Selective Strictness and Parametricity in Structural Operational Semantics

Janis Voigtländer

Institut für Theoretische Informatik
Technische Universität Dresden
01062 Dresden, Germany
voigt@tcs.inf.tu-dresden.de

Patricia Johann*

Department of Computer Science
Rutgers University
Camden, NJ 08102 USA
pjohann@crab.rutgers.edu

Abstract

Parametric polymorphism constrains the behavior of pure functional programs in a way that allows the derivation of interesting theorems about them solely from their types, i.e., virtually for free. The formal background of such 'free theorems' is well developed for extensions of the Girard-Reynolds polymorphic lambda calculus by *algebraic datatypes* and *general recursion*, provided the resulting calculus is endowed with either a purely strict or a purely nonstrict semantics. But modern functional languages like Clean and Haskell, while using nonstrict evaluation by default, provide means to enforce strict evaluation of subcomputations at will, and thus give the advanced programmer explicit control over the evaluation order. Such *selective strictness*, of course, does not remain without semantic consequences, and indeed breaks standard parametricity results. This paper develops an operational semantics for a core calculus representing all the language features emphasized above. Its main achievement is the characterization of observational equivalence with respect to this operational semantics by a carefully constructed logical relation. This establishes the formal basis for new parametricity results, as illustrated by several example applications including the first complete correctness proof for short cut fusion in the presence of selective strictness.

Keywords: Clean, Haskell, extensionality principles, fixpoint recursion, functional programming languages, identity extension, lambda calculus, logical relations, mixing strict and nonstrict semantics, parametric polymorphism, program transformations, *seq*, short cut fusion, theorems for free, types

1 Introduction

To support the production of software that is rapidly prototyped, reliable, and maintainable, both programmers and programming language designers need to have at their disposal techniques for reasoning effectively about program semantics. One technique which is suitable for reasoning about programs in polymorphically typed

*Supported in part by National Science Foundation grant CCF-0429072.

functional languages is based on parametricity properties [Rey83] — more colorfully known as free theorems [Wad89] — associated with polymorphic functions.[1] A *parametricity property* formalizes the intuition that a polymorphic function must behave uniformly, i.e., must use the same algorithm to compute its result regardless of the concrete type at which it is instantiated. The parametricity property for a polymorphic function can be derived solely from the type of that function, with no knowledge whatsoever of the function's actual definition; this is the sense in which parametricity properties are obtained 'for free'. But a polymorphic function is only guaranteed to satisfy its parametricity property if a *parametricity theorem* [Wad89] — originally called *abstraction theorem* [Rey83] — holds for the underlying language of which it is part. Thus parametricity properties are not actually 'free' at all: the illusion of freeness is merely a reflection of the ease with which parametricity properties can be derived *once the considerable task of proving a parametricity theorem which guarantees that they hold* has been completed.

In addition, some popular applications which are typically justified by cleverly instantiating parametricity properties actually require programming languages of interest to satisfy properties which are stronger than those guaranteed solely by their parametricity theorems. In particular, the past decade or so has seen the development of a number of parametricity-based techniques for automatically transforming modular-but-inefficient programs in nonstrict functional languages such as Haskell [Pey03] into efficient-but-monolithic equivalents [GLP93, TM95, Joh02, Sve02, Voi02, GJUV05]. Automatic transformation is essential to mitigating the inherent tension between the design and development of programs that are easy to reason about and the run-time efficiency of those programs. But the semantic correctness of parametricity-based program transformations, with respect to some notion of equivalence in which we are interested, does not always follow from parametricity theorems alone.

The key to stating and proving the parametricity theorem for a given language (or, more precisely, for a given model of a language) is to interpret the types of the language according to certain systematically-constructed relations, called *logical relations* [Plo73, Fri75, Rey83, Sta85]. A parametricity theorem then asserts that every closed term of closed type is related to itself by the relational interpretation of its type.[2] But proving correctness of parametricity-based program transformations in a given model often requires more than just reflexivity of the logical relation: the relational interpretation of each type — including ∀-types — must coincide exactly with the model's notion of equivalence at that type. When this is the case, i.e., when there is a logical relation which coincides with the notion of equivalence we are interested in, we say that we have a *parametric model* of that notion of equivalence. Constructing such a parametric model — and, therefore, proving correctness of

[1]As is standard, we take the term 'polymorphic' to refer to parametric polymorphism, as opposed to *ad hoc* polymorphism of the kind supported, for example, by type classes [WB89].

[2]More precisely, this description applies only in an operational semantics framework. For a denotational model, the parametricity theorem asserts that *the interpretation of* every closed term of closed type is related to itself by the relational interpretation of its type.

certain parametricity-based program transformations — thus entails ensuring that a strengthening of the parametricity theorem holds. The required strengthening is closely related to Reynolds' *identity extension lemma* [Rey83].

1.1 Parametric models and program transformations

The Girard-Reynolds polymorphic lambda calculus $\lambda^\forall$ [Gir72, Rey74], which is also known as 'System F' and provides the theoretical underpinning for many polymorphically typed functional languages, is well-known to admit parametric models [BFSS90, Has91, RR94]. Results derived from parametricity thus hold unconditionally in $\lambda^\forall$. But for calculi that more closely resemble modern functional languages the story is not so simple. For instance, adding a fixpoint primitive to a calculus, thus capturing general recursive definitions, weakens its parametricity properties by imposing strictness and continuity conditions on (some of the) functions appearing in those properties [Wad89, LP96]. The impact on identity extension and related strengthenings of parametricity theorems is also highly nontrivial. Moreover, to help programmers control the time and space behavior of programs, nonstrict languages often provide primitives for selectively forcing strict evaluation in computations. Such primitives can further compromise parametricity-based results such as the correctness of program transformations; see, e.g., Appendix B of [Voi02], [JV04, JV06], and Section 2 below.

To study the impact of fixpoint recursion, Pitts [Pit00] constructed an operational model for the calculus PolyPCF obtained by adding a fixpoint primitive and an algebraic datatype to $\lambda^\forall$ and endowing the result with a nonstrict semantics. Pitts proved that his model is parametric in the sense discussed above, paving the way for its use in [Joh02, Joh03, Joh05] to give the first fully satisfactory correctness proofs for *short cut fusion* [GLP93] and related program transformations. Despite recent progress in denotational semantics [RS04, BMP06, Møg06], we take this as strong evidence that the operational setting is currently the best available for studying parametricity and program transformations based on it.

Nevertheless, our studies [JV04, JV06] of the (additional) impact of selective strictness on parametricity were first performed in a denotational setting. This is primarily because working in a denotational setting allowed a more intuitive initial approach to the problem. Concretely, to understand how free theorems and parametricity-based program transformations are affected by the presence of the polymorphic strict evaluation primitive *seq* in Haskell, we started with the standardly accepted naive denotational model for Haskell and constructed a logical relation for which a parametricity theorem could be proved. We then used this logical relation to derive appropriate variations of standard free theorems and to recover (partial and total) correctness of some well-known parametricity-based program transformations in the presence of *seq*. This shed significant light on the behavior of such transformations in 'real Haskell' (as opposed to 'Haskell minus *seq*', which was the target of previous correctness arguments) by identifying preconditions under which they could safely be applied.

But despite providing important insights, the work in [JV04, JV06] has two significant shortcomings. The first is that the underlying model could not be shown to be parametric. That is, the statement corresponding to the identity extension lemma for the logical relation which we offer as witness that the naive model is parametric remains a conjecture, and the proofs regarding program transformations depend on this conjecture. The second shortcoming is that the status of the naive denotational model with respect to the observable behavior of Haskell programs is unclear. That is, the operational semantics that implementations of Haskell are, according to the language definition, expected to satisfy is not in any way tied into the denotational semantics. In particular, equivalence in the naive denotational setting between a given Haskell program and the new program obtained by applying parametricity-based transformations to it is insufficient to allow us to draw any conclusions at all about the relationship between the observable behavior of the given program and the observable behavior of the transformed program.

1.2 This paper

In this paper we study the calculus PolySeq which is obtained by adding a Haskell-like strictness primitive to Pitts' PolyPCF. Our key contribution is the construction of a *parametric model of observational equivalence* for PolySeq. Our model is similar to Pitts' operational semantics-based parametric model for PolyPCF [Pit00], but his construction has been refined to accommodate the extra constraints on relational interpretations of types imposed by selective strictness. While these constraints are in some sense 'just' operational analogues of denotational ones discussed in [JV04, JV06], their translation and incorporation into the operational semantics is delicate, and is accomplished rather differently than we anticipated in those earlier papers. Moreover, our construction starts from a small-step semantics, rather than from a big-step semantics *à la* Pitts. This allows us to more explicitly model the operational behavior of *seq*, while at the same time providing some new insights into techniques for modularly constructing parametric models of observational equivalence for extensions of $\lambda^{\forall}$ that support multiple additional language features.

Constructing a parametric model of PolySeq observational equivalence is a highly nontrivial undertaking precisely because the impact of *seq* on termination is subtle and complex. At first glance, it may appear that this impact can be accounted for by extending the operationally-based techniques of [Pit00] along the lines suggested for 'Lazy PCF' in the conclusion of that paper. But *seq* impacts the termination behavior of programs in ways that go beyond just making it possible to observe termination of whole programs at function types, or, indeed, at any type. In fact, *seq* can be used to force evaluation of *any* term of *any* type appearing at *any* place in a program, so that termination of both intermediate and 'top-level' computations of any type becomes observable. Since a parametric model of PolySeq observational equivalence must identify all terms of all types which exhibit the same observable behavior in arbitrary program contexts, this must be taken into account when defining the logical relation which witnesses parametricity of the model. We accomplish this for the model constructed in this paper by enforcing a *convergence-reflection* property at all types, in addition to appropriately restricting the relations over which quantification is performed when defining the relational interpretation of ∀-types.

An important secondary contribution of this paper is to show how the model we construct here can be used to prove the correctness, with respect to observational equivalence, of parametricity-based transformations on PolySeq programs. This will be exemplified for the classical short cut fusion technique [GLP93] and ties up the loose ends from [JV04, JV06] mentioned in the previous subsection. Although we focus in this paper on a calculus whose only algebraic datatypes are lists, all of the results here carry over to extensions of PolySeq with non-list algebraic datatypes; in particular, our results extend easily to prove correctness of short cut fusion for non-list algebraic datatypes. The techniques introduced in this paper can also be used to prove correctness of parametricity-based transformations which fuse consumers of algebraic data structures with producers parameterized over substitution values [Joh02], and which are category-theoretic duals of short cut fusion for algebraic datatypes [TM95, Sve02].

Whereas the denotational development in [JV04, JV06] was carried out in terms of an inequational notion of program definedness, the model constructed here is based on a notion of true program equivalence. This may seem odd, given that the impact of *seq* on a parametricity-based program equivalence is to potentially make one side of equivalence less defined than the other, and thus that the larger goal of our current research is to extend the model of PolySeq constructed here to accommodate an appropriate notion of semantic approximation. Such an extension would give a full porting of the results from [JV04, JV06] to the operational setting, and would transfer to the inequational setting all the advantages of working in an operationally-based model. But rather than moving from Pitts' equational model for PolyPCF to an inequational model for PolySeq in one fell swoop, we consider the impact of selective strictness on the construction of an operationally-based parametric model and the impact of moving from studying program equivalence to studying program inequivalence in such a model in two independent steps. This is reasonable in light of preliminary results which suggest that many of the same issues that arise in the inequational development for PolySeq are likely to arise in an inequational development for PolyPCF itself as well. It therefore makes sense to first isolate and identify the impact of *seq*, as we do in the present paper, in order to clearly delineate it from those other issues.

The ultimate goal of the line of research advanced in this paper is the development of tools for reasoning about parametricity properties of, and parametricity-based transformations on programs in, real programming languages rather than toy calculi. This provides another point in favor of an operational approach such as the one taken in this paper. For while the Glasgow Haskell Compiler [GHC] uses a variant of $\lambda^{\forall}$ as its intermediate language Core, a well-defined denotational semantics is not currently known even for relevant subsets of Core. It is therefore unclear whether results derived relative to any particular denotational model of, say, PolySeq would eventually shed any light at all on parametricity properties of Core. On the other hand, we derive our results in this paper relative to an operational semantics that implementations like GHC are expected to satisfy, so that the parametricity results we prove for PolySeq do indeed provide insights into those of Core and, by extension, full Haskell.

The remainder of this paper is structured as follows. Section 2 informally discusses selective strictness and how it can break parametricity. Our formal study of this issue begins in Section 3 with the syntax and semantics of PolySeq. In Section 4 we study PolySeq termination, introduce restrictions on relations to accommodate the fixpoint and selective strictness primitives, and examine their interplay. Based on the aforementioned restrictions, and indeed driven by them, we define our logical relation in Section 5. In Section 6 we prove our main technical result (Corollary 6.8), namely that the logical relation characterizes PolySeq observational equivalence. In Section 7 we use this result to establish extensionality principles (Lemmas 7.6, 7.7, and 7.10), to enumerate terms up to observational equivalence (Lemmas 7.15 and 7.17), and to prove correctness of short cut fusion (Theorem 7.18). Section 8 concludes, explores the relationship of our work to other work in the area, and offers directions for further research. Throughout, proofs that are either routine, too technical for the main part of the paper, or essentially repeated or elaborated from [Pit00] are deferred to Appendix A.

2 Selective strictness breaks parametricity

In PolyPCF, as in other nonstrict languages, function arguments are evaluated only when required. But evaluation can be explicitly forced in the presence of a strict evaluation primitive such as Haskell's *seq*. This primitive is denotationally specified as follows in the language definition [Pey03]:[3]

$$
\begin{aligned}
&\mathit{seq} :: \forall \alpha\ \beta.\ \alpha \to \beta \to \beta \\
&\mathit{seq}\ \bot\ b = \bot \\
&\mathit{seq}\ \ a\ b = \ b \quad \text{if } a \neq \bot
\end{aligned}
$$

Here $\bot$ is the undefined value corresponding to a nonterminating computation or a runtime error, such as might be obtained as the result of a failed pattern match. The operational behavior of *seq* is to evaluate its first argument before returning its second argument. Note that *seq* can be applied at all types.

[3] Actually, the type given there for *seq* is just $\alpha \to \beta \to \beta$. But the semantics of α and β occurring free therein is exactly an implicit universal quantification. For clarity, we prefer to make all quantification over type variables explicit. This is supported by most Haskell implementations via the keyword **forall**. Note that while adding or omitting outermost quantifications (as here for *seq*'s type) is just a matter of syntactic convenience, the positioning of 'inner $\forall$s' is crucial for functions like *build* in Figure 1 below (see also the footnote on the next page).

A prototypical example of a function which uses *seq* — in fact, the one which is probably discussed most frequently on the Haskell Mailing List [HML] — is:

$$\begin{aligned}
&\mathit{foldl}' :: \forall\alpha\ \beta.\ (\beta \to \alpha \to \beta) \to \beta \to [\alpha] \to \beta\\
&\mathit{foldl}'\ f\ z\ []\qquad\quad = z\\
&\mathit{foldl}'\ f\ z\ (h : t) = \mathit{seq}\ z'\ (\mathit{foldl}'\ f\ z'\ t)\\
&\qquad\qquad\qquad\quad\ \ \mathbf{where}\ z' = f\ z\ h
\end{aligned}$$

Here *seq* ensures that the accumulating parameter is computed immediately in each recursive step rather than constructing a complex closure, representing the overall accumulation, which would be computed only at the end of the call to foldl'. Thus, in many situations foldl' offers a useful and easily obtained efficiency improvement over the Haskell prelude function *foldl*. Further examples of programs which make use of selective strictness via *seq* can be found in [THLP98]. Note that other means of explicitly introducing strictness in Haskell programs — e.g., strict datatypes, the strict application function \$!, and the recently introduced bang patterns — are all definable in terms of *seq*. Similarly, language features for selective strictness in Clean [CLR] are interdefinable with *seq* [EM06].

The impact of selective strictness on the semantics of, and reasoning techniques for, languages like Clean and Haskell can be severe. This impact has been studied, for example, in [HK05] and [EM06], as well as in our own recent work [JV04, JV06]. It has also been noticed somewhat more in passing in [Voi02], [DJ04], and [DHJG06]. That the mixture of nonstrict and strict evaluation is currently a topic of great interest in programming languages research is further evidenced by recent work in the areas of program verification [ABB$^+$05] and implementation [RMP06]. Our specific focus in this paper is, of course, on the impact of selective strictness on parametric polymorphism in nonstrict languages.

The classic example of a parametricity-based program transformation is the short cut fusion rule [GLP93]. This rule eliminates intermediate lists from compositions of list producers written in terms of *build* and list consumers written in terms of *foldr* using the following rule for appropriately typed g, c, and n:

$$\mathit{foldr}\ c\ n\ (\mathit{build}\ g) = g\ c\ n \tag{1}$$

Definitions of *foldr* and *build* are given in Figure 1. The function *foldr*, which is standard in the Haskell prelude, takes as input a function c, a value n, and a list l, and produces a value by replacing all occurrences of (:) in l by c and any occurrence of [] in l by n. For instance, $\mathit{foldr}\ (+)\ 0\ l$ sums the (numeric) elements of the list l. The function *build*, on the other hand, takes as input a polymorphic function g providing a type-independent template for constructing 'abstract' lists, and applies it to the list constructors (:) and [] to get a corresponding 'concrete' list.[4] For example, $\mathit{build}\ (\lambda\, c\ n \to c\ 4\ (c\ 9\ n))$ produces the list [4,9]. Applying rule (1)

[4] Taking a polymorphic function as argument, *build* has a *rank-2 type* [Lei83]. While such higher-rank types are not covered by the current Haskell standard [Pey03], they are actually supported by most implementations. For recent work on type inference in this setting see [VWP06].

to a composition matching its left-hand side yields a corresponding instance of the right-hand side which avoids constructing intermediate lists produced by *build* g and immediately consumed by *foldr* c n. This is accomplished by applying g to the (:)- and []-replacement functions c and n directly.

$$\begin{array}{l} \mathit{foldr} :: \forall\alpha\ \beta.\ (\alpha \to \beta \to \beta) \to \beta \to [\alpha] \to \beta \\ \mathit{foldr}\ c\ n = f\ \textbf{where}\ \begin{array}{ll} f\ [] & = n \\ f\ (h : t) & = c\ h\ (f\ t) \end{array} \end{array}$$

$$\begin{array}{l} \mathit{build} :: \forall\alpha.\ (\forall\beta.\ (\alpha \to \beta \to \beta) \to \beta \to \beta) \to [\alpha] \\ \mathit{build}\ g = g\ (:)\ [] \end{array}$$

Figure 1: Haskell functions for short cut fusion.

The short cut fusion rule is derived from the parametricity property for g, or, more accurately, for g's type. But the simple instantiation in which $g = \mathit{seq}$, $c = \bot$, and $n = 0$ shows that (1) is no longer an equivalence if *seq* is present. The intuitive reason for this breakdown of short cut fusion lies in the differences in definedness and strictness properties of the arguments supplied to g before and after applying the short cut fusion rule. The list constructors (:) and [] passed to g by *build* are both non-$\bot$, and so is any value obtained by combining them. But since no such *a priori* guarantees exist for their replacement functions c and n, the same use of *seq* inside g might result in the undefined value $\bot$ on the right-hand side of (1) and a non-$\bot$ value on the left-hand side.

Before publication of [JV04, JV06], a folk theorem had long held that parametricity properties remain valid in the presence of *seq* if all of the functions appearing in them (where one is free to make a choice) are strict and total. But as shown there, this is not the case: although strictness and totality of the consumer (*foldr* c n) just happen to be sufficient for recovering correctness of the short cut fusion rule when *seq* is present, they are not enough to recover the parametricity properties of all polymorphic functions in this situation. In fact, it is probably this happenstance vis-a-vis the short cut fusion rule that is responsible for the failure of the folklore approach to parametricity in the presence of *seq* having gone unnoticed for so long. In [JV04, JV06] we gave constraints which guarantee that parametricity properties of polymorphic functions hold in the denotational setting considered there, even in the presence of *seq*. In this paper we translate these constraints to the operational setting and show how their operational counterparts can be used to recover parametricity properties of polymorphic functions — as well as correctness of program transformations based on parametricity — without the shortcomings of the denotationally-based development of [JV04, JV06] discussed in Section 1.1. Our results for short cut fusion in particular can be found in Section 7.5.

3 PolySeq

As a testbed for exploring the impact of selective strictness on parametricity results in an operational setting we use a concrete programming language, PolySeq. It is an extension of the Girard-Reynolds calculus $\lambda^\forall$ by an algebraic datatype of lists and by primitives for general recursion and selective strictness that can be applied at all types. As in $\lambda^\forall$ and PolyPCF, and differently than in Clean and Haskell, all typing is explicit in the syntax of terms. That is, lambda-bound variables always come with an attached type, and with regard to polymorphism both type generalization and specialization are made explicit. Another, purely syntactic difference from Clean and Haskell is that there is only one mechanism to perform pattern matching, namely by case expressions. And rather than using recursive function equations, recursion is made explicit using a fixpoint primitive in the standard way. As an example, consider the following 'translation' of the Haskell function *foldl′* from the introduction:

$$\begin{aligned}
\mathit{foldl'} = \Lambda\alpha.\Lambda\beta.\mathbf{fix}(\lambda g :: (\beta \to \alpha \to \beta) \to \beta \to \alpha\text{-}\mathit{list} \to \beta.\lambda f :: \beta \to \alpha \to \beta.\lambda z :: \beta.\\
\lambda l :: \alpha\text{-}\mathit{list}.\mathbf{case}\ l\ \mathbf{of}\ \{\mathbf{nil}\ \Rightarrow z;\\
h : t \Rightarrow \mathbf{seq}(\underline{f\ z\ h}, g\ f\ (\underline{f\ z\ h})\ t)\})
\end{aligned}$$

Note that the sharing of the two underlined expressions, which was present in the Haskell version via the binding '**where** $z' = f\ z\ h$', is lost in PolySeq, which does not provide any sharing construct. But this difference has no impact at all on any semantic properties we are going to study. For while implementations of Haskell typically apply a lazy evaluation strategy, the language definition [Pey03] only mandates that the semantics be nonstrict (apart from where selective strictness is used, of course), without committing to either call-by-name or call-by-need. Clearly, such a commitment is unnecessary, as it would have no impact on the observable behavior of programs. Indeed, choosing between call-by-name and call-by-need can only impact program properties regarding time and space usage, neither of which is under study here or specified in [Pey03]. Put differently, the semantics of *foldl′* in Haskell is invariant under replacing its second defining equation from the introduction by

$$\mathit{foldl'}\ f\ z\ (h : t) = \mathit{seq}\ (\underline{f\ z\ h})\ (\mathit{foldl'}\ f\ (\underline{f\ z\ h})\ t)\,,$$

so it makes no sense to complicate PolySeq by modeling a sharing construct.

