equiv M^*[x := V^+]$.

(ii) $\Gamma \vdash_{\mathsf{Need}} M : A$ if and only if $\Gamma^*;\ - \vdash_{\mathsf{Aff}} M^* : A^*$, and
$\Gamma \vdash_{\mathsf{Need}} V : A$ if and only if $\Gamma^*;\ - \vdash_{\mathsf{Aff}} V^+ : A^+$.

(iii) If $M \xrightarrow[\mathsf{Need}]{} N$ then $M^* \xrightarrow[\mathsf{Aff}]{} N^*$.

(iv) $M \mapsto_{\mathsf{Need}} N$ if and only if $M^* \mapsto_{\mathsf{Aff}} N^*$.

Proof: The proof is as for Let, noting that reduction by (G) in Need translates to $(!\mathrm{W})$ in Aff; completeness of the translation for standard reduction follows from Lemma 7.2.17. ♦

These results support the choice of affine answer in Section 6.5. Taking the naïve definition of answers

$$A ::= \lambda x.\, M \mid !A \mid \text{let } !x = M \text{ in } A \quad .$$

A simple example is the Need term

$$\text{let } x = I\, I \text{ in } x \quad ,$$

which is clearly not an answer. Unfortunately, its steadfast translation would be an answer according to the above definition, as we have

$$(\text{let } x = I\, I \text{ in } x)^* \equiv \text{let } !x = (I\, I)^* \text{ in } !x \quad .$$

Part III

Separate Modes for Weakening and Contraction

Chapter 8

The separated-linear lambda calculus

... the rest sounds like something dreamt up by a vindictive bureaucrat on acid while closely inspecting something Hieronymous Bosch painted on one of his bleak but imaginative days.

Still, it has its attractions.

(Banks)

IN this chapter we develop SLin, a separated-linear lambda calculus. In contrast to the linear and affine calculi, which control weakening and contraction through a single mode, and which we refer to by contrast as the *unitary* modal calculi, we now separate this functionality into separate modes, one for each operation. Thus we have two different modal prefixes for both types and terms. We must also segregate typing assumptions by whether they allow both operations, neither, only weakening or only contraction, and so we have four distinct zones. This system is motivated by Jacobs' work on models of linear logic with separate modes [37], which we discuss in more detail below.

Figure 8.1 presents the types and terms of the separated-linear lambda calculus SLin. A type is a base type, a linear function, or one of two modal types. As in Name and Val, basic terms include variables, function abstractions and applications. We have a number of new term forms, however, for the introduction and elimination of the new modal operators. The prefixes $!^w$ and $!^c$ denote terms which one may discard or duplicate, respectively. We refer to each prefix as the *complement* of the other. The corresponding $!^w$- and $!^c$-eliminators strip prefixes from expressions for discarding, or for copying into another term. However, the remaining eliminator has no corresponding prefix, and is a bit mysterious until one considers the structure of the typing rules.

Consistent with our convention in Lin, in a judgment

$$\Gamma;\ \Delta;\ \Phi;\ \Psi \vdash_{\mathsf{SLin}} M : A\ \ ,$$

we take the leftmost environment Γ to contain fully intuitionistic assumptions, and Ψ to contain fully linear assumptions. Moreover, Δ will contain assumptions which we allow to be weakened, but not contracted, and Φ will contain assumptions which may be contracted, but not weakened. These four-zoned judgments can lead to a confusing level of clutter in typing rules and trees, so we do not actually use them in practice, but instead introduce some shorthand notation. First, we will let $\mathcal{E}, \mathcal{F}, \mathcal{G}$ range over quadruples $(\Gamma;\ \Delta;\ \Phi;\ \Psi)$ of the various contexts, and write $\mathcal{E}, \mathcal{F}$ to denote the component-wise concatenation of the two quadruples. We also adapt the notation of Jacobs [37], and use the marker $\blacktriangleright_l$ (respectively, $\blacktriangleright_c$, $\blacktriangleright_w$, $\blacktriangleright_i$) to indicate additions to the linear (relevant, affine, intuitionistic) zone of a quadruple. For example, if

$$\mathcal{E} \equiv \Gamma_0;\ \Delta_0;\ \Phi_0;\ \Psi_0 \qquad \text{and} \qquad \mathcal{F} \equiv \Gamma_1;\ \Delta_1;\ \Phi_1;\ \Psi_1$$

then $\mathcal{E}, \mathcal{F} \equiv \Gamma_0, \Gamma_1;\ \Delta_0, \Delta_1;\ \Phi_0, \Phi_1;\ \Psi_0, \Psi_1$, and by

$$\mathcal{E}, \blacktriangleright_l x_0 : A_0, x_1 : A_1, \blacktriangleright_w y : B$$

we mean

$$\Gamma_0;\ \Delta_0, y : B;\ \Phi_0;\ \Psi_0, x_0 : A_0, x_1 : A_1\ \ .$$

The role of the "extra" eliminator structure is now easier to see: we must be able to eliminate variables from each of the four zones, and this eliminator corresponds to the intuitionistic zone. We have no intuitionistic prefix because such an amalgamation of the $!^w$ and $!^c$ prefixes is exactly what we are trying to avoid with this calculus. We wish to enable weakening and contraction separately from each other, and so an intuitionistic term is one marked as both weakenable and

Syntactic domains

$$
\begin{array}{lrl}
\text{Types} & A, B & ::= Z \mid A \multimap B \mid\, !^w A \mid\, !^c A \\
\text{Terms} & L, M, N & ::= x \mid \lambda x.\, M \mid M\ N \\
 & & \mid\ \text{let } !^w x = M \text{ in } N \mid\, !^w M \\
 & & \mid\ \text{let } !^c x = M \text{ in } N \mid\, !^c M \\
 & & \mid\ \text{let } !^i x = M \text{ in } N
\end{array}
$$

Typing rules

The typing axiom

$$\dfrac{}{\blacktriangleright^{l} x : A \vdash x : A}\ \text{Id}$$

Substructure rules

$$\dfrac{\mathcal{E}, \blacktriangleright^{c} x : A, y : A \vdash M : B}{\mathcal{E}, \blacktriangleright^{c} z : A \vdash M[x := z, y := z] : B}\ \text{Contraction} \qquad \dfrac{\mathcal{E} \vdash M : B}{\mathcal{E}, \blacktriangleright^{w} x : A \vdash M : B}\ \text{Weakening}$$

$$\dfrac{\mathcal{E}, \blacktriangleright^{i} x : A, y : A \vdash M : B}{\mathcal{E}, \blacktriangleright^{i} z : A \vdash M[x := z, y := z] : B}\ \text{Contraction/W} \qquad \dfrac{\mathcal{E} \vdash M : B}{\mathcal{E}, \blacktriangleright^{i} x : A \vdash M : B}\ \text{Weakening/C}$$

$$\dfrac{\mathcal{E}, \blacktriangleright^{l} x : A \vdash M : B}{\mathcal{E}, \blacktriangleright^{w} x : A \vdash M : B}\ \text{Dereliction!}^w \qquad \dfrac{\mathcal{E}, \blacktriangleright^{l} x : A \vdash M : B}{\mathcal{E}, \blacktriangleright^{c} x : A \vdash M : B}\ \text{Dereliction!}^c$$

$$\dfrac{\mathcal{E}, \blacktriangleright^{c} x : A \vdash M : B}{\mathcal{E}, \blacktriangleright^{i} x : A \vdash M : B}\ \text{Dereliction!}^w/C \qquad \dfrac{\mathcal{E}, \blacktriangleright^{w} x : A \vdash M : B}{\mathcal{E}, \blacktriangleright^{i} x : A \vdash M : B}\ \text{Dereliction!}^c/W$$

Term structure rules

$$\dfrac{\blacktriangleright^{i} \Gamma, \blacktriangleright^{c} \Phi \vdash M : A}{\blacktriangleright^{i} \Gamma, \blacktriangleright^{c} \Phi \vdash M : !^c A}\ !^c\text{-I} \qquad \dfrac{\mathcal{E} \vdash M : !^c A \qquad \mathcal{F}, \blacktriangleright^{c} x : A \vdash N : B}{\mathcal{E}, \mathcal{F} \vdash \text{let } !^c x = M \text{ in } N : B}\ !^c\text{-E}$$

$$\dfrac{\blacktriangleright^{i} \Gamma, \blacktriangleright^{w} \Phi \vdash M : A}{\blacktriangleright^{i} \Gamma, \blacktriangleright^{w} \Phi \vdash M : !^w A}\ !^w\text{-I} \qquad \dfrac{\mathcal{E} \vdash M : !^w A \qquad \mathcal{F}, \blacktriangleright^{w} x : A \vdash N : B}{\mathcal{E}, \mathcal{F} \vdash \text{let } !^w x = M \text{ in } N : B}\ !^w\text{-E}$$

$$\dfrac{\mathcal{E} \vdash M : !^w !^c A \qquad \mathcal{F}, \blacktriangleright^{i} x : A \vdash N : B}{\mathcal{E}, \mathcal{F} \vdash \text{let } !^i x = M \text{ in } N : B}\ !^i\text{-E}^a$$

$$\dfrac{\mathcal{E} \vdash M : !^c !^w A \qquad \mathcal{F}, \blacktriangleright^{i} x : A \vdash N : B}{\mathcal{E}, \mathcal{F} \vdash \text{let } !^i x = M \text{ in } N : B}\ !^i\text{-E}^b$$

$$\dfrac{\mathcal{E}, \blacktriangleright^{l} x : A \vdash M : B}{\mathcal{E} \vdash \lambda x.\, M : A \multimap B}\ \multimap\text{-I} \qquad \dfrac{\mathcal{E} \vdash M : A \multimap B \qquad \mathcal{F} \vdash N : A}{\mathcal{E}, \mathcal{F} \vdash M\ N : B}\ \multimap\text{-E}$$

Figure 8.1: Syntax and typing rules for the separated-linear lambda calculus.

β-reduction rules

$$
\begin{array}{lll}
(\beta\multimap) & (\lambda x.\, M)\, N & \xrightarrow[\mathsf{SLin}]{} M[x := N] \\
(\beta!^c) & \text{let } !^c x = !^c M \text{ in } N & \xrightarrow[\mathsf{SLin}]{} N[x := M] \\
(\beta!^w) & \text{let } !^w x = !^w M \text{ in } N & \xrightarrow[\mathsf{SLin}]{} N[x := M] \\
(\beta!^i_{cw}) & \text{let } !^i x = !^c!^w M \text{ in } N & \xrightarrow[\mathsf{SLin}]{} N[x := M] \\
(\beta!^i_{wc}) & \text{let } !^i x = !^w!^c M \text{ in } N & \xrightarrow[\mathsf{SLin}]{} N[x := M]
\end{array}
$$

Weakening rule

$$
(\mathrm{W}!^i) \qquad \text{let } !^i x = !^w M \text{ in } N \xrightarrow[\mathsf{SLin}]{} N,\ x \notin \mathrm{fv}(N)
$$

Commuting conversion rules

$$
(!^\square\multimap) \qquad (\text{let } !^\square x = L \text{ in } M)\, N \xrightarrow[\mathsf{SLin}]{} \text{let } !^\square x = L \text{ in } (M\ N)
$$

$$
(!^\square!^\triangle) \qquad \text{let } !^\square x = (\text{let } !^\triangle y = L \text{ in } M) \text{ in } N \xrightarrow[\mathsf{SLin}]{} \text{let } !^\triangle y = L \text{ in } (\text{let } !^\square x = M \text{ in } N)
$$

where $!^\square$ and $!^\triangle$ each range over $!^w$, $!^c$, $!^i$.

$$
(!^i!^\square!^w) \qquad \text{let } !^i x = !^\square(\text{let } !^i y = !^w L \text{ in } M) \text{ in } N \xrightarrow[\mathsf{SLin}]{} \text{let } !^i y = !^w L \text{ in let } !^i x = !^\square M \text{ in } N
$$

where $!^\square$ ranges over $!^w$ and $!^c$.

Figure 8.2: Reduction in the separated-linear lambda calculus.

contractable: that is, of either form $!^w!^c M$ or $!^c!^w M$; the $!^i$-eliminator does not distinguish between the two. Equally we might have two separate eliminators for the two permutations of the prefixes; this issue is an insubstantial question of syntax. What the intuitionistic fourth zone reflects is that the composititon of the modes is distinct from either mode individually, and that in the type system we do not distinguish between the order of the composition.

The typing rules are straightforward. The Id rule introduces a linear variable. We do not have introduction rules for other zones, but rather, the Dereliction rules. Since maintaining an assumption in (for example) the linear zone entails *not* performing certain activities, we can correctly move an assumption from a more restrictive zone to a less restrictive zone at any time, but not vice-versa. While the variable in the subject of the derelicted assumption will not (yet) be used in the newly enabled manner, the typing is still appropriate, and we can subsequently apply the newly enabled rules. We have two each of the Weakening and Contraction rules : one where the operation is performed from a zone where the other operation is not allowed (for example, the Weakening rule adds a variable into the zone where weakening but not contraction is allowed) and one where the operation is performed in the intuitionistic zone. Introduction of a modal prefix is allowed only if a term contains no free variables on which the operation enabled by the prefix is not allowed. For example, the $!^w$-I rule requires that the contractable and linear zones be empty; a term under a $!^w$ prefix should be discardable, but if such a free variable were discarded we would violate the typing assumptions.

The reduction rules for SLin are given in Figure 8.2. The main reduction rules for SLin are the five β rules and the weakening rule. As usual, we have a β rule whenever an elimination rule occurs immediately below the corresponding introduction rule; we have two such rules for the $!^i$-eliminator since the two prefixes to be eliminated may arrive in either order. The weakening rule allows us to take advantage of the fact that a term prefixed by $!^w$ may be discarded immediately if it is not to be used, without necessarily finding a $!^c$ prefix for a $!^i$-eliminator.

The remaining reduction rules are commuting rules for managing the additional structure. The rôle of the $(!^\square\multimap)$ and $(!^\square!^\triangle)$ rules are the same as for the similarly named rules of Lin. In the left-hand sides of the former, a modal elimination rule can block a $(\beta\multimap)$ step by hiding an abstraction in the left subterm; in the left-hand sides of the latter, a modal elimination rule can block a $(\beta!)$ step by hiding a correctly prefixed term. In both cases, the rule expands the scope of the variable bound by the blocking eliminator to free any blocked reduction at that point. The $(!^i!^\square!^w)$ rules extend the $(!^\square!^\triangle)$ rules to the cases where an amalgamated

Well-typed left-hand side

$$
\dfrac{
\dfrac{
\dfrac{\dfrac{\vdots\,\mathcal{P}_1}{\blacktriangleright_i\Gamma \vdash L : A_0}}{\blacktriangleright_i\Gamma \vdash\, !^c L : !^c A_0}\;!^c\text{-I}
\qquad
\dfrac{\dfrac{\vdots\,\mathcal{P}_2}{\blacktriangleright_i\Delta, \blacktriangleright_c y : A_0 \vdash M_0 : B}}{\blacktriangleright_i\Delta, \blacktriangleright_c y : A_0 \vdash\, !^c M_0 :!^c B}\;!^c\text{-I}
}{\dfrac{\blacktriangleright_i\Delta \vdash \text{let } !^c y = !^c L \text{ in } !^c M_0 :!^c B}{\blacktriangleright_i\Gamma, \Delta \vdash\, !^w(\text{let } !^c y = !^c L \text{ in } !^c M_0) :!^w !^c B}\;!^w\text{-I}}\;!^c\text{-E}
\qquad
\dfrac{\dfrac{\vdots\,\mathcal{P}_3}{\mathcal{E} \vdash N : A}}{\mathcal{E}, \blacktriangleright_i x : B \vdash N : A}\;\text{W/C}
}{\mathcal{E}, \blacktriangleright_i\Gamma, \Delta \vdash \text{let } !^i x = !^w(\text{let } !^c y = !^c L \text{ in } !^c M_0) \text{ in } N : A}\;!^i\text{-E}_1
$$

Ill-typed right-hand side

$$
\dfrac{
\dfrac{\dfrac{\vdots\,\mathcal{P}_1}{\blacktriangleright_i\Gamma \vdash L : A_0}}{\blacktriangleright_i\Gamma \vdash\, !^c L : !^c A_0}\;!^c\text{-I}
\qquad
\dfrac{
\dfrac{\dfrac{\dfrac{\vdots\,\mathcal{P}_2}{\blacktriangleright_i\Delta, \blacktriangleright_c y : A_0 \vdash M_0 : B}}{\blacktriangleright_i\Delta, \blacktriangleright_c y : A_0 \vdash\, !^c M_0 : !^c B}\;!^c\text{-I}}{\blacktriangleright_i\Delta, \blacktriangleright_c y : A_0 \vdash\, !^w !^c M_0 : !^w !^c B}\;\text{(Illegal step)}
\qquad
\dfrac{\vdots\,\mathcal{P}_3}{\mathcal{E} \vdash N : A}
}{\mathcal{E}, \blacktriangleright_i\Delta, \blacktriangleright_c y : A_0 \vdash \text{let } !^i x = !^w !^c M_0 \text{ in } N : A}\;\text{W/C}
}{\mathcal{E}, \blacktriangleright_i\Gamma, \Delta \vdash \text{let } !^c y = !^c L \text{ in let } !^i x = !^w !^c M_0 \text{ in } N : A}\;!^i\text{-E}_1
$$

Figure 8.3: Example of the failure of subject reduction in an omitted $(!^i!^\square!^w)$ rule. We abbreviate "Weakening/C" as "W/C".

---

eliminator "sees" one of the two modal prefixes required for a reduction step, but under that prefix another binding can hide the second prefix. Again, the $(!^i!^\square!^w)$ rules simply expand the scope of the inner binding and remove it from a position where it may no longer block the reduction of the outer binding.

Some rules might seem to be missing from the $(!^i!^\square!^w)$ set, for example a rule:

$$\text{let } !^i x = !^w(\text{let } !^c y = !^c L \text{ in } M) \text{ in } N \tag{8.1}$$
$$\to \quad \text{let } !^c y = !^c L \text{ in let } !^i x = !^w M \text{ in } N \ . \tag{8.2}$$

This rule — as would the similar rule with the $!^w$'s and $!^c$'s exchanged — disallows the subject reduction property. By throwing away part of a discardable term, but keeping another part which is not explicitly discardable but which we remove from under a prefix which makes it discardable, we can disrupt the allowed usages. For example, we consider the case where x does not appear free in N, and moreover take $M \equiv\, !^c M_0$. So on the left-hand side we have some typing as shown in Figure 8.3. However, L is not discardable and is not bound to a discardable variable: possibly, L will reduce to an expression $!^w L_0$ whose prefix can be removed and the body discarded, but nonetheless L by itself cannot in general be discarded.[1] So when trying to type the right-hand side of Expression (8.2), we fail because the variable y which eliminates the prefix from L prevents the addition of the $!^w$ prefix to $!^c M_0$. The problem becomes clearer if we allow the $(\beta!^i)$ contraction of the new inner eliminator: since $x \notin \text{fv}(N)$ we have

$$
\begin{aligned}
& \text{let } !^c y = !^c L \text{ in } (\text{let } !^i x = !^w !^c M_0 \text{ in } N) \\
\xrightarrow[(\beta!^i)]{} \quad & \text{let } !^c y = !^c L \text{ in } N \ ,
\end{aligned}
$$

and since $y \notin \text{fv}(N)$ as well, we would clearly have to perform weakening on a variable which is exclusively contractable. These problems do not arise for the $(!^\square!^\triangle)$ rules, since (for example) they would not move a $!^c$-eliminator from under a $!^w$ prefix.

A further suspicion can be raised about what $(!^i!^\square!^w)$ rules do remain: why do we require the innermost prefix on the subterm L? Although their presence is somewhat unattractive, without them we lose confluence. Consider the less restrictive rule

$$
\begin{aligned}
& \text{let } !^i x = !^w(\text{let } !^i y = L \text{ in } M) \text{ in } N \\
\to \quad & \text{let } !^i y = L \text{ in let } !^i x = !^w M \text{ in } N
\end{aligned}
$$

[1] Note that we are not implying that L cannot be typed using only discardable assumptions; in fact it would be trivial to do so with Dereliction rules. What we are stating is that in Expression (8.2) the subterm $!^c L$ is not explicitly discardable, and that y is not discardable in $M \equiv !^c M_0$.

and the term

$$\text{let } !^i x = !^w(\text{let } !^i y = (\text{let } !^c z = L_0 \text{ in } L_1) \text{ in } M) \text{ in } N \ \ .$$

We can apply the new rule at the top level, and the $(!^i!^c)$ rule to the subterm

$$\text{let } !^i y = (\text{let } !^c z = L_0 \text{ in } L_1) \text{ in } M \ \ .$$

After the latter contraction we can no longer apply a $(!^i!^{\Box}!^w)$ rule at the top level, since we have just restricted

$$\text{let } !^i x = !^w(\text{let } !^c z = L_0 \text{ in let } !^i y = L_1 \text{ in } M) \text{ in } N$$

from being the left-hand side of a $(!^i!^{\Box}!^w)$ rule for the sake of subject reduction. Since we could have for example $L_0 \equiv w$ for some free variable w, confluence would not hold in general.

Based on his earlier work on models for the substructural lambda calculi [36], Jacobs identified a category of models for a variant of linear logic with separate exponential connectives for each of weakening and contraction which generalizes affine and relevance logic as special cases [37]. Jacobs limited the scope of his study to the model theory, giving only a brief sketch of a possible syntax for the logic and not considering its relation to intuitionistic systems. It would be fair to characterize the calculus we present here as a syntax for an intuitionistic fragment of Jacobs' system, if with a quite different syntax than Jacobs' proposed format. Aside from Jacobs' model, we know of no other research similar to our system. The results of this and the following chapter have also appeared in a slightly different format in an unpublished technical report [50], and have been summarized in a workshop proceedings, albeit in an earlier, less complete form [49].

8.1 Substitution properties

Typing proofs and reduction sequences in SLin are both well-behaved under substitution.

Lemma 8.1.1. Let M, N be SLin terms and A, B be SLin types.

1. If $\mathcal{E}, \blacktriangleright^{l} x : B \vdash M : A$ and $\mathcal{F} \vdash N : B$.
   then $\mathcal{E}, \mathcal{F} \vdash M[x := N] : A$.

2. If $\mathcal{E}, \blacktriangleright^{c} x : B \vdash M : A$ and $\blacktriangleright^{i} \Gamma, \blacktriangleright^{c} \Delta \vdash N : B$.
   then $\mathcal{E}, \blacktriangleright^{i} \Gamma, \blacktriangleright^{c} \Delta \vdash M[x := N] : A$.

3. If $\mathcal{E}, \blacktriangleright^{w} x : B \vdash M : A$ and $\blacktriangleright^{i} \Gamma, \blacktriangleright^{w} \Delta \vdash N : B$.
   then $\mathcal{E}, \blacktriangleright^{i} \Gamma, \blacktriangleright^{w} \Delta \vdash M[x := N] : A$.

4. If $\mathcal{E}, \blacktriangleright^{i} x : B \vdash M : A$ and $\blacktriangleright^{i} \Gamma \vdash N : B$.
   then $\mathcal{E}, \blacktriangleright^{i} \Gamma \vdash M[x := N] : A$.

Lemma 8.1.2. Let $L, M, N \in \mathsf{SLin}$ where $M \xrightarrow[\mathsf{SLin}]{} N$. Then:

1. $M[x := L] \xrightarrow[\mathsf{SLin}]{} N[x := L]$.

2. $L[x := M] \xrightarrow[\mathsf{SLin}]{}\!\!\!\!\!\rightarrow L[x := N]$ and

Proof: The typing result is an easy induction on M, as is the first clause of the reduction result. The second clause, as usual, is just compatible closure. ⧫

8.2 Subject reduction

With the above properties of substitution, subject reduction follows easily.

Proposition 8.2.1. SLin satisfies the subject reduction property.

Proof: By induction on the depth of the typing tree. For the β rules we use Lemma 8.1.1; for the other rules we need simply rearrange the typing trees. ⧫

Syntactic domains As in SLin, noting as well:

$$\text{Prefixes} \quad P \ ::= \text{let } !^i x = M \text{ in} \mid \text{let } !^w x = M \text{ in} \mid \text{let } !^c x = M \text{ in}$$

β reduction rules (As in SLin)

$$\begin{array}{lllll}
(\beta\multimap) & (\lambda x.\, M)\, N & \xrightarrow[\mathsf{SLin}^\star]{} & M[x := N] \\
(\beta!^c) & \text{let } !^c x = !^c M \text{ in } N & \xrightarrow[\mathsf{SLin}^\star]{} & N[x := M] \\
(\beta!^w) & \text{let } !^w x = !^w M \text{ in } N & \xrightarrow[\mathsf{SLin}^\star]{} & N[x := M] \\
(\beta!^i_{cw}) & \text{let } !^i x = !^c !^w M \text{ in } N & \xrightarrow[\mathsf{SLin}^\star]{} & N[x := M] \\
(\beta!^i_{wc}) & \text{let } !^i x = !^w !^c M \text{ in } N & \xrightarrow[\mathsf{SLin}^\star]{} & N[x := M]
\end{array}$$

Weakening reduction rule (As in SLin)

$$(\mathrm{W}!^i) \qquad \text{let } !^i x = !^w M \text{ in } N \ \xrightarrow[\mathsf{SLin}^\star]{} \ N,\ x \notin \mathrm{fv}(N)$$

Commuting conversion rules

$$\begin{array}{lll}
(\star!^\square\multimap) & (P^n \text{ let } !^\square x = L \text{ in } M)\, N \ \xrightarrow[\mathsf{SLin}^\star]{} \ P^n \text{ let } !^\square x = L \text{ in } (M\ N) \\
(\star!^\square!^\triangle) & \text{let } !^\square x = (P^n \text{ let } !^\triangle y = L \text{ in } M) \text{ in } N \\
& \qquad \xrightarrow[\mathsf{SLin}^\star]{} \ P^n \text{ let } !^\triangle y = L \text{ in } (\text{let } !^\square x = M \text{ in } N) \\
& \text{where } !^\square \text{ and } !^\triangle \text{ each range over } !^w, !^c, !^i.
\end{array}$$

$$\begin{array}{ll}
(\star!^i!^\square!^w) & \text{let } !^i x = !^\square({}^{\mathrm{i}w}\mathcal{P}\, M) \text{ in } N \ \xrightarrow[\mathsf{SLin}^\star]{} \ {}^{\mathrm{i}w}\mathcal{P}\, (\text{let } !^i x = !^\square M \text{ in } N) \\
& \text{where } !^\square \text{ ranges over } !^w \text{ and } !^c.
\end{array}$$

Damage function

$$\begin{array}{rcl}
\mathbf{d}(\sigma, 0) & = & 0 \\
\mathbf{d}(\sigma : M \xrightarrow[(\beta,\mathrm{W})]{} N, m) & = & -1,\ m > 0 \text{ and top-level } \sigma \\
\mathbf{d}(\sigma : M \xrightarrow[(\star!^\square\multimap)]{} N, m) & = & 0 \\
\mathbf{d}(\sigma : M \xrightarrow[(\star!^\square!^\triangle)]{} N, m) & = & n,\ m > 0,\ \text{top-level } \sigma \text{ and } n \text{ prefixes moved} \\
\mathbf{d}(\sigma : M \xrightarrow[(\star!^i!^\square!^w)]{} N, m) & = & n,\ m > 0 \text{ top-level } \sigma \text{ and } n \text{ prefixes moved} \\
\mathbf{d}(P\, M \to P\, N, m) & = & \mathbf{d}(M \to N, m-1),\ m > 0 \\
\mathbf{d}(M \to N, m) & = & 0,\ \text{other compatible closure}
\end{array}$$

Figure 8.4: The generalized separated-linear lambda calculus.

8.3 Generalized reduction and developments

8.3.1 Generalized reduction

Figure 8.4 describes $\mathsf{SLin}^\star$, a variation of the SLin reduction rules. $\mathsf{SLin}^\star$ and SLin share a common term language, but as in the generalizations of the other calculi, $\mathsf{SLin}^\star$ allows single $(!^\square\multimap, !^\square!^\triangle, !^i!^\square!^w)$ steps to move more than one eliminator prefix at a time; we take the *rank* of a reduction step as before. Again, single-step $\mathsf{SLin}^\star$ reduction relates more terms than single-step SLin reduction, although their reflexive, transitive closures are clearly equal:

Lemma 8.3.1. Let $M, N \in \mathsf{SLin}$. We have $M \xrightarrow[\mathsf{SLin}]{} \!\!\!\!\!\rightarrow N$ if and only if $M \xrightarrow[\mathsf{SLin}^\star]{} \!\!\!\!\!\rightarrow N$.

Proof: As usual, n-rank steps in $\mathsf{SLin}^\star$ correspond to n successive steps in SLin. ♦

8.3.2 Marked reduction

Figure 8.5 shows the syntax and Figure 8.6 the reduction rules for the marked variant $\mathsf{SLin}^{\star\prime}$ of $\mathsf{SLin}^\star$. Compatible closure in $\mathsf{SLin}^{\star\prime}$ is more complicated than usual, and we present these closure rules in Figure 8.7. We refer to reduction by rules $(\beta\multimap_0, \beta!^\square{}_0, \star!^\square\multimap_0, \star!^\square\multimap^<_0, \star!^\square!^\triangle{}_0, \star!^\square!^{\triangle<}_0, !^i!^\square!^w{}_0, !^i!^\square!^{w<}_0)$ as $\mathsf{SLin}^{\star\prime}_0$. We have the usual correspondence between the marked and unmarked systems.

Syntactic domains

Terms	L', M', N'	$::= x \mid \lambda x.\, M' \mid !^w M' \mid !^c M' \mid M'\, N' \mid \mathcal{P}\, M$
		$\mid\ {}^{[\beta\multimap]}(\lambda x.\, M')\, N'$
		$\mid\ {}^{[\star !^c \multimap_n]}(\mathcal{P}^{n-1}\, {}^{\mathrm{c}}\mathcal{P}\, M')\, N'$
		$\mid\ {}^{[\star !^w \multimap_n]}(\mathcal{P}^{n-1}\, {}^{\mathrm{w}}\mathcal{P}\, M')\, N'$
		$\mid\ {}^{[\star !^i \multimap_n]}(\mathcal{P}^{n-1}\, {}^{\mathrm{i}}\mathcal{P}\, M')\, N'$
General eliminator prefixes	$\mathcal{P}$	$::= {}^{\mathrm{i}}\mathcal{P} \mid {}^{\mathrm{w}}\mathcal{P} \mid {}^{\mathrm{c}}\mathcal{P}$
$!^i$-eliminator prefixes	${}^{\mathrm{i}}\mathcal{P}$	$::= \text{let } !^i x = M' \text{ in}$
		$\mid\ {}^{\mathrm{i}^w}\mathcal{P} \mid {}^{\mathrm{i}^c}\mathcal{P}$
		$\mid\ {}^{[\star !^i !^w{}_n]}\text{let } !^i x = (\mathcal{P}^{n-1}\, {}^{\mathrm{w}}\mathcal{P}\, M') \text{ in}$
		$\mid\ {}^{[\star !^i !^c{}_n]}\text{let } !^i x = (\mathcal{P}^{n-1}\, {}^{\mathrm{c}}\mathcal{P}\, M') \text{ in}$
		$\mid\ {}^{[\star !^i !^i{}_n]}\text{let } !^i x = (\mathcal{P}^{n-1}\, {}^{\mathrm{i}}\mathcal{P}\, M') \text{ in}$
– with a $!^w$-prefixed bound term	${}^{\mathrm{i}^w}\mathcal{P}$	$::= \text{let } !^i y = !^w L' \text{ in}$
		$\mid\ {}^{[\beta !^i_{cw}]}\text{let } !^i x = !^w !^c M' \text{ in}$
		$\mid\ {}^{[!^i !^w !^w]}\text{let } !^i x = !^w({}^{\mathrm{i}^w}\mathcal{P}\, M') \text{ in}$
– with a $!^c$-prefixed bound term	${}^{\mathrm{i}^c}\mathcal{P}$	$::= \text{let } !^i y = !^c L' \text{ in}$
		$\mid\ {}^{[\beta !^i_{wc}]}\text{let } !^i x = !^c !^w M' \text{ in}$
		$\mid\ {}^{[!^i !^c !^w]}\text{let } !^i x = !^c({}^{\mathrm{i}^w}\mathcal{P}\, M') \text{ in}$
$!^c$-eliminator prefixes	${}^{\mathrm{c}}\mathcal{P}$	$::= \text{let } !^c x = M' \text{ in} \mid {}^{\mathrm{c}^c}\mathcal{P}$
		$\mid\ {}^{[\star !^c !^w{}_n]}\text{let } !^c x = (\mathcal{P}^{n-1}\, {}^{\mathrm{w}}\mathcal{P}\, M') \text{ in}$
		$\mid\ {}^{[\star !^c !^c{}_n]}\text{let } !^c x = (\mathcal{P}^{n-1}\, {}^{\mathrm{c}}\mathcal{P}\, M') \text{ in}$
		$\mid\ {}^{[\star !^c !^i{}_n]}\text{let } !^c x = (\mathcal{P}^{n-1}\, {}^{\mathrm{i}}\mathcal{P}\, M') \text{ in}$
– with a $!^c$-prefixed bound term	${}^{\mathrm{c}^c}\mathcal{P}$	$::= \text{let } !^c y = !^c L' \text{ in}$
		$\mid\ {}^{[\beta !^c]}\text{let } !^c x = !^c M' \text{ in}$
$!^w$-eliminator prefixes	${}^{\mathrm{w}}\mathcal{P}$	$::= \text{let } !^w x = M' \text{ in} \mid {}^{\mathrm{w}^w}\mathcal{P}$
		$\mid\ {}^{[\star !^w !^w{}_n]}\text{let } !^w x = (\mathcal{P}^{n-1}\, {}^{\mathrm{w}}\mathcal{P}\, M') \text{ in}$
		$\mid\ {}^{[\star !^w !^c{}_n]}\text{let } !^w x = (\mathcal{P}^{n-1}\, {}^{\mathrm{c}}\mathcal{P}\, M') \text{ in}$
		$\mid\ {}^{[\star !^w !^i{}_n]}\text{let } !^w x = (\mathcal{P}^{n-1}\, {}^{\mathrm{i}}\mathcal{P}\, M') \text{ in}$
– with a $!^w$-prefixed bound term	${}^{\mathrm{w}^w}\mathcal{P}$	$::= \text{let } !^w y = !^w L' \text{ in}$
		$\mid\ {}^{[\beta !^w]}\text{let } !^w x = !^w M' \text{ in}$

Figure 8.5: Syntax of the marked generalized separated-linear lambda calculus.

Main reduction rules

$$
\begin{array}{llcl}
(\beta\multimap_0) & {}^{[\beta\multimap]}(\lambda x.\, M')\, N' & \xrightarrow[\mathsf{SLin}^{\star\prime}]{} & M'[x := N'] \\
(\beta\multimap_1) & (\lambda x.\, M')\, N' & \xrightarrow[\mathsf{SLin}^{\star\prime}]{} & M'[x := N'] \\
(\beta!^c_0) & {}^{[\beta!^c]}\mathrm{let}\ !^c x = !^c M' \ \mathrm{in}\ N' & \xrightarrow[\mathsf{SLin}^{\star\prime}]{} & N'[x := M'] \\
(\beta!^c_1) & \mathrm{let}\ !^c x = !^c M' \ \mathrm{in}\ N' & \xrightarrow[\mathsf{SLin}^{\star\prime}]{} & N'[x := M'] \\
(\beta!^w_0) & {}^{[\beta!^w]}\mathrm{let}\ !^w x = !^w M' \ \mathrm{in}\ N' & \xrightarrow[\mathsf{SLin}^{\star\prime}]{} & N'[x := M'] \\
(\beta!^w_1) & \mathrm{let}\ !^w x = !^w M' \ \mathrm{in}\ N' & \xrightarrow[\mathsf{SLin}^{\star\prime}]{} & N'[x := M'] \\
(\beta!^i_{cw\,0}) & {}^{[\beta!^i_{cw}]}\mathrm{let}\ !^i x = !^w !^c M' \ \mathrm{in}\ N' & \xrightarrow[\mathsf{SLin}^{\star\prime}]{} & N'[x := M'] \\
(\beta!^i_{cw\,1}) & \mathrm{let}\ !^i x = !^w !^c M' \ \mathrm{in}\ N' & \xrightarrow[\mathsf{SLin}^{\star\prime}]{} & N'[x := M'] \\
(\beta!^i_{wc\,0}) & {}^{[\beta!^i_{wc}]}\mathrm{let}\ !^i x = !^c !^w M' \ \mathrm{in}\ N' & \xrightarrow[\mathsf{SLin}^{\star\prime}]{} & N'[x := M'] \\
(\beta!^i_{wc\,1}) & \mathrm{let}\ !^i x = !^c !^w M' \ \mathrm{in}\ N' & \xrightarrow[\mathsf{SLin}^{\star\prime}]{} & N'[x := M'] \\
(\mathrm{W}!^i_1) & \mathrm{let}\ !^i x = !^w M' \ \mathrm{in}\ N' & \xrightarrow[\mathsf{SLin}^{\star\prime}]{} & N',\ x \notin \mathrm{fv}(N')
\end{array}
$$

($!^\square\multimap$) rules

$$
\begin{array}{llcl}
(\star!^\square\multimap_0) & {}^{[\star!^\square\multimap_n]}(\mathcal{P}^{n-1}\, {}^\square\mathcal{P}\, M')\, N' & \xrightarrow[\mathsf{SLin}^{\star\prime}]{} & \mathcal{P}^{n-1}\, {}^\square\mathcal{P}\, (M'\ N') \\
(\star!^\square\multimap^<_0) & {}^{[\star!^\square\multimap_n]}(\mathcal{P}^{m}\, M')\, N' & \xrightarrow[\mathsf{SLin}^{\star\prime}]{} & \mathcal{P}^{m}\, ({}^{[\star!^\square\multimap_{n-m}]}M'\ N'), \\
 & & & 0 < m < n \\
(\star!^\square\multimap_1) & (\mathcal{P}^{n-1}\, {}^\square\mathcal{P}\, M')\, N' & \xrightarrow[\mathsf{SLin}^{\star\prime}]{} & \mathcal{P}^{n-1}\, {}^\square\mathcal{P}\, (M'\ N') \\
(\star!^\square\multimap^>_1) & {}^{[\star!^\square\multimap_n]}(\mathcal{P}^{n-1}\, {}^\square\mathcal{P}\, \mathcal{R}^m\, {}^\square\mathcal{R}\, M')\, N' & & \\
 & & \xrightarrow[\mathsf{SLin}^{\star\prime}]{} & \mathcal{P}^{n-1}\, {}^\square\mathcal{P}\, \mathcal{R}^m\, {}^\square\mathcal{R}\, (M'\ N'), \\
 & & & m \geq 0
\end{array}
$$

where $!^\square$ ranges over $!^w$, $!^c$, $!^i$
and correspondingly ${}^\square\mathcal{P}$ to ${}^{\mathrm{w}}\mathcal{P}$, ${}^{\mathrm{c}}\mathcal{P}$, ${}^{\mathrm{i}}\mathcal{P}$ (likewise ${}^\square\mathcal{R}$).

($!^\square!^\triangle$) rules

$$
\begin{array}{llcl}
(\star!^\square!^\triangle{}_0) & {}^{[\star!^\square!^\triangle{}_n]}\mathrm{let}\ !^\square x = (\mathcal{P}^{n-1}\, {}^\Delta\mathcal{P}\, M')\ \mathrm{in}\ N' & \xrightarrow[\mathsf{SLin}^{\star\prime}]{} & \mathcal{P}^{n-1}\, {}^\Delta\mathcal{P}\, (\mathrm{let}\ !^\square x = M'\ \mathrm{in}\ N') \\
(\star!^\square!^{\triangle<}_{\ 0}) & {}^{[\star!^\square!^\triangle{}_n]}\mathrm{let}\ !^\square x = (\mathcal{P}^{m}\, M')\ \mathrm{in}\ N' & & \\
 & & \xrightarrow[\mathsf{SLin}^{\star\prime}]{} & \mathcal{P}^{m}\, ({}^{[\star!^\square!^\triangle{}_{n-m}]}\mathrm{let}\ !^\square x = M', \\
 & & & \qquad\qquad \mathrm{in}\ N') \\
 & & & \text{where } 0 < m < n \\
(\star!^\square!^\triangle{}_1) & \mathrm{let}\ !^\square x = (\mathcal{P}^{n-1}\, {}^\Delta\mathcal{P}\, M')\ \mathrm{in}\ N' & \xrightarrow[\mathsf{SLin}^{\star\prime}]{} & \mathcal{P}^{n-1}\, {}^\Delta\mathcal{P}\, (\mathrm{let}\ !^\square x = M'\ \mathrm{in}\ N') \\
(\star!^\square!^{\triangle>}_{\ 1}) & {}^{[\star!^\square!^\triangle{}_n]}\mathrm{let}\ !^\square x = (\mathcal{P}^{n-1}\, {}^\Delta\mathcal{P}\, {}^\square\mathcal{R}^m\, \mathcal{R}\, M')\ \mathrm{in}\ N' & & \\
 & & \xrightarrow[\mathsf{SLin}^{\star\prime}]{} & \mathcal{P}^{n-1}\, {}^\Delta\mathcal{P}\, \mathcal{R}^m\, {}^\square\mathcal{R}\, (\mathrm{let}\ !^\square x = M'\ \mathrm{in}\ N'), \\
 & & & \text{where } m \geq 0
\end{array}
$$

where $!^\square$ and $!^\triangle$ each range over $!^w$, $!^c$, $!^i$
and correspondingly ${}^\square\mathcal{P}$ to ${}^{\mathrm{w}}\mathcal{P}$, ${}^{\mathrm{c}}\mathcal{P}$, ${}^{\mathrm{i}}\mathcal{P}$ (and ${}^\square\mathcal{R}$).

($!^i!^\square!^w$) rules

$$
\begin{array}{llcl}
(!^i!^\square!^w{}_0) & & & {}^{[\star!^i!^\square!^w{}_n]}\mathrm{let}\ !^i x = !^\square({}^{\mathrm{i}w}\mathcal{P}^{n}\, M')\ \mathrm{in}\ N' \\
 & & \xrightarrow[\mathsf{SLin}^{\star\prime}]{} & {}^{\mathrm{i}w}\mathcal{P}^{n}\, (\mathrm{let}\ !^i x = !^\square M'\ \mathrm{in}\ N') \\
(!^i!^\square!^{w<}_{\ 0}) & & & {}^{[\star!^i!^\square!^w{}_n]}\mathrm{let}\ !^i x = !^\square({}^{\mathrm{i}w}\mathcal{P}^{m}\, {}^{\mathrm{i}w}\mathcal{R}^{n-m}\, M')\ \mathrm{in}\ N' \\
 & & \xrightarrow[\mathsf{SLin}^{\star\prime}]{} & {}^{\mathrm{i}w}\mathcal{P}^{m}\, ({}^{[\star!^i!^\square!^w{}_{n-m}]}\, {}^{\mathrm{i}w}\mathcal{R}^{n-m}\ \mathrm{let}\ !^i x = !^\square M'\ \mathrm{in}\ N') \\
 & & & \text{where } 0 < m < n \\
(!^i!^\square!^w{}_1) & \mathrm{let}\ !^i x = !^\square({}^{\mathrm{i}w}\mathcal{P}^{n}\, M')\ \mathrm{in}\ N' & \xrightarrow[\mathsf{SLin}^{\star\prime}]{} & {}^{\mathrm{i}w}\mathcal{P}^{n}\, (\mathrm{let}\ !^i x = !^\square M'\ \mathrm{in}\ N') \\
(!^i!^\square!^{w>}_{\ 1}) & & & {}^{[\star!^i!^\square!^w{}_n]}\mathrm{let}\ !^i x = !^\square({}^{\mathrm{i}w}\mathcal{P}^{n}\, {}^{\mathrm{i}w}\mathcal{R}^{m}\, M')\ \mathrm{in}\ N' \\
 & & \xrightarrow[\mathsf{SLin}^{\star\prime}]{} & {}^{\mathrm{i}w}\mathcal{P}^{n}\, {}^{\mathrm{i}w}\mathcal{R}^{m}\, (\mathrm{let}\ !^i x = !^\square M'\ \mathrm{in}\ N') \\
 & & & \text{where } 0 < m
\end{array}
$$

where $!^\square$ ranges over $!^w$ and $!^c$

Figure 8.6: $(\beta, !^\square\multimap, !^i!^\square!^w)$ reduction rules for the marked generalized separated-linear lambda calculus.

