

# A Generic Operational Metatheory for Algebraic Effects

Patricia Johann  
Dep. of Comput. and Inf. Sci.  
University of Strathclyde  
Glasgow G1 1XH, UK  
patricia@cis.strath.ac.uk

Alex Simpson  
School of Informatics  
University of Edinburgh  
Edinburgh EH8 9AB, UK  
Alex.Simpson@ed.ac.uk

Janis Voigtländer  
Inst. for Comput. Sci.  
University of Bonn  
53012 Bonn, Germany  
jv@iai.uni-bonn.de

**Abstract**—We provide a syntactic analysis of contextual preorder and equivalence for a polymorphic programming language with effects. Our approach applies uniformly across a range of algebraic effects, and incorporates, as instances: errors, input/output, global state, nondeterminism, probabilistic choice, and combinations thereof. Our approach is to extend Plotkin and Power’s structural operational semantics for algebraic effects (FoSSaCS 2001) with a primitive “basic preorder” on ground type computation trees. The basic preorder is used to derive notions of contextual preorder and equivalence on program terms. Under mild assumptions on this relation, we prove fundamental properties of contextual preorder (hence equivalence) including extensionality properties and a characterisation via applicative contexts, and we provide machinery for reasoning about polymorphism using relational parametricity.

## I. INTRODUCTION

The goal of this paper is to give a systematic account of contextual equivalence for extensions of a functional language with non-functional effects, including errors, input/output, global state, (finite) nondeterminism, probabilistic choice, and combinations thereof. It thus fits into a tradition that includes studies by Mason and Talcott [1], Pitts and Stark [2], and Lassen [3], who respectively consider functional languages with mutable S-expressions, local state, and countable nondeterminism. The main novelty here is that, rather than considering each different kind of effect individually, we provide a general operational framework that can cater for a whole range of effects uniformly. Using this, we obtain *generic operational metatheorems* stating properties of contextual equivalence that hold for every instantiation of our framework to effects of interest.

The programming language we use as a vehicle for this is an extension of Plotkin’s PCF [4] with polymorphism and effects. Although, in the present paper, we consider a call-by-name version only, we believe our methods adapt also to call-by-value (see Section IX for further discussion).

The first difficulty we need to overcome is that the operational semantics of effects are not usually presented

in a uniform manner. For example, in structural operational semantics [5]: pure functional languages are formalised using deterministic transitions  $M \mapsto M'$ ; nondeterminism introduces branching; languages with store require transitions  $(M, s) \mapsto (M', s')$  between state-carrying configurations; languages with input/output use labelled transitions, e.g.,  $M \xrightarrow{!3} M'$  (“output 3”); etc. To exaggerate only slightly, every new feature requires its own format.

To address this problem, we build on ideas of Plotkin and Power. In a major programme of research, see, e.g., [6], [7], [8], [9], they identify the class of *algebraic effects*, which are effects defined via well-behaved effect-triggering operations, and which include all the examples mentioned above. Algebraic effects can be given a uniform operational presentation, based on reducing a program to a *computation tree* (as defined in Section III below) that records all the effects (potentially) encountered during execution [6], [7]. If taken in isolation, the computation-tree presentation treats effect operations merely as uninterpreted syntax. Indeed, it is determined by the signature of effect operations alone. To furnish the effect operations with meaning, Plotkin and Power [6], [7] require a denotational semantics, with a suitable *computational monad* [10]. Their main result is then a generic *computational adequacy* theorem, relating the denotation of a term to that of its computation tree.

Our goal is a systematic study of contextual equivalence  $=_{ctx}$ , which is an operational notion. Accordingly, we propose a syntactic method of ascribing meaning to effect operations. We augment computation trees by identifying the minimum machinery needed to define  $=_{ctx}$ : an assumed equivalence relation  $=_{basic}$  on ground type trees. In fact, for greater generality, we assume a preorder  $\sqsubseteq_{basic}$  on ground type trees, and use it to define contextual preorder  $\sqsubseteq_{ctx}$ .

Our aim is to obtain generic operational metatheorems for  $\sqsubseteq_{ctx}$  (and  $=_{ctx}$ ), applying to any signature of effect operations together with a specified preorder  $\sqsubseteq_{basic}$ . In order to do so, we need to assume that  $\sqsubseteq_{basic}$  satisfies two properties: it must be *admissible* and *compositional*, as defined in Section IV. These properties arise naturally when, as is frequently the case, the  $\sqsubseteq_{basic}$  relation can be described as an *observational preorder* determined by a

This research was carried out during visits of Voigtländer to the Universities of Strathclyde and Edinburgh, supported by a SICSA Distinguished Visiting Fellowship. Simpson’s and Johann’s research is supported by EPSRC grants “Linear Observations and Computational Effects” and “Categorical Foundations for Indexed Programming”, respectively.

*decomposable* family of *open observations* on ground type trees, see Section V. The main examples of algebraic effects all have observational preorders induced in such a way. Thus our assumptions, and hence our theorems, are indeed applicable.

We prove our results using the powerful machinery of  $\top\top$ -closed relations, developed by Pitts and Stark [11], [12], [2]. This adapts naturally to our setting, by using the basic relation  $\sqsubseteq_{basic}$  to induce the required closure operations. The definitions are given in Section VI, whose culmination is the characterisation of contextual preorder as a logical relation (Theorem VI.6). The proof has the usual structure [2], [11], [12], but makes essential use of the admissibility and compositionality of  $\sqsubseteq_{basic}$  to deal with effects.

Our main operational metatheorems about contextual preorder are presented in Section VII. For example, we prove extensionality properties, as well as a “context lemma” (Theorem VII.2) characterising contextual equivalence at arbitrary type via ground type applicative contexts, cf. Milner [13]. For call-by-value languages with effects, such results are known to hold only in restricted form, e.g., the “ciu” theorem of [1]. In contrast, in our call-by-name setting, the theorems remain valid in their classic form. We also establish various results about contextual equivalence between effect operations. For example, effect operations commute with evaluation contexts. This again connects with the work of Plotkin and Power, but whereas for them, the (denotational) property of commuting with evaluation contexts is taken as the definition of an algebraic operation, in our setting the operational analogue holds as a consequence of our operational semantics.

The inclusion of polymorphism in our language allows us to exploit one of the strengths of the logical-relation machinery: it yields principles for reasoning about relational parametricity [11], [12]. In Section VIII, we present one consequence of relational parametricity in our call-by-name effectful setting. The type  $\forall X. (\tau \rightarrow X) \rightarrow X$  is characterised as Moggi’s type  $T(\tau)$  [10], where  $T$  is a computational monad encapsulating the effects present in the language. This provides an operational vindication of a type identity that has previously been demonstrated denotationally for general monadic effects [14].

This paper is intended to be the first in a sequence, which will further extend the generic approach to wider classes of effects and languages. Some specific ideas in this direction and a discussion of other related work appear in Section IX.

For space reasons, all proofs are omitted from this conference version of the paper.

## II. THE LANGUAGE

Our language is a polymorphic version of Plotkin’s PCF [4] with the addition of operations for manipulating effects. The syntax of *types* and *terms* is given in Figure 1, where  $\alpha$  and  $x$  range over disjoint countably infinite sets of

*type variables* and *term variables*, respectively. The syntax is standard except for the effect operations. We shall consider such operations as being specified by a signature  $\Sigma$  that assigns an *arity* to each effect operation  $\sigma$ . We allow four different formats. The simplest case is that of a finite arity operation  $\sigma$ , specified as  $\sigma : \alpha^n \rightarrow \alpha$  for some  $n \in \mathbb{N}$ . Infinite arity operations are specified as  $\sigma : \alpha^{\text{Nat}} \rightarrow \alpha$ . The remaining two cases are parameterised versions of the above. A parameterised finite arity operation is specified as  $\sigma : \text{Nat} \times \alpha^n \rightarrow \alpha$ , and a parameterised infinitary operation is specified as  $\sigma : \text{Nat} \times \alpha^{\text{Nat}} \rightarrow \alpha$ . Computationally interesting examples of different cases are given below.

Types are assigned to terms according to the axioms and rules in Figure 2, where  $\Gamma$  ranges over *typing environments* of the form  $\vec{\alpha}, x_1 :: \tau_1, \dots, x_m :: \tau_m$  for a finite list  $\vec{\alpha}$  of distinct type variables,  $m \in \mathbb{N}$  (where  $\mathbb{N}$  is the set of natural numbers including 0), a list  $\vec{x} = x_1, \dots, x_m$  of distinct term variables, and types  $\tau_1, \dots, \tau_m$  whose free variables are in  $\vec{\alpha}$ . The typing is relative to the signature  $\Sigma$ . In a typing judgement of the form  $\Gamma \vdash M :: \tau$ , with  $\Gamma$  as above, we require that the free variables of the term  $M$  are in  $\vec{\alpha}, \vec{x}$  and that the free variables of the type  $\tau$  are in  $\vec{\alpha}$ . We write  $Typ$  for the set of closed types. Given  $\tau \in Typ$ , we write  $Term(\tau)$  for the set of terms  $M$  for which  $\emptyset \vdash M :: \tau$  is derivable, where  $\emptyset$  is the empty typing environment. Further, we set  $Term = \bigcup_{\tau \in Typ} Term(\tau)$ .