3.1 Syntax and typing

The syntax of PolySeq types and terms is given in Figure 2, where α and x range over disjoint countably infinite sets of *type variables* and *term variables*, respectively. The only difference to Figure 1 in [Pit00], apart from the slightly different notation, is the addition of the new term former **seq**. Alongside τ and M, we later also let σ and A, B, C, F, G, H, L, N, R, T, and V range over the syntactic categories of types and terms, respectively. Moreover, β and c, f, g, h, l, n, t, and y are later used as additional type and term variables, respectively, and all the mentioned conventions apply to versions with indices or primes as well. To reduce the need for brackets, function types and function applications are read right- and left-associative, respectively, so that $\tau_1 \to \tau_2 \to \tau_3$ means $\tau_1 \to (\tau_2 \to \tau_3)$, while $F\ A\ B$ means $(F\ A)\ B$. The constructions $\forall\alpha.-$, $\lambda x :: \tau.-$, $\Lambda\alpha.-$, and $\mathbf{case}\ M\ \mathbf{of}\ \{\mathbf{nil} \Rightarrow M';\ x : x' \Rightarrow -\}$ are binders for α, x, and x'. We identify types and terms up to renaming of bound (type and term) variables. The concept of free variables in a type or term is defined in the usual way. For example, α is a free variable in $\mathbf{nil}_\alpha$, but not in $\Lambda\alpha.\mathbf{nil}_\alpha$. We write *Typ* for the set of closed types, that is, those having no free variables. The result of capture-avoiding substitution of a type τ' for all free occurrences of a type variable α in a type τ or a term M is denoted by $\tau[\tau'/\alpha]$ or $M[\tau'/\alpha]$, respectively. Similarly, $M[M'/x]$ denotes the result of capture-avoiding substitution of a term M' for all free occurrences of a term variable x in a term M. Additionally, we use substitution for lists of distinct variables (e.g., $M_2[H/h, T/t]$ and $\tau[\vec{\tau}/\vec{\alpha}]$) and substitution for type and term variables at once (e.g., $M[\vec{\sigma}/\vec{\alpha}, \vec{N}/\vec{x}]$).

Types	τ	$::=$	α	type variable
		$\mid$	$\tau \to \tau$	function type
		$\mid$	$\forall\alpha.\tau$	$\forall$-type
		$\mid$	τ-*list*	list type
Terms	M	$::=$	x	term variable
		$\mid$	$\lambda x :: \tau.M$	function abstraction
		$\mid$	$M\ M$	function application
		$\mid$	$\Lambda\alpha.M$	type generalization
		$\mid$	M_τ	type specialization
		$\mid$	$\mathbf{nil}_\tau$	empty list
		$\mid$	$M : M$	non-empty list
		$\mid$	$\mathbf{case}\ M\ \mathbf{of}\ \{\mathbf{nil} \Rightarrow M;\ x : x \Rightarrow M\}$	case expression
		$\mid$	$\mathbf{fix}(M)$	fixpoint recursion
		$\mid$	$\mathbf{seq}(M, M)$	strictness primitive

Figure 2: Syntax of the PolySeq language.

Types are assigned to (some) terms according to the axioms and rules in Figure 3, where Γ ranges over *typing environments* of the form $\vec{\alpha}, x_1 :: \tau_1, \ldots, x_m :: \tau_m$ for a finite list $\vec{\alpha}$ of distinct type variables, $m \in \mathbb{N}$, a list $\vec{x} = x_1, \ldots, x_m$ of distinct term variables, and types $\tau_1, \ldots, \tau_m$ whose free variables are in $\vec{\alpha}$. The only difference of note to Figure 2 in [Pit00] is the addition of the typing rule for **seq**. In a typing judgement of the form $\Gamma \vdash M :: \tau$, with Γ as above, we require that M's free variables are in $\vec{\alpha}, \vec{x}$ and that τ's free variables are in $\vec{\alpha}$. As in [Pit00], the explicit type information in the syntax of function abstractions and empty lists ensures that for every Γ and M there is at most one τ with $\Gamma \vdash M :: \tau$. Given $\tau \in Typ$, we write $Term(\tau)$ for the set of terms M for which $\varnothing \vdash M :: \tau$ is derivable, where $\varnothing$ is the empty typing environment. Further, we set $Term = \bigcup_{\tau \in Typ} Term(\tau)$.

$$\Gamma, x :: \tau \vdash x :: \tau$$

$$\frac{\Gamma, x :: \tau \vdash M :: \tau'}{\Gamma \vdash (\lambda x :: \tau.M) :: \tau \to \tau'} \qquad \frac{\Gamma \vdash F :: \tau \to \tau' \qquad \Gamma \vdash A :: \tau}{\Gamma \vdash F\ A :: \tau'}$$

$$\frac{\alpha, \Gamma \vdash M :: \tau}{\Gamma \vdash \Lambda\alpha.M :: \forall\alpha.\tau} \qquad \frac{\Gamma \vdash G :: \forall\alpha.\tau}{\Gamma \vdash G_{\tau'} :: \tau[\tau'/\alpha]}$$

$$\Gamma \vdash \mathbf{nil}_\tau :: \tau\text{-}\mathit{list} \qquad \frac{\Gamma \vdash H :: \tau \qquad \Gamma \vdash T :: \tau\text{-}\mathit{list}}{\Gamma \vdash (H : T) :: \tau\text{-}\mathit{list}}$$

$$\frac{\Gamma \vdash L :: \tau\text{-}\mathit{list} \qquad \Gamma \vdash M_1 :: \tau' \qquad \Gamma, h :: \tau, t :: \tau\text{-}\mathit{list} \vdash M_2 :: \tau'}{\Gamma \vdash \mathbf{case}\ L\ \mathbf{of}\ \{\mathbf{nil} \Rightarrow M_1;\ h : t \Rightarrow M_2\} :: \tau'}$$

$$\frac{\Gamma \vdash F :: \tau \to \tau}{\Gamma \vdash \mathbf{fix}(F) :: \tau} \qquad \frac{\Gamma \vdash A :: \tau \qquad \Gamma \vdash B :: \tau'}{\Gamma \vdash \mathbf{seq}(A, B) :: \tau'}$$

Figure 3: PolySeq type assignment relation.

3.2 Operational semantics

Our semantics for PolySeq follows Plotkin's style of structural operational semantics [Plo04]. In particular, and in contrast to [Pit00], we start from a small-step rather than from a big-step formulation. As pointed out below, the two approaches are equivalent in a precise sense, and indeed [Pit00] also makes essential use of a small-step semantics: indirectly in the proof of Theorem 3.6 and (thus) in the structural termination relation $\top$. Working directly with a small-step semantics avoids such indirection (and the proof burden that comes with it), and it is interesting to see that an approach without any big-step overlay is itself sufficient for constructing the desired parametric model.

PolySeq *values* are given by the following grammar:

$$V ::= \lambda x :: \tau.M \mid \Lambda\alpha.M \mid \mathbf{nil}_\tau \mid M : M.$$

Note that adding **seq** to a language does not introduce any new values. The subset of *Term* consisting of all elements that respect the above grammar is denoted by *Value*. Further, given $\tau \in \mathit{Typ}$, we set $\mathit{Value}(\tau) = \mathit{Value} \cap \mathit{Term}(\tau)$.

The remaining ingredients for setting up a small-step semantics are redex/reduct-pairs and a notion of reduction in context. The former are just as in the proof of Theorem 3.6 in [Pit00], except that an appropriate pair involving **seq** is added.

Definition 3.1. Let $\tau \in \mathit{Typ}$ and $R, R' \in \mathit{Term}(\tau)$. We write $R \leadsto R'$ for the following pairs:

R	R'	if
$(\lambda x :: \tau'.N)\ A$	$N[A/x]$	$x :: \tau' \vdash N :: \tau$
$(\Lambda\alpha.N)_{\tau'}$	$N[\tau'/\alpha]$	$\alpha \vdash N :: \tau''$
$\mathbf{case}\ \mathbf{nil}_{\tau'}\ \mathbf{of}\ \{\mathbf{nil} \Rightarrow M;\ h : t \Rightarrow M'\}$	M	$h :: \tau', t :: \tau'\text{-}\mathit{list} \vdash M' :: \tau$
$\mathbf{case}\ H : T\ \mathbf{of}\ \{\mathbf{nil} \Rightarrow M;\ h : t \Rightarrow M'\}$	$M'[H/h, T/t]$	$h :: \tau', t :: \tau'\text{-}\mathit{list} \vdash M' :: \tau$
$\mathbf{fix}(F)$	$F\ \mathbf{fix}(F)$	
$\mathbf{seq}(V, M)$	M	$V \in \mathit{Value}$,

where x, h, and t are term variables, α is a type variable, $\tau' \in \mathit{Typ}$, $A, H \in \mathit{Term}(\tau')$, $M \in \mathit{Term}(\tau)$, $T \in \mathit{Term}(\tau'\text{-}\mathit{list})$, $F \in \mathit{Term}(\tau \to \tau)$, and the further types and terms that occur in the table are subject to the restrictions recorded on the right. $\diamond$

It is essential that V is a value in the last pair above, because otherwise one would not ensure the intended semantics of **seq**, which is to first evaluate its first argument before reducing to the second.

To describe reduction in context, we use the machinery of evaluation contexts introduced in [FFKD87]. Following [HS97], we represent these contexts as stacks of evaluation frames. Compared to [Pit00], an appropriate additional kind of evaluation frame is introduced to account for **seq**.

Definition 3.2. The grammar for *evaluation frame stacks* is

$$S ::= \mathit{Id} \mid S \circ E\,,$$

where E ranges over *evaluation frames*:

$$E ::= (-\ M) \mid -_\tau \mid (\mathbf{case}\ -\ \mathbf{of}\ \{\mathbf{nil} \Rightarrow M;\ x : x \Rightarrow M\}) \mid \mathbf{seq}(-, M).$$

If a stack comprises a single evaluation frame E, then we denote it by E rather than $\mathit{Id} \circ E$. Moreover, given an evaluation frame E and a term M, we write $E\{M\}$ for the term that results from replacing '$-$' by M in E. $\diamond$

Argument and result types are assigned to (some) evaluation frame stacks according to the axiom and rules in Figure 4, where Γ again ranges over typing environments, with well-formedness conditions similar to those for term typing judgements. The only difference of note to Figure 6 in [Pit00] is the addition of the typing rule for the new evaluation frame. As in [Pit00], for every Γ, S, and τ there is at most one τ' with $\Gamma \vdash S :: \tau \multimap \tau'$; this satisfies all needs for type uniqueness we will encounter. Given $\tau, \tau' \in \mathit{Typ}$, we write $\mathit{Stack}(\tau, \tau')$ for the set of evaluation frame stacks S for which $\varnothing \vdash S :: \tau \multimap \tau'$ is derivable. Since we will later want to restrict our attention to evaluation frame stacks which return results of list type, we set, for every $\tau \in \mathit{Typ}$, $\mathit{LStack}(\tau) = \bigcup_{\tau' \in \mathit{Typ}} \mathit{Stack}(\tau, \tau'\text{-}\mathit{list})$.

The (typed) operations of concatenating two evaluation frame stacks and of applying an evaluation frame stack to a term are given as follows.

$$\Gamma \vdash \mathit{Id} :: \tau \multimap \tau$$

$$\frac{\Gamma \vdash S :: \tau' \multimap \tau'' \qquad \Gamma \vdash A :: \tau}{\Gamma \vdash S \circ (-\ A) :: (\tau \to \tau') \multimap \tau''} \qquad \frac{\Gamma \vdash S :: \tau[\tau'/\alpha] \multimap \tau''}{\Gamma \vdash S \circ -_{\tau'} :: (\forall \alpha.\tau) \multimap \tau''}$$

$$\frac{\Gamma \vdash S :: \tau' \multimap \tau'' \qquad \Gamma \vdash M_1 :: \tau' \qquad \Gamma, h :: \tau, t :: \tau\text{-}\mathit{list} \vdash M_2 :: \tau'}{\Gamma \vdash S \circ (\mathbf{case}\ -\ \mathbf{of}\ \{\mathbf{nil} \Rightarrow M_1;\ h : t \Rightarrow M_2\}) :: \tau\text{-}\mathit{list} \multimap \tau''}$$

$$\frac{\Gamma \vdash S :: \tau' \multimap \tau'' \qquad \Gamma \vdash B :: \tau'}{\Gamma \vdash S \circ \mathbf{seq}(-, B) :: \tau \multimap \tau''}$$

Figure 4: Typing evaluation frame stacks.

Definition 3.3. Let $\tau \in \mathit{Typ}$. Given $S \in \mathit{LStack}(\tau)$, we define for every $\tau' \in \mathit{Typ}$ and $S' \in \mathit{Stack}(\tau', \tau)$ the concatenation $(S \mathbin{@} S') \in \mathit{LStack}(\tau')$ by induction on the structure of S' as follows:

$$\begin{array}{lcl} S \mathbin{@} \mathit{Id} & = & S \\ S \mathbin{@} (S'' \circ E) & = & (S \mathbin{@} S'') \circ E. \end{array}$$

Moreover, we define for every $\tau' \in \mathit{Typ}$, $S \in \mathit{Stack}(\tau', \tau)$, and $M \in \mathit{Term}(\tau')$ the application $(S\ M) \in \mathit{Term}(\tau)$ by induction on the structure of S as follows:

$$\begin{array}{lcl} \mathit{Id}\ M & = & M \\ (S' \circ E)\ M & = & S'\ (E\{M\}). \end{array}$$

◇

The transition relation induced by the choice of values, redex/reduct-pairs, and evaluation frames is defined in the following (standard) way.

Definition 3.4. Let $\tau_1, \tau_2, \tau' \in \mathit{Typ}$, $S_1 \in \mathit{Stack}(\tau_1, \tau')$, $M_1 \in \mathit{Term}(\tau_1)$, $S_2 \in \mathit{Stack}(\tau_2, \tau')$, and $M_2 \in \mathit{Term}(\tau_2)$. We write $(S_1, M_1) \rightarrowtail (S_2, M_2)$ for the following pairs:

(S_1, M_1)	(S_2, M_2)	if
$(S, E\{N\})$	$(S \circ E, N)$	$N \notin \mathit{Value}$
$(S \circ E, V)$	$(S, E\{V\})$	$V \in \mathit{Value}$
(S, R)	(S, R')	$R \rightsquigarrow R'$,

where S is an evaluation frame stack, E is an evaluation frame, and the terms that occur in the table are subject to the restrictions recorded on the right. ◇

Intuitively, the first two transition rules navigate a term to detect the next redex to be reduced, while the third rule performs a small-step reduction in a given evaluation context. Note that $\rightarrowtail$ is deterministic, but not terminating (due to **fix**). We denote by $\rightarrowtail^t$, with $t \in \mathbb{N}$, the t-fold composition of $\rightarrowtail$, and by $\rightarrowtail^*$ its reflexive, transitive closure. Evaluation of a term to a value is then captured as follows.

Definition 3.5. Given $M \in \mathit{Term}$ and $V \in \mathit{Value}$ of the same type, we write $M \Downarrow V$ if $(\mathit{Id}, M) \rightarrowtail^* (\mathit{Id}, V)$. Given $M \in \mathit{Term}$, we write $M{\Downarrow}$ if there is some V with $M \Downarrow V$, and $M{\Uparrow}$ otherwise. In the former case we say that M *converges*, and in the latter we say that it *diverges*. Note that every value converges. $\diamond$

Thus, we have provided a stack-based abstract machine for PolySeq. The evaluation relation $\Downarrow$ induced by this machine is the same as the one we would obtain via defining a big-step semantics by adding the rule

$$\frac{A \Downarrow V \qquad B \Downarrow V'}{\mathbf{seq}(A, B) \Downarrow V'}$$

to Figure 3 in [Pit00]. The proof of this fact is very similar to that of (3) inside the proof of Theorem 3.6 in [Pit00]; we do not give it here. Note that the small-step semantics is as good an operational foundation for reasoning about the impact of **seq** as is a big-step one. And since it is a bit more low-level, the small-step semantics more immediately reflects the operational behavior of **seq** in actual Haskell implementations.[5] That is because the evaluation frame $\mathbf{seq}(-, M)$ and the reduction $\mathbf{seq}(V, M) \rightsquigarrow M$ make it explicit that **seq** first evaluates its first argument, before turning to the second one, while no such order is imposed in the above big-step rule.

Before moving on to the intended notion of operational equivalence, we give three observations about termination issues and the existence of a 'polymorphic bottom'. The first two observations follow easily from the definitions of $\Downarrow$ and $\rightarrowtail$, the third one arises by combination of the first two.

Observation 3.6. For every $\tau \in \mathit{Typ}$: $\mathbf{fix}(\lambda x :: \tau.x){\Uparrow}$. ■

Observation 3.7. For every $R, R' \in \mathit{Term}$, if $R \rightsquigarrow R'$, then $R{\Downarrow} \Leftrightarrow R'{\Downarrow}$. ■

Observation 3.8. Let $\Omega = \Lambda\alpha.\mathbf{fix}(\lambda x :: \alpha.x) \in \mathit{Term}(\forall\alpha.\alpha)$. While $\Omega{\Downarrow}$, for every $\tau \in \mathit{Typ}$: $\Omega_\tau{\Uparrow}$. ■

Being able to describe to what value, if any, a term evaluates is usually not enough to reason about the operational behavior of a programming language. In particular, stipulating that two terms are to be considered equivalent (if and) only if both evaluate to the same value is a much too strong requirement. It would outlaw, for example, consideration of quicksort and heapsort implementations as semantically equivalent, on the grounds that their representations as function abstractions would necessarily be different. Such a situation would be highly undesirable, given that two different algorithms performing the same computational task (such as sorting a list) should clearly be equated by any reasonable semantics. For this reason, it is standard to allow only *particular observations* to be made about terms in the language (with comparison of function abstraction representations not being one of

[5] Actually, in Haskell **seq** is not just a term former with two mandatory arguments, as in PolySeq, but itself a valid term of type $\forall\alpha.\forall\beta.\alpha \to \beta \to \beta$. Of course, such a term can also be formed from our primitive as $\mathit{seq} = \Lambda\alpha.\Lambda\beta.\lambda x :: \alpha.\lambda y :: \beta.\mathbf{seq}(x, y)$, which then behaves like Haskell's *seq*.

them), but to require that two terms are considered equivalent (if and) only if they lead to the same observations *in every possible context*.

Following [Pit00], we choose evaluation of terms of list type to the empty list as the only observation possible for PolySeq. Actually, due to the presence of **seq**, this allows the observation of termination at arbitrary types, in a sense later made precise in Corollary 4.16. So in contrast to the situation in the setting without **seq**, the initial choice to observe termination only at list types has no impact on the derived notion of observational equivalence here. The important point is that we can still observe only termination at arbitrary types, and not the full representation of any obtained value. To capture observations in context, we again follow the treatment in [Pit00]. That is, we specify a number of desirable properties our notion of semantic equality for PolySeq should have, and then ask for the largest relation with those properties. Of course, reasoning about semantic equality only makes sense if we have at least an equivalence relation, i.e., one that is reflexive, transitive, and symmetric. But in addition to that, the intended notion of equality should also be a congruence, i.e., should be compatible with all term formers and with substitution, in a sense made precise below. This is where the 'context closure' of observational equivalence is ensured. To tie in the possible observations themselves, we impose an adequacy property, reflecting the above discussion about '**nil**-termination'.

Definition 3.9. Let the relation $\mathcal{E}$ comprise 4-tuples of the form (Γ, M, M', τ) with $\Gamma \vdash M :: \tau$ and $\Gamma \vdash M' :: \tau$. We write $\Gamma \vdash M\ \mathcal{E}\ M' :: \tau$ when the tuple (Γ, M, M', τ) is in $\mathcal{E}$, and we abbreviate this to $M\ \mathcal{E}\ M'$ if $\Gamma = \varnothing$ since τ is then uniquely determined as the closed type of both M and M'.

1. $\mathcal{E}$ is *adequate* if for every $\tau \in \mathit{Typ}$ and $L, L' \in \mathit{Term}(\tau\text{-}\mathit{list})$:

$$L\ \mathcal{E}\ L' \Rightarrow (L \Downarrow \mathbf{nil}_\tau \Leftrightarrow L' \Downarrow \mathbf{nil}_\tau).$$

2. $\mathcal{E}$ is *compatible* if it is closed under the axioms and rules in Figure 5, which differs from Figure 4 in [Pit00] only by the new rule for **seq**.

3. $\mathcal{E}$ is *substitutive* if it is closed under the rules in Figure 6, where $\Gamma[\tau'/\alpha]$ is the typing environment obtained from Γ by replacing every $x :: \sigma$ therein by $x :: \sigma[\tau'/\alpha]$.

4. $\mathcal{E}$ is *reflexive* if for every environment Γ, term M, and type τ with $\Gamma \vdash M :: \tau$:

$$\Gamma \vdash M\ \mathcal{E}\ M :: \tau.$$

5. $\mathcal{E}$ is *transitive* if $\mathcal{E};\mathcal{E} \subseteq \mathcal{E}$, where *relation composition* $\mathcal{E}_1;\mathcal{E}_2$ is defined by:

$$\Gamma \vdash M\ (\mathcal{E}_1;\mathcal{E}_2)\ M' :: \tau \Leftrightarrow \exists M''.\ \Gamma \vdash M\ \mathcal{E}_1\ M'' :: \tau \wedge \Gamma \vdash M''\ \mathcal{E}_2\ M' :: \tau.$$

6. $\mathcal{E}$ is *symmetric* if $\mathcal{E}^{-1} \subseteq \mathcal{E}$, where *relation reciprocation* $\mathcal{E}^{-1}$ is defined by:

$$\Gamma \vdash M\ \mathcal{E}^{-1}\ M' :: \tau \Leftrightarrow \Gamma \vdash M'\ \mathcal{E}\ M :: \tau. \qquad \diamond$$

$$\Gamma, x :: \tau \vdash x \ \mathcal{E} \ x :: \tau$$

$$\frac{\Gamma, x :: \tau \vdash M \ \mathcal{E} \ M' :: \tau'}{\Gamma \vdash (\lambda x :: \tau.M) \ \mathcal{E} \ (\lambda x :: \tau.M') :: \tau \to \tau'}$$

$$\frac{\Gamma \vdash F \ \mathcal{E} \ F' :: \tau \to \tau' \qquad \Gamma \vdash A \ \mathcal{E} \ A' :: \tau}{\Gamma \vdash (F \ A) \ \mathcal{E} \ (F' \ A') :: \tau'}$$

$$\frac{\alpha, \Gamma \vdash M \ \mathcal{E} \ M' :: \tau}{\Gamma \vdash \Lambda\alpha.M \ \mathcal{E} \ \Lambda\alpha.M' :: \forall\alpha.\tau} \qquad \frac{\Gamma \vdash G \ \mathcal{E} \ G' :: \forall\alpha.\tau}{\Gamma \vdash G_{\tau'} \ \mathcal{E} \ G'_{\tau'} :: \tau[\tau'/\alpha]}$$

$$\Gamma \vdash \mathbf{nil}_\tau \ \mathcal{E} \ \mathbf{nil}_\tau :: \tau\text{-}\mathit{list} \qquad \frac{\Gamma \vdash H \ \mathcal{E} \ H' :: \tau \qquad \Gamma \vdash T \ \mathcal{E} \ T' :: \tau\text{-}\mathit{list}}{\Gamma \vdash (H : T) \ \mathcal{E} \ (H' : T') :: \tau\text{-}\mathit{list}}$$

$$\frac{\Gamma \vdash L \ \mathcal{E} \ L' :: \tau\text{-}\mathit{list} \qquad \Gamma \vdash M_1 \ \mathcal{E} \ M_1' :: \tau' \qquad \Gamma, h :: \tau, t :: \tau\text{-}\mathit{list} \vdash M_2 \ \mathcal{E} \ M_2' :: \tau'}{\begin{array}{c}\Gamma \vdash (\mathbf{case} \ L \ \mathbf{of} \ \{\mathbf{nil} \Rightarrow M_1; \ h : t \Rightarrow M_2\}) \\ \mathcal{E} \ (\mathbf{case} \ L' \ \mathbf{of} \ \{\mathbf{nil} \Rightarrow M_1'; \ h : t \Rightarrow M_2'\}) :: \tau'\end{array}}$$

$$\frac{\Gamma \vdash F \ \mathcal{E} \ F' :: \tau \to \tau}{\Gamma \vdash \mathbf{fix}(F) \ \mathcal{E} \ \mathbf{fix}(F') :: \tau} \qquad \frac{\Gamma \vdash A \ \mathcal{E} \ A' :: \tau \qquad \Gamma \vdash B \ \mathcal{E} \ B' :: \tau'}{\Gamma \vdash \mathbf{seq}(A, B) \ \mathcal{E} \ \mathbf{seq}(A', B') :: \tau'}$$

Figure 5: Compatibility properties.

$$\frac{\alpha, \Gamma \vdash M \ \mathcal{E} \ M' :: \tau}{\Gamma[\tau'/\alpha] \vdash M[\tau'/\alpha] \ \mathcal{E} \ M'[\tau'/\alpha] :: \tau[\tau'/\alpha]}$$

$$\frac{\Gamma, x :: \tau \vdash M \ \mathcal{E} \ M' :: \tau' \qquad \Gamma \vdash N \ \mathcal{E} \ N' :: \tau}{\Gamma \vdash M[N/x] \ \mathcal{E} \ M'[N'/x] :: \tau'}$$

Figure 6: Substitutivity properties.

