# FULL ABSTRACTION AND THE CONTEXT LEMMA

TREVOR JIM* AND ALBERT R. MEYER†

**Abstract.** It is impossible to add a combinator to PCF to achieve full abstraction for models such as Berry's stable domains in a way analogous to the addition of the "parallel-or" combinator that achieves full abstraction for the familiar cpo model. In particular, we define a general notion of rewriting system of the kind used for evaluating simply typed $\lambda$-terms in Scott's PCF. Any simply typed $\lambda$-calculus with such a "PCF-like" rewriting semantics is shown necessarily to satisfy Milner's Context Lemma. A simple argument demonstrates that any denotational semantics that is adequate for PCF, and in which certain simple Boolean functionals exist, cannot be fully abstract for *any* extension of PCF satisfying the Context Lemma. An immediate corollary is that stable domains cannot be fully abstract for any extension of PCF definable by PCF-like rules.

**Key words.** Stable functions, full abstraction, context lemma, PCF, standardization.

**AMS subject classifications.** 03B40 68N15 68Q40 68Q50 68Q55

## 1. Introduction.

A paradigmatic example of a functional programming language is PCF, Scott's simply typed $\lambda$-calculus for recursive functions on the integers [35]. Many categories of denotational meaning are known to *adequately* reflect the computational behavior of PCF in a precise technical sense, namely, a PCF term evaluates to the numeral $\underline{n}$ iff it means the integer $n$. But typically there are pairs of terms with distinct meanings that nevertheless are computationally indistinguishable in PCF. For example, with the semantics based on cpo's, PCF must be extended with a "parallel-or" combinator in order to express enough computations to be *fully abstract*, *i.e.*, semantical distinctions and computational distinctions between terms coincide [32, 31].

The problem of giving a semantical description of a fully abstract model of unextended PCF remained open for nearly fifteen years (*cf.* [28, 9, 29, 37]); it has only recently been solved [3, 21]. During that time, efforts to construct spaces of "sequential" functions corresponding to those definable in the original PCF without parallelism led to the discovery of a number of new domains suitable for denotational semantics. Although none are fully abstract for PCF, one motivation for the development of spaces such as the *stable* functions, *bistable* functions, *sequential algorithms* [6, 5, 9, 8, 16], and most recently the *strongly stable* functions [14], was that they captured various aspects of sequentiality and so seemed "closer" to full abstraction for unextended PCF than the popular cpo model.

The stable function model in particular has a simple definition and attractive category-theoretic properties. Its only apparent technical peculiarity is that stable domains of functions are not partially ordered pointwise; in general, the stable ordering strictly refines the pointwise ordering. Nevertheless, just as for the cpo model, the elements of stable domains of type $\sigma \to \tau$ are actually total functions from elements of type $\sigma$ to elements of type $\tau$. Likewise, there is a natural notion of finite and effective elements of stable domains, and these domains yield an adequate least fixed-point model for PCF. Further, they form a Cartesian Closed Category with solutions for domain equations [6]. This category was also independently discovered and used in

* MIT LABORATORY FOR COMPUTER SCIENCE. E-MAIL: TREVOR@THEORY.LCS.MIT.EDU.
WORK SUPPORTED BY ARO GRANT DAAL03–89–G–0071.
† MIT LABORATORY FOR COMPUTER SCIENCE. E-MAIL: MEYER@THEORY.LCS.MIT.EDU.
WORK SUPPORTED BY ONR GRANT N00014–89–J–1988 AND NSF GRANT 8819761–CCR.

constructing a model of polymorphic $\lambda$-calculus [17]. So the stable domains seem to offer a setting for a theory for higher-order recursive computation with many of the attractions of the cpo category.

However, one important result about cpo's is not known for stable domains, namely, full abstraction with respect to some extension of PCF analogous to the parallel-or extension which Plotkin and Sazonov provided for the cpo model. What might a symbolic-evaluator for an extended PCF look like if it was well matched—fully abstract—with the stable model? We conclude that such an evaluator will have to be unusual looking: it cannot be specified by the kind of term-rewriting based evaluation rules known for PCF and its extensions.

The significance of this negative result hinges heavily on how drastic we judge it to go beyond the scope of PCF-like rules. It is of course possible that some operational behavior that we declare to be non-PCF-like, in our technical sense, will nevertheless offer a useful extension of PCF for which stable domains are fully abstract. For example, Bloom [11] provides such an extension for complete lattice models, though he goes on to criticize the rather complex algorithmic specification of the combinators in his extension. (The general benefits of structured approaches to operational semantics and connections to full abstraction are discussed in [27, 12].)

To illustrate the generality of our notion of PCF-like rules, we note that the standard extensions of PCF by parallel-or and existential combinators are easily seen to be PCF-like. For example, we can define an evaluator for Plotkin's $\exists$ constant [31] while remaining within a term rewriting discipline, as follows. Let $p : \iota \to o$ be an "integer predicate" variable, and use the rules:

$$\begin{aligned} \exists p &\to \mathtt{cond}\,(p\underline{n})\,\mathtt{tt}\,\Omega, \\ \exists p &\to \mathtt{cond}\,(p\Omega)\,\Omega\,\mathtt{ff}. \end{aligned}$$

The resulting PCF-like language no longer has a confluent rewriting system, though it remains single-valued, *viz.*, every term rewrites to at most one numeral. In general, our PCF-like rules need not even be single-valued.

A substantial technical contribution of this paper is a simple, modest restriction on the format of rewrite rules which is sufficient to guarantee Milner's Context Lemma [28] for languages defined by such rules. Informally, this "Approximation" Context Lemma requires that if two phrases $M, N$ of the same syntactic functional type yield visibly distinct computational outcomes when used in *some* language context, then there are actual parameters of appropriate argument type, such that $M$ and $N$ each simply *applied* to these arguments, yield visibly distinct computational outcomes. This property, more perspicuously dubbed *operational extensionality* by Bloom [10, 11], has been identified by many authors as technically significant in program semantics [38, 30, 25, 1, 19, 2, 36]. The key to the proof of the Context Lemma is a new Standard Reduction Theorem 4.4 for PCF-like rewrite systems.

Our work borrows much from Bloom [10, 11]. The second author raised the question of whether there is a "reasonable" extension of PCF that would yield a fully abstract evaluator for lattice models [33, 34]. In answering this question, Bloom emphasized how the Context Lemma and full abstraction were incompatible with *single-valued* evaluators for the lattice model. He also characterized a general class of *consistent* rewrite rules that ensured the soundness of the Context Lemma. However, in order to encompass the computational behavior of the $\exists$ combinator, Bloom needed to develop an auxiliary notion of "observation calculi".

Our PCF-like rules are, in an appropriate sense, as powerful as Bloom's observational calculi, and strictly subsume the class of consistent rules. In particular, consistent rules are necessarily confluent and hence single-valued; as Bloom remarks [10], introducing a `join` combinator with simple multiple-valued rewrite rules yields a PCF extension both fully abstract for the lattice model and also satisfying the Context Lemma. Our wish to simplify Bloom's criteria while dealing with nonconfluent rewriting systems forced us, instead, to a rather elaborate theory of standard reductions.

As an aside, we also point out that it is questionable whether the (bi)stable and similar domains are closer to full abstraction for PCF. In particular, although some operationally valid equations that fail in the cpo model do hold, for example, in the stable model, we note in Corollary 3.3 that the converse also happens: some equations that hold in the cpo model fail in the stable model. The cpo, stable and likewise the bistable models thus offer information about the operational behavior of PCF terms that is not apparently comparable, and it is hard to see how to judge which is a more accurate model.

The outline of our argument is as follows: in §2 we formulate the key concepts of observational approximation, adequacy, and full abstraction in a general setting. Then in §3, Theorem 3.2, we give a short proof that any denotational semantics that is adequate for PCF, and in which a certain simple Boolean functional exists, cannot be fully abstract for extensions of PCF satisfying the Context Lemma. The Boolean functional is obviously not continuous in Scott's sense, but it is stably continuous, and so does appear in the stable model. We also formulate a Comparability Context Lemma which applies to the bistable domains. Section 4 gives our general notion of term rewriting systems of the kind used for symbolic evaluation of PCF terms. Then in §5, we show that any such system defines an observational approximation relation that must satisfy the Context Lemma [28]. An immediate corollary is Theorem 5.5 that there is no extension of PCF defined by PCF-like rewriting rules for which the stable domain semantics is fully abstract. We state, but do not prove, a similar result for the bistable domains.

**2. Adequacy and Full Abstraction.** Concepts concerning program behavior, such as observational congruence, adequacy, and full abstraction, can usefully be defined in a general setting consisting of:

- a set $\mathcal{L}$, called a *language*, whose elements, $M, N, \ldots$, are called *terms*;
- partial operators $C[\cdot]$ on terms called *contexts*; and
- a set $\mathcal{O}$, called a *notion of observation*, whose elements are predicates on terms called *observations*. When an observation is true of a term, the term is said to *yield* the observation.

We will work with languages whose operational behavior is specified by (possibly nondeterministic) symbolic evaluation of terms, so we further assume a binary relation, "evaluates to", on terms. For such languages, $\mathcal{O}_{\text{eval}}$ captures the familiar notion of observing the final output of an evaluation:

$$\mathcal{O}_{\text{eval}} \quad = \quad \{\text{ “evaluates to } O\text{”} \mid O \text{ is an output term}\}.$$

Here the output terms are those terms regarded as observable "output values". These typically include the ground constants (integers, truth values, ...); $\lambda$-abstractions and finite lists of output values might also be included.

There are other notions of observation based on evaluation. For instance, $\mathcal{O}_{\text{lazy}}$ consists of the single predicate true of exactly those terms whose evaluation can terminate. And notions of observation can be based on semantics of terms, *e.g.*,

$$\mathcal{O}_{\rm int} \quad = \quad \{\text{“has the meaning of } O\text{”} \mid O \text{ is an output term}\}.$$

In this paper, however, we will be mainly concerned with $\mathcal{O}_{\rm eval}$.

Any notion of observation induces a preordering on terms called *observational approximation*. Intuitively, one term approximates another if, according to the chosen notion of observation, the approximated term exhibits at least as much observable behavior when used in any program as the approximating term.

DEFINITION 2.1. *Let $\mathcal{L}$ be a language with a notion of observation $\mathcal{O}$. A term $M$* observationally approximates *a term $N$, written $M \sqsubseteq_{\rm obs} N$, if for all contexts $C[\cdot]$, whenever $C[M]$ is a term yielding an observation from $\mathcal{O}$, then $C[N]$ is a term yielding it as well. $M$ and $N$ are* observationally congruent, *written $M \equiv_{\rm obs} N$, iff $M \sqsubseteq_{\rm obs} N$ and $N \sqsubseteq_{\rm obs} M$.*

Observational approximation provides precise meaning for questions such as, “Does my code meet a specification?” or “Will my new implementation of a module change the behavior of the program?”

In languages like PCF with applicative syntax and a suitable notion of closed terms, analysis of observational approximation can be simplified by appealing to a *Context Lemma*:

DEFINITION 2.2. *Let $\mathcal{L}$ be a language with a notion of observation $\mathcal{O}$. We say a term $M$* applicatively approximates *a term $N$, written $M \sqsubseteq_{\rm app} N$, iff for all vectors of closed terms, $\vec{P}$, whenever $M\vec{P}$ is a term yielding an observation, $N\vec{P}$ is a term yielding it as well.* The Approximation Context Lemma[1] *holds if for all closed terms $M$ and $N$,*

$$M \sqsubseteq_{\rm app} N \quad \textit{iff} \quad M \sqsubseteq_{\rm obs} N.$$

A fundamental result of Milner [28] is that under $\mathcal{O}_{\rm eval}$ with numerals taken as the output terms, PCF itself, as well as its extension with parallel-or, satisfies the Approximation Context Lemma. We will see later that the Approximation Context Lemma holds for *all* languages defined in a “PCF-like” operational discipline, including, of course, PCF and its familiar extensions.

One method for proving observational approximations is by developing an abstract meaning, $[\![M]\!]$, of a term $M$ that is adequate to determine its observations.

DEFINITION 2.3. *A* meaning function *for a language $\mathcal{L}$ is a function $[\![\cdot]\!]$ from terms $M$ to values $[\![M]\!]$ in some set, partially ordered by a relation $\sqsubseteq$. A meaning function is* compositional *iff for all terms $M, N$ and contexts $C[\cdot]$, if $[\![M]\!] \sqsubseteq [\![N]\!]$ and $C[M]$ is a term, then $C[N]$ is a term and $[\![C[M]]\!] \sqsubseteq [\![C[N]]\!]$.*

*A meaning function is* adequate[2] *for a notion of observation $\mathcal{O}$ iff for all terms $M, N$ and all observations $obs \in \mathcal{O}$,*

$$\big([\![M]\!] \sqsubseteq [\![N]\!] \text{ and } obs(M)\big) \quad \textit{implies} \quad obs(N).$$

[1] In particular when $\mathcal{O}$ is $\mathcal{O}_{\rm eval}$, Bloom [10] calls this “operational extensionality” while Milner [28] uses simply “the Context Lemma”. We use the more descriptive “Approximation Context Lemma” because we will later consider Context Lemmas that are not based on approximation.

[2] As with the Context Lemma, we might more descriptively call this “approximation adequate”; but we will use only the version of adequacy based on approximation, and call it simply adequacy for brevity.

Adequacy and compositionality guarantee that the meanings accurately predict observational approximation.

LEMMA 2.4. *A compositional meaning function* $[\![\cdot]\!]$ *is adequate for a notion of observation iff for all terms* $M$ *and* $N$,

$$[\![M]\!] \sqsubseteq [\![N]\!] \quad \textit{implies} \quad M \sqsubseteq_{\text{obs}} N.$$

The ordering on adequate meanings may be strictly finer than observational approximation. In the ideal situation, known as *full abstraction*, the two orderings coincide:

DEFINITION 2.5. *Let* $[\![\cdot]\!]$ *be a meaning function for a language* $\mathcal{L}$ *with a notion of observation* $\mathcal{O}$. *We say* $[\![\cdot]\!]$ *is* approximation fully abstract[3] *if for all terms* $M$ *and* $N$,

$$[\![M]\!] \sqsubseteq [\![N]\!] \quad \textit{iff} \quad M \sqsubseteq_{\text{obs}} N.$$

*It is* equationally fully abstract *if for all* $M$ *and* $N$,

$$[\![M]\!] = [\![N]\!] \quad \textit{iff} \quad M \equiv_{\text{obs}} N.$$

Approximation full abstraction trivially implies adequacy for compositional meaning functions. Assuming that each output term evaluates to itself, it follows immediately that if $[\![\cdot]\!]$ is adequate for $\mathcal{O}_{\text{eval}}$ and $[\![O]\!] \sqsubseteq [\![M]\!]$, then $M$ evaluates to $O$, for any output term $O$. If, in addition, the meaning function is *sound* for the evaluator, we easily obtain a familiar (*cf.* [27]) alternate characterization of adequacy:

DEFINITION 2.6. *A meaning function* $[\![\cdot]\!]$ *is* sound *for an "evaluates to" relation if for all terms* $M$ *and* $N$,

$$M \textit{ evaluates to } N \quad \textit{implies} \quad [\![M]\!] = [\![N]\!].$$

LEMMA 2.7. *A sound, compositional meaning function* $[\![\cdot]\!]$ *is adequate for* $\mathcal{O}_{\text{eval}}$ *iff*

$$[\![O]\!] = [\![M]\!] \quad \textit{iff} \quad M \textit{ evaluates to } O,$$

*for all terms* $M$ *and output terms* $O$.

This paper focuses specifically on the language PCF and its extensions. The precise (usual) definitions of PCF syntax and semantics appear in Appendix A, and we provide only a quick review here.

PCF is a simply typed $\lambda$-calculus with Boolean and natural number ground types, numerals $\underline{n}$ for $n \geq 0$, Boolean constants `tt` and `ff`, and simple arithmetic, recursion, and conditional operators. The evaluation relation $\twoheadrightarrow$ of the language is given by term rewriting rules.

DEFINITION 2.8. *An* extension of PCF *is a simply typed language together with a set of rewrite rules, such that the types, typed constants, and rewrite rules of the extension include those of PCF. The extension is* conservative *iff for all PCF terms* $M$, *and all terms* $N$ *in the extension,*

$$M \twoheadrightarrow_{\text{extended}} N \quad \textit{iff} \quad M \twoheadrightarrow_{\text{PCF}} N.$$

Observational congruence, adequacy, *etc.*, for PCF and its extensions will be defined with respect to $\mathcal{O}_{\text{eval}}$, where we take the rewriting relation $\twoheadrightarrow$ as the "evaluates to" relation, and the output terms are the ground constants `tt`, `ff`, and $\underline{n}$ for $n \geq 0$.

The results of the next section, which examines full abstraction for models of extensions of PCF, require that we prove facts about the meanings of terms while

[3] Stoughton [37] calls this "inequationally fully abstract."

knowing very little about the extensions or the models. We will only have adequacy, conservativity, and a few other assumptions to work with. The following lemma shows that this gives us enough to reason about the unextended terms of the language.

