Operationally-Based Theories of Program Equivalence

Andrew Pitts

Contents

1 Introduction

This article describes some mathematical methods for verifying properties of programs in higher-order, functional languages. We focus on methods for reasoning about equivalence of expressions. Such methods are often based upon a denotational semantics of the programming language in question, using the mathematical theory of domains (Scott 1982; Plotkin 1981a). Here I will describe some methods which are based upon *operational semantics* (Plotkin 1981b). These operationally-based techniques have several attractive features. For example, there is relatively little mathematical overhead involved in developing the basic theory—in contrast with that needed to develop the existence and properties of recursively defined domains, the *sine qua non* of denotational semantics. On the other hand, domain theory provides an extremely powerful tool for analysing recursive program constructs. I believe that any serious attempt to develop a useful theory for verification of program properties has to involve *both* operational and denotational techniques.

Highlights The main purpose of this article is to advertise the usefulness, for proving equivalences between functional programs, of co-inductive techniques more familiar in the context of concurrency theory (de Roever 1978; Park 1981; Milner 1989). They were imported into the world of lambda calculus and functional programming by several people: see Dybjer and Sander (1989); Abramsky (1990); Howe (1989, Howe (1996); Egidi, Honsell, and della Rocca (1992); and Gordon 1994. I will also present proofs of some 'domain-theoretic' properties of the operational semantics. To keep things simple, but non-trivial, the example programming language used throughout is an extension of PCF (Plotkin 1977) with products and lazy lists. The technical highlights are:

- An 'operational extensionality' theorem (Theorem 3.8) for the example programming language. This is a generalisation of the context lemma of Milner (1977) and characterises ground contextual equivalence as a certain co-inductively defined notion of bisimilarity. This result yields a co-induction principle for proving instances of contextual equivalence, whose utility we illustrate with several examples (section 3). The use of *ground* contextual equivalence introduces some differences between the appropriate notion of bisimilarity and the 'applicative bisimulation' studied by Abramsky (and Howe) for 'lazy' lambda calculi. As far as I know Gordon (1995a) was the first person to give an operational extensionality theorem for ground contextual equivalence in non-strict, recursively typed languages. His notion of bisimilarity is based upon a labelled transition system for the language. Here we use a notion of bisimilarity based simply upon the evaluation (or 'big-step') semantics. Each approach has its uses. The proof of operational extensionality we give uses an adaptation of a method due to Howe (1989, Howe (1996); we postpone it to an Appendix in order not to interrupt the paper's flow. However, it is the technique rather than the result for the particular example language which is important, and so we urge readers not to neglect this Appendix.

- A proof of some order-theoretic properties of fixpoint recursion with respect to the contextual preorder (section 4). They are syntactic analogues of the ω-chain completeness and continuity properties used in domain-theoretic denotational semantics. Although these properties of the contextual preorder can be derived from a computationally adequate denotational semantics of the language, in keeping with the spirit of this article I give a proof directly from the operational semantics. Mason, Smith, and Talcott (1996) carry out a similar program based on a transition (or 'small-step') semantics, whereas here I use the evaluation semantics.

Prerequisites We assume the reader is familiar with some flavour of functional programming. The textbooks by Abelson and Susman (1985), Paulson (1991), and Bird and Wadler (1988) all provide good introductions. I will also assume familiarity with the use of inductive definitions to specify the syntax and operational semantics of programming languages (especially ones based upon typed lambda calculus). The recent text books by Gunter (1992) and Winskel (1993) both provide good introductions to this topic (and much else besides).

Acknowledgements I have had many stimulating discussions with Andrew Gordon on the topic of operationally-based notions of bisimilarity and co-induction.

His lecture notes (1995b) provide a somewhat different perspective on many of the topics covered here, and are recommended. Much of the material which follows is a reworking of other people's work: sources are given in the text as appropriate. Any errors are, of course, all my own work.

2 Contextual Equivalence

Loosely speaking, two expressions M and M' of a programming language are contextually equivalent if any occurrences of M and M' in complete programs can be interchanged without affecting the results of executing the programs. To formalise this for a particular language (as will be done in Definition 2.9 below), one has to specify precisely how programs are executed, i.e. specify an operational semantics, and one has to specify what the observable results of execution should be. These two key ingredients of contextual equivalence account for the fact that it is often referred to in the literature as *observational*, or *operational*, *equivalence*. As we shall see later (section 5), changing either of the parameters may or may not affect the properties of the resulting notion of contextual equivalence.

For most of this article we study properties of contextual equivalence with respect to a simple functional programming language for recursively defined, higher order functions and lazy lists. We coin the acronym PCFL for this language—standing for 'Programming Computable Functions on pairs and lazy Lists'. As the name suggests, PCFL is obtained from PCF (Plotkin 1977), the mother of all toy programming languages, by adding type constructors for pairs and lazy lists. It has the property of being extremely simple (so that the theory to be developed is not obscured by too many syntactical and semantical complications) whilst containing some potentially infinite data structures, for which co-inductive techniques seem particularly effective.

PCFL syntax

PCFL is a language of terms of various types (integers, booleans, function, product, and list types). Function definitions and recursive definitions in PCFL are handled anonymously, rather than through some explicit mechanism of environments binding identifiers to their definitions. This simplifies the theoretical development at the expense of making PCFL terms somewhat unwieldy.

The terms are built up from constants (for boolean and integer values and for the empty list) and variables, using the constructs which are given in Figure 1 and whose intended meaning is as follows. $\mathsf{if}\ B\ \mathsf{then}\ M\ \mathsf{else}\ N$ is a term which evaluates like M or N, according to whether the boolean term B evaluates to true or false. $M\,\mathsf{op}\,N$ is a binary operation or relation applied to two integer expressions. $\lambda x\,.\,F(x)$ is a name for the function mapping x to $F(x)$. $F\,A$ is the function F applied to the argument A. $\mathsf{fix}\,x\,.\,F(x)$ is a recursively defined term, solving the

$$
\begin{array}{rlll}
M ::= & x & & \text{variables} \\
\mid & \underline{b} & & \text{booleans} \\
\mid & \mathsf{if}\ M\ \mathsf{then}\ M\ \mathsf{else}\ M & & \text{boolean conditional} \\
\mid & \underline{n} & & \text{numerals} \\
\mid & M\ \mathsf{op}\ M & & \text{arithmetic operation} \\
\mid & \lambda x\,.\,M & & \text{function abstraction} \\
\mid & M\ M & & \text{function application} \\
\mid & \mathsf{fix}\,x\,.\,M & & \text{fixpoint recursion} \\
\mid & \langle M, M\rangle & & \text{pairing} \\
\mid & \mathsf{fst}(M) & & \text{first projection} \\
\mid & \mathsf{snd}(M) & & \text{second projection} \\
\mid & \mathsf{nil} & & \text{empty list} \\
\mid & M :: M & & \text{cons list} \\
\mid & \mathsf{case}\ M\ \mathsf{of}\ \{\mathsf{nil} \to M \mid x :: x \to M\} & & \text{list conditional}
\end{array}
$$

where

$$
\begin{array}{rl}
x \in \mathit{Var} & \text{a fixed, infinite set of variables,} \\
b \in \mathbb{B} \stackrel{\text{def}}{=} \{\mathsf{true}, \mathsf{false}\} & \text{the set of booleans,} \\
n \in \mathbb{Z} \stackrel{\text{def}}{=} \{\ldots, -2, 1, 0, 1, 2, \ldots\} & \text{the set of integers,} \\
\mathsf{op} \in \{=, -, =, \leq, \ldots\} & \text{a fixed, finite set of arithmetic} \\
& \text{operation and relation symbols.}
\end{array}
$$

Figure 1: PCFL syntax

fixpoint equation $x = F(x)$. $\langle M, N\rangle$ is the ordered pair with first and second components M and N. $\mathsf{fst}(P)$ is the first component of the pair P. $\mathsf{snd}(P)$ is the second component of the pair P. $H :: T$ is the list with head H and tail T. Finally, $\mathsf{case}\,L\,\mathsf{of}\,\{\mathsf{nil} \to M \mid h{::}t \to N(h,t)\}$ is a term which evaluates like M or $N(H,T)$, according to whether the term L of list type evaluates to the empty list nil, or to a non-empty list $H :: T$.

More precisely, the **PCFL terms** are given by the syntax trees generated by the grammar in Figure 1, modulo α-equivalence. Recall that two expressions in a calculus with variable binding constructs are called **α-equivalent** if they are syntactically identical up to renaming of bound variables. In PCFL, function abstraction, fixpoint recursion and list destructors are variable binding constructs: occurrences of x in M are bound in $\lambda x\,.\,M$ and $\mathsf{fix}\,x\,.\,M$, whilst occurrences of h and t in N are bound in $\mathsf{case}\ L\ \mathsf{of}\ \{\mathsf{nil} \to M \mid h :: t \to N\}$. Any other occurrences of variables are free.

Warning Any reasonable semantics of a programming language with binding constructs will identify α-equivalent expressions. So since we are here concerned with semantic rather than implementation issues, we take the terms of the language PCFL to be α-equivalence classes of syntax trees. It would probably be better, both from an implementation as well as a semantic point of view, to use a more abstract form of representation without explicit bound variables—such as de Bruijn's notation (see Barendregt 1984, Appendix C). However such a representation tends to be hard to read, so we will stick with the more familiar form of syntax given in Figure 1. But be warned that *we will not make a notational distinction between a PCFL syntax tree and the term (α-equivalence class) it determines.*

Notation 2.1. We will use the following notation for the finite set of free variables of a PCFL term:

$$\mathit{fvar}(M) \stackrel{\text{def}}{=} \text{the set of free variables of } M.$$

If M and N are PCFL terms and x is a variable, then $N[M/x]$ will denote the PCFL term resulting from substituting M for all free occurrences of x in N. As usual with calculi with variable binding constructs, this operation of substitution is induced by textual substitution at the level of syntax trees, taking care to avoid capture of free variables (i.e. one must pick a representative tree for N whose bound variables are not in $\mathit{fvar}(M)$). More generally, given a list $M_1, \ldots, M_n$ of terms and a list $x_1, \ldots, x_n$ of distinct variables

$$N[M_1/x_1, \ldots, M_n/x_n], \quad \text{or just} \quad N[\vec{M}/\vec{x}]$$

will denote the result of simultaneously substituting each term M_i in the list for all free occurrences in N of the corresponding variable x_i.

PCFL type assignment

The terms of Plotkin's PCF (1977) contain explicit type information. For PCFL we have chosen to leave out type information from the terms. Nevertheless, PCFL is a typed language, in the sense that we will only consider a term to be well formed if it can be assigned a type, given an assignment of types to the free variables occurring in the term. **PCFL-types** are given by the following grammar:

$$\begin{array}{rlll} \sigma ::= & \gamma & & \text{ground type} \\ \mid & \sigma \to \sigma' & & \text{function type} \\ \mid & \sigma \times \sigma' & & \text{product type} \\ \mid & [\sigma] & & \text{list type} \end{array}$$

where

$$\begin{array}{rll} \gamma ::= & bool & \text{type of booleans} \\ \mid & int & \text{type of integers.} \end{array}$$

A PCFL **typing assertion** takes the form

$$\Gamma \vdash M : \sigma \tag{2.1}$$

where Γ is a finite partial function from variables to types, M is a PCFL term, and σ is a type. The **type assignment relation** for PCFL consists of all typing assertions that can be derived from the axioms and rules in Figure 2. If (2.1) is derivable, we simply say that it is **valid**. The notation $\Gamma, x : \sigma$ used in the rule ($\vdash$ abs) denotes the partial function which *properly* extends Γ by mapping x to σ. Implicit in its use is the assumption that x is not in $dom(\Gamma)$, the domain of definition of Γ. With this notational convention, strictly speaking the side condition on the rule is unnecessary, but has been included for people who only look at Figure 2 without reading the preceding sentence. Similar remarks apply to the rules ($\vdash$ fix) and ($\vdash$ case). If $dom(\Gamma)$ consists of the distinct variables $x_1, \ldots, x_n$ and $\Gamma(x_i) = \sigma_i$ say, then we will sometimes write (2.1) as $x_1 : \sigma_1, \ldots, x_n : \sigma_n \vdash M : \sigma$.

Lemma 2.2. *(i) If $\Gamma \vdash M : \sigma$ is a valid typing assertion, then $fvar(M) \subseteq dom(\Gamma)$.*

(ii) If $\Gamma \vdash M : \sigma$ and $x \notin dom(\Gamma)$, then $\Gamma, x : \sigma' \vdash M : \sigma$ (for any σ').

(iii) If $\Gamma, \Gamma' \vdash M : \sigma$ and $fvar(M) \subseteq dom(\Gamma)$, then $\Gamma \vdash M : \sigma$. Here (and elsewhere), Γ, Γ' indicates the union of the partial functions Γ and Γ', under the assumption that their domains of definition are disjoint.

(iv) If $\Gamma \vdash M_i : \sigma_i$ for $i = 1, \ldots, n$ and $\Gamma, x_1 : \sigma_1, \ldots, x_n : \sigma_n \vdash N : \sigma$, then $\Gamma \vdash N[M_1/x_1, \ldots, M_n/x_n] : \sigma$.

Proof. Parts (i), (ii) and (iii) are proved by induction on the derivation of $\Gamma \vdash M : \sigma$. Part (iv) is proved by induction on the derivation of $\Gamma, x_1 : \sigma_1, \ldots, x_n : \sigma_n \vdash N : \sigma$, using part (ii). □

Definition 2.3. Let $Exp_\sigma(\Gamma)$ denote the set of PCFL terms that can be assigned type σ, given Γ:

$$Exp_\sigma(\Gamma) \stackrel{\text{def}}{=} \{M \mid \Gamma \vdash M : \sigma\}.$$

By part (i) of the above lemma, any $M \in Exp_\sigma(\Gamma)$ has its free variables contained in $dom(\Gamma)$. In particular, in the case Γ is the empty partial function, M is a **closed term**, that is, one with no free variables. (A term which does have free variables is called **open**.) We will write Exp_σ for $Exp_\sigma(\Gamma)$ in this case. The elements of Exp_σ are called the closed PCFL terms of type σ. A closed term is **typeable** if it belongs to Exp_σ for some type σ.

$$\Gamma \vdash x : \sigma \quad \text{(if } \Gamma \text{ is defined at } x \text{ with value } \sigma\text{)} \qquad (\vdash \text{var})$$

$$\Gamma \vdash \underline{b} : bool \quad (\text{if } b \in \mathbb{B}) \qquad (\vdash \text{bool})$$

$$\frac{\Gamma \vdash B : bool \qquad \Gamma \vdash M_1 : \sigma \qquad \Gamma \vdash M_2 : \sigma}{\Gamma \vdash \mathsf{if}\ B\ \mathsf{then}\ M_1\ \mathsf{else}\ M_2 : \sigma} \qquad (\vdash \text{cond})$$

$$\Gamma \vdash \underline{n} : int \quad (\text{if } n \in \mathbb{Z}) \qquad (\vdash \text{int})$$

$$\frac{\Gamma \vdash M_1 : int \qquad \Gamma \vdash M_2 : int}{\Gamma \vdash M_1\ \mathsf{op}\ M_2 : \gamma}\ (\text{if } \gamma \text{ is the result type of } \mathsf{op}) \qquad (\vdash \text{op})$$

$$\frac{\Gamma, x : \sigma \vdash M : \sigma'}{\Gamma \vdash \lambda x\,.\,M : \sigma \to \sigma'}\ (x \notin dom(\Gamma)) \qquad (\vdash \text{abs})$$

$$\frac{\Gamma \vdash F : \sigma \to \sigma' \qquad \Gamma \vdash A : \sigma}{\Gamma \vdash F\,A : \sigma'} \qquad (\vdash \text{app})$$

$$\frac{\Gamma, x : \sigma \vdash F : \sigma}{\Gamma \vdash \mathsf{fix}\, x\,.\,F : \sigma}\ (x \notin dom(\Gamma)) \qquad (\vdash \text{fix})$$

$$\frac{\Gamma \vdash M_1 : \sigma \qquad \Gamma \vdash M_2 : \sigma'}{\Gamma \vdash \langle M_1, M_2 \rangle : \sigma \times \sigma'} \qquad (\vdash \text{pair})$$

$$\frac{\Gamma \vdash P : \sigma \times \sigma'}{\Gamma \vdash \mathsf{fst}(P) : \sigma} \qquad (\vdash \text{fst})$$

$$\frac{\Gamma \vdash P : \sigma \times \sigma'}{\Gamma \vdash \mathsf{snd}(P) : \sigma'} \qquad (\vdash \text{snd})$$

$$\Gamma \vdash \mathsf{nil} : [\sigma] \qquad (\vdash \text{nil})$$

$$\frac{\Gamma \vdash H : \sigma \qquad \Gamma \vdash T : [\sigma]}{\Gamma \vdash H :: T : [\sigma]} \qquad (\vdash \text{cons})$$

$$\frac{\Gamma \vdash L : [\sigma] \quad \Gamma \vdash M_1 : \sigma' \quad \Gamma, h : \sigma, t : [\sigma] \vdash M_2 : \sigma'}{\Gamma \vdash \mathsf{case}\ L\ \mathsf{of}\ \{\mathsf{nil} \to M_1 \mid h :: t \to M_2\} : \sigma'}\ (h, t \notin dom(\Gamma)) \quad (\vdash \text{case})$$

Figure 2: Rules for type assignment in PCFL

Evaluation of PCFL terms

Typically, a program in a typed functional language consists of some definitions (usually, recursive definitions) of data of various types, together with a term of a type whose values are printable (integers, booleans, character strings, etc) to

$$C \Downarrow C \qquad (\Downarrow \text{can})$$

$$\frac{B \Downarrow \mathsf{true} \qquad M_1 \Downarrow C}{\mathsf{if}\ B\ \mathsf{then}\ M_1\ \mathsf{else}\ M_2 \Downarrow C} \qquad (\Downarrow \text{cond1})$$

$$\frac{B \Downarrow \mathsf{false} \qquad M_2 \Downarrow C}{\mathsf{if}\ B\ \mathsf{then}\ M_1\ \mathsf{else}\ M_2 \Downarrow C} \qquad (\Downarrow \text{cond2})$$

$$\frac{M_1 \Downarrow \underline{n_1} \qquad M_2 \Downarrow \underline{n_2}}{M_1\ \mathsf{op}\ M_2 \Downarrow \underline{c}}\ (\text{if } c = n_1\ \mathsf{op}\ n_2) \qquad (\Downarrow \text{op})$$

$$\frac{F \Downarrow \lambda x\,.\,M \qquad M[A/x] \Downarrow C}{F\,A \Downarrow C} \qquad (\Downarrow \text{app})$$

$$\frac{F[\mathsf{fix}\,x\,.\,F/x] \Downarrow C}{\mathsf{fix}\,x\,.\,F \Downarrow C} \qquad (\Downarrow \text{fix})$$

$$\frac{P \Downarrow \langle M_1, M_2\rangle \qquad M_1 \Downarrow C}{\mathsf{fst}(P) \Downarrow C} \qquad (\Downarrow \text{fst})$$

$$\frac{P \Downarrow \langle M_1, M_2\rangle \qquad M_2 \Downarrow C}{\mathsf{snd}(P) \Downarrow C} \qquad (\Downarrow \text{snd})$$

$$\frac{L \Downarrow \mathsf{nil} \qquad M_1 \Downarrow C}{\mathsf{case}\ L\ \mathsf{of}\ \{\mathsf{nil} \to M_1 \mid h :: t \to M_2\} \Downarrow C} \qquad (\Downarrow \text{case1})$$

$$\frac{L \Downarrow H :: T \qquad M_2[H/h, T/t] \Downarrow C}{\mathsf{case}\ L\ \mathsf{of}\ \{\mathsf{nil} \to M_1 \mid h :: t \to M_2\} \Downarrow C} \qquad (\Downarrow \text{case2})$$

Figure 3: Rules for evaluating PCFL terms

be evaluated modulo the given definitions. In PCFL definitions are given anonymously within a term, and we take the types with printable values to be just the ground types, $bool$ and int. Therefore, a PCFL **program** is defined to be a closed term of ground type.

Executing such a program consists of evaluating the term to see which integer or boolean it denotes, if any. The process of evaluation will usually involve evaluation of subexpressions of non-ground type. There are at least two standard ways to specify this process of evaluation: by means of a transition relation between terms and by means of an evaluation relation between terms and terms in canonical form. Both are examples of the structural operational semantics of Plotkin (1981b), in as much as the inductive definition of the relation follows the structure of the term being evaluated. Here we will use the second (and more abstract) approach, and give an inductively defined **evaluation relation**.The relation takes the form

$$M \Downarrow C$$

where M and C are closed, typeable PCFL terms and C is in **canonical form**:

$$C ::= \underline{b} \mid \underline{n} \mid \lambda x \,.\, M \mid \langle M, M\rangle \mid \mathsf{nil} \mid M :: M.$$

The evaluation relation is inductively defined by the axioms and rules in Figure 3.

The canonical forms and evaluation rules embody certain choices which have been made about how PCFL programs behave: evaluation does not continue under a lambda abstraction, or within the components of a pair or list-cons; and an argument is passed unevaluated to the body of a lambda abstraction in function application. Of course, these choices affect the properties of contextual equivalence for PCFL. Other choices and their effect on theories of program equivalence will be discussed in section 5.

Proposition 2.4. *Evaluation is deterministic and preserves typing, that is*

(i) (Determinacy) If $M \Downarrow C$ and $M \Downarrow C'$, then $C = C'$.

(ii) (Subject reduction) If $M \Downarrow C$ and $M \in Exp_\sigma$, then $C \in Exp_\sigma$.

Proof. Both properties can easily be proved by induction on the derivation of $M \Downarrow C$ (using Lemma 2.2(iv)). ☐

Exercise 2.5. Terms of type $[\sigma]$ in PCFL are notations for potentially infinite lists of the data described by terms of type σ. Here is how the standard example of the infinite list of natural numbers, $[0, 1, 2, \ldots]$ can be coded in PCFL. Consider the terms

$$\begin{aligned} nats &\stackrel{\text{def}}{=} \mathsf{fix}\, \ell \,.\, \underline{0} :: map(\lambda x \,.\, x + \underline{1})\ell \\ map &\stackrel{\text{def}}{=} \mathsf{fix}\, m \,.\, \lambda f \,.\, \lambda \ell \,. \\ &\qquad\qquad \mathsf{case}\ \ell\ \mathsf{of}\ \{\mathsf{nil} \to \mathsf{nil} \mid h :: t \to f h :: m f t\} \end{aligned}$$

$$\begin{aligned} head &\stackrel{\text{def}}{=} \lambda \ell \,.\, \mathsf{case}\ \ell\ \mathsf{of}\ \{\mathsf{nil} \to \bot \mid h :: t \to h\} \\ tail &\stackrel{\text{def}}{=} \lambda \ell \,.\, \mathsf{case}\ \ell\ \mathsf{of}\ \{\mathsf{nil} \to \mathsf{nil} \mid h :: t \to t\} \\ \bot &\stackrel{\text{def}}{=} \mathsf{fix}\, x \,.\, x \end{aligned}$$

Show that for any types σ and σ'

$$\begin{aligned} &\emptyset \vdash \bot : \sigma \\ &\emptyset \vdash head : [\sigma] \to \sigma \\ &\emptyset \vdash tail : [\sigma] \to [\sigma] \\ &\emptyset \vdash map : (\sigma \to \sigma') \to ([\sigma] \to [\sigma']) \end{aligned}$$

and hence that $\emptyset \vdash nats : [int]$. Prove that $nats$ is a notation for the infinite list $[0, 1, 2, \ldots]$ in the sense that for all $n \in \mathbb{N}$, $head(tail^n\ nats) \Downarrow \underline{n}$, where

$$tail^0 \stackrel{\text{def}}{=} \lambda x \,.\, x$$
$$tail^{n+1} \stackrel{\text{def}}{=} \lambda x \,.\, tail(tail^n\ x).$$

PCFL contexts

Recall the informal definition of contextual equivalence with which we began this section. For our example language PCFL, so far we have decided upon what constitutes a program (namely, a closed term of ground type) and what the observable results of execution should be (namely, the integer or boolean constant to which the program evaluates, if any). It remains to formalise the notion of interchanging occurrences of terms in programs. To do so, we use 'contexts'—syntax trees containing parameters (or place-holders, or 'holes') which yield a term when the parameters are replaced by terms. Thus the **PCFL contexts**, $\mathcal{C}$, are the syntax trees generated by the grammar in Figure 1 augmented by the clause

$$\mathcal{C} ::= \cdots \mid \mathsf{p}$$

where p ranges over some fixed set of parameters. Note that the syntax trees of PCFL terms are particular contexts, namely the ones with no occurrences of parameters.

Context substitution $\mathcal{C}'[\mathcal{C}/\mathsf{p}]$ will denote the PCFL context obtained from a context $\mathcal{C}'$ by replacing all occurrences of p with the context $\mathcal{C}$. It should be emphasised that this form of substitution may well involve capture of free variables in $\mathcal{C}$ by binding variables in $\mathcal{C}'$. For example, if $\mathcal{C} = x$ and $\mathcal{C}' = \lambda x \,.\, \mathsf{p}$, then $\mathcal{C}'[\mathcal{C}/\mathsf{p}] = \lambda x \,.\, x$. For this reason, the operation of substituting $\mathcal{C}$ for p does not preserve the relation $\equiv^\alpha$ of α-equivalence. For example if x and y are distinct variables, then $\lambda x \,.\, \mathsf{p} \equiv^\alpha \lambda y \,.\, \mathsf{p}$, but $(\lambda x \,.\, \mathsf{p})[x/\mathsf{p}] = \lambda x \,.\, x \not\equiv^\alpha \lambda y \,.\, x = (\lambda y \,.\, \mathsf{p})[x/\mathsf{p}]$. However, one can easily prove by induction on the structure of $\mathcal{C}'$ that

$$\mathcal{C}_1 \equiv^\alpha \mathcal{C}_2 \Rightarrow \mathcal{C}'[\mathcal{C}_1/\mathsf{p}] \equiv^\alpha \mathcal{C}'[\mathcal{C}_1/\mathsf{p}].$$

In other words, substituting α-equivalent contexts results in α-equivalent contexts. It follows that the operation of substituting for a parameter in a context induces a well-defined operation on α-equivalence classes of PCFL syntax trees, that is, on PCFL terms.

It is possible to give a treatment of contexts and contextual equivalence which does not descend below the level of abstraction of α-equivalence classes of expressions (or equivalently, a treatment which applies to expressions using de Bruijn indices rather than explicit bound variables), at the expense of introducing 'function variables'. The interested reader is referred to (Pitts 1994, Section 4).