It is easy to see that every compatible relation $\mathcal{E}$ is also reflexive. Our intended notion of equivalence is the largest relation satisfying all six properties from Definition 3.9. We call it *observational equivalence* and write it as $=_{obs}$. Its existence, however, is only stipulated for now. It could be proved in a direct manner here by characterizing $=_{obs}$ as the union of all adequate, compatible, and substitutive relations. But since we are interested in a more constructive and ultimately more useful characterization, we defer the proof of the existence of $=_{obs}$ to Theorem 6.7, where it is characterized by a relation inductively derived based on the type structure of PolySeq. Of course, this means that up to that point in Section 6 we may not assume anything about $=_{obs}$, and we will not do so. When we finally have returned to $=_{obs}$, we will often use its reflexivity, transitivity, and symmetry without explicit mention.

4 PolySeq termination

In this section we study various aspects of termination in PolySeq. These are essential for characterizing observational equivalence in the presence of **fix** and **seq** in the way we aim to do. Regarding **fix**, it is long known [Wad89, LP96] that parametricity can only be achieved by restricting attention to relations that are admissible in a sense corresponding to the concepts of strictness and continuity in denotational semantics. The main technical contribution of [Pit00] was an account of such admissibility in the operational setting, based on a closure operator arising from **nil**-termination. As we will demonstrate, adding **seq** does not break any of that machinery (in particular, Lemma 4.14 can still be established below), but it requires more. The reason for the latter is that with a strictness primitive that can be applied at all types, new restrictions must be imposed on relational interpretations of types. This was already observed in [LP96] and [PLST98], and made more rigorous in [JV04, JV06], but all in a denotational setting only. Here we present a corresponding operational account (Definition 4.17), show how it interacts with the aforementioned operational machinery for fixpoint admissibility (Lemma 4.18), and show how it guarantees the key property needed to achieve parametricity in the presence of **seq** (Lemma 4.19).

4.1 General properties of nil-termination

First, we fix a special notation for expressing that evaluating a particular term in a particular context described by an evaluation frame stack leads to the empty list.

Definition 4.1. Let $\tau, \tau' \in \mathit{Typ}$, $S \in \mathit{Stack}(\tau, \tau'\text{-}\mathit{list})$, and $M \in \mathit{Term}(\tau)$. We write $S \top M$ if $(S, M) \rightarrowtail^* (\mathit{Id}, \mathbf{nil}_{\tau'})$. ◇

Note that, rather than taking the described behavior as *definition* of $\top$, Pitts defines $\top$ via a syntactic system of structural rules (see Figure 7 of [Pit00]) and then *proves* that $S \top M$ if and only if $(S, M) \rightarrowtail^* (\mathit{Id}, \mathbf{nil}_{\tau'})$ inside his Theorem 3.6. Our approach is simpler in that it obviates the need for an extra set of syntactic rules and an attendant proof. The key point is, of course, that we still get all the properties of $\top$ that we need. In particular, the structural properties present in Figure 7 of [Pit00], plus corresponding ones having to do with **seq**, are all embodied in Observations 4.2 and 4.3 below. Moreover, since these follow generically from Definitions 3.4, 3.5, and 4.1 (and from determinism of $\rightarrowtail$), without considering the concrete sets of redex/reduct pairs and evaluation frames at hand, our approach promises to be more amenable to (further) extensions of the calculus. That the definition via $\rightarrowtail$ is no less structural than one obtained via extra syntactic rules is also evident from the proof of Lemma 4.13 below, which as Theorem 3.7 of [Pit00] was Pitts' main motivation for characterizing **nil**-termination structurally.

Observation 4.2. For every $\tau \in \mathit{Typ}$, $L \in \mathit{Term}(\tau\text{-}\mathit{list})$: $\mathit{Id} \top L \Leftrightarrow L \Downarrow \mathbf{nil}_\tau$. ■

Observation 4.3. Let $\tau \in \mathit{Typ}$ and $S \in \mathit{LStack}(\tau)$.

1. For every $\tau' \in \mathit{Typ}$, $M \in \mathit{Term}(\tau')$, and evaluation frame E with $E\{M\} \in \mathit{Term}(\tau)$: $S \top E\{M\} \Leftrightarrow S \circ E \top M$.

2. For every $R, R' \in \mathit{Term}(\tau)$ with $R \leadsto R'$: $S \top R \Leftrightarrow S \top R'$. ■

By repeated applications of Observation 4.3(1), we also have the following.

Corollary 4.4. For every $\tau, \tau' \in \mathit{Typ}$, $S \in \mathit{Stack}(\tau, \tau')$, $S' \in \mathit{LStack}(\tau')$, and $M \in \mathit{Term}(\tau)$:

$$(S' \mathbin{@} S) \top M \Leftrightarrow S' \top S\ M. \qquad ■$$

The following lemma shows that **nil**-termination is respected in a certain sense by evaluation of the term put in context.

Lemma 4.5. For every $\tau \in \mathit{Typ}$, $S \in \mathit{LStack}(\tau)$, and $M \in \mathit{Term}(\tau)$:

$$S \top M \Leftrightarrow \exists V \in \mathit{Value}(\tau).\ M \Downarrow V \wedge S \top V. \qquad \square$$

The proof, by two inductions over $\rightarrowtail^*$-sequences, is given in Appendix A. An immediate consequence is the following 'strictness of stacks' result.

Corollary 4.6. For every $\tau \in \mathit{Typ}$ and $M \in \mathit{Term}(\tau)$, if $M\Uparrow$, then for every $S \in \mathit{LStack}(\tau)$, $S \top M$ does not hold. ■

4.2 Termination for fix and $\top\top$-closedness

The key role of $\top$ in [Pit00] is its use in defining an order-reversing Galois connection, and the use of the induced closure operator in characterizing a class of relations that admit a form of fixpoint induction. This is repeated in the following three definitions.

Definition 4.7. Given $\tau, \tau' \in \mathit{Typ}$, we define

$$\mathit{Rel}(\tau, \tau') = \mathcal{P}(\mathit{Term}(\tau) \times \mathit{Term}(\tau'))$$

and

$$\mathit{StRel}(\tau, \tau') = \mathcal{P}(\mathit{LStack}(\tau) \times \mathit{LStack}(\tau')).$$

Further, we set $\mathit{Rel} = \bigcup_{\tau,\tau' \in \mathit{Typ}} \mathit{Rel}(\tau, \tau')$. ◇

Definition 4.8. Let $\tau, \tau' \in \mathit{Typ}$. Given $r \in \mathit{Rel}(\tau, \tau')$, we define $r^\top \in \mathit{StRel}(\tau, \tau')$ by

$$(S, S') \in r^\top \;\text{ iff }\; \forall (M, M') \in r.\; S \top M \;\Leftrightarrow\; S' \top M'.$$

Similarly, given $s \in \mathit{StRel}(\tau, \tau')$, we define $s^\top \in \mathit{Rel}(\tau, \tau')$ by

$$(M, M') \in s^\top \;\text{ iff }\; \forall (S, S') \in s.\; S \top M \;\Leftrightarrow\; S' \top M'.$$ ◇

The following properties are standard, for every $\tau, \tau' \in \mathit{Typ}$ and $r, r_1, r_2 \in \mathit{Rel}(\tau, \tau')$:

$$r \;\subseteq\; r^{\top\top} \tag{2}$$
$$(r^{\top\top})^\top \;=\; r^\top \tag{3}$$
$$r_1 \subseteq r_2 \;\Rightarrow\; r_1^{\top\top} \subseteq r_2^{\top\top}. \tag{4}$$

Definition 4.9. A relation $r \in \mathit{Rel}$ is ⊤⊤-*closed* if $r^{\top\top} = r$. ◇

Note that by (2) and (3), respectively, the condition $r^{\top\top} = r$ is equivalent to $r^{\top\top} \subseteq r$ and to the existence of some $r' \in \mathit{Rel}$ with $r = (r')^{\top\top}$.

Several important properties of ⊤⊤-closed relations can be established now. The first is that they respect adequate and compatible relations. The second is that they relate every pair of (appropriately typed) diverging terms.

Lemma 4.10. Let $\mathcal{E}$ be an adequate and compatible relation, let $\tau, \tau' \in \mathit{Typ}$, and $r \in \mathit{Rel}(\tau, \tau')$. If r is ⊤⊤-closed, then for every $M_1 \in \mathit{Term}(\tau)$ and $M_2, M_3 \in \mathit{Term}(\tau')$:

$$(M_1, M_2) \in r \wedge M_2 \;\mathcal{E}\; M_3 \;\Rightarrow\; (M_1, M_3) \in r.$$

Proof: The desired $(M_1, M_3) \in r$ follows from ⊤⊤-closedness of r and the following reasoning for every $(S, S') \in r^\top$:

$$\begin{array}{rcll}
S \top M_1 & \Leftrightarrow & S' \top M_2 & \text{by } (S,S') \in r^\top \text{ and } (M_1, M_2) \in r \\
 & \Leftrightarrow & \mathit{Id} \top S'\, M_2 & \text{by Corollary 4.4} \\
 & \Leftrightarrow & S'\, M_2 \Downarrow \mathbf{nil}_{\tau''} & \text{by Observation 4.2} \\
 & \Leftrightarrow & S'\, M_3 \Downarrow \mathbf{nil}_{\tau''} & \\
 & \Leftrightarrow & S' \top M_3 & \text{by Observation 4.2 and Corollary 4.4.}
\end{array}$$

Here $\tau'' \in \mathit{Typ}$ is such that $S' \in \mathit{Stack}(\tau', \tau''\text{-}\mathit{list})$, and the fourth equivalence follows from $M_2 \;\mathcal{E}\; M_3$, because $\mathcal{E}$ is compatible and adequate. ■

Lemma 4.11. For every $\tau, \tau' \in \mathit{Typ}$, $M \in \mathit{Term}(\tau)$, $M' \in \mathit{Term}(\tau')$, and $r \in \mathit{Rel}(\tau, \tau')$, if $M\Uparrow$, $M'\Uparrow$, and r is $\top\top$-closed, then $(M, M') \in r$.

Proof: By Corollary 4.6, $M\Uparrow$ and $M'\Uparrow$ imply that for every $(S, S') \in r^\top$: $S \top M \Leftrightarrow S' \top M'$. Thus, we have $(M, M') \in r^{\top\top} = r$. ■

The previous lemma provides a kind of base case for fixpoint induction. To establish this principle in full, we first need to look at the finite unwindings of a fixpoint.

Definition 4.12. Let $\tau \in \mathit{Typ}$ and $F \in \mathit{Term}(\tau \to \tau)$. By induction on $n \in \mathbb{N}$, we define $\mathbf{fix}^{(n)}(F) \in \mathit{Term}(\tau)$ as follows:

$$\mathbf{fix}^{(n)}(F) = \begin{cases} \mathbf{fix}(\lambda x :: \tau.x) & \text{if } n = 0 \\ F\ \mathbf{fix}^{(n-1)}(F) & \text{otherwise.} \end{cases} \qquad \diamond$$

Then the following important termination property holds for **fix** and its unwindings.

Lemma 4.13. For every $\tau \in \mathit{Typ}$, $S \in \mathit{LStack}(\tau)$, and $F \in \mathit{Term}(\tau \to \tau)$:

$$S \top \mathbf{fix}(F) \Leftrightarrow \exists n \in \mathbb{N}.\ S \top \mathbf{fix}^{(n)}(F). \qquad \square$$

The proof uses Observations 3.6 and 3.7 and Corollary 4.6, and is given in Appendix A.

Just as in [Pit00], the previous unwinding lemma can now be used to establish a fixpoint induction principle for binary relations. It justifies the claim that $\top\top$-closed relations have appropriate admissibility properties, and indeed corresponds to what is identified as the necessary parametricity property of **fix** in Sections 7 and 5 of [Wad89] and [LP96], respectively.

Lemma 4.14. Let $\tau, \tau' \in \mathit{Typ}$, $F \in \mathit{Term}(\tau \to \tau)$, $F' \in \mathit{Term}(\tau' \to \tau')$, and $r \in \mathit{Rel}(\tau, \tau')$. If

$$\forall (A, A') \in r.\ (F\ A, F'\ A') \in r \qquad (5)$$

and r is $\top\top$-closed, then $(\mathbf{fix}(F), \mathbf{fix}(F')) \in r$.

Proof: By the definitions of $\mathbf{fix}^{(0)}(\cdots)$, Observation 3.6, and Lemma 4.11, we have $(\mathbf{fix}^{(0)}(F), \mathbf{fix}^{(0)}(F')) \in r$. Using (5) and induction on natural numbers, it follows from this that for every $n \in \mathbb{N}$:

$$(\mathbf{fix}^{(n)}(F), \mathbf{fix}^{(n)}(F')) \in r. \qquad (6)$$

The desired $(\mathbf{fix}(F), \mathbf{fix}(F')) \in r$ then follows from $\top\top$-closedness of r and the following reasoning for every $(S, S') \in r^\top$:

$$\begin{array}{rcll} S \top \mathbf{fix}(F) & \Leftrightarrow & \exists n \in \mathbb{N}.\ S \top \mathbf{fix}^{(n)}(F) & \text{by Lemma 4.13} \\ & \Leftrightarrow & \exists n \in \mathbb{N}.\ S' \top \mathbf{fix}^{(n)}(F') & \text{by } (S, S') \in r^\top \text{ and (6)} \\ & \Leftrightarrow & S' \top \mathbf{fix}(F') & \text{by Lemma 4.13.} \end{array}$$

■

4.3 Termination for seq and convergence-reflection

To handle the strictness primitive, we ultimately need an analogue of Lemma 4.14 for **seq**. In Sections 5 of [JV04] and [JV06], the relevant parametricity property of **seq** was identified as saying that for appropriate relations r_1 and r_2, terms A and A' related by r_1, and terms B and B' related by r_2, we should have that $\mathbf{seq}(A, B)$ and $\mathbf{seq}(A', B')$ are related by r_2. It turns out that $\top\top$-closedness is not a strong enough restriction on r_1 and r_2 to achieve this, just as strictness and continuity were not enough in the denotational setting. To see why, we first need the following key statement, playing a similar role for **seq** as Lemma 4.13 does for **fix**, as well as a corollary of this statement.

Lemma 4.15. Let $\tau \in \mathit{Typ}$, $S \in \mathit{LStack}(\tau)$, $A \in \mathit{Term}$, and $B \in \mathit{Term}(\tau)$. Then: $S \top \mathbf{seq}(A, B) \Leftrightarrow A\Downarrow \wedge S \top B$.

Proof: Let $\tau' \in \mathit{Typ}$ be such that $A \in \mathit{Term}(\tau')$. Then we reason as follows:

$$
\begin{array}{lll}
 & S \top \mathbf{seq}(A, B) & \\
\Leftrightarrow & S \circ \mathbf{seq}(-, B) \top A & \text{by Observation 4.3(1)} \\
\Leftrightarrow & \exists V \in \mathit{Value}(\tau').\ A \Downarrow V \wedge S \circ \mathbf{seq}(-, B) \top V & \text{by Lemma 4.5} \\
\Leftrightarrow & \exists V \in \mathit{Value}(\tau').\ A \Downarrow V \wedge S \top B & \text{by Observation 4.3.}
\end{array}
$$
■

Corollary 4.16. For every $M \in \mathit{Term}$ and $\tau \in \mathit{Typ}$: $M\Downarrow \Leftrightarrow \mathbf{seq}(-, \mathbf{nil}_\tau) \top M$.

Proof: We reason as follows:

$$
\begin{array}{llll}
M\Downarrow & \Leftrightarrow & \mathit{Id} \top \mathbf{seq}(M, \mathbf{nil}_\tau) & \text{by Lemma 4.15 and } \mathit{Id} \top \mathbf{nil}_\tau \\
 & \Leftrightarrow & \mathbf{seq}(-, \mathbf{nil}_\tau) \top M & \text{by Observation 4.3(1).}
\end{array}
$$
■

The corollary essentially says that in the presence of **seq**, observing **nil**-termination of terms of list type suffices to observe general termination of arbitrary terms.

Now, we can give a counterexample to the parametricity property of **seq** mentioned at the start of this subsection. Let $r_1 = \{(\Omega_{\forall\alpha.\alpha}, \Omega)\}^{\top\top}$ and $r_2 = \{(M, M) \mid M \in \mathit{Term}(\forall\alpha.\alpha)\}^{\top\top}$, both of which are in $\mathit{Rel}(\forall\alpha.\alpha, \forall\alpha.\alpha)$, and let $A = \Omega_{\forall\alpha.\alpha}$ and $A' = B = B' = \Omega$. Clearly, r_1 and r_2 are $\top\top$-closed, and $(A, A') \in r_1$ and $(B, B') \in r_2$ hold by (2). So we would expect that $(\mathbf{seq}(A, B), \mathbf{seq}(A', B')) \in r_2$ also holds. But this does not hold, as is argued in the following. Note that for every $S \in \mathit{LStack}(\forall\alpha.\alpha)$ we have $(S, S) \in \{(M, M) \mid M \in \mathit{Term}(\forall\alpha.\alpha)\}^{\top}$. So to have $(\mathbf{seq}(\Omega_{\forall\alpha.\alpha}, \Omega), \mathbf{seq}(\Omega, \Omega)) \in \{(M, M) \mid M \in \mathit{Term}(\forall\alpha.\alpha)\}^{\top\top}$ would imply that for every such S we have $S \top \mathbf{seq}(\Omega_{\forall\alpha.\alpha}, \Omega) \Leftrightarrow S \top \mathbf{seq}(\Omega, \Omega)$. But this is contradicted by Observation 3.8, Lemma 4.15, and Corollary 4.16.

The reason for the failure described above is that the given r_1 relates a diverging term to a converging term. To repair this failure, we introduce a new restriction on relations, namely convergence-reflection. This restriction is a relatively direct translation of the bottom-reflection restriction discussed in the denotational setting of [JV04, JV06]. In particular, in contrast to what was envisaged in those earlier papers, the new restriction is not enforced by a variant of $\top\top$-closure or by a completely new closure operator. But of course, the new restriction must at least nicely coexist with $\top\top$-closure in a certain sense, to be discussed below the following definition.

Definition 4.17. We say that $r \in \mathit{Rel}$ is *convergence-reflecting* if for every $(M, M') \in r$: $M\Downarrow \Leftrightarrow M'\Downarrow$. For given $\tau, \tau' \in \mathit{Typ}$, the restriction of $\mathit{Rel}(\tau, \tau')$ to convergence-reflecting relations is denoted by $\mathit{Rel}^{\Downarrow}(\tau, \tau')$. We set $\mathit{Rel}^{\Downarrow} = \bigcup_{\tau,\tau' \in \mathit{Typ}} \mathit{Rel}^{\Downarrow}(\tau, \tau')$. ◇

Note that the r_1 used in the counterexample above is also evidence for the fact that not every $\top\top$-closed relation is convergence-reflecting. However, we can show that the notions of $\top\top$-closure and convergence-reflection are compatible in the sense that the former preserves the latter. This is essential, as otherwise the modular way in which the two restrictions are introduced (and later preserved) would break down.

Lemma 4.18. For every $r \in \mathit{Rel}^{\Downarrow}$, $r^{\top\top} \in \mathit{Rel}^{\Downarrow}$ also holds.

Proof: For arbitrary $\tau \in \mathit{Typ}$ and every $(M, M') \in r$, we have:

$$\begin{array}{rcll} \mathbf{seq}(-, \mathbf{nil}_\tau) \top M & \Leftrightarrow & M\Downarrow & \text{by Corollary 4.16} \\ & \Leftrightarrow & M'\Downarrow & \text{since } r \text{ is convergence-reflecting} \\ & \Leftrightarrow & \mathbf{seq}(-, \mathbf{nil}_\tau) \top M' & \text{by Corollary 4.16.} \end{array}$$

Thus, $(\mathbf{seq}(-, \mathbf{nil}_\tau), \mathbf{seq}(-, \mathbf{nil}_\tau)) \in r^\top$, and consequently for every $(N, N') \in r^{\top\top}$:

$$\mathbf{seq}(-, \mathbf{nil}_\tau) \top N \Leftrightarrow \mathbf{seq}(-, \mathbf{nil}_\tau) \top N' ,$$

which by Corollary 4.16, applied twice as above, is equivalent to $N\Downarrow \Leftrightarrow N'\Downarrow$. ■

A further connection between the concepts of convergence-reflection and $\top\top$-closure is established in the following lemma, providing the sought after parametricity property of **seq**, and thus the analogue for **seq** of Lemma 4.14.

Lemma 4.19. Let $r_1 \in \mathit{Rel}^{\Downarrow}$, $r_2 \in \mathit{Rel}$, $(A, A') \in r_1$, and $(B, B') \in r_2$. If r_2 is $\top\top$-closed, then $(\mathbf{seq}(A, B), \mathbf{seq}(A', B')) \in r_2$.

Proof: The desired $(\mathbf{seq}(A, B), \mathbf{seq}(A', B')) \in r_2$ follows from $\top\top$-closedness of r_2 and the following reasoning for every $(S, S') \in r_2^\top$:

$$\begin{array}{rcll} S \top \mathbf{seq}(A, B) & \Leftrightarrow & A\Downarrow \wedge S \top B & \text{by Lemma 4.15} \\ & \Leftrightarrow & A'\Downarrow \wedge S \top B & \text{by } (A, A') \in r_1 \in \mathit{Rel}^{\Downarrow} \\ & \Leftrightarrow & A'\Downarrow \wedge S' \top B' & \text{by } (S, S') \in r_2^\top \text{ and } (B, B') \in r_2 \\ & \Leftrightarrow & S' \top \mathbf{seq}(A', B') & \text{by Lemma 4.15.} \end{array}$$
■

Since in the previous lemma one relation must be convergence-reflecting and the other one must be $\top\top$-closed, we actually have to impose (and preserve) both restrictions on all relations in our type-based characterization of observational equivalence.

This is so because **seq** can be applied at all types according to the corresponding typing rule in Figure 3, where no restrictions are imposed on τ or τ'. Hence, the relational interpretation of every type must be able to fulfill either of the two roles r_1 and r_2 in Lemma 4.19, and so must adhere to both restrictions. This is what will drive, in Section 5, the development of the relational interpretation of PolySeq types.

But first we need to set up a few more auxiliary statements. The first is an analogue of Lemma 4.5 for general termination.

Lemma 4.20. For every $\tau, \tau' \in \mathit{Typ}$, $S \in \mathit{Stack}(\tau, \tau')$, and $M \in \mathit{Term}(\tau)$:

$$(S\ M){\Downarrow} \Leftrightarrow \exists V \in \mathit{Value}(\tau).\ M \Downarrow V \wedge (S\ V){\Downarrow}.$$

Proof: We reason as follows:

$$\begin{array}{lll} & (S\ M){\Downarrow} & \\ \Leftrightarrow & \mathbf{seq}(-, \mathbf{nil}_\tau) \top S\ M & \text{by Corollary 4.16} \\ \Leftrightarrow & (\mathbf{seq}(-, \mathbf{nil}_\tau) \mathbin{@} S) \top M & \text{by Corollary 4.4} \\ \Leftrightarrow & \exists V \in \mathit{Value}(\tau).\ M \Downarrow V \wedge (\mathbf{seq}(-, \mathbf{nil}_\tau) \mathbin{@} S) \top V & \text{by Lemma 4.5} \\ \Leftrightarrow & \exists V \in \mathit{Value}(\tau).\ M \Downarrow V \wedge (S\ V){\Downarrow}\,, & \end{array}$$

where the last equivalence is again by Corollaries 4.4 and 4.16. ∎

Lemma 4.20 implies that for every $M \in \mathit{Term}$ and evaluation frame E with $E\{M\} \in \mathit{Term}$, if $E\{M\}{\Downarrow}$, then $M{\Downarrow}$. Letting $M = F$ and $E = (-\ A)$, we have the following.

