Math. Struct. in Comp. Science (2000), *vol.* 10, *pp.* 321–359. Printed in the United Kingdom

Parametric Polymorphism and Operational Equivalence

ANDREW M. PITTS

Cambridge University Computer Laboratory,
Pembroke Street, Cambridge, CB2 3QG, UK

Received 18 December 1998; revised 28 September 1999

Studies of the mathematical properties of impredicative polymorphic types have for the most part focused on the *polymorphic lambda calculus* of Girard-Reynolds, which is a calculus of *total* polymorphic functions. This paper considers polymorphic types from a functional programming perspective, where the partialness arising from the presence of fixpoint recursion complicates the nature of potentially infinite ('lazy') data types. An approach to Reynolds' notion of *relational parametricity* is developed which works directly on the syntax of a programming language, using a novel closure operator to relate operational behaviour to parametricity properties of types. Working with an extension of Plotkin's PCF with $\forall$-types, lazy lists and existential types, we show by example how the resulting logical relation can be used to prove properties of polymorphic types up to operational equivalence.

1. Introduction

> *'It turns out that virtually any basic type of interest can be encoded within F_2* [polymorphic lambda calculus]*. Similarly, product types, sum types, existential types, and some recursive types, can be encoded within F_2: polymorphism has an amazing expressive power.'*
>
> Cardelli (1997, page 2225)

It is a widely held view—typified by the above quotation—that the *polymorphic lambda calculus* (PLC) of Girard (1972) and Reynolds (1974) plays a foundational role for the statics, that is, the type systems of functional programming languages analogous to the one played by the untyped lambda calculus for the dynamics of such languages. The technical justification for this view rests on the encoding of a wide class of data type constructions as PLC types: see for example (Böhm and Berarducci 1985), (Reynolds and Plotkin 1993) and (Girard 1989, Chapter 11). However, these results cannot just be applied 'off the shelf' to deduce properties of functional programming languages equipped with polymorphic types. This is because PLC is a theory of *total* polymorphic functions—a consequence of the fact that β-reduction of typeable PLC terms is strongly normalising (Girard 1972); whereas functional programming languages typically feature various mechanisms for making general forms of recursive definitions, both at the level of expressions and at the level of types. The first kind of definition entails the presence of 'partial' expressions, that is, ones whose evaluation does not terminate; and then the second kind of definition may throw up types whose values involve partiality in complicated ways, through the use of non-strict constructors. Can such 'lazy' data types be encoded with combinations of function- and $\forall$-types? It is this question which motivates the results presented here.

A specific example may help to bring this issue into sharper focus. Consider the type $numlist$ of lazy lists of numbers in a non-strict functional programming language, such as the Haskell language $\langle$`www.haskell.org`$\rangle$. The canonical forms of this type are the empty list, **nil**, and cons-expressions, $H{:}T$, where the head H (of type num) and the tail T (of type $numlist$ again) are not necessarily in canonical form (and therefore their evaluation may not terminate). Thus expressions of this type can represent finite lists (such as $0 : \mathbf{nil}$), properly infinite lists (such as $\ell = 0 : \ell$), or 'partial' lists (such as $0 : \Omega$, where Ω is a divergent expression of type $numlist$). Suppose now that the language is augmented with $\forall$-types. (We consider why one might want to do so in a moment.) In PLC, the type

$$L(\tau) \stackrel{\text{def}}{=} \forall\,\alpha\,(\alpha \to (\tau \to \alpha \to \alpha) \to \alpha) \quad (\alpha \text{ not free in } \tau)$$

encodes finite τ-lists—in the sense that the closed β-normal forms of $L(\tau)$ are in bijection with finite lists of closed β-normal forms of type τ. But what is the situation in the functional programming language? Can uses of the lazy list type $numlist$ always be replaced by the polymorphic type $L(num)$? More precisely, are $numlist$ and $L(num)$ 'operationally isomorphic', in the sense that there are functions in the language from $numlist$ to $L(num)$ and back again that are mutually inverse up to some reasonable notion of operational equivalence of expressions? Or is $numlist$ operationally isomorphic to some some other polymorphic type, or to no such type?

The reader will not find the answer to such questions in the literature, as far as I know. Partly this is because it is hard to construct denotational models of both impredicative polymorphism and fixpoint recursion. Such models do exist (see (Coquand, Gunter, and Winskel 1987; Coquand, Gunter, and Winskel 1989) for one style of model and (Abadi and Plotkin 1990) for another), but there is not much in the way of useful analysis of the properties of polymorphic types in these models. On the other hand for pure PLC, Reynolds' notion of *relational parametricity* (Reynolds 1983; Ma and Reynolds 1992) turns out to provide a very powerful tool for such an analysis. There are models of PLC supporting a relationally parametric structure (Bainbridge, Freyd, Scedrov, and Scott 1990), and in such models polymorphic encodings of data type constructions have strong properties; indeed they have category-theoretic universal properties characterising the constructions uniquely up to isomorphism (Hasegawa 1991; Hasegawa 1994; Abadi, Cardelli, and Curien 1993; Plotkin and Abadi 1993). Can one extend this relational approach to encompass fixpoint recursion? Unpublished work of Plotkin (1993) indicates one way to do that model-theoretically. Here we show that a relatively simple, syntactic approach is possible.

Should one care? Well, for one thing the results presented here provide a basis for obtaining some 'free theorems' (Wadler 1989) up to operational equivalence (and modulo some restrictions to do with strictness) in languages like ML and Haskell that combine higher order functions, fixpoint recursion and *predicative* polymorphism. However, the power of the relational approach really shows when considering fully impredicative $\forall$-types. Since the type reconstruction problem is undecidable in this case (Wells 1994) and explicit labelling with type information is considered cumbersome, most higher order typed languages meant for human programmers eschew fully impredicative polymorphism. However, it seems that impredicative polymorphism is a useful feature of explicitly typed intermediate languages in compilers (Harper and Lillibridge 1993;

Morrisett, Walker, Crary, and Glew 1998). Also, there is foundational interest in knowing, in the presence of fixpoint recursion, to what extent various kinds of type can be reduced to pure polymorphic types.

As Wadler (1989, Sec. 7) and Plotkin (1993) point out, extending relational parametricity to cope with fixpoint recursion seems to necessitate working not with arbitrary relations, but with ones that are at least *admissible* in the domain-theoretic sense, i.e. that are bottom-relating and closed under taking limits of chains of related elements. In this paper a relational framework for polymorphism and fixpoint recursion is developed which is based upon operational rather than denotational semantics. This allows one to avoid some of the complexities of the domain-theoretic approach. In particular, it turns out that questions of admissibility of relations can be treated implicitly, via an operationally-defined notion of closure. This closure operator allows us to tie operational behaviour to parametricity properties of types and it is the main technical contribution of the paper. We use it to obtain a straightforward and apparently quite powerful method for proving properties of Morris-style contextual equivalence of types and terms involving impredicative polymorphism and fixpoint recursion; and one which is based only upon the syntax and operational semantics of the language. (See (Pitts 1997b; Pitts and Stark 1998) for previous results of this kind.)

The plan of the paper is as follows. In the next section we introduce PolyPCF, an extension of PCF (Plotkin 1977) with lazy lists and $\forall$-types which will serve as the vehicle for examining the issues raised above. We define a notion of *observational congruence* for PolyPCF terms based upon observing convergence of evaluation in all contexts of list-type, but not of function- or $\forall$-type. However, instead of working with a conventional Morris-style definition of contextual equivalence, we define this observational congruence without mentioning PolyPCF contexts explicitly, making use of the 'relational' approach of (Lassen 1998). Not only does this avoid some low-level technicalities with binding in contexts, but it also makes it easier to state the relevant properties of logical relations later on. Section 3 introduces the closure operation on (binary) relations between PolyPCF terms mentioned above. We use it in Section 4 to present our syntactic version of relational parametricity. An action of the PolyPCF types on binary relations between closed PolyPCF terms (of the same closed type) is defined. This gives rise to a certain binary logical relation which is shown to characterise PolyPCF observational congruence (Theorem 4.15). Section 5 shows how the logical relation can be used to deduce extensionality properties of PolyPCF observational congruence. Whereas these 'context lemmas' can be proved in a number of different ways in addition to the one given here, the same does not seem to be true of the results in the next two sections where we really exploit the relational parametricity properties of observational congruence established earlier. In Section 6 we show, amongst other things, that in PolyPCF it is indeed the case that $\alpha\,list$ is observationally isomorphic to the pure polymorphic type $\forall\,\alpha'\,(\alpha' \to (\alpha \to \alpha' \to \alpha') \to \alpha')$; and in Section 7 we show that existential types, with a standard operational semantics, are definable in PolyPCF up to observational congruence. Finally, Section 8 considers some directions in which the results presented here might usefully be extended.

Types	τ	$::=$	α	type variable
		$\mid$	$\tau \to \tau$	function type
		$\mid$	$\forall\, \alpha\, (\tau)$	$\forall$-type
		$\mid$	$\tau\, \mathit{list}$	list type
Terms	M	$::=$	x	variable
		$\mid$	$\lambda\, x : \tau\, (M)$	function abstraction
		$\mid$	$M\, M$	function application
		$\mid$	$\Lambda\, \alpha\, (M)$	type generalisation
		$\mid$	$M\, \tau$	type specialisation
		$\mid$	$\mathbf{fix}(M)$	fixpoint recursion
		$\mid$	$\mathbf{nil}_\tau$	empty list
		$\mid$	$M :: M$	non-empty list
		$\mid$	$\mathbf{case}\, M\, \mathbf{of}\, \{\mathbf{nil} \Rightarrow M \mid x :: x \Rightarrow M\}$	case expression

Notes

(i) α and x range over disjoint countably infinite sets TyVar and Var of *type variables* and *variables* respectively.

(ii) The constructions $\forall\, \alpha\, (-)$, $\lambda\, x : \tau\, (-)$, $\Lambda\, \alpha\, (-)$, and $\mathbf{case}\, M\, \mathbf{of}\, \{\mathbf{nil} \Rightarrow M' \mid x :: x' \Rightarrow (-)\}$ are binders. We will identify types and terms up to renaming of bound variables and bound type variables.

(iii) We write $\mathit{ftv}(e)$ for the finite set of free type variables of an expression e (be it a type or a term) and $\mathit{fv}(M)$ for the finite set of free variables of a term M.

(iv) The result of capture-avoiding substitution of a type τ for all free occurrences of a type variable α in e (a type or a term) will be denoted $e[\tau/\alpha]$. Similarly, $M[M'/x]$ denotes the result of capture-avoiding substitution of a term M' for all free occurrences of the variable x in M.

Fig. 1. Syntax of the PolyPCF language

2. Polymorphic PCF

To explore the issues raised in the Introduction we use a programming language, PolyPCF, combining the Girard-Reynolds polymorphic lambda calculus with that veteran of studies in programming languages, PCF (Plotkin 1977). Recall that PCF is a simply typed, call-by-name lambda calculus equipped with fixpoint recursion and some basic operations on ground types of natural numbers and booleans. To this we add $\forall$-types from the Girard-Reynolds polymorphic lambda calculus and a type constructor for lists. For reasons of parsimony we do without the ground types of natural numbers and booleans, because the role they play in the theory can be taken by the list types. The syntax of PolyPCF types and terms is given in Fig. 1. We are only interested in terms that can be assigned types. The PolyPCF type assignment relation is given by the axioms and rules in Fig. 2, all of which are quite standard.

Notation 2.1. A type τ is *closed* if $\mathit{ftv}(\tau) = \emptyset$; whereas a term M is closed if $\mathit{fv}(M) = \emptyset$, whether or not it also has free type variables. We write Typ for the set of closed PolyPCF types, i.e. those having no free type variables. Given $\tau \in \mathit{Typ}$, we write $\mathit{Term}(\tau)$ for the set of closed PolyPCF terms M for which $\emptyset \vdash M : \tau$ is derivable from the axioms and rules in Fig. 2.

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

$$\frac{\Gamma, x : \tau_1 \vdash M : \tau_2}{\Gamma \vdash \lambda\, x : \tau_1\, (M) : \tau_1 \to \tau_2} \qquad \frac{\Gamma \vdash F : \tau_1 \to \tau_2 \quad \Gamma \vdash A : \tau_1}{\Gamma \vdash F\, A : \tau_2}$$

$$\frac{\Gamma, \alpha \vdash M : \tau}{\Gamma \vdash \Lambda\, \alpha\, (M) : \forall\, \alpha\, (\tau)} \qquad \frac{\Gamma \vdash G : \forall\, \alpha\, (\tau_1)}{\Gamma \vdash G\, \tau_2 : \tau_1[\tau_2/\alpha]}$$

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

$$\Gamma \vdash \mathbf{nil}_\tau : \tau \mathit{list} \qquad \frac{\Gamma \vdash H : \tau \quad \Gamma \vdash T : \tau \mathit{list}}{\Gamma \vdash H :: T : \tau \mathit{list}}$$

$$\frac{\Gamma \vdash L : \tau_1 \mathit{list} \quad \Gamma \vdash M_1 : \tau_2 \quad \Gamma, h : \tau_1, t : \tau_1 \mathit{list} \vdash M_2 : \tau_2}{\Gamma \vdash \mathbf{case}\, L\, \mathbf{of}\, \{\mathbf{nil} \Rightarrow M_1 \mid h :: t \Rightarrow M_2\} : \tau_2}$$

Notes

(i) Typing judgements take the form $\Gamma \vdash M : \tau$ where

— the *typing environment* Γ is a pair A, Δ with A a finite subset of TyVar and Δ a function defined on a finite subset $\mathit{dom}(\Delta)$ of Var and mapping each $x \in \mathit{dom}(\Delta)$ to a type with free type variables in A;

— M is a term with $\mathit{ftv}(M) \subseteq A$ and $\mathit{fv}(M) \subseteq \mathit{dom}(\Delta)$;

— τ is a type with $\mathit{ftv}(\tau) \subseteq A$.

The PolyPCF *type assignment relation* consists of all such judgements inductively defined by the above axioms and rules.

(ii) The notation $\Gamma, x : \tau$ indicates the typing environment obtained from the typing environment $\Gamma = A, \Delta$ by properly extending the function Δ by mapping $x \notin \mathit{dom}(\Delta)$ to τ. Similarly, Γ, α is the typing environment obtained by properly extending A with an $\alpha \notin A$.

(iii) The explicit type information included in the syntax of function abstractions and empty lists ensures that, given Γ and M, there is at most one τ for which $\Gamma \vdash M : \tau$ holds.

Fig. 2. PolyPCF type assignment relation

We give the operational semantics of PolyPCF in terms of an inductively defined relation of *evaluation*. It takes the form $M \Downarrow V$, where M and V are closed terms of the same closed type (i.e. $M, V \in \mathit{Term}(\tau)$ for some $\tau \in \mathit{Typ}$) and where V is a *value*:

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

The evaluation relation is inductively defined by the axioms and rules in Fig. 3. Note that function application is given a call-by-name semantics and that the evaluation rule for type specialisations, $G\, \tau$, is dictated by our choice of values at $\forall$-types—we choose not to evaluate 'under the Λ'. Evaluation is deterministic: given M, there is at most one V for which $M \Downarrow V$ holds; and of course the rule for **fix** entails that there may be no such V.

$$V \Downarrow V \quad (V \text{ a value})$$

$$\frac{F \Downarrow \lambda\, x : \tau\,(M) \quad M[A/x] \Downarrow V}{F\, A \Downarrow V} \qquad \frac{G \Downarrow \Lambda\, \alpha\,(M) \quad M[\tau/\alpha] \Downarrow V}{G\, \tau \Downarrow V}$$

$$\frac{F\, \mathbf{fix}(F) \Downarrow V}{\mathbf{fix}(F) \Downarrow V}$$

$$\frac{L \Downarrow \mathbf{nil}_\tau \quad M_1 \Downarrow V}{\mathbf{case}\, L\, \mathbf{of}\, \{\mathbf{nil} \Rightarrow M_1 \mid h :: t \Rightarrow M_2\} \Downarrow V} \qquad \frac{L \Downarrow H :: T \quad M_2[H/h, T/t] \Downarrow V}{\mathbf{case}\, L\, \mathbf{of}\, \{\mathbf{nil} \Rightarrow M_1 \mid h :: t \Rightarrow M_2\} \Downarrow V}$$

Fig. 3. PolyPCF evaluation relation

Next we define a suitable notion of program equivalence for PolyPCF. Recall that two terms of a programming language are regarded as *contextually equivalent* if, roughly speaking, they are interchangeable in any program without affecting the observable behaviour of the program upon execution. Of course, to make this a precise notion one has to choose what constitutes an executable program and what behaviour should be observable. For PCF, Plotkin (Plotkin 1977) chooses 'program' to mean 'closed term of ground type' and the observable behaviour of such a program to be the constant (integer or boolean) to which it evaluates, if any. Since we have replaced ground types with list types, here we take a program to be a closed term of list type and we observe whether or not it evaluates to **nil**. (In fact, just observing termination of evaluation at list types leads to the same notion of contextual equivalence. Another reasonable choice would be to observe termination of evaluation at *any* type; this gives rise to a different notion of contextual equivalence, analogous to the one studied in (Abramsky 1990), but which can be analysed using the same kind of operational techniques introduced in this paper; see Section 8.)

