Labeling techniques and typed fixed-point operators

John C. Mitchell
Department of Computer Science
Stanford University
mitchell@cs.stanford.edu

My Hoang
SAP Technology, Inc.
950 Tower Lane, 16th Floor
Foster City, CA 94404
my.hoang@sap-ag.de

Brian Howard
Department of Computing & Information Sciences
Kansas State University
bhoward@cis.ksu.edu

July 2, 1996

Abstract

Labeling techniques for untyped lambda calculus were developed by Lévy, Hyland, Wadsworth and others in the 1970's. A typical application is the proof of confluence from finiteness of developments: by labeling each subterm with an natural number indicating the maximum number of times this term may participate in a reduction step, we obtain a strongly-normalizing approximation of β, η-reduction. Confluence then follows by a syntactic "compactness" argument (repeated in the introduction of this paper).

This paper presents applications of labeling to typed lambda calculi with fixed-point operators, including confluence and completeness of left-most reduction for PCF (an "applied" lambda calculus with fixed-point operators and numeric and boolean operations) and a confluence proof for a variant of typed lambda calculus with subtyping. Labeling is simpler for typed calculi than untyped calculi because the fixed-point operator is the only source of nontermination. We can also use the method of logical relations for the labeled typed calculus, extended with basic operations like addition and conditional. This method would not apply to untyped lambda calculus.

1 Introduction

This paper presents a proof method for extensions of typed lambda calculus with fixed-point operators that is based on labeled reduction. The main idea, in brief, is that we may prove properties of a typed functional language by first labeling every occurrence of the fixed-point operator with a non-negative integer giving the maximum number of times this operator may be used in computation. This produces a language with "bounded recursion." We may prove confluence (the Church-Rosser property) and termination of reduction with bounded recursion, for example, using the same techniques that would apply for pure typed lambda calculus. While some properties, such as termination, may fail when labels are removed, others, such as confluence, may be preserved. Although this paper (prepared in connection with the first author's invited talk at the Higher-Order Operational Techniques workshop) is primarily concerned with operational techniques, there are also applications of labeling in denotational semantics.

We develop two sets of results. The first are for typed lambda calculus extended with fixed-point operators and any set of additional operations satisfying certain conditions. The second are for typed lambda calculus with subtyping. In the first case, we prove confluence and termination of labeled calculus, then use these to show confluence for the unlabeled calculus and completeness of left-most reduction. While many readers may have assumed that these properties of lambda calculus also hold when types, recursion and operations such as integer arithmetic are added, we were unable to find any prior proofs of these properties in the literature. For the second system, we observe that confluence fails for β,η-reduction in the presence of subtyping. This problem is repaired by adding an intuitive but unusual reduction rule. We then use the labeling technique to extend our confluence proof to typed lambda calculus with subtyping and fixed-point operators. This use of labeling is technically interesting since the confluence proof for typed lambda calculus with subtyping relies on termination.

The inspiration for our use of labeling comes from the untyped lambda calculus, where labeling techniques were developed by Levy, Hyland, Wadsworth and others in the 1970's [Bar84, Hyl76, Lév75, Wad76]. A good illustrative example is the proof of confluence from finiteness of developments: by labeling each subterm with an natural number indicating the maximum number of times this term may participate in a reduction step, we obtain a strongly-normalizing approximation of β,η-reduction. Confluence then follows by a syntactic "compactness" argument. While the techniques for proving properties of labeled systems differ between typed and untyped systems, the transfer from labeled to unlabeled versions are similar. Since this argument provides the basis for the unifying technique of this paper, we illustrate it briefly using untyped lambda calculus. Since this description is meant only to provide intuition, we will not be concerned with the details. The main properties used here will be proved later for typed systems.

The main reduction (symbolic evaluation) rule of untyped lambda calculus is β-reduction,

$$(\beta) \qquad (\lambda x.U)V \to [V/x]U$$

where $[V/x]U$ is the result of substituting expression V for all free occurrences of variable x in U. (See [Bar84] for full presentation and discussion.) It is easy to find untyped lambda terms that may be reduced indefinitely without reaching a *normal form,* a term that cannot be further reduced. For example, we have

$$(\lambda x.\, xx)(\lambda x.\, xx) \to [(\lambda x.\, xx)/x]xx \equiv (\lambda x.\, xx)(\lambda x.\, xx)$$

However, we can eliminate infinite sequences of reduction by labeling each term with a natural

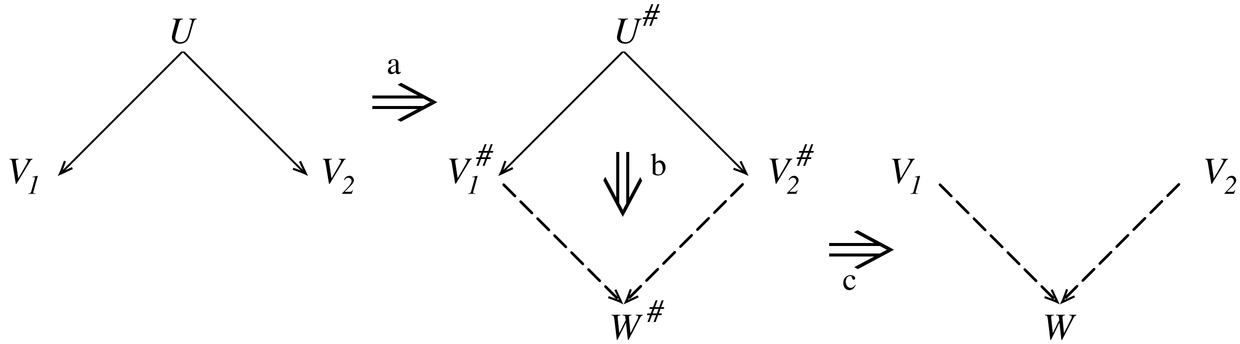

Figure 1: Confluence proof using labeling.

number and limiting reduction to positive numbers as follows:

$$(\beta_{n+1}) \qquad (\lambda x.U)^{n+1}V \to ([V^n/x]U)^n$$

where V^n is result of replacing each label ℓ in V by the minimum of ℓ and n. For example, we have

$$(\lambda x.\, xx)^{n+1}(\lambda x.\, xx)^{n+m} \to [(\lambda x.\, xx)^n/x]xx \equiv (\lambda^n x.\, xx)^n(\lambda^n x.\, xx)^n$$

For this example, it is easy to see that since the labels decrease, repeated labeled reduction must eventually halt. In fact, labeled reduction is confluent and terminating on untyped lambda terms [Bar84].

The main properties connecting labeled and unlabeled reduction are given below, using $U^\#$ for an arbitrary labeling of U and $\natural(L)$ for the result of erasing labels from L:

> **Projection:** If $U^\# \twoheadrightarrow L$, for any unlabeled term U and labeled term L, then $U \twoheadrightarrow \natural(L)$.
>
> **Lifting:** If $U_1, \ldots, U_k \twoheadrightarrow V$, for any unlabeled terms $U_1, \ldots, U_k, V$, then there exist labelings $U_1^\#, \ldots, U_k^\#, V^\#$ of these terms such that $U_i^\# \twoheadrightarrow V^\#$ for $1 \le i \le k$.

We will use analogous properties of typed labeling in each of our labeling proofs. Intuitively, lifting and projection are very natural properties if we think of labeling as restricting the use of some computational resource and erasing the label as dropping this restriction. For example, we can consider projection as a formal statement of the intuitive property that if a program computes a result when we allocate at most n computation steps to the program, the the program will compute the same result if we run it without any such limit.

Using lifting and projection, along with confluence of labeled reduction, we can prove confluence of untyped lambda calculus as follows. Assume that untyped term U reduces to terms V_1 and V_2. We must show that there exists some term W with $V_1 \twoheadrightarrow W$ and $V_2 \twoheadrightarrow W$. The argument has three steps, also shown diagrammatically in Figure 1.

(a) Lift both reductions to obtain labeled reductions from some $U^\#$ to some $V_1^\#$ and $V_2^\#$.

(b) By confluence of labeled reduction, $V_1^\#$ and $V_2^\#$ must reduce to some common (labeled) term $W^\#$.

(c) Project these labeled reductions to obtain unlabeled reductions as desired.

This proves confluence for untyped lambda calculus.

After defining the appropriate versions of typed lambda calculus in Section 2, we review a general theorem based on the logical relation method in Section 3. Confluence and completeness of left-most reduction for typed lambda calculi with fixed point are covered in Section 4, with subtyping considered in Section 5.

2 Syntax, Equations and Reduction Rules

2.1 Typed lambda terms

Typed lambda calculus may be defined with empty, one-element, disjoint union, product and function types. For simplicity, we will work only with function types in this paper. The type expressions of typed lambda calculus with function types, $\lambda^{\rightarrow}$, are given by the grammar

$$\sigma \ ::= \ b \,|\, \sigma \rightarrow \sigma$$

where b may be any type constant. The extension of certain properties to product types appears in [Mit96], along with some discussion of the failure of confluence with initial (one-element) types.

The compound expressions and their types are defined using inference rules. The rules use typing assertions of the form

$$\Gamma \triangleright M : \tau,$$

where Γ is a *type assignment* of the form

$$\Gamma = \{x_1\colon \sigma_1, \ldots, x_k\colon \sigma_k\},$$

with no x_i occurring twice. Intuitively, the assertion $\Gamma \triangleright M : \tau$ says that if variables $x_1, \ldots, x_k$ have types $\sigma_1, \ldots, \sigma_k$ (respectively), then M is a well-formed term of type τ. If Γ is any type assignment, we will write $\Gamma, x\colon \sigma$ for the type assignment

$$\Gamma, x\colon \sigma \ = \ \Gamma \cup \{x\colon \sigma\}.$$

In doing so, we always assume that x does not appear in Γ.

The syntax of terms depends on the choice of type and term constants. A $\lambda^{\rightarrow}$ *signature* $\Sigma = \langle B, C\rangle$ consists of

- A set B whose elements are called *base types* or *type constants*.
- A collection C of pairs of the form $\langle c, \sigma\rangle$, where σ is a $\lambda^{\rightarrow}$ type expression over B and no c occurs in two distinct pairs.

A symbol c occurring in some pair $\langle c, \sigma\rangle \in C$ is called a *term constant of type* σ. We generally write $c\colon \sigma$ if $\langle c, \sigma\rangle \in C$. Note that the type and term constants must be consistent, in that the type of each term constant may only contain the given type constants.

Example 2.1 An example $\lambda^{\rightarrow}$ signature is Σ_{PCF}, which gives us the language PCF (without cartesian products). This signature provides symbols for natural number and boolean operations, together with fixed-point operators at all types.

type constants:	$nat, bool$
term constants:	$0, 1, 2, 3, 4, \ldots : nat$
	$true, false\colon bool$
	$plus\colon nat \rightarrow nat \rightarrow nat$
	$Eq?\colon nat \rightarrow nat \rightarrow bool$
	$cond_\sigma\colon bool \rightarrow \sigma \rightarrow \sigma \rightarrow \sigma$ each type σ
	$fix_\sigma\colon (\sigma \rightarrow \sigma) \rightarrow \sigma$ each type σ

We may write terms over this signature in a more familiar form using syntactic sugar such as

$$\begin{array}{lcl} M + N & \stackrel{def}{=} & \mathit{plus}\, M\, N \\ \texttt{if } M \texttt{ then } N \texttt{ else } P & \stackrel{def}{=} & \mathit{cond}_\sigma\, M\, N\, P \end{array}$$

where the type subscript on *cond* is determined by the types of N and P. ■

The $\lambda^\rightarrow$ terms over signature Σ and their types are defined simultaneously using axioms and inference rules. For each term constant c of type σ, we have the axiom

$$(cst) \qquad \emptyset \rhd c : \sigma$$

We assume some countably infinite set *Var* of variables $\{v_0, v_1, \ldots\}$. Variables are given types by the axiom

$$(var) \qquad x{:}\,\sigma \rhd x : \sigma$$

where σ must be a $\lambda^\rightarrow$ type over Σ. The rule

$$(add\ var) \qquad \frac{\Gamma \rhd M : \sigma}{\Gamma, x{:}\,\tau \rhd M : \sigma}$$

allows us to add an additional hypothesis to the typing context. For lambda abstraction, we have

$$(\rightarrow \text{Intro}) \qquad \frac{\Gamma, x{:}\,\sigma \rhd M : \tau}{\Gamma \rhd (\lambda x{:}\,\sigma.\, M) : \sigma \rightarrow \tau}$$

Intuitively, this rule says that if M specifies a result of type τ for every $x{:}\,\sigma$, then the expression $\lambda x{:}\,\sigma.\, M$ defines a function of type $\sigma \rightarrow \tau$. Function applications are written according to the rule

$$(\rightarrow \text{Elim}) \qquad \frac{\Gamma \rhd M : \sigma \rightarrow \tau,\ \Gamma \rhd N : \sigma}{\Gamma \rhd MN : \tau}$$

which says that we may apply any function with type $\sigma \rightarrow \tau$ to an argument of type σ to produce a result of type τ.