Let $M' \xrightarrow[\mathsf{SLin}]{} N'$. Then:

- $\lambda x.\, M' \to \lambda x.\, N'$,
- $!^{w} M' \xrightarrow[\mathsf{SLin}]{} !^{w} N'$,
- $!^{c} M' \xrightarrow[\mathsf{SLin}]{} !^{c} N'$,
- $L'\ M' \xrightarrow[\mathsf{SLin}]{} L'\ N'$,
- $M'\ L' \xrightarrow[\mathsf{SLin}]{} N'\ L'$,
- ${}^{[\beta\multimap]}(\lambda x.\, L')\ M' \xrightarrow[\mathsf{SLin}]{} {}^{[\beta\multimap]}(\lambda x.\, L')\ N'$,
- ${}^{[\star !^{\Box}\multimap_n]}(\mathcal{P}^{n-1}\ {}^{\Box}\mathcal{P}\, L')\ M' \xrightarrow[\mathsf{SLin}]{} {}^{[\star !^{\Box}\multimap_n]}(\mathcal{P}^{n-1}\ {}^{\Box}\mathcal{P}\, L')\ N'$,
  where $!^{\Box}$ ranges over $!^{w}$, $!^{c}$, $!^{i}$ with corresponding ${}^{\Box}\mathcal{P}$.
- ${}^{[\star !^{\Box} !^{\triangle}{}_n]}\text{let } !^{\Box}x = (\mathcal{P}^{n-1}\ {}^{\Delta}\mathcal{P}\, L') \text{ in } M' \xrightarrow[\mathsf{SLin}]{} {}^{[\star !^{\Box} !^{\triangle}{}_n]}\text{let } !^{\Box}x = (\mathcal{P}^{n-1}\ {}^{\Delta}\mathcal{P}\, L') \text{ in } N'$,
  and ${}^{[\star !^{\Box} !^{\triangle}{}_n]}\text{let } !^{\Box}x = (\mathcal{P}^{n-1}\ {}^{\Delta}\mathcal{P}\, M') \text{ in } L' \xrightarrow[\mathsf{SLin}]{} {}^{[\star !^{\Box} !^{\triangle}{}_n]}\text{let } !^{\Box}x = (\mathcal{P}^{n-1}\ {}^{\Delta}\mathcal{P}\, N') \text{ in } L'$,
  where $!^{\Box}$ and $!^{\triangle}$ each range over $!^{w}$, $!^{c}$, $!^{i}$ with corresponding ${}^{\Delta}\mathcal{P}$.

Let $\sigma : \mathcal{P}^{n}\ {}^{\Box}\mathcal{P}\, M' \xrightarrow[\mathsf{SLin}]{} N'$, where $!^{\Box}$ and $!^{\triangle}$ each range over $!^{w}$, $!^{c}$, $!^{i}$ and with corresponding ${}^{\Delta}\mathcal{P}$:

- ${}^{[\star !^{\Box}\multimap_n]}(\mathcal{P}^{n}\ {}^{\Box}\mathcal{P}\, M')\ L' \xrightarrow[\mathsf{SLin}^{\star\prime}]{} {}^{[\star !^{\Box}\multimap_{n+\mathbf{d}(\sigma)}]}(N')\ L'$.
- ${}^{[\star !^{\triangle} !^{\Box}{}_n]}\text{let } !^{\triangle}x = (\mathcal{P}^{n}\ {}^{\Box}\mathcal{P}\, M') \text{ in } L' \xrightarrow[\mathsf{SLin}^{\star\prime}]{} {}^{[\star !^{\triangle} !^{\Box}{}_{n+\mathbf{d}(\sigma)}]}\text{let } !^{\triangle}x = N' \text{ in } L'$.

Let $\sigma : {}^{¡^{w}}\mathcal{P}^{n}\ M' \xrightarrow[\mathsf{SLin}]{} N'$ and let $!^{\Box}$ range over $!^{w}$, $!^{c}$:

- ${}^{[\star !^{i} !^{\Box} !^{w}{}_n]}\text{let } !^{i}x = ({}^{¡^{w}}\mathcal{P}^{n}\ M') \text{ in } L' \xrightarrow[\mathsf{SLin}^{\star\prime}]{} {}^{[\star !^{i} !^{\Box} !^{w}{}_{n+\mathbf{d}(\sigma)}]}\text{let } !^{i}x = N' \text{ in } L'$.

Figure 8.7: Compatible closure in $\mathsf{SLin}^{\star\prime}$.

Definition 8.3.2. Let $M' \in \mathsf{SLin}^{\star\prime}$; then we take $|M'|$ to be the $\mathsf{SLin}^{\star}$ term formed simply by erasing all markings in M'. ◊

Lemma 8.3.3. Let $\sigma' : M' \xrightarrow[\mathsf{SLin}^{\star\prime}]{} N'$. Then $|M'| \xrightarrow[\mathsf{SLin}^{\star}]{} |N'|$.

Proof: Trivial, since every rule of $\mathsf{SLin}^{\star}$ can be retrieved by simply erasing the markings. ♦

In the above lemma, we refer to the sequence $|M'| \xrightarrow[\mathsf{SLin}^{\star}]{} |N'|$ as $|\sigma'|$, and define projection of multistep reduction sequences accordingly as the concatenation of the projections of the single steps.

Lemma 8.3.4. Let $M' \in \mathsf{SLin}^{\star\prime}$ and $\sigma : |M'| \xrightarrow[\mathsf{SLin}^{\star}]{} \!\!\!\!\rightarrow N'$. Then there exists some $N' \in \mathsf{SLin}^{\star\prime}$, $N \equiv |N'|$, such that $\sigma' : M' \xrightarrow[\mathsf{SLin}^{\star\prime}]{} \!\!\!\!\rightarrow N'$ and $\sigma \equiv |\sigma'|$.

Proof: Again, immediate from the correspondence of the rules in the calculi. ♦

We also have the usual substitution results for $\mathsf{SLin}^{\star\prime}$.

Lemma 8.3.5. Let $L', M', N' \in \mathsf{SLin}^{\star\prime}$, with $M' \xrightarrow[\mathsf{SLin}^{\star\prime}]{} N'$. Then:

- $M'[x := L'] \xrightarrow[\mathsf{SLin}^{\star\prime}]{} N'[x := L']$.
- $L'[x := M'] \xrightarrow[\mathsf{SLin}^{\star\prime}]{} \!\!\!\!\rightarrow L'[x := N']$.

Proof: The first result is just compatible closure, and the second follows by induction on the structure of L'. ♦

It will be useful to have a more specific lemma about $\mathsf{SLin}^{\star\prime}_0$ reduction.

Lemma 8.3.6. Let $L', M', N' \in \mathsf{SLin}^{\star\prime}_0$, with $M' \xrightarrow[\mathsf{SLin}^{\star\prime}_0]{} N'$. Then:

- $M'[x := L'] \xrightarrow[\mathsf{SLin}^{\star\prime}_0]{} N'[x := L']$.

- $L'[x := M'] \xrightarrow[\mathsf{SLin}^{\star\prime}_0]{} L'[x := N']$.

Proof: Again the first result is by compatible closure, and the second by structural induction. ♦

As usual, there is a correspondence between marked terms and sets of references to redexes of the corresponding unmarked term. A *path* is a finite, possibly empty sequence of symbols referring to subterms of a term structure; a path is *valid* for a term if it actually corresponds to that term's structure. We use the same symbols for paths as in previous chapters, since all of the (say) eliminators have a common structure. Repeating the previous examples, in the term $M \equiv (\lambda x.\, M_0)\ M_1$, we take $\underline{@1}$ to mean the left subterm $(\lambda x.\, M_0)$, and $\underline{@2}$ the right subterm M_1 (or just $\underline{!1}$ or $\underline{\lambda 1}$ for terms with only one child), and so $\underline{@1}$ is a valid path referring to (indexing) the term $\lambda x.\, M_0$, $\underline{@1}\,\underline{\lambda 1}$ indexes the term M_0, and $\underline{@2}$ indexes the term M_1. In the term let $!^{\square}x = M$ in N, we have subterm $\underline{\ell 1}$ and $\underline{\ell 2}$. For a term M, we let $\mathsf{p}(M)$ be the set of valid paths into M, and let γ, δ range over paths. We write $\gamma \succcurlyeq \delta$ if there exists some δ_0 such that $\gamma \equiv \delta\delta_0$, and write $\gamma \succ \delta$ if $\delta_0 \equiv \epsilon$.

We index $(\beta, !^i!^{\square}!^w)$ redex by a single path, and $(!^{\square}\!\multimap, !^{\square}!^{\triangle})$ redexes by a pair of a path and a positive integer:

Definition 8.3.7. An *indexing set* $\bar{\gamma}$ for a term $M \in \mathsf{SLin}^{\star}$ is a subset of $\mathsf{p}(M) \cup (\mathsf{p}(M) \times \mathbb{N})$ where:

1. If $\gamma \in \bar{\gamma}$, then γ indexes a $(\beta, !^i!^{\square}!^w)$ redex in M.
2. If $(\gamma, n) \in \bar{\gamma}$, then γ indexes a $(!^{\square}\!\multimap, !^{\square}!^{\triangle})$ redex in M which moves n eliminator prefixes.
3. If $(\gamma, n_0), (\gamma, n_1) \in \bar{\gamma}$, then $n_0 = n_1$.

◊

It is clear from the definition of the rules that redexes specified by different rules will never coincide at the top-level: that is, if $(\gamma_0, n_0), \gamma_1 \in \bar{\gamma}$, then $\gamma_0 \not\equiv \gamma_1$. Clause (3) of the definition allows us to make the claim that indexing sets and marked terms are in one-to-one correspondence; without that restriction we would be able to mark certain terms as more than one redex. The following lemma is then clear:

Lemma 8.3.8. If $\bar{\gamma}$ be an indexing set for M, then there exists some marked term M' where $|M'| \equiv M$ and every redex indexed in $\bar{\gamma}$ is marked in M'. If M' is a marked term, then there exists an indexing set $\bar{\gamma}$ for $|M'|$ such that every redex marked in M' is indexed.

So it is sensible to write $(|M'|, \bar{\gamma}) \equiv M'$ for the appropriate indexing set, and we adopt this notation henceforward.

The restriction of Clause (3) also means that we cannot track every generalized redex in a term simultaneously, but since we are interested in $\mathsf{SLin}^{\star}$ only for the insight it allows into SLin, such a restriction is acceptable.

8.3.3 Developments and their finiteness

We define residuals and developments as usual. Let $\sigma' : (|M'|, \bar{\gamma}) \equiv M' \xrightarrow[\mathsf{SLin}^{\star\prime}]{} N'$; then there exists some $\bar{\delta}$ such that $N' \equiv (|N'|, \bar{\delta})$. Then we define $\bar{\delta}$ to be the *residual(s)* of the redexes in $\bar{\gamma}$ with respect to σ'. A *development* of a marked term is a reduction sequence which contracts only marked redexes, *viz.* a $\mathsf{SLin}^{\star\prime}_0$ sequence. A development is *finite* if it has a finite number of steps, and a finite development is *complete* if its final result is unmarked. So to show that all developments are finite (Corollary 8.3.17), we will show that $\mathsf{SLin}^{\star\prime}_0$ isstrongly normalizing (Lemma 8.3.16).

As usual, for strong normalization of $\mathsf{SLin}^{\star\prime}_0$, we use a decoration of the marked calculus with weights on variable occurrences. We refer to the weighted calculus as $\mathsf{SLin}^{\star\bullet}$ and present the norm $\|\,.\,\| : \mathsf{SLin}^{\star\bullet} \to \mathbb{N}$ in Figure 8.8. We take $\dot{M}, \dot{N}$ and so forth to range over $\mathsf{SLin}^{\star\bullet}$ terms. We take the same reduction rules in $\mathsf{SLin}^{\star\bullet}$ as in $\mathsf{SLin}^{\star\prime}$, as refer to $\mathsf{SLin}^{\star\bullet}_0$ as for $\mathsf{SLin}^{\star\prime}_0$. Once again, we have the usual substitution results.

Lemma 8.3.9. Let $\dot{L}, \dot{M}, \dot{N} \in \mathsf{SLin}^{\star\bullet}$, with $\dot{M} \xrightarrow[\mathsf{SLin}^{\star\bullet}]{} \dot{N}$. Then:

$$\begin{array}{rcl} \|x\|^i &=& i \\ \|\lambda x.\,\dot{M}\| &=& \|\dot{M}\| \\ \|\dot{M}\,\dot{N}\| &=& 2\|\dot{M}\| + 2\|\dot{N}\| \\ \|\dot{P}\,\dot{N}\| &=& 2\|\dot{P}\| + \|\dot{N}\| \\ \|\text{let } !^{\square}x = \dot{M} \text{ in}\| &=& \|\dot{M}\| \\ \|!^{\square}\dot{M}\| &=& \|\dot{M}\| \end{array}$$

Figure 8.8: The norm $\|\,.\,\|$ on $\mathsf{SLin}^{\star\bullet}$ terms.

- $\dot{M}[x := \dot{L}] \xrightarrow[\mathsf{SLin}^{\star\bullet}]{} \dot{N}[x := \dot{L}]$.
- $\dot{L}[x := \dot{M}] \xrightarrow[\mathsf{SLin}^{\star\bullet}]{} \!\!\!\!\!\rightarrow \dot{L}[x := \dot{N}]$.

Proof: The first result is again just compatible closure, and the second follows by induction on the structure of $\dot{L}$. ♦

Just as we can separate a marking from a term, we can separate weights from a marked term. Given a term $M \in \mathsf{SLin}^\star$ we define the variable-paths $\Xi(M)$ to be the set

$$\Xi(M) = \{\gamma \in \mathsf{p}(M) : M|_\gamma \equiv x,\ \text{any } x\}\ ,$$

that is, all paths indexing variables. Then a weighting of a marked term $M' \in \mathsf{SLin}^{\star\prime}$ is just any total function I from the variable-paths of $|M'|$ to integers:

$$I : \Xi(|M'|) \to \mathbb{N}\ .$$

The projection and lifting results for $\mathsf{SLin}^{\star\bullet}$ into $\mathsf{SLin}^{\star\prime}$, and by extension $\mathsf{SLin}^\star$ as well, are clear. We will need a lemma describing the interaction of substitution and norms for particular terms.

Lemma 8.3.10. Let $\dot{M}, \dot{N} \in \mathsf{SLin}^{\star\bullet}$ such that for all occurrences of x^i in $\dot{M}$ we have $\|\dot{N}\| < i$. Then $\|\dot{M}[x := \dot{N}]\| \leq \|\dot{M}\|$.

Proof: Trivially, by induction; equality arises only when x is not free in $\dot{M}$. ♦

We identify a particular class of weightings relevant to marked reduction.

Definition 8.3.11. A term $\dot{M} \in \mathsf{SLin}^{\star\bullet}$ has a *decreasing weighting* if it meets all of the following conditions:

1. For every subterm ${}^{[\beta\multimap]}(\lambda x.\,\dot{M}_0)\ \dot{M}_1$, and for all occurrences of x^i within $\dot{M}_0$, we have $\|\dot{M}_1\| < 1$.
2. For every subterm ${}^{[\beta!^\square]}\text{let } !^\square x = !^\square\dot{M}_0 \text{ in } \dot{M}_1$ where $!^\square$ ranges over $!^w, !^c$, and for all occurrences of x^i within $\dot{M}_1$, we have $\|\dot{M}_0\| < 1$.
3. For every subterm ${}^{[\beta!^i]}\text{let } !^i x = !^\square !^\triangle \dot{M}_0 \text{ in } \dot{M}_1$ with $!^\square$ and $!^\triangle$ dual, and for all occurrences of x^i within $\dot{M}_1$, we have $\|\dot{M}_0\| < 1$.

◊

As suggested by the name, the norm is decreased by $\mathsf{SLin}^{\star\bullet}_0$ reduction of terms with a decreasing weighting. Moreover, $\mathsf{SLin}^{\star\bullet}_0$-reducing a term will again have a decreasing weighting.

Lemma 8.3.12. Let $\dot{M} \xrightarrow[\mathsf{SLin}^{\star\bullet}_0]{} \dot{N}$, where $\dot{M}$ has a decreasing weighting. Then $\|\dot{M}\| > \|\dot{N}\|$.

Proof: By a structural induction down to the various possible top-level contractions. We show the base cases for the groups of redexes, and a representative of the trivial induction steps.

1. *Top level marked* $(\beta\multimap)$ *contraction.* Just as in the top-level (I) and (V) cases of Lemma 3.3.10. We have

$$\begin{array}{rcl} \dot{M} & \equiv & {}^{[\beta\multimap]}(\lambda x.\, \dot{M}_0)\ \dot{M}_1 \text{ and} \\ \dot{N} & \equiv & \dot{M}_0[x := \dot{M}_1]\ \ . \end{array}$$

By the definition of a decreasing weighting we have that the weighting of every occurrence of x^i in $\dot{M}_0$ is greater than $||\dot{M}_1||$; by Lemma 8.3.10 we also have $||\dot{M}_0[x := \dot{M}_1]|| \leq ||\dot{M}_0||$. Then the result follows with

$$\begin{array}{rl} & ||^{[\beta\multimap]}(\lambda x.\, \dot{M}_0)\ \dot{M}_1|| \\ = & 2||\dot{M}_0|| + 2||\dot{M}_1|| \\ > & ||\dot{M}_0|| \\ \geq & ||\dot{M}_0[x := \dot{M}_1]||\ \ . \end{array}$$

2. *Top level marked* $(\beta!^{\Box})$ *contractions,* $!^{\Box}$ *ranging over* $!^w, !^c$. We have $\dot{M} \equiv {}^{[\beta!^{\Box}]}\text{let } !^{\Box}x = !^{\Box}\dot{M}_0 \text{ in } \dot{M}_1$ and $\dot{N} \equiv \dot{M}_1[x := \dot{M}_0]$. We can use the definition of a decreasing weighting and Lemma 8.3.10 to show that $||\dot{M}_1[x := \dot{M}_0]|| \leq ||\dot{M}_1||$. Then similarly we have the result with

$$\begin{array}{rl} & ||^{[\beta!^{\Box}]}\text{let } !^{\Box}x = !^{\Box}\dot{M}_0 \text{ in } \dot{M}_1|| \\ = & 2||\dot{M}_0|| + ||\dot{M}_1|| \\ > & ||\dot{M}_1|| \\ \geq & ||\dot{M}_1[x := \dot{M}_0]||\ \ . \end{array}$$

3. *Top level marked* $(\beta!^i)$ *contractions.* We have

$$\dot{M} \equiv {}^{[\beta!^i]}\text{let } !^i x = !^{\Box}!^{\triangle}\dot{M}_0 \text{ in } \dot{M}_1$$

with $!^{\Box}, !^{\triangle}$ dual, and $\dot{N} \equiv \dot{M}_1[x := \dot{M}_0]$. Aside from the syntax, this case is just as in the previous one: by the definition of a decreasing weighting plus Lemma 8.3.10 we have $||\dot{M}_1[x := \dot{M}_0]|| \leq ||\dot{M}_1||$. Then similarly we have the result with

$$\begin{array}{rl} & ||^{[\beta!^i]}\text{let } !^i x = !^{\Box}!^{\triangle}\dot{M}_0 \text{ in } \dot{M}_1|| \\ = & 2||\dot{M}_0|| + ||\dot{M}_1|| \\ > & ||\dot{M}_1|| \\ \geq & ||\dot{M}_1[x := \dot{M}_0]||\ \ . \end{array}$$

4. *Top level marked* $(\star!^{\Box}\multimap)$ *contractions.* We consider the case of steps of rank 1; higher ranked steps can be seen as just multiple rank 1 steps.

$$\begin{array}{rl} & ||^{[\star!^{\Box}\multimap_1]}(\text{let } !^{\Box}x = \dot{L} \text{ in } \dot{M})\ \dot{N}|| \\ = & 4||\dot{L}|| + 2||\dot{M}|| + 2||\dot{N}|| \\ > & 2||\dot{L}|| + 2||\dot{M}|| + 2||\dot{N}|| \\ = & ||\text{let } !^{\Box}x = \dot{L} \text{ in } (\dot{M}\ \dot{N})||\ \ . \end{array}$$

5. *Top level marked* $(\star!^{\Box}!^{\triangle})$ *contraction.* Again considering rank 1 steps, this case is straightforward:

$$\begin{array}{rl} & ||^{[\star!^{\Box}!^{\triangle}_1]}\text{let } !^{\Box}x = \text{let } !^{\triangle}y = \dot{L} \text{ in } \dot{M} \text{ in } \dot{N}|| \\ = & 4||\dot{L}|| + 2||\dot{M}|| + ||\dot{N}|| \\ > & 2||\dot{L}|| + 2||\dot{M}|| + ||\dot{N}|| \\ = & ||\text{let } !^{\triangle}y = \dot{L} \text{ in let } !^{\Box}x = \dot{M} \text{ in } \dot{N}||\ \ . \end{array}$$

6. *Top level marked* $(\star!^i!^{\Box}!^w)$ *contraction.* We have a simple analysis just as in the case for $(\star!^{\Box}!^{\triangle})$ steps:

$$\begin{array}{rl} & ||^{[\star!^i!^{\Box}!^w{}_n]}\text{let } !^i x = !^{\Box}(\text{let } !^i y = !^w\dot{L} \text{ in } \dot{M}) \text{ in } \dot{N}|| \\ = & 4||\dot{L}|| + 2||\dot{M}|| + ||\dot{N}|| \\ > & 2||\dot{L}|| + 2||\dot{M}|| + ||\dot{N}|| \\ = & ||\text{let } !^i y = !^w\dot{L} \text{ in let } !^i x = !^{\Box}\dot{M} \text{ in } \dot{N}||\ \ , \end{array}$$

again with n-rank steps just as n 1-rank steps.

For non-top level reductions, the inductive step is simple since the norm of a term increases directly with the norm of its subcomponents. Considering application, let $\dot{L}\,\dot{M} \to \dot{L}\,\dot{N}$ with $||\dot{M}|| > ||\dot{N}||$, and we have

$$\begin{array}{rl} & ||\dot{L}\,\dot{M}|| \\ = & 2||\dot{L}|| + 2||\dot{M}|| \\ > & 2||\dot{L}|| + 2||\dot{N}|| \\ = & ||\dot{L}\,\dot{N}|| \ . \end{array}$$

The other cases are equally simple. ♦

The following argument occurs twice in the subsequent lemma, so we present it just once right here.

Lemma 8.3.13. Let $\dot{M}, \dot{N} \in \mathsf{SLin}^{\star\bullet}$ both have decreasing weighting where for all occurrences of x^i in $\dot{M}$ we have $i > ||\dot{N}||$. Then $\dot{M}[x := \dot{N}]$ has a decreasing weighting.

Proof: Let $\dot{L}$ be a marked β-redex in $\dot{M}[x := \dot{N}]$, so either $\dot{L} \equiv {}^{[\beta\multimap]}(\lambda y.\, \dot{L}_0)\, \dot{L}_1$, $\dot{L} \equiv {}^{[\beta!^{\square}]}\mathrm{let}\ !^{\square}y = !^{\square}\dot{L}_1 \ \mathrm{in}\ \dot{L}_0$ or $\dot{L} \equiv {}^{[\beta!^i]}\mathrm{let}\ !^i y = !^{\square}!^{\triangle}\dot{L}_1 \ \mathrm{in}\ \dot{L}_0$ for $!^{\square}, !^{\triangle}$ dual in the latter case. If the considered occurrence of $\dot{L}$ comes from $\dot{M}$, the result follows from Lemma 8.3.10: since $||\dot{L}_1[x := \dot{N}]|| < ||\dot{L}_1||$ by the hypothesis, the condition for decreasing weighting involving y continues to hold. If the considered occurrence of $\dot{L}$ comes from $\dot{N}$, then the result is trivial since it is unchanged. ♦

Lemma 8.3.14. Let $\dot{M} \xrightarrow[\mathsf{SLin}^{\star\bullet}_0]{} \dot{N}$, and let $\dot{M}$ have a decreasing weighting. Then $\dot{N}$ has a decreasing weighting.

This lemma has the same role and reasoning as Barendregt's proof of the similar result for call-by-name [9, Lemma 11.2.18.(ii)], although our system is somewhat more complicated due to the larger number of rules.

Proof: We first consider each of the four possible top-level reduction rules separately, and then compatible closure. In each of the first four cases, we show that every marked $(\beta\multimap, \beta!)$ redex in $\dot{N}$ again satisfies the criteria for a decreasing weighting.

1. *Marked top-level $(\beta\multimap)$ reduction.* So $\dot{M} \equiv {}^{[\beta\multimap]}(\lambda y.\, \dot{M}_0)\ \dot{M}_1$, and $\dot{N} \equiv \dot{M}_0[y := \dot{M}_1]$ has a decreasing weighting by Lemma 8.3.13.
2. *Marked top-level $(\beta!^{\square})$ reduction.* By the same reasoning as for marked top-level $(\beta\multimap)$ reduction.
3. *Marked top-level $(\star!^{\square}\multimap)$ reduction.* So $\dot{M} \equiv {}^{[!^{\square}\multimap]}(\dot{P}\,\dot{M}_0)\ \dot{M}_1$, and we again let $\dot{L}$ be a marked $(\beta\multimap, \beta!^{\square})$-redex in $\dot{N}$; again $\dot{L}$ is necessarily the residual of a marked redex $\dot{L}_0$ with decreasing weighting in $\dot{M}$. If $\dot{L}_0$ is a subterm of P, $\dot{M}_0$ or $\dot{M}_1$ then the result is trivial, since $\dot{L}$ is just the same as $\dot{L}_0$, just moved around within the term. If on the other hand $\dot{L}_0 \equiv (P\,\dot{M}_0)$, then we must also have $\dot{L}_0 \equiv {}^{[\beta!^{\square}]}\mathrm{let}\ !^{\square}y = !^{\square}\dot{L}_1 \ \mathrm{in}\ \dot{M}_0$ and so $\dot{L} \equiv {}^{[\beta!^{\square}]}\mathrm{let}\ !^{\square}y = !^{\square}\dot{L}_1 \ \mathrm{in}\ (\dot{M}_0\ \dot{M}_1)$; since y does not appear free in $\dot{M}_1$ the decreasing weighting condition on $\dot{L}$ is again satisfied.
4. *Marked top-level $(\star!^{\square}!^{\triangle}, \star!^{i}!^{\square}!^{w})$ reduction.* By a similar analysis as for marked top-level $(\star!^{\square}\multimap)$ reduction.

For compatible closure we proceed by structural induction on $\dot{M}$. The base case of a top-level redex is covered above. The inductive steps correspond to the compatible closure of the redex within non-contracted contexts; these steps are non-trivial only for enclosure within marked $(\beta\multimap, \beta!^{\square})$ redexes, which we consider presently.

1. *Marked top-level $(\beta\multimap)$ redexes.* This case is exactly as in the (I) and (V) cases of Lemma 3.3.11. We have ${}^{[\beta\multimap]}(\lambda x.\, \dot{M}_0)\ \dot{M}_1$, with two further cases depending on which subterm is contracted.
   (a) *Reduction $\dot{M}_0 \to \dot{N}_0$.* So $\dot{N} \equiv {}^{[\beta\multimap]}(\lambda x.\, \dot{N}_0)\ \dot{M}_1$. Let x^i occur in $\dot{N}_0$; then we must have x^i occurring in $\dot{M}_0$ as well since reduction will not create new weightings of a free variable, and by the initial conditions this $i > ||\dot{M}_1||$.

(b) *Reduction* $\dot{M}_1 \to \dot{N}_1$. So $\dot{N} \equiv {}^{[\beta\multimap]}(\lambda x.\, \dot{M}_0)\, \dot{N}_1$. Let x^i occur in $\dot{M}_0$; then by the initial decreasing weighting condition we have $i > \|\dot{M}_1\|$. Moreover by Lemma 8.3.12 we have that $\|\dot{M}_1\| > \|\dot{N}_1\|$, and so $i > \|\dot{N}_1\|$ and the weighting is again decreasing.

2. *Marked top-level* $(\beta!^{\Box})$ *redexes.* By similar reasoning as for marked top-level $(\beta\multimap)$ redexes.

The cases of commuting conversion reduction rules is clear, and the lemma follows. ♦

Lemma 8.3.15. Let $M' \in \mathsf{SLin}^{\star\prime}$. Then there exists a term $\dot{M} \in \mathsf{SLin}^{\star\bullet}$, $|\dot{M}| \equiv M'$, such that $\dot{M}$ has a decreasing weighting.

Proof: We can use the same numbering strategy as in Lemma 3.3.13, which is clearly applicable here since the terms and reduction rules have essentially the same structure. ♦

Lemma 8.3.16. $\mathsf{SLin}^{\star\prime}_0$ reduction is strongly normalizing.

Proof: By Lemma 8.3.15 every term $M' \in \mathsf{SLin}^{\star\prime}$ has a decreasing weighting $\dot{M}$, and so by Lemma 8.3.12 and Lemma 8.3.14 all $\mathsf{SLin}^{\star\prime}_0$-sequences from M' are of length at most $\|\dot{M}\|$. ♦

Corollary 8.3.17, $\mathrm{FD}(\mathsf{SLin}^{\star})$**.** All $\mathsf{SLin}^{\star}$-developments are finite.

Proof: Immediate, since developments and $\mathsf{SLin}^{\star\prime}_0$ reduction are the same thing. ♦

8.3.4 Unique completions

Lemma 8.3.18. $\mathsf{SLin}^{\star\prime}_0$ reduction is weakly confluent.

Proof: We take $M \xrightarrow[\mathsf{SLin}^{\star\prime}_0]{} M_0$, $M \xrightarrow[\mathsf{SLin}^{\star\prime}_0]{} M_1$ and construct the N such that

$$\begin{array}{lcl} M_0 & \xrightarrow[\mathsf{SLin}^{\star\prime}_0]{}\!\!\!\!\rightarrow & N \ , \text{ and} \\ M_1 & \xrightarrow[\mathsf{SLin}^{\star\prime}_0]{}\!\!\!\!\rightarrow & N \ . \end{array}$$

We consider first the case of a top-level redex.

- *Top-level* $(\beta\multimap_0, \beta!^{\Box}_0)$ *steps.* For these top-level steps the result is immediate by Lemma 8.3.6.
- *Top-level* $(\star!^{\Box}\multimap_0, \star!^{\Box}!^{\triangle}_0, \star!^{i}!^{\Box}!^{w}_0)$ *steps.* Just as in the corresponding cases of Lemma 5.3.14; the additional syntax makes no particular difference.

Cases with redexes in disjoint subterms and under compatible closure follow immediately. ♦

Corollary 8.3.19. $\mathsf{SLin}^{\star\prime}_0$ reduction is confluent.

Proof: Follows from Lemma 8.3.16 and Corollary 8.3.19. ♦

Proposition 8.3.20, $\mathrm{FD!}(\mathsf{SLin}^{\star})$**.** All $\mathsf{SLin}^{\star}$-developments are finite and can be extended to complete developments; all complete $\mathsf{SLin}^{\star}$-developments end in the same term.

Proof: Finiteness and extension follow from $\mathrm{FD}(\mathsf{SLin})$, and unique completion are by Lemma 2.1.4 with Corollary 8.3.19 since development of markings and $\mathsf{SLin}^{\star\prime}_0$ reduction are the same. ♦

8.4 Confluence

Joinability in SLin is just as in the previous calculi. Given two ranked redexes at the same subterm, we take the larger of the two ranks.

Lemma 8.4.1. All $\mathsf{SLin}^\star$-markings are joinable.

Proof: As for $\mathsf{Let}^\star$. ♦

Proposition 8.4.2. Reduction in SLin — excluding $(\mathrm{W}!^i)$ steps — is confluent.

Proof: By Lemma 2.1.12 with Proposition 8.3.20 and Lemma 8.4.1. ♦

Confluence follows from the technical developments of Section 8.3.2. We have from there in Proposition 8.4.2 that the subset of SLin which excludes only $(\mathrm{W}!^i)$ steps is confluent. The result for the full system follows from the following two lemmas:

Lemma 8.4.3. Reduction of SLin terms by $(\mathrm{W}!^i)$ steps is confluent.

Proof: Trivial, just as in Lemma 4.3.1: we can show by inspection that if

$$\begin{array}{ccl} M & \xrightarrow[(\mathrm{W}!^i)]{} & M_1 \text{ and} \\ M & \xrightarrow[(\mathrm{W}!^i)]{} & M_2 \ , \end{array}$$

then we have some N such that for both i, $M_i \xrightarrow[(\mathrm{W}!^i)]{}^{=} N$. The full result follows by induction. ♦

Lemma 8.4.4. Let $M, M_1, M_2 \in \mathsf{SLin}$, with $M \xrightarrow[(\mathrm{W}!^i)]{} M_1$ and $M \xrightarrow[\mathsf{SLin}]{} M_2$, with the latter not by a $(\mathrm{W}!^i)$ step. Then there exists some N such that $M_1 \xrightarrow[\mathsf{SLin}]{}^{=} N$ and $M_2 \xrightarrow[(\mathrm{W}!^i)]{}^{=} N$ where the former is not by a $(\mathrm{W}!^i)$ step (where the $^=$ indicates reflexive closure).

Proof: The result is clear in the style of Lemma 4.3.2. ♦

Proposition 8.4.5. SLin is confluent.

Proof: Follows from Proposition 8.4.2, Lemma 8.4.3 and Lemma 8.4.4. ♦

8.5 Standard evaluation

The notion of evaluation in SLin fulfills a prediction made by Jacobs [37]: that although the order of application of $!^c$ and $!^w$ may be indistinguishable in many sensible models, the terms $!^c!^wM$ and $!^w!^cM$ may reasonably have different operational interpretations[2]. The difference is in the evaluation of the $!^i$-eliminator, whose evaluation depends on the order in which the two prefixes are encountered. When the $!^c$ prefix is outermost, we reduce its subterm to find the expected $!^w$ prefix, and immediately substitute its body for the variable bound by the eliminator: thus the evaluation context forms

$$E ::= \cdots \mid \text{let } !^ix = E \text{ in } M \mid \text{let } !^ix = !^cE \text{ in } M \ .$$

Conversely, when the $!^w$ prefix is outermost, we delay any further evaluation of the subterm under the prefix, and instead proceed with the evaluation of the body of the eliminator. Only when the eliminator-bound variable is demanded do we resume evaluation under the $!^w$ prefix: this strategy is manifested in the evaluation contexts

$$\begin{array}{rl} E ::= \cdots & \mid \text{let } !^ix = E \text{ in } M \\ & \mid \text{let } !^ix = !^wM \text{ in } E \\ & \mid \text{let } !^ix = !^wE \text{ in } E[x] \ , \end{array}$$

which are reminiscent of Need evaluation.

[2] To be specific: Jacobs wrote that a term of the form $!^c!^wM$ would be "less efficient, because it involves doing nothing many times" than one of the form $!^w!^cM$, although here the operational interpretation of neither ordering of the prefixes seems necessarily more efficient than the other. In his development of the models, he first presents separate models where one operator distributes over the other but not vice-versa, and then describes a model where the two combinations are essentially equivalent.

Syntactic domains

$$
\begin{array}{lll}
\text{Evaluation contexts} & E & ::= [\,] \mid E\ M \\
& & \mid\ \text{let } !^c x = E \text{ in } M \mid \text{let } !^w x = E \text{ in } M \\
& & \mid\ \text{let } !^i x = E \text{ in } M \mid \text{let } !^i x = !^c E \text{ in } M \\
& & \mid\ \text{let } !^i x = !^w M \text{ in } E \mid \text{let } !^i x = !^w E \text{ in } E[x] \\
\text{Answers} & A & ::= \lambda x.\, M \mid !^w M \mid !^c M \mid \text{let } !^i x = !^w M \text{ in } A
\end{array}
$$

Reduction rules

$$
\begin{array}{llll}
(\beta\text{-}\!\circ) & (\lambda x.\, M)\ N & \xmapsto[\mathsf{SLin}]{} & M[x := N] \\
(\beta !^c) & \text{let } !^c x = !^c N \text{ in } M & \xmapsto[\mathsf{SLin}]{} & M[x := N] \\
(\beta !^w) & \text{let } !^w x = !^w N \text{ in } M & \xmapsto[\mathsf{SLin}]{} & M[x := N] \\
(\beta !^i_{cw}) & \text{let } !^i x = !^c !^w N \text{ in } M & \xmapsto[\mathsf{SLin}]{} & M[x := N] \\
(\beta !^i_{wc}) & \text{let } !^i x = !^w !^c N \text{ in } E[x] & \xmapsto[\mathsf{SLin}]{} & (E[x])[x := N]
\end{array}
$$

$$
(!^i\text{-}\!\circ) \quad (\text{let } !^i x = !^w M \text{ in } A)\ N \ \xmapsto[\mathsf{SLin}]{}\ \text{let } !^i x = !^w M \text{ in } (A\ N)
$$

$$
\begin{array}{ll}
(!^\square !^i) & \text{let } !^\square x = (\text{let } !^i y = !^w M \text{ in } A) \text{ in } N \\
& \qquad \xmapsto[\mathsf{SLin}]{}\ \text{let } !^i y = !^w M \text{ in } (\text{let } !^\square x = A \text{ in } N) \\
& \text{where } !^\square \text{ ranges over } !^w, !^c, !^i.
\end{array}
$$

$$
\begin{array}{ll}
(!^i !^w !^w) & \text{let } !^i x = !^w(\text{let } !^i y = !^w M \text{ in } A) \text{ in } E[x] \\
& \qquad \xmapsto[\mathsf{SLin}]{}\ \text{let } !^i y = !^w M \text{ in let } !^i x = !^w A \text{ in } E[x] \\
(!^i !^c !^w) & \text{let } !^i x = !^c(\text{let } !^i y = !^w M \text{ in } A) \text{ in } N \\
& \qquad \xmapsto[\mathsf{SLin}]{}\ \text{let } !^i y = !^w M \text{ in let } !^i x = !^c A \text{ in } N
\end{array}
$$

Figure 8.9: Evaluation in SLin.

Arguably, we ought to treat the $!^w$-eliminator is the same manner: allowing the $(\beta !^w)$ rule in evaluation only when the bound variable is demanded, and including an evaluation context of the form

$$E ::= \cdots \mid \text{let } !^i x = !^w M \text{ in } E \ .$$

Ultimately there is little difference between the two. Since such an x appears at most once in the body of the eliminator, we are not duplicating work when we apply $(\beta !^w)$: in either case, M would be evaluated only when x is demanded. Moreover, it will become clear that neither configuration would affect the translations of the intuitionistic systems into SLin. We treat the $!^c$ prefix and eliminator similarly: we seek permission to copy the bound term immediately, and once the term is reduced to the prefix, and we have that permission, duplicate immediately.

We have the notion of a demanded variable as in Need and Aff, and the usual, straightforward unique evaluation context lemma for SLin, although the larger language of terms means that this proof, as well as most of the proofs in this section, is notably more complicated.

Lemma 8.5.1. Let $M \in \mathsf{SLin}$. Then exactly one of the following is true:

1. M is an answer in SLin.

2. M demands some $x \in \text{fv}(M)$.

3. There exists a unique top-level standard SLin redex Δ, plus some unique evaluation context E such that $M \equiv E[\Delta]$.

Proof: By structural induction on M.

- $M \equiv x$. Clearly only Clause 2 is possible, with $E \equiv [\,]$.

- $M \equiv \lambda x.\, M_0$. All function are answers, so we have Clause 1 and clearly none other.

Syntactic domains As in SLin.

Reduction rules

$$
\begin{array}{lll}
(\beta{\multimap}) & (\lambda x.\, M)\; N & \xrightarrow[\mathsf{SLin}]{} \; M[x := N] \\
(\beta!^c) & \text{let } !^c x = !^c N \text{ in } M & \xrightarrow[\mathsf{SLin}]{} \; M[x := N] \\
(\beta!^w) & \text{let } !^w x = !^w N \text{ in } M & \xrightarrow[\mathsf{SLin}]{} \; M[x := N] \\
(\beta!^i_{cw}) & \text{let } !^i x = !^c !^w N \text{ in } M & \xrightarrow[\mathsf{SLin}]{} \; M[x := N] \\
(\beta!^i_{wc}) & \text{let } !^i x = !^w !^c N \text{ in } E[x] & \xrightarrow[\mathsf{SLin}]{} \; (E[x])[x := N] \\
 & & \\
(\star!^i{\multimap}) & ({}^{\mathrm{i}w}\mathcal{P}^n\; A)\; N & \xrightarrow[\mathsf{SLin}]{} \; {}^{\mathrm{i}w}\mathcal{P}^n\,(A\; N) \\
 & & \\
(\star!^\Box!^i) & \text{let } !^\Box x = ({}^{\mathrm{i}w}\mathcal{P}^n\; A) \text{ in } M & \xrightarrow[\mathsf{SLin}]{} \; {}^{\mathrm{i}w}\mathcal{P}^n\,(\text{let } !^\Box x = A \text{ in } M) \\
 & \quad \text{where } !^\Box \text{ ranges over } !^w,\, !^c,\, !^i. & \\
 & & \\
(!^i!^w!^w) & \text{let } !^i x = !^w({}^{\mathrm{i}w}\mathcal{P}\, A) \text{ in } E[x] & \xrightarrow[\mathsf{SLin}]{} \; {}^{\mathrm{i}w}\mathcal{P}\,(\text{let } !^i x = !^w A \text{ in } E[x]) \\
(!^i!^c!^w) & \text{let } !^i x = !^c({}^{\mathrm{i}w}\mathcal{P}\, A) \text{ in } N & \xrightarrow[\mathsf{SLin}]{} \; {}^{\mathrm{i}w}\mathcal{P}\,(\text{let } !^i x = !^c A \text{ in } N)
\end{array}
$$

Figure 8.10: Evaluation in $\mathsf{SLin}^\star$.

- $M \equiv M_0\; M_1$. Clause 1 does not apply; no application can be an answer. Applying the induction hypothesis on M_0, if Clause 1 applies for the subterm, then the top-level redex is a top-level $(\beta{\multimap}, !^i{\multimap})$ standard step. If Clause 2 or 3 applies for the subterm, then the same applies for the whole term as well.

- $M \equiv !^w M_0$. Then clearly only Clause 1 applies.

- $M \equiv (\text{let } !^w x = M_0 \text{ in } M_1)$. Trivially, M is not an answer; we consider M_0 according to the induction hypothesis. If Clause 1 holds for M_0, then depending on the two possible forms for a well-typed subterm we have a top-level standard $(\beta!^w, !^w!^i)$ step. Otherwise, whichever clause holds for M_0 holds for M as well.

- $M \equiv !^c M_0$ *or* $M \equiv (\text{let } !^c x = M_0 \text{ in } M_1)$. As in the previous two cases.

- $M \equiv (\text{let } !^i x = M_0 \text{ in } M_1)$. This final case is rather complicated, and is reminiscent of the final case in the proof of the unique evaluation context lemma for Need, Lemma 4.6.1. We consider M_0 according to the induction hypothesis. If Clause 2 or 3 applies to M_0, then the same applies to M. Otherwise if M_0 is an answer, we have an analysis of its exact form:

  - $M_0 \equiv !^w N_0$. This first case is the most complicated. Considering M_1 inductively, if M_1 demands x, then we must further consider the clause which applies for N_0. If N_0 is an answer, then M is a top-level standard $(\beta!^i, !^i!^w!^w)$ step, with the specific rule depending on the form of N_0. If N_0 demands a free variable or contains a standard redex, then the same applies to M.

    Otherwise, if M_1 does not demand x, then whatever rule applied for M_1 applies for M as well.

  - $M_0 \equiv !^c N_0$. If N_0 is an answer than we have M a top-level redex, and the specific rule is dictated by the form of N_0:

    * $N_0 \equiv !^w N_1$. Here, the $(\beta!^i)$ rule applies.
    * $N_0 \equiv (\text{let } !^i y = !^w N_1 \text{ in } A)$. In this case, we can apply the $(!^i!^c!^w)$ rule.

    Otherwise if Clause 2 or 3 applies to N_0, then the same applies to M.

  - $M_0 \equiv (\text{let } !^i y = !^w N_0 \text{ in } A)$. We have that M is a top-level standard $(!^i!^i)$ step.

No other form of M is possible, and the proof is complete. ♦

Figure 8.10 shows the notion of reduction for $\mathsf{SLin}^\star$, derived from evaluation in SLin just by moving to the corresponding $\star$-steps of arbitrary rank.

