

1  
2  
3  
4  
5  
6  
7  
8

# PROOF-RELEVANT LOGICAL RELATIONS FOR NAME GENERATION

NICK BENTON, MARTIN HOFMANN, AND VIVEK NIGAM

Microsoft Research, Cambridge, UK  
*e-mail address:* nick@microsoft.com

LMU, Munich, Germany  
*e-mail address:* hofmann@ifi.lmu.de

UFPB, João Pessoa, Brazil  
*e-mail address:* vivek.nigam@gmail.com

---

ABSTRACT. Pitts and Stark’s  $\nu$ -calculus is a paradigmatic total language for studying the problem of contextual equivalence in higher-order languages with name generation. Models for the  $\nu$ -calculus that validate basic equivalences concerning names may be constructed using functor categories or nominal sets, with a dynamic allocation monad used to model computations that may allocate fresh names. If recursion is added to the language and one attempts to adapt the models from (nominal) sets to (nominal) domains, however, the direct-style construction of the allocation monad no longer works. This issue has previously been addressed by using a monad that combines dynamic allocation with continuations, at some cost to abstraction.

This paper presents a direct-style model of a  $\nu$ -calculus-like language with recursion using the novel framework of *proof-relevant logical relations*, in which logical relations also contain objects (or proofs) demonstrating the equivalence of (the semantic counterparts of) programs. Apart from providing a fresh solution to an old problem, this work provides an accessible setting in which to introduce the use of proof-relevant logical relations, free of the additional complexities associated with their use for more sophisticated languages.

9  
10  
11  
12  
13  
14  
15  
16  
17  
18

## INTRODUCTION

Reasoning about contextual equivalence in higher-order languages that feature dynamic allocation of names, references, objects or keys is challenging. Pitts and Stark’s  $\nu$ -calculus boils the problem down to its purest form, being a total, simply-typed lambda calculus with just names and booleans as base types, an operation `new` that generates fresh names, and equality testing on names. The full equational theory of the  $\nu$ -calculus is surprisingly complex and has been studied both operationally and denotationally, using logical relations [Sta94, PS98], environmental bisimulations [BK13] and nominal game semantics [AGM<sup>+</sup>04, Tze12].

Even before one considers the ‘exotic’ equivalences that arise from the (partial) encapsulation of names within closures, there are two basic equivalences that hold for essentially all forms of

---

*1998 ACM Subject Classification:* D.3.3: Programming Languages; F.3.2: Logic and Meanings of Programs.

*Key words and phrases:* logical relations, parametricity, program transformation.

A preliminary version of this work was presented at the 11th International Conference on Typed Lambda Calculi and Applications, TLCA 2013, 26-28 June 2013, Eindhoven, The Netherlands.

19 generativity:

$$(\text{let } x \leftarrow \text{new in } e) = e, \text{ provided } x \text{ is not free in } e. \quad (\text{Drop})$$

$$(\text{let } x \leftarrow \text{new in let } y \leftarrow \text{new in } e) = (\text{let } y \leftarrow \text{new in let } x \leftarrow \text{new in } e) \quad (\text{Swap}).$$

20 The (Drop) equivalence says that removing the generation of unused names preserves behaviour;  
 21 this is sometimes called the ‘garbage collection’ rule. The (Swap) equivalence says that the order in  
 22 which names are generated is immaterial. These two equations also appear as structural congruences  
 23 for name restriction in the  $\pi$ -calculus.

24 Denotational models for the  $\nu$ -calculus validating (Drop) and (Swap) may be constructed using  
 25 (pullback-preserving) functors in  $\text{Set}^{\mathbf{W}}$ , where  $\mathbf{W}$  is the category of finite sets and injections  
 26 [Sta94], or in FM-sets [GP02]. These models use a dynamic allocation monad to interpret possibly-  
 27 allocating computations. One might expect that moving to  $\text{Cpo}^{\mathbf{W}}$  or FM-cpos would allow such  
 28 models to adapt straightforwardly to a language with recursion, and indeed Shinwell, Pitts and Gab-  
 29 bay originally proposed [SPG03] a dynamic allocation monad over FM-cpos. However, it turned  
 30 out that the underlying FM-cppo of the proposed monad does not actually have least upper bounds  
 31 for all finitely-supported chains. A counter-example is given in Shinwell’s thesis [Shi04, page 86].  
 32 To avoid the problem, Shinwell and Pitts [Shi04, SP05] moved to an *indirect-style* model, using a  
 33 *continuation monad* [PS98]:  $(-)^{\top\top} \stackrel{\text{def}}{=} (- \rightarrow 1_{\perp}) \rightarrow 1_{\perp}$  to interpret computations. In particular, one  
 34 shows that two programs are equivalent by proving that they co-terminate when supplied with the  
 35 same (or equivalent) continuations. The CPS approach was also adopted by Benton and Leperchey  
 36 [BL05], and by Bohr and Birkedal [BB06], for modelling languages with references.

37 In the context of our on-going research on the semantics of effect-based program transforma-  
 38 tions [BKHB06], we have been led to develop *proof-relevant* logical relations [BHN14]. These  
 39 interpret types not merely as partial equivalence relations, as is commonly done, but as a proof-  
 40 relevant generalization thereof: *setoids*. A setoid is like a category all of whose morphisms are  
 41 isomorphisms (a groupoid) with the difference that no equations between these morphisms are im-  
 42 posed. The objects of a setoid establish that values inhabit semantic types, whilst its morphisms are  
 43 understood as explicit proofs of semantic equivalence. This paper shows how we can use proof-  
 44 relevant logical relations to give a direct-style model of a language with name generation and re-  
 45 cursion, validating (Drop) and (Swap). Apart from providing a fresh approach to an old problem,  
 46 our aim in doing this is to provide a comparatively accessible presentation of proof-relevant logi-  
 47 cal relations in a simple setting, free of the extra complexities associated with specialising them to  
 48 abstract regions and effects [BHN14].

49 Although our model validates the two most basic equations for name generation, it is – like sim-  
 50 ple functor categories in the total case – still far from fully abstract. Many of the subtler contextual  
 51 equivalences of the  $\nu$ -calculus still hold in the presence of recursion; one naturally wonders whether  
 52 the more sophisticated methods used to prove those equivalences carry over to the proof-relevant  
 53 setting. We will show one such method, Stark’s *parametric functors*, which are a categorical version  
 54 of Kripke logical relations, does indeed generalize smoothly, and can be used to establish a non-  
 55 trivial equivalence involving encapsulation of fresh names. Moreover, the proof-relevant version is  
 56 naturally transitive, which is, somewhat notoriously, not generally true of ordinary logical relations.