Thus given $\Gamma \vdash M_1 : \tau$ and $\Gamma \vdash M_2 : \tau$ in PolyPCF, we can say that the terms M_1 and M_2 are contextually equivalent, and write $\Gamma \vdash M_1 =_{\mathrm{ctx}} M_2 : \tau$, if for any context $\mathcal{M}[-]$ for which $\mathcal{M}[M_1], \mathcal{M}[M_2] \in \mathit{Term}(\tau'\,\mathit{list})$ for some $\tau' \in \mathit{Typ}$, it is the case that

$$\mathcal{M}[M_1] \Downarrow \mathbf{nil}_{\tau'} \;\Leftrightarrow\; \mathcal{M}[M_2] \Downarrow \mathbf{nil}_{\tau'}.$$

As usual, a *context* $\mathcal{M}[-]$ means a PolyPCF term with a subterm replaced by the placeholder '$-$'; and then $\mathcal{M}[M]$ indicates the term that results from replacing the placeholder with the term M. This is a textual substitution which may well involve capture of free variables in M by binders in $\mathcal{M}[-]$. So, unlike terms, contexts are not identified up to renaming of bound variables. Although this might seem like a minor syntactic matter, it is an indication that the notion of 'context' occurring in the above definition of contextual equivalence is rather too concrete. Perhaps a better indication is the fact that the *substitutivity property* of PolyPCF contextual equivalence

$$\Gamma, x : \tau_1 \vdash M =_{\mathrm{ctx}} M' : \tau_2 \;\;\&\;\; \Gamma \vdash N =_{\mathrm{ctx}} N' : \tau_1 \;\Rightarrow\; \Gamma \vdash M[N/x] =_{\mathrm{ctx}} M'[N'/x] : \tau_2$$

is by no means an immediate consequence of the above definition of $=_{\mathrm{ctx}}$. This is because $M[N/x]$ is not of the form $\mathcal{M}_N[M]$ for some context $\mathcal{M}_N[-]$ (uniformly in M). Nevertheless, we can regard $M[N/x]$ as a use of M 'in context'; or in other words, it is reasonable to demand that the above substitutivity property holds of a notion of PolyPCF contextual equivalence *by definition*.

Partly for these reasons, we next develop a slightly more abstract, 'relational' treatment of PolyPCF contextual equivalence which avoids explicit use of contexts, following (Gordon 1998; Lassen 1998). This approach has the advantage of focussing on the key properties we want for a notion of semantic equality of PolyPCF programs, by taking them as the definition (Definition 2.2) and then deducing the existence of a *greatest*[†] relation with those properties (Theorem 2.3). This way of doing things makes it easier to state and prove the fundamental properties of the logical relation to be defined in Section 4. However, to readers more used to the traditional approach to contextual equivalence outlined above, what follows may seem unfamiliar and rather technical—in which case they may want to skip to Examples 2.6–2.9 on first reading and take references to PolyPCF observational congruence, $=_{\mathrm{obs}}$ (defined below), to mean contextual equivalence $=_{\mathrm{ctx}}$, since the two relations do indeed coincide (Remark 2.4).

Definition 2.2. Suppose $\mathcal{E}$ is a set of 4-tuples (Γ, M, M', τ) satisfying

$$\Gamma \vdash M \mathrel{\mathcal{E}} M' : \tau \;\Rightarrow\; (\Gamma \vdash M : \tau \;\;\&\;\; \Gamma \vdash M' : \tau) \qquad (1)$$

where we write $\Gamma \vdash M \mathrel{\mathcal{E}} M' : \tau$ instead of $(\Gamma, M, M', \tau) \in \mathcal{E}$.

(i) $\mathcal{E}$ is *compatible* if it is closed under the axioms and rules in Fig. 4. It is *substitutive* if it is closed under the rules in Fig. 5. (All these axioms and rules are intended to apply only to 4-tuples satisfying the well-formedness condition (1)).

(ii) It is not hard to see that compatible relations are automatically reflexive. A PolyPCF *precongruence* is a compatible, substitutive relation which is also transitive. A PolyPCF *congruence* is a precongruence which is also symmetric.

(iii) $\mathcal{E}$ is *adequate* if for all closed types $\tau \in \mathit{Typ}$ and closed terms $M, M' \in \mathit{Term}(\tau\,\mathit{list})$

$$\emptyset \vdash M \mathrel{\mathcal{E}} M' : \tau\,\mathit{list} \;\Rightarrow\; (M \Downarrow \mathbf{nil}_\tau \Leftrightarrow M' \Downarrow \mathbf{nil}_\tau).$$

Theorem 2.3 (PolyPCF observational congruence). There is a largest adequate, compatible and substitutive relation. It is an equivalence relation and hence is the largest adequate PolyPCF congruence relation. We call it *PolyPCF observational congruence* and write it as $=_{\mathrm{obs}}$.

Proof. We take $=_{\mathrm{obs}}$ to be the union of all adequate, compatible and substitutive relations. The collection of adequate relations is closed under the operations of non-empty union, relational composition and reciprocation; moreover, the identity relation is adequate, compatible and substitutive. These facts imply that $=_{\mathrm{obs}}$ is itself compatible (hence reflexive), substitutive, transitive and symmetric; see (Lassen 1998, Lemma 3.7.1 and Section 3.8). □

Remark 2.4. The substitutivity properties of Fig. 5 are just as important a part of the laws of equational logic appropriate to the language of PolyPCF as are the compatibility properties of Fig. 4. It is for this reason that we have made substitutivity a part of the definition of $=_{\mathrm{obs}}$. Nevertheless, it can be dispensed with in the definition since in fact $=_{\mathrm{obs}}$ *is the largest relation*

[†] Greatest, because one wishes to identify programs as much as possible, i.e. unless there are observable reasons for not doing so.

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

$$\frac{\Gamma, x : \tau_1 \vdash M \mathrel{\mathcal{E}} M' : \tau_2}{\Gamma \vdash \lambda\, x : \tau_1\,(M) \mathrel{\mathcal{E}} \lambda\, x : \tau_1\,(M') : \tau_1 \to \tau_2} \qquad \frac{\Gamma \vdash F \mathrel{\mathcal{E}} F' : \tau_1 \to \tau_2 \quad \Gamma \vdash A \mathrel{\mathcal{E}} A' : \tau_1}{\Gamma \vdash (F\,A) \mathrel{\mathcal{E}} (F'\,A') : \tau_2}$$

$$\frac{\alpha, \Gamma \vdash M \mathrel{\mathcal{E}} M' : \tau}{\Gamma \vdash \Lambda\,\alpha\,(M) \mathrel{\mathcal{E}} \Lambda\,\alpha\,(M') : \forall\,\alpha\,(\tau)} \quad \frac{\Gamma \vdash G \mathrel{\mathcal{E}} G' : \forall\,\alpha\,(\tau_1)}{\Gamma \vdash (G\,\tau_2) \mathrel{\mathcal{E}} (G'\,\tau_2) : \tau_1[\tau_2/\alpha]}$$

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

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

$$\frac{\Gamma \vdash L \mathrel{\mathcal{E}} L' : \tau_1\mathit{list} \quad \Gamma \vdash M_1 \mathrel{\mathcal{E}} M_1' : \tau_2 \quad \Gamma, h : \tau_1, t : \tau_1\mathit{list} \vdash M_2 \mathrel{\mathcal{E}} M_2' : \tau_2}{\Gamma \vdash (\mathbf{case}\,L\,\mathbf{of}\,\{\mathbf{nil} \Rightarrow M_1 \mid h :: t \Rightarrow M_2\}) \mathrel{\mathcal{E}} (\mathbf{case}\,L'\,\mathbf{of}\,\{\mathbf{nil} \Rightarrow M_1' \mid h :: t \Rightarrow M_2'\}) : \tau_2}$$

Fig. 4. Compatibility properties

$$\frac{\alpha, \Gamma \vdash M \mathrel{\mathcal{E}} M' : \tau_1}{\Gamma[\tau_2/\alpha] \vdash M[\tau_2/\alpha] \mathrel{\mathcal{E}} M'[\tau_2/\alpha] : \tau_1[\tau_2/\alpha]}$$

$$\frac{\Gamma, x : \tau_1 \vdash M \mathrel{\mathcal{E}} M' : \tau_2 \quad \Gamma \vdash N \mathrel{\mathcal{E}} N' : \tau_1}{\Gamma \vdash M[N/x] \mathrel{\mathcal{E}} M'[N'/x] : \tau_2}$$

Fig. 5. Substitutivity properties

which is merely adequate and compatible. The fact that the largest adequate and compatible relation is also substitutive follows once one knows that it relates β-convertible terms. This in turn can be deduced as a corollary of the properties of the logical relation of Section 4 (see also (Lassen 1998, Proposition 4.3.3) for a more direct proof). Once one knows that $=_{\mathrm{obs}}$ is the largest adequate and compatible relation, it is straightforward to see that it is indeed the same as the notion of contextual equivalence $=_{\mathrm{ctx}}$ which we mentioned at the start of this section.

We conclude this section with some examples of properties of PolyPCF types with respect to observational congruence. It does not seem easy to prove such properties directly from the definition of observational congruence (or using the more concrete notion of contextual equivalence). The logical relation of the next section will provide the means to prove them.

Notation 2.5. In the case of closed terms of closed type, we just write $M_1 =_{\mathrm{obs}} M_2 : \tau$ for $\emptyset \vdash M_1 =_{\mathrm{obs}} M_2 : \tau$.

Example 2.6 (Polymorphic null type). Consider the type

$$\mathit{null} \overset{\mathrm{def}}{=} \forall\,\alpha\,(\alpha).$$

In PolyPCF there is a closed term of this type, namely $\Omega \overset{\text{def}}{=} \Lambda\,\alpha\,(\mathbf{fix}(\lambda\,x : \alpha\,(x)))$. This is a 'polymorphic bottom' since for each $\tau \in \mathit{Typ}$ it is not hard to see that $\Omega\,\tau$ diverges, in the sense that there is no V for which $\Omega\,\tau \Downarrow V$ holds. In fact, up to observational congruence, Ω is the only closed term of type null. In other words, we claim that for all $G \in \mathit{Term}(\mathit{null})$, one has $G =_{\mathrm{obs}} \Omega : \mathit{null}$. The claim will be proved in Section 6.

Example 2.7 (Polymorphic unit type). Consider the type

$$\mathit{unit} \overset{\text{def}}{=} \forall\,\alpha\,(\alpha \to \alpha).$$

As well as the 'bottom' term $\Omega\,\mathit{unit}$, this type contains a polymorphic identity function $\Lambda\,\alpha\,(\lambda\,x : \alpha\,(x))$. But that is all: we claim that if $G \in \mathit{Term}(\mathit{unit})$, then either $G =_{\mathrm{obs}} (\Omega\,\mathit{unit}) : \mathit{unit}$ or $G =_{\mathrm{obs}} \Lambda\,\alpha\,(\lambda\,x : \alpha\,(x)) : \mathit{unit}$. The claim will be proved in Section 6.

Example 2.8 (Polymorphic lists). Consider the polymorphic list type

$$L(\alpha) \overset{\text{def}}{=} \forall\,\alpha'\,(\alpha' \to (\alpha \to \alpha' \to \alpha') \to \alpha').$$

Define terms I and J as follows:

$$\begin{aligned}
I \quad &\overset{\text{def}}{=} \quad \Lambda\,\alpha\,(\mathbf{fix}\,(\lambda\,i : \alpha\mathit{list} \to L(\alpha)\,(\lambda\,\ell : \alpha\mathit{list}\,(\\
&\qquad\qquad \Lambda\,\alpha'\,(\lambda\,x' : \alpha'\,(\lambda\,f : \alpha \to \alpha' \to \alpha'\,(\\
&\qquad\qquad\qquad \mathbf{case}\,\ell\,\mathbf{of}\,\{\mathbf{nil} \Rightarrow x' \mid h :: t \Rightarrow f\,h(i\,t\,\alpha'\,x'\,f)\}))))))) \\
J \quad &\overset{\text{def}}{=} \quad \Lambda\,\alpha\,(\lambda\,p : L(\alpha)\,(p\,(\alpha\mathit{list})\,(N\,\alpha)\,(C\,\alpha)))
\end{aligned}$$

where $N \overset{\text{def}}{=} \Lambda\,\alpha\,(\mathbf{nil}_\alpha)$ and $C \overset{\text{def}}{=} \Lambda\,\alpha\,(\lambda\,h : \alpha\,(\lambda\,t : \alpha\mathit{list}\,(h :: t)))$. Then I and J are closed terms of types $\forall\,\alpha\,(\alpha\mathit{list} \to L(\alpha))$ and $\forall\,\alpha\,(L(\alpha) \to \alpha\mathit{list})$ respectively. We claim that these terms constitute an isomorphism between $\alpha\mathit{list}$ and $L(\alpha)$ up to observational congruence, polymorphically in α. In other words, the following observational congruences hold:

$$\begin{aligned}
\alpha,\,\ell : \alpha\mathit{list} \quad &\vdash \quad J\,\alpha\,(I\,\alpha\,\ell) =_{\mathrm{obs}} \ell : \alpha\mathit{list} \\
\alpha,\,g : L(\alpha) \quad &\vdash \quad I\,\alpha\,(J\,\alpha\,g) =_{\mathrm{obs}} g : L(\alpha).
\end{aligned}$$

The claim will be proved in Section 6.

Example 2.9 (Existential types). The previous example shows that in PolyPCF, inductive list types are observationally isomorphic to polymorphic list types. In Section 7 we give another example of this phenomenon by extending PolyPCF with existential types $\exists\,\alpha\,(\tau)$, equipped with a standard operational semantics, and proving that they are observationally isomorphic to the polymorphic types $\forall\,\alpha'\,(\forall\,\alpha\,(\tau \to \alpha') \to \alpha')$ (where $\alpha' \notin \mathit{ftv}(\tau)$).

3. $\top\top$-Closed relations

We aim to characterise PolyPCF observational congruence (defined in Theorem 2.3) in terms of a binary 'logical relation' incorporating a notion of relational parametricity analogous to that introduced by Reynolds (Reynolds 1983) for the pure polymorphic lambda calculus. This result about PolyPCF observational congruence will be established in the next section and applied in Sections 5, 6, and 7. But first we have to set up some technical machinery to do with 'admissibility' properties of relations between PolyPCF terms. To see why, consider the simple example of the type $null = \forall\,\alpha\,(\alpha)$ in Example 2.6. By analogy with a relationally parametric model of the polymorphic lambda calculus, we want the relation of observational congruence at type $null$, i.e. $\{(G, G') \mid G =_{\mathrm{obs}} G' : null\}$, to coincide with

$$\textstyle\bigcap_{\tau,\tau' \in \mathit{Typ}} \{(G, G') \mid \forall\, r \in \mathcal{R}(\tau, \tau')\,((G\,\tau, G\,\tau') \in r)\} \tag{2}$$

where $\mathcal{R}(\tau, \tau')$ is some set of binary relations between closed terms of type τ and of type τ' in PolyPCF. What kind of binary relations should we take for $\mathcal{R}$? If (2) is to coincide with observational congruence, certainly $\mathcal{R}$ cannot consist of *all* binary relations, for then $\mathcal{R}(\tau, \tau')$ would contain the empty relation and hence (2) would be empty, whereas $\{(G, G') \mid G =_{\mathrm{obs}} G' : null\}$ contains (Ω, Ω), where Ω is the 'polymorphic bottom' term defined in Example 2.6. In fact we will have to restrict the parameterising relations r in the definition of relations like (2) to be at least 'admissible for fixpoint induction', in some way. In domain theory, a subset of a domain is said to be *admissible* if it contains the least element of the domain and is closed under taking least upper bounds of chains in the domain. It is perfectly possible to make use of a direct, syntactic version of this notion by considering relations between terms which are closed under the limits of certain syntactically definable chains, e.g. those generated by the finite unfoldings of a fixpoint term, or by syntactically definable projection functions. See (Birkedal and Harper 1997) for an example of this approach to 'syntactic admissibility'. Here we take a more indirect, but ultimately more useful approach (already present implicitly in (Pitts and Stark 1998)), making use of a certain closure operation on term-relations. Not only do such $\top\top$-closed relations, as we shall call them, have good admissibility properties (see Theorem 3.11), they also enjoy an important property whose denotational analogue is automatic: they respect our chosen notion of semantic equality, observational congruence (see Corollary 3.15).

Remark 3.1. Throughout this paper we concentrate on *binary* relations, rather than ones of other arities, because we are interested in using relational parametricity to deduce properties of the *binary* relation of observational congruence: it remains to be seen whether the theory of $\top\top$-closed relations presented here can be adapted to relations of other arities. However, even given the use of binary relations, the reader may wonder why in (2) we need to consider relations between different types—why not just quantify over $Rel(\tau, \tau)$, as τ ranges over Typ? The answer is that the extra generality of using $Rel(\tau, \tau')$ for potentially different domain and codomain types τ, τ' makes for a sharper characterisation of observational congruence which we exploit in several of the examples given in Sections 6 and 7.

The notion of $\top\top$-closure is the key technical contribution of this paper. It is induced by a Galois connection between term relations and relations between PolyPCF *evaluation contexts* (Felleisen and Hieb 1992)—those contexts $\mathcal{E}[-]$ with a single occurrence of the placeholder, '$-$', in the position where the next subexpression will be evaluated. The Galois connection arises in a simple manner from the '**nil**-termination' relation, $\mathcal{E}[M] \Downarrow \mathbf{nil}_\tau$, which is part of the evaluation relation of Fig. 3. To aid analysis of **nil**-termination, we use the following reformulation of evaluation contexts as stacks of 'evaluation frames' (cf. (Harper and Stone 1996) and (Pitts and Stark 1998, Section 3)).

Definition 3.2. (Frame Stacks) The grammar for PolyPCF *frame stacks* is

$$S ::= Id \mid S \circ F$$

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

$$\frac{\Gamma \vdash S : \tau' \multimap \tau'' \quad \Gamma \vdash A : \tau}{\Gamma \vdash S \circ (-\,A) : (\tau \to \tau') \multimap \tau''} \qquad \frac{\Gamma \vdash S : \tau'[\tau/\alpha] \multimap \tau''}{\Gamma \vdash S \circ (-\,\tau) : \forall\,\alpha\,(\tau') \multimap \tau''}\ \alpha \text{ not free in } \Gamma$$

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

Fig. 6. Typing frame stacks

where F ranges over *frames*:

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

Thus frames stacks are essentially finite list of frames, and we will often refer to the *length* of S meaning the length of the corresponding list.

We use the judgement $\Gamma \vdash S : \tau \multimap \tau'$ to indicate the argument and result type of a frame stack. Here Γ is a typing environment, as defined in Fig. 2, and we assume similar well-formedness conditions as in Note (i) of that figure (free variables and free type variables of all expressions in the judgement are listed in Γ). The axioms and rules inductively defining this judgement are given in Fig. 6. Unlike PolyPCF terms, we have not included explicit type information in the syntax of frame stacks. For example, *Id* is not tagged with a type. However, it is not hard to see that, given Γ, S, and τ, there is at most one τ' for which $\Gamma \vdash S : \tau \multimap \tau'$ holds. This property is enough for our purposes, since the argument type τ of a frame stack S will always be supplied in any particular situation in which we use S.