Lemma 8.5.2. Let $M, N \in \mathsf{SLin}/\mathsf{SLin}^\star$. Then $M \xrightarrow[\mathsf{SLin}]{}\!\!\!\!\!\rightarrow N$ if and only if

$$M \xrightarrow[\mathsf{SLin}^\star]{}\!\!\!\!\!\rightarrow N \ .$$

Proof: Clear from the definitions of the rules; we simply exchange each n-rank $\star$ step in $\mathsf{Need}^\star$ for n ordinary steps in Need. ♦

Lemma 8.5.3. Let $M \in \mathsf{SLin}/\mathsf{SLin}^\star$. Then exactly one of the following is true:

1. M is an answer in $\mathsf{SLin}/\mathsf{SLin}^\star$.
2. M demands some $x \in \mathrm{fv}(M)$.
3. There exists at least one top-level standard $\mathsf{SLin}^\star$ redex Δ, plus some unique evaluation context E such that $M \equiv E[\Delta]$.

Proof: As in Lemma 8.5.1. ♦

Lemma 8.5.4. Let A be a $\mathsf{SLin}/\mathsf{SLin}^\star$ answer. Then $A[x := M]$ is also an answer in $\mathsf{SLin}/\mathsf{SLin}^\star$.

Proof: Easy; free variables do not appear in the definition of answers, and so the result is an easy structural induction, observing that abstractions (respectively $!^w$-prefixed, $!^c$-prefixed terms) remain abstractions (respectively $!^w$-prefixed, $!^c$-prefixed terms). ♦

Lemma 8.5.5. Let $M, N \in \mathsf{SLin}^\star$ where $M[x := N]$ is a $\mathsf{SLin}/\mathsf{SLin}^\star$ answer and *either*

1. N is not an answer, or
2. M does not demand x,

or both. Then M is a $\mathsf{SLin}/\mathsf{SLin}^\star$ answer.

Proof: Clear from analysis of the particular M. ♦

Lemma 8.5.6. Let $M, N \in \mathsf{SLin}/\mathsf{SLin}^\star$, where M demands x. Then $M[y := N]$ demands x.

Proof: Clear by induction of the evaluation context E such that $M \equiv E[x]$. ♦

Lemma 8.5.7. Let $M, N \in \mathsf{SLin}^\star$ where $M[y := N]$ demands x and *either*

1. N does not demand x, or
2. M does not demand y,

or both. Then M demands x.

Proof: By analysis of M and the E where $E[x] \equiv M[y := N]$. ♦

Lemma 8.5.8. Let $M \xrightarrow[\mathsf{SLin}^\star]{} N$ be top-level. Then $M[x := L] \xrightarrow[\mathsf{SLin}^\star]{} N[x := L]$ is also top-level and standard.

Proof: Trivial, by inspection of the reduction rules. ♦

Lemma 8.5.9. Let $\sigma : M \underset{\mathsf{i}}{\rightarrow} N$ in $\mathsf{SLin}^\star$ with $z \in \mathrm{fv}(M, N)$. Then M demands z if and only if N demands z.

Proof: We take $M \equiv E[z]$ and $\sigma : M \equiv C[\Delta] \underset{\mathsf{i}}{\rightarrow} C[\Delta'] \equiv N$, and proceed with an analysis of C. If the top-level structure of C is that of a non-empty evaluation context (for example $C \equiv C_0\ M_0$), then we have the result by the induction hypothesis; if its top-level structure is not that of an evaluation context (for example $C \equiv M_0\ C_0$), then the result is immediate. Otherwise, for top-level σ we consider the specific rule which applies.

1. *We have* $M \equiv (\mathrm{let}\ !^i x = !^w !^c M_0 \text{ in } M_1) \xrightarrow[\mathsf{i}\text{-}(\beta!^i_{wc})]{} M_1[x := M_0] \equiv N$*, where* M_1 *does not demand* x. We must have M_1 demanding z, since no subterm of M_0 could be in evaluation position; the result then follows from Lemma 8.5.6 and Lemma 8.5.7.

2. *We have* $M \equiv (\text{let } !^i x = !^w M_0 \text{ in } M_1) \xrightarrow[\text{i-(W}!^i)]{} M_1 \equiv N$ *where* $x \notin \text{fv}(M_1)$. Trivially, if $x \notin \text{fv}(M_1)$ then it must be that M_1 demands z.

3. $M \equiv ({}^{i^w}\!\mathcal{P}^n\ M_1)\ M_2 \xrightarrow[\text{i-}(\star !^i \multimap)]{} {}^{i^w}\!\mathcal{P}^n\ (M_1\ M_2) \equiv N$ *where* M_1 *is not an answer.* We have two possibilities. If M_1 demands z, then the result is immediate. Otherwise, taking

$$ {}^{i^w}\!\mathcal{P}_j \equiv \text{let } !^i x_j = !^w L_j \text{ in } \ , $$

we have a strictly decreasing sequence a_k, $1 \leq k \leq m$, $1 \leq a_k \leq n$, such that M_1 demands $x_{(a_1)}$, $L_{(a_j)}$ demands $x_{(a_{j+1})}$ and $L_{(m)}$ demands z. This arrangement would clearly be preserved from one side of the reduction arrow to the other.

4. $M \equiv (\mathcal{R}^n\ {}^{i^w}\!\mathcal{P}\ M_1)\ M_2 \xrightarrow[\text{i-}(\star !^i \multimap)]{} \mathcal{R}^n\ {}^{i^w}\!\mathcal{P}\ (M_1\ M_2) \equiv N$ *where not all of the* $\mathcal{R}$ *are of the form* ${}^{i^w}\!\mathcal{P}$. This case is similar to the previous one. We must have some $0 \leq m < n$ such that the $\mathcal{R}^n$ can be partitioned as

$$ \mathcal{R}^n \equiv {}^{i^w}\!\mathcal{R}^m\ \mathcal{R}_{m+1}\ \mathcal{R}^{n-m-1} \ , $$

where $\mathcal{R}_{m+1}$ is *not* of the form ${}^{i^w}\!\mathcal{R}_{m+1}$. Hence, its bound term L is in evaluation position. Then either L demands z, or we have a series a_k among the bound variables and terms of ${}^{i^w}\!\mathcal{R}^m$ as in the previous case which end in a bound term demanding z.

5. *We have*

$$ \begin{array}{rcl} M & \equiv & (\mathcal{P}^n \text{ let } !^\square x = M_0 \text{ in } M_1)\ M_2 \\ & \xrightarrow[\text{i-}(!^\square \multimap)]{} & \mathcal{P}^n\ (\text{let } !^\square x = M_0 \text{ in } M_1\ M_2) \\ & \equiv & N \ , \end{array} $$

where $!^\square$ *ranges over* $!^w, !^c$. As in Case 4.

6. *We have*

$$ \begin{array}{rcl} M & \equiv & \text{let } !^\square x = ({}^{i^w}\!\mathcal{P}^n\ M_1) \text{ in } M_2 \\ & \xrightarrow[\text{i-}(\star !^\square !^i)]{} & {}^{i^w}\!\mathcal{P}^n\ (\text{let } !^\square x = M_1 \text{ in } M_2) \\ & \equiv & N \ , \end{array} $$

where M_1 *is not an answer.* As in Case 3, since M_1 must itself be demanded.

7. *We have*

$$ \begin{array}{rcl} M & \equiv & \text{let } !^\square x = (\mathcal{P}^n\ M_1) \text{ in } M_2 \\ & \xrightarrow[\text{i-}(\star !^\square !^\triangle)]{} & \mathcal{P}^n\ (\text{let } !^\square x = M_1 \text{ in } M_2) \\ & \equiv & N \ , \end{array} $$

where not all $\mathcal{P}$ *are of the form* ${}^{i^w}\!\mathcal{P}$. As in Case 4.

8. *We have*

$$ \begin{array}{rcl} M & \equiv & \text{let } !^i x = !^c({}^{i^w}\!\mathcal{P}^n\ M_1) \text{ in } M_2 \\ & \xrightarrow[\text{i-}(\star !^i !^c !^w)]{} & {}^{i^w}\!\mathcal{P}^n\ (\text{let } !^i x = !^c M_1 \text{ in } M_2) \equiv N \ . \end{array} $$

This case is reminiscent of Case 6. We have either M_2 demanding z, or else some chain of demand as in Case 3.

9. *We have*

$$ \begin{array}{rcl} M & \equiv & \text{let } !^i x = !^w({}^{i^w}\!\mathcal{P}^n\ M_2) \text{ in } M_3 \\ & \xrightarrow[\text{i-}(\star !^i !^w !^w)]{} & {}^{i^w}\!\mathcal{P}^n\ (\text{let } !^i x = !^w M_2 \text{ in } M_3) \\ & \equiv & N \ , \end{array} $$

where M_2 *is not an answer.* We have either M_2 demanding z, or else M_2 demanding x with one of the arrangements of the previous case.

10. *We have*

$$\begin{array}{rcl} M & \equiv & \text{let } !^i x = !^w ({}^{\mathrm{i}^w}\mathcal{P}^n\, A) \text{ in } M \\ & \xrightarrow[\texttt{i-}(\star !^i !^w !^w)]{} & {}^{\mathrm{i}^w}\mathcal{P}^n\, (\text{let } !^i x = !^w A \text{ in } M) \\ & \equiv & N \end{array}$$

where M does not demand x. As before, except that the only possible arrangement is that M demands z.

Finally, it is not possible for a top-level $(\beta\multimap, \beta!^c, \beta!^w, \beta!^i_{cw})$ step to be internal, so the proof is complete. ♦

We follow the usual technique for the standardization result.

Lemma 8.5.10. *Reduction in* $\mathsf{SLin}^\star$ *satisfies **P0**:* All $\mathsf{SLin}^\star$-developments are finite and can be extended to a complete development, all complete $\mathsf{SLin}^\star$-developments of the same term and marking end in the same term, and $\mathsf{SLin}^\star$ has joinable markings with joining operator $\Cup$.

Proof: From FD!($\mathsf{SLin}^\star$) and Lemma 8.4.1. ♦

Lemma 8.5.11. *Reduction in* $\mathsf{SLin}^\star$ *satisfies **P1**:* Let $\sigma : M \underset{\mathrm{i}}{\twoheadrightarrow} N$ in $\mathsf{SLin}^\star$. Then M is an answer if and only if N is an answer.

Proof: By structural induction on M.

- *We have* $M \equiv (\text{let } !^i x = !^w !^c M_0 \text{ in } M_1) \xrightarrow[\texttt{i-}(\beta !^i_{wc})]{} M_1[x := M_0] \equiv N$, *where M_1 does not demand x.* M is an answer if and only if M_1 is an answer. Since M_1 does not demand x, by Lemma 8.5.4 and Lemma 8.5.5 we have that M_1 is an answer if and only if $M_1[x := M_0] \equiv N$ is an answer.

- *We have* $M \equiv (\text{let } !^i x = !^w M_0 \text{ in } M_1) \xrightarrow[\texttt{i-}(\mathrm{W}!^i)]{} M_1 \equiv N$ *where* $x \notin \mathrm{fv}(M_1)$. Clearly both sides are answers exactly when M_1 is an answer.

- *We have* $M \equiv (\mathcal{P}^n\, M_1)\, M_2 \xrightarrow[\texttt{i-}(\star !^\square \multimap)]{} \mathcal{P}^n\, (M_1\, M_2) \equiv N$ *where M_1 is not an answer.* Trivially, neither side is an answer.

- *We have*

$$\begin{array}{rcl} M & \equiv & \text{let } !^\square x = (\mathcal{P}^n\, M_1) \text{ in } M_2 \\ & \xrightarrow[\texttt{i-}(\star !^\square !^\triangle)]{} & \mathcal{P}^n\, (\text{let } !^\square x = M_1 \text{ in } M_2) \\ & \equiv & N\ , \end{array}$$

  where M_1 is not an answer. Again, neither side is an answer, since even if $!^\square$ were $!^i$, we would still not have its bound term on either side under a $!^w$ prefix.

- *We have*

$$\begin{array}{rcl} M & \equiv & \text{let } !^i x = !^\square ({}^{\mathrm{i}^w}\mathcal{P}^n\, M_1) \text{ in } M_2 \\ & \xrightarrow[\texttt{i-}(\star !^i !^\square !^w)]{} & {}^{\mathrm{i}^w}\mathcal{P}^n\, (\text{let } !^i x = !^\square M_1 \text{ in } M_2) \\ & \equiv & N \end{array}$$

  where $!^\square$ ranges over $!^w, !^c$ and either M_1 is not an answer, M_2 does not demand x, or both. If $!^\square$ is $!^c$ then the neither side is an answer. Otherwise both sides are clearly answers exactly when M_2 is an answer.

We have compatible closure by the induction hypothesis and definition of answer, and so the proof is complete. ♦

Lemma 8.5.12. *Reduction in* $\mathsf{SLin}^\star$ *satisfies **P2**:* Let γ, δ mark distinct internal $\mathsf{SLin}^\star$ redexes in M, $\sigma : M \xrightarrow[(\gamma)]{} N$. Then every member of δ/σ is internal.

Proof: Again, by induction on the structure of M. If $\gamma \equiv \delta$ then there are no residuals, and the result is vacuously true, so we assume from now on that the two are different in some way, *i.e.* in rank if not position. We begin with the case where γ is top-level, excluding as usual those rules of which top-level steps cannot be internal, and excluding as well $(\mathrm{W}!^i)$ steps which cannot be marked.

- *We have* $M \equiv (\text{let } !^i x = !^w !^c M_0 \text{ in } M_1) \xrightarrow[\text{i-}(\beta!^i_{wc})]{} M_1[x := M_0] \equiv N$, *where* M_1 *does not demand* x. If δ indexes a redex in M_1, then the residual is just δ again, which clearly remains internal: if it is within a non-evaluation context at first, it will continue to be so under the substitution of a non-demanded variable. For δ indexing a term is M_0 the result is again clear; if M_1 does not demand x, then it does not demand the replacement for x.

- *We have* $M \equiv ({}^{\text{i}^w}\!\mathcal{P}^n\, M_1)\, M_2 \xrightarrow[\text{i-}(\star!^i\multimap)]{} {}^{\text{i}^w}\!\mathcal{P}^n\, (M_1\, M_2) \equiv N$ *where* M_1 *is not an answer.* A variety of cases arise; at the risk of being pedantic we catalog the possibilities by considering the explicit structure of the path δ.

  - $\delta \equiv \epsilon$. So δ is a $(\star!^\square\multimap)$ step of different rank m than γ but is also top-level. We compare m and n: if $m < n$, then δ/σ is empty. If $m > n$, then the single residual is the $(\star!^\square\multimap)$ in context $({}^{\text{i}^w}\!\mathcal{P}^n\, [\,])$ with rank $(m - n)$. This step is clearly internal if and only if γ is internal as well.
  - $\delta \succcurlyeq \underline{@1}$. We distinguish three cases, respectively, reduction of a bound term, top-level reduction of a subterm $({}^{\text{i}^w}\!\mathcal{P}_m \cdots {}^{\text{i}^w}\!\mathcal{P}_n\, M_1)$, and reduction within M_1.
    * $\delta \equiv \underline{@1}(\underline{\ell 2}^m)\underline{\ell 1}\underline{!1}\delta_0$, $m < n$, *i.e.* reduction of the bound term L in ${}^{\text{i}^w}\!\mathcal{P}_{m+1} \equiv (\text{let } !^i x = !^w L \text{ in})$. Since this redex is internal in M, either δ_0 is internal to L or if M_1 does not demand x. Whichever condition is true in M would continue to be true in N, so the residual will be internal as well.
    * $\delta \equiv \underline{@1}(\underline{\ell 2}^m), m < n$. We have two possibilities, either a $(\star!^i!^w!^w)$ redex or a $(\beta!^i_{wc})$ redex. Taking $L_2 \equiv {}^{\text{i}^w}\!\mathcal{P}_{m+1} \cdots {}^{\text{i}^w}\!\mathcal{P}_n\, M_1$, in the first case we have
    $$({}^{\text{i}^w}\!\mathcal{P}_m \cdots {}^{\text{i}^w}\!\mathcal{P}_n\, M_1) \equiv \text{let } !^i x_m = !^w({}^{\text{i}^w}\!\mathcal{R}\, L_1) \text{ in } L_2 \ \ ,$$
    Since this redex is internal, we must have either that L_2 does not demand x, or that L_1 is an answer, or both. L_2 demands x if and only if ${}^{\text{i}^w}\!\mathcal{P}_{m+1} \cdots {}^{\text{i}^w}\!\mathcal{P}_n\, (M_1\, M_2)$ demands x, so the residual is internal as well. For a $(\beta!^i_{wc})$ step, we have
    $$({}^{\text{i}^w}\!\mathcal{P}_m \cdots {}^{\text{i}^w}\!\mathcal{P}_n\, M_1) \equiv \text{let } !^i x_m = !^w !^c L_1 \text{ in } L_2 \ \ ,$$
    which if internal means L_2 does not demand x, so again neither does ${}^{\text{i}^w}\!\mathcal{P}_{m+1} \cdots {}^{\text{i}^w}\!\mathcal{P}_n\, (M_1\ M_2)$, and so the residual is internal.
    * $\delta \equiv \underline{@1}(\underline{\ell 2}^n)\delta_0$. Since $(({}^{\text{i}^w}\!\mathcal{P}^n\, [\,])\ M_2)$ is an evaluation context, δ_0 must be internal to M_1, and so the residual must be internal as well.
  - $\delta \succcurlyeq \underline{@2}$. Contraction within M_2 in both M and N is clearly internal.

- *We have* $M \equiv (\mathcal{R}^n\, {}^{\text{i}^w}\!\mathcal{P}\, M_1)\ M_2 \xrightarrow[\text{i-}(\star!^i\multimap)]{} \mathcal{R}^n\, {}^{\text{i}^w}\!\mathcal{P}\, (M_1\ M_2) \equiv N$ *where not all of the* $\mathcal{R}$ *are of the form* ${}^{\text{i}^w}\!\mathcal{P}$.

- *We have*
$$\begin{array}{rcl} M & \equiv & (\mathcal{P}^n \text{ let } !^\square x = M_0 \text{ in } M_1)\ M_2 \\ & \xrightarrow[\text{i-}(\star!^\square\multimap)]{} & \mathcal{P}^n\,(\text{let } !^\square x = M_0 \text{ in } M_1\ M_2) \\ & \equiv & N \end{array}$$
where $!^\square$ *ranges over* $!^w, !^c$.

  These two cases rely on exactly the same reasoning as in the first $(!^i\multimap)$ case. We must only distinguish the different $!^\square$ eliminator prefixes since their evaluation behavior, and thus the internal redexes which they admit, differ from a $!^i$-eliminator with an $!^w$-prefixed argument. It is clear though that the criteria for these redexes to be internal is preserved in both M and N.

- *We have*
$$\begin{array}{rcl} M & \equiv & \text{let } !^\square x = ({}^{\text{i}^w}\!\mathcal{P}^n\, M_1) \text{ in } M_2 \\ & \xrightarrow[\text{i-}(\star!^\square!^i)]{} & {}^{\text{i}^w}\!\mathcal{P}^n\,(\text{let } !^\square x = M_1 \text{ in } M_2) \\ & \equiv & N\ \ , \end{array}$$
where M_1 *is not an answer.* We again analyze the structure of δ.

- $\delta \equiv \epsilon$. As in the $(!^{\Box}\!\multimap)$ cases. A $(!^{\Box}!^{i})$ redex of lesser rank would have no residuals, while a $(!^{\Box}!^{\triangle})$ redex of greater rank would clearly remain internal: since both δ and γ are internal in M, either M_1 is not an answer, M_2 does not demand x, or both. Each condition would hold in N as in M, and hence the residual of δ would be internal.
- $\delta \succcurlyeq \underline{\ell 1}$.
  - $\delta \succcurlyeq \underline{\ell 1}(\underline{\ell 2}^m)\underline{\ell 1}$, $m < n$.
  - $\delta \equiv \underline{\ell 1}(\underline{\ell 2}^m)$, $m < n$. Both of these cases are as in the corresponding case for a $(!^{\Box}\!\multimap)$ step.
  - $\delta \equiv \underline{\ell 1}(\underline{\ell 2}^m)\delta_0$. So δ_0 indexes an internal redex of M_1, which clearly makes the residual $(\underline{\ell 2}^n)\underline{\ell 1}\delta_0$ of δ again internal.
- $\delta \succcurlyeq \underline{\ell 2}$. Since M_1 is not an answer, contractions of M_2 in N are clearly internal.

• *We have*

$$\begin{array}{ccl} M & \equiv & \text{let } !^{\Box}x = (\mathcal{P}^n\; M_1) \text{ in } M_2 \\ & \xrightarrow[\mathtt{i}\text{-}(\star!^{\Box}!^{\triangle})]{} & \mathcal{P}^n\; (\text{let } !^{\Box}x = M_1 \text{ in } M_2) \\ & \equiv & N \end{array}$$

where not all $\mathcal{P}$ *are of the form* ${}^{\mathtt{i}^w}\mathcal{P}$. This case is again like the first case of internal $(\star!^{\Box}!^{i})$ reduction, noting variations in the criteria for internal contraction, which again γ preserves.

• *We have*

$$\begin{array}{ccl} M & \equiv & (\text{let } !^{i}x = !^{\Box}(\text{let } !^{i}y = !^{w}M_1 \text{ in } M_2) \text{ in } M_3) \\ & \xrightarrow[\mathtt{i}\text{-}(!^{i}!^{\Box}!^{w})]{} & (\text{let } !^{i}y = !^{w}M_1 \text{ in } (\text{let } !^{i}x = !^{\Box}M_2 \text{ in } M_3)) \\ & \equiv & N \end{array}$$

where $!^{\Box}$ *ranges over* $!^{w}, !^{c}$.

- $\delta \equiv \epsilon$. As for $(\star!^{\Box}!^{\triangle})$ steps of the same location but different rank.
- $\delta \succcurlyeq \underline{\ell 1}$. Since $!^{\Box}$-prefixed expressions cannot be top-level redexes, we in fact have $\delta \succcurlyeq \underline{\ell 1}\,!\underline{1}$.
  - $\delta \equiv \underline{\ell 1}\,!\underline{1}$. As in the $(\star!^{i}\!\multimap)$ case, we have either a $(\beta!^{i}_{wc})$ or $(\star!^{i}!^{w}!^{w})$ step, which is clearly again internal after contraction of γ.
  - $\delta \equiv \underline{\ell 1}\,!\underline{1}$. This step is internal either if δ_1 is itself internal in the bound term, or if the bound variable is not demanded, or both. The two conditions are clearly both preserved by contracting γ, so the residual is again internal.
  - $\delta \succcurlyeq \underline{\ell 1}\,!\underline{1}$. If M_2 demands x, then δ_1 must index an internal redex in M_1; otherwise if M_2 does not demand x the residual is trivially internal.
- $\delta \succcurlyeq \underline{\ell 2}$. Trivially, δ_1 lust be internal in M_2, and so the residual of δ will be internal as well in N.

It is also possible that δ will be top-level when σ is not. In that case we have the result by a similar inspection of the individual cases; by the preceding Lemma 8.5.9 and Lemma 8.5.11 on the effect of internal contractions on demand and answerhood, it is clear that the forms of subterms required for δ to be internal are upheld by contraction of the redex at γ. For δ, γ in distinct subterms it again follows from Lemma 8.5.9 and Lemma 8.5.11 that if δ had been in a non-evaluation context, contracting γ would not cause the context to become an evaluation context. Finally, for δ, γ in the same subterm the result is immediate by the induction hypothesis. ♦

Lemma 8.5.13. *Reduction in* $\mathsf{SLin}^{\star}$ *satisfies* ***P3****:* Let $\sigma : M_0 \xrightarrow[\mathtt{i}]{} N_0$ and let $N_0 \mapsto N_1$, both in $\mathsf{SLin}^{\star}$. Then M_0 has a standard redex: there exists some M_1 such that $M_0 \mapsto M_1$.

Proof: So we have:

$$\begin{aligned}\sigma : M_0 \equiv C_0[\Delta_0] &\underset{\mathrm{i}}{\rightarrow} C_0[\Delta_0'] \equiv N_0 \\ N_0 \equiv E_1[\Delta_1] &\mapsto E_1[\Delta_1'] \equiv N_1 \ ,\end{aligned}$$

and we seek to construct or otherwise show the existence of the E, Δ such that $M_0 \equiv E[\Delta]$ and Δ is a standard step. We proceed by structural induction on M_0, and perform an analysis on C_0 and E_1.

We begin as usual where $C_0 \equiv [\,]$, excluding as usual the steps which cannot be both top-level and internal.

- *We have* $M_0 \equiv (\text{let } !^i x = !^w !^c M_1 \text{ in } M_2) \xrightarrow[\text{i-}(\beta !^i_{wc})]{} M_2[x := M_1] \equiv N_0$, *where* M_2 *does not demand* x. Since σ is internal, we must have that M_2 does not demand x. By Lemma 8.5.5, since $M_2[x := M_1]$ has a standard redex, we know that M_2 is not an answer, and similarly by Lemma 8.5.7 it could not demand any other variable, so by Lemma 8.5.3 M_2 must have a standard redex, which would correspond to the contraction in N_0.

- *We have* $M_0 \equiv (\text{let } !^i x = !^w M_1 \text{ in } M_2) \xrightarrow[\text{i-}(\mathrm{W}!^i)]{} M_2 \equiv N_0$ *where* $x \notin \mathrm{fv}(M_2)$. Trivially, $\Delta \equiv \Delta_1$ and $E \equiv \text{let } !^i x = !^w M_1 \text{ in } E_1$.

- *We have* $M_0 \equiv ({}^{\mathrm{i}^w}\mathcal{P}^n\, M_1)\, M_2 \xrightarrow[\text{i-}(\star !^i \multimap)]{} {}^{\mathrm{i}^w}\mathcal{P}^n\, (M_1\, M_2) \equiv N_0$ *where* M_1 *is not an answer.* $\Delta \equiv \Delta_1$, and E depends on E_1:

  - *We have* $E_1 \equiv {}^{\mathrm{i}^w}\mathcal{P}^n\, (E_2\, M_2)$. Then $E \equiv ({}^{\mathrm{i}^w}\mathcal{P}^n\, E_2)\, M_2$.
  - *We have*

    $$E_1 \equiv {}^{\mathrm{i}^w}\mathcal{P}^{m-1}\, \Big(\text{let } !^i x_m = !^w E_2 \text{ in } {}^{\mathrm{i}^w}\mathcal{P}^n_{m+1}\, (M_1\, M_2)\Big) \ ,$$

    where ${}^{\mathrm{i}^w}\mathcal{P}^n_{m+1}\, (M_1\, M_2)$ *demands* x_m. Then $E \equiv ({}^{\mathrm{i}^w}\mathcal{P}^{m-1}\, \text{let } !^i x_m = !^w E_2 \text{ in } {}^{\mathrm{i}^w}\mathcal{P}^n_{m+1}\, M_1)\, M_2$.

- *We have* $M_0 \equiv (\mathcal{R}^n\, {}^{\mathrm{i}^w}\mathcal{P}\, M_1)\, M_2 \xrightarrow[\text{i-}(\star !^i \multimap)]{} \mathcal{R}^n\, {}^{\mathrm{i}^w}\mathcal{P}\, (M_1\, M_2) \equiv N_0$ *where not all of the* $\mathcal{R}$ *are of the form* ${}^{\mathrm{i}^w}\mathcal{P}$, and

- *We have*

  $$\begin{aligned} M_0 \quad &\equiv \quad (\mathcal{P}^n\, \text{let } !^\square x = M_0 \text{ in } M_1)\, M_2 \\ &\xrightarrow[\text{i-}(!^\square \multimap)]{} \quad \mathcal{P}^n\, (\text{let } !^\square x = M_0 \text{ in } M_1\, M_2) \\ &\equiv \quad N_0 \end{aligned}$$

  where $!^\square$ *ranges over* $!^w, !^c$.

  Both of these cases are as in the first $(!^i \multimap)$ case, the difference being that for the $\mathcal{R}$ not of the form ${}^{\mathrm{i}^w}\mathcal{P}$ the standard step would be in the bound term, rather than the free term, or else the β rule for a $(\text{let } !^w x = !^w M \text{ in})$, $(\text{let } !^c x = !^c M \text{ in})$ or $(\text{let } !^i x = !^c !^w M \text{ in})$ prefix.

- *We have*

  $$\begin{aligned} M_0 \quad &\equiv \quad \text{let } !^\square x = ({}^{\mathrm{i}^w}\mathcal{P}^n\, M_1) \text{ in } M_2 \\ &\xrightarrow[\text{i-}(\star !^\square !^i)]{} \quad {}^{\mathrm{i}^w}\mathcal{P}^n\, (\text{let } !^\square x = M_1 \text{ in } M_2) \\ &\equiv \quad N_0 \ , \end{aligned}$$

  where M_1 *is not an answer.* Since M_1 is not an answer, either the standard redex is within it, or is within a bound term in the ${}^{\mathrm{i}^w}\mathcal{P}^n$. In both cases we have $\Delta \equiv \Delta_1$. In the former case we have simply $E_1 \equiv {}^{\mathrm{i}^w}\mathcal{P}^n\, (\text{let } !^\square x = E_2 \text{ in } M_2)$, and so $E \equiv \text{let } !^\square x = ({}^{\mathrm{i}^w}\mathcal{P}^n\, E_2) \text{ in } M_2$. In the latter case we have

  $$N_0 \equiv {}^{\mathrm{i}^w}\mathcal{P}^m\, \text{let } !^i x_m = !^w E_2[\Delta_1] \text{ in } {}^{\mathrm{i}^w}\mathcal{P}^n_{m+1}\, (\text{let } !^\square x = M_1 \text{ in } M_2)$$

  where $({}^{\mathrm{i}^w}\mathcal{P}^n_{m+1}\, (\text{let } !^\square x = M_1 \text{ in } M_2))$ demands x_m. Then we can construct

  $$E \equiv \text{let } !^\square x = {}^{\mathrm{i}^w}\mathcal{P}^m\, (\text{let } !^i x_m = !^w E_2[\Delta_1] \text{ in } ({}^{\mathrm{i}^w}\mathcal{P}^n_{m+1}\, M_1)) \text{ in } M_2 \ .$$

- *We have*

$$\begin{array}{rcl} M_0 & \equiv & \text{let } !^{\Box}x = (\mathcal{P}^n\, M_1) \text{ in } M_2 \\ & \xrightarrow[\texttt{i-}(\star!^{\Box}!^{i})]{} & \mathcal{P}^n\,(\text{let } !^{\Box}x = M_1 \text{ in } M_2) \\ & \equiv & N_0\ , \end{array}$$

where not all $\mathcal{P}$ *are of the form* ${}^{\mathrm{i}^w}\mathcal{P}$. Similar to the previous case, again with the caveat that non-${}^{\mathrm{i}^w}\mathcal{P}$ prefixes have a different evaluation behavior.

- *We have*

$$\begin{array}{rcl} M_0 & \equiv & \text{let } !^{i}x = !^{\Box}({}^{\mathrm{i}^w}\mathcal{P}^n\, M_1) \text{ in } M_2 \\ & \xrightarrow[\texttt{i-}(!^{i}!^{\Box}!^{w})]{} & {}^{\mathrm{i}^w}\mathcal{P}^n\,(\text{let } !^{i}x = !^{\Box}M_1 \text{ in } M_2) \\ & \equiv & N_0 \end{array}$$

where M_1 *is not an answer and* $!^{\Box}$ *ranges over* $!^w, !^c$. If $!^{\Box}$ is $!^c$, then either the redex is within M_1, or is by demand within a bound term of the ${}^{\mathrm{i}^w}\mathcal{P}$. In the latter case we have an analysis an in the previous similar cases of this proof. If $!^{\Box}$ is $!^i$, then we either have the standard step within M_2, or M_2 demanding x and one of the arrangements for reduction under the $!^w$.

- *We have*

$$\begin{array}{rcl} M_0 & \equiv & \text{let } !^{i}x = !^{w}({}^{\mathrm{i}^w}\mathcal{P}^n\, A) \text{ in } M_1 \\ & \xrightarrow[\texttt{i-}(!^{i}!^{w}!^{w})]{} & {}^{\mathrm{i}^w}\mathcal{P}^n\,(\text{let } !^{i}x = !^{w}A \text{ in } M_1) \\ & \equiv & N_0 \end{array}$$

where M_1 *does not demand* x. Trivially, the standard redex in each case in within M_1.

We can have no other top-level internal steps.

If $E_1 \equiv [\,]$, then the result is immediate, with $E \equiv [\,]$ as well.

Otherwise the result is clear. If the topmost configuration of C_0 is that of an evaluation context — that is, if C_0 and E_0 both have the same immediate structure such as

$$\begin{array}{rcl} C_0 & \equiv & C_1\ L \text{ and} \\ E_1 & \equiv & E_1\ L\ , \end{array}$$

then we have the result by the induction hypothesis. Finally, when the internal and standard contractions are in different subterms, we have the result immediately by Lemma 8.5.9 and Lemma 8.5.11. ⧫

Lemma 8.5.14. *Reduction in* $\mathsf{SLin}^{\star}$ *satisfies* ***P4****:* Let $\sigma : M_0 \underset{\mathrm{i}}{\twoheadrightarrow} N_0$ and $M_0 \overset{\delta}{\longmapsto} N_1$ in $\mathsf{SLin}^{\star}$. Then δ/σ contains a single element δ_1 which is a standard redex of N_0: $N_0 \overset{\delta_1}{\longmapsto} N$.

Proof: The analysis is similar to that of the previous proof, by a structural induction on M. Where both contractions are in the same subterm we have result immediately from the induction hypothesis. We have three primary cases which remain: where one or the other step is top-level, and where each is in a distinct child of the top-level term. We take

$$\begin{array}{rcl} \sigma : M_0 \equiv & C_0[\Delta_0] \underset{\mathrm{i}}{\twoheadrightarrow} C_0[\Delta_0'] & \equiv N_0 \\ \sigma : M_0 \equiv & E_1[\Delta_1] \mapsto E_1[\Delta_1'] & \equiv N_1\ , \end{array}$$

and construct in each case the required E and Δ.

If the standard step is top-level, then the result is immediate with Lemma 8.5.9 and Lemma 8.5.11 by a consideration of the structure of the terms following from each possible redex.

If the internal step is top-level, we have the following cases.

- *We have* $M_0 \equiv (\text{let } !^{i}x = !^{w}!^{c}M_1 \text{ in } M_2) \xrightarrow[\texttt{i-}(\beta!^{i}_{wc})]{} M_2[x := M_1] \equiv N_0$, *where* M_1 *does not demand* x. Since M_1 does not demand x, the standard step must be within M_1, so we have the result by Lemma 8.5.6 and Lemma 8.5.8.

- *We have* $M_0 \equiv (\text{let } !^i x = !^w M_1 \text{ in } M_2) \xrightarrow[\text{i-(W}!^i)]{} M_2 \equiv N_0$ *where* $x \notin \text{fv}(M_2)$. We can have only $E_1 \equiv (\text{let } !^i x = !^w M_1 \text{ in } E_2)$, and then E is just E_2.

- *We have* $M_0 \equiv ({}^{\mathrm{i}w}\mathcal{P}^n\ M_1)\ M_2 \xrightarrow[\text{i-}(\star !^i \multimap)]{} {}^{\mathrm{i}w}\mathcal{P}^n\ (M_1\ M_2) \equiv N_0$ *where* M_1 *is not an answer.* Either the standard step is within M_1 or a bound term of the ${}^{\mathrm{i}w}\mathcal{P}^n$. In the former case we have

$$E_1 \equiv ({}^{\mathrm{i}w}\mathcal{P}^n\ E_2)\ M_2\ \ ,$$

  and so $E \equiv {}^{\mathrm{i}w}\mathcal{P}^n\ (E_2\ M_2)$. In the latter case we have some

$${}^{\mathrm{i}w}\mathcal{P}_{m+1} \cdots {}^{\mathrm{i}w}\mathcal{P}_n\ M_1$$

  demanding x and

$$E_1 \equiv ({}^{\mathrm{i}w}\mathcal{P}^{m-1}\ \text{let } !^i x_m = !^w E_2 \text{ in } {}^{\mathrm{i}w}\mathcal{P}_{m+1} \cdots {}^{\mathrm{i}w}\mathcal{P}_n\ M_1)\ M_2$$

  so then $E \equiv {}^{\mathrm{i}w}\mathcal{P}^{m-1}\ \text{let } !^i x_m = !^w E_2 \text{ in } {}^{\mathrm{i}w}\mathcal{P}_{m+1} \cdots {}^{\mathrm{i}w}\mathcal{P}_n\ (M_1\ M_2)$.

- *We have* $M_0 \equiv (\mathcal{P}^n\ M_1)\ M_2 \xrightarrow[\text{i-}(\star !^{\square} \multimap)]{} \mathcal{P}^n\ (M_1\ M_2) \equiv N_0$ *where not all of the* $\mathcal{P}$ *are of the form* ${}^{\mathrm{i}w}\mathcal{P}$. As in the previous case, taking into account the different evaluation behavior of the non ${}^{\mathrm{i}w}\mathcal{P}$ prefixes.

- *We have*

$$\begin{array}{rcl} M_0 & \equiv & \text{let } !^{\square} x = ({}^{\mathrm{i}w}\mathcal{P}^n\ M_1) \text{ in } M_2 \\ & \xrightarrow[\text{i-}(\star !^{\square} !^i)]{} & {}^{\mathrm{i}w}\mathcal{P}^n\ (\text{let } !^{\square} x = M_1 \text{ in } M_2) \\ & \equiv & N_0\ \ , \end{array}$$

  where M_1 *is not an answer,* and

- *We have*

$$\begin{array}{rcl} M_0 & \equiv & \text{let } !^{\square} x = (\mathcal{P}^n\ M_1) \text{ in } M_2 \\ & \xrightarrow[\text{i-}(\star !^{\square} !^{\triangle})]{} & \mathcal{P}^n\ (\text{let } !^{\square} x = M_1 \text{ in } M_2) \\ & \equiv & N_0\ \ , \end{array}$$

  where not all $\mathcal{P}$ *are of the form* ${}^{\mathrm{i}w}\mathcal{P}$.

  Both of these cases are by a similar analysis as for the $(!^{\square}!^{\triangle})$ cases in the previous proof, with the obvious construction of the reshuffled evaluation contexts as in the $(!^{\square}\multimap)$ steps immediately above.

- *We have*

$$\begin{array}{rcl} M_0 & \equiv & \text{let } !^i x = !^{\square}({}^{\mathrm{i}w}\mathcal{P}^n\ M_1) \text{ in } M_2 \\ & \xrightarrow[\text{i-}(!^i !^{\square} !^w)]{} & \left({}^{\mathrm{i}w}\mathcal{P}^n\ (\text{let } !^i x = !^{\square} M_1 \text{ in } M_2)\right) \\ & \equiv & N_0 \end{array}$$

  where M_1 *is not an answer and* $!^{\square}$ *ranges over* $!^w, !^c$. If $!^{\square}$ is $!^c$, then either the redex is within M_1, or is by demand within a bound term of the ${}^{\mathrm{i}w}\mathcal{P}$, and we have the same reshuffling of the evaluation contexts as in the $(!^{\square}!^{\triangle})$ steps. Otherwise if $!^{\square}$ is $!^w$ we have either the redex in $M_2 \equiv E_2[\Delta_1]$ and then $E \equiv \left({}^{\mathrm{i}w}\mathcal{P}^n\ (\text{let } !^i x = !^w M_1 \text{ in } E_2)\right)$, or else M_2 demanding x and an arrangement as for when $!^{\square}$ is $!^c$.

- *We have*

$$\begin{array}{rcl} M_0 & \equiv & \text{let } !^i x = !^w({}^{\mathrm{i}w}\mathcal{P}^n\ A) \text{ in } M_1 \\ & \xrightarrow[\text{i-}(!^i !^w !^w)]{} & {}^{\mathrm{i}w}\mathcal{P}^n\ (\text{let } !^i x = !^w A \text{ in } M_1) \\ & \equiv & N_0 \end{array}$$

  where M_1 *does not demand* x. Again, this final case is straightforward. We must have the standard step within $M_1 \equiv E_2[\Delta_1]$, and its residual in N_0 is just $({}^{\mathrm{i}w}\mathcal{P}^n\ (\text{let } !^i x = !^w A \text{ in } E_2))$.

If the two redexes are in different subterms, then the standard step — being outside of the effect of the internal step — could not be duplicated or discarded, and so the residual is unique; by Lemma 8.5.9 and Lemma 8.5.11 it is again a standard step. ♦

Lemma 8.5.15. Let $M \underset{(\beta\multimap,\beta!^{\square},!^{\square}\multimap,!^{\square}!^{\triangle},!^{i}!^{\square}!^{w})}{\twoheadrightarrow} A$; then there exists some A_0 such that

$$M \underset{\mathsf{Need}^\star}{\mapsto\!\!\!\twoheadrightarrow} A_0 \underset{(\beta\multimap,\beta!^{\square},!^{\square}\multimap,!^{\square}!^{\triangle},!^{i}!^{\square}!^{w})}{\twoheadrightarrow} A \ .$$

Proof: By the general technique of Lemma 2.1.20, here with Lemma 8.5.10, **P1**, **P2**, **P3** and **P4**. ♦

Lemma 8.5.16. Let $M \underset{(\mathrm{W}!^{i})}{\longrightarrow} M_0 \underset{\mathsf{SLin}^\star}{\longmapsto} N$. Then there exists some N_0 such that

$$M \underset{\mathsf{SLin}^\star}{\longmapsto}^{=} N_0 \underset{(\mathrm{W}!^{i})}{\twoheadrightarrow} N \ .$$

Proof: By analysis of the relative positions of the original redexes, in the same manner as for $\mathsf{Need}^\star$ and $\mathsf{Aff}^\star$in Lemma 4.6.15 and Lemma 6.5.16, respectively. ♦

Lemma 8.5.17. Let $\sigma : M \underset{\mathsf{SLin}^\star}{\twoheadrightarrow} A$; then there exists some A_0 such that

$$M \underset{\mathsf{SLin}^\star}{\mapsto\!\!\!\twoheadrightarrow} A_0 \underset{\mathsf{SLin}^\star}{\twoheadrightarrow} A \ .$$

Proof: As for $\mathsf{Need}^\star$ and $\mathsf{Aff}^\star$, by induction on the number of non-standard $\mathsf{SLin}^\star$ steps in σ. If the last such step in σ is a $(\mathrm{W}!^{i})$ step, then we reduce the problem to the inductive hypothesis by Lemma 8.5.16 and Lemma 8.5.11. If the last non-standard step is an internal $(\beta\multimap, \beta!^{\square}, !^{\square}\multimap, !^{\square}!^{\triangle}, !^{i}!^{\square}!^{w})$ step, then the result follows from Lemma 8.5.15. ♦

We can now deduce the main standardization result for call-by-need.

Proposition 8.5.18, $\mathrm{Std}(\mathsf{SLin})$**.** The relation $\underset{\mathsf{SLin}}{\longmapsto}$ is a standard strategy for $\underset{\mathsf{SLin}}{\longrightarrow}$ reduction to answers.

Proof: By Lemma 8.5.1 for uniqueness and termination at answers; we have completeness with Lemma 8.5.17 and Lemma 8.3.1. ♦

8.6 Typing without the structural rules

It is also possible to formulate the typing rules for SLin without structural rules. Rather than a single Id rule and four Dereliction rules, we use one axiom link per zone; rather than explicit weakening and contraction rules, we make the different abilities of different zones implicit in other rules. We present the typing relation $\vdash_{\mathsf{SRF}}$ in Figure 8.11. This style of rule is the same as in Barber's formulation of DILL [7].

Permission to weaken variables in certain zones in implicit in the axiom rules. For example in Id_l,

$$\frac{}{\blacktriangleright^{i}\Gamma, \blacktriangleright^{w}\Delta, \blacktriangleright^{l}x : A \vdash_{\mathsf{SRF}} x : A}\ \mathrm{Id}_l$$

the variables in Γ and Δ are quite clearly unused, which is acceptable since we allow weakening in those two zones. On the other hand, no unused assumptions appear in the zones where weakening is not allowed. Permission to contract is implicit in the elimination rules with two inferences above the bar. For example in $\multimap$-E, we can use variables in Γ and Φ to type the same variables in both subterms of the application, which is acceptable since contraction is allowed in those two zones. On the other hand, in the other zones we must use separate environments for the different subterms.