57 Section 1 sketches the language with which we will be working, and a naive ‘raw’ domain-  
 58 theoretic semantics for it. This semantics does not validate interesting equivalences, but is adequate.  
 59 By constructing a realizability relation between it and the more abstract semantics we subsequently  
 60 introduce, we will be able to show adequacy of the more abstract semantics. In Section 2 we intro-  
 61 duce our category of setoids; these are predomains where there is a (possibly-empty) set of ‘proofs’  
 62 witnessing the equality of each pair of elements. We then describe pullback-preserving functors

$$\begin{array}{c}
\frac{}{\Gamma, x : \tau \vdash_v x : \tau} \quad \frac{}{\Gamma \vdash_v b : \mathbf{bool}} \quad \frac{}{\Gamma \vdash_v i : \mathbf{int}} \quad \frac{\Gamma, f : \tau \rightarrow \tau', x : \tau \vdash_c e : \tau'}{\Gamma \vdash_v \mathbf{rec } f x = e : \tau \rightarrow \tau'} \\
\frac{\Gamma \vdash_v v : \mathbf{int} \quad \Gamma \vdash_v v' : \mathbf{int}}{\Gamma \vdash_v v + v' : \mathbf{int}} \quad \frac{\Gamma \vdash_v v : \tau \quad \Gamma \vdash_v v' : \tau \quad \tau \in \{\mathbf{int}, \mathbf{name}\}}{\Gamma \vdash_v v = v' : \mathbf{bool}} \quad \frac{\Gamma \vdash_v v : \tau}{\Gamma \vdash_c v : \tau} \\
\frac{}{\Gamma \vdash_c \mathbf{new} : \mathbf{name}} \quad \frac{\Gamma \vdash_c e : \tau \quad \Gamma, x : \tau \vdash_c e' : \tau'}{\Gamma \vdash_c \mathbf{let } x \leftarrow e \mathbf{ in } e' : \tau'} \quad \frac{\Gamma \vdash_v v : \tau \rightarrow \tau' \quad \Gamma \vdash_v v' : \tau}{\Gamma \vdash_c v v' : \tau'} \\
\frac{\Gamma \vdash_v v : \mathbf{bool} \quad \Gamma \vdash_c e : \tau \quad \Gamma \vdash_c e' : \tau}{\Gamma \vdash_c \mathbf{if } v \mathbf{ then } e \mathbf{ else } e' : \tau}
\end{array}$$

Figure 1: Typing rules for language with recursion and name generation

63 from the category of worlds  $\mathbf{W}$  into the category of setoids. Such functors will interpret types of  
64 our language in the more abstract semantics, with morphisms between them interpreting terms. The  
65 interesting construction here is that of a dynamic allocation monad over the category of pullback-  
66 preserving functors. Section 7 shows how the abstract semantics is defined and related to the more  
67 concrete one. Section 8 then shows how the semantics may be used to establish basic equivalences  
68 involving name generation. Section 9 describes how proof-relevant parametric functors can validate  
69 a more subtle equivalence involving encapsulation of new names.

70

## 1. SYNTAX AND SEMANTICS

71 We work with an entirely conventional CBV language, featuring recursive functions and base types  
72 that include names, equipped with equality testing and fresh name generation (here  $+$  is just a  
73 representative operation on integers):

$$\begin{aligned}
\tau &:= \mathbf{int} \mid \mathbf{bool} \mid \mathbf{name} \mid \tau \rightarrow \tau' \\
v &:= x \mid b \mid i \mid \mathbf{rec } f x = e \mid v + v' \mid v = v' \\
e &:= v \mid \mathbf{new} \mid \mathbf{let } x \leftarrow e \mathbf{ in } e' \mid v v' \mid \mathbf{if } v \mathbf{ then } e \mathbf{ else } e' \\
\Gamma &:= x_1 : \tau_1, \dots, x_n : \tau_n
\end{aligned}$$

74 The expression  $\mathbf{rec } f x = e$  stands for an anonymous function which satisfies the recursive equation  
75  $f(x) = e$  where typically, both  $x$  and  $f$  will occur in  $e$ . In the special case where  $f$  does not occur in  
76  $e$  the construct degenerates to function abstraction. We thus introduce the abbreviation:

$$\mathbf{fun } x.e \triangleq \mathbf{rec } f x = e \quad \text{where } f \text{ does not occur in } e.$$

77 There are typing judgements for values,  $\Gamma \vdash_v v : \tau$ , and computations,  $\Gamma \vdash_c e : \tau$ , defined in an  
78 unsurprising way; these are shown in Figure 1. We will often elide the subscript on turnstiles.

79 We define a simple-minded concrete denotational semantics  $\llbracket \cdot \rrbracket$  for this language using predo-  
80 mains ( $\omega$ -cpos) and continuous maps. For types we take

$$\begin{aligned}
\llbracket \mathbf{int} \rrbracket &= \mathbb{Z} & \llbracket \mathbf{bool} \rrbracket &= \mathbb{B} & \llbracket \mathbf{name} \rrbracket &= \mathbb{N} \\
\llbracket \tau \rightarrow \tau' \rrbracket &= \llbracket \tau \rrbracket \rightarrow (\mathbb{N} \rightarrow \mathbb{N} \times \llbracket \tau' \rrbracket)_\perp \\
\llbracket x_1 : \tau_1, \dots, x_n : \tau_n \rrbracket &= \llbracket \tau_1 \rrbracket \times \dots \times \llbracket \tau_n \rrbracket
\end{aligned}$$

81 and there are then conventional clauses defining

$$\llbracket \Gamma \vdash_v v : \tau \rrbracket : \llbracket \Gamma \rrbracket \rightarrow \llbracket \tau \rrbracket \quad \text{and} \quad \llbracket \Gamma \vdash_c e : \tau \rrbracket : \llbracket \Gamma \rrbracket \rightarrow (\mathbb{N} \rightarrow \mathbb{N} \times \llbracket \tau \rrbracket)_\perp$$

82 Note that this semantics just uses naturals to interpret names, and a state monad over names to  
83 interpret possibly-allocating computations. For allocation we take

$$\llbracket \Gamma \vdash_c \text{new} : \text{name} \rrbracket(\eta) = [\lambda n.(n + 1, n)]$$

84 returning the next free name and incrementing the name supply. This semantics validates no inter-  
85 esting equivalences involving names, but is adequate for the obvious operational semantics. Our  
86 more abstract semantics,  $\llbracket \cdot \rrbracket$ , will be related to  $\llbracket \cdot \rrbracket$  in order to establish *its* adequacy.

87

## 2. SETOIDS

88 We define the *category of setoids*,  $Std$ , as the exact completion of the category of predomains, see  
89 [CFS87, BCRS98]. We give here an elementary description of this category using the language of  
90 dependent types. A *setoid*  $A$  consists of a predomain  $|A|$  and for any two  $x, y \in |A|$  a set  $A(x, y)$  of  
91 “proofs” (that  $x$  and  $y$  are equal). The set of triples  $X = \{(x, y, p) \mid p \in A(x, y)\}$  must itself be a  
92 predomain, *i.e.*, there has to be an order relation  $\leq$  such that  $(X, \leq)$  is a predomain. The first and  
93 second projections out of the set of triples must be continuous. Furthermore, there are continuous  
94 functions  $r_A : \prod x \in |A|.A(x, x)$  and  $s_A : \prod x, y \in |A|.A(x, y) \rightarrow A(y, x)$  and  $t_A : \prod x, y, z.A(x, y) \times$   
95  $A(y, z) \rightarrow A(x, z)$ , witnessing reflexivity, symmetry and transitivity; note that, unlike the case of  
96 *groupoids*, no equations involving  $r$ ,  $s$  and  $t$  are imposed.