We say M *is a* $\lambda^\rightarrow$ *term over signature* Σ *with type* τ *in context* Γ if $\Gamma \rhd M : \tau$ is either a typing axiom for Σ, or follows from axioms by rules $(add\ var)$, $(\rightarrow$ Intro$)$ and $(\rightarrow$ Elim$)$. As an expository convenience, we will often write $\Gamma \rhd M : \tau$ to mean that "$\Gamma \rhd M : \tau$ is derivable," in much the same way as one often writes a formula $\forall x.P(x)$, in logic, as a way of saying "$\forall x.P(x)$ is true." A proof of a typing assertion is called a *typing derivation.*

2.2 Equations

We write equations between typed lambda terms in a form that includes the assignment of types to variables. Since the types of terms will be used in the equational proof system, we also include the types of terms. Specifically, a typed equation has the form

$$\Gamma \rhd M = N : \tau$$

where we assume that M and N have type τ in context Γ. Intuitively, the equation $\{x_1{:}\,\sigma_1, \ldots, x_k{:}\,\sigma_k\} \rhd M = N : \tau$ means that for all type-correct values of the variables $x_1{:}\,\sigma_1, \ldots, x_k{:}\,\sigma_k$, expressions M and N denote the same element of type τ. Another way of writing this equation might be

$$\forall x_1{:}\,\sigma_1 \ldots \forall x_k{:}\,\sigma_k.\, M = N{:}\,\tau.$$

A "structural" rule for equations is

$$(add\ var) \qquad \frac{\Gamma \triangleright M = N : \sigma}{\Gamma, x{:}\,\tau \triangleright M : N\sigma}$$

which lets us add variables to the type assignment. We also have an axiom and inference rules making provable equality an equivalence relation and a congruence.

$$(ref) \qquad \Gamma \triangleright M = M : \sigma$$

$$(sym) \qquad \frac{\Gamma \triangleright M = N : \sigma}{\Gamma \triangleright N = M : \sigma}$$

$$(trans) \qquad \frac{\Gamma \triangleright M = N : \sigma,\ \Gamma \triangleright N = P : \sigma}{\Gamma \triangleright M = P : \sigma}$$

$$(\xi) \qquad \frac{\Gamma, x{:}\,\sigma \triangleright M = N : \tau}{\Gamma \triangleright \lambda x{:}\,\sigma.\, M = \lambda x{:}\,\sigma.\, N : \sigma \to \tau}$$

$$(\nu) \qquad \frac{\Gamma \triangleright M_1 = M_2 : \sigma \to \tau,\ \Gamma \triangleright N_1 = N_2 : \sigma}{\Gamma \triangleright M_1 N_1 = M_2 N_2 : \tau}$$

For $\lambda^{\to}$, three axioms remain. The first describes renaming of bound variables, while the other two specify that the introduction and elimination rules are "inverses" of each other. The axiom for renaming bound variables is

$$(\alpha) \qquad \Gamma \triangleright \lambda x{:}\,\sigma.\, M = \lambda y{:}\,\sigma.[y/x]M : \sigma \to \tau, \text{ provided } y \notin FV(M)$$

The second axiom, (β), shows how to evaluate a function application using substitution.

$$(\beta) \qquad \Gamma \triangleright (\lambda x{:}\,\sigma.\, M)N = [N/x]M : \tau$$

Finally, we have an axiom for equating extensionally equivalent function expressions,

$$(\eta) \qquad \Gamma \triangleright \lambda x{:}\,\sigma.(Mx) = M : \sigma \to \tau, \text{ provided } x \notin FV(M)$$

It is easy to see that if $x \notin FV(M)$, then by (β) we have $(\lambda x{:}\,\sigma.\, Mx)y = My$ for any argument $y{:}\,\sigma$. Therefore M and $\lambda x{:}\,\sigma.\, Mx$ define the same function.

For PCF, we add several "non-logical" axioms. (These are "non-logical" in the sense that they are specific axioms about term constants, not "logical" axioms that apply to all languages based on typed lambda calculus.) For natural numbers and conditional, we have infinitely many equational axioms:

$$\begin{array}{l} 0+0=0 \quad 0+1=1 \quad 1+0=1 \quad \ldots \quad 3+2=5 \quad \ldots \\ Eq?\,0\,0 = true \quad Eq?\,0\,1 = false \quad Eq?\,1\,0 = false \quad Eq?\,1\,1 = true \quad \ldots \\ \texttt{if } true \texttt{ then } M \texttt{ else } N = M \qquad \texttt{if } false \texttt{ then } M \texttt{ else } N = N \end{array}$$

The axiom for *fix* is

$$(fix) \qquad fix_\sigma = \lambda f{:}\,\sigma \to \sigma.\, f(fix_\sigma\, f)$$

from which it is esy to derive equations such as $fix_\sigma M = M(fix_\sigma\, M)$.

2.3 Reduction, Convertibility and Confluence

Reduction is a "directed" form of equational reasoning that corresponds to symbolic evaluation of programs. In simply-typed lambda calculus, we orient each of the equational axioms except (α). Reductions for term forms associated with products, sums and other simple types may be found in [Gun92, Mit96] and elsewhere. We discuss PCF reduction below.

While we are only interested in reducing typed terms, we may define reduction on simply-typed terms without mentioning types. Since reduction models program evaluation, this is a way of emphasizing that $\lambda^{\rightarrow}$ evaluation may be done without examining the types of terms. We will also see that the type of a term does not change as it is reduced. However, the connections between typed and untyped reduction may become more subtle in the presence of subtyping.

For clarity, we repeat the equational axioms in their reduction form.

$$(\beta)_{red} \qquad (\lambda x\colon\sigma.\, M)N \rightarrow [N/x]M,$$

$$(\eta)_{red} \qquad \lambda x\colon\sigma.\, Mx \rightarrow M, \text{ provided } x \notin FV(M).$$

A term of the form $(\lambda x\colon\sigma.\, M)N$ is called a *β-redex* and $\lambda x\colon\sigma.\, Mx$ an *η-redex*. We say M *β,η-reduces to N in one step,* written $M \rightarrow_{\beta,\eta} N$, if N can be obtained by applying (β) or (η) to some subterm of M. The reduction relation $\twoheadrightarrow_{\beta,\eta}$ is the reflexive and transitive closure of one-step β,η-reduction.

It can be shown that one step reduction preserves type.

Lemma 2.2 *If $\Gamma \triangleright M : \sigma$, and $M \rightarrow_{\beta,\eta} N$, then $\Gamma \triangleright N : \sigma$.*

It follows by an easy induction that $\twoheadrightarrow_{\beta,\eta}$ also preserves type. This is often called the *subject reduction property,* based on terminology that regards $M\colon\sigma$ as a "sentence" whose subject is M and predicate is σ.

It is useful to write $\Gamma \triangleright M \twoheadrightarrow N : \sigma$ when $\Gamma \triangleright M : \sigma$ is well-typed and $M \twoheadrightarrow N$. We know by the Lemma above that in this case, we also have $\Gamma \triangleright N : \sigma$. A term M is in *β,η-normal form* if there is no N with $M \rightarrow_{\beta,\eta} N$.

The main theorems about β,η-reduction are confluence and strong normalization. These may be proved using the technique of logical relations (summarized in Section 3).

- **Confluence** : β,η-Reduction is confluent on $\lambda^{\rightarrow}$ terms.
- **Strong Normalization**: There is no infinite sequence $M_0 \rightarrow_{\beta,\eta} M_1 \rightarrow_{\beta,\eta} M_2 \rightarrow_{\beta,\eta} \ldots$ of β,η-reductions on $\lambda^{\rightarrow}$ terms.

The second property is called "normalization" since it states that every term may be reduced to a normal form (a term that cannot be reduced further). The "strong" part of the statement is that a normal form is reached by *any* sequence of reductions. In contrast, weak normalization is the property that every term may be reduced to a normal form by some sequence of reductions, but not necessarily all.

For PCF, we adopt several "non-logical" reductions based on the corresponding equational axioms. Orienting the equations above in the intuitively plausible direction gives us

$$\begin{array}{l}
0+0 \rightarrow 0 \quad 0+1 \rightarrow 1 \quad 1+0 \rightarrow 1 \quad \ldots \quad 3+2 \rightarrow 5 \quad \ldots \\
Eq?\,0\,0 \rightarrow \mathit{true} \quad Eq?\,0\,1 \rightarrow \mathit{false} \quad Eq?\,1\,0 \rightarrow \mathit{false} \quad Eq?\,1\,1 \rightarrow \mathit{true} \quad \ldots \\
\texttt{if } \mathit{true} \texttt{ then } M \texttt{ else } N = M \qquad \texttt{if } \mathit{false} \texttt{ then } M \texttt{ else } N = N
\end{array}$$

The reduction axiom for *fix* is

$$(\mathit{fix}) \qquad\qquad \mathit{fix}_\sigma \quad \rightarrow \quad \lambda f\!:\sigma\rightarrow\sigma.\, f(\mathit{fix}_\sigma\, f)$$

from which it is easy to derive reductions such as $\mathit{fix}_\sigma M \twoheadrightarrow M(\mathit{fix}_\sigma\, M)$. It is easy to see that *fix*-reduction destroys strong normalization. However, it may be shown using the techniques given in this paper that various versions of PCF with arithmetic, boolean operations and fixed-point operators are confluent. (See [Mit96] for further information.)

It is worth noting that β,η-reduction is not confluent on *pre-terms,* strings that look like terms but are not necessarily well-typed. To see this, consider the pre-term

$$\lambda x\!:\sigma.(\lambda y\!:\tau.\, y)\, x$$

Using β-reduction, we may simplify this to $\lambda x\!:\sigma.\, x$, while η-reduction gives us $\lambda y\!:\tau.\, y$. Since these normal forms differ by more than names of bound variables when $\sigma \neq \tau$, confluence fails for pre-terms.

One consequence of this example, which is taken from [vD80, Ned73], is that confluence for typed lambda calculus does not follow immediately from the confluence of untyped lambda calculus, even though the typed terms could be considered as a subset of the untyped terms (*c.f.* [Bar84, Appendix A]). The reason is that the simple "proof" of confluence for typed lambda calculus by appeal to the Church-Rosser theorem for untyped lambda calculus applies to pre-terms as well as typed terms. Since this leads to an incorrect conclusion for pre-terms, it is not a correct proof for typed terms. The reader familiar with other presentations of typed lambda calculus may wonder whether this is still the case if we do not write type expressions in typed terms, but use variables that are each given a fixed type. In this alternate presentation of $\lambda^{\rightarrow}$, α-conversion must be restricted so that we only replace one bound variable by another with the same type. With this restriction on α-conversion, the example demonstrating failure of confluence still applies. Thus confluence for $\lambda^{\rightarrow}$ does not seem to follow from the Church-Rosser theorem for untyped β,η-reduction directly. It is worth noting, however, that if we drop η-reduction, then we *do* have confluence for β-reduction on $\lambda^{\rightarrow}$ pre-terms.

The convertibility relation $\leftrightarrow_{\beta,\eta}$ on typed terms is the least type-respecting equivalence relation containing reduction $\twoheadrightarrow_{\beta,\eta}$. For typographical simplicity, we will drop the β,η subscripts for the rest of this paragraph. Conversion can be visualized by saying that $\Gamma \rhd M \leftrightarrow N : \sigma$ iff there is a sequence of terms $M_0,\ldots,M_k$ with $\Gamma \rhd M_i : \sigma$ such that

$$M \equiv M_0 \twoheadrightarrow M_1 \twoheadleftarrow \ldots \twoheadrightarrow M_k \equiv N.$$

In this picture, the directions of $\twoheadrightarrow$ and $\twoheadleftarrow$ need not be regarded as significant. However, by reflexivity and transitivity of $\twoheadrightarrow$, this order of reduction and "backward reduction" is completely general.

A consequence of confluence is the following connection between reduction and provable equality.

Corollary 2.3 *An equation $\Gamma \rhd M = N : \tau$ is provable from the axioms of $\lambda^{\rightarrow}$ iff $\Gamma \rhd M \leftrightarrow N : \tau$ iff there is some term P with $M \twoheadrightarrow_{\beta,\eta} P$ and $N \twoheadrightarrow_{\beta,\eta} P$.*

This illustrates one of the general interests in confluence, namely, confluence implies a connection between reduction and equational reasoning. This connection may be used to prove the consistency of the equational proof system or, more generally, to show that certain equations are not provable from a set of equational axioms. The other general reason for studying confluence is that when we regard reduction as a model of program execution, confluence implies that the result of a computation is independent of evaluation order.

2.4 Subtyping

2.4.1 Programming language motivation

Subtyping appears in a variety of programming languages. An early form of subtyping appears in the Fortran treatment of "mixed mode" arithmetic: arithmetic expressions may be written using combinations of integer and real (floating point) expressions, with integers converted to real numbers as needed. The conversion of integers to reals has some of the properties that are typical of subtyping, since we generally think of the mathematical integers as a subset of the real numbers. However, conversion in programs involves changing the representation of a number, which is not typical of subtyping with records or objects. Fortran mixed mode arithmetic also goes beyond basic subtyping since (i) Fortran provides implicit conversion from reals to integers by truncation, which is different since this operation changes the value of the number that is represented, and (ii) Fortran also provides overloaded operations, such as $+ \colon \mathit{int} \times \mathit{int} \rightarrow \mathit{int}$ and $+ \colon \mathit{real} \times \mathit{real} \rightarrow \mathit{real}$.