Notation 2.6. Most of the time we will use contexts only involving a single parameter, which we write as $-$. We write $\mathcal{C}[-]$ to indicate that $\mathcal{C}$ is a context containing no parameters other than $-$. If M is a PCFL term, then $\mathcal{C}[M]$ will denote the term resulting from choosing a representative syntax tree for M, substituting it for the parameter in $\mathcal{C}$, and forming the α-equivalence class of the resulting PCFL syntax tree (which from the remarks above, is independent of the choice of representative for M).

Typed contexts We will assume given a function that assigns types to parameters. We write $-_\sigma$ to indicate that a parameter $-$ has type σ. Just as we only consider a PCFL term to be well-formed if it can be assigned a type, we will restrict attention to contexts that can be typed. The relation

$$\Gamma \vdash \mathcal{C} : \sigma$$

assigning a type σ to a context $\mathcal{C}$ given a finite partial function Γ assigning types to variables, is inductively generated by axioms and rules just like those in Figure 2 together with the following axiom for parameters:

$$\Gamma \vdash -_\sigma : \sigma. \tag{2.2}$$

Warning: when the axioms and rules of Figure 2 are applied to syntax trees rather than α-equivalence classes of syntax trees (as is the case when typing contexts), it should be borne in mind that they enforce a separation between free and bound variables and hence are not closed under α-equivalence. For example, if $x \neq y$, then $x : \mathit{int} \vdash \lambda y \,.\, -_{\mathit{int}} : \mathit{int} \to \mathit{int}$ is a valid typing assertion, whereas $x : \mathit{int} \vdash \lambda x \,.\, -_{\mathit{int}} : \mathit{int} \to \mathit{int}$ is not.

Definition 2.7. Let $\mathit{Ctx}_\sigma(\Gamma)$ denote the set of PCFL contexts that can be assigned type σ, given Γ:

$$\mathit{Ctx}_\sigma(\Gamma) \stackrel{\text{def}}{=} \{\mathcal{C} \mid \Gamma \vdash \mathcal{C} : \sigma\}.$$

We write Ctx_σ for $\mathit{Ctx}_\sigma(\emptyset)$. Given $\mathcal{C}[-_\sigma] \in \mathit{Ctx}_{\sigma'}(\Gamma)$, we write $\mathit{traps}(\mathcal{C}[-_\sigma])$ for the set of variables that occur in $\mathcal{C}[-_\sigma]$ associated to binders containing the hole $-_\sigma$ within their scope. Thus any free variables of M in $\mathit{traps}(\mathcal{C}[-_\sigma])$ become bound in $\mathcal{C}[M]$.

The operation $M \mapsto \mathcal{C}[M]$ of substituting a PCFL term for a parameter in a context to obtain a new PCFL term respects typing in the following sense.

Lemma 2.8. *Suppose $M \in \mathit{Exp}_\sigma(\Gamma, \Gamma')$, $\mathcal{C}[-_\sigma] \in \mathit{Ctx}_{\sigma'}(\Gamma)$, and that $\mathit{dom}(\Gamma') \subseteq \mathit{traps}(\mathcal{C}[-_\sigma])$. Then $\mathcal{C}[M] \in \mathit{Exp}_{\sigma'}(\Gamma)$.*

Proof. By induction on the derivation of $\Gamma \vdash \mathcal{C}[-_\sigma] : \sigma'$. □

Definition 2.9 (Ground contextual equivalence). As usual, let Γ be a finite partial function from variables to PCFL types. Given $M, M' \in \mathit{Exp}_\sigma(\Gamma)$, we write

$$\Gamma \vdash M \leq^{\mathrm{gnd}}_\sigma M'$$

to mean that for all $\mathcal{C}[-_\sigma] \in \mathit{Ctx}_{\mathit{bool}}$ with $\mathit{dom}(\Gamma) \subseteq \mathit{traps}(\mathcal{C}[-_\sigma])$

$$\forall b \in \mathbb{B}\,(\mathcal{C}[M] \Downarrow \underline{b} \Rightarrow \mathcal{C}[M'] \Downarrow \underline{b})$$

and for all $\mathcal{C}[-_\sigma] \in \mathit{Ctx}_{\mathit{int}}$ with $\mathit{dom}(\Gamma) \subseteq \mathit{traps}(\mathcal{C}[-_\sigma])$

$$\forall n \in \mathbb{Z}\,(\mathcal{C}[M] \Downarrow \underline{n} \Rightarrow \mathcal{C}[M'] \Downarrow \underline{n}).$$

(Note that by virtue of Lemma 2.8, $\mathcal{C}[M]$ and $\mathcal{C}[M']$ are indeed closed terms of type γ when $\mathcal{C}[-_\sigma] \in \mathit{Ctx}_\gamma$ satisfies $\mathit{dom}(\Gamma) \subseteq \mathit{traps}(\mathcal{C}[-_\sigma])$.)

The relation $\leq^{\mathrm{gnd}}$ will be called the **ground contextual preorder** between PCFL terms (of the same type, given a typing of their free variables). **Ground contextual equivalence** is the symmetrization of this relation:

$$\Gamma \vdash M \cong^{\mathrm{gnd}}_\sigma M' \overset{\mathrm{def}}{\Leftrightarrow} (\Gamma \vdash M \leq^{\mathrm{gnd}}_\sigma M' \;\&\; \Gamma \vdash M' \leq^{\mathrm{gnd}}_\sigma M).$$

In section 5 we will consider some variations on the notion of contextual equivalence, in which contexts of non-ground type are used. Until then we will drop the adjective 'ground' and just refer to $\leq^{\mathrm{gnd}}$ and $\cong^{\mathrm{gnd}}$ as the **contextual preorder** and **contextual equivalence**. For closed terms $M, M' \in \mathit{Exp}_\sigma$, we will just write

$$M \leq^{\mathrm{gnd}}_\sigma M' \quad \text{and} \quad M \cong^{\mathrm{gnd}}_\sigma M'$$

for $\emptyset \vdash M \leq^{\mathrm{gnd}}_\sigma M'$ and $\emptyset \vdash M \cong^{\mathrm{gnd}}_\sigma M'$ respectively.

Remark 2.10. The relations of contextual preorder and contextual equivalence remain the same if in Definition 2.9 we restrict to contexts yielding terms of type bool only, or of type int only, or restrict attention to evaluation to a fixed integer or boolean constant, or just to convergence to something. For example

$$\Gamma \vdash M \leq^{\mathrm{gnd}}_\sigma M' \Leftrightarrow \forall \mathcal{C}[-]\,(\mathcal{C}[M] \Downarrow \underline{42} \Rightarrow \mathcal{C}[M'] \Downarrow \underline{42}). \tag{2.3}$$

To see this, given any context $\mathcal{C}[-]$, for each $n \in \mathbb{Z}$ note that the context

$$\mathcal{C}_n[-] \overset{\mathrm{def}}{=} \mathsf{if}\ \mathcal{C}[-] = \underline{n}\ \mathsf{then}\ \underline{42}\ \mathsf{else}\ \underline{0}$$

has the property that for all M

$$\mathcal{C}_n[M] \Downarrow \underline{42} \Leftrightarrow \mathcal{C}[M] \Downarrow \underline{n}.$$

Similarly, the contexts

$$\mathcal{C}_{\mathsf{true}}[-] \overset{\mathrm{def}}{=} \mathsf{if}\ \mathcal{C}[-]\ \mathsf{then}\ \underline{42}\ \mathsf{else}\ \underline{0}$$
$$\mathcal{C}_{\mathsf{false}}[-] \overset{\mathrm{def}}{=} \mathsf{if}\ \mathcal{C}[-]\ \mathsf{then}\ \underline{0}\ \mathsf{else}\ \underline{42}$$

satisfy

$$\mathcal{C}_{\mathsf{true}}[M] \Downarrow \underline{42} \Leftrightarrow \mathcal{C}[M] \Downarrow \mathsf{true}$$
$$\mathcal{C}_{\mathsf{false}}[M] \Downarrow \underline{42} \Leftrightarrow \mathcal{C}[M] \Downarrow \mathsf{false}.$$

Property (2.3) follows immediately.

Exercise 2.11. Writing $M{\Downarrow_{int}}$ to mean $\exists n \in \mathbb{Z}\,(M \Downarrow \underline{n})$, show that

$$\Gamma \vdash M \leq^{\mathrm{gnd}}_{\sigma} M' \Leftrightarrow \forall \mathcal{C}[-]\,(\mathcal{C}[M]{\Downarrow_{int}} \Rightarrow \mathcal{C}[M']{\Downarrow_{int}}).$$

[Hint: note that $\perp \stackrel{\text{def}}{=} \mathsf{fix}\, x\,.\,x$ is a closed term (typeable to any type) which does not evaluate to anything. (Why?) Given any $\mathcal{C}[-]$, use $\perp$ to define a new context $\mathcal{C}'[-]$ satisfying for any M that $\mathcal{C}'[M]{\Downarrow_{int}}$ if and only if $\mathcal{C}[M] \Downarrow \underline{42}$. Now use Remark 2.10.]

Properties of PCFL contextual equivalence

Having given the precise definition of contextual equivalence for our example language PCFL we must now develop its theory—its general properties which one can use to establish that particular PCFL terms are, or are not, contextually equivalent. To show that two terms are *not* contextually equivalent is usually quite easy: one just has to find a suitable context of ground type for which the terms yield different results when the rules for evaluation in Figure 3 are applied. For example, the fact that $\underline{0} :: (\underline{1} :: \mathsf{nil})$ and $\underline{0} :: \mathsf{nil}$ are contextually inequivalent terms of type $[int]$, is witnessed by the context $\mathcal{C}[-] \stackrel{\text{def}}{=} head(tail{-})$, where $head$ and $tail$ are the terms defined in Exercise 2.5. For $\mathcal{C}[\underline{0} :: (\underline{1} :: \mathsf{nil})] \Downarrow \underline{1}$, whereas $\mathcal{C}[\underline{0} :: \mathsf{nil}]$ does not evaluate to anything.

The job of establishing that a contextual equivalence *does* hold can be much harder. For example, $\cong^{\mathrm{gnd}}$ satisfies β-conversion:

$$\Gamma \vdash (\lambda x\,.\,M)\,A \cong^{\mathrm{gnd}}_{\sigma'} M[A/x]$$

where $\Gamma, x : \sigma \vdash M : \sigma'$ and $\Gamma \vdash A : \sigma$. However, it is not immediately obvious from Definition 2.9 why this is so. The problem lies mainly in the quantification over all contexts that occurs in the definition of $\leq^{\mathrm{gnd}}$ and $\cong^{\mathrm{gnd}}$. One might try to construct a proof which proceeds by induction on the structure of contexts, but it is not so easy to find a sufficiently strong inductive hypothesis to make all the steps (involving evaluation of subexpressions at non-ground types) go through. We will take up the challenge of such problems seriously in the next section when we introduce PCFL bisimilarity—another, and more tractable, notion of equivalence which turns out to coincide with PCFL contextual equivalence. We conclude this section by stating some of the properties of $\leq^{\mathrm{gnd}}$ and $\cong^{\mathrm{gnd}}$ that will be proved in these notes.

(In)equational logic

$$\Gamma \vdash M : \sigma \Rightarrow \Gamma \vdash M \leq^{\mathrm{gnd}}_{\sigma} M \tag{2.4}$$

$$(\Gamma \vdash M \leq^{\mathrm{gnd}}_{\sigma} M' \ \&\ \Gamma \vdash M' \leq^{\mathrm{gnd}}_{\sigma} M'') \Rightarrow \Gamma \vdash M' \leq^{\mathrm{gnd}}_{\sigma} M'' \tag{2.5}$$

$$(\Gamma \vdash M \leq^{\mathrm{gnd}}_{\sigma} M' \ \&\ \Gamma \vdash M' \leq^{\mathrm{gnd}}_{\sigma} M) \Leftrightarrow \Gamma \vdash M \cong^{\mathrm{gnd}}_{\sigma} M' \tag{2.6}$$

$$\Gamma, x : \sigma \vdash M \leq^{\mathrm{gnd}}_{\sigma'} M' \Rightarrow \Gamma \vdash \lambda x \,.\, M \leq^{\mathrm{gnd}}_{\sigma \to \sigma'} \lambda x \,.\, M' \tag{2.7}$$

$$\Gamma, x : \sigma \vdash M \leq^{\mathrm{gnd}}_{\sigma} M' \Rightarrow \Gamma \vdash \mathsf{fix}\, x \,.\, M \leq^{\mathrm{gnd}}_{\sigma} \mathsf{fix}\, x \,.\, M' \tag{2.8}$$

$$\begin{aligned}(\Gamma \vdash L \leq^{\mathrm{gnd}}_{[\sigma]} L' \ &\&\ \Gamma \vdash M \leq^{\mathrm{gnd}}_{\sigma'} M' \\ &\&\ \Gamma, h : \sigma, t : [\sigma] \vdash N \leq^{\mathrm{gnd}}_{\sigma'} N') \\ &\quad \Rightarrow \Gamma \vdash \ (\mathsf{case}\ L\ \mathsf{of}\ \{\mathsf{nil} \to M \mid h :: t \to N\}) \leq^{\mathrm{gnd}}_{\sigma'} \\ &\qquad\qquad (\mathsf{case}\ L'\ \mathsf{of}\ \{\mathsf{nil} \to M' \mid h :: t \to N'\})\end{aligned} \tag{2.9}$$

$$(\Gamma \vdash M \leq^{\mathrm{gnd}}_{\sigma} M' \ \&\ \Gamma \subseteq \Gamma') \Rightarrow \Gamma' \vdash M \leq^{\mathrm{gnd}}_{\sigma} M' \tag{2.10}$$

$$\begin{aligned}\Gamma \vdash M \leq^{\mathrm{gnd}}_{\sigma} M' \ &\&\ \Gamma, x : \sigma \vdash N : \sigma' \\ &\Rightarrow \Gamma \vdash N[M/x] \leq^{\mathrm{gnd}}_{\sigma'} N[M'/x]\end{aligned} \tag{2.11}$$

$$\begin{aligned}\Gamma \vdash M : \sigma \ &\&\ \Gamma, x : \sigma \vdash N \leq^{\mathrm{gnd}}_{\sigma'} N' \\ &\Rightarrow \Gamma \vdash N[M/x] \leq^{\mathrm{gnd}}_{\sigma'} N'[M/x]\end{aligned} \tag{2.12}$$

Properties (2.4)–(2.9) are all straightforward consequences of the definition of $\leq^{\mathrm{gnd}}$ and $\cong^{\mathrm{gnd}}$. By contrast, (2.12) is not so straightforward to establish, because the operation $N \mapsto N[M/x]$ is not necessarily of the form $N \mapsto \mathcal{C}[N]$ for some context $\mathcal{C}[-]$. The fact that (2.12) holds is intimately tied up with the fact that PCFL contextual equivalence satisfies the β-rule (2.13) given below: see Lemma A.10.

$\boldsymbol{\beta}$-rules

$$(\Gamma, x : \sigma \vdash M : \sigma' \ \&\ \Gamma \vdash A : \sigma) \Rightarrow \Gamma \vdash (\lambda x \,.\, M)\, A \cong^{\mathrm{gnd}}_{\sigma'} M[A/x] \tag{2.13}$$

$$\begin{aligned}&(\Gamma \vdash M : \sigma \ \&\ \Gamma \vdash M' : \sigma') \Rightarrow \\ &\qquad (\Gamma \vdash \mathsf{fst}(\langle M, M' \rangle) \cong^{\mathrm{gnd}}_{\sigma} M \ \&\ \Gamma \vdash \mathsf{snd}(\langle M, M' \rangle) \cong^{\mathrm{gnd}}_{\sigma'} M')\end{aligned} \tag{2.14}$$

$$\begin{aligned}&(\Gamma \vdash M : \sigma' \ \&\ \Gamma, h : \sigma, t : [\sigma] \vdash N : \sigma') \Rightarrow \\ &\qquad \Gamma \vdash \mathsf{case}\ \mathsf{nil}\ \mathsf{of}\ \{\mathsf{nil} \to M \mid h :: t \to N\} \cong^{\mathrm{gnd}}_{\sigma'} M\end{aligned} \tag{2.15}$$

$$(\Gamma \vdash H : \sigma \ \&\ \Gamma \vdash T : [\sigma] \ \&\ \Gamma \vdash M : \sigma' \ \&\ \Gamma, h : \sigma, t : [\sigma] \vdash N : \sigma') \Rightarrow \tag{2.16}$$

$$\Gamma \vdash \mathsf{case}\ H :: T\ \mathsf{of}\ \{\mathsf{nil} \to M \mid h :: t \to N\} \cong^{\mathrm{gnd}}_{\sigma'} N[H/h, T/t]$$

$$(\Gamma \vdash M : \sigma \ \&\ \Gamma \vdash M' : \sigma) \Rightarrow \tag{2.17}$$
$$(\Gamma \vdash \mathsf{if\ true\ then}\ M\ \mathsf{else}\ M' \cong^{\mathrm{gnd}}_{\sigma} M\ \&$$
$$\Gamma \vdash \mathsf{if\ false\ then}\ M\ \mathsf{else}\ M' \cong^{\mathrm{gnd}}_{\sigma} M')$$

$$(n\ \mathsf{op}\ n' = c) \Rightarrow \emptyset \vdash \underline{n}\ \mathsf{op}\ \underline{n'} \cong^{\mathrm{gnd}}_{\gamma} \underline{c} \tag{2.18}$$

The fact that these β-rules are valid follows from the characterisation of $\cong^{\mathrm{gnd}}$ in terms of PCFL bisimilarity to be given in the next section (Theorem 3.8). For in each case, (closed instantiations of) the term on the left hand side of $\cong^{\mathrm{gnd}}$ evaluates to a canonical C if and only if (closed instantiations of) the right hand term evaluates to the *same* canonical form. Thus each of (2.13)–(2.18) follows from the fact, shown in Proposition 3.9, that 'Kleene equivalence' is contained in the relation of contextual equivalence, together with the first of the following extensionality properties.

Extensionality properties

For all $N, N' \in \mathit{Exp}_{\sigma}(x_1 : \sigma_1, \ldots, x_n : \sigma_n)$:

$$x_1 : \sigma_1, \ldots, x_n : \sigma_n \vdash N \leq^{\mathrm{gnd}}_{\sigma} N' \Leftrightarrow$$
$$\forall M_1 \in \mathit{Exp}_{\sigma_1}, \ldots, M_n \in \mathit{Exp}_{\sigma_n}\ (N[\vec{M}/\vec{x}] \leq^{\mathrm{gnd}}_{\sigma} N'[\vec{M}/\vec{x}]) \tag{2.19}$$

For all $M, M' \in \mathit{Exp}_{\gamma}$ (γ a ground type):

$$M \leq^{\mathrm{gnd}}_{\gamma} M' \Leftrightarrow \forall c\,(M \Downarrow \underline{c} \Rightarrow M' \Downarrow \underline{c}) \tag{2.20}$$

For all $F, F' \in \mathit{Exp}_{\sigma \to \sigma'}$:

$$F \leq^{\mathrm{gnd}}_{\sigma \to \sigma'} F' \Leftrightarrow \forall A \in \mathit{Exp}_{\sigma}\,(F\,A \leq^{\mathrm{gnd}}_{\sigma'} F'\,A) \tag{2.21}$$

For all $P, P' \in \mathit{Exp}_{\sigma \times \sigma'}$:

$$P \leq^{\mathrm{gnd}}_{\sigma \times \sigma'} P' \Leftrightarrow$$
$$\mathsf{fst}(P) \leq^{\mathrm{gnd}}_{\sigma} \mathsf{fst}(P')\ \&\ \mathsf{snd}(P) \leq^{\mathrm{gnd}}_{\sigma'} \mathsf{snd}(P') \tag{2.22}$$

For all $L, L' \in \mathit{Exp}_{[\sigma]}$:

$$L \leq^{\mathrm{gnd}}_{[\sigma]} L' \Leftrightarrow (L \Downarrow \mathsf{nil} \Rightarrow L' \Downarrow \mathsf{nil})\ \&$$
$$\forall H, T\,(L \Downarrow H :: T \Rightarrow \exists H', T'\,(L' \Downarrow H' :: T'\ \&$$
$$H \leq^{\mathrm{gnd}}_{\sigma} H'\ \&\ T \leq^{\mathrm{gnd}}_{[\sigma]} T')) \tag{2.23}$$

Analogous extensionality properties hold by construction for the notion of PCFL similarity introduced in the next section. Thus (2.19)–(2.23) will follow once we have proved that this coincides with $\leq^{\mathrm{gnd}}$ (Theorem 3.8).

The η-rule for functions and the surjective pairing rule for products follow by combining these extensionality properties with the corresponding β-rules:

$$(\Gamma \vdash F : \sigma \to \sigma' \;\&\; x \notin dom(\Gamma)) \Rightarrow \Gamma \vdash F \cong^{\mathrm{gnd}}_{\sigma\to\sigma'} (\lambda x \,.\, F)\, x \tag{2.24}$$

$$\Gamma \vdash P : \sigma \times \sigma' \Rightarrow \Gamma \vdash P \cong^{\mathrm{gnd}}_{\sigma\times\sigma'} \langle \mathsf{fst}(P), \mathsf{snd}(P)\rangle \tag{2.25}$$

Unfolding recursive terms

$$\Gamma, x : \sigma \vdash M : \sigma \Rightarrow \Gamma \vdash \mathsf{fix}\, x \,.\, M \cong^{\mathrm{gnd}}_{\sigma} M[\mathsf{fix}\, x \,.\, M/x] \tag{2.26}$$

This holds for the same reason as the β-rules given above—it is an instance of 'Kleene equivalence', and so will follow from Proposition 3.9 below.

Syntactic bottom

The term $\bot \stackrel{\mathrm{def}}{=} \mathsf{fix}\, x \,.\, x$ acts as a least element with respect to $\leq^{\mathrm{gnd}}$:

$$\Gamma \vdash M : \sigma \Rightarrow \Gamma \vdash \bot \leq^{\mathrm{gnd}}_{\sigma} M \tag{2.27}$$

As for the previous property, this will be deduced from Proposition 3.9.

Rational completeness and syntactic continuity

In addition to the unfolding property (2.26), terms of the form $\mathsf{fix}\, x \,.\, F$ enjoy a least prefixed point property: if $F \in Exp_\sigma(x : \sigma)$ and $M \in Exp_\sigma$, then

$$F[M/x] \leq^{\mathrm{gnd}}_{\sigma} M \Rightarrow \mathsf{fix}\, x \,.\, F \leq^{\mathrm{gnd}}_{\sigma} M. \tag{2.28}$$

Sands (1995, Appendix) gives a direct, operationally-based proof of the analogous property of recursive functions definitions, making use of a transition relation rather than just an evaluation relation. We will deduce this least prefixed point property from a stronger property which we now explain.

For each natural number n, let $\mathsf{fix}^{(n)} x \,.\, F$ be the term given as follows:

$$\begin{aligned} \mathsf{fix}^{(0)} x \,.\, F &\stackrel{\mathrm{def}}{=} \bot, \\ \mathsf{fix}^{(n+1)} x \,.\, F &\stackrel{\mathrm{def}}{=} F[\mathsf{fix}^{(n)} x \,.\, F/x]. \end{aligned}$$

It follows from (2.11) and (2.27) that these terms form an ascending chain

$$\bot = \mathsf{fix}^{(0)} x \,.\, F \leq^{\mathrm{gnd}}_{\sigma} \mathsf{fix}^{(1)} x \,.\, F \leq^{\mathrm{gnd}}_{\sigma} \cdots .$$

We claim that $\mathsf{fix}\, x \,.\, F$ is a least upper bound with respect to $\leq^{\mathrm{gnd}}$ for this chain. In other words, for each $M \in Exp_\sigma$

$$\mathsf{fix}\, x \,.\, F \leq^{\mathrm{gnd}}_{\sigma} M \Leftrightarrow \forall n\, (\mathsf{fix}^{(n)} x \,.\, F \leq^{\mathrm{gnd}}_{\sigma} M). \tag{2.29}$$

Thus the collection of PCFL terms preordered by $\leq^{\text{gnd}}$ enjoys a restricted amount of chain-completeness. Moreover, the operations of PCFL preserve these least upper bounds: for each context $\mathcal{C}[-]$ it is the case that

$$\mathcal{C}[\mathsf{fix}\, x\,.\,F] \leq^{\text{gnd}}_{\sigma} M \Leftrightarrow \forall n\,(\mathcal{C}[\mathsf{fix}^{(n)} x\,.\,F] \leq^{\text{gnd}}_{\sigma} M). \tag{2.30}$$

Properties (2.29) and (2.30) will be proved in Section 4, using operationally-based methods (which seem different from those used for the same purpose in (Smith 1992; Mason, Smith, and Talcott 1996)). Such properties can also be established via an adequate denotational semantics of PCFL—see (Pitts 1994) for example.

Note that (2.28) can be deduced from (2.29), since if $F[M/x] \leq^{\text{gnd}}_{\sigma} M$, then one can show by induction on n that $\mathsf{fix}^{(n)} x\,.\,F \leq^{\text{gnd}}_{\sigma} M$. The base case $n = 0$ is just (2.27); and the induction step follows from the hypothesis using (2.11) and (2.5).

3 Similarity and Bisimilarity

Look again at the extensionality properties (2.19)–(2.23) which we claim hold of the contextual preorder, $\leq^{\text{gnd}}$. Property (2.21) expresses the preordered version of a familiar extensionality property for functions—namely that two functions are equal if (and only if) they yield equal results when applied to any argument. In particular, (2.21) serves to express $\leq^{\text{gnd}}$ at a function type $(\sigma \to \sigma')$ in terms of $\leq^{\text{gnd}}$ at a simpler type (σ'). For the simply typed language PCF, $\to$ is the only type constructor and a typical PCF type takes the form $\sigma_1 \to (\sigma_2 \to \ldots (\sigma_n \to \gamma) \ldots)$ with γ a ground type and $n \geq 0$. Consequently, for this simpler language one can express the contextual preorder at any type in terms of application and evaluation of terms of ground type:

Milner's Context Lemma for PCF. *For any closed PCF terms M and M'*

$$M \leq^{\text{gnd}}_{\sigma_1 \to (\sigma_2 \to \ldots (\sigma_n \to \gamma) \ldots)} M' \Leftrightarrow$$
$$\forall A_1, \ldots, A_n, c\,(M A_1 \ldots A_n \Downarrow \underline{c} \Rightarrow M' A_1 \ldots A_n \Downarrow \underline{c}).$$

Proof. See Milner (1977). □

For PCFL with its types of (potentially infinite) lists, $[\sigma]$, the situation is not so straightforward. Property (2.23) does not serve to define $\leq^{\text{gnd}}$ at a list type $[\sigma]$ in terms of $\leq^{\text{gnd}}$ at type σ, since the occurrence $T \leq^{\text{gnd}}_{[\sigma]} T'$ on the right hand side of the bi-implication is not necessarily 'simpler' (for any measure of simplicity) than the occurrence $L \leq^{\text{gnd}}_{[\sigma]} L'$ on the left hand side. (For example, when $L = \mathsf{fix}\,\ell\,.\,\underline{0} :: \ell \in \mathit{Exp}_{[int]}$, then $L \Downarrow \underline{0} :: L$, so T in this case is syntactically identical to L.) In other words, there may be many binary relations between closed PCFL terms which satisfy (2.23). The crucial observation is not just that $\leq^{\text{gnd}}$ is such a relation, but that it is the *greatest* such—indeed, is the greatest relation satisfying just the left-to-right implication in (2.23).