**EXAMPLE 1 (ERROR).** Let  $Err$  be a set of error labels. For each  $e \in Err$ , the signature contains a zero-arity operation (i.e., constant)  $raise_e : \alpha^0 \rightarrow \alpha$ .

**EXAMPLE 2 (NONDETERMINISM).** The signature contains a single binary choice operation  $or : \alpha^2 \rightarrow \alpha$ .

**EXAMPLE 3 (GLOBAL STATE).** Let  $Loc$  be a set of locations. For each  $l \in Loc$ , we include operations  $lookup_l : \alpha^{\text{Nat}} \rightarrow \alpha$  and  $update_l : \text{Nat} \times \alpha \rightarrow \alpha$ . These denote actions that can be performed on  $l$  considered as a storage cell:  $lookup_l(\lambda n :: \text{Nat}. M)$  assigns the content of location  $l$  to variable  $n$  and proceeds as  $M$ ;  $update_l(N; M)$  evaluates  $N$ , assigns the result to  $l$ , and proceeds as  $M$ .

**EXAMPLE 4 (INPUT/OUTPUT).** The signature contains effect operations  $read : \alpha^{\text{Nat}} \rightarrow \alpha$  and  $write : \text{Nat} \times \alpha \rightarrow \alpha$ . Intuitively,  $read(\lambda n :: \text{Nat}. M)$  reads a number from input, assigns it to  $n$  and proceeds as  $M$ ; and  $write(N; M)$  evaluates  $N$  to a number, outputs it, and proceeds as  $M$ .

The examples above do not involve any parameterised infinitary operations, i.e., operations of arity  $\text{Nat} \times \alpha^{\text{Nat}} \rightarrow \alpha$ . Our main reason for including such operations is that they provide us with a single “hardest case” to consider in proofs. In addition, the formulation of some of the above operations may seem cumbersome. For example, in place of the polymorphic  $update_l$ , one might prefer to use an assignment command  $l := M$  of unit type. We adopt the

**Types**  $\tau ::= \alpha \mid \tau \rightarrow \tau \mid \forall \alpha. \tau \mid \text{Nat}$   
**Terms**  $M ::= x \mid \lambda x :: \tau. M \mid M M \mid \Lambda \alpha. M \mid M_\tau \mid \mathbf{fix}(M) \mid Z \mid S(M) \mid$   
 $\mathbf{case} M \mathbf{of} \{Z \Rightarrow M; S(x) \Rightarrow M\} \mid \sigma(M, \dots, M) \mid \sigma(M; M, \dots, M)$

Figure 1. Syntax of the language.

$$\begin{array}{c}
\frac{}{\Gamma, x :: \tau \vdash x :: \tau} \quad \frac{\Gamma, x :: \tau \vdash M :: \tau'}{\Gamma \vdash (\lambda x :: \tau. M) :: \tau \rightarrow \tau'} \quad \frac{\Gamma \vdash M :: \tau \rightarrow \tau' \quad \Gamma \vdash N :: \tau}{\Gamma \vdash (M N) :: \tau'} \\
\\
\frac{\alpha, \Gamma \vdash M :: \tau}{\Gamma \vdash \Lambda \alpha. M :: \forall \alpha. \tau} \quad \frac{\Gamma \vdash G :: \forall \alpha. \tau}{\Gamma \vdash G_{\tau'} :: \tau[\tau'/\alpha]} \quad \frac{\Gamma \vdash F :: \tau \rightarrow \tau}{\Gamma \vdash \mathbf{fix}(F) :: \tau} \\
\\
\frac{}{\Gamma \vdash Z :: \text{Nat}} \quad \frac{\Gamma \vdash M :: \text{Nat}}{\Gamma \vdash S(M) :: \text{Nat}} \quad \frac{\Gamma \vdash M :: \text{Nat} \quad \Gamma \vdash M_1 :: \tau \quad \Gamma, x :: \text{Nat} \vdash M_2 :: \tau}{\Gamma \vdash \mathbf{case} M \mathbf{of} \{Z \Rightarrow M_1; S(x) \Rightarrow M_2\} :: \tau} \\
\\
\frac{\sigma : \alpha^n \rightarrow \alpha \quad \forall i. \Gamma \vdash M_i :: \tau}{\Gamma \vdash \sigma(M_1, \dots, M_n) :: \tau} \quad \frac{\sigma : \alpha^{\text{Nat}} \rightarrow \alpha \quad \Gamma \vdash M_1 :: \text{Nat} \rightarrow \tau}{\Gamma \vdash \sigma(M_1) :: \tau} \\
\\
\frac{\sigma : \text{Nat} \times \alpha^n \rightarrow \alpha \quad \Gamma \vdash M :: \text{Nat} \quad \forall i. \Gamma \vdash M_i :: \tau}{\Gamma \vdash \sigma(M; M_1, \dots, M_n) :: \tau} \quad \frac{\sigma : \text{Nat} \times \alpha^{\text{Nat}} \rightarrow \alpha \quad \Gamma \vdash M :: \text{Nat} \quad \Gamma \vdash M_1 :: \text{Nat} \rightarrow \tau}{\Gamma \vdash \sigma(M; M_1) :: \tau}
\end{array}$$

Figure 2. Type assignment relation.

former because it fits nicely into our polymorphic type theory, and does not require a unit type. The equivalence of the two styles is a general property of algebraic effects [9].

### III. OPERATIONAL SEMANTICS

The subset of *Term* whose elements (called *values*) respect the following grammar is denoted by *Value*:

$$V ::= \lambda x :: \tau. M \mid \Lambda \alpha. M \mid Z \mid S(V).$$

We write  $Value(\tau)$  for the values of type  $\tau$ . The elements of  $Value(\text{Nat})$  are the *numerals*  $Z, S(Z), S^2(Z), \dots$ , for which we use the shorthand  $\bar{0}, \bar{1}, \bar{2}, \dots$ .

The operational semantics of our language needs to specify both how closed terms evaluate to values and also any effectful behaviour that may occur during execution. Following Plotkin and Power [7], we achieve the latter by abstracting away from the nature of the effects performed. We use a syntactically defined *computation tree* to capture every effect operation that could potentially occur during the given execution, irrespective of how the effect operations are themselves interpreted. The operational semantics will determine a function  $|\cdot|$  mapping every closed term  $M \in Term(\tau)$  to a computation tree  $|M|$  of type  $\tau$ . We first define the notion of computation tree, and then the machinery via which the computation tree of a term is determined.

**DEFINITION III.1.** A *computation tree* of closed type  $\tau$  is a labelled tree each node of which is one of the following.

- A leaf node labelled  $\perp$ .
- A leaf node labelled  $V$ , where  $V \in Value(\tau)$ .
- A node labelled  $\sigma$  with children  $t_1, \dots, t_n$  indexed by  $\{1, \dots, n\}$ , where  $\sigma$  is an effect operation  $\sigma : \alpha^n \rightarrow \alpha$ .

- A node labelled  $(\sigma, m)$  with children  $t_1, \dots, t_n$  indexed by  $\{1, \dots, n\}$ , where  $\sigma : \text{Nat} \times \alpha^n \rightarrow \alpha$ .
- A node labelled  $\sigma$  with children  $t_0, t_1, t_2, \dots$  indexed by  $\mathbb{N}$ , where  $\sigma : \alpha^{\text{Nat}} \rightarrow \alpha$ .
- A node labelled  $(\sigma, m)$  with children  $t_0, t_1, t_2, \dots$  indexed by  $\mathbb{N}$ , where  $\sigma : \text{Nat} \times \alpha^{\text{Nat}} \rightarrow \alpha$ .

It is immediate that every subtree of a computation tree is itself a computation tree of the same type. We define a partial ordering on computation trees (of the same type) by  $t_1 \sqsubseteq t_2$  if  $t_1$  can be obtained from  $t_2$  by pruning (possibly infinitely many) subtrees and labelling the pruning points with  $\perp$ . It is standard that this order endows the set  $Trees(\tau)$  of trees of type  $\tau$  with the structure of an  $\omega$ -complete partial order ( $\omega\text{cpo}$ ), with least element  $\perp$ .