A cleaner example of subtyping appears in Pascal subranges or the closely related range constraints of Ada (see [Hor84], for example). The Pascal subrange [1..10] containing the integers between 1 and 10 is subtype of the integers. If x is a variable of type $[1..10]$, and y of type integer, then we can assign y the value of x since every integer between 1 and 10 is an integer. More powerful examples of subtyping appear in typed object-oriented languages such as Eiffel [Mey92] and C++ [ES90]. In these languages, a class of objects, which may be regarded for the moment as a form of type, is placed in a subtype hierarchy. An object of a subtype may be used in place of one of any supertype, since the subtype relation guarantees that all required operations are implemented. Moreover, although the representation of objects of one type may differ from the representation of objects of a subtype, the representations are generally compatible in a way that eliminates the need for conversion from one to another.

2.4.2 Typed lambda calculus with subtyping

The notation $A <: B$ is commonly used to indicate that A is a subtype of B, since $<:$ provides a rough ASCII approximation of the subset symbol "$\subseteq$". We use $\lambda^{\rightarrow}_{<:}$ to denote the extension of $\lambda^{\rightarrow}$ with subtyping. Surprisingly, reduction for $\lambda^{\rightarrow}_{<:}$ is substantially more complicated than for $\lambda^{\rightarrow}$.

Like $\lambda^{\rightarrow}$, the definition of $\lambda^{\rightarrow}_{<:}$ terms depends on a signature, which now includes subtype assumptions about the type symbols. Formally, a $\lambda^{\rightarrow}_{<:}$ *signature* is a triple $\Sigma = \langle B, \mathit{Sub}, C \rangle$ with B a set of type constants, Sub a set of subtyping assertions $b <: b'$ between type constants $b, b' \in B$, and C a set of term constants, each with a specified type written using $\rightarrow$ and type constants from B.

The $\lambda^{\rightarrow}_{<:}$ type expressions over signature $\langle B, \mathit{Sub}, C \rangle$ are the same as the $\lambda^{\rightarrow}$ type expressions over signature $\langle B, C \rangle$. Note that we only consider subtype assumptions are between atomic type names. A consequence of this property is given in Lemma 2.4 below.

The distinguishing feature of $\lambda^{\rightarrow}_{<:}$ is the subtype relation, defined from the signature by the following axiom and inference rules.

(*ref* $<:$) $$\tau <: \tau$$

(*trans* $<:$) $$\frac{\rho <: \sigma, \ \ \sigma <: \tau}{\rho <: \tau}$$

($\rightarrow$ $<:$) $$\frac{\rho <: \tau, \ \ \tau' <: \rho'}{\tau \rightarrow \tau' <: \rho \rightarrow \rho'}$$

If we think of subtyping as an ordering, the last rule "says" that $\rightarrow$ is monotonic in its second argument, but antimonotonic in its first.

We write $\Sigma \vdash \sigma <: \tau$ if the subtype assertion $\sigma <: \tau$ is provable from assertions in *Sub* using the axiom and inference rules given above. It is easy to show that if $\Sigma \vdash \sigma <: \tau$, then the type expressions σ and τ must contain the same number and parenthesization of $\rightarrow$'s. To state this precisely, we let the *matching* relation on types be the least relation satisfying the following conditions:

$$\begin{array}{rll} b & \text{matches} & b' \quad \text{for any type constants } b, b' \\ \sigma_1 \rightarrow \sigma_2 & \text{matches} & \tau_1 \rightarrow \tau_2 \quad \text{whenever } \sigma_i \text{ matches } \tau_i \quad (i = 1, 2) \end{array}$$

Lemma 2.4 *For any $\lambda^{\rightarrow}_{<:}$ signature Σ, if $\Sigma \vdash \sigma <: \tau$ then σ matches τ.*

2.4.3 Terms

The terms of $\lambda^{\rightarrow}_{<:}$ are given by the same typing rules as in $\lambda^{\rightarrow}$, except for the additional rule:

$$(subsumption) \qquad \frac{\Gamma \triangleright M : \sigma, \ \Sigma \vdash \sigma <: \tau}{\Gamma \triangleright M : \tau}$$

In summary, the typing rules of $\lambda^{\rightarrow}_{<:}$ are (*cst*), (*var*), ($\rightarrow$ Intro), ($\rightarrow$ Elim), (*add var*) and (*subsumption*).

2.4.4 Equations

The equational proof system of $\lambda^{\rightarrow}_{<:}$ consists of exactly the same axioms and proof rules as $\lambda^{\rightarrow}$ without subtyping. Specifically, we have (*ref*), (*sym*) and (*trans*) making provable equality an equivalence relation, the technical rule (*add var*) for adding variables to the type assignment, axioms (α), (β) and (η) for lambda abstraction and application, and (ξ) and (ν) making provable equality a congruence relation.

There is a common typing confusion associated with the extensionality axiom, (η), that illustrates the importance of writing types as part of equations. The extensionality axiom has the form

$$\Gamma \triangleright \lambda x\colon \tau.(Mx) = M, \quad x \text{ not free in } M$$

but it is not clear that the two terms involved necessarily have the same type. For example, suppose M has the form $\lambda y\colon \tau'.N$ with $\tau <: \tau'$. Then M has type $\tau' \rightarrow \rho$ but $\lambda x\colon \tau.(Mx)$ does not. Thus, in writing $\lambda x\colon \tau.(Mx) = M$, it could appear that we are equating terms with different types. However, since $\tau' \rightarrow \rho <: \tau \rightarrow \rho$, both have type $\tau \rightarrow \rho$. Our axiom scheme

$$\Gamma \triangleright \lambda x\colon \tau.(Mx) = M : \tau \rightarrow \rho, \quad x \text{ not free in } M,$$

applies whenever $\Gamma \triangleright \lambda x\colon \tau.(Mx) : \tau \rightarrow \rho$ and $\Gamma \triangleright M : \tau \rightarrow \rho$ are both derivable, which will be the case whenever $\Gamma \triangleright M : \tau \rightarrow \rho$ is derivable.

A derived rule. A general principle about subtyping and equality is given by the inference rule

$$(subsumption\ eq) \qquad \frac{\Gamma \triangleright M = N : \tau, \ \Sigma \vdash \tau <: \rho}{\Gamma \triangleright M = N : \rho}$$

which we do *not* need to add to the proof system since it is derivable from reflexivity, application and (β). More specifically, if $\tau <: \rho$, then we have the typing

$$\Gamma \triangleright \lambda x\colon \tau.\, x : \tau \to \rho$$

and so by reflexivity, the equation

$$\Gamma \triangleright \lambda x\colon \tau.\, x = \lambda x\colon \tau.\, x : \tau \to \rho.$$

If $\Gamma \triangleright M = N : \tau$, then by applying the identity to each side, we may prove

$$\Gamma \triangleright (\lambda x\colon \tau.\, x)M = (\lambda x\colon \tau.\, x)N : \rho$$

which gives us the conclusion of $(\textit{subsumption eq})$ by (β) and transitivity.

Failure of Confluence For any $\lambda^{\to}_{<:}$ term $\Gamma \triangleright \lambda x\colon \sigma.\, M : \sigma \to \tau$, with $\rho <: \sigma$, we can prove the equation

$$\Gamma \triangleright \lambda x\colon \sigma.\, M = \lambda x\colon \rho.\, M : \rho \to \tau$$

One way to develop some intuition for this equation is to suppose that we give the lambda abstraction $\Gamma \triangleright \lambda x\colon \sigma.\, M : \sigma \to \tau$ type $\rho \to \tau$ by subsumption. If we then apply this function to an argument $\Gamma \triangleright N : \rho$ and β-reduce the resulting term,

$$(\lambda x\colon \sigma.\, M)N \to [N/x]M,$$

we will substitute a term of type ρ for the bound variable of type σ. This suggests that if we change the type of $\lambda x\colon \sigma.\, M$ by subsumption, then we have a term that is "functionally equivalent" to the term $\lambda x\colon \rho.\, M$ with the formal parameter x given type ρ instead of σ. In other words, subsumption effectively changes the types of lambda-bound variables.

We can prove the equation above by applying $\lambda x\colon \sigma.\, M$ to a free variable $x\colon \rho$ and then lambda-abstracting x. We begin with the typing derivation

$\Gamma \triangleright \lambda x\colon \sigma.\, M : \sigma \to \tau$	by assumption
$\Gamma, x\colon \rho \triangleright \lambda x\colon \sigma.\, M : \sigma \to \tau$	by $(\textit{add var})$
$\Gamma, x\colon \rho \triangleright x : \sigma$	by $(\textit{var}), (\textit{subsumption}), (\textit{add var})$
$\Gamma, x\colon \rho \triangleright (\lambda x\colon \sigma.\, M)x : \tau$	by application
$\Gamma \triangleright \lambda x\colon \rho.(\lambda x\colon \sigma.\, M)x : \rho \to \tau$	by $(\to$ Intro$)$

At this point, we apply a "trick," using (β) to show

$$\Gamma \triangleright \lambda x\colon \rho.(\lambda x\colon \sigma.\, M)x = \lambda x\colon \rho.\, M : \rho \to \tau$$

and (η) to show

$$\Gamma \triangleright \lambda x\colon \rho.(\lambda x\colon \sigma.\, M)x = \lambda x\colon \sigma.\, M : \rho \to \tau$$

giving us the desired equation by transitivity. Since both of these terms would be in normal form when M is a normal form, the last few steps also show that confluence fails for β, η-reduction on $\lambda^{\to}_{<:}$ terms. We return to confluence for $\lambda^{\to}_{<:}$ in Section 5.

3 Reduction properties of typed lambda terms

Confluence, normalization and other properties of typed reduction may be proved using logical relations. It will be useful to review a general theorem from [Mit96] that has these two properties as corollaries. The reason for considering the general theorem is that we are interested in both confluence (since labeled confluence implies unlabeled confluence) and termination (since this is used for completeness of left-most reduction). Since the proof of the main theorem in this section requires substantial machinery that is not related to labeling, we simply state the results in this section without proof.

A *property of typed lambda terms,* or *predicate on terms* is a set of typed lambda terms. If $\mathcal{S} = \{S^\sigma\}$ is a predicate, with S^σ the set of all terms of type σ in $\mathcal{S}$, then we write $\mathcal{S}(M)$ to indicate that $M \in \mathcal{S}$ and similarly $S^\sigma(M)$ to indicate that $M \in \mathcal{S}$ and $M\colon\sigma$.

A predicate $\mathcal{S}$ is *type-closed* if the following three conditions are satisfied.

(a) If $\mathcal{S}(M_1), \ldots, \mathcal{S}(M_k)$, then $\mathcal{S}(xM_1 \ldots M_k)$, where x is any variable of the appropriate type,

(b) If $S^\tau(Mx)$ holds for every variable x of type σ, then $S^{\sigma\to\tau}(M)$,

(c) If $S^\sigma(N)$ and $S^b(([N/x]M)N_1 \ldots N_k)$, then $S^b((\lambda x\colon\sigma.M)NN_1 \ldots N_k)$ for each type constant b.

In verifying that a particular predicate satisfies (b), we generally assume $S(Mx)$ for some x not free in M and show $S(M)$.

Theorem 3.1 *If $\mathcal{S}$ is any type-closed property of typed $\lambda^\to$ terms, then $\mathcal{S}(M)$ for every term $\Gamma \triangleright M : \sigma$.*

Strong normalization is the predicate SN defined by

$$SN(M) \text{ iff there is no infinite sequence of reductions from } M.$$

We may regard confluence as a predicate on typed $\lambda^\to$ terms by defining

$$CR(M) \text{ iff } \forall M_1, M_2[\ M \twoheadrightarrow M_1 \wedge M \twoheadrightarrow M_2 \supset \exists N.\ M_1 \twoheadrightarrow N \wedge M_2 \twoheadrightarrow N].$$

In verifying that CR is type-closed, the most difficult condition is (b). While it is easy to see that $SN(Mx) \supset SN(M)$, it is not immediate that $CR(Mx) \supset CR(M)$, since there may be reductions of Mx that do not apply to M.

Lemma 3.2 *The predicate CR is type-closed.*

Combining Lemma 3.2 with Theorem 3.1, we have confluence of β, η-reduction.

Theorem 3.3 (Confluence) *β, η-Reduction is confluent on typed lambda terms.*

3.1 Extending the type-closed method to constants

We prove properties of labeled and other reduction by extending the "type-closed" method to $\lambda^\to$ terms with constants. With constants that may have associated reduction rules, the proof that every term has a type-closed property $\mathcal{S}$ has four main parts:

1. Define a predicate $\mathcal{P}$ on the applicative structure of terms, from $\mathcal{S}$, by

$$\begin{array}{lll} P^b(M) & \text{iff} & \mathcal{S}(M) \\ P^{\sigma \to \tau}(M) & \text{iff} & \forall N.\, P^\sigma(N) \text{ implies } P^\tau(MN) \end{array}$$

2. Show that $P^\sigma(M)$ implies $\mathcal{S}(M)$.

2′. Show that $P^\sigma(c)$ for each term constant $c{:}\,\sigma$.

3. Show that $\mathcal{P}$ is admissible, a technical property defined in [Mit96]. This allows us to conclude that $P^\sigma(M)$ holds for every well-typed term $\Gamma \triangleright M : \sigma$.

Theorem 3.1 above is proved using steps 1, 2 and 3. If there are no reduction rules for constants, then we can treat constants just like variables. However, with reduction rules, condition (a) of the definition of type-closed will may fail for constant c in place of variable x. (For example, M may be strongly normalizing and $\mathit{fix}\, M$ not strongly normalizing.) Therefore, we must treat constants by some other means.