Notation 3.3. Given closed PolyPCF types $\tau, \tau' \in \mathit{Typ}$, we write $\mathit{Stack}(\tau, \tau')$ for the set of frame stacks S for which $\emptyset \vdash S : \tau \multimap \tau'$ is derivable from the axioms and rules in Fig. 6. We will be particularly interested in the case when τ' is a list type, so we write

$$\mathit{Stack}(\tau) \overset{\text{def}}{=} \textstyle\bigcup_{\tau' \in \mathit{Typ}} \mathit{Stack}(\tau, \tau'\mathit{list}).$$

Definition 3.4 (Applying a frame stack to a term). The analogue for frame stacks of the operation of filling the hole of an evaluation context with a term is given by the operation $S, M \mapsto S\,M$, of *applying a frame stack to term*. It is defined by induction on the length of the stack:

$$\begin{cases} \mathit{Id}\,M & \overset{\text{def}}{=}\ M \\ (S \circ F)\,M & \overset{\text{def}}{=}\ S\,(F[M]) \end{cases}$$

where $F[M]$ is the term which results from replacing '$-$' by M in the frame F. Note that if $S \in \mathit{Stack}(\tau, \tau')$ and $M \in \mathit{Term}(\tau)$, then $S\,M \in \mathit{Term}(\tau')$.

The following lemma shows that, unlike PolyPCF function application, the above notion of application is strict in its second argument.

$$\frac{S = S' \circ (-\,A) \qquad S' \top M[A/x]}{S \top \lambda x : \tau\,(M)} \qquad \frac{S \circ (-\,A) \top F}{S \top F\,A}$$

$$\frac{S = S' \circ (-\,\tau) \qquad S' \top M[\tau/\alpha]}{S \top \Lambda\,\alpha\,(M)} \qquad \frac{S \circ (-\,\tau) \top G}{S \top G\,\tau}$$

$$\frac{S \circ (-\,\mathbf{fix}(F)) \top F}{S \top \mathbf{fix}(F)}$$

$$\frac{S = \mathit{Id}}{S \top \mathbf{nil}_\tau} \qquad \frac{S = S' \circ (\mathbf{case}\ -\ \mathbf{of}\ \{\mathbf{nil} \Rightarrow M_1 \mid h :: t \Rightarrow M_2\}) \qquad S' \top M_1}{S \top \mathbf{nil}_\tau}$$

$$\frac{S = S' \circ (\mathbf{case}\ -\ \mathbf{of}\ \{\mathbf{nil} \Rightarrow M_1 \mid h :: t \Rightarrow M_2\}) \qquad S' \top M_2[H/h, T/t]}{S \top H :: T}$$

Fig. 7. PolyPCF structural termination relation

Lemma 3.5.

$$S\,M \Downarrow V \quad \Leftrightarrow \quad \exists\, V'\,(M \Downarrow V' \ \ \&\ \ S\,V' \Downarrow V)$$

Proof. The result follows by induction on the length of frame stacks from the corresponding property of frames, namely:

$$F[M] \Downarrow V \quad \Leftrightarrow \quad \exists\, V'\,(M \Downarrow V' \ \ \&\ \ F[V'] \Downarrow V).$$

For each of the three kinds of frame, this property is a simple consequence of the inductive definition of $\Downarrow$ in Fig. 3. □

Theorem 3.6 (A structural induction principle for PolyPCF termination). For all closed types $\tau, \tau' \in \mathit{Typ}$, for all frame stacks $S \in \mathit{Stack}(\tau, \tau'\,\mathit{list})$, and for all closed terms $M \in \mathit{Term}(\tau)$, we have

$$S\,M \Downarrow \mathbf{nil}_{\tau'} \quad \Leftrightarrow \quad S \top M$$

where the relation $(-) \top (-)$ is inductively defined by the rules in Fig. 7.

Proof. The way we have defined $(-) \top (-)$ is good for the use we will make of the theorem, but perhaps obscures its origin, which lies in Felleisen's evaluation-context based presentation of structural operational semantics (Felleisen and Hieb 1992) and Krivine's stack-based abstract machines for evaluating lambda terms (see (Amadio and Curien 1998, page 184) for example). Thus we can define a transition relation between (stack,term)-pairs, $(S, M) \rightarrow (S', M')$, by case analysis on the structure of M and then S, as follows:

$$\begin{array}{rl}
(S, F[M]) \to (S \circ F, M) & \text{if } M \text{ is not a value} \\
(S \circ F, V) \to (S, F[V]) & \text{if } V \text{ is a value} \\
(S, R) \to (S, R') & \text{where } (R, R') \text{ is one of the following (redex,reduct)-pairs:}
\end{array}$$

R	R'
$(\lambda\, x : \tau\, (M))\, A$	$M[A/x]$
$(\Lambda\, \alpha\, (M))\, \tau$	$M[\tau/\alpha]$
$\mathbf{fix}(F)$	$F\, \mathbf{fix}(F)$
$\mathbf{case}\, \mathbf{nil}_\tau\, \mathbf{of}\, \{\mathbf{nil} \Rightarrow M_1 \mid h :: t \Rightarrow M_2\}$	M_1
$\mathbf{case}\, H :: T\, \mathbf{of}\, \{\mathbf{nil} \Rightarrow M_1 \mid h :: t \Rightarrow M_2\}$	$M_2[H/h, T/t]$.

It is not hard to see that $S \top M$ holds if and only if there is a finite sequence of transitions $(S, M) \to^* (Id, \mathbf{nil}_\tau)$ (for some τ). So the theorem follows once we prove

$$S\, M \Downarrow V \quad \Leftrightarrow \quad (S, M) \to^* (Id, V). \tag{3}$$

This can be deduced using the following lemmas:

$$M \Downarrow V \;\Rightarrow\; \forall S\, .\, (S, M) \to^* (S, V) \tag{4}$$

$$(S, M) \to (S', M') \;\Rightarrow\; \forall V\, (S'\, M' \Downarrow V \;\Rightarrow\; S\, M \Downarrow V) \tag{5}$$

$$(S, S'\, M) \to^* (S@S', M) \tag{6}$$

where $S@S'$ denotes the result of appending S' to S:

$$\begin{cases} S@Id & \stackrel{\text{def}}{=}\; S \\ S@(S' \circ F) & \stackrel{\text{def}}{=}\; (S@S') \circ F. \end{cases}$$

(4) follows by induction on the derivation of $M \Downarrow V$; (5) by case analysis of $\to$; and (6) by induction on the length of S'. Then (4) and (6), together with the determinacy of $\to$, imply the left-to-right implication in (3). The reverse implication follows by repeated use of (5) starting from the fact that $Id\, V = V$ and hence $Id\, V \Downarrow V$. □

The relation $(-) \top (-)$ gives a *structural* characterisation of $\mathbf{nil}$-termination of PolyPCF evaluation because of the syntax-directed nature of the rules in Fig. 7. This facilitates inductive proofs involving termination. In particular it helps with proving the following important property of termination. (For statements and proofs of analogous properties see for example: (Mason, Smith, and Talcott 1996, Section 4.3), (Pitts and Stark 1998, Theorem 3.2), (Birkedal and Harper 1997, Section 3.1), and (Lassen 1998, Section 4.5).)

Theorem 3.7 (Unwinding Theorem for PolyPCF). For any $\tau \in \mathit{Typ}$ and $F \in \mathit{Term}(\tau \to \tau)$, define terms $\mathbf{fix}^{(n)}(F) \in \mathit{Term}(\tau)$ for each $n \geq 0$ as follows.

$$\begin{cases} \mathbf{fix}^{(0)}(F) & \stackrel{\text{def}}{=}\; \mathbf{fix}(\lambda\, x : \tau\, (x)) \\ \mathbf{fix}^{(n+1)}(F) & \stackrel{\text{def}}{=}\; F\, \mathbf{fix}^{(n)}(F). \end{cases}$$

Then for any $S \in \mathit{Stack}(\tau)$ it is the case that

$$S \top \mathbf{fix}(F) \;\Leftrightarrow\; \exists\, n \geq 0\, (S \top \mathbf{fix}^{(n)}(F)).$$

Proof. Note that it follows from the inductive definition of $(-) \top (-)$ (Fig. 7) that $S \top \mathbf{fix}^{(0)}(F)$ holds for no S. Consequently, given any $\tau, \tau', \tau'' \in \mathit{Typ}$, $x : \tau \vdash S : \tau' \multimap \tau''\mathit{list}$ and $x : \tau \vdash M : \tau'$ one can prove

$$S[\mathbf{fix}^{(0)}(F)/x] \top M[\mathbf{fix}^{(0)}(F)/x] \;\Rightarrow\; \forall\, A \in \mathit{Term}(\tau)\, (S[A/x] \top M[A/x])$$

by induction on the derivation of $(-) \top (-)$ (for all τ', τ'', S, and M simultaneously)[‡]. From this it follows by induction on $n \geq 0$ that for all S and M

$$S[\mathbf{fix}^{(n)}(F)/x] \top M[\mathbf{fix}^{(n)}(F)/x] \;\Rightarrow\; S[\mathbf{fix}^{(n+1)}(F)/x] \top M[\mathbf{fix}^{(n+1)}(F)/x]. \tag{7}$$

Now one can prove the key property

$$S[\mathbf{fix}(F)/x] \top M[\mathbf{fix}(F)/x] \;\Rightarrow\; \exists\, n \geq 0\, (S[\mathbf{fix}^{(n)}/x] \top M[\mathbf{fix}^{(n)}(F)/x]) \tag{8}$$

by induction on the derivation of $(-) \top (-)$ for all τ', τ'', S, and M simultaneously, using (7) when it comes to the rule for **fix** terms. Clearly the left-to-right implication of the theorem is an instance of (8); and the converse implication is a special case of

$$S[\mathbf{fix}^{(n)}(F)/x] \top M[\mathbf{fix}^{(n)}(F)/x] \;\Rightarrow\; S[\mathbf{fix}(F)/x] \top M[\mathbf{fix}(F)/x] \tag{9}$$

which, once again, can be proved by induction on the derivation of $(-) \top (-)$ for all τ', S, M, and n simultaneously. □

Definition 3.8 (Term- and stack-relations). A PolyPCF *term-relation* is a binary relation between (typeable) closed terms. Given closed PolyPCF types $\tau, \tau' \in \mathit{Typ}$, we write

$$\mathit{Rel}(\tau, \tau')$$

for the set of term-relations that are subsets of $\mathit{Term}(\tau) \times \mathit{Term}(\tau')$. A PolyPCF *stack-relation* is a binary relation between (typeable) frame stacks whose result types are list types. We write

$$\mathit{StRel}(\tau, \tau')$$

for the set of stack-relations that are subsets of $\mathit{Stack}(\tau) \times \mathit{Stack}(\tau')$. (Recall from Notation 3.3 that the elements of $\mathit{Stack}(\tau)$ are frame stacks S satisfying $\emptyset \vdash S : \tau \multimap \tau''\mathit{list}$ for some τ''.)

Using the $(-) \top (-)$ relation we can manufacture a stack-relation from a term-relation and vice versa, as follows.

Definition 3.9 (The $(-)^{\top}$ operation on relations). Given any $\tau, \tau' \in \mathit{Typ}$ and $r \in \mathit{Rel}(\tau, \tau')$, define $r^{\top} \in \mathit{StRel}(\tau, \tau')$ by

$$(S, S') \in r^{\top} \quad \overset{\text{def}}{\Leftrightarrow} \quad \forall\, (M, M') \in r\, (S \top M \Leftrightarrow S' \top M');$$

[‡] We are interested in the special case when $x \notin \mathit{fv}(S)$ and $M = x$, but the more general statement is needed to make the induction go through; similarly for (7), (8) and (9).

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

$$(M, M') \in s^{\top} \quad \overset{\text{def}}{\Leftrightarrow} \quad \forall\, (S, S') \in s\, (S \top M \Leftrightarrow S' \top M').$$

Just from the form of the definition of the operations $r \mapsto r^{\top}$, $s \mapsto s^{\top}$ (i.e. without using any properties of the termination relation $(-) \top (-)$) it is clear that one has a Galois connection with respect to inclusion:

$$r \subseteq s^{\top} \;\Leftrightarrow\; s \subseteq r^{\top}. \tag{10}$$

Hence the operations $(-)^{\top}$ are inclusion-reversing

$$\begin{aligned} r_1 \subseteq r_2 \;&\Rightarrow\; (r_2)^{\top} \subseteq (r_1)^{\top} \\ s_1 \subseteq s_2 \;&\Rightarrow\; (s_2)^{\top} \subseteq (s_1)^{\top} \end{aligned}$$

and $r \mapsto r^{\top\top}$ is a closure operator for term-relations, i.e. is inclusion-preserving

$$r_1 \subseteq r_2 \;\Rightarrow\; (r_1)^{\top\top} \subseteq (r_2)^{\top\top} \tag{11}$$

inflationary

$$r \subseteq r^{\top\top}$$

and idempotent

$$r^{\top\top} = (r^{\top\top})^{\top\top}.$$

Definition 3.10 ($\top\top$-Closed term-relations). A term-relation r is $\top\top$-*closed* if $r = r^{\top\top}$, or equivalently if $r^{\top\top} \subseteq r$, or equivalently if $r = s^{\top}$ for some stack-relation s, or equivalently if $r = (r')^{\top\top}$ for some term-relation r'.

The next result is a PolyPCF version (for binary relations) of Scott's Fixpoint Induction Principle (Scott 1993). It justifies the claim that $\top\top$-closed term-relations have appropriate 'admissibility' properties.

Theorem 3.11 (PolyPCF fixpoint induction). Suppose $r \in \mathit{Rel}(\tau, \tau')$, $F \in \mathit{Term}(\tau \to \tau)$, and $F' \in \mathit{Term}(\tau' \to \tau')$ satisfy

$$\forall (A, A') \in r\,.\,(F\,A, F'\,A') \in r. \tag{12}$$

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

Proof. Recall the definition of the terms $\mathbf{fix}^{(n)}(F)$ from Theorem 3.7. From the definition of $(-) \top (-)$ in Fig. 7, it follows that $S \top \mathbf{fix}^{(0)}(F)$ does not hold for any $S \in \mathit{Stack}(\tau)$. So we have that $(\mathbf{fix}^{(0)}(F), \mathbf{fix}^{(0)}(F)) \in s^{\top}$ for any $s \in \mathit{StRel}(\tau, \tau')$. Hence in particular taking $s = r^{\top}$, we have

$$(\mathbf{fix}^{(0)}(F), \mathbf{fix}^{(0)}(F')) \in r^{\top\top} = r.$$

Using (12), it follows from this by induction on n that

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

holds for all $n \geq 0$. So for any $(S, S') \in r^{\top}$ we have

$$\begin{array}{lll} S \top \mathbf{fix}(F) & \Leftrightarrow \ \exists\, n \geq 0\, (S \top \mathbf{fix}^{(n)}(F)) & \text{by Theorem 3.7} \\ & \Leftrightarrow \ \exists\, n \geq 0\, (S' \top \mathbf{fix}^{(n)}(F')) & \text{since } (\mathbf{fix}^{(n)}(F), \mathbf{fix}^{(n)}(F')) \in r \text{ and } (S, S') \in r^{\top} \\ & \Leftrightarrow \ S' \top \mathbf{fix}(F') & \text{by Theorem 3.7.} \end{array}$$

Thus by definition of $(-)^{\top}$, $(\mathbf{fix}(F), \mathbf{fix}(F)) \in r^{\top\top} = r$, as required. □

Another property of $\top\top$-closed term-relations we will need is that they respect various notions of semantic equivalence for PolyPCF. We have already introduced our primary such notion, observational congruence, in Section 2. We will also need to consider a simpler notion, that of 'Kleene equivalence'[§]

Definition 3.12 (Kleene equivalence). For each closed type $\tau \in \mathit{Typ}$ and closed terms $M, M' \in \mathit{Term}(\tau)$ we write $M =_{\mathrm{kl}} M' : \tau$ to mean $\forall\, V\, (M \Downarrow V \ \Leftrightarrow \ M' \Downarrow V)$.

Example 3.13. Each of the four pairs (R, R') mentioned in the last clause of the definition of $\to$ in the proof of Theorem 3.6 is an example of a Kleene equivalence. This follows immediately from the definition of $\Downarrow$ in Fig. 3.

Although it is by no means immediate from their definitions, it is in fact the case that Kleene equivalence implies observational congruence (see Corollary 4.16). The converse is false: for example, $\lambda\, x : \tau\, (\mathbf{fix}(\lambda\, x' : \tau'\, (x')))$ and $\mathbf{fix}(\lambda\, f : \tau \to \tau'\, (f))$ are not Kleene equivalent terms of type $\tau \to \tau'$ (because the first is a value whereas the second does not evaluate to anything), but the results of Section 5 show that they are observationally congruent (cf. Example 5.3).

Lemma 3.14. If $M =_{\mathrm{kl}} M' : \tau$ or $M =_{\mathrm{obs}} M' : \tau$, then for any $S \in \mathit{Stack}(\tau)$

$$S \top M \ \Leftrightarrow \ S \top M'.$$

Proof. First suppose that $M =_{\mathrm{kl}} M' : \tau$. Then

$$\begin{array}{lll} S \top M & \Leftrightarrow \exists\, V, \tau'\, (M \Downarrow V \ \ \& \ \ S\, V \Downarrow \mathbf{nil}_{\tau'}) & \text{by Theorem 3.6 and Lemma 3.5} \\ & \Leftrightarrow \exists\, V, \tau'\, (M' \Downarrow V \ \ \& \ \ S\, V \Downarrow \mathbf{nil}_{\tau'}) & \text{since } M =_{\mathrm{kl}} M' : \tau \\ & \Leftrightarrow S \top M' & \text{by Theorem 3.6 and Lemma 3.5.} \end{array}$$

The argument for observational congruence is slightly different. If $M =_{\mathrm{obs}} M' : \tau$, then by the compatibility properties which are part of the definition of $=_{\mathrm{obs}}$, we also have $S\, M =_{\mathrm{obs}} S\, M' : \tau'\, \mathit{list}$ for any $S \in \mathit{Stack}(\tau, \tau'\, \mathit{list})$ and $\tau' \in \mathit{Typ}$. Since $=_{\mathrm{obs}}$ is adequate (Definition 2.2(iii)), we have

$$S\, M \Downarrow \mathbf{nil}_{\tau'} \ \Leftrightarrow \ S\, M' \Downarrow \mathbf{nil}_{\tau'}$$

from which the result follows by Theorem 3.6. □

In view of this lemma, it follows from the definition of $(-)^{\top}$ that if $(M_1, M_1') \in s^{\top}$, then $(M_2, M_1') \in s^{\top}$ for any M_2 which is either Kleene equivalent to M_1, or observationally congruent to it; and similarly for M_1'. Thus we have:

[§] I believe this terminology, in the context of notions of program equivalence, was introduced by Robert Harper.