LEMMA 2.9. *If a model is adequate for a conservative extension of PCF, then it is also adequate for PCF.*

*Proof.* Suppose a model $[\![\cdot]\!]$ is adequate for a conservative extension of PCF, and $[\![M]\!] \sqsubseteq [\![N]\!]$ for some PCF terms $M, N$. All models are compositional, so $[\![C[M]]\!] \sqsubseteq [\![C[N]]\!]$ for any PCF context $C[\cdot]$. So for any ground PCF constant $c$, if $C[M] \twoheadrightarrow_{\mathrm{extended}} c$, then $C[N] \twoheadrightarrow_{\mathrm{extended}} c$ by adequacy. And then by conservativity, if $C[M] \twoheadrightarrow_{\mathrm{PCF}} c$, then $C[N] \twoheadrightarrow_{\mathrm{PCF}} c$. Hence, $M \sqsubseteq^{\mathrm{PCF}}_{\mathrm{obs}} N$. □

We will further require that our models be sound, and that the ground types $o$ and $\iota$ be interpreted as the flat cpos $\{tt, ff\}_\perp$ and $\{0, 1, \ldots\}_\perp$, with the standard interpretation of $\mathtt{tt}$, $\mathtt{ff}$, and the numerals $\underline{n}$. Such models will be called *models with Booleans* (though they are indeed also models with integers). Two models with Booleans of particular interest are the cpo model $\mathcal{C}[\![\cdot]\!]$ and the stable model $\mathcal{S}[\![\cdot]\!]$. Both models are adequate but not fully abstract for PCF.

The additional information about the ground types of models with Booleans is in fact enough to determine the meanings of ground PCF terms.

LEMMA 2.10. *The meaning of any closed PCF term of ground type is the same in all models with Booleans that are adequate for PCF.*

*Proof.* Let $M$ be a closed PCF term of type $o$ (the case $M : \iota$ is similar). In PCF, exactly one of the following holds: (1) $M \twoheadrightarrow_{\mathrm{PCF}} \mathtt{tt}$; (2) $M \twoheadrightarrow_{\mathrm{PCF}} \mathtt{ff}$; or (3) neither (1) nor (2) holds. And by Lemma 2.7, $M \twoheadrightarrow_{\mathrm{PCF}} \mathtt{tt}$ iff $[\![M]\!] = [\![\mathtt{tt}]\!] = tt$ for *any* model with Booleans $[\![\cdot]\!]$ adequate for PCF. Similarly, cases (2) and (3) imply $[\![M]\!] = ff$ and $[\![M]\!] = \perp$ respectively. □

Thus we can use any particular adequate model with Booleans, like the familiar cpo model, to discover the meaning of ground PCF terms for arbitrary adequate models with Booleans. We have less to say about terms of higher type. But the following notions are useful:

DEFINITION 2.11. *Let $\tau$ be a first-order type, that is, a type of the form $\sigma_1 \to \cdots \to \sigma_n$, where $\sigma_j$ is a ground type for $1 \le j \le n$. Let $[\![\cdot]\!]_i$ for $i = 1, 2$ be type frames such that $\sqsubseteq_1$ on $[\![\sigma_j]\!]_1$ equals $\sqsubseteq_2$ on $[\![\sigma_j]\!]_2$, and let $f_i \in [\![\tau]\!]_i$. Then $f_1$* pointwise approximates *$f_2$, written $f_1 \sqsubseteq_{\mathrm{pnt}} f_2$, iff for all $d_j \in [\![\sigma_j]\!]_1$,*

$$f_1(d_1)\cdots(d_n) \quad \sqsubseteq_1 \quad f_2(d_1)\cdots(d_n).$$

It follows immediately from Lemma 2.10 that the functions that are the meanings of a PCF term of first-order type agree *pointwise* in all models with Booleans that are adequate for PCF. So we can use the meaning of a first-order PCF term in some particular model to reason about its meaning in any adequate model with Booleans.

However, pointwise equality is not quite the same as equality of functions. For example, consider the conditional constant $\mathtt{cond}_o : o \to o \to o \to o$. Now $\mathcal{S}[\![\mathtt{cond}_o]\!] \equiv_{\mathrm{pnt}} \mathcal{C}[\![\mathtt{cond}_o]\!]$. But the stable domain does not contain parallel-or, so the stable and cpo meanings of $o \to o \to o$ are different. Thus, $\mathcal{S}[\![\mathtt{cond}_o]\!] \neq \mathcal{C}[\![\mathtt{cond}_o]\!]$ since the two functions have different codomains.

Nevertheless, it follows immediately from the definitions that pointwise approximation has the following useful property:

LEMMA 2.12. *Let $[\![\cdot]\!]$ be a model with Booleans that is adequate for PCF, and let $M$ and $N$ be closed PCF terms of first-order type. Then*

$$[\![M]\!] \sqsubseteq_{\text{pnt}} [\![N]\!] \quad \textit{implies} \quad M \sqsubseteq_{\text{app}} N.$$

**3. Failures of Full Abstraction.** Our first theorem hinges on the presence of certain simple functionals over the Booleans.

DEFINITION 3.1. *Let True be the constant tt function on the flat Booleans, and True! be the strict constant tt function. A* true-separator *is a function $f$ satisfying*

$$\begin{aligned} f(\mathit{True}) &= \mathit{tt}, \\ f(\mathit{True!}) &= \mathit{ff}. \end{aligned}$$

THEOREM 3.2. *Let $[\![\cdot]\!]$ be a model with Booleans that is adequate for some conservative extension of PCF satisfying the Approximation Context Lemma. If $[\![\cdot]\!]$ contains a true-separator, it is not equationally fully abstract.*

*Proof.* Define the terms

$$\begin{aligned} \texttt{True} &\stackrel{\text{def}}{\equiv} \lambda x.\texttt{tt}, \\ \texttt{True!} &\stackrel{\text{def}}{\equiv} \lambda x.\texttt{cond}\, x\, \texttt{tt}\, \texttt{tt}. \end{aligned}$$

By the definition of model with Booleans, $[\![\texttt{True}]\!] = \mathit{True}$. And by Lemma 2.10, $[\![\texttt{cond}]\!] \equiv_{\text{pnt}} \mathcal{C}[\![\texttt{cond}]\!]$, so by definition of model with Booleans, we have $[\![\texttt{True!}]\!] = \mathit{True!}$. Then $\texttt{True!} \sqsubseteq_{\text{app}} \texttt{True}$ by Lemmas 2.9 and 2.12. So by the Approximation Context Lemma, $\texttt{True!} \sqsubseteq_{\text{obs}} \texttt{True}$.

We conclude that there is no term $P$ defining a true-separator; otherwise `True!` and `True` yield distinct observations in the context $(P\ [\cdot])$, contradicting the fact that $\texttt{True!} \sqsubseteq_{\text{obs}} \texttt{True}$.

However, we can define a true-separator *detector*, $D$, as follows:

$$D \stackrel{\text{def}}{\equiv} \lambda x.\texttt{cond}\,(x\,\texttt{True})\,(\texttt{cond}\,(x\,\texttt{True!})\,\Omega^o\,\texttt{tt})\,\Omega^o,$$

where $\Omega^o$ is the divergent term $(\texttt{Y}_o(\lambda z^o.z))$. By Lemma 2.10, $[\![\Omega^o]\!] = \mathcal{C}[\![\Omega^o]\!] = \bot$, and so

$$[\![D]\!](f) \quad = \quad \begin{cases} \mathit{tt} & \text{if } f \text{ is a true-separator,} \\ \bot & \text{otherwise.} \end{cases}$$

Now $[\![\lambda x.\Omega^o]\!]$ is the constant $\bot$ function, so $[\![D]\!] \neq [\![\lambda x.\Omega^o]\!]$, since they differ exactly on arguments that are true-separators. But since true-separators are not definable by terms, $D$ and $\lambda x.\Omega^o$ are applicatively congruent. Then by the Approximation Context Lemma, they are observationally congruent, contradicting equational full abstraction. ◻

COROLLARY 3.3. *If a stable function model with Booleans is adequate for a conservative extension of PCF that satisfies the Approximation Context Lemma, then the model is not equationally fully abstract.*

*Proof.* Every stable function model with Booleans contains a true-separator *truesep*, defined as follows:

$$\mathit{truesep}(g) \quad = \quad \begin{cases} \mathit{tt} & \text{if } g = \mathit{True}, \\ \mathit{ff} & \text{if } g = \mathit{True!}, \\ \bot & \text{otherwise.} \end{cases}$$

□

COROLLARY 3.4. *The PCF equations valid in the stable model do not include those valid in the cpo model.*

*Proof.* Just note that $\mathcal{C}[\![D]\!] = \mathcal{C}[\![\lambda x.\Omega^o]\!]$, but $\mathcal{S}[\![D]\!] \neq \mathcal{S}[\![\lambda x.\Omega^o]\!]$. □

Our proof of Corollary 3.3 of course takes advantage of the notable fact that the stable ordering of functions differs from the pointwise ordering, *e.g.*, the pair of functions *True* and *True!* are ordered pointwise but are stable-incomparable. In fact, the first few lines of the proof of Theorem 3.2 already show that *in*equational full abstraction is incompatible with the Approximation Context Lemma for any model in which *True* and *True!* are incomparable; the rest of the proof justifies the stronger conclusion that *equational* full abstraction fails as well.

We remark that the authors of [14] have informed us that their *strongly stable* models are adequate models with Booleans for PCF, and that *truesep* is strongly stable, so Theorem 3.3 and Corollary 3.4 hold for strongly stable models.

Berry realized that altering the pointwise ordering of functions caused difficulties, and he proposed from the start an additional *bistable* model which combines stability with the pointwise ordering. Since the counterexample of Corollary 3.3 relies on the non-pointwise stable ordering, it does not apply to the bistable model.

There is, however, an interesting counterexample to the full abstraction of the bistable model that provides a starting point for extending our results. The counterexample, noted in [16], has its roots in the fundamental motivation behind stable models, *viz.*, to eliminate elements like parallel-or. Consider the following definition:

DEFINITION 3.5. *Let lor be the or-function that is strict in its left argument, and ror be the or-function that is strict in its right argument. An* or-separator *is a function $f$ satisfying*

$$\begin{aligned} f(lor) &= tt, \\ f(ror) &= ff. \end{aligned}$$

The cpo model contains a parallel-or function which bounds the left- and right-strict or-functions, and thus, by monotonicity, cannot contain an or-separator. Since the cpo model is adequate for PCF, an or-separator is not definable in PCF. On the other hand, the stable and bistable models do not contain parallel-or, and in fact, both contain or-separators.

Thus in extending the results to the bistable model, one might try to use an or-separator in the role played by the true-separator in the stable case. Since neither *lor* nor *ror* applicatively approximates the other, an argument based on the Approximation Context Lemma will not work; but a similar argument based on a notion of *observational comparability* does apply:

DEFINITION 3.6. *Let $\mathcal{L}$ be a language with a notion of observation $\mathcal{O}$. Terms $M$ and $N$ are* directly comparable *provided the set of observations yielded by $M$ is setwise comparable to that yielded by $N$. The terms are* observationally comparable, *written $M \sim_{\text{obs}} N$, if for all contexts $C[\cdot]$, the terms $C[M]$ and $C[N]$ are directly comparable. They are* applicatively comparable, *written $M \sim_{\text{app}} N$, if for all vectors $\vec{P}$ of closed terms, $M\vec{P}$ and $N\vec{P}$ are directly comparable. $\mathcal{L}$ with $\mathcal{O}$ is said to* satisfy the Comparability Context Lemma *if for all closed terms $M$ and $N$,*

$$M \sim_{\text{app}} N \quad \textit{iff} \quad M \sim_{\text{obs}} N.$$

THEOREM 3.7. *Let $[\![\cdot]\!]$ be a model with Booleans that is adequate for some conservative extension of PCF satisfying the Comparability and Approximation Context Lemmas. If $[\![\cdot]\!]$ contains an or-separator, it is not equationally fully abstract.*

*Proof.* Consider the terms

$$\begin{aligned}\mathtt{lor} &\stackrel{\text{def}}{\equiv} \lambda xy.\mathtt{cond}\, x\, \mathtt{tt}\, (\mathtt{cond}\, y\, \mathtt{tt}\, \mathtt{ff}),\\ \mathtt{ror} &\stackrel{\text{def}}{\equiv} \lambda xy.\mathtt{cond}\, y\, \mathtt{tt}\, (\mathtt{cond}\, x\, \mathtt{tt}\, \mathtt{ff}).\end{aligned}$$

By Lemmas 2.9, 2.10 and 2.12, we have $[\![\mathtt{lor}]\!] = lor$, $[\![\mathtt{ror}]\!] = ror$, and $\mathtt{lor} \sim_{\text{app}} \mathtt{ror}$. So by the Comparability Context Lemma, $\mathtt{lor} \sim_{\text{obs}} \mathtt{ror}$.

We conclude that there is no term $P$ defining an or-separator; otherwise $\mathtt{lor}$ and $\mathtt{ror}$ yield distinct observations in the context $(P\ [\cdot])$, contradicting the fact that $\mathtt{lor} \sim_{\text{obs}} \mathtt{ror}$.

However, we can define an or-separator *detector* as follows:

$$D \stackrel{\text{def}}{\equiv} \lambda x.\mathtt{cond}\,(x\ \mathtt{lor})\,(\mathtt{cond}\,(x\ \mathtt{ror})\,\Omega^o\,\mathtt{tt})\,\Omega^o.$$

By Lemma 2.10,

$$[\![D]\!](f) = \begin{cases} \mathtt{tt} & \text{if } f \text{ is an or-separator,}\\ \perp & \text{otherwise.}\end{cases}$$

Now $[\![D]\!] \neq [\![\lambda x.\Omega^o]\!]$, since they differ exactly on arguments that are or-separators. But since or-separators are not definable by terms, $D$ and $[\![\lambda x.\Omega^o]\!]$ are applicatively congruent. Then by the Approximation Context Lemma, they are observationally congruent, contradicting equational full abstraction. □

COROLLARY 3.8. *If a bistable model with Booleans is adequate for a conservative extension of PCF that satisfies the Comparability and Approximation Context Lemmas, then the model is not equationally fully abstract.*

*Proof.* Every bistable model with Booleans contains an or-separator *orsep*, defined as follows:

$$orsep(g) = \begin{cases} tt & \text{if } g = lor,\\ \mathit{ff} & \text{if } g = ror,\\ \perp & \text{otherwise.}\end{cases}$$

□

COROLLARY 3.9 ([23]). *The PCF equations valid in the bistable model do not include those valid in the cpo model.*

*Proof.* Just note that $\mathcal{C}[\![D]\!] = \mathcal{C}[\![\lambda x.\Omega^o]\!]$, but $\mathcal{B}[\![D]\!] \neq \mathcal{B}[\![\lambda x.\Omega^o]\!]$, where $\mathcal{B}[\![\cdot]\!]$ is the bistable model of [6]. □

The PCF-like languages, defined in the next section, do not satisfy the Comparability Context Lemma. In fact, an or-separator constant can defined through the following PCF-like rules:

$$\begin{aligned}\mathtt{orsep}(f) &\to \mathtt{cond}\,(f\ \mathtt{tt}\ \Omega^o)\,(\mathtt{cond}\,(f\ \mathtt{ff}\ \mathtt{tt})\,(\mathtt{cond}\,(f\ \mathtt{ff}\ \mathtt{ff})\,\mathtt{tt}\,\Omega^o)\,\Omega^o)\,\Omega^o,\\ \mathtt{orsep}(f) &\to \mathtt{cond}\,(f\ \Omega^o\ \mathtt{tt})\,(\mathtt{cond}\,(f\ \mathtt{tt}\ \mathtt{ff})\,(\mathtt{cond}\,(f\ \mathtt{ff}\ \mathtt{ff})\,\mathtt{ff}\,\Omega^o)\,\Omega^o)\,\Omega^o.\end{aligned}$$

Thus we will have to restrict the class of rules we consider if we wish to apply Theorem 3.7. The *consistent* rules of Bloom [11] are an important, natural candidate for the restricted class. We do not know whether the Comparability Context Lemma holds for them. However, we can prove that an or-separator is not definable in consistent systems by a method involving a notion of comparability based on logical relations, as we indicate at the end of the next section.

**4. PCF-like rewrite systems.** Symbolic evaluators for PCF terms are often presented as term rewriting systems. In this section, we give the basic definitions for such systems, and give our criteria for calling such a system "PCF-like". Our evaluator for PCF is given in Appendix A.

A *rewrite rule* is a pair $l \to r$ of terms of the same type, such that the free variables of the right-hand side $r$ are included in those of the left-hand side $l$. We write $M \xrightarrow{\Delta}_{\pi} N$ if for some subterm $\Delta$ of $M$, $\Delta \to \Delta'$ is an instance of the rule $\pi$, and $N$ is obtained from $M$ by replacing $\Delta$ with $\Delta'$. We will omit $\Delta$ or $\pi$ as convenient.

Since all of our languages are simply typed $\lambda$-calculi, we will always include $\beta$-reduction in the rewrite rules of the language. Additionally, we may specify some set $\Theta$ of *$\delta$-rules* defining the behavior of the constants. Together, $\Theta$ and $\beta$ define the *rewriting relation* $\to_{\Theta,\beta}$ on the language $\mathcal{L}$. We omit $\Theta$ and $\beta$ when they can be recovered from context.