The main idea in extending Theorem 3.1 is that in treating constant c, we may need some properties of $\mathcal{P}$ in proving $P^\sigma(c)$. We therefore insert an extra step $(2')$ in the middle of the proof. (The arguments in steps 1, 2 and 3 are essentially the same with or without term constants.) Rather that try to identify a sufficient condition for step $2'$ to succeed, we simply state a theorem that says the method works when it works. We will use this for labeled fixed points and other operations.

Theorem 3.4 *Let $\mathcal{S}$ be a type-closed property of $\lambda^{\to}$ terms and let $\mathcal{P}$ be the predicate on terms defined from $\mathcal{S}$ as above. If $P(c_1), \ldots, P(c_k)$, for term constants $c_1, \ldots, c_k$, then $\mathcal{P}(M)$ and therefore $\mathcal{S}(M)$ holds for every well-typed $\lambda^{\to}$ term M over $c_1, \ldots, c_k$.*

4 Reduction with *fix* and additional operations

In this section, we analyze reduction for typed lambda calculus with *fix* and other operations. An important idea for handling *fix* is to label each occurrence of a fixed-point operator in a term with a bound on the number of reductions that may be applied. This produces a confluent and terminating reduction relation.

4.1 Labeled reduction

There are two versions of labeled reduction, one used to prove semantic results such as the Approximation Theorem and Computational Adequacy (see [Mit96]), and the other used to prove reduction properties. After defining both forms of reduction, we will focus on the simpler version used for reduction properties.

If we begin with simply-typed lambda calculus over signature Σ, containing fixed-point constants at some types, then the labeled terms over Σ are defined using the extended signature $\Sigma_{lab} \supseteq \Sigma$ obtained by adding a constant, fix_σ^n, for each type σ that has a fixed point operator in Σ and for each natural number $n \geq 0$. The constant fix_σ^n is called the *labeled fixed-point operator with label* n. For semantic results, it is common to also add a constant $\perp_\sigma$ for each type σ that has a fixed-point operator. We use the notation $\Sigma_{lab,\perp}$ for the signature $\Sigma_{lab,\perp} \supseteq \Sigma_{lab} \supseteq \Sigma$ with labeled fixed-point operators and constants $\perp_\sigma$ at appropriate types. We say M is a *labeled term over* Σ if M is a well-typed term over $\Sigma_{lab,\perp}$ and M does not contain any *fix* without a label. We write $lab(M)$ for the set of labeled terms that become syntactically identical to M when we replace each fix_σ^n by fix_σ.

The two forms of reduction for labeled fixed-point operators are:

$$(lab_+) \qquad \mathit{fix}_\sigma^{n+1} \to \lambda f\colon \sigma \to \sigma.\, f(\mathit{fix}_\sigma^n f)$$

$$(lab_0) \qquad \mathit{fix}_\sigma^{0} \to \lambda f\colon \sigma \to \sigma.\, \perp_\sigma$$

We will call reduction using the first reduction axiom *positive labeled reduction*, or simply *labeled reduction* for short, and refer to this using the symbol lab_+. The second reduction will be indicated by lab_0. Reduction with both reduction axioms will be denoted $lab_{+,0}$. We use $lab_{+(0)}$ for either lab_+ or $lab_{+,0}$. It is also possible to add reduction rules for $\perp$, such as $(\perp_{\sigma\to\tau} M) \to \perp_\tau$, but we will not consider this.

In semantic analysis, we may wish to treat fix_σ as one of its semantic approximants, $\lambda f\colon \sigma \to \sigma.\, f^n(\perp)$. This is precisely what happens if we label an occurrence of fix_σ with the number n and use $lab_{+,0}$ reduction. The problem with full $lab_{+,0}$ reduction, for the purpose of analyzing reduction without labels, is that the reduction of $\mathit{fix}_\sigma^0\, M$ to $(\lambda f\colon \sigma \to \sigma.\, \perp_\sigma)\, M$ allows us to discard the subterm M on subsequent β-reduction. This makes it impossible to use confluence of $\beta, lab_{+,0}$-reduction to prove confluence of β, fix-reduction, for example, in any direct way. Therefore, both forms reduction are useful. Since this paper is primarily concerned with reduction properties, we will be primarily concerned with lab_+.

We develop some general connections between labeled reduction and unlabeled reduction below, in a setting that allows for a set $\mathcal{R}$ of additional rewrite axioms. Like the common rewrite axioms for addition and conditional, for example, these must be *left-linear.* This means that if $L \to R$ is a reduction, then no term variable may occur more than once in the left-hand side, L. Just to clarify notation, we write $\mathcal{R}, \beta, lab_+$ for the combination of reduction given by a set $\mathcal{R}$ of reduction axioms, β-reduction and positive labeled reduction lab_+. Two basic and useful connections are given in the following lemmas.

Lemma 4.1 *Suppose $\mathcal{R}$ is left-linear and $M \xrightarrow{\mathcal{R},\beta,\mathit{fix}}\!\!\!\rightarrow N$. There is a natural number k such that if $M^{\#} \in \mathit{lab}(M)$, with each label in $M^{\#}$ at least k, then $M^{\#} \xrightarrow{\mathcal{R},\beta,\mathit{lab}_+}\!\!\!\rightarrow N^{\#}$ for some $N^{\#} \in \mathit{lab}(N)$.*

Lemma 4.2 *If $M^{\#} \xrightarrow{\mathcal{R},\beta,\mathit{lab}_+}\!\!\!\rightarrow N^{\#}$, where $M^{\#} \in \mathit{lab}(M)$ and $N^{\#} \in \mathit{lab}(N)$, then $M \xrightarrow{\mathcal{R},\beta,\mathit{fix}}\!\!\!\rightarrow N$.*

Both are proved by easy inductions on the length of reduction sequences. In Lemma 4.1, the number k is essentially the length of the reduction sequence from M to N, since any single reduction could be carried out by labeling any *fix* that is reduced with any number at least 1. The importance of assuming that $\mathcal{R}$ is left-linear is discussed below. Since $\mathit{lab}_{+,0}$ contains lab_+, the first lemma also holds for $\mathcal{R}, \beta, \mathit{proj}, \mathit{lab}_{+,0}$. However, Lemma 4.2 fails for $\mathit{lab}_{+,0}$, since the rule for fix^0 cannot be simulated using unlabeled *fix*. These two lemmas are used to prove the basic connection between confluence of labeled and unlabeled reduction.

Proposition 4.3 *If $\mathcal{R}, \beta, \mathit{lab}_+$-reduction is confluent on labeled terms, and $\mathcal{R}$ is left-linear, then $\mathcal{R}, \beta, \mathit{fix}$-reduction is confluent on unlabeled terms.*

Proof Suppose $N \xleftarrow{\mathcal{R},\beta,\mathit{fix}}\!\!\!\leftarrow M \xrightarrow{\mathcal{R},\beta,\mathit{fix}}\!\!\!\rightarrow P$. By Lemma 4.1, there are labeling $M^{\#}$, $N^{\#}$, and $P^{\#}$ of these terms (respectively) such that $N^{\#} \xleftarrow{\mathcal{R},\beta,\mathit{lab}_+}\!\!\!\leftarrow M^{\#} \xrightarrow{\mathcal{R},\beta,\mathit{lab}_+}\!\!\!\rightarrow P^{\#}$. Since $\mathcal{R}, \beta, \mathit{lab}_+$-reduction is confluent, $N^{\#}$ and $P^{\#}$ must have a common reduct $Q^{\#}$. By Lemma 4.2, we may erase labels to obtain $N \xrightarrow{\mathcal{R},\beta,\mathit{fix}}\!\!\!\rightarrow Q \xleftarrow{\mathcal{R},\beta,\mathit{fix}}\!\!\!\leftarrow P$. This shows that $\mathcal{R}, \beta, \mathit{fix}$-reduction is confluent. ∎

It is important to understand how left-linearity is used in the proof of Lemma 4.1. The critical step is the base case for a reduction from $\mathcal{R}$. The property that we may choose labels arbitrarily, as long as they are all big enough, may fail if we have a non-linear rule, since some reduction rule may be applicable only if two subterms have the same label. For example, consider the following algebraic rewrite rules for $\mathit{Eq?}\colon \mathit{nat} \to \mathit{nat} \to \mathit{bool}$, $\mathit{succ}\colon \mathit{nat} \to \mathit{nat}$ and $\mathit{true}, \mathit{false}\colon \mathit{nat}$.

$$\begin{array}{lcl} \mathit{Eq?}\, x\, x & \to & \mathit{true} \\ \mathit{Eq?}\, x\, (\mathit{succ}\, x) & \to & \mathit{false} \end{array}$$

Lemma 4.1 fails if $\mathcal{R}$ consists solely of either one of these rules, or both of them. These rules, together, also provide a counter-example to Proposition 4.3, if we drop the assumption that $\mathcal{R}$ is left-linear, since we may reduce $\mathit{Eq?}\,(\mathit{fix}\ \mathit{succ})\,(\mathit{fix}\ \mathit{succ})$ to both *true* and *false*. The first reduction is immediate, while the second proceeds by

$$\mathit{Eq?}\,(\mathit{fix}\ \mathit{succ})\,(\mathit{fix}\ \mathit{succ}) \twoheadrightarrow \mathit{Eq?}\,(\mathit{fix}\ \mathit{succ})\,(\mathit{succ}\,(\mathit{fix}\ \mathit{succ})).$$

If we label $\mathit{Eq?}\,(\mathit{fix}\ \mathit{succ})\,(\mathit{fix}\ \mathit{succ})$ in any way, then at most one of these reductions will be possible.

4.2 Termination and confluence of labeled reduction

As an application of the Theorem 3.4, we show that labeled reduction is strongly normalizing. Since Theorem 3.4 is not stated for a relation $\mathcal{R}$ satisfying a specific set of conditions, we will use PCF to illustrate the technique. We use *pcf* to indicate reduction using the axioms of PCF, and pcf_+ for PCF reduction with lab_+ replacing *fix*-reduction. It is a simple matter to extend the argument to show that $\mathit{pcf}_{+,0}$-reduction, with lab_0 added, is terminating.

We begin by defining the property we wish to establish, namely

$$\mathcal{S}(M) \ \text{ iff } \ \mathit{pcf}_+\text{-reduction is terminating from } M$$

where M may be any labeled PCF term, *i.e.*, any $\lambda^{\rightarrow}$ term over the signature type constants *nat* and *bool*, numerals, boolean constants *true* and *false*, addition, equality test *Eq?* on *nat*, conditional on each type, and labeled fixed point constants fix_σ^n for each type σ and all $n \geq 0$. Let $\mathcal{T}$ be the set of typed terms over this signature, constructed using infinitely many variables at each type.

Let $\mathcal{P} \subseteq \mathcal{T}$ be the typed predicate defined from $\mathcal{S}$ as above. We must show $\mathcal{P}(c)$ for each constant of the signature. The numerals and boolean constants are covered by the fact that $P^b(c)$ iff $\mathcal{S}(c)$, since all of these constants are normal forms. It remains to show that $\mathcal{P}$ holds for each of the remaining constants.

We will use the subsidiary notion of *elimination context,* given by the following grammar:

$$\mathcal{E}[\,] \quad ::= \quad [\,] \mid \mathcal{E}[\,]M$$

It is easy to see that for $\lambda^{\rightarrow}$, every elimination context has the form $\mathcal{E}[\,] \equiv [\,]M_1 \ldots M_k$ for some sequence of terms, with $\mathcal{E}[N] \equiv NM_1 \ldots M_k$. This form of context is often called an "applicative context" since it applies a term to a series of arguments. However, we use the phrase "elimination context" since the generalization to other types would involve the elimination term forms. For example, an elimination context for $\lambda^{\times,\rightarrow}$ would allow us to apply projection functions (but not to form pairs, since pairing is the introduction form for products). We say $\mathcal{E}[\,]$ is a σ, τ-elimination context if $\mathcal{E}[M]$ has type τ whenever M has type σ. We write $\mathcal{P}(\mathcal{E}[\,])$ if $\mathcal{E}[\,] \equiv [\,]M_1 \ldots M_k$ and $\mathcal{P}(M_i)$ holds for $1 \leq i \leq k$.

A useful fact to notice is that by Theorem 3.4, we have $\mathcal{P}(M)$ for every term without constants. A consequence is that for every type σ and every type constant $b \in \{int, bool\}$, there is a σ, b-elimination context $\mathcal{E}[\,]$ with $\mathcal{P}(\mathcal{E}[\,])$ and therefore $\mathcal{S}(\mathcal{E}[\,])$. This is easily verified by induction on types, using a variable wherever we need a term M with $P^\sigma(M)$.

The following lemma will be useful in reasoning about constants. The first two parts do not depend on the choice of $\mathcal{S}$, only the definition of $\mathcal{P}$ from $\mathcal{S}$. Part (iii) relies only on the fact that $\mathcal{S}$ is closed under reduction.