Corollary 3.15. The $\top\top$-closed term-relations respect both observational congruence and Kleene equivalence. In other words, if $r \in \mathit{Rel}(\tau, \tau')$ is $\top\top$closed, then for $\sim$ equal to either $=_{\mathrm{obs}}$ or $=_{\mathrm{kl}}$, we have

$$M_1 \sim M_2 : \tau \ \ \& \ \ (M_2, M_3) \in r \ \ \& \ \ M_3 \sim M_4 : \tau' \ \Rightarrow \ (M_1, M_4) \in r.$$

4. A syntactical version of relational parametricity

In order to characterise PolyPCF observational congruence in terms of parametric logical relations we introduce a notion of 'action' of the PolyPCF types on term-relations. This is defined by induction on the structure of types, using the following operations on term-relations, the first of which is characteristic of all the various notions of 'logical' relations right back to their beginning (Plotkin 1973).

Definition 4.1 (Action of $\rightarrow$ on term-relations). Given $r_1 \in \mathit{Rel}(\tau_1, \tau_1')$ and $r_2 \in \mathit{Rel}(\tau_2, \tau_2')$, we define $r_1 \rightarrow r_2 \in \mathit{Rel}(\tau_1 \rightarrow \tau_2, \tau_1' \rightarrow \tau_2')$ by:

$$(F, F') \in r_1 \rightarrow r_2 \overset{\mathrm{def}}{\Leftrightarrow} \forall\, (A, A') \in r_1\, ((F\, A, F'\, A') \in r_2).$$

Definition 4.2 (Action of $\forall$ on term-relations). Let τ_1 and τ_1' be PolyPCF types with at most a single free type variable, α say. Suppose R is a function mapping term-relations $r \in \mathit{Rel}(\tau_2, \tau_2')$ (any $\tau_2, \tau_2' \in \mathit{Typ}$) to term-relations $R(r) \in \mathit{Rel}(\tau_1[\tau_2/\alpha], \tau_1'[\tau_2'/\alpha])$. Then we can form a term-relation $\forall\, r\, (R(r)) \in \mathit{Rel}(\forall\, \alpha\, (\tau_1), \forall\, \alpha\, (\tau_1'))$ as follows:

$$(G, G') \in \forall\, r\, (R(r)) \overset{\mathrm{def}}{\Leftrightarrow} \forall\, \tau_2, \tau_2' \in \mathit{Typ}\, (\forall\, r \in \mathit{Rel}(\tau_2, \tau_2')\, ((G\, \tau_2, G'\, \tau_2') \in R(r))).$$

Definition 4.3 (Action of $(-)\mathit{list}$ on term-relations). Given $\tau, \tau' \in \mathit{Typ}$, $r_1 \in \mathit{Rel}(\tau, \tau')$ and $r_2 \in \mathit{Rel}(\tau\, \mathit{list}, \tau'\, \mathit{list})$, define $1 + (r_1 \times r_2) \in \mathit{Rel}(\tau\, \mathit{list}, \tau'\, \mathit{list})$ by:

$$\begin{aligned} &1 + (r_1 \times r_2) \overset{\mathrm{def}}{=} \\ &\qquad \{(\mathbf{nil}_\tau, \mathbf{nil}_{\tau'})\} \cup \{(H :: T, H' :: T') \mid (H, H') \in r_1 \ \&\ (T, T') \in r_2\}. \end{aligned}$$

Note that the subset relation makes $\mathit{Rel}(\tau\, \mathit{list}, \tau'\, \mathit{list})$ into a complete lattice and that, for each r_1, the function $r_2 \mapsto (1 + (r_1 \times r_2))^{\top\top}$ is monotone (cf. (11)). Therefore we can form its greatest (post-)fixed point:

$$(r_1)\mathit{list} \overset{\mathrm{def}}{=} \nu\, r_2\, (1 + (r_1 \times r_2))^{\top\top}.$$

Thus $(r_1)\mathit{list}$ is the unique term-relation satisfying

$$(r_1)\mathit{list} \ = \ (1 + (r_1 \times (r_1)\mathit{list}))^{\top\top} \tag{13}$$

$$\forall\, r_2\, (r_2 \subseteq (1 + (r_1 \times r_2))^{\top\top} \ \Rightarrow \ r_2 \subseteq (r_1)\mathit{list}). \tag{14}$$

The appearance of $(-)^{\top\top}$ in this definition and the reason for using a greatest fixed point (rather than a least one) are discussed in Remark 4.4.

Combining the preceding definitions, for each PolyPCF type τ and each list $\vec{\alpha} = \alpha_1, \ldots, \alpha_n$ of distinct type variables containing the free type variables of τ, we define a function from tuples of term-relations to term-relations

$$r_1 \in \mathit{Rel}(\tau_1, \tau_1'), \ldots, r_n \in \mathit{Rel}(\tau_n, \tau_n') \mapsto \Delta_\tau(\vec{r}/\vec{\alpha}) \in \mathit{Rel}(\tau[\vec{\tau}/\vec{\alpha}], \tau[\vec{\tau}'/\vec{\alpha}']). \tag{19}$$

$$\begin{aligned}
\Delta_{\alpha_i}(\vec{r}/\vec{\alpha}) &\overset{\text{def}}{=} r_i && (15)\\
\Delta_{\tau\to\tau'}(\vec{r}/\vec{\alpha}) &\overset{\text{def}}{=} \Delta_\tau(\vec{r}/\vec{\alpha}) \to \Delta_{\tau'}(\vec{r}/\vec{\alpha}) && (16)\\
\Delta_{\forall\,\alpha\,(\tau)}(\vec{r}/\vec{\alpha}) &\overset{\text{def}}{=} \forall\, r\,(\Delta_\tau(r^{\top\top}/\alpha, \vec{r}/\vec{\alpha})) && (17)\\
\Delta_{\tau\,list}(\vec{r}/\vec{\alpha}) &\overset{\text{def}}{=} (\Delta_\tau(\vec{r}/\vec{\alpha}))\mathit{list} && (18)
\end{aligned}$$

Fig. 8. Definition of the logical relation Δ

The definition proceeds by induction on the structure of τ and is given in Fig. 8. We will see below (Lemma 4.11) that these functions have the important property that they yield a $\top\top$-closed term-relation when applied to a tuple of such relations. The '$n = 0$' case of (19) means that for each *closed* type $\tau \in \mathit{Typ}$ we can apply Δ_τ to the empty tuple of term-relations to obtain a term-relation $\Delta_\tau() \in \mathit{Rel}(\tau, \tau)$. We will see below (Theorem 4.15) that $\Delta_\tau()$ coincides with the relation of observational congruence (defined in Theorem 2.3) at type τ:

$$M_1 =_{\text{ctx}} M_2 : \tau \;\Leftrightarrow\; (M_1, M_2) \in \Delta_\tau() \qquad (M_1, M_2 \in \mathit{Term}(\tau)). \tag{20}$$

This, together with the definition of Δ at $\forall$-types is what permits us to deduce results like those in Examples 2.6–2.8 (see Sections 6 and 7).

Remark 4.4. In Fig. 8, the closure operation $(-)^{\top\top}$ has been carefully combined with the operations of Definitions 4.1–4.3 to ensure that (20) holds. The way that $(-)^{\top\top}$ appears in the definition of $\Delta_{\tau\,list}$ used in clause (18) reflects the fact that for PolyPCF observational congruence we observe termination at list types. The use of '$r^{\top\top}/\alpha$' in clause (17) is a way of ensuring that the quantification is restricted to range over $\top\top$-closed relations. As remarked at the beginning of Section 3, some such restriction to do with admissibility is necessary if such actions are to characterise $=_{\text{obs}}$; and we saw in Theorem 3.11 that being $\top\top$-closed ensures that a term-relation has good admissibility properties. Finally, it is worth pointing out that if one changes the greatest fixed point used in Definition 4.3 to a least fixed point, thereby obtaining a different action, call it Δ', one can still deduce the coincidence of $\Delta'_\tau()$ with $=_{\text{obs}}$. This corresponds to the fact that denotationally, the relation of equality for recursive data types has a mixed inductive/coinductive character: see (Pitts 1996); and in the case of algebraic data types like $\tau\,\mathit{list}$, the mixed inductive/co-inductive character of equality becomes a *simultaneous* one. Here we have concentrated on the co-inductive aspect of $=_{\text{obs}}$ at list types rather than its inductive aspect, because that is more useful for proving semantic equalities between potentially infinite lists: see Section 6.

As is typical for logical relations, in order to prove (20) we need to extend $\Delta_\tau()$ to a relation between *open* terms. We do this via closing substitutions.

Definition 4.5. (Logical relation on open terms) Suppose $\Gamma \vdash M : \tau$ and $\Gamma \vdash M' : \tau$ hold, with $\Gamma = \alpha_1, \ldots, \alpha_m,\, x_1 : \tau_1, \ldots, x_n : \tau_n$ say. Write

$$\Gamma \vdash M \mathrel{\Delta} M' : \tau \tag{21}$$

to mean: given any $\sigma_i, \sigma'_i \in \mathit{Typ}$ and $r_i \in \mathit{Rel}(\sigma_i, \sigma'_i)$ (for $i = 1..m$) with each r_i $\top\top$-closed, then for any $(N_j, N'_j) \in \Delta_{\tau_j}(\vec{r}/\vec{\alpha})$ (for $j = 1..n$) it is the case that

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

(The restriction to $\top\top$-closed relations in this definition accords with the definition of Δ at $\forall$-types.)

Proposition 4.6 ('Fundamental Property' of the logical relation). The relation (21) between open PolyPCF terms is compatible and substitutive, in the sense of Definition 2.2(i).

The proof of this proposition occupies most of the rest of this section. We begin with some lemmas expressing general properties of the constructions in Definitions 4.1–4.3 with respect to the Galois connection $(-)^\top$ introduced in Section 3.

Lemma 4.7. Suppose $r_1 \in \mathit{Rel}(\tau_1, \tau'_1)$ and $r_2 \in \mathit{Rel}(\tau_2, \tau'_2)$. Consider $r_1 \to r_2$ as in Definition 4.1.

(i) Suppose given values $\lambda\, x : \tau_1\, (M)$ and $\lambda\, x : \tau'_1\, (M')$ of types $\tau_1 \to \tau_2$ and $\tau'_1 \to \tau'_2$ respectively, satisfying

$$\forall\, (A, A') \in r_1\, ((M[A/x], M'[A'/x]) \in r_2). \qquad (22)$$

If r_2 is $\top\top$-closed, then $(\lambda\, x : \tau_1\, (M), \lambda\, x : \tau'_1\, (M')) \in r_1 \to r_2$.

(ii) If $(A, A') \in r_1$ and $(S, S') \in r_2^\top$, then $(S \circ (-\, A), S' \circ (-\, A')) \in (r_1 \to r_2)^\top$.

(iii) If r_2 is $\top\top$-closed, then so is $r_1 \to r_2$.

Proof. For part (i), we have to show for all $(A, A') \in r_1$ that

$$((\lambda\, x : \tau_1\, (M))\, A, (\lambda\, x : \tau'_1\, (M'))\, A') \in r_2. \qquad (23)$$

From Example 3.13 we have the Kleene equivalences

$$(\lambda\, x : \tau_1\, (M))\, A =_{\mathrm{kl}} M[A/x] : \tau_2 \quad \text{and} \quad (\lambda\, x : \tau'_1\, (M'))\, A' =_{\mathrm{kl}} M'[A'/x] : \tau_2$$

and by Corollary 3.15, r_2 respects $=_{\mathrm{kl}}$. Hence (23) follows from assumption (22).

To prove part (ii), suppose $(A, A') \in r_1$ and $(S, S') \in r_2^\top$. Then note that for any $(F, F') \in r_1 \to r_2$

$$\begin{aligned} S \circ (-\, A) \top F &\Leftrightarrow S \top F\, A && \text{by definition of } (-) \top (-) \text{ (Fig. 7)} \\ &\Leftrightarrow S' \top F'\, A' && \text{since } (F\, A, F'\, A') \in r_2 \text{ and } (S, S') \in r_2^\top \\ &\Leftrightarrow S' \circ (-\, A') \top F' && \text{by definition of } (-) \top (-). \end{aligned}$$

Since this is so for any such (F, F'), by definition of $(-)^\top$ we have $(S \circ (-\, A), S' \circ (-\, A')) \in (r_1 \to r_2)^\top$.

Finally, to prove part (ii), suppose $(F, F') \in (r_1 \to r_2)^{\top\top}$, $(A, A') \in r_1$, and $(S, S') \in r_2^\top$. Then by part (ii) we have $(S \circ (-\, A), S' \circ (-\, A')) \in (r_1 \to r_2)^\top$ and hence

$$S \circ (-\, A) \top F \;\Leftrightarrow\; S' \circ (-\, A') \top F'.$$

Therefore

$$S \top F\, A \;\Leftrightarrow\; S' \top F'\, A'.$$

Since this holds for all such (S, S'), we have $(F\,A, F'\,A') \in (r_2^{\top})^{\top} = r_2$; and since this is so for all $(A, A') \in r_1$, it follows that $(F, F') \in r_1 \to r_2$. Thus we have succeeded in proving that $(r_1 \to r_2)^{\top\top} \subseteq r_1 \to r_2$, i.e. that $r_1 \to r_2$ is $\top\top$-closed. □

Lemma 4.8. Let τ_1 and τ_1' be PolyPCF types with at most a single free type variable, α say. Suppose R is a function mapping term-relations $r \in \mathit{Rel}(\tau_2, \tau_2')$ (any $\tau_2, \tau_2' \in \mathit{Typ}$) to term-relations $R(r) \in \mathit{Rel}(\tau_1[\tau_2/\alpha], \tau_1'[\tau_2'/\alpha])$. Let $\forall\, r\,(R(r))$ be as in Definition 4.2.

(i) Suppose given values $\Lambda\,\alpha\,(M)$ and $\Lambda\,\alpha\,(M')$ of types $\forall\,\alpha\,(\tau_1)$ and $\forall\,\alpha\,(\tau_1')$ respectively, satisfying

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

If each $R(r)$ is $\top\top$-closed, then $(\Lambda\,\alpha\,(M), \Lambda\,\alpha\,(M')) \in \forall\, r\,(R(r))$.

(ii) If $r_2 \in \mathit{Rel}(\tau_2, \tau_2')$ and $(S, S') \in R(r_2)^{\top}$, then $(S \circ (-\,\tau_2), S' \circ (-\,\tau_2')) \in (\forall\, r\,(R(r)))^{\top}$.

(iii) If each $R(r)$ is $\top\top$-closed, then so is $\forall\, r\,(R(r))$.

Proof. The proof is similar to that for Lemma 4.7, but using the property

$$S \circ (-\,\tau) \top G \;\Leftrightarrow\; S \top G\,\tau$$

(which follows immediately from the definition of $(-) \top (-)$ in Fig. 7) and the Kleene equivalence $(\Lambda\,\alpha\,(M))\,\tau =_{\mathrm{kl}} M[\tau/\alpha]$. □

Lemma 4.9. Suppose $r \in \mathit{Rel}(\tau, \tau')$ and consider $(r)\mathit{list}$ as in Definition 4.3.

(i) $(\mathbf{nil}_{\tau}, \mathbf{nil}_{\tau'}) \in (r)\mathit{list}$; and if $(H, H') \in r$ and $(T, T') \in (r)\mathit{list}$, then $(H :: T, H' :: T') \in (r)\mathit{list}$.

(ii) $(\mathit{Id}, \mathit{Id}) \in ((r)\mathit{list})^{\top}$.

(iii) $(r)\mathit{list}$ is $\top\top$-closed.

Proof. In view of (13), $(r)\mathit{list}$ contains $1 + (r \times (r)\mathit{list})$, from which part (i) follows immediately.

Note that from (13) we have

$$((r)\mathit{list})^{\top} = ((1 + (r \times (r)\mathit{list}))^{\top\top})^{\top} = (1 + (r \times (r)\mathit{list}))^{\top}.$$

So for part (ii), it suffices to show for all $(L, L') \in 1 + (r \times (r)\mathit{list})$ that $\mathit{Id} \top L \Leftrightarrow \mathit{Id} \top L'$. But if $(L, L') \in 1 + (r \times (r)\mathit{list})$, then

either$L = \mathbf{nil}_{\tau}$ and $L' = \mathbf{nil}_{\tau'}$, in which case both $\mathit{Id} \top L$ and $\mathit{Id} \top L'$ hold;

or$L = H :: T$ and $L' = H' :: T'$ (for some $(H, H') \in r$ and $(T, T') \in (r)\mathit{list}$), in which case neither $\mathit{Id} \top L$, nor $\mathit{Id} \top L'$ hold.

Finally, for part (iii) just note that by (13), $(r)\mathit{list}$ is $\top\top$-closed because it is of the form $(r')^{\top\top}$ for some r'. □

Lemma 4.10. Suppose given closed types $\tau_1, \tau_1', \tau_2, \tau_2' \in \mathit{Typ}$, term-relations $r_1 \in \mathit{Rel}(\tau_1, \tau_1')$, $r_2 \in \mathit{Rel}(\tau_2, \tau_2')$, closed terms $(M_1, M_1') \in r_2$, and open terms

$$h : \tau_1, t : \tau_1\,\mathit{list} \vdash M_2 : \tau_2 \quad \text{and} \quad h' : \tau_1', t' : \tau_1'\,\mathit{list} \vdash M_2' : \tau_2'$$

satisfying

$$\forall\,(H, H') \in r_1, (T, T') \in (r_1)\mathit{list}\,((M_2[H/h, T/t], M_2'[H'/h', T'/t']) \in r_2). \tag{24}$$

Writing

$$match \overset{\text{def}}{=} \{\mathbf{nil} \Rightarrow M_1 \mid h :: t \Rightarrow M_2\} \quad \text{and} \quad match' \overset{\text{def}}{=} \{\mathbf{nil} \Rightarrow M_1' \mid h' :: t' \Rightarrow M_2'\}$$

we have:

(i) If $(S, S') \in (r_2)^\top$, then $(S \circ (\mathbf{case} - \mathbf{of}\ match), S' \circ (\mathbf{case} - \mathbf{of}\ match')) \in ((r_1)list)^\top$.
(ii) If $(L, L') \in (r_1)list$ and r_2 is $\top\top$-closed, then $(\mathbf{case}\, L\, \mathbf{of}\ match, \mathbf{case}\, L'\, \mathbf{of}\ match') \in r_2$.