The $\delta$-rules of PCF have a particularly simple form:

DEFINITION 4.1. *A* linear ground $\delta$-rule *is a rewrite rule of the form*

$$\delta m_1 m_2 \cdots m_n \to P,$$

*where each $m_i$ is either a ground constant $c_i$ or a variable $x_i$. The variables $x_i$ must be distinct. A* PCF-like rewrite system *is a language $\mathcal{L}$ together with a set $\Theta$ of linear ground $\delta$-rules on the constants of $\mathcal{L}$.*

Note that this definition of "PCF-like" is meant to be generous. In particular, although the system for pure, unextended PCF is both single-valued—every term reduces to at most one constant—and confluent, PCF-like systems in general may be multiple-valued and nonconfluent.

An interesting example of a multiple-valued PCF-like system arises in [10]. There, Bloom defines an extension of PCF that is both fully abstract and denotationally universal for the lattice model of PCF. The key to the construction amounts to the addition of operators $\top : o$ and $\texttt{join} : o \to o \to o$ with rules

$$\begin{array}{rcll} \texttt{join}\, x\, y & \to & x, & \\ \texttt{join}\, x\, y & \to & y, & \\ \texttt{join}\, \underline{n_1}\, \underline{n_2} & \to & \top, & \underline{n_1} \neq \underline{n_2}, \\ \top & \to & \underline{n}, & n \geq 0. \end{array}$$

Plotkin [31] extended parallel PCF by an existential operator, $\exists : (\iota \to o) \to o$, to achieve a language that is fully abstract and denotationally universal for the cpo model. There, $\exists$ is defined by the deductive rules

$$\frac{p\underline{n} \twoheadrightarrow \texttt{tt}}{\exists p \to \texttt{tt}}, \qquad \frac{p\Omega \twoheadrightarrow \texttt{ff}}{\exists p \to \texttt{ff}},$$

where $\twoheadrightarrow$ is the reflexive transitive closure of $\to$. Because he wanted to be able to specify constants like $\exists$, Bloom [11] introduced *observation calculi* as a definition of "PCF-like" deductive rules.

But note that if we give up confluence, it is possible to define an $\exists$ constant while remaining in a term rewriting discipline.[4] One such definition was given in the introduction; we provide here a second implementation, which uses the parallel-or combinator $\texttt{por}$.

$$\begin{array}{rcl} \exists p & \to & \texttt{por}\,(p0)\,\big(\exists(\lambda x.p(\texttt{succ}\, x))\big), \\ \exists p & \to & \texttt{cond}\,(p\Omega)\,\texttt{tt}\,\texttt{ff}. \end{array}$$

[4] The question of whether $\exists$ can be defined in a *confluent* term rewriting discipline remains open.

This kind of rewriting is more straightforward, but actually as powerful as the deductive discipline.

Since PCF-like systems are not confluent in general, we will not be able to use confluence in our proof of the Context Lemma. Instead we will rely on a *standardization theorem*, which states that if a term $M$ rewrites to a term $N$, then there is a "standard" reduction from $M$ to $N$. Thus we only need consider these standard reductions in our proof.

Typically, the standard reductions are a class of reductions with a particularly nice structure. For instance, in the pure, typed $\lambda$-calculus, a standard reduction is one in which redexes are contracted from left to right.

The definition of standard reductions in PCF-like rewrite systems is more complicated because they admit the upwards creation of redexes, *cf.* [20]. However, there is a simple inductive characterization of those standard reductions that end at a ground constant. This will be sufficient to follow the proof of the Context Lemma given in the next section, so we defer the general definition of standard reductions, and the proof of the Standardization Theorem, to Appendix C.

Before defining the standard reductions to ground constants, we introduce some useful notation. Consider the set of indices

$$\{\, i \mid m_i \text{ is a constant } c_i \text{ in rule } \theta : \delta\vec{m} \to P \,\}.$$

These indices identify what we call the *critical* arguments of $\theta$, since the rule $\theta$ applies to a term $\delta\vec{Q}$ iff $Q_i \equiv c_i$ for $i$ in the set. For expository purposes it will be convenient to separate the critical and non-critical arguments of a constant $\delta$ (relative to some linear ground $\delta$-rule $\theta$).

NOTATION 4.2. *Let $\theta : \delta\vec{m} \to P$ be a linear ground $\delta$-rule with $j$ critical arguments and $k$ non-critical arguments. Then for vectors $\vec{A} \equiv A_1 \cdots A_j$ and $\vec{B} \equiv B_1 \cdots B_k$, we let*

$$\delta_\theta\langle \vec{A}, \vec{B}\rangle \quad \stackrel{\text{def}}{\equiv} \quad \delta\vec{Q},$$

*where $\vec{Q}$ is the interleaving of $\vec{A}$ and $\vec{B}$ such that the $A_i$'s appear at the critical indices of $\vec{Q}$. We drop the subscript $\theta$ when it can be recovered from context.*

Note that we do not require that $\delta\vec{Q}$ be an instance of $\delta\vec{m}$; we will want to use the $\delta\langle\cdot,\cdot\rangle$ notation on terms that we anticipate becoming $\theta$-redexes over the course of a reduction.

In this notation, we write linear ground $\delta$-rules as

$$\theta \quad : \quad \delta\langle \vec{c}, \vec{x}\rangle \to P$$

or even

$$\theta \quad : \quad \delta\langle \vec{c}, \vec{x}\rangle \to P(\vec{x})$$

when we wish to make the dependence of $P$ on $\vec{x}$ explicit.

DEFINITION 4.3. *The* standard reductions to ground constants *in a PCF-like rewrite system are defined inductively as follows. We will write $M \twoheadrightarrow_s c$ for a standard reduction of a term $M$ to a ground constant $c$.*

- *If $c$ is a ground constant, then the 0-step reduction $c \twoheadrightarrow c$ is standard.*

- *If $M_1, M_2, \ldots, M_n$ are terms, and $c$ is a ground constant, then a reduction*

$$\begin{array}{rcl}(\lambda x M_1) M_2 M_3 \cdots M_n & \to_\beta & M_1[x := M_2] M_3 \cdots M_n \\ & \twoheadrightarrow_s & c\end{array}$$

  *is standard.*
- *If $C_1, C_2, \ldots, C_n, \vec{D}, \vec{E}$ are terms, and $c, c_1, c_2, \ldots, c_n$ are ground constants, then a reduction of the following form is standard:*

$$\begin{array}{llcl}\boldsymbol{\sigma}_1 : & \delta_\theta \langle C_1 C_2 \cdots C_n, \vec{D}\rangle \vec{E} & \twoheadrightarrow & \delta_\theta \langle c_1 C_2 \cdots C_n, \vec{D}\rangle \vec{E} \\ \boldsymbol{\sigma}_2 : & & \twoheadrightarrow & \delta_\theta \langle c_1 c_2 \cdots C_n, \vec{D}\rangle \vec{E} \\ \vdots & & \twoheadrightarrow & \cdots \\ \boldsymbol{\sigma}_n : & & \twoheadrightarrow & \delta_\theta \langle c_1 c_2 \cdots c_n, \vec{D}\rangle \vec{E} \\ & & \to_\theta & P_\theta(\vec{D})\vec{E} \\ & & \twoheadrightarrow_s & c,\end{array}$$

  *where for $1 \leq i \leq n$, the subreduction $\boldsymbol{\sigma}_i$ consists of a standard reduction from the subterm $C_i$ to the ground constant $c_i$.*

THEOREM 4.4 (STANDARDIZATION). *For any PCF-like rewrite system, if $M \twoheadrightarrow N$, then there is a standard reduction $M \twoheadrightarrow_s N$.*

Note that if we require our rules to be *non-overlapping*, then they are a special case of *orthogonal* rewrite systems, for which both confluence and standardization have been known for some time [20]. Similarly, confluence and standardization have been known for the systems of Bloom [11], which restrict our systems by allowing only so-called *consistent* overlaps at the root. However, it is not clear whether $\exists$ can be defined in such systems, and we certainly lose the ability to define interesting non-confluent systems, such as PCF extended with `join`.

**5. The Context Lemma.** Once standardization is known, the Context Lemma can be proved by a straightforward adaptation of Bloom's proof for his observation calculi [11]. First we recall the following basic facts about substitutions.

LEMMA 5.1 (SUBSTITUTION LEMMA). *If $x \not\equiv y$ and $y \notin \mathrm{FV}(L)$, then*

$$M[x := L]\big[y := N[x := L]\big] \quad \equiv \quad M[y := N][x := L].$$

LEMMA 5.2. *If $x \notin \mathrm{FV}(P)$, then*

$$P\big[\vec{y} := \vec{N}[x := M]\big] \quad \equiv \quad (P[\vec{y} := \vec{N}])[x := M].$$

The Context Lemma will follow immediately from this next result.

LEMMA 5.3. *Suppose $C$ is a ground term, $c$ is a ground constant, $M$ and $N$ are closed terms of the same type, and $M \sqsubseteq_{\mathrm{app}} N$. If $C[x := M] \twoheadrightarrow c$, then $C[x := N] \twoheadrightarrow c$.*

*Proof.* By Standardization, $C[x := M] \twoheadrightarrow_s c$. We show $C[x := N] \twoheadrightarrow c$ by induction on the length of the reduction $C[x := M] \twoheadrightarrow_s c$.

1. The only reduction $C[x := M] \twoheadrightarrow_s c$ of length zero is $c \twoheadrightarrow c$. Then one of the following holds:
   (a) $C \equiv c$. Then clearly $C[x := N] \equiv c \twoheadrightarrow c$.
   (b) $C \equiv x$ and $M \equiv c$. Here $C[x := N] \twoheadrightarrow c$ because $M \sqsubseteq_{\mathrm{app}} N$.

For the induction, we consider subcases on the form of $C$.

2. $C \equiv (\lambda y C_1)C_2 \cdots C_n$. Assume $x \not\equiv y$ (the case $x \equiv y$ is similar). Since $M$ is closed, we have

$$C[x := M] \quad \equiv \quad \big(\lambda y(C_1[x := M])\big)C_2[x := M] \cdots C_n[x := M].$$

Then the reduction $C[x := M] \twoheadrightarrow_s c$ is of the form

$$\begin{array}{rcl} C[x := M] & \equiv & \big(\lambda y(C_1[x := M])\big)C_2[x := M] \cdots C_n[x := M] \\ & \rightarrow_\beta & (C_1[x := M])\big[y := C_2[x := M]\big]C_3[x := M] \cdots C_n[x := M] \\ & \twoheadrightarrow_s & c. \end{array}$$

By the Substitution Lemma,

$$(C_1[x := M])\big[y := C_2[x := M]\big] \quad \equiv \quad (C_1[y := C_2])[x := M],$$

so our reduction can be rewritten

$$\begin{array}{rcl} C[x := M] & \equiv & \big((\lambda y C_1)C_2 \cdots C_n\big)[x := M] \\ & \rightarrow_\beta & \big((C_1[y := C_2])C_3 \cdots C_n\big)[x := M] \\ & \twoheadrightarrow_s & c. \end{array}$$

Now by $\beta$-reduction, the fact that $N$ is closed, and the Substitution Lemma,

$$\begin{array}{rcl} C[x := N] & \equiv & \big((\lambda y C_1)C_2 \cdots C_n\big)[x := N] \\ & \rightarrow_\beta & \big((C_1[y := C_2])C_3 \cdots C_n\big)[x := N]. \end{array}$$

And by induction,

$$\big((C_1[y := C_2])C_3 \cdots C_n\big)[x := N] \quad \twoheadrightarrow \quad c.$$

Thus we have a reduction $C[x := N] \twoheadrightarrow c$ as desired.

3. $C \equiv \delta C_1 \cdots C_n$. Then the reduction $C[x := M] \twoheadrightarrow_s c$ must contract the head $\delta$ by some rule $\theta : \delta_\theta\langle \vec{d}, \vec{y}\rangle \rightarrow P(\vec{y})$ (where each $d_i$ is a ground constant). Accordingly, we rewrite $C$ as

$$C \quad \equiv \quad \delta_\theta\langle \vec{D}, \vec{E}\rangle \vec{F}.$$

Then the reduction $C[x := M] \twoheadrightarrow_s c$ is of the form

$$\begin{array}{rcl} C[x := M] & \equiv & \delta_\theta\langle \vec{D}[x := M], \vec{E}[x := M]\rangle \vec{F}[x := M] \\ & \twoheadrightarrow & \delta_\theta\langle \vec{d}, \vec{E}[x := M]\rangle \vec{F}[x := M] \\ & \rightarrow_\theta & P(\vec{E}[x := M])\vec{F}[x := M] \\ & \twoheadrightarrow_s & c, \end{array}$$

where each $D_i[x := M] \twoheadrightarrow_s d_i$ in turn. By Lemma 5.2,

$$P(\vec{E}[x := M]) \quad \equiv \quad P(\vec{E})[x := M],$$

so the reduction can be rewritten

$$\begin{array}{rcl} C[x := M] & \equiv & (\delta_\theta\langle \vec{D}, \vec{E}\rangle \vec{F})[x := M] \\ & \twoheadrightarrow & (\delta_\theta\langle \vec{d}, \vec{E}\rangle \vec{F})[x := M] \\ & \to_\theta & (P(\vec{E})\vec{F})[x := M] \\ & \twoheadrightarrow_s & c. \end{array}$$

Again by Lemma 5.2,

$$P(\vec{E}[x := N])\vec{F}[x := N] \quad \equiv \quad (P(\vec{E})\vec{F})[x := N].$$

And by induction, $(P(\vec{E})\vec{F})[x := N] \twoheadrightarrow c$, and $D_i[x := N] \twoheadrightarrow d_i$. Thus we have found a reduction

$$\begin{array}{rcl} C[x := N] & \equiv & (\delta_\theta\langle \vec{D}, \vec{E}\rangle \vec{F})[x := N] \\ & \twoheadrightarrow & (\delta_\theta\langle \vec{d}, \vec{E}\rangle \vec{F})[x := N] \\ & \to_\theta & (P(\vec{E})\vec{F})[x := N] \\ & \twoheadrightarrow & c. \end{array}$$

4. $C \equiv xC_1 \cdots C_n$. Then consider the term

$$C' \stackrel{\text{def}}{\equiv} MC_1 \cdots C_n.$$

Note that $C[x := M] \equiv C'[x := M]$, so $C'[x := M] \twoheadrightarrow_s c$. Moreover $C'$ must be of a form considered in the two previous cases, and so by the previous argument we conclude $C'[x := N] \twoheadrightarrow c$. Now consider the applicative *context*

$$C''[\cdot] \stackrel{\text{def}}{\equiv} [\cdot]C_1[x := N] \cdots C_n[x := N].$$

Since $C''[M] \equiv C'[x := N]$, we have $C''[M] \twoheadrightarrow c$. Finally, $M \sqsubseteq_{\text{app}} N$ implies $C''[N] \twoheadrightarrow c$; and

$$\begin{array}{rcl} C''[N] & \equiv & NC_1[x := N] \cdots C_n[x := N] \\ & \equiv & C[x := N], \end{array}$$

so $C[x := N] \twoheadrightarrow c$.

Note that we need not consider the case $C \equiv yC_1 \cdots C_n$, where $y \not\equiv x$, since then $C[x := M]$ can never reduce to a ground constant. □

THEOREM 5.4 (APPROXIMATION CONTEXT LEMMA). *In any PCF-like rewrite system,*

$$M \sqsubseteq_{\text{obs}} N \quad \textit{iff} \quad M \sqsubseteq_{\text{app}} N$$

*for all closed terms $M$ and $N$.*

*Proof.*

($\Longrightarrow$) Trivial.

($\Longleftarrow$) It is sufficient to show the following: for all ground contexts $C[\cdot]$ and ground constants $c$, if $C[M] \twoheadrightarrow c$, then $C[N] \twoheadrightarrow c$.

Remember that the action of placing a term into the "holes" of a context differs from substitution only in that free variables of the term can be captured. But $M$ and $N$ are closed, with no free variables to capture; so for any context $C[\cdot]$,

$$\begin{array}{rrcl} & C[M] & \equiv & (C[x])[x := M], \\ \text{and} & C[N] & \equiv & (C[x])[x := N], \end{array}$$

where $x$ is a fresh variable. So by Lemma 5.3, if $C[M] \twoheadrightarrow c$, then $C[N] \twoheadrightarrow c$ as well. $\Box$

We now have immediately from Corollary 3.3:

THEOREM 5.5. *Every stable function model with Booleans that is adequate for a conservative extension of PCF defined by PCF-like rewrite rules is not equationally fully abstract.*

We remark that a simple sufficient condition to ensure that an extension of PCF by PCF-like rules is conservative is that $\delta$-rules whose left-hand sides involve no new (non-PCF) constants must be exactly the rules of PCF.