Lemma 4.4 *Let $M \in T^\sigma$ and let b be nat or bool.*
(i) *If $P^b(\mathcal{E}[M])$ for every σ, b-context $\mathcal{E}[\,]$ with $\mathcal{P}(\mathcal{E}[\,])$, then $P^\sigma(M)$.*
(ii) *If $P^\sigma(M)$ and $\mathcal{E}[\,]$ is a σ, b-context with $\mathcal{P}(\mathcal{E}[\,])$, then $P^b(\mathcal{E}[M])$.*
(iii) *If $P^\sigma(M)$ and $M \xrightarrow{pcf_+}\!\!\!\!\rightarrow N$, then $P^\sigma(N)$.*

Proof We prove (i) by induction on the type of M. If $M\!:b$, then the only b, b-elimination context is $\mathcal{E}[\,] = [\,]$ and the lemma clearly holds. If $M\!:\sigma \rightarrow \tau$, then every relevant $\sigma \rightarrow \tau, b$-elimination context has the form $\mathcal{E}[[\,]N_0]$, where $P^\sigma(N_0)$ and $\mathcal{E}[\,]$ is a τ, b-elimination context with $\mathcal{P}(\mathcal{E}[\,])$. Let $N \in T^\sigma$ be any term with $P^\sigma(N)$, and consider any τ, b-elimination context $\mathcal{E}[\,]$ with $\mathcal{P}(\mathcal{E}[\,])$. By the inductive hypothesis for $MN \in T^\tau$, we have $P^\tau(MN)$. Since N was chosen arbitrarily, if follows that $P^{\sigma \rightarrow \tau}(M)$. The proof for term $M\!:\sigma \times \tau$ is similar. This proves (i). Part (ii) may be proved by an easy induction on types.

We prove part (iii) from (i) and (ii) using the fact that if $\mathcal{S}(M)$ and $M \xrightarrow{pcf_+}\!\!\!\!\rightarrow N$, then $\mathcal{S}(N)$. More specifically, suppose $P^\sigma(M)$ and let $\mathcal{E}[\,]$ be any σ, b-elimination context with $\mathcal{P}(\mathcal{E}[\,])$. By (ii), we have $P^b(\mathcal{E}[M])$. If $M \xrightarrow{pcf_+}\!\!\!\!\rightarrow N$, then $\mathcal{E}[M] \xrightarrow{pcf_+}\!\!\!\!\rightarrow \mathcal{E}[N]$. Since $P^b(Q)$ iff $Q \in \mathcal{S}$, we therefore have $P^b(\mathcal{E}[N])$. Since this is true for every σ, b-context $\mathcal{E}[\,]$ with $\mathcal{P}(\mathcal{E}[\,])$, we have $P^\sigma(N)$ by (i). This proves the lemma. ∎

Lemma 4.5 *For $\mathcal{P}$ defined from strong normalization of pcf_+-reduction, we have $P(+)$, $P(Eq?)$, $P(\texttt{if \ then \ else \ })$, and $P(fix_\sigma^n)$ for each type σ and natural number $n \geq 0$.*

Proof For addition, we assume $P^{nat}(M)$ and $P^{nat}(N)$ and show $P^{nat}(M+N)$. It suffices to show that $M+N$ is strongly normalizing. However, this follows immediately since M and N are both strongly normalizing and the reduction axioms for $+$ only produce a numeral from two numerals. The proof for equality test, $Eq?$, is essentially the same.

For conditional, we assume $P^{bool}(B)$, $P^\sigma(M)$, $P^\sigma(N)$ and demonstrate $P^\sigma(\texttt{if } B \texttt{ then } M \texttt{ else } N)$. By Lemma 4.4, it suffices to show that if $\mathcal{E}[\texttt{if } B \texttt{ then } M \texttt{ else } N] \in T^b$, for $\mathcal{P}(\mathcal{E}[\,])$ and $b \equiv nat$ or $b \equiv bool$, then $\mathcal{E}[\texttt{if } B \texttt{ then } M \texttt{ else } N]$ is strongly normalizing. However, this is an easy case analysis, according to whether $B \xrightarrow{pcf_+}\!\!\!\!\rightarrow true$, $B \xrightarrow{pcf_+}\!\!\!\!\rightarrow false$, or neither, using (ii) of Lemma 4.4.

The remaining case is a labeled fixed-point constant fix_σ^n. For this, we use induction on the label. The base case, for fix_σ^0, is easy since there is no associated reduction. For fix_σ^{n+1}, we assume $P^{\sigma\to\sigma}(M)$ and let $\mathcal{E}[\,]$ be any elimination context with $\mathcal{P}(\mathcal{E}[\,])$ and $\mathcal{E}[fix_\sigma^{n+1}M] \in T^b$. We prove the lemma by showing that $\mathcal{E}[fix_\sigma^{n+1}M]$ is strongly normalizing. If there is an infinite reduction sequence from this term, it must begin with steps of the form

$$\begin{aligned} &\mathcal{E}[fix_\sigma^{n+1}M] \xrightarrow{pcf_+}\!\!\!\!\rightarrow \mathcal{E}'[fix_\sigma^{n+1}M'] \xrightarrow{lab_+} \mathcal{E}'[(\lambda f\colon\sigma\to\sigma.\, f(fix_\sigma^n f))M'] \\ &\xrightarrow{pcf_+}\!\!\!\!\rightarrow \mathcal{E}''[(\lambda f\colon\sigma\to\sigma.\, f(fix_\sigma^n f))M''] \xrightarrow{\beta} \mathcal{E}''[M''(fix_\sigma^n M'')] \xrightarrow{pcf_+}\!\!\!\!\rightarrow \quad \ldots \end{aligned}$$

However, by the induction hypothesis and (ii) of Lemma 4.4, we have $P^{\sigma\to\sigma}(M'')$, $P^\sigma(fix_\sigma^n M'')$ and $\mathcal{P}(\mathcal{E}''[\,])$, which implies that an infinite reduction sequence of this form is impossible. This completes the induction step and proves the lemma. ∎

Since Lemma 4.5 is easily extended to $pcf_{+,0}$, we have the following corollary of Theorem 3.4.

Theorem 4.6 *Both pcf_+ and $pcf_{+,0}$-reduction are strongly normalizing on labeled PCF terms.*

We also wish to show that both forms of labeled PCF reduction are confluent. A general theorem that could be applied, if we did not have conditional at all types, is given in [BT88, BTG89]. The general theorem is that if typed lambda calculus is extended with a confluent, algebraic rewrite system, the result is confluent. The proof of this is nontrivial, but is substantially simplified if we assume that $\mathcal{R}$ is left-linear. That is in fact the case we are interested in, since left-linearity is essential to Proposition 4.3 anyway. An adaptation of this proof to typed lambda calculus with labeled fixed-point operators has been carried out in [HM90, How92]. For the purposes of proving our desired results as simply as possible, however, it suffices to prove weak confluence, since confluence then follows by Newman's Lemma [Bar84, Mit96]. To emphasize the fact that the argument does not depend heavily on the exact natural number and boolean operations of PCF, we show that adding $\beta, lab_{+(0)}$ to any weakly confluent reduction system preserves weak confluence, as long as there are no symbols in common between $\mathcal{R}$ and the reduction axioms of $\beta, lab_{+(0)}$. This is essentially an instance of *orthogonal rewrite systems,* as they are called in the literature on algebraic rewrite systems [Toy87], except that application is shared. A related confluence property of untyped lambda calculus is Mitschke's δ-reduction theorem [Bar84], which shows that under certain syntactic conditions on $\mathcal{R}$, untyped $\mathcal{R},\beta$-reduction is confluent.

Lemma 4.7 *Let $\mathcal{R}$ be a set of reduction axioms of the form $L \to R$, where L and R are lambda terms of the same type and L does not contain an application xM of a variable to an argument or any of the symbols λ, fix or any fix_σ^n. If $\mathcal{R}$ is weakly confluent, then $\mathcal{R}, \beta, lab_{+(0)}$ is weakly confluent.*

Proof The proof is a case analysis on pairs of redexes. We show two cases involving β-reduction and reduction from $\mathcal{R}$. The significance of the assumptions on the form of L is that these guarantee that if a subterm of a substitution instance $[M_1, \ldots, M_k/x_1, \ldots, x_k]L$ of some left-hand-side contains a $\beta, lab_{+(0)}$ redex, then this redex must be entirely within one of $M_1, \ldots, M_k$. The lemma would also hold with any other hypotheses guaranteeing this property.

The first case is an $\mathcal{R}$-redex inside a β-redex. This gives a term of the form

$$(\lambda y\colon \sigma.(\ldots [M_1, \ldots, M_k/x_1, \ldots, x_k]L \ldots))N$$

reducing to either of the terms

$$(\ldots [N/y][M_1, \ldots, M_k/x_1, \ldots, x_k]L \ldots) \qquad (\lambda y\colon \sigma.(\ldots [M_1, \ldots, M_k/x_1, \ldots, x_k]R \ldots))N$$

the first by β-reduction and the second by an axiom $L \to R$ from $\mathcal{R}$. It is easy to see that both reduce in one step to

$$(\ldots [N/y][M_1, \ldots, M_k/x_1, \ldots, x_k]R \ldots)$$

The other possible interaction between β-reduction and $\mathcal{R}$ begins with a term of the form

$$(\ldots [M_1, \ldots, M_k/x_1, \ldots, x_k]L \ldots)$$

where one of $M_1, \ldots, M_k$ contains a β-redex. If $M_i \to M_i'$, then this term reduces to either of the two terms

$$(\ldots [M_1, \ldots, M_i', \ldots, M_k/x_1, \ldots, x_k]L \ldots) \qquad (\ldots [M_1, \ldots, M_i, \ldots, M_k/x_1, \ldots, x_k]R \ldots)$$

in one step. We can easily reduce the first to

$$(\ldots [M_1, \ldots, M_i', \ldots, M_k/x_1, \ldots, x_k]R \ldots)$$

by one $\mathcal{R}$ step. Since we can also reach this term by some number of β-reductions, one for each occurrence of x_i in R, local confluence holds. The cases for other reductions in a substitution instance of the left-hand-side of an $\mathcal{R}$ axiom are very similar. ∎

By Lemma Newman's Lemma [Bar84, Mit96], it follows that pcf_+ and $pcf_{+,0}$ are confluent. We may use Proposition 4.3 to derive confluence of pcf-reduction from confluence of pcf_+. This gives us the following theorem.

Theorem 4.8 *The reductions pcf_+ and $pcf_{+,0}$ are confluent on labeled PCF terms and pcf-reduction is confluent on ordinary (unlabeled) PCF terms.*

4.3 Completeness of leftmost reduction

The final result in this section is that if $\mathcal{R}$ is a set of "left-normal" rules (defined below), and $\mathcal{R}, \beta, lab_+$ is confluent and terminating, then the leftmost reduction strategy is complete for finding $\mathcal{R}, \beta, \mathit{fix}$-normal forms. Since the PCF rules are left-normal, left-most reduction is complete for finding normal forms of PCF programs.

A reduction axiom is *left-normal* if all the variables in the left-hand side of the rule appear to the right of all the term constants. For example, all of the PCF rules for natural numbers and booleans are left-normal. If we permute the arguments of conditional, however, putting the boolean argument at the end, then we would have reduction axioms

$$\begin{array}{rcl} \mathit{cond}\ x\, y\, \mathit{true} & \longrightarrow & x, \\ \mathit{cond}\ x\, y\, \mathit{false} & \longrightarrow & y. \end{array}$$

These are not left-normal since x and y appear to the left of the constants *true* and *false*. Intuitively, if we have left-normal rules, then may safely evaluate the arguments of a function from left to right. In many cases, it is possible to replace a set of rules with essentially equivalent left-normal ones by either permuting arguments or introducing auxiliary function symbols (see [Mit96]). However, this is not possible for inherently "non-sequential" functions such as parallel-or.

We will prove the completeness of left-most reduction using strong normalization and confluence of positive labeled reduction, and the correspondence between labeled and unlabeled reduction. Since labeled *fix* reduction may terminate with fix^0 where unlabeled *fix* reduction could continue, our main lemma is that any fix^0 occurring to the left of the left-most redex will remain after reducing the left-most redex.

Lemma 4.9 *Let $\mathcal{R}$ be a set of left-normal reduction axioms with no left-hand-side containing fix^0. If $M \xrightarrow{\mathcal{R},\beta,lab_+} N$ by the left-most reduction step, and fix^0 occurs to the left of the left-most redex of M, then fix^0 must also occur in N, to the left of the left-most redex if N is not in normal form.*

Proof Suppose $M \equiv \mathcal{C}[L] \to \mathcal{C}[R] \equiv N$ by contracting the left-most $\mathcal{R},\beta,lab_+$-redex and assume fix^0 occurs in context $\mathcal{C}[\ ]$ to the left of $[\ ]$. If $N \equiv \mathcal{C}[R]$ is in normal form, then fix^0 occurs in N since fix^0 occurs in $\mathcal{C}[\ ]$. We therefore assume N is not in normal form and show that fix^0 occurs to the left of the left-most redex in N.

Suppose, for the sake of deriving a contradiction, that fix^0 occurs within, or to the right of, the left-most redex of N. Since a term beginning with fix^0 is not a redex, the left-most redex of N must begin with some symbol to the left of fix^0, which is to the left of R when we write N as $\mathcal{C}[R]$. The proof proceeds by considering each possible form of redex.

Suppose the left-most redex in N is $(\lambda x{:}\,\sigma.\, N_1)N_2$, with fix^0 and therefore R occurring to the right of this λ. Since fix^0 must occur between λ and R, R cannot be $(\lambda x{:}\,\sigma.\, N_1)N_2$, and so a redex of the form $(\lambda x{:}\,\sigma.\, N_1')N_2'$ must occur in M to the left of L. This contradicts the assumption that L is the leftmost redex in M. The analogous case for *fix* is straightforward.