Corollary 4.21. For every $\tau, \tau' \in \mathit{Typ}$, $F \in \mathit{Term}(\tau \to \tau')$, and $A \in \mathit{Term}(\tau)$, if $(F\ A){\Downarrow}$, then $F{\Downarrow}$. ∎

Moreover, if $M = \Omega_{\forall\alpha.\alpha}$ and $E = -_\tau$, then Observation 3.8 gives the following.

Corollary 4.22. For every $\tau \in \mathit{Typ}$: $(\Omega_{\forall\alpha.\alpha})_\tau{\Uparrow}$. ∎

5 The logical relation

The key to parametricity results, and to our characterization of observational equivalence by a logical relation, is to build relational interpretations of types by induction on the type structure. Starting from an interpretation of type variables by relations (between terms), this requires defining a *relational action* for each of the ways of forming PolySeq types. Such an action takes an appropriate number of relations and produces a new one as the interpretation for the compound type. During this propagation of relations up the type hierarchy it is essential that the restrictions needed to accommodate **fix** and **seq** — namely, ⊤⊤-closedness and convergence-reflection — are preserved. In our development convergence-reflection will always hold by construction, whereas preservation of ⊤⊤-closedness is established *a posteriori*.

The main characteristic of all logical relations, from the very beginning [Fri75, Plo73, Rey83, Sta85] up to newer accounts [Pit00, Pit05, Ahm06, DHJG06], is that for two functions to be related they must map related arguments to related results. In the presence of **seq** the relational action for function types must additionally enforce the requirement that two function terms are only related if either both converge or both diverge. This corresponds to the symmetrized version of the additional restriction on the relational interpretation of function types in [JV04, JV06], transferred to the operational setting.

Definition 5.1. Given $\tau_1, \tau_1', \tau_2, \tau_2' \in \mathit{Typ}$, $r_1 \in \mathit{Rel}(\tau_1, \tau_1')$, and $r_2 \in \mathit{Rel}(\tau_2, \tau_2')$, we define $(r_1 \to r_2) \in \mathit{Rel}^{\Downarrow}(\tau_1 \to \tau_2, \tau_1' \to \tau_2')$ by

$$(F, F') \in (r_1 \to r_2) \;\text{ iff }\; (F{\Downarrow} \Leftrightarrow F'{\Downarrow}) \wedge \forall (A, A') \in r_1.\ (F\ A, F'\ A') \in r_2$$

for every $F \in \mathit{Term}(\tau_1 \to \tau_2)$ and $F' \in \mathit{Term}(\tau_1' \to \tau_2')$. ◇

The relational action corresponding to $\forall$-types usually relates two polymorphic terms if respective instances, at arbitrary types, are related by the images, under a given relation-to-relation mapping, of certain relations between terms of the types at which instantiation occurs. The relation-to-relation mapping will be derived from the compound action of the body of the $\forall$-type on relations interpreting the type variable from its head. The range of relations over which the quantification occurs, i.e., the concretization of 'certain' above, depends on the primitives the calculus supports, at arbitrary types, in extension of the Girard-Reynolds calculus $\lambda^{\forall}$. As we have argued, **fix** and **seq** mandate relations to be ⊤⊤-closed and convergence-reflecting. Since the later construction of the logical relation will ensure that the relation-to-relation mapping always first ⊤⊤-closes its argument relation (cf. clause (10) in Definition 5.4), only convergence-reflection needs to be explicitly enforced in the quantification of relations below. The aforementioned ⊤⊤-closure of the argument relation will not affect its convergence-reflection property due to Lemma 4.18. To ensure that also the result of the relational action is convergence-reflecting, we explicitly enforce this property in a manner analogous to the way it was enforced in Definition 5.1. That this explicit enforcement is strictly necessary here is argued below the following definition.

Definition 5.2. Let τ_1 and τ_1' be types with at most a single free variable, α say. Suppose R is a function that maps every $\tau_2, \tau_2' \in \mathit{Typ}$ and $r \in \mathit{Rel}^{\Downarrow}(\tau_2, \tau_2')$ to an $R_{\tau_2,\tau_2'}(r) \in \mathit{Rel}(\tau_1[\tau_2/\alpha], \tau_1'[\tau_2'/\alpha])$. Then we define $(\forall R) \in \mathit{Rel}^{\Downarrow}(\forall\alpha.\tau_1, \forall\alpha.\tau_1')$ by

$$\begin{aligned}(G, G') \in (\forall R) \;\text{ iff }\; & (G{\Downarrow} \Leftrightarrow G'{\Downarrow}) \\ & \wedge \forall \tau_2, \tau_2' \in \mathit{Typ}, r \in \mathit{Rel}^{\Downarrow}(\tau_2, \tau_2').\ (G_{\tau_2}, G'_{\tau_2'}) \in R_{\tau_2,\tau_2'}(r)\end{aligned}$$

for every $G \in \mathit{Term}(\forall\alpha.\tau_1)$ and $G' \in \mathit{Term}(\forall\alpha.\tau_1')$. We also write $\forall R$ as $\forall r.R(r)$, suppressing reference to τ_2 and τ_2'. ◇

Assume the condition $G{\Downarrow} \Leftrightarrow G'{\Downarrow}$ were dropped from the previous definition. Then the relation $\forall R$ would no longer be guaranteed to be convergence-reflecting. To see this, consider $\tau_1 = \tau_1' = \alpha$ and let R be the function that maps every $\tau_2, \tau_2' \in$

Typ and $r \in \mathit{Rel}^{\Downarrow}(\tau_2, \tau_2')$ to $r^{\top\top}$. This function is well-behaved insofar as it maps every convergence-reflecting relation to one that is both $\top\top$-closed and convergence-reflecting (cf. Lemma 4.18). Nevertheless, $\forall R$ would not be convergence-reflecting, because it would relate $G = \Omega$ and $G' = \Omega_{\forall\alpha.\alpha}$. This is because for every $\tau_2, \tau_2' \in \mathit{Typ}$ and $r \in \mathit{Rel}^{\Downarrow}(\tau_2, \tau_2')$, we have $(\Omega_{\tau_2}, (\Omega_{\forall\alpha.\alpha})_{\tau_2'}) \in r^{\top\top}$ by Observation 3.8, Lemma 4.11, and Corollary 4.22. But $\Omega\Downarrow$ and $\Omega_{\forall\alpha.\alpha}\Uparrow$ by Observation 3.8.

The relational action for list types is a straightforward structural lifting, appropriately combined with $\top\top$-closure. No special care is needed with respect to convergence-reflection, because it is satisfied automatically.

Definition 5.3. Given $\tau, \tau' \in \mathit{Typ}$ and $r \in \mathit{Rel}(\tau, \tau')$, we define $\mathit{lift}(r) \in \mathit{Rel}(\tau\text{-}\mathit{list}, \tau'\text{-}\mathit{list})$ as the greatest (post-)fixpoint (with respect to set inclusion) of the mapping $s \mapsto (1 + (r \times s))^{\top\top}$ for $s \in \mathit{Rel}(\tau\text{-}\mathit{list}, \tau'\text{-}\mathit{list})$, where

$$1 + (r \times s) = \{(\mathbf{nil}_\tau, \mathbf{nil}_{\tau'})\} \cup \{(H : T, H' : T') \mid (H, H') \in r \wedge (T, T') \in s\}$$

for every such s. The existence of the greatest fixpoint is guaranteed by monotonicity of the mapping $s \mapsto (1+(r\times s))^{\top\top}$ with respect to set inclusion, which in turn follows from (4). Note that the fixpoint property

$$\mathit{lift}(r) = (1 + (r \times \mathit{lift}(r)))^{\top\top}, \tag{7}$$

the observation that $1+(r\times \mathit{lift}(r))$ is convergence-reflecting (since it only relates values), and Lemma 4.18 imply that $\mathit{lift}(r)$ is convergence-reflecting, and thus actually $\mathit{lift}(r) \in \mathit{Rel}^{\Downarrow}(\tau\text{-}\mathit{list}, \tau'\text{-}\mathit{list})$. ◇

Combining the relational actions, the logical relation Δ is defined by induction on the structure of PolySeq types. It maps a type and a list containing convergence-reflecting relations as interpretations for the type's free variables to a new relation. The new relation is convergence-reflecting by construction, i.e., because the relational actions always deliver convergence-reflecting relations.

Definition 5.4. For every type τ, $n \in \mathbb{N}$, list $\vec{\alpha} = \alpha_1, \ldots, \alpha_n$ of distinct type variables containing the free variables of τ, lists $\vec{\tau} = \tau_1, \ldots, \tau_n$ and $\vec{\tau}' = \tau_1', \ldots, \tau_n'$ of closed types, and list $\vec{r} = r_1, \ldots, r_n$ with $r_i \in \mathit{Rel}^{\Downarrow}(\tau_i, \tau_i')$ for every $1 \le i \le n$, we define $\Delta_\tau(\vec{r}/\vec{\alpha}) \in \mathit{Rel}^{\Downarrow}(\tau[\vec{\tau}/\vec{\alpha}], \tau[\vec{\tau}'/\vec{\alpha}])$ by induction on the structure of τ as follows:

$$\Delta_{\alpha_i}(\vec{r}/\vec{\alpha}) = r_i \tag{8}$$

$$\Delta_{\tau'\to\tau''}(\vec{r}/\vec{\alpha}) = \Delta_{\tau'}(\vec{r}/\vec{\alpha}) \to \Delta_{\tau''}(\vec{r}/\vec{\alpha}) \tag{9}$$

$$\Delta_{\forall\alpha.\tau'}(\vec{r}/\vec{\alpha}) = \forall r.\Delta_{\tau'}(\vec{r}, r^{\top\top}/\vec{\alpha}, \alpha) \tag{10}$$

$$\Delta_{\tau'\text{-}\mathit{list}}(\vec{r}/\vec{\alpha}) = \mathit{lift}(\Delta_{\tau'}(\vec{r}/\vec{\alpha})). \tag{11}$$

Note that without loss of generality the variable bound in the head of $\forall\alpha.\tau'$ in clause (10) can be assumed to not occur in $\vec{\alpha}$. Note also that the mapping $r \mapsto \Delta_{\tau'}(\vec{r}, r^{\top\top}/\vec{\alpha}, \alpha)$ in the right-hand side of that clause is well-defined, given that it will only be invoked for convergence-reflecting relations r by Definition 5.2. Because then by Lemma 4.18 also $r^{\top\top}$ is convergence-reflecting, so that it can be used as an argument for $\Delta_{\tau'}$. ◇

To ultimately establish that $\Delta_\tau(\vec{r}/\vec{\alpha})$ is $\top\top$-closed (in addition to being convergence-reflecting) provided every relation in $\vec{r}$ is, we have to show how $\top\top$-closedness is pushed up the type hierarchy by the relational actions for function and $\forall$-types. This is the task of the following two lemmas.

Lemma 5.5. For every $r_1, r_2 \in \mathit{Rel}$, if r_2 is $\top\top$-closed, then so is $r_1 \to r_2$.

Proof: We have to show that $(F, F') \in (r_1 \to r_2)^{\top\top}$ implies $(F, F') \in (r_1 \to r_2)$, i.e.,

$$F\Downarrow \;\Leftrightarrow\; F'\Downarrow$$

and

$$\forall (A, A') \in r_1.\ (F\ A, F'\ A') \in r_2.$$

The former holds because $(r_1 \to r_2)^{\top\top}$ is convergence-reflecting by Lemma 4.18. The latter follows from $\top\top$-closedness of r_2 if we can show that $(A, A') \in r_1$ and $(F, F') \in (r_1 \to r_2)^{\top\top}$ together imply $(F\ A, F'\ A') \in r_2^{\top\top}$. To do so, we reason for every $(S, S') \in r_2^{\top}$ as follows:

$$\begin{array}{rcll} S \top F\ A & \Leftrightarrow & S \circ (-\ A) \top F & \text{by Observation 4.3(1)} \\ & \Leftrightarrow & S' \circ (-\ A') \top F' & \\ & \Leftrightarrow & S' \top F'\ A' & \text{by Observation 4.3(1).} \end{array}$$

Here the second equivalence holds by $(F, F') \in (r_1 \to r_2)^{\top\top}$ and $(S \circ (-\ A), S' \circ (-\ A')) \in (r_1 \to r_2)^{\top}$. The latter is established by reasoning for every $(N, N') \in (r_1 \to r_2)$ as follows:

$$\begin{array}{rcll} S \circ (-\ A) \top N & \Leftrightarrow & S \top N\ A & \text{by Observation 4.3(1)} \\ & \Leftrightarrow & S' \top N'\ A' & \\ & \Leftrightarrow & S' \circ (-\ A') \top N' & \text{by Observation 4.3(1).} \end{array}$$

Here the second equivalence holds by $(S, S') \in r_2^{\top}$ and $(N\ A, N'\ A') \in r_2$. The latter follows from $(N, N') \in (r_1 \to r_2)$ and $(A, A') \in r_1$ by the definition of $r_1 \to r_2$. ■

The main difference between the previous proof and that of Lemma 4.7(iii) in [Pit00] is the extra proof obligation arising from convergence-reflection, which is resolved using Lemma 4.18.

Lemma 5.6. Let R be as in Definition 5.2. If $R_{\tau_2,\tau_2'}(r)$ is $\top\top$-closed for every $\tau_2, \tau_2' \in \mathit{Typ}$ and $r \in \mathit{Rel}^{\Downarrow}(\tau_2, \tau_2')$, then $\forall R$ is also $\top\top$-closed. □

The proof, using Observation 4.3(1) and Lemma 4.18, is very similar to that of Lemma 5.5. For completeness' sake, it is given in Appendix A.

The following lemma gives the desired statement about propagation of $\top\top$-closedness. It is the analogue of Lemma 4.11 in [Pit00], with the proof making additional use of Lemma 4.18.

Lemma 5.7. Let τ, $\vec{\alpha}$, and $\vec{r}$ be as in Definition 5.4. If every relation in $\vec{r}$ is $\top\top$-closed, then so is $\Delta_\tau(\vec{r}/\vec{\alpha})$.

Proof: The statement follows by induction on the structure of τ, using Definition 5.4. The induction base with τ being a type variable is trivial, because every relation in $\vec{r}$ is assumed to be $\top\top$-closed. The induction step for function types follows from Lemma 5.5. The induction step for $\forall$-types follows from Lemmas 4.18 and 5.6. The induction step for list types is trivial by (7). ■

By inductions similar to those in the previous proof we easily obtain the following two results.

Observation 5.8. Let τ, $\vec{\alpha}$, and $\vec{r}$ be as in Definition 5.4. Moreover, let $r' \in Rel^{\Downarrow}$ and let α' be a type variable not occurring in $\vec{\alpha}$ (and hence not occurring free in τ). Then:

$$\Delta_\tau(\vec{r}/\vec{\alpha}) = \Delta_\tau(\vec{r}, r'/\vec{\alpha}, \alpha').$$ ■

Observation 5.9. Let τ, $\vec{\alpha}$, and $\vec{r}$ be as in Definition 5.4. Moreover, let α' be a type variable not occurring in $\vec{\alpha}$ and let τ' be a type with free variables in $\vec{\alpha}, \alpha'$. Then:

$$\Delta_{\tau'[\tau/\alpha']}(\vec{r}/\vec{\alpha}) = \Delta_{\tau'}(\vec{r}, \Delta_\tau(\vec{r}/\vec{\alpha})/\vec{\alpha}, \alpha').$$ ■

6 Proving the main result

In this section we first prove the parametricity theorem and then prove its strengthening corresponding to identity extension. Proofs of parametricity theorems typically proceed by induction on typing derivations, which in our case would mean by induction on the axioms and rules in Figure 3. Instead, our proof will be based on the closely related system describing compatibility in Figure 5. In any case, considerations similar to those in [JV04, JV06] are necessary. In particular, since the relational action for function types was changed by adding a convergence-reflection restriction, a stronger statement than in the case of the 'standard' logical relation must be proved for the rule in whose conclusion a function type appears, i.e., for the compatibility rule for function abstractions. The extra proof obligation thus arising in the following analogue of Lemma 4.7(i) in [Pit00] is met by using the fact that every function abstraction is a value and thus trivially converges.

Lemma 6.1. Let $\tau_1, \tau_1', \tau_2, \tau_2' \in Typ$, $r_1 \in Rel(\tau_1, \tau_1')$, and $r_2 \in Rel(\tau_2, \tau_2')$. Let x be a term variable and M and M' be terms such that $x :: \tau_1 \vdash M :: \tau_2$ and $x :: \tau_1' \vdash M' :: \tau_2'$. If

$$\forall (A, A') \in r_1.\ (M[A/x], M'[A'/x]) \in r_2 \tag{12}$$

and r_2 is $\top\top$-closed, then $(\lambda x :: \tau_1.M, \lambda x :: \tau_1'.M') \in (r_1 \to r_2)$.

Proof: We have to show

$$(\lambda x :: \tau_1.M)\Downarrow \Leftrightarrow (\lambda x :: \tau_1'.M')\Downarrow$$

and

$$\forall (A, A') \in r_1.\ ((\lambda x :: \tau_1.M)\ A, (\lambda x :: \tau_1'.M')\ A') \in r_2.$$

The former holds because both $\lambda x :: \tau_1.M$ and $\lambda x :: \tau_1'.M'$ are values. The latter follows from $\top\top$-closedness of r_2 and the following reasoning for every $(A, A') \in r_1$ and $(S, S') \in r_2^\top$:

$$\begin{array}{rcll} S \top (\lambda x :: \tau_1.M)\ A & \Leftrightarrow & S \top M[A/x] & \text{by Observation 4.3(2)} \\ & \Leftrightarrow & S' \top M'[A'/x] & \text{by } (S, S') \in r_2^\top \text{ and (12)} \\ & \Leftrightarrow & S' \top (\lambda x :: \tau_1'.M')\ A' & \text{by Observation 4.3(2).} \end{array}$$ ■

Similarly, we need an analogue of Lemma 4.8(i) in [Pit00], for type generalizations. In contrast to the situation in [JV04, JV06], we have now imposed an explicit convergence-reflection restriction on the relational action for ∀-types as well. Consequently, an extra proof obligation now also arises in the following 'compatibility lemma' for type generalizations, but can be met just as for function abstractions.

Lemma 6.2. Let τ_1, τ_1', α, and R be as in Definition 5.2, and let M and M' be terms such that $\alpha \vdash M :: \tau_1$ and $\alpha \vdash M' :: \tau_1'$. If

$$\forall \tau_2, \tau_2' \in \mathit{Typ}, r \in \mathit{Rel}^{\Downarrow}(\tau_2, \tau_2').\ (M[\tau_2/\alpha], M'[\tau_2'/\alpha]) \in R_{\tau_2,\tau_2'}(r)$$

and $R_{\tau_2,\tau_2'}(r)$ is $\top\top$-closed for every $\tau_2, \tau_2' \in \mathit{Typ}$ and $r \in \mathit{Rel}^{\Downarrow}(\tau_2, \tau_2')$, then $(\Lambda\alpha.M, \Lambda\alpha.M') \in (\forall R)$. ■

The proof is completely analogous to that of Lemma 6.1 above. We omit it here.

As is typical in proofs about logical relations, we have to generalize the statement we ultimately want to prove from terms and types without free variables to those with free variables. Hence, we also need to extend the logical relation itself to open terms, which is as usual done via closing substitutions.

Definition 6.3. Let $n, m \in \mathbb{N}$, let $\vec{\alpha}$ be a list of n type variables, $\vec{x} = x_1, \ldots, x_m$ be a list of term variables, $\tau_1, \ldots, \tau_m$ be types, and $\Gamma = \vec{\alpha}, x_1 :: \tau_1, \ldots, x_m :: \tau_m$. Given terms M and M' and a type τ with $\Gamma \vdash M :: \tau$ and $\Gamma \vdash M' :: \tau$, we write

$$\Gamma \vdash M\ \Delta\ M' :: \tau$$

if for every pair of lists $\vec{\sigma} = \sigma_1, \ldots, \sigma_n$ and $\vec{\sigma}' = \sigma_1', \ldots, \sigma_n'$ of closed types and every list $\vec{r} = r_1, \ldots, r_n$ of $\top\top$-closed $r_i \in \mathit{Rel}^{\Downarrow}(\sigma_i, \sigma_i')$, we have that for every pair of lists $\vec{N} = N_1, \ldots, N_m$ and $\vec{N}' = N_1', \ldots, N_m'$ with $(N_j, N_j') \in \Delta_{\tau_j}(\vec{r}/\vec{\alpha})$ for every $1 \leq j \leq m$, the following holds:

$$(M[\vec{\sigma}/\vec{\alpha}, \vec{N}/\vec{x}], M'[\vec{\sigma}'/\vec{\alpha}, \vec{N}'/\vec{x}]) \in \Delta_\tau(\vec{r}/\vec{\alpha}).$$ ◇

Now, we can go about proving the fundamental property of Δ, namely that it is reflexive. That is, we prove the following parametricity theorem.

Theorem 6.4. The relation Δ is compatible, and thus reflexive. In particular, for every $\tau \in \mathit{Typ}$ and $M \in \mathit{Term}(\tau)$: $(M, M) \in \Delta_\tau()$.

Proof: To prove that Δ is compatible, we have to show that it is closed under each of the axioms and rules in Figure 5. The axiom $\Gamma, x :: \tau \vdash x \ \Delta \ x :: \tau$ is trivially satisfied due to the way Δ is defined. Also by that definition, to establish the rule

$$\frac{\Gamma \vdash A \ \Delta \ A' :: \tau \qquad \Gamma \vdash B \ \Delta \ B' :: \tau'}{\Gamma \vdash \mathbf{seq}(A, B) \ \Delta \ \mathbf{seq}(A', B') :: \tau'}$$

it suffices to show that for Γ as in Definition 6.3, types τ and τ', terms A, A', B, and B' with $\Gamma \vdash A :: \tau$, $\Gamma \vdash A' :: \tau$, $\Gamma \vdash B :: \tau'$, and $\Gamma \vdash B' :: \tau'$, lists $\vec{\sigma} = \sigma_1, \ldots, \sigma_n$ and $\vec{\sigma}' = \sigma'_1, \ldots, \sigma'_n$ of closed types, list $\vec{r} = r_1, \ldots, r_n$ of $\top\top$-closed $r_i \in \mathit{Rel}^{\Downarrow}(\sigma_i, \sigma'_i)$, list $\vec{N} = N_1, \ldots, N_m$ of $N_j \in \mathit{Term}(\tau_j[\vec{\sigma}/\vec{\alpha}])$, and list $\vec{N}' = N'_1, \ldots, N'_m$ of $N'_j \in \mathit{Term}(\tau_j[\vec{\sigma}'/\vec{\alpha}])$,

$$(A[\vec{\sigma}/\vec{\alpha}, \vec{N}/\vec{x}], A'[\vec{\sigma}'/\vec{\alpha}, \vec{N}'/\vec{x}]) \in \Delta_\tau(\vec{r}/\vec{\alpha}) \tag{13}$$

and

$$(B[\vec{\sigma}/\vec{\alpha}, \vec{N}/\vec{x}], B'[\vec{\sigma}'/\vec{\alpha}, \vec{N}'/\vec{x}]) \in \Delta_{\tau'}(\vec{r}/\vec{\alpha}) \tag{14}$$

imply

$$(\mathbf{seq}(A, B)[\vec{\sigma}/\vec{\alpha}, \vec{N}/\vec{x}], \mathbf{seq}(A', B')[\vec{\sigma}'/\vec{\alpha}, \vec{N}'/\vec{x}]) \in \Delta_{\tau'}(\vec{r}/\vec{\alpha}).$$

By substitution, the latter is equivalent to

$$(\mathbf{seq}(A[\vec{\sigma}/\vec{\alpha}, \vec{N}/\vec{x}], B[\vec{\sigma}/\vec{\alpha}, \vec{N}/\vec{x}]), \mathbf{seq}(A'[\vec{\sigma}'/\vec{\alpha}, \vec{N}'/\vec{x}], B'[\vec{\sigma}'/\vec{\alpha}, \vec{N}'/\vec{x}])) \in \Delta_{\tau'}(\vec{r}/\vec{\alpha}).$$

But this is indeed implied by (13) and (14) due to Lemma 4.19, taking into account that $\Delta_\tau(\vec{r}/\vec{\alpha})$ is convergence-reflecting (by Definition 5.4) and $\Delta_{\tau'}(\vec{r}/\vec{\alpha})$ is $\top\top$-closed (by Lemma 5.7). The remaining axiom and rules from Figure 5 are established in a similar fashion in Appendix A, using Lemmas 4.14, 4.18, 5.7, 6.1, and 6.2, Observation 5.9, (2), (7), and an auxiliary lemma about case expressions, which is also proved in Appendix A based on (3), (7), and Observation 4.3. □

We can use the technique of the preceding proof, together with Lemma 5.7 and Observation 5.9, to show that Δ is also closed under the rules in Figure 6. This gives the following.