Because we are unable to prove a Comparability Context Lemma for consistent PCF-like rewrite rules, Corollary 3.8 cannot be applied. Nevertheless, our analysis of comparability can be extended to show:

THEOREM 5.6. *Every bistable model with Booleans that is adequate for a conservative extension of PCF defined by consistent PCF-like rewrite rules is not equationally fully abstract.*

**6. Conclusions and Future Work.** We have extended the metatheory of term rewriting semantics for simply typed $\lambda$-calculi and have shown that certain denotational models, in particular those based on stable and strongly stable domains, cannot be fully abstract for such operational semantics. Our proof exploits the lack of order-extensionality in these domains.

We conjecture that our methods and results will extend to untyped versions of PCF-like languages. Extensions to lazy and call-by-value languages also seem plausible, though with more difficulties, since higher order terms now yield observations and the notion of lazy model is more technical.

The category of sequential algorithms [7] is technically not a model in our sense, but is like the stable model in that it is a Cartesian Closed Category with partially ordered function objects that are not pointwise ordered. We believe that with some minor modifications our results will apply to it as well.

Although we are able to show the failures of some order-extensional models, like the bistable models, the extensional embedding methods of [13] offer a more sophisticated way to restore order-extensionality which, for example, guarantees that the theory of the extensionally embedded models includes that of cpo's. We do not know whether these models can avoid the kind of failure of full abstraction that we have identified.

Finally, the work of Cartwright, Curien, and Felleisen raises some interesting issues [15]. They extend PCF with a higher-order operator, `catch`, that distinguishes functions based on their sequential evaluation: when applied to a functional expression, `catch` returns the index of the argument that the expression would evaluate first, if it were applied. This is enough, for example, to distinguish `True` and `True!`, since `True!` uses its (sole) argument first, while `True` does not use its argument at all.

The resulting language is fully abstract with an extension of the sequential algorithms model; however, it does not satisfy the Context Lemma (we still have `True!` $\sqsubseteq_{\mathrm{app}}$ `True`, but no longer `True!` $\sqsubseteq_{\mathrm{obs}}$ `True`). But by further adding "errors" to the language and model, they recover the Context Lemma, and retain full abstraction.

There are two reasons that this does not contradict our results. First, although extensional, their language is not defined by PCF-like rules; whether it can be defined by such rules is an interesting open question. Second, extensionality is only achieved by adding "error" elements to the ground types of the model, and thus their model

$$\begin{array}{rcll}
\mathtt{tt},\mathtt{ff} & : & o & \\
\underline{n} & : & \iota & \text{for each integer } n \geq 0 \\
\mathtt{succ},\mathtt{pred} & : & \iota \to \iota & \\
\mathtt{zero?} & : & \iota \to o & \\
\mathtt{cond}_o & : & o \to o \to o \to o & \\
\mathtt{cond}_\iota & : & o \to \iota \to \iota \to \iota & \\
\mathtt{Y}_\sigma & : & (\sigma \to \sigma) \to \sigma & \text{for each type } \sigma
\end{array}$$

FIG. 1. *Constants of PCF*

does not meet our technical definition of "model with Booleans".

**Appendix A. PCF.** Because we will work with both PCF and its extensions, we give the general definitions for simply typed $\lambda$-calculi. A language is parameterized by its ground types and typed constants; for instance, PCF's ground types are the Booleans $o$ and the numerals $\iota$, and its constants are listed in Figure 1.

The set of *types* of the language is the least set containing the ground types and $(\sigma \to \tau)$ for types $\sigma$ and $\tau$. The set of *first-order* types is the least set containing the ground types and $(\sigma \to \tau)$ for ground types $\sigma$ and first-order types $\tau$.

The typed *terms* of the language are defined inductively:

- A constant $\delta^\sigma$ is a term of type $\sigma$.
- A variable $x^\sigma$ is a term of type $\sigma$.
- If $M$ is a term of type $(\sigma \to \tau)$ and $N$ is a term of type $\sigma$, then $(MN)$ is a term of type $\tau$.
- If $M$ is a term of type $\tau$, then $(\lambda x^\sigma M)$ is a term of type $(\sigma \to \tau)$.

We omit types and parentheses whenever possible, adopting the standard conventions of association: application associates to the left, and "$\to$" associates to the right. We will use $M, N, P, \ldots$ to denote arbitrary terms; $x, y, z, \ldots$ to denote arbitrary variables; and $\sigma, \tau, \gamma, \ldots$ to denote arbitrary types. $\delta$ will always denote a constant, and $c$ will always be a ground constant. The binary relation symbol $\equiv$ denotes syntactic equality.

Free and bound variables are defined as usual, and we consider terms that are identical modulo a change of bound variables to be syntactically identical. A term is *closed* if it has no free variables; otherwise it is *open*. A *program* is a closed term of ground type.

A *substitution* is a typed-respecting mapping of variables to terms. Substitutions are extended to terms as usual (taking care to avoid capture of free variables), and are written postfix, so that $M\rho$ is the application of the substitution $\rho$ to the term $M$. We call $M\rho$ an *instance* of $M$. If $\vec{x} \equiv x_1, \ldots, x_n$ and $\vec{N} \equiv N_1, \ldots, N_n$, then $[\vec{x} := \vec{N}]$ is the substitution that maps each $x_i$ to $N_i$ (simultaneously), and is the identity otherwise. A special case is $[x := N]$, so that $M[x := N]$ is the result of substituting $N$ for $x$ in $M$. Sometimes we write $M \equiv M(\vec{x})$, with the intent that $M(\vec{N}) \equiv M[\vec{x} := \vec{N}]$.

A *context* $C[\cdot]$ is a term with some "holes". $C[M]$ denotes the result of putting $M$ into the holes of $C[\cdot]$, which may cause free variables of $M$ to become bound. We say $C[\cdot]$ is a *program context* for $M$ if $C[M]$ is a closed term of ground type.

The interpreter of the language is defined via a rewrite system; any set of $\delta$-rules, together with the classical rule $(\beta)$, induces the one-step reduction relation $\to$. The relation $\twoheadrightarrow$ is the reflexive transitive closure of $\to$. Figure 2 gives the $\delta$-rules for PCF.

$$
\begin{array}{rcl}
\mathtt{cond}\,\mathtt{tt}\; x\; y & \rightarrow & x \\
\mathtt{cond}\,\mathtt{ff}\; x\; y & \rightarrow & y \\
\\
\mathtt{zero?}\,\underline{0} & \rightarrow & \mathtt{tt} \\
\mathtt{zero?}\,\underline{n+1} & \rightarrow & \mathtt{ff} \\
\\
\mathtt{succ}\,\underline{n} & \rightarrow & \underline{n+1} \\
\\
\mathtt{pred}\,\underline{0} & \rightarrow & \underline{0} \\
\mathtt{pred}\,\underline{n+1} & \rightarrow & \underline{n} \\
\\
\mathtt{Y}\, f & \rightarrow & f(\mathtt{Y}\, f)
\end{array}
$$

FIG. 2. *Rewrite rules for PCF*

**Appendix B. Simply Typed Models.** Here we develop the general framework for function-based models of simply typed $\lambda$-calculi.

A *type frame* $\{[\![\sigma]\!]\}$ is collection of sets indexed by type such that $[\![\sigma \rightarrow \tau]\!]$ is a set of functions from $[\![\sigma]\!]$ to $[\![\tau]\!]$. The sets $[\![\sigma]\!]$ are called *domains*, and the elements of each $[\![\sigma]\!]$ are called *meanings* or *values* of type $\sigma$.

Since our discussion focuses on issues of adequacy and full abstraction, we also require the following:

- there is a partial order $\sqsubseteq_\sigma$ associated with each domain $[\![\sigma]\!]$;
- the functions of $[\![\sigma \rightarrow \tau]\!]$ are monotone with respect to the orderings $\sqsubseteq_\sigma$ and $\sqsubseteq_\tau$;
- and the relation $\sqsubseteq_{\sigma\rightarrow\tau}$ refines the pointwise relation on functions $f, g \in [\![\sigma \rightarrow \tau]\!]$, *i.e.*,

$$f \sqsubseteq_{\sigma\rightarrow\tau} g \quad \text{implies} \quad f(d) \sqsubseteq_\tau g(d) \text{ for all } d \in [\![\sigma]\!].$$

The last two conditions say that function application is monotone in both arguments; this implies that models, defined below, are compositional.

An *environment* is a type-respecting mapping from variables to values. If $\rho$ is an environment, then the environment $\rho[x := d]$ is $\rho$ with the value of $x$ updated to $d$:

$$\rho[x := d](y) \quad = \quad \begin{cases} d & \text{if } y \equiv x, \\ \rho(y) & \text{otherwise.} \end{cases}$$

An *interpretation* is a type-respecting mapping from constants to values. For a given type frame $\{[\![\sigma]\!]\}$ and interpretation $\mathcal{I}$ we can try to define a *model*, $[\![\cdot]\!]$, that is a mapping from each term to a meaning with respect to an environment, satisfying the following conditions:

$$[\![\delta]\!]\rho = \mathcal{I}(\delta) \tag{B.1}$$

$$[\![x]\!]\rho = \rho(x) \tag{B.2}$$

$$[\![(MN)]\!]\rho = ([\![M]\!]\rho)([\![N]\!]\rho) \tag{B.3}$$

$$([\![\lambda x M]\!]\rho)(d) = [\![M]\!]\rho[x := d] \tag{B.4}$$

Implicit in condition (B.4) is the requirement that the function defined to be ($[\![\lambda x M]\!]\rho$) must be an element in an element of the type frame. In other words, a model is a type frame that is closed under lambda-definability. Such closure certainly does not hold for all type frames (*cf.* [26]).

The meaning of a closed term is the same in any environment:

$$[\![M]\!]\rho \quad = \quad [\![M]\!]\rho'$$

for all closed $M$ and arbitrary $\rho, \rho'$. Therefore we sometimes write $[\![M]\!]$ for the meaning of a closed term $M$, omitting the environment.

**Continuity.** We give the standard definitions for cpo's and continuous functions, then define the cpo model of PCF.

A *partial order* or *poset* is a set $D$ together with a binary relation $\sqsubseteq$ that is reflexive, transitive, and anti-symmetric. We will refer to the partial order $\langle D, \sqsubseteq\rangle$ as just $D$. A subset $X \subseteq D$ is *directed* if every finite subset of $X$ has an upper bound in $X$. A partial order $D$ is a *complete partial order* or *cpo* if it has a least element $\perp_D$ and every directed subset $X \subseteq D$ has a least upper bound $\sqcup X$. We omit the subscript $D$ in $\perp_D$ when it can be recovered from context. For any set $X$ we define the cpo $X_\perp$, with elements $X \cup \{\perp_X\}$, ordered $x \sqsubseteq y$ iff $x = y$ or $x = \perp_X$.

A function $f : D \to E$ between posets is *monotone* if $f(x) \sqsubseteq_E f(y)$ whenever $x \sqsubseteq_D y$. We say $f$ is *continuous* if it is monotone and $f(\sqcup X) = \sqcup f(X)$ for every directed $X \subseteq D$.

The set $D \to_c E$ of continuous functions from cpo $D$ to cpo $E$ is a cpo under the pointwise order $\sqsubseteq_{\mathrm{p}}$, defined as follows:

$$f \sqsubseteq_{\mathrm{p}} g \quad \text{iff} \quad f(x) \sqsubseteq_E g(x) \text{ for all } x \in D.$$

If $D$ is a cpo and $f : D \to D$ is continuous, then $f$ has a least fixed point $\mathit{fix}(f)$. The function $\mathit{fix}$ itself is continuous, which will allow us to interpret the recursion operator $\mathtt{Y}$.

Now we define the cpo model $\mathcal{C}[\![\cdot]\!]$ of PCF, based on continuous functions and cpos. First we construct a type frame with ground domains $\mathcal{C}[\![o]\!] = \{\mathit{tt}, \mathit{ff}\}_\perp$ and $\mathcal{C}[\![\iota]\!] = \{0, 1, 2, \ldots\}_\perp$, and higher-order domains $\mathcal{C}[\![\sigma \to \tau]\!] = \mathcal{C}[\![\sigma]\!] \to_c \mathcal{C}[\![\tau]\!]$. The cpo model of PCF is then the model $\mathcal{C}[\![\cdot]\!]$ associated with $\{\mathcal{C}[\![\sigma]\!]\}$ and the *standard interpretation*: the ground constants are interpreted in the obvious way; the constants $\mathtt{Y}_\sigma$ are interpreted as *least* fixed-point operators; and the interpretation of the remaining function constants is determined by the condition that the rewrite rules of Figure 2 be valid as equations.

THEOREM B.1 (PLOTKIN [31], SAZONOV [32]). *The cpo model $\mathcal{C}[\![\cdot]\!]$ is adequate but not fully abstract for PCF.*

**Stability.** If $D$ is a partial order and $X \subseteq D$, then $X$ is *bounded* or *consistent* if there is an element $y \in D$ such that $x \sqsubseteq y$ for all $x \in X$. If elements $x$ and $y$ are consistent we will write $x \uparrow y$. We say $D$ is *bounded complete* if every bounded subset $X \subseteq D$ has a least upper bound $\sqcup X$.

An element $a \in D$ is *compact* if, for every directed $X \subseteq D$ with $a \sqsubseteq \sqcup X$, there is some $x \in X$ such that $a \sqsubseteq x$. We define $\mathbf{K}D$, the *kernel* of $D$, to be the set of compact elements of $D$. The cpo $D$ is *algebraic* if, for every $x \in D$, the set $\downarrow x = \{\, a \in \mathbf{K}D \mid a \sqsubseteq x \,\}$ is directed and $\sqcup \downarrow x = x$.

The greatest lower bound of a set $X$ is denoted $\sqcap X$. A cpo is *distributive* if $x \sqcap (y \sqcup z) = (x \sqcap y) \sqcup (x \sqcap z)$ whenever $y$ and $z$ are consistent. An algebraic cpo $D$

| Function | $tt$ | $\mathit{ff}$ | $\perp$ |
|---|---|---|---|
| *True* | $tt$ | $tt$ | $tt$ |
| *False* | $\mathit{ff}$ | $\mathit{ff}$ | $\mathit{ff}$ |
| *True!* | $tt$ | $tt$ | $\perp$ |
| *False!* | $\mathit{ff}$ | $\mathit{ff}$ | $\perp$ |
| *Id* | $tt$ | $\mathit{ff}$ | $\perp$ |
| *Not* | $\mathit{ff}$ | $tt$ | $\perp$ |
| $(tt \Rightarrow tt)$ | $tt$ | $\perp$ | $\perp$ |
| $(tt \Rightarrow \mathit{ff})$ | $\mathit{ff}$ | $\perp$ | $\perp$ |
| $(\mathit{ff} \Rightarrow tt)$ | $\perp$ | $tt$ | $\perp$ |
| $(\mathit{ff} \Rightarrow \mathit{ff})$ | $\perp$ | $\mathit{ff}$ | $\perp$ |
| *Bot* | $\perp$ | $\perp$ | $\perp$ |

FIG. 3. *Boolean functions*

*has property I* if $\downarrow a$ is finite for each $a \in \mathbf{K}D$. A *dI-domain* is a distributive, bounded complete cpo that has property I.

A continuous function $f$ between dI-domains is *stable* if whenever $x \uparrow y$, we have that $f(x \sqcap y) = f(x) \sqcap f(y)$. We let $D \rightarrow_s E$ be the set of stable functions between dI-domains $D$ and $E$. As noted in [6], $D \rightarrow_s E$ ordered pointwise is *not* a dI-domain; accordingly we define the stable ordering $\sqsubseteq_{\mathrm{s}}$:

$$f \sqsubseteq_{\mathrm{s}} g \qquad \text{iff} \qquad f(x) = f(y) \sqcap g(x) \text{ whenever } x \sqsubseteq y.$$

If $D$ and $E$ are dI-domains, then $D \rightarrow_s E$ is a dI-domain under the stable order.

It must be noted that the stable order is quite different from the pointwise order. For instance, consider the monotone Boolean functions, listed in Figure 3. These functions are both continuous and stable, and so they are elements of both the continuous and stable type frames. However, the stable ordering of $o \rightarrow o$ (Figure 5) is different from its pointwise ordering (Figure 4). In particular, consider *True*, the constant $tt$ function, and *True!*, the strict constant $tt$ function. Although $\mathit{True!} \sqsubseteq_{\mathrm{p}} \mathit{True}$, we have $\mathit{True!} \not\sqsubseteq_{\mathrm{s}} \mathit{True}$, since $\perp \sqsubseteq_{\mathrm{s}} tt$ but

$$\mathit{True!}(\perp) = \perp \quad \neq \quad tt = (\mathit{True!}(tt) \sqcap \mathit{True}(\perp)).$$

(It is this that permits the existence of the function *truesep* that was needed in Corollary 3.3.)

Nevertheless, a stable model $\mathcal{S}[\![\cdot]\!]$ of PCF, based on dI-domains and stable functions, can be defined in much the same way as the cpo model. The ground domains $\mathcal{S}[\![o]\!]$ and $\mathcal{S}[\![\iota]\!]$ of the stable type frame are identical to the ground domains of the cpo model. At higher types, however, we use stable functions: $\mathcal{S}[\![\sigma \rightarrow \tau]\!] = \mathcal{S}[\![\sigma]\!] \rightarrow_s \mathcal{S}[\![\tau]\!]$. Then we let $\mathcal{S}[\![\cdot]\!]$ be the model associated with the stable type frame and the (stable) standard interpretation (*cf.* the interpretation of the cpo model).