Proof. To prove part (i), assume given $(S, S') \in (r_2)^\top$. As noted in the proof of the previous lemma, we have $((r_1)list)^\top = (1 + (r_1 \times (r_1)list))^\top$. So it suffices to check for all $(L, L') \in 1 + (r_1 \times (r_1)list)$ that

$$S \circ (\mathbf{case} - \mathbf{of}\ match) \top L \;\Leftrightarrow\; S' \circ (\mathbf{case} - \mathbf{of}\ match') \top L'. \tag{25}$$

But if $(L, L') \in 1 + (r_1 \times (r_1)list)$, then

either$L = \mathbf{nil}_{\tau_1}$ and $L' = \mathbf{nil}_{\tau_1'}$, in which case by definition of $(-) \top (-)$, (25) holds if and only if

$$S \top M_1 \;\Leftrightarrow\; S' \top M_1'$$

which is the case because $(S, S') \in (r_2)^\top$ and $(M_1, M_1') \in r_2$;

or$L = H :: T$ and $L' = H' :: T'$ for some $(H, H') \in r_1$ and $(T, T') \in (r_1)list$, in which case by definition of $(-) \top (-)$, (25) holds if and only if

$$S \top M_2[H/h, T/t] \;\Leftrightarrow\; S' \top M_2'[H'/h', T'/t']$$

which is the case because of property (24).

For part (ii), if $(L, L') \in (r_1)list$, then for any $(S, S') \in (r_2)^\top$ we have

$$\begin{aligned} S \top (\mathbf{case}\, L\, \mathbf{of}\ match) &\Leftrightarrow S \circ (\mathbf{case} - \mathbf{of}\ match) \top L && \text{by definition of } (-) \top (-) \\ &\Leftrightarrow S' \circ (\mathbf{case} - \mathbf{of}\ match') \top L' && \text{by part (i)} \\ &\Leftrightarrow S' \top (\mathbf{case}\, L'\, \mathbf{of}\ match') && \text{by definition of } (-) \top (-). \end{aligned}$$

Since this holds for all $(S, S') \in (r_2)^\top$, we have that $(\mathbf{case}\, L\, \mathbf{of}\ match, \mathbf{case}\, L'\, \mathbf{of}\ match') \in (r_2)^{\top\top}$. So if $r_2 = (r_2)^{\top\top}$, we have the desired conclusion. □

We are now in a position to prove an important property of the action of PolyPCF types on term-relations given in Fig. 8.

Lemma 4.11. For each open type τ, with free type variables in $\vec{\alpha}$ say, if the term-relations $\vec{r}$ are $\top\top$-closed, then so is the term-relation $\Delta_\tau(\vec{r}/\vec{\alpha})$ defined in Fig. 8. In particular for each closed type τ, $\Delta_\tau() \in Rel(\tau, \tau)$ is $\top\top$-closed.

Proof. By induction on the structure of τ, using clauses (15)–(18) of the definition of Δ. The base case of a type variable is trivial because each r_i is assumed to be $\top\top$-closed. The induction step for function types follows from Lemma 4.7(iii). The induction step for $\forall$-types follows from Lemma 4.8(iii). The induction step for list types is trivial since by (13), $(\Delta_\tau(\vec{r}/\vec{\alpha}))list$ is equal to a term-relation of the form $(r')^{\top\top}$. □

Lemma 4.12. Assuming $\mathit{ftv}(\tau) \subseteq \{\vec{\alpha}, \vec{\alpha}'\}$ and $\mathit{ftv}(\vec{\tau}') \subseteq \{\vec{\alpha}\}$ (so that $\mathit{ftv}(\tau[\vec{\tau}'/\vec{\alpha}']) \subseteq \{\vec{\alpha}\}$), then for all $\vec{r}$

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

Proof. This follows immediately from the definition of Δ in Fig. 8, by induction on the structure of τ. □

We can now complete the proof of the fundamental property of the logical relation of Definition 4.5.

Proof of Proposition 4.6 To prove that the relation (21) is compatible, we have to verify that it is closed under each of the axioms and rules in Fig. 4 corresponding to the various term-forming constructs of the language. The compatibility axiom for variables is trivially satisfied because of the way (21) is defined. The compatibility rule for function abstraction follows from Lemma 4.7(i) and Lemma 4.11. The compatibility rule for function application follows directly from clause (16) in Fig. 8. The compatibility rule for type generalisation follows from Lemma 4.8(i) and Lemma 4.11. The compatibility rule for type specialisation follows from clause (17) in Fig. 8 together with Lemmas 4.11 and 4.12. The compatibility rule for fixpoint terms follows from from Theorem 3.11 together with clause (16) in Fig. 8 and Lemma 4.11. The compatibility rules for list values follow from Lemma 4.9(i) and Lemma 4.11. Finally, the compatibility rule for case expressions follows from Lemmas 4.10(ii), 4.11 and 4.12.

So (21) is compatible. It is also closed under the two substitutivity rules in Fig. 5. The first one follows from the substitution property of Lemma 4.12. The second one follows immediately from the way (21) is defined from the functions $\vec{r} \mapsto \Delta_{\tau}(\vec{r}/\vec{\alpha})$ via closing substitutions. □

Note that since Δ is compatible, it is necessarily a reflexive relation and hence in particular we have:

Corollary 4.13. For all $\tau \in \mathit{Typ}$ and $M \in \mathit{Term}(\tau)$ it is the case that $(M, M) \in \Delta_{\tau}()$.

We need the corresponding property for frame stacks.

Lemma 4.14. For all $\tau \in \mathit{Typ}$ and all $S \in \mathit{Stack}(\tau)$, it is the case that $(S, S) \in (\Delta_{\tau}())^{\top}$.

Proof. This follows by induction on the length of the frame stack S. The base case uses Lemma 4.9(ii); the induction steps use Lemmas 4.7, 4.8, and 4.10. □

Theorem 4.15. The logical relation (21) coincides with PolyPCF observational congruence:

$$\Gamma \vdash M =_{\mathrm{obs}} M' : \tau \;\Leftrightarrow\; \Gamma \vdash M \,\Delta\, M' : \tau. \tag{26}$$

Proof. Using the substitutivity properties of $=_{\mathrm{obs}}$ (which it possesses by definition) and of Δ (which it possesses by Proposition 4.6), Corollary 3.15 and Lemma 4.11 combine to give

$$\Gamma \vdash M_1 =_{\mathrm{obs}} M_2 : \tau \;\;\&\;\; \Gamma \vdash M_2 \,\Delta\, M_3 : \tau \;\;\&\;\; \Gamma \vdash M_3 =_{\mathrm{obs}} M_4 : \tau' \;\Rightarrow \\ \Gamma \vdash M_1 \,\Delta\, M_4 : \tau. \tag{27}$$

Since $=_{\mathrm{obs}}$ and Δ are both compatible (the former by definition, the latter by Proposition 4.6), they are both reflexive and so we can take $M_1 = M_3 = M_3 = M$ and $M_4 = M'$ in (27) to deduce the left-to-right implication in (26).

For the converse implication, in view of Proposition 4.6 it just suffices to show that Δ is adequate, in the sense of Definition 2.2(iii). For then Δ is adequate, compatible and substitutive and hence is contained in the largest such relation, which by definition is $=_{\text{obs}}$. But if τ is any closed type and $\emptyset \vdash L \mathrel{\Delta} L' : \tau\mathit{list}$, i.e. $(L, L') \in \Delta_{\tau\mathit{list}}()$, then by Lemma 4.9(ii) we have $\mathit{Id} \top L \Leftrightarrow \mathit{Id} \top L'$ and therefore $L \Downarrow \mathbf{nil}_\tau \;\Leftrightarrow L' \Downarrow \mathbf{nil}_\tau$ by Theorem 3.6. □

Theorem 4.15 allows one to prove many properties of PolyPCF observational congruence to do with extensionality and relational parametricity; we pursue some of these properties in the next two sections. It also allows us to deduce some basic instances of observational congruence via the following result. Recall the notion of Kleene equivalence, $=_{\text{kl}}$, from Definition 3.12.

Corollary 4.16. For all $\tau \in \mathit{Typ}$ and $M, M' \in \mathit{Term}(\tau)$, if $M =_{\text{kl}} M' : \tau$, then $M =_{\text{obs}} M' : \tau$.

Proof. By Theorem 4.15 we have that $(-) =_{\text{obs}} (-) : \tau$ coincides with $\Delta_\tau()$. Therefore it is closed under composition with $=_{\text{kl}}$ by Corollary 3.15 and Lemma 4.11. Thus since $=_{\text{obs}}$ is reflexive we have

$$\begin{aligned} M =_{\text{kl}} M' : \tau \;&\Rightarrow\; M =_{\text{kl}} M' : \tau \;\;\&\;\; M' =_{\text{obs}} M' : \tau \\ &\Rightarrow\; M =_{\text{obs}} M' : \tau. \end{aligned}$$

□

Thus the redex-reduct pairs (R, R') mentioned in the last clause of the definition of $\to$ in the proof of Theorem 3.6 are all instances of observational congruence (cf. Example 3.13):

$$(\lambda\, x : \tau_1\, (M))\, A \;=_{\text{obs}}\; M[A/x] : \tau_2 \tag{28}$$

$$(\Lambda\, \alpha\, (M))\, \tau_2 \;=_{\text{obs}}\; M[\tau_2/\alpha] : \tau_1[\tau_2/\alpha] \tag{29}$$

$$\mathbf{case}\, \mathbf{nil}_{\tau_1}\, \mathbf{of}\, \{\mathbf{nil} \Rightarrow M_1 \mid h :: t \Rightarrow M_2\} \;=_{\text{obs}}\; M_1 : \tau_2 \tag{30}$$

$$\mathbf{case}\, H :: T\, \mathbf{of}\, \{\mathbf{nil} \Rightarrow M_1 \mid h :: t \Rightarrow M_2\} \;=_{\text{obs}}\; M_2[H/h, T/t] : \tau_2 \tag{31}$$

$$\mathbf{fix}(F) \;=_{\text{obs}}\; F\, \mathbf{fix}(F) : \tau. \tag{32}$$

The following characterisation of $=_{\text{obs}}$ will be useful in Section 6.

Corollary 4.17. Given any closed type $\tau \in \mathit{Typ}$ and closed terms $M, M' \in \mathit{Term}(\tau)$, write $M =_{\text{ciu}} M' : \tau$ to mean

$$\forall\, S \in \mathit{Stack}(\tau)\, (S \top M \;\Leftrightarrow\; S \top M').$$

(This is the ‘uses’ part of the notion of ‘closed instantiations of uses’ (ciu) equivalence of Mason and Talcott (1991).) Then

$$M =_{\text{ciu}} M' : \tau \;\Leftrightarrow M =_{\text{obs}} M' : \tau.$$

Proof. The fact that $=_{\text{obs}}$ is contained in $=_{\text{ciu}}$ follows (via Theorem 3.6) from the fact that $=_{\text{obs}}$ is, by definition, an adequate PolyPCF congruence relation. For the converse implication, by Theorem 4.15, it suffices to show that $=_{\text{ciu}}$ is contained in $\Delta_\tau()$. But it is evident from the definition of $=_{\text{ciu}}$ that any $\top\top$-closed term-relation, and hence in particular $\Delta_\tau()$, is closed under composition with $=_{\mathrm{ciu}}$. So now we can argue as in the proof of Corollary 4.16 (using the evident fact that $=_{\mathrm{ciu}}$ is reflexive) to deduce that $=_{\mathrm{ciu}}$ implies $=_{\mathrm{obs}}$. □

5. Extensionality properties of observational congruence

In this section we use Theorem 4.15 to establish various extensionality principles for PolyPCF observational congruence. Together, these generalise the 'context lemma' of Milner (1977) from PCF to PolyPCF. The first result reduces observational congruence for open terms to that for closed terms of closed type. Then restricting attention to closed terms, we give a separate extensionality principle for each of the three ways of forming PolyPCF types—functions, $\forall$-types, and list-types.

Theorem 5.1. Given $\Gamma \vdash M : \tau$ and $\Gamma \vdash M' : \tau$, with $\Gamma = \alpha_1, \dots, \alpha_m, x_1 : \tau_1, \dots, x_n : \tau_n$ say, then $\Gamma \vdash M =_{\mathrm{obs}} M' : \tau$ if and only if for all $\sigma_i \in \mathit{Typ}$ $(i = 1..m)$ and all $N_j \in \mathit{Term}(\tau_j[\vec{\sigma}/\vec{\alpha}])$ $(j = 1..n)$, it is the case that

$$M[\vec{\sigma}/\vec{\alpha}, \vec{N}/\vec{x}] =_{\mathrm{obs}} M'[\vec{\sigma}/\vec{\alpha}, \vec{N}/\vec{x}] : \tau[\vec{\sigma}/\vec{\alpha}]. \tag{33}$$

Proof. The 'only if' direction holds because by definition $=_{\mathrm{obs}}$ satisfies the substitutivity properties of Fig. 5 (and because it is reflexive).

For the 'if' direction, note that by Theorem 4.15 Δ is reflexive and in particular $\Gamma \vdash M' \;\Delta\; M' : \tau$ holds. Thus by Definition 4.5, for all $\top\top$-closed $r_i \in \mathit{Rel}(\sigma_i, \sigma_i')$ $(i = 1..m)$ and all $(N_j, N_j') \in \Delta_{\tau_j}(\vec{r}/\vec{\alpha})$ $(j = 1..n)$ we have $(M'[\vec{\sigma}/\vec{\alpha}, \vec{N}/\vec{x}], M'[\vec{\sigma}'/\vec{\alpha}, \vec{N}'/\vec{x}]) \in \Delta_\tau(\vec{r}/\vec{\alpha})$. Since $\Delta_\tau(\vec{r}/\vec{\alpha})$ is $\top\top$-closed (by Lemma 4.11), from this, (33), and Corollary 3.15 we conclude that $(M[\vec{\sigma}/\vec{\alpha}, \vec{N}/\vec{x}], M'[\vec{\sigma}'/\vec{\alpha}, \vec{N}'/\vec{x}]) \in \Delta_\tau(\vec{r}/\vec{\alpha})$. Thus by definition of the logical relation on open terms (Definition 4.5) we have $\Gamma \vdash M \;\Delta\; M' : \tau$ and hence $\Gamma \vdash M =_{\mathrm{obs}} M' : \tau$ by Theorem 4.15. □

Theorem 5.2 (Function type extensionality). For all $\tau_1 \to \tau_2 \in \mathit{Typ}$ and $F, F' \in \mathit{Term}(\tau_1 \to \tau_2)$

$$F =_{\mathrm{obs}} F' : \tau_1 \to \tau_2 \quad \Leftrightarrow \quad \forall\, A \in \mathit{Term}(\tau_1)\, (F\, A =_{\mathrm{obs}} F'\, A : \tau_2). \tag{34}$$

Proof.

$$\begin{aligned}
F =_{\mathrm{obs}} F' : \tau_1 \to \tau_2 &\Leftrightarrow (F, F') \in \Delta_{\tau_1 \to \tau_2}() && \text{by Theorem 4.15}\\
&\Leftrightarrow \forall\,(A, A') \in \Delta_{\tau_1}()\,((F\,A, F'\,A') \in \Delta_{\tau_2}()) && \text{by definition of } \Delta_{\tau_1 \to \tau_2}()\\
&\Leftrightarrow \forall\, A, A' \in \mathit{Term}(\tau_1)\,(A =_{\mathrm{obs}} A' : \tau_1 \Rightarrow \\
&\qquad\qquad F\,A =_{\mathrm{obs}} F'\,A' : \tau_2) && \text{by Theorem 4.15.}
\end{aligned}$$

Since this holds for all F, F' and since $=_{\mathrm{obs}}$ is reflexive and transitive, (34) follows. □

Example 5.3. Consider the 'polymorphic bottom' $\Omega \in \mathit{Term}(\forall\,\alpha\,(\alpha))$ introduced in Example 2.6:

$$\Omega \overset{\text{def}}{=} \Lambda\,\alpha\,(\mathbf{fix}(\lambda\, x : \alpha\,(x))).$$

For any $\tau_1, \tau_2 \in \mathit{Typ}$ and $A \in \mathit{Term}(\tau_1)$, evaluation of both $(\lambda\, x : \tau_1\, (\Omega\, \tau_2))\, A$ and $\Omega\, (\tau_1 \to \tau_2)\, A$ diverges, so

$$(\lambda\, x : \tau_1\, (\Omega\, \tau_2))\, A \;=_{\mathrm{kl}}\; \Omega\, (\tau_1 \to \tau_2)\, A.$$

Hence by Corollary 4.16

$$\forall\, A \in \mathit{Term}(\tau_1)\, ((\lambda\, x : \tau_1\, (\Omega\, \tau_2))\, A =_{\mathrm{obs}} \Omega\, (\tau_1 \to \tau_2)\, A : \tau_2)$$

and so by the above theorem, $\lambda\, x : \tau_1\, (\Omega\, \tau_2) =_{\mathrm{obs}} \Omega\, (\tau_1 \to \tau_2) : \tau_1 \to \tau_2$.