Lemma 6.5. The relation Δ is substitutive. ■

One further important property that Δ should have if it is to characterize observational equivalence — indeed the very property tying in the observational aspect — is adequacy with respect to **nil**-termination. It is established next.

Lemma 6.6. The relation Δ is adequate.

Proof: Let $\tau \in \mathit{Typ}$ and $L, L' \in \mathit{Term}(\tau\text{-}\mathit{list})$. If $L \;\Delta\; L'$, then $(L, L') \in (1 + (r \times \mathit{lift}(r)))^{\top\top}$ by Definition 6.3, clause (11), and (7), where $r = \Delta_\tau()$. Now, the desired $L \Downarrow \mathbf{nil}_\tau \;\Leftrightarrow\; L' \Downarrow \mathbf{nil}_\tau$ follows by applying Observation 4.2 (twice) and combining $(L, L') \in (1 + (r \times \mathit{lift}(r)))^{\top\top}$ with $(\mathit{Id}, \mathit{Id}) \in (1 + (r \times \mathit{lift}(r)))^{\top}$, where to establish the latter it suffices to show $\mathit{Id} \top N \;\Leftrightarrow\; \mathit{Id} \top N'$ for every $(N, N') \in (1 + (r \times \mathit{lift}(r)))$. We do so by case distinction.

Case a: $N = N' = \mathbf{nil}_\tau$. Then both $\mathit{Id} \top N$ and $\mathit{Id} \top N'$ hold.

Case b: $N = H : T$ and $N' = H' : T'$ for some $(H, H') \in r$ and $(T, T') \in \mathit{lift}(r)$. Then neither $\mathit{Id} \top N$ nor $\mathit{Id} \top N'$ holds. ■

Finally, we can show that Δ is not just any adequate, compatible, and substitutive relation, but is actually exactly the one we are looking for.

Theorem 6.7. The relation Δ is the largest adequate, compatible, and substitutive relation. It is also reflexive, transitive, and symmetric.

Proof: For the first statement, by Theorem 6.4 and Lemmas 6.5 and 6.6, it remains to prove that Δ subsumes every adequate, compatible, and substitutive relation. Let $\mathcal{E}$ be such a relation, let Γ, M, M', and τ be as in Definition 6.3, and assume $\Gamma \vdash M \;\mathcal{E}\; M' :: \tau$. Further, let $\vec{\sigma} = \sigma_1, \ldots, \sigma_n$ and $\vec{\sigma}' = \sigma'_1, \ldots, \sigma'_n$ be lists of closed types, $\vec{r} = r_1, \ldots, r_n$ be a list of $\top\top$-closed $r_i \in \mathit{Rel}^{\Downarrow}(\sigma_i, \sigma'_i)$, and $\vec{N} = N_1, \ldots, N_m$ and $\vec{N}' = N'_1, \ldots, N'_m$ be lists with $(N_j, N'_j) \in \Delta_{\tau_j}(\vec{r}/\vec{\alpha})$ for every $1 \leq j \leq m$. Since Δ is reflexive (cf. Theorem 6.4), we have $\Gamma \vdash M \;\Delta\; M :: \tau$, which by Definition 6.3 implies that

$$(M[\vec{\sigma}/\vec{\alpha}, \vec{N}/\vec{x}], M[\vec{\sigma}'/\vec{\alpha}, \vec{N}'/\vec{x}]) \in \Delta_\tau(\vec{r}/\vec{\alpha}). \tag{15}$$

Moreover, $\Gamma \vdash M \;\mathcal{E}\; M' :: \tau$ and the substitutivity of $\mathcal{E}$ imply that

$$M[\vec{\sigma}'/\vec{\alpha}, \vec{N}'/\vec{x}] \;\mathcal{E}\; M'[\vec{\sigma}'/\vec{\alpha}, \vec{N}'/\vec{x}]. \tag{16}$$

Since (15) and (16) combine into $(M[\vec{\sigma}/\vec{\alpha}, \vec{N}/\vec{x}], M'[\vec{\sigma}'/\vec{\alpha}, \vec{N}'/\vec{x}]) \in \Delta_\tau(\vec{r}/\vec{\alpha})$ by Lemmas 4.10 and 5.7, and since this is obtained independently of the choice of the lists $\vec{\sigma}$, $\vec{\sigma}'$, $\vec{r}$, $\vec{N}$, and $\vec{N}'$ above, we indeed have the desired $\Gamma \vdash M \;\Delta\; M' :: \tau$ by Definition 6.3.

For the second statement, note that since Δ is compatible, it is also reflexive. Moreover, it is easy to see that the collection of adequate, compatible, and substitutive relations is closed under the operations of relation composition and relation reciprocation. This implies that $\Delta;\Delta$ and Δ^{-1} are adequate, compatible, and substitutive relations, and are thus subsumed by the largest such relation, i.e., $\Delta;\Delta \subseteq \Delta$ and $\Delta^{-1} \subseteq \Delta$. But this means that Δ is also transitive and symmetric. ■

With the above characterization, we have established that Δ coincides with our intended notion of equivalence as discussed at the end of Section 3.2. So the following corollary can be read either as a definition of $=_{obs}$ in terms of Δ or as a coincidence statement between $=_{obs}$ and Δ. In the latter reading, we take $=_{obs}$ to be characterized as the union of all adequate, compatible, and substitutive relations and Δ to be characterized independently based on the type structure of PolySeq.

Corollary 6.8. Let Γ, M, M', and τ be as in Definition 6.3. Then:

$$\Gamma \vdash M =_{obs} M' :: \tau \;\Leftrightarrow\; \Gamma \vdash M \; \Delta \; M' :: \tau.$$

In particular, for every $\tau \in \mathit{Typ}$ and $M, M' \in \mathit{Term}(\tau)$:

$$M =_{obs} M' \;\Leftrightarrow\; (M, M') \in \Delta_\tau(). \qquad \blacksquare$$

Together with Observations 5.8 and 5.9, the statement about closed types in the previous corollary implies a statement corresponding to what is usually referred to as the identity extension lemma. This statement says that for every type τ with free variables in $\vec{\alpha} = \alpha_1, \ldots, \alpha_n$, list $\vec{\tau} = \tau_1, \ldots, \tau_n$ of closed types, and list $\vec{r} = r_1, \ldots, r_n$ of relations with

$$r_i = \{(M, M') \mid M, M' \in \mathit{Term}(\tau_i) \wedge M =_{obs} M'\},$$

we have:

$$\Delta_\tau(\vec{r}/\vec{\alpha}) = \{(M, M') \mid M, M' \in \mathit{Term}(\tau[\vec{\tau}/\vec{\alpha}]) \wedge M =_{obs} M'\}.$$

7 Applications

In this section we show how our characterization of observational equivalence by a logical relation can be used to study the semantics of PolySeq. Our study progresses from relatively basic statements about notions from the operational semantics in Section 3.2 to the correctness of parametricity-based program transformations.

7.1 Exploring observational equivalence

Since for every $\tau \in \mathit{Typ}$, $\Delta_\tau()$ is convergence-reflecting by Definition 5.4, we obtain the following simple consequence of Corollary 6.8.

Corollary 7.1. For every $M, M' \in \mathit{Term}$ of the same type, if $M =_{obs} M'$, then $M\Downarrow \;\Leftrightarrow\; M'\Downarrow$. $\blacksquare$

Note that the above does not hold in the setting without **seq**. In particular, Examples 5.3 and 5.5 in [Pit00] provide observational equivalences (of PolyPCF) that do not adhere to convergence-reflection. Namely, they state that for every $\tau_1, \tau_2 \in \mathit{Typ}$, $\lambda x :: \tau_1.\Omega_{\tau_2} =_{obs} \Omega_{\tau_1 \to \tau_2}$, and that for every type τ with at most a single free variable, α say, $\Lambda\alpha.\Omega_\tau =_{obs} \Omega_{\forall\alpha.\tau}$. In light of Observation 3.8 and Corollary 7.1, neither of the two can be true in PolySeq. This motivates turning attention to extensionality principles, because that is where Pitts' Examples 5.3 and 5.5 came from. But before doing so, we look at three different ways of establishing observational equivalences that work independently of whether **seq** is present in the language or not.

First, note that the reverse implication does not hold in Corollary 7.1: just because two terms of the same type have the same convergence or divergence behavior, they are not necessarily observationally equivalent. The reason is that two terms that are both convergent can have otherwise completely different behavior. But all divergent terms of the same type are observationally equivalent for sure, as established by the following lemma.

Lemma 7.2. For every $M, M' \in \mathit{Term}$ of the same type, if $M\Uparrow$ and $M'\Uparrow$, then $M =_{obs} M'$.

Proof: By Corollary 6.8, it suffices to show that $(M, M') \in \Delta_\tau()$, where τ is the type of both M and M'. But this follows from Lemmas 4.11 and 5.7. ■

To establish observational equivalence of two convergent terms, we may use one of the following two lemmas. Each states that a certain operational notion from Section 3.2 implies observational equivalence.

Lemma 7.3. For every $M \in \mathit{Term}$ and $V \in \mathit{Value}$, if $M \Downarrow V$, then $M =_{obs} V$.

Proof: By Corollary 6.8, it suffices to show that $(M, V) \in \Delta_\tau()$, where τ is the type of both M and V. By Lemma 5.7, $\Delta_\tau()$ is $\top\top$-closed, i.e., is equal to $(\Delta_\tau())^{\top\top}$. Then it suffices to reason for every $(S, S') \in (\Delta_\tau())^{\top}$ as follows:

$$\begin{array}{rcll} S \top M & \Leftrightarrow & S \top V & \text{by } M \Downarrow V \text{ and Lemma 4.5} \\ & \Leftrightarrow & S' \top V & \text{by } (S, S') \in (\Delta_\tau())^{\top} \text{ and } (V, V) \in \Delta_\tau()\,, \end{array}$$

where $(V, V) \in \Delta_\tau()$ holds by Theorem 6.4. ■

Lemma 7.4. For every $R, R' \in \mathit{Term}$, if $R \leadsto R'$, then $R =_{obs} R'$.

Proof: Analogous to the proof of Lemma 7.3 above, but using Observation 4.3(2) instead of Lemma 4.5. ■

Combining Lemmas 7.3 and 7.4 with the fact that $=_{obs}$ is compatible, we obtain the following.

Corollary 7.5. For every $A, B \in \mathit{Term}$, if $A\Downarrow$, then $\mathbf{seq}(A, B) =_{obs} B$. ■

7.2 Extensionality principles

This subsection deals with extensionality principles. The standard one for function types — namely that two functions are 'the same' if (and only if) they return the same results for every possible argument — has to be revised for PolySeq. Since **seq** allows an additional observation to be made about function terms, namely checking them for convergence without applying them to any argument, it is additionally necessary that the two function terms in question are either both converging or both diverging.

Lemma 7.6. Let $\tau_1, \tau_2 \in \mathit{Typ}$. Then for every $F, F' \in \mathit{Term}(\tau_1 \to \tau_2)$:

$$F =_{obs} F' \;\text{ iff }\; (F\Downarrow \Leftrightarrow F'\Downarrow) \wedge \forall A \in \mathit{Term}(\tau_1).\; F\;A =_{obs} F'\;A.$$

Proof: The left-to-right implication follows from Corollary 7.1 and the compatibility of $=_{obs}$. For the right-to-left implication, assume $F\Downarrow \Leftrightarrow F'\Downarrow$ and

$$\forall A \in \mathit{Term}(\tau_1).\; F\;A =_{obs} F'\;A. \tag{17}$$

To show $F =_{obs} F'$, it suffices by Corollary 6.8 and clause (9) to prove that $(F, F') \in \Delta_{\tau_1}() \to \Delta_{\tau_2}()$. Since $F\Downarrow \Leftrightarrow F'\Downarrow$ is known, it remains by Definition 5.1 to show that for every $(M, M') \in \Delta_{\tau_1}()$: $(F\;M, F'\;M') \in \Delta_{\tau_2}()$. But this follows by Lemmas 4.10 (for the adequate and compatible relation $=_{obs}$) and 5.7 from

$$(F\;M, F\;M') \in \Delta_{\tau_2}() \wedge F\;M' =_{obs} F'\;M'.$$

Here the first conjunct follows from $(F, F) \in \Delta_{\tau_1}() \to \Delta_{\tau_2}()$ (cf. Theorem 6.4 and clause (9)) by Definition 5.1, while the second one follows from (17). ■

The standard extensionality principle for ∀-types states that two polymorphic terms are observationally equivalent if and only if all their corresponding instances are. In PolySeq we must enrich this principle by an explicit convergence-reflection check, just as we did with the standard extensionality principle for function types above. The reason, of course, is that **seq** can also be used at ∀-types.

Lemma 7.7. Let τ be a type with at most a single free variable, α say. Then for every $G, G' \in \mathit{Term}(\forall\alpha.\tau)$:

$$G =_{obs} G' \;\text{ iff }\; (G\Downarrow \Leftrightarrow G'\Downarrow) \wedge \forall \tau' \in \mathit{Typ}.\; G_{\tau'} =_{obs} G'_{\tau'}. \qquad ■$$

The proof is completely analogous to that of Lemma 7.6 above. We omit it here.

One important consequence of the previous lemma is that in PolySeq it is not true for every $G \in \mathit{Term}(\forall\alpha.\tau)$ that if for every $\tau' \in \mathit{Typ}$, $G_{\tau'} =_{obs} \Omega_{\tau[\tau'/\alpha]}$, then $G =_{obs} \Omega_{\forall\alpha.\tau}$. A counterexample is exactly the term $G = \Lambda\alpha.\Omega_\tau$ from Example 5.5 in [Pit00]. (Note that Example 5.5 in [Pit00] was shown above — below Corollary 7.1 — to fail in the presence of **seq**.) This might seem a bit surprising, given that the refuted implication is the direct translation into the operational setting of laws (2) and (5) in [JV04] and [JV06], respectively. The resolution of this apparent contradiction is that PolySeq combines selective strictness and impredicative polymorphism, while Haskell has only the former (and Pitts' PolyPCF has only the latter). Ultimately, this is also the reason why we had to explicitly enforce $G\Downarrow \Leftrightarrow G'\Downarrow$ in the relational action for ∀-types (cf. Definition 5.2), whereas a corresponding enforcement was not necessary in [JV04, JV06].

The extensionality principle for list types is essentially a coinduction principle. To state it, we need the following notion.

Definition 7.8. Let $\tau \in \mathit{Typ}$ and $s \in \mathit{Rel}(\tau\text{-}\mathit{list}, \tau\text{-}\mathit{list})$. We say that s is a *bisimulation* if for every $(L, L') \in s$ the following hold:

1. $L \Downarrow \mathbf{nil}_\tau$ if and only if $L' \Downarrow \mathbf{nil}_\tau$,
2. if $L \Downarrow H : T$ (for some H and T), then there are H' and T' with $L' \Downarrow H' : T'$, $H =_{obs} H'$, and $(T, T') \in s$, and (conversely)
3. if $L' \Downarrow H' : T'$ (for some H' and T'), then there are H and T with $L \Downarrow H : T$, $H =_{obs} H'$, and $(T, T') \in s$. ◇

Note that it is also possible to define the notion of bisimulation more generally, with an arbitrary relation to tie corresponding list elements together rather than doing so by $=_{obs}$ at some type τ. But the specialized definition above suffices for our purposes and we prefer not to add unnecessary complication. A particularly simple example of a bisimulation is the following.

Lemma 7.9. For every $\tau \in \mathit{Typ}$, the relation $\{(L, L') \mid L, L' \in \mathit{Term}(\tau\text{-}\mathit{list}) \wedge L =_{obs} L'\}$ is a bisimulation.

Proof: Straightforward, using adequacy and compatibility of $=_{obs}$, Corollary 7.1, and Lemmas 7.3 and 7.4. ■

The extensionality principle for list types then amounts to the statement that the bisimulation from the previous lemma is the greatest one at a given $\tau \in \mathit{Typ}$.

Lemma 7.10. Let $\tau \in \mathit{Typ}$. Then for every $L, L' \in \mathit{Term}(\tau\text{-}\mathit{list})$, $L =_{obs} L'$ if and only if (L, L') is contained in some bisimulation.

Proof: In light of Lemma 7.9, it suffices to prove the if-direction. For this, by Corollary 6.8 and clause (11), it suffices to show that $\mathit{lift}(\Delta_\tau())$ subsumes every bisimulation in $\mathit{Rel}(\tau\text{-}\mathit{list}, \tau\text{-}\mathit{list})$. Let $s \in \mathit{Rel}(\tau\text{-}\mathit{list}, \tau\text{-}\mathit{list})$ be a bisimulation and $(L, L') \in s$. Then we establish $(L, L') \in (1 + (\Delta_\tau() \times s))^{\top\top}$ by reasoning for every $(S, S') \in (1 + (\Delta_\tau() \times s))^{\top}$ as follows:

$$
\begin{array}{lll}
 & S \top L & \\
\Leftrightarrow & \exists V \in \mathit{Value}(\tau\text{-}\mathit{list}).\ L \Downarrow V \wedge S \top V & \text{by Lemma 4.5} \\
\Leftrightarrow & (L \Downarrow \mathbf{nil}_\tau \wedge S \top \mathbf{nil}_\tau) \vee \exists H, T.\ L \Downarrow H : T \wedge S \top H : T & \text{by definition} \\
\Leftrightarrow & (L' \Downarrow \mathbf{nil}_\tau \wedge S' \top \mathbf{nil}_\tau) \vee \exists H', T'.\ L' \Downarrow H' : T' \wedge S' \top H' : T' & \\
\Leftrightarrow & \exists V' \in \mathit{Value}(\tau\text{-}\mathit{list}).\ L' \Downarrow V' \wedge S' \top V' & \text{by definition} \\
\Leftrightarrow & S' \top L' & \text{by Lemma 4.5.}
\end{array}
$$

Here the third equivalence follows from s being a bisimulation, $(L, L') \in s$, $(S, S') \in (1 + (\Delta_\tau() \times s))^{\top}$, the definition of $1 + (\Delta_\tau() \times s)$, and Corollary 6.8. Thus, for every bisimulation s in $\mathit{Rel}(\tau\text{-}\mathit{list}, \tau\text{-}\mathit{list})$, we have $s \subseteq (1 + (\Delta_\tau() \times s))^{\top\top}$. This implies $s \subseteq \mathit{lift}(\Delta_\tau())$ by the definition of $\mathit{lift}(\Delta_\tau())$ as the greatest post-fixpoint of the mapping $s \mapsto (1 + (\Delta_\tau() \times s))^{\top\top}$. ■

The intuition for no relevant changes being necessary for list types, neither here in the extensionality principle nor prior to that in the development of the logical relation, is that $\mathbf{seq}(A, B)$ with A of list type can always be 'simulated' by ($\mathbf{case}\ A\ \mathbf{of}\ \{\mathbf{nil} \Rightarrow B;\ h : t \Rightarrow B\}$). Hence, inclusion of **seq** into the language has no real impact on what can happen at list types.

7.3 Manufacturing permissible relations

For applications of the logical relation that make more essential use of the power inherent in the quantification over relations in clause (10), such as those applications studied in the next two subsections, we need a source of appropriately restricted relations. In the setting without **seq**, Pitts has identified a source of $\top\top$-closed relations by considering graphs of evaluation frame stacks, defined as follows.

Definition 7.11. For every $\tau, \tau' \in \mathit{Typ}$ and $S \in \mathit{Stack}(\tau, \tau')$, we define $\mathit{graph}_S \in \mathit{Rel}(\tau, \tau')$ by

$$(M, M') \in \mathit{graph}_S \;\text{ iff }\; S\ M =_{obs} M'. \qquad \diamond$$

Note that graph_S is a true relation, not just a function mapping terms to terms. That is, the same M may be related to several different M'.

Since in PolySeq we have to ensure that relational interpretations of types are not only $\top\top$-closed but also convergence-reflecting, we need to restrict attention to just particular stacks that give rise to such relations. This motivates the following definition.

Definition 7.12. Given $\tau, \tau' \in \mathit{Typ}$ and $S \in \mathit{Stack}(\tau, \tau')$, we say that S is *total* if for every $V \in \mathit{Value}(\tau)$: $(S\ V)\Downarrow$. $\diamond$

Then we can establish the following.

Lemma 7.13. For every $\tau, \tau' \in \mathit{Typ}$ and $S \in \mathit{Stack}(\tau, \tau')$, graph_S is $\top\top$-closed. Moreover, if S is total, then graph_S is convergence-reflecting as well.

Proof: By Corollary 6.8, we first have to show that $(M, M') \in (\mathit{graph}_S)^{\top\top}$ implies $(S\ M, M') \in \Delta_{\tau'}()$. By Lemma 5.7, $\Delta_{\tau'}()$ is $\top\top$-closed, so it suffices to show that $(M, M') \in (\mathit{graph}_S)^{\top\top}$ and $(S', S'') \in (\Delta_{\tau'}())^{\top}$ imply $S' \top S\ M \Leftrightarrow S'' \top M'$. But this follows from Corollary 4.4 and the following reasoning for every $(N, N') \in \mathit{graph}_S$, establishing $(S' \mathbin{@} S, S'') \in (\mathit{graph}_S)^{\top}$ from $(S', S'') \in (\Delta_{\tau'}())^{\top}$:

$$\begin{array}{llll} (S' \mathbin{@} S) \top N & \Leftrightarrow & S' \top S\ N & \text{by Corollary 4.4} \\ & \Leftrightarrow & S'' \top N' & \text{by } (S', S'') \in (\Delta_{\tau'}())^{\top} \text{ and } (S\ N, N') \in \Delta_{\tau'}()\,, \end{array}$$

where $(S\ N, N') \in \Delta_{\tau'}()$ follows from $(N, N') \in \mathit{graph}_S$ by Corollary 6.8.