Although we have fewer $\vdash_{\mathsf{SRF}}$ typing rules in total, they are individually more complicated than the corresponding $\vdash_{\mathsf{SLin}}$ rules. Our shorthand with $\mathcal{E}$, $\blacktriangleright^{l}$, etc. is not so useful here since the pairwise interactions of zones is more important and must be made explicit. Nonetheless, it is clear that the schemes produce judgments

$$\dfrac{}{\blacktriangleright^{i}\Gamma, x : A, \blacktriangleright^{w}\Delta \vdash_{\mathsf{SRF}} x : A}\,\mathrm{Id}_i \qquad \dfrac{}{\blacktriangleright^{i}\Gamma, \blacktriangleright^{w}\Delta, x : A \vdash_{\mathsf{SRF}} x : A}\,\mathrm{Id}_w$$

$$\dfrac{}{\blacktriangleright^{i}\Gamma, \blacktriangleright^{w}\Delta, \blacktriangleright^{c}x : A \vdash_{\mathsf{SRF}} x : A}\,\mathrm{Id}_c \qquad \dfrac{}{\blacktriangleright^{i}\Gamma, \blacktriangleright^{w}\Delta, \blacktriangleright^{l}x : A \vdash_{\mathsf{SRF}} x : A}\,\mathrm{Id}_l$$

$$\dfrac{\mathcal{E}, \blacktriangleright^{l}x : A \vdash_{\mathsf{SRF}} M : B}{\mathcal{E} \vdash_{\mathsf{SRF}} \lambda x.\, M : A \multimap B}\,\multimap\text{-I}$$

$$\dfrac{\blacktriangleright^{i}\Gamma, \blacktriangleright^{c}\Phi \vdash_{\mathsf{SRF}} M : A}{\blacktriangleright^{i}\Gamma, \blacktriangleright^{c}\Phi \vdash_{\mathsf{SRF}} M : !^{c}A}\,!^{c}\text{-I} \qquad \dfrac{\blacktriangleright^{i}\Gamma, \blacktriangleright^{w}\Phi \vdash_{\mathsf{SRF}} M : A}{\blacktriangleright^{i}\Gamma, \blacktriangleright^{w}\Phi \vdash_{\mathsf{SRF}} M : !^{w}A}\,!^{w}\text{-I}$$

$$\dfrac{\blacktriangleright^{i}\Gamma, \blacktriangleright^{w}\Delta_0, \blacktriangleright^{c}\Phi, \blacktriangleright^{l}\Psi_0 \vdash_{\mathsf{SRF}} M : A \multimap B \qquad \blacktriangleright^{i}\Gamma, \blacktriangleright^{w}\Delta_1, \blacktriangleright^{c}\Phi, \blacktriangleright^{l}\Psi_1 \vdash_{\mathsf{SRF}} N : A}{\blacktriangleright^{i}\Gamma, \blacktriangleright^{w}\Delta_0, \Delta_1, \blacktriangleright^{c}\Phi, \blacktriangleright^{l}\Psi_0, \Psi_1 \vdash_{\mathsf{SRF}} M\, N : B}\,\multimap\text{-E}$$

$$\dfrac{\blacktriangleright^{i}\Gamma, \blacktriangleright^{w}\Delta_0, \blacktriangleright^{c}\Phi, \blacktriangleright^{l}\Psi_0 \vdash_{\mathsf{SRF}} M : !^{c}A \qquad \blacktriangleright^{i}\Gamma, \blacktriangleright^{w}\Delta_1, \blacktriangleright^{c}\Phi, x : A, \blacktriangleright^{l}\Psi_1 \vdash_{\mathsf{SRF}} N : B}{\blacktriangleright^{i}\Gamma, \blacktriangleright^{w}\Delta_0, \Delta_1, \blacktriangleright^{c}\Phi, \blacktriangleright^{l}\Psi_0, \Psi_1 \vdash_{\mathsf{SRF}} \text{let } !^{c}x = M \text{ in } N : B}\,!^{c}\text{-E}$$

$$\dfrac{\blacktriangleright^{i}\Gamma, \blacktriangleright^{w}\Delta_0, \blacktriangleright^{c}\Phi, \blacktriangleright^{l}\Psi_0 \vdash_{\mathsf{SRF}} M : !^{w}A \qquad \blacktriangleright^{i}\Gamma, \blacktriangleright^{w}\Delta_1, x : A, \blacktriangleright^{c}\Phi, \blacktriangleright^{l}\Psi_1 \vdash_{\mathsf{SRF}} N : B}{\blacktriangleright^{i}\Gamma, \blacktriangleright^{w}\Delta_0, \Delta_1, \blacktriangleright^{c}\Phi, \blacktriangleright^{l}\Psi_0, \Psi_1 \vdash_{\mathsf{SRF}} \text{let } !^{w}x = M \text{ in } N : B}\,!^{w}\text{-E}$$

$$\dfrac{\mathcal{E} \vdash_{\mathsf{SRF}} M : !^{w}!^{c}A \qquad \mathcal{F}, \blacktriangleright^{i}x : A \vdash_{\mathsf{SRF}} N : B}{\mathcal{E}, \mathcal{F} \vdash_{\mathsf{SRF}} \text{let } !^{i}x = M \text{ in } N : B}\,!^{i}\text{-E}_1$$

$$\dfrac{\mathcal{E} \vdash_{\mathsf{SRF}} M : !^{c}!^{w}A \qquad \mathcal{F}, \blacktriangleright^{i}x : A \vdash_{\mathsf{SRF}} N : B}{\mathcal{E}, \mathcal{F} \vdash_{\mathsf{SRF}} \text{let } !^{i}x = M \text{ in } N : B}\,!^{i}\text{-E}_2$$

Figure 8.11: Omitting typing rules: the relation $\vdash_{\mathsf{SRF}}$ for SLin.

of the same types, and we believe that the shorter, less intricate rules of $\vdash_{\mathsf{SLin}}$ allow simpler, easier to read proofs, even if a few are longer.

However it should be noted that the structural-rule free formulation does allow a refinement of the Curry-Howard correspondence. Since under $\vdash_{\mathsf{SLin}}$ we can rearrange the location of structural rules within a proof fairly arbitrarily without changing a term, several proofs of a term's type can be given. In other words, each well-typed term can be Curry-Howard correspondent with many proofs in the logic. With the structural-rule free formulation, we in fact have an isomorphism between closed terms and theorems (To see that isomorphism fails for open terms in general one need only note that all four Id rules in $\vdash_{\mathsf{SRF}}$ prove the same thing).

Chapter 9

Three separated translations

FOR the separated-linear calculus, we need new separated translations. The adaptations of Girard's original mappings are straightforward, and not surprisingly behave exactly as their unitary counterparts. What is more surprising is the behavior of a combination of the two: we can now soundly and completely map call-by-need into the same calculus as call-by-name and call-by-value/let. This hybrid translation is not possible in the unitary calculi because it involves using the weakening and contraction prefixes separately. We consider the separated standard translation in Section 9.1, the separated steadfast translation in Section 9.2, and the hybrid translation in Section 9.3.

9.1 The separated standard translation

Figure 9.1 illustrates the standard translation from Name to SLin. This translation is not especially surprising, and is largely the same as the translation into Lin. Recalling the intention that ! was just divided into two components composable into the original, we simply replace occurrences of ! in the translation with $!^c!^w$, and occurrences of !-eliminators with $!^i$-eliminators. The intuition behind the translation remains the same as in the unseparated case. The argument of every application is explicitly allowed to be either discarded or copied; functions are translated to linear functions which use their arguments exactly once, namely in a $!^i$-elimination which retrieves the original intuitionistic argument.

It is reasonable to ask whether we might instead translate an argument M as $!^w!^c M^\circ$: for much of what follows here, such an exchange is fine. However, the results on standard evaluation would no longer hold; it is there that the operational distinction between $!^w!^c M$ and $!^c!^w M$ becomes important.

The results on both separated translations follow just as in the unitary cases; the new SLin structures do not expose any new redexes in the images of the translations. We verify the result at length for $\circ$ in this section.

Lemma 9.1.1. The set of terms ranged over by S in the grammar of Figure 9.1 contains the image of $\circ$ on Need terms and is closed under SLin-reduction.

Proof: The first claim is clear; the second follows by inspection of the rules, again using the fact that since $S ::= x$, the set is closed under substitution of terms S for variables. ♦

The soundness and completeness results for the translation from Name into SLin are nearly identical to the results from Name into Lin, since the only possible reduction steps, in effect, are those corresponding to simply replacing '!' in the latter with '$!^c!^w$' in the former.

Lemma 9.1.2. The translation † from the subset of SLin ranged over by terms S in the grammar of Figure 9.1 to Name preserves substitution, is a right-inverse to $\circ$ and is sound with respect to types and reduction:

(i) $(S[x := T])^\dagger \equiv S^\dagger[x := T^\dagger]$.

(ii) Let $M \in \mathsf{Name}$. Then $M \equiv M^{\circ\dagger}$.

(iii) If $\Gamma \vdash_{\mathsf{SLin}} S : A$ then $\blacktriangleright_i \Gamma^\dagger \vdash_{\mathsf{Name}} S^\dagger : A^\dagger$.

(iv) If $S \xrightarrow[\mathsf{SLin}]{} \!\!\!\!\!\rightarrow T$ then $S^\dagger \xrightarrow[\mathsf{Name}]{} \!\!\!\!\!\rightarrow T^\dagger$.

(v) If $S \mapsto\!\!\!\!\!\!\rightarrow_{\mathsf{SLin}} T$ then $S^\dagger \mapsto\!\!\!\!\!\!\rightarrow_{\mathsf{Name}} T^\dagger$.

Proof: The first clause is true because † does not alter variables, and the second clause is clear from a simple structural induction and case analysis on Need terms. The third case is exactly as in proofs for the non-separated translations.

The fourth case is rather straightforward since only a handful of reduction rules are applicable to $\circ$-images.

Standard translation

Of types

$$\begin{array}{rcl}(A \to B)^\circ & \equiv & (!^c!^w A^\circ) \multimap B^\circ \\ Z^\circ & \equiv & Z\end{array}$$

Of terms

$$\begin{array}{rcl}x^\circ & \equiv & x \\ (\lambda x.\, M)^\circ & \equiv & \lambda y.\, \text{let } !^i x = y \text{ in } M^\circ \\ (M\ N)^\circ & \equiv & M^\circ\, (!^c!^w N^\circ)\end{array}$$

Of typing environments

$$(x : A)^\circ \quad \equiv \quad x : A^\circ$$

Figure 9.1: The standard translation from Name into SLin.

Reduction-closed images of the standard translation

$$S, T \ ::= x \mid \lambda y.\, \text{let } !^i x = y \text{ in } S \mid S\ !^c!^w T \mid \text{let } !^i x = !^c!^w S \text{ in } T$$

A right-inverse for the standard translation

$$\begin{array}{rcl}x^\dagger & \equiv & x \\ (\lambda y.\, \text{let } !^i x = y \text{ in } S)^\dagger & \equiv & \lambda x.\, S^\dagger \\ (S\ !^c!^w T)^\dagger & \equiv & S^\dagger\, T^\dagger \\ (\text{let } !^i x = !^c!^w T \text{ in } S)^\dagger & \equiv & S^\dagger[x := T^\dagger]\end{array}$$

Figure 9.2: The inverse standard translation from SLin into Name.

- $(\lambda y.\, \text{let } !^i x = y \text{ in } S)\ !^c!^w T \xrightarrow[(\beta\multimap)]{} \text{let } !^i x = !^c!^w T \text{ in } S$. We have

$$\begin{array}{rcl}((\lambda y.\, \text{let } !^i x = y \text{ in } S)\ !^c!^w T)^\dagger & \equiv & (\lambda x.\, S^\dagger)\, T^\dagger \quad \text{and} \\ (\text{let } !^i x = !^c!^w T \text{ in } S)^\dagger & \equiv & S^\dagger[x := T^\dagger]\ ,\end{array}$$

  which are clearly related by $(\beta \to)$.

- $\text{let } !^i x = !^c!^w T \text{ in } S \xrightarrow[(\beta !^i)]{} S[x := T]$. Then we have

$$\begin{array}{rcl}(\text{let } !^i x = !^c!^w T \text{ in } S)^\dagger & \equiv & S^\dagger[x := T^\dagger] \\ & \equiv & (S[x := T])^\dagger\ .\end{array}$$

- $(\text{let } !^i x = !^w!^c T_0 \text{ in } S)\ !^w!^c T_1 \xrightarrow[(!^i\multimap)]{} \text{let } !^i x = !^w!^c T_0 \text{ in } S\ !^w!^c T_1$. We have the standard images of each side equivalent in Name since

$$(S^\dagger[x := !^w!^c T_0^\dagger])\ !^w!^c T_1^\dagger \equiv (S^\dagger\ !^w!^c T_1^\dagger)[x := !^w!^c T_0^\dagger]$$

  and $x \notin \text{fv}(!^w!^c T_1)$.

- *Given*

$$\begin{array}{l}\text{let } !^i y = (\text{let } !^i x = !^w!^c T_0 \text{ in } !^w!^c T_1) \text{ in } S \\ \xrightarrow[(!^i!^i)]{} \quad \text{let } !^i x = !^w!^c T_0 \text{ in let } !^i y = !^w!^c T_1 \text{ in } S\ .\end{array}$$

  We again have the standard images of the two sides identical since $x \notin \text{fv}(S)$, and so

$$S^\dagger[y := !^w!^c T_1^\dagger[x := !^w!^c T_0^\dagger]] \equiv S^\dagger[y := !^w!^c T_1^\dagger][x := !^w!^c T_0^\dagger]\ .$$

As usual, compatible closure is trivial, and Clause (iv) is complete. For Clause (v) we can apply at the top level the standard $(\beta\multimap, \beta !^i)$ rules, with soundness just as

Steadfast translation

Of types

$$A^* \equiv\ !^c!^w A^+$$

$$\begin{array}{rcl} (A \to B)^+ & \equiv & (A^*) \multimap (B^*) \\ Z^+ & \equiv & Z \end{array}$$

Of terms

$$\begin{array}{rcl} V^* & \equiv & !^c!^w V^+ \\ (M\ N)^* & \equiv & (\text{let } !^i z = M^* \text{ in } z)\ N^* \\ (\text{let } x = M \text{ in } N)^* & \equiv & \text{let } !^i x = M^* \text{ in } N^* \end{array}$$

$$\begin{array}{rcl} x^+ & \equiv & x \\ (\lambda x.\, M)^+ & \equiv & \lambda y.\, \text{let } !^i x = y \text{ in } M^* \end{array}$$

Figure 9.3: The steadfast translation from Val into SLin.

in Clause (iv); only evaluation contexts of the form $(E\ S_0)$ can be drawn from a term S, so compatible closure under evaluation contexts follows by the obvious structural induction. ♦

Proposition 9.1.3, Separated standard translation of Name.
Let $M, N \in \mathsf{Name}$.

1. $(M[x := N])^\circ \equiv M^\circ[x := N^\circ]$.
2. $\Gamma \vdash_{\mathsf{Name}} M : A$ if and only if $\blacktriangleright_i \Gamma^\circ \vdash_{\mathsf{SLin}} M^\circ : A^\circ$.
3. $M \xrightarrow[\mathsf{Name}]{} N$ if and only if $M^\circ \xrightarrow[\mathsf{SLin}]{} N^\circ$.
4. $M \mapsto_{\mathsf{Name}} N$ if and only if $M^\circ \mapsto_{\mathsf{SLin}} N^\circ$.

Proof: The proof is as before: the first clause follows by a straightforward induction. Soundness of the second clause follows by an analysis of each Name typing rule, and completeness by Lemma 9.1.1 and Lemma 9.1.2.(ii) and (iii). For soundness in the third clause we take $(\lambda x.\, M)\ N \xrightarrow[(\beta\to)]{} M[x := N]$, and in SLin,

$$\begin{array}{rl} & ((\lambda x.\, M)\ N)^\circ \\ \equiv & (\lambda y.\, \text{let } !^i x = y \text{ in } M^\circ)\ !^c!^w N^\circ \\ (\beta\multimap) & \text{let } !^i x = !^c!^w N^\circ \text{ in } M^\circ \\ (\beta!) & M^\circ[x := N^\circ]\ , \end{array}$$

and by the first clause we have $(M[x := N])^\circ \equiv M^\circ[x := N^\circ]$. Completeness follows from Lemma 9.1.1 and Lemma 9.1.2.(ii) and .(iv). Likewise, Clause 4 follows from the fact that $\circ$ preserves evaluation positions, and Lemma 9.1.2.(v). ♦

9.2 The separated steadfast translation

Figure 9.3 presents the steadfast translation from Val into SLin. As with the standard translation, the steadfast translation into SLin is only superficially distinct from the steadfast translation into Lin, and most of the results follow just as in the non-separated case.

Proposition 9.2.1, The separated steadfast translation of Val. Let $M, N \in \mathsf{Val}$.

1. $(M[x := V])^* \equiv M^*[x := V^+]$.
2. $\Gamma \vdash_{\mathsf{Val}} M : A$ if and only if $\blacktriangleright_i \Gamma^* \vdash_{\mathsf{SLin}} M^* : A^*$.

Hybrid translation

Of types

$$\begin{array}{rcl} A^{\circledast} & \equiv & !^{c} A^{\oplus} \\ (A \to B)^{\oplus} & \equiv & (!^{w} A^{\circledast}) \multimap B^{\circledast} \\ Z^{\oplus} & \equiv & Z \end{array}$$

Of terms

$$\begin{array}{rcl} V^{\circledast} & \equiv & !^{c} V^{\oplus} \\ (M\ N)^{\circledast} & \equiv & (\text{let } !^{c} z = M^{\circledast} \text{ in } z)\ (!^{w} N^{\circledast}) \\ (\text{let } x = M \text{ in } N)^{\circledast} & \equiv & \text{let } !^{i} x = !^{w} M^{\circledast} \text{ in } N^{\circledast} \\ x^{\oplus} & \equiv & x \\ (\lambda x.\, M)^{\oplus} & \equiv & \lambda y.\, \text{let } !^{i} x = y \text{ in } M^{\circledast} \end{array}$$

Of typing environments

$$(\Gamma, x : A)^{\circledast} \equiv \Gamma^{\circledast}, x : A^{\oplus}$$

Figure 9.4: The hybrid translation from Need into SLin.

3. If $M \twoheadrightarrow_{\mathsf{Val}} N$ then $M^{*} \twoheadrightarrow_{\mathsf{SLin}} N^{*}$.
4. $M \mapsto\!\!\!\twoheadrightarrow_{\mathsf{Val}} N$ if and only if $M^{*} \mapsto\!\!\!\twoheadrightarrow_{\mathsf{SLin}} N^{*}$.

Proof: Again, just as in the unitary case. ♦

As with the non-separated case, the translation is not complete for general reduction sequences. The same counterexample suffices; we have:

$$\begin{array}{cl} & (I\ P\ V)^{*} \\ \equiv & (\text{let } !^{i} z = (\text{let } !^{i} w = !^{c}!^{w}(\lambda y.\, \text{let } !^{i} x = y \text{ in } !^{c}!^{w} x) \text{ in } w)\ P^{*} \\ & \quad \text{in } z)\ (!^{c}!^{w} V^{+}) \\ \xrightarrow[(\beta !^{i})]{} & (\text{let } !^{i} z = (\lambda y.\, \text{let } !^{i} x = y \text{ in } !^{c}!^{w} x)\ P^{*} \text{ in } z)\ (!^{c}!^{w} V^{+}) \\ \xrightarrow[(\beta \multimap)]{} & (\text{let } !^{i} z = (\text{let } !^{i} x = P^{*} \text{ in } !^{c}!^{w} x) \text{ in } z)\ (!^{c}!^{w} V^{+}) \\ \xrightarrow[(!^{i}!^{i})]{} & (\text{let } !^{i} x = P^{*} \text{ in let } !^{i} z = !^{c}!^{w} x \text{ in } z)\ (!^{c}!^{w} V^{+}) \\ \xrightarrow[(\beta !^{i})]{} & (\text{let } !^{i} x = P^{*} \text{ in } x)\ (!^{c}!^{w} V^{+}) \\ \equiv & (P\ V)^{*} \end{array}$$

where $I\ P\ V \not\twoheadrightarrow_{\mathsf{Val}} P\ V$ in general. The same results as for Val hold for Let.

9.3 The hybrid translation

Figure 9.4 describes the hybrid translation motivated in the introduction. We translate all values to be copyable but not necessarily discardable, and all function arguments to be discardable, but not necessarily copyable. So a call-by-need function mapping the type A to the type B is translated to a copyable linear function from the image of A made explicitly discardable, into the image of B: that is, into the type $!^{c}((!^{w} A^{\circledast}) \multimap B^{\circledast})$.

We now begin a series of lemmas which lead to Proposition 9.3.10, the soundness and completeness result for the hybrid transformation. As we believe this translation to be entirely new, we pay special attention to the pedantic details in this section. We will use the grammars of Figure 9.5 and mappings of Figure 9.6, which we characterize by the following lemmas:

Lemma 9.3.1. Let $M \in \mathsf{Need}$ and $M^{\circledast} \twoheadrightarrow_{\mathsf{SLin}} N$. Then N matches some S in the grammar of Figure 9.5. Moreover if and $M^{\circledast} \mapsto\!\!\!\twoheadrightarrow_{\mathsf{SLin}} N$, N matches some $\mathbb{S}$ in that grammar.

Lemma 9.3.2. Let $M \in \mathsf{Need}$. Then $(M^{\circledast})^{\natural} \equiv M$.

Reduction-closed images of Need types and terms

$$
\begin{array}{lrl}
\text{Types} & A, B & ::= \,!^c \bar{A} \\
 & \bar{A}, \bar{B} & ::= (!^w A) \multimap B \mid Z \\
\text{Terms} & S, T & ::= \,!^c U \mid P\ (!^w S) \\
 & & \mid \ \text{let}\ !^i x = \,!^w S \text{ in } T \\
 & & \mid \ \text{let}\ !^c x = S \text{ in } (D[x]\ !^w T)\ ,\ x \notin \text{fv}(D, T) \\
\text{Values} & U & ::= x \mid \lambda y.\, \text{let}\ !^i x = y \text{ in } S \\
\text{Application prefixes} & P & ::= U \mid \text{let}\ !^i x = \,!^w S \text{ in } P \\
 & & \mid \ \text{let}\ !^c x = S \text{ in } D[x]\ ,\ x \notin \text{fv}(D) \\
 & D & ::= [\,] \mid D\ (!^w S)
\end{array}
$$

Evaluation-closed images of Need types and terms

$$
\begin{array}{lrl}
\text{Terms} & \mathcal{S}, \mathcal{T} & ::= \,!^c \mathcal{U} \mid \mathcal{P}\ (!^w \mathcal{S}) \\
 & & \mid \ \text{let}\ !^i x = \,!^w \mathcal{S} \text{ in } \mathcal{T} \\
\text{Values} & \mathcal{U} & ::= x \mid \lambda y.\, \text{let}\ !^i x = y \text{ in } \mathcal{S} \\
 & \mathcal{P} & ::= \mathcal{U} \mid \text{let}\ !^i x = \,!^w \mathcal{S} \text{ in } \mathcal{P} \\
 & & \mid \ \text{let}\ !^c x = \mathcal{S} \text{ in } x \\
\text{Answers} & \mathcal{A} & ::= \,!^c \lambda y.\, \text{let}\ !^i x = y \text{ in } \mathcal{S} \\
 & & \mid \ \text{let}\ !^i x = \,!^w \mathcal{S} \text{ in } \mathcal{A} \\
S\text{-Evaluation Contexts} & \mathcal{E} & ::= [\,] \mid \mathcal{G}\ (!^w \mathcal{S}) \\
 & & \mid \ \text{let}\ !^i x = \,!^w \mathcal{S} \text{ in } \mathcal{E} \\
 & & \mid \ \text{let}\ !^i x = \,!^w \mathcal{E} \text{ in } \mathcal{E}[x] \\
P\text{-Evaluation Contexts} & \mathcal{G} & ::= [\,] \mid \text{let}\ !^i x = \,!^w \mathcal{S} \text{ in } \mathcal{G} \\
 & & \mid \ \text{let}\ !^i x = \,!^w \mathcal{E} \text{ in } \mathcal{G}[x] \\
 & & \mid \ \text{let}\ !^c x = \mathcal{E} \text{ in } x
\end{array}
$$

Figure 9.5: SLin reduction- and evaluation- closed images of the $\circledast$-image of Need terms.

The inverse translation

Of types

$$
\begin{array}{rcl}
(!^c \bar{A})^\natural & \equiv & \bar{A}^\bowtie \\
((!^w A) \multimap B)^\bowtie & \equiv & A^\natural \to B^\natural \\
Z^\bowtie & \equiv & Z
\end{array}
$$

Of terms

$$
\begin{array}{rcl}
(!^c U)^\natural & \equiv & U^\natural \\
(P\ (!^w S))^\natural & \equiv & P^\bowtie\ S^\natural \\
(\text{let}\ !^i x = S \text{ in } T)^\natural & \equiv & \text{let } x = S^\natural \text{ in } T^\natural \\
(\text{let}\ !^c x = S \text{ in } (D[x])\ T)^\natural & \equiv & (D^\bowtie[S^\natural])\ T^\natural \\
x^\bowtie & \equiv & x \\
(\lambda y.\, \text{let}\ !^i x = y \text{ in } S)^\bowtie & \equiv & \lambda x.\, S^\natural \\
(\text{let}\ !^c x = S \text{ in } D[x])^\bowtie & \equiv & D^\bowtie[S^\natural] \\
(\text{let}\ !^i x = S \text{ in } P)^\bowtie & \equiv & \text{let}\ !^i x = S^\natural \text{ in } P^\bowtie \\
[\,]^\bowtie & \equiv & [\,] \\
(D\ (!^w S))^\bowtie & \equiv & D^\bowtie\ S^\natural
\end{array}
$$

Of typing environments

$$
\begin{array}{rcl}
(\Gamma, x : \bar{A})^\natural & \equiv & \Gamma^\natural, x : \bar{A}^\bowtie \\
(\blacktriangleright_i \Gamma, \blacktriangleright_w \Delta, \blacktriangleright_c \Phi, \blacktriangleright_l \Psi)^\natural & \equiv & \Gamma^\natural, \Delta^\natural, \Phi^\natural, \Psi^\natural
\end{array}
$$

Figure 9.6: The inverse hybrid translation from SLin to Need.

Lemma 9.3.3. Every SLin answer and evaluation context which we can form from terms $\mathcal{S}$ are given by $\mathcal{A}$ and $\mathcal{E}$, respectively; every SLin evaluation context which we can form from terms $\mathcal{P}$ is given by $\mathcal{G}$.

Lemma 9.3.4. For all U and $\mathcal{U}$, $U^{\bowtie}$ and $\mathcal{U}^{\bowtie}$ are values in Need; for all $\mathcal{A}$, $\mathcal{A}^{\bowtie}$ is an answer in Need; for all $\mathcal{E}$ and $\mathcal{G}$, $\mathcal{E}^{\natural}$ and $\mathcal{G}^{\natural}$ are evaluation contexts in Need.

Proof: The above four results are all clear, in some cases with the obvious structural induction, from an easy inspection of the definitions. ♦

Lemma 9.3.5. Let $\mathcal{E} \vdash_{\mathsf{SLin}} S :!^c \bar{A}$ and $\mathcal{F}, \blacktriangleright_c x : \bar{A} \vdash_{\mathsf{SLin}} D[x] : \bar{B}$.
Then $(\mathcal{E}, \mathcal{F})^{\natural} \vdash_{\mathsf{Need}} D^{\natural}[S^{\natural}] : \bar{B}^{\natural}$.

Proof: Since $x \notin \mathrm{fv}(D)$ we have $\mathcal{E} \vdash_{\mathsf{SLin}} \mathrm{let}\ !^c z = S \text{ in } z : \bar{A}$; then by Lemma 8.1.1, $\mathcal{E}, \mathcal{F} \vdash_{\mathsf{SLin}} D[\mathrm{let}\ !^c z = S \text{ in } z] : \bar{B}$. The result is immediate as

$$(D[\mathrm{let}\ !^c z = S \text{ in } z])^{\bowtie} \equiv D^{\bowtie}[S^{\natural}] \ .$$

♦

Lemma 9.3.6. Let D, S, T be as in Figure 9.5. Then:

$$D^{\bowtie}[\mathrm{let}\ x = S^{\natural} \text{ in } T^{\natural}] \xrightarrow[(C)]{} \!\!\!\!\!\rightarrow \mathrm{let}\ x = S^{\natural} \text{ in } D^{\bowtie}[T^{\natural}] \ .$$

Proof: By induction on D. ♦

Under the preceding lemmas, completeness for $\circledast$ is implied by the following soundness result for $\natural$:

Lemma 9.3.7. Let S, T, U and so forth be as in Figure 9.5.

(i) $(S[x := U])^{\natural} \equiv S^{\natural}[x := U^{\bowtie}]$,
$(P[x := U])^{\bowtie} \equiv P^{\bowtie}[x := U^{\bowtie}]$ and
$(D[y][x := U])^{\bowtie} \equiv (D^{\bowtie}[y])[x := U^{\bowtie}]$.

(ii) If $\mathcal{E} \vdash_{\mathsf{SLin}} S : A$ then $\mathcal{E}^{\natural} \vdash_{\mathsf{Need}} S^{\natural} : A^{\natural}$,
if $\mathcal{E} \vdash_{\mathsf{SLin}} P : A$ then $\mathcal{E}^{\natural} \vdash_{\mathsf{Need}} P^{\bowtie} : A^{\bowtie}$,
if $\mathcal{E} \vdash_{\mathsf{SLin}} U : \bar{A}$ then $\mathcal{E}^{\natural} \vdash_{\mathsf{Need}} U^{\bowtie} : \bar{A}^{\bowtie}$ and
if $\mathcal{E} \vdash_{\mathsf{SLin}} D[x] : \bar{A}$ then $\mathcal{E}^{\natural} \vdash_{\mathsf{Need}} D[x]^{\bowtie} : \bar{A}^{\bowtie}$.

(iii) If $S \xrightarrow[\mathsf{SLin}]{}\!\!\!\!\rightarrow T$ then $S^{\natural} \xrightarrow[\mathsf{Need}]{}\!\!\!\!\rightarrow T^{\natural}$.

(iv) If $S \xmapsto[\mathsf{SLin}]{}\!\!\!\!\rightarrow T$ then $S^{\natural} \xmapsto[\mathsf{Need}]{}\!\!\!\!\rightarrow T^{\natural}$.

Proof: Clause (i) is straightforward; note that we may in fact have $x \equiv y$ in the final equivalence.

For Clause (ii), we proceed by structural induction on the depth of the typing proof. The reasoning for the image of the structural SLin rules for all four syntactic groups is the same, and is largely trivial; we write out that reasoning once for the Weakening and Contraction rules in Figure 9.7 and for the Dereliction rules in Figure 9.8. We present the reasoning for terms S, P and U in Figures 9.9, 9.10 and 9.11, respectively. For terms of the form $D[x]$, where $D \equiv [\,]$ we have just the Id rule for Need; for $D \equiv D_0\ (!^w S)$ we have an induction as in $\multimap$-E in Figure 9.9, replacing P with $D[x]$. In many of these cases we may have structural rules mingled within the base trees in the figures, for example in a typing proof of a term S which ends in a $\multimap$-E step we might have

$$\dfrac{\begin{array}{c}\vdots\ \mathcal{P}_1\\ \mathcal{E} \vdash P : (!^w A) \multimap B\end{array} \quad \dfrac{\dfrac{\begin{array}{c}\vdots\ \mathcal{P}_2\\ \blacktriangleright_i \Gamma, \blacktriangleright_w \Delta \vdash S : A\end{array}}{\blacktriangleright_i \Gamma, \blacktriangleright_w \Delta \vdash !^w S : !^w A}\ !^w\text{-I}}{\blacktriangleright_i \Gamma, \blacktriangleright_w \Delta, x : A_0 \vdash !^w S : !^w A}\ \text{Wkning}}{\mathcal{E}, \blacktriangleright_i \Gamma, \blacktriangleright_w \Delta, x : A_0 \vdash P\ (!^w S) : B}\ \multimap\text{-E} \ ,$$

but it is clear that these rules can simply be shifted out of the way; in this example we have simply

$$\dfrac{\begin{array}{c}\vdots\ \mathcal{P}_1\\ \mathcal{E} \vdash P : (!^w A) \multimap B\end{array} \quad \dfrac{\dfrac{\begin{array}{c}\vdots\ \mathcal{P}_2\\ \blacktriangleright_i \Gamma, \blacktriangleright_w \Delta \vdash S : A\end{array}}{\blacktriangleright_i \Gamma, \blacktriangleright_w \Delta, x : A_0 \vdash S : A}\ \text{Wkning}}{\blacktriangleright_i \Gamma, \blacktriangleright_w \Delta, x : A_0 \vdash !^w S : !^w A}\ !^w\text{-I}}{\mathcal{E}, \blacktriangleright_i \Gamma, \blacktriangleright_w \Delta, x : A_0 \vdash P\ (!^w S) : B}\ \multimap\text{-E} \ .$$

$$
\dfrac{\dfrac{\vdots\,\mathcal{P}}{\mathcal{E} \vdash_{\mathsf{SLin}} M : B}}{\mathcal{E}, \blacktriangleright^{w} x : A \vdash_{\mathsf{SLin}} M : B}\ \text{Weakening}
\quad \xrightarrow{\natural,\bowtie} \quad
\dfrac{\dfrac{\dfrac{\vdots\,\mathcal{P}^{\natural}}{\mathcal{E}^{\natural} \vdash_{\mathsf{Need}} M^{\natural} : B^{\natural}}}{\mathcal{E}^{\natural}, x : A^{\bowtie} \vdash_{\mathsf{Need}} M^{\natural} : B^{\natural}}\ \text{Weakening}}{(\mathcal{E}, \blacktriangleright^{w} x : A)^{\natural} \vdash_{\mathsf{Need}} M^{\natural} : B^{\natural}}\ \equiv
$$

$$
\dfrac{\dfrac{\vdots\,\mathcal{P}}{\mathcal{E} \vdash_{\mathsf{SLin}} M : B}}{\mathcal{E}, \blacktriangleright^{i} x : A \vdash_{\mathsf{SLin}} M : B}\ \text{Weakening/C}
\quad \xrightarrow{\natural,\bowtie} \quad
\dfrac{\dfrac{\dfrac{\vdots\,\mathcal{P}^{\natural}}{\mathcal{E}^{\natural} \vdash_{\mathsf{Need}} M^{\natural} : B^{\natural}}}{\mathcal{E}^{\natural}, x : A^{\bowtie} \vdash_{\mathsf{Need}} M^{\natural} : B^{\natural}}\ \text{Weakening}}{(\mathcal{E}, \blacktriangleright^{i} x : A)^{\natural} \vdash_{\mathsf{Need}} M^{\natural} : B^{\natural}}\ \equiv
$$

$$
\dfrac{\dfrac{\vdots\,\mathcal{P}}{\mathcal{E}, \blacktriangleright^{c} x : A, y : A \vdash_{\mathsf{SLin}} M : B}}{\mathcal{E}, \blacktriangleright^{c} z : A \vdash_{\mathsf{SLin}} M[x := z, y := z] : B}\ \text{Contraction}
$$

$$
\xrightarrow{\natural,\bowtie} \quad
\dfrac{\dfrac{\dfrac{\dfrac{\vdots\,\mathcal{P}^{\natural}}{(\mathcal{E}, \blacktriangleright^{c} x : A, y : A)^{\natural} \vdash_{\mathsf{Need}} M^{\natural} : B^{\natural}}}{\mathcal{E}^{\natural}, x : A^{\bowtie}, y : A^{\bowtie} \vdash_{\mathsf{Need}} M^{\natural} : B^{\natural}}\ \equiv}{\mathcal{E}^{\natural}, z : A^{\bowtie} \vdash_{\mathsf{Need}} M^{\natural}[x := z, y := z] : B^{\natural}}\ \text{Cntr.}}{(\mathcal{E}, \blacktriangleright^{c} z : A)^{\natural} \vdash_{\mathsf{Need}} (M[x := z, y := z])^{\natural} : B^{\natural}}\ \equiv
$$

$$
\dfrac{\dfrac{\vdots\,\mathcal{P}}{\mathcal{E}, \blacktriangleright^{i} x : A, y : A \vdash_{\mathsf{SLin}} M : B}}{\mathcal{E}, \blacktriangleright^{i} z : A \vdash_{\mathsf{SLin}} M[x := z, y := z] : B}\ \text{Contraction/W}
$$

$$
\xrightarrow{\natural,\bowtie} \quad
\dfrac{\dfrac{\dfrac{\dfrac{\vdots\,\mathcal{P}^{\natural}}{(\mathcal{E}, \blacktriangleright^{i} x : A, y : A)^{\natural} \vdash_{\mathsf{Need}} M^{\natural} : B^{\natural}}}{\mathcal{E}^{\natural}, x : A^{\bowtie}, y : A^{\bowtie} \vdash_{\mathsf{Need}} M^{\natural} : B^{\natural}}\ \equiv}{\mathcal{E}^{\natural}, z : A^{\bowtie} \vdash_{\mathsf{Need}} M^{\natural}[x := z, y := z] : B^{\natural}}\ \text{Cntr.}}{(\mathcal{E}, \blacktriangleright^{i} z : A)^{\natural} \vdash_{\mathsf{Need}} (M[x := z, y := z])^{\natural} : B^{\natural}}\ \equiv
$$

Figure 9.7: Soundness of $\natural$ for (completeness of $\circledast$ from) Weakening and Contraction rules. In this and the next figure we slightly abuse our metavariable notation by writing M, N to range over any of the subsets S, U and so forth of SLin; we moreover handwave a bit by not writing out explicitly that the terms and types on the right-hand side of the type inference statements of the trees on the right-hand side of the arrows could also be translated with $\bowtie$.

$$\dfrac{\dfrac{\vdots\ \mathcal{P}}{\mathcal{E}, \blacktriangleright^{\iota} x : A \vdash_{\mathsf{SLin}} M : B}}{\mathcal{E}, \blacktriangleright^{w} x : A \vdash_{\mathsf{SLin}} M : B}\text{Dereliction!}^{w} \quad \xrightarrow{\natural,\bowtie} \quad \dfrac{\dfrac{\dfrac{\vdots\ \mathcal{P}^{\natural}}{(\mathcal{E}, \blacktriangleright^{\iota} x : A)^{\natural} \vdash_{\mathsf{Need}} M^{\natural} : B^{\natural}}}{\mathcal{E}^{\natural}, x : A^{\bowtie} \vdash_{\mathsf{Need}} M^{\natural} : B^{\natural}}\equiv}{(\mathcal{E}, \blacktriangleright^{w} x : A)^{\natural} \vdash_{\mathsf{Need}} M^{\natural} : B^{\natural}}\equiv$$

$$\dfrac{\dfrac{\vdots\ \mathcal{P}}{\mathcal{E}, \blacktriangleright^{\iota} x : A \vdash_{\mathsf{SLin}} M : B}}{\mathcal{E}, \blacktriangleright^{c} x : A \vdash_{\mathsf{SLin}} M : B}\text{Dereliction!}^{c} \quad \xrightarrow{\natural,\bowtie} \quad \dfrac{\dfrac{\dfrac{\vdots\ \mathcal{P}^{\natural}}{(\mathcal{E}, \blacktriangleright^{\iota} x : A)^{\natural} \vdash_{\mathsf{Need}} M^{\natural} : B^{\natural}}}{\mathcal{E}^{\natural}, x : A^{\bowtie} \vdash_{\mathsf{Need}} M^{\natural} : B^{\natural}}\equiv}{(\mathcal{E}, \blacktriangleright^{c} x : A)^{\natural} \vdash_{\mathsf{Need}} M^{\natural} : B^{\natural}}\equiv$$

$$\dfrac{\dfrac{\vdots\ \mathcal{P}}{\mathcal{E}, \blacktriangleright^{c} x : A \vdash_{\mathsf{SLin}} M : B}}{\mathcal{E}, \blacktriangleright^{i} x : A \vdash_{\mathsf{SLin}} M : B}\text{Dereliction!}^{w}/C \quad \xrightarrow{\natural,\bowtie} \quad \dfrac{\dfrac{\dfrac{\vdots\ \mathcal{P}^{\natural}}{(\mathcal{E}, \blacktriangleright^{c} x : A)^{\natural} \vdash_{\mathsf{Need}} M^{\natural} : B^{\natural}}}{\mathcal{E}^{\natural}, x : A^{\bowtie} \vdash_{\mathsf{Need}} M^{\natural} : B^{\natural}}\equiv}{(\mathcal{E}, \blacktriangleright^{i} x : A)^{\natural} \vdash_{\mathsf{Need}} M^{\natural} : B^{\natural}}\equiv$$

$$\dfrac{\dfrac{\vdots\ \mathcal{P}}{\mathcal{E}, \blacktriangleright^{w} x : A \vdash_{\mathsf{SLin}} M : B}}{\mathcal{E}, \blacktriangleright^{i} x : A \vdash_{\mathsf{SLin}} M : B}\text{Dereliction!}^{c}/W \quad \xrightarrow{\natural,\bowtie} \quad \dfrac{\dfrac{\dfrac{\vdots\ \mathcal{P}^{\natural}}{(\mathcal{E}, \blacktriangleright^{w} x : A)^{\natural} \vdash_{\mathsf{Need}} M^{\natural} : B^{\natural}}}{\mathcal{E}^{\natural}, x : A^{\bowtie} \vdash_{\mathsf{Need}} M^{\natural} : B^{\natural}}\equiv}{(\mathcal{E}, \blacktriangleright^{i} x : A)^{\natural} \vdash_{\mathsf{Need}} M^{\natural} : B^{\natural}}\equiv$$

Figure 9.8: Soundness of $\natural$ for (completeness of $\circledast$ from) Dereliction rules.

For Clause (iii), we have soundness for top level reduction of S terms as shown in Figure 9.12, and top-level P terms as shown in Figure 9.13. D and U terms are reducible only within their S- and P-subterms. Compatible closure is straightforward as the translations depend only on the top-level structure of terms, and not their subterms.

For Clause (iv), we have soundness with the preceding lemmas about preservation of answers and evaluation contexts as shown in Figures 9.14 and 9.15; compatible closure follows again by the preservation lemmas. ♦

Lemma 9.3.8. Let A be a call-by-need answer and E be a call-by-need evaluation context. Then $A^{\circledast}$ is a separated-linear answer, and if we take $[\,]^{\circledast} \equiv [\,]$ then $E^{\circledast}$ is a separated-linear evaluation context.

Proof: Simple, by induction on the size of the structures. ♦

Corollary 9.3.9. Let E be a call-by-need evaluation context. Then there exists a separated-linear evaluation context E_0 such that for all x we have $(E[x])^{\circledast} \equiv E_0[x]$.

Proof: This E_0 is simply $E^{\circledast}[!^{c}[\,]]$. ♦

Proposition 9.3.10. Let $M, N \in \mathsf{Need}$.

(i) $(M[x := V])^{\circledast} \equiv M^{\circledast}[x := V^{\oplus}]$.

(ii) $\Gamma \vdash_{\mathsf{Need}} M : A$ if and only if $\blacktriangleright^{i}\Gamma^{\circledast} \vdash_{\mathsf{SLin}} M^{\circledast} : A^{\circledast}$, and $\Gamma \vdash_{\mathsf{Need}} V : A$ if and only if $\blacktriangleright^{i}\Gamma^{\circledast} \vdash_{\mathsf{SLin}} V^{\oplus} : A^{\oplus}$.

(iii) $M \xrightarrow[\mathsf{Need}]{} \!\!\!\!\rightarrow N$ if and only if $M^{\circledast} \xrightarrow[\mathsf{SLin}]{} \!\!\!\!\rightarrow N^{\circledast}$.