Greatest post-fixed points of monotone operators

Recall that a **complete lattice** is a partially ordered set $(X, \leq)$ for which every subset $S \subseteq X$ has a least upper bound, $\bigvee S$, with respect to $\leq$:

$$\forall x \in X \, (\bigvee S \leq x \Leftrightarrow \forall s \in S \, (s \leq x)).$$

(As is well known, this is equivalent to requiring that every subset has a greatest lower bound.)

A **monotone operator** on $(X, \leq)$ is a function $\Phi : X \longrightarrow X$ satisfying

$$\forall x, x' \in X \, (x \leq x' \Rightarrow \Phi(x) \leq \Phi(x')).$$

The **greatest post-fixed point** of Φ is the (necessarily unique) element $\nu(\Phi)$ of X satisfying

$$\nu(\Phi) \leq \Phi(\nu(\Phi)) \tag{3.1}$$

$$\forall x \in X \, (x \leq \Phi(x) \Rightarrow x \leq \nu(\Phi)). \tag{3.2}$$

Theorem 3.1 (Tarski-Knaster Fixed Point Theorem). *Every monotone operator Φ on a complete lattice $(X, \leq)$ possesses a greatest post-fixed point, $\nu(\Phi)$. This element is in fact the greatest element of the set $\{x \in X \mid x = \Phi(x)\}$ of fixed points of Φ.*

Proof. The proof is probably familiar to you, but in case not, here it is.

The monotonicity of Φ ensures that the least upper bound of any set of post-fixed points for Φ is again a post-fixed point. It follows that

$$\nu(\Phi) \stackrel{\text{def}}{=} \bigvee \{x \in X \mid x \leq \Phi(x)\}$$

is the greatest post-fixed point of Φ. Since

$$x = \Phi(x) \Rightarrow x \leq \Phi(x) \Rightarrow x \leq \nu(\Phi)$$

to prove the second sentence of the theorem, it suffices to see that $\nu(\Phi)$ *is* a fixed point of Φ. Since it is a post-fixed point, it suffices to show that $\Phi(\nu(\Phi)) \leq \nu(\Phi)$. But since $\nu(\Phi) \leq \Phi(\nu(\Phi))$ and Φ is monotone, one has that $\Phi(\nu(\Phi)) \leq \Phi(\Phi(\nu(\Phi)))$, that is, $\Phi(\nu(\Phi))$ is a post-fixed point of Φ. Since $\nu(\Phi)$ is the greatest such we have $\Phi(\nu(\Phi)) \leq \nu(\Phi)$, as required. $\square$

Definition 3.2. Throughout this section we will be concerned with one particular complete lattice, $(\mathit{Rel}, \leq)$. The elements of Rel are type-indexed families $\mathcal{R} = (\mathcal{R}_\sigma \mid \sigma)$ of binary relations $\mathcal{R}_\sigma$ between the closed PCFL terms of type σ. Thus each component of $\mathcal{R}$ is a subset $\mathcal{R}_\sigma \subseteq \mathit{Exp}_\sigma \times \mathit{Exp}_\sigma$. The partial ordering on Rel is defined to be set-theoretic inclusion in each component:

$$\mathcal{R} \leq \mathcal{R}' \stackrel{\text{def}}{\Leftrightarrow} \forall \sigma \, (\mathcal{R}_\sigma \subseteq \mathcal{R}'_\sigma).$$

Clearly, the least upper bound of a subset of Rel is given by set-theoretic union in each component.

PCFL simulations and bisimulations

Given $\mathcal{R} \in Rel$, the elements $\langle \mathcal{R} \rangle$ and $[\mathcal{R}]$ of Rel are defined as follows.

$$B \, \langle \mathcal{R} \rangle_{bool} \, B' \overset{\text{def}}{\Leftrightarrow} \forall b \in \mathbb{B} \, (B \Downarrow \underline{b} \Rightarrow B' \Downarrow \underline{b}) \tag{3.3a}$$

$$N \, \langle \mathcal{R} \rangle_{int} \, N' \overset{\text{def}}{\Leftrightarrow} \forall n \in \mathbb{Z} \, (N \Downarrow \underline{n} \Rightarrow N' \Downarrow \underline{n}) \tag{3.3b}$$

$$F \, \langle \mathcal{R} \rangle_{\sigma \to \sigma'} \, F' \overset{\text{def}}{\Leftrightarrow} \forall A \in Exp_{\sigma} \, (F \, A \, \mathcal{R}_{\sigma'} \, F' \, A) \tag{3.3c}$$

$$P \, \langle \mathcal{R} \rangle_{\sigma \times \sigma'} \, P' \overset{\text{def}}{\Leftrightarrow} \mathsf{fst}(P) \, \mathcal{R}_{\sigma} \, \mathsf{fst}(P') \;\&\; \mathsf{snd}(P) \, \mathcal{R}_{\sigma'} \, \mathsf{snd}(P') \tag{3.3d}$$

$$\begin{aligned} L \, \langle \mathcal{R} \rangle_{[\sigma]} \, L' \overset{\text{def}}{\Leftrightarrow} \;& (L \Downarrow \mathsf{nil} \Rightarrow L' \Downarrow \mathsf{nil}) \\ & \& \; \forall H, T \, (L \Downarrow H :: T \Rightarrow \\ & \quad \exists H', T' \, (L' \Downarrow H' :: T' \;\&\; H \, \mathcal{R}_{\sigma} \, H' \;\&\; T \, \mathcal{R}_{[\sigma]} \, T')) \end{aligned} \tag{3.3e}$$

$$B \, [\mathcal{R}]_{bool} \, B' \overset{\text{def}}{\Leftrightarrow} \forall b \in \mathbb{B} \, (B \Downarrow \underline{b} \Leftrightarrow B' \Downarrow \underline{b}) \tag{3.4a}$$

$$N \, [\mathcal{R}]_{int} \, N' \overset{\text{def}}{\Leftrightarrow} \forall n \in \mathbb{Z} \, (N \Downarrow \underline{n} \Leftrightarrow N' \Downarrow \underline{n}) \tag{3.4b}$$

$$F \, [\mathcal{R}]_{\sigma \to \sigma'} \, F' \overset{\text{def}}{\Leftrightarrow} \forall A \in Exp_{\sigma} \, (F \, A \, \mathcal{R}_{\sigma'} \, F' \, A) \tag{3.4c}$$

$$P \, [\mathcal{R}]_{\sigma \times \sigma'} \, P' \overset{\text{def}}{\Leftrightarrow} \mathsf{fst}(P) \, \mathcal{R}_{\sigma} \, \mathsf{fst}(P') \;\&\; \mathsf{snd}(P) \, \mathcal{R}_{\sigma'} \, \mathsf{snd}(P') \tag{3.4d}$$

$$\begin{aligned} L \, [\mathcal{R}]_{[\sigma]} \, L' \overset{\text{def}}{\Leftrightarrow} \;& (L \Downarrow \mathsf{nil} \Leftrightarrow L' \Downarrow \mathsf{nil}) \\ & \& \; \forall H, T \, (L \Downarrow H :: T \Rightarrow \\ & \quad \exists H', T' \, (L' \Downarrow H' :: T' \;\&\; H \, \mathcal{R}_{\sigma} \, H' \;\&\; T \, \mathcal{R}_{[\sigma]} \, T')) \\ & \& \; \forall H', T' \, (L' \Downarrow H' :: T' \Rightarrow \\ & \quad \exists H, T \, (L \Downarrow H :: T \;\&\; H \, \mathcal{R}_{\sigma} \, H' \;\&\; T \, \mathcal{R}_{[\sigma]} \, T')). \end{aligned} \tag{3.4e}$$

Clearly, $\mathcal{R} \mapsto \langle \mathcal{R} \rangle$ and $\mathcal{R} \mapsto [\mathcal{R}]$ are both monotone operators on Rel. So we can apply Theorem 3.1 and form their greatest (post-)fixed points.

Definition 3.3. A family of relations $\mathcal{S} \in Rel$ satisfying $\mathcal{S} \leq \langle \mathcal{S} \rangle$ will be called a **PCFL simulation**; the greatest such will be called **PCFL similarity** and written $\preceq$. A family of relations $\mathcal{B} \in Rel$ satisfying $\mathcal{B} \leq [\mathcal{B}]$ will be called a **PCFL bisimulation**; the greatest such will be called **PCFL bisimilarity** and written $\simeq$.

Let us spell out what the conditions $\mathcal{S} \leq \langle \mathcal{S} \rangle$ and $\mathcal{B} \leq [\mathcal{B}]$ mean. A PCFL simulation $\mathcal{S}$ is specified by a type-indexed family of binary relations, $\mathcal{S}_{\sigma} \subseteq Exp_{\sigma} \times Exp_{\sigma}$, satisfying the conditions in Figure 4. A PCFL bisimulation is specified by a type-indexed family of binary relations, $\mathcal{B}_{\sigma} \subseteq Exp_{\sigma} \times Exp_{\sigma}$, satisfying the conditions in Figure 5.

Note in particular that the notion of (bi)simulation requires one to consider evaluation at ground and list types, but not at product and function types. The reason for this is that we wish to obtain a notion of bisimilarity which coincides with PCFL contextual equivalence as defined in the previous section. Some variations on the definition of $\cong^{\text{gnd}}$ and the corresponding changes in a coextensive notion of bisimilarity will be considered in 5.

$$(B\,\mathcal{S}_{bool}\,B' \;\&\; B \Downarrow \underline{b}) \Rightarrow B' \Downarrow \underline{b} \qquad \text{(sim 1)}$$

$$(N\,\mathcal{S}_{int}\,N' \;\&\; N \Downarrow \underline{n}) \Rightarrow N' \Downarrow \underline{n} \qquad \text{(sim 2)}$$

$$F\,\mathcal{S}_{\sigma\to\sigma'}\,F' \Rightarrow \forall A \in Exp_\sigma\,(F\,A\,\mathcal{S}_{\sigma'}\,F'\,A) \qquad \text{(sim 3)}$$

$$P\,\mathcal{S}_{\sigma\times\sigma'}\,P' \Rightarrow (\mathsf{fst}(P)\,\mathcal{S}_\sigma\,\mathsf{fst}(P') \;\&\; \mathsf{snd}(P)\,\mathcal{S}_{\sigma'}\,\mathsf{snd}(P')) \qquad \text{(sim 4)}$$

$$(L\,\mathcal{S}_{[\sigma]}\,L' \;\&\; L \Downarrow \mathsf{nil}) \Rightarrow L' \Downarrow \mathsf{nil} \qquad \text{(sim 5)}$$

$$(L\,\mathcal{B}_{[\sigma]}\,L' \;\&\; L \Downarrow H :: T) \Rightarrow \qquad \text{(sim 6)}$$
$$\exists H', T'\,(L' \Downarrow H' :: T' \;\&\; H\,\mathcal{B}_\sigma\,H' \;\&\; T\,\mathcal{B}_{[\sigma]}\,T')$$

Figure 4: Simulation conditions

$$(B\,\mathcal{B}_{bool}\,B' \;\&\; B \Downarrow \underline{b}) \Rightarrow B' \Downarrow \underline{b} \qquad \text{(bis 1a)}$$

$$(B\,\mathcal{B}_{bool}\,B' \;\&\; B' \Downarrow \underline{b'}) \Rightarrow B \Downarrow \underline{b'} \qquad \text{(bis 1b)}$$

$$(N\,\mathcal{B}_{int}\,N' \;\&\; N \Downarrow \underline{n}) \Rightarrow N' \Downarrow \underline{n} \qquad \text{(bis 2a)}$$

$$(N\,\mathcal{B}_{int}\,N' \;\&\; N' \Downarrow \underline{n'}) \Rightarrow N \Downarrow \underline{n'} \qquad \text{(bis 2b)}$$

$$F\,\mathcal{B}_{\sigma\to\sigma'}\,F' \Rightarrow \forall A \in Exp_\sigma\,(F\,A\,\mathcal{B}_{\sigma'}\,F'\,A) \qquad \text{(bis 3)}$$

$$P\,\mathcal{B}_{\sigma\times\sigma'}\,P' \Rightarrow (\mathsf{fst}(P)\,\mathcal{B}_\sigma\,\mathsf{fst}(P') \;\&\; \mathsf{snd}(P)\,\mathcal{B}_{\sigma'}\,\mathsf{snd}(P')) \qquad \text{(bis 4)}$$

$$(L\,\mathcal{B}_{[\sigma]}\,L' \;\&\; L \Downarrow \mathsf{nil}) \Rightarrow L' \Downarrow \mathsf{nil} \qquad \text{(bis 5a)}$$

$$(L\,\mathcal{B}_{[\sigma]}\,L' \;\&\; L' \Downarrow \mathsf{nil}) \Rightarrow L \Downarrow \mathsf{nil} \qquad \text{(bis 5b)}$$

$$(L\,\mathcal{B}_{[\sigma]}\,L' \;\&\; L \Downarrow H :: T) \Rightarrow \qquad \text{(bis 6a)}$$
$$\exists H', T'\,(L' \Downarrow H' :: T' \;\&\; H\,\mathcal{B}_\sigma\,H' \;\&\; T\,\mathcal{B}_{[\sigma]}\,T')$$

$$(L\,\mathcal{B}_{[\sigma]}\,L' \;\&\; L' \Downarrow H' :: T') \Rightarrow \qquad \text{(bis 6b)}$$
$$\exists H, T\,(L \Downarrow H :: T \;\&\; H\,\mathcal{B}_\sigma\,H' \;\&\; T\,\mathcal{B}_{[\sigma]}\,T').$$

Figure 5: Bisimulation conditions

Remark 3.4. Note that by Theorem 3.1, PCFL similarity and bisimilarity are fixed points (rather than just post-fixed points) of their associated monotone operators, that is, $\preceq\, = \langle\preceq\rangle$ and $\simeq\, = [\simeq]$.

Proposition 3.5 (Co-induction principle for $\simeq$ and $\preceq$). *Given $M, M' \in Exp_\sigma$, to prove that $M \simeq_\sigma M'$ holds, it suffices to find a PCFL bisimulation $\mathcal{B}$ such that $M\,\mathcal{B}_\sigma\,M'$. Similarly, to prove $M \preceq_\sigma M'$, it suffices to find a PCFL simulation $\mathcal{S}$ with $M\,\mathcal{S}_\sigma\,M'$.*

Proof. If $\mathcal{B} \leq [\mathcal{B}]$, then $\mathcal{B} \leq\, \simeq$ (since $\simeq$ is the greatest post-fixed point of $[-]$), so that $\mathcal{B}_\sigma \subseteq \simeq_\sigma$. Thus if $M \mathrel{\mathcal{B}_\sigma} M'$, then $M \simeq_\sigma M'$. □

Once we have proved that bisimilarity and contextual equivalence coincide for PCFL this proposition will provide a powerful tool for proving contextual equivalences. For the moment, we use it to establish some basic facts about (bi)similarity, the last of which depends very much upon the deterministic nature of evaluation in PCFL.

Proposition 3.6. *PCFL similarity is a preorder and PCFL bisimilarity is the equivalence relation induced by it. In other words, for all types σ and all closed terms $M, M', M'' \in \mathit{Exp}_\sigma$, one has:*

(i) $M \preceq_\sigma M$

(ii) $(M \preceq_\sigma M' \;\&\; M' \preceq_\sigma M'') \Rightarrow M \preceq_\sigma M''$

(iii) $M \simeq_\sigma M' \Leftrightarrow (M \preceq_\sigma M' \;\&\; M' \preceq_\sigma M)$.

Proof. Note that the element of Rel whose component at type σ is $\{(M, M) \mid M \in \mathit{Exp}_\sigma\}$ is trivially a PCFL simulation. So (i) holds by Proposition 3.5. Similarly, to prove (ii), it suffices to check that

$$\{(M, M'') \in \mathit{Exp}_\sigma \times \mathit{Exp}_\sigma \mid \exists M' \in \mathit{Exp}_\sigma\,(M \preceq_\sigma M' \;\&\; M' \preceq_\sigma M'')\}$$

determines a PCFL simulation. But this follows immediately from the fact that $\preceq$ is itself a PCFL simulation.

For (iii), note that since $\simeq$ satisfies the bisimulation conditions in Figure 5, both $\{(M, M') \mid M \simeq_\sigma M\}$ and $\{(M, M') \mid M' \simeq_\sigma M\}$ determine PCFL simulations. Hence both are contained in $\preceq$, and thus we have the left-to-right implication in (iii). Conversely, the fact that $\preceq$ is a PCFL simulation and the fact that evaluation in PCFL is deterministic (Proposition 2.4(i)) together imply that $\{(M, M') \mid M \preceq_\sigma M' \;\&\; M' \preceq_\sigma M\}$ satisfies the conditions in Figure 5, and hence is contained in $\simeq_\sigma$. □

We extend $\preceq$ and $\simeq$ from closed terms to all typeable PCFL terms by considering closed instantiations of open terms. (Cf. the property (2.19), which we are claiming the contextual preorder satisfies.) It is convenient to introduce a notation for this process.

Definition 3.7. Suppose $\mathcal{R} \in \mathit{Rel}$. For any finite partial function Γ assigning types to variables

$$\Gamma : x_1 \mapsto \sigma_1, x_2 \mapsto \sigma_2, \ldots, x_n \mapsto \sigma_n$$

for any type σ, and for any terms $N, N' \in \mathit{Exp}_\sigma(\Gamma)$, define

$$\Gamma \vdash N \mathrel{\mathcal{R}^\circ_\sigma} N' \overset{\text{def}}{\Leftrightarrow} \forall M_1 \in \mathit{Exp}_{\sigma_1}, \ldots, M_n \in \mathit{Exp}_{\sigma_n}\,(N[\vec{M}/\vec{x}] \mathrel{\mathcal{R}_\sigma} N'[\vec{M}/\vec{x}]) \quad (3.5)$$

We will call $\mathcal{R}^\circ$ the **open extension** of $\mathcal{R}$. Applying this construction to $\preceq$ and $\simeq$, we get relations $\preceq^\circ$ and $\simeq^\circ$ on open terms, which we will still call similarity and bisimilarity respectively.

Armed with these definitions, we can state the co-inductive characterisation of contextual equivalence for PCFL.

Theorem 3.8 (Operational Extensionality for PCFL). *Contextual preorder (respectively, equivalence) coincides with similarity (respectively, bisimilarity):*

$$\Gamma \vdash M \leq^{\mathrm{gnd}}_\sigma M' \Leftrightarrow \Gamma \vdash M \preceq^\circ_\sigma M'$$
$$\Gamma \vdash M \cong^{\mathrm{gnd}}_\sigma M' \Leftrightarrow \Gamma \vdash M \simeq^\circ_\sigma M'.$$

In particular, the following **co-induction principle** *for $\cong^{\mathrm{gnd}}$ holds:*

> *To prove that two closed PCFL terms are contextually equivalent, it suffices to find a PCFL bisimulation which relates them.*

The proof of this theorem will be given in the next section. In the rest of this section we explore some of its consequences and then give some examples of the use of co-induction to prove contextual equivalences.

First note that in view of the definition of $\preceq^\circ$ from $\preceq$, the extensionality property (2.19) of $\leq^{\mathrm{gnd}}$ is an immediate consequence of Theorem 3.8. The other extensionality properties (2.20)–(2.23) also follow from the theorem, using the fact that $\preceq = \langle \preceq \rangle$.

Proposition 3.9 (Kleene equivalence). *For each type σ consider the following binary relations on Exp_σ:*

$$M \leq^{\mathrm{kl}}_\sigma M' \overset{\mathrm{def}}{\Leftrightarrow} \forall C\,(M \Downarrow C \Rightarrow M' \Downarrow C)$$
$$M \cong^{\mathrm{kl}}_\sigma M' \overset{\mathrm{def}}{\Leftrightarrow} M \leq^{\mathrm{kl}}_\sigma M' \;\&\; M' \leq^{\mathrm{kl}}_\sigma M.$$

If $M \cong^{\mathrm{kl}}_\sigma M'$ holds we will say that M and M' are **Kleene equivalent**.[1] *Then*

$$M \leq^{\mathrm{kl}}_\sigma M' \Rightarrow M \preceq_\sigma M' \tag{3.6}$$
$$M \cong^{\mathrm{kl}}_\sigma M' \Rightarrow M \simeq_\sigma M'. \tag{3.7}$$

Hence in view of Theorem 3.8, Kleene equivalent closed PCFL terms are contextually equivalent.

Proof. Property (3.7) follows from property (3.6) by Proposition 3.6(iii). For (3.6), it suffices to check that the relations $\{(M, M') \mid M \leq^{\mathrm{kl}}_\sigma M'\}$ (for each type σ) satisfy the conditions in Figure 4.

[1] Following Harper (1995), this terminology is adopted from the logic of partially defined expressions, where two such expressions are commonly said to be 'Kleene equivalent' if the first is defined if and only if the second is, and in that case they are equal.

Clearly the conditions (sim 1) and (sim 2) follow immediately from the definition of $\leq^{\mathrm{kl}}$. The conditions (sim 5) and (sim 6) are almost as straightforward, just needing the additional fact that $\leq^{\mathrm{kl}}$ is reflexive (which is evident from the definition of $\leq^{\mathrm{kl}}$).

To verify the condition (sim 4), suppose that $P \leq^{\mathrm{kl}}_{\sigma_1 \times \sigma_2} P'$ holds. For any C, if $\mathsf{fst}(P) {\Downarrow} C$ then this evaluation can only have been deduced by an application of rule ($\Downarrow$ fst) in Figure 3, so there are terms $M_i \in \mathit{Exp}_{\sigma_i}$ $(i = 1, 2)$ such that $P {\Downarrow} \langle M_1, M_2 \rangle$ and $M_1 \Downarrow C$. Then since $P \leq^{\mathrm{kl}}_{\sigma_1 \times \sigma_2} P'$, it is the case that $P' \Downarrow \langle M_1, M_2 \rangle$ and hence that $\mathsf{fst}(P') \Downarrow C$. Thus for any C, $\mathsf{fst}(P) \Downarrow C$ implies $\mathsf{fst}(P') \Downarrow C$, which is to say that $\mathsf{fst}(P) \leq^{\mathrm{kl}}_{\sigma_1} \mathsf{fst}(P')$. Similarly, one can deduce that $\mathsf{snd}(P) \leq^{\mathrm{kl}}_{\sigma_2} \mathsf{snd}(P')$. Thus condition (sim 4) does indeed hold.

The proof of the simulation condition at function types, (sim 3), is like that for product types, and is omitted. $\square$

The following Kleene equivalences all follow immediately from the definition of evaluation in PCFL (where we have suppressed type information).

$$
\begin{aligned}
(\lambda x \,.\, M)A &\cong^{\mathrm{kl}} M[A/x] \\
\mathsf{fst}(\langle M, M' \rangle) &\cong^{\mathrm{kl}} M \\
\mathsf{snd}(\langle M, M' \rangle) &\cong^{\mathrm{kl}} M' \\
\mathsf{case\ nil\ of}\ \{\mathsf{nil} \to M \mid h :: t \to N\} &\cong^{\mathrm{kl}} M
\end{aligned}
$$

$$
\begin{aligned}
\mathsf{case}\ H :: T\ \mathsf{of}\ \{\mathsf{nil} \to M \mid h :: t \to N\} &\cong^{\mathrm{kl}} N[H/h, T/t] \\
\mathsf{if\ true\ then}\ M\ \mathsf{else}\ M' &\cong^{\mathrm{kl}} M \\
\mathsf{if\ false\ then}\ M\ \mathsf{else}\ M' &\cong^{\mathrm{kl}} M' \\
\underline{n}\ \mathsf{op}\ \underline{n'} &\cong^{\mathrm{kl}} \underline{n\ \mathsf{op}\ n'} \\
\mathsf{fix}\, x \,.\, M &\cong^{\mathrm{kl}} M[\mathsf{fix}\, x \,.\, M/x].
\end{aligned}
$$

By the proposition, these are also valid for PCFL similarity; and since $\preceq^{\circ}$ is defined from $\preceq$ by taking closed instantiations, it follows that $\preceq^{\circ}$ satisfies the β rules (2.13)–(2.18) and the unfolding rule (2.26). Thus by (one half of) Theorem 3.8, $\leq^{\mathrm{gnd}}$ satisfies these rules as well.

Similarly, the fact that $\bot = \mathsf{fix}\, x \,.\, x$ does not evaluate, immediately implies

$$\bot \leq^{\mathrm{kl}}_{\sigma} M$$

holds for any $M \in \mathit{Exp}_{\sigma}$. Hence property (2.27) is also a consequence of the proposition combined with Theorem 3.8.

Proposition 3.10 (Co-induction at list types). *For any type τ, call a binary relation $\mathcal{R} \subseteq Exp_{[\tau]} \times Exp_{[\tau]}$ a $[\tau]$*-**bisimulation** *if whenever $L \mathrel{\mathcal{R}} L'$*

$$L \Downarrow \mathsf{nil} \Rightarrow L' \Downarrow \mathsf{nil} \tag{3.8}$$

$$L' \Downarrow \mathsf{nil} \Rightarrow L \Downarrow \mathsf{nil} \tag{3.9}$$

$$L \Downarrow H :: T \Rightarrow \exists H', T' \, (H \cong^{\mathrm{gnd}}_{\tau} H' \;\&\; T \mathrel{\mathcal{R}} T') \tag{3.10}$$

$$L' \Downarrow H' :: T' \Rightarrow \exists H, T \, (H \cong^{\mathrm{gnd}}_{\tau} H' \;\&\; T \mathrel{\mathcal{R}} T'). \tag{3.11}$$

Then for any $L, L' \in Exp_{[\tau]}$, $L \cong^{\mathrm{gnd}}_{[\tau]} L'$ if and only if there is some $[\tau]$-bisimulation $\mathcal{R}$ with $L \mathrel{\mathcal{R}} L'$.