Theorem 5.4 ($\forall$-type extensionality). For all $\forall\, \alpha\, (\tau) \in \mathit{Typ}$ and $G, G' \in \mathit{Term}(\forall\, \alpha\, (\tau))$

$$G =_{\mathrm{obs}} G' : \forall\, \alpha\, (\tau) \quad \Leftrightarrow \quad \forall\, \tau' \in \mathit{Typ}\, (G\, \tau' =_{\mathrm{obs}} G'\, \tau' : \tau[\tau'/\alpha]).$$

Proof. The left-to-right implication follows from the fact that $=_{\mathrm{obs}}$ is a PolyPCF congruence (Definition 2.2(ii)). Conversely, suppose

$$\forall\, \tau' \in \mathit{Typ}\, (G\, \tau' =_{\mathrm{obs}} G'\, \tau' : \tau[\tau'/\alpha]) \tag{35}$$

holds. To show that $G =_{\mathrm{obs}} G' : \forall\, \alpha\, (\tau)$, by Theorem 4.15 it suffices to prove that $(G, G') \in \Delta_{\forall\, \alpha\, (\tau)}()$, i.e. that $(G\, \tau_1, G'\, \tau_2) \in \Delta_\tau(r^{\top\top}/\alpha)$ for all $r \in \mathit{Rel}(\tau_1, \tau_2)$ and $\tau_1, \tau_2 \in \mathit{Typ}$. But since $\Delta_{\forall\, \alpha\, (\tau)}()$ is reflexive (Corollary 4.13) we have $(G, G) \in \Delta_{\forall\, \alpha\, (\tau)}()$ and hence for any r, $(G\, \tau_1, G\, \tau_2) \in \Delta_\tau(r^{\top\top}/\alpha)$. Thus by (35) we have

$$(G\, \tau_1, G\, \tau_2) \in \Delta_\tau(r^{\top\top}/\alpha) \;\;\&\;\; G\, \tau_2 =_{\mathrm{obs}} G'\, \tau_2 : \tau[\tau_2/\alpha]$$

and hence by Lemma 4.11 and Corollary 3.15 that $(G\, \tau_1, G'\, \tau_2) \in \Delta_\tau(r^{\top\top}/\alpha)$, as required for $(G, G') \in \Delta_{\forall\, \alpha\, (\tau)}()$. □

Example 5.5. Let Ω be as in Example 5.3. For any closed $\forall$-type $\forall\, \alpha\, (\tau)$ and any $\tau' \in \mathit{Typ}$, evaluation of both $(\Lambda\, \alpha\, (\Omega\, \tau))\, \tau'$ and $\Omega\, (\forall\, \alpha\, (\tau))\, \tau'$ diverges and so

$$(\Lambda\, \alpha\, (\Omega\, \tau))\, \tau' \;=_{\mathrm{kl}}\; \Omega\, (\forall\, \alpha\, (\tau))\, \tau.'$$

Hence by Corollary 4.16 and the above theorem, it is the case that $\Lambda\, \alpha\, (\Omega\, \tau) =_{\mathrm{obs}} \Omega\, (\forall\, \alpha\, (\tau)) : \forall\, \alpha\, (\tau)$.

The extensionality principle for list types is more complicated than the above principles for function- and $\forall$-types. We will recover the characterisation of observational congruence of lazy lists in terms of a notion of *bisimilarity* to be found, for example, in (Gordon 1995) or (Pitts 1997a). Recall from Section 4 that for a closed type $\tau \in \mathit{Typ}$, the term-relation $\Delta_{\tau\,\mathit{list}}()$ is $(\Delta_\tau())\mathit{list}$ and hence is given by the following greatest fixed point.

$$\Delta_{\tau\,\mathit{list}}() \;=\; \nu\, r\, (1 + (\Delta_\tau() \times r))^{\top\top}.$$

We will show that this coincides with a different greatest fixed point, which is defined in terms of PolyPCF evaluation, $(-) \Downarrow (-)$ and which aids calculations.

Definition 5.6. Given $\tau, \tau' \in \mathit{Typ}$ and $r \in \mathit{Rel}(\tau, \tau')$, call a term-relation $s \in \mathit{Rel}(\tau\,\mathit{list}, \tau'\,\mathit{list})$ an *r-simulation* if it satisfies that whenever $(L, L') \in s$ then

$$\begin{aligned} L \Downarrow \mathbf{nil}_\tau \;&\Rightarrow\; L' \Downarrow \mathbf{nil}_{\tau'} \\ L \Downarrow H :: T \;&\Rightarrow\; \exists\, H', T'\, (L' \Downarrow H' :: T' \;\;\&\;\; (H, H') \in r \;\;\&\;\; (T, T') \in s). \end{aligned}$$

We say that s is an *r-bisimulation* if both it and its reciprocal $s^{op} = \{(L', L) \mid (L, L') \in s\}$ are r-simulations.

Proposition 5.7. If $r \in \mathit{Rel}(\tau, \tau')$ is $\top\top$-closed, then the term-relation $(r)\mathit{list} \in \mathit{Rel}(\tau\,\mathit{list}, \tau'\,\mathit{list})$ of Definition 4.3 is the greatest r-bisimulation.

Proof. Writing

$$\begin{aligned}\Phi_r(s) &\overset{\text{def}}{=} \{(L, L') \mid \quad (L \Downarrow \mathbf{nil}_\tau \Rightarrow L' \Downarrow \mathbf{nil}_{\tau'})\ \&\ \forall H, T\, (L \Downarrow H :: T \Rightarrow \\ &\qquad\qquad \exists\, H', T'\, (L' \Downarrow H' :: T'\ \&\ (H, H') \in r\ \&\ (T, T') \in s)\} \\ \Psi_r(s) &\overset{\text{def}}{=} \Phi_r(s) \cap (\Phi_r(s^{op}))^{op}\end{aligned}$$

we have to prove that $\nu\, s\, \Psi_r(s) = \nu\, s\, (1 + (r \times s))^{\top\top}$. This can be achieved via the following lemmas,which hold for any term-relations $r \in \mathit{Rel}(\tau, \tau')$ and $s \in \mathit{Rel}(\tau\,\mathit{list}, \tau'\,\mathit{list})$.

(i) $1 + (r \times s) \subseteq \Psi_r(s) \subseteq (1 + (r \times s))^{\top\top}$.
(ii) If r and s are $\top\top$-closed, then so is $\Psi_r(s)$.

The proof of (i) is straightforward. For (ii) we first prove

$$(\mathit{Id}, \mathit{Id}) \in \Psi_r(s)^\top \tag{36}$$

$$(\mathit{IsCons}_\tau, \mathit{IsCons}_{\tau'}) \in \Psi_r(s)^\top \tag{37}$$

$$(S, S') \in r^\top \Rightarrow (S \circ \mathit{Head}_\tau, S' \circ \mathit{Head}_{\tau'}) \in \Psi_r(s)^\top \tag{38}$$

$$(S, S') \in s^\top \Rightarrow (S \circ \mathit{Tail}_\tau, S' \circ \mathit{Tail}_{\tau'}) \in \Psi_r(s)^\top \tag{39}$$

where

$$\begin{aligned}\mathit{IsCons}_\tau &\overset{\text{def}}{=} \mathit{Id} \circ \mathbf{case}-\mathbf{of}\,\{\mathbf{nil} \Rightarrow \Omega\,\tau\mathit{list} \mid h :: t \Rightarrow \mathbf{nil}_\tau\} \\ \mathit{Head}_\tau &\overset{\text{def}}{=} \mathbf{case}-\mathbf{of}\,\{\mathbf{nil} \Rightarrow \Omega\,\tau \mid h :: t \Rightarrow h\} \\ \mathit{Tail}_\tau &\overset{\text{def}}{=} \mathbf{case}-\mathbf{of}\,\{\mathbf{nil} \Rightarrow \Omega\,\tau \mid h :: t \Rightarrow t\}\end{aligned}$$

and Ω is as in Example 2.6. From (38) we get

$$(L, L') \in \Psi_r(s)^{\top\top} \Rightarrow (\mathit{Head}_\tau[L], \mathit{Head}_{\tau'}[L']) \in r^{\top\top} \tag{40}$$

and similarly from (39)

$$(L, L') \in \Psi_r(s)^{\top\top} \Rightarrow (\mathit{Tail}_\tau[L], \mathit{Tail}_{\tau'}[L']) \in s^{\top\top} \tag{41}$$

(where we use the notation $-[-]$ from Definition 3.4 for applying a frame to a term). Then (36), (37), (40) and (41) can be used to prove (ii), making use of Corollary 3.15 applied to the Kleene equivalences

$$\left.\begin{aligned}\mathit{Head}_\tau[L] &=_{\text{kl}} H \\ \mathit{Tail}_\tau[L] &=_{\text{kl}} T\end{aligned}\right\} \quad \text{if } L \Downarrow H :: T.$$

Now if s is an r-bisimulation, i.e. if $s \subseteq \Psi_r(s)$, then by (i), $s \subseteq (1 + (r \times s))^{\top\top}$ and thus $s \subseteq (r)\mathit{list}$, by definition of $(r)\mathit{list}$. So to see that $(r)\mathit{list}$ is the greatest r-bisimulation, it just remains to check that it is itself an r-bisimulation. It is only now that we use the assumption that

r is $\top\top$-closed. Note that by (13), $(r)\mathit{list}$ is also $\top\top$-closed; so by (ii), $\Psi_r((r)\mathit{list})$ is $\top\top$-closed and hence

$$\begin{aligned}\Psi_r((r)\mathit{list}) &= \Psi_r((r)\mathit{list})^{\top\top} \\ &\supseteq (1 + (r \times (r)\mathit{list}))^{\top\top} && \text{by (i)} \\ &= (r)\mathit{list}\end{aligned}$$

as required. □

Theorem 5.8 (List type extensionality). For all $\tau \in \mathit{Typ}$ and $L, L' \in \mathit{Term}(\tau\mathit{list})$, $L =_{\mathrm{obs}} L' : \tau\mathit{list}$ if and only if $(L, L') \in s$ for some $s \in \mathit{Rel}(\tau\mathit{list}, \tau\mathit{list})$ satisfying that whenever $(M, M') \in s$ then

— $M \Downarrow \mathbf{nil}_\tau$ if and only if $M' \Downarrow \mathbf{nil}_\tau$
— if $M \Downarrow H :: T$, then $M' \Downarrow H' :: T'$ for some H' and T' with $H =_{\mathrm{obs}} H' : \tau$ and $(T, T') \in s$
— if $M' \Downarrow H' :: T'$, then $M \Downarrow H :: T$ for some H and T with $H =_{\mathrm{obs}} H' : \tau$ and $(T, T') \in s$.

Proof. Since by Theorem 4.15 $(-) =_{\mathrm{obs}} (-) : \tau$ coincides with $\Delta_\tau()$, a term-relation $s \in \mathit{Rel}(\tau\mathit{list}, \tau\mathit{list})$ with the above property is precisely a $\Delta_\tau()$-bisimulation (Definition 5.6). Thus L and L' are related by some such term-relation if and only if they are related by the greatest $\Delta_\tau()$-bisimulation, which by the previous proposition is $(\Delta_\tau())\mathit{list}$. This is by definition $\Delta_{\tau\mathit{list}}()$, and by Theorem 4.15 again, this is $(-) =_{\mathrm{obs}} (-) : \tau\mathit{list}$. □

This theorem provides a coinduction principle for PolyPCF list types which can be used to prove properties of lazy lists like those considered in (Pitts 1997a, Section 3) for example. We use it in the next section to prove Example 2.8 concerning polymorphic versus inductive list types.

6. Examples: null, unit and list types

In this section we use Theorem 4.15 to give proofs of the properties of null, unit, and list types claimed in Examples 2.6–2.8.

Null Type

Proof of Example 2.6. Suppose G is a closed term of type $\mathit{null} \stackrel{\text{def}}{=} \forall\,\alpha\,(\alpha)$. We have to show that $G =_{\mathrm{obs}} \Omega : \mathit{null}$, where $\Omega \stackrel{\text{def}}{=} \Lambda\,\alpha\,(\mathbf{fix}(\lambda\,x : \alpha\,(x)))$. By Theorem 5.4 and (29), it suffices to show for all $\tau \in \mathit{Typ}$ that $G\,\tau =_{\mathrm{obs}} \mathbf{fix}(\lambda\,x : \tau\,(x)) : \tau$. For this, by Corollary 4.17 it suffices to show for all $S \in \mathit{Stack}(\tau)$ that $S \top (G\,\tau)$ does not hold, because evaluation of $\mathbf{fix}(\lambda\,x : \tau\,(x))$ does not converge.

From Corollary 4.13 we have $(G, G) \in \Delta_{\forall\,\alpha\,(\alpha)}() = \forall\,r\,(r^{\top\top})$. In other words, for all $\tau, \tau' \in \mathit{Typ}$ and $r \in \mathit{Rel}(\tau, \tau')$ we have

$$(G\,\tau, G\,\tau') \in r^{\top\top}. \tag{42}$$

Given τ, we use (42) with $\tau' = \tau\mathit{list}$ and r the one-element term-relation

$$r \stackrel{\text{def}}{=} \{(\Omega\,\tau\,,\, \Omega\,(\tau\mathit{list}))\}.$$

For any $S \in \mathit{Stack}(\tau)$, let $S' \in \mathit{Stack}(\tau\,\mathit{list})$ be a frame stack that diverges when applied to any term of type $\tau\,\mathit{list}$, say

$$S' \overset{\text{def}}{=} \mathit{Id} \circ (\mathbf{case}\ -\ \mathbf{of}\,\{\mathbf{nil} \Rightarrow \Omega\,(\tau\,\mathit{list}) \mid h :: t \Rightarrow \Omega\,(\tau\,\mathit{list})\}).$$

Now neither $S \top \Omega\,\tau$ nor $S' \top \Omega\,(\tau\,\mathit{list})$ hold, because of the divergence properties of Ω. Therefore by definition of r, we have $(S, S') \in r^{\top}$. Combining this with (42) yields $S \top (G\,\tau) \Leftrightarrow S' \top (G\,\tau\,\mathit{list})$. But S' was chosen so that $S' \top L$ does not hold for any $L \in \mathit{Term}(\tau\,\mathit{list})$. Therefore $S \top (G\,\tau)$ does not hold either, as required. □

In order to prove Examples 2.7 and 2.8 we use the following source of $\top\top$-closed term-relations.

Lemma 6.1 (Graphs of frame stacks are $\top\top$-closed). For all $\tau, \tau' \in \mathit{Typ}$, suppose $S \in \mathit{Stack}(\tau, \tau')$ is a frame stack with argument type τ and result type τ'. Then the term-relation $\mathit{graph}_S \in \mathit{Rel}(\tau, \tau')$ defined by

$$\mathit{graph}_S \overset{\text{def}}{=} \{(M, M') \mid S\,M =_{\text{obs}} M' : \tau'\}.$$

is $\top\top$-closed. (The application operation $S, M \mapsto S\,M$ was given in Definition 3.4.)

Proof. We have to show that $(\mathit{graph}_S)^{\top\top} \subseteq \mathit{graph}_S$. Note that by Theorem 4.15

$$\mathit{graph}_S = \{(M, M') \mid (S\,M, M') \in \Delta_{\tau'}()\}. \tag{43}$$

Let $S'@S$ denote the result of appending the frames in S to a frame stack S' (as defined in the proof of Theorem 3.6). Then an induction on the length of S yields

$$(S'@S) \top M \Leftrightarrow S' \top (S\,M). \tag{44}$$

From (43) and (44) we get

$$(S', S'') \in (\Delta_{\tau'}())^{\top} \Rightarrow (S'@S, S'') \in (\mathit{graph}_S)^{\top}$$

and hence that

$$(N, N') \in (\mathit{graph}_S)^{\top\top} \Rightarrow (S\,N, N') \in (\Delta_{\tau'}())^{\top\top}.$$

But by Lemma 4.11, $\Delta_{\tau'}()$ is $\top\top$-closed. Therefore if $(N, N') \in (\mathit{graph}_S)^{\top\top}$, then $(S\,N, N') \in \Delta_{\tau'}()$ and hence $(N, N') \in \mathit{graph}_S$, as required. □

Unit Type

Proof of Example 2.7. Suppose G is a closed term of type $\mathit{unit} \overset{\text{def}}{=} \forall\,\alpha\,(\alpha \to \alpha)$. Combining Theorems 5.2 and 5.4 with Examples 5.3 and 5.5, and with the beta-conversions (28) and (29), to establish the claim in this example it suffices to show for all $\tau \in \mathit{Typ}$ and $M \in \mathit{Term}(\tau)$ that either

$$G\,\tau\,M =_{\text{obs}} \Omega\,\tau : \tau \tag{45}$$

or

$$G\,\tau\,M =_{\text{obs}} M : \tau. \tag{46}$$

Given $\tau \in \mathit{Typ}$ and $M \in \mathit{Term}(\tau)$, let $S \in \mathit{Stack}(\mathit{unitlist}, \tau)$ be the frame stack

$$S \overset{\text{def}}{=} \mathit{Id} \circ (\textbf{case} - \textbf{of}\, \{\textbf{nil} \Rightarrow M \mid h :: t \Rightarrow M\})$$

and consider $\mathit{graph}_S \in \mathit{Rel}(\mathit{unitlist}, \tau)$ as in Lemma 6.1. By Corollary 4.13 we have $(G, G) \in \Delta_{\mathit{unit}}() = \forall\, r\, (r^{\top\top} \to r^{\top\top})$. So since by Lemma 6.1 graph_S is $\top\top$-closed, we have

$$(G\, \mathit{unitlist}, G\, \tau) \in \mathit{graph}_S \to \mathit{graph}_S. \tag{47}$$

Using the beta-conversion (30), we have $(\textbf{nil}_{\mathit{unit}}, M) \in \mathit{graph}_S$. Therefore from (47) we get that $(G\, \mathit{unitlist}\, \textbf{nil}_{\mathit{unit}}, G\, \tau\, M) \in \mathit{graph}_S$, i.e. that

$$\textbf{case}\, (G\, \mathit{unitlist}\, \textbf{nil}_{\mathit{unit}})\, \textbf{of}\, \{\textbf{nil} \Rightarrow M \mid h :: t \Rightarrow M\} \;=_{\text{obs}}\; G\, \tau\, M : \tau. \tag{48}$$

Now either $G\, \mathit{unitlist}\, \textbf{nil}_{\mathit{unit}} \Downarrow V$ for some V, or not. In the first case we get

$$\textbf{case}\, (G\, \mathit{unitlist}\, \textbf{nil}_{\mathit{unit}})\, \textbf{of}\, \{\textbf{nil} \Rightarrow M \mid h :: t \Rightarrow M\} \;=_{\text{kl}}\; M : \tau$$

and in the second we get

$$\textbf{case}\, (G\, \mathit{unitlist}\, \textbf{nil}_{\mathit{unit}})\, \textbf{of}\, \{\textbf{nil} \Rightarrow M \mid h :: t \Rightarrow M\} \;=_{\text{kl}}\; \Omega\, \tau : \tau.$$