The mechanism for evaluating a term to its computation tree uses *redex/reduct-pairs* and a notion of *reduction in context*. We define these appropriately for call-by-name evaluation. The redex/reduct-pairs are written  $R \rightsquigarrow R'$  (with  $R, R' \in Term$ ) and listed in the following table:

$R$	$R'$
$(\lambda x :: \tau. M) A$	$M[A/x]$
$(\Lambda \alpha. M)_\tau$	$M[\tau/\alpha]$
$\mathbf{case} \bar{0} \mathbf{of} \{Z \Rightarrow M_1; S(x) \Rightarrow M_2\}$	$M_1$
$\mathbf{case} \bar{n} + \bar{1} \mathbf{of} \{Z \Rightarrow M_1; S(x) \Rightarrow M_2\}$	$M_2[\bar{n}/x]$
$\mathbf{fix}(F)$	$F \mathbf{fix}(F)$

To describe reduction in context, we use the notions of *evaluation frames* and *evaluation stacks*, given by the grammars

$$\begin{aligned}
E &::= (- M) \mid -_\tau \mid S(-) \mid \sigma(-; M, \dots, M) \mid \\
&\quad (\mathbf{case} - \mathbf{of} \{Z \Rightarrow M; S(x) \Rightarrow M\}) \\
S &::= Id \mid S \circ E \quad ,
\end{aligned}$$

where the evaluation frame  $\sigma(-; M, \dots, M)$  is only present for  $\sigma \in \Sigma$  with  $\sigma : \text{Nat} \times \alpha^n \rightarrow \alpha$  or  $\sigma : \text{Nat} \times \alpha^{\text{Nat}} \rightarrow \alpha$ . If an evaluation stack comprises a single evaluation frame  $E$ , we denote it by  $E$  rather than  $Id \circ E$ . Given an evaluation frame  $E$  and a term  $M$ , we write  $E\{M\}$  for the term that results from replacing “ $-$ ” by  $M$  in  $E$ . Similarly, given a stack  $S$  and term  $M$ , the application  $S\{M\} \in \text{Term}(\tau')$  is defined by induction on the structure of  $S$  via the equations  $Id\{M\} = M$  and  $(S' \circ E)\{M\} = S'\{E\{M\}\}$ .

A type assignment relation  $\Gamma \vdash S :: \tau \multimap \tau'$  (where  $\Gamma$  again ranges over typing environments) assigns argument and result types to certain evaluation stacks. We define  $\Gamma \vdash S :: \tau \multimap \tau'$  to hold if  $\Gamma, x :: \tau \vdash S\{x\} :: \tau'$ , where  $x$  is fresh for  $\Gamma$ . Given  $\tau, \tau' \in \text{Typ}$ , we write  $\text{Stack}(\tau, \tau')$  for the set of stacks  $S$  for which  $\emptyset \vdash S :: \tau \multimap \tau'$  is derivable. We set, for every  $\tau \in \text{Typ}$ ,  $\text{Stack}(\tau) = \text{Stack}(\tau, \text{Nat})$ .

Now a *transition*  $(S_1, M_1) \multimap (S_2, M_2)$  (with  $M_1, M_2 \in \text{Term}(\tau)$  and  $S_1, S_2 \in \text{Stack}(\tau, \tau')$ , for some  $\tau$  and  $\tau'$ ) is possible for exactly the following combinations:

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

To define the computation tree associated with a term  $M$ , one evaluates the term as the configuration  $(Id, M)$ . If this evaluation terminates with a value, that value is the tree. If it stops at an effect operation, then that operation is used to label the current node in the tree, and the procedure continues with appropriate stack/term configurations for each of the children nodes. If ever one gets to a nonterminating configuration, then the current node gets labelled with  $\perp$ . The definition below implements this rigorously.

**DEFINITION III.2.** We define a family of functions  $|\cdot|_{\tau, \tau'}$ , indexed by  $\tau, \tau' \in \text{Typ}$ , mapping  $S \in \text{Stack}(\tau, \tau')$ , and  $M \in \text{Term}(\tau)$  to a computation tree  $|S, M|_{\tau, \tau'} \in \text{Trees}(\tau')$ . These functions are given by the canonical (i.e., least in the pointwise order on functions into  $\text{Trees}(\tau')$ ) solution of the following recursive equation (omitting subscripts).

$$|S, M| = \begin{cases} M & \text{if } S = Id \text{ and } M \in \text{Value} \\ |S', M'| & \text{if } (S, M) \multimap (S', M') \\ \sigma(|S, M_1|, \dots, |S, M_n|) & \text{if } M = \sigma(M_1, \dots, M_n), \sigma : \alpha^n \rightarrow \alpha \\ \sigma(|S, M' \bar{0}|, |S, M' \bar{1}|, |S, M' \bar{2}|, \dots) & \text{if } M = \sigma(M'), \sigma : \alpha^{\text{Nat}} \rightarrow \alpha \\ (\sigma, m)(|S, M_1|, \dots, |S, M_n|) & \text{if } M = \sigma(\bar{m}; M_1, \dots, M_n), \\ & \sigma : \text{Nat} \times \alpha^n \rightarrow \alpha \\ (\sigma, m)(|S, M' \bar{0}|, |S, M' \bar{1}|, |S, M' \bar{2}|, \dots) & \text{if } M = \sigma(\bar{m}; M'), \sigma : \text{Nat} \times \alpha^{\text{Nat}} \rightarrow \alpha \end{cases}$$

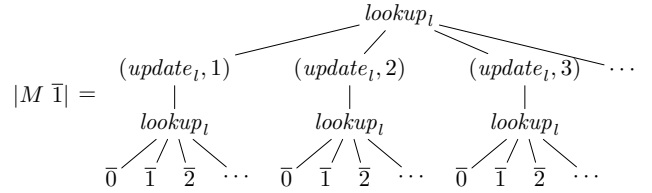
Finally, for  $M \in \text{Term}(\tau)$ , we define  $|M| = |Id, M|$ .

Note that the different cases in the above definition of  $|S, M|$  are exhaustive and mutually exclusive. Also, because we demand the canonical solution, we obtain  $|S, M| = \perp$  when  $(S, M)$  diverges under  $\multimap$ . In particular,  $|S, \Omega_\tau| = \perp$ , where  $\Omega_\tau = \text{fix}(\lambda x :: \tau. x)$ .

**EXAMPLE 3 (CONTINUED).** In the signature for global state, define  $M = \text{fix}(\lambda f :: \text{Nat} \rightarrow \text{Nat}. \lambda n :: \text{Nat}. B)$  where  $B$  is

case  $n$  of  $\{Z \Rightarrow \text{lookup}_l(\lambda c :: \text{Nat}. c);$   
 $S(n') \Rightarrow \text{lookup}_l(\lambda c :: \text{Nat}. \text{update}_l(S(c); f n'))\}$ .

Then  $M \in \text{Term}(\text{Nat} \rightarrow \text{Nat})$  and:



In this example, the computation tree contains branches that could not possibly arise according to the intended interpretation of the operations. For instance, the subtree  $(\text{update}_l, 1)(\text{lookup}_l(\bar{0}, \bar{1}, \bar{2}, \dots))$  represents the action: first, store the value 1 in the location  $l$ ; next, read the content of  $l$ ; if this is 0, return  $\bar{0}$ ; if it is 1, return  $\bar{1}$ ; if it is 2, return  $\bar{2}$ ; etc. In this case, only the second branch from the choice point is possible, and the value returned will be  $\bar{1}$ . The reason for including such redundancy is that computation trees treat operations as syntax. Without specific interpretation, it is possible that all branches might be relevant. Indeed, if the example is transcribed to the signature for input/output then, in the corresponding subtree  $(\text{write}, 1)(\text{read}(\bar{0}, \bar{1}, \bar{2}, \dots))$ , all branches are relevant.

#### IV. CONTEXTUAL PREORDER

Our operational semantics treats effect operations merely as uninterpreted syntax. In this section we show that we can uniformly account for the properties of specific effects not by directly defining how they behave, but instead by using computation trees to define a notion of contextual equivalence (and, more generally, preorder) between programs.

To define contextual preorder on programs, we require, as given, a basic preorder relation  $\sqsubseteq_{\text{basic}}$  on ground type computation trees  $\text{Trees}(\text{Nat})$ . We illustrate some natural choices for  $\sqsubseteq_{\text{basic}}$  in the context of our running examples.

**EXAMPLE 1 (CONTINUED).** The only ground type computation trees are  $\perp$ , numerals  $\bar{n}$ , and errors  $\text{raise}_e$ . The natural preorder is the expected flat partial ordering:  $x \sqsubseteq_{\text{basic}}^{\text{err}} y$  iff  $x = \perp$  or  $x = y$ .