(iv) $M \underset{\mathsf{Need}}{\longmapsto}\!\!\!\!\rightarrow N$ if and only if $M^{\circledast} \underset{\mathsf{SLin}}{\longmapsto}\!\!\!\!\rightarrow N^{\circledast}$.

$$\dfrac{\begin{array}{c}\vdots\,\mathcal{P}\\ {\blacktriangleright_i}\Gamma, {\blacktriangleright_c}\Phi \vdash_{\mathsf{SLin}} U : \bar{A}\end{array}}{{\blacktriangleright_i}\Gamma, {\blacktriangleright_c}\Phi \vdash_{\mathsf{SLin}} !^c U : !^c\bar{A}}\,!^c\text{-I} \quad \xrightarrow{\natural} \quad \dfrac{\begin{array}{c}\vdots\,\mathcal{P}^{\bowtie}\\ ({\blacktriangleright_i}\Gamma, {\blacktriangleright_c}\Phi)^\natural \vdash_{\mathsf{Need}} U^{\bowtie} : \bar{A}^{\bowtie}\end{array}}{({\blacktriangleright_i}\Gamma, {\blacktriangleright_c}\Phi)^\natural \vdash_{\mathsf{Need}} (!^c U)^\natural : (!^c\bar{A})^\natural}\equiv$$

$$\dfrac{\begin{array}{c}\vdots\,\mathcal{P}_1\\ \mathcal{E} \vdash_{\mathsf{SLin}} S :!^c\bar{A}_0\end{array} \quad \dfrac{\begin{array}{c}\vdots\,\mathcal{P}_2\\ \mathcal{F}, {\blacktriangleright_c}x : \bar{A}_0 \vdash_{\mathsf{SLin}} D[x] : (!^w A) \multimap B\end{array} \quad \dfrac{\begin{array}{c}\vdots\,\mathcal{P}_3\\ {\blacktriangleright_i}\Gamma, {\blacktriangleright_w}\Delta \vdash_{\mathsf{SLin}} T : A\end{array}}{{\blacktriangleright_i}\Gamma, {\blacktriangleright_w}\Delta \vdash_{\mathsf{SLin}} !^w T : !^w A}\,!^w\text{-I}}{\mathcal{F}, {\blacktriangleright_i}\Gamma, {\blacktriangleright_w}\Delta, {\blacktriangleright_c}x : \bar{A}_0 \vdash_{\mathsf{SLin}} D[x]\, !^w T : B}\,\multimap\text{-E}}{\mathcal{E}, \mathcal{F}, {\blacktriangleright_i}\Gamma, {\blacktriangleright_w}\Delta \vdash_{\mathsf{SLin}} \text{let } !^c x = S \text{ in } D[x]\, !^w T : B}\,!^c\text{-E}$$

$$\xrightarrow{\natural} \quad \dfrac{\dfrac{\begin{array}{c}\vdots\,\mathcal{R}\\ (\mathcal{E}, \mathcal{F})^\natural \vdash_{\mathsf{Need}} (D^{\bowtie}[S^\natural]) : A^\natural \to B^\natural\end{array} \quad \begin{array}{c}\vdots\,{\mathcal{P}_3}^\natural\\ ({\blacktriangleright_i}\Gamma, {\blacktriangleright_w}\Delta)^\natural \vdash_{\mathsf{Need}} T^\natural : A^\natural\end{array}}{(\mathcal{E}, \mathcal{F})^\natural, ({\blacktriangleright_i}\Gamma, {\blacktriangleright_w}\Delta)^\natural \vdash_{\mathsf{Need}} (D^{\bowtie}[S^\natural])\, T^\natural : B^\natural}\,\to\text{-E}}{(\mathcal{E}, \mathcal{F}, {\blacktriangleright_i}\Gamma, {\blacktriangleright_w}\Delta)^\natural \vdash_{\mathsf{Need}} (\text{let } !^c x = S \text{ in } D[x]\, !^w T)^\natural : B^\natural}\equiv$$

$$\text{where } \mathcal{R} \equiv \dfrac{\dfrac{\begin{array}{c}\vdots\,\mathcal{P}_1\\ \mathcal{E} \vdash_{\mathsf{SLin}} S :!^c\bar{A}_0\end{array} \quad \begin{array}{c}\vdots\,\mathcal{P}_2\\ \mathcal{F}, {\blacktriangleright_c}x : \bar{A}_0 \vdash_{\mathsf{SLin}} D[x] : (!^w A) \multimap B\end{array}}{(\mathcal{E}, \mathcal{F})^\natural \vdash_{\mathsf{Need}} (D^{\bowtie}[S^\natural]) : (!^c\bar{A})^\natural \to B^\natural}\,\text{Lemma 9.3.5}}{(\mathcal{E}, \mathcal{F})^\natural \vdash_{\mathsf{Need}} (D^{\bowtie}[S^\natural]) : A^\natural \to B^\natural}\equiv$$

$$\dfrac{\dfrac{\begin{array}{c}\vdots\,\mathcal{P}_1\\ \mathcal{E} \vdash_{\mathsf{SLin}} S : !^c\bar{A}\end{array}}{\mathcal{E} \vdash_{\mathsf{SLin}} !^w S : !^w !^c\bar{A}}\,!^w\text{-I} \quad \begin{array}{c}\vdots\,\mathcal{P}_2\\ {\blacktriangleright_i}\Gamma, {\blacktriangleright_w}\Delta, {\blacktriangleright_i}x : \bar{A} \vdash_{\mathsf{SLin}} T : B\end{array}}{\mathcal{E}, {\blacktriangleright_i}\Gamma, {\blacktriangleright_w}\Delta \vdash_{\mathsf{SLin}} \text{let } !^i x = !^w S \text{ in } T : B}\,!^i\text{-E}_1$$

$$\xrightarrow{\natural} \quad \dfrac{\dfrac{\dfrac{\begin{array}{c}\vdots\,{\mathcal{P}_1}^\natural\\ \mathcal{E}^\natural \vdash_{\mathsf{Need}} S^\natural : (!^c\bar{A})^\natural\end{array}}{\mathcal{E}^\natural \vdash_{\mathsf{Need}} S^\natural : \bar{A}^{\bowtie}}\equiv \quad \dfrac{\begin{array}{c}\vdots\,{\mathcal{P}_2}^\natural\\ ({\blacktriangleright_i}\Gamma, {\blacktriangleright_w}\Delta, {\blacktriangleright_i}x : \bar{A})^\natural \vdash_{\mathsf{Need}} T^\natural : B^\natural\end{array}}{({\blacktriangleright_i}\Gamma, {\blacktriangleright_w}\Delta)^\natural, x : \bar{A}^{\bowtie} \vdash_{\mathsf{Need}} T^\natural : B^\natural}\equiv}{\mathcal{E}^\natural, ({\blacktriangleright_i}\Gamma, {\blacktriangleright_w}\Delta)^\natural \vdash_{\mathsf{Need}} \text{let } x = S^\natural \text{ in } T^\natural : B^\natural}\,\text{Let}}{(\mathcal{E}, {\blacktriangleright_i}\Gamma, {\blacktriangleright_w}\Delta)^\natural \vdash_{\mathsf{Need}} (\text{let } !^i x = !^w S \text{ in } T)^\natural : B^\natural}\equiv$$

$$\dfrac{\begin{array}{c}\vdots\,\mathcal{P}_1\\ \mathcal{E} \vdash_{\mathsf{SLin}} P : (!^w A) \multimap B\end{array} \quad \dfrac{\begin{array}{c}\vdots\,\mathcal{P}_2\\ {\blacktriangleright_i}\Gamma, {\blacktriangleright_w}\Delta \vdash_{\mathsf{SLin}} S : A\end{array}}{{\blacktriangleright_i}\Gamma, {\blacktriangleright_w}\Delta \vdash_{\mathsf{SLin}} !^w S : !^w A}\,!^w\text{-I}}{\mathcal{E}, {\blacktriangleright_i}\Gamma, {\blacktriangleright_w}\Delta \vdash_{\mathsf{SLin}} P\,(!^w S) : B}\,\multimap\text{-E}$$

$$\xrightarrow{\natural} \quad \dfrac{\dfrac{\dfrac{\begin{array}{c}\vdots\,{\mathcal{P}_1}^\natural\\ \mathcal{E}^\natural \vdash_{\mathsf{Need}} P^\natural : ((!^w A) \multimap B)^\natural\end{array}}{\mathcal{E}^\natural \vdash_{\mathsf{Need}} P^\natural : A^\natural \to B^\natural}\equiv \quad \begin{array}{c}\vdots\,{\mathcal{P}_2}^\natural\\ ({\blacktriangleright_i}\Gamma, {\blacktriangleright_w}\Delta)^\natural \vdash_{\mathsf{Need}} S^\natural : A^\natural\end{array}}{\mathcal{E}^\natural, ({\blacktriangleright_i}\Gamma, {\blacktriangleright_w}\Delta)^\natural \vdash_{\mathsf{Need}} P^{\bowtie}\, S^\natural : B^\natural}\,\to\text{-E}}{(\mathcal{E}, {\blacktriangleright_i}\Gamma, {\blacktriangleright_w}\Delta)^\natural \vdash_{\mathsf{Need}} (P\,(!^w S))^\natural : B^\natural}\equiv$$

Figure 9.9: Soundness of $\natural$ for (completeness of $\circledast$ from) typing judgments for terms S.

$$\dfrac{\begin{array}{c}\vdots\,\mathcal{P}_1\\ \mathcal{E}\vdash_{\mathsf{SLin}} S : !^c\bar{A}\end{array}\quad \begin{array}{c}\vdots\,\mathcal{P}_2\\ \mathcal{F}, \blacktriangleright^{c} x:\bar{A}\vdash_{\mathsf{SLin}} D[x]:\bar{B}\end{array}}{\mathcal{E},\mathcal{F}\vdash_{\mathsf{SLin}} \text{let } !^c x = S \text{ in } D[x] : \bar{B}}\,!^c\text{-E}$$

$$\xrightarrow{\bowtie}\quad \dfrac{\dfrac{\begin{array}{c}\vdots\,\mathcal{P}_1\\ \mathcal{E}\vdash_{\mathsf{SLin}} S : !^c\bar{A}\end{array}\quad \begin{array}{c}\vdots\,\mathcal{P}_2\\ \mathcal{F}, \blacktriangleright^{c} x:\bar{A}\vdash_{\mathsf{SLin}} D[x]:\bar{B}\end{array}}{(\mathcal{E},\mathcal{F})^\natural \vdash_{\mathsf{Need}} D^{\bowtie}[S^\natural] : \bar{B}^{\bowtie}}\,\text{Lemma 9.3.5}}{(\mathcal{E},\mathcal{F})^\natural\vdash_{\mathsf{Need}} (\text{let } !^c x = S \text{ in } D[x])^{\bowtie} : \bar{B}^{\bowtie}}\,\equiv$$

$$\dfrac{\dfrac{\begin{array}{c}\vdots\,\mathcal{P}_1\\ \blacktriangleright^{i}\Gamma, \blacktriangleright^{w}\Delta\vdash_{\mathsf{SLin}} S : !^c\bar{A}\end{array}}{\blacktriangleright^{i}\Gamma, \blacktriangleright^{w}\Delta\vdash_{\mathsf{SLin}} !^w S : !^w !^c\bar{A}}\,!^w\text{-I}\quad \begin{array}{c}\vdots\,\mathcal{P}_2\\ \mathcal{F}, \blacktriangleright^{i} x:\bar{A}\vdash_{\mathsf{SLin}} P:\bar{B}\end{array}}{\blacktriangleright^{i}\Gamma, \blacktriangleright^{w}\Delta, \mathcal{F}\vdash_{\mathsf{SLin}} \text{let } !^i x = !^w S \text{ in } P : \bar{B}}\,!^i\text{-E}_1$$

$$\xrightarrow{\bowtie}\quad \dfrac{\dfrac{\dfrac{\begin{array}{c}\vdots\,{\mathcal{P}_1}^{\natural}\\ (\blacktriangleright^{i}\Gamma, \blacktriangleright^{w}\Delta)^\natural\vdash_{\mathsf{Need}} S^\natural : (!^c\bar{A})^\natural\end{array}}{(\blacktriangleright^{i}\Gamma, \blacktriangleright^{w}\Delta)^\natural\vdash_{\mathsf{Need}} S^\natural : \bar{A}^{\bowtie}}\,\equiv\quad \dfrac{\begin{array}{c}\vdots\,{\mathcal{P}_2}^{\bowtie}\\ (\mathcal{F}, \blacktriangleright^{i} x:\bar{A})^\natural\vdash_{\mathsf{Need}} P^{\bowtie}:\bar{B}^{\bowtie}\end{array}}{\mathcal{F}^\natural, x:\bar{A}^{\bowtie}\vdash_{\mathsf{Need}} P^{\bowtie}:\bar{B}^{\bowtie}}\,\equiv}{(\blacktriangleright^{i}\Gamma, \blacktriangleright^{w}\Delta)^\natural, \mathcal{F}^\natural\vdash_{\mathsf{Need}} \text{let } x = S^\natural \text{ in } P^{\bowtie} : \bar{B}^{\bowtie}}\,\text{Let}}{(\blacktriangleright^{i}\Gamma, \blacktriangleright^{w}\Delta, \mathcal{F})^\natural\vdash_{\mathsf{Need}} (\text{let } !^i x = !^w S \text{ in } P)^{\bowtie} : \bar{B}^{\bowtie}}\,\equiv$$

Figure 9.10: Soundness of $\bowtie$ for (completeness of $\oplus$ from) typing judgments for terms P.

$$\dfrac{}{\blacktriangleright^{l} x:\bar{A}\vdash_{\mathsf{SLin}} x:\bar{A}}\,\text{Id}\quad\xrightarrow{\bowtie}\quad\dfrac{}{x:\bar{A}^{\bowtie}\vdash_{\mathsf{Need}} x:\bar{A}^{\bowtie}}\,\text{Id}$$

$$\dfrac{\dfrac{\dfrac{\dfrac{}{\blacktriangleright^{l} y: !^w !^c\bar{A}\vdash_{\mathsf{SLin}} y: !^w !^c\bar{A}}\,\text{Id}\quad \begin{array}{c}\vdots\,\mathcal{P}\\ \mathcal{E}, \blacktriangleright^{i} x:\bar{A}\vdash_{\mathsf{SLin}} S:B\end{array}}{\mathcal{E}, \blacktriangleright^{l} y: !^w !^c\bar{A}\vdash_{\mathsf{SLin}} \text{let } !^i x = y \text{ in } S : B}\,!^i\text{-E}}{\mathcal{E}\vdash_{\mathsf{SLin}} \lambda y.\,\text{let } !^i x = y \text{ in } S : (!^w !^c\bar{A})\multimap B}\,\multimap\text{-I}}{\mathcal{E}\vdash_{\mathsf{SLin}} \lambda y.\,\text{let } !^i x = y \text{ in } S : (!^w A)\multimap B}\,\equiv$$

$$\xrightarrow{\bowtie}\quad \dfrac{\dfrac{\dfrac{\begin{array}{c}\vdots\,\mathcal{P}^\natural\\ (\mathcal{E}, \blacktriangleright^{i} x:\bar{A})^\natural\vdash_{\mathsf{Need}} S^\natural : B^\natural\end{array}}{\mathcal{E}^\natural, x:\bar{A}^{\bowtie}\vdash_{\mathsf{Need}} S^\natural : B^\natural}\,\equiv}{\mathcal{E}^\natural\vdash_{\mathsf{Need}} \lambda x.\,S^\natural : \bar{A}^{\bowtie}\to B^\natural}\,\to\text{-I}}{\mathcal{E}^\natural\vdash_{\mathsf{Need}} (\lambda y.\,\text{let } !^i x = y \text{ in } S)^{\bowtie} : ((!^w A)\multimap B)^{\bowtie}}\,\equiv$$

Figure 9.11: Soundness of $\bowtie$ for (completeness of $\oplus$ from) typing judgments for terms U.

$$\begin{array}{ll} & (\lambda y.\,\mathrm{let}\ !^i x = y\ \mathrm{in}\ S)\,(!^w T) \\ \xrightarrow[(\beta\multimap)]{} & \mathrm{let}\ !^i x = !^w T\ \mathrm{in}\ S \end{array} \quad \xrightarrow{\natural} \quad \begin{array}{ll} & ((\lambda y.\,\mathrm{let}\ !^i x = y\ \mathrm{in}\ S)\,(!^w T))^\natural \\ \equiv & (\lambda x.\,S^\natural)\,T^\natural \\ \xrightarrow[(I)]{} & \mathrm{let}\ x = T^\natural\ \mathrm{in}\ S^\natural \\ \equiv & (\mathrm{let}\ !^i x = !^w T\ \mathrm{in}\ S)^\natural \end{array}$$

$$\begin{array}{ll} & (\mathrm{let}\ !^i x = !^w S\ \mathrm{in}\ P)\,(!^w T) \\ \xrightarrow[(!^i\multimap)]{} & \mathrm{let}\ !^i x = !^w S\ \mathrm{in}\ P\ (!^w T) \end{array} \quad \xrightarrow{\natural} \quad \begin{array}{ll} & ((\mathrm{let}\ !^i x = !^w S\ \mathrm{in}\ P)\,(!^w T))^\natural \\ \equiv & (\mathrm{let}\ x = S^\natural\ \mathrm{in}\ P^\bowtie)\,T^\natural \\ \xrightarrow[(C)]{} & \mathrm{let}\ x = S^\natural\ \mathrm{in}\ P^\bowtie\,T^\natural \\ \equiv & (\mathrm{let}\ !^i x = !^w S\ \mathrm{in}\ P\ (!^w T))^\natural \end{array}$$

$$\begin{array}{ll} & (\mathrm{let}\ !^c x = S\ \mathrm{in}\ D[x])\,(!^w T) \\ \xrightarrow[(!^c\multimap)]{} & \mathrm{let}\ !^c x = S\ \mathrm{in}\ D[x]\ (!^w T) \end{array} \quad \xrightarrow{\natural} \quad \begin{array}{ll} & ((\mathrm{let}\ !^c x = S\ \mathrm{in}\ D[x])\,(!^w T))^\natural \\ \equiv & D^\bowtie[S^\natural]\,T^\natural \\ \equiv & (\mathrm{let}\ !^c x = S\ \mathrm{in}\ D[x]\ (!^w T))^\natural \end{array}$$

$$\begin{array}{ll} & \mathrm{let}\ !^i x = !^w !^c U\ \mathrm{in}\ S \\ \xrightarrow[(\beta !^i)]{} & S[x := U] \end{array} \quad \xrightarrow{\natural} \quad \begin{array}{lll} & (\mathrm{let}\ !^i x = !^w !^c U\ \mathrm{in}\ S)^\natural & \\ \equiv & \mathrm{let}\ x = U^\bowtie\ \mathrm{in}\ S^\natural & \\ \xrightarrow[(V)]{} & S^\natural[x := U^\bowtie] & \text{by Lemma 9.3.4} \\ \equiv & (S[x := U])^\natural & \text{by Lemma 9.3.7.(i)} \end{array}$$

$$\begin{array}{ll} & \mathrm{let}\ !^c x = !^c U\ \mathrm{in}\ D[x]\ !^w S \\ \xrightarrow[(\beta !^c)]{} & D[U]\ !^w S \end{array} \quad \xrightarrow{\natural} \quad \begin{array}{ll} & (\mathrm{let}\ !^c x = !^c U\ \mathrm{in}\ D[x]\ !^w S)^\natural \\ \equiv & D^\bowtie[U^\bowtie]\ S^\natural \\ \equiv & (D[U]\ !^w S)^\natural \end{array}$$

$$\begin{array}{ll} & \mathrm{let}\ !^i x = !^w(\mathrm{let}\ !^i y = !^w S\ \mathrm{in}\ T)\ \mathrm{in}\ T_0 \\ \xrightarrow[(!^i !^w !^w)]{} & \mathrm{let}\ !^i y = !^w S\ \mathrm{in}\ \mathrm{let}\ !^i x = !^w T\ \mathrm{in}\ T_0 \end{array}$$

$$\xrightarrow{\natural} \quad \begin{array}{ll} & (\mathrm{let}\ !^i x = !^w(\mathrm{let}\ !^i y = !^w S\ \mathrm{in}\ T)\ \mathrm{in}\ T_0)^\natural \\ \equiv & \mathrm{let}\ x = (\mathrm{let}\ y = S^\natural\ \mathrm{in}\ T^\natural)\ \mathrm{in}\ T_0^\natural \\ \xrightarrow[(A)]{} & \mathrm{let}\ y = S^\natural\ \mathrm{in}\ \mathrm{let}\ x = T^\natural\ \mathrm{in}\ T_0^\natural \\ \equiv & (\mathrm{let}\ !^i y = !^w S\ \mathrm{in}\ \mathrm{let}\ !^i x = !^w T\ \mathrm{in}\ T_0)^\natural \end{array}$$

$$\begin{array}{ll} & \mathrm{let}\ !^c x = (\mathrm{let}\ !^i y = !^w S_0\ \mathrm{in}\ S_1)\ \mathrm{in}\ D_1[x]\ !^w T_1 \\ \xrightarrow[(!^c !^i)]{} & \mathrm{let}\ !^i y = !^w S_0\ \mathrm{in}\ \mathrm{let}\ !^c x = S_1\ \mathrm{in}\ D_1[x]\ !^w T_1 \end{array}$$

$$\xrightarrow{\natural} \quad \begin{array}{ll} & (\mathrm{let}\ !^c x = (\mathrm{let}\ !^i y = !^w S_0\ \mathrm{in}\ S_1)\ \mathrm{in}\ D_1[x]\ !^w T_1)^\natural \\ \equiv & \mathrm{let}\ x = (\mathrm{let}\ y = S_0^\natural\ \mathrm{in}\ S_1^\natural)\ \mathrm{in}\ D_1^\bowtie[x]\ T_1^\natural \\ \xrightarrow[(A)]{} & \mathrm{let}\ y = S_0^\natural\ \mathrm{in}\ \mathrm{let}\ x = S_1^\natural\ \mathrm{in}\ D_1^\bowtie[x]\ T_1^\natural \\ \equiv & (\mathrm{let}\ !^i y = !^w S_0\ \mathrm{in}\ \mathrm{let}\ !^c x = S_1\ \mathrm{in}\ D_1[x]\ !^w T_1)^\natural \end{array}$$

$$\begin{array}{ll} & \mathrm{let}\ !^c x = (\mathrm{let}\ !^c y = S\ \mathrm{in}\ D_0[y]\ !^w T_0)\ \mathrm{in}\ D_1[x]\ !^w T_1 \\ \xrightarrow[(!^c !^c)]{} & \mathrm{let}\ !^c y = S\ \mathrm{in}\ \mathrm{let}\ !^c x = D_0[y]\ !^w T_0\ \mathrm{in}\ D_1[x]\ !^w T_1 \end{array}$$

$$\xrightarrow{\natural} \quad \begin{array}{ll} & (\mathrm{let}\ !^c x = (\mathrm{let}\ !^c y = S\ \mathrm{in}\ D_0[y]\ !^w T_0)\ \mathrm{in}\ D_1[x]\ !^w T_1)^\natural \\ \equiv & \mathrm{let}\ x = (\mathrm{let}\ y = S^\natural\ \mathrm{in}\ D_0^\bowtie[y]\ T_0^\natural)\ \mathrm{in}\ D_1^\natural[x]\ T_1^\natural \\ \xrightarrow[(A)]{} & \mathrm{let}\ y = S^\natural\ \mathrm{in}\ \mathrm{let}\ x = D_0^\bowtie[y]\ T_0^\natural\ \mathrm{in}\ D_1^\natural[x]\ T_1^\natural \\ \equiv & (\mathrm{let}\ !^c y = S\ \mathrm{in}\ \mathrm{let}\ !^c x = D_0[y]\ !^w T_0\ \mathrm{in}\ D_1[x]\ !^w T_1)^\natural \end{array}$$

$$\begin{array}{ll} & \mathrm{let}\ !^i x = !^w S\ \mathrm{in}\ T \\ \xrightarrow[(\mathrm{W}!^i)]{} & T \\ & \text{where } x \notin \mathrm{fv}(T) \end{array} \quad \xrightarrow{\natural} \quad \begin{array}{ll} & (\mathrm{let}\ !^i x = !^w S\ \mathrm{in}\ T)^\natural \\ \equiv & \mathrm{let}\ x = S^\natural\ \mathrm{in}\ T^\natural \\ \xrightarrow[(G)]{} & T^\natural \end{array}$$

Figure 9.12: Soundness of $\natural$ for reduction of terms S.

$$
\begin{array}{ll}
 & \text{let } !^i x = !^w(\text{let } !^i y = !^w S_0 \text{ in } S_1) \text{ in } P \\
\xrightarrow[(!^i!^w!^w)]{} & \text{let } !^i y = !^w S_0 \text{ in let } !^i x = !^w S_1 \text{ in } P
\end{array}
$$

$$
\xrightarrow{\bowtie}
\begin{array}{ll}
 & (\text{let } !^i x = !^w(\text{let } !^i y = !^w S_0 \text{ in } S_1) \text{ in } P)^{\bowtie} \\
\equiv & \text{let } x = (\text{let } y = S_0^{\natural} \text{ in } S_1^{\natural}) \text{ in } P^{\bowtie} \\
\xrightarrow[(A)]{} & \text{let } y = S_0^{\natural} \text{ in let } x = S_1^{\natural} \text{ in } P^{\bowtie} \\
\equiv & (\text{let } !^i y = !^w S_0 \text{ in let } !^i x = !^w S_1 \text{ in } P)^{\bowtie}
\end{array}
$$

$$
\begin{array}{ll}
 & \text{let } !^i x = !^w !^c U \text{ in } P \\
\xrightarrow[(\beta !^i)]{} & P[x := U]
\end{array}
$$

$$
\xrightarrow{\bowtie}
\begin{array}{lll}
 & (\text{let } !^i x = !^w !^c U \text{ in } P)^{\bowtie} & \\
\equiv & \text{let } x = U^{\bowtie} \text{ in } P^{\bowtie} & \\
\xrightarrow[(V)]{} & P^{\bowtie}[x := U^{\bowtie}] & \text{by Lemma 9.3.4} \\
\equiv & (P[x := U])^{\bowtie} &
\end{array}
$$

$$
\begin{array}{ll}
 & \text{let } !^c x = (\text{let } !^i y = !^w S_0 \text{ in } S_1) \text{ in } D[x] \\
\xrightarrow[(!^c!^i)]{} & \text{let } !^i y = !^w S_0 \text{ in let } !^c x = S_1 \text{ in } D[x]
\end{array}
$$

$$
\xrightarrow{\bowtie}
\begin{array}{lll}
 & (\text{let } !^c x = (\text{let } !^i y = !^w S_0 \text{ in } S_1) \text{ in } D[x])^{\bowtie} & \\
\equiv & D^{\bowtie}[\text{let } y = S_0^{\natural} \text{ in } S_1^{\natural}] & \\
\xrightarrow[(C)]{} & \text{let } y = S_0^{\natural} \text{ in } D^{\bowtie}[S_1^{\natural}] & \text{by Lemma 9.3.6} \\
\equiv & (\text{let } !^i y = !^w S_0 \text{ in let } !^c x = S_1 \text{ in } D[x])^{\bowtie} &
\end{array}
$$

$$
\begin{array}{ll}
 & \text{let } !^c x = !^c U \text{ in } D[x] \\
\xrightarrow[(\beta !^i)]{} & D[U]
\end{array}
\quad
\xrightarrow{\bowtie}
\begin{array}{ll}
 & (\text{let } !^c x = !^c U \text{ in } D[x])^{\bowtie} \\
\equiv & D^{\bowtie}[U^{\bowtie}] \\
\equiv & D[U]^{\bowtie}
\end{array}
$$

$$
\begin{array}{ll}
 & \text{let } !^i x = !^w S \text{ in } P \\
\xrightarrow[(\mathrm{W}!^i)]{} & P \\
 & \text{where } x \notin \mathrm{fv}(P)
\end{array}
\quad
\xrightarrow{\bowtie}
\begin{array}{ll}
 & (\text{let } !^i x = !^w S \text{ in } P)^{\bowtie} \\
\equiv & \text{let } x = S^{\natural} \text{ in } P^{\bowtie} \\
\xrightarrow[(G)]{} & P^{\bowtie}
\end{array}
$$

Figure 9.13: Soundness of $\bowtie$ for reduction of terms P.

$$
\begin{array}{rl}
 & (\lambda y.\,\mathrm{let}\ !^{i}x = y\ \mathrm{in}\ \mathcal{S})\ (!^{w}\mathcal{T}) \\
\xmapsto[(\beta\multimap)]{} & \mathrm{let}\ !^{i}x = !^{w}\mathcal{T}\ \mathrm{in}\ \mathcal{S}
\end{array}
$$

$$
\xrightarrow{\natural}\quad
\begin{array}{rl}
 & ((\lambda y.\,\mathrm{let}\ !^{i}x = y\ \mathrm{in}\ \mathcal{S})\ (!^{w}\mathcal{T}))^{\natural} \\
\equiv & (\lambda x.\,\mathcal{S}^{\natural})\ \mathcal{T}^{\natural} \\
\xmapsto[(I)]{} & \mathrm{let}\ x = \mathcal{T}^{\natural}\ \mathrm{in}\ \mathcal{S}^{\natural} \\
\equiv & (\mathrm{let}\ !^{i}x = !^{w}\mathcal{T}\ \mathrm{in}\ \mathcal{S})^{\natural}
\end{array}
$$

$$
\begin{array}{rl}
 & (\mathrm{let}\ !^{i}x = !^{w}\mathcal{S}\ \mathrm{in}\ \mathcal{A})\ (!^{w}\mathcal{T}) \\
\xmapsto[(!^{i}\multimap)]{} & \mathrm{let}\ !^{i}x = !^{w}\mathcal{S}\ \mathrm{in}\ \mathcal{A}\ (!^{w}\mathcal{T})
\end{array}
$$

$$
\xrightarrow{\natural}\quad
\begin{array}{rll}
 & ((\mathrm{let}\ !^{i}x = !^{w}\mathcal{S}\ \mathrm{in}\ \mathcal{A})\ (!^{w}\mathcal{T}))^{\natural} & \\
\equiv & (\mathrm{let}\ x = \mathcal{S}^{\natural}\ \mathrm{in}\ \mathcal{A}^{\bowtie})\ \mathcal{T}^{\natural} & \\
\xmapsto[(C)]{} & \mathrm{let}\ x = \mathcal{S}^{\natural}\ \mathrm{in}\ \mathcal{A}^{\bowtie}\ \mathcal{T}^{\natural} & \text{Lemma 9.3.4} \\
\equiv & (\mathrm{let}\ !^{i}x = !^{w}\mathcal{S}\ \mathrm{in}\ \mathcal{A}\ (!^{w}\mathcal{T}))^{\natural} &
\end{array}
$$

$$
\begin{array}{rl}
 & \mathrm{let}\ !^{i}x = !^{w}!^{c}\mathcal{U}\ \mathrm{in}\ \mathcal{E}[x] \\
\xmapsto[(\beta!^{i})]{} & \mathcal{E}[x][x := \mathcal{U}]
\end{array}
$$

$$
\xrightarrow{\natural}\quad
\begin{array}{rll}
 & (\mathrm{let}\ !^{i}x = !^{w}!^{c}\mathcal{U}\ \mathrm{in}\ \mathcal{E}[x])^{\natural} & \\
\equiv & \mathrm{let}\ x = \mathcal{U}^{\bowtie}\ \mathrm{in}\ \mathcal{E}[x]^{\natural} & \\
\xmapsto[(V)]{} & \mathcal{E}[x]^{\natural}[x := \mathcal{U}^{\bowtie}] & \text{Lemma 9.3.4} \\
\equiv & (\mathcal{E}[x][x := \mathcal{U}])^{\natural} & \text{Lemma 9.3.7.(i)}
\end{array}
$$

$$
\begin{array}{rl}
 & \mathrm{let}\ !^{i}x = !^{w}(\mathrm{let}\ !^{i}y = !^{w}\mathcal{S}\ \mathrm{in}\ \mathcal{A})\ \mathrm{in}\ \mathcal{E}[x] \\
\xmapsto[(!^{i}!^{w}!^{w})]{} & \mathrm{let}\ !^{i}y = !^{w}\mathcal{S}\ \mathrm{in}\ \mathrm{let}\ !^{i}x = !^{w}\mathcal{A}\ \mathrm{in}\ \mathcal{E}[x]
\end{array}
$$

$$
\xrightarrow{\natural}\quad
\begin{array}{rl}
 & (\mathrm{let}\ !^{i}x = !^{w}(\mathrm{let}\ !^{i}y = !^{w}\mathcal{S}\ \mathrm{in}\ \mathcal{A})\ \mathrm{in}\ \mathcal{E}[x])^{\natural} \\
\equiv & \mathrm{let}\ x = (\mathrm{let}\ y = \mathcal{S}^{\natural}\ \mathrm{in}\ \mathcal{A}^{\natural})\ \mathrm{in}\ \mathcal{E}[x]^{\natural} \\
\xmapsto[(A)]{} & \mathrm{let}\ y = \mathcal{S}^{\natural}\ \mathrm{in}\ \mathrm{let}\ x = \mathcal{A}^{\natural}\ \mathrm{in}\ \mathcal{E}[x]^{\natural} \\
\equiv & (\mathrm{let}\ !^{i}y = !^{w}\mathcal{S}\ \mathrm{in}\ \mathrm{let}\ !^{i}x = !^{w}\mathcal{A}\ \mathrm{in}\ \mathcal{E}[x])^{\natural}
\end{array}
$$

Figure 9.14: Soundness of $\natural$ for evaluation of terms S.

$$
\begin{array}{ll}
 & \text{let } !^i x = (\text{let } !^i y = !^w \mathcal{S} \text{ in } \mathcal{A}) \text{ in } \mathcal{G}[x] \\
\xmapsto[(!^i !^w)]{} & \text{let } !^i y = !^w \mathcal{S} \text{ in let } !^i x = \mathcal{A} \text{ in } \mathcal{G}[x]
\end{array}
$$

$$
\begin{array}{lll}
 & & (\text{let } !^i x = (\text{let } !^i y = !^w \mathcal{S} \text{ in } \mathcal{A}) \text{ in } \mathcal{G}[x])^{\bowtie} \\
 & \equiv & \text{let } x = (\text{let } y = \mathcal{S}^{\natural} \text{ in } \mathcal{A}^{\natural}) \text{ in } \mathcal{G}[x]^{\bowtie} \\
\xrightarrow{\bowtie} & \xmapsto[(A)]{} & \text{let } y = \mathcal{S}^{\natural} \text{ in let } x = \mathcal{A}^{\natural} \text{ in } \mathcal{G}[x]^{\bowtie} \quad \text{Lemma 9.3.4} \\
 & \equiv & (\text{let } !^i y = !^w \mathcal{S} \text{ in let } !^i x = \mathcal{A} \text{ in } \mathcal{G}[x])^{\bowtie}
\end{array}
$$

$$
\begin{array}{ll}
 & \text{let } !^i x = !^w !^c \mathcal{U} \text{ in } \mathcal{G}[x] \\
\xmapsto[(\beta !^i)]{} & \mathcal{G}[x][x := \mathcal{U}]
\end{array}
$$

$$
\begin{array}{lll}
 & & (\text{let } !^i x = !^w !^c \mathcal{U} \text{ in } \mathcal{G}[x])^{\bowtie} \\
 & \equiv & \text{let } x = \mathcal{U}^{\bowtie} \text{ in } \mathcal{G}[x]^{\bowtie} \\
\xrightarrow{\bowtie} & \xmapsto[(V)]{} & \mathcal{G}[x]^{\bowtie}[x := \mathcal{U}^{\bowtie}] \qquad \text{Lemma 9.3.4} \\
 & \equiv & (\mathcal{G}[x][x := \mathcal{U}])^{\bowtie}
\end{array}
$$

$$
\begin{array}{ll}
 & \text{let } !^c x = !^w (\text{let } !^i y = !^w \mathcal{S} \text{ in } \mathcal{A}) \text{ in } x \\
\xmapsto[(!^i !^w !^w)]{} & \text{let } !^i y = !^w \mathcal{S} \text{ in let } !^c x = !^w \mathcal{A} \text{ in } x
\end{array}
$$

$$
\begin{array}{lll}
 & & (\text{let } !^c x = !^w (\text{let } !^i y = !^w \mathcal{S} \text{ in } \mathcal{A}) \text{ in } x)^{\bowtie} \\
\xrightarrow{\bowtie} & \equiv & \text{let } y = \mathcal{S}^{\natural} \text{ in } \mathcal{A}^{\natural} \\
 & \equiv & (\text{let } !^i y = !^w \mathcal{S} \text{ in let } !^c x = !^w \mathcal{A} \text{ in } x)^{\bowtie}
\end{array}
$$

$$
\begin{array}{ll}
 & \text{let } !^c x = !^c \mathcal{U} \text{ in } x \\
\xmapsto[(\beta !^i)]{} & \mathcal{U}
\end{array}
$$

$$
\begin{array}{lll}
 & & (\text{let } !^c x = !^c \mathcal{U} \text{ in } x)^{\bowtie} \\
\xrightarrow{\bowtie} & \equiv & \mathcal{U}^{\bowtie}
\end{array}
$$

Figure 9.15: Soundness of $\bowtie$ for evaluation of terms P.

$$
\dfrac{}{x : A \vdash_{\mathsf{Need}} x : A}\,\text{Id}
\quad\xrightarrow{\circledast}\quad
\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\blacktriangleright^{l}\, x : A^{\oplus} \vdash_{\mathsf{SLin}} x : A^{\oplus}}\,\text{Id}}{\blacktriangleright^{c}\, x : A^{\oplus} \vdash_{\mathsf{SLin}} x : A^{\oplus}}\,\text{Dereliction!}^{c}}{\blacktriangleright^{c}\, x : A^{\oplus} \vdash_{\mathsf{SLin}} !^{c}x :!^{c}A^{\oplus}}\,!^{c}\text{-I}}{\blacktriangleright^{i}\, x : A^{\oplus} \vdash_{\mathsf{SLin}} !^{c}x :!^{c}A^{\oplus}}\,\text{Dereliction!}^{w}/C}{\blacktriangleright^{i}\, (x : A)^{\circledast} \vdash_{\mathsf{SLin}} x^{\circledast} : A^{\circledast}}\,\equiv
$$

$$
\dfrac{\begin{array}{c}\vdots\ \mathcal{P}\\ \Gamma \vdash_{\mathsf{Need}} M : A\end{array}}{\Gamma, x : B \vdash_{\mathsf{Need}} M : A}\,\text{Weakening}
\quad\xrightarrow{\circledast}\quad
\dfrac{\dfrac{\begin{array}{c}\vdots\ \mathcal{P}^{\circledast}\\ \blacktriangleright^{i}\, \Gamma^{\circledast} \vdash_{\mathsf{SLin}} M^{\circledast} : A^{\circledast}\end{array}}{\blacktriangleright^{i}\, \Gamma^{\circledast}, x : B^{\oplus} \vdash_{\mathsf{SLin}} M^{\circledast} : A^{\circledast}}\,\text{Weakening}/C}{\blacktriangleright^{i}\, (\Gamma, x : B)^{\circledast} \vdash_{\mathsf{SLin}} M^{\circledast} : A^{\circledast}}\,\equiv
$$

$$
\dfrac{\begin{array}{c}\vdots\ \mathcal{P}\\ \Gamma, x : B, y : B \vdash_{\mathsf{Need}} M : A\end{array}}{\Gamma, z : B \vdash_{\mathsf{Need}} M[x := z, y := z] : A}\,\text{Contraction}
$$

$$
\xrightarrow{\circledast}\quad
\dfrac{\dfrac{\dfrac{\begin{array}{c}\vdots\ \mathcal{P}^{\circledast}\\ \blacktriangleright^{i}\, (\Gamma, x : B, y : B)^{\circledast} \vdash_{\mathsf{Need}} M^{\circledast} : A^{\circledast}\end{array}}{\blacktriangleright^{i}\, \Gamma^{\circledast}, x : B^{\oplus}, y : B^{\oplus} \vdash_{\mathsf{Need}} M^{\circledast} : A^{\circledast}}\,\equiv}{\blacktriangleright^{i}\, \Gamma^{\circledast}, z : B^{\oplus} \vdash_{\mathsf{Need}} M^{\circledast}[x := z, y := z] : A^{\circledast}}\,\text{Contraction}/W}{\blacktriangleright^{i}\, (\Gamma, z : B)^{\circledast} \vdash_{\mathsf{Need}} (M[x := z, y := z])^{\circledast} : A^{\circledast}}\,\equiv
$$

Figure 9.16: Soundness of $\circledast$ for structural typing rules.

Proof: Clause (i) is trivial since all values are translated with the same prefix. Soundness in Clause (ii) follows by a straightforward induction on the depth of the proof tree; we illustrate the reasoning for each Need typing rule in Figures 9.16 and 9.17. Soundness in Clause (iii) follows by a similar analysis of each rule, shown in Figure 9.18. Finally, soundness for Clause (iv) follows by the analyses of each rule in Figure 9.19, Lemma 9.3.8 and Corollary 9.3.9.