Proof. First note that by Theorem 3.8, $\cong^{\mathrm{gnd}}$ is a PCFL bisimulation (since it coincides with $\simeq$). In particular it satisfies conditions (bis 5a)–(bis 6b) of Figure 5, and therefore $\{(L, L') \mid L \cong^{\mathrm{gnd}}_{[\tau]} L'\}$ is a $[\tau]$-bisimulation. This gives the 'only if' part of the proposition.

Conversely, if $\mathcal{R}$ is a $[\tau]$-bisimulation, then the fact that $\cong^{\mathrm{gnd}}$ satisfies the conditions in Figure 5 and $\mathcal{R}$ satisfies (3.8)–(3.11) implies that

$$\mathcal{B}_\sigma \stackrel{\text{def}}{=} \begin{cases} \mathcal{R} \cup \{(L, L') \mid L \cong^{\mathrm{gnd}}_{[\tau]} L'\} & \text{if } \sigma = [\tau] \\ \{(M, M') \mid M \cong^{\mathrm{gnd}}_{\sigma} M'\} & \text{otherwise} \end{cases}$$

defines a PCFL bisimulation. Thus if $L \mathrel{\mathcal{R}} L'$, then $L \mathrel{\mathcal{B}_{[\tau]}} L'$, so $L \cong^{\mathrm{gnd}}_{[\tau]} L'$ by Theorem 3.8. □

Examples

Here is a graded series of examples illustrating the use of the co-induction principle of Proposition 3.10.

Example 3.11. For any type τ, the following contextual equivalence is valid.

$$f : \tau \to \tau, x : \tau \vdash map\, f\, (iterate\, f\, x) \cong^{\mathrm{gnd}}_{[\tau]} iterate\, f\, (f\, x) \tag{3.12}$$

where

$$\begin{aligned} map &\stackrel{\text{def}}{=} \mathsf{fix}\, m \,.\, \lambda f \,.\, \lambda \ell \,. \\ &\qquad\quad \mathsf{case}\; \ell \;\mathsf{of}\; \{\mathsf{nil} \to \mathsf{nil} \mid h :: t \to (f\, h) :: (m\, f\, t)\} \\ iterate &\stackrel{\text{def}}{=} \mathsf{fix}\, i \,.\, \lambda f \,.\, \lambda x \,.\, x :: (i\, f\, (f\, x)). \end{aligned}$$

(Note that $\emptyset \vdash map : (\tau \to \tau) \to ([\tau] \to [\tau])$ and $\emptyset \vdash iterate : (\tau \to \tau) \to \tau \to [\tau]$, for any type τ.)

Proof. Intuitively, $\mathit{iterate}\, f\, x$ is a notation for the infinite list $[x, fx, f(fx), \dots]$ and $\mathit{map}\, f$ applies f to each component of a list. So one would expect that both $\mathit{map}\, f\,(\mathit{iterate}\, f\, x)$ and $\mathit{iterate}\, f\,(f\, x)$ denote the list $[fx, f(fx), f(f(fx)), \dots]$. So the contextual equivalence (3.12) is intuitively reasonable. Let us see how to prove that it holds using the co-inductive characterisation of $\cong^{\mathrm{gnd}}$.

First note that in view of the extensionality property (2.19) of $\cong^{\mathrm{gnd}}$ (which as noted above, follows from Theorem 3.8), to prove (3.12) it suffices to show for all τ, $F \in \mathit{Exp}_{\tau\to\tau}$, and $M \in \mathit{Exp}_\tau$ that

$$\mathit{map}\, F\,(\mathit{iterate}\, F\, M) \simeq_{[\tau]} \mathit{iterate}\, F\,(F\, M). \tag{3.13}$$

We deduce this from Proposition 3.10 by constructing a suitable $[\tau]$-bisimulation, $\mathcal{R}$. In fact we do not have to look very far for $\mathcal{R}$ in this case, since the pairs of terms we are interested in already constitute a $[\tau]$-bisimulation! Let us verify that for

$$\mathcal{R} \stackrel{\text{def}}{=} \{(\mathit{map}\, F\,(\mathit{iterate}\, F\, M), \mathit{iterate}\, F\,(F\, M)) \mid F \in \mathit{Exp}_{\tau\to\tau}\ \&\ M \in \mathit{Exp}_\tau\}$$

if $L\, \mathcal{R}\, L'$, then conditions (3.8)–(3.11) are satisfied.

If $L\, \mathcal{R}\, L'$, then by definition of $\mathcal{R}$ we have that $L = \mathit{map}\, F\,(\mathit{iterate}\, F\, M)$ and $L' = \mathit{iterate}\, F\,(F\, M)$, for some terms M and F. Using the evaluation rules in Figure 3 together with the definitions of $\mathit{iterate}$ and map, one obtains

$$\begin{aligned} \mathit{iterate}\, F\, M &\Downarrow M :: (\mathit{iterate}\, F\,(F\, M)) \\ L' = \mathit{iterate}\, F\,(F\, M) &\Downarrow (F\, M) :: (\mathit{iterate}\, F\,(F\,(F\, M))) \\ L = \mathit{map}\, F\,(\mathit{iterate}\, F\, M) &\Downarrow (F\, M) :: (\mathit{map}\, F\,(\mathit{iterate}\, F\,(F\, M))). \end{aligned}$$

So by determinacy of evaluation (Proposition 2.4(i)), if $L \Downarrow C$, then $C = H :: T$ with $H = F\, M$ and $T = \mathit{map}\, F\,(\mathit{iterate}\, F\,(F\, M))$. But we know that $L' \Downarrow H' :: T'$ with $H' = F\, M = H$ and $T' = \mathit{iterate}\, F\,(F\,(F\, M))$, and hence with $H \cong^{\mathrm{gnd}}_\tau H'$ (since $\cong^{\mathrm{gnd}}$ is reflexive) and $T\, \mathcal{R}\, T'$ (by definition of $\mathcal{R}$). So conditions (3.8) and (3.10) are satisfied (the first one trivially, because by determinacy of $\Downarrow$, L does not evaluate to nil). A symmetrical argument starting with the assumption that $L' \Downarrow C'$ shows that conditions (3.9) and (3.11) are also satisfied by $\mathcal{R}$. So $\mathcal{R}$ is indeed a $[\tau]$-bisimulation, and since it relates the terms in which we are interested, the proof is complete. □

The next example makes use of mathematical induction in order to verify that a particular relation has the properties required of a bisimulation.

Example 3.12. Define

$$
\begin{aligned}
zip &\stackrel{\text{def}}{=} \mathsf{fix}\, z\,.\,\lambda\ell\,.\,\lambda\ell'\,.\,\mathsf{case}\ \ell\ \mathsf{of}\\
&\qquad \{\mathsf{nil} \to \mathsf{nil}\\
&\qquad \mid h :: t \to \mathsf{case}\ \ell'\ \mathsf{of}\ \ \{\mathsf{nil} \to \mathsf{nil}\\
&\qquad\qquad\qquad\qquad\qquad \mid h' :: t' \to \langle h, h'\rangle :: z\, t\, t'\}\}\\
\mathit{from} &\stackrel{\text{def}}{=} \mathsf{fix}\, f\,.\,\lambda x\,.\,\lambda y\,.\,x :: (f\,(x+y)\,y)\\
suc &\stackrel{\text{def}}{=} \lambda x\,.\,x + \underline{1}\\
plus &\stackrel{\text{def}}{=} \lambda z\,.\,\mathsf{fst}(z) + \mathsf{snd}(z)\\
nats &\stackrel{\text{def}}{=} \mathsf{fix}\,\ell\,.\,\underline{0} :: (map\ suc\ \ell)
\end{aligned}
$$

where map is as in the previous example. (Note that $\emptyset \vdash zip : [\sigma]\to([\sigma']\to[\sigma\times\sigma'])$ for any types σ and σ'; and $\emptyset \vdash \mathit{from} : \mathit{int} \to (\mathit{int} \to [\mathit{int}])$, $\emptyset \vdash suc : \mathit{int} \to \mathit{int}$, $\emptyset \vdash plus : (\mathit{int} \times \mathit{int}) \to \mathit{int}$, and $\emptyset \vdash nats : [\mathit{int}]$.) Then

$$map\ plus\ (zip\ nats\ nats) \cong^{\text{gnd}}_{[\mathit{int}]} \mathit{from}\ \underline{0}\,\underline{2} \tag{3.14}$$

Proof. Consider the following closed PCFL terms, defined by induction on $n \in \mathbb{N}$:

$$
\begin{array}{lll}
N_0 \stackrel{\text{def}}{=} \underline{0} & E_0 \stackrel{\text{def}}{=} \underline{0} & L_0 \stackrel{\text{def}}{=} nats\\
N_{n+1} \stackrel{\text{def}}{=} suc\ N_n & E_{n+1} \stackrel{\text{def}}{=} E_n + \underline{2} & L_{n+1} \stackrel{\text{def}}{=} map\ suc\ L_n.
\end{array}
$$

From the definition of from and E_n it follows directly that

$$\mathit{from}\ E_n\ \underline{2} \Downarrow E_n :: (\mathit{from}\ E_{n+1}\ \underline{2}) \tag{3.15}$$

From the definition of map, $nats$, L_n, and N_n it follows by induction on n that

$$L_n \Downarrow N_n :: L_{n+1}$$

Therefore, using the definition of zip, we have that

$$zip\ L_n\ L_n \Downarrow \langle N_n, N_n\rangle :: (zip\ L_{n+1}\ L_{n+1})$$

and from the definition of map that

$$map\ plus\ (zip\ L_n\ L_n) \Downarrow plus\ \langle N_n, N_n\rangle :: (map\ plus\ (zip\ L_{n+1}\ L_{n+1})) \tag{3.16}$$

Finally, note that by induction on n, $plus\ \langle N_n, N_n\rangle$ is Kleene equivalent to E_n, and hence by Proposition 3.9

$$plus\ \langle N_n, N_n\rangle \cong^{\text{gnd}}_{\mathit{int}} E_n \tag{3.17}$$

So if we define $\mathcal{R} \subseteq Exp_{[\mathit{int}]} \times Exp_{[\mathit{int}]}$ by

$$\mathcal{R} \stackrel{\text{def}}{=} \{(map\ plus\ (zip\ L_n\ L_n), \mathit{from}\ E_n\ \underline{2}) \mid n \in \mathbb{N}\}$$

then properties (3.15), (3.16), and (3.17) together with determinacy of evaluation (Proposition 2.4(i)) show that $\mathcal{R}$ is a $[\mathit{int}]$-bisimulation. Since by definition of L_0 and E_0, $\mathcal{R}$ relates the two terms of type $[\mathit{int}]$ in which we are interested, the proof of (3.14) via Proposition 3.10 is complete. □

In the next example, in order to verify that a certain relation is a bisimulation we make use of the congruence property of $\cong^{\mathrm{gnd}}$, namely that if two terms are contextually equivalent and they are substituted for a parameter in a context, the resulting terms are also contextually equivalent. (This is an easy consequence of the definition of contextual equivalence.)

Example 3.13 (The '$take$-lemma'). For any type τ and terms $L, L' \in Exp_{[\tau]}$, the following property holds.

$$\forall n \in \mathbb{N}\,(take\;\underline{n}\;L \cong^{\mathrm{gnd}}_{[} \tau]take\;\underline{n}\;L') \Rightarrow L \cong^{\mathrm{gnd}}_{[\tau]} L' \tag{3.18}$$

where

$$\begin{aligned} take \stackrel{\text{def}}{=}\;\; & \mathsf{fix}\, f\,.\,\lambda x\,.\,\lambda \ell\,.\,\mathsf{if}\; x = \underline{0}\;\mathsf{then\;nil\;else} \\ & \qquad \mathsf{case}\;\ell\;\mathsf{of}\;\{\mathsf{nil} \to \mathsf{nil} \mid h :: t \to h :: (f\,(x - \underline{1})\,t)\} \end{aligned}$$

(Note that $\emptyset \vdash take : int \to ([\tau] \to [\tau])$, for any type τ.)

This property allows one to establish instances of contextual equivalence between list expressions by appeal to mathematical induction: to prove $L \cong^{\mathrm{gnd}}_{[\tau]} L'$, it suffices to prove

$$take\;\underline{0}\;L \cong^{\mathrm{gnd}}_{[\tau]} take\;\underline{0}\;L' \tag{3.19}$$

$$take\;\underline{n}\;L \cong^{\mathrm{gnd}}_{[\tau]} take\;\underline{n}\;L' \Rightarrow take\;\underline{n+1}\;L \cong^{\mathrm{gnd}}_{[\tau]} take\;\underline{n+1}\;L' \tag{3.20}$$

For the informal theory of equality of functional programs discussed by Bird and Wadler (1988), this induction principle is called the '$take$-lemma' (see *loc. cit.*, section 7.5.1). It is used to justify equalities like the ones in the previous examples. But why is the $take$-lemma valid? As we now show, if "equal" means contextually equivalent, then one can prove the validity of this principle by appealing to the co-inductive characterisation of $\cong^{\mathrm{gnd}}$ at list types given by Proposition 3.10. (See also Gordon 1995b, Section 4.6.)

Proof. Fixing the type τ, define $\mathcal{R} \subseteq Exp_{[\tau]} \times Exp_{[\tau]}$ by:

$$\mathcal{R} \stackrel{\text{def}}{=} \{(L, L') \mid \forall n \in \mathbb{N}\,(take\;\underline{n}\;L \cong^{\mathrm{gnd}}_{[\tau]} take\;\underline{n}\;L')\}$$

I claim that $\mathcal{R}$ satisfies the conditions (3.8)–(3.11) required of a $[\tau]$-bisimulation.

First note that the evaluation rules in Figure 3 imply the following properties of $take$. For all $n \in \mathbb{N}$, $H \in Exp_\tau$, and $L, T \in Exp_{[\tau]}$:

$$take\;\underline{n+1}\;L \Downarrow \mathsf{nil} \Leftrightarrow L \Downarrow \mathsf{nil} \tag{3.21}$$

$$take\;\underline{n+1}\;L \Downarrow H :: T \Leftrightarrow \exists T'\,(L \Downarrow H :: T' \;\&\; T = take\;(\underline{n+1} - \underline{1})\;T') \tag{3.22}$$

Now suppose $L\;\mathcal{R}\;L'$, that is, for all $n \in \mathbb{N}$, $take\;\underline{n}\;L \cong^{\mathrm{gnd}}_{[\tau]} take\;\underline{n}\;L'$.

To see that $\mathcal{R}$ satisfies (3.8), suppose $L \Downarrow \mathsf{nil}$. Then by (3.21), $take\;\underline{1}\;L \Downarrow \mathsf{nil}$. Since $L\;\mathcal{R}\;L'$, by definition of $\mathcal{R}$, $take\;\underline{1}\;L \cong^{\mathrm{gnd}}_{[\tau]} take\;\underline{1}\;L'$. Since $\cong^{\mathrm{gnd}}$ is a PCFL bisimulation (by Theorem 3.8), it follows that $\mathit{take}\,\underline{1}\,L' \Downarrow \mathsf{nil}$ and hence by (3.21) again, that $L' \Downarrow \mathsf{nil}$. A symmetrical argument shows that $\mathcal{R}$ also satisfies (3.9).

To see that $\mathcal{R}$ satisfies (3.10), suppose $L \Downarrow H :: T$. Then by (3.22), for any $n \in \mathbb{N}$ we have $\mathit{take}\,\underline{n+1}\,L \Downarrow H :: (\mathit{take}\,(\underline{n+1} - \underline{1})\,T)$. Since $L\,\mathcal{R}\,L'$, by definition of $\mathcal{R}$, $\mathit{take}\,\underline{n+1}\,L \cong^{\mathrm{gnd}}_{[\tau]} \mathit{take}\,\underline{n+1}\,L'$. So since $\cong^{\mathrm{gnd}}$ is a PCFL bisimulation, it follows that there are terms H', T'' with

$$\mathit{take}\,\underline{n+1}\,L' \Downarrow H' :: T'' \;\&\; H \cong^{\mathrm{gnd}}_{\tau} H' \;\&\; \mathit{take}\,(\underline{n+1} - \underline{1})\,T \cong^{\mathrm{gnd}}_{[\tau]} T''.$$

By (3.22) again, $L' \Downarrow H' :: T'$ for some term T' with $T'' = \mathit{take}\,(\underline{n+1} - \underline{1})\,T'$. We have to check that $T\,\mathcal{R}\,T'$. Note that for all n we have $\mathit{take}\,(\underline{n+1} - \underline{1})\,T \cong^{\mathrm{gnd}}_{[\tau]} T'' = \mathit{take}\,(\underline{n+1} - \underline{1})\,T'$. To conclude from this that $\forall n \in \mathbb{N}\,(\mathit{take}\,\underline{n}\,T \cong^{\mathrm{gnd}}_{[\tau]} \mathit{take}\,\underline{n}\,T')$, we use the congruence property (2.11) of $\cong^{\mathrm{gnd}}$ (which as we noted in section 2 is an easy consequence of the definition of $\cong^{\mathrm{gnd}}$). It allows us to infer $\mathit{take}\,(\underline{n+1} - \underline{1})\,T \cong^{\mathrm{gnd}}_{[\tau]} \mathit{take}\,\underline{n}\,T$ from the fact that $(\underline{n+1} - \underline{1}) \cong^{\mathrm{gnd}}_{\mathit{int}} \underline{n}$. (The latter holds by Proposition 3.9, since evidently $\underline{n+1} - \underline{1}$ is Kleene equivalent to $\underline{n}$.) A symmetrical argument shows that $\mathcal{R}$ also satisfies (3.11).

We have now established that $\mathcal{R}$ is a $[\tau]$-bisimulation. So by Proposition 3.10, for all $L, L' \in \mathit{Exp}_{[\tau]}$, if $L\,\mathcal{R}\,L'$ then $L \cong^{\mathrm{gnd}}_{[\tau]} L'$, as required for (3.18). □

The next example is somewhat more challenging than the previous ones, in as much as the verification of the bisimulation condition involves an induction on the **depths of proofs** of evaluation. We write

$$M \Downarrow^n C$$

to indicate that there is a proof tree for $M \Downarrow C$ (built out of the axioms and rules in Figure 3) whose depth is less than or equal to n. If the reader prefers, one can give a slightly more abstract definition of the relations $\Downarrow^n$ ($n \in \mathbb{N}$): they are simultaneously inductively defined by axioms and rules obtained from those in Figure 3 by replacing $\Downarrow$ by $\Downarrow^n$ in each axiom and each hypothesis of a rule, and by replacing $\Downarrow$ by $\Downarrow^{n+1}$ in the conclusion of each rule. Clearly it is the case that

$$M \Downarrow C \Leftrightarrow \exists n \in \mathbb{N}\,(M \Downarrow^n C) \tag{3.23}$$

Example 3.14. For any type τ, the following contextual equivalence is valid.

$$u : \tau \to \mathit{bool}, v : \tau \to \tau, \ell : [\tau] \vdash \mathit{filter}\,u\,(\mathit{map}\,v\,\ell) \cong^{\mathrm{gnd}}_{[\tau]} \mathit{map}\,v\,(\mathit{filter}\,(u \circ v)\,\ell) \tag{3.24}$$

where map is as in Example 3.11 and

$$\begin{aligned} \mathit{filter} &\stackrel{\text{def}}{=} \mathsf{fix}\,f\,.\,\lambda u\,.\,\lambda \ell\,.\,\mathsf{case}\,\ell\,\mathsf{of} \\ &\qquad \{\mathsf{nil} \to \mathsf{nil} \mid h :: t \to \mathsf{if}\,u\,h\,\mathsf{then}\,h :: (f\,u\,t)\,\mathsf{else}\,f\,u\,t\} \\ u \circ v &\stackrel{\text{def}}{=} \lambda x\,.\,u\,(v\,x) \end{aligned}$$

(Note that $\emptyset \vdash \mathit{filter} : (\tau \to \mathit{bool}) \to ([\tau] \to [\tau])$ and that $u : \tau' \to \tau'', v : \tau \to \tau' \vdash u \circ v : \tau \to \tau''$, for any types τ, τ', τ''.)

Proof. Intuitively, the expression $\mathit{filter}\, u$ is a notation for the function on lists which removes elements of the list which fail the boolean test u. It is an inherently partial function, because as one progressively evaluates an input lazy list, one may never find an element passing the boolean test with which to begin the output list. It is mainly for this reason that (3.24) is harder to prove than the previous examples.

Just as in Example 3.11, to prove (3.24) it suffices to prove for all types τ, and closed terms $U \in \mathit{Exp}_{\tau \to bool}$ and $V \in \mathit{Exp}_{\tau \to \tau}$, that

$$\mathit{filter}\, U\, (\mathit{map}\, V\, L) \simeq_{[\tau]} \mathit{map}\, V\, (\mathit{filter}\, (U \circ V)\, L) \tag{3.25}$$

holds for all $L \in \mathit{Exp}_{[\tau]}$. Given τ, U, and V, define $\mathcal{R} \subseteq \mathit{Exp}_{[\tau]} \times \mathit{Exp}_{[\tau]}$ by:

$$\mathcal{R} \stackrel{\text{def}}{=} \{(\mathit{filter}\, U\, (\mathit{map}\, V\, L), \mathit{map}\, V\, (\mathit{filter}\, (U \circ V)\, L)) \mid L \in \mathit{Exp}_{[\tau]}\}.$$

To establish (3.25), by Proposition 3.10 it suffices to show that this $\mathcal{R}$ is a $[\tau]$-bisimulation. Instead of proving that the conditions (3.8)–(3.11) hold for $\mathcal{R}$ directly (which does not seem possible—try it and see), we can deduce them via (3.23), using the following properties of $\Downarrow^n$:

$$\forall L\, (\mathit{filter}\, U\, (\mathit{map}\, V\, L) \Downarrow^n \mathsf{nil} \Rightarrow \mathit{map}\, V\, (\mathit{filter}\, (U \circ V)\, L) \Downarrow \mathsf{nil}) \tag{3.26}$$

$$\forall L\, (\mathit{map}\, V\, (\mathit{filter}\, (U \circ V)\, L) \Downarrow^n \mathsf{nil} \Rightarrow \mathit{filter}\, U\, (\mathit{map}\, V\, L) \Downarrow \mathsf{nil}) \tag{3.27}$$

$$\begin{aligned}&\forall L, H, T\, (\mathit{filter}\, U\, (\mathit{map}\, V\, L) \Downarrow^n H :: T \Rightarrow \\ &\qquad \exists T'\, (\mathit{map}\, V\, (\mathit{filter}\, (U \circ V)\, L) \Downarrow H :: T'\ \&\ T\, \mathcal{R}\, T'))\end{aligned} \tag{3.28}$$

$$\begin{aligned}&\forall L, H, T'\, (\mathit{map}\, V\, (\mathit{filter}\, (U \circ V)\, L) \Downarrow^n H :: T' \Rightarrow \\ &\qquad \exists T\, (\mathit{filter}\, U\, (\mathit{map}\, V\, L) \Downarrow H :: T\ \&\ T\, \mathcal{R}\, T'))\end{aligned} \tag{3.29}$$

Each of (3.26)–(3.29) can be proved by induction on n. We give the argument for (3.28) and leave the other three as exercises.

So assume inductively that (3.28) holds for all $n < m$. If $\mathit{filter}\, U\, (\mathit{map}\, V\, L) \Downarrow^m H :: T$, then by definition of filter, $m \geq 2$ and there are terms H_1, T_1 so that

$$\mathit{map}\, V\, L \Downarrow^{m-2} H_1 :: T_1 \tag{3.30}$$

$$\mathsf{if}\ U\, H_1\ \mathsf{then}\ H_1 :: (\mathit{filter}\, U\, T_1)\ \mathsf{else}\ \mathit{filter}\, U\, T_1 \Downarrow^{m-2} H :: T \tag{3.31}$$

Then by definition of map, for (3.30) to hold, it must be the case that $m \geq 4$ and there are terms H_2, T_2 so that

$$L \Downarrow^{m-4} H_2 :: T_2 \tag{3.32}$$

$$H_1 = V\, H_2 \quad \text{and} \quad T_1 = \mathit{map}\, V\, T_2. \tag{3.33}$$

On the other hand, since (3.31) holds, it must be the case that either $U\, H_1 \Downarrow \mathsf{true}$ or $U\, H_1 \Downarrow \mathsf{false}$. We treat each case separately.

Case $U\ H_1 \Downarrow \mathsf{true}$. In this case (3.31) holds because

$$H = H_1 \quad \text{and} \quad T = \mathit{filter}\, U\, T_1. \tag{3.34}$$

Combining this with (3.33) we get $(U \circ V)\ H_2 \Downarrow \mathsf{true}$. Then together with (3.32), this yields

$$\mathit{filter}\ (U \circ V)\ L \Downarrow H_2 :: \mathit{filter}\ (U \circ V)\ T_2$$

and hence

$$\mathit{map}\, V\ (\mathit{filter}\ (U \circ V)\ L) \Downarrow V\ H_2 :: (\mathit{map}\, V\ (\mathit{filter}\ (U \circ V)\ T_2)).$$

Since by (3.33) and (3.34) we have $H = V\ H_2$ and $T = \mathit{filter}\, U\ (\mathit{map}\, V\, T_2)$, it follows that

$$\begin{gathered}\mathit{map}\, V\ (\mathit{filter}\ (U \circ V)\ L) \Downarrow H :: (\mathit{map}\, V\ (\mathit{filter}\ (U \circ V)\ T_2)) \\ T\ \mathcal{R}\ \mathit{map}\, V\ (\mathit{filter}\ (U \circ V)\ T_2).\end{gathered}$$

So the conclusion of the $n = m$ case of (3.28) holds with $T' = \mathit{map}\, V\ (\mathit{filter}\ (U \circ V)\ T_2)$.