**EXAMPLE 2 (CONTINUED).** Different choices for  $\sqsubseteq_{\text{basic}}$  correspond to different paradigms for nondeterministic computation. We consider just two.

*Ordinary nondeterminism.*: Define the set  $results(t)$  of possible results of a computation tree  $t$  by:

$$results(t) = \{n \mid t \text{ has a leaf labelled with } \bar{n}\} \cup \{\perp \mid t \text{ has a leaf labelled with } \perp \text{ or } t \text{ has an infinite branch}\}$$

Define:  $t \sqsubseteq_{basic}^{nd} t'$  iff either (i)  $\perp \in results(t)$  and  $results(t) - \{\perp\} \subseteq results(t')$ , or (ii)  $results(t) = results(t')$ . This is the well-known Egli-Milner ordering.

*Probabilistic choice.*: We consider the *or* operation as performing a fair probabilistic choice (coin toss) between the two alternatives. Given  $t \in Trees(\text{Nat})$  and  $n \in \text{Nat}$ , define  $\mathbf{P}(t = \bar{n})$  to be the probability, within the tree  $t$  (considered as a Markov chain), of reaching a leaf labelled with  $\bar{n}$ . Define  $t \sqsubseteq_{basic}^{pr} t'$  iff, for all  $n$ ,  $\mathbf{P}(t = \bar{n}) \leq \mathbf{P}(t' = \bar{n})$ .

EXAMPLE 3 (CONTINUED). Define  $States = \text{Loc} \rightarrow \mathbb{N}$ . We define a partial function  $exec$  from  $Trees(\text{Nat}) \times States$  to  $\mathbb{N} \times States$ , where  $exec(t, s)$  represents the result of “executing”  $t$  starting from state  $s$ . The idea is that  $exec(t, s) = (n, s')$  iff the execution returns value  $\bar{n}$  with  $s'$  the state at termination. Formally,  $exec$  is the least-defined partial function satisfying the recursive equations:

$$\begin{aligned} exec(\bar{n}, s) &= (n, s) \\ exec(lookup_l(t_0, t_1, \dots), s) &= exec(t_{s(l)}, s) \\ exec(update_l(m)(t), s) &= exec(t, s[l := m]) \end{aligned}$$

where  $s[l := m]$  is the evident state obtained from  $s$  by modifying  $l$ . One sees that the definition of  $exec$  is such that redundant branches of the computation tree, as discussed at the end of Section III, are ignored. Define  $t \sqsubseteq_{basic}^{st} t'$  iff: for every  $s \in States$ , if  $exec(t, s)$  is defined then so is  $exec(t', s)$  and  $exec(t, s) = exec(t', s)$ .

EXAMPLE 4 (CONTINUED). An *i/o trace* is a finite sequence of elements of three kinds: an *input*  $?n$ , an *output*  $!n$ , or a *return*  $\cdot n$ , where, in each case,  $n \in \mathbb{N}$ . The sets  $io(t)$  of (not necessarily completed) *i/o traces* of computation trees  $t$  are defined as the smallest satisfying:

$$\begin{aligned} io(\bar{n}) &= \{\cdot n, \varepsilon\} \\ io(read(t_0, t_1, \dots)) &= \{(?n)\alpha \mid \alpha \in io(t_n)\} \cup \{\varepsilon\} \\ io(write, n)(t) &= \{(!n)\alpha \mid \alpha \in io(t)\} \cup \{\varepsilon\} \end{aligned}$$

where  $\varepsilon$  is the empty sequence, and we concatenate by juxtaposition. Define  $t \sqsubseteq_{basic}^{io} t'$  iff  $io(t) \subseteq io(t')$ .

Note that in Examples 1 and 4, the basic preorders coincide with the partial order on  $Trees(\text{Nat})$  defined in Section III.

The above definitions of the  $\sqsubseteq_{basic}$  preorder make use of the standard devices one meets in the usual presentations of operational semantics (e.g., the set  $States$  of global states). What we have gained, with respect to the standard presentations, is that each example now fits into the same

uniform format; its definition consists of two components: a signature, and a preorder  $\sqsubseteq_{basic}$  on  $Trees(\text{Nat})$ .

We henceforth assume given a signature and  $\sqsubseteq_{basic}$  preorder, and we use this data to define *contextual preorder* for our language. We expect that the contextual preorder should satisfy natural compatibility (precongruence) conditions, and it should be the largest such relation consistent with the relation  $\sqsubseteq_{basic}$  at type  $\text{Nat}$ .

DEFINITION IV.1. A *well-typed relation*  $\mathcal{E}$  comprises 4-tuples of the form  $(\Gamma, M, M', \tau)$  with  $\Gamma \vdash M :: \tau$  and  $\Gamma \vdash M' :: \tau$ . We write  $\Gamma \vdash M \mathcal{E} M' :: \tau$  when the tuple  $(\Gamma, M, M', \tau)$  is in  $\mathcal{E}$ , and we abbreviate this to  $M \mathcal{E} M' :: \tau$  if  $\Gamma = \emptyset$  (and we often omit  $\tau$ , which is uniquely determined). The notions of *reflexivity*, *transitivity* and *symmetry* apply to well-typed relations in the obvious way.

DEFINITION IV.2. A well-typed relation  $\mathcal{E}$  is said to be *compatible* if it is closed under the axioms and rules in Figure 3. It is said to be  $\sqsubseteq_{basic}$ -adequate if for every  $M, M' \in Term(\text{Nat})$ , we have that  $M \mathcal{E} M'$  implies  $|M| \sqsubseteq_{basic} |M'|$ .

Note that every compatible relation is reflexive.

The following result is standard (see, e.g., [3], [11], [12]).

PROPOSITION IV.3. *There exists a largest  $\sqsubseteq_{basic}$ -adequate compatible relation. This relation is a preorder. It is an equivalence relation if  $\sqsubseteq_{basic}$  is.*

*Contextual preorder*  $\sqsubseteq_{ctx}$  is defined to be the largest  $\sqsubseteq_{basic}$ -adequate compatible relation, as given by the proposition above. *Contextual equivalence*  $=_{ctx}$  is then defined as  $(\sqsubseteq_{ctx}) \cap (\supseteq_{ctx})$ . In fact, contextual equivalence itself arises as a special case of a contextual preorder, namely  $=_{ctx}$  is the contextual preorder obtained by taking the equivalence relation  $(\sqsubseteq_{basic}) \cap (\supseteq_{basic})$  as the basic preorder.

The main goal of this paper is to establish fundamental properties of contextual preorder and equivalence in the general context we have developed thus far: a signature of effect operations together with a primitive preorder  $\sqsubseteq_{basic}$  on ground type computation trees. However, we will need to impose two further conditions on the preorder  $\sqsubseteq_{basic}$  for the results to hold. The first exploits the  $\omega\text{cpo}$  structure on  $Trees(\text{Nat})$ , discussed in Section III.

DEFINITION IV.4. We say that  $\sqsubseteq_{basic}$  is *admissible* if, for all ascending chains  $(t_n), (t'_n)$  in  $Trees(\text{Nat})$ , if  $t_n \sqsubseteq_{basic} t'_n$ , for all  $n$ , then  $(\bigsqcup_{n \geq 0} t_n) \sqsubseteq_{basic} (\bigsqcup_{n \geq 0} t'_n)$ .

The second condition requires a substitution operation on ground type computation trees. Given  $t, t_0, t_1, \dots \in Trees(\text{Nat})$ , define  $t\{t_n/\bar{n}\}_n$  to be the computation tree obtained by replacing, for every  $n \in \mathbb{N}$ , each node labelled  $\bar{n}$  in  $t$  with the tree  $t_n$  as a subtree rooted at that node.