For completeness we rely on the subset of SLin and mapping from it to Need given in Figures 9.5 and 9.6. It is easy to verify that this grammar includes all $\circledast$-images of Need terms and is closed under general SLin reduction, that the mapping $\natural$ is right-inverse to $\circledast$, and that $\natural$ is sound for typing proofs and reduction sequences, which together imply completeness in Clauses (ii) and (iii). Finally, completeness in Clause (iv) follows since $\natural$ takes SLin answers and evaluation contexts to Need answers and evaluation contexts as well. ♦

Why is $\circledast$ complete for general reduction from Need into SLin while $*$ from Val into Lin, from Let into Lin, or from Need into Aff, which produce similar reduction-closed images of terms, are not? The counterexamples for completeness of $*$ for arbitrary reduction sequences exploits the fact that we cannot distinguish an eliminator introduced by the translation for massaging the left-hand side of an application from an eliminator which is the translation of a let-binding or an abstraction. Under $\circledast$ we do not have this confusion: the former is translated as a $!^{c}$-eliminator while the latter becomes a $!^{i}$-eliminator, which can be distinguished in the inverse translation.

$$\dfrac{\begin{matrix}\vdots\;\mathcal{P}\\ \Gamma, x : A \vdash_{\mathsf{Need}} M : B\end{matrix}}{\Gamma \vdash_{\mathsf{Need}} \lambda x.\, M : A \to B}\;\to\text{-I}$$

$$\overset{\circledast}{\longrightarrow}\quad \dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\blacktriangleright_{l}\, y :!^{w} A^{\circledast} \vdash_{\mathsf{SLin}} y :!^{w}!^{c} A^{\oplus}}\;\text{Id} \qquad \begin{matrix}\vdots\;\mathcal{P}^{\circledast}\\ \blacktriangleright_{i}\, \Gamma^{\circledast}, x : A^{\oplus} \vdash_{\mathsf{SLin}} M^{\circledast} : B^{\circledast}\end{matrix}}{\blacktriangleright_{i}\, \Gamma^{\circledast}, \blacktriangleright_{l}\, y :!^{w} A^{\circledast} \vdash_{\mathsf{SLin}} \text{let } !^{i}x = y \text{ in } M^{\circledast} : B^{\circledast}}\;!^{i}\text{-E}}{\blacktriangleright_{i}\, \Gamma^{\circledast} \vdash_{\mathsf{SLin}} \lambda y.\, \text{let } !^{i}x = y \text{ in } M^{\circledast} : (!^{w} A^{\circledast}) \multimap B^{\circledast}}\;\multimap\text{-I}}{\blacktriangleright_{i}\, \Gamma^{\circledast} \vdash_{\mathsf{SLin}} !^{c}\lambda y.\, \text{let } !^{i}x = y \text{ in } M^{\circledast} : !^{c}((!^{w} A^{\circledast}) \multimap B^{\circledast})}\;!^{c}\text{-I}}{\blacktriangleright_{i}\, \Gamma \vdash_{\mathsf{SLin}} (\lambda x.\, M)^{\circledast} : (A \to B)^{\circledast}}\;\equiv$$

$$\dfrac{\begin{matrix}\vdots\;\mathcal{P}_1\\ \Gamma \vdash_{\mathsf{Need}} M : A \to B\end{matrix} \quad \begin{matrix}\vdots\;\mathcal{P}_2\\ \Delta \vdash_{\mathsf{Need}} N : A\end{matrix}}{\Gamma, \Delta \vdash_{\mathsf{Need}} M\ N : B}\;\to\text{-E}$$

$$\overset{\circledast}{\longrightarrow}\quad \dfrac{\dfrac{\dfrac{\dfrac{\begin{matrix}\vdots\;\mathcal{P}_1^{\circledast}\\ \blacktriangleright_{i}\, \Gamma^{\circledast} \vdash_{\mathsf{SLin}} M^{\circledast} : (A \to B)^{\circledast}\end{matrix}}{\blacktriangleright_{i}\, \Gamma^{\circledast} \vdash_{\mathsf{SLin}} M^{\circledast} :!^{c}((!^{w} A^{\circledast}) \multimap B^{\circledast})}\;\equiv \qquad \mathcal{R}}{\blacktriangleright_{i}\, \Gamma^{\circledast} \vdash_{\mathsf{SLin}} (\text{let } !^{c}z = M^{\circledast} \text{ in } z) : (!^{w} A^{\circledast}) \multimap B^{\circledast}}\;!^{c}\text{-E} \qquad \begin{matrix}\vdots\;\mathcal{P}_2^{\circledast}\\ \Delta^{\circledast} \vdash_{\mathsf{SLin}} N^{\circledast} : A^{\circledast}\end{matrix}}{\blacktriangleright_{i}\, \Gamma^{\circledast}, \Delta^{\circledast} \vdash_{\mathsf{SLin}} (\text{let } !^{c}z = M^{\circledast} \text{ in } z)\ N^{\circledast} : B^{\circledast}}\;\multimap\text{-E}}{\blacktriangleright_{i}\, (\Gamma, \Delta)^{\circledast} \vdash_{\mathsf{SLin}} (M\ N)^{\circledast} : B^{\circledast}}\;\equiv$$

where $\mathcal{R} \equiv \dfrac{\dfrac{}{\blacktriangleright_{l}\, z : (!^{w} A^{\circledast}) \multimap B^{\circledast} \vdash_{\mathsf{SLin}} z : (!^{w} A^{\circledast}) \multimap B^{\circledast}}\;\text{Id}}{\blacktriangleright_{c}\, z : (!^{w} A^{\circledast}) \multimap B^{\circledast} \vdash_{\mathsf{SLin}} z : (!^{w} A^{\circledast}) \multimap B^{\circledast}}\;\text{Dereliction}!^{c}$

$$\dfrac{\begin{matrix}\vdots\;\mathcal{P}_1\\ \Gamma \vdash_{\mathsf{Need}} M : B\end{matrix} \quad \begin{matrix}\vdots\;\mathcal{P}_2\\ \Delta, x : B \vdash_{\mathsf{Need}} N : A\end{matrix}}{\Gamma, \Delta \vdash_{\mathsf{Need}} \text{let } x = M \text{ in } N : A}\;\to\text{Let}$$

$$\overset{\circledast}{\longrightarrow}\quad \dfrac{\dfrac{\dfrac{\dfrac{\begin{matrix}\vdots\;\mathcal{P}_1^{\circledast}\\ \blacktriangleright_{i}\, \Gamma^{\circledast} \vdash_{\mathsf{SLin}} M^{\circledast} : B^{\circledast}\end{matrix}}{\blacktriangleright_{i}\, \Gamma^{\circledast} \vdash_{\mathsf{SLin}} !^{w} M^{\circledast} :!^{w} B^{\circledast}}\;!^{w}\text{-I}}{\blacktriangleright_{i}\, \Gamma^{\circledast} \vdash_{\mathsf{SLin}} !^{w} M^{\circledast} :!^{w}!^{c} B^{\oplus}}\;\equiv \qquad \dfrac{\begin{matrix}\vdots\;\mathcal{P}_2^{\circledast}\\ \blacktriangleright_{i}\, (\Delta, x : B)^{\circledast} \vdash_{\mathsf{SLin}} N^{\circledast} : A^{\circledast}\end{matrix}}{\blacktriangleright_{i}\, \Delta^{\circledast}, x : B^{\oplus} \vdash_{\mathsf{SLin}} N^{\circledast} : A^{\circledast}}\;\equiv}{\blacktriangleright_{i}\, \Gamma^{\circledast}, \Delta^{\circledast} \vdash_{\mathsf{SLin}} (\text{let } !^{i}x = !^{w} M^{\circledast} \text{ in } N^{\circledast}) : A^{\circledast}}\;\to\text{Let}}{\blacktriangleright_{i}\, (\Gamma, \Delta)^{\circledast} \vdash_{\mathsf{SLin}} (\text{let } x = M \text{ in } N)^{\circledast} : A^{\circledast}}\;\equiv$$

Figure 9.17: Soundness of $\circledast$ for logical typing rules.

$$
\begin{array}{ll}
 & (\lambda x.\, M)\; N \\
\xrightarrow[(I)]{} & \mathrm{let}\; x = N \;\mathrm{in}\; M
\end{array}
$$

$$
\begin{array}{lll}
 & & ((\lambda x.\, M)\; N)^{\circledast} \\
 & \equiv & (\mathrm{let}\; !^{c} z = !^{c}(\lambda x.\, \mathrm{let}\; !^{i} y = x \;\mathrm{in}\; M^{\circledast}) \;\mathrm{in}\; z)\; (!^{w} N^{\circledast}) \\
\xrightarrow{\circledast} & \xrightarrow[(\beta !^{c})]{} & (\lambda x.\, \mathrm{let}\; !^{i} y = x \;\mathrm{in}\; M^{\circledast})\; (!^{w} N^{\circledast}) \\
 & \xrightarrow[(\beta \multimap)]{} & \mathrm{let}\; !^{i} x = !^{w} N^{\circledast} \;\mathrm{in}\; M^{\circledast} \\
 & \equiv & (\mathrm{let}\; x = N \;\mathrm{in}\; M)^{\circledast}
\end{array}
$$

$$
\begin{array}{ll}
 & \mathrm{let}\; x = V \;\mathrm{in}\; M \\
\xrightarrow[(V)]{} & M[x := V]
\end{array}
$$

$$
\begin{array}{lll}
 & & (\mathrm{let}\; x = V \;\mathrm{in}\; M)^{\circledast} \\
 & \equiv & \mathrm{let}\; !^{i} x = !^{w} !^{c} V^{\oplus} \;\mathrm{in}\; M^{\circledast} \\
\xrightarrow{\circledast} & \xrightarrow[(\beta !^{i})]{} & M^{\circledast}[x := V^{\oplus}] \\
 & \equiv & (M[x := V])^{\circledast}
\end{array}
$$

$$
\begin{array}{ll}
 & (\mathrm{let}\; x = L \;\mathrm{in}\; M)\; N \\
\xrightarrow[(C)]{} & \mathrm{let}\; x = L \;\mathrm{in}\; (M\; N)
\end{array}
$$

$$
\begin{array}{lll}
 & & ((\mathrm{let}\; x = L \;\mathrm{in}\; M)\; N)^{\circledast} \\
 & \equiv & (\mathrm{let}\; !^{c} z = (\mathrm{let}\; !^{i} x = !^{w} L^{\circledast} \;\mathrm{in}\; M^{\circledast}) \;\mathrm{in}\; z)\; (!^{w} N^{\circledast}) \\
\xrightarrow{\circledast} & \xrightarrow[(!^{c} !^{i})]{} & (\mathrm{let}\; !^{i} x = !^{w} L^{\circledast} \;\mathrm{in}\; \mathrm{let}\; !^{c} z = M^{\circledast} \;\mathrm{in}\; z)\; (!^{w} N^{\circledast}) \\
 & \xrightarrow[(!^{i} \multimap)]{} & \mathrm{let}\; !^{i} x = !^{w} L^{\circledast} \;\mathrm{in}\; (\mathrm{let}\; !^{c} z = M^{\circledast} \;\mathrm{in}\; z)\; (!^{w} N^{\circledast}) \\
 & \equiv & (\mathrm{let}\; x = L \;\mathrm{in}\; (M\; N))^{\circledast}
\end{array}
$$

$$
\begin{array}{ll}
 & \mathrm{let}\; x = (\mathrm{let}\; y = L \;\mathrm{in}\; M) \;\mathrm{in}\; N \\
\xrightarrow[(A)]{} & \mathrm{let}\; y = L \;\mathrm{in}\; \mathrm{let}\; x = M \;\mathrm{in}\; N
\end{array}
$$

$$
\begin{array}{lll}
 & & (\mathrm{let}\; x = (\mathrm{let}\; y = L \;\mathrm{in}\; M) \;\mathrm{in}\; N)^{\circledast} \\
\xrightarrow{\circledast} & \equiv & \mathrm{let}\; !^{i} x = !^{w}(\mathrm{let}\; !^{i} y = !^{w} L^{\circledast} \;\mathrm{in}\; M^{\circledast}) \;\mathrm{in}\; N^{\circledast} \\
 & \xrightarrow[(!^{i} !^{w} !^{w})]{} & \mathrm{let}\; !^{i} y = !^{w} L^{\circledast} \;\mathrm{in}\; \mathrm{let}\; !^{i} x = !^{w} M^{\circledast} \;\mathrm{in}\; N^{\circledast} \\
 & \equiv & (\mathrm{let}\; y = L \;\mathrm{in}\; \mathrm{let}\; x = M \;\mathrm{in}\; N)^{\circledast}
\end{array}
$$

$$
\begin{array}{ll}
 & \mathrm{let}\; x = M \;\mathrm{in}\; N \\
\xrightarrow[(G)]{} & N\;,\; x \notin \mathrm{fv}(N)
\end{array}
$$

$$
\begin{array}{lll}
 & & (\mathrm{let}\; x = M \;\mathrm{in}\; N)^{\circledast} \\
\xrightarrow{\circledast} & \equiv & \mathrm{let}\; !^{i} x = !^{w} M^{\circledast} \;\mathrm{in}\; N^{\circledast} \\
 & \xrightarrow[(\mathrm{W}!^{i})]{} & N^{\circledast}
\end{array}
$$

Figure 9.18: Soundness of $\circledast$ for general reduction.

$$
\begin{array}{ll}
 & (\lambda x.\, M)\; N \\
\xrightarrow[(I)]{} & \text{let } x = N \text{ in } M
\end{array}
$$

$$
\begin{array}{lll}
 & & ((\lambda x.\, M)\; N)^{\circledast} \\
 & \equiv & (\text{let } !^c z = !^c(\lambda x.\, \text{let } !^i y = x \text{ in } M^{\circledast}) \text{ in } z)\; (!^w N^{\circledast}) \\
\xrightarrow{\circledast} & \xrightarrow[(\beta !^c)]{} & (\lambda x.\, \text{let } !^i y = x \text{ in } M^{\circledast})\; (!^w N^{\circledast}) \\
 & \xrightarrow[(\beta \multimap)]{} & \text{let } !^i x = !^w N^{\circledast} \text{ in } M^{\circledast} \\
 & \equiv & (\text{let } x = N \text{ in } M)^{\circledast}
\end{array}
$$

$$
\begin{array}{ll}
 & \text{let } x = V \text{ in } E[x] \\
\xrightarrow[(V)]{} & E[x][x := V]
\end{array}
$$

$$
\begin{array}{lll}
 & & (\text{let } x = V \text{ in } E[x])^{\circledast} \\
 & \equiv & \text{let } !^i x = !^w !^c V^{\oplus} \text{ in } (E[x])^{\circledast} \\
\xrightarrow{\circledast} & \xrightarrow[(\beta !^i)]{} & E[x]^{\circledast}[x := V^{\oplus}] \\
 & \equiv & (E[x][x := V])^{\circledast}
\end{array}
$$

$$
\begin{array}{ll}
 & (\text{let } x = M \text{ in } A)\; N \\
\xrightarrow[(C)]{} & \text{let } x = M \text{ in } (A\; N)
\end{array}
$$

$$
\begin{array}{lll}
 & & ((\text{let } x = M \text{ in } A)\; N)^{\circledast} \\
 & \equiv & (\text{let } !^c z = (\text{let } !^i x = !^w M^{\circledast} \text{ in } A^{\circledast}) \text{ in } z)\; (!^w N^{\circledast}) \\
\xrightarrow{\circledast} & \xrightarrow[(!^c !^i)]{} & (\text{let } !^i x = !^w M^{\circledast} \text{ in let } !^c z = A^{\circledast} \text{ in } z)\; (!^w N^{\circledast}) \\
 & \xrightarrow[(!^i \multimap)]{} & \text{let } !^i x = !^w M^{\circledast} \text{ in } (\text{let } !^c z = A^{\circledast} \text{ in } z)\; (!^w N^{\circledast}) \\
 & \equiv & (\text{let } x = M \text{ in } (A\; N))^{\circledast}
\end{array}
$$

$$
\begin{array}{ll}
 & \text{let } x = (\text{let } y = M \text{ in } A) \text{ in } E[x] \\
\xrightarrow[(A)]{} & \text{let } y = M \text{ in let } x = A \text{ in } E[x]
\end{array}
$$

$$
\begin{array}{lll}
 & & (\text{let } x = (\text{let } y = M \text{ in } A) \text{ in } E[x])^{\circledast} \\
 & \equiv & \text{let } !^i x = !^w(\text{let } !^i y = !^w M^{\circledast} \text{ in } A^{\circledast}) \text{ in } (E[x])^{\circledast} \\
\xrightarrow{\circledast} & \xrightarrow[(!^i !^w !^w)]{} & \text{let } !^i y = !^w M^{\circledast} \text{ in let } !^i x = !^w A^{\circledast} \text{ in } (E[x])^{\circledast} \\
 & \equiv & (\text{let } y = M \text{ in let } x = A \text{ in } E[x])^{\circledast}
\end{array}
$$

Figure 9.19: Soundness of $\circledast$ for top-level standard reduction. Lemma 9.3.8 and Corollary 9.3.9 guarantee the applicability of the standard SLin rules.

Chapter 10

Conclusion

To ships swaying
rivers shift
oceans fall and
mountains drift
(Stipe)

WE have attempted to refine the classical lambda calculi with mechanisms to monitor their consumption of space resources. At the most general level, we have focused on the logical systems underlying the calculi's type systems: from the resource-oblivious minimal logic, via the minimal intuitionistic fragments of linear and affine logic, finally to a separated-linear logic encompassing all of the others. Our ultimate results in the separated-linear calculus provide a sufficiently rich basis to contrast the call-by-name, by-value and by-need parameter-passing mechanisms at the level of reduction sequences. The more general statement of the commuting conversion rules which we present here allowed a more useful statement on the finiteness of developments in the resource-conscious calculi. This result in turn allowed more concise proofs of the relevant theorems for less restricted systems than had been previously possible.

Perhaps the most glaring shortcoming in our present work is its treatment of call-by-value: where the separated translations of call-by-name and call-by-need are sound and complete for general reduction, the translations of call-by-value — both unitary and separated — are not complete, just as the unitary translation of call-by-need into affine lambda is not complete. The question of how one might complete these translations remains open.

The material in Part III raises the interesting question of the nature of the separated-linear *logic*, which is a separate issue from the associated lambda calculus which we addressed here. We have implied the existence of such a system by Curry-Howard correspondence to our system's typing rules, but we know of no presentation of the syntactic/proof-theoretic properties of the implied logic. To be sure, Jacobs has formulated a model of a separated system, but with only a conjectural sketch of the associated sequent calculus, and no consideration of its properties, or of intuitionistic or minimal fragments. We view the development of a separated-linear logic to be a worthwhile, if perhaps academic, point of future work.

We have not considered eta reduction. In the linear calculus, under the obvious eta rules

$$\begin{array}{lll} (\eta{\multimap}) & \lambda x.\, M\, x & \to\ M \\ (\eta!) & \text{let } !x = M \text{ in } !x & \to\ M \end{array}$$

by themselves, neither Girard translation is even sound. Soundness, and for the standard translation completeness, can be restored with an extra commuting conversion rule

$$({\multimap}!) \qquad \text{let } !x = M \text{ in } L\, N \quad\to\quad L\,(\text{let } !x = M \text{ in } N)\ ,$$

but then confluence in the extended Lin is lost. In a term $(\text{let } !x = L \text{ in } M_0\ M_1)\ N$ we have the top-level (!!) step

$$(\text{let } !x = L \text{ in } M_0\ M_1)\ N \to \text{let } !x = L \text{ in } (M_0\ M_1\ N)$$

as well as the $({\multimap}!)$ step

$$(\text{let } !x = L \text{ in } M_0\ M_1)\ N \to M_0\ (\text{let } !x = L \text{ in } M_1)\ N\ .$$

Unfortunately, it is not generally possible to rejoin the two terms, such as in the case where M_0 is a single free variable. Sabry and Wadler extend a subset of Lin with $(\eta!)$ and a somewhat artificial eta rule for abstractions [68],

$$\lambda x.\, \text{let } !y = x \text{ in } M\ !y \to M\ .$$

Their extension is designed to admit a reflection based on the steadfast translation from the computational lambda calculus λ_c [57, 58]. Since λ_c has no rule (C) as in Let, their subset of Lin excludes the $(!\multimap)$ rule; since they are interested only in reduction sequences relevant to the $*$-image of λ_c sequences, the above $(\eta\multimap)$ rule is sufficient. The question of finding a natural, confluent extension of Lin for eta reduction which is sound and perhaps complete for the $*$-image of reduction sequences remains open.

Obvious next steps in applying the separated-linear calculus are the development of some type inference system and operational model. For example, one might hope that corresponding to the finer grain of the resource usage classification in the separated-linear system, we would find finer analyses in the style of Mossin, Turner and Wadler [59, 74].

In the following sections we discuss a pair of more technical questions in some detail: the relationship of our work to the Barendregt cube of types, and the integration of sum and product types into our various frameworks.

10.1 Non-simple types

An interesting question for the call-by-need and various linear systems we have discussed here is their extension to the framework of pure type systems, in particular to the systems of the Barendregt cube [10, 11]. Bloo, Kamareddine and Nederpelt [17] have pointed out a problem with the naïve extension of call-by-need style reduction to the cube[1], namely a loss of subject reduction in certain corners. The problem arises from the fact that in pure type systems, *all* abstractions and applications are handled by the same mechanism. So if we move directly in this framework from the call-by-name (β) rule to the call-by-need rules, then we have the (I) rule creating let-bindings for type variables as well as for term variables. It is then the (C, A) rules which account for the loss of subject reduction in type systems which relate two different sorts. For example in $\lambda 2$ we have the (Church) typing [17, Example 4.10]

$$\begin{array}{rcl} B : *, z : B & \vdash & (\lambda A : *.\, \lambda y : A.\, (\lambda x : A.\, x)\, y)\, B\, z : B \\ & \underset{(I)}{=} & (\text{let } A : * = B \text{ in } \lambda y : A.\, (\lambda x : A.\, x)\, y)\, z : B \end{array}$$

but when we change the scope of A with the (C) rule the type of

$$(\lambda y : A.\, (\lambda x : A.\, x)\, y) \ ,$$

which is now applied directly to $z : B$, is $A \to A \to A$, which is no longer reduction-equivalent to a function accepting B:

$$\begin{array}{rl} & (\text{let } A : * = B \text{ in } \lambda y : A.\, (\lambda x : A.\, x)\, y)\, z \\ \xrightarrow[(C)]{} & \text{let } A : * = B \text{ in } ((\lambda y : A.\, (\lambda x : A.\, x)\, y)\, z) \end{array}$$

but

$$B : *, z : B \nvdash \text{let } A : * = B \text{ in } ((\lambda y : A.\, (\lambda x : A.\, x)\, y)\, z) : B$$

since

$$B : *, z : B, A : * \vdash \lambda y : A.\, (\lambda x : A.\, x)\, y : A \to A \to A \underset{\mathsf{Need2}}{\neq} B \to B \to B \ .$$

The various modal eliminators of the linear calculi would suffer the same problems with their commuting conversion rules. Bloo, Kamareddine and Nederpelt [17] solve this problem by introducing *definitions* as well as declarations in the typing contexts. Of course, working specifically in $\lambda 2$, we could simply distinguish applications of the two abstractions, treating type abstraction with the usual β rule, and term abstraction with let-bindings. While sufficient for $\lambda 2$, that technique would fail for the members of the cube where types depend on terms. Another solution at the level of abstraction of the cube which would be likely more practical would be the restriction of "harmful" commuting conversion steps to variables of certain sorts, perhaps by identifying as part of an extended PTS specification pairs of sorts which should or should not be mingled.

[1] Their work used the item notation and generalized reduction relation $\hookrightarrow$ which we discussed in Chapter 4; in this discussion we stay in the standard notation and use the call-by-need reduction relation.

Additional syntax

Types	A	$::= \cdots \mid A \times B \mid A + B$
Terms	L, M, N	$::= \cdots \mid (M, N) \mid \pi_i\, M \mid \mathsf{inl}\, M \mid \mathsf{inr}\, M$
		$\mid\ \text{case}\ L\ \text{in}\ \mathsf{inl}\, x : M,\ \mathsf{inr}\, y : N$
Evaluation Contexts	E	$::= \cdots \mid \pi_i\, E \mid \text{case}\ E\ \text{in}\ \mathsf{inl}\, x : M,\ \mathsf{inr}\, y : N$

Additional typing rules

$$\frac{\Gamma \vdash M : A \qquad \Delta \vdash N : B}{\Gamma, \Delta \vdash (M, N) : A \times B}\ \times\text{-I} \qquad \frac{\Gamma \vdash M : A_1 \times A_2}{\Gamma \vdash \pi_i\, M : A_i}\ \times\text{-E}$$

$$\frac{\Gamma \vdash M : A}{\Gamma \vdash \mathsf{inl}\, M : A + B}\ +\text{-I} \qquad \frac{\Gamma \vdash N : B}{\Gamma \vdash \mathsf{inr}\, N : A + B}\ +\text{-I}$$

$$\frac{\Gamma, x : A_1 \vdash M : B \qquad \Gamma, y : A_2 \vdash N : B \qquad \Delta \vdash L : A_1 + A_2}{\Gamma, \Delta \vdash (\text{case}\ L\ \text{in}\ \mathsf{inl}\, x : M,\ \mathsf{inr}\, y : N) : B}\ +\text{-E}$$

Additional reduction rules

$$\begin{array}{lllll}
(\beta\times) & \pi_i\,(M_1, M_2) & \to & M_i \\
(\beta+) & \text{case}\ (\mathsf{inl}\, L)\ \text{in}\ \mathsf{inl}\, x : M,\ \mathsf{inr}\, y : N & \to & M[x := L] \\
 & \text{case}\ (\mathsf{inr}\, L)\ \text{in}\ \mathsf{inl}\, x : M,\ \mathsf{inr}\, y : N & \to & N[y := L]
\end{array}$$

Additional evaluation rules Same as the reduction rules.

Figure 10.1: Products and sums in the call-by-name calculus.

Additional syntax As for extending call-by-name, except:

Terms	L, M, N	$::= \cdots \mid \pi_i\, M \mid \text{case}\ L\ \text{in}\ \mathsf{inl}\, x : M,\ \mathsf{inr}\, y : N$
Values	V, W	$::= \cdots \mid (V, W) \mid \mathsf{inl}\, V \mid \mathsf{inr}\, W$

Additional typing rules As for call-by-name, except

$$\frac{\Gamma \vdash V : A \qquad \Delta \vdash W : B}{\Gamma, \Delta \vdash (V, W) : A \times B}\ \times\text{-I}$$

Additional reduction rules

$$\begin{array}{lllll}
(\beta\times) & \pi_i\,(V_1, V_2) & \to & V_i \\
(\beta+) & \text{case}\ (\mathsf{inl}\, V)\ \text{in}\ \mathsf{inl}\, x : M,\ \mathsf{inr}\, y : N & \to & M[x := V] \\
 & \text{case}\ (\mathsf{inr}\, W)\ \text{in}\ \mathsf{inl}\, x : M,\ \mathsf{inr}\, y : N & \to & N[y := W]
\end{array}$$

Additional evaluation rules Same as the reduction rules.

Figure 10.2: Products and sums in the call-by-value and call-by-let calculi.

10.2 Product and sum types

The extension for products in each of the four systems is straightforward. For call-by-name, call-by-value and call-by-need the respective extensions $\mathsf{Name}^\times$, $\mathsf{Val}^\times$ and $\mathsf{Need}^\times$ appear in Figure 10.1, Figure 10.2 and Figure 10.3, and have all been previously studied. In the call-by-value and call-by-need systems, since pairs are copyable we require in the language that they contain only values. A term (M, N) is then just an abbreviation for $(\lambda x.\, \lambda y.\, (x, y))\ M\ N$. In call-by-name and value, we have only a single new rule for projection; in call-by-name we also have a new commuting conversion rule which again allows two type eliminator structures to move past each other.

10.2.1 The unitary calculus and translations

The extension $\mathsf{Lin}^\&$ of the linear lambda calculus for additive products in Figure 10.4 is just that of our previous work [51, 52]. The & operator is the additive or projective product of linear logic. In its typing introduction rule, we use the same linear environment Γ to type both components of the pair, so eventually only

Additional syntax As for extending call-by-value, plus:

$$\text{Answers} \quad A \ ::= \cdots \mid (V, W)$$

Additional typing rules As for call-by-value.

Additional reduction rules

$$\begin{array}{lll}
(\mathrm{I}\times) & \pi_i\,(V_1, V_2) & \to \ V_i \\
(\mathrm{A}\times) & \pi_i\,(\text{let } x = M \text{ in } N) & \to \ \text{let } x = M \text{ in } \pi_i\,N \\
(\mathrm{I}+) & \text{case } (\mathsf{inl}\,V) \text{ in } \mathsf{inl}\,x : M,\ \mathsf{inr}\,y : N & \to \ M[x := V] \\
 & \text{case } (\mathsf{inr}\,W) \text{ in } \mathsf{inl}\,x : M,\ \mathsf{inr}\,y : N & \to \ N[y := W] \\
(\mathrm{A}+) & \text{case } (\text{let } z = L_1 \text{ in } L_2) & \to \ \text{let } z = L_1 \text{ in case } L_2 \\
 & \quad \text{in } \mathsf{inl}\,x : M,\ \mathsf{inr}\,y : N & \qquad\qquad \text{in } \mathsf{inl}\,x : M,\ \mathsf{inr}\,y : N
\end{array}$$

Additional evaluation rules Same as the reduction rules, except:

$$\begin{array}{lll}
(\mathrm{A}\times) & \pi_i\,(\text{let } x = M \text{ in } A) & \to \ \text{let } x = M \text{ in } \pi_i\,A \\
(\mathrm{A}+) & \text{case } (\text{let } z = L \text{ in } A) & \mapsto \ \text{let } z = L \text{ in } (\text{case } A \text{ in } \mathsf{inl}\,x : M,\ \mathsf{inr}\,y : N) \\
 & \quad \text{in } \mathsf{inl}\,x : M,\ \mathsf{inr}\,y : N &
\end{array}$$

Figure 10.3: Products and sums in the call-by-need calculus.

one component of the pair can be used, since linear assumptions can only be used once. Additive products contrast with multiplicative or tensor products, written $\otimes$, where separate environments are used for typing the components, and where each component must be used:

$$\frac{\Gamma; \Phi \vdash M : A \qquad \Delta; \Psi \vdash N : B}{\Gamma, \Delta; \Phi, \Psi \vdash (M \otimes N) : A \otimes B} \otimes\text{-I}$$

$$\frac{\Gamma; \Phi \vdash M : A_1 \otimes A_2 \qquad \Delta; \Psi, x : A_1, y : A_2 \vdash N : B}{\Gamma, \Delta; \Phi, \Psi \vdash \text{let } x \otimes y = M \text{ in } N : B} \otimes\text{-E}$$

Recalling the equivalences in Girard's models of the logic, tensor and projective pairs are related by the equation

$$!(A \mathbin{\&} B) = (!A) \otimes (!B) \ , \tag{10.1}$$

where $=$ denotes equality in Girard's models [26]. We will use this relationship towards treating the steadfast translation.

For the standard translation we can extend Girard's original translation as he presents it for types to terms. For types he gives

$$(A \times B)^\circ \equiv A^\circ \mathbin{\&} B^\circ$$

which we extend to terms as shown in Figure 10.4. This extension is trivially sound and complete for mapping types and reduction sequences into $\mathsf{Lin}^{\&}$.

Proposition 10.2.1. Let $M, N \in \mathsf{Name}^\times$.

1. $(M[x := N])^\circ \equiv M^\circ[x := N^\circ]$.
2. $\Gamma \vdash_{\mathsf{Name}^\times} M : A$ if and only if $\Gamma^\circ; - \vdash_{\mathsf{Lin}^\&} M^\circ : A^\circ$.
3. $M \xrightarrow[\mathsf{Name}^\times]{} \!\!\!\!\rightarrow N$ if and only if $M^\circ \xrightarrow[\mathsf{Name}^\times]{} \!\!\!\!\rightarrow N^\circ$. ♦

The steadfast translation of pairs is trickier. Girard's formulation specifies

$$(A \times B)^* \equiv A^* \otimes B^*$$

but this formulation is not quite what we need. Our steadfast translation of terms relies crucially on the $*$-image of all values and only values being !-prefixed, so that we can emulate the call-by-value copying discipline through the mode. The translation of terms which this translation of types suggests is incompatible with that scheme, since pairs are considered values in $\mathsf{Val}^\times$. It is here that we make use of Equation 10.1. Since the components of pairs in call-by-value are always again

Additional syntax

$$\begin{array}{lrl} \text{Types} & A & ::= \cdots \mid A \,\&\, B \\ \text{Terms} & L, M, N & ::= \cdots \mid (M, N) \mid \pi_i\, M \\ \text{Evaluation Contexts} & E & ::= \cdots \mid \pi_i\, E \end{array}$$

Additional typing rules

$$\frac{\Phi; \Gamma \vdash M : A \qquad \Phi; \Gamma \vdash N : B}{\Phi; \Gamma \vdash (M, N) : A \,\&\, B}\;\&\text{-I} \qquad\qquad \frac{\Phi; \Gamma \vdash M : A_1 \,\&\, A_2}{\Phi; \Gamma \vdash \pi_i\, M : A_i}\;\&\text{-E}$$

Additional reduction rules

$$\begin{array}{lll} (\beta\&) & \pi_i\,(M_1, M_2) & \to\; M_i \\ (\&!) & \pi_i\,(\text{let } !x = M \text{ in } N) & \to\; \text{let } !x = M \text{ in } \pi_i\, N \end{array}$$

Additional evaluation rules

$$\begin{array}{lll} (\beta\&) & \pi_i\,(M_1, M_2) & \mapsto\; M_i \\ (\&!) & \pi_i\,(\text{let } !x = M \text{ in } A) & \mapsto\; \text{let } !x = M \text{ in } \pi_i\, A \end{array}$$

Standard translation

$$\begin{array}{rcl} (A \,\&\, B)^\circ & \equiv & A^\circ \,\&\, B^\circ \\ (M, N)^\circ & \equiv & (M^\circ, N^\circ) \\ (\pi_i\, M)^\circ & \equiv & \pi_i\, M^\circ \end{array}$$

Figure 10.4: Additive products in the linear lambda calculus, $\mathsf{Lin}^{\&}$.

values, and hence have !-prefixed translations, we can reason both on types and on terms as follows:

$$(A \times B)^* \equiv A^* \otimes B^* \equiv (!A^+) \otimes (!B^+) = !(A^+ \,\&\, B^+)\ .$$

This translation of types suggests a translation of terms which we used in our previous work [51, 52]

$$\begin{array}{rcl} (V, W)^+ & \equiv & (V^+, W^+) \\ (\pi_i\, M)^* & \equiv & \pi_i\,(\text{let } !z = M^* \text{ in } z)\ . \end{array}$$

Unfortunately this translation is again fundamentally flawed; the second clause preserves neither types nor reductions, and variations which reintroduce a ! prefix would seem to disrupt both the convention of only value-images having prefixes, and the translation of standard sequences. Taking

$$M \equiv (V_1, V_2) : A_1 \,\&\, A_2\ ,$$

then

$$M^* \equiv (V_1, V_2)^* \equiv !(V_1^+, V_2^+) : !(A_1^+, A_2^+)$$

and also $V_1^* \equiv !V_1^+ : !A_1^+$. But while

$$\pi_i\, M \equiv \pi_i\,(V_1, V_2) \xrightarrow[\mathsf{Val}^\times]{} V_i\ ,$$

from the translation of $\pi_i\, M$ we have

$$\begin{array}{rl} & (\pi_i\, M)^* \\ \equiv & \pi_i\,(\text{let } !z = !(V_1^+, V_2^+) \text{ in } z) \\ \xrightarrow[(\beta!)]{} & \pi_i\,(V_1^+, V_2^+) \\ \xrightarrow[(\beta\&)]{} & V_i \\ \not\equiv & V_i^*\ . \end{array}$$

We would argue that the problem is one of adapting the model-theoretic concepts to the syntax. Simply because the syntactic forms $!(V_1^+, V_2^+)$ and $(V_1^* \otimes V_2^*)$ have the same denotation in reasonable models does not, of course, make the two forms

Syntax As in Lin, with:

$$
\begin{array}{lrl}
\text{Types} & A, B & ::= \cdots \mid A \,\&\, B \mid A \otimes B \\
\text{Terms} & L, M, N & ::= \cdots \mid (V, W) \mid \pi_i\, M \mid M \otimes N \mid \text{let } x \otimes y = M \text{ in } N \\
\text{Evaluation contexts} & E & ::= \cdots \mid \pi_i\, E \mid \text{let } x \otimes y = E \text{ in } M \mid \text{let } x \otimes y = !E \text{ in } M \\
 & & \mid\ \text{let } !x = E \otimes N \text{ in } M \mid \text{let } !x = V \otimes E \text{ in } M
\end{array}
$$

Typing judgments As in Lin, plus the following:

For additive products

$$
\frac{\Phi;\ \Gamma \vdash M : A \qquad \Phi;\ \Gamma \vdash N : B}{\Phi;\ \Gamma \vdash (M, N) : A \,\&\, B}\ \&\text{-I}
\qquad\qquad
\frac{\Phi;\ \Gamma \vdash M : A_1 \,\&\, A_2}{\Phi;\ \Gamma \vdash \pi_i\, M : A_i}\ \&\text{-E}
$$

For multiplicative products

$$
\frac{\Phi;\ \Gamma \vdash M : A \qquad \Psi;\ \Delta \vdash N : B}{\Phi, \Psi;\ \Gamma, \Delta \vdash M \otimes N : A \otimes B}\ \otimes\text{-I}
$$

$$
\frac{\Phi;\ \Gamma \vdash M : A_1 \otimes A_2 \qquad \Psi;\ \Delta, x : A_1, y : A_2 \vdash N : B}{\Phi, \Psi;\ \Gamma, \Delta \vdash \text{let } x \otimes y = M \text{ in } N : B}\ \otimes\text{-E}
$$

For isomorphic types

$$
\frac{\Phi;\ \Gamma \vdash M : !(A \,\&\, B)}{\Phi;\ \Gamma \vdash M : (!A) \otimes (!B)}\ \otimes\cong\&
$$

Reduction rules As in Lin, plus the following:

$$
\begin{array}{lll}
(\beta\&) & \pi_i\, (M_1, M_2) & \to\ M_1 \\
(\beta\otimes) & \text{let } x \otimes y = M \otimes N \text{ in } L & \to\ L[x := M, y := N] \\
(!\otimes) & \text{let } !x = (\text{let } y \otimes z = L \text{ in } M) \text{ in } N & \\
 & & \to\ \text{let } y \otimes z = L \text{ in let } !x = M \text{ in } N \\
(\otimes!) & \text{let } y \otimes z = (\text{let } !x = L \text{ in } M) \text{ in } N & \\
 & & \to\ \text{let } !x = L \text{ in } (\text{let } y \otimes z = M \text{ in } N) \\
(\otimes\otimes) & \text{let } x \otimes y = (\text{let } z \otimes w = L \text{ in } M) \text{ in } N & \\
 & & \to\ \text{let } z \otimes w = L \text{ in } (\text{let } x \otimes y = M \text{ in } N) \\
(\&!) & \pi_i\, (\text{let } !x = M \text{ in } N) & \to\ \text{let } !x = M \text{ in } (\pi_i\, N) \\
(\&\otimes) & \pi_i\, (\text{let } x \otimes y = M \text{ in } N) & \to\ \text{let } x \otimes y = M \text{ in } (\pi_i\, N) \\
(\cong \otimes \&) & \text{let } !x = (!M) \otimes (!N) \text{ in } L & \to\ \text{let } !x = !(M \,\&\, N) \text{ in } L \\
(\cong \& \otimes) & \text{let } x \otimes y = !(M \,\&\, N) \text{ in } L & \to\ \text{let } x \otimes y = (!M) \otimes (!N) \text{ in } L
\end{array}
$$

Evaluation rules As in Lin, plus the following:

$$
\begin{array}{lll}
(\beta\&) & \pi_i\, (M_1, M_2) & \mapsto\ M_1 \\
(\beta\otimes) & \text{let } x \otimes y = M \otimes N \text{ in } L & \mapsto\ L[x := M, y := N] \\
(\cong \otimes \&) & \text{let } !x = (!M) \otimes (!N) \text{ in } L & \mapsto\ \text{let } !x = !(M \,\&\, N) \text{ in } L \\
(\cong \& \otimes) & \text{let } x \otimes y = !(M \,\&\, N) \text{ in } L & \mapsto\ \text{let } x \otimes y = (!M) \otimes (!N) \text{ in } L
\end{array}
$$

Figure 10.5: The linear lambda calculus with both products, $\mathsf{Lin}^{\&,\otimes}$.

easily interchangeable. We propose the syntax of Figure 10.5 for the $\mathsf{Lin}^{\&,\otimes}$ calculus which incorporates both of the two linear logic multiplicatives. The key rules for expressing the isomorphism in the operational syntax are the $(\cong \otimes \&, \cong \& \otimes)$ rules: given one of the terms forms associated with the isomorphism on types, we allow a conversion to the "other" term suggested by the types' relationship. The typing rule $\otimes\cong\&$ allows the two types to be converted freely to each other; the double line means that the inference is valid both downward and upward.

We need not change our previous standard translation into $\mathsf{Lin}^{\&}$; $\mathsf{Lin}^{\&,\otimes}$ is clearly a conservative extension of $\mathsf{Lin}^{\&}$. The formulation of $\mathsf{Lin}^{\&,\otimes}$ also admits a sound extension of the steadfast translation, spelled out in Figure 10.6. The difficult case of translating the projections relies on the isomorphism. The translation works by forcing the $!(M, N)$ representation of a pair to the $(!M) \otimes (!N)$ form. We must explicitly discard the unneeded component, which is immediately possible since both are already !-prefixed. Finally we can return the remaining

Standard translation As for $\mathsf{Lin}^{\&}$

Steadfast translation As from Val into Lin, plus:

Of types

$$(A \times B)^+ \quad \equiv \quad (A^+ \mathbin{\&} B^+)$$

Of terms

$$\begin{array}{rcl} (V, W)^+ & \equiv & (V^+, W^+) \\ (\pi_i\, M)^* & \equiv & \text{let } x_1 \otimes x_2 = M^* \text{ in let } !z = x_{3-i} \text{ in } x_i \end{array}$$

Figure 10.6: Extending the Girard translations for $\mathsf{Lin}^{\&,\otimes}$.

component, which now has the proper type. For the completeness result we will use the following grammar:

$$\begin{array}{rl} S, T & ::= \;!U \mid U\; S \mid (\text{let } !x = S \text{ in } x)\; T \mid \text{let } !x = S \text{ in } T \\ & \;\;\mid \text{ let } x_1 \otimes x_2 = S \text{ in let } !z = x_{3-i} \text{ in } x_i \\ & \;\;\mid \text{ let } x_1 \otimes x_2 = (!U_0) \otimes (!U_1) \text{ in let } !z = x_{3-i} \text{ in } x_i \\ U & ::= x \mid \lambda y.\, \text{let } !x = y \text{ in } S \mid (U_0, U_1) \end{array}$$

This grammar is clearly equivalent to the set of standard $\mathsf{Lin}^{\&,\otimes}$-reduction closed $*$-images of $\mathsf{Val}^{\times}$ terms.

Lemma 10.2.2. Let $M \in \mathsf{Val}^{\times}$ and $N \in \mathsf{Lin}^{\&,\otimes}$ such that $M^* \underset{\mathsf{Lin}^{\&,\otimes}}{\longmapsto\!\!\!\!\twoheadrightarrow} N$. Then N matches some term S of the grammar above.

Proof: As usual, by inspection of the grammar productions and reduction rules. ♦

We extend the inverse steadfast translation from Lin into Let of Section 7.2.1 to relate terms of this grammar to $\mathsf{Val}^{\times}$ terms as follows:

$$\begin{array}{rcl} (\text{let } x_1 \otimes x_2 = S \text{ in let } !z = x_{3-i} \text{ in } x_i)^\ddagger & \equiv & \pi_i\,(S^\ddagger) \\ (\text{let } x_1 \otimes x_2 = (!U_0) \otimes (!U_1) \text{ in let } !z = x_{3-i} \text{ in } x_i)^\ddagger & \equiv & \pi_i\,(U_0^\#, U_1^\#) \\ (U_1, U_2)^\# & \equiv & (U_1^\ddagger, U_2^\ddagger) \end{array}$$

As usual, $\ddagger$ can be exchanged with substitution of values, and is a right-inverse to $*$ on $\mathsf{Val}^{\times}$ terms:

Lemma 10.2.3. Let $S, U \in \mathsf{Lin}^{\&,\otimes}$ according to the grammar above. Then

$$(S[x := U])^\ddagger \equiv S^\ddagger[x := U^\#] \;\; .$$

Proof: Trivial. ♦

Lemma 10.2.4. Let $S, T \in \mathsf{Lin}^{\&,\otimes}$ according to the above grammar.

1. If $\Phi; \Gamma \vdash_{\mathsf{Lin}^{\&,\otimes}} S : A$ then $\Phi^\ddagger, \Gamma^\ddagger \vdash_{\mathsf{Val}^{\times}} S^\ddagger : A^\ddagger$.
2. If $S \underset{\mathsf{Lin}^{\&,\otimes}}{\longmapsto\!\!\!\!\twoheadrightarrow} T$ then $S^\ddagger \underset{\mathsf{Val}^{\times}}{\longmapsto\!\!\!\!\twoheadrightarrow} T^\ddagger$.

Proof: We consider only cases not covered by the analysis of Lemma 7.2.4. For the typing result the straightforward structural induction on typings of terms S is clear. Two new top-level standard steps are possible.

1. *Reduction by the* $(\beta\otimes)$ *rule of a term*

$$\text{let } x_1 \otimes x_2 = (!U_0) \otimes (!U_1) \text{ in let } !z = x_{3-i} \text{ in } x_i \;\; .$$

We have

$$\begin{array}{rl} & (\text{let } x_1 \otimes x_2 = (!U_0) \otimes (!U_1) \text{ in let } !z = x_{3-i} \text{ in } x_i)^\ddagger \\ \equiv & \pi_i\,(U_0^\#, U_1^\#) \\ \underset{(\beta\times)}{\longmapsto} & U_i^\# \\ \equiv & (!U_i)^\ddagger \;\; . \end{array}$$

2. *Reduction by rule* $(\cong \mathbin{\&} \otimes)$ *of a term*

$$\text{let } x_1 \otimes x_2 = !(U_0, U_1) \text{ in let } !z = x_{3-i} \text{ in } x_i \ \ .$$

We have simply:

$$\begin{array}{rl} & (\text{let } x_1 \otimes x_2 = !(U_0, U_1) \text{ in let } !z = x_{3-i} \text{ in } x_i)^{\ddagger} \\ \equiv & \pi_i\,(U_0^{\#}, U_1^{\#}) \\ \equiv & (\text{let } x_1 \otimes x_2 = (!U_0) \otimes (!U_1) \text{ in let } !z = x_{3-i} \text{ in } x_i)^{\ddagger} \ \ . \end{array}$$

Compatible closure in evaluation contexts follows by structural induction on S to find the standard step; multistep standard reduction follows by the trivial induction on the length of the standard step. ♦

Proposition 10.2.5, Steadfast translation of $\mathsf{Val}^{\times}$ into $\mathsf{Lin}^{\&,\otimes}$.
Let $M, N, V \in \mathsf{Val}^{\times}$. Then:

1. $M[x := V]^* \equiv M^*[x := V^+]$.
2. $\Gamma \vdash_{\mathsf{Val}^{\times}} M : A$ if and only if $\Gamma^*; - \vdash_{\mathsf{Lin}^{\&,\otimes}} M^* : A^*$.
3. If $M \xrightarrow[\mathsf{Val}^{\times}]{} \!\!\!\!\!\rightarrow N$ then $M \xrightarrow[\mathsf{Lin}^{\&,\otimes}]{} \!\!\!\!\!\rightarrow N$.
4. $M \underset{\mathsf{Val}^{\times}}{\longmapsto\!\!\!\!\rightarrow} N$ if and only if $M \underset{\mathsf{Lin}^{\&,\otimes}}{\longmapsto\!\!\!\!\rightarrow} N$.