Case $U\ H_1 \Downarrow \mathsf{false}$. In this case (3.31) holds because $m \geq 3$ and $\mathit{filter}\, U\, T_1 \Downarrow^{m-3} H :: T$. Hence by (3.33), $\mathit{filter}\, U\ (\mathit{map}\, V\, T_2) \Downarrow^{m-3} H :: T$. So by induction hypothesis there is some T_3 with

$$\mathit{map}\, V\ (\mathit{filter}\ (U \circ V)\ T_2) \Downarrow H :: T_3 \tag{3.35}$$

$$T\ \mathcal{R}\ T_3. \tag{3.36}$$

By definition of map, for (3.35) to hold, it must be the case that there are terms H_4, T_4 with

$$\mathit{filter}\ (U \circ V)\ T_2 \Downarrow H_4 :: T_4 \tag{3.37}$$

$$H = V\ H_4 \quad \text{and} \quad T_3 = \mathit{map}\, V\, T_4. \tag{3.38}$$

Since we are assuming that $U\ H_1 \Downarrow \mathsf{false}$, by (3.33) we also have $(U \circ V)\ H_2 \Downarrow \mathsf{false}$. Then by (3.32) and (3.37), we have $\mathit{filter}\ (U \circ V)\ L \Downarrow H_4 :: T_4$ and hence

$$\mathit{map}\, V\ (\mathit{filter}\ (U \circ V)\ L) \Downarrow V\ H_4 :: (\mathit{map}\, V\, T_4).$$

So by (3.36) and (3.38)

$$\mathit{map}\, V\ (\mathit{filter}\ (U \circ V)\ L) \Downarrow H :: T_3\ \&\ T\ \mathcal{R}\ T_3.$$

Thus the conclusion of the $n = m$ case of (3.28) holds with $T' = T_3$. □

Exercises

Consider the following PCFL terms.

$$\begin{aligned}
map &\stackrel{\text{def}}{=} \mathsf{fix}\, m\,.\,\lambda u\,.\,\lambda \ell\,.\,\mathsf{case}\;\ell\;\mathsf{of}\;\{\mathsf{nil} \to \mathsf{nil} \mid h :: t \to (u\,h) :: (m\,u\,t)\}\\
nats &\stackrel{\text{def}}{=} \mathsf{fix}\, \ell\,.\,\underline{0} :: map(\lambda x\,.\,x + \underline{1})\ell\\
from &\stackrel{\text{def}}{=} \mathsf{fix}\, f\,.\,\lambda x\,.\,\lambda y\,.\,x :: (f\;(x+y)\;y)\\
append &\stackrel{\text{def}}{=} \mathsf{fix}\, a\,.\,\lambda \ell\,.\,\lambda \ell'\,.\,\mathsf{case}\;\ell\;\mathsf{of}\;\{\mathsf{nil} \to \ell' \mid h :: t \to h :: (a\;t\;\ell')\}\\
interleave &\stackrel{\text{def}}{=} \mathsf{fix}\, i\,.\,\lambda \ell\,.\,\lambda \ell'\,.\,\mathsf{case}\;\ell\;\mathsf{of}\;\{\mathsf{nil} \to \ell' \mid h :: t \to h :: (i\;\ell'\,t)\}\\
val &\stackrel{\text{def}}{=} \lambda x\,.\,x :: \mathsf{nil}\\
lift &\stackrel{\text{def}}{=} \mathsf{fix}\, f\,.\,\lambda u\,.\,\lambda \ell\,.\,\mathsf{case}\;\ell\;\mathsf{of}\;\{\mathsf{nil} \to \mathsf{nil} \mid h :: t \to append(u\,h)(f\;\;u\,t)\}\\
u \circ v &\stackrel{\text{def}}{=} \lambda x\,.\,u\,(v\,x).
\end{aligned}$$

Prove the following contextual equivalences.

$$\emptyset \vdash nats \cong^{\text{gnd}}_{[int]} from\,\underline{0}\,\underline{1} \tag{3.39}$$

$$x : int \vdash from\,x\,\underline{1} \cong^{\text{gnd}}_{[int]} interleave\,(from\,x\,\underline{2})\,(from\,(x+1)\,\underline{2}) \tag{3.40}$$

$$x : \tau, u : \tau \to [\tau'] \vdash lift\,u\,(val\,x) \cong^{\text{gnd}}_{[\tau']} f\,x \tag{3.41}$$

$$\ell : [\tau] \vdash lift\;val\;\ell \cong^{\text{gnd}}_{[\tau]} \ell \tag{3.42}$$

$$u : \tau \to [\tau'], v : \tau' \to [\tau''], \ell : [\tau] \vdash$$
$$lift\,((lift\,v) \circ u)\,\ell \cong^{\text{gnd}}_{[\tau'']} lift\,v\,(lift\,u\,\ell) \tag{3.43}$$

These last three equivalences are respectively the β, η and associativity identities for the Kleisli triple corresponding to the (strong) monad structure of lazy lists (see Moggi 1991 and Wadler 1992). Prove (3.41) by applying Proposition 3.9. Prove (3.42) and (3.43) by constructing suitable bisimulations. (3.43) is quite challenging: you will need to employ techniques like those in Example 3.14, involving an induction over depths of proofs of evaluation, in order to verify the bisimulation conditions. (See also Gordon 1995b, Section 4.5.)

4 Rational Completeness and Syntactic Continuity

In this section we prove that the PCFL contextual preorder $\leq^{\text{gnd}}$ satisfies the properties (2.29) and (2.30) mentioned at the end of section 2—namely that each fixpoint term $\mathsf{fix}\,x\,.\,F$ is the least upper bound with respect to $\leq^{\text{gnd}}$ of a canonically associated chain of approximations, and these least upper bounds are preserved by the PCFL language constructs.

Although it is beyond the scope of these notes to pursue the topic, these properties form the basis for transferring various domain-theoretic verification techniques (such as the induction principle of Scott 1993, section 3) from the denotational semantics of a functional language to the language itself equipped with an operational semantics. Which is not to say that the denotational semantics of a language is without its uses. Indeed one way to establish (2.29) and (2.30) is via a computationally adequate denotational semantics of PCFL: cf. (Pitts 1994). Here we give a proof directly from the operational semantics of PCFL, as specified by the evaluation relation of Figure 3. Mason, Smith, and Talcott (1996) achieve similar results for an untyped, call-by-value functional language, making use of a one-step transition relation rather than an evaluation relation. The differences between the language treated in *loc. cit.* and the one used in these notes are not particularly relevant to the proof of the properties in question: the method given below could easily be adapted to untyped languages and/or ones with call-by-value function application. It can also be used to prove completeness and continuity properties for some of the variations on contextual preordering and similarity mentioned in section 5.

As in *loc. cit.*, the proof of (2.29) and (2.30) given here hinges upon a certain 'compactness' property of $\Downarrow$ with respect to fixpoint terms (Corollary 4.6). However, we deduce this compactness property from an apparently stronger property of evaluation, Proposition 4.5, which seems to be new. Unfortunately, it is beyond the scope of these notes to present further applications of Proposition 4.5.

Notational conventions. Throughout this section we will consider a particular fixpoint term $\mathsf{fix}\, x \,.\, F$, of type τ say, and use the following abbreviations.

$$\begin{aligned} F_0 &\stackrel{\text{def}}{=} \mathsf{fix}^{(0)} x \,.\, F \stackrel{\text{def}}{=} \bot \stackrel{\text{def}}{=} \mathsf{fix}\, x \,.\, x \\ F_{n+1} &\stackrel{\text{def}}{=} \mathsf{fix}^{(n+1)} x \,.\, F \stackrel{\text{def}}{=} F[F_n/x] \\ F_\omega &\stackrel{\text{def}}{=} \mathsf{fix}\, x \,.\, F \ . \end{aligned}$$

We will only consider PCFL contexts involving parameters of type τ. As usual, we write $\mathcal{M}[\vec{\mathsf{p}}]$ to indicate such a context whose parameters are contained in the list $\vec{\mathsf{p}} = \mathsf{p}_1, \ldots, \mathsf{p}_k$ of pairwise distinct parameters. Given a k-tuple $\vec{n} = (n_1, \ldots, n_k)$ of natural numbers then we will make the following abbreviations.

$$\begin{aligned} \mathcal{C}[F_{\vec{n}}] &\stackrel{\text{def}}{=} \mathcal{C}[F_{n_1}, \ldots, F_{n_k}] \\ \mathcal{C}[F_\omega] &\stackrel{\text{def}}{=} \mathcal{C}[F_\omega, \ldots, F_\omega] \ . \end{aligned}$$

Finally, the length of a list $\vec{\mathsf{p}}$ of parameters will be denoted by $|\vec{\mathsf{p}}|$.

Definition 4.1. For each k, we partially order the set $\mathbb{N}^k$ of k-tuples of natural numbers componentwise from the usual ordering on $\mathbb{N}$:

$$\vec{n} \le \vec{n}' \stackrel{\text{def}}{\Leftrightarrow} (n_1 \le n_1' \ \& \ldots \& \ n_k \le n_k') \ .$$

A subset $I \subseteq \mathbb{N}^k$ is said to be **cofinal** in $\mathbb{N}^k$ if and only if for all $\vec{n} \in \mathbb{N}^k$ there is some $\vec{n}' \in I$ with $\vec{n} \leq \vec{n}'$. We will write $\mathcal{P}_{\mathrm{cof}}(\mathbb{N}^k)$ for the set of all such cofinal subsets of $\mathbb{N}^k$.

Note that by induction on n using (2.27) and (2.11), one can prove

$$F_n \leq^{\mathrm{gnd}} F_{n+1} : \tau \quad \text{and} \quad F_n \leq^{\mathrm{gnd}} F_\omega : \tau$$

and hence that for each unary context $\mathcal{C}[\mathsf{p}]$, there is a $\leq^{\mathrm{gnd}}$-ascending chain

$$\mathcal{C}[F_0] \leq^{\mathrm{gnd}} \mathcal{C}[F_1] \leq^{\mathrm{gnd}} \mathcal{C}[F_2] \leq^{\mathrm{gnd}} \cdots$$

bounded above by $\mathcal{C}[F_\omega]$. We aim to show that $\mathcal{C}[F_\omega]$ is in fact the least upper bound of this chain. If that is the case, then note that more generally if $\mathcal{C}$ involves several parameters $\vec{\mathsf{p}}$, then for any cofinal subset $I \subseteq \mathbb{N}^{|\vec{\mathsf{p}}|}$, $\mathcal{C}[F_\omega]$ will be the least upper bound of the set $\{\mathcal{C}[F_{\vec{n}}] \mid \vec{n} \in I\}$. In fact it turns out to be convenient to prove this stronger least upper bound property directly and then deduce (2.30) (and hence also (2.29)) as a special case.

The following notion of evaluation is somewhat technical: its introduction is justified below by Proposition 4.5.

Definition 4.2 (Evaluation of contexts, mod F). Given PCFL contexts $\mathcal{M}[\vec{\mathsf{p}}]$ and $\mathcal{C}[\vec{\mathsf{p}}']$, we write $\mathcal{M}[\vec{\mathsf{p}}] \Downarrow^F \mathcal{C}[\vec{\mathsf{p}}']$ to mean that for all $I \in \mathcal{P}_{\mathrm{cof}}(\mathbb{N}^{|\vec{\mathsf{p}}|})$

$$\{\vec{n}\vec{n}' \mid \vec{n} \in I \;\&\; \mathcal{M}[F_{\vec{n}}] \Downarrow \mathcal{C}[F_{\vec{n}'}]\} \in \mathcal{P}_{\mathrm{cof}}(\mathbb{N}^{|\vec{\mathsf{p}}|+|\vec{\mathsf{p}}'|}) \ .$$

Note that the relation $\mathcal{M}[\vec{\mathsf{p}}] \Downarrow^F \mathcal{C}[\vec{\mathsf{p}}']$ is preserved under renaming the parameters $\vec{\mathsf{p}}$ and, independently, the parameters $\vec{\mathsf{p}}'$. As the following lemma shows, the relation is also preserved under addition or subtraction of extra parameters.

Lemma 4.3.

$$\mathcal{M}[\vec{\mathsf{p}}] \Downarrow^F \mathcal{C}[\vec{\mathsf{p}}'] \Leftrightarrow \mathcal{M}[\vec{\mathsf{p}}\vec{\mathsf{q}}] \Downarrow^F \mathcal{C}[\vec{\mathsf{p}}'\vec{\mathsf{q}}'] \ .$$

Proof. This property follows from the definition of $\Downarrow^F$ together with simple properties of cofinal subsets of $\mathbb{N}^k$. □

Lemma 4.4. *The relation $\Downarrow^F$ satisfies the following analogues of the axioms and rules for PCFL evaluation given in Figure 3.*

(i) If $\mathcal{C}$ is in canonical form (that is, a constant, lambda abstraction, pair, or cons expression), then $\mathcal{C}[\vec{\mathsf{p}}] \Downarrow^F \mathcal{C}[\vec{\mathsf{p}}]$.

(ii) If $\mathcal{B}[\vec{\mathsf{p}}] \Downarrow^F \mathsf{true}[\,]$ and $\mathcal{M}_1[\vec{\mathsf{p}}] \Downarrow^F \mathcal{C}[\vec{\mathsf{p}}']$, then $(\mathsf{if}\,\mathcal{B}\,\mathsf{then}\,\mathcal{M}_1\,\mathsf{else}\,\mathcal{M}_2)[\vec{\mathsf{p}}] \Downarrow^F \mathcal{C}[\vec{\mathsf{p}}']$.

(iii) If $\mathcal{B}[\vec{\mathsf{p}}] \Downarrow^F \mathsf{false}[\,]$ and $\mathcal{M}_2[\vec{\mathsf{p}}] \Downarrow^F \mathcal{C}[\vec{\mathsf{p}}']$, then $(\mathsf{if}\,\mathcal{B}\,\mathsf{then}\,\mathcal{M}_1\,\mathsf{else}\,\mathcal{M}_2)[\vec{\mathsf{p}}] \Downarrow^F \mathcal{C}[\vec{\mathsf{p}}']$.

(iv) If $\mathcal{M}_i[\vec{\mathsf{p}}] \Downarrow^F \underline{n_i}[\,]$ for $i = 1, 2$, then $(\mathcal{M}_1\,\mathsf{op}\,\mathcal{M}_2)[\vec{\mathsf{p}}] \Downarrow^F \underline{c}[\,]$, where $c = n_1\,\mathsf{op}\,n_2$.

(v) If $\mathcal{F}[\vec{\mathsf{p}}] \Downarrow^F (\lambda x \,.\, \mathcal{M})[\vec{\mathsf{p}}']$ *and* $\mathcal{M}[\mathcal{A}/x][\vec{\mathsf{p}}\vec{\mathsf{p}}'] \Downarrow^F \mathcal{C}[\vec{\mathsf{p}}'']$*, then* $(\mathcal{F}\,\mathcal{A})[\vec{\mathsf{p}}] \Downarrow^F \mathcal{C}[\vec{\mathsf{p}}'']$.

(vi) If $\mathcal{M}[\mathsf{fix}\, x \,.\, \mathcal{M}/x][\vec{\mathsf{p}}] \Downarrow^F \mathcal{C}[\vec{\mathsf{p}}']$*, then* $(\mathsf{fix}\, x \,.\, \mathcal{M})[\vec{\mathsf{p}}] \Downarrow^F \mathcal{C}[\vec{\mathsf{p}}']$.

(vii) If $\mathcal{P}[\vec{\mathsf{p}}] \Downarrow^F \langle \mathcal{M}_1, \mathcal{M}_2 \rangle[\vec{\mathsf{p}}']$ *and* $\mathcal{M}_1[\vec{\mathsf{p}}'] \Downarrow^F \mathcal{C}[\vec{\mathsf{p}}'']$*, then* $\mathsf{fst}(\mathcal{P})[\vec{\mathsf{p}}] \Downarrow^F \mathcal{C}[\vec{\mathsf{p}}'']$.

(viii) If $\mathcal{P}[\vec{\mathsf{p}}] \Downarrow^F \langle \mathcal{M}_1, \mathcal{M}_2 \rangle[\vec{\mathsf{p}}']$ *and* $\mathcal{M}_2[\vec{\mathsf{p}}'] \Downarrow^F \mathcal{C}[\vec{\mathsf{p}}'']$*, then* $\mathsf{snd}(\mathcal{P})[\vec{\mathsf{p}}] \Downarrow^F \mathcal{C}[\vec{\mathsf{p}}'']$.

(ix) If $\mathcal{L}[\vec{\mathsf{p}}] \Downarrow^F \mathsf{nil}[\,]$ *and* $\mathcal{M}_1[\vec{\mathsf{p}}] \Downarrow^F \mathcal{C}[\vec{\mathsf{p}}']$*, then*

$$(\mathsf{case}\ \mathcal{L}\ \mathsf{of}\ \{\mathsf{nil} \to \mathcal{M}_1 \mid h :: t \to \mathcal{M}_2\})[\vec{\mathsf{p}}] \Downarrow^F \mathcal{C}[\vec{\mathsf{p}}'].$$

(x) If $\mathcal{L}[\vec{\mathsf{p}}] \Downarrow^F (\mathcal{H} :: \mathcal{T})[\vec{\mathsf{p}}']$ *and* $\mathcal{M}_2[\mathcal{H}/h, \mathcal{T}/t][\vec{\mathsf{p}}\vec{\mathsf{p}}'] \Downarrow^F \mathcal{C}[\vec{\mathsf{p}}'']$*, then*

$$(\mathsf{case}\ \mathcal{L}\ \mathsf{of}\ \{\mathsf{nil} \to \mathcal{M}_1 \mid h :: t \to \mathcal{M}_2\})[\vec{\mathsf{p}}] \Downarrow^F \mathcal{C}[\vec{\mathsf{p}}''].$$

Proof. Each property follows from combining the corresponding evaluation rule in Figure 3 with the definition of $\Downarrow^F$. We give the argument for the last case (x), and leave the others as exercises for the reader. So suppose

$$\mathcal{L}[\vec{\mathsf{p}}] \Downarrow^F (\mathcal{H} :: \mathcal{T})[\vec{\mathsf{p}}'] \tag{4.1}$$

$$\mathcal{M}_2[\mathcal{H}/h, \mathcal{T}/t][\vec{\mathsf{p}}\vec{\mathsf{p}}'] \Downarrow^F \mathcal{C}[\vec{\mathsf{p}}''] \tag{4.2}$$

In order to verify that $(\mathsf{case}\ \mathcal{L}\ \mathsf{of}\ \{\mathsf{nil} \to \mathcal{M}_1 \mid h :: t \to \mathcal{M}_2\})[\vec{\mathsf{p}}] \Downarrow^F \mathcal{C}[\vec{\mathsf{p}}'']$, we have to show for any $I \in \mathcal{P}_{\mathrm{cof}}(\mathbb{N}^{|\vec{\mathsf{p}}|})$ that

$$\{\vec{n}\vec{n}'' \mid \vec{n} \in I \ \&\ (\mathsf{case}\ \mathcal{L}\ \mathsf{of}\ \{\mathsf{nil} \to \mathcal{M}_1 \mid h :: t \to \mathcal{M}_2\})[F_{\vec{n}}] \Downarrow \mathcal{C}[F_{\vec{n}''}]\} \tag{4.3}$$

is a cofinal subset of $\mathbb{N}^{|\vec{\mathsf{p}}|+|\vec{\mathsf{p}}''|}$. But given such an I, by (4.1)

$$I' \stackrel{\mathrm{def}}{=} \{\vec{n}\vec{n}' \mid \vec{n} \in I \ \&\ \mathcal{L}[F_{\vec{n}}] \Downarrow (\mathcal{H} :: \mathcal{T})[F_{\vec{n}'}]\}$$

is a cofinal subset of $\mathbb{N}^{|\vec{\mathsf{p}}|+|\vec{\mathsf{p}}'|}$. Then by (4.2) applied to I'

$$I'' \stackrel{\mathrm{def}}{=} \{\vec{n}\vec{n}'\vec{n}'' \mid \vec{n}\vec{n}' \in I' \ \&\ \mathcal{M}_2[\mathcal{H}/h, \mathcal{T}/t][F_{\vec{n}\vec{n}'}] \Downarrow \mathcal{C}[F_{\vec{n}''}]\}$$

is a cofinal subset of $\mathbb{N}^{|\vec{\mathsf{p}}|+|\vec{\mathsf{p}}'|+|\vec{\mathsf{p}}''|}$ and hence

$$I''' \stackrel{\mathrm{def}}{=} \{\vec{n}\vec{n}'' \mid \exists \vec{n}' \,(\vec{n}\vec{n}'\vec{n}'' \in I'')\}$$

is a cofinal subset of $\mathbb{N}^{|\vec{\mathsf{p}}|+|\vec{\mathsf{p}}''|}$.

Now if $\vec{n}\vec{n}'' \in I'''$, then $\vec{n} \in I$ and for some $\vec{n}'$ it is the case that

$$\mathcal{L}[F_{\vec{n}}] \Downarrow (\mathcal{H} :: \mathcal{T})[F_{\vec{n}'}] = \mathcal{H}[F_{\vec{n}'}] :: \mathcal{T}[F_{\vec{n}'}]$$
$$\mathcal{M}_2[F_{\vec{n}}][\mathcal{H}[F_{\vec{n}'}]/h, \mathcal{T}[F_{\vec{n}'}]/t] = \mathcal{M}_2[\mathcal{H}/h, \mathcal{T}/t][F_{\vec{n}\vec{n}'}] \Downarrow \mathcal{C}[F_{\vec{n}''}]$$

and hence by ($\Downarrow$ case2)

$$(\mathsf{case}\ \mathcal{L}\ \mathsf{of}\ \{\mathsf{nil} \to \mathcal{M}_1 \mid h :: t \to \mathcal{M}_2\})[F_{\vec{n}}] \Downarrow \mathcal{C}[F_{\vec{n}''}].$$

Thus (4.3) contains I''' and hence is also a cofinal subset of $\mathbb{N}^{|\vec{\mathsf{p}}|+|\vec{\mathsf{p}}''|}$, as required. $\square$

Proposition 4.5. *For all PCFL contexts $\mathcal{M}[\vec{\mathsf{p}}]$, if $\mathcal{M}[F_\omega] \Downarrow C$ then there is a context $\mathcal{C}[\vec{\mathsf{p}}']$ with $C = \mathcal{C}[F_\omega]$ and $\mathcal{M}[\vec{\mathsf{p}}] \Downarrow^F \mathcal{C}[\vec{\mathsf{p}}']$.*

Proof. The proof is by induction on the derivation of $\mathcal{M}[F_\omega] \Downarrow C$. More precisely, we show that

$$\mathcal{E} \stackrel{\text{def}}{=} \{(M, C) \mid \forall \mathcal{M}[\vec{\mathsf{p}}]\,(M = \mathcal{M}[F_\omega] \Rightarrow \exists \mathcal{C}[\vec{\mathsf{p}}']\,(C = \mathcal{C}[F_\omega]\ \&\ \mathcal{M}[\vec{\mathsf{p}}] \Downarrow^F \mathcal{C}[\vec{\mathsf{p}}']))\}$$

is closed under the axioms and rules of Figure 3 inductively defining $\Downarrow$.

Case ($\Downarrow$ can). Since F_ω is not in canonical form, if C is a closed term in canonical form and $C = \mathcal{C}[F_\omega]$, then the context $\mathcal{C}$ must itself be in canonical form, and hence $\mathcal{C}[\vec{\mathsf{p}}] \Downarrow^F \mathcal{C}[\vec{\mathsf{p}}]$ by Lemma 4.4(i). Hence $(C, C) \in \mathcal{E}$ for any closed canonical C.

Case ($\Downarrow$ fix). Suppose that $(M[\mathsf{fix}\,x\,.\,M/x], C)$ is an element of $\mathcal{E}$; we wish to show that $(\mathsf{fix}\,x\,.\,M, C)$ is too. So suppose

$$\mathsf{fix}\,x\,.\,M = \mathcal{M}[F_\omega] \tag{4.4}$$

for some context $\mathcal{M}[\vec{\mathsf{p}}]$. We have to find $\mathcal{C}[\vec{\mathsf{p}}']$ such that $C = \mathcal{C}[F_\omega]$ and $\mathcal{M}[\vec{\mathsf{p}}] \Downarrow^F \mathcal{C}[\vec{\mathsf{p}}']$. For (4.4) to hold it must be the case that either

(A) $\mathcal{M} = \mathsf{p}_i$ is one of the parameters in $\vec{\mathsf{p}}$ and $\mathsf{fix}\,x\,.\,M = F_\omega (= \mathsf{fix}\,x\,.\,F)$—without loss of generality we may assume that $M = F$;

or

(B) $\mathcal{M}$ is of the form $\mathsf{fix}\,x\,.\,\mathcal{M}'[\vec{\mathsf{p}}]$ and $M = \mathcal{M}'[F_\omega]$.

We consider each case in turn.

(A). Let $\mathcal{N}[\mathsf{p}]$ be the context $F[\mathsf{p}/x]$. Thus $\mathcal{N}[F_\omega] = F[F_\omega/x] = M[\mathsf{fix}\,x\,.\,M/x]$. Then since by the induction hypothesis $(M[\mathsf{fix}\,x\,.\,M/x], C) \in \mathcal{E}$, there is some context $\mathcal{C}[\vec{\mathsf{p}}']$ with

$$C = \mathcal{C}[F_\omega] \tag{4.5}$$

$$\mathcal{N}[\mathsf{p}] \Downarrow^F \mathcal{C}[\vec{\mathsf{p}}'] \tag{4.6}$$

In view of (4.5), to complete this case it suffices to show that $\mathcal{M}[\vec{\mathsf{p}}] \Downarrow^F \mathcal{C}[\vec{\mathsf{p}}']$. Since $\mathcal{M} = \mathsf{p}_i$, by Lemma 4.3 this is equivalent to showing that $\mathsf{p}[\mathsf{p}] \Downarrow^F \mathcal{C}[\vec{\mathsf{p}}']$. So for each $I \in \mathcal{P}_{\mathrm{cof}}(\mathbb{N})$ we have to show that

$$\{n\vec{n}' \mid n \in I\ \&\ F_n \Downarrow \mathcal{C}[F_{\vec{n}'}]\} \tag{4.7}$$

is a cofinal subset of $\mathbb{N}^{1+|\vec{\mathsf{p}}'|}$. But if I is cofinal in $\mathbb{N}$, so is

$$J \stackrel{\text{def}}{=} \{n \mid n+1 \in I\}$$

and hence from (4.6)

$$J' \stackrel{\text{def}}{=} \{n\vec{n}' \mid n \in J \;\&\; \mathcal{N}[F_n] \Downarrow \mathcal{C}[F_{\vec{n}'}]\}$$

is a cofinal subset of $\mathbb{N}^{1+|\vec{\mathsf{p}}'|}$. But if $n\vec{n}' \in J'$ then $n+1 \in I$ and

$$\begin{aligned} \mathcal{N}[F_n] &= F[F_n/x] && \text{by definition of } \mathcal{N} \\ &= F_{n+1} && \text{by definition of } F_{n+1}. \end{aligned}$$

Thus if $n\vec{n}' \in J'$ then $(n+1)\vec{n}'$ is an element of (4.7). Since J' is cofinal, it follows that (4.7) is as well. This completes the induction step in case (A).