DEFINITION IV.5. We say that  $\sqsubseteq_{basic}$  is *compositional* if,

$$\begin{array}{c}
\frac{\Gamma, x :: \tau \vdash x \mathcal{E} x :: \tau}{\Gamma \vdash \mathbf{fix}(F) \mathcal{E} \mathbf{fix}(F') :: \tau} \quad \frac{\Gamma \vdash Z \mathcal{E} Z :: \text{Nat}}{\Gamma \vdash (\lambda x :: \tau. M) \mathcal{E} (\lambda x :: \tau. M') :: \tau \rightarrow \tau'} \quad \frac{\Gamma \vdash M \mathcal{E} M' :: \text{Nat}}{\Gamma \vdash S(M) \mathcal{E} S(M') :: \text{Nat}} \\
\frac{\Gamma \vdash F \mathcal{E} F' :: \tau \rightarrow \tau'}{\Gamma \vdash (F A) \mathcal{E} (F' A') :: \tau'} \quad \frac{\Gamma \vdash A \mathcal{E} A' :: \tau}{\Gamma \vdash \Lambda \alpha. M \mathcal{E} \Lambda \alpha. M' :: \forall \alpha. \tau} \quad \frac{\alpha, \Gamma \vdash M \mathcal{E} M' :: \tau}{\Gamma \vdash G_{\tau'} \mathcal{E} G'_{\tau'} :: \tau[\tau'/\alpha]} \\
\frac{\Gamma \vdash M \mathcal{E} M' :: \text{Nat} \quad \Gamma \vdash M_1 \mathcal{E} M'_1 :: \tau \quad \Gamma, x :: \text{Nat} \vdash M_2 \mathcal{E} M'_2 :: \tau}{\Gamma \vdash (\mathbf{case} M \mathbf{of} \{Z \Rightarrow M_1; S(x) \Rightarrow M_2\}) \mathcal{E} (\mathbf{case} M' \mathbf{of} \{Z \Rightarrow M'_1; S(x) \Rightarrow M'_2\}) :: \tau} \\
\frac{\sigma : \text{Nat} \times \alpha^{\text{Nat}} \rightarrow \alpha \quad \Gamma \vdash M \mathcal{E} M' :: \text{Nat} \quad \Gamma \vdash M_1 \mathcal{E} M'_1 :: \text{Nat} \rightarrow \tau}{\Gamma \vdash \sigma(M; M_1) \mathcal{E} \sigma(M'; M'_1) :: \tau}
\end{array}$$

... the evident similar rules for other arities of  $\sigma$

Figure 3. Compatibility properties.

whenever  $t \sqsubseteq_{\text{basic}} t'$  and  $t_n \sqsubseteq_{\text{basic}} t'_n$ , for all  $n$ , then it holds that  $t\{t_n/\bar{n}\}_n \sqsubseteq_{\text{basic}} t'\{t'_n/\bar{n}\}_n$

Admissible and compositional preorders enjoy useful closure properties. If  $\sqsubseteq_{\text{basic}}$  is admissible and compositional, then so is its converse  $\sqsupseteq_{\text{basic}}$ . Also, the intersection of any family of admissible and compositional preorders is again admissible and compositional. Thus, when  $\sqsubseteq_{\text{basic}}$  is admissible and compositional, so is  $=_{\text{basic}}$ .

In the next section we shall see that all the instances of  $\sqsubseteq_{\text{basic}}$  relations considered above, for the various running examples, are indeed admissible and compositional.

## V. OBSERVATIONAL PREORDERS

In this section we describe a useful and rather general method of specifying  $\sqsubseteq_{\text{basic}}$  relations and establishing their admissibility and compositionality. This defines  $\sqsubseteq_{\text{basic}}$  via a collection of possible ‘‘observations’’ on ground type computation, and thus implements the contextual preorder  $\sqsubseteq_{\text{ctx}}$  as an ‘‘observational preorder’’.

**DEFINITION V.1.** An *observation* is a subset of  $\text{Trees}(\text{Nat})$ .

The idea is that the chosen subset of  $\text{Trees}(\text{Nat})$  represents those computations that we observe to satisfy some property being tested for.

**DEFINITION V.2.** Given a family  $\mathcal{O}$  of observations, the preorder  $\sqsubseteq_{\text{basic}}^{\mathcal{O}}$  is defined by:  $t \sqsubseteq_{\text{basic}}^{\mathcal{O}} t'$  iff, for all  $O \in \mathcal{O}$ , it holds that  $t \in O$  implies  $t' \in O$ .

The basic preorders defined in the previous section, for our running examples, all arise from families of observations.

**EXAMPLE 1 (CONTINUED).** The family of observations is  $\mathcal{O}^{\text{err}} = \{\{\bar{n}\} \mid n \in \mathbb{N}\} \cup \{\{\text{raise}_e\} \mid e \in \text{Err}\}$ .

**EXAMPLE 2 (CONTINUED).**

*Ordinary nondeterminism.* The family is  $\mathcal{O}^{\text{nd}} = \{\diamond n \mid n \in \mathbb{N}\} \cup \{\square F \mid F \subseteq \mathbb{N} \text{ and } F \text{ finite}\}$ , where  $\diamond n = \{t \mid n \in \text{results}(t)\}$  and  $\square F = \{t \mid \text{results}(t) \subseteq F\}$ .

*Probabilistic choice.* The family of observations is  $\mathcal{O}^{\text{pr}} = \{\mathbf{P}_{n,r} \mid n \in \mathbb{N}, r \in [0, 1]\}$  where  $\mathbf{P}_{n,r} = \{t \mid \mathbf{P}(t = \bar{n}) > r\}$ .

**EXAMPLE 3 (CONTINUED).** The family of observations is  $\mathcal{O}^{\text{st}} = \{s \mapsto (n, s') \mid s, s' \in \text{States}, n \in \mathbb{N}\}$  where  $s \mapsto (n, s') = \{t \mid \text{exec}(t, s) = (n, s')\}$ .

**EXAMPLE 4 (CONTINUED).** The family of observations is  $\mathcal{O}^{\text{io}} = \{\langle \alpha \rangle \mid \alpha \text{ an i/o trace}\}$ , where  $\langle \alpha \rangle = \{t \mid \alpha \in \text{io}(t)\}$ .

In addition to the conceptual appeal of defining contextual preorder through specifying observations, this approach provides a convenient way of establishing the admissibility and compositionality of  $\sqsubseteq_{\text{basic}}$ .

**DEFINITION V.3.** An observation  $O$  is *open* (in the Scott topology) if it is upwards closed as a subset of  $\text{Trees}(\text{Nat})$  and, for every ascending chain  $(t_n)$  of trees,  $(\bigsqcup_{n \geq 0} t_n) \in O$  implies there exists  $n \in \mathbb{N}$  with  $t_n \in O$ .

Openness is an intuitively reasonable requirement to place on observations. It reflects that, to observe that a tree  $t_\omega$  satisfies a property, we must use only finitely much information about  $t_\omega$ .

**PROPOSITION V.4.** If  $\mathcal{O}$  is a family of open observations, then  $\sqsubseteq_{\text{basic}}^{\mathcal{O}}$  is admissible.

The converse does not hold. Indeed, there is a simple characterisation of preorders definable via open observations.

**PROPOSITION V.5.** A preorder  $\sqsubseteq_{\text{basic}}$  is definable as  $\sqsubseteq_{\text{basic}}^{\mathcal{O}}$  for some family of open observations  $\mathcal{O}$ , if and only if, for every  $t \in \text{Trees}(\text{Nat})$ , the set  $\{t' \mid t' \not\sqsubseteq_{\text{basic}} t\}$  is open.

The following example of an interesting preorder violating this condition was suggested by Paul Levy.

EXAMPLE 2 (CONTINUED). The *inclusion preorder* for nondeterminism is defined by  $t \sqsubseteq_{basic}^{inclusion} t'$  iff  $results(t) \subseteq results(t')$ . Then  $\sqsubseteq_{basic}^{inclusion}$  is not determined by a family of open observations since  $\perp \not\sqsubseteq_{basic}^{inclusion} \bar{0}$ . Nevertheless, it is admissible and compositional.

DEFINITION V.6. A family of observations,  $\mathcal{O}$ , is called *decomposable* if, whenever  $t\{t_n/\bar{n}\}_n \in O \in \mathcal{O}$ , there exist subfamilies  $\mathcal{O}'$  and  $(\mathcal{O}'_n)$  of  $\mathcal{O}$  such that:

- (i)  $t \in \bigcap \mathcal{O}'$  and  $t_n \in \bigcap \mathcal{O}'_n$  for all  $n$ ; and
- (ii) whenever  $t'$  and  $(t'_n)$  are such that  $t' \in \bigcap \mathcal{O}'$  and  $t'_n \in \bigcap \mathcal{O}'_n$  for all  $n$ , then it holds that  $t'\{t'_n/\bar{n}\}_n \in O$ .

PROPOSITION V.7. *The relation  $\sqsubseteq_{basic}^{\mathcal{O}}$  is compositional if and only if  $\mathcal{O}$  is a decomposable family of observations.*

Propositions V.4 and V.7 provide a uniform means for specifying admissible and compositional  $\sqsubseteq_{basic}$  relations: such relations are obtained as  $\sqsubseteq_{basic}^{\mathcal{O}}$  whenever  $\mathcal{O}$  is a decomposable family of open observations. One gain here is that openness and decomposability often turn out to be easy to establish in practice. Indeed, Propositions V.4 and V.7 serve the simplification of reducing properties of binary relations between trees to properties of sets of trees.