Proof: The substitution result is a straightforward induction. For the other three cases, we only treat cases not covered in Proposition 7.2.5 and Proposition 10.2.1. For the new cases of the typing result we consider each of the new term forms. For a typing of a term

$$\dfrac{\begin{array}{c}\vdots\,\mathcal{F} \\ \Gamma \vdash_{\mathsf{Val}^{\times}} V : A\end{array} \quad \begin{array}{c}\vdots\,\mathcal{G} \\ \Delta \vdash_{\mathsf{Val}^{\times}} W : B\end{array}}{\Gamma, \Delta \vdash_{\mathsf{Val}^{\times}} (V, W) : A \times B}\ \times\text{-I} \ \ ,$$

we can construct the $\mathsf{Lin}^{\&,\otimes}$ typing tree

$$\dfrac{\dfrac{\dfrac{\begin{array}{c}\vdots\,\mathcal{F}^+ \\ \Gamma^*; - \vdash_{\mathsf{Lin}^{\&,\otimes}} V^+ : A^+\end{array}}{\vdots}}{\Gamma^*\Delta^*; - \vdash_{\mathsf{Lin}^{\&,\otimes}} V^+ : A^+} \quad \dfrac{\dfrac{\begin{array}{c}\vdots\,\mathcal{G}^+ \\ \Gamma^*; - \vdash_{\mathsf{Lin}^{\&,\otimes}} W^+ : B^+\end{array}}{\vdots}}{\Gamma^*\Delta^*; - \vdash_{\mathsf{Lin}^{\&,\otimes}} W^+ : B^+}\ \text{Weakening}}{\Gamma^*, \Delta^*; - \vdash_{\mathsf{Lin}^{\&,\otimes}} (V^+, W^+) : A^+ \mathbin{\&} B^+}\ \&\text{-I} \ \ .$$

For terms $\pi_i\, M$ we have

$$\dfrac{\begin{array}{c}\vdots\,\mathcal{F} \\ \Gamma \vdash_{\mathsf{Val}^{\times}} M : A_1 \mathbin{\&} A_2\end{array}}{\Gamma \vdash_{\mathsf{Val}^{\times}} \pi_i\, M : A_i}\ \times\text{-E} \ \ ,$$

we first build partial trees $\mathcal{G}_0, \mathcal{G}$

$$\mathcal{G}_0 \equiv \dfrac{}{-; x_{3-i} : !A_{3-i}{}^+ \vdash_{\mathsf{Lin}^{\&,\otimes}} x_{3-i} : !A_{3-i}{}^+}\ \text{Id}$$

$$\mathcal{G} \equiv \dfrac{\vdots\,\mathcal{G}_0 \qquad \dfrac{\dfrac{}{-; x_i : !A_i{}^+ \vdash_{\mathsf{Lin}^{\&,\otimes}} x_i : !A_i{}^+}\ \text{Id}}{z : A_{3-i}; x_i : !A_i{}^+ \vdash_{\mathsf{Lin}^{\&,\otimes}} x_i : !A_i{}^+}\ \text{Weakening}}{-; x_1 : !A_1{}^+, x_2 : !A_2{}^+ \vdash_{\mathsf{Lin}^{\&,\otimes}} \text{let } !z = x_{3-i} \text{ in } x_i : !A_i{}^+}\ !\text{-E}$$

which we can use to build

$$\dfrac{\dfrac{\dfrac{\dfrac{\begin{array}{c}\vdots\,\mathcal{F}^* \\ \Gamma^*; - \vdash_{\mathsf{Lin}^{\&,\otimes}} M^* : (A_1 \times A_2)^*\end{array}}{\Gamma^*; - \vdash_{\mathsf{Lin}^{\&,\otimes}} M^* : !(A_1^+ \mathbin{\&} A_2^+)}\ \equiv}{\Gamma^*; - \vdash_{\mathsf{Lin}^{\&,\otimes}} M^* : (!A_1^+) \otimes (!A_2^+)}\ \cong \qquad \vdots\,\mathcal{G}}{\Gamma^*; - \vdash_{\mathsf{Lin}^{\&,\otimes}} \text{let } x_1 \otimes x_2 = M^* \text{ in let } !z = x_{3-i} \text{ in } x_i : !A_i{}^+}\ \otimes\text{-E}}{\Gamma^*; - \vdash_{\mathsf{Lin}^{\&,\otimes}} (\pi_i\, M)^* : A_i^*}\ \equiv \ \ .$$

Additional syntax

$$\begin{array}{lrcl} \text{Types} & A & ::= & \cdots \mid A \mathbin{\&} B \\ \text{Terms} & L, M, N & ::= & \cdots \mid (M, N) \mid \pi_i\, M \end{array}$$

Additional typing judgments

$$\frac{\mathcal{E} \vdash M : A \qquad \mathcal{E} \vdash N : B}{\mathcal{E} \vdash (M, N) : A \mathbin{\&} B}\ \&\text{-I} \qquad\qquad \frac{\mathcal{E} \vdash M : A_1 \mathbin{\&} A_2}{\mathcal{E} \vdash \pi_i\, M : A_i}\ \&\text{-E}$$

Additional reduction rules

$$\begin{array}{llcl} (\beta\&) & \pi_i\,(M_1, M_2) & \to & M_i \\ (\&!^i) & \pi_i\,(\text{let } !^i x = M \text{ in } N) & \to & \text{let } !^i x = M \text{ in } \pi_i\, N \\ (\&!^c) & \pi_i\,(\text{let } !^c x = M \text{ in } N) & \to & \text{let } !^c x = M \text{ in } \pi_i\, N \\ (\&!^w) & \pi_i\,(\text{let } !^w x = M \text{ in } N) & \to & \text{let } !^w x = M \text{ in } \pi_i\, N \end{array}$$

Additional evaluation rule

$$\begin{array}{llcl} (\beta\&) & \pi_i\,(M_1, M_2) & \mapsto & M_i \\ (\&!^i) & \pi_i\,(\text{let } !^i x = !^w M \text{ in } A) & \mapsto & \text{let } !^i x = !^w M \text{ in } \pi_i\, A \end{array}$$

Figure 10.7: Products in the separated-linear lambda calculus.

Completeness follows from the first clause of Lemma 10.2.4.

For soundness of general reduction, we translate a $(\beta\times)$ contraction

$$\pi_i\,(U_1, U_2) \to U_i$$

as

$$\begin{array}{rl} & (\pi_i\,(U_1, U_2))^* \\ \equiv & \text{let } x_1 \otimes x_2 = !(U_1^+, U_2^+) \text{ in let } !z = x_{3-i} \text{ in } x_i \\ \xrightarrow[(\simeq\&\otimes)]{} & \text{let } x_1 \otimes x_2 = (!U_1^+) \otimes (!U_2^+) \text{ in let } !z = x_{3-i} \text{ in } x_i \\ \xrightarrow[(\beta\otimes)]{} & \text{let } !z = !U_{3-i}^+ \text{ in } !U_i^+ \\ \xrightarrow[(\beta!)]{} & (U_i)^* \ , \end{array}$$

and compatible closure is trivial. For standard reduction we have exactly the same case as above for soundness, and completeness by Lemma 10.2.4. ♦

10.2.2 The separated-linear calculus and hybrid translations

In extending the hybrid translation, we have the possibility of using the style of pairs from either the standard or steadfast translation. We consider a simple extension in the former style first. We again include in the language of types the additive product constructor $\&$ as presented by Girard, as in Figure 10.7. The extensions to the translations for product types and terms are straightforward as well, and appear in Figure 10.8. In previous work we studied the extension to products of the unitary calculus, and the results for the separated Girard translations follow as they do there.

We again have soundness and completeness for this extension to $\circledast$. We begin towards the proof as usual with the inverse translation, shown in Figure 10.9.

Lemma 10.2.6. Let $M \in \mathsf{Need}$ extended for products. Then $M^{\circledast\natural} \equiv M$.

Proof: Clear by an analysis of the two new cases. ♦

Lemma 10.2.7. Let A be a type, and S, T be terms in SLin extended for products as restricted in the grammars of Figure 10.9.

1. Let $M \in \mathsf{Need}$; then $(M^{\circledast})^\natural \equiv M$.
2. If $\mathcal{E} \vdash_{\mathsf{SLin}} S : A$ then $\mathcal{E}^\natural \vdash_{\mathsf{Need}} S^\natural : A^\natural$.
3. If $S \xrightarrow[\mathsf{SLin}]{}\!\!\!\!\!\rightarrow T$ then $S^{\circledast} \xrightarrow[\mathsf{Need}]{}\!\!\!\!\!\rightarrow T^{\circledast}$.

Standard translation

$$
\begin{array}{rcl}
(A \times B)^\circ & \equiv & A^\circ \mathbin{\&} B^\circ \\
(M, N)^\circ & \equiv & (M^\circ, N^\circ) \\
(\pi_i\, M)^\circ & \equiv & \pi_i\, M^\circ
\end{array}
$$

Steadfast translation

$$
\begin{array}{rcl}
(A \times B)^+ & \equiv & A^* \mathbin{\&} B^* \\
(V, W)^+ & \equiv & (V^+, W^+) \\
(\pi_i\, M)^* & \equiv & \pi_i\,(\text{let } !^i z = M^* \text{ in } z)
\end{array}
$$

Hybrid translation

$$
\begin{array}{rcl}
(A \times B)^\oplus & \equiv & A^\circledast \mathbin{\&} B^\circledast \\
(V, W)^\oplus & \equiv & (V^\circledast, W^\circledast) \\
(\pi_i\, M)^\circledast & \equiv & \pi_i\,(\text{let } !^c z = M^\circledast \text{ in } z)
\end{array}
$$

Figure 10.8: Translations of products into SLin.

Syntax As before, plus:

$$
\begin{array}{rcl}
\bar{A} & ::= & \cdots \mid \bar{A}_0 \mathbin{\&} \bar{A}_1 \\
S, T & ::= & \cdots \mid \pi_i\, K \mid \pi_i\, x \\
U & ::= & \cdots \mid (!^c U, !^c U) \\
K & ::= & U \mid \text{let } !^c x = S \text{ in } x \\
 & \mid & \text{let } !^i x = !^w S \text{ in } K
\end{array}
$$

Translation As before, plus:

$$
\begin{array}{rcl}
(A_1 \mathbin{\&} A_2)^{\bowtie} & \equiv & A_1^\natural \times A_2^\natural \\
(\pi_i\, K)^\natural & \equiv & \pi_i\, K^\natural \\
(\text{let } !^c x = S \text{ in } \pi_i\, x)^\natural & \equiv & \pi_i\, S^\natural \\
(!^c U_0, !^c U_1)^{\bowtie} & \equiv & (U_0^{\bowtie}, U_1^{\bowtie}) \\
(\pi_i\, [\,])^\natural & \equiv & \pi_i\, [\,] \\
(\text{let } !^i x = !^w S \text{ in } G)^\natural & \equiv & \text{let } x = S^\natural \text{ in } G^\natural \\
(\text{let } !^i x = !^w S \text{ in } K)^{\bowtie} & \equiv & \text{let } x = S^\natural \text{ in } K^{\bowtie}
\end{array}
$$

Figure 10.9: The extension for products of inverse hybrid translation and structures of Figure 9.6.

Of terms S

$$\dfrac{\begin{array}{c}\vdots\ \mathcal{P}\\ \mathcal{E}\vdash_{\mathsf{SLin}} K : A_1 \,\&\, A_2\end{array}}{\mathcal{E}\vdash_{\mathsf{SLin}} \pi_i\,K : A_i}\,\&\text{-E} \quad \overset{\natural}{\to} \quad \dfrac{\dfrac{\dfrac{\begin{array}{c}\vdots\ \mathcal{P}^{\bowtie}\\ \mathcal{E}^\natural \vdash_{\mathsf{Need}} K^{\bowtie} : (A_1 \,\&\, A_2)^{\bowtie}\end{array}}{\mathcal{E}^\natural\vdash_{\mathsf{Need}} K^{\bowtie} : A_1^\natural\times A_2^\natural}\equiv}{\mathcal{E}^\natural\vdash_{\mathsf{Need}}\pi_i\,K^{\bowtie}:A_i^\natural}\times\text{-E}}{\mathcal{E}^\natural\vdash_{\mathsf{Need}}(\pi_i\,K)^\natural : A_i^\natural}\equiv$$

$$\dfrac{\begin{array}{c}\vdots\ \mathcal{P}_1\\ \mathcal{E}\vdash_{\mathsf{SLin}} S : !^c A\end{array}\qquad \begin{array}{c}\vdots\ \mathcal{P}_2\\ \mathcal{F}, {\blacktriangleright_c} x : A\vdash_{\mathsf{SLin}} \pi_i\,x : B\end{array}}{\mathcal{E},\mathcal{F}\vdash_{\mathsf{SLin}} \mathrm{let}\ !^c x = S\ \mathrm{in}\ \pi_i\,x : B}\,!^c\text{-E}$$

$$\overset{\natural}{\to}\quad \dfrac{\dfrac{\dfrac{\begin{array}{c}\vdots\ \mathcal{P}_1^\natural\\ \mathcal{E}^\natural\vdash_{\mathsf{Need}} S^\natural : (!^c A)^\natural\end{array}}{\mathcal{E}^\natural\vdash_{\mathsf{Need}} S^\natural : A^{\bowtie}}\equiv \qquad \dfrac{\begin{array}{c}\vdots\ \mathcal{P}_2^\natural\\ (\mathcal{F}, {\blacktriangleright_c} x : A)^\natural\vdash_{\mathsf{Need}} (\pi_i\,x)^\natural : B^\natural\end{array}}{\mathcal{F}^\natural, x : A^{\bowtie}\vdash_{\mathsf{Need}} \pi_i\,x : B^\natural}\equiv}{\mathcal{E}^\natural,\mathcal{F}^\natural\vdash_{\mathsf{Need}} \mathrm{let}\ x = S^\natural\ \mathrm{in}\ \pi_i\,x : B^\natural}\,\text{Let}}{(\mathcal{E},\mathcal{F})^\natural\vdash_{\mathsf{Need}} (\mathrm{let}\ !^c x = S\ \mathrm{in}\ \pi_i\,x)^\natural : B^\natural}\equiv$$

Figure 10.10: Soundness of $\natural$ for typings.

Proof: Clause (1) is clear from the definitions of the two transformations. For the remaining clauses, we extend the proof of Lemma 9.3.7 with the reasoning on types shown in Figures 10.10 and 10.11, and on reductions in Figures 10.12, and 10.13. ♦

Proposition 10.2.8. Let A be a type, M, N terms and V a value in SLin extended for products.

1. $(M[x := V])^{\circledast} \equiv M^{\circledast}[x := V^{\oplus}]$.
2. $\Gamma \vdash_{\mathsf{Need}} M : A$ if and only if ${\blacktriangleright_\iota}\Gamma^{\circledast} \vdash_{\mathsf{Need}} M^{\circledast} : A^{\circledast}$.
3. $M \underset{\mathsf{Need}}{\twoheadrightarrow} N$ if and only if $M^{\circledast} \underset{\mathsf{SLin}}{\twoheadrightarrow} N^{\circledast}$.

Proof: Clause 1 follows as before. For Clause 2 soundness is straightforward, if somewhat complicated, and is shown in Figure 10.14. Completeness follows from Clauses (1) and (2) of Lemma 10.2.7. For Clause 3 soundness is again straightforward, if somewhat complicated, and is shown in Figure 10.15. Completeness follows from Clauses (1) and (3) of Lemma 10.2.7. ♦

The version of $\&$ is not the same as that presented by Jacobs, who suggests a rule

$$\dfrac{{\blacktriangleright_\iota}\Gamma \vdash M : A \quad {\blacktriangleright_\iota}\Gamma \vdash N : B}{{\blacktriangleright_\iota}\Gamma \vdash (M, N) : A \,\&\, B}$$

[36, p. 103]. Such a rule requires other *permeability* rules [26, 27] which in turn require polarities to be attached to our formulas. Our formulation of $\&$-I is analogous to Girard's, and we believe that this rule is less complicated and more appropriate for our setting.

A steadfast-style hybrid translation. Steadfast translation-style pairs for the hybrid translation are more difficult. One could imagine using the various connectives and correspondences suggested by Jacobs [37], in particular the *relevance tensor* product $\otimes_c$. However the appropriate behavior of these connectives at the term level is not completely clear, and we do not explore this topic further.

10.2.3 Sum types and expressions

The treatment of sums in the standard translation is straightforward [51, 52]. On types we have

$$(A + B)^{\circ} \quad \equiv \quad (!A^{\circ}) \oplus (!B^{\circ})$$

Of terms U

$$
\dfrac{
 \dfrac{\begin{array}{c}\vdots\,\mathcal{P}_1\\ \blacktriangleright^i\Gamma,\blacktriangleright^c\Delta\vdash_{\mathsf{SLin}} U_0:\bar{A}_0\end{array}}{\blacktriangleright^i\Gamma,\blacktriangleright^c\Delta\vdash_{\mathsf{SLin}} !^cU_0 :!^c\bar{A}_0}\;!^c\text{-I}
 \quad
 \dfrac{\begin{array}{c}\vdots\,\mathcal{P}_2\\ \blacktriangleright^i\Gamma,\blacktriangleright^c\Delta\vdash_{\mathsf{SLin}} U_1:\bar{A}_1\end{array}}{\blacktriangleright^i\Gamma,\blacktriangleright^c\Delta\vdash_{\mathsf{SLin}} !^cU_1 :!^c\bar{A}_1}\;!^c\text{-I}
}{\blacktriangleright^i\Gamma,\blacktriangleright^c\Delta\vdash_{\mathsf{SLin}} (!^cU_0,!^cU_1) :!^c\bar{A}_0\&!^c\bar{A}_1}\;\&\text{-I}
$$

$$
\xrightarrow{\bowtie}\quad
\begin{array}{c}
\dfrac{
 \dfrac{\dfrac{\begin{array}{c}\vdots\,\mathcal{P}_1^{\bowtie}\\ (\blacktriangleright^i\Gamma,\blacktriangleright^c\Delta)^\natural \vdash_{\mathsf{Need}} U_0^{\bowtie}:\bar{A}_0^{\bowtie}\end{array}}{\Gamma^\natural,\Delta^\natural\vdash_{\mathsf{Need}} U_0^{\bowtie}:\bar{A}_0^{\bowtie}}\;\equiv}{\check{\Gamma}^\natural,\check{\Delta}^\natural\vdash_{\mathsf{Need}}\check{U}_0^{\bowtie}:\bar{A}_0^{\bowtie}}\;\equiv
 \quad
 \dfrac{\dfrac{\begin{array}{c}\vdots\,\mathcal{P}_2^{\bowtie}\\ (\blacktriangleright^i\Gamma,\blacktriangleright^c\Delta)^\natural \vdash_{\mathsf{Need}} U_1^{\bowtie}:\bar{A}_1^{\bowtie}\end{array}}{\Gamma^\natural,\Delta^\natural\vdash_{\mathsf{Need}} U_1^{\bowtie}:\bar{A}_1^{\bowtie}}\;\equiv}{\tilde{\Gamma}^\natural,\tilde{\Delta}^\natural\vdash_{\mathsf{Need}}\tilde{U}_1^{\bowtie}:\bar{A}_1^{\bowtie}}\;\equiv
}{\check\Gamma^\natural,\check\Delta^\natural,\tilde\Gamma^\natural,\tilde\Delta^\natural\vdash_{\mathsf{Need}}(\check U_0^{\bowtie},\tilde U_1^{\bowtie}):\bar A_0^{\bowtie}\times\bar A_1^{\bowtie}}\;\times\text{-I}\\
\vdots\,\text{Contraction}\\
\dfrac{\Gamma^\natural,\Delta^\natural\vdash_{\mathsf{Need}} (U_0^{\bowtie},U_1^{\bowtie}):\bar A_0^{\bowtie}\times\bar A_1^{\bowtie}}{(\blacktriangleright^i\Gamma,\blacktriangleright^c\Delta)^\natural\vdash_{\mathsf{Need}}(!^cU_0,!^cU_1)^{\bowtie}:(!^c\bar A_0\&!^c\bar A_1)^{\bowtie}}\;\times\text{-I}
\end{array}
$$

Of terms K

$$
\dfrac{\begin{array}{c}\vdots\,\mathcal{P}\\ \mathcal{E}\vdash_{\mathsf{SLin}} S:!^c\bar A\\ \vdots\,\mathcal{P}\\ \mathcal{E}\vdash_{\mathsf{SLin}} S:!^c\bar A\end{array} \quad \dfrac{\dfrac{}{\blacktriangleright^i x:\bar A\vdash_{\mathsf{SLin}} x:\bar A}\;\text{Id}}{\blacktriangleright^c x:\bar A\vdash_{\mathsf{SLin}} x:\bar A}\;\text{Dereliction}!^c}{\mathcal E\vdash_{\mathsf{SLin}} \text{let } !^cx = S \text{ in } x : \bar A}\;!^c\text{-E}
$$

$$
\xrightarrow{\bowtie}\quad
\dfrac{\dfrac{\begin{array}{c}\vdots\,\mathcal P^\natural\\ \mathcal E^\natural\vdash_{\mathsf{Need}} S^\natural:!^c\bar A^\natural\end{array}}{\mathcal E^\natural\vdash_{\mathsf{Need}} S^\natural:\bar A^{\bowtie}}\;\equiv}{\mathcal E^\natural\vdash_{\mathsf{Need}}(\text{let }!^cx=S\text{ in }x)^{\bowtie}:\bar A^{\bowtie}}\;\equiv
$$

$$
\dfrac{\dfrac{\begin{array}{c}\vdots\,\mathcal P_1\\ \blacktriangleright^i\Gamma,\blacktriangleright^w\Delta\vdash_{\mathsf{SLin}} S:!^c\bar A\end{array}}{\blacktriangleright^i\Gamma,\blacktriangleright^w\Delta\vdash_{\mathsf{SLin}} !^wS:!^w!^c\bar A}\;!^c\text{-I} \quad \begin{array}{c}\vdots\,\mathcal P_2\\ \mathcal E,\blacktriangleright^i x:\bar A\vdash_{\mathsf{SLin}} K:\bar B\end{array}}{\mathcal E,\blacktriangleright^i\Gamma,\blacktriangleright^w\Delta\vdash_{\mathsf{SLin}} \text{let } !^ix = !^wS \text{ in } K}\;!^i\text{-E}
$$

$$
\xrightarrow{\bowtie}\quad
\dfrac{\dfrac{\dfrac{\begin{array}{c}\vdots\,\mathcal P_1^\natural\\ (\blacktriangleright^i\Gamma,\blacktriangleright^w\Delta)^\natural\vdash_{\mathsf{Need}} S^\natural:!^c\bar A^\natural\end{array}}{\Gamma^\natural,\Delta^\natural\vdash_{\mathsf{Need}} S^\natural:\bar A^{\bowtie}}\;\equiv \quad \dfrac{\begin{array}{c}\vdots\,\mathcal P_2\\ (\mathcal E,\blacktriangleright^ix:\bar A)^\natural\vdash_{\mathsf{Need}} K^{\bowtie}:\bar B^{\bowtie}\end{array}}{\mathcal E^\natural,x:\bar A^{\bowtie}\vdash_{\mathsf{Need}} K^{\bowtie}:\bar B^{\bowtie}}\;\equiv}{\mathcal E^\natural,\Gamma^\natural,\Delta^\natural\vdash_{\mathsf{Need}} \text{let } x = S^\natural \text{ in } K^{\bowtie}:\bar B^{\bowtie}}\;\text{Let}}{(\mathcal E,\Gamma,\Delta)^\natural\vdash_{\mathsf{Need}}(\text{let } !^ix=!^wS \text{ in } K)^{\bowtie}:\bar B^{\bowtie}}\;\equiv
$$

Figure 10.11: Soundness of $\bowtie$ for typings.

Of terms S

$$
\begin{array}{rl}
 & \pi_i\,(!^c U_0, !^c U_1) \\
\xrightarrow[(\beta\&)]{} & !^c U_i
\end{array}
\quad \overset{\natural}{\to} \quad
\begin{array}{rl}
 & (\pi_i\,(!^c U_0, !^c U_1))^\natural \\
\equiv & \pi_i\,(U_0^{\bowtie}, U_1^{\bowtie}) \\
\xrightarrow[(I\times)]{} & U_i^{\bowtie} \\
\equiv & (!^c U_i)^\natural
\end{array}
$$

$$
\begin{array}{rl}
 & \pi_i\,(\text{let } !^c x = S \text{ in } x) \\
\xrightarrow[(\&!^c)]{} & \text{let } !^c x = S \text{ in } \pi_i\, x
\end{array}
\quad \overset{\natural}{\to} \quad
\begin{array}{rl}
 & (\pi_i\,(\text{let } !^c x = S \text{ in } x))^\natural \\
\equiv & \pi_i\,(S^\natural) \\
\equiv & (\text{let } !^c x = S \text{ in } \pi_i\, x)^\natural
\end{array}
$$

$$
\begin{array}{rl}
 & \pi_i\,(\text{let } !^i x = !^w S \text{ in } K) \\
\xrightarrow[(\&!^i)]{} & \text{let } !^i x = !^w S \text{ in } \pi_i\, K
\end{array}
\quad \overset{\natural}{\to} \quad
\begin{array}{rl}
 & (\pi_i\,(\text{let } !^i x = !^w S \text{ in } K))^\natural \\
\equiv & \pi_i\,(\text{let } x = S^\natural \text{ in } K^\natural) \\
\xrightarrow[(A\times)]{} & \text{let } x = S^\natural \text{ in } \pi_i\, K^\natural \\
\equiv & (\text{let } !^i x = !^w S \text{ in } \pi_i\, K)^\natural
\end{array}
$$

$$
\begin{array}{rl}
 & \text{let } !^c x = (\text{let } !^i y = !^w S \text{ in } T) \text{ in } \pi_i\, x \\
\xrightarrow[(!^c!^i)]{} & \text{let } !^i y = !^w S \text{ in let } !^c x = T \text{ in } \pi_i\, x
\end{array}
$$

$$
\overset{\natural}{\to} \quad
\begin{array}{rl}
 & (\text{let } !^c x = (\text{let } !^i y = !^w S \text{ in } T) \text{ in } \pi_i\, x)^\natural \\
\equiv & \pi_i\,(\text{let } y = S^\natural \text{ in } T^\natural) \\
\xrightarrow[(A\times)]{} & \text{let } y = S^\natural \text{ in let } !^c x = T^\natural \text{ in } \pi_i\, x \\
\equiv & (\text{let } !^i y = !^w S \text{ in let } !^c x = T \text{ in } \pi_i\, x)^\natural
\end{array}
$$

$$
\begin{array}{rl}
 & \text{let } !^c x = (\text{let } !^c y = S \text{ in } D[y]\; !^w T) \text{ in } \pi_i\, x \\
\xrightarrow[(!^c!^c)]{} & \text{let } !^c y = S \text{ in let } !^c x = D[y]\; !^w T \text{ in } \pi_i\, x
\end{array}
$$

$$
\overset{\natural}{\to} \quad
\begin{array}{rl}
 & (\text{let } !^c x = (\text{let } !^c y = S \text{ in } D[y]\; !^w T) \text{ in } \pi_i\, x)^\natural \\
\equiv & \pi_i\,(D^{\bowtie}[S^\natural]\, T^\natural) \\
\equiv & (\text{let } !^c y = S \text{ in let } !^c x = D[y]\; !^w T \text{ in } \pi_i\, x)^\natural
\end{array}
$$

$$
\begin{array}{rl}
 & \text{let } !^c x = (\text{let } !^c y = !^w S \text{ in } \pi_j\, y) \text{ in } \pi_i\, x \\
\xrightarrow[(!^c!^c)]{} & \text{let } !^c y = !^w S \text{ in let } !^c x = \pi_j\, y \text{ in } \pi_i\, x
\end{array}
$$

$$
\overset{\natural}{\to} \quad
\begin{array}{rl}
 & (\text{let } !^c x = (\text{let } !^c y = !^w S \text{ in } \pi_j\, y) \text{ in } \pi_i\, x)^\natural \\
\equiv & \pi_i\, \pi_j\, S^\natural \\
\equiv & (\text{let } !^c y = !^w S \text{ in let } !^c x = \pi_j\, y \text{ in } \pi_i\, x)^\natural
\end{array}
$$

$$
\begin{array}{rl}
 & \text{let } !^c x = (\text{let } !^c y = !^w S \text{ in } \pi_i\, y) \text{ in } D[x]\; !^w T \\
\xrightarrow[(!^c!^c)]{} & \text{let } !^c y = !^w S \text{ in let } !^c x = \pi_i\, y \text{ in } D[x]\; !^w T
\end{array}
$$

$$
\overset{\natural}{\to} \quad
\begin{array}{rl}
 & (\text{let } !^c x = (\text{let } !^c y = !^w S \text{ in } \pi_i\, y) \text{ in } D[x]\; !^w T)^\natural \\
\equiv & D^{\bowtie}[\pi_i\, S^\natural]\, T^\natural \\
\equiv & (\text{let } !^c y = !^w S \text{ in let } !^c x = \pi_i\, y \text{ in } D[x]\; !^w T)^\natural
\end{array}
$$

Figure 10.12: Soundness of $\natural$ for reduction.

Of terms K

$$
\begin{array}{rl}
 & \text{let } !^c x = (\text{let } !^i y = !^w S \text{ in } T) \text{ in } x \\
\xrightarrow[(!^c !^i)]{} & \text{let } !^i y = !^w S \text{ in let } !^c x = T \text{ in } x
\end{array}
$$

$$
\begin{array}{rrl}
 & & (\text{let } !^c x = (\text{let } !^i y = !^w S \text{ in } T) \text{ in } x)^\natural \\
\stackrel{\bowtie}{\to} & \equiv & \text{let } y = S^\natural \text{ in } T^\natural \\
 & \equiv & (\text{let } !^i y = !^w S \text{ in let } !^c x = T \text{ in } x)^\natural
\end{array}
$$

$$
\begin{array}{rl}
 & \text{let } !^c x = (\text{let } !^c y = S \text{ in } D[y] \; !^w T) \text{ in } x \\
\xrightarrow[(!^c !^c)]{} & \text{let } !^c y = S \text{ in let } !^c x = D[y] \; !^w T \text{ in } x
\end{array}
$$

$$
\begin{array}{rrl}
 & & (\text{let } !^c x = (\text{let } !^c y = S \text{ in } D[y] \; !^w T) \text{ in } x)^\natural \\
\stackrel{\bowtie}{\to} & \equiv & D^\bowtie[S^\natural] \; !^w T^\natural \\
 & \equiv & (\text{let } !^c y = S \text{ in let } !^c x = D[y] \; !^w T \text{ in } x)^\natural
\end{array}
$$

$$
\begin{array}{rl}
 & \text{let } !^c x = (\text{let } !^c y = S \text{ in } y) \text{ in } x \\
\xrightarrow[(!^c !^c)]{} & \text{let } !^c y = S \text{ in let } !^c x = y \text{ in } x
\end{array}
$$

$$
\begin{array}{rrl}
 & & (\text{let } !^c x = (\text{let } !^c y = S \text{ in } y) \text{ in } x)^\natural \\
\stackrel{\bowtie}{\to} & \equiv & S^\natural \\
 & \equiv & (\text{let } !^c y = S \text{ in let } !^c x = y \text{ in } x)^\natural
\end{array}
$$

$$
\begin{array}{rl}
 & \text{let } !^i x = !^w(\text{let } !^i y = !^w S \text{ in } T) \text{ in } K \\
\xrightarrow[(!^i !^w !^i_w)]{} & \text{let } !^i y = !^w S \text{ in let } !^i x = !^w T \text{ in } K
\end{array}
$$

$$
\begin{array}{rrl}
 & & (\text{let } !^i x = !^w(\text{let } !^i y = !^w S \text{ in } T) \text{ in } K)^\natural \\
 & \equiv & \text{let } x = (\text{let } y = S^\natural \text{ in } T^\natural) \text{ in } K^\bowtie \\
\stackrel{\bowtie}{\to} & \xrightarrow[(A)]{} & \text{let } y = S^\natural \text{ in let } x = T^\natural \text{ in } K^\bowtie \\
 & \equiv & (\text{let } !^i y = !^w S \text{ in let } !^i x = !^w T \text{ in } K)^\natural
\end{array}
$$

Figure 10.13: Soundness of $\bowtie$ for reduction.

$$\dfrac{\begin{array}{c}\vdots\,\mathcal{P}_1\\ \Gamma \vdash_{\mathsf{Need}} V : A\end{array}\quad \begin{array}{c}\vdots\,\mathcal{P}_2\\ \Delta \vdash_{\mathsf{Need}} W : B\end{array}}{\Gamma,\Delta \vdash_{\mathsf{Need}} (V,W) : A\times B}\times\text{-I}$$

$$\xrightarrow{\circledast}\quad \dfrac{\dfrac{\dfrac{\begin{array}{c}\vdots\,\mathcal{P}_1^{\circledast}\\ \blacktriangleright^i \Gamma^{\circledast} \vdash_{\mathsf{SLin}} V^{\circledast} : A^{\circledast}\\ \vdots\,\text{Weakening}\\ \blacktriangleright^i \Gamma^{\circledast},\Delta^{\circledast} \vdash_{\mathsf{SLin}} V^{\circledast} : A^{\circledast}\end{array}\quad \begin{array}{c}\vdots\,\mathcal{P}_2^{\circledast}\\ \blacktriangleright^i \Delta^{\circledast} \vdash_{\mathsf{SLin}} W^{\circledast} : B^{\circledast}\\ \vdots\,\text{Weakening}\\ \blacktriangleright^i \Gamma^{\circledast},\Delta^{\circledast} \vdash_{\mathsf{SLin}} W^{\circledast} : B^{\circledast}\end{array}}{\blacktriangleright^i \Gamma^{\circledast},\Delta^{\circledast} \vdash_{\mathsf{SLin}} (V^{\circledast},W^{\circledast}) : A^{\circledast} \,\&\, B^{\circledast}}\&\text{-I}}{\blacktriangleright^i \Gamma^{\circledast},\Delta^{\circledast} \vdash_{\mathsf{SLin}} !^c(V^{\circledast},W^{\circledast}) :!^c(A^{\circledast} \,\&\, B^{\circledast})}\,!^c\text{-I}}{\blacktriangleright^i (\Gamma,\Delta)^{\circledast} \vdash_{\mathsf{SLin}} (V,W)^{\circledast} : (A\times B)^{\circledast}}\equiv$$

$$\dfrac{\begin{array}{c}\vdots\,\mathcal{P}\\ \Gamma \vdash_{\mathsf{Need}} M : A_1\times A_2\end{array}}{\Gamma,\Delta \vdash_{\mathsf{Need}} \pi_i\, M : A_i}\times\text{-E}$$

$$\xrightarrow{\circledast}\quad \dfrac{\dfrac{\dfrac{\begin{array}{c}\vdots\,\mathcal{P}^{\circledast}\\ \blacktriangleright^i \Gamma^{\circledast} \vdash_{\mathsf{SLin}} M^{\circledast} : (A_1\times A_2)^{\circledast}\end{array}}{\blacktriangleright^i \Gamma^{\circledast} \vdash_{\mathsf{SLin}} M^{\circledast} :!^c(A_1^{\circledast} \,\&\, A_2^{\circledast})}\equiv \quad \dfrac{\dfrac{}{\blacktriangleright^l x : A_1^{\circledast} \,\&\, A_2^{\circledast} \vdash_{\mathsf{SLin}} x : A_1^{\circledast} \,\&\, A_2^{\circledast}}\text{Id}}{\blacktriangleright^c x : A_1^{\circledast} \,\&\, A_2^{\circledast} \vdash_{\mathsf{SLin}} x : A_1^{\circledast} \,\&\, A_2^{\circledast}}\text{Dereliction}!^c}{\blacktriangleright^i \Gamma^{\circledast} \vdash_{\mathsf{SLin}} \text{let } !^c z = M^{\circledast} \text{ in } z : A_1^{\circledast} \,\&\, A_2^{\circledast}}\,!^c\text{-E}}{\dfrac{\blacktriangleright^i \Gamma^{\circledast} \vdash_{\mathsf{SLin}} \pi_i\,(\text{let } !^c z = M^{\circledast} \text{ in } z) : A_i^{\circledast}}{\blacktriangleright^i \Gamma^{\circledast} \vdash_{\mathsf{SLin}} (\pi_i\, M)^{\circledast} : A_i^{\circledast}}\equiv}\&\text{-E}$$

Figure 10.14: Soundness of $\circledast$ for typing judgments in the extension of SLin to products.

$$\begin{array}{llcll}
 & & & & (\pi_i\,(V_1,V_2))^{\circledast}\\
 & \pi_i\,(V_1,V_2) & & \equiv & \pi_i\,(\text{let } !^c z = !^c(V_1^{\circledast},V_2^{\circledast}) \text{ in } z)\\
\xrightarrow[(I\times)]{} & V_i & \xrightarrow{\circledast} & \xrightarrow[(\beta !^c)]{} & \pi_i\,(V_1^{\circledast},V_2^{\circledast})\\
 & & & \xrightarrow[(\beta\&)]{} & V_i^{\circledast}
\end{array}$$

$$\begin{array}{llcll}
 & & & & (\pi_i\,(\text{let } x = M \text{ in } N))^{\circledast}\\
 & & & \equiv & \pi_i\,(\text{let } !^c z = \text{let } !^i x = !^w M^{\circledast} \text{ in } N^{\circledast} \text{ in } z)\\
 & \pi_i\,(\text{let } x = M \text{ in } N) & & \xrightarrow[(!^c !^i)]{} & \pi_i\,(\text{let } !^i x = !^w M^{\circledast} \text{ in let } !^c z = N^{\circledast} \text{ in } z)\\
\xrightarrow[(A\times)]{} & \text{let } x = M \text{ in } \pi_i\, N & \xrightarrow{\circledast} & \xrightarrow[(\& !^i)]{} & \text{let } !^i x = !^w M^{\circledast} \text{ in } \pi_i\,(\text{let } !^c z = N^{\circledast} \text{ in } z)\\
 & & & \equiv & (\text{let } x = M \text{ in } \pi_i\, N)^{\circledast}
\end{array}$$

Figure 10.15: Soundness of $\circledast$ for reduction in the extension of SLin to products.

using the tensor sum operator [26], and the straightforward extension to terms. The extension of the steadfast translation is less clear. We again need to preserve a translation of values which is !-prefixed, and again Girard's translation does not:

$$(A+B)^* \equiv A^* \oplus B^* \ \ .$$

Unfortunately, the relation of $\oplus$ to another connective by means of an exponential connective is not as helpful as the one linking $\otimes$ and & via !: instead, we have

$$?(A \oplus B) = (?A)⅋(?B) \ \ .$$

Unfortunately, neither the ? mode nor the ⅋ connective appear in the fragment of linear logic we wish to consider: ? is defined from ! by negation, and ⅋ makes sense only if we allow multiple conclusions on the right-hand side of $\vdash$. In the separated calculus, we can extend the hybrid translation in the standard style, but the difficulties of the unitary case would seem to extend to the separated system as well.

10.2.4 General syntactic pairs and sums

It is somewhat irksome to restrict the syntactic form of pairs and copairs in Val and Need because we lose the correspondence with call-by-name terms. Just as the unitary steadfast translation from Need into Aff is unsatisfying because it introduces a separate target language, such a treatment of the two datatypes is unsatisfying due to its separate source syntax. In this section we propose an alternate formulation of products and sums in call-by-value and call-by-need, although we leave open the problems of verifying their various properties and of adapting the translations into linear systems from the amended calculi.

For a call-by-value system the restriction is straightforward. We allow terms to be pairs of any two terms, but values to be pairs only of other values, and likewise for copairs.

$$\begin{array}{llcl} \text{Terms} & L, M, N & ::= & \cdots \mid (M, N) \mid \pi_i\, M \mid \mathsf{inl}\, M \mid \mathsf{inr}\, N \\ & & & \mid\ \text{case}\ L\ \text{in}\ \mathsf{inl}\, x : M,\ \mathsf{inr}\, y : N \\ \text{Values} & V, W & ::= & \cdots \mid (V, W) \mid \mathsf{inl}\, V \mid \mathsf{inr}\, W \end{array}$$

To ensure that standard evaluation does not get stuck, we extend the notion of evaluation context.

$$\text{Evaluation contexts} \quad E \ ::= \cdots \mid (E, M) \mid (V, E) \mid \mathsf{inl}\, E \mid \mathsf{inr}\, E$$

The choice to evaluate the left component of pairs first is arbitrary; it would be equally appropriate to pursue the opposite order.

The steadfast translation of pairs seems adaptable to this formulation. Rather than taking (M, N) to a projective pair, we map it to a tensor product:

$$(M, N)^* \equiv M^* \otimes N^* \ \ .$$

The requirement that a pair's components be values for the pair itself to be a value is preserved by the translation as far as copying of function arguments is concerned. To force a pair-image into a !-prefixed projective pair, its components must first be reduced to a !-prefixed terms as well. We would need to vary the translation of projection to similarly force a pair's components:

$$(\pi_i)^* \equiv \text{let } !x = M^* \text{ in let } y_1 \otimes y_2 = !x \text{ in let } !z = y_{3-i} \text{ in } y_i \ \ .$$

Unfortunately the $\mathsf{Lin}^{\&,\otimes}$ reduction rules we have described is not sufficient; if a pair's component in another pair, the ($\cong \otimes\&$) rule is not applicable, and so a term such as

$$(\pi_i\,((V_1, V_2), V_3))^* \equiv \text{let } !x = ((!V_1^+ \otimes !V_2^+) \otimes !V_3^+) \text{ in } \cdots$$

would be stuck. Several sets of further ($\cong$) rules for $\mathsf{Lin}^{\&,\otimes}$ suggest themselves, although it is not completely clear exactly which such extensions would be best to admit elegant standard reduction and other results.

The modification to call-by-need is more interesting. Unlike in call-by-value, we do not want to force evaluation of a pair or copair's components before they are needed, but we still do not want to copy the components until they are actually reduced to values. One solution is to abolish the general (V) rule, and forward separate rules for different types of values:

$$\begin{array}{lllll}
(Vx) & \text{let } x = y \text{ in } L & \to & L[x := y] \\
(V\to) & \text{let } x = \lambda y.\, M \text{ in } L & \to & L[x := \lambda y.\, M] \\
(V\times) & \text{let } x = (M, N) \text{ in } L & \to & \text{let } y = M \\
 & & & \quad \text{in let } z = N \text{ in } L[x := (y, z)] \\
(V+) & \text{let } x = \mathsf{inl}\, M \text{ in } L & \to & \text{let } y = M \text{ in } L[x := \mathsf{inl}\, y] \\
 & \text{let } x = \mathsf{inr}\, N \text{ in } L & \to & \text{let } z = N \text{ in } L[x := \mathsf{inr}\, z]
\end{array}$$

For sum types, we also have a new (I) rule:

$$\begin{array}{llll}
(I+) & \text{case } \mathsf{inl}\, L \text{ in } \mathsf{inl}\, x : M,\ \mathsf{inr}\, y : N & \to & \text{let } x = L \text{ in } M \\
 & \text{case } \mathsf{inr}\, L \text{ in } \mathsf{inl}\, x : M,\ \mathsf{inr}\, y : N & \to & \text{let } y = L \text{ in } N
\end{array}$$

Some technical problems with this result are obvious: we no longer have a substitution result

$$\text{If } M \to N \text{ then } M[x := V] \to N[x := V].$$

since different values are treated differently, suggesting that we might discard the (Vx) rule as in standard evaluation. It is also not at all clear how one might modify the steadfast translation of pairs into some extension of Aff using only the usual connectives and reduction rules. These difficulties notwithstanding, this alternative formulation of product and sum expressions seems to satisfyingly alleviate the syntactic restrictions of the previous formulations.

Appendix A
Recursion

In this appendix we consider an extension of the call-by-name, call-by-value and linear calculi for recursion. The material here is a revision of the results presented in work with Odersky, Turner and Wadler [52]. Braüner has also presented a recursive linear lambda calculus similar to the first of the two we present [18]. His work mainly addresses the properties of an extension for recursion of the standard translation from Name to Lin in models for these systems rather than in their reduction theories. His formulation is essentially the same as our $\mathsf{Lin}_{\mathrm{rec}}$ calculus, although since we first of all separate linear and intuitionistic assumptions in typing judgments, and moreover avoid term structures corresponding to structural rules, while he does neither, the syntaxes of our respective systems appear quite different at first glance. Braüner comments that the standard translation from call-by-name into his system is sound for reduction, but addresses neither completeness nor call-by-value systems and the steadfast translation. Rather, Braüner is concerned mainly with the categorical models of recursion in the two systems $\mathsf{Name}_{\mathrm{rec}}$ and $\mathsf{Lin}_{\mathrm{rec}}$. Preliminary versions of the results in this section have appeared in an unpublished workshop proceedings [48].

We consider a recursive linear lambda calculus first in Section A.1, and show the system to satisfy subject reduction and confluence. In Section A.2 we tackle the usual recursive call-by-name system, and its translation by $\circ$ into $\mathsf{Lin}_{\mathrm{rec}}$, and in Section A.3 recursive call-by-value and the steadfast translation. Finally, in Section A.4 we consider an alternate recursive linear lambda calculus whose existence we had earlier conjectured, along with its translations from the call-by-name and call-by-value systems, and discuss why this calculus is not especially interesting.

A.1 Recursive Linear Lambda

We can extend the linear lambda calculus for recursion on a single bound variable in the standard manner. We extend Lin as shown in Figure A.1 to obtain the recursive linear lambda calculus $\mathsf{Lin}_{\mathrm{rec}}$. The extension satisfies the same basic properties as the non-recursive system.