97 We should explain what continuity of a dependent function like  $t(-, -)$  is: if  $(x_i)_i$  and  $(y_i)_i$   
98 and  $(z_i)_i$  are ascending chains in  $A$  with suprema  $x, y, z$  and  $p_i \in A(x_i, y_i)$  and  $q_i \in A(y_i, z_i)$  are  
99 proofs such that  $(x_i, y_i, p_i)_i$  and  $(y_i, z_i, q_i)_i$  are ascending chains, too, with suprema  $(x, y, p)$  and  
100  $(y, z, q)$  then  $(x_i, z_i, t(p_i, q_i))_i$  is an ascending chain of proofs (by monotonicity of  $t(-, -)$ ) and its  
101 supremum is  $(x, z, t(p, q))$ . Formally, such dependent functions can be reduced to non-dependent  
102 ones using pullbacks, that is  $t$  would be a function defined on the pullback of the second and first  
103 projections from  $\{(x, y, p) \mid p \in A(x, y)\}$  to  $|A|$ , but we find the dependent notation to be much  
104 more readable. If  $p \in A(x, y)$  we may write  $p : x \sim y$  or simply  $x \sim y$ . We also omit  $| - |$   
105 wherever appropriate. We remark that “setoids” also appear in constructive mathematics and formal  
106 proof, see *e.g.*, [BCP03], but the proof-relevant nature of equality proofs is not exploited there and  
107 everything is based on sets (types) rather than predomains. A morphism from setoid  $A$  to setoid  
108  $B$  is an equivalence class of pairs  $f = (f_0, f_1)$  of continuous functions where  $f_0 : |A| \rightarrow |B|$  and  
109  $f_1 : \prod x, y \in |A|.A(x, y) \rightarrow B(f_0(x), f_0(y))$ . Two such pairs  $f, g : A \rightarrow B$  are *identified* if there exists a  
110 continuous function  $\mu : \prod a \in |A|.B(f_0(a), g_0(a))$ .

111 The following is folklore, see also [BCRS98].

112 **Proposition 2.1.** The category of setoids is cartesian-closed. Cartesian product is given pointwise.  
113 The function space  $A \Rightarrow B$  of setoids  $A$  and  $B$  is given as follows: the underlying predomain  $|A \Rightarrow B|$   
114 comprises pairs  $(f_0, f_1)$  which are *representatives* of morphisms from  $A$  to  $B$ . That is,  $f_0 : |A| \rightarrow |B|$   
115 and  $f_1 : \prod x, y \in |A|.A(x, y) \rightarrow B(f_0(x), f_0(y))$  are continuous functions with the pointwise ordering.  
116 The proof set  $(A \Rightarrow B)((f_0, f_1), (f'_0, f'_1))$  comprises witnesses of the equality of  $(f_0, f_1)$  and  $(f'_0, f'_1)$   
117 qua morphisms, *i.e.*, continuous functions  $\mu : \prod a \in |A|.B(f_0(a), g_0(a))$ .

118 *Proof.* The evaluation morphism  $(A \Rightarrow B) \times A \rightarrow B$  sends  $(f_0, f_1)$  and  $a$  to  $f_0(a)$ . If  $h : C \times$   
119  $A \rightarrow B$  is a morphism represented by  $(h_0, h_1)$  then the morphism  $\lambda(h) : C \rightarrow A \Rightarrow B$  may  
120 be represented by  $(\lambda(h)_0, \lambda(h)_1)$  where  $\lambda(h)_0(c) = (f_0, f_1)$  and  $f_0(a) = h_0(c, a)$  and  $f_1(a, a', p) =$

121  $h_1((c, a), (c, a'), (r(c), p))$ . Likewise,  $\lambda(h)_1(c, c', p) = \mu$  where  $\mu(a) = h_1((c, a), (c', a), (p, r(a)))$ . The  
 122 remaining verifications are left to the reader.  $\square$

123 **Definition 2.2.** A setoid  $D$  is *pointed* if  $|D|$  has a least element  $\perp$  and such that there is also a least  
 124 proof  $\perp \in D(\perp, \perp)$ .

125 If  $D$  is pointed we write  $\perp$  for the obvious global element  $1 \rightarrow D$  returning  $\perp$ . A morphism  
 126  $f : D \rightarrow D'$  with  $D, D'$  both pointed is *strict* if  $f\perp = \perp$ .

127 **Theorem 2.3.** Let  $D$  be a setoid such that  $|D|$  has a least element  $\perp$  and such that there is also a least  
 128 proof  $\perp \in D(\perp, \perp)$ . Then there is a morphism of setoids  $Y : [D \Rightarrow D] \rightarrow D$  satisfying the following  
 129 equations (written using  $\lambda$ -calculus notation, which is meaningful in cartesian-closed categories).

$$\begin{aligned}
 f(Y(f)) &= Y(f) && \text{(Fixpoint)} \\
 f(Y(g \circ f)) &= Y(f \circ g) && \text{(Dinaturality)} \\
 f(Y(g)) &= Y(h) \text{ if } f \text{ is strict and } fg = hf && \text{(Uniformity)} \\
 Y(f^n) &= Y(f) && \text{(Power)} \\
 Y(\lambda x.f(x, x)) &= Y(\lambda x.Y(\lambda y.f(x, y))) && \text{(Diagonal)} \\
 Y(\lambda \vec{x}. \vec{t}(\vec{x})) &= \langle Y(s), \dots, Y(s) \rangle && \text{(Amalgamation)} \\
 &\text{when } t_i(y, \dots, y) = s(y) \text{ for } i = 1, \dots, n \text{ and } \vec{t} = \langle t_1, \dots, t_n \rangle
 \end{aligned}$$

130 *Proof.* To define the morphism  $Y$  suppose we are given  $f = (f_0, f_1) \in |D \Rightarrow D|$ . For each  $i \in \mathbb{N}$  we  
 131 define  $d_i \in |D|$  by  $d_0 = \perp$  and  $d_{i+1} = f_0(d_i)$ . We then put  $Y(f) = \sup_i d_i$ .

132 Now suppose that  $f' = (f'_0, f'_1) \in |D \Rightarrow D|$  and  $q : f \sim f'$ , i.e.,  $q : \Pi d.D(f_0(d), f'_0(d))$ .  
 133 Let  $d'_i$  be defined analogously to  $d_i$  so that  $Y(f') = \sup_i d'_i$ . By induction on  $i$  we define proofs  
 134  $p_i : d_i \sim d'_i$ . We put  $p_0 = \perp$  (the least proof) and, inductively,  $p_{i+1} = t(f_1(p_i), q(d'_i))$  (transitivity).  
 135 Notice that  $f_1(p_i) : d_{i+1} \sim f_0(d'_i)$  and  $q(d'_i) : f_0(d'_i) \sim d'_{i+1}$ . Now let  $(d, d', p)$  be the supremum of  
 136 the chain  $(d_i, d'_i, p_i)$ . By continuity of the projections we have that  $d = Y(f)$  and  $d' = Y(f')$  and  
 137 thus  $p : Y(f) \sim Y(f')$ . The passage from  $q$  to  $p$  witnesses that  $Y$  is indeed a (representative of a)  
 138 morphism.