(B). By assumption, $M[\mathsf{fix}\, x\,.\,M/x] = \mathcal{M}'[\mathsf{fix}\, x\,.\,\mathcal{M}'/x][F_\omega]$. Then by the induction hypothesis $(M[\mathsf{fix}\, x\,.\,M/x], C) \in \mathcal{E}$, there is some context $\mathcal{C}[\vec{\mathsf{p}}']$ with

$$C = \mathcal{C}[F_\omega]$$
$$\mathcal{M}'[\mathsf{fix}\, x\,.\,\mathcal{M}'/x][\vec{\mathsf{p}}] \Downarrow \mathcal{C}[\vec{\mathsf{p}}']$$

and hence by Lemma 4.4(vi)

$$\mathcal{M}[\vec{\mathsf{p}}] = \mathsf{fix}\, x\,.\,\mathcal{M}'[\vec{\mathsf{p}}] \Downarrow \mathcal{C}[\vec{\mathsf{p}}'].$$

This completes the induction step in case (B).

Thus in either case we have $(\mathsf{fix}\, x\,.\,M, C) \in \mathcal{E}$, and so we have completed the induction step for case ($\Downarrow$ fix).

Case ($\Downarrow$ cond1). Suppose that (B, true) and (M_1, C) are both elements of $\mathcal{E}$. We have to show that $(\mathsf{if}\; B\; \mathsf{then}\; M_1\; \mathsf{else}\; M_2, C) \in \mathcal{E}$. So suppose

$$\mathsf{if}\; B\; \mathsf{then}\; M_1\; \mathsf{else}\; M_2 = \mathcal{M}[F_\omega] \tag{4.8}$$

for some context $\mathcal{M}[\vec{\mathsf{p}}]$. We have to find $\mathcal{C}[\vec{\mathsf{p}}']$ such that $C = \mathcal{C}[F_\omega]$ and $\mathcal{M}[\vec{\mathsf{p}}] \Downarrow^F \mathcal{C}[\vec{\mathsf{p}}']$.

For (4.8) to hold it must be the case that $\mathcal{M} = \mathsf{if}\; \mathcal{B}\; \mathsf{then}\; \mathcal{M}_1\; \mathsf{else}\; \mathcal{M}_2$ for some contexts $\mathcal{B}$, and $\mathcal{M}_i$ $(i = 1, 2)$ satisfying

$$B = \mathcal{B}[F_\omega] \tag{4.9}$$
$$M_i = \mathcal{M}_i[F_\omega] \quad (i = 1, 2) \tag{4.10}$$

Since $(B, \mathsf{true}) \in \mathcal{E}$, (4.9) implies that there is some context $\mathcal{C}_1[\vec{\mathsf{p}}_1]$ with

$$\mathsf{true} = \mathcal{C}_1[F_\omega] \tag{4.11}$$
$$\mathcal{B}[\vec{\mathsf{p}}] \Downarrow^F \mathcal{C}_1[\vec{\mathsf{p}}_1] \tag{4.12}$$

Now (4.11) can only hold because $\mathcal{C}_1 = \mathsf{true}$, in which case from (4.12) and Lemma 4.3 we conclude that

$$\mathcal{B}[\vec{\mathsf{p}}] \Downarrow^F \mathsf{true}[\,] \tag{4.13}$$

Since $(M_1, C) \in \mathcal{E}$, (4.10) implies that there is some context $\mathcal{C}[\vec{\mathsf{p}}']$ with

$$C = \mathcal{C}[F_\omega] \tag{4.14}$$

$$\mathcal{M}_1[\vec{\mathsf{p}}] \Downarrow^F \mathcal{C}[\vec{\mathsf{p}}']. \tag{4.15}$$

Applying Lemma 4.4(ii) to (4.13) and (4.15) yields

$$\mathcal{M}[\vec{\mathsf{p}}] = (\mathsf{if}\ \mathcal{B}\ \mathsf{then}\ \mathcal{M}_1\ \mathsf{else}\ \mathcal{M}_2)[\vec{\mathsf{p}}] \Downarrow^F \mathcal{C}[\vec{\mathsf{p}}']$$

which together with (4.14) is the desired conclusion.

Remaining cases: they are all similar to the previous case, using the appropriate clause of Lemma 4.4 in each case. □

Corollary 4.6 (A 'compactness' property of evaluation). *For any PCFL context $\mathcal{M}[\mathsf{p}]$ of ground type, if $\mathcal{M}[\mathsf{fix}\,x\,.\,F] \Downarrow \underline{c}$, then $\mathcal{M}[\mathsf{fix}^{(n)}x\,.\,F] \Downarrow \underline{c}$ for some $n \in \mathbb{N}$.*

Proof. Suppose $\mathcal{M}[F_\omega] \Downarrow \underline{c}$. Then by the previous proposition there is a context $\mathcal{C}[\vec{\mathsf{p}}']$ such that $\mathcal{M}[\mathsf{p}] \Downarrow^F \mathcal{C}[\vec{\mathsf{p}}']$ and $\mathcal{C}[F_\omega] = \underline{c}$. The latter equation can only hold because $\mathcal{C} = \underline{c}$, and hence (using Lemma 4.3) we have $\mathcal{M}[\mathsf{p}] \Downarrow^F \underline{c}[\,]$. Taking $I = \mathbb{N}$ in the definition of $\Downarrow^F$, this means that $\{n \mid \mathcal{M}[F_n] \Downarrow \underline{c}\}$ is a cofinal subset of $\mathbb{N}$. So in particular it is a non-empty subset, that is, there is some $n \in \mathbb{N}$ with $\mathcal{M}[F_n] \Downarrow \underline{c}$, as required. □

We can now complete the proof of the rational completeness (2.29) and syntactic continuity property (2.30) of the PCFL contextual preorder.

Theorem 4.7. *For any fixpoint term $\mathsf{fix}\,x\,.\,F \in \mathit{Exp}_\tau$, define the terms $\mathsf{fix}^{(n)}x\,.\,F \in \mathit{Exp}_\tau$ ($n \in \mathbb{N}$) by*

$$\mathsf{fix}^{(0)}x\,.\,F \stackrel{\text{def}}{=} \bot \stackrel{\text{def}}{=} \mathsf{fix}\,x\,.\,x$$

$$\mathsf{fix}^{(n+1)}x\,.\,F \stackrel{\text{def}}{=} F[\mathsf{fix}^{(n)}x\,.\,F/x].$$

Then for any type σ, context $\mathcal{C}[-_\tau] \in \mathit{Ctx}_\sigma$, and term $M \in \mathit{Exp}_\sigma$

$$\mathcal{C}[\mathsf{fix}\,x\,.\,F] \leq^{\text{gnd}} M : \sigma \Leftrightarrow \forall n \in \mathbb{N}\,(\mathcal{C}[\mathsf{fix}^{(n)}x\,.\,F] \leq^{\text{gnd}} M : \sigma).$$

Proof. We mentioned at the beginning of this section that property (2.27) of $\perp$ combined with the precongruence property of $\leq^{\text{gnd}}$ imply that $\mathsf{fix}^{(n)} x \,.\, F \;\leq^{\text{gnd}} \mathsf{fix}\, x \,.\, F \;:\; \tau$ and hence that $\mathcal{C}[\mathsf{fix}^{(n)} x \,.\, F] \;\leq^{\text{gnd}}\; \mathcal{C}[\mathsf{fix}\, x \,.\, F] \;:\; \sigma$. The $\Rightarrow$ direction of the theorem follows from this by transitivity of $\leq^{\text{gnd}}$.

Conversely, suppose $\mathcal{C}[\mathsf{fix}^{(n)} x \,.\, F] \leq^{\text{gnd}} M : \sigma$ holds for all $n \in \mathbb{N}$. We wish to show that $\mathcal{C}[\mathsf{fix}\, x \,.\, F] \leq^{\text{gnd}} M : \sigma$, that is, for any context $\mathcal{N}[-]$ of ground type and any constant $\underline{c}$, if $\mathcal{N}[\mathcal{C}[\mathsf{fix}\, x \,.\, F]] \Downarrow \underline{c}$ then $\mathcal{N}[\mathcal{C}[M]] \Downarrow \underline{c}$. But if $\mathcal{N}[\mathcal{C}[\mathsf{fix}\, x \,.\, F]] \Downarrow \underline{c}$, then by Corollary 4.6 applied to the context $\mathcal{N}[\mathcal{C}[-]]$, there is some $n \in \mathbb{N}$ with $\mathcal{N}[\mathcal{C}[\mathsf{fix}^{(n)} x \,.\, F]] \Downarrow \underline{c}$. Since $\mathcal{C}[\mathsf{fix}^{(n)} x \,.\, F] \leq^{\text{gnd}} M : \sigma$, it follows that $\mathcal{N}[M] \Downarrow \underline{c}$, as required. $\square$

Exercise 4.8. In view of the Operational Extensionality Theorem 3.8, we could have stated the above theorem using PCFL similarity, $\preceq$, rather than $\leq^{\text{gnd}}$. Give a direct proof that $\preceq$ satisfies the rational completeness and syntactic continuity properties by proving that $\mathcal{S} \in \mathit{Rel}$ is a PCFL simulation, where for each type σ we define

$$\begin{aligned}\mathcal{S}_\sigma \stackrel{\text{def}}{=} \{(M, M') \mid \exists \mathcal{M}[\vec{\mathsf{p}}] \in \mathit{Ctx}_\sigma, I \in \mathcal{P}_{\text{cof}}(\mathbb{N}^{|\vec{\mathsf{p}}|})\, (M = \mathcal{M}[F_\omega]\ \& \\ \forall \vec{n} \in I\, (\mathcal{M}[F_{\vec{n}}] \preceq_\sigma M'))\}.\end{aligned}$$

Exercise 4.9. Prove the following converse of Proposition 4.5, by induction on the derivation of $\mathcal{M}[F_{\vec{n}}] \Downarrow C$:

> *F*or all contexts $\mathcal{M}[\vec{\mathsf{p}}]$ and all $\vec{n} \in \mathbb{N}^{|\vec{\mathsf{p}}|}$, if $\mathcal{M}[F_{\vec{n}}] \Downarrow C$ then there is some canonical context $\mathcal{C}[\vec{\mathsf{p}}']$ and some $\vec{n}' \in \mathbb{N}^{|\vec{\mathsf{p}}'|}$ with $C = \mathcal{C}[F_{\vec{n}'}]$ and $\mathcal{M}[F_\omega] \Downarrow \mathcal{C}[F_\omega]$.

5 Further Directions

In this section we discuss, very briefly and without going into details, the extent to which the co-inductive characterisation of PCFL contextual equivalence in terms of bisimilarity is stable with respect to change of program equivalence, or of programming language.

'Lazy' contextual equivalence

As the name suggests, the definition of PCFL ground contextual equivalence (Definition 2.9) involves observing convergence of evaluation only in contexts of *ground* type. A strictly finer notion of contextual equivalence is obtained if we relax this condition and observe convergence at any type.

$$\Gamma \vdash M \cong^{\text{lazy}} M' : \sigma \stackrel{\text{def}}{\Leftrightarrow} \forall \tau, \mathcal{C}[-_\sigma] \in \mathit{Ctx}_\tau\, (\mathcal{C}[M]\Downarrow \Leftrightarrow \mathcal{C}[M']\Downarrow)$$

where by $M{\Downarrow}$ we mean $\exists C\,(M \Downarrow C)$. Clearly

$$\Gamma \vdash M \cong^{\text{lazy}} M' : \sigma \Rightarrow \Gamma \vdash M \cong^{\text{gnd}} M' : \sigma.$$

However, the converse does not hold. For example we have

$$\lambda x\,.\,\bot \cong^{\text{gnd}} \bot : \sigma \to \sigma' \tag{5.1}$$

by (2.21); but $\lambda x\,.\,\bot \not\cong^{\text{lazy}} \bot : \sigma \to \sigma'$, because the left-hand side does evaluate whereas the right-hand side does not. Similarly,

$$\langle \bot, \bot \rangle \cong^{\text{gnd}} \bot : \sigma \times \sigma' \tag{5.2}$$

holds by (2.21), whereas $\langle \bot, \bot \rangle \not\cong^{\text{lazy}} \bot : \sigma \times \sigma'$.

It is possible to modify the notion of bisimulation to get a co-inductive characterisation of $\cong^{\text{lazy}}$.

Theorem 5.1. *Let $\simeq'$ be the greatest element of Rel satisfying conditions* (bis 1a), (bis 1b), (bis 2a), (bis 2b), (bis 5a), (bis 5b), (bis 6a)*, and* (bis 6b) *from Figure 5 together with the following conditions at function and product types:*

$$\begin{aligned}
&(F\,\mathcal{B}_{\sigma\to\sigma'}\,F' \;\&\; F \Downarrow \lambda x\,.\,M) \Rightarrow && \text{(bis 3a)}\\
&\qquad \exists \lambda x\,.\,M'\,(F' \Downarrow \lambda x\,.\,M' \;\&\; \forall A \in \mathit{Exp}_\sigma\,(M[A/x]\,\mathcal{B}_{\sigma'}\,M'[A/x]))\\
&(F\,\mathcal{B}_{\sigma\to\sigma'}\,F' \;\&\; F' \Downarrow \lambda x\,.\,M') \Rightarrow && \text{(bis 3b)}\\
&\qquad \exists \lambda x\,.\,M\,(F \Downarrow \lambda x\,.\,M \;\&\; \forall A \in \mathit{Exp}_\sigma\,(M[A/x]\,\mathcal{B}_{\sigma'}\,M'[A/x]))\\
&(P\,\mathcal{B}_{\sigma\times\sigma'}\,P' \;\&\; P \Downarrow \langle M_1, M_2\rangle) \Rightarrow && \text{(bis 4a)}\\
&\qquad \exists M_1', M_2'\,(P' \Downarrow \langle M_1', M_2'\rangle \;\&\; M_1\,\mathcal{B}_\sigma\,M_1' \;\&\; M_2\,\mathcal{B}_{\sigma'}\,M_2')\\
&(P\,\mathcal{B}_{\sigma\times\sigma'}\,P' \;\&\; P' \Downarrow \langle M_1', M_2'\rangle) \Rightarrow && \text{(bis 4b)}\\
&\qquad \exists M_1, M_2\,(P \Downarrow \langle M_1, M_2\rangle \;\&\; M_1\,\mathcal{B}_\sigma\,M_1' \;\&\; M_2\,\mathcal{B}_{\sigma'}\,M_2').
\end{aligned}$$

Defining $\simeq'^{\circ}$ from $\simeq$ as in Definition 3.7, then

$$\Gamma \vdash M \cong^{\text{lazy}} M' : \sigma \Leftrightarrow \Gamma \vdash M \simeq'^{\circ} M'.$$

The relation $\simeq'$ is a version for PCFL of Abramsky's notion of **applicative bisimulation** which he developed in his work with Ong on the untyped, 'lazy' lambda calculus (Abramsky 1990; Abramsky and Ong 1993). The above theorem can be proved using the operationally-based methods developed by Howe (1989, Howe (1996) and which we employ in the Appendix. to prove the coincidence of $\cong^{\text{gnd}}$ and $\simeq$.

Convergence testing

The η-rule (2.24) and the surjective pairing property (2.25) say that every closed PCFL term of function or product type is ground contextually equivalent to a term in canonical form. This is the essential difference between $\cong^{\mathrm{gnd}}$ and $\cong^{\mathrm{lazy}}$, and we can remove it by augmenting PCFL syntax with term-formers for testing convergence to canonical form at such types. Consider the extension of PCFL whose terms are given by the grammar of Figure 1 extended as follows:

$$M ::= \cdots \mid \mathsf{ispr}(M) \mid \mathsf{isfn}(M).$$

The type assignment and evaluation rules for ispr and isfn are:

$$\frac{\Gamma \vdash P : \sigma \times \sigma'}{\Gamma \vdash \mathsf{ispr}(P) : \mathit{bool}} \qquad \frac{\Gamma \vdash F : \sigma \to \sigma'}{\Gamma \vdash \mathsf{isfn}(P) : \mathit{bool}}$$

$$\frac{P \Downarrow \langle M_1, M_2 \rangle}{\mathsf{ispr}(P) \Downarrow \mathsf{true}} \qquad \frac{F \Downarrow \lambda x \,.\, M}{\mathsf{isfn}(F) \Downarrow \mathsf{true}}.$$

Then Theorem 5.1 continues to hold, but now it is the case that $\cong^{\mathrm{lazy}}$ coincides with our original notion of ground contextual equivalence, $\cong^{\mathrm{gnd}}$.

Note. Analogous convergence testers for ground and list types already exist in PCFL, namely

$$\begin{aligned}
\mathit{isbool}[-_{\mathit{bool}}] &\stackrel{\mathrm{def}}{=} \mathsf{if}\ -_{\mathit{bool}}\ \mathsf{then\ true\ else\ true}\\
\mathit{isint}[-_{\mathit{int}}] &\stackrel{\mathrm{def}}{=} \mathsf{if}\ -_{\mathit{int}} = 0\ \mathsf{then\ true\ else\ true}\\
\mathit{islist}[-_{[\sigma]}] &\stackrel{\mathrm{def}}{=} \mathsf{case}\ -_{[\sigma]}\ \mathsf{of}\ \{\mathsf{nil} \to \mathsf{true} \mid h :: t \to \mathsf{true}\}.
\end{aligned}$$

An alternative way to alter PCFL to make $\cong^{\mathrm{lazy}}$ and $\cong^{\mathrm{gnd}}$ coincide (while still retaining the validity of Theorem 5.1) is to use the elimination forms for product and function types which correspond systematically to their introduction forms of pairing and function abstraction respectively. (See Martin-Löf 1984, Preface; and the 'do-it-yourself' type theory of Backhouse, Chisholm, Malcolm, and Saaman 1989.)

For product types the eliminator takes the form

$$\mathsf{split}\ P\ \mathsf{as}\ \langle x_1, x_2 \rangle\ \mathsf{in}\ E$$

with free occurrences of x_1 and x_2 in E bound in the elimination term. Its typing and evaluation rules are as follows.

$$\frac{\Gamma \vdash P : \sigma_1 \times \sigma_2 \qquad \Gamma, x_1 : \sigma_1, x_2 : \sigma_2 \vdash E : \sigma}{\Gamma \vdash \mathsf{split}\ P\ \mathsf{as}\ \langle x_1, x_2 \rangle\ \mathsf{in}\ E : \sigma}$$

$$\frac{P \Downarrow \langle M_1, M_2 \rangle \qquad E[M_1/x_1, M_2/x_2] \Downarrow C}{\mathsf{split}\ P\ \mathsf{as}\ \langle x_1, x_2 \rangle\ \mathsf{in}\ E \Downarrow C}$$

The projections fst and snd, and the convergence tester ispr are all definable from it:

$$\begin{aligned}
p : \sigma_1 \times \sigma_2 &\vdash \mathsf{fst}(p) \cong^{\mathrm{lazy}} \mathsf{split}\ p\ \mathsf{as}\ \langle x_1, x_2 \rangle\ \mathsf{in}\ x_1 : \sigma_1\\
p : \sigma_1 \times \sigma_2 &\vdash \mathsf{snd}(p) \cong^{\mathrm{lazy}} \mathsf{split}\ p\ \mathsf{as}\ \langle x_1, x_2 \rangle\ \mathsf{in}\ x_2 : \sigma_2\\
p : \sigma_1 \times \sigma_2 &\vdash \mathsf{ispr}(p) \cong^{\mathrm{lazy}} \mathsf{split}\ p\ \mathsf{as}\ \langle x_1, x_2 \rangle\ \mathsf{in\ true} : \mathit{bool}
\end{aligned}$$

For function types, the systematic eliminator involves some extra syntactic complications. It is probably for this reason that it is less well-known in functional programming than in Type Theory. It takes the form

$$\mathsf{funsplit}\ F\ \mathsf{as}\ \lambda x\,.\,\xi(x)\ \mathsf{in}\ E$$

where ξ belongs to a new syntactic category of **function variables**. Free occurrences of ξ in E are bound in the elimination term and x is a bound variable (really it is just a dummy variable to make the syntax more readable). The typing and evaluation rules for the function eliminator are as follows.

$$\frac{\Gamma \vdash F : \sigma_1 \to \sigma_2 \qquad \Gamma, \xi(\sigma_1) : \sigma_2 \vdash E : \sigma}{\Gamma \vdash \mathsf{funsplit}\ F\ \mathsf{as}\ \lambda x\,.\,\xi(x)\ \mathsf{in}\ E : \sigma}$$

$$\frac{F \Downarrow \lambda x\,.\,M \qquad E[(x)M/\xi] \Downarrow C}{\mathsf{funsplit}\ P\ \mathsf{as}\ \lambda x\,.\,\xi(x)\ \mathsf{in}\ E \Downarrow C}.$$

The typing rule makes use of extended typing assumptions to the left of $\vdash$ that involve assigning 'arities' to function variables. For example, in the rule above $\xi(\sigma_1) : \sigma_2$ is an assumption that ξ is a unary function variable which applies to terms of type σ_1 to yield terms of type σ_2. The evaluation rule makes use of an extended notion of substitution, namely that of substituting a 'meta-abstraction' $(x)M$ for a function variable ξ in a term E: we leave its definition to the imagination of the reader. Function application and the convergence tester for function types are definable using the function eliminator:

$$f : \sigma_1 \to \sigma_2, a : \sigma_1 \vdash f\ a \cong^{\mathrm{lazy}} \mathsf{funsplit}\ f\ \mathsf{as}\ \lambda x\,.\,\xi(x)\ \mathsf{in}\ \xi(a) : \sigma_2$$
$$f : \sigma_1 \to \sigma_2 \vdash \mathsf{isfn}(f) \cong^{\mathrm{lazy}} \mathsf{funsplit}\ f\ \mathsf{as}\ \lambda x\,.\,\xi(x)\ \mathsf{in}\ \mathsf{true} : \mathit{bool}$$

Note. The systematically derived eliminator for list types is the case expression which we built into the original syntax of PCFL. We have been discussing how to augment the syntax of PCFL to make convergence at compound types more observable. Going in the opposite direction, it is possible to remove observability of convergence at list types, without reducing expressive power, by replacing the case expression by $\mathsf{head}(L)$ and $\mathsf{tail}(L)$ expressions, together with a semi-decision test for emptiness $\mathsf{isnil}(L)$ (which is boolean-valued[2] and diverges unless $L \Downarrow \mathsf{nil}$). The properties of $\cong^{\mathrm{gnd}}$ are altered thereby—for example, an 'η-rule' for lists becomes valid:

$$L \cong^{\mathrm{gnd}} \mathsf{nil} : [\sigma] \quad \vee \quad \exists H, T\,(L \cong^{\mathrm{gnd}} H :: T : [\sigma]).$$

In order to retain the validity of the Operational Extensionality Theorem 3.8, one has to change the notion of bisimilarity by replacing conditions (bis 5a), (bis 5b), (bis 6a), and (bis 6b) by one analogous to that for product types:

$$L\ \mathcal{B}_{[\sigma]}\ L' \Rightarrow$$
$$\mathsf{isnil}(L)\ \mathcal{B}_{\mathit{bool}}\ \mathsf{isnil}(L')\ \&\ \mathsf{head}(L)\ \mathcal{B}_{\sigma}\ \mathsf{head}(L')\ \&\ \mathsf{tail}(L)\ \mathcal{B}_{[\sigma]}\ \mathsf{tail}(L').$$

[2]Really $\mathsf{isnil}(L)$ should be of unit type, but we did not include a unit type in PCFL.

Strict function application

The rule ($\Downarrow$ app) in Figure 3 describes **non-strict**, or **call-by-name** function application. The **strict**, or **call-by-value** rule is

$$\frac{F \Downarrow \lambda x \,.\, M \qquad A \Downarrow C \qquad M[C/x] \Downarrow D}{F\,A \Downarrow D}$$

If one alters the notion of evaluation by replacing ($\Downarrow$ app) by this rule, then of course the properties of $\cong^{\mathrm{gnd}}$ change. The Operational Extensionality Theorem can be retained provided one alters the notion of bisimilarity at function types appropriately, by using conditions (bis 3a) and (bis 3b) given above with the universal quantification which occurs in them restricted to range over closed terms in *canonical form*. The proof of this and other operationally-based properties for strict functional languages can be developed along the lines indicated in these notes by systematically restricting the use of substitution of terms to substitution of terms in canonical form.[3] In other words, one carries along the idea that variables in strict languages implicitly range over values (that is, canonical forms). See also (Egidi, Honsell, and della Rocca 1992).

Recursive types, polymorphic types, no types

We built only one kind of recursive data—lazy lists—into our example language PCFL, because it is sufficient to illustrate some of the complications which such a feature introduces. An important complication is that in going from a simply typed language like PCF to ones with more complex type systems, one may loose the ability to define a notion of interest (such as some extensional notion of program equivalence, for example) by induction on the structure of the types. The co-inductive techniques used to prove the Operational Extensionality Theorem 3.8 for PCFL were originally developed for untyped languages. We have seen how they adapt to one simple form of recursive data, and in fact they extend very easily to give similar results for languages with general forms of recursively defined type. See Gordon (1995a, 1996), for example. As Rees (1994) shows, they can also be used to give operational extensionality results for languages with polymorphic types.

Languages with state

One can extend the methods described in this article to lambda-calculus based imperative programming languages—such as Idealised Algol (Reynolds 1981), Scheme (Abelson and Susman 1985), or Standard ML (Milner, Tofte, and Harper 1990). For work based directly on contextual equivalence for a Scheme-like language, see (Mason and Talcott 1991; Mason and Talcott 1992). For work employing various notions of bisimilarity and operationally-based logical relations applied

[3] so one should replace fixpoint terms by recursive *function* terms in a strict version of PCFL.

to ML- and Algol-like languages, see (Pitts and Stark 1993; Ritter and Pitts 1995; Stark 1995; Pitts 1996; Crole and Gordon 1996).

Concerning the status of Operational Extensionality theorems for functional languages with state, the situation is as follows. With some simplifying assumptions, an evaluation relation for such languages can take the general form

$$s, M \Downarrow s', C$$

where s is a state, M an expression to be evaluated, C the canonical form resulting from evaluation, and s' the state which results from the evaluation. A state might give the current values (which may well be complicated objects, such as closures, in the case of Scheme or Standard ML) of some storage locations, for example. If the language is such that it can be given an operational semantics in which the shape (the number of locations, say) of the final state s' is always the same as that of the initial state s, then it seems that a co-inductive characterisation of contextual equivalence can usually be given. This is the case if the language has global variables, but no constructs for locally declared state. Less trivially, it is also the case for 'block-structured' languages, such as Algol. On the other hand, languages like Standard ML, which involve dynamically created references, certainly do not have this nice property that the 'state shape' does not grow under evaluation. Accordingly, for Standard ML there are various notions of bisimilarity known which are congruences for the language and (hence) are contained in contextual equivalence, but so far none is known which actually coincides with contextual equivalence.