Now, let S be total and let $(M, M') \in \mathit{graph}_S$, i.e., $S\ M =_{obs} M'$. By Corollary 7.1, this implies that $(S\ M)\Downarrow \Leftrightarrow M'\Downarrow$. So it remains to show that $M\Downarrow \Leftrightarrow (S\ M)\Downarrow$. But this follows from totality of S by Lemma 4.20. ■

7.4 Enumerating terms up to observational equivalence

One application of parametricity and extensionality results presented in [Pit00], and one which is also popular in the Haskell community (as various discussions on [HML] indicate), is to show that certain types have only a small number of inhabitants which differ with respect to observational equivalence. In particular, Pitts' Examples 2.6 and 2.7 state, respectively, that in the absence of **seq**, every element of $\mathit{Term}(\forall\alpha.\alpha)$ is observationally equivalent to Ω, and that every element of $\mathit{Term}(\forall\alpha.\alpha \to \alpha)$ is observationally equivalent either to $\Omega_{\forall\alpha.\alpha\to\alpha}$ or to $\Lambda\alpha.\lambda x :: \alpha.x$. We will revisit these examples for PolySeq and observe interesting differences. But first we need the following auxiliary lemma.

Lemma 7.14. For every $\tau \in \mathit{Typ}$ and $M, M' \in \mathit{Term}(\tau)$, if $M\Uparrow$ and $(M', M') \in \{(M, M)\}^{\top\top}$, then $M'\Uparrow$ as well.

Proof: From $(M', M') \in \{(M, M)\}^{\top\top}$ follows that for every $(S, S') \in \{(M, M)\}^{\top}$: $S \top M' \Leftrightarrow S' \top M'$. Choose $S = \mathbf{seq}(-, \mathbf{nil}_\tau)$ and $S' = \mathbf{seq}(-, \Omega_{\tau\text{-}list})$, which is a valid choice due to the following reasoning:

$$\begin{array}{rcll} \mathbf{seq}(-, \mathbf{nil}_\tau) \top M & \Leftrightarrow & M\Downarrow & \text{by Corollary 4.16} \\ & \Leftrightarrow & M\Downarrow \wedge \mathit{Id} \top \Omega_{\tau\text{-}list} & \text{by } M\Uparrow \\ & \Leftrightarrow & \mathit{Id} \top \mathbf{seq}(M, \Omega_{\tau\text{-}list}) & \text{by Lemma 4.15} \\ & \Leftrightarrow & \mathbf{seq}(-, \Omega_{\tau\text{-}list}) \top M & \text{by Observation 4.3(1).} \end{array}$$

Now, $\mathbf{seq}(-, \mathbf{nil}_\tau) \top M' \Leftrightarrow \mathbf{seq}(-, \Omega_{\tau\text{-}list}) \top M'$ implies

$$M'\Downarrow \Leftrightarrow (M'\Downarrow \wedge \mathit{Id} \top \Omega_{\tau\text{-}list})$$

by using Observation 4.3(1), Lemma 4.15, and Corollary 4.16 in a manner similar to that above. Since $\mathit{Id} \top \Omega_{\tau\text{-}list}$ does not hold (cf. Observations 3.8 and 4.2), this implies $M'\Uparrow$. ■

Now, we can show that although not all elements of $\mathit{Term}(\forall\alpha.\alpha)$ are observationally equivalent in an impredicative calculus including **seq**, they do separate into exactly two equivalence classes.

Lemma 7.15. For every $G \in \mathit{Term}(\forall\alpha.\alpha)$, either $G =_{obs} \Omega$ or $G =_{obs} \Omega_{\forall\alpha.\alpha}$.

Proof: By Theorem 6.4 and Definition 5.4, we have $(G, G) \in (\forall r.r^{\top\top})$. By Definition 5.2, this implies that for every $\tau \in \mathit{Typ}$ and $r \in \mathit{Rel}^{\Downarrow}(\tau, \tau)$: $(G_\tau, G_\tau) \in r^{\top\top}$. For fixed τ, choose $r = \{(\Omega_\tau, \Omega_\tau)\}$, which obviously is convergence-reflecting. Then we get $G_\tau\Uparrow$ by Observation 3.8 and Lemma 7.14. Since this is so for every $\tau \in \mathit{Typ}$, the claim follows by Observation 3.8, Corollary 4.22, and Lemmas 7.2 and 7.7. ■

To provide a similar account for the type $\forall\alpha.\alpha \to \alpha$, we first need a further auxiliary lemma. Its statement should also hold for Pitts' setting without **seq**, but a proof would be much more complicated there, as it could not make use of the convergence-reflection of relational interpretations of types. In fact, there seems to be no way in PolyPCF to prove the statement solely based on the reflexivity of the logical relation as below.

Lemma 7.16. Let τ be a type with at most a single free variable, α say. Then for every $G \in \mathit{Term}(\forall\alpha.\tau)$:

$$(\exists\tau' \in \mathit{Typ}.\ G_{\tau'}\Downarrow) \Rightarrow (\forall\tau' \in \mathit{Typ}.\ G_{\tau'}\Downarrow).$$

Proof: By Theorem 6.4 and clause (10), we have $(G, G) \in (\forall r.\Delta_\tau(r^{\top\top}/\alpha))$. By Definition 5.2, this implies that for every $\tau_1, \tau_2 \in \mathit{Typ}$ and $r \in \mathit{Rel}^{\Downarrow}(\tau_1, \tau_2)$: $(G_{\tau_1}, G_{\tau_2}) \in \Delta_\tau(r^{\top\top}/\alpha)$. For fixed τ_1 and τ_2, choose $r = \varnothing$, which obviously is convergence-reflecting. By Lemma 4.18 and Definition 5.4, $\Delta_\tau(r^{\top\top}/\alpha)$ is then also convergence-reflecting. This means that for every $\tau_1, \tau_2 \in \mathit{Typ}$: $G_{\tau_1}\Downarrow \Leftrightarrow G_{\tau_2}\Downarrow$. ■

Now, we can show that in PolySeq, $\mathit{Term}(\forall\alpha.\alpha \to \alpha)$ is factorized into four, rather than two, equivalence classes with respect to $=_{obs}$. The two additional ones arise exactly from the failures of Pitts' Examples 5.3 and 5.4 in the presence of **seq**, as observed below Corollary 7.1.

Lemma 7.17. For every $G \in \mathit{Term}(\forall\alpha.\alpha \to \alpha)$, exactly one of the following holds:

1. $G =_{obs} \Omega_{\forall\alpha.\alpha\to\alpha}$,
2. $G =_{obs} \Lambda\alpha.\Omega_{\alpha\to\alpha}$,
3. $G =_{obs} \Lambda\alpha.\lambda x :: \alpha.\Omega_\alpha$, or
4. $G =_{obs} \Lambda\alpha.\lambda x :: \alpha.x$.

Proof: By Theorem 6.4 and Definition 5.4, we have $(G, G) \in (\forall r.r^{\top\top} \to r^{\top\top})$. By Definitions 5.1 and 5.2, this implies the following:

$$\begin{array}{l}\forall\tau_1, \tau_2 \in \mathit{Typ}, r \in \mathit{Rel}^{\Downarrow}(\tau_1, \tau_2), M_1 \in \mathit{Term}(\tau_1), M_2 \in \mathit{Term}(\tau_2).\\ \quad (M_1, M_2) \in r^{\top\top} \Rightarrow (G_{\tau_1}\ M_1, G_{\tau_2}\ M_2) \in r^{\top\top}.\end{array} \tag{18}$$

From this, we get that

$$\forall\tau \in \mathit{Typ}, M \in \mathit{Term}(\tau).\ M\Uparrow \Rightarrow (G_\tau\ M)\Uparrow \tag{19}$$

by instantiating $\tau_1 = \tau_2 = \tau$, $r = \{(M, M)\}$, and $M_1 = M_2 = M$, and by using (2) and Lemma 7.14.

Now, we prove that

$$\forall\tau \in \mathit{Typ}, M \in \mathit{Term}(\tau).\ G_\tau\ M =_{obs} \Omega_\tau \tag{20}$$

or

$$\forall\tau \in \mathit{Typ}, M \in \mathit{Term}(\tau).\ G_\tau\ M =_{obs} M\,, \tag{21}$$

by contradiction. Assume that neither (20) nor (21) holds. Then there exist $\tau_1, \tau_2 \in \mathit{Typ}$, $M_1 \in \mathit{Term}(\tau_1)$, and $M_2 \in \mathit{Term}(\tau_2)$ with

$$\neg(G_{\tau_1}\ M_1 =_{obs} \Omega_{\tau_1})$$

and

$$\neg(G_{\tau_2}\ M_2 =_{obs} M_2). \tag{22}$$

From these observational inequivalences we get $(G_{\tau_1}\ M_1)\Downarrow$, $M_1\Downarrow$, and $M_2\Downarrow$ by Observation 3.8, Lemma 7.2, and (19). Using (18) with $r = \mathit{graph}_{\mathbf{seq}(-,M_2)}$, which is $\top\top$-closed and convergence-reflecting by $M_2\Downarrow$, Observation 3.7, and Lemma 7.13, we get that $(M_1, M_2) \in \mathit{graph}_{\mathbf{seq}(-,M_2)}$ implies $(G_{\tau_1}\ M_1, G_{\tau_2}\ M_2) \in \mathit{graph}_{\mathbf{seq}(-,M_2)}$, i.e.,

$$\mathbf{seq}(M_1, M_2) =_{obs} M_2 \;\Rightarrow\; \mathbf{seq}(G_{\tau_1}\ M_1, M_2) =_{obs} G_{\tau_2}\ M_2.$$

But by $M_1\Downarrow$, $(G_{\tau_1}\ M_1)\Downarrow$, and Corollary 7.5, the precondition of this implication is fulfilled, whereas its conclusion contradicts (22). Consequently, the assumption that neither (20) nor (21) holds was wrong. From this, the claim follows by Observation 3.8, Corollary 7.1, and Lemmas 7.2, 7.4, 7.6, 7.7, and 7.16. ■

7.5 Correctness of short cut fusion

The PolySeq equivalents of the Haskell functions *foldr* and *build* from Figure 1 in Section 2 are as follows:

$$\mathit{foldr} = \Lambda\alpha.\Lambda\beta.\lambda c :: \alpha \to \beta \to \beta.\lambda n :: \beta.\mathbf{fix}(\lambda f :: \alpha\text{-}\mathit{list} \to \beta.\lambda l :: \alpha\text{-}\mathit{list}.\\ \mathbf{case}\ l\ \mathbf{of}\ \{\mathbf{nil} \Rightarrow n;\ h : t \Rightarrow c\ h\ (f\ t)\})$$

and

$$\mathit{build} = \Lambda\alpha.\lambda g :: \forall\beta.(\alpha \to \beta \to \beta) \to \beta \to \beta.g_{\alpha\text{-}\mathit{list}}\ (\lambda h :: \alpha.\lambda t :: \alpha\text{-}\mathit{list}.h : t)\ \mathbf{nil}_\alpha.$$

Due to the explicit syntactic representation of type specialization in PolySeq, the short cut fusion rule (1) becomes a bit more verbose now. It says that for every $\tau, \tau' \in \mathit{Typ}$, $G \in \mathit{Term}(\forall\beta.(\tau \to \beta \to \beta) \to \beta \to \beta)$, $C \in \mathit{Term}(\tau \to \tau' \to \tau')$, and $N \in \mathit{Term}(\tau')$:

$$(\mathit{foldr}_\tau)_{\tau'}\ C\ N\ (\mathit{build}_\tau\ G) =_{obs} G_{\tau'}\ C\ N \tag{23}$$

Of course, just as noted for 'Haskell with *seq*' in Section 2, the rule does not hold unconstrained in PolySeq. For example, it is easy to see that the above observational equivalence does not hold if G is $\Lambda\beta.\lambda c :: \tau \to \beta \to \beta.\lambda n :: \beta.\mathbf{seq}(c, n)$, C is $\Omega_{\tau\to\tau'\to\tau'}$, and N is any converging element of $\mathit{Term}(\tau')$.

But correctness of short cut fusion can be recovered, even in PolySeq with its selective strictness primitive, by imposing restrictions on N and C as follows.

Theorem 7.18. If $N\Downarrow$ and for every $A \in \mathit{Term}(\tau)$ and $B \in \mathit{Term}(\tau')$, $(C\ A\ B)\Downarrow$, then (23) holds.

Proof: By Theorem 6.4 and Definition 5.4, we have

$$(G, G) \in (\forall r.(\Delta_\tau(r^{\top\top}/\beta) \to (r^{\top\top} \to r^{\top\top})) \to (r^{\top\top} \to r^{\top\top})).$$

By Lemma 4.18, Definitions 5.1 and 5.2, and Observation 5.8, this implies that for every $r \in \mathit{Rel}^{\Downarrow}(\tau\text{-}\mathit{list}, \tau')$, $C_1 \in \mathit{Term}(\tau \to \tau\text{-}\mathit{list} \to \tau\text{-}\mathit{list})$, $C_2 \in \mathit{Term}(\tau \to \tau' \to \tau')$, $N_1 \in \mathit{Term}(\tau\text{-}\mathit{list})$, and $N_2 \in \mathit{Term}(\tau')$:

$$\begin{aligned} & C_1\Downarrow \Leftrightarrow C_2\Downarrow \\ & \wedge (\forall (A_1, A_2) \in \Delta_\tau().\ (C_1\ A_1)\Downarrow \Leftrightarrow (C_2\ A_2)\Downarrow \\ & \qquad \wedge \forall (B_1, B_2) \in r^{\top\top}.\ (C_1\ A_1\ B_1, C_2\ A_2\ B_2) \in r^{\top\top}) \\ & \wedge (N_1, N_2) \in r^{\top\top} \\ \Rightarrow\ & (G_{\tau\text{-}\mathit{list}}\ C_1\ N_1, G_{\tau'}\ C_2\ N_2) \in r^{\top\top}. \end{aligned}$$

We instantiate $C_1 = \lambda h :: \tau.\lambda t :: \tau\text{-}\mathit{list}.h : t$, $C_2 = C$, $N_1 = \mathbf{nil}_\tau$, $N_2 = N$, and

$$r = \{(L, M) \mid L \in \mathit{Term}(\tau\text{-}\mathit{list}) \wedge M \in \mathit{Term}(\tau') \wedge (\mathit{foldr}_\tau)_{\tau'}\ C\ N\ L =_{obs} M\}.$$

By the definition of *foldr*, Lemma 7.4, and the compatibility of $=_{obs}$, this r equals graph_S, where

$$\begin{aligned} S = \mathbf{case} - \mathbf{of}\ \{&\mathbf{nil} \Rightarrow N; \\ & h : t \Rightarrow C\ h\ (\mathbf{fix}(\lambda f :: \tau\text{-}\mathit{list} \to \tau'.\lambda l :: \tau\text{-}\mathit{list}. \\ & \qquad \mathbf{case}\ l\ \mathbf{of}\ \{\mathbf{nil} \Rightarrow N;\ h' : t' \Rightarrow C\ h'\ (f\ t')\})\ t)\}. \end{aligned}$$

Since S is total by Observation 3.7 and the preconditions on N and C, the chosen r is $\top\top$-closed and convergence-reflecting by Lemma 7.13. Hence, that choice was justified (i.e., r really is in $\mathit{Rel}^{\Downarrow}(\tau\text{-}\mathit{list}, \tau')$), and occurrences of $r^{\top\top}$ in the implication displayed above can be replaced by r itself. Then the claim follows from the definition of *build*, Lemma 7.4, the compatibility of $=_{obs}$, and Observation 3.7, provided we can establish that $C\Downarrow$, that for every $A_2 \in \mathit{Term}(\tau)$, $(C\ A_2)\Downarrow$, that

$$\begin{aligned} & \forall (A_1, A_2) \in \Delta_\tau(), B_1 \in \mathit{Term}(\tau\text{-}\mathit{list}), B_2 \in \mathit{Term}(\tau'). \\ & \quad (\mathit{foldr}_\tau)_{\tau'}\ C\ N\ B_1 =_{obs} B_2 \\ & \quad \Rightarrow (\mathit{foldr}_\tau)_{\tau'}\ C\ N\ ((\lambda h :: \tau.\lambda t :: \tau\text{-}\mathit{list}.h : t)\ A_1\ B_1) =_{obs} C\ A_2\ B_2\,, \end{aligned}$$

and that

$$(\mathit{foldr}_\tau)_{\tau'}\ C\ N\ \mathbf{nil}_\tau =_{obs} N.$$

But the two ‘convergence conditions’ follow from the precondition on C by Corollary 4.21, whereas the other two statements follow from Corollary 6.8, the definition of *foldr*, Lemma 7.4, and the compatibility of $=_{obs}$. ■

Note that the above proof relies on Corollary 6.8 rather than just on the weaker Theorem 6.4. This is exactly where the proofs in [JV04, JV06] had to resort to an unproved conjecture, because the statement corresponding to the identity extension lemma for the (denotational-style) logical relation offered there could not be shown to hold. This has now been remedied in the operational setting. The preconditions on N and C imposed in Theorem 7.18 of course closely correspond to the denotational ones suggested for total correctness of short cut fusion in [JV04] and [JV06]. For pragmatic discussions of the implications these preconditions have in practice, consult the latter paper.

8 Conclusion and related/future work

We have constructed a parametric model of observational equivalence for a nonstrict polymorphic lambda calculus integrating general recursion, an algebraic datatype, and selective strictness. This puts earlier — more intuition-based than formally-derived — accounts [JV04, JV06] of free theorems and parametricity-based program transformations for functional languages mixing all these features, like Clean and Haskell, on a firm theoretical (but at the same time implementation-adequate) basis. The obvious next step is to follow the earlier accounts' move from an equational to an inequational setting, that is, to characterize observational *approximation* by a logical relation. For some applications such a move is essential, for example for proving safe applicability of the transformations from [Voi02] in a language including selective strictness. As motivated in [JV04, JV06] based on practical concerns regarding the transformation from [Sve02], moving from an equational to an inequational development is of interest even for a setting in which selective strictness is precluded. Regarding the actual realization of an inequational version of the kind of operational development given in this paper, some technical challenges still lie ahead. For example, the denotational-style logical relation developed in [JV04, JV06] uses the semantic approximation relation inside its definition. Directly translating this into the operational setting would mean using observational approximation inside the definition of the logical relation. In the current paper, however, we could define the logical relation without referring to observational equivalence. Of course, this way of doing things is preferable, as it then establishes the logical relation as a completely independent characterization of the semantic notion in which we are interested. But whether it is feasible in the inequational setting remains open for now.

In retrospect, an interesting relationship can be observed between the restrictions introduced in [JV04, JV06] to accommodate selective strictness — or, more precisely, their equational and operational-style incarnations coming forward in the present paper — and ideas from both Pitts' original paper on PolyPCF [Pit00] and his extended study in [Pit05] of what is essentially 'Call-by-value PolyPCF'. Firstly, Figure 9 of [Pit00] proposes to use roughly the following as relational interpretation of function types in a call-by-name calculus 'Lazy PCF' in which termination at function types is observable:

$$\{(\lambda x :: \tau_1.M, \lambda x :: \tau_1'.M') \mid \forall (A, A') \in r_1.\ (M[A/x], M'[A'/x]) \in r_2\}^{\top\top}$$

Since function abstractions are values, and thus converge, the relation of which the $\top\top$-closure is taken here is clearly convergence-reflecting. By our Lemma 4.18, the proposed relational interpretation of function types is then itself also convergence-reflecting. So the above is quite reminiscent of $r_1 \to r_2$ in our Definition 5.1. For a calculus incorporating impredicative polymorphic types, that is, for 'Lazy PolyPCF' rather than 'Lazy PCF', one would of course have to ensure convergence-reflection of the relational interpretation of $\forall$-types as well, either explicitly as we do in Definition 5.2 or more indirectly in a manner similar to that above. But even then, one would not yet have a logical relation appropriate for PolySeq. For while the aforementioned adaptations account for observability of whole program termination at arbitrary types in 'Lazy PolyPCF', they do not fully capture the impact of selective strictness in PolySeq, namely that termination of arbitrary intermediate computations becomes observable as well. To account for that, we have imposed convergence-reflection not only on the *result* of each relational action, but also on the relations over which we *quantify* in the relational action for ∀-types. And interestingly, something similar happens in Pitts' account for 'Call-by-value PolyPCF' in [Pit05]. The enforcement is again done quite indirectly rather than in the direct way we have preferred in the present paper (and which is indeed much more preferable when it comes to applying the logical relation as in the previous section), but the basic idea is the same. Nevertheless, as a whole, the logical relation from [Pit05] is also not appropriate for PolySeq. In particular, the relational action for function types given there requires relatedness of function results only for related function arguments that are values. Regarding the extensionality principle for function types, this would imply that $A \in \mathit{Term}(\tau_1)$ in Lemma 7.6 is replaced by $A \in \mathit{Value}(\tau_1)$. And while this gives a correct principle for a purely strict calculus, it is wrong for PolySeq. That the logical relation characterizing PolySeq observational equivalence can ultimately be understood as a kind of blend of the logical relations for 'Lazy PolyPCF' and 'Call-by-value PolyPCF' is not entirely surprising, as it corresponds to the fact that intuitively 'Haskell with *seq*' is situated somewhere between 'Haskell without *seq*' and a purely strict language like ML. But, of course, finding just the right blend, or equivalently, identifying the precise position selective strictness occupies between pure nonstrictness and pure strictness was the challenging task we set out to solve with this paper. Interestingly, the picture drawn up above finds a complement, and is tied back to denotational semantics, in recent domain-theoretic work by Møgelberg [Møg06]. There, he sketches a program logic with the potential to transfer the results of [Pit05] to a denotational setting. And for the relational interpretations of types put forward he appeals to our results in [JV04].

Pitts' study of parametric polymorphism for the purely strict version of PolyPCF was mainly motivated by concerns regarding existential types (∃-types) rather than ∀-types, and a similar motivation underlies other recent work on operationally-based logical relations for purely strict calculi [Ahm06].[6] Hence, it would be interesting to see how things change when ∃-types are integrated into PolySeq as well. It is easy to see that the observationally isomorphic encoding of ∃-types by ∀-types provided for purely nonstrict PolyPCF in Section 7 of [Pit00] breaks down in PolySeq (just as the encoding of list types from Example 2.8 of that paper does). So if we want to reason about existential types, they must be explicitly added to the calculus, and their interplay with selective strictness must be studied very carefully. It may very well be that the Haskell community, which is currently trying to improve the language's covering of existential types as part of the 'Haskell Prime' standardization process, is in for some surprises here, just as was the case with respect to the impact of selective strictness on ∀-polymorphism when [JV04] appeared.

[6]Interestingly, [Ahm06] studies an inequational logical relation as we did in [JV04, JV06].

References

[ABB+05] A. Abel, M. Benke, A.L. Bove, R.J.M. Hughes, and U. Norell. Verifying Haskell programs using constructive type theory. In *Haskell Workshop, Proceedings*, pages 62–73. ACM Press, 2005.

[Ahm06] A. Ahmed. Step-indexed syntactic logical relations for recursive and quantified types. In *European Symposium on Programming, Proceedings*, volume 3924 of *LNCS*, pages 69–83. Springer-Verlag, 2006.

[BFSS90] E.S. Bainbridge, P.J. Freyd, A. Scedrov, and P.J. Scott. Functorial polymorphism. *Theoretical Computer Science*, 70:35–64, 1990. Corrigendum *Ibid.*, 71:431, 1990.

[BMP06] L. Birkedal, R.E. Møgelberg, and R.L. Petersen. Parametric domain-theoretic models of polymorphic intuitionistic / linear lambda calculus. In *Mathematical Foundations of Programming Semantics 2005, Proceedings*, volume 155 of *ENTCS*, pages 191–217. Elsevier B.V., 2006.

[CLR] Clean Language Report 2.0 (`http://www.cs.ru.nl/~clean/CleanExtra/report20/index.html`).

[DHJG06] N.A. Danielsson, R.J.M. Hughes, P. Jansson, and J. Gibbons. Fast and loose reasoning is morally correct. In *Principles of Programming Languages, Proceedings*, pages 206–217. ACM Press, 2006.

[DJ04] N.A. Danielsson and P. Jansson. Chasing bottoms — A case study in program verification in the presence of partial and infinite values. In *Mathematics of Program Construction, Proceedings*, volume 3125 of *LNCS*, pages 85–109. Springer-Verlag, 2004.