139 Equations ‘‘Diagonal’’ and ‘‘Dinaturality’’ follow directly from the validity of these properties  
 140 for the least fixpoint combinator for cpos. For the sake of completeness we prove the second one.  
 141 Assume  $f, g \in |D \Rightarrow D|$  and let  $d_i = (f_0 g_0)^i(\perp)$  and  $e_i = (g_0 f_0)^i(\perp)$ . We have  $d_i \leq f_0(e_i)$  and  
 142  $f_0(e_i) \leq d_{i+1}$ . It follows that  $Y(fg)$  and  $Y(gf)$  are actually equal. Equation ‘‘Fixpoint’’ is a direct  
 143 consequence of dinaturality (take  $g = \text{id}$ ).

144 Amalgamation and uniformity are also valid for the least fixpoint combinator, but cannot be  
 145 directly inherited since the equational premises only holds up to  $\sim$ . As a representative example  
 146 we show amalgamation. So assume elements  $t_i \in |D^n \Rightarrow D|$  and  $s \in |D \Rightarrow D|$  and proofs  $p_k :$   
 147  $\Pi d.D((t_k)_0(d, \dots, d), s(d))$ . Consider  $d_i = \vec{t}_0^i(\perp, \dots, \perp)$  and  $e_i = s_0^i(\perp)$ . By induction on  $i$  and using  
 148 the  $p_k$  we construct proofs  $d_i \sim (e_i, \dots, e_i)$ . The desired proof of  $Y(\vec{t}) \sim (Y(s), \dots, Y(s))$  is obtained  
 149 as the supremum of these proofs as in the definition of the witness that  $Y$  is a morphism above.

150 Equation ‘‘Power’’, finally, can be deduced from amalgamation and dinaturality or alternatively  
 151 inherited directly from the least fixpoint combinator.  $\square$

152 The above equational axioms for the fixpoint combinator are taken from [SP00] where they are  
 153 shown to imply certain completeness properties; in particular, it follows that the category of setoids  
 154 is an ‘‘iteration theory’’ in the sense of Bloom and Esik [BÉ93]. For us they are important since  
 155 the category of setoids is not cpo-enriched in any reasonable way, so that the usual order-theoretic  
 156 characterisation of  $Y$  is not available. Concretely, the equations help for example to justify various  
 157 loop optimisations when loops are expressed using the fixpoint combinator.

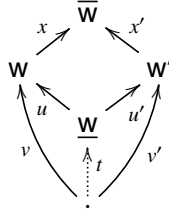
158 **Definition 2.4.** A setoid  $D$  is *discrete* if for all  $x, y \in D$  we have  $|D(x, y)| \leq 1$  and  $|D(x, y)| = 1 \iff$   
 159  $x = y$ .

160 Thus, in a discrete setoid proof-relevant equality and actual equality coincide and moreover any  
 161 two equality proofs are actually equal (proof irrelevance).

162

### 3. FINITE SETS AND INJECTIONS

163 *Pullback squares* are a central notion in our framework. As it will become clear later, they are  
 164 the “proof-relevant” component of logical relations. Recall that a morphism  $u$  in a category is a  
 165 monomorphism if  $ux = ux'$  implies  $x = x'$  for all morphisms  $x, x'$ . Two morphisms with common  
 166 co-domain are called a co-span and two morphisms with common domain are called span. A com-  
 167 muting square  $xu = x'u'$  of morphisms is a *pullback* if whenever  $xv = x'v'$  there is unique  $t$  such  
 168 that  $v = ut$  and  $v' = u't$ . This can be visualized as follows:



169 We write  $x \diamond_u^{x'} w'$  or  $w_u^x \diamond_{u'}^{x'} w'$  (when  $w^{(')} = \text{dom}(x^{(')})$ ) for such a pullback square. We call the common  
 170 codomain of  $x$  and  $x'$  the *apex* of the pullback, written  $\bar{w}$ , while the common domain of  $u, u'$  is the  
 171 *low point* of the square, written  $\underline{w}$ . A pullback square  $w_u^x \diamond_{u'}^{x'} w'$  with apex  $\bar{w}$  is *minimal* if whenever  
 172 there is another pullback  $w_{u_1}^{x_1} \diamond_{u'_1}^{x'_1} w'$  over the same span and with apex  $\bar{w}_1$ , then there is a unique  
 173 morphism  $t : \bar{w} \rightarrow \bar{w}_1$  such that  $x_1 = tx$  and  $x'_1 = tx'$ .

174 A category has pullbacks if every co-span can be completed to a pullback, which is necessarily  
 175 unique up to isomorphism.

176 **Definition 3.1.** A category of worlds,  $\mathcal{C}$ , is a category with pullbacks where any span  $u : \underline{w} \rightarrow$   
 177  $w, u' : \underline{w} \rightarrow w'$  can be completed to a minimal pullback square. Furthermore, there is a subcategory  
 178  $\mathcal{I}$  of  $\mathcal{C}$  full on objects which is a poset, *i.e.*,  $|\mathcal{I}(X, Y)| \leq 1$ . The morphisms in  $\mathcal{I}$  are called inclusions.  
 179 Moreover, any morphism  $u$  in  $\mathcal{C}$  can be factored as  $i_1; u_1$  and as  $u_2; i_2$  where  $i_1, i_2$  are inclusions and  
 180  $u_1, u_2$  are isomorphisms.

181 **Proposition 3.2.** In a category of worlds all morphisms are monomorphisms and if  $w_u^x \diamond_{u'}^{x'} w'$  with  
 182 apex  $\bar{w}$  is a minimal pullback then the morphisms  $x$  and  $x'$  are *jointly epic*, *i.e.* for any  $f, g : \bar{w} \rightarrow w_1$ ,  
 183 if  $fx = gx$  and  $fx' = gx'$ , then  $f = g$ .

184 *Proof.* First we show that any morphism  $u : w \rightarrow w'$  is a monomorphism. Let  $w_u^x \diamond_{u'}^{x'} w'$  be a  
 185 completion of the span  $u, u$  to a (minimal) pullback. If  $ua = ub =: h$ , then  $xh = x'h$ . So, the pullback  
 186 property furnishes a unique map  $c$  such that  $uc = h$ . Thus  $c = a = b$ , so  $u$  is a monomorphism.

187 Now suppose that  $w_u^x \diamond_{u'}^{x'} w'$  is a minimal pullback and  $fx = gx =: h$  and  $fx' = gx' =: h'$ . Then  
 188 we claim that  $w_u^h \diamond_{u'}^{h'} w'$  is a pullback: if  $ht = h't'$ , then since  $f, g$  are monomorphisms by the above,  
 189 we have  $xt = x't'$ , so we can appeal to the pullback property of the original square.

190 Minimality of  $w_u^x \diamond_{u'}^{x'} w'$  furnishes a unique map  $k$  such that  $h = kx$  and  $h' = kx'$ . But since  $f$   
 191 and  $g$  also have that property ( $h = fx$  and  $h' = fx'$  and similarly for  $g$ ), we conclude  $f = g = k$ .  $\square$

192 **Proposition 3.3.** The category  $\mathbf{W}$  with finite sets of natural numbers as objects and injective func-  
 193 tions for morphisms and inclusions for the subcategory of inclusion ( $\mathcal{I}$ ) is a category of worlds.