THEOREM B.2 (BERRY [6]). *The stable model $\mathcal{S}[\![\cdot]\!]$ is adequate but not fully abstract for PCF.*

## Appendix C. Standard reductions in PCF-like rewrite systems.

**C.1. Preliminaries.** This appendix gives a full definition of standard reductions and proof of the Standardization Theorem. In this section we sketch out some of the basic terminology of rewriting systems. Section C.2 introduces descendants, which

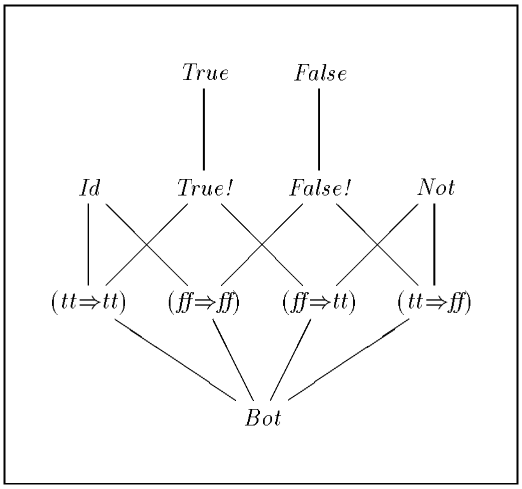


FIG. 4. *Pointwise ordering of* $o \to o$

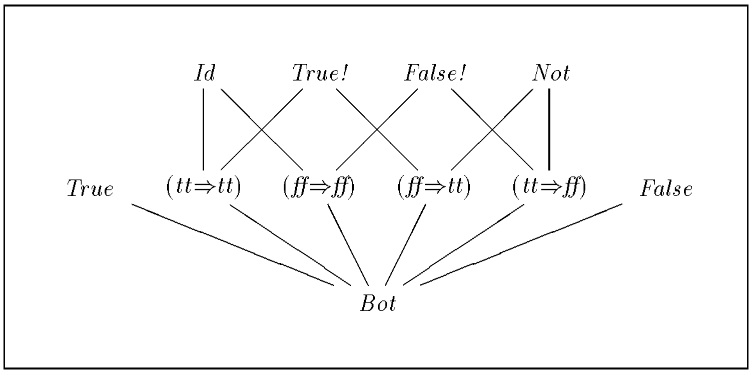


FIG. 5. *Stable ordering of* $o \to o$

allow us to trace subterms from step to step in a reduction. In §C.3 we show that a very weak form of confluence holds for PCF-like systems; this property will be essential in proving the Standardization Theorem. Section C.4 introduces labelled rewrite systems, and proves that they are strongly normalizing. The labelled systems will be used in the proof of Standardization. The standard reductions are defined in §C.5, and Standardization is proved in §C.6. The proof is a variation of Klop's proof for the pure $\lambda$-calculus [24], and involves a rewriting system on reductions. The system successively rewrites non-standard reduction paths to "more standard" paths; Standardization is proved by showing that the system is strongly normalizing, and that normal forms are standard reductions.

Our presentation of the machinery used to state and prove Standardization is necessarily brief. Much of the material is covered in more depth in standard references [4, 24]. Throughout we will work with a PCF-like rewrite system given by a language, $\mathcal{L}$, and set, $\Theta$, of linear ground $\delta$-rules.

We assume that the reader is familiar with the following terminology. The notation $M \subset N$ denotes that $M$ is a *subterm* of $N$. A subterm may appear several times in a term; multiple *occurrences* of a subterm can be distinguished by their *paths*, which specify the exact position of a subterm inside the term. When we speak of a subterm $M \subset N$ we implicitly mean a particular occurrence of $M$ in $N$; the disambiguating paths are omitted.

Note that $M \to N$ iff there is an instance $\Delta \to \Delta'$ of a rule $\pi$ such that $\Delta \subset M$, and $N$ is obtained from $M$ by replacing $\Delta$ with $\Delta'$. We will write $M \xrightarrow{\Delta}_\pi N$ in this case, and we call $\Delta$ a *($\pi$)-redex* and $\Delta'$ its *($\pi$)-contractum.*

A *reduction (path)* $\boldsymbol{\sigma}$ is a sequence

$$\boldsymbol{\sigma} : M_1 \xrightarrow{\Delta_1}_{\pi_1} M_2 \xrightarrow{\Delta_2}_{\pi_2} M_3 \xrightarrow{\Delta_3}_{\pi_3} \cdots .$$

We will use $\boldsymbol{\sigma}, \boldsymbol{\tau}, \ldots$ to refer to reduction paths. Two reductions are *coinitial* if they start in the same term, and *cofinal* if they end in the same term.

**C.2. Descendants.** Consider some possible effects of a reduction $M \to N$ on a subterm $\Delta \subset M$:

- $\Delta$ could be erased, as in $(\lambda x.y)\Delta \to y$.
- $\Delta$ could be copied to some instances in $N$, as in $(\lambda x.\delta xx)\Delta \to \delta\Delta\Delta$.
- $\Delta$ could be left untouched and in its original position, as in $\Delta((\lambda x.x)y) \to \Delta y$.
- The contracted redex might occur within $\Delta$, transforming it into a syntactically different subterm in the same position.

In order to define and prove standardization, we will need to speak precisely about these cases, so we introduce *descendants*, which let us track a subterm throughout a reduction. We will not define descendants in their full generality, but only for certain subterms of interest. Our definition is equivalent to the standard definition [24] on those subterms.

Descendants are introduced via an annotated rewrite system derived from $\mathcal{L}$ and $\Theta$, in which some $\lambda$'s and $\delta$'s are marked with a $*$. Thus we define the language $\mathcal{L}_*$, whose symbols are those of $\mathcal{L}$, with the addition of $\lambda_*$, and $\delta_*^\sigma$ for each constant $\delta^\sigma$ of $\mathcal{L}$. The terms of $\mathcal{L}_*$ are defined inductively:

- A constant $\delta^\sigma$ or $\delta_*^\sigma$ is a term of type $\sigma$.
- A variable $x^\sigma$ is a term of type $\sigma$.
- If $M$ is a term of type $(\sigma \to \tau)$ and $N$ is a term of type $\sigma$, then $(MN)$ is a term of type $\tau$.
- If $M$ is a term of type $\tau$, then $(\lambda x^\sigma M)$ and $(\lambda_* x^\sigma M)$ are terms of type $(\sigma \to \tau)$.

The *erasure* $|M| \in \mathcal{L}$ of $M \in \mathcal{L}_*$ is obtained from $M$ by leaving out the $*$'s. Substitution for the language is defined in the obvious way (with $\lambda_*$'s binding variables just as $\lambda$'s). The rules of the new system include $\beta$ and the rule scheme $\beta_*$:

$$\beta_* : (\lambda_* x M)N \to M[x := N].$$

Similarly, the $\delta$-rules $\Theta_*$ of the system are derived from the rules $\Theta$. If $\theta$ is a rule of $\Theta$,

$$\theta : \delta\langle \vec{c}, \vec{x}\rangle \to P(\vec{x}),$$

then $\Theta_*$ contains all rules of the form $\theta'$ and $\theta_*$:

$$\begin{aligned} \theta' : \delta\langle \vec{c}', \vec{x}\rangle &\to P(\vec{x}), \\ \theta_* : \delta_*\langle \vec{c}', \vec{x}\rangle &\to P(\vec{x}), \end{aligned}$$

where $\vec{c}'$ is any vector of $\mathcal{L}_*$ ground constants such that $|\vec{c}'| \equiv \vec{c}$.

There is a strong connection between the systems. Any $\Theta_*$-reduction path $\boldsymbol{\sigma}$,

$$\boldsymbol{\sigma} : M_1 \xrightarrow{\Delta_1}_{\pi_1} M_2 \xrightarrow{\Delta_2}_{\pi_2} M_3 \xrightarrow{\Delta_3}_{\pi_3} \cdots,$$

*projects* to a $\Theta$-reduction path $|\boldsymbol{\sigma}|$:

$$|\boldsymbol{\sigma}| : |M_1| \xrightarrow{|\Delta_1|}_{|\pi_1|} |M_2| \xrightarrow{|\Delta_2|}_{|\pi_2|} |M_3| \xrightarrow{|\Delta_3|}_{|\pi_3|} \cdots.$$

Conversely, for any $M \in \mathcal{L}_*$ and $\Theta$-reduction path $\boldsymbol{\sigma} : |M| \to \cdots$, there is a unique *lift* of $\boldsymbol{\sigma}$ to a $\Theta_*$-reduction path $\boldsymbol{\sigma}' : M \to \cdots$ such that $\boldsymbol{\sigma} \equiv |\boldsymbol{\sigma}'|$.

We will be interested in tracing subterms of the form $(\lambda x.M_1)M_2$ or $\delta M_1 \cdots M_n$ throughout a reduction; that is, $\beta$-redexes and possible $\delta$-redexes. Accordingly, we introduce the following terminology. A subterm $(\lambda x.M_1)M_2$ or $\delta M_1 \cdots M_n$ of $M$ is called a *predescendant of* $M$. If $\mathcal{F}$ is a set of predescendants of $M \in \mathcal{L}$, we write $(M, \mathcal{F})$ for the $\mathcal{L}_*$ term derived from $M$ by marking the head $\lambda$ or $\delta$ of each predescendant in $\mathcal{F}$ with a $*$.

Definition C.1. *Suppose $\boldsymbol{\sigma} : M \to \cdots \to N$ is a $\Theta$-reduction path.*

(i) *If $\Delta$ is a predescendant of $M$, its set of* descendants *in $N$ relative to $\boldsymbol{\sigma}$, written $(\Delta/\boldsymbol{\sigma})$, is defined as follows.*
*Let $M' \equiv (M, \{\Delta\})$ and lift $\boldsymbol{\sigma}$ to $\boldsymbol{\sigma}' : M' \to \cdots \to N'$. If $\Delta \equiv (\lambda x M_1)M_2$ (resp. $\Delta \equiv \delta M_1 \cdots M_n$), then $(\Delta/\boldsymbol{\sigma}) \stackrel{\text{def}}{=} \mathcal{F}$, where $\mathcal{F}$ is the unique set of subterms of $N$ of the form $(\lambda x M_1')M_2'$ (resp. $\delta M_1' \cdots M_n'$), such that $N' \equiv (N, \mathcal{F})$.*

(ii) *If $\mathcal{F}$ is a set of predescendants of $M$, its descendants $\mathcal{F}/\boldsymbol{\sigma}$ are defined*

$$\mathcal{F}/\boldsymbol{\sigma} \stackrel{\text{def}}{=} \bigcup \{\, \Delta/\boldsymbol{\sigma} \mid \Delta \in \mathcal{F} \,\}.$$

(iii) *$\Delta \subset M$ is an* ancestor *of $\Delta' \subset N$ if $\Delta' \in \Delta/\boldsymbol{\sigma}$.*

For a given reduction $M_1 \to M_2 \to M_3 \to \cdots$, we will sometimes speak of descendants and ancestors for subterms of terms $M_i$ and $M_j$, where $i$ and $j$ are any indices such that $j \geq i$. We do not specify the reduction from $M_i$ to $M_j$, as it can be recovered from context.

Note C.2.

(i) *If $M \xrightarrow{\Delta} N$, then $\Delta$ has no descendants in $N$.*

(ii) *If $M \xrightarrow{\Delta}_\theta N$, where $\Delta \equiv \delta\langle \vec{c}, \vec{B}\rangle$, then no $c_i$ has a descendant in $N$.*

We mention that the following important property holds for our PCF-like systems, since it does not hold for all rewrite systems [24].

Note C.3. *If $\Delta \subset M$ and $M \to N$, then descendants of $\Delta$ in $N$ are disjoint.* Disjointness of descendants does not extend to $\twoheadrightarrow$, as we indicate here:

$$\begin{aligned}(\lambda y.(\lambda x.yx)y)(\lambda z.\delta_* z) &\to_\beta (\lambda x.(\lambda z.\delta_* z)x)(\lambda z.\delta_* z)\\ &\to_\beta (\lambda x.\delta_* x)(\lambda z.\delta_* z)\\ &\to_\beta \delta_*(\lambda z.\delta_* z).\end{aligned}$$

Definition C.4. *Suppose $M_i$ is a term in a reduction $\boldsymbol{\sigma}$,*

$$\boldsymbol{\sigma} : M_1 \xrightarrow{\Delta_1}_{\pi_1} M_2 \xrightarrow{\Delta_2}_{\pi_2} M_3 \xrightarrow{\Delta_3}_{\pi_3} \cdots.$$

(i) *We say $\Delta \subset M_i$ is $(\pi)$-*contracted (in $\boldsymbol{\sigma}$) *if for some $j \geq i$, $\Delta_j$ is a descendant of $\Delta$ and $\pi_j = \pi$.*

(ii) *We say* $\Delta \subset M_i$ *is* active (in $\boldsymbol{\sigma}$) *if there is a* $\Delta' \subset \Delta$ *that is contracted in* $\boldsymbol{\sigma}$.

Sometimes it will be useful to specify a set of subterms of some term $M$, and consider reductions from $M$ in which only those subterms are contracted. Such reductions are called *developments*. Because we work with systems in which a subterm can contract by more than one rule, our definition of developments extends the standard definition by specifying a rule for each redex contracted in a development.

DEFINITION C.5. *Suppose the following:* $\boldsymbol{\sigma}$ *is a reduction from* $M$ *to* $N$; $\mathcal{F}$ *is a set of subterms of* $M$; *and* $\Pi$ *is a mapping that takes each* $\Delta \in \mathcal{F}$ *to a rule* $\pi_\Delta$.

(i) *We call* $\boldsymbol{\sigma}$ *a* development *of* $\mathcal{F}$ *from* $M$ *by* $\Pi$, *written* $\boldsymbol{\sigma} : (M, \mathcal{F}) \overset{\Pi}{\twoheadrightarrow} N$, *if each redex* $\Delta'$ *contracted in* $\boldsymbol{\sigma}$ *is a descendant of some* $\Delta \in \mathcal{F}$, *and* $\Delta'$ *is contracted by rule* $\pi_\Delta$.

(ii) *We say a development* $\boldsymbol{\sigma}$ *is a* complete development, *written* $\boldsymbol{\sigma} : (M, \mathcal{F}) \underset{\mathrm{cpl}}{\overset{\Pi}{\twoheadrightarrow}} N$, *if* $\mathcal{F}/\boldsymbol{\sigma} = \emptyset$.

*When* $\Pi$ *is evident from context, we will omit mention of it.*

NOTE C.6. *If* $\mathcal{F}$ *is a set of* $n$ *disjoint redexes of* $M$, *then clearly all complete developments of* $\mathcal{F}$ *from* $M$ *are of length* $n$ *and are cofinal.*

**C.3. Properties related to confluence.** Note C.6 is a special case of a much stronger theorem, the Finite Developments theorem. We will not need to prove the Finite Developments theorem in its full generality; this section proves a weaker result that will be sufficient for our application.

DEFINITION C.7. *We say two* $\delta$*-redexes* $\Delta_1$ *and* $\Delta_2$ overlap *if either*

(i) *they share the same head* $\delta$, *or*

(ii) *one* $\Delta_i$ *appears as a critical argument of the other.*

*Note that in case* (ii), *the* $\Delta_i$ *must be a ground constant.*

Often, rewrite systems are constrained to avoid overlapping redexes; such systems are guaranteed to be confluent. Because we allow overlapping rules, our systems are not confluent in general. However, they do satisfy the following much weaker property, which will be essential in our proof of standardization.

LEMMA C.8. *Suppose* $\boldsymbol{\sigma}_1 : M_0 \overset{\Delta_1}{\rightarrow} M_1$ *and* $\boldsymbol{\sigma}_2 : M_0 \overset{\Delta_2}{\rightarrow} M_2$, *where* $\Delta_1$ *and* $\Delta_2$ *do not overlap. Then complete developments of* $\Delta_2/\boldsymbol{\sigma}_1$ *from* $M_1$ *and* $\Delta_1/\boldsymbol{\sigma}_2$ *from* $M_2$ *are finite and cofinal.*

*Proof.* For each of the various cases on the relative positions of $\Delta_1$ and $\Delta_2$ in $M_0$, we find a term $M_3$ that is the final term of every complete development of $\Delta_1/\boldsymbol{\sigma}_2$ and $\Delta_2/\boldsymbol{\sigma}_1$:

$$\begin{array}{ccc} M_0 & \xrightarrow{\Delta_1} & M_1 \\ \downarrow{\scriptstyle \Delta_2} & & \downarrow\!\!\downarrow{\scriptstyle \Delta_2/\sigma_1} \\ M_2 & \xrightarrow[]{\Delta_1/\sigma_2}\!\!\!\!\rightarrow & M_3 \end{array}$$

1. $\Delta_1$ and $\Delta_2$ are disjoint. Then $M_0$, $M_1$, and $M_2$ can be written

$$\begin{array}{rcl} M_0 & \equiv & \cdots\Delta_1\cdots\Delta_2\cdots, \\ M_1 & \equiv & \cdots\Delta_1'\cdots\Delta_2\cdots, \\ M_2 & \equiv & \cdots\Delta_1\cdots\Delta_2'\cdots, \end{array}$$

where $\Delta_1'$ and $\Delta_2'$ are the respective contractums of $\Delta_1$ and $\Delta_2$. Now defining

$$M_3 \stackrel{\text{def}}{\equiv} \cdots\Delta_1'\cdots\Delta_2'\cdots,$$

we see that the only complete development of $\Delta_2/\boldsymbol{\sigma}_1$ is $M_1 \xrightarrow{\Delta_2} M_3$, and the only complete development of $\Delta_1/\boldsymbol{\sigma}_2$ is $M_2 \xrightarrow{\Delta_1} M_3$, as desired.