Refined notions of bisimulation

One very important topic has not been treated in these notes—namely the development of various refinements of the notion of bisimulation (such as 'bisimulation-up-to-bisimilarity') which can make the job of establishing specific instances of applicative bisimilarity much easier. This topic is addressed in (Gordon 1995b, section 4), to which the reader is referred.

A Proof of the Operational Extensionality Theorem

This appendix is devoted to the proof of Theorem 3.8, which says that the PCFL ground contextual preorder (Definition 2.9) coincides with the open extension of PCFL similarity (Definition 3.3). The proof will be split into two parts:

(a) Proof that the open extension $\preceq^{\circ}$ of similarity (Definition 3.7) is a PCFL precongruence.

(b) Proof that the ground contextual preorder $\leq^{\mathrm{gnd}}$, when restricted to closed terms, is a PCFL simulation.

For part (a) we employ an adaptation of a method due to Howe (1989, Howe (1996). From part (a) we show that one can easily deduce that $\Gamma \vdash M \preceq^{\circ}_{\sigma} M'$ implies $\Gamma \vdash M \leq^{\mathrm{gnd}}_{\sigma} M'$, and part (b) gives the converse.

Congruence properties of similarity

Roughly speaking, a PCFL congruence is a binary relation between (open) terms which respects the usual laws of equational reasoning. Thus the relation should be an equivalence relation which is preserved by the operation of substituting a term for a parameter in a context. When dealing with typed terms (as we are) it is natural to restrict to relations which only relate terms of the same type. Since typing takes place in the presence of an assignment of types to free variables, we include some 'structural' properties (such as weakening and preservation under the operation of substituting terms for free variables) in the definition of congruence. Also, since we are interested in properties of the contextual preorder, we place the emphasis on the notion of 'precongruence'—which is a congruence minus the symmetry property. The following definition formulates the notion of PCFL precongruence solely with PCFL terms, rather than with PCFL contexts. The lemma which follows it gives the precise sense in which a precongruence respects the operation of substitution into contexts.

Suppose $\mathcal{R}$ is family of binary relations $\mathcal{R}_{\Gamma,\sigma} \subseteq \mathit{Exp}_\sigma(\Gamma) \times \mathit{Exp}_\sigma(\Gamma)$, indexed by variable typings Γ and types σ. As usual, we will write $\Gamma \vdash M \,\mathcal{R}_\sigma\, M'$ to indicate that a pair of terms (M, M') is in the relation $\mathcal{R}_{\Gamma,\sigma}$.

Definition A.1. $\mathcal{R}$ is a **PCFL precongruence relation** if it has the following properties.

$$(\Gamma \vdash M \,\mathcal{R}_\sigma\, M' \;\&\; \Gamma \subseteq \Gamma') \Rightarrow \Gamma' \vdash M \,\mathcal{R}_\sigma\, M' \tag{A.1}$$

$$(\Gamma \vdash M : \sigma \;\&\; \Gamma, x : \sigma \vdash N \,\mathcal{R}_{\sigma'}\, N') \Rightarrow \Gamma \vdash N[M/x] \,\mathcal{R}_{\sigma'}\, N'[M/x] \tag{A.2}$$

$$\Gamma \vdash M : \sigma \Rightarrow \Gamma \vdash M \,\mathcal{R}_\sigma\, M \tag{A.3}$$

$$(\Gamma \vdash M \,\mathcal{R}_\sigma\, M' \;\&\; \Gamma \vdash M' \,\mathcal{R}_\sigma\, M'') \Rightarrow \Gamma \vdash M \,\mathcal{R}_\sigma\, M'' \tag{A.4}$$

$$(\Gamma \vdash M \,\mathcal{R}_\sigma\, M' \;\&\; \Gamma, x : \sigma \vdash N : \sigma') \Rightarrow \Gamma \vdash N[M/x] \,\mathcal{R}_{\sigma'}\, N[M'/x] \tag{A.5}$$

$$\Gamma, x : \sigma \vdash M \,\mathcal{R}_{\sigma'}\, M' \Rightarrow \Gamma \vdash \lambda x \,.\, M \,\mathcal{R}_{\sigma \to \sigma'}\, \lambda x \,.\, M' \tag{A.6}$$

$$\Gamma, x : \sigma \vdash M \,\mathcal{R}_\sigma\, M' \Rightarrow \Gamma \vdash \mathsf{fix}\, x \,.\, M \,\mathcal{R}_\sigma\, \mathsf{fix}\, x \,.\, M' \tag{A.7}$$

$$\begin{aligned} &\Gamma \vdash L \,\mathcal{R}_\sigma\, L' \;\&\; \Gamma \vdash M \,\mathcal{R}_{\sigma'}\, M' \;\&\; \Gamma, h : \sigma, t : [\sigma] \vdash N \,\mathcal{R}_{\sigma'}\, N' \\ &\quad\Rightarrow \Gamma \vdash \; (\mathsf{case}\; L \;\mathsf{of}\; \{\mathsf{nil} \to M \mid h :: t \to N\}) \,\mathcal{R}_{\sigma'} \\ &\qquad\qquad (\mathsf{case}\; L' \;\mathsf{of}\; \{\mathsf{nil} \to M' \mid h :: t \to N'\}) \end{aligned} \tag{A.8}$$

$\mathcal{R}$ is a **PCFL congruence relation** if in addition it is symmetric:

$$\Gamma \vdash M \,\mathcal{R}_\sigma\, M' \Rightarrow \Gamma \vdash M' \,\mathcal{R}_\sigma\, M.$$

Modulo (A.4), property (A.5) is equivalent to saying that the non variable-binding syntax constructors of PCFL preserve the precongruence relation. For example, for application one has:

$$(\Gamma \vdash F \mathcal{R}_{\sigma \to \sigma'} F' \mathbin{\&} \Gamma \vdash A \mathcal{R}_{\sigma} A') \Rightarrow \Gamma \vdash (F\, A) \mathcal{R}_{\sigma'} (F'\, A').$$

Then (A.6)–(A.8) extend this preservation property to the variable-binding constructs of the language. As the following lemma shows, these properties are all special cases of preservation of the precongruence relation by the operation of substituting for a parameter in a context.

Lemma A.2. *Suppose that $\mathcal{R}$ is a PCFL precongruence relation. Suppose further that $\Gamma, \Gamma' \vdash M \mathcal{R}_{\sigma} M'$, that $\mathcal{C}[-_{\sigma}] \in Ctx_{\sigma'}(\Gamma)$, and that the variables in $dom(\Gamma')$ all occur as the bound variables of binders in $\mathcal{C}$ which contain the parameter $-_{\sigma}$ within their scope. (Cf. the statement of Lemma 2.8; in particular by that lemma, $\mathcal{C}[M]$ and $\mathcal{C}[M']$ are elements of $Exp_{\sigma'}(\Gamma)$.) Then $\Gamma \vdash \mathcal{C}[M] \mathcal{R}_{\sigma'} \mathcal{C}[M']$.*

Proof. The proof is by induction on the derivation of $\Gamma \vdash \mathcal{C}[-_{\sigma}] : \sigma'$, and is omitted. □

We aim to show:

Theorem A.3. *PCFL similarity, $\preceq^{\circ}$, is a precongruence, and hence (by Proposition 3.6(iii)) PCFL bisimilarity is a congruence.*

It is possible to prove Theorem A.3 by indirect means, making use of a domain-theoretic denotational semantics for PCFL. Abramsky (1990) takes such a route to prove the congruence property for his notion of applicative bisimulation for the untyped lambda calculus. The proof we give here is based directly upon the operational semantics of PCFL and is a minor adaptation of the method given by Howe (1989, Howe (1996). An adaptation is needed because Howe's proof concerns notions of bisimulation matching contextual equivalences in which convergence of function and product expressions to canonical form is observable. This is not the case for PCFL contextual equivalence as defined above. This is the reason why the η-rules (2.24) and (2.25) hold. It is pleasant that such properties hold, but the main reason for choosing to treat this variant of contextual equivalence here is to allow a direct comparison with Milner and Plotkin's classic work on PCF (Milner 1977; Plotkin 1977). Gordon (1995a) gets a result like Theorem 3.8 for PCFL contextual equivalence (indeed for a language with general forms of recursive type), but for a notion of bisimilarity (a very useful notion, as *loc. cit.* shows) based upon a certain labelled transition system.

Proofs of congruence for bisimilarities arising from labelled transition systems for reactive systems (such as for CCS (Milner 1989, 4.4), for example) suggest the following strategy for proving Theorem A.3. Let $\mathcal{S}$ denote the 'precongruence closure' of $\preceq^{\circ}$, that is, the smallest PCFL precongruence containing $\preceq^{\circ}$. To see that $\preceq^{\circ}$ is a precongruence, it would suffice to show that $\mathcal{S} = \preceq^{\circ}$; and by definition $\mathcal{S}$

$$\Gamma, x : \sigma \vdash x \preceq^* N : \sigma \quad (\text{if} \quad \Gamma, x : \sigma \vdash x \preceq^\circ_\sigma N) \qquad (\preceq^* \text{var})$$

$$\Gamma, x : \sigma \vdash \underline{b} \preceq^*_{bool} N \quad (\text{if} \quad \Gamma \vdash \underline{b} \preceq^\circ_{bool} N) \qquad (\preceq^* \text{bool})$$

$$\frac{\begin{array}{c}\Gamma \vdash B \preceq^*_{bool} B' \\ \Gamma \vdash M_1 \preceq^*_\sigma M_1' \\ \Gamma \vdash M_2 \preceq^*_\sigma M_2'\end{array}}{\Gamma \vdash \begin{pmatrix}\mathsf{if}\ B\ \mathsf{then} \\ M_1\ \mathsf{else}\ M_2\end{pmatrix} \preceq^*_\sigma N} \quad \begin{array}{l}(\text{if} \quad \Gamma \vdash \mathsf{if}\ B'\ \mathsf{then} \\ \qquad M_1'\ \mathsf{else}\ M_2' \preceq^\circ_\sigma N)\end{array} \qquad (\preceq^* \text{cond})$$

$$\Gamma \vdash \underline{n} \preceq^*_{int} N \quad (\text{if} \quad \Gamma \vdash \underline{n} \preceq^\circ_{int} N) \qquad (\preceq^* \text{int})$$

$$\frac{\begin{array}{c}\Gamma \vdash M_1 \preceq^*_{int} M_1' \\ \Gamma \vdash M_2 \preceq^*_{int} M_2'\end{array}}{\Gamma \vdash M_1\ \mathsf{op}\ M_2 \preceq^*_\gamma N} \quad (\text{if} \quad \Gamma \vdash M_1'\ \mathsf{op}\ M_2' \preceq^\circ_\gamma N) \qquad (\preceq^* \text{op})$$

$$\frac{\Gamma, x : \sigma \vdash M \preceq^*_{\sigma'} M'}{\Gamma \vdash \lambda x \,.\, M \preceq^*_{\sigma \to \sigma'} N} \quad (\text{if} \quad \Gamma \vdash \lambda x \,.\, M' \preceq^\circ_{\sigma \to \sigma'} N) \qquad (\preceq^* \text{abs})$$

$$\frac{\begin{array}{c}\Gamma \vdash F \preceq^*_{\sigma \to \sigma'} F' \\ \Gamma \vdash A \preceq^*_\sigma A'\end{array}}{\Gamma \vdash F\, A \preceq^*_{\sigma'} N} \quad (\text{if} \quad \Gamma \vdash F'\, A' \preceq^\circ_{\sigma'} N) \qquad (\preceq^* \text{app})$$

$$\frac{\Gamma, x : \sigma \vdash M \preceq^*_\sigma M'}{\Gamma \vdash \mathsf{fix}\, x \,.\, M \preceq^*_\sigma N} \quad (\text{if} \quad \Gamma \vdash \mathsf{fix}\, x \,.\, M' \preceq^\circ_\sigma N) \qquad (\preceq^* \text{fix})$$

Figure 6: Definition of $\preceq^*$, begun

contains $\preceq^\circ$, it would be enough to prove that $\mathcal{S} \subseteq \preceq^\circ$. Since $\mathcal{S}$ satisfies (A.2), to establish this inclusion, it is enough to prove that $\mathcal{S}$ restricted to closed terms is a PCFL simulation. Unfortunately, it is not clear how to prove this. Instead we follow Howe, and define an auxiliary relation, $\preceq^*$, which is not quite the precongruence closure (for one thing, it is not transitive), but which permits the proof-strategy we have outlined to go through.

Definition A.4. The relation

$$\Gamma \vdash M \preceq^*_\sigma N \qquad (M, N \in \mathit{Exp}_\sigma(\Gamma))$$

is inductively defined by the axioms and rules in Figures 6 and 7.

Lemma A.5. *(i) If* $\Gamma \vdash M \preceq^*_\sigma M'$ *and* $\Gamma \vdash M' \preceq^\circ_\sigma M''$*, then* $\Gamma \vdash M \preceq^*_\sigma M''$.

(ii) If $\Gamma \vdash M : \sigma$*, then* $\Gamma \vdash M \preceq^*_\sigma M$.

(iii) If $\Gamma \vdash M \preceq^\circ_\sigma M'$*, then* $\Gamma \vdash M \preceq^*_\sigma M'$.

$$\frac{\begin{array}{c}\Gamma \vdash M_1 \preceq^*_\sigma M'_1 \\ \Gamma \vdash M_2 \preceq^*_{\sigma'} M'_2\end{array}}{\Gamma \vdash \langle M_1, M_2\rangle \preceq^*_{\sigma\times\sigma'} N} \text{(if} \quad \Gamma \vdash \langle M'_1, M'_2\rangle \preceq^\circ_{\sigma\times\sigma'} N) \qquad (\preceq^* \text{pair})$$

$$\frac{\Gamma \vdash P \preceq^*_{\sigma\times\sigma'} P'}{\Gamma \vdash \mathsf{fst}(P) \preceq^*_\sigma N} \text{(if} \quad \Gamma \vdash \mathsf{fst}(P') \preceq^\circ_\sigma N) \qquad (\preceq^* \text{fst})$$

$$\frac{\Gamma \vdash P \preceq^*_{\sigma\times\sigma'} P'}{\Gamma \vdash \mathsf{snd}(P) \preceq^*_{\sigma'} N} \text{(if} \quad \Gamma \vdash \mathsf{snd}(P') \preceq^\circ_{\sigma'} N) \qquad (\preceq^* \text{snd})$$

$$\Gamma \vdash \mathsf{nil} \preceq^*_\sigma N \quad \text{(if} \quad \Gamma \vdash \mathsf{nil} \preceq^\circ_\sigma N) \qquad (\preceq^* \text{nil})$$

$$\frac{\begin{array}{c}\Gamma \vdash H \preceq^*_\sigma H' \\ \Gamma \vdash T \preceq^*_{[\sigma]} T'\end{array}}{\Gamma \vdash H :: T \preceq^*_{[\sigma]} N} \text{(if} \quad \Gamma \vdash H' :: T' \preceq^\circ_{[\sigma]} N) \qquad (\preceq^* \text{cons})$$

$$\frac{\begin{array}{c}\Gamma \vdash L \preceq^*_{[\sigma]} L' \\ \Gamma \vdash M_1 \preceq^*_{\sigma'} M'_1 \\ \Gamma, h : \sigma, t : [\sigma] \vdash M_2 \preceq^*_{\sigma'} M'_2\end{array}}{\Gamma \vdash \left(\begin{array}{l}\mathsf{case}\ L\ \mathsf{of}\ \{\mathsf{nil}{\to}M_1 \\ \qquad\qquad \mid h :: t{\to}M_2\}\end{array}\right) \preceq^*_{\sigma'} N} \begin{array}{l}(\text{if}\ \Gamma \vdash \mathsf{case}\ L'\ \mathsf{of} \\ \{\mathsf{nil}{\to}M'_1 \mid h :: t{\to}M'_2\} \\ \preceq^\circ_{\sigma'} N)\end{array} \qquad (\preceq^* \text{case})$$

Figure 7: Definition of $\preceq^*$, completed

(iv) If $\Gamma \vdash M \preceq^*_\sigma M'$ *and* $\Gamma, x : \sigma \vdash N \preceq^*_{\sigma'} N'$, *then* $\Gamma \vdash N[M/x] \preceq^*_{\sigma'} N'[M'/x]$.

Proof. Part (i) is proved by induction on the derivation of $\Gamma \vdash M \preceq^*_\sigma M'$ from the axioms and rules in Figures 6 and 7. Part (ii) is proved by induction on the derivation of $\Gamma \vdash M : \sigma$ from the axioms and rules in Figure 2. Part (iii) follows from the first two parts. Part (iv) is proved by induction on the derivation of $\Gamma, x : \sigma \vdash N \preceq^*_{\sigma'} N'$, using part (iii) and the fact (evident from the definition of $\preceq^\circ$ from $\preceq$) that $\preceq^\circ$ satisfies property (A.2). □

Lemma A.6. *(i) If* $\emptyset \vdash \underline{b} \preceq^*_{bool} B$, *then* $B \Downarrow \underline{b}$.

(ii) If $\emptyset \vdash \underline{n} \preceq^*_{int} N$, *then* $N \Downarrow \underline{n}$.

(iii) If $\emptyset \vdash F \preceq^*_{\sigma\to\sigma'} F'$, *then for all* $A \in Exp_\sigma$, $\emptyset \vdash F\ A \preceq^*_{\sigma'} F'\ A$.

(iv) If $\emptyset \vdash P \preceq^*_{\sigma\times\sigma'} P'$, *then* $\emptyset \vdash \mathsf{fst}(P) \preceq^*_\sigma \mathsf{fst}(P')$ *and* $\emptyset \vdash \mathsf{snd}(P) \preceq^*_{\sigma'} \mathsf{snd}(P')$.

(v) If $\emptyset \vdash \mathsf{nil} \preceq^*_{[\sigma]} L$, *then* $L \Downarrow \mathsf{nil}$.

(vi) If $\emptyset \vdash H :: T \preceq^*_{[\sigma]} L$, *then* $L \Downarrow H' :: T'$ *for some* H', T' *with* $\emptyset \vdash H \preceq^*_\sigma H'$ *and* $\emptyset \vdash T \preceq^*_{[\sigma]} T'$.

Proof. For part (i), if $\emptyset \vdash \underline{b} \preceq^*_{bool} B$ holds, it must have been deduced using ($\preceq^*$ bool), so $\emptyset \vdash \underline{b} \preceq^\circ_{bool} B$ holds, that is, $\underline{b} \preceq_{bool} B$. Then since $\preceq$ is a simulation and $\underline{b} \Downarrow \underline{b}$, it follows from condition (sim 1) in Figure 4 that $B \Downarrow \underline{b}$, as required. The argument for parts (ii) and (v) is similar.

Part (iii) follows by applying rule ($\preceq^*$ app) from Figure 6 with $A' = A$ and $N = F\,A$, using the reflexivity of $\preceq^*$ (Lemma A.5(ii)) and $\preceq^\circ$ (via Proposition 3.6(i)). Similarly, part (iv) follows by applying the rules ($\preceq^*$ fst) and ($\preceq^*$ snd).

Finally for part (vi), if $\emptyset \vdash H :: T \preceq^*_{[\sigma]} L$ holds it can only have been deduced by an application of rule ($\preceq^*$ cons). So there are terms H'' and T'' with

$$\emptyset \vdash H \preceq^*_\sigma H'' \tag{A.9}$$
$$\emptyset \vdash T \preceq^*_{[\sigma]} T'' \tag{A.10}$$
$$\emptyset \vdash H'' :: T'' \preceq^\circ_{[\sigma]} L$$

and hence

$$H'' :: T'' \preceq_{[\sigma]} L. \tag{A.11}$$

The simulation property (sim 6a) of $\preceq$ applied to (A.11) and $H'' :: T'' \Downarrow H'' :: T''$ implies that there are further terms H' and T' with $L \Downarrow H' :: T'$, $H'' \preceq_\sigma H'$, and $T'' \preceq_{[\sigma]} T'$. The last two properties combined with (A.9), (A.10), and Lemma A.5(i) yield $\emptyset \vdash H \preceq^*_\sigma H'$ and $\emptyset \vdash T \preceq^*_{[\sigma]} T'$, as required. □

The following lemma gives the key property of $\preceq^*$ permitting the proof of Theorem A.3 to go through. It is the analogue of (Howe 1989, Theorem 1). In the proof of the lemma we will make use of the Kleene preorder, $\leq^{\mathrm{kl}}$, defined in Proposition 3.9 together with the fact, established in that proposition, that $\leq^{\mathrm{kl}}$ is contained in PCFL similarity.

Lemma A.7. *If $M \Downarrow C$ and $\emptyset \vdash M \preceq^*_\sigma N$, then $\emptyset \vdash C \preceq^*_\sigma N$.*

Proof. The proof is by induction on the derivation of $M \Downarrow C$. For once we will give the details of the induction proof in some detail, since it is quite delicate. To be more precise, we will show that

$$\mathcal{E} \stackrel{\text{def}}{=} \{(M, C) \mid \forall \sigma, N\, (\emptyset \vdash M \preceq^*_\sigma N \Rightarrow \emptyset \vdash C \preceq^*_\sigma N)\}$$

is closed under the axioms and rules in Figure 3 and hence contains the evaluation relation, as required.

Case ($\Downarrow$ can). Trivial.

Case ($\Downarrow$ cond1). Suppose that (B, true) and (M_1, C) are in $\mathcal{E}$. We have to show that $(\mathsf{if}\ B\ \mathsf{then}\ M_1\ \mathsf{else}\ M_2, C) \in \mathcal{E}$. So suppose

$$\emptyset \vdash \mathsf{if}\ B\ \mathsf{then}\ M_1\ \mathsf{else}\ M_2 \preceq^*_\sigma N \tag{A.12}$$

This can only have been deduced by an application of rule ($\preceq^*$ cond), so there are terms B', M'_1, and M'_2 with

$$\emptyset \vdash B \preceq^*_{bool} B', \quad \emptyset \vdash M_i \preceq^*_\sigma M'_i \quad (\text{for } i = 1, 2) \tag{A.13}$$

and

$$\mathsf{if}\ B'\ \mathsf{then}\ M'_1\ \mathsf{else}\ M'_2 \preceq_\sigma N \tag{A.14}$$

Since $(B, \mathsf{true}), (M_1, C) \in \mathcal{E}$, from (A.13) we get

$$\emptyset \vdash \mathsf{true} \preceq^*_{bool} B' \tag{A.15}$$

$$\emptyset \vdash C \preceq^*_\sigma M'_1 \tag{A.16}$$

By Lemma A.6(i), from (A.15) we have $B' \Downarrow \mathsf{true}$. Hence by definition of $\leq^{\mathrm{kl}}$,

$$M'_1 \leq^{\mathrm{kl}}_\sigma \mathsf{if}\ B'\ \mathsf{then}\ M'_1\ \mathsf{else}\ M'_2$$

holds. Therefore by Proposition 3.9 we have

$$M'_1 \preceq_\sigma \mathsf{if}\ B'\ \mathsf{then}\ M'_1\ \mathsf{else}\ M'_2$$

which combined with (A.14) and transitivity of $\preceq_\sigma$ yields $M'_1 \preceq_\sigma N$. Lemma A.5(i) plus (A.16) implies $\emptyset \vdash C \preceq^*_\sigma N$. So we have shown that (A.12) implies $\emptyset \vdash C \preceq^*_\sigma N$, for any N and σ. Thus $(\mathsf{if}\ B\ \mathsf{then}\ M_1\ \mathsf{else}\ M_2, C) \in \mathcal{E}$, as required.

Case ($\Downarrow$ cond2) is similar to the previous case.

Case ($\Downarrow$ op). Suppose $(M_i, \underline{n_i}) \in \mathcal{E}$ for $i = 1, 2$, and that

$$\emptyset \vdash M_1\ \mathsf{op}\ M_2 \preceq^*_\gamma N \tag{A.17}$$

We must show that

$$\emptyset \vdash \underline{c} \preceq^*_\gamma N \tag{A.18}$$

where $c \stackrel{\mathrm{def}}{=} n_1\ \mathsf{op}\ n_2$. Now (A.17) must have been deduced by an application of rule ($\preceq^*$ op) to

$$\emptyset \vdash M_i \preceq^*_{int} M'_i \quad (i = 1, 2) \tag{A.19}$$

$$M'_1\ \mathsf{op}\ M'_2 \preceq_\gamma N \tag{A.20}$$

for some terms M'_1, M'_2. Since $(M_i, \underline{n_i}) \in \mathcal{E}$, from (A.19) it follows that $\emptyset \vdash \underline{n_i} \preceq^*_{int} M'_i$ and hence by Lemma A.6(ii) that $M'_i \Downarrow \underline{n_i}$. Thus by rule ($\Downarrow$ op), $M'_1\ \mathsf{op}\ M_2 \Downarrow \underline{c}$ and therefore $\underline{c} \leq^{\mathrm{kl}}_\gamma M'_1\ \mathsf{op}\ M_2$ (by definition of $\leq^{\mathrm{kl}}$). Then from Proposition 3.9, (A.20), and transitivity of $\preceq$, we have that $\underline{c} \preceq_\gamma N$ and hence (by Lemma A.5(iii)) that (A.18) does indeed hold.

Case ($\Downarrow$ app). Suppose $(F, \lambda x\,.\,M), (M[A/x], C) \in \mathcal{E}$ and that

$$\emptyset \vdash F\,A \preceq^*_{\sigma'} N \tag{A.21}$$

We must show that

$$\emptyset \vdash C \preceq^*_{\sigma'} N \tag{A.22}$$

Now (A.21) must have been deduced by an application of ($\preceq^*$ app) to

$$\emptyset \vdash F \preceq^*_{\sigma\to\sigma'} F' \tag{A.23}$$

$$\emptyset \vdash A \preceq^*_{\sigma} A' \tag{A.24}$$

and

$$F'\,A' \preceq_{\sigma'} N \tag{A.25}$$

for some terms F', A'. Since $(F, \lambda x\,.\,M) \in \mathcal{E}, \emptyset \vdash \lambda x\,.\,M \preceq^*_{\sigma'} F'$ holds by (A.23). This can only have been derived by an application of ($\preceq^*$ abs) to

$$x : \sigma \vdash M \preceq^*_{\sigma'} M' \tag{A.26}$$

and

$$\lambda x\,.\,M' \preceq_{\sigma\to\sigma'} F' \tag{A.27}$$

for some term M'. Applying Lemma A.5(iv) to (A.24) and (A.26), we have that $\emptyset \vdash M[A/x] \preceq^*_{\sigma'} M'[A'/x]$. Then since $(M[A/x], C) \in \mathcal{E}$, it follows from this that

$$\emptyset \vdash C \preceq^*_{\sigma'} M'[A'/x] \tag{A.28}$$

Since $\preceq$ is a PCFL simulation, from property (sim 3) in Figure 4 applied to (A.27), we get

$$(\lambda x\,.\,M')A' \preceq_{\sigma'} F'\,A' \tag{A.29}$$

Note that by definition of $\leq^{\mathrm{kl}}$, we always have $M'[A'/x] \leq^{\mathrm{kl}}_{\sigma'} (\lambda x\,.\,M')A'$ and hence by Proposition 3.9, $\emptyset \vdash M'[A'/x] \preceq_{\sigma'} (\lambda x\,.\,M')A'$. Combining this with (A.25), (A.29) and transitivity of $\preceq$, we get $M'[A,/x] \preceq_{\sigma'} N$. Lemma A.5(i) applied to this and (A.28) yields (A.22), as required.