194 *Proof.* Given  $f : X \rightarrow Z$  and  $g : Y \rightarrow Z$  forming a co-span in  $\mathbf{W}$ , we form their pullback as  
 195  $X \xleftarrow{f^{-1}} fX \cap gY \xrightarrow{g^{-1}} Y$ . This is minimal when  $fX \cup gY = Z$ . Conversely, given a span  $Y \xleftarrow{f} X \xrightarrow{g} Z$ ,  
 196 we can complete to a minimal pullback by

$$(Y \setminus fX) \uplus fX \xrightarrow{[in_1, in_3 \circ f^{-1}]} (Y \setminus fX) + (Z \setminus gX) + X \xleftarrow{[in_2, in_3 \circ g^{-1}]} (Z \setminus gX) \uplus gX$$

197 where  $[-, -]$  is case analysis on the disjoint union  $Y = (Y \setminus fX) \uplus fX$ . Thus a minimal pullback  
 198 square in  $\mathbf{W}$  is of the form:

$$\begin{array}{ccc} & X'_1 \cup X'_2 & \\ x \nearrow & & \nwarrow x' \\ X_1 \cong X'_1 & & X_2 \cong X'_2 \\ u \searrow & & \nearrow u' \\ & X'_1 \cap X'_2 & \end{array}$$

199 The factorization property is straightforward.  $\square$

200 An object  $w$  of  $\mathbf{W}$  models a set of generated/allocated names, with injective maps corresponding  
 201 to renamings and extensions with newly generated names.

202 In  $\mathbf{W}$ , a minimal pullback corresponds to a *partial bijection* between  $X_1$  and  $X_2$ , as used in  
 203 other work on logical relations for generativity [PS93, BKBH07]. We write  $u : x \hookrightarrow y$  to mean that  
 204  $u$  is a subset inclusion and also use the notation  $x \hookrightarrow y$  to denote the subset inclusion map from  $x$   
 205 to  $y$ . Of course, the use of this notation implies that  $x \subseteq y$ . Note that if we have a span  $u, u'$  then  
 206 we can choose  $x, x'$  so that  $\begin{smallmatrix} x \\ \diamond \\ x' \\ u \\ \diamond \\ u' \end{smallmatrix}$  is a minimal pullback and one of  $x$  and  $x'$  is an inclusion. To do  
 207 that, we simply replace the apex of any minimal pullback completion with an isomorphic one. The  
 208 analogous property holds for completion of co-spans to pullbacks.

209 In this paper, we fix the category of worlds to be  $\mathbf{W}$ . The general definitions, in particular that  
 210 of setoid-valued functors that we are going to give, make sense in other settings. For example, in  
 211 our treatment of proof-relevant logical relations for reasoning about stateful computation [BHN14],  
 212 we build a category of worlds from partial equivalence relations on heaps.

#### 213 4. SETOID-VALUED FUNCTORS

214 A functor  $A$  from the category of worlds  $\mathbf{W}$  to the category of setoids comprises as usual for  
 215 each  $w \in \mathbf{W}$  a setoid  $Aw$  and for each  $u : w \rightarrow w'$  a morphism of setoids  $Au : Aw \rightarrow Aw'$   
 216 preserving identities and composition. This means that there exist continuous functions of type  
 217  $\Pi a. Aw(a, (Aid) a)$ ; and for any two morphisms  $u : w \rightarrow w_1$  and  $v : w_1 \rightarrow w_2$  a continuous function  
 218 of type  $\Pi a. Aw_2(Av(Au a), A(vu) a)$ .

219 If  $u : w \rightarrow w'$  and  $a \in Aw$  we may write  $u.a$  or even  $ua$  for  $Au(a)$  and likewise for proofs in  
 220  $Aw$ . Note that there is a proof of equality of  $(uv).a$  and  $u.(v.a)$ . In the sequel, we shall abbreviate  
 221 ‘setoid-valued functor(s)’ as ‘s.v.f.’.

222 Intuitively, s.v.f. will become the denotations of types. Thus, an element of  $Aw$  is a value  
 223 involving at most the names in  $w$ . If  $u : w \rightarrow w_1$  then  $Aw \ni a \mapsto u.a \in Aw_1$  represents renaming  
 224 and possible weakening by names not “actually” occurring in  $a$ . Note that due to the restriction to  
 225 injective functions identification of names (“contraction”) is precluded. This is in line with Stark’s  
 226 use of set-valued functors on the category  $\mathbf{W}$  to model fresh names.

227 **Definition 4.1.** We call an s.v.f.,  $A$ , *pullback-preserving* if for every pullback square  $\mathbf{w}_u^x \diamond_u^{x'} \mathbf{w}'$  with  
 228 apex  $\bar{\mathbf{w}}$  and low point  $\underline{\mathbf{w}}$  the diagram  $\mathbf{A}w_{Au}^{Ax} \diamond_{Au'}^{Ax'} \mathbf{A}w'$  is a pullback in *Std*. This means that there is a  
 229 continuous function of type

$$\Pi a \in \mathbf{A}w. \Pi a' \in \mathbf{A}w'. \mathbf{A}\bar{\mathbf{w}}(x.a, x'.a') \rightarrow \Sigma \underline{a} \in \mathbf{A}\underline{\mathbf{w}}. \mathbf{A}w(u.\underline{a}, a) \times \mathbf{A}w'(u'.\underline{a}, a')$$

230 Thus, if two values  $a \in \mathbf{A}w$  and  $a' \in \mathbf{A}w'$  are equal in a common world  $\bar{\mathbf{w}}$  then this can only be  
 231 the case because there is a value in the “intersection world”  $\underline{\mathbf{w}}$  from which both  $a, a'$  arise.

232 Note that the ordering on worlds and world morphisms is discrete so that continuity only refers  
 233 to the  $\mathbf{A}w'(u.a, u.a')$  argument.

234 The following proposition is proved using a pullback of the form  ${}^u \diamond_v^u$ .

235 **Proposition 4.2.** If  $A$  is a pullback preserving s.v.f.,  $u : \mathbf{w} \rightarrow \mathbf{w}'$  and  $a, a' \in \mathbf{A}w$ , there is a continuous  
 236 function  $\mathbf{A}w'(u.a, u.a') \rightarrow \mathbf{A}w(a, a')$ . Moreover, the “common ancestor”  $\underline{a}$  of  $a$  and  $a'$  is unique up  
 237 to  $\sim$ .

238 All the s.v.f. that we define in this paper will turn out to be pullback-preserving. However, for  
 239 the results described in this paper pullback preservation is not needed. Thus, we will not use it  
 240 any further, but note that there is always the option to require that property should the need arise  
 241 subsequently.

242 Morphisms between functors are natural transformations in the usual sense; they serve to in-  
 243 terpret terms with variables and functions. In more explicit terms, a morphism from s.v.f.  $A$  to  $B$  is  
 244 an equivalence class of pairs  $e = (e_0, e_1)$  where  $e_0$  and  $e_1$  are continuous functions of the following  
 245 types:

$$\begin{aligned} e_0 &: \Pi \mathbf{w}. \mathbf{A}w \rightarrow \mathbf{B}w \\ e_1 &: \Pi \mathbf{w}. \Pi \mathbf{w}'. \Pi x : \mathbf{w} \rightarrow \mathbf{w}'. \Pi a \in \mathbf{A}w. \Pi a' \in \mathbf{A}w'. \mathbf{A}w'(x.a, a') \rightarrow \mathbf{B}w'(x.e_0(a), e_0(a')) \end{aligned}$$

246 Notice that worlds are discretely order, thus continuity only refers to the dependency of  $a, a'$  etc.

247 Two morphisms  $e = (e_0, e_1), e' = (e'_0, e'_1)$  are identified if there is a continuous function:

$$\mu : \Pi \mathbf{w}. \Pi a \in \mathbf{A}w. \mathbf{B}w(e(a), e'(a))$$

248 where as in the case of setoids, we omit subscripts where appropriate. These morphisms compose  
 249 in the obvious way and so the s.v.f. and morphisms between them form a category.