Lemma A.1.1. Let $M, N \in \mathsf{Lin}_{\mathrm{rec}}$.

(i) Let $\Phi;\ \Gamma, x : A \vdash_{\mathsf{Lin}_{\mathrm{rec}}} M : B$ and $\Psi;\ \Delta \vdash_{\mathsf{Lin}_{\mathrm{rec}}} N : A$.
Then $\Phi, \Psi;\ \Gamma, \Delta \vdash_{\mathsf{Lin}_{\mathrm{rec}}} M[x := N] : B$.

(ii) Let $\Phi, x : A;\ \Gamma \vdash_{\mathsf{Lin}_{\mathrm{rec}}} M : B$ and $\Psi;\ - \vdash_{\mathsf{Lin}_{\mathrm{rec}}} N : A$.
Then $\Phi, \Psi;\ \Gamma \vdash_{\mathsf{Lin}_{\mathrm{rec}}} M[x := N] : B$.

Proof: By the usual structural inductions. ♦

Lemma A.1.2. Let $L, M, N \in \mathsf{Lin}_{\mathrm{rec}}$, and let $M \to N$. Then:

1. $M[x := L] \to N[x := L]$.
2. $L[x := M] \twoheadrightarrow L[x := N]$.

Proposition A.1.3. $\mathsf{Lin}_{\mathrm{rec}}$ satisfies subject reduction.

Proof: For the new reduction rule we have by Lemma A.1.1.(ii). For the remaining rules the analysis is just as in the result for Lin. ♦

Proposition A.1.4. $\mathsf{Lin}_{\mathrm{rec}}$ is confluent.

Proof: We extend the marking technique of general reduction $\mathsf{Lin}^{\star}$ of Section 5.3.2. We have a new step $\underline{\mu 1}$ and two new reduction rules

$$\begin{array}{lccc} (\mathrm{Rec}_0) & {}^{[\underline{\mu 1}]}\mu x.\,\dot{M} & \to & \dot{M}[x := \mu x.\,|\dot{M}|] \\ (\mathrm{Rec}_1) & \mu x.\,\dot{M} & \to & \dot{M}[x := \mu x.\,\dot{M}] \end{array}$$

Syntactic domains As for Lin, plus the following:

$$\text{Terms} \qquad L, M, N \quad ::= \quad \ldots \mid \mu x.\, M$$

Typing judgments As for Lin, plus the following:

$$\frac{\Phi,\, x : A;\; - \vdash M : A}{\Phi;\; - \vdash \mu x.\, M : A}\,\text{Rec}$$

Reduction relation As for Lin, plus the following:

$$\text{(Rec)} \qquad \mu x.\, M \quad \to \quad M[x := \mu x.\, M]$$

Figure A.1: The linear lambda calculus with recursion $\mathsf{Lin}_{\mathrm{rec}}$.

The deduction of FD!($\mathsf{Lin}^{\star}_{\mathrm{rec}}$) has only one difficulty, namely, the adaptation of the norm used to show strong normalization of marked reduction. We extend the norm $||\,.\,||$ as follows:

$$\begin{aligned} ||^{[\underline{\mu 1}]}\mu x.\, \dot{M}|| \quad &= \quad 1 + ||\dot{M}[x := \mu x.\, \dot{M}]|| \\ ||\mu x.\, \dot{M}|| \quad &= \quad ||\dot{M}|| \ , \end{aligned}$$

and keep the notion of decreasing weighting as before. It is trivial to observe that the norm is decreasing on marked reduction, and that marked reduction preserves decreasing weightings. We can construct a decreasing weighting for any term as in the previous system by solving a recurrence relation, and strong normalization of $\mathsf{Lin}^{\star}_{\mathrm{rec}0}{}'$ follows. For weak confluence of $\mathsf{Lin}^{\star}_{\mathrm{rec}0}{}'$ we need only Lemma 5.3.14 and Lemma A.1.2, and so full confluence follows with normalization. We then have strong finiteness of developments, and confluence follows as usual. ♦

A.2 Recursive Call-by-Name

Proposition A.2.1. $\mathsf{Name}_{\mathrm{rec}}$ has the subject reduction property. ♦

Proposition A.2.2. $\mathsf{Name}_{\mathrm{rec}}$ is confluent. ♦

Proposition A.2.3. The translation $\circ$ from $\mathsf{Name}_{\mathrm{rec}}$ to $\mathsf{Lin}_{\mathrm{rec}}$ preserves substitutions, preserves typing judgments and preserves reduction sequences: For $M, N \in \mathsf{Name}_{\mathrm{rec}}$,

(i) $(M[x := N])^{\circ} \equiv M^{\circ}[x := N^{\circ}]$.

(ii) $\Gamma \vdash_{\mathsf{Name}_{\mathrm{rec}}} M : A$ if and only if $\Gamma^{\circ};\ - \vdash_{\mathsf{Lin}_{\mathrm{rec}}} M^{\circ} : A^{\circ}$.

(iii) $M \xrightarrow[\mathsf{Name}_{\mathrm{rec}}]{} \!\!\!\!\!\rightarrow N$ if and only if $M^{\circ} \xrightarrow[\mathsf{Name}_{\mathrm{rec}}]{} \!\!\!\!\!\rightarrow N^{\circ}$.

Proof: Clauses (i) and (ii) are as before. For soundness in Clause (iii) we take

$$\mu x.\, M \xrightarrow[\mathsf{Name}_{\mathrm{rec}}]{} M[x := \mu x.\, M] \ ,$$

and in $\mathsf{Lin}_{\mathrm{rec}}$ we have

$$\begin{aligned} & (\mu x.\, M)^{\circ} \\ \equiv \quad & \mu x.\, M^{\circ} \\ \xrightarrow{\text{(Rec)}} \quad & M^{\circ}[x := \mu x.\, M^{\circ}] \\ \equiv \quad & (M[x := \mu x.\, M)^{\circ} \ . \end{aligned}$$

Completeness is also easy; we extend the grammar of recursion-closed $\circ$-images by

$$S ::= \cdots \mid \mu x.\, S \ ,$$

and the map † by

$$(\mu x.\, S)^{\dagger} \equiv \mu x.\, S^{\dagger} \ .$$

Clearly † is again right-inverse to $\circ$, and preserves the (Rec) step. ♦

Syntactic domains As for $\mathsf{Lin}_{\mathrm{rec}}$.
Typing judgments As for Lin, plus the following:

$$\frac{\Phi,\, x : A;\ - \vdash M :!A}{\Phi;\ - \vdash \mu x.\, M :!A}\ \mathrm{Rec}_v$$

Reduction relation As for Lin, plus the following:

$$(\mathrm{Rec}_v) \qquad \mu x.\, M \quad \to \quad M[x := \mathrm{let}\ !y = \mu x.\, M\ \mathrm{in}\ y]$$

Figure A.2: The alternate linear lambda calculus with recursion $\mathsf{Lin}_{\mathrm{rec}\,v}$.

A.3 Recursive Call-by-Value

Proposition A.3.1. $\mathsf{Val}_{\mathrm{rec}}$ has the subject reduction property. ♦

Proposition A.3.2. $\mathsf{Val}_{\mathrm{rec}}$ is confluent. ♦

Proposition A.3.3. The translation $*$ from $\mathsf{Val}_{\mathrm{rec}}$ to $\mathsf{Lin}_{\mathrm{rec}}$ preserves substitution of values, preserves typing judgments and preserves reduction sequences: For $M, N, V \in \mathsf{Val}_{\mathrm{rec}}$,

(i) $(M[x := V])^* \equiv M^*[x := V^+]$.

(ii) $\Gamma \vdash_{\mathsf{Val}_{\mathrm{rec}}} M : A$ if and only if $\Gamma^*;\ - \vdash_{\mathsf{Val}_{\mathrm{rec}}} M^* : A^*$.

(iii) If $M \xrightarrow[\mathsf{Val}_{\mathrm{rec}}]{} \!\!\!\!\!\rightarrow N$ then $M^* \xrightarrow[\mathsf{Lin}_{\mathrm{rec}}]{} \!\!\!\!\!\rightarrow N^*$.

Proof: Clauses (i) and (ii) are as usual. For soundness in Clause (iii) we take $\mu f.\, \lambda x.\, M \xrightarrow[\mathsf{Val}_{\mathrm{rec}}]{} \!\!\!\!\!\rightarrow \lambda x.\, M[f := \mu f.\, \lambda x.\, M]$, and then in $\mathsf{Lin}_{\mathrm{rec}}$ we have

$$\begin{array}{ll} & !\mu f.\, \lambda z.\, \mathrm{let}\ !x = z\ \mathrm{in}\ M^* \\ \xrightarrow[(\mathrm{Rec})]{} & !(\lambda z.\, \mathrm{let}\ !x = z\ \mathrm{in}\ M^*)[f := \mu f.\, \lambda z.\, \mathrm{let}\ !x = z\ \mathrm{in}\ M^*] \\ \equiv & (\lambda x.\, M[f := \mu f.\, \lambda x.\, M])^*\ . \end{array}$$

We have the same counterexamples to completeness as before. ♦

A.4 An Alternate Recursive Linear Lambda

In earlier work with Odersky, Turner and Wadler [51] we had conjectured that two different recursion operators were possible in the linear lambda calculus: the μ operator we use in $\mathsf{Lin}_{\mathrm{rec}}$, and in addition a second operator μ_v of type

$$\frac{\Phi,\, x : A;\ - \vdash M : !A}{\Phi;\ - \vdash \mu_v x.\, M : !A}\ \mathrm{Rec}_v\ ,$$

or equivalently

$$\frac{\Phi;\ x :!A \vdash M : !A}{\Phi;\ - \vdash \mu_v x.\, M : !A}\ \mathrm{Rec}_v\ .$$

The motivation for two systems is that one rule — the one we use here in $\mathsf{Lin}_{\mathrm{rec}}$ — seems to correspond to the standard translation of the call-by-name Rec rule, while the Rec_v rule above seems to correspond to the steadfast translation of the call-by-value Rec rule. In some sense, the two seem to be equivalent, but at the same time, some problems exist with the alternate operator. The Rec_v typing rule gives rise to the $\mathsf{Lin}_{\mathrm{rec}\,v}$ calculus of Figure A.2. We can map terms from one recursive linear calculus into the other with the maps $\flat$ and $\sharp$ of Figure A.3.

We find that the two maps are inverses of each other modulo convertibility. Starting from $\mathsf{Lin}_{\mathrm{rec}}$ the result is simple:

Lemma A.4.1. Let $M \in \mathsf{Lin}_{\mathrm{rec}}$. Then $M^{\flat\sharp} \xrightarrow[\mathsf{Lin}_{\mathrm{rec}}]{} \!\!\!\!\!\rightarrow M$.

From $\mathsf{Lin}_{\mathrm{rec}}$ to $\mathsf{Lin}_{\mathrm{rec}\,v}$

$$\begin{array}{rcl} x^\flat & \equiv & x \\ (\lambda x.\, M)^\flat & \equiv & \lambda x.\, M^\flat \\ (M\ N)^\flat & \equiv & M^\flat\ N^\flat \\ (\mathrm{let}\ x = M\ \mathrm{in}\ N)^\flat & \equiv & \mathrm{let}\ x = M^\flat\ \mathrm{in}\ N^\flat \\ (\mu x.\, M)^\flat & \equiv & \mathrm{let}\ !y = \mu_v x.\, !M^\flat\ \mathrm{in}\ y \end{array}$$

From $\mathsf{Lin}_{\mathrm{rec}\,v}$ to $\mathsf{Lin}_{\mathrm{rec}}$

$$\begin{array}{rcl} x^\sharp & \equiv & x \\ (\lambda x.\, M)^\sharp & \equiv & \lambda x.\, M^\sharp \\ (M\ N)^\sharp & \equiv & M^\sharp\ N^\sharp \\ (\mathrm{let}\ x = M\ \mathrm{in}\ N)^\sharp & \equiv & \mathrm{let}\ x = M^\sharp\ \mathrm{in}\ N^\sharp \\ (\mu_v x.\, M)^\sharp & \equiv & !\mu x.\, \mathrm{let}\ !z = M^\sharp\ \mathrm{in}\ z \end{array}$$

Figure A.3: Maps between the alternate recursive linear lambda calculi.

Proof: All cases are trivial aside from the recursion binding:

$$\begin{array}{rl} & (\mu x.\, M)^{\flat\sharp} \\ \equiv & \mathrm{let}\ !y = !\mu x.\, (\mathrm{let}\ !z = !M^{\flat\sharp}\ \mathrm{in}\ z)\ \mathrm{in}\ y \\ \xrightarrow[(\beta !)]{} & \mu x.\, \mathrm{let}\ !z = !M^{\flat\sharp}\ \mathrm{in}\ z \\ \xrightarrow[(\beta !)]{} & \mu x.\, M^{\flat\sharp} \quad . \end{array}$$

♦

Unfortunately, when we begin with a term in $\mathsf{Lin}_{\mathrm{rec}\,v}$ the development is less clear. The image under the composition of the two maps is then convertible only if we also consider an odd variant of the eta reduction rule $(¡\eta)$,

$$(¡\eta) \qquad !\mathrm{let}\ !x = M\ \mathrm{in}\ x \to M \quad .$$

Lemma A.4.2. Let $M \in \mathsf{Lin}_{\mathrm{rec}\,v}$. Then $M^{\flat\sharp} \xrightarrow[(¡\eta)]{} M$.

Proof: All cases are again trivial aside from the recursion binding:

$$\begin{array}{rl} & (\mu_v x.\, M)^{\sharp\flat} \\ \equiv & !\mathrm{let}\ !y = (\mu_v x.\, !\mathrm{let}\ !z = M^{\sharp\flat}\ \mathrm{in}\ z)\ \mathrm{in}\ y \\ \xrightarrow[(¡\eta)]{} & \mu_v x.\, !\mathrm{let}\ !z = M^{\sharp\flat}\ \mathrm{in}\ z \\ \xrightarrow[(¡\eta)]{} & \mu_v x.\, M^{\sharp\flat} \quad . \end{array}$$

♦

It is also easy to verify that $\flat$ is sound for reduction without the $(¡\eta)$ rule, and that $\sharp$ is sound for equality with the $(¡\eta)$ rule in $\mathsf{Lin}_{\mathrm{rec}}$. Since the two systems $\mathsf{Lin}_{\mathrm{rec}}$ and $\mathsf{Lin}_{\mathrm{rec}\,v}$ do not totally correspond, the soundness and completeness results of Proposition A.2.3 and Proposition A.3.3 for $\mathsf{Lin}_{\mathrm{rec}}$ do not transfer directly to analogous results for translations into $\mathsf{Lin}_{\mathrm{rec}\,v}$. Nonetheless, the alternate translations are also sound and complete.

Proposition A.4.3. The translation $\bar{\circ}$ from $\mathsf{Name}_{\mathrm{rec}}$ to $\mathsf{Lin}_{\mathrm{rec}\,v}$ preserves substitutions, preserves typing judgments and preserves reduction sequences: For $M, N \in \mathsf{Name}_{\mathrm{rec}}$,

(i) $(M[x := N])^{\bar{\circ}} \equiv (M^\circ)[x := N^{\bar{\circ}}]$.

(ii) $\Gamma \vdash_{\mathsf{Name}_{\mathrm{rec}}} M : A$ if and only if $\Gamma^{\bar{\circ}};\ - \vdash_{\mathsf{Lin}_{\mathrm{rec}\,v}} M^{\bar{\circ}} : A^{\bar{\circ}}$.

(iii) $M \xrightarrow[\mathsf{Name}_{\mathrm{rec}}]{} N$ if and only if $M^\circ \xrightarrow[\mathsf{Lin}_{\mathrm{rec}\,v}]{} N^\circ$.

Standard translation As for $\circ$ from Name, plus:

$$(\mu x.\,M)^{\bar{\circ}} \quad\equiv\quad \text{let } !y = \mu x.\,!M^{\bar{\circ}} \text{ in } y$$

Steadfast translation As for $*$ from Val or Let, plus:

$$(\mu f.\,\lambda x.\,M)^{\bar{+}} \quad\equiv\quad \text{let } !z = (\mu f.\,!\lambda y.\,\text{let } !x = y \text{ in } M^{\bar{*}}) \text{ in } z$$

Figure A.4: Extensions to the translations for mapping $\mathsf{Name}_{\mathrm{rec}}$ and $\mathsf{Val}_{\mathrm{rec}}$ into $\mathsf{Lin}_{\mathrm{rec}\,v}$.

Proof: Clauses (i) and (ii) follow by the usual arguments. For soundness in Clause (iii) we take $\mu x.\,M \xrightarrow[\mathsf{Name}_{\mathrm{rec}}]{} M[x := \mu x.\,M]$, and translating into $\mathsf{Lin}_{\mathrm{rec}\,v}$ we have:

$$\begin{array}{rl}
 & (\mu x.\,M)^{\bar{\circ}} \\
\equiv & \text{let } !y = \mu x.\,!M^{\bar{\circ}} \text{ in } y \\
\xrightarrow[(\mathrm{Rec}_v)]{} & \text{let } !y = !M^{\bar{\circ}}[x := \text{let } !y = \mu x.\,!M^{\bar{\circ}} \text{ in } y] \text{ in } y \\
\xrightarrow[(\beta !)]{} & M^{\bar{\circ}}[x := \text{let } !y = \mu x.\,!M^{\bar{\circ}} \text{ in } y] \\
\equiv & (M[x := \mu x.\,M])^{\bar{\circ}} \ .
\end{array}$$

For completeness we extend the grammar of reduction-closed $\circ$-images of Name terms by

$$S ::= \cdots \mid \text{let } !y = \mu x.\,!S \text{ in } y$$

and define the right-inverse map $\bar{\dagger}$ to be the extension of $\dagger$ by

$$(\text{let } !y = \mu x.\,!S \text{ in } y)^{\bar{\dagger}} \equiv \mu x.\,S^{\bar{\dagger}} \ .$$

Then we have the usual correspondence,

$$\begin{array}{rl}
 & (\text{let } !y = \mu x.\,!S \text{ in } y)^{\bar{\dagger}} \\
\equiv & \mu x.\,S^{\bar{\dagger}} \\
\xrightarrow[(\mathrm{Rec})]{} & S[x := \mu x.\,S^{\bar{\dagger}}] \\
\equiv & (\text{let } !y = !S[x := \text{let } !y = \mu x.\,!S \text{ in } y] \text{ in } y)^{\bar{\dagger}} \ .
\end{array}$$

♦

Proposition A.4.4. The translation $\bar{*}$ from $\mathsf{Val}_{\mathrm{rec}}$ to $\mathsf{Lin}_{\mathrm{rec}\,v}$ preserves substitution of values, preserves typing judgments and preserves reduction sequences: For $M, N \in \mathsf{Name}_{\mathrm{rec}}$,

(i) $(M[x := V])^{\bar{*}} \equiv (M^{\bar{*}})[x := V^{\bar{+}}]$.

(ii) $\Gamma \vdash_{\mathsf{Val}_{\mathrm{rec}}} M : A$ if and only if $\Gamma^{\bar{*}};\ - \vdash_{\mathsf{Lin}_{\mathrm{rec}\,v}} M^{\bar{*}} : A^{\bar{*}}$.

(iii) If $M \xrightarrow[\mathsf{Val}_{\mathrm{rec}}]{}\!\!\!\!\rightarrow N$ then $M^{\bar{*}} \xrightarrow[\mathsf{Lin}_{\mathrm{rec}\,v}]{}\!\!\!\!\rightarrow N^{\bar{*}}$.

Proof: Clauses (i) and (ii) follow by the usual arguments. For soundness in Clause (iii) we have the same reasoning as in the previous result, using $\mu x.\,\lambda y.\,M_0$ rather than just $\mu x.\,M$. ♦

Ultimately, the $\mathsf{Lin}_{\mathrm{rec}\,v}$ is merely a curiosity. While it is nearly equivalent to $\mathsf{Lin}_{\mathrm{rec}}$, the $(¡\eta)$ rule which seems necessary is flawed. The rule abruptly changes the evaluation properties of an answer; Barber has pointed out that its reverse would violate subject reduction, leading to a suspect notion of equality, and moreover that the rule is not sound for the usual models of intuitionistic linear logic [8].

Glossary of Metavariables

Header

- A — Answers (main calculi).
  Types (main calculi).
  Formulas (logics).
- B — Types (main calculi).
  Formulas (logics).
- C — Contexts (unmarked calculi).
- C' — Contexts (marked calculi).
- D — Function-holed contexts (SLin).
- E — Evaluation contexts (main calculi).
- $\mathcal{E}$ — Quaternary typing contexts (SLin).
- $\mathcal{F}$ — Quaternary typing contexts (SLin).
- f — Recurrence relations (marked and weighted calculi).
- $\mathcal{G}$ — Quaternary typing contexts (SLin).
- H — Speculative contexts (Name, Let, Lin).
- $\mathfrak{H}$ — (V, C, A, G)-normal applicable forms (Need).
- I — Identity function (all calculi).
- i — Integers, positive integers (N/A).
- j — Integers, positive integers (N/A).
- L — Terms (basic (unmarked) calculi).
- L' — Marked terms (marked calculi).
- $\dot{L}$ — Marked, weighted terms (marked and weighted calculi).
- $\mathfrak{L}$ — (V, C, A, G)-normal terms (Need).
- M — Terms (basic (unmarked) calculi).
- M' — Marked terms (marked calculi).
- $\dot{M}$ — Marked, weighted terms (marked and weighted calculi).
- $\mathfrak{M}$ — (V, C, A, G)-normal terms (Need).
- m — Integers, positive integers (N/A).
- N — Terms (basic (unmarked) calculi).
- N' — Marked terms (marked calculi).
- $\dot{N}$ — Marked, weighted terms (marked and weighted calculi).
- $\mathfrak{N}$ — (V, C, A, G)-normal terms (Need).
- n — Integers, positive integers (N/A).
- P — Binding prefixes (marked and unmarked calculi with bindings).
  Translation images of applied terms (Lin, Aff, SLin).
  Applicable prefixes, $\circledast$-image (SLin).
- $\mathcal{P}$ — Typing proof trees (type systems).
  Markable binding prefixes (marked generalized systems).
- Q' — Contexts or terms (marked calculi).
- $\mathcal{R}$ — Markable binding prefixes (marked generalized systems).
- R — Reduction rules (all calculi).
- S — Balanced prefixes (item notation).
  Translation images of terms (Lin, Aff, SLin).
- s — Steps (unmarked calculi).
- T — Translation images of terms (Lin, Aff, SLin).
- t — Steps (unmarked calculi).
- U — Translation images of values (Lin, Aff, SLin).
- V — Values (unmarked calculi).
- V' — Marked values (marked calculi).
- $\dot{V}$ — Marked, weighted values (weighted and marked calculi).
- $\mathfrak{V}$ — (V, C, A, G)-normal values (Need).
- W — Values (unmarked calculi).
- W' — Marked values (marked calculi).

$\dot{W}$ Marked, weighted values (weighted and marked calculi).

x Variable (all calculi).

y Variable (all calculi).

Z Basic types (type systems).
Atomic formulas (logics).

z Variable (all calculi).

Γ Typing environments (type systems).
Multisets of formulas (logics).

γ Paths (unmarked systems).

$\bar{\gamma}$ Sets of paths (unmarked systems).

Δ Typing environments (type systems).
Multisets of formulas (logics).
Top-level redexes (all calculi).

Δ' Contracted top-level redexes (all calculi).

δ Paths (unmarked systems).

$\bar{\delta}$ Sets of paths (unmarked systems).

ρ Paths (unmarked systems).

$\bar{\rho}$ Sets of paths (unmarked systems).

σ Reduction sequences (unmarked systems).

σ' Reduction sequences (marked systems).

τ Reduction sequences (unmarked systems).

τ' Reduction sequences (marked systems).

Φ Typing environments (type systems).
Multisets of formulas (logics).

φ Paths (unmarked systems).

$\bar{\varphi}$ Sets of paths (unmarked systems).

Ψ Typing environments (type systems).
Multisets of formulas (logics).

ψ Paths (unmarked systems).

$\bar{\psi}$ Sets of paths (unmarked systems).

$!^{\square}, !^{\triangle}$ Range over $!^{w}, !^{c}, !^{i}$ as allowed by context (SLin).

Glossary of Symbolic Notation

This glossary lists all symbolic notation with the following exceptions: Names of reduction rules are listed in the index under "Reduction, rule, *rule name*." Names of typing judgment rules are listed in the index under "Typing rule, *rule name*." Where certain notations may be parameterized by calculi or rules indexed elsewhere, we give only the unparameterized form, plus any special variant forms whose meaning might not be immediately clear from the context. Metavariables are listed in a separate glossary.

$\mathbf{d}$ (Damage function) *In the calculi with generalized commuting conversion rules, returns the number of bindings introduced or removed from the top* n *bindings by a reduction sequence.*

Ξ (Variable-paths function) *Given a term in* $\mathsf{SLin}^{\star\bullet}$, *returns the set of all paths into that term which index variables.*

$\urcorner$ (Remarking operator) *In the simply-marked calculi, removes markings for commuting conversion steps from (former) redexes whose inner binding was removed by a* β *(or similar) step.*

$\pitchfork$ (Let contraction) *In call-by-need, removes let-bindings by substituting the bound term for the bound variable in the body, regardless of whether the bound term has been reduced to a value.*

$\overline{\ell\lambda}$ (Let expansion) *In call-by-let, reverses the effect of the* (I) *rule, removing let-bindings by converting each binding* $(\mathrm{let}\ x = M\ \mathrm{in}\ N)$ *into an application* $(\lambda x.\, N)\, M$.

$\_|\_$ (Index of a subterm with a path) *In* Let *and* Lin, *returns the subexpression of a term indexed by the particular path.*

$\mathcal{C}^{-}_{-}$ (Context, construction of from paths) *In* Let *and* Lin, *converts a term into a context by replacing the subexpression indexed by the path with a hole.*

$[\,]/\_$ (Path, construction of from contexts) *In* Let *and* Lin, *returns the path which indexes the hole of a context.*

$-$ (Type environment, empty) *Some type environments contain no assumptions.*

$!$ (Mode, !) *Mode allowing Weakening and Contraction in the linear and affine systems.*

$|\,.\,|$ (Projection) *Simple removal of marks to convert a term of a marked system to the underlying unmarked term.*

$\|\,.\,\|$ (Weighted marked term, norm on) *Simple removal of marks to convert a term of a marked system to the underlying unmarked term.*

∂ (Path, valid for a term) *The set of paths which validly index subexpressions in the given term.*

$\circ$ (Standard translation) *The first of Girard's two translations from intuitionistic logic (lambda calculi) into intuitionistic linear logic (lambda calcului).*

$\dagger$ (Standard translation, inverse) *A right-inverse of the standard translation.*

$*$ (Steadfast translation) *The second of Girard's two translations from intuitionistic logic (lambda calculi) into intuitionistic linear logic (lambda calcului).*

$\ddagger$ (Steadfast translation, inverse) *A right-inverse of the steadfast translation.*

$+$ (Steadfast translation, of values) *A separate map for the steadfast translation of values under the outermost* $!$.

$\#$ (Steadfast translation, inverse, of values) *A right-inverse of the steadfast translation of values.*

$\blacktriangleright_l, \blacktriangleright_c, \blacktriangleright_w, \blacktriangleright_i$ (Zone, marker) *Markers for the (respectively) linear, relevant, affine and intuitionistic zones of a separated-linear typing context.*

$/$ (Residual) *Written as an operator; given redexes* $\bar{\gamma}$ *and a reduction sequence* σ*, we write* $\bar{\gamma}/\sigma$ *to indicate the residuals of* $\bar{\gamma}$ *with respect to* σ.

$\vdash$ (Typing rule) *Typing judgement relation.*

$\vdash_{\mathsf{SRF}}$ (Separated-linear lambda calculus) *Typing judgement relation for* SLin *without structural rules.*

$\multimap$ (Linear implication.) *The linear version of classical implication.*

Index

Bibliography

[1] M. Abadi, L. Cardelli, P.-L. Curien and J.-J. Lévy, Explicit substitutions, in: *Conf. Rec. POPL'90: 18th ACM Symp. on Principles of Programming Languages* (San Francisco, California, January 1990, ACM Press): p. 82.

[2] S. Abramsky, Computational interpretations of linear logic, *Theoretical Computer Science* **111** (1993) 3–57: p. 6, 99.

[3] Z. M. Ariola and M. Felleisen, The call-by-need lambda calculus (Technical Report CIS-TR-94-23, Department of Computer Science, University of Oregon, October 1994): p. 71, 81, 104.

[4] Z. M. Ariola and M. Felleisen, The call-by-need lambda calculus, to appear in: *J. Functional Programming* (Cambridge University Press, appearing 1997): p. 71, 81, 104.

[5] Z. M. Ariola, M. Felleisen, J. Maraist, M. Odersky and P. Wadler, A call-by-need lambda calculus, in: *Conf. Rec. POPL'95: 22nd ACM Symp. on Principles of Programming Languages* (San Francisco, California, January 1995, ACM Press) 233–246: p. 12, 71, 75, 82, 104.

[6] Z. M. Ariola and J. W. Klop, Cyclic lambda graph rewriting, in: *Proc. LICS'94: IEEE Conf. on Logic in Computer Science* (Paris, France, 1994): p. 81.

[7] A. Barber, DILL — dual intuitionistic linear logic (to appear; draft paper, Dual intuitionistic linear logic, available from the author, October 1995): p. 6, 10, 98, 148.

[8] A. Barber, private email (15 January 1996): p. 197.

[9] H. P. Barendregt, *The Lambda Calculus: Its Syntax and Semantics*, volume 103 of *Studies in Logic and the Foundations of Computer Science* (North-Holland Publishing Company, 1981): p. 17, 20, 21, 23, 24, 26, 37, 135.

[10] H. P. Barendregt, Lambda calculi with types, in: *Handbook of Logic in Computer Science* (Volume 2, Oxford University Press, Walton St., Oxford, England, 1992): p. 83, 170.

[11] H. P. Barendregt and K. Hemerik, Types in lambda calculus and programming languages, in: *Proc. ESOP'90: European Symp. on Programming* (Springer-Verlag, Lecture Notes in Computer Science 432, 1990) 1–35: p. 170.

[12] G. Bellin and P. J. Scott, On the π-calculus and linear logic, *Theoretical Computer Science* **135** (1994) 11–65: p. 6.

[13] P. N. Benton, G. Bierman, V. de Paiva and M. Hyland, Type assignment for intuitionistic linear logic (Technical Report 262, Computing Laboratory, University of Cambridge, August 1992): p. 99.

[14] P. N. Benton, Strong normalisation for the linear term calculus, *J. Functional Programming* **5**:1 (January 1995) 65–80: p. 99.

[15] G. Bierman, What is a categorical model of intuitionistic linear logic?, in: *Proc. Second International Conference on Typed Lambda Calculus* (Edinburgh, Scotland, April 1995, Springer Verlag, Lecture Notes in Computer Science 902): p. 99.

[16] G. Bierman, *On Intuitionistic Linear Logic* (Ph.D. Thesis, University of Cambridge Computer Laboratory, December 1993; also appears as Technical Report 346, Computing Laboratory, University of Cambridge, August 1994): p. 99.

[17] R. Bloo, F. Kamareddine and R. Nederpelt, The Barendregt cube with definitions and generalised reduction, *Information and Computation* **126** 2 (May 1996) 123–143: p. 82, 83, 170.

[18] T. Braüner, The Girard translation extended with recursion, in: *Proc. CSL'94: 8th Workshop on Computer Science Logic* (Kazimierz, Poland, September 1994, Springer Verlag, Lecture Notes in Computer Science 933; extended report in: Technical Report RS-95-13, BRICS Group, Dept. of Computer Science, University of Århus, February 1995): p. 193.

[19] L. E. J. Brouwer, *Brouwer's Cambridge Lectures on Intuitionism* (D. van Dalen, editor, Cambridge University Press, 1981): p. 9.

[20] A. Church, *The Calculi of Lambda Conversion* (Princeton University Press, 1941): p. 21.

[21] H. B. Curry and R. Feys, *Combinatory Logic*, vol. 1 (North-Holland, Amsterdam, 1958): p. 2, 6, 17, 26.

[22] O. Danvy and J. Hatcliff, Thunks (continued), in: *Proc. WSA'92: Workshop on Static Analysis* (1992; extended version in: Technical Report CIS-92-28, Kansas State University, 1992): p. 81.

[23] M. Felleisen and R. Hieb, The revised report on the syntactic theories of sequential control and state, *Theoretical Computer Science* (1992): p. 26.

[24] J. Field, On laziness and optimality in lambda interpreters: Tools for specification and analysis, in: *Conf. Rec. POPL'90: 18th ACM Symp. on Principles of Programming Languages* (San Francisco, California, January 1990, ACM Press): p. 82.

[25] G. Gentzen, *The Collected Papers of Gerhard Gentzen*, in *Studies in Logic and the Foundations of Computer Science* (M. E. Szabo, editor, North-Holland Publishing Company, 1969): ?.

[26] J.-Y. Girard, Linear logic, *Theoretical Computer Science* **50** (1987) 1–102: p. 2, 6, 87, 98, 111, 172, 179.

[27] J.-Y. Girard, On the unity of logic, *Annals of Pure and Applied Logic* **59** (1993) 201–217: p. 6, 87, 98, 179.

[28] J.-Y. Girard, Y. Lafont and P. Taylor, *Proofs and Types* (Cambridge University Press, 1989): p. 6.

[29] G. Gonthier, M. Abadi and J.-J. Levy, The geometry of optimal lambda reduction, in: *Conf. Rec. POPL'92: 18th ACM Symp. on Principles of Programming Languages* (Albuquerque, New Mexico, January 1992, ACM Press): p. 111.

[30] J. Hatcliff and O. Danvy, Thunks and the lambda-calculus (DIKU Technical Report 95/3, Datalogisk Institut, Københavns Universitet, 1995): p. 81.

[31] A. Heyting, *Intuitionism: An Introduction*, in *Studies in Logic and the Foundations of Computer Science* (North-Holland Publishing Company, third printing, 1976): p. 8.

[32] J. R. Hindley, The equivalence of complete reductions, *Transactions of the American Mathematical Society* **229** (1977) 227–248: p. 24.

[33] S. Holmström, A linear functional language (Draft paper, Chalmers University of Technology, 1988): p. 99.

[34] W. Howard, The formulae-as-types notion of construction, in: *To H. B. Curry: Essays on Combinatory Logic, Lambda-Calculus and Formalism* (J. R. Hindley and J. P. Seldin, eds., Academic Press, New York and London, 1980): p. 2, 6.

[35] R. J. M. Hughes, Why function programming matters, *The Computer Journal* **32** 2 (1989) 98-107: p. 1.

[36] B. Jacobs, Semantics of lambda-I and of other substructure lambda calculi, in: *Typed Lambda Calculus and Applications* (Springer LNCS 664, 1994) 195-208: p. 3, 127, 179.

[37] B. Jacobs, Semantics of weakening and contraction, *Annals of Pure and Applied Logic* **69** (1994) 73–106: p. 3, 123, 127, 137, 179.

[38] T. P. Jensen, Strictness analysis in logical form, in: *Conf. Rec. FPCA'91: Fifth ACM Symp. on Functional Programming Languages and Computer Architecture* (Cambridge, England, ACM Press, 1991): p. 12.

[39] M. B. Josephs, The semantics of lazy functional languages, *Theoretical Computer Science* **68** (1989) 105–111: p. 81.

[40] F. Kamareddine and R. Nederpelt, A useful λ notation, *Theoretical Computer Science* **155** 1 (Feb. 1996) 85–109: p. 82.

[41] J. W. Klop, *Combinatory Reduction Systems* (Ph.D. thesis, State University of Utrecht, the Netherlands, 1980): p. 17.

[42] T.-M. Kuo and P. Mishra, Strictness analysis: a new perspective based on type inference, in: *Conf. Rec. FPCA'89: Fourth ACM Symp. on Functional Programming Languages and Computer Architecture* (ACM Press, 1989): p. 12.

[43] Y. Lafont, The linear abstract machine, *Theoretical Computer Science* **59** (1988) 157–180: p. 99.

[44] J. Launchbury, A natural semantics for lazy evaluation, in: *Conf. Rec. POPL'93: 19th ACM Symp. on Principles of Programming Languages* (Charleston, South Carolina, January 1993, ACM Press): p. 82.

[45] P. Lincoln and J. Mitchell, Operational aspects of linear lambda calculus, in: *Proc. LICS'92: Seventh IEEE Symp. on Logic in Computer Science* (Santa Cruz, California, June 1992, IEEE Press): p. 99.

[46] I. Mackie, *Lilac: a Functional Programming Language Based on Linear Logic* (Master's Thesis, Imperial College London, 1991): p. 99.

[47] I. Mackie, *The Geometry of Implementation* (Ph.D. Thesis, Imperial College London, 1994): p. 3, 111.

[48] J. Maraist, Linear term reduction, recursion and the Girard translations, in *Proc. Fifth Workshop on Functional-Logic Programming* (Schloß Rauischholzhausen, Marburg, Germany, 1996): p. 193.

[49] J. Maraist, Finer control of weakening and contraction: a separated-linear lambda calculus, in: *Proc. Workshop on Multi-Paradigm Logic Programming* (Technical Report 96-28, Fachbereichs Informatik, Technische Universität Berlin, September 1996): p. 127.

[50] J. Maraist, Separating weakening and contraction in a linear lambda calculus (unabridged) (Technical Report 25/96, Fakultät für Informatik, Universität Karlsruhe, September 1996): p. 127.

[51] J. Maraist, M. Odersky, D. N. Turner and P. Wadler, Call-by-name, call-by-value, call-by-need and the linear lambda calculus, in: *Proc. MFPS'95: Eleventh Conf. on the Mathematical Foundations of Programming Semantics* (New Orleans, March/April 1995, Elsevier Publishers, ENTCS 1; extended report in: Technical Report 5/95, Fakultät für Informatik, Universität Karlsruhe, and in: Technical Report FP-94-12, Department of Computing Science, University of Glasgow; March 1995): p. 3, 31, 98, 111, 116, 171, 172, 179, 195.

[52] J. Maraist, M. Odersky, D. N. Turner and P. Wadler, Call-by-name, call-by-value, call-by-need and the linear lambda calculus, submitted to: *Theoretical Computer Science*, special issue on papers presented at MFPS'95 (Elsevier Publishers): p. 3, 31, 98, 111, 171, 172, 179, 193.

[53] J. Maraist, M. Odersky and P. Wadler, The call-by-need lambda calculus (Unabridged) (Technical Report 28/94, Fakultät für Informatik, Universität Karlsruhe and Technical Report FP-94-11, Department of Computing Science, University of Glasgow; October 1994): p. 12, 63, 71, 75, 82, 104.

[54] J. Maraist, M. Odersky and P. Wadler, The call-by-need lambda calculus, submitted to: *Journal of Functional Programming* (Cambridge University Press, submitted August 1995, revised/resubmitted January 1997): p. 63, 71, 82, 104.

[55] L. Maranget, Optimal derivations in weak lambda-calculi and in orthogonal term rewriting systems, in: *Proc. POPL'91: 19th ACM Symp. on Principles of Programming Languages* (Orlando, Florida, January 1991, ACM Press) 255–269: p. 82.

[56] R. Milner, A theory of type polymorphism in programming, *Journal of Computer and System Sciences* **17** (December 1978) 348–375: p. 12.

[57] E. Moggi, Computational lambda-calculus and monads (Technical Report ECS-LFCS-88-66, Laboratory for the Foundations of Computer Science, University of Edinburgh, October 1988): p. 169.

[58] E. Moggi, Computational lambda-calculus and monads, in: *Proc. LICS'89: Fourth IEEE Symp. on Logic in Computer Science* (IEEE Press, Asilomar, California, June 1989): p. 169.

[59] C. Mossin, D. N. Turner and P. Wadler, Once upon a type, in: *Conf. Rec. FPCA'95: Seventh ACM Symp. on Functional Programming and Computer Architecture* (San Diego, California, June 1995, ACM Press): p. 12, 111, 170.

[60] M. H. A. Newman, On theories with a combinatorial definition of "equivalence," *Annals of Mathematics* **2** 43 (1942) 223–243: p. 21.

[61] C. Okasaki, P. Lee and D. Tarditi, Call-by-need and continuation-passing style, *Lisp and Symbolic Computation* **7** (1994) 57-81: p. 21.

[62] G. D. Plotkin, Call-by-name, call-by-value and the λ calculus, *Theoretical Computer Science* **1** (1975) 125–159: p. 1, 26, 30, 31.

[63] S. Purushothaman and J. Seaman, An adequate operational semantics of sharing in lazy evaluation, in: *Proc. ESOP'92, Fourth European Symp. on Programming* (B. Krieg-Brückner, editor, *Lecture Notes in Computer Science 582*, Springer-Verlag, New York, February 1992) 435–450: p. 82.

[64] J. C. Reynolds, The discoveries of continuations, *Lisp and Symbolic Computation* **6**:3/4 (November 1993) 233–248: p. 31.

[65] S. R. della Rocca and L. Roversi, Lambda calculus and intuitionistic linear logic (manuscript, July 1994). Available from L. Roversi, University of Pisa, rover@di.unipi.it: p. 99.

[66] Mark Ryan and Martin Sadler, Valuation systems and consequence relations, in: *Handbook of Logic in Computer Science* (Volume 1, Oxford University Press, Walton St., Oxford, England, 1992): p. 6.

[67] A. Sabry and M. Felleisen, Reasoning about programs in continuation-passing style, *Lisp and Symbolic Computation* **6**:(3/4) (November 1993) 289–360: p. 31.

[68] A. Sabry and P. Wadler, A reflection on call-by-value (Technical Report, Department of Computer and Information Science, University of Oregon, April 1996): p. 169.

[69] A. Scedrov, A brief guide to linear logic, *Current Trends in Computer Science* (G. Rozenberg and A. Salomaa, World Science Publishing Co., 1993): p. 6.

[70] H. Schellinx, *The Noble Art of Linear Decorating* (Ph.D. dissertation, Institute for Logic, Language, and Computation, University of Amsterdam, 1994): p. 111.

[71] R. A. G. Seely, Linear logic, $*$-autonomous categories, and cofree coalgebras, in: *Categories in Computer Science and Logic* (AMS Contemporary Mathematics 92, June 1989): p. 99.

[72] Q. Tarentino, *Pulp Fiction* (1993): p. 50.

[73] A. S. Troelstra, *Lectures on Linear Logic* (CSLI Lecture Notes, 1992): p. 99, 111.

[74] D. N. Turner and P. Wadler, Operational interpretations of linear logic (manuscript, July 1996): p. 170.

[75] P. Wadler, Linear types can change the world!, in: M. Broy and C. Jones, eds., *Programming Concepts and Methods* (North Holland, Sea of Galilee, Israel, April 1990): p. 99.

[76] P. Wadler, Is there a use for linear logic?, in: *Proc. PEPM'91: ACM Workshop on Partial Evaluation and Semantics-Based Program Manipulation*, (New Haven, Connecticut, June 1991, ACM Press): p. 99.

[77] P. Wadler, A syntax for linear logic, in: *Proc. MFPS'93: Ninth Int. Conf. on the Mathematical Foundations of Programming Semantics* (New Orleans, Louisiana, April 1993, Springer Verlag, Lecture Notes in Computer Science 802): p. 10, 98, 99.

[78] P. Wadler, A taste of linear logic, in: *Proc. MFCS'93: Mathematical Foundations of Computer Science* (Gdansk, Poland, August 1993, Springer Verlag, Lecture Notes in Computer Science 711): p. 98, 99.

[79] P. Wadler, Lazy vs. Strict, *ACM Computing Surveys* (June 1996): p. 1.

[80] C. Wadsworth, *Semantics and Pragmatics of the Lambda-Calculus* (Ph.D. Thesis, Oxford University, 1971): p. 81.

[81] N. Yoshida, Optimal reduction in weak-lambda-calculus with shared environments, in: *Conf. Rec. FPCA'93: Symp. on Functional Programming Languages and Computer Architecture* (Copenhagen, Denmark, June 1993, ACM Press): p. 82.

I. Banks, *The Crow Road* (Abacus, 1992).

J. C. Chase, *Frenchmen, Desire, Good Children and Other Streets of New Orleans* (MacMillian, 1949, revised 1960, 1979).

M. Stipe, "Feeling gravitys pull," in: R.E.M., *Fables of the Reconstruction/Reconstruction of the Fables* (IRS, 1984).

J. M. Straczynski, "Parliament of Dreams," *Babylon 5* **1** 5 (1994).