[EM06] M. van Eekelen and M. de Mol. Proof tool support for explicit strictness. In *Implementation and Application of Functional Languages 2005, Selected Papers*, LNCS, to appear. Springer-Verlag, 2006.

[FFKD87] M. Felleisen, D.P. Friedman, E. Kohlbecker, and B. Duba. A syntactic theory of sequential control. *Theoretical Computer Science*, 52:205–237, 1987.

[Fri75] H. Friedman. Equality between functionals. In *Logic Colloquium '72–73, Proceedings*, pages 22–37. Springer-Verlag, 1975.

[GHC] The Glasgow Haskell Compiler (`http://www.haskell.org/ghc`).

[Gir72] J.-Y. Girard. *Interprétation functionelle et élimination des coupures dans l'arithmétique d'ordre supérieure.* PhD thesis, Université Paris VII, 1972.

[GJUV05] N. Ghani, P. Johann, T. Uustalu, and V. Vene. Monadic augment and generalised short cut fusion. In *International Conference on Functional Programming, Proceedings*, pages 294–305. ACM Press, 2005.

[GLP93] A. Gill, J. Launchbury, and S.L. Peyton Jones. A short cut to deforestation. In *Functional Programming Languages and Computer Architecture, Proceedings*, pages 223–232. ACM Press, 1993.

[Has91] R. Hasegawa. Parametricity of extensionally collapsed term models of polymorphism and their categorical properties. In *Theoretical Aspects of Computer Software, Proceedings*, volume 526 of *LNCS*, pages 495–512. Springer-Verlag, 1991.

[HK05] W.L. Harrison and R.B. Kieburtz. The logic of demand in Haskell. *Journal of Functional Programming*, 15:837–891, 2005.

[HML] The Haskell Mailing List Archive (`http://www.mail-archive.com/haskell@haskell.org`).

[HS97] R. Harper and C. Stone. An interpretation of Standard ML in type theory. Technical Report CMU-CS-97-147, Carnegie Mellon University, 1997.

[Joh02] P. Johann. A generalization of short-cut fusion and its correctness proof. *Higher-Order and Symbolic Computation*, 15:273–300, 2002.

[Joh03] P. Johann. Short cut fusion is correct. *Journal of Functional Programming*, 13:797–814, 2003.

[Joh05] P. Johann. On proving the correctness of program transformations based on free theorems for higher-order polymorphic calculi. *Mathematical Structures in Computer Science*, 15:201–229, 2005.

[JV04] P. Johann and J. Voigtländer. Free theorems in the presence of *seq*. In *Principles of Programming Languages, Proceedings*, pages 99–110. ACM Press, 2004.

[JV06] P. Johann and J. Voigtländer. The impact of *seq* on free theorems-based program transformations. *Fundamenta Informaticae*, 69:63–102, 2006.

[Lei83] D. Leivant. Polymorphic type inference. In *Principles of Programming Languages, Proceedings*, pages 88–98. ACM Press, 1983.

[LP96] J. Launchbury and R. Paterson. Parametricity and unboxing with unpointed types. In *European Symposium on Programming, Proceedings*, pages 204–218. Springer-Verlag, 1996.

[Møg06] R.E. Møgelberg. Interpreting polymorphic FPC into domain theoretic models of parametric polymorphism. In *International Colloquium on Automata, Languages and Programming, Proceedings*, LNCS, to appear. Springer-Verlag, 2006.

[Pey03] S.L. Peyton Jones, editor. *Haskell 98 Language and Libraries: The Revised Report*. Cambridge University Press, 2003.

[Pit00] A.M. Pitts. Parametric polymorphism and operational equivalence. *Mathematical Structures in Computer Science*, 10:321–359, 2000.

[Pit05] A.M. Pitts. Typed operational reasoning. In B.C. Pierce, editor, *Advanced Topics in Types and Programming Languages*, pages 245–289. MIT Press, 2005.

[Plo73] G.D. Plotkin. Lambda-definability and logical relations. Memorandum SAI-RM-4, University of Edinburgh, 1973.

[Plo04] G.D. Plotkin. A structural approach to operational semantics. *Journal of Logic and Algebraic Programming*, 60-61:17–139, 2004.

[PLST98] S.L. Peyton Jones, J. Launchbury, M.B. Shields, and A.P. Tolmach. Bridging the gulf: A common intermediate language for ML and Haskell. In *Principles of Programming Languages, Proceedings*, pages 49–61. ACM Press, 1998.

[Rey74] J.C. Reynolds. Towards a theory of type structure. In *Colloque sur la Programmation, Proceedings*, pages 408–423. Springer-Verlag, 1974.

[Rey83] J.C. Reynolds. Types, abstraction and parametric polymorphism. In *Information Processing, Proceedings*, pages 513–523. Elsevier Science Publishers B.V., 1983.

[RMP06] B. Rudiak-Gould, A. Mycroft, and S.L. Peyton Jones. Haskell is not not ML. In *European Symposium on Programming, Proceedings*, volume 3924 of *LNCS*, pages 38–53. Springer-Verlag, 2006.

[RR94] E.P. Robinson and G. Rosolini. Reflexive graphs and parametric polymorphism. In *Logic in Computer Science, Proceedings*, pages 364–371. IEEE Computer Society Press, 1994.

[RS04] G. Rosolini and A. Simpson. Using synthetic domain theory to prove operational properties of a polymorphic programming language based on strictness. Manuscript, 2004.

[Sta85] R. Statman. Logical relations and the typed lambda-calculus. *Information and Control*, 65:85–97, 1985.

[Sve02] J. Svenningsson. Shortcut fusion for accumulating parameters & zip-like functions. In *International Conference on Functional Programming, Proceedings*, pages 124–132. ACM Press, 2002.

[THLP98] P.W. Trinder, K. Hammond, H.-W. Loidl, and S.L. Peyton Jones. Algorithm + Strategy = Parallelism. *Journal of Functional Programming*, 8:23–60, 1998.

[TM95] A. Takano and E. Meijer. Shortcut deforestation in calculational form. In *Functional Programming Languages and Computer Architecture, Proceedings*, pages 306–313. ACM Press, 1995.

[Voi02] J. Voigtländer. Concatenate, reverse and map vanish for free. In *International Conference on Functional Programming, Proceedings*, pages 14–25. ACM Press, 2002.

[VWP06] D. Vytiniotis, S. Weirich, and S.L. Peyton Jones. Boxy types: inference for higher-rank types and impredicativity. In *International Conference on Functional Programming, Proceedings*, to appear. ACM Press, 2006.

[Wad89] P. Wadler. Theorems for free! In *Functional Programming Languages and Computer Architecture, Proceedings*, pages 347–359. ACM Press, 1989.

[WB89] P. Wadler and S. Blott. How to make *ad-hoc* polymorphism less *ad hoc*. In *Principles of Programming Languages, Proceedings*, pages 60–76. ACM Press, 1989.

A Additional proofs

Proof of Lemma 4.5: For fixed $\tau, \tau' \in \mathit{Typ}$ and $S \in \mathit{Stack}(\tau, \tau'\text{-}\mathit{list})$, we first prove that for every $V \in \mathit{Value}(\tau)$, $t \in \mathbb{N}$, $\tau'' \in \mathit{Typ}$, $M \in \mathit{Term}(\tau'')$, and $S' \in \mathit{Stack}(\tau'', \tau)$:

$$(S', M) \rightarrowtail^t (\mathit{Id}, V) \;\Rightarrow\; (S \mathbin{@} S', M) \rightarrowtail^* (S, V) \tag{24}$$

by induction on t. The induction base ($t = 0$) is straightforward, using that $(S', M) = (\mathit{Id}, V)$ implies $S' = \mathit{Id}$ and $M = V$. For the induction step ($t \to t + 1$), we proceed by case distinction on the first transition in

$$(S', M) \rightarrowtail (S'', M') \rightarrowtail^t (\mathit{Id}, V)$$

as follows:

Case a: $M = E\{M'\}$ and $S'' = S' \circ E$, where E is an evaluation frame and $M' \notin \mathit{Value}$. Then by $(S'', M') \rightarrowtail^t (\mathit{Id}, V)$ and the induction hypothesis for t we know that:

$$(S \mathbin{@} (S' \circ E), M') \rightarrowtail^* (S, V).$$

From this, the induction claim follows by $(S \mathbin{@} S', M) \rightarrowtail ((S \mathbin{@} S') \circ E, M')$.

Case b: $S' = S'' \circ E$ and $M' = E\{M\}$, where E is an evaluation frame and $M \in \mathit{Value}$. Then by $(S'', M') \rightarrowtail^t (\mathit{Id}, V)$ and the induction hypothesis for t we know that:

$$(S \mathbin{@} S'', E\{M\}) \rightarrowtail^* (S, V).$$

From this, the induction claim follows by $((S \mathbin{@} S'') \circ E, M) \rightarrowtail (S \mathbin{@} S'', E\{M\})$.

Case c: $S'' = S'$ and $M \rightsquigarrow M'$. Then by $(S'', M') \rightarrowtail^t (\mathit{Id}, V)$ and the induction hypothesis for t we know that:

$$(S \mathbin{@} S', M') \rightarrowtail^* (S, V).$$

From this, the induction claim follows by $(S \mathbin{@} S', M) \rightarrowtail (S \mathbin{@} S', M')$.

Now, for the same fixed τ, τ', and S, we prove that for every $t \in \mathbb{N}$, $\tau'' \in \mathit{Typ}$, $M \in \mathit{Term}(\tau'')$, and $S' \in \mathit{Stack}(\tau'', \tau)$:

$$(S \mathbin{@} S', M) \rightarrowtail^t (\mathit{Id}, \mathbf{nil}_{\tau'}) \Rightarrow \exists V \in \mathit{Value}(\tau).\ (S', M) \rightarrowtail^* (\mathit{Id}, V) \wedge S \top V \qquad (25)$$

by induction on t. The induction base ($t = 0$) is straightforward, using that $(S \mathbin{@} S', M) = (\mathit{Id}, \mathbf{nil}_{\tau'})$ implies $S = S' = \mathit{Id} \in \mathit{Stack}(\tau'\text{-}\mathit{list}, \tau'\text{-}\mathit{list})$ and $M = \mathbf{nil}_{\tau'}$. For the induction step ($t \to t+1$), we proceed by case distinction on the first transition in

$$(S \mathbin{@} S', M) \rightarrowtail (S'', M') \rightarrowtail^t (\mathit{Id}, \mathbf{nil}_{\tau'})$$

as follows:

Case a: $M = E\{M'\}$ and $S'' = (S \mathbin{@} S') \circ E = S \mathbin{@} (S' \circ E)$, where E is an evaluation frame and $M' \notin \mathit{Value}$. Then by $(S'', M') \rightarrowtail^t (\mathit{Id}, \mathbf{nil}_{\tau'})$ and the induction hypothesis for t, we know that there exists a $V \in \mathit{Value}(\tau)$ such that:

$$(S' \circ E, M') \rightarrowtail^* (\mathit{Id}, V) \wedge S \top V.$$

From this, the induction claim follows by $(S', M) \rightarrowtail (S' \circ E, M')$.

Case b: $S \mathbin{@} S' = S'' \circ E$ and $M' = E\{M\}$, where E is an evaluation frame and $M \in \mathit{Value}$. Then we perform a further case distinction on S' being equal to Id or not.

Case b.1: $S' = \mathit{Id}$. Then clearly $\tau'' = \tau$ and thus $M \in \mathit{Value}(\tau)$. Moreover, $(S', M) \rightarrowtail^* (\mathit{Id}, M)$. The induction claim then follows from $(S \mathbin{@} S', M) \rightarrowtail^{t+1} (\mathit{Id}, \mathbf{nil}_{\tau'})$, which implies $S \top M$.

Case b.2: $S' = S''' \circ E$ for some evaluation frame stack S''' with $S \mathbin{@} S''' = S''$. Then by $(S'', M') \rightarrowtail^t (\mathit{Id}, \mathbf{nil}_{\tau'})$ and the induction hypothesis for t, we know that there exists a $V \in \mathit{Value}(\tau)$ such that:

$$(S''', M') \rightarrowtail^* (\mathit{Id}, V) \wedge S \top V.$$

From this, the induction claim follows by $(S', M) \rightarrowtail (S''', M')$.

Case c: $S'' = S \mathbin{@} S'$ and $M \rightsquigarrow M'$. Then by $(S'', M') \rightarrowtail^t (Id, \mathbf{nil}_{\tau'})$ and the induction hypothesis for t, we know that there exists a $V \in \mathit{Value}(\tau)$ such that:

$$(S', M') \rightarrowtail^* (Id, V) \wedge S \top V$$

From this, the induction claim follows by $(S', M) \rightarrowtail (S', M')$.

Finally, the lemma follows by combining (24) and (25), both specialized to the case $\tau'' = \tau$ and $S' = Id$, with the definitions of $\top$ and $\Downarrow$. ■

Proof of Lemma 4.13: For the proof we need the notion of substitution in evaluation frames and evaluation frame stacks, introduced as follows. Let x be a term variable. The concept of x occurring free in an evaluation frame (stack) is defined in the obvious way. The result of substituting a term M for all free occurrences of x in an evaluation frame E is denoted by $E[M/x]$. Substitution in an evaluation frame stack S, denoted by $S[M/x]$, is by corresponding substitution in all evaluation frames constituting S.

Let $\tau \in \mathit{Typ}$ and $F \in \mathit{Term}(\tau \to \tau)$. We will use the fact that for every $\tau' \in \mathit{Typ}$, term N with $x :: \tau \vdash N :: \tau'$, and $n \in \mathbb{N}$:

$$N[\mathbf{fix}^{(n)}(F)/x] \in \mathit{Value} \Leftrightarrow N[\mathbf{fix}(F)/x] \in \mathit{Value}. \tag{26}$$

Let $\tau'' \in \mathit{Typ}$. First, we prove that for every $n, t \in \mathbb{N}$, $\tau' \in \mathit{Typ}$, evaluation frame stack S with $x :: \tau \vdash S :: \tau' \multimap \tau''\text{-}\mathit{list}$, and term M with $x :: \tau \vdash M :: \tau'$:

$$\begin{aligned} &(S[\mathbf{fix}^{(n)}(F)/x], M[\mathbf{fix}^{(n)}(F)/x]) \rightarrowtail^t (Id, \mathbf{nil}_{\tau''}) \\ \Rightarrow\ &(S[\mathbf{fix}^{(n+1)}(F)/x], M[\mathbf{fix}^{(n+1)}(F)/x]) \rightarrowtail^t (Id, \mathbf{nil}_{\tau''}) \end{aligned} \tag{27}$$

by induction on n. The induction step $(n \to n+1)$ follows from

$$\begin{aligned} &S[\mathbf{fix}^{(n+1)}(F)/x] = S'[\mathbf{fix}^{(n)}(F)/x]\,, && M[\mathbf{fix}^{(n+1)}(F)/x] = M'[\mathbf{fix}^{(n)}(F)/x]\,, \\ &S[\mathbf{fix}^{(n+2)}(F)/x] = S'[\mathbf{fix}^{(n+1)}(F)/x]\,, \text{ and } && M[\mathbf{fix}^{(n+2)}(F)/x] = M'[\mathbf{fix}^{(n+1)}(F)/x]\,, \end{aligned}$$

where $S' = S[(F\ x)/x]$ and $M' = M[(F\ x)/x]$. For the induction base $(n = 0)$, we prove that for every $t \in \mathbb{N}$, $\tau' \in \mathit{Typ}$, evaluation frame stack S with $x :: \tau \vdash S :: \tau' \multimap \tau''\text{-}\mathit{list}$, and term M with $x :: \tau \vdash M :: \tau'$:

$$\begin{aligned} &(S[\mathbf{fix}^{(0)}(F)/x], M[\mathbf{fix}^{(0)}(F)/x]) \rightarrowtail^t (Id, \mathbf{nil}_{\tau''}) \\ \Rightarrow\ &(S[\mathbf{fix}^{(1)}(F)/x], M[\mathbf{fix}^{(1)}(F)/x]) \rightarrowtail^t (Id, \mathbf{nil}_{\tau''}) \end{aligned}$$

by induction on t. The induction base $(t = 0)$ is straightforward, using that $(S[\mathbf{fix}^{(0)}(F)/x], M[\mathbf{fix}^{(0)}(F)/x]) = (Id, \mathbf{nil}_{\tau''})$ implies $S = Id$ and $M = \mathbf{nil}_{\tau''}$. For the induction step $(t \to t+1)$, we proceed by case distinction on the first transition in

$$(S[\mathbf{fix}^{(0)}(F)/x], M[\mathbf{fix}^{(0)}(F)/x]) \rightarrowtail (S', M') \rightarrowtail^t (Id, \mathbf{nil}_{\tau''})$$

as follows:

Case a: $M[\mathbf{fix}^{(0)}(F)/x] = E\{M'\}$ and $S' = S[\mathbf{fix}^{(0)}(F)/x] \circ E$, where E is an evaluation frame and $M' \notin \mathit{Value}$. Then, by the definitions of $\mathbf{fix}^{(0)}(F)$ and evaluation frames, clearly $M = E'\{N\}$ for some evaluation frame E' and term N with $E = E'[\mathbf{fix}^{(0)}(F)/x]$ and $M' = N[\mathbf{fix}^{(0)}(F)/x]$. Consequently, $M[\mathbf{fix}^{(1)}(F)/x] = (E'[\mathbf{fix}^{(1)}(F)/x])\{N[\mathbf{fix}^{(1)}(F)/x]\}$. Since $M' = N[\mathbf{fix}^{(0)}(F)/x] \notin \mathit{Value}$, by (26) also $N[\mathbf{fix}^{(1)}(F)/x] \notin \mathit{Value}$ holds. Thus, we have:

$$\begin{aligned}&(S[\mathbf{fix}^{(1)}(F)/x], M[\mathbf{fix}^{(1)}(F)/x])\\ \rightarrowtail\ &(S[\mathbf{fix}^{(1)}(F)/x] \circ (E'[\mathbf{fix}^{(1)}(F)/x]), N[\mathbf{fix}^{(1)}(F)/x]).\end{aligned}$$

Together with $(S', M') \rightarrowtail^t (\mathit{Id}, \mathbf{nil}_{\tau''})$ and the induction hypothesis for t, this implies the induction claim $(S[\mathbf{fix}^{(1)}(F)/x], M[\mathbf{fix}^{(1)}(F)/x]) \rightarrowtail^{t+1} (\mathit{Id}, \mathbf{nil}_{\tau''})$.

Case b: $S[\mathbf{fix}^{(0)}(F)/x] = S' \circ E$ and $M' = E\{M[\mathbf{fix}^{(0)}(F)/x]\}$, where E is an evaluation frame and $M[\mathbf{fix}^{(0)}(F)/x] \in \mathit{Value}$. Then clearly $S = S'' \circ E'$ for some evaluation frame stack S'' and evaluation frame E' with $S' = S''[\mathbf{fix}^{(0)}(F)/x]$ and $E = E'[\mathbf{fix}^{(0)}(F)/x]$. Since $M[\mathbf{fix}^{(0)}(F)/x] \in \mathit{Value}$, by (26) also $M[\mathbf{fix}^{(1)}(F)/x] \in \mathit{Value}$ holds. Thus, we have:

$$\begin{aligned}&(S[\mathbf{fix}^{(1)}(F)/x], M[\mathbf{fix}^{(1)}(F)/x])\\ \rightarrowtail\ &(S''[\mathbf{fix}^{(1)}(F)/x], (E'[\mathbf{fix}^{(1)}(F)/x])\{M[\mathbf{fix}^{(1)}(F)/x]\}).\end{aligned}$$

Together with $(S', M') \rightarrowtail^t (\mathit{Id}, \mathbf{nil}_{\tau''})$ and the induction hypothesis for t, this implies the induction claim $(S[\mathbf{fix}^{(1)}(F)/x], M[\mathbf{fix}^{(1)}(F)/x]) \rightarrowtail^{t+1} (\mathit{Id}, \mathbf{nil}_{\tau''})$.

Case c: $S' = S[\mathbf{fix}^{(0)}(F)/x]$ and $M[\mathbf{fix}^{(0)}(F)/x] \rightsquigarrow M'$. Then we have $M \neq x$, because otherwise $M' = ((\lambda x :: \tau.x)\ \mathbf{fix}(\lambda x :: \tau.x))$ by the definitions of $\mathbf{fix}^{(0)}(F)$ and $\rightsquigarrow$, which would be in contradiction to $(S', M') \rightarrowtail^t (\mathit{Id}, \mathbf{nil}_{\tau''})$ by Observations 3.6 and 3.7 and Corollary 4.6. From $M \neq x$ and $M[\mathbf{fix}^{(0)}(F)/x] \rightsquigarrow M'$ follows, by the definitions of $\mathbf{fix}^{(0)}(F)$ and $\rightsquigarrow$, the existence of a term N such that $M' = N[\mathbf{fix}^{(0)}(F)/x]$ and $M[\mathbf{fix}^{(1)}(F)/x] \rightsquigarrow N[\mathbf{fix}^{(1)}(F)/x]$, and consequently:

$$(S[\mathbf{fix}^{(1)}(F)/x], M[\mathbf{fix}^{(1)}(F)/x]) \rightarrowtail (S[\mathbf{fix}^{(1)}(F)/x], N[\mathbf{fix}^{(1)}(F)/x]).$$

Together with $(S', M') \rightarrowtail^t (\mathit{Id}, \mathbf{nil}_{\tau''})$ and the induction hypothesis for t, this implies the induction claim $(S[\mathbf{fix}^{(1)}(F)/x], M[\mathbf{fix}^{(1)}(F)/x]) \rightarrowtail^{t+1} (\mathit{Id}, \mathbf{nil}_{\tau''})$.

Now, using (27), we prove that for every $t \in \mathbb{N}$, $\tau' \in \mathit{Typ}$, evaluation frame stack S with $x :: \tau \vdash S :: \tau' \multimap \tau''\text{-}\mathit{list}$, and term M with $x :: \tau \vdash M :: \tau'$:

$$\begin{aligned}&(S[\mathbf{fix}(F)/x], M[\mathbf{fix}(F)/x]) \rightarrowtail^t (\mathit{Id}, \mathbf{nil}_{\tau''})\\ \Rightarrow\ &\exists n \in \mathbb{N}.\ (S[\mathbf{fix}^{(n)}(F)/x], M[\mathbf{fix}^{(n)}(F)/x]) \rightarrowtail^* (\mathit{Id}, \mathbf{nil}_{\tau''})\end{aligned} \tag{28}$$

by induction on t. The induction base ($t = 0$) is straightforward, using that $(S[\mathbf{fix}(F)/x], M[\mathbf{fix}(F)/x]) = (\mathit{Id}, \mathbf{nil}_{\tau''})$ implies $S = \mathit{Id}$ and $M = \mathbf{nil}_{\tau''}$. For the induction step ($t \to t + 1$), we proceed by case distinction on the first transition in

$$(S[\mathbf{fix}(F)/x], M[\mathbf{fix}(F)/x]) \rightarrowtail (S', M') \rightarrowtail^t (\mathit{Id}, \mathbf{nil}_{\tau''})$$

as follows:

Case a: $M[\mathbf{fix}(F)/x] = E\{M'\}$ and $S' = S[\mathbf{fix}(F)/x] \circ E$, where E is an evaluation frame and $M' \notin \mathit{Value}$. Then, by the definition of evaluation frames, clearly $M = E'\{N\}$ for some evaluation frame E' and term N with $E = E'[\mathbf{fix}(F)/x]$ and $M' = N[\mathbf{fix}(F)/x]$. By $(S', M') \rightarrowtail^t (\mathit{Id}, \mathbf{nil}_{\tau''})$ and the induction hypothesis for t, we then know that there exists an $n \in \mathbb{N}$ such that:

$$(S[\mathbf{fix}^{(n)}(F)/x] \circ (E'[\mathbf{fix}^{(n)}(F)/x]), N[\mathbf{fix}^{(n)}(F)/x]) \rightarrowtail^* (\mathit{Id}, \mathbf{nil}_{\tau''}).$$

To establish the induction claim, it thus suffices to show for any such n that:

$$\begin{aligned} &(S[\mathbf{fix}^{(n)}(F)/x], M[\mathbf{fix}^{(n)}(F)/x]) \\ \rightarrowtail\ &(S[\mathbf{fix}^{(n)}(F)/x] \circ (E'[\mathbf{fix}^{(n)}(F)/x]), N[\mathbf{fix}^{(n)}(F)/x]). \end{aligned}$$

But this follows from $M[\mathbf{fix}^{(n)}(F)/x] = (E'[\mathbf{fix}^{(n)}(F)/x])\{N[\mathbf{fix}^{(n)}(F)/x]\}$ and $N[\mathbf{fix}^{(n)}(F)/x] \notin \mathit{Value}$, where the latter is established by (26) from $M' = N[\mathbf{fix}(F)/x] \notin \mathit{Value}$.