It remains to consider a redex SL' with S some substitution and $L' \to R'$ in $\mathcal{R}$. We assume fix^0 and therefore R occur to the right of the first symbol of SL'. Since fix^0 is not in L', by hypothesis, and the rule is left-normal, all symbols to the right of fix^0, including R if it occurs within L', must be the result of substituting terms for variables in L'. It follows that we have a redex in M to the left of L, again a contradiction. This proves the lemma. ∎

Theorem 4.10 *Suppose $\mathcal{R},\beta,lab_+$ is confluent and terminating, with $\mathcal{R}$ left-linear and left-normal. If $M \overset{\mathcal{R},\beta,\mathit{fix}}{\twoheadrightarrow} N$ and N is a normal form, then there is a $\mathcal{R},\beta,\mathit{fix}$-reduction from M to N that contracts the leftmost redex at each step.*

Proof If $M \overset{\mathcal{R},\beta,\mathit{fix}}{\twoheadrightarrow} N$ then, by Lemma 4.1 there exist labelings $M^\#$ and $N^\#$ of these terms such that $M^\# \overset{\mathcal{R},\beta,lab_+}{\twoheadrightarrow} N^\#$. Since $\mathcal{R},\beta,lab_+$-reduction is confluent and terminating, we may reduce $M^\#$ to $N^\#$ by reducing the left-most redex at each step.

We show that the left-most $\mathcal{R},\beta,lab_+$-reduction of $M^\#$ to $N^\#$ is also the left-most $\mathcal{R},\beta,\mathit{fix}$-reduction of M to N (when labels are removed). It is easy to see that this is the case if no term in the reduction sequence has fix^0 to the left of the left-most redex, since this is the only term that would be an unlabeled redex without being a labeled one. Therefore, we assume that the final k steps of the reduction have the form

$$M_k^\# \equiv \mathcal{C}[L] \to \mathcal{C}[R] \equiv M_{k-1}^\# \to M_{k-2}^\# \to \ldots \to N^\#,$$

where fix^0 occurs to the left of L in M_k. But by Lemma 4.9 and induction on k, fix^0 must also be present in $N^{\#}$, which contradicts the fact that N is a $\mathcal{R}, \beta, \mathit{fix}$-normal form. It follows from Lemma 4.1 that by erasing labels we obtain a leftmost reduction from M to N. ∎

A related theorem in [Klo80] shows that left-most reduction is normalizing for the untyped lambda calculus extended with any left-normal, linear and non-overlapping term rewriting system. Our proof is simpler since we assume termination, and since type restrictions make *fix* the only source of potential nontermination.

5 Confluence for lambda calculi with subtyping

5.1 Failures of confluence with subtyping

As noted in Section 2.4, β, η reduction fails to be confluent in the presence of subtyping. Specifically, if $\sigma <: \tau$ and $M \equiv \lambda y : \sigma . (\lambda x : \tau . N)\, y$, then

$$M \twoheadrightarrow \lambda x : \sigma . N$$

by β-reduction and

$$M \twoheadrightarrow \lambda x : \tau . N$$

by η-reduction. If N is a closed normal form, then both $\lambda x : \sigma . N$ and $\lambda x : \tau . N$ will be distinct normal forms. Since these two distinct normal forms differ only in the type annotations of λ-bound variables, one possible solution could be to add a reduction rule that allows us to change the type-tags of variables.

Based on this example, we consider the following reduction:

$$(\textit{subtype}) \qquad\qquad \lambda x : \tau . M \rightarrow \lambda x : \sigma . M \qquad \text{if } \sigma <: \tau$$

Although (*subtype*) restores confluence for the counter-example considered above, confluence still fails. Specifically, Let N be a normal-form term and consider any types σ_1, σ_2, τ such that $\sigma_1 <: \tau$ and $\sigma_2 <: \tau$ and there is no type ρ such that $\rho <: \sigma_1$, $\rho <: \sigma_2$. Then, by (*subtype*) $\lambda x : \tau . N$ can be reduced to both $\lambda x : \sigma_1 . N$ and $\lambda x : \sigma_2 . N$ but there is no common term to which the latter two terms can be reduced to.

5.2 Reduction System for $\lambda^{\rightarrow}_{<:}$

To understand our formulation of a confluent reduction system in the presence of subtyping, it is useful to recall the connection between the equational theory of a calculus and its reduction system, stated as Corollary 2.3. Specifically, since we take equality as given by the proof system and formulate a confluent reduction system so that convertibility will coincide with provable equality. From this point of view, it is appropriate to look to provable equality for inspiration.

In the presence of subtyping, there are are interesting subtleties in the way one needs to think about equality. In $\lambda^{\rightarrow}$ without subtyping, we include the context Γ and type σ in equations $\Gamma \triangleright M = N : \sigma$ simply to indicate the common typing of both terms and to facilitate writing "well-typed" inference rules for axiomatizing the true equations. Since the $\lambda^{\rightarrow}$ type σ is uniquely determined from Γ and M or N, the type does not have much to do with *how* M and N might be equal. With subtyping, however, terms M and N may have many different types (under the same assumptions about free variables) and so it is *a priori* possible for them to be equal at one type but different at another type.

This dependence of equality on the context and type implies that reduction would also need to be dependent on the context and type if we there is to be the natural connection between equations and reduction. With this in mind, we formulate reduction for $\lambda^{\rightarrow}_{<:}$ in the form $\Gamma \triangleright M \xrightarrow{\sigma} N$ with the dependence of the reduction on the context Γ and the type σ at which M and N are being considered made explicitly. It would be possible in our reduction system that $\Gamma \triangleright M \xrightarrow{\sigma} N$ but that $\Gamma \triangleright M \stackrel{\tau}{\not\rightarrow} N$ for a different type τ, *i.e.*, redexes strongly depend on the type. Intuitively, we take as basic redexes β, η and the rule (*subtype*) with the restriction that $\Gamma \triangleright M \xrightarrow{\sigma} N$ only if both M and N have type σ in the context Γ. The individual reduction relations $\Gamma \triangleright \cdot \xrightarrow{\sigma} \cdot$ for each context Γ and type σ can then be shown to be confluent. in particular, suppose $\sigma_1 <: \tau$ and $\sigma_2 <: \tau$ and there is no type ρ such that $\rho <: \sigma_1$, $\rho <: \sigma_2$, as above. If N has type δ, we would have

$$\lambda x\colon \tau \,.\, N \xrightarrow{\sigma_1 \to \delta} \lambda x\colon \sigma_1 \,.\, N$$

and

$$\lambda x\colon \tau \,.\, N \xrightarrow{\sigma_2 \to \delta} \lambda x\colon \sigma_2 \,.\, N$$

but not $\lambda x\colon \tau \,.\, N \xrightarrow{\sigma_1 \to \delta} \lambda x\colon \sigma_2 \,.\, N$, thus blocking the failure of confluence with the untyped version of $(subtype)$ above.

However, defining the reduction rules to depend on the type σ leads to a problem in applying the reduction rules to subterms that are redexes, since it is not clear at what type to consider the redexes. In more detail, consider a term $N \equiv (\cdots M \cdots)$ with a subterm M, and suppose that we wish to reduce N at type σ. The type σ of the whole term does not determine the type at which reductions on the subterm M can be performed and one therefore needs to specify which *typed* redexes can be simplified inside terms. Our solution is that we can consider M to be a redex of type τ, if the whole term $(\cdots M \cdots)$ can be given type σ assuming that the subterm M is of type τ. Thus, the type at which we can apply the reductions for M are not all types of M but only those with which we can give the enclosing term the indicated type. These ideas are formalized in a proof system for $\Gamma \triangleright M \xrightarrow{\sigma} N$ defined below.

The reduction relation $\Gamma \triangleright M \xrightarrow{\sigma} N$, where Γ is a context and σ a type, is given by the following rules.

$$(\beta) \qquad \frac{\Gamma \triangleright (\lambda x\colon \sigma \,.\, M)\, N : \tau}{\Gamma \triangleright (\lambda x\colon \sigma \,.\, M)\, N \xrightarrow{\tau} [N/x]\, M}$$

$$(\eta) \qquad \frac{\Gamma \triangleright M : \sigma \to \tau \qquad x \notin FV(M)}{\Gamma \triangleright (\lambda x\colon \sigma \,.\, M\, x) \xrightarrow{\sigma \to \tau} M}$$

$$(<:) \qquad \frac{\Gamma \triangleright \lambda x\colon \tau \,.\, M : \sigma \to \delta \qquad \sigma <: \tau,\ \sigma \not\equiv \tau}{\Gamma \triangleright (\lambda x\colon \tau \,.\, M) \xrightarrow{\sigma \to \delta} (\lambda x\colon \sigma \,.\, M)}$$

$$(app1) \qquad \frac{\Gamma \triangleright M \xrightarrow{\sigma \to \tau} N \qquad \Gamma \triangleright P : \sigma}{\Gamma \triangleright M\, P \xrightarrow{\tau} N\, P}$$

$$(app2) \qquad \frac{\Gamma \triangleright M : \sigma \to \tau \qquad \Gamma \triangleright P_1 \xrightarrow{\sigma} P_2}{\Gamma \triangleright M\, P_1 \xrightarrow{\tau} M\, P_2}$$

$$(abs) \qquad \frac{\Gamma, x\colon \sigma \triangleright M \xrightarrow{\tau} N \qquad \sigma_1 <: \sigma}{\Gamma \triangleright \lambda x\colon \sigma \,.\, M \xrightarrow{\sigma_1 \to \tau} \lambda x\colon \sigma \,.\, N}$$

5.3 Confluence of Reduction System

Confluence of $\lambda^{\to}_{<:}$ is proved using confluence of β,η-reduction for untyped λ-calculus. In broad outline, we consider the untyped term resulting from erasing the types of λ-bound variables in $\lambda^{\to}_{<:}$ terms and use the common reduct given by confluence of untyped λ-calculus to extract the common reduct for the reduction in $\lambda^{\to}_{<:}$.

The *type erasure,* $Erase(M)$, of a term, M, is defined by

$$\begin{array}{rcl} Erase(x) & = & x \\ Erase(M\,N) & = & Erase(M)\,Erase(N) \\ Erase(\lambda x\colon\sigma\,.\,M) & = & \lambda x.\,Erase(M) \end{array}$$

We can also reconstruct a typed $\lambda^{\rightarrow}_{<:}$ term from the erasure of any $\lambda^{\rightarrow}_{<:}$ term. We may identify such erasures using the following inference system for assigning types to untyped terms. (var) $\Gamma \triangleright x : \sigma$ if $x\colon\sigma \in \Gamma$

$(addvar)$ $$\frac{\Gamma \triangleright U : \tau}{\Gamma, x\colon\sigma \triangleright U : \tau} \qquad \text{if } x \notin Dom(\Gamma)$$

$(\rightarrow Intro)$ $$\frac{\Gamma, x\colon\sigma \triangleright U : \tau}{\Gamma \triangleright \lambda x.\,U : \sigma \rightarrow \tau}$$

$(\rightarrow Elim)$ $$\frac{\Gamma \triangleright U : \sigma \rightarrow \tau \qquad \Gamma \triangleright V : \sigma}{\Gamma \triangleright U\,V : \tau}$$

$(subsump)$ $$\frac{\Gamma \triangleright U : \sigma \qquad \Sigma \vdash \sigma <: \tau}{\Gamma \triangleright U : \tau}$$

With these definitions, we can now state and prove the various lemmas needed to establish the confluence of the reduction system for $\lambda^{\rightarrow}_{<:}$.

Our first lemma shows that the type-erasure of a term of $\lambda^{\rightarrow}_{<:}$ can be given the same type in the type system for untyped λ-terms.

Lemma 5.1 *If* $\Gamma \triangleright M : \sigma$ *then* $\Gamma \triangleright Erase(M) : \sigma$.

Proof By induction on the proof of $\Gamma \triangleright M : \sigma$. ∎

The next lemma states that the typed reduction rules can be mimicked as β,η-reductions on the untyped terms resulting from their type-erasure.

Lemma 5.2 *If* $\Gamma \triangleright M \xrightarrow{\sigma} N$ *then* $Erase(M) \xrightarrow{\beta,\eta}\!\!\!\!\rightarrow Erase(N)$

Proof By induction on the proof of $\Gamma \triangleright M \xrightarrow{\sigma} N$. ∎

We next establish the converse, *i.e.*, that β,η-reductions on untyped terms can be mimicked as typed reductions on the corresponding terms. To prove this, we need two technical propositions stating some obvious properties of substitutions.

Proposition 5.3 *If* $\Gamma, x\colon\sigma \triangleright M : \tau$ *and* $\Gamma \triangleright N : \sigma$ *then* $\Gamma \triangleright [N/x]M : \tau$

Proof By induction on the proof of $\Gamma, x\colon\sigma \triangleright M : \tau$. ∎

Proposition 5.4 $Erase([N/x]\,M) \equiv [Erase(N)/x]Erase(M)$.

Proof By induction on the structure of M. ∎

Lemma 5.5 *If* $\Gamma \triangleright M : \sigma$ *and* $Erase(M) \xrightarrow{\beta,\eta} U$ *then* $\exists N$ *such that* $\Gamma \triangleright N : \sigma$, $\Gamma \triangleright M \xrightarrow{\sigma} N$, *and* $Erase(N) \equiv U$.

Proof By induction on the structure of M.

Case $M \equiv x$**:** Vacuously true, since $Erase(M) = x$ and $x \stackrel{\beta,\eta}{\not\longrightarrow} U$, for any term U.