250

## 5. INSTANCES OF SETOID-VALUED FUNCTORS

251 We now describe some concrete functors that will allow us to interpret types of the  $\nu$ -calculus as  
 252 s.v.f. The simplest one endows any predomain with the structure of an s.v.f. where the equality  
 253 is proof-irrelevant and coincides with standard equality. The second one generalises the function  
 254 space of setoids and is used to interpret function types. The third one is used to model dynamic  
 255 allocation and is the only one that introduces proper proof-relevance.

256 **5.1. Base types.** For each predomain  $D$  we can define a constant s.v.f., denoted  $D$  as well, with  
 257  $Dw$  defined as the discrete setoid over  $D$  and  $Du$  as the identity. These constant s.v.f. serve as  
 258 denotations for base types like booleans or integers.

259 The s.v.f.  $N$  of names is given by  $Nw = \mathbf{w}$  where  $\mathbf{w}$  on the right hand side stands for the  
 260 discrete setoid over the discrete predomain of names in  $\mathbf{w}$ , and  $Nu = u$  for  $u : \mathbf{w} \rightarrow \mathbf{w}'$ . Thus, e.g.  
 261  $N\{1, 2, 3\} = \{1, 2, 3\}$ .



262 **5.2. Cartesian closure.** The category of s.v.f. is cartesian closed which follows from general  
 263 known properties on functor categories. The construction of product and function space follows  
 264 the usual pattern. We give it here explicitly for convenience.

265 Let  $A$  and  $B$  be s.v.f. The product  $A \times B$  is given by taking a pointwise product of setoids.  
 266 For the sake of completeness, we note that  $(A \times B)\mathbf{w} = A\mathbf{w} \times B\mathbf{w}$  (product predomain) and  $(A \times$   
 267  $B)\mathbf{w}((a, b), (a', b')) = A\mathbf{w}(a, a') \times B\mathbf{w}(b, b')$ . This defines a cartesian product on the category of s.v.f.  
 268 More generally, we can define indexed products  $\prod_{i \in I} A_i$  of a family  $(A_i)_i$  of s.v.f. We write  $1$  for the  
 269 empty indexed product and  $()$  for the only element of  $1\mathbf{w}$ . Note that  $1$  is the terminal object in the  
 270 category of s.v.f.

271 The function space  $A \Rightarrow B$  is the s.v.f. given as follows.  $|(A \Rightarrow B)\mathbf{w}|$  contains pairs  $(f_0, f_1)$   
 272 where  $f_0(u) \in |A\mathbf{w}_1 \Rightarrow B\mathbf{w}_1|$  for each  $\mathbf{w}_1$  and  $u : \mathbf{w} \rightarrow \mathbf{w}_1$ . If  $u : \mathbf{w} \rightarrow \mathbf{w}_1$  and  $v : \mathbf{w}_1 \rightarrow \mathbf{w}_2$  then

$$f_1(u, v) \in (A\mathbf{w}_1 \Rightarrow B\mathbf{w}_2)([A v \Rightarrow B\mathbf{w}_2] f_0(vu), [A\mathbf{w}_1 \Rightarrow B v] f_0(u))$$

273 where

$$\begin{aligned} [A v \Rightarrow B\mathbf{w}_2] &: (A\mathbf{w}_2 \Rightarrow B\mathbf{w}_2) \rightarrow (A\mathbf{w}_1 \Rightarrow B\mathbf{w}_2) \\ [A\mathbf{w}_1 \Rightarrow B v] &: (A\mathbf{w}_1 \Rightarrow B\mathbf{w}_1) \rightarrow (A\mathbf{w}_1 \Rightarrow B\mathbf{w}_2) \end{aligned}$$

274 are the obvious composition morphisms.

275 A proof in  $(A \Rightarrow B)\mathbf{w}((f_0, f_1), (f'_0, f'_1))$  is a function  $g$  that for each  $u : \mathbf{w} \rightarrow \mathbf{w}_1$  yields a proof  
 276  $g(u) \in (A\mathbf{w}_1 \Rightarrow B\mathbf{w}_1)(f_0(u), f'_0(u))$ .

277 The order on objects and proofs is pointwise as usual. The following is now clear from the  
 278 definitions.

279 **Proposition 5.1.** The category of s.v.f. is cartesian-closed.

280 **Definition 5.2.** An s.v.f.  $D$  is *pointed* if  $D\mathbf{w}$  is pointed for each  $\mathbf{w}$  and the transition maps  $Du :$   
 281  $D\mathbf{w} \rightarrow D\mathbf{w}_1$  for  $u : \mathbf{w} \rightarrow \mathbf{w}_1$  are strict.

282 **Theorem 5.3.** If  $D$  is a pointed s.v.f. then there exists a morphism  $Y : (D \Rightarrow D) \rightarrow D$  satisfying the  
 283 equations from Theorem 2.3 understood relative to the cartesian-closed structure of the category of  
 284 s.v.f.

285 *Proof.* The fixpoint combinator on the level of s.v.f. is defined pointwise: Given world  $\mathbf{w}$  and  
 286  $(f_0, f_1) \in (D \Rightarrow D)\mathbf{w}$  we define

$$Y\mathbf{w}(f_0, f_1) = Y(f_0(id_{\mathbf{w}}))$$

287 where  $Y$  is the setoid fixpoint combinator from Theorem 2.3. The translation of proofs is obvious.  
 288 We need to show that this defines a natural transformation. So, let  $u : \mathbf{w} \rightarrow \mathbf{w}_1$  and  $(f_0, f_1) \in (D \Rightarrow$   
 289  $D)\mathbf{w}$ . Put  $f := f_0(id_{\mathbf{w}})$  and  $g := f_0(u)$ . We need to construct a proof that  $Du(Y(f)) \sim Y(g)$ . Now,  $f_1$   
 290 furnishes a proof of  $(Du)f = g$  and  $Du$  is strict by assumption on  $D$  so that “Uniformity” furnishes  
 291 the desired proof.

292 The laws from Theorem 2.3 can be directly inherited. □

293 **Definition 5.4.** A s.v.f.  $A$  is discrete if  $A\mathbf{w}$  is a discrete setoid for every world  $\mathbf{w}$ .

294 The constructions presented so far only yield discrete s.v.f., *i.e.*, proof relevance is merely  
 295 propagated but never actually created. This is not so for the next operator on s.v.f., which is to  
 296 model dynamic allocation.