2. $\Delta_1 \subset \Delta_2$. Then there is a unique descendant $\Delta_2'$ of $\Delta_2$ in $M_1$, and we consider three subcases.
   (a) $\Delta_2 \equiv (\lambda x.\cdots\Delta_1\cdots)N$. Then we can write $M_0$, $M_1$, and $M_2$ as

$$\begin{aligned} M_0 &\equiv \cdots((\lambda x.\cdots\Delta_1\cdots)N)\cdots,\\ M_1 &\equiv \cdots((\lambda x.\cdots\Delta_1'\cdots)N)\cdots,\\ M_2 &\equiv \cdots((\cdots\Delta_1\cdots)[x:=N])\cdots,\end{aligned}$$

   where $\Delta_1'$ is the contractum of $\Delta_1$, and $\Delta_2' \equiv (\lambda x.\cdots\Delta_1'\cdots)N$. If we take

$$M_3 \stackrel{\text{def}}{\equiv} \cdots((\cdots\Delta_1'\cdots)[x:=N])\cdots,$$

   then the only complete development of $\Delta_2/\boldsymbol{\sigma}_1$ is $M_1 \xrightarrow{\Delta_2'}_\beta M_3$. Furthermore, substitutivity holds for PCF-like rewrite systems; that is,

$$M \xrightarrow{\Delta} M' \implies M[x:=N] \xrightarrow{\Delta'} M'[x:=N],$$

   where $\Delta'$ is $\Delta$ with any free occurrences of $x$ replaced by $N$. Thus the only complete development of $\Delta_1/\boldsymbol{\sigma}_2$ is $M_2 \rightarrow M_3$.
   (b) $\Delta_2 \equiv (\lambda x.N)(\cdots\Delta_1\cdots)$. Then $M_0$, $M_1$, and $M_2$ can be written

$$\begin{aligned} M_0 &\equiv \cdots((\lambda x.N)(\cdots\Delta_1\cdots))\cdots,\\ M_1 &\equiv \cdots((\lambda x.N)(\cdots\Delta_1'\cdots))\cdots,\\ M_2 &\equiv \cdots(N[x:=(\cdots\Delta_1\cdots)])\cdots,\end{aligned}$$

   where $\Delta_1'$ is the contractum of $\Delta_1$, and $\Delta_2' \equiv (\lambda x.N)(\cdots\Delta_1'\cdots)$. Defining

$$M_3 \stackrel{\text{def}}{\equiv} \cdots(N[x:=(\cdots\Delta_1'\cdots)])\cdots,$$

   we see that the only complete development of $\Delta_2/\boldsymbol{\sigma}_1$ is $M_1 \xrightarrow{\Delta_2'}_\beta M_3$. Furthermore, descendants of $\Delta_1$ in $M_2$ are disjoint, and any contraction of them in turn is a reduction $M_2 \xrightarrow{\Delta_1} \cdots \xrightarrow{\Delta_1} M_3$.
   (c) $\Delta_2 \equiv \delta_\theta\langle\cdots,\cdots(\cdots\Delta_1\cdots)\cdots\rangle$. Then we write $M_0$, $M_1$, and $M_2$ as

$$\begin{aligned} M_0 &\equiv \cdots(\delta_\theta\langle\cdots,\cdots(\cdots\Delta_1\cdots)\cdots\rangle)\cdots,\\ M_1 &\equiv \cdots(\delta_\theta\langle\cdots,\cdots(\cdots\Delta_1'\cdots)\cdots\rangle)\cdots,\\ M_2 &\equiv \cdots(P_\theta(\cdots(\cdots\Delta_1\cdots)\cdots))\cdots,\end{aligned}$$

   where $\Delta_1'$ is the contractum of $\Delta_1$, and $\Delta_2' \equiv \delta_\theta\langle\cdots,\cdots(\cdots\Delta_1'\cdots)\cdots\rangle$. Defining

$$M_3 \stackrel{\text{def}}{\equiv} \cdots(P_\theta(\cdots(\cdots\Delta_1'\cdots)\cdots))\cdots,$$

we see that the only complete development of $\Delta_2/\boldsymbol{\sigma}_1$ is $M_1 \xrightarrow{\Delta_2'}_\theta M_3$. And just as in case 2(b), the descendants of $\Delta_1$ in $M_2$ are disjoint, so by contracting them in turn we find a reduction $M_2 \xrightarrow{\Delta_1} \cdots \xrightarrow{\Delta_1} M_3$.

3. $\Delta_2 \subset \Delta_1$. This case is handled exactly as case 2.

$\square$

**C.4. A labelled $\lambda$-calculus.** For any PCF-like rewrite system, there is a corresponding *labelled* PCF-like system that is strongly normalizing. The labelling technique has led to some of the simplest proofs for many syntactic properties, and we will use it in our proof of standardization. This section introduces labelled calculi and proves that they are strongly normalizing.

The labelled system is similar to the system that we introduced earlier to define descendants. However, the systems are also different in important ways, since they are intended for different purposes. In the labelled system, we will mark $\delta$'s with nonnegative integers instead of $*$'s, and we will not need to mark $\lambda$'s. Furthermore, we do not allow unmarked $\delta$'s. The reasons for this will become apparent in what follows.

For any PCF-like language $\mathcal{L}$, the language $\mathcal{L}_{\mathbb{N}}$ is just the PCF-like language with constants $\delta_n^\sigma$ for each constant $\delta^\sigma$ of $\mathcal{L}$ and each $n \in \mathbb{N}$.

Notation C.9.

(i) *If $M \in \mathcal{L}_{\mathbb{N}}$, then $|M| \in \mathcal{L}$ is the term derived from $M$ by erasing the labels on the constants.*

(ii) *If $M \in \mathcal{L}$, then $M^n \in \mathcal{L}_{\mathbb{N}}$ is the term derived from $M$ by labelling each constant with $n$.*

The $\delta$-rules $\Theta_{\mathbb{N}}$ of the labelled calculus are defined as follows. If $\theta$ is a rule of $\Theta$,

$$\theta : \delta\langle \vec{c}, \vec{x}\rangle \to P(\vec{x}),$$

then $\Theta_{\mathbb{N}}$ contains all rules of the form $\theta_{\mathbb{N}}$:

$$\theta_{\mathbb{N}} : \delta_{n+1}\langle \vec{c}', \vec{x}\rangle \to P^n(\vec{x}),$$

where $\vec{c}'$ is a vector of $\mathcal{L}_{\mathbb{N}}$ ground constants such that $|\vec{c}'| \equiv \vec{c}$. Note that there is no rule for any $\delta_0$.

The projection $|\boldsymbol{\sigma}|$ of a $\Theta_{\mathbb{N}}$-reduction path $\boldsymbol{\sigma}$ is defined in the obvious way. And any finite $\Theta$-reduction $\boldsymbol{\sigma}$ can be lifted to a $\Theta_{\mathbb{N}}$-reduction $\boldsymbol{\sigma}'$ such that $\boldsymbol{\sigma} \equiv |\boldsymbol{\sigma}'|$ (*e.g.*, label each constant in the first term of $\boldsymbol{\sigma}$ by the length of $\boldsymbol{\sigma}$).

Definition C.10. *A term $M$ is* strongly normalizable *(SN) if all reductions starting at $M$ are finite.*

Theorem C.11 (Strong Normalization). *Every $\mathcal{L}_{\mathbb{N}}$ term is strongly normalizable.*

The rest of this section lays out the proof of strong normalization. We use a straightforward extension of the method of [18].

Definition C.12. *The notion of* strong computability *(SC) of a term is defined by induction as follows:*

(i) *A term of ground type is SC iff it is SN.*

(ii) *A term $M^{(\sigma\to\tau)}$ is SC iff, for every SC term $N^\sigma$, the term $(MN)^\tau$ is SC.*

Note C.13. *By definition C.12(ii) a term $M$ is SC iff, for all vectors $\vec{N}$ of SC terms driving $M$ to ground type, the term $M\vec{N}$ is SC. And by definition C.12(i), such an $M\vec{N}$ is SC iff it is SN.*

DEFINITION C.14. *An* atom *is a variable or a constant $\delta_n$ with no rule.*

LEMMA C.15.

(i) *If $a$ is an atom and $\vec{N}$ is a vector of SN terms, then the term $a\vec{N}$ is SC.*

(ii) *Every SC term $M$ is SN.*

*Proof.* By induction on the type of $a\vec{N}$ and $M$.

1. Basis: $a\vec{N}$ and $M$ have ground type.
   (i) Since each $N_i$ is SN, $a\vec{N}$ must be SN, and therefore SC by definition C.12(i).
   (ii) By definition C.12(i).
2. Induction: $a\vec{N}$ and $M$ have type $\sigma \to \tau$.
   (i) Let $P^\sigma$ be SC. By the induction hypothesis (ii), $P$ is SN. Then by induction, the term $(a\vec{N}P)^\tau$ is SC. Therefore so is $a\vec{N}$ by definition C.12(ii).
   (ii) Let $x^\sigma$ be a variable not occurring in $M$. By the induction hypothesis (i), $x$ is SC. Then $(Mx)^\tau$ is SC, and therefore SN by induction. But any subterm of an SN term is SN, so $M$ is SN as well.

□

LEMMA C.16. *If $N$ is SC and $M[x := N]$ is SC, then so is $(\lambda x M)N$.*

*Proof.* Let $\vec{P} \equiv P_1, \ldots, P_n$ be a vector of SC terms driving $M$ to ground type. Since $M[x := N]$ is SC, the term

$$(M[x := N])\vec{P} \tag{C.5}$$

is SN by Note C.13. The lemma follows from Note C.13 if we can prove that

$$(\lambda x M)N\vec{P} \tag{C.6}$$

is SN.

Now since (C.5) is SN, all of its subterms are SN, including $M[x := N], \vec{P}$. Furthermore by hypothesis and the preceding lemma, $N$ is SN. Therefore an infinite reduction from (C.6) cannot consist entirely of contractions in $M, N, P_1, \ldots, P_n$. So an infinite reduction of (C.6) must have the form

$$\begin{array}{rcl} (\lambda x M)NP_1 \cdots P_n & \twoheadrightarrow & (\lambda x M')N'P'_1 \cdots P'_n \\ & \to & M'[x := N']P'_1 \cdots P'_n \\ & \twoheadrightarrow & \ldots \end{array}$$

(where $M \twoheadrightarrow M'$, etc.) From the reductions $M \twoheadrightarrow M'$ and $N \twoheadrightarrow N'$ we have

$$M[x := N] \twoheadrightarrow M'[x := N']$$

Then we can construct an infinite reduction from (C.5) as follows:

$$\begin{array}{rcl} M[x := N]P_1 \cdots P_n & \twoheadrightarrow & M'[x := N']P'_1 \cdots P'_n \\ & \twoheadrightarrow & \ldots \end{array}$$

But this contradicts the fact that (C.5) is SN. Therefore there is no infinite reduction from (C.6); it must be SN. □

LEMMA C.17. *Consider a constant $\delta$ and a vector $\vec{N}$ of SC terms driving $\delta$ to ground type. If for each rule $\theta$ on $\delta$,*

$$\theta : \delta_\theta\langle \vec{c}, \vec{x}\rangle \to P_\theta(\vec{x}),$$

*where* $\delta\vec{N} \equiv \delta_\theta\langle\vec{N}_1, \vec{N}_2\rangle\vec{N}_3$, *we have that*

$$P_\theta(\vec{N}_2)\vec{N}_3 \tag{C.7}$$

*is SC, then* $\delta\vec{N}$ *is SC.*

*Proof.* We must show that $\delta\vec{N}$ is SN. Since the $\vec{N}$ are SC, by Lemma C.15 they are SN. Therefore any infinite reduction from $\delta\vec{N}$ must look like

$$\begin{array}{rcl} \delta_\theta\langle\vec{N}_1, \vec{N}_2\rangle\vec{N}_3 & \twoheadrightarrow & \delta_\theta\langle\vec{c}, \vec{N}_2{}'\rangle\vec{N}_3{}' \\ & \rightarrow & P_\theta(\vec{N}_2{}')\vec{N}_3{}' \\ & \twoheadrightarrow & \cdots \end{array}$$

where $\vec{N}_1 \twoheadrightarrow \vec{c}$, $\vec{N}_2 \twoheadrightarrow \vec{N}_2{}'$, etc. But then we can construct an infinite reduction from (C.7) as follows:

$$\begin{array}{rcl} P_\theta(\vec{N}_2)\vec{N}_3 & \twoheadrightarrow & P_\theta(\vec{N}_2{}')\vec{N}_3{}' \\ & \twoheadrightarrow & \cdots \end{array}$$

But as (C.7) is SC, by Lemma C.15 it is SN, a contradiction. Therefore $\delta\vec{N}$ is SN. □

Lemma C.18. *For any term* $M$ *and substitution* $\rho \equiv [\vec{x} := \vec{N}]$, *where each* $N_i$ *is SC, the term* $M\rho$ *is SC.*

*Proof.* The proof is by induction on the lexicographic ordering of $(m, M)$, where $m$ is the maximum $\delta$-index appearing in $M$.

1. $M$ is a variable $x_i$. Then $M\rho$ is $N_i$ and the result follows.
2. $M$ is an atom distinct from $x_1, \ldots, x_n$. Then $M\rho \equiv M$ which is SC by Lemma C.15. Note that this includes all constants $\delta_0$.
3. $M \equiv \delta_{m+1}$. Then $M\rho \equiv \delta_{m+1}$. Thus it is sufficient to show that for any vector $\vec{N}'$ of SC terms driving $\delta_{m+1}$ to ground type, the term $\delta_{m+1}\vec{N}'$ is SC. Consider any rule $\theta$ on $\delta_{m+1}$:

   $$\theta : \delta_{m+1}\langle\vec{c}, \vec{x}\rangle \rightarrow P(\vec{x}).$$

   By construction of the labelled calculus, no constants in $P$ are labelled with an index greater than $m$. Thus we can apply the induction hypothesis to $P$. If we rewrite $\delta_{m+1}\vec{N}'$ as $\delta_{m+1}\langle\vec{N}_1{}', \vec{N}_2{}'\rangle\vec{N}_3{}'$, by induction $P(\vec{N}_2{}')$ is SC. Then by the definition of SC, the term $P(\vec{N}_2{}')\vec{N}_3{}'$ is SC. Therefore by Lemma C.17, $\delta_{m+1}\vec{N}'$ is SC.
4. $M \equiv \lambda y^\sigma M_1$. Then $M\rho \equiv \lambda y(M_1\rho)$, neglecting changes in bound variables. To show that $M\rho$ is SC we must show that for all SC terms $N^\sigma$, the term $(M\rho)N$ is SC. But $(M\rho)N \equiv (\lambda y(M_1\rho))N$, and

   $$(M_1\rho)[y := N] \quad \equiv \quad M_1[x_1 := N_1]\cdots[x_n := N_n][y := N]$$

   which is SC by induction. Therefore $(\lambda y(M_1\rho))N$ is SC by Lemma C.16.
5. $M \equiv M_1M_2$. Then $M\rho \equiv (M_1\rho)(M_2\rho)$, and $M_1\rho$ and $M_2\rho$ are SC by induction. Therefore $M\rho$ is SC by definition C.12(ii).

□

*Proof of Theorem C.11 (Strong Normalization):* By Lemma C.18, every term $M$ is SC (just let $\vec{x}$ be empty). Then by Lemma C.15, $M$ is strongly normalizing. □

**C.5. Standard Reductions.** Our definition of standard reductions is similar to that of [20], with a few important differences. The "linear ground" restriction imposed on our systems gives us a particularly simple class of rewrite rules, and this simplicity carries over to the definition of standard reductions. On the other hand, the systems of [20] do not include $\lambda$-abstraction, and forbid overlapping rewrite rules, which we allow.

Overlapping rules do not add much complication to the definition of standard reductions, but they are more of an obstacle in the proof of standardization. Overlapping systems are not confluent in general, so we cannot use confluence and related properties in our proof. This is offset by the fact that we consider only typed systems.