Case $M \equiv M_1\, M_2$**:** Since $\Gamma \triangleright M : \sigma$, we have that

$$\begin{array}{ll} \Gamma \triangleright M_1 : \tau \rightarrow \rho & \text{where } \rho <: \sigma \\ \Gamma \triangleright M_2 : \tau & \end{array} \tag{1}$$

Hence $\tau \rightarrow \rho <: \rho \rightarrow \sigma$ and by $(subsump)$,

$$\Gamma \triangleright M_1 : \tau \rightarrow \sigma \tag{2}$$

Suppose that $Erase(M) \xrightarrow{\beta,\eta} U$. Since $Erase(M) = Erase(M_1)\, Erase(M_2)$, the reduction to U could be because $Erase(M)$ itself is a β-redex, or we perform a β,η-reduction inside $Erase(M_1)$ or $Erase(M_2)$. We consider each of these cases separately.

Case I: Suppose that $Erase(M_1) \equiv \lambda x.\, V$ and $U \equiv [Erase(M_2)/x]V$. Since $Erase(M_1) \equiv \lambda x.\, V$, it follows that M_1 is of the form $\lambda x\colon \tau'.\, M_1'$ with $Erase(M_1') = V$. From (2), it follows that

$$\Gamma, x : \tau' \triangleright M_1' : \sigma \tag{3}$$

with $\tau <: \tau'$. Using (1) with $\tau <: \tau'$, we get that

$$\Gamma \triangleright M_2 : \tau' \tag{4}$$

Take N to be the term $[M_2/x]M_1'$. From judgments (3), (4), using Proposition 5.3, it follows that $\Gamma \triangleright N : \sigma$. Since $M \equiv (\lambda x\colon \tau'.\, M_1')\, M_2$, we can use (β) to get that $\Gamma \triangleright M \xrightarrow{\sigma} N$. Finally,

$$\begin{array}{rcl} Erase(N) & \equiv & Erase([M_2/x]M_1') \\ & \equiv & [Erase(M_2)/x]Erase(M_1') \end{array}$$

by Proposition 5.4, and we thus get that $Erase(N) \equiv U$.

Case II: Suppose that $Erase(M_1) \xrightarrow{\beta,\eta} V_1'$ and $U \equiv V_1'\, Erase(M_2)$. By (2), we can use induction hypothesis on M_1 to get that there is a term N' such that $\Gamma \triangleright N' : \tau \rightarrow \sigma$, $\Gamma \triangleright M_1 \xrightarrow{\tau \rightarrow \sigma} N'$ and $Erase(N') \equiv V_1'$.

Take $N \equiv N'\, M_2$. From (1), we get that $\Gamma \triangleright N : \sigma$. Using *(app1)*, we get that $\Gamma \triangleright M \xrightarrow{\sigma} N$. Finally,

$$\begin{array}{rcl} Erase(N) & \equiv & Erase(N')\, Erase(M_2) \\ & \equiv & V_1'\, Erase(M_2) \\ & \equiv & U \end{array}$$

Case III: The only possibility remaining is that $Erase(M_2) \xrightarrow{\beta,\eta} V_2'$ and $U \equiv Erase(M_1)\, V_2'$. This case is dealt similarly as Case II.

Case $M \equiv \lambda x : \sigma_1 \,.\, M_1$**:** Since $\Gamma \rhd M : \sigma$, we have that $\sigma \equiv \sigma_2 \to \tau$, and

$$\Gamma, x : \sigma_1 \rhd M_1 : \tau' \quad \text{where } \sigma_2 <: \sigma_1 \text{ and } \tau' <: \tau, \tag{5}$$

By $(subsump)$,

$$\Gamma, x : \sigma_1 \rhd M_1 : \tau \tag{6}$$

Suppose that $Erase(M) \xrightarrow{\beta,\eta} U$. Since $Erase(M) = \lambda x.\, Erase(M_1)$, the reduction to U could be because $Erase(M)$ itself is a η-redex, or we perform a β,η-reduction inside $Erase(M_1)$. We consider each of these cases separately.

Case I: Suppose $Erase(M_1) \equiv V\,x$ for some term V,
then $Erase(M) \equiv \lambda x.\, V\,x$ and $Erase(M) \xrightarrow{\eta} V$, *i.e.*, $U \equiv V$.
Since $Erase(M_1) \equiv V\,x$, we have that $M_1 \equiv M_1'\,x$ where $Erase(M_1') \equiv V$. By judgment (6), $\Gamma, x : \sigma_1 \rhd M_1'\,x : \tau$, *i.e.*, $\Gamma \rhd M_1' : \sigma_1' \to \tau'$ where
$\sigma_1 <: \sigma_1'$ and $\tau' <: \tau$. By (5), $\sigma_2 <: \sigma_1$, hence $\Gamma \rhd M_1' : \sigma_2 \to \tau$, *i.e.*,
$\Gamma \rhd M_1' : \sigma$.
Take $N \equiv M_1'$, we have that $\Gamma \rhd N : \sigma$.
Since $\Gamma \rhd M : \sigma$ where $\sigma \equiv \sigma_2 \to \tau$ and $M \equiv \lambda x : \sigma_1 \,.\, M_1'\,x$, we have the type judgment $\Gamma \rhd \lambda x : \sigma_1 \,.\, M_1'\,x : \sigma_2 \to \tau$.
By $(<:)$, and (5), $\Gamma \rhd \lambda x : \sigma_1 \,.\, M_1'\,x \xrightarrow{\sigma_2 \to \tau} \lambda x : \sigma_2 \,.\, M_1'\,x$.
Hence, by (η), $\Gamma \rhd \lambda x : \sigma_2 \,.\, M_1'\,x \xrightarrow{\sigma_2 \to \tau} M_1'$, *i.e.*, $\Gamma \rhd M \xrightarrow{\sigma} N$.
Finally,

$$\begin{array}{rcl} Erase(N) & \equiv & Erase(M_1') \\ & \equiv & V \\ & \equiv & U \end{array}$$

Case II: Suppose that $Erase(M_1) \xrightarrow{\beta,\eta} V$. We have $U \equiv \lambda x.\, V$. By (6), we can use the induction hypothesis on M_1 to get that there is a term N_1 such that $\Gamma, x : \sigma_1 \rhd N_1 : \tau, \Gamma, x : \sigma_1 \rhd M_1 \xrightarrow{\tau} N_1$, and $Erase(N_1) \equiv V$.
Take $N \equiv \lambda x : \sigma_1 \,.\, N_1$.
Using (abs) on $\Gamma, x : \sigma_1 \rhd N_1 : \tau$, we get $\Gamma \rhd \lambda x : \sigma_1 \,.\, N_1 : \sigma_1 \to \tau$.
Since $\sigma_2 <: \sigma_1$, by $(<:)$, we have that $\Gamma \rhd \lambda x : \sigma_1 \,.\, N_1 : \sigma_2 \to \tau$, *i.e.*,
$\Gamma \rhd N : \sigma$.
Now, using (abs), we get that $\Gamma \rhd \lambda x : \sigma_1 \,.\, M_1 \xrightarrow{\sigma_2 \to \tau} \lambda x : \sigma_1 \,.\, N_1$, *i.e.*, $\Gamma \rhd M \xrightarrow{\sigma} N$.
Finally,

$$\begin{array}{rcl} Erase(N) & \equiv & \lambda x.\, Erase(N_1) \\ & \equiv & \lambda x.\, V \\ & \equiv & U \end{array}$$

■

Corollary 5.6 *If* $\Gamma \rhd M : \sigma$ *and* $Erase(M) \xrightarrow{\beta,\eta}\!\!\!\!\rightarrow U$ *then* $\exists\, N$ *such that* $\Gamma \rhd N : \sigma$, $\Gamma \rhd M \xrightarrow{\sigma}\!\!\!\!\rightarrow N$, *and* $Erase(N) \equiv U$.

Proof By induction on the number of reduction step of $Erase(M) \xrightarrow{\beta,\eta}\!\!\!\!\rightarrow U$ and Lemma 5.5. ■

Lemma 5.7 *If U is a $\beta,\eta-$normal form then*

$$U \equiv \ \lambda x_1 \ldots \lambda x_n \,.\, y\, V_1 \ldots V_k$$

where $n \geq 0, k \geq 0$, y is a variable, and $V_1, \ldots, V_k$ are $\beta,\eta-$normal forms.

Proof By induction on the structure of U. ■

Lemma 5.8 *If U is a $\beta,\eta-$normal form and $\Gamma \triangleright U : \sigma$, then there exists a term N such that $\Gamma \triangleright N : \sigma$, $Erase(N) \equiv U$, and for any term P such that $\Gamma \triangleright P : \sigma$ and $Erase(P) \equiv U$, we have that $\Gamma \triangleright P \stackrel{\sigma}{\longrightarrow\!\!\!\!\rightarrow} N$.*

Proof Let U be a β,η-normal form. By Lemma 5.7,

$$U \ \equiv \ \lambda x_1 \ldots \lambda x_n \,.\, y\, V_1 \ldots V_k, \qquad n \geq 0, k \geq 0$$

We proceed by induction on the length of U. Suppose $\Gamma \triangleright U : \sigma$, then

$$\sigma \equiv \sigma_1 \rightarrow \sigma_2 \rightarrow \cdots \rightarrow \sigma_n \rightarrow \rho \tag{7}$$
$$\Gamma' \triangleright y\, V_1 \ldots V_k : \rho \qquad \text{where} \quad \Gamma' \equiv \Gamma, x_i : \sigma_i, \quad i = 1, \ldots, n \tag{8}$$

By (8), we have

$$y : \rho_1 \rightarrow \cdots \rightarrow \rho_k \rightarrow \rho' \in \Gamma' \tag{9}$$
$$\Gamma' \triangleright V_i : \rho_i', \qquad \text{where} \quad \rho_i' <: \rho_i, \quad i = 1, \ldots, k \tag{10}$$
$$\rho' <: \rho \tag{11}$$

Hence, $\Gamma' \triangleright V_i : \rho_i$.
By induction hypothesis applied to the shorter terms V_i, there exist terms $\Gamma' \triangleright N_i : \rho_i$ with the stated property.
Take $N \equiv \ \lambda x_1 : \sigma_1 \ldots \lambda x_n : \sigma_n \,.\, y\, N_1 \ldots N_k$.
Since $\Gamma' \triangleright N_i : \rho_i$ and $y : \rho_1 \rightarrow \cdots \rightarrow \rho_k \rightarrow \rho' \in \Gamma'$, using $(\rightarrow Elim)$ k times, we get that

$$\Gamma' \triangleright y\, N_1 \ldots N_k : \rho',$$

by (11),

$$\Gamma' \triangleright y\, N_1 \ldots N_k : \rho.$$

Using $(\rightarrow Intro)$ n times, we get

$$\Gamma \triangleright \lambda x_1 : \sigma_1 \ldots \lambda x_n : \sigma_n \,.\, y\, N_1 \ldots N_k : \sigma_1 \rightarrow \sigma_2 \rightarrow \cdots \rightarrow \sigma_n \rightarrow \rho,$$

i.e.,

$$\Gamma \triangleright N : \sigma.$$

Clearly, $Erase(N) \equiv U$.
Now, suppose $\Gamma \triangleright P : \sigma$ and $Erase(P) \equiv U$.
Then, $P \equiv \ \lambda x_1 : \sigma_1' \ldots \lambda x_n : \sigma_n' \,.\, y\, P_1 \ldots P_k$ with $Erase(P_i) \equiv V_i$, $i = 1, \ldots, k$.
From $\Gamma \triangleright P : \sigma$, we have Γ, $x_1 : \sigma_1', \ldots, x_n : \sigma_n' \triangleright y\, P_1 \ldots P_k : \tau'$ and $\sigma_1' \rightarrow \cdots \rightarrow \sigma_n' \rightarrow \tau' \ <: \ \sigma$, *i.e.*, $\sigma_i <: \sigma_i'$ for $i = 1, \ldots, n$ and $\tau' <: \rho$.