297

## 6. DYNAMIC ALLOCATION MONAD

298 Before we define the dynamic allocation monad we recall Stark's definition of a dynamic allocation  
 299 monad for the category of *set-valued functors* on the category of worlds. For set-valued functor  $A$ ,  
 300 Stark defines a set-valued functor  $TA$  by  $TAw = \{(w_1, a) \mid w \subseteq w_1, a \in Aw_1\}/\sim$  where  $(w_1, a) \sim$   
 301  $(w'_1, a')$  iff there exist maps  $x : w_1 \rightarrow \bar{w}$ ,  $x' : w'_1 \rightarrow \bar{w}$  for some  $\bar{w}$  satisfying  $x.i = x'.i'$  and  
 302  $x.a = x'.a'$  where  $i : w \hookrightarrow w_1$  and  $i' : w \hookrightarrow w'_1$  are the inclusion maps.

303 Our dynamic allocation monad for s.v.f. essentially mimics this definition, the difference being  
 304 that the maps  $i, i'$  witnessing equivalence of elements now become proofs of  $\sim$ -equality. Addition-  
 305 ally, our definition is based on predomains and involves a bottom element for recursion.

306 **6.1. Definition of the monad.** Let  $A$  be an s.v.f.. We put

$$|TAw| = \{(w_1, a) \mid w \subseteq w_1 \wedge a \in Aw_1\}_\perp$$

307 Thus, a non-bottom element of  $TAw$  consists of an extension of  $w$  together with an element of  $A$   
 308 taken at that extension. Note that the extension is not existentially quantified but part of the element.

309 The ordering is given by  $(w_1, a) \leq (w'_1, a')$  if  $w_1 = w'_1$  and  $a \leq a'$  in  $Aw_1$  and of course,  $\perp$  is  
 310 the least element of  $TAw$ .

311 The proofs are defined as follows. First,  $TAw(\perp, \perp) = \{\perp\}$  and second, the elements of  
 312  $TAw((w_1, a), (w'_1, a'))$  are triples  $(x, x', p)$  where  $x, x'$  complete the inclusions  $u : w \hookrightarrow w_1$  and  
 313  $u' : w \hookrightarrow w'_1$  to a commuting square

$$\begin{array}{ccc} & \bar{w} & \\ x \nearrow & & \nwarrow x' \\ w_1 & & w'_1 \\ u \searrow & w & \swarrow u' \end{array}$$

314 with  $\bar{w} = \text{cod}(x) = \text{cod}(x')$ . The third component  $p$  then is a proof that  $a$  and  $a'$  are equal  
 315 when transported to  $\bar{w}$ , formally,  $p \in A\bar{w}(x.a, x'.a')$ . The ordering is again discrete in  $x, x'$  and  
 316 inherited from  $A$  in  $p$ . Formally,  $((w_1, a), (w'_1, a'), (x, x', p)) \leq ((w_1, b), (w'_1, b'), (x, x', q))$  when  
 317  $(x.a, x'.a', p) \leq (x.b, x'.b', q)$  in  $A\text{cod}(x)$  and of course  $(\perp, \perp, \perp)$  is the least element. No  $\leq$ -relation  
 318 exists between triples with different mediating co-span. In particular, in an ascending chain of  
 319 proofs the witnessing spans are always the same, which is the intuitive reason why they can be  
 320 patched together to form a suprema.

321 Consider, for example, that  $w = \{0\}$ ,  $w_1 = \{0, 1, 2\}$ ,  $w'_1 = \{0, 2, 3\}$ . Then, both  $c = (w_1, (0, 2))$   
 322 and  $c' = (w'_1, (0, 3))$  are elements of  $T(N \times N)w$ , and  $(x, x', p) \in T(N \times N)w(c, c')$  is a proof that the  
 323 two are equal where  $x : w_1 \rightarrow \bar{w} = \{0, 1, 2, 3\}$  sends  $0 \mapsto 0, 1 \mapsto 1, 2 \mapsto 2$  and  $x' : w'_1 \rightarrow \bar{w}$  sends  
 324  $0 \mapsto 0, 2 \mapsto 3, 3 \mapsto 2$ . The proof  $p$  is the canonical proof by reflexivity. Note that, in this case, the  
 325 order relation is trivial. It becomes more interesting when the type of values  $A$  is a function space.

326 Next, we define the morphism part. Assume that  $u : w \rightarrow q$  is a morphism in  $\mathbf{W}$ . We want  
 327 to construct a morphism  $Au : TAw \rightarrow TAq$  in  $Std$ . So let  $(w_1, a) \in TAw$  and  $i : w \hookrightarrow w_1$  be the  
 328 inclusion. We complete the span  $i, u$  to a minimal pullback

$$\begin{array}{ccc} w_1 & \xrightarrow{u_1} & q_1 \\ i \uparrow & & \uparrow j \\ w & \xrightarrow{u} & q \end{array}$$

329 with  $j$  an inclusion as indicated. We then define  $TAu(w_1, a) = (q_1, u_1.a)$ . We assume a function  
 330 that returns such completions to minimal pullbacks in some chosen way. The particular choice is  
 331 unimportant.

332 Picking up the previous example and letting  $u : w \rightarrow q = \{0, 1\}$  be  $0 \mapsto 1$  then a possible  
 333 completion to a minimal pullback would be

$$\begin{array}{ccc} w_1 = \{0, 1, 2\} & \xrightarrow{0 \mapsto 1, 1 \mapsto 2, 2 \mapsto 3} & \{0, 1, 2, 3\} = q_1 \\ \uparrow i & & \uparrow j \\ w = \{0\} & \xrightarrow{0 \mapsto 1} & \{0, 1\} = q \end{array}$$

334 Note that the following square where the additional name 1 in  $q$  is identified with a name already  
 335 existing in  $w_1$  is *not* a pullback

$$\begin{array}{ccc} w_1 = \{0, 1, 2\} & \xrightarrow{0 \mapsto 1, 1 \mapsto 0, 2 \mapsto 3} & \{0, 1, 2, 3\} \\ \uparrow i & & \uparrow j \\ w = \{0\} & \xrightarrow{0 \mapsto 1} & \{0, 1\} = q \end{array} \quad (6.1)$$

336 Adding extra garbage into  $q_1$  like so would result in a pullback that is not minimal.

$$\begin{array}{ccc} w_1 = \{0, 1, 2\} & \xrightarrow{0 \mapsto 1, 1 \mapsto 2, 2 \mapsto 3} & \{0, 1, 2, 3, 4, 5\} \\ \uparrow i & & \uparrow j \\ w = \{0\} & \xrightarrow{0 \mapsto 1} & \{0, 1\} = q \end{array}$$

337 If  $(x, x', p)$  is a proof of  $(w_1, a) \sim (w'_1, a')$  then we obtain a proof,  $(q'_1, u'_1.a')$ , that  $TAu(w_1, a) \sim$   
 338  $TAu(w'_1, a')$  as follows. We first complete the span  $xi, u$  to a minimal pullback with apex  $\bar{q}$  and  
 339 upper arrow  $\bar{u} : \text{cod}(x) = \bar{w} \rightarrow \bar{q}$ . Now minimality of the pullbacks apexed at  $q_1$  and  $q'_1$  furnishes  
 340 morphisms  $y : q_1 \rightarrow \bar{q}$  and  $y' : q'_1 \rightarrow \bar{q}$  so that  $yj = y'j'$  (where  $j' : q \hookrightarrow q'_1$ ). We then have  
 341  $(y, y', \bar{u}.p) : TAu(w_1, a) \sim TAu(w'_1, a')$  as required. This shows that the passage  $(w_1, a) \mapsto (w'_1, a')$   
 342 defines indeed a morphism of setoids.

$$\begin{array}{ccccc} & & \bar{w} & \xrightarrow{\bar{u}} & \bar{q} \\ & & \swarrow x' & & \swarrow y' \\ & & w'_1 & \xrightarrow{u'_1} & q'_1 \\ w_1 & \xrightarrow{u_1} & q_1 & & \\ \uparrow i & \nearrow i' & \uparrow j & \nearrow j' & \\ w & \xrightarrow{u} & q & & \end{array}$$

343 The functor laws amount to similar constructions of  $\sim$ -witnesses and are left to the reader. The  
 344 following is direct from the definitions.