Then by Corollary 4.16 and (48), the first possibility yields (46), whereas the second yields (45). □

List Types

Proof of Example 2.8. Let $L(\alpha)$, I and J be as defined in Example 2.8. By Theorem 5.1, to prove

$$\alpha,\, \ell : \alpha\mathit{list} \vdash J\, \alpha\, (I\, \alpha\, \ell) =_{\text{obs}} \ell : \alpha\mathit{list}$$
$$\alpha,\, g : L(\alpha) \vdash I\, \alpha\, (J\, \alpha\, g) =_{\text{obs}} g : L(\alpha)$$

it suffices to show for all $\tau \in \mathit{Typ}$, $L \in \mathit{Term}(\tau\mathit{list})$, and $G \in \mathit{Term}(L(\tau))$ that

$$J\, \tau\, (I\, \tau\, L) \;=_{\text{obs}}\; L : \tau\mathit{list} \tag{49}$$

and

$$I\, \tau\, (J\, \tau\, G) \;=_{\text{obs}}\; G : L(\tau).$$

For the latter, in view of the definition of $L(\tau)$ it suffices to show for all $\tau' \in \mathit{Typ}$, $M' \in \mathit{Term}(\tau')$, and $F \in \mathit{Term}(\tau \to \tau' \to \tau')$ that

$$I\, \tau\, (J\, \tau\, G)\, \tau'\, M'\, F \;=_{\text{obs}}\; G\, \tau'\, M'\, F : \tau'. \tag{50}$$

We tackle (49) first. Applying the beta-conversion properties (28), (29), and (32) to the definitions of I and J yields

$$I\, \tau\, L\, \tau'\, M'\, F \;=_{\text{obs}}\; \textbf{case}\, L\, \textbf{of}\, \{\textbf{nil} \Rightarrow M' \mid h :: t \Rightarrow F\, h\, (I\, \tau\, t\, \tau'\, M'\, F)\} : \tau\mathit{list} \tag{51}$$

(for all L, M', and F of appropriate type) and then

$$J\, \tau\, (I\, \tau\, L) \;=_{\text{obs}}\; \textbf{case}\, L\, \textbf{of}\, \{\textbf{nil} \Rightarrow \textbf{nil}_\tau \mid h :: t \Rightarrow h :: (J\, \tau\, (I\, \tau\, t))\} : \tau\mathit{list}. \tag{52}$$

From (52) it follows that $s \overset{\text{def}}{=} \{(L, L') \mid L =_{\text{obs}} J\,\tau\,(I\,\tau\,L') : \tau\mathit{list}\}$ satisfies the bisimulation conditions in Theorem 5.8 and hence by that theorem it is contained in $=_{\text{obs}}$. Since $(J\,\tau\,(I\,\tau\,L), L) \in s$, we have (49).

Turning to the proof of (50), consider the frame stack $S \in \mathit{Stack}(\tau\mathit{list}, \tau')$ defined by

$$S \overset{\text{def}}{=} \mathit{Id} \circ (\mathbf{case} - \mathbf{of}\,\{\mathbf{nil} \Rightarrow M' \mid h :: t \Rightarrow (F\,h)\,(I\,\tau\,t\,\tau'\,M'\,F)\}).$$

In view of (51), we have $S\,L =_{\text{obs}} I\,\tau\,L\,\tau'\,M'\,F : \tau\mathit{list}$ and therefore

$$r_{M',F} \overset{\text{def}}{=} \{(L, M'') \mid I\,\tau\,L\,\tau'\,M'\,F =_{\text{obs}} M'' : \tau'\}$$

is a $\top\top$-closed member of $\mathit{Rel}(\tau\mathit{list}, \tau')$ by Lemma 6.1. So for each $G \in \mathit{Term}(L(\tau))$, since by Corollary 4.13

$$(G, G) \in \Delta_{L(\tau)}() = \forall\,r\,(r^{\top\top} \to (\Delta_\tau() \to r^{\top\top} \to r^{\top\top}) \to r^{\top\top})$$

we have that

$$(G\,\tau\mathit{list}, G\,\tau') \in r_{M',F} \to (\Delta_\tau() \to r_{M',F} \to r_{M',F}) \to r_{M',F}.$$

From (51) and the definition of $r_{M',F}$ we get that $N \overset{\text{def}}{=} \Lambda\,\alpha\,(\mathbf{nil}_\alpha)$ and $C \overset{\text{def}}{=} \Lambda\,\alpha\,(\lambda\,h : \alpha\,(\lambda\,t : \alpha\mathit{list}\,(h :: t)))$ satisfy

$$(N\,\tau, M') \in r_{M',F} \qquad \text{and} \qquad (C\,\tau, F) \in \Delta_\tau() \to r_{M',F} \to r_{M',F}$$

and hence

$$(G\,\tau\mathit{list}\,(N\,\tau)\,(C\,\tau)\,,\,G\,\tau'\,M'\,F) \in r_{M',F}.$$

So by definition of $r_{M',F}$ we have $I\,\tau(G\,\tau\mathit{list}\,(N\,\tau)\,(C\,\tau))\,\tau'\,M'\,F =_{\text{obs}} G\,\tau'\,M'\,F : \tau'$, from which (50) follows by definition of J. □

7. Example: existential types

Example 2.8 shows that in PolyPCF, inductive lists types are observationally isomorphic to polymorphic list types. In this section we give another example of this phenomenon by extending PolyPCF with existential types $\exists\,\alpha\,(\tau)$ (Mitchell and Plotkin 1988) equipped with a standard operational semantics, and proving that they are observationally isomorphic to the polymorphic types $\forall\,\alpha'\,(\forall\,\alpha\,((\tau \to \alpha') \to \alpha'))$ (where $\alpha' \notin \mathit{ftv}(\tau)$).

PolyPCF with Existential Types

Extend the grammar of PolyPCF (Fig. 1) with a type-former for existential types

$$\begin{array}{rcll} \tau & ::= & \cdots & \\ & \mid & \exists\,\alpha\,(\tau) & \exists\text{-type} \end{array}$$

and operations for constructing and deconstructing terms of such types

$$\begin{array}{rcll} M & ::= & \cdots & \\ & \mid & \mathbf{pack}\,\tau, M\,\mathbf{as}\,\exists\,\alpha\,(\tau) & \exists\text{-type constructor} \\ & \mid & \mathbf{open}\,M\,\mathbf{as}\,\alpha, x\,\mathbf{in}\,M & \exists\text{-type destructor.} \end{array}$$

The explicit typing information in **pack**ed terms is there to preserve the 'uniqueness of type' property mentioned in Note (iii) in Fig. 2. Free occurrences of α in τ become bound in the type $\exists\,\alpha\,(\tau)$; and free occurrences of α and x in M become bound in the term $\mathbf{open}\,E\,\mathbf{as}\,\alpha, x\,\mathbf{in}\,M$. The type assignment relation of Fig. 2 is extended by adding the rules

$$\frac{\Gamma \vdash M : \tau_1[\tau_2/\alpha]}{\Gamma \vdash \mathbf{pack}\,\tau_2, M\,\mathbf{as}\,\exists\,\alpha\,(\tau_1) : \exists\,\alpha\,(\tau_1)}$$

$$\frac{\Gamma \vdash E : \exists\,\alpha\,(\tau_1) \quad \Gamma, \alpha, x : \tau_1 \vdash M : \tau_2}{\Gamma \vdash \mathbf{open}\,E\,\mathbf{as}\,\alpha, x\,\mathbf{in}\,M : \tau_2}.$$

Recall that such rules are only applied to judgements that are well-formed in the sense of Note (i) in Fig. 2. In particular, in the second of the two rules given above, since the notation Γ, α means Γ *properly* extended by α, α does not occur free in Γ and hence not in τ_2 either (by well-formedness of the conclusion of the rule). This condition that α is not allowed to occur free in τ_2 in the second rule is what distinguishes an existential type from a type-indexed dependent sum, where there is free access both to the type component as well as the term component of a **pack**ed term: see (Mitchell and Plotkin 1988, p 474 *et seq*) for a discussion of this point.

We extend the evaluation relation of Fig. 3 by declaring that **pack**ed terms are values

$$\mathbf{pack}\,\tau, M\,\mathbf{as}\,\exists\,\alpha\,(\tau) \Downarrow \mathbf{pack}\,\tau, M\,\mathbf{as}\,\exists\,\alpha\,(\tau)$$

and adding the following rule for $\exists$-type destructors:

$$\frac{E \Downarrow \mathbf{pack}\,\tau', M'\,\mathbf{as}\,\exists\,\alpha\,(\tau) \quad M[\tau'/\alpha, M'/x] \Downarrow V}{\mathbf{open}\,E\,\mathbf{as}\,\alpha, x\,\mathbf{in}\,M \Downarrow V}.$$

The associated notion of *observational congruence* is just as in Theorem 2.3, though of course the compatibility properties in Fig. 4 have to be extended with obvious clauses for the two new term-forming operations.

Turning to the material in Section 3, we have to extend the notion of frame stack (Definition 3.2) by adding a new form of frame:

$$F \;::=\; \cdots \mid (\mathbf{open} - \mathbf{as}\,\alpha, x\,\mathbf{in}\,M)$$

with typing rule

$$\frac{\Gamma \vdash S : \tau' \multimap \tau'' \qquad \Gamma, \alpha, x : \tau \vdash M : \tau'}{\Gamma \vdash S \circ (\mathbf{open} - \mathbf{as}\,\alpha, x\,\mathbf{in}\,M) : \exists\,\alpha\,(\tau) \multimap \tau''}.$$

Then the results in that section continue to hold provided we extend the definition of $(-) \top (-)$ by adding the following two rules to those in Fig. 7

$$\frac{S = S' \circ (\mathbf{open} - \mathbf{as}\,\alpha, x\,\mathbf{in}\,M) \qquad S' \top M[\tau'/\alpha, M'/x]}{S \top \mathbf{pack}\,\tau', M'\,\mathbf{as}\,\exists\,\alpha\,(\tau)}$$

$$\frac{S \circ (\mathbf{open} - \mathbf{as}\,\alpha, x\,\mathbf{in}\,M) \top E}{S \top \mathbf{open}\,E\,\mathbf{as}\,\alpha, x\,\mathbf{in}\,M}$$

(or equivalently, by adding the single (redex,reduct)-pair (R, R') where

$$R = \mathbf{open}\,(\mathbf{pack}\,\tau', M'\,\mathbf{as}\,\exists\,\alpha\,(\tau))\,\mathbf{as}\,\alpha, x\,\mathbf{in}\,M \quad \text{and} \quad R' = M[\tau'/\alpha, M'/x]$$

to the definition of the transition relation in the proof of Theorem 3.6). In particular, we have the closure operation $(-)^{\top\top}$ on term-relations for the extended language, with its various properties.

Now we come to the delicate part. We extend the logical relation of Section 4 to the language with existential types by adding the following clause to Fig. 8.

$$\Delta_{\exists\,\alpha\,(\tau)}(\vec{r}/\vec{\alpha}) \overset{\text{def}}{=} (\exists\, r\,(\Delta_\tau(r^{\top\top}/\alpha, \vec{r}/\vec{\alpha})))^{\top\top}. \tag{53}$$

As well as the closure operation $(-)^{\top\top}$, this makes use of the following action of $\exists$ on term-relations.

Definition 7.1 (Action of $\exists$ on term-relations). Let τ_1 and τ_1' be PolyPCF types with at most a single free type variable, α say. Suppose R is a function mapping term-relations $r \in \mathit{Rel}(\tau_2, \tau_2')$ (any $\tau_2, \tau_2' \in \mathit{Typ}$) to term-relations $R(r) \in \mathit{Rel}(\tau_1[\tau_2/\alpha], \tau_1'[\tau_2'/\alpha])$. Then we can form a term-relation $\exists\, r\,(R(r)) \in \mathit{Rel}(\exists\,\alpha\,(\tau_1), \exists\,\alpha\,(\tau_1'))$ as follows.

$$\begin{aligned}\exists\, r\,(R(r)) \overset{\text{def}}{=} \{(&\mathbf{pack}\,\tau_2, M\,\mathbf{as}\,\exists\,\alpha\,(\tau_1), \mathbf{pack}\,\tau_2', M'\,\mathbf{as}\,\exists\,\alpha\,(\tau_1')) \mid \\ &\exists\, r \in \mathit{Rel}(\tau_2, \tau_2')\,((M, M') \in R(r))\}.\end{aligned}$$

Note that $\exists\, r\,(R(r))$ only contains values (i.e. **pack**ed terms). Thus it makes some sense to take the $\top\top$-closure of this construct in (53) when defining the logical relation at $\exists$-types. (The use of $r^{\top\top}/\alpha$, rather than r/α in (53) is for the same reasons as in clause (16): see Remark 4.4.) However, the precise justification for the definition is that it permits Proposition 4.6 (the Fundamental Property of the logical relation) to go through for the extended language. The compatibility properties of the extended logical relation with respect to $\exists$-type constructors and destructors follow from properties of Definition 7.1 with respect to $(-)^{\top\top}$ which are analogous to those for list types in Lemmas 4.9 and 4.10; we omit the details.

From the fundamental property for Δ we deduce that Theorem 4.15 (characterisation of $=_{\mathrm{obs}}$ in terms of the logical relation), Corollary 4.16 ($=_{\mathrm{kl}}$ implies $=_{\mathrm{obs}}$), Corollary 4.17 (coincidence of $=_{\mathrm{obs}}$ with $=_{\mathrm{ciu}}$) and the results of Section 5 go through for the extended language. Two simple consequences of these results which we will need in a moment are beta- and eta-conversions for $\exists$-types:

$$\Gamma \vdash \mathbf{open}\,(\mathbf{pack}\,\tau', M'\,\mathbf{as}\,\exists\,\alpha\,(\tau))\,\mathbf{as}\,\alpha, x\,\mathbf{in}\,M =_{\mathrm{obs}} M[\tau'/\alpha, M'/x] : \tau'' \tag{54}$$

$$\Gamma \vdash E =_{\mathrm{obs}} \mathbf{open}\,E\,\mathbf{as}\,\alpha, x\,\mathbf{in}\,(\mathbf{pack}\,\alpha, x\,\mathbf{as}\,\exists\,\alpha\,(\tau)) : \exists\,\alpha\,(\tau) \tag{55}$$

where

$$\Gamma \vdash M' : \tau[\tau'/\alpha] \qquad \Gamma, \alpha, x : \tau \vdash M : \tau'' \qquad \Gamma \vdash E : \exists\,\alpha\,(\tau).$$

These observational congruences hold because their closed instances are valid Kleene equivalences.

Definability of $\exists$-types

We aim to show that each existential type

$$\varepsilon \overset{\text{def}}{=} \exists\,\alpha\,(\tau)$$

is in bijection, up to $=_{\mathrm{obs}}$, with the polymorphic type

$$\varepsilon' \overset{\mathrm{def}}{=} \forall\,\alpha'\,(\forall\,\alpha\,(\tau \to \alpha') \to \alpha')$$

(where $\alpha' \notin \mathit{ftv}(\tau)$). For simplicity, we assume that $\exists\,\alpha\,(\tau)$ is closed. Define $I \in \mathit{Term}(\varepsilon \to \varepsilon')$ and $J \in \mathit{Term}(\varepsilon' \to \varepsilon)$ to be the terms

$$\begin{aligned} I &\overset{\mathrm{def}}{=} \lambda\,e : \varepsilon\,(\Lambda\,\alpha'\,(\lambda\,g : \forall\,\alpha\,(\tau \to \alpha')\,(\mathbf{open}\,e\,\mathbf{as}\,\alpha, x\,\mathbf{in}\,g\,\alpha\,x)))\\ J &\overset{\mathrm{def}}{=} \lambda\,g' : \varepsilon'\,(g'\,\varepsilon\,P),\\ \text{where}\quad P &\overset{\mathrm{def}}{=} \Lambda\,\alpha\,(\lambda\,x : \tau\,(\mathbf{pack}\,\alpha, x\,\mathbf{as}\,\varepsilon)) \in \mathit{Term}(\forall\,\alpha\,(\tau \to \varepsilon)). \end{aligned}$$

Theorem 7.2. ε is observationally isomorphic to ε' in the sense that $J \circ I \overset{\mathrm{def}}{=} \lambda\,e : \varepsilon\,(J\,(I\,e))$ and $I \circ J \overset{\mathrm{def}}{=} \lambda\,g : \varepsilon'\,(I\,(J\,g))$ are observationally congruent to the identity functions on ε and ε' respectively.