The standard reductions of [20] are based on "outside-in" reductions. Informally, outside-in reductions are reductions in which no subterm of a term reduces before the term itself contracts, unless the subterm reduces outside-in and contributes towards making the term a redex. For example, consider the PCF reduction

$$\begin{aligned} \texttt{cond}\,(\texttt{zero?}\,0)\,M\,N &\rightarrow \texttt{cond}\,\texttt{tt}\,M\,N \\ &\rightarrow M. \end{aligned}$$

The reduction is standard, even though the term $\texttt{cond}\,(\texttt{zero?}\,0)\,M\,N$ contracts after its subterm $(\texttt{zero?}\,0)$, because it is the contraction of $(\texttt{zero?}\,0)$ that turns the $\texttt{cond}$ term into a redex.

There is a natural way of testing whether or not a reduction is outside-in: first, identify "outermost" subterms that contract; each of these identifies subterms that must reduce before the outer subterm itself contracts. By iterating the process, we can identify a subterm or subterms that must reduce before any others, if the reduction is to be outside-in. This idea is the basis of our definition of standard reductions.

For each term in a reduction, we identify a *principal redex*, and call a reduction standard if the redex contracted at each step is the principal redex. For the pure $\lambda$-calculus, the principal redex for some $M_i$ will simply be the leftmost redex of $M_i$ contracted in the reduction.

For systems with constants, we must allow reductions to take place in the critical arguments of some $\delta$-terms. To find the principal redex, then, we start by considering the leftmost contracted subterm; if it is a $\delta$-term, we then consider critical arguments in which contractions take place, etc. Eventually, consideration of these *preprincipal* subterms leads to the principal redex.

DEFINITION C.19. *Let $M_i$ be a term in a reduction path $\boldsymbol{\sigma}$,*

$$\boldsymbol{\sigma} : M_1 \overset{\Delta_1}{\rightarrow} M_2 \overset{\Delta_2}{\rightarrow} M_3 \overset{\Delta_3}{\rightarrow} \cdots.$$

*The subterms of $M_i$ that are* preprincipal in $\boldsymbol{\sigma}$ *are defined inductively:*

(i) *If $\Delta$ is the leftmost subterm of $M_i$ contracted in $\boldsymbol{\sigma}$, then $\Delta$ is preprincipal in $\boldsymbol{\sigma}$.*

(ii) *If $\delta_\theta\langle\vec{A},\vec{B}\rangle$ is preprincipal and $\theta$-contracted in $\boldsymbol{\sigma}$, and $\Delta$ is the leftmost contracted subterm of $\vec{A}$, then $\Delta$ is preprincipal in $\boldsymbol{\sigma}$.*

*We write* $\mathrm{pp}_\sigma(\Delta)$ *if $\Delta$ is preprincipal in $\boldsymbol{\sigma}$.*

This next lemma is essential in showing an important property of the preprincipal subterms: they are linearly ordered by $\subset$ (see the following note):

LEMMA C.20. *Let $M_i$ be a term in a reduction path $\boldsymbol{\sigma}$,*

$$\boldsymbol{\sigma} : M_1 \overset{\Delta_1}{\rightarrow} M_2 \overset{\Delta_2}{\rightarrow} M_3 \overset{\Delta_3}{\rightarrow} \cdots,$$

*and let $\Delta$ be a preprincipal subterm of $M_i$. If $\Delta \not\equiv \Delta_i$, then $\Delta$ is not contained in $\Delta_i$, and has a unique, preprincipal descendant $\Delta' \subset M_{i+1}$.*

*Proof.* By induction on how $\mathrm{pp}_\sigma(\Delta)$.

(i) $\mathrm{pp}_\sigma(\Delta)$ because $\Delta$ is the leftmost contracted subterm of $M_i$. Clearly $\Delta_i$ does not contain $\Delta$, else $\Delta$ would not be leftmost. So either $\Delta_i \subset \Delta$, or $\Delta_i$ is disjoint and to the right of $\Delta$. In either case, $\Delta$ has a unique descendant $\Delta'$ in $M_{i+1}$. Furthermore, the contraction of $\Delta_i$ can only introduce terms inside of or to the right of $\Delta'$, and so $\Delta'$ must be the leftmost contracted subterm of $M_{i+1}$. Thus $\mathrm{pp}_\sigma(\Delta')$.

(ii) $\mathrm{pp}_\sigma(\Delta)$ because $M_i$ contains a preprincipal, $\theta$-contracted subterm, $\delta_\theta\langle\vec{A},\vec{B}\rangle$, where $\Delta$ is the leftmost subterm of $\vec{A}$ contracted in $\boldsymbol{\sigma}$.
Now $\Delta_i \not\equiv \delta_\theta\langle\vec{A},\vec{B}\rangle$, else by Note C.2(ii), $\Delta$ would have no descendant in $M_{i+1}$, contradicting the fact that it is contracted in $\boldsymbol{\sigma}$.
Then by induction, $\Delta_i$ does not contain $\delta_\theta\langle\vec{A},\vec{B}\rangle$. So either $\Delta_i$ is in $\vec{A}$, is in $\vec{B}$, or is entirely disjoint. If $\Delta_i$ is in $\vec{A}$, it cannot contain $\Delta$ because $\Delta$ is leftmost. And if $\Delta_i$ is in $\vec{B}$ or is disjoint, it clearly does not contain $\Delta$.
Again by induction, $\delta_\theta\langle\vec{A},\vec{B}\rangle$ has a unique, preprincipal descendant, which must be of the form $\delta_\theta\langle\vec{A}',\vec{B}'\rangle$. And since $\Delta$ is not contained in $\Delta_i$, $\Delta$ must have unique descendant $\Delta'$ that is the leftmost contracted subterm of $\vec{A}'$. And therefore $\mathrm{pp}_\sigma(\Delta')$.

□

NOTE C.21.

(i) *By Lemma C.20, every preprincipal subterm contracts exactly once in $\boldsymbol{\sigma}$. Thus the $\theta$ and $\vec{A}$ of Definition C.19(ii) are unique.*

(ii) *By* (i), *we conclude that if $\Delta_1$ and $\Delta_2$ are distinct, preprincipal subterms of $M_i$, then either $\Delta_1 \subset \Delta_2$ or $\Delta_2 \subset \Delta_1$.*

DEFINITION C.22. *Suppose $\boldsymbol{\sigma}$ is a reduction path,*

$$\boldsymbol{\sigma} : M_1 \stackrel{\Delta_1}{\to} M_2 \stackrel{\Delta_2}{\to} M_3 \stackrel{\Delta_3}{\to} \cdots.$$

(i) *We define the* principal redex $\mathrm{pr}_\sigma(M_i)$ *to be the innermost preprincipal subterm of $M_i$. By Note C.21(ii), this is well defined.*

(ii) *We say $\boldsymbol{\sigma}$ is a* standard reduction *if for all $i$, $\Delta_i \equiv \mathrm{pr}_\sigma(M_i)$.*

In Definition 4.3, we gave a simple definition of "standard reductions to ground constants." The standard reductions we define here are much more general: they apply to reductions between arbitrary terms, and even to infinite reductions.

It is not immediately obvious that the general definition we give here is equivalent to our earlier definition in those cases where the earlier definition applies. This equivalence is established by the next two lemmas.

LEMMA C.23. *Suppose $\boldsymbol{\sigma} : C[M_1] \stackrel{\Delta_1}{\to} C[M_2] \stackrel{\Delta_2}{\to} \cdots \stackrel{\Delta_{n-1}}{\to} C[M_n] \to \cdots$ is a standard reduction, $\Delta_i \subset M_i$ for $1 \le i < n$, $C[\cdot]$ is a context with a single hole, and no subterm of $M_n$ contracts in $\boldsymbol{\sigma}$. Then the reduction $\boldsymbol{\sigma}' : M_1 \stackrel{\Delta_1}{\to} M_2 \stackrel{\Delta_2}{\to} \cdots \stackrel{\Delta_{n-1}}{\to} M_n$ is a standard reduction.*

*Proof.* Suppose $\Delta \subset M_i$. Since no subterm of $M_n$ contracts in $\boldsymbol{\sigma}$, we have

$$\Delta \text{ contracts in } \boldsymbol{\sigma} \text{ iff } \Delta \text{ contracts in } \boldsymbol{\sigma}'. \tag{C.8}$$

Now we show that if $\Delta$ is preprincipal in $\boldsymbol{\sigma}$, then $\Delta$ is preprincipal in $\boldsymbol{\sigma}'$, by induction on the definition of preprincipal.

- $\Delta$ is the leftmost subterm of $C[M_i]$ contracted in $\boldsymbol{\sigma}$. Then $\Delta$ is the leftmost subterm of $M_i$ contracted in $\boldsymbol{\sigma}$. And then by (C.8), $\Delta$ is the leftmost subterm of $M_i$ contracted in $\boldsymbol{\sigma}'$, so $\Delta$ is preprincipal in $\boldsymbol{\sigma}'$.
- $\Delta$ is the leftmost subterm of $\vec{A}$ contracted in $\boldsymbol{\sigma}$, where $\Delta' = \delta_\theta\langle\vec{A},\vec{B}\rangle$ is preprincipal in $\boldsymbol{\sigma}$.
  If $\Delta' \subset M_i$, then by induction, $\Delta'$ is preprincipal in $\boldsymbol{\sigma}'$. And by (C.8), $\Delta$ is the leftmost subterm of $\vec{A}$ contracted in $\boldsymbol{\sigma}'$, so $\Delta$ is preprincipal in $\boldsymbol{\sigma}'$.
  Otherwise, $M_i$ is properly contained by $\Delta'$. Then we must have $M_i \subset A_j$ for some $A_j \in \vec{A}$. So $\Delta$ is the leftmost subterm of $M_i$ contracted in $\boldsymbol{\sigma}$. And then by (C.8), $\Delta$ is the leftmost subterm of $M_i$ contracted in $\boldsymbol{\sigma}'$, so $\Delta$ is preprincipal in $\boldsymbol{\sigma}'$.

Finally, we show that if $\Delta$ is principal in $\boldsymbol{\sigma}$, then $\Delta$ is principal in $\boldsymbol{\sigma}'$. This is sufficient to show that $\boldsymbol{\sigma}'$ is standard.

By way of contradiction assume $\Delta$ is principal in $\boldsymbol{\sigma}$ but not in $\boldsymbol{\sigma}'$. We have already shown that $\Delta$ is preprincipal in $\boldsymbol{\sigma}'$, so it must be that $\Delta$ is not the *innermost* preprincipal subterm of $M_i$ in $\boldsymbol{\sigma}'$. Then $\Delta$ must be of the form $\delta_\theta\langle\vec{A},\vec{B}\rangle$, and there must be a leftmost subterm $\Delta'$ of $\vec{A}$ contracted in $\boldsymbol{\sigma}'$. Then by (C.8), $\Delta'$ is the leftmost subterm of $\vec{A}$ contracted in $\boldsymbol{\sigma}$. But then $\Delta'$ would be preprincipal in $\boldsymbol{\sigma}$, contradicting that $\Delta$ is principal in $\boldsymbol{\sigma}$. ∎

In the following lemma, we abbreviate "standard reduction to ground constant" by SRTGC.

LEMMA C.24. *If $\boldsymbol{\sigma} : M \twoheadrightarrow_s c$, then $\boldsymbol{\sigma}$ is a* SRTGC.

*Proof.* By induction on the length of $\boldsymbol{\sigma}$.

If $|\boldsymbol{\sigma}| = 0$, then $\boldsymbol{\sigma}$ must be of the form $c \twoheadrightarrow c$, which is a SRTGC.

If $|\boldsymbol{\sigma}| > 0$, consider cases on the structure of $M$:

- $M \equiv (\lambda x M_1)M_2\vec{N}$: because $\boldsymbol{\sigma}$ ends in a ground constant, we know $(\lambda x M_1)M_2$ must be contracted in $\boldsymbol{\sigma}$. Therefore it is the principal subterm of $M$, and $\boldsymbol{\sigma}$ must be of the form $(\lambda x M_1)M_2\vec{N} \rightarrow (M_1[x := M_2])\vec{N} \twoheadrightarrow_s c$. By induction the reduction $(M_1[x := M_2])\vec{N} \twoheadrightarrow_s c$ is a SRTGC, so $\boldsymbol{\sigma}$ is a SRTGC.
- $M \equiv \delta\vec{N}$: Since $\boldsymbol{\sigma}$ ends in $c$, we know the head $\delta$ must be contracted. That is, $\delta\vec{N}$ must be of the form $\delta_\theta\langle\vec{C},\vec{D}\rangle\vec{E}$, where $\Delta \equiv \delta_\theta\langle\vec{C},\vec{D}\rangle$ is the leftmost subterm of $M$ contracted (by $\theta$) in $\boldsymbol{\sigma}$. By the definition of standard reductions, the only subterms that contract before $\Delta$ in $\boldsymbol{\sigma}$ are contained in $\vec{C}$, and moreover reduction in $\vec{C}$ proceeds from left to right. Therefore $\boldsymbol{\sigma}$ is of the form

$$\begin{array}{lllll}
\boldsymbol{\sigma}_1 : & \delta_\theta\langle C_1C_2\cdots C_n,\vec{D}\rangle\vec{E} & \twoheadrightarrow & \delta_\theta\langle c_1C_2\cdots C_n,\vec{D}\rangle\vec{E} \\
\boldsymbol{\sigma}_2 : & & \twoheadrightarrow & \delta_\theta\langle c_1c_2\cdots C_n,\vec{D}\rangle\vec{E} \\
\vdots & & \twoheadrightarrow & \cdots \\
\boldsymbol{\sigma}_n : & & \twoheadrightarrow & \delta_\theta\langle c_1c_2\cdots c_n,\vec{D}\rangle\vec{E} \\
 & & \rightarrow_\theta & P_\theta(\vec{D})\vec{E} \\
 & & \twoheadrightarrow_s & c,
\end{array}$$

  By Lemma C.23, the reductions $\boldsymbol{\sigma}_i' : C_i \twoheadrightarrow c_i$ (derived from $\boldsymbol{\sigma}_i$ in the obvious way) are standard reductions. Then by induction, each $\boldsymbol{\sigma}_i'$ is a SRTGC, and therefore $\boldsymbol{\sigma}$ is a SRTGC.
- $M \equiv x\vec{N}$: no such term can reduce to a ground constant, so we do not need to consider this case.

□

The following theorem is the main result of this appendix.

THEOREM C.25 (STANDARDIZATION). *If $M \twoheadrightarrow N$ is a finite reduction in a PCF-like rewrite system, then there is a standard reduction from $M$ to $N$.*

**C.6. Path-reduction.** This section gives our proof of Standardization. It is based on a proof in [24] for the pure $\lambda$-calculus, which introduced a sort of meta-reduction: a reduction relation on reduction paths. This *path-reduction* rewrites non-standard reductions into "more standard" reductions. The following results motivate the definition of path-reduction.

LEMMA C.26. *Let $\boldsymbol{\sigma}$ be a reduction path,*

$$\boldsymbol{\sigma} : M_1 \xrightarrow{\Delta_1} M_2 \xrightarrow{\Delta_2} M_3 \xrightarrow{\Delta_3} \cdots,$$

*and let $\Delta \equiv \mathrm{pr}_\sigma(M_i)$. If $\Delta_i \not\equiv \Delta$, then $\Delta$ has a unique descendant $\Delta' \subset M_{i+1}$, and $\Delta' \equiv \mathrm{pr}_\sigma(M_{i+1})$.*

*Proof.* Lemma C.20 proves uniqueness. To show $\Delta' \equiv \mathrm{pr}_\sigma(M_{i+1})$, by the definition of $\mathrm{pr}_\sigma$ and Lemma C.20 it suffices to note the following: if $\Delta_1 \subset \Delta_2 \subset M$ have unique descendants $\Delta'_1, \Delta'_2 \subset M'$, where $M \to M'$, then $\Delta'_1 \subset \Delta'_2$. □

COROLLARY C.27. *Suppose $\boldsymbol{\sigma}$ is a reduction path,*

$$\boldsymbol{\sigma} : M_1 \xrightarrow{\Delta_1} M_2 \xrightarrow{\Delta_2} \cdots \xrightarrow{\Delta_{n-1}} M_n.$$

*Then $\boldsymbol{\sigma}$ is standard iff there is no $j$ such that $\Delta_j$ is the descendant of $\mathrm{pr}_\sigma(M_{j-1})$.*

The corollary suggests a possible way to transform a non-standard reduction into a standard reduction: successively "swap" the contraction of a principal redex with the contraction of a non-principal redex at the previous step. If we reach a reduction in which each principal redex contracts as soon as it becomes principal, we will have found a standard reduction.