All the families of observations given for the examples above are easily shown to satisfy the openness and decomposability conditions. Thus the  $\sqsubseteq_{basic}$  relations for our running examples are all both admissible and compositional.

## VI. A LOGICAL RELATION CHARACTERISATION OF $\sqsubseteq_{ctx}$

We now take up our main task of establishing properties of contextual preorder. *For the remainder of the paper, we assume that  $\sqsubseteq_{basic}$  is admissible and compositional.*

This section develops the main technical tool: a characterisation of  $\sqsubseteq_{ctx}$  via a logical relation. The presence of recursion and effect operations in our language means we need to restrict to a well-behaved class of term relations. For this, we use the closure operator approach of Pitts and Stark [2], [11], [12]. This treats terms and stacks as complementary ingredients which combine to reduce relations to a universal “test” relation. This adapts naturally to our setting, by using  $\sqsubseteq_{basic}$  as the test relation.

Given  $\tau, \tau' \in Typ$ , we define  $Rel(\tau, \tau') = \mathcal{P}(Term(\tau) \times Term(\tau'))$  and  $Rel^\top(\tau, \tau') = \mathcal{P}(Stack(\tau) \times Stack(\tau'))$ . For  $r \in Rel(\tau, \tau')$  and  $s \in Rel^\top(\tau, \tau')$  we define  $r^\top \in Rel^\top(\tau, \tau')$  and  $s^\top \in Rel(\tau, \tau')$  by:

$$\begin{aligned} (S, S') \in r^\top &\text{ iff } \forall (M, M') \in r. |S, M| \sqsubseteq_{basic} |S', M'| \\ (M, M') \in s^\top &\text{ iff } \forall (S, S') \in s. |S, M| \sqsubseteq_{basic} |S', M'| \end{aligned}$$

As for any order-preserving Galois connection, we always have  $r \subseteq r^{\top\top}$ ,  $(r^{\top\top})^\top = r^\top$ , and  $r_1 \subseteq r_2 \Rightarrow r_1^{\top\top} \subseteq r_2^{\top\top}$ .

We say that  $r$  is  $\top\top$ -closed if  $r^{\top\top} = r$ . An important property of  $\top\top$ -closed relations is that they respect contextual preorder.

LEMMA VI.1. *Suppose  $r \in Rel(\tau_1, \tau_2)$  is  $\top\top$ -closed. For all  $M_1, M'_1 \in Term(\tau_1)$  and  $M_2, M'_2 \in Term(\tau_2)$ , if  $M'_1 \sqsubseteq_{ctx} M_1$ ,  $(M_1, M_2) \in r$  and  $M_2 \sqsubseteq_{ctx} M'_2$ , then  $(M'_1, M'_2) \in r$ .*

We now define the logical relation, which maps a type and a list containing relations as interpretations for the type’s free variables to a new relation.

DEFINITION VI.2. For every type  $\tau$ ,  $n \in \mathbb{N}$ , list  $\vec{\alpha} = \alpha_1, \dots, \alpha_n$  of distinct type variables containing the free variables of  $\tau$ , lists  $\vec{\tau} = \tau_1, \dots, \tau_n$  and  $\vec{\tau}' = \tau'_1, \dots, \tau'_n$  of closed types, and list  $\vec{r} = r_1, \dots, r_n$  with  $r_i \in Rel(\tau_i, \tau'_i)$  for every  $1 \leq i \leq n$ , we define  $\Delta_\tau(\vec{r}/\vec{\alpha}) \in Rel(\tau[\vec{\tau}/\vec{\alpha}], \tau[\vec{\tau}'/\vec{\alpha}])$  by induction on the structure of  $\tau$  by:

$$\begin{aligned} \Delta_{\alpha_i}(\vec{r}/\vec{\alpha}) &= r_i \\ \Delta_{\tau' \rightarrow \tau''}(\vec{r}/\vec{\alpha}) &= \{(F, F') \mid \forall (A, A') \in \Delta_{\tau'}(\vec{r}/\vec{\alpha}). \\ &\quad (F A, F' A') \in \Delta_{\tau''}(\vec{r}/\vec{\alpha})\} \\ \Delta_{\forall \alpha. \tau'}(\vec{r}/\vec{\alpha}) &= \{(G, G') \mid \forall \tau_2, \tau'_2 \in Typ, r \in Rel(\tau_2, \tau'_2). \\ &\quad (G_{\tau_2}, G'_{\tau'_2}) \in \Delta_{\tau'}(\vec{r}, r^{\top\top}/\vec{\alpha}, \alpha)\} \\ \Delta_{Nat}(\vec{r}/\vec{\alpha}) &= \{(Id, Id)\}^\top \end{aligned}$$

LEMMA VI.3. *Let  $\tau$ ,  $\vec{\alpha}$ , and  $\vec{r}$  be as in Definition VI.2. If every relation in  $\vec{r}$  is  $\top\top$ -closed, then so is  $\Delta_\tau(\vec{r}/\vec{\alpha})$ .*

To characterise  $\sqsubseteq_{ctx}$  via the logical relation from Definition VI.2, we first have to lift the latter from closed terms to a relation on terms possibly containing free variables, that is, to a well-typed relation in the sense of Definition IV.1.

DEFINITION VI.4. Given  $\Gamma = \vec{\alpha}, x_1 :: \tau_1, \dots, x_m :: \tau_m$  such that  $\Gamma \vdash M :: \tau$  and  $\Gamma \vdash M' :: \tau$ , we write  $\Gamma \vdash M \Delta M' :: \tau$  if  $(M[\vec{\sigma}/\vec{\alpha}, \vec{N}/\vec{x}], M'[\vec{\sigma}'/\vec{\alpha}, \vec{N}'/\vec{x}]) \in \Delta_\tau(\vec{r}/\vec{\alpha})$  for every  $\vec{\sigma} = \sigma_1, \dots, \sigma_n$  and  $\vec{\sigma}' = \sigma'_1, \dots, \sigma'_n$  with all types closed, every  $\vec{r} = r_1, \dots, r_n$  where each  $r_i \in Rel(\sigma_i, \sigma'_i)$  is  $\top\top$ -closed, and every  $\vec{N} = N_1, \dots, N_m$  and  $\vec{N}' = N'_1, \dots, N'_m$  where each  $(N_j, N'_j) \in \Delta_{\tau_j}(\vec{r}/\vec{\alpha})$ .

The main step in proving that  $\Delta$  and  $\sqsubseteq_{ctx}$  coincide is to establish that  $\Delta$  enjoys the fundamental property of logical relations, namely that it is reflexive. As usual, one proves, by induction on terms, the stronger statement that  $\Delta$  is compatible. The new features in our setting are: the admissibility of  $\sqsubseteq_{basic}$  is used in the case of the fixpoint operator, and the compositionality of  $\sqsubseteq_{basic}$  is needed to prove compatibility for the effect operations.

THEOREM VI.5. *The relation  $\Delta$  is compatible.*

THEOREM VI.6. *The relations  $\Delta$  and  $\sqsubseteq_{ctx}$  coincide.*

## VII. PROPERTIES OF CONTEXTUAL PREORDER

We now present our main operational metatheorems about contextual preorder. Since, by remarks in Sections IV and V,

contextual equivalence arises as the contextual preorder derived from  $=_{basic}$ , which is admissible and compositional whenever  $\sqsubseteq_{basic}$  is, all results apply equally to contextual equivalence. The first result reduces  $\sqsubseteq_{ctx}$  on open terms and types to its restriction to closed terms and types.

**THEOREM VII.1.**  $\alpha_1, \dots, \alpha_m, x_1 :: \tau_1, \dots, x_n :: \tau_n \vdash M \sqsubseteq_{ctx} M' :: \tau$  iff for all  $\sigma_i \in Typ$  ( $i = 1, \dots, m$ ) and all  $N_j \in Term(\tau_j[\vec{\sigma}/\vec{\alpha}])$  ( $j = 1, \dots, n$ ), it is the case that  $M[\vec{\sigma}/\vec{\alpha}, \vec{N}/\vec{x}] \sqsubseteq_{ctx} M'[\vec{\sigma}/\vec{\alpha}, \vec{N}/\vec{x}]$ .

The next result is our “context lemma”. Contextual preorder at closed terms and types can be reduced to the ground type relation  $\sqsubseteq_{basic}$  merely by considering “applicative” stacks. We say that an evaluation stack is *applicative* if it is built from evaluation frames of the form  $(- M)$  and  $-_\tau$  only.

**THEOREM VII.2.** Let  $M, M' \in Term(\tau)$ . Then  $M \sqsubseteq_{ctx} M'$  iff for every applicative stack  $S \in Stack(\tau)$ , it holds that  $|S, M| \sqsubseteq_{basic} |S, M'|$ .