Case ($\Downarrow$ fix). Suppose $(M[\mathsf{fix}\, x \,.\, M/x], C) \in \mathcal{E}$ and that

$$\emptyset \vdash \mathsf{fix}\, x \,.\, M \preceq^*_\sigma N \tag{A.30}$$

We must show that

$$\emptyset \vdash C \preceq^*_\sigma N \tag{A.31}$$

Now (A.30) must have been deduced by an application of ($\preceq^*$ fix) to

$$x : \sigma \vdash M \preceq^*_\sigma M' \tag{A.32}$$

and

$$\mathsf{fix}\, x \,.\, M' \preceq_\sigma N \tag{A.33}$$

for some term M'. Applying ($\preceq^*$ fix) to (A.32) and $\mathsf{fix}\, x \,.\, M' \preceq_\sigma \mathsf{fix}\, x \,.\, M'$ (using the fact that $\preceq$ is reflexive), we get $\emptyset \vdash \mathsf{fix}\, x \,.\, M \preceq^*_\sigma \mathsf{fix}\, x \,.\, M'$. Applying Lemma A.5(iv) to this and (A.32) yields $\emptyset \vdash M[\mathsf{fix}\, x \,.\, M/x] \preceq^*_\sigma M'[\mathsf{fix}\, x \,.\, M'/x]$. Then since $(M[\mathsf{fix}\, x \,.\, M/x], C) \in \mathcal{E}$, we deduce that

$$\emptyset \vdash C \preceq^*_\sigma M'[\mathsf{fix}\, x \,.\, M'/x] \tag{A.34}$$

Note that by definition of $\leq^{\mathrm{kl}}$, one always has $M'[\mathsf{fix}\, x \,.\, M'/x] \leq^{\mathrm{kl}}_\sigma \mathsf{fix}\, x \,.\, M'$, and hence also $M'[\mathsf{fix}\, x \,.\, M'/x] \preceq_\sigma \mathsf{fix}\, x \,.\, M'$ (by Proposition 3.9). Combining this with (A.33), we get $M'[\mathsf{fix}\, x \,.\, M'/x] \preceq_\sigma N$. Applying Lemma A.5(i) to this and (A.34) yields (A.31), as required.

Case ($\Downarrow$ fst). Suppose $(P, \langle M_1, M_2\rangle), (M_1, C) \in \mathcal{E}$ and that

$$\emptyset \vdash \mathsf{fst}(P) \preceq^*_{\sigma_1} N \tag{A.35}$$

We must show that

$$\emptyset \vdash C \preceq^*_{\sigma_1} N \tag{A.36}$$

Now (A.35) must have been deduced by an application of ($\preceq^*$ fst) to

$$\emptyset \vdash P \preceq^*_{\sigma_1 \times \sigma_2} P' \tag{A.37}$$

and

$$\mathsf{fst}(P') \preceq_{\sigma_1} N \tag{A.38}$$

for some term P'. Since $(P, \langle M_1, M_2\rangle) \in \mathcal{E}$, from (A.37) we get

$$\emptyset \vdash \langle M_1, M_2\rangle \preceq^*_{\sigma_1 \times \sigma_2} P'.$$

This must have been deduced by an application of ($\preceq^*$ pair) to

$$\emptyset \vdash M_i \preceq^*_{\sigma_i} M'_i \quad (i = 1, 2) \tag{A.39}$$

$$\langle M'_1, M'_2 \rangle \preceq_{\sigma_1 \times \sigma_2} P' \tag{A.40}$$

for some terms M'_1, M'_2. Since $(M_1, C) \in \mathcal{E}$, from (A.39) we get

$$\emptyset \vdash C \preceq^*_{\sigma_1} M'_1 \tag{A.41}$$

Since $\preceq$ is a PCFL simulation, it satisfies property (sim 4) in Figure 4, and so (A.40) implies

$$\mathsf{fst}(\langle M'_1, M'_2 \rangle) \preceq_{\sigma_1} \mathsf{fst}(P') \tag{A.42}$$

Note that by definition of $\leq^{\mathrm{kl}}$, one always has $M'_1 \leq^{\mathrm{kl}}_{\sigma_1} \mathsf{fst}(\langle M'_1, M'_2 \rangle)$, and hence and hence also $M'_1 \preceq_{\sigma_1} \mathsf{fst}(\langle M'_1, M'_2 \rangle)$ (by Proposition 3.9). Combining this with (A.38) and (A.42) we get $M'_1 \preceq_{\sigma_1} N$. Applying Lemma A.5(i) to this and (A.41) yields (A.36), as required.

Case ($\Downarrow$ snd) is similar to the previous case.

Case ($\Downarrow$ case1) is similar to the case for ($\Downarrow$ cond1), but using the fact (evident from the definition of $\leq^{\mathrm{kl}}$) that if $L' \Downarrow \mathsf{nil}$, then $M'_1 \leq^{\mathrm{kl}}_{\sigma'} \mathsf{case}\ L'\ \mathsf{of}\ \{\mathsf{nil} \to M'_1 \mid h :: t \to M'_2\}$.

Case ($\Downarrow$ case2) is similar to the case for ($\Downarrow$ app), but using the fact (evident from the definition of $\leq^{\mathrm{kl}}$) that if $L' \Downarrow H' :: T'$, then

$$M'_2[H'/h, T'/t] \leq^{\mathrm{kl}}_{\sigma'} \mathsf{case}\ L'\ \mathsf{of}\ \{\mathsf{nil} \to M_1 \mid h :: t \to M_2\}.$$

This completes the proof of Lemma A.7. □

Proposition A.8. *For all* Γ, σ, M, N

$$\Gamma \vdash M \preceq^\circ_\sigma N \Leftrightarrow \Gamma \vdash M \preceq^*_\sigma N.$$

Proof. We have already proved the left to right implication in part (iii) of Lemma A.5. For the converse, note that by part (iv) of that lemma, and by the construction of $\preceq^\circ$ from $\preceq$ (Definition 3.7), it suffices to prove the implication just for closed terms:

$$\emptyset \vdash M \preceq^*_\sigma N \Rightarrow M \preceq_\sigma N.$$

By the co-induction principle for $\preceq$ (Proposition 3.5), it suffices to show that $\mathcal{S}$ is a PCFL simulation, where

$$\mathcal{S}_\sigma \stackrel{\text{def}}{=} \{(M, N) \mid \emptyset \vdash M \preceq^*_\sigma N\}.$$

But the fact that $\mathcal{S} \leq \langle \mathcal{S} \rangle$ follows immediately by combining Lemmas A.6 and A.7. □

We can now complete the proof of Theorem A.3.

Proof of Theorem A.3. We have seen that $\preceq$, and hence also $\preceq^\circ$, is reflexive and transitive. So it just remains to see that $\preceq^\circ$ has properties (A.1), (A.2), and (A.5)–(A.8) of Definition A.1. The weakening property (A.1) is an immediate consequence of the construction of $\preceq^\circ$ from $\preceq$. For the other properties, it suffices by Proposition A.8 to check that they hold for $\preceq^*$. The substitution properties (A.2) and (A.5) are both instances of Lemma A.5(iv) (using reflexivity of $\preceq^*$, established in part (ii) of that lemma). Finally, (A.6)–(A.8) hold for $\preceq^*$ by construction. For example, if $\Gamma, x : \sigma \vdash M \preceq^*_{\sigma'} M'$, then by ($\preceq^*$ abs) (taking $N = \lambda x \,.\, M'$ and using the fact that $\preceq^\circ$ is reflexive) one has $\Gamma \vdash \lambda x \,.\, M \preceq^*_{\sigma \to \sigma'} \lambda x \,.\, M'$. □

Corollary A.9. *For all* Γ, σ, M, N

$$\Gamma \vdash M \preceq^\circ_\sigma N \Rightarrow \Gamma \vdash M \leq^{\mathrm{gnd}}_\sigma N.$$

Proof. Suppose $\Gamma \vdash M \preceq^\circ_\sigma N$ and that $\mathcal{C}[-_\sigma]$ is a context for which $\mathcal{C}[M]$ and $\mathcal{C}[N]$ are closed terms of ground type, γ say. Since by Theorem A.3 $\preceq^\circ$ is a precongruence relation, it follows from Lemma A.2 that $\emptyset \vdash \mathcal{C}[M] \preceq^\circ_\gamma \mathcal{C}[N]$, that is, $\mathcal{C}[M] \preceq_\gamma \mathcal{C}[N]$. So if $\mathcal{C}[M] \Downarrow \underline{c}$, then by the simulation properties (sim 1) and (sim 2) of $\preceq$ it is also the case that $\mathcal{C}[N] \Downarrow \underline{c}$. Since this holds for any $\mathcal{C}[-_\sigma]$, we have that $\Gamma \vdash M \leq^{\mathrm{gnd}}_\sigma N$. □

The PCFL contextual preorder is a simulation

Referring back to the beginning of this section, we have now completed part (a) of the proof of Theorem 3.8, and it remains to prove part (b)—the fact that $\leq^{\mathrm{gnd}}$ is a PCFL simulation. The reader will be relieved to know that this part is quite straightforward in comparison with part (a).

Define $\mathcal{S} \in \mathit{Rel}$ by:

$$\mathcal{S}_\sigma \stackrel{\text{def}}{=} \{(M, N) \mid \emptyset \vdash M \leq^{\mathrm{gnd}}_\sigma N\}.$$

We wish to show that $\mathcal{S} \leq \langle \mathcal{S} \rangle$. We check each of the simulation properties in Figure 4 in turn.

Property (sim 1). Suppose $M \,\mathcal{S}_{\mathit{bool}}\, N$ and that $M \Downarrow \underline{b}$. Applying the definition of $\leq^{\mathrm{gnd}}$ with the context $\mathcal{C}[-_{\mathit{bool}}] \stackrel{\text{def}}{=} -_{\mathit{bool}}$ shows that $N \Downarrow \underline{b}$.

Property (sim 2). is just like the previous case.

Property (sim 3). Suppose $F\ \mathcal{S}_{\sigma\to\sigma'}\ F'$ and that $A \in \mathit{Exp}_\sigma$. Given any context $\mathcal{C}[-_{\sigma'}]$ for which $\mathcal{C}[F\ A]$ and $\mathcal{C}[F'\ A]$ are closed terms of ground type, let $\mathcal{C}'[-_{\sigma\to\sigma'}]$ be the context obtained by substituting $(-_{\sigma\to\sigma'})\ A$ for $-_{\sigma'}$ throughout $\mathcal{C}$. Thus $\mathcal{C}'[F] = \mathcal{C}[F\ A]$ and similarly with F' for F. Then

$$\begin{aligned}
\mathcal{C}[F\ A] \Downarrow \underline{c} &\Rightarrow \mathcal{C}'[F] \Downarrow \underline{c} && \text{since } \mathcal{C}'[F] = \mathcal{C}[F\ A] \\
&\Rightarrow \mathcal{C}'[F'] \Downarrow \underline{c} && \text{since } \emptyset \vdash F \leq^{\mathrm{gnd}}_{\sigma\to\sigma'} F' \\
&\Rightarrow \mathcal{C}[F'\ A] \Downarrow \underline{c} && \text{since } \mathcal{C}'[F'] = \mathcal{C}[F'\ A].
\end{aligned}$$

Thus $\emptyset \vdash F\ A \leq^{\mathrm{gnd}}_{\sigma'} F'\ A$, that is, $F\ A\ \mathcal{S}_{\sigma'}\ F'\ A$, as required.

Property (sim 4). The proof is like the previous case, but using $\mathcal{C}'[-_{\sigma_1\times\sigma_2}] \stackrel{\mathrm{def}}{=} \mathcal{C}[\mathsf{fst}(-_{\sigma_1\times\sigma_2})]$, and then $\mathcal{C}'[-_{\sigma_1\times\sigma_2}] \stackrel{\mathrm{def}}{=} \mathcal{C}[\mathsf{snd}(-_{\sigma_1\times\sigma_2})]$.

Property (sim 5). Suppose $L\ \mathcal{S}_{[\sigma]}\ L'$ and that $L \Downarrow \mathsf{nil}$. Consider the context

$$\mathcal{C}[-_{[\sigma]}] \stackrel{\mathrm{def}}{=} \mathsf{case}\ -_{[\sigma]}\ \mathsf{of}\ \{\mathsf{nil} \to \mathsf{true} \mid h :: t \to \mathsf{false}\}.$$

Note that $\mathcal{C}[L] \Downarrow \mathsf{true}$ if and only if $L \Downarrow \mathsf{nil}$, and similarly for L'. So if $L \Downarrow \mathsf{nil}$, since $\emptyset \vdash L \leq^{\mathrm{gnd}}_{[\sigma]} L'$, it follows that $\mathcal{C}[L'] \Downarrow \mathsf{true}$ and hence $L' \Downarrow \mathsf{nil}$.

Property (sim 6). Suppose $L\ \mathcal{S}_{[\sigma]}\ L'$ and that $L \Downarrow H :: T$. Arguing just as in the previous case, we have that $L \Downarrow H' :: T'$ for some terms H', T'. We have to show that $\emptyset \vdash H \leq^{\mathrm{gnd}}_{\sigma} H'$ and $\emptyset \vdash T \leq^{\mathrm{gnd}}_{[\sigma]} T'$.

We make use of PCFL expressions for the functions for taking the head and tail of a list:

$$\begin{aligned}
\mathit{head} &\stackrel{\mathrm{def}}{=} \lambda\ell\,.\,\mathsf{case}\ \ell\ \mathsf{of}\ \{\mathsf{nil} \to \bot \mid h :: t \to h\} \\
\mathit{tail} &\stackrel{\mathrm{def}}{=} \lambda\ell\,.\,\mathsf{case}\ \ell\ \mathsf{of}\ \{\mathsf{nil} \to \mathsf{nil} \mid h :: t \to t\}
\end{aligned}$$

where

$$\bot \stackrel{\mathrm{def}}{=} \mathsf{fix}\,x\,.\,x.$$

Since $L \Downarrow H :: T$, it follows from the definition of $\cong^{\mathrm{kl}}$ (in Proposition 3.9) that

$$\emptyset \vdash H \cong^{\mathrm{kl}}_{\sigma} \mathit{head}\ L \qquad \text{and} \qquad \emptyset \vdash T \cong^{\mathrm{kl}}_{[\sigma]} \mathit{tail}\ L$$

and similarly for L', H', T'. We saw in Proposition 3.9 that $\leq^{\mathrm{kl}}$ is contained in $\preceq$. Hence by Corollary A.9 we have

$$\emptyset \vdash H \cong^{\mathrm{gnd}}_{\sigma} \mathit{head}\ L \qquad \text{and} \qquad \emptyset \vdash T \cong^{\mathrm{gnd}}_{[\sigma]} \mathit{tail}\ L$$

and similarly for L', H', T'. By an argument similar to that given above for property (sim 3) of $\leq^{\mathrm{gnd}}$, the fact that $\emptyset \vdash L \leq^{\mathrm{gnd}}_{[\sigma]} L'$ holds implies that

$$\emptyset \vdash \mathit{head}\, L \leq^{\mathrm{gnd}}_{\sigma} \mathit{head}\, L' \qquad \text{and} \qquad \emptyset \vdash \mathit{tail}\, L \leq^{\mathrm{gnd}}_{[\sigma]} \mathit{tail}\, L'.$$

Putting all these fact together we have:

$$\begin{gathered} H \cong^{\mathrm{gnd}} \mathit{head}\, L \leq^{\mathrm{gnd}} \mathit{head}\, L' \cong^{\mathrm{gnd}} H' \\ T \cong^{\mathrm{gnd}} \mathit{tail}\, L \leq^{\mathrm{gnd}} \mathit{tail}\, L' \cong^{\mathrm{gnd}} T' \end{gathered}$$

so that $\emptyset \vdash H \leq^{\mathrm{gnd}}_{\sigma} H'$ and $\emptyset \vdash T \leq^{\mathrm{gnd}}_{[\sigma]} T'$, as required.

This completes the verification that $\leq^{\mathrm{gnd}}$ restricted to closed terms is a PCFL simulation. Hence we have that it is contained in PCFL similarity:

$$\emptyset \vdash M \leq^{\mathrm{gnd}}_{\sigma} N \Rightarrow \emptyset \vdash M \preceq^{\circ}_{\sigma} N \tag{A.43}$$

In order to complete the proof of the Operational Extensionality Theorem, we must extend this implication from closed to open terms. To do this we need to verify that the substitutivity property (2.12) holds for $\leq^{\mathrm{gnd}}$. As mentioned on page 254, this property is essentially a consequence of the fact that β-conversion holds up to contextual equivalence for PCFL.

Lemma A.10. $\leq^{\mathrm{gnd}}$ *satisfies property* (2.12)*, that is, if* $\Gamma, x : \sigma \vdash N \leq^{\mathrm{gnd}}_{\sigma'} N'$*, then it is the case that* $\Gamma \vdash N[M/x] \leq^{\mathrm{gnd}}_{\sigma'} N'[M/x]$ *holds, for any* $M \in \mathit{Exp}_{\sigma}(\Gamma)$.

Proof. For any $\mathcal{C}[-_{\sigma'}]$, since $\mathcal{C}[(\lambda x\,.\,N)M]$ is of the form $\mathcal{C}'[N]$ with $\mathcal{C}'[-_{\sigma'}] \stackrel{\mathrm{def}}{=} \mathcal{C}[(\lambda x\,.\,)M]$, it follows that $\Gamma, x : \sigma \vdash N \leq^{\mathrm{gnd}}_{\sigma'} N'$ implies

$$\Gamma \vdash (\lambda x\,.\,N)M \leq^{\mathrm{gnd}}_{\sigma'} (\lambda x\,.\,N')M. \tag{A.44}$$

We noted on page 263 that as a consequence of Proposition 3.9 $\simeq^{\circ}$ satisfies the β-rule for function application; hence by Corollary A.9 one has

$$\Gamma \vdash N[M/x] \leq^{\mathrm{gnd}}_{\sigma'} (\lambda x\,.\,N)M \quad \text{and} \quad \Gamma \vdash (\lambda x\,.\,N')M \leq^{\mathrm{gnd}}_{\sigma'} N'[M/x].$$

Combining these with (A.44) and transitivity of $\leq^{\mathrm{gnd}}$ yields $\Gamma \vdash N[M/x] \leq^{\mathrm{gnd}}_{\sigma'} N'[M/x]$. $\square$

Suppose $x : \sigma_1, \ldots, x_n : \sigma_n \vdash M \leq^{\mathrm{gnd}}_{\sigma} N$ holds. Then for any $M_i \in \mathit{Exp}_{\sigma_i}$ ($i = 1, \ldots, n$), by applying the lemma repeatedly we get $M[\vec{M}/\vec{x}] \leq^{\mathrm{gnd}}_{\sigma} N[\vec{M}/\vec{x}]$, and hence by (A.43) that $M[\vec{M}/\vec{x}] \preceq_{\sigma} N[\vec{M}/\vec{x}]$. Thus by definition of $\preceq^{\circ}$, we have $\Gamma \vdash M \preceq^{\circ}_{\sigma} N$. Therefore the converse of Corollary A.9 does indeed hold and we have completed the proof of Theorem 3.8.

References

Abelson, H. and G. J. Susman (1985). *Structure and Interpretation of Computer Programs*. MIT Press.

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

Abramsky, S. and C.-H. L. Ong (1993). Full abstraction in the lazy lambda calculus. *Information and Computation 105*, 159–267.

Backhouse, R., P. Chisholm, G. Malcolm, and E. Saaman (1989). Do-it-yourself type theory. *Formal Aspects of Computing 1*, 19–84.

Barendregt, H. P. (1984). *The Lambda Calculus: Its Syntax and Semantics* (revised ed.). North-Holland.

Bird, R. and P. Wadler (1988). *Introduction to Functional Programming*. Prentice-Hall.

Crole, R. L. and A. D. Gordon (1996). Relating operational and denotational semantics for input/output effects. Technical Report 1996/5, University of Leicester Department of Mathematics and Computer Science.

de Roever, W. P. (1978). On backtracking and greatest fixed points. In E. J. Neuhold (Ed.), *Formal Description of Programming Concepts*, pp. 621–639. North-Holland, Amsterdam.

Dybjer, P. and H. P. Sander (1989). A functional programming approach to the specification and verification of concurrent systems. *Formal Aspects of Computing 1*, 303–319.

Egidi, L., F. Honsell, and S. R. della Rocca (1992). Operational, denotational and logical descriptions: a case study. *Fundamenta Informaticae 26*, 149–169.

Gordon, A. D. (1994). *Functional Programming and Input/Output*. Distinguished Dissertations in Computer Science. Cambridge University Press.

Gordon, A. D. (1995a). 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. (1995b, June). Bisimilarity as a theory of functional programming. Minicourse. Notes Series BRICS-NS-95-3, BRICS, Department of Computer Science, University of Aarhus.

Gordon, A. D. and G. D. Rees (1996, January). Bisimilarity for a first-order calculus of objects with subtyping. In *Conference Record of the 23rd ACM Symposium on Principles of Programming Languages, St Petersburg Beach, Florida*, pp. 386–395. ACM Press.

Gunter, C. A. (1992). *Semantics of Programming Languages: Structures and Techniques*. Foundations of Computing. MIT Press.

Harper, R. (1995). A relational proof of correctness of CPS conversion. Draft Version of 9 June, 1995.

Howe, D. J. (1989). Equality in lazy computation systems. In *4th Annual Symposium on Logic in Computer Science*, pp. 198–203. IEEE Computer Society Press, Washington.

Howe, D. J. (1996, February). Proving congruence of bisimulation in functional programming languages. *Information and Computation 124*(2), 103–112.

Martin-Löf, P. (1984). *Intuitionistic Type Theory*. Bibliopolis, Napoli.

Mason, I. A., S. F. Smith, and C. L. Talcott (1996). From operational semantics to domain theory. *Information and Computation*. To appear. Revised and extended version of (Smith 1992).

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

Mason, I. A. and C. L. Talcott (1992). References, local variables and operational reasoning. In *Proceedings of the 7th Annual Symposium on Logic in Computer Science*, pp. 186–197. IEEE Computer Society Press.

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

Milner, R. (1989). *Communication and Concurrency*. Prentice Hall.

Milner, R., M. Tofte, and R. Harper (1990). *The Definition of Standard ML*. MIT Press.

Moggi, E. (1991, July). Notions of computation and monads. *Information and Computation 93*(1), 55–92.

Park, D. (1981). Concurrency and automata on infinite sequences. In P. Deussen (Ed.), *Proceedings of the 5th GI-Conference on Theoretical Computer Science*, Volume 104 of *Lecture Notes in Computer Science*, pp. 167–183. Springer-Verlag, Berlin.

Paulson, L. C. (1991). *ML for the Working Programmer*. Cambridge University Press.

Pitts, A. M. (1994, December). Some notes on inductive and co-inductive techniques in the semantics of functional programs. Notes Series BRICS-NS-94-5, BRICS, Department of Computer Science, University of Aarhus. vi+135 pp, draft version.

Pitts, A. M. (1996). Reasoning about local variables with operationally-based logical relations. In *11th Annual Symposium on Logic in Computer Science*, pp. 152–163. IEEE Computer Society Press, Washington.

Pitts, A. M. and I. D. B. Stark (1993). Observable properties of higher order functions that dynamically create local names, or: What's new? In *Mathematical Foundations of Computer Science, Proc. 18th Int. Symp., Gdańsk, 1993*, Volume 711 of *Lecture Notes in Computer Science*, pp. 122–141. Springer-Verlag, Berlin.

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

Plotkin, G. D. (1981a). Post-graduate lecture notes in advanced domain theory (incorporating the "Pisa Notes"). Dept. of Computer Science, Univ. of Edinburgh.

Plotkin, G. D. (1981b). A structural approach to operational semantics. Technical Report DAIMI FN-19, Aarhus University.

Rees, G. (1994, April). Observational equivalence for a polymorphic lambda calculus. Unpublished note.

Reynolds, J. C. (1981). The essence of Algol. In J. W. de Bakker and J. C. van Vliet (Eds.), *Algorithmic Languages. Proceedings of the International Symposium on Algorithmic Languages*, pp. 345–372. North-Holland, Amsterdam.

Ritter, E. and A. M. Pitts (1995). A fully abstract translation between a λ-calculus with reference types and Standard ML. In *2nd Int. Conf. on Typed Lambda Calculus and Applications, Edinburgh, 1995*, Volume 902 of *Lecture Notes in Computer Science*, pp. 397–413. Springer-Verlag, Berlin.

Sands, D. (1995). Total correctness by local improvement in the transformation of functional programs. To appear (a short version appears in the proceedings of POPL'95).

Scott, D. S. (1982). Domains for denotational semantics. In M. Nielson and E. M. Schmidt (Eds.), *Automata, Languages and Programming, Proceedings 1982*, Volume 140 of *Lecture Notes in Computer Science*. Springer-Verlag, Berlin.

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

Smith, S. F. (1992). From operational to denotational semantics. In S. Brookes *et al* (Ed.), *7th International Conference on Mathematical Foundations of Programming Semantics, Pittsburgh PA*, Volume 598 of *Lecture notes in Computer Science*, pp. 54–76. Springer-Verlag, Berlin.

Stark, I. D. B. (1995). *Names and Higher-Order Functions*. Ph. D. thesis, University of Cambridge. Also published as Technical Report 363, University of Cambridge Computer Laboratory, April 1995.

Wadler, P. (1992). Comprehending monads. *Mathematical Structures in Computer Science 2*, 461–493.

Winskel, G. (1993). *The Formal Semantics of Programming Languages*. Foundations of Computing. Cambridge, Massachusetts: The MIT Press.