Case b: $S[\mathbf{fix}(F)/x] = S' \circ E$ and $M' = E\{M[\mathbf{fix}(F)/x]\}$, where E is an evaluation frame and $M[\mathbf{fix}(F)/x] \in \mathit{Value}$. Then clearly $S = S'' \circ E'$ for some evaluation frame stack S'' and evaluation frame E' with $S' = S''[\mathbf{fix}(F)/x]$ and $E = E'[\mathbf{fix}(F)/x]$. By $(S', M') \rightarrowtail^t (\mathit{Id}, \mathbf{nil}_{\tau''})$ and the induction hypothesis for t, we then know that there exists an $n \in \mathbb{N}$ such that:

$$(S''[\mathbf{fix}^{(n)}(F)/x], (E'[\mathbf{fix}^{(n)}(F)/x])\{M[\mathbf{fix}^{(n)}(F)/x]\}) \rightarrowtail^* (\mathit{Id}, \mathbf{nil}_{\tau''}).$$

To establish the induction claim, it thus suffices to show for any such n that:

$$\begin{aligned} &(S[\mathbf{fix}^{(n)}(F)/x], M[\mathbf{fix}^{(n)}(F)/x]) \\ \rightarrowtail\ &(S''[\mathbf{fix}^{(n)}(F)/x], (E'[\mathbf{fix}^{(n)}(F)/x])\{M[\mathbf{fix}^{(n)}(F)/x]\}). \end{aligned}$$

But this follows from $S[\mathbf{fix}^{(n)}(F)/x] = S''[\mathbf{fix}^{(n)}(F)/x] \circ (E'[\mathbf{fix}^{(n)}(F)/x])$ and $M[\mathbf{fix}^{(n)}(F)/x] \in \mathit{Value}$, where the latter is established by (26) from $M[\mathbf{fix}(F)/x] \in \mathit{Value}$.

Case c: $S' = S[\mathbf{fix}(F)/x]$ and $M[\mathbf{fix}(F)/x] \leadsto M'$. Then we perform a further case distinction on M being equal to x or not.

Case c.1: $M = x$. Then we have $M' = F\ \mathbf{fix}(F)$ by the definition of $\leadsto$. By $(S', M') \rightarrowtail^t (\mathit{Id}, \mathbf{nil}_{\tau''})$ and the induction hypothesis for t, we then know that there exists an $n \in \mathbb{N}$ such that:

$$(S[\mathbf{fix}^{(n)}(F)/x], F\ \mathbf{fix}^{(n)}(F)) \rightarrowtail^* (\mathit{Id}, \mathbf{nil}_{\tau''}).$$

By (27) and the definition of $\mathbf{fix}^{(n+1)}(F)$, we get that for any such n:

$$(S[\mathbf{fix}^{(n+1)}(F)/x], \mathbf{fix}^{(n+1)}(F)) \rightarrowtail^* (\mathit{Id}, \mathbf{nil}_{\tau''})\,,$$

which suffices to establish the induction claim.

Case c.2: $M \neq x$. Then from $M[\mathbf{fix}(F)/x] \rightsquigarrow M'$ follows, by the definition of $\rightsquigarrow$, the existence of a term N such that $M' = N[\mathbf{fix}(F)/x]$ and $M[\mathbf{fix}^{(n)}(F)/x] \rightsquigarrow N[\mathbf{fix}^{(n)}(F)/x]$ for every $n \in \mathbb{N}$. By $(S', M') \rightarrowtail^t (Id, \mathbf{nil}_{\tau''})$ and the induction hypothesis for t, we also know that given any such N, there exists an $n \in \mathbb{N}$ such that:

$$(S[\mathbf{fix}^{(n)}(F)/x], N[\mathbf{fix}^{(n)}(F)/x]) \rightarrowtail^* (Id, \mathbf{nil}_{\tau''}).$$

To establish the induction claim, it thus suffices to show for any such n that:

$$(S[\mathbf{fix}^{(n)}(F)/x], M[\mathbf{fix}^{(n)}(F)/x]) \rightarrowtail (S[\mathbf{fix}^{(n)}(F)/x], N[\mathbf{fix}^{(n)}(F)/x]).$$

But this follows from $M[\mathbf{fix}^{(n)}(F)/x] \rightsquigarrow N[\mathbf{fix}^{(n)}(F)/x]$.

Again using (27), we now prove that for every $n, t \in \mathbb{N}$, $\tau' \in \mathit{Typ}$, evaluation frame stack S with $x :: \tau \vdash S :: \tau' \multimap \tau''\text{-}\mathit{list}$, and term M with $x :: \tau \vdash M :: \tau'$:

$$\begin{array}{rl} & (S[\mathbf{fix}^{(n)}(F)/x], M[\mathbf{fix}^{(n)}(F)/x]) \rightarrowtail^t (Id, \mathbf{nil}_{\tau''}) \\ \Rightarrow & (S[\mathbf{fix}(F)/x], M[\mathbf{fix}(F)/x]) \rightarrowtail^* (Id, \mathbf{nil}_{\tau''}) \end{array} \qquad (29)$$

by induction on t. The induction base ($t = 0$) is straightforward, using that $(S[\mathbf{fix}^{(n)}(F)/x], M[\mathbf{fix}^{(n)}(F)/x]) = (Id, \mathbf{nil}_{\tau''})$ implies $S = Id$ and $M = \mathbf{nil}_{\tau''}$. For the induction step ($t \to t+1$), we proceed by case distinction on the first transition in

$$(S[\mathbf{fix}^{(n)}(F)/x], M[\mathbf{fix}^{(n)}(F)/x]) \rightarrowtail (S', M') \rightarrowtail^t (Id, \mathbf{nil}_{\tau''})$$

as follows:

Case a: $M[\mathbf{fix}^{(n)}(F)/x] = E\{M'\}$ and $S' = S[\mathbf{fix}^{(n)}(F)/x] \circ E$, where E is an evaluation frame and $M' \notin \mathit{Value}$. Then, by the definitions of $\mathbf{fix}^{(n)}(F)$ and evaluation frames, there are two cases to consider:

Case a.1: $M = E'\{N\}$ for some evaluation frame E' and term N with $E = E'[\mathbf{fix}^{(n)}(F)/x]$ and $M' = N[\mathbf{fix}^{(n)}(F)/x]$. Then $M[\mathbf{fix}(F)/x] = (E'[\mathbf{fix}(F)/x])\{N[\mathbf{fix}(F)/x]\}$. Since $M' = N[\mathbf{fix}^{(n)}(F)/x] \notin \mathit{Value}$, by (26) also $N[\mathbf{fix}(F)/x] \notin \mathit{Value}$ holds. Thus, we have:

$$(S[\mathbf{fix}(F)/x], M[\mathbf{fix}(F)/x]) \rightarrowtail (S[\mathbf{fix}(F)/x] \circ (E'[\mathbf{fix}(F)/x]), N[\mathbf{fix}(F)/x]).$$

Together with $(S', M') \rightarrowtail^t (Id, \mathbf{nil}_{\tau''})$ and the induction hypothesis for t, this implies the induction claim $(S[\mathbf{fix}(F)/x], M[\mathbf{fix}(F)/x]) \rightarrowtail^* (Id, \mathbf{nil}_{\tau''})$.

Case a.2: $M = x$, $n > 0$, $E = (-\ \mathbf{fix}^{(n-1)}(F))$, and $M' = F$. Then by $(S', M') \rightarrowtail^t (Id, \mathbf{nil}_{\tau''})$ and (27) we know that:

$$(S[\mathbf{fix}^{(n)}(F)/x] \circ (-\ \mathbf{fix}^{(n)}(F)), F) \rightarrowtail^t (Id, \mathbf{nil}_{\tau''}).$$

The induction claim follows from this by the induction hypothesis for t and

$$\begin{aligned} (S[\mathbf{fix}(F)/x], \mathbf{fix}(F)) &\rightarrowtail (S[\mathbf{fix}(F)/x], F\ \mathbf{fix}(F)) \\ &\rightarrowtail (S[\mathbf{fix}(F)/x] \circ (-\ \mathbf{fix}(F)), F)\,, \end{aligned}$$

where the second transition is valid due to $M' = F \notin \mathit{Value}$.

Case b: $S[\mathbf{fix}^{(n)}(F)/x] = S' \circ E$ and $M' = E\{M[\mathbf{fix}^{(n)}(F)/x]\}$, where E is an evaluation frame and $M[\mathbf{fix}^{(n)}(F)/x] \in \mathit{Value}$. Then clearly $S = S'' \circ E'$ for some evaluation frame stack S'' and evaluation frame E' with $S' = S''[\mathbf{fix}^{(n)}(F)/x]$ and $E = E'[\mathbf{fix}^{(n)}(F)/x]$. Since $M[\mathbf{fix}^{(n)}(F)/x] \in \mathit{Value}$, by (26) also $M[\mathbf{fix}(F)/x] \in \mathit{Value}$ holds. Thus, we have:

$$(S[\mathbf{fix}(F)/x], M[\mathbf{fix}(F)/x]) \rightarrowtail (S''[\mathbf{fix}(F)/x], (E'[\mathbf{fix}(F)/x])\{M[\mathbf{fix}(F)/x]\}).$$

Together with $(S', M') \rightarrowtail^t (\mathit{Id}, \mathbf{nil}_{\tau''})$ and the induction hypothesis for t, this implies the induction claim $(S[\mathbf{fix}(F)/x], M[\mathbf{fix}(F)/x]) \rightarrowtail^* (\mathit{Id}, \mathbf{nil}_{\tau''})$.

Case c: $S' = S[\mathbf{fix}^{(n)}(F)/x]$ and $M[\mathbf{fix}^{(n)}(F)/x] \rightsquigarrow M'$. Then we perform a further case distinction on M being equal to x or not.

Case c.1: $M = x$. Then we have $n > 0$, because otherwise $M' = ((\lambda x :: \tau.x)\ \mathbf{fix}(\lambda x :: \tau.x))$ by the definitions of $\mathbf{fix}^{(0)}(F)$ and $\rightsquigarrow$, which would contradict $(S', M') \rightarrowtail^t (\mathit{Id}, \mathbf{nil}_{\tau''})$ by Observations 3.6 and 3.7 and Corollary 4.6. From $M = x$, $n > 0$, and $M[\mathbf{fix}^{(n)}(F)/x] \rightsquigarrow M'$ follows, by the definitions of $\mathbf{fix}^{(n)}(F)$ and $\rightsquigarrow$, that $F = (\lambda x :: \tau.F')$ and $M' = F'[\mathbf{fix}^{(n-1)}(F)/x]$ for some term F' with $x :: \tau \vdash F' :: \tau$.[7] Then by $(S', M') \rightarrowtail^t (\mathit{Id}, \mathbf{nil}_{\tau''})$ and (27) we know that:

$$(S[\mathbf{fix}^{(n)}(F)/x], F'[\mathbf{fix}^{(n)}(F)/x]) \rightarrowtail^t (\mathit{Id}, \mathbf{nil}_{\tau''}).$$

The induction claim follows from this by the induction hypothesis for t and

$$\begin{aligned}(S[\mathbf{fix}(F)/x], \mathbf{fix}(F)) &\rightarrowtail (S[\mathbf{fix}(F)/x], F\ \mathbf{fix}(F)) \\ &\rightarrowtail (S[\mathbf{fix}(F)/x], F'[\mathbf{fix}(F)/x]).\end{aligned}$$

Case c.2: $M \neq x$. Then from $M[\mathbf{fix}^{(n)}(F)/x] \rightsquigarrow M'$ follows, by the definitions of $\mathbf{fix}^{(n)}(F)$ and $\rightsquigarrow$, the existence of a term N such that $M' = N[\mathbf{fix}^{(n)}(F)/x]$ and $M[\mathbf{fix}(F)/x] \rightsquigarrow N[\mathbf{fix}(F)/x]$, and consequently:

$$(S[\mathbf{fix}(F)/x], M[\mathbf{fix}(F)/x]) \rightarrowtail (S[\mathbf{fix}(F)/x], N[\mathbf{fix}(F)/x]).$$

Together with $(S', M') \rightarrowtail^t (\mathit{Id}, \mathbf{nil}_{\tau''})$ and the induction hypothesis for t, this implies the induction claim $(S[\mathbf{fix}(F)/x], M[\mathbf{fix}(F)/x]) \rightarrowtail^* (\mathit{Id}, \mathbf{nil}_{\tau''})$.

Finally, the lemma follows by combining the definition of $\top$ with (28) and (29) in the special case that $M = x$ and x is not a free variable of S. ■

[7] Note that we are free to assume here that the term variable bound in F is x, since terms are identified up to renaming of bound variables. Also note that we know the type of x in this binding, given that $F \in \mathit{Term}(\tau \to \tau)$.

Proof of Lemma 5.6: We have to show that $(G, G') \in (\forall R)^{\top\top}$ implies $(G, G') \in (\forall R)$, i.e.,

$$G\Downarrow \Leftrightarrow G'\Downarrow$$

and

$$\forall \tau_2, \tau_2' \in Typ, r \in Rel^{\Downarrow}(\tau_2, \tau_2').\ (G_{\tau_2}, G'_{\tau_2'}) \in R_{\tau_2,\tau_2'}(r).$$

The former holds because $(\forall R)^{\top\top}$ is convergence-reflecting by Lemma 4.18. The latter follows from $\top\top$-closedness of each $R_{\tau_2,\tau_2'}(r)$ if, for every $\tau_2, \tau_2' \in Typ$ and $r \in Rel^{\Downarrow}(\tau_2, \tau_2')$, we can show that $(G_{\tau_2}, G'_{\tau_2'}) \in (R_{\tau_2,\tau_2'}(r))^{\top\top}$. To do so, we reason for every $(S, S') \in (R_{\tau_2,\tau_2'}(r))^{\top}$ as follows:

$$\begin{array}{rcll} S \top G_{\tau_2} & \Leftrightarrow & S \circ -_{\tau_2} \top G & \text{by Observation 4.3(1)} \\ & \Leftrightarrow & S' \circ -_{\tau_2'} \top G' & \\ & \Leftrightarrow & S' \top G'_{\tau_2'} & \text{by Observation 4.3(1).} \end{array}$$

Here the second equivalence holds by $(G, G') \in (\forall R)^{\top\top}$ and $(S \circ -_{\tau_2}, S' \circ -_{\tau_2'}) \in (\forall R)^{\top}$. The latter is established by reasoning for every $(N, N') \in (\forall R)$ as follows:

$$\begin{array}{rcll} S \circ -_{\tau_2} \top N & \Leftrightarrow & S \top N_{\tau_2} & \text{by Observation 4.3(1)} \\ & \Leftrightarrow & S' \top N'_{\tau_2'} & \\ & \Leftrightarrow & S' \circ -_{\tau_2'} \top N' & \text{by Observation 4.3(1).} \end{array}$$

Here the second equivalence holds by $(S, S') \in (R_{\tau_2,\tau_2'}(r))^{\top}$ and $(N_{\tau_2}, N'_{\tau_2'}) \in R_{\tau_2,\tau_2'}(r)$. The latter follows from $(N, N') \in (\forall R)$ by the definition of $\forall R$. ■

Lemma A.1 (essentially Lemma 4.10(ii) in [Pit00]). Let $\tau_1, \tau_1', \tau_2, \tau_2' \in Typ$, $r_1 \in Rel(\tau_1, \tau_1')$, and $r_2 \in Rel(\tau_2, \tau_2')$. Let h and t be term variables and M_2 and M_2' be terms such that $h :: \tau_1, t :: \tau_1\text{-}list \vdash M_2 :: \tau_2$ and $h :: \tau_1', t :: \tau_1'\text{-}list \vdash M_2' :: \tau_2'$. If

$$\forall (H, H') \in r_1, (T, T') \in lift(r_1).\ (M_2[H/h, T/t], M_2'[H'/h, T'/t]) \in r_2 \tag{30}$$

and r_2 is $\top\top$-closed, then for every $(L, L') \in lift(r_1)$ and $(M_1, M_1') \in r_2$:

$$(\mathbf{case}\ L\ \mathbf{of}\ \{\mathbf{nil} \Rightarrow M_1;\ h : t \Rightarrow M_2\}, \mathbf{case}\ L'\ \mathbf{of}\ \{\mathbf{nil} \Rightarrow M_1';\ h : t \Rightarrow M_2'\}) \in r_2.$$

Proof: The desired $(\mathbf{case}\ L\ \mathbf{of}\ match, \mathbf{case}\ L'\ \mathbf{of}\ match') \in r_2$, with $match$ abbreviating $\{\mathbf{nil} \Rightarrow M_1;\ h : t \Rightarrow M_2\}$ and $match'$ abbreviating $\{\mathbf{nil} \Rightarrow M_1';\ h : t \Rightarrow M_2'\}$, follows from $\top\top$-closedness of r_2 and the following reasoning for every $(S, S') \in r_2^{\top}$:

$$\begin{array}{rcll} S \top \mathbf{case}\ L\ \mathbf{of}\ match & \Leftrightarrow & S \circ (\mathbf{case}\ -\ \mathbf{of}\ match) \top L & \text{by Observation 4.3(1)} \\ & \Leftrightarrow & S' \circ (\mathbf{case}\ -\ \mathbf{of}\ match') \top L' & \\ & \Leftrightarrow & S' \top \mathbf{case}\ L'\ \mathbf{of}\ match' & \text{by Observation 4.3(1).} \end{array}$$

Here the second equivalence holds by $(L, L') \in lift(r_1)$ and $(S \circ (\mathbf{case}\ -\ \mathbf{of}\ match), S' \circ (\mathbf{case}\ -\ \mathbf{of}\ match')) \in lift(r_1)^{\top}$, where the latter is established as follows.

By (3) and (7), we have $lift(r_1)^{\top} = (1 + (r_1 \times lift(r_1)))^{\top}$. So it suffices to show $S \circ (\mathbf{case}\ -\ \mathbf{of}\ match) \top N \Leftrightarrow S' \circ (\mathbf{case}\ -\ \mathbf{of}\ match') \top N'$ for every $(N, N') \in (1 + (r_1 \times lift(r_1)))$. We do so by case distinction.

Case a: $N = \mathbf{nil}_{\tau_1}$ and $N' = \mathbf{nil}_{\tau_1'}$. Then:

$$
\begin{array}{lll}
 & S \circ (\mathbf{case} - \mathbf{of}\ \mathit{match}) \top \mathbf{nil}_{\tau_1} & \\
\Leftrightarrow & S \top M_1 & \text{by Observation 4.3} \\
\Leftrightarrow & S' \top M_1' & \text{by } (S, S') \in r_2^{\top} \text{ and } (M_1, M_1') \in r_2 \\
\Leftrightarrow & S' \circ (\mathbf{case} - \mathbf{of}\ \mathit{match}') \top \mathbf{nil}_{\tau_1'} & \text{by Observation 4.3.}
\end{array}
$$

Case b: $N = H : T$ and $N' = H' : T'$ for some $(H, H') \in r_1$ and $(T, T') \in \mathit{lift}(r_1)$. Then:

$$
\begin{array}{lll}
 & S \circ (\mathbf{case} - \mathbf{of}\ \mathit{match}) \top H : T & \\
\Leftrightarrow & S \top M_2[H/h, T/t] & \text{by Observation 4.3} \\
\Leftrightarrow & S' \top M_2'[H'/h, T'/t] & \text{by } (S, S') \in r_2^{\top} \text{ and (30)} \\
\Leftrightarrow & S' \circ (\mathbf{case} - \mathbf{of}\ \mathit{match}') \top H' : T' & \text{by Observation 4.3.}
\end{array}
$$

■

Proof of Theorem 6.4, continued: The rule

$$\frac{\Gamma, x :: \tau \vdash M \ \Delta \ M' :: \tau'}{\Gamma \vdash (\lambda x :: \tau.M) \ \Delta \ (\lambda x :: \tau.M') :: \tau \to \tau'}$$

follows from the definition of Δ, clause (9) in Definition 5.4, and Lemmas 5.7 and 6.1. The rule

$$\frac{\Gamma \vdash F \ \Delta \ F' :: \tau \to \tau' \qquad \Gamma \vdash A \ \Delta \ A' :: \tau}{\Gamma \vdash (F\ A) \ \Delta \ (F'\ A') :: \tau'}$$

follows from the definition of Δ, clause (9), and Definition 5.1. The rule

$$\frac{\alpha, \Gamma \vdash M \ \Delta \ M' :: \tau}{\Gamma \vdash \Lambda\alpha.M \ \Delta \ \Lambda\alpha.M' :: \forall\alpha.\tau}$$

follows from the definition of Δ, clause (10), and Lemmas 4.18, 5.7, and 6.2. The rule

$$\frac{\Gamma \vdash G \ \Delta \ G' :: \forall\alpha.\tau}{\Gamma \vdash G_{\tau'} \ \Delta \ G'_{\tau'} :: \tau[\tau'/\alpha]}$$

follows from the definition of Δ, clause (10), Definition 5.2, Lemma 5.7, and Observation 5.9. The axiom $\Gamma \vdash \mathbf{nil}_\tau \ \Delta \ \mathbf{nil}_\tau :: \tau\text{-}\mathit{list}$ and the rule

$$\frac{\Gamma \vdash H \ \Delta \ H' :: \tau \qquad \Gamma \vdash T \ \Delta \ T' :: \tau\text{-}\mathit{list}}{\Gamma \vdash (H : T) \ \Delta \ (H' : T') :: \tau\text{-}\mathit{list}}$$

follow from the definition of Δ, clause (11), and the fact that for every $r \in \mathit{Rel}$, $(1 + (r \times \mathit{lift}(r))) \subseteq \mathit{lift}(r)$ by (2) and (7). The rule

$$\frac{\Gamma \vdash L \ \Delta \ L' :: \tau\text{-}\mathit{list} \qquad \Gamma \vdash M_1 \ \Delta \ M_1' :: \tau' \qquad \Gamma, h :: \tau, t :: \tau\text{-}\mathit{list} \vdash M_2 \ \Delta \ M_2' :: \tau'}{\begin{array}{c}\Gamma \vdash (\mathbf{case}\ L\ \mathbf{of}\ \{\mathbf{nil} \Rightarrow M_1;\ h : t \Rightarrow M_2\}) \\ \Delta \ (\mathbf{case}\ L'\ \mathbf{of}\ \{\mathbf{nil} \Rightarrow M_1';\ h : t \Rightarrow M_2'\}) :: \tau'\end{array}}$$

follows from the definition of Δ, clause (11), and Lemmas 5.7 and A.1. The rule

$$\frac{\Gamma \vdash F \ \Delta \ F' :: \tau \to \tau}{\Gamma \vdash \mathbf{fix}(F) \ \Delta \ \mathbf{fix}(F') :: \tau}$$

follows from the definition of Δ, clause (9), Definition 5.1, and Lemmas 4.14 and 5.7. This completes the proof. ■