The proof is in two halves. First, it is shown that the result holds if  $S$  ranges over *all* stacks, rather than just applicative ones. This weaker result, which is a “ciu” theorem, in the sense of [1], follows by a standard application of the logical relation  $\Delta$ , cf. [11], [12]. Second, making crucial use of the compositionality of  $\sqsubseteq_{basic}$ , it is shown that applicative stacks are just as discriminative as arbitrary stacks.

Since the only applicative stack in  $Stack(\text{Nat})$  is  $Id$ , it is immediate from Theorem VII.2 that contextual preorder at ground type coincides with  $\sqsubseteq_{basic}$ .

**COROLLARY VII.3.** Let  $M, M' \in Term(\text{Nat})$ . Then  $M \sqsubseteq_{ctx} M'$  iff  $|M| \sqsubseteq_{basic} |M'|$ .

We also have that functions behave extensionally at both type and term levels.

**THEOREM VII.4.**

- a) For every  $G, G' \in Term(\forall\alpha.\tau)$  we have  $G \sqsubseteq_{ctx} G'$  if and only if for all  $\tau' \in Typ$  it holds that  $G_{\tau'} \sqsubseteq_{ctx} G'_{\tau'}$ .
- b) For every  $F, F' \in Term(\tau_1 \rightarrow \tau_2)$  we have  $F \sqsubseteq_{ctx} F'$  if and only if for every  $A \in Term(\tau_1)$  it holds that  $F A \sqsubseteq_{ctx} F' A$ .

The reduction pairs in the table in Section III preserve contextual equivalence.

**THEOREM VII.5.**  $L =_{ctx} R$ , for any reduction-pair  $(L, R)$ .

Using the results above, one can show that the usual  $\beta$ - and  $\eta$ -equalities, for both types and terms, hold up to  $=_{ctx}$ .

The next result shows that effect operations commute with stacks, which is the fundamental property that characterises effect operations as being *algebraic* in the work of Plotkin and Power [7]. We state the commutativity result for just one of the four possible arities for  $\sigma$ .

**THEOREM VII.6.** Let  $\sigma : \text{Nat} \times \alpha^{\text{Nat}} \rightarrow \alpha$ ,  $M \in Term(\text{Nat})$ ,  $M_1 \in Term(\text{Nat} \rightarrow \tau)$ , and  $S \in Stack(\tau)$ . Then

$$S\{\sigma(M; M_1)\} =_{ctx} \sigma(M; \lambda x :: \text{Nat}. S\{M_1 x\})$$

A further useful result is that inequations between effect operations hold at all types if and only if they hold at ground type. For space reasons, we omit this result from this conference paper, since its formulation requires introducing a syntax for algebraic terms over infinitary operations. One consequence of the result is that inequations between effect operations, such as the equational axioms considered by Plotkin and Power, see, e.g., [8], can be read off from their ground type instances, as specified by  $\sqsubseteq_{basic}$ .

To end the section, we make a few remarks about the necessity or otherwise of the assumed conditions on  $\sqsubseteq_{basic}$ . As observed by a reviewer, results that assert specific contextual equivalences, such as Theorems VII.5 and VII.6, hold for an arbitrary preorder  $\sqsubseteq_{basic}$ , even in the absence of admissibility and compositionality. This is because, by the results of this section, they hold if  $\sqsubseteq_{basic}$  is taken to be tree equality, which is obviously admissible and compositional. Since all other preorders are coarser than equality, so are the corresponding contextual equivalences.

In contrast, we believe that, under mild assumptions, the statement of Corollary VII.3 can be shown to be equivalent to the compositionality of  $\sqsubseteq_{basic}$ . In any case, Corollary VII.3 does require *some* conditions to be imposed on  $\sqsubseteq_{basic}$ , hence so does Theorem VII.2.

## VIII. REASONING ABOUT RELATIONAL PARAMETRICITY

A major application of  $\top\top$ -closed logical relations in the literature is to reason about parametricity properties of polymorphism [11], [12], [15], [16], [17]. In this section, we use the logical relation of Section VI to obtain one example of a consequence of parametricity for our language. We show that, in our effectful setting, the type  $\forall\alpha.(\tau \rightarrow \alpha) \rightarrow \alpha$  acts like Moggi’s “monadic” type  $T(\tau)$ . This phenomenon has previously been demonstrated in a denotational setting in [14]. Operationally, in our call-by-name language, it means that this type represents effectful computations which return, as values, suspended terms of type  $\tau$ .

Define  $T(\tau)$  as an abbreviation for  $\forall\alpha.(\tau \rightarrow \alpha) \rightarrow \alpha$ . Given a term  $\Gamma \vdash M :: \tau$ , we define  $[M]$  to be  $\Lambda\alpha. \lambda f :: \tau \rightarrow \alpha. (f M)$ . Note that  $\Gamma \vdash [M] :: T(\tau)$ . Intuitively, the term  $[M]$  is the “think” of type  $T(\tau)$  that suspends the evaluation of the term  $M$ . Also, given terms  $\Gamma \vdash M :: T(\tau)$  and  $\Gamma, x :: \tau \vdash N :: \tau'$ , define  $\text{let } x \leftarrow M \text{ in } N$  to be  $M_{\tau'} (\lambda x :: \tau. N)$ . Clearly  $\Gamma \vdash \text{let } x \leftarrow M \text{ in } N :: \tau'$ . (In the context of our call-by-name language, it is correct to allow an arbitrary type  $\tau'$  here rather than restrict to types of the form  $T(\tau')$ .) Operationally, the term  $[M]$ , for closed  $M$ , is a value. Also, the context  $\text{let } x \leftarrow - \text{ in } N$  is given by the (applicative)

stack  $Id \circ (- (\lambda x :: \tau. N)) \circ -_{\tau'}$ . Thus, the commutativity of effect operations with stacks yields the expected eager sequencing behaviour of Moggi’s “let” operation; for example, for  $\sigma : \alpha^{\text{Nat}} \rightarrow \alpha$ , we have:

$\text{let } x \leftarrow \sigma(M) \text{ in } N =_{ctx} \sigma(\lambda n :: \text{Nat. let } x \leftarrow (Mn) \text{ in } N)$ .

The theorem below states the two fundamental contextual equivalences relating the above derived constructs. Taken together, these properties assert the correct universal property for the type operator  $T(-)$ : it is left adjoint to the forgetful functor from the category of stacks (between closed types) to the category of terms (both up to  $=_{ctx}$ ).

**THEOREM VIII.1.**

- a) Given  $\Gamma \vdash M :: \tau$  and  $\Gamma, x :: \tau \vdash N :: \tau'$ , we have  $\Gamma \vdash \text{let } x \leftarrow [M] \text{ in } N =_{ctx} N[M/x]$ .
- b) Given  $\Gamma \vdash M :: T(\tau)$  and  $\Gamma \vdash S :: T(\tau) \multimap \tau'$ , we have  $\Gamma \vdash \text{let } x \leftarrow M \text{ in } S\{[x]\} =_{ctx} S\{M\}$ .

The proof, which follows the lines of the denotational proof in [14], requires a lemma, which we highlight because it is useful in other applications of relational parametricity: the graphs, up to  $\sqsubseteq_{ctx}$ , of functions given by stacks are  $\top\top$ -closed. (The necessity of working up to  $\sqsubseteq_{ctx}$  is explained by Lemma VI.1.) For every  $S \in \text{Stack}(\tau, \tau')$ , define  $\text{left-graph}_S \in \text{Rel}(\tau, \tau')$  and  $\text{right-graph}_S \in \text{Rel}(\tau', \tau)$  by:

$$\begin{aligned} (M, M') \in \text{left-graph}_S & \text{ iff } S\{M\} \sqsubseteq_{ctx} M' \\ (M', M) \in \text{right-graph}_S & \text{ iff } M' \sqsubseteq_{ctx} S\{M\}. \end{aligned}$$

**LEMMA VIII.2.** Let  $\tau, \tau' \in \text{Typ}$  and  $S \in \text{Stack}(\tau, \tau')$ . Then  $\text{left-graph}_S$  and  $\text{right-graph}_S$  are  $\top\top$ -closed.