Hence, $\Gamma,\, x_1 : \sigma_1, \ldots, x_n : \sigma_n \rhd y\, P_1 \ldots P_k : \rho$.
Since $y : \rho_1 \rightarrow \cdots \rightarrow \rho_k \rightarrow \rho' \in \Gamma'$, $\Gamma' \rhd P_i : \rho_i''$ with $\rho_i'' <: \rho_i$, we have $\Gamma' \rhd P_i : \rho_i$.
By induction hypothesis, $\Gamma' \rhd P_i \xrightarrow{\rho_i}\!\!\!\!\rightarrow N_i$. By repeated use of rule $(app2)$ and the fact that $\Gamma' \rhd y : \rho_1 \rightarrow \cdots \rightarrow \rho_k \rightarrow \rho$, we have $\Gamma' \rhd y\, P_1 \ldots P_k \xrightarrow{\rho}\!\!\!\!\rightarrow y\, N_1 \ldots N_k$.
By repeated use of rule (abs), we get

$$\begin{array}{l} \Gamma \rhd \lambda x_1 : \sigma_1 \ldots \lambda x_n : \sigma_n \,.\, y\, P_1 \ldots P_k \\ \qquad \xrightarrow{\sigma_1 \rightarrow \cdots \rightarrow \sigma_n \rightarrow \rho}\!\!\!\!\rightarrow \lambda x_1 : \sigma_1 \ldots \lambda x_n : \sigma_n \,.\, y\, N_1 \ldots N_k \end{array} \tag{12}$$

Now, since $\sigma_n <: \sigma_n'$, by $(<:)$

$$\begin{array}{l} \Gamma,\, x_1 : \sigma_1, \ldots, x_{n-1} : \sigma_{n-1} \rhd \lambda x_n : \sigma_n' \,.\, y\, P_1 \ldots P_k \\ \qquad\qquad \xrightarrow{\sigma_n \rightarrow \rho} \lambda x_n : \sigma_n \,.\, y\, P_1 \ldots P_k \end{array}$$

By repeating this argument, we get

$$\begin{array}{l} \Gamma \rhd \lambda x_1 : \sigma_1' \ldots \lambda x_n : \sigma_n' \,.\, y\, P_1 \ldots P_k \\ \qquad \xrightarrow{\sigma_1 \rightarrow \cdots \rightarrow \sigma_n \rightarrow \rho} \lambda x_1 : \sigma_1 \ldots \lambda x_n : \sigma_n \,.\, y\, P_1 \ldots P_k \end{array} \tag{13}$$

Using the reduction (13) followed by (12), we get that

$$\Gamma \rhd P \xrightarrow{\sigma}\!\!\!\!\rightarrow N$$

■

Lemma 5.9 *Let U be any untyped term such that $\Gamma \rhd U : \sigma$ for some context Γ and type σ. Then, U is strongly-normalizing under β, η-reduction.*

Proof By Corollary 22 of [Mit91], it follows that U is the erasure of a simply-typed λ-term. Since simply-typed λ-calculus is strongly normalizing under β, η-reduction, the result follows. ■

Theorem 5.10 *Suppose that $\Gamma \rhd M : \sigma$ and $\Gamma \rhd M \xrightarrow{\sigma}\!\!\!\!\rightarrow N_1, \Gamma \rhd M \xrightarrow{\sigma}\!\!\!\!\rightarrow N_2$, then there is a term $\Gamma \rhd P : \sigma$ such that $\Gamma \rhd N_1 \xrightarrow{\sigma}\!\!\!\!\rightarrow P$ and $\Gamma \rhd N_2 \xrightarrow{\sigma}\!\!\!\!\rightarrow P$.*

Proof Suppose $\Gamma \rhd M \xrightarrow{\sigma}\!\!\!\!\rightarrow N_1, \Gamma \rhd M \xrightarrow{\sigma}\!\!\!\!\rightarrow N_2$.
By Lemmas 5.2 and 5.1, we have $\Gamma \rhd Erase(M) \xrightarrow{\beta,\eta}\!\!\!\!\rightarrow Erase(N_1)$,
$\Gamma \rhd Erase(M) \xrightarrow{\beta,\eta}\!\!\!\!\rightarrow Erase(N_2)$, and $\Gamma \rhd Erase(N_1) : \sigma, \Gamma \rhd Erase(N_2) : \sigma$.
By confluence of β, η-reduction for untyped $\lambda-$calculus, there exists a term U' such that $Erase(N_1) \xrightarrow{\beta,\eta}\!\!\!\!\rightarrow U', Era$
By Corollary 5.6, there exist P_1', P_2' such that $\Gamma \rhd P_1' : \sigma, \Gamma \rhd P_2' : \sigma, \Gamma \rhd N_1 \xrightarrow{\sigma}\!\!\!\!\rightarrow P_1'$,
$\Gamma \rhd N_2 \xrightarrow{\sigma}\!\!\!\!\rightarrow P_2'$, and $Erase(N_1) \equiv U', Erase(N_2) \equiv U'$.
Thus, by Lemma 5.1 $\Gamma \rhd U' : \sigma$. By Lemma 5.9, there exists a term U of $\beta, \eta-$normal form such that $\Gamma \rhd U : \sigma, U' \xrightarrow{\beta,\eta}\!\!\!\!\rightarrow U$.
By Corollary 5.6, there exists P_1, P_2 such that $\Gamma \rhd P_1 : \sigma, \Gamma \rhd P_2 : \sigma, \Gamma \rhd P_1' \xrightarrow{\sigma}\!\!\!\!\rightarrow P_1$,
$\Gamma \rhd P_2' \xrightarrow{\sigma}\!\!\!\!\rightarrow P_2$, and $Erase(P_1) \equiv U, Erase(P_2) \equiv U$.
Now, by Lemma 5.8 there exists a term P such that $\Gamma \rhd P : \sigma$, and $\Gamma \rhd P_1 \xrightarrow{\sigma}\!\!\!\!\rightarrow P$,
$\Gamma \rhd P_2 \xrightarrow{\sigma}\!\!\!\!\rightarrow P$, *i.e.*, there exists P such that $\Gamma \rhd N_1 \xrightarrow{\sigma}\!\!\!\!\rightarrow P, \Gamma \rhd N_2 \xrightarrow{\sigma}\!\!\!\!\rightarrow P$.
This can be illustrated as in Figure 2.

■

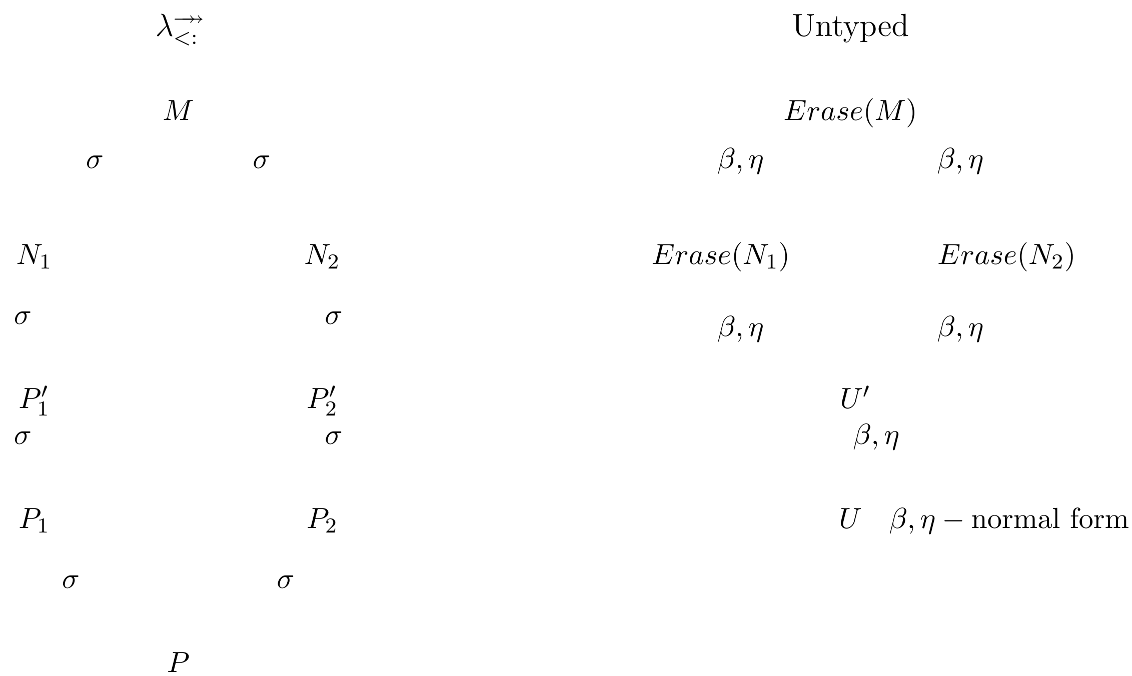

Figure 2: Proof of Theorem 5.10

5.4 Reduction with subtyping and recursion

In the last part of this paper, we consider reduction for $\lambda^{\rightarrow,fix}_{<:}$, the language obtained by adding fixed-point operators to $\lambda^{\rightarrow}_{<:}$. One reason this system deserves separate consideration is that our confluence proof for $\lambda^{\rightarrow}_{<:}$ relies heavily on the strong-normalization property. Since strong normalization fails for $\lambda^{\rightarrow,fix}_{<:}$, it is therefore not easy to see, a priori, whether reduction for $\lambda^{\rightarrow,fix}_{<:}$ is likely to be confluent.

We prove confluence of the reduction system for $\lambda^{\rightarrow,fix}_{<:}$ by using the corresponding calculus $\lambda^{\rightarrow,lab}_{<:}$ with all fixed point operators labeled. The reduction relation $\Gamma \triangleright M \xrightarrow{\sigma}_{lab} N$ on $\lambda^{\rightarrow,lab}_{<:}$ is the same as reduction for $\lambda^{\rightarrow}_{<:}$, with the expected rule for labeled fixed points,

$$(fix_{lab}) \qquad \Gamma \triangleright fix^{n+1}_{\sigma} \xrightarrow{\tau}_{lab} \lambda f : \sigma \to \sigma \, . \, f \, (fix^{n}_{\sigma} \, f) \qquad \text{if } \Gamma \triangleright fix^{n+1}_{\sigma} : \tau$$

It is relatively straightforward to verify the lifting and projection properties of labeling for $\lambda^{\rightarrow,fix}_{<:}$. These may be stated as follows, writing $L \in lab(M)$ if L is a labeled term with $\natural(L) = M$.

Lemma 5.11 *Suppose that* $\Gamma \triangleright M \xrightarrow{\sigma}\!\!\!\!\rightarrow N$. *Then there exists a natural number* k *such that if* $M^{\#} \in lab(M)$ *with each label in* M *at least* k *then* $\Gamma \triangleright M^{\#} \xrightarrow{\sigma}\!\!\!\!\rightarrow_{lab} N^{\#}$.

Proof If $\Gamma \triangleright M \xrightarrow{\sigma}\!\!\!\!\rightarrow N$ then there exist terms $M_1, \ldots, M_k$, $k \geq 0$ such that $M \equiv M_0$, $\Gamma \triangleright M_i \xrightarrow{\sigma} M_{i+1}$, $M_k \equiv N$. Proof by induction on k. ∎

Lemma 5.12 *If* $\Gamma \triangleright M^{\#} \xrightarrow{\sigma}_{lab} N^{\#}$ *then* $\Gamma \triangleright M \xrightarrow{\sigma} N$ *where* $M^{\#} \in lab(M)$, $N^{\#} \in lab(N)$.

Proof By induction on the proof of $\Gamma \triangleright M^{\#} \xrightarrow{\sigma}_{lab} N^{\#}$. ∎

Corollary 5.13 *Confluence of* $\xrightarrow{\sigma}\!\!\!\!\rightarrow_{lab}$ *implies confluence of* $\xrightarrow{\sigma}\!\!\!\!\rightarrow$.

5.5 Confluence proof using labeled reduction

As shown in the previous section, confluence of the reduction system for $\lambda_{<:}^{\rightarrow,fix}$ follows from the confluence of labeled reduction. In this section we establish the confluence of the reduction system for $\lambda^{\rightarrow,lab}$ thereby proving confluence for λ-calculus with subtyping and fixed points.

Just as in Section 5.3, we prove the confluence of labeled reduction by considering untyped λ-terms and using confluence properties of β,η-reduction on them. However, there is one main step in extending the ideas from the proof for $\lambda_{<:}^{\rightarrow}$ to that for $\lambda_{<:}^{\rightarrow,lab}$. In $\lambda_{<:}^{\rightarrow}$ the only additional basic reduction rule besides β and η was (<:) which could be easily mimicked in the untyped λ-calculus (as a 0-step reduction!). However, in $\lambda_{<:}^{\rightarrow,lab}$, we have an additional set of reduction rules, namely those of labeled fixed points that need to be suitably mimicked as reduction on untyped terms. One possibility is to introduce labeled fixed point constants in the target untyped calculus and include their reduction rules in the untyped calculus. However, this introduces some extraneous complication, since we would have to establish confluence of untyped lambda calculus with labeled fixed-point reduction. A simpler solution is to take the target untyped λ-calculus with only include β,η-reductions and to mimic labeled fixed-point reductions through the definition of the erasure function. More specifically, we translate labeled fixed points by performing their bounded number of unwindings completely and erasing the type-annotations of the λ-bound variables from the resulting term. This reduces all labeled fixed points to terms involving fix_{σ}^{0} which we include as constants in the untyped λ-calculus. The inclusion of these constants does not interfere with the confluence of the system since there are no associated reduction rules with these constants, and thus the reduction system is still only β,η on untyped terms.

5.5.1 Type System for Untyped Terms

We consider untyped terms which may include constants of the form fix_{σ}^{0}. Thus, the typing rules for untyped terms are those given in Section 5.3 together with the following additional axiom.

$$(fix) \qquad\qquad \phi \triangleright fix_{\sigma}^{0} : (\sigma \rightarrow \sigma) \rightarrow \sigma$$

5.5.2 Type Erasure

The *type erasure* of a term M, written $Erase(M)$, is defined as follows.

$$\begin{array}{rcl} Erase(x) & = & x \\ Erase(fix_{\sigma}^{0}) & = & fix_{\sigma}^{0} \\ Erase(fix_{\sigma}^{n+1}) & = & \lambda f.\, f\,((Erase(fix_{\sigma}^{n}))\, f) \\ Erase(M\,N) & = & Erase(M)\, Erase(N) \\ Erase(\lambda x : \sigma\,.\, M) & = & \lambda x.\, Erase(M) \end{array}$$

5.5.3 Proof

Lemma 5.14 *If* $\Gamma \triangleright M : \sigma$ *then* $\Gamma \triangleright Erase(M) : \sigma$.

Proof By induction on the proof of $\Gamma \triangleright M : \sigma$. ■

Lemma 5.15 *If* $\Gamma \triangleright M \xrightarrow{\sigma}_{lab} N$ *then* $Erase(M) \overset{\beta,\eta}{\twoheadrightarrow} Erase(N)$