Proof. Using the various extensionality results of Section 5, it suffices to prove for all $E \in \mathit{Term}(\varepsilon)$ that

$$J\,(I\,E) \;=_{\mathrm{obs}}\; E : \varepsilon \tag{56}$$

and for all $G' \in \mathit{Term}(\varepsilon')$, $\tau' \in \mathit{Typ}$, and $G \in \mathit{Term}(\forall\,\alpha\,(\tau \to \tau'))$ that

$$I\,(J\,G')\,\tau'\,G \;=_{\mathrm{obs}}\; G'\,\tau'\,G : \tau'. \tag{57}$$

Using the definitions of I and J and applying the beta-conversions (28) and (29) several times we get

$$J\,(I\,E) \;=_{\mathrm{obs}}\; \mathbf{open}\,E\,\mathbf{as}\,\alpha, x\,\mathbf{in}\,(\mathbf{pack}\,\alpha, x\,\mathbf{as}\,\varepsilon) : \varepsilon$$

so that (56) holds because of the eta-conversion (55). Similarly, after various beta-conversions (57) is equivalent to

$$\mathbf{open}\,(G'\,\varepsilon\,P)\,\mathbf{as}\,\alpha, x\,\mathbf{in}\,G\,\alpha\,x \;=_{\mathrm{obs}}\; G'\,\tau'\,G : \tau'. \tag{58}$$

To see that this holds, consider the frame stack $S \in \mathit{Stack}(\varepsilon, \tau')$ given by

$$S \overset{\mathrm{def}}{=} \mathit{Id} \circ (\mathbf{open} - \mathbf{as}\,\alpha, x\,\mathbf{in}\,G\,\alpha\,x)$$

and the $\top\top$-closed term-relation $\mathit{graph}_S \in \mathit{Rel}(\varepsilon, \tau')$ associated with it as in Lemma 6.1. We claim that

$$(P, G) \in \forall\,r\,(\Delta_\tau(r^{\top\top}/\alpha) \to \mathit{graph}_S). \tag{59}$$

For, given any $\tau_2, \tau_2' \in \mathit{Typ}$, $r \in \mathit{Rel}(\tau_2, \tau_2')$, and $(M, M') \in \Delta_\tau(r^{\top\top}/\alpha)$, we have

$$\begin{aligned} ((\mathbf{pack}\,\tau_2, M\,\mathbf{as}\,\varepsilon), (\mathbf{pack}\,\tau_2', M'\,\mathbf{as}\,\varepsilon)) &\in \exists\,r\,(\Delta_\tau(r^{\top\top}/\alpha)) && \text{by Definition 7.1}\\ &\subseteq (\exists\,r\,(\Delta_\tau(r^{\top\top}/\alpha)))^{\top\top} &&\\ &\overset{\mathrm{def}}{=} \Delta_\varepsilon() && \text{since } \varepsilon \overset{\mathrm{def}}{=} \exists\,\alpha\,(\tau). \end{aligned}$$

Hence by Theorem 4.15 (for the extended language)

$$(\mathbf{pack}\,\tau_2, M\,\mathbf{as}\,\varepsilon) =_{\mathrm{obs}} (\mathbf{pack}\,\tau_2', M'\,\mathbf{as}\,\varepsilon) : \varepsilon.$$

From this and the definition of P we get via the beta-conversions (28) and (29) that

$$P\,\tau_2\,M \;=_{\mathrm{obs}}\; \mathbf{pack}\,\tau_2', M'\,\mathbf{as}\,\varepsilon : \varepsilon. \tag{60}$$

By (54) we have $((\mathbf{pack}\,\tau_2', M'\,\mathbf{as}\,\varepsilon), G\,\tau_2'\,M') \in \mathit{graph}_S$. Then since graph_S is $\top\top$-closed (Lemma 6.1) from this and (60) we conclude via Corollary 3.15 that $(P\,\tau_2\,M, G\,\tau_2'\,M') \in \mathit{graph}_S$. Since this holds for all r and $(M, M') \in \Delta_\tau(r^{\top\top}/\alpha)$, we do indeed have (59).

Now since $\Delta_{\varepsilon'}()$ is reflexive (Corollary 4.13),

$$(G', G') \in \Delta_{\varepsilon'}() = \forall\, r'\,(\forall\, r\,(\Delta_\tau(r^{\top\top}/\alpha) \to {r'}^{\top\top}) \to {r'}^{\top\top}).$$

Hence $(G'\,\varepsilon, G'\,\tau') \in \forall\, r\,(\Delta_\tau(r^{\top\top}/\alpha) \to \mathit{graph}_S) \to \mathit{graph}_S$ (using the fact that $\mathit{graph}_S = (\mathit{graph}_S)^{\top\top}$). So by (59) we have $(G'\,\varepsilon\,P, G'\,\tau'\,G) \in \mathit{graph}_S$. Therefore by definition of S we do indeed have (58). □

8. Conclusion

The notion of contextual equivalence of programs has a final, as opposed to initial, character—in that terms are identified as much as possible within some observational framework. Therefore it is reasonable to expect $\forall$-types to have strong parametricity properties with respect to such a notion of equivalence. The work of Mitchell and Moggi on the maximally consistent model of PLC (see Mitchell 1996, Section 9.3.2 *et seq*) vindicates this expectation, and the work presented here provides further evidence, this time in a context more directly relevant to functional programming. It seems that in the presence of fixpoints, polymorphic types have very rich properties up to contextual equivalence and that operationally-based logical relations provide a convenient way of proving these properties. To not obscure the ideas with too many syntactic details, we chose here to focus just upon the definability up to observational congruence of list types and existential types. But similar results can be derived using our techniques for other common type constructs, such as products, sums, and covariant recursive types built from them. In the case of covariant recursive types (sometimes called 'algebraic data types') one cannot proceed as we did in Section 4 and define the requisite logical relation $(\Delta_\tau \mid \tau \in \mathit{Typ})$ by induction on the structure of τ—because the clause for a recursive type involves the relation at structurally more complicated types. Instead one can define all the relations simultaneously using the Tarski fixed point theorem: covariance of the recursive types ensures the monotonicity of the operator whose fixed point specifies the required family of relations. In the case of general mixed-variance recursive types, this method is not available (because the operator is no longer monotone) and the existence of logical relations of the kind we need is much harder to establish. One way to proceed is via a syntactical analogue of the technique developed in (Pitts 1996) for recursively defined domains. This has been carried out for a single, top-level recursive type by (Birkedal and Harper 1997) (although without the benefit of the $(-)^\top$ machinery). Indeed, using the analysis of Freyd (1992), general recursive types à la Plotkin's FPC with their usual operational semantics (Plotkin 1985) should be observationally isomorphic to PolyPCF types.

These kind of applications are certainly just a small selection of the results which can be proved using the machinery of Sections 3 and 4 (see also (Pitts and Stark 1998) for example). The Galois connection $(-)^\top$ between term-relations and stack-relations (Definition 3.9) seems the most interesting ingredient of that machinery. One of its roles is to tie the operational semantics

Vanilla PCF
(call-by-name evaluation; termination at function types is not observable)

$$\Delta_{\tau_1 \to \tau_2}(\vec{r}) \overset{\text{def}}{=} \Delta_{\tau_1}(\vec{r}) \to \Delta_{\tau_2}(\vec{r})$$

where in general

$$r_1 \to r_2 \overset{\text{def}}{=} \{(F, F') \mid \forall\,(A, A') \in r_1\,((F\,A, F'\,A') \in r_2)\}.$$

'Lazy' PCF
(call-by-name evaluation; termination at function types is observable)

$$\Delta_{\tau_1 \to \tau_2}(\vec{r}) \overset{\text{def}}{=} (\lambda\,\Delta_{\tau_1}(\vec{r})\,(\Delta_{\tau_2}(\vec{r})))^{\top\top}$$

where in general

$$\begin{aligned} \lambda\,r_1\,(r_2) \overset{\text{def}}{=} \quad & \{(\lambda\,x:\tau_1\,(M), \lambda\,x:\tau_1\,(M')) \mid \\ & \quad \forall\,(A, A') \in r_1\,((M[A/x], M'[A'/x]) \in r_2)\}. \end{aligned}$$

Call-by-value PCF
(call-by-value evaluation; hence termination at function types is necessarily observable)

$$\Delta_{\tau_1 \to \tau_2}(\vec{r}) \overset{\text{def}}{=} (\lambda_v\,\Delta_{\tau_1}(\vec{r})\,(\Delta_{\tau_2}(\vec{r})))^{\top\top}$$

where in general

$$\begin{aligned} \lambda_v\,r_1\,(r_2) \overset{\text{def}}{=} \quad & \{(\lambda\,x:\tau_1\,(M), \lambda\,x:\tau_1\,(M')) \mid \\ & \quad \forall\,(V, V') \in r_1 \text{ with } V, V' \text{ values } ((M[V/x], M'[V'/x]) \in r_2)\}. \end{aligned}$$

Fig. 9. Some actions of → on term-relations

into the logical relation. This idea is reinforced in Fig. 9, where we mention some alternative actions of → on term-relations (cf. Definition 4.1) which fit contextual equivalence for 'lazy' and call-by-value PCF. Of course in each case, the definition of $- \top -$ and hence of $(-)^{\top\top}$, changes to match the changed operational semantics and/or observational scenario; and in the second case the notion of frame stack is different as well. The full details of this style of logical relation for a call-by-value version of PolyPCF can be found in (Pitts 1998), which uses it to explore extensionality principles for existential types.

As mentioned in Section 3, another role of the $(-)^{\top}$ operation is to provide a syntactic version of the domain-theoretic notion of admissibility (i.e. of a subset being bottom-containing and closed under least upper bounds of ascending chains). The recent upsurge in operational techniques in the semantics of higher order programming languages has been fuelled to a certain extent by developing syntactical versions of domain-theoretic methods (see (Mason, Smith, and Talcott 1996) and (Birkedal and Harper 1997) for example). Here it may be interesting to go in the opposite direction. The Galois connection $(-)^{\top}$ arose from purely operational considerations (in fact, as a way of dealing with dynamic allocation of local state in the logical relation introduced in (Pitts and Stark 1998)); but it may be useful to use a denotational version of $(-)^{\top}$ for 'extensional collapses' when constructing models of polymorphism and recursion. Denotationally, strict continuous functions play the role of frame stacks (evaluation contexts). So given domains D and D', choosing to take 'observations' in the two-element domain $I = \{\bot, \top\}$ with $\bot \sqsubseteq \top$, we may consider the evident Galois connection between relations $R \subseteq D \times D'$ and relations $S \subseteq (D \multimap I) \times (D' \multimap I)$ induced by

$$f \top d \overset{\text{def}}{\Leftrightarrow} f(d) = \top$$

where $D \multimap I$ denotes the usual domain of strict continuous functions from D to I. Which relations on $D \times D'$ are closed for this Galois connection? It is not hard to see that such relations are admissible in the usual sense. Winskel (private communication) has given a simple example to show that the converse is false; and Abadi (2000) gives an interesting, inductive characterisation of the $\top\top$-closed relations.

References

Abadi, M. (2000). $\top\top$-Closed relations and admissibility. *Mathematical Structures in Computer Science 10*, ?–?.

Abadi, M., L. Cardelli, and P.-L. Curien (1993). Formal parametric polymorphism. *Theoretical Computer Science 121*, 9–58.

Abadi, M. and G. D. Plotkin (1990). A PER model of polymorphism and recursive types. In *5th Annual Symposium on Logic in Computer Science*, pp. 355–365. IEEE Computer Society Press, Washington.

Abramsky, S. (1990). The lazy λ-calculus. In D. A. Turner (Ed.), *Research Topics in Functional Programming*, Chapter 4, pp. 65–117. Addison Wesley.

Amadio, R. M. and P.-L. Curien (1998). *Domains and Lambda-Calculi*. Cambridge University Press.

Bainbridge, E. S., P. J. Freyd, A. Scedrov, and P. J. Scott (1990). Functorial polymorphism. *Theoretical Computer Science 70*, 35–64. Corrigendum in 71(1990) 431.

Birkedal, L. and R. Harper (1997). Relational interpretation of recursive types in an operational setting (Summary). In M. Abadi and T. Ito (Eds.), *Theoretical Aspects of Computer Software, Third International Symposium, TACS'97, Sendai, Japan, September 23 - 26, 1997, Proceedings*, Volume 1281 of *Lecture Notes in Computer Science*. Springer-Verlag, Berlin.

Böhm, C. and A. Berarducci (1985). Automatic synthesis of typed λ-programs on term algebras. *Theoretical Computer Science 39*, 135–154.

Cardelli, L. (1997). Type systems. In *CRC Handbook of Computer Science and Engineering*, Chapter 103, pp. 2208–2236. CRC Press.

Coquand, T., C. A. Gunter, and G. Winskel (1987). DI-domains as a model of polymorphism. In M. Main, A. Melton, M. Mislove, and D. Schmidt (Eds.), *Mathematical Foundations of Programming Language Semantics*, Volume 298 of *Lecture Notes in Computer Science*, pp. 344–363. Springer-Verlag, Berlin.

Coquand, T., C. A. Gunter, and G. Winskel (1989). Domain theoretic models of polymorphism. *Information and Computation 81*, 123–167.

Felleisen, M. and R. Hieb (1992). The revised report on the syntactic theories of sequential control and state. *Theoretical Computer Science 103*, 235–271.

Freyd, P. J. (1992). Remarks on algebraically compact categories. In M. P. Fourman, P. T. Johnstone, and A. M. Pitts (Eds.), *Applications of Categories in Computer Science, Proceedings LMS Symposium, Durham, UK, 1991*, Volume 177 of *LMS Lecture Note Series*, pp. 95–106. Cambridge University Press.

Girard, J.-Y. (1972). *Interprétation fonctionelle et élimination des coupures dans l'arithmetique d'ordre supérieur*. Ph. D. thesis, Université Paris VII. Thèse de doctorat d'état.

Girard, J.-Y. (1989). *Proofs and Types*. Cambridge University Press. Translated and with appendices by Y. Lafont and P. Taylor.

Gordon, A. D. (1995). Bisimilarity as a theory of functional programming. In *Eleventh Conference on the Mathematical Foundations of Programming Semantics, New Orleans, 1995*, Volume 1 of *Electronic Notes in Theoretical Computer Science*. Elsevier.

Gordon, A. D. (1998). Operational equivalences for untyped and polymorphic object calculi. In A. D. Gordon and A. M. Pitts (Eds.), *Higher Order Operational Techniques in Semantics*, Publications of the Newton Institute, pp. 9–54. Cambridge University Press.

Harper, R. and M. Lillibridge (1993). Explicit polymorphism and CPS conversion. In *20th SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, pp. 206–219. ACM Press.

Harper, R. and C. Stone (1996). A type-theoretic account of Standard ML 1996 (version 2). Technical Report CMU–CS–96–136R, Carnegie Mellon University, Pittsburgh, PA.

Hasegawa, R. (1991). Parametricity of extensionally collapsed term models of polymorphism and their categorical properties. In T. Ito and A. R. Meyer (Eds.), *Theoretical Aspects of Computer Software*, Volume 526 of *Lecture Notes in Computer Science*, pp. 495–512. Springer-Verlag, Berlin.

Hasegawa, R. (1994). Categorical data types in parametric polymorphism. *Mathematical Structures in Computer Science 4*, 71–110.

Lassen, S. B. (1998). *Relational Reasoning about Functions and Nondeterminism*. Ph. D. thesis, Department of Computer Science, University of Aarhus.

Ma, Q. and J. C. Reynolds (1992). Types, abstraction, and parametric polymorphism, part 2. In S. Brookes, M. Main, A. Melton, M. Mislove, and D. A. Schmidt (Eds.), *Mathematical Foundations of Programming Semantics, Proceedings 1991*, Volume 598 of *Lecture Notes in Computer Science*, pp. 1–40. Springer-Verlag, Berlin.

Mason, I. A., S. F. Smith, and C. L. Talcott (1996). From operational semantics to domain theory. *Information and Computation 128*(1), 26–47.

Mason, I. A. and C. L. Talcott (1991). Equivalence in functional languages with effects. *Journal of Functional Programming 1*, 287–327.

Milner, R. (1977). Fully abstract models of typed lambda-calculi. *Theoretical Computer Science 4*, 1–22.

Mitchell, J. C. (1996). *Foundations for Programming Languages*. Foundations of Computing series. MIT Press.

Mitchell, J. C. and G. D. Plotkin (1988). Abtract types have existential types. *ACM Transactions on Programming Languages and Systems 10*, 470–502.

Morrisett, G., D. Walker, K. Crary, and N. Glew (1998). From System F to typed assembly language. In *25rd SIGPLAN-SIGACT Symposium on Principles of Programming Languages*. ACM Press.

Pitts, A. M. (1996). Relational properties of domains. *Information and Computation 127*, 66–90.

Pitts, A. M. (1997a). Operationally-based theories of program equivalence. In P. Dybjer and A. M. Pitts (Eds.), *Semantics and Logics of Computation*, Publications of the Newton Institute, pp. 241–298. Cambridge University Press.

Pitts, A. M. (1997b). Reasoning about local variables with operationally-based logical relations. In P. W. O'Hearn and R. D. Tennent (Eds.), *Algol-Like Languages*, Volume 2, Chapter 17, pp. 173–193. Birkhauser. First appeared in *Proceedings 11th Annual IEEE Symposium on Logic in Computer Science*, Brunswick, NJ, July 1996, pp 152–163.

Pitts, A. M. (1998). Existential types: Logical relations and operational equivalence. In K. G. Larsen, S. Skyum, and G. Winskel (Eds.), *Automata, Languages and Programming, 25th International Colloquium, ICALP'98, Aalborg, Denmark, July 1998, Proceedings*, Volume 1443 of *Lecture Notes in Computer Science*, pp. 309–326. Springer-Verlag, Berlin.

Pitts, A. M. and I. D. B. Stark (1998). Operational reasoning for functions with local state. In A. D. Gordon and A. M. Pitts (Eds.), *Higher Order Operational Techniques in Semantics*, Publications of the Newton Institute, pp. 227–273. Cambridge University Press.

Plotkin, G. D. (1973). Lambda-definability and logical relations. Memorandum SAI-RM-4, School of Artificial Intelligence, University of Edinburgh.

Plotkin, G. D. (1977). LCF considered as a programming language. *Theoretical Computer Science 5*, 223–255.

Plotkin, G. D. (1985). Lectures on predomains and partial functions. Notes for a course given at the Center for the Study of Language and Information, Stanford.

Plotkin, G. D. (1993). Second order type theory and recursion. Notes for a talk at the Scott Fest.

Plotkin, G. D. and M. Abadi (1993). A logic for parametric polymorphism. In M. Bezem and J. F. Groote (Eds.), *Typed Lambda Calculus and Applications*, Volume 664 of *Lecture Notes in Computer Science*, pp. 361–375. Springer-Verlag, Berlin.

Reynolds, J. C. (1974). Towards a theory of type structure. In *Paris Colloquium on Programming*, Volume 19 of *Lecture Notes in Computer Science*, pp. 408–425. Springer-Verlag, Berlin.

Reynolds, J. C. (1983). Types, abstraction and parametric polymorphism. In R. E. A. Mason (Ed.), *Information Processing 83*, pp. 513–523. North-Holland, Amsterdam.

Reynolds, J. C. and G. D. Plotkin (1993). On functors expressible in the polymorphic typed lambda calculus. *Information and Computation 105*, 1–29.

Scott, D. S. (1993). A type-theoretical alternative to ISWIM, CUCH, OWHY. *Theoretical Computer Science 121*, 411–440.

Wadler, P. (1989). Theorems for free! In *Fourth International Conference on Functional Programming Languages and Computer Architecture*, London, UK.

Wells, J. B. (1994). Typability and type-checking in the second-order λ-calculus are equivalent and undecidable. In *Proceedings, 9th Annual IEEE Symposium on Logic in Computer Science*, Paris, France, pp. 176–185. IEEE Computer Society Press.