Finally, again following [14], we give a simple description of the logical relation  $\Delta$  at types  $T(\tau)$ . This indicates how Definition VI.2 can be extended in the case of languages with a primitive  $T(-)$  type constructor.

$$\Delta_{T(\tau)}(\vec{r}/\vec{\alpha}) = \{([M], [M']) \mid (M, M') \in \Delta_{\tau}(\vec{r}/\vec{\alpha})\}^{\top\top}$$

## IX. RELATED AND FUTURE WORK

The idea of providing a uniform operational metatheory for a family of language features has been popular in concurrency theory, where, e.g., general operational rule formats have been studied (see [18] for a survey), and principled methods of deriving behavioural equivalence from reduction rules have been identified [19], [20]. For higher-order languages, possibly involving concurrency, state, and polymorphism, the theory of *environmental bisimulations*, has emerged as a general tool for analysing contextual equivalence [21], [22]. However, the empirical robustness of this theory has not yet been explained via any uniform operational metatheory. Indeed, as far as we know, our work is the first to attempt a systematic operational metatheory that applies to the setting of functional languages with effects. To this end, we have combined two previously distinct research

areas: Plotkin and Power’s theory of algebraic effects [6], [7], [8], [9], and Pitts and Stark’s logical-relation-based approach to operational reasoning [2], [11], [12]. By adapting the latter to a range of effects, we provide further evidence of its versatility, see also [15], [16], [17]. Our results have been obtained for a call-by-name language. For call-by-value, the statements of the metatheorems need adjusting. For example, the context lemma does not hold as formulated in Theorem VII.2, since  $\Omega_{\text{Nat} \rightarrow \text{Nat}}$  and  $\lambda x :: \text{Nat. } \Omega_{\text{Nat}}$  are not call-by-value contextually equivalent. Nevertheless, we believe our methods will adapt straightforwardly to establish the correct metatheorems in a call-by-value setting, simply by adjusting the definition of the logical relation in the appropriate way, cf. [12].

Plotkin and Power originally considered computation trees in tandem with a denotational semantics given via a computational monad  $T$  on  $\omega\text{cpos}$  [7]. Plotkin [6] uses such a denotational semantics to define contextual preorder for a (call-by-value) language with effects. His definition does not adapt directly to our polymorphic language, because it is not known how to model the combination of polymorphism and effects in  $\omega\text{cpos}$ . Nevertheless, one can instead use the monad  $T$  to give a denotational definition of a  $\sqsubseteq_{basic}$  relation on computation trees, and thereby derive a contextual preorder as in Section IV. It can be shown that any such denotationally determined  $\sqsubseteq_{basic}$  relation is automatically admissible and compositional. Thus all our operational metatheorems apply. Plotkin raises the question of whether context-lemma-like results are available for such denotationally determined contextual preorders over his (monomorphic) call-by-value language [6]. Our Theorem VII.2 answers the analogous question affirmatively, for our (polymorphic) call-by-name language.

Our existing theory has certain limitations in its applicability. Indeed, our running examples pretty much cover the range of effects catered for, except for one aspect: we can also deal with combinations of effects. A combination of instances of our framework can itself be considered an instance by amalgamating the signatures of effect operations, and then specifying a suitable admissible and compositional  $\sqsubseteq_{basic}$  relation on the resulting trees. While this is easily done in individual cases, a more compelling story would be to show that the “sum” and “tensor” operations, in [23], for combining Lawvere theories have analogues in our setting for combining  $\sqsubseteq_{basic}$  relations, and that these operations correctly account for the principal examples of combinations of algebraic effects. A further interesting development would show that a similar methodological approach to combination is available via constructions on the families of observations of Section V. This might be possible by viewing observations as being generated by modal operators, and by considering methods of combining modal logics. It is also plausible that viewing observations as modalities might lead to a uniform method of deriving logical characterizations of

contextual equivalence.

Our current theory does not cover all effects that have been described as “algebraic” in the literature. For example, the admissibility requirement on  $\sqsubseteq_{basic}$  rules out countable nondeterminism [3]. Also, local state [8] is not incorporated because it requires a type of scopable locations to be used as a parameter and arity in effect operations. While our permitted arities could potentially be generalised beyond finite and Nat, cf. [6], the inclusion of more general arities would raise tricky issues about how to index the branching in computation trees, and how to ensure the compatibility of  $\sqsubseteq_{basic}$  with respect to such indices. While such issues should be solvable in the case of ground-type local store, they are likely to present significant problems for more complex effects such as higher-order store.

A promising direction for further research is to extend our theory to allow effect operations that are blatantly non-algebraic, such as exception handlers and other control primitives. These can be added to our operational semantics by specifying new evaluation frames and reduction rules for them, but leaving the computation trees and the crucial  $\sqsubseteq_{basic}$  relations unchanged. Thus we envisage a general theory in which effect operations are separated into two groups: the algebraic operations, which are the building blocks of the computation tree, and which give rise to the “observables” of behaviour; and the control primitives, which contribute to the reduction rules. While the new reduction rules for control will invalidate our operational metatheorems in their present form, we believe that the methodology of using a logical relation to analyse the contextual preorder derived from a  $\sqsubseteq_{basic}$  relation on computation trees will still be applicable to obtain the correct metatheorems at this generality.

#### ACKNOWLEDGMENTS

We thank Paul Levy, Gordon Plotkin and the anonymous reviewers for helpful suggestions.

#### REFERENCES

- [1] I. Mason and C. Talcott, “Equivalence in functional languages with effects,” *Journal of Functional Programming*, vol. 1, pp. 287–327, 1991.
- [2] A. Pitts and I. Stark, “Operational reasoning for functions with local state,” in *Higher Order Operational Techniques in Semantics, Proceedings*, 1998, pp. 227–273.
- [3] S. Lassen, “Relational reasoning about functions and nondeterminism,” Ph.D. dissertation, University of Aarhus, 1998.
- [4] G. Plotkin, “LCF considered as a programming language,” *Theoretical Computer Science*, vol. 5, pp. 223–255, 1977.
- [5] —, “A structural approach to operational semantics,” *Journal of Logic and Algebraic Programming*, vol. 60–61, pp. 17–139, 2004.
- [6] —, “Adequacy for infinitary algebraic effects,” in *Conference on Algebra and Coalgebra in Computer Science, Proceedings*, 2009, pp. 1–2.
- [7] G. Plotkin and J. Power, “Adequacy for algebraic effects,” in *Foundations of Software Science and Computation Structures, Proceedings*, 2001, pp. 1–24.
- [8] —, “Notions of computation determine monads,” in *Foundations of Software Science and Computation Structures, Proceedings*, 2002, pp. 342–356.
- [9] —, “Algebraic operations and generic effects,” *Applied Categorical Structures*, vol. 11, pp. 69–94, 2003.
- [10] E. Moggi, “Computational lambda-calculus and monads,” in *Logic in Computer Science, Proceedings*, 1989, pp. 14–23.
- [11] A. Pitts, “Parametric polymorphism and operational equivalence,” *Mathematical Structures in Computer Science*, vol. 10, pp. 321–359, 2000.
- [12] —, “Typed operational reasoning,” in *Advanced Topics in Types and Programming Languages*, B. Pierce, Ed. MIT Press, 2005, pp. 245–289.
- [13] R. Milner, “Fully abstract models of typed  $\lambda$ -calculi,” *Theoretical Computer Science*, vol. 4, pp. 1–22, 1977.
- [14] R. Møgelberg and A. Simpson, “Relational parametricity for computational effects,” in *Logic in Computer Science, Proceedings*, 2007, pp. 346–355.
- [15] P. Johann, “Short cut fusion is correct,” *Journal of Functional Programming*, vol. 13, pp. 797–814, 2003.
- [16] P. Johann and J. Voigtländer, “A family of syntactic logical relations for the semantics of Haskell-like languages,” *Information and Computation*, vol. 207, pp. 341–368, 2009.
- [17] J. Voigtländer and P. Johann, “Selective strictness and parametricity in structural operational semantics, inequationally,” *Theoretical Computer Science*, vol. 388, pp. 290–318, 2007.
- [18] L. Aceto, W. Fokkink, and C. Verhoef, “Structural operational semantics,” in *Handbook of Process Algebra*. Elsevier, 2001, pp. 197–292.
- [19] J. Leifer and R. Milner, “Deriving bisimulation congruences for reactive systems,” in *Conference on Concurrency Theory, Proceedings*, 2000, pp. 243–258.
- [20] P. Sewell, “From rewrite rules to bisimulation congruences,” *Theoretical Computer Science*, vol. 274, pp. 183–230, 2002.
- [21] D. Sangiorgi, N. Kobayashi, and E. Sumii, “Environmental bisimulations for higher-order languages,” in *Logic in Computer Science, Proceedings*, 2007, pp. 293–302.
- [22] E. Sumii, “A complete characterization of observational equivalence in polymorphic lambda-calculus with general references,” in *Computer Science Logic, Proceedings*, 2009, pp. 455–469.
- [23] M. Hyland, G. Plotkin, and J. Power, “Combining effects: Sum and tensor,” *Theoretical Computer Science*, vol. 357, pp. 70–99, 2006.