DEFINITION C.28. *Suppose $\boldsymbol{\sigma}$ is a non-standard reduction, that is, there is some $j$ such that*

$$\boldsymbol{\sigma} : \cdots \to M_{j-1} \xrightarrow{\Delta_{j-1}} M_j \xrightarrow{\Delta_j} M_{j+1} \to \cdots$$

*where $\Delta_j$ is the descendant of $\Delta'_j \equiv \mathrm{pr}_\sigma(M_{j-1})$. The subpath*

$$M_{j-1} \xrightarrow{\Delta_{j-1}} M_j \xrightarrow{\Delta_j} M_{j+1}$$

*is called the* path-redex at step $j$. *Note that $\Delta'_j$ and $\Delta_{j-1}$ do not overlap, and furthermore, by Lemma C.26, $\Delta_j$ is the unique descendant of $\Delta'_j$. Therefore by Lemma C.8, we can find a sequence*

$$M_{j-1} \xrightarrow{\Delta'_j} M'_j \xrightarrow{\Delta'_{j-1}} \cdots \xrightarrow{\Delta'_{j-1}} M_{j+1},$$

*where the $\Delta'_{j-1}$ are the descendants of $\Delta_{j-1}$. Such a sequence is called a* path-contractum. *Finally, we define* path-reduction*: $\boldsymbol{\sigma} \underset{\text{path}}{\xrightarrow{j}} \boldsymbol{\sigma}'$ if $\boldsymbol{\sigma}'$ is obtained from $\boldsymbol{\sigma}$ by replacing the path-redex at step $j$ by a corresponding path-contractum. We will drop the index $j$ when convenient.*

Clearly, path-reduction preserves initial and final terms, and any path-reduction normal form is a standard reduction. Moreover, the next two lemmas show that path-reduction is strongly normalizing.

LEMMA C.29. *Suppose* $\boldsymbol{\sigma} \xrightarrow[\text{path}]{j} \boldsymbol{\sigma}'$, *where*

$$\begin{array}{lcl}\boldsymbol{\sigma} & : & M_1 \to \cdots \to M_{j-1} \xrightarrow{\Delta_{j-1}} M_j \xrightarrow{\Delta_j} M_{j+1} \to \cdots, \\ \boldsymbol{\sigma}' & : & M_1 \to \cdots \to M_{j-1} \xrightarrow{\Delta_j'} M_j' \xrightarrow{\Delta_{j-1}'} \cdots \xrightarrow{\Delta_{j-1}'} M_{j+1} \to \cdots.\end{array}$$

*Then for* $i \neq j$, *the following hold:*

(i) *If* $\Delta \subset M_i$ *is not contracted in* $\boldsymbol{\sigma}$, *then it is not contracted in* $\boldsymbol{\sigma}'$.
(ii) *If* $\Delta \subset M_i$ *is contracted in* $\boldsymbol{\sigma}$ *and* $\mathrm{pp}_\sigma(\Delta)$, *then* $\Delta$ *is contracted in* $\boldsymbol{\sigma}'$.
(iii) *If* $\Delta \subset M_i$ *is preprincipal in* $\boldsymbol{\sigma}$, *then it is preprincipal in* $\boldsymbol{\sigma}'$.
(iv) $\mathrm{pr}_\sigma(M_i) \equiv \mathrm{pr}_{\sigma'}(M_i)$.

*Proof.*

(i) Just note that path-reduction only permutes the order of contraction of subterms; it does not introduce new contractions.
(ii) It is clear that if $\Delta$ contracts in $\boldsymbol{\sigma}$ and does not contract in $\boldsymbol{\sigma}'$, then $\Delta$ is either $\Delta_{j-1}$ or one of its ancestors. Thus we only need consider $\Delta_{j-1}$.
If $\Delta_{j-1}$ does not contract in $\boldsymbol{\sigma}'$, then it must be contained in $\Delta_j'$. But $\Delta_j'$ is the principal redex of $M_{j-1}$, that is, the innermost preprincipal subterm of $M_{j-1}$. So if $\Delta_{j-1}$ is not contracted in $\boldsymbol{\sigma}'$, it is not preprincipal in $\boldsymbol{\sigma}$.
(iii) We use induction on how $\mathrm{pp}_\sigma(\Delta)$.
  1. $\mathrm{pp}_\sigma(\Delta)$ because $\Delta$ is the leftmost contracted subterm of $M_i$. By (ii), $\Delta$ is contracted in $\boldsymbol{\sigma}'$, and by (i), it is the leftmost contracted subterm of $M_i$ in $\boldsymbol{\sigma}'$. Therefore $\mathrm{pp}_{\sigma'}(\Delta)$.
  2. $\mathrm{pp}_\sigma(\Delta)$ because $\mathrm{pp}_\sigma(\delta_\theta\langle\vec{A},\vec{B}\rangle)$, and $\Delta$ is the leftmost contracted subterm of $\vec{A}$. By induction, $\mathrm{pp}_{\sigma'}(\delta_\theta\langle\vec{A},\vec{B}\rangle)$, and by (i) and (ii), $\Delta$ is the leftmost subterm of $\vec{A}$ contracted in $\boldsymbol{\sigma}'$. Therefore $\mathrm{pp}_{\sigma'}(\Delta)$.
(iv) This follows from (i), (iii), and the definition of $\mathrm{pr}_\sigma$.

□

LEMMA C.30. *If* $\boldsymbol{\sigma}$ *is a finite reduction, then there is no infinite path-reduction starting from* $\boldsymbol{\sigma}$.

*Proof.* Consider a path-reduction

$$\boldsymbol{\sigma} \equiv \boldsymbol{\sigma}_1 \underset{\text{path}}{\to} \boldsymbol{\sigma}_2 \underset{\text{path}}{\to} \boldsymbol{\sigma}_3 \underset{\text{path}}{\to} \cdots.$$

It is not hard to see that the reduction could have been carried out in the labelled system; that is, if $\boldsymbol{\sigma}_i'$ is a labelled reduction such that $|\boldsymbol{\sigma}_i'| \equiv \boldsymbol{\sigma}_i$, and $\boldsymbol{\sigma}_i \xrightarrow[\text{path}]{j} \boldsymbol{\sigma}_{i+1}$, then there is a labelled reduction $\boldsymbol{\sigma}_{i+1}'$ such that $|\boldsymbol{\sigma}_{i+1}'| \equiv \boldsymbol{\sigma}_{i+1}$, and $\boldsymbol{\sigma}_i' \xrightarrow[\text{path}]{j} \boldsymbol{\sigma}_{i+1}'$. Thus we can find labelled reductions $\boldsymbol{\sigma}_1', \boldsymbol{\sigma}_2', \boldsymbol{\sigma}_3', \ldots$ such that $|\boldsymbol{\sigma}_i'| \equiv \boldsymbol{\sigma}_i$, and

$$\boldsymbol{\sigma}_1' \underset{\text{path}}{\to} \boldsymbol{\sigma}_2' \underset{\text{path}}{\to} \boldsymbol{\sigma}_3' \underset{\text{path}}{\to} \cdots.$$

And because labelled reduction is strongly normalizing, and each $\boldsymbol{\sigma}_i'$ begins with the same $\mathcal{L}_{\mathbb{N}}$ term, each $\boldsymbol{\sigma}_i$ is finite.

Furthermore, the path-reduction can be thought of as constructing a tree of terms, with each path from root to leaf corresponding to a reduction $\boldsymbol{\sigma}_i$. Each contracted path-redex introduces a branching in the tree. For example, if $\boldsymbol{\sigma}_i \xrightarrow[\text{path}]{j} \boldsymbol{\sigma}_{i+1}$, then the root-to-leaf path corresponding to $\boldsymbol{\sigma}_{i+1}$ is obtained by branching off of the root-to-leaf path of $\boldsymbol{\sigma}_i$ at depth $j-1$. The situation is depicted in the following figure, where the root of the tree is displayed at the left and the leaves are displayed at the right:

$$\begin{array}{llllllllll} M_1 \longrightarrow \cdots \longrightarrow & M_{j-1} & \xrightarrow{\Delta_{j-1}} & M_j & \xrightarrow{\Delta_j} & M_{j+1} \longrightarrow \cdots \longrightarrow M_n : \boldsymbol{\sigma}_i \\ & \downarrow{\scriptstyle \Delta'_j} & & & & \\ & M'_j & \xrightarrow{\Delta'_{j-1}} & \cdots & \xrightarrow{\Delta'_{j-1}} & M_{j+1} \longrightarrow \cdots \longrightarrow M_n : \boldsymbol{\sigma}_{i+1} \end{array}$$

By Lemma C.29(iv), the tree is a binary tree, and we have just seen that there is no infinite path from the root. Then by König's Lemma, the tree is finite, so the number of different reductions given by the tree must be finite. □

*Proof of Theorem C.25 (Standardization):* If $\boldsymbol{\sigma} : M \twoheadrightarrow N$ is a finite reduction in a PCF-like system, we can obtain a standard reduction from $M$ to $N$ just by finding a path-reduction normal form of $\boldsymbol{\sigma}$. □

The following results are included for completeness.

LEMMA C.31. *Path reduction is confluent.*

*Proof.* Suppose $\boldsymbol{\sigma} \underset{\text{path}}{\xrightarrow{i}} \boldsymbol{\sigma}_i$ and $\boldsymbol{\sigma} \underset{\text{path}}{\xrightarrow{j}} \boldsymbol{\sigma}_j$, where $i \neq j$. We show that there is a $\boldsymbol{\sigma}'$ such that $\boldsymbol{\sigma}_i \underset{\text{path}}{\rightarrow} \boldsymbol{\sigma}'$, and $\boldsymbol{\sigma}_j \underset{\text{path}}{\rightarrow} \boldsymbol{\sigma}'$. Thus the reflexive closure of path reduction satisfies the diamond property, and therefore path reduction is confluent (*cf.* [4] Lemma 3.2.2).

First note that if $|i-j| \geq 2$, then the path redexes at steps $i$ and $j$ are disjoint. In this case, we may contract the redexes in any order and reach a common $\boldsymbol{\sigma}'$.

Finally we show $|i-j| \geq 2$. By way of contradiction assume that $i = j+1$. Then $M_{j-1} \xrightarrow{\Delta_{j-1}} M_j \xrightarrow{\Delta_j} M_i \xrightarrow{\Delta_i} M_{i+1}$ is a subpath of $\boldsymbol{\sigma}$. But by the definition of path redex, $\Delta_j$ must be the principal redex of $M_j$ and $\Delta_i$ must be the descendent of the principal redex of $M_j$. This is impossible because $\Delta_j$ has no descendants in $M_i$ (*cf.* Note C.2(i)). □

COROLLARY C.32. *Path-reduction normal forms are unique.*

**Acknowledgments.** We are grateful to G. Berry, B. Bloom, P.-L. Curien, J.-J. Lévy, G.D. Plotkin, and Scott Smith for helpful discussions. The comments of a referee motivated some technical improvements.

REFERENCES

[1] S. ABRAMSKY, *The lazy lambda calculus*, in Research Topics in Functional Programming, D. L. Turner, ed., Addison-Wesley Publishing Co., 1989.

[2] ———, *Domain theory in logical form*, Ann. Pure Appl. Logic, 51 (1991), pp. 1–77.

[3] S. ABRAMSKY, P. MALACARIA, AND R. JAGADEESAN, *Full abstraction for PCF*, in Theoretical Aspects of Computer Software, M. Hagiya and J. C. Mitchell, eds., vol. 789 of Lecture Notes in Computer Science, Springer-Verlag, Apr. 1994, pp. 1–15.

[4] H. P. BARENDREGT, *The Lambda Calculus: Its Syntax and Semantics*, vol. 103 of Studies in Logic and the Foundations of Mathematics, North-Holland, Amsterdam, second ed., 1984.

[5] G. BERRY, *Séquentialité de l'evaluation formelle des lambda-expressions*, in Program Transformations, $3^{eme}$ Colloque International sur la programmation, B. Robinet, ed., 1978, pp. 67–80.

[6] ———, *Stable models of typed lambda-calculi*, in Automata, Languages and Programming: Fifth Colloquium, G. Ausiello and C. Böhm, eds., vol. 62 of Lecture Notes in Computer Science, Springer-Verlag, July 1978, pp. 72–89.

[7] G. BERRY AND P.-L. CURIEN, *Sequential algorithms on concrete data structures*, Theoretical Comput. Sci., 20 (1982), pp. 265–321.

[8] ———, *Theory and practice of sequential algorithms: the kernel of the programming language CDS*, in Algebraic Methods in Semantics, M. Nivat and J. C. Reynolds, eds., Cambridge Univ. Press, 1985, ch. 2, pp. 35–87.

[9] G. Berry, P.-L. Curien, and J.-J. Lévy, *Full abstraction for sequential languages: the state of the art*, in Algebraic Methods in Semantics, M. Nivat and J. C. Reynolds, eds., Cambridge Univ. Press, 1985, ch. 3, pp. 89–132.

[10] B. Bloom, *Can LCF be topped? Flat lattice models of typed lambda calculus (preliminary report)*, in Third Annual Symposium on Logic in Computer Science [22], pp. 282–295.

[11] ———, *Can LCF be topped? Flat lattice models of typed $\lambda$-calculus*, Information and Computation, 87 (1990), pp. 263–300.

[12] B. Bloom, S. Istrail, and A. R. Meyer, *Bisimulation can't be traced (preliminary report)*, in Conference Record of the Fifteenth Annual ACM Symposium on Principles of Programming Languages, 1988, pp. 229–239. Also appears as MIT Technical Memo MIT/LCS/TM–345; submitted for journal publication.

[13] A. Bucciarelli and T. Ehrhard, *Extensional embedding of a strongly stable model of PCF*, in Automata, Languages and Programming: Eighteenth Colloquium, July 1991.

[14] ———, *Sequentiality in an extensional framework*, Information and Computation, 110 (1994).

[15] R. Cartwright, P.-L. Curien, and M. Felleisen, *Fully abstract semantics for observably sequential languages*, Information and Computation, 111 (1994).

[16] P.-L. Curien, *Categorical Combinators, Sequential Algorithms and Functional Programming*, Birkhäuser, 1993. Second edition.

[17] J.-Y. Girard, *The system F of variable types, fifteen years later*, Theoretical Comput. Sci., 45 (1986), pp. 152–192.

[18] J. R. Hindley and J. P. Seldin, *Introduction to Combinators and $\lambda$-calculus*, vol. 1 of London Math. Soc. Student Texts, Cambridge Univ. Press, 1986.

[19] D. J. Howe, *Equality in lazy computation systems*, in Proceedings, Fourth Annual Symposium on Logic in Computer Science, IEEE, 1989, pp. 198–203.

[20] G. Huet and J.-J. Lévy, *Computations in nonambiguous term rewriting systems*, Tech. Report 359, INRIA, Rocqencourt, France, 1979.

[21] J. Hyland and C.-H. Ong, *Pi-calculus, dialogue games and PCF*, in Conference Record of FPCA '95: Conference on Functional Programming Languages and Computer Architecture, ACM Press, June 1995, pp. 96–107.

[22] IEEE, *Third Annual Symposium on Logic in Computer Science*, 1988.

[23] T. Jim and A. R. Meyer, *Communication in the* Types *electronic forum.* 17 June, 1989.

[24] J. W. Klop, *Combinatory reduction systems*, Tract 127, Mathematisch Centrum, Amsterdam, 1980.

[25] I. Mason and C. Talcott, *Programming, transforming, and proving with function abstractions and memories*, in Automata, Languages and Programming: *16$^{th}$* International Colloquium, G. Ausiello, M. Dezani-Ciancaglini, and S. R. D. Rocca, eds., vol. 372 of Lecture Notes in Computer Science, Springer-Verlag, 1989.

[26] A. R. Meyer, *What is a model of the lambda calculus?*, Information and Control, 52 (1982), pp. 87–122.

[27] ———, *Semantical paradigms: Notes for an invited lecture, with two appendices by Stavros Cosmadakis*, in Third Annual Symposium on Logic in Computer Science [22], pp. 236–253.

[28] R. Milner, *Fully abstract models of the typed lambda calculus*, Theoretical Comput. Sci., 4 (1977), pp. 1–22.

[29] K. Mulmuley, *Full Abstraction and Semantic Equivalence*, ACM Doctoral Dissertation Award, 1986, MIT Press, 1987.

[30] C.-H. L. Ong, *The Lazy Lambda Calculus: An Investigation into the Foundations of Functional Programming*, PhD thesis, Imperial College, University of London, 1988.

[31] G. D. Plotkin, *LCF considered as a programming language*, Theoretical Comput. Sci., 5 (1977), pp. 223–256.

[32] V. Sazonov, *Expressibility of functions in D. Scott's LCF language*, Algebra i Logika, 15 (1976), pp. 308–330. (Russian).

[33] D. S. Scott, *Continuous lattices*, in Toposes, Algebraic Geometry and Logic, F. W. Lawvere, ed., vol. 274 of Lecture Notes in Mathematics, Springer-Verlag, 1972, pp. 97–136.

[34] ———, *Data types as lattices*, SIAM J. Comput., 5 (1976), pp. 522–587.

[35] ———, *A type theoretical alternative to CUCH, ISWIM, OWHY*, Theoretical Comput. Sci., 121 (1993), pp. 411–440.

[36] S. Smith, *From operational to denotational semantics*, in Mathematical Foundations of Programming Semantics, vol. 598 of Lecture Notes in Computer Science, Springer-Verlag, 1992, pp. 54–76.

[37] A. STOUGHTON, *Fully Abstract Models of Progamming Languages*, Research Notes in Theoretical Computer Science, Pitman/Wiley, 1988. Revision of PhD thesis, Dept. of Computer Science, Univ. Edinburgh, Report No. CST–40–86, 1986.

[38] C. TALCOTT, *Programming and proving with function and control abstractions*, Tech. Report STAN–CS–89–1288, Stanford Univ., 1988.