345 **Proposition 6.1.**  $T$  is a strong monad on the category of s.v.f. The unit sends  $v \in Aw$  to  $(w, v) \in$   
 346  $(TA)w$ . The multiplication sends  $(w_1, (w_2, v)) \in (TTA)w$  to  $(w_2, v) \in TAw$ . The strength map sends  
 347  $(a, (w_1, b)) \in (A \times TB)w$  to  $(w_1, (a.i, b))$  where  $i : w \hookrightarrow w_1$ .

348 Notice that if we had taken any arbitrary commuting square like the one shown in Equation 6.1,  
 349 then preservation of proofs could not be guaranteed because names in extensions would be captured  
 350 in an arbitrary way. Minimality, on the other hand, is a mere convenience.

351 **6.2. Comparison with cpo-valued functors.** The flawed attempt at defining a dynamic allocation  
 352 monad for FM-domains discussed by Shinwell [Shi04] and mentioned in the introduction can be  
 353 reformulated in terms of cpo-valued functors and further highlights the importance of proof-relevant  
 354 equality.

355 Given a cpo-valued functor  $A$  one may construct a poset-valued functor  $T_{sp}A$  which has for  
 356 underlying set equivalence classes of pairs  $(w_1, a)$  with  $w \subseteq w_1$  and  $a \in Aw_1$ . As in Stark's definition  
 357 above, we have  $(w_1, a) \sim (w'_1, a')$  if there are morphisms  $x, x'$  such that  $xi = x'i'$  and  $x.a = x'.a'$   
 358 where  $i : wrightarrow w_1, i' : w \rightarrow w'_1$  are the inclusions. As for the ordering, the only reasonable  
 359 choice is to decree that on representatives  $(w_1, a) \leq (w'_1, a')$  if  $x.a \leq x'.a'$  for some co-span  $x, x'$   
 360 with  $xi = x'i'$  where  $i, i'$  are the inclusions as above. However, while this defines a partial order  
 361 it is not clear why it should have suprema of ascending chains because the witnessing spans might  
 362 not match up so that they can be pasted to a witnessing span for the limit of the chain. Indeed,  
 363 Shinwell's thesis [Shi04] contains a concrete counterexample, which is due to Pitts.

364 The counterexample takes the following form in our notation: We define the cpo-valued functor  
 365  $A$  by  $Aw := (\mathcal{P}(w), \subseteq)$ . So the elements of  $Aw$  are subsets of  $w$  ordered by inclusion, hence a finite  
 366 cpo. Let us now examine  $T_{sp}A$ . An element of  $T_{sp}Aw$  is an  $\sim$ -equivalence class of pairs  $(w_1, U)$  where  
 367  $U \subseteq w_1, w \subseteq w_1$ . Furthermore,  $(w_1, U) \sim (w'_1, U')$  whenever  $U = U'$  and the ordering  $\leq$  on  $T_{sp}Aw$  is  
 368  $(w_1, U) \leq (w'_1, U')$  whenever  $U \subseteq U'$ . Let  $t_n$  be the equivalence class of  $(\{0, \dots, n-1\}, \{0, \dots, n-1\})$ .  
 369 We have  $t_n \in T_{sp}A\emptyset$  for all  $n$  and  $t_n \leq t_m \iff n \leq m$ . From this it is clear that the ascending chain  
 370  $t_0 \leq t_1 \leq \dots$  does not have a least upper bound in  $T_{sp}A\emptyset$  for if  $(w_1, U)$  were such an upper bound  
 371 then  $|U| \geq n$  would have to hold for all  $n$ .

372 The transition to proof relevance that we have made allows us to define the order on repre-  
 373 sentatives as we have done and thus to bypass these difficulties. We view  $A$  above as a s.v.f. with  
 374 underlying cpo  $Aw = w$  and, trivial, *i.e.*, discrete equality. Now applying our dynamic allocation  
 375 monad  $T$  to  $A$  yields the s.v.f.  $TAW$  whose underlying cpo contains in addition to  $\perp$ , pairs  $(w_1, U)$   
 376 where  $U \subseteq w_1$  with ordering  $(w_1, U) \leq (w'_1, U')$  if  $w_1 = w'_1$  and  $U \subseteq U'$ . A proof that an element  
 377  $(w_1, U)$  is equal to the element  $(w'_1, U')$  is given by a triple  $(w_2, u, u')$  such that  $u : w_1 \rightarrow w_2$  and  
 378  $u' : w'_1 \rightarrow w_2$  and moreover  $u(U) = u'(U')$ . The ordering in these proofs is the discrete one. Now  
 379 the sequence shown above is not an ascending chain and thus is no longer a counter-example to  
 380 completeness.

## 381 7. OBSERVATIONAL EQUIVALENCE AND FUNDAMENTAL LEMMA

382 We now construct the machinery that connects the concrete language with the denotational machin-  
 383 ery introduced in Section 1. The semantics of types, written using  $\llbracket \cdot \rrbracket$ , as s.v.f. is defined inductively  
 384 as follows:

- 385 • For basic types  $\llbracket \tau \rrbracket$  is the corresponding discrete s.v.f..
- 386 •  $\llbracket \tau \rightarrow \tau' \rrbracket$  is defined as the function space  $\llbracket \tau \rrbracket \rightarrow T\llbracket \tau' \rrbracket$ , where  $T$  is the dynamic allocation  
 387 monad.
- 388 • For typing context  $\Gamma$  we define  $\llbracket \Gamma \rrbracket$  as the indexed product of s.v.f.  $\prod_{x \in \text{dom}(\Gamma)} \llbracket \Gamma(x) \rrbracket$ .

389 To each term in context  $\Gamma \vdash e : \tau$  we can associate a morphism  $\llbracket e \rrbracket$  from  $\llbracket \Gamma \rrbracket$  to  $T\llbracket \tau \rrbracket$  by  
 390 interpreting the syntax in the category of s.v.f. using cartesian closure, the fixpoint combinator, and  
 391 the fact that  $T$  is a strong monad. We omit most of the straightforward but perhaps slightly tedious  
 392 definition and only give the clauses for “new” and “let” here:

$$\llbracket \text{new} \rrbracket \mathbf{w} = (\mathbf{w} \cup \{n + 1\}, n + 1)$$

393 where  $n = \max(\mathbf{w})$  and  $\max(\mathbf{w}) = \max(\{n \mid n \in \mathbf{w}\})$ , i.e., the greatest number in the world  $\mathbf{w}$ .

394 If  $f_1 : \llbracket \Gamma \rrbracket \rightarrow T\llbracket \tau_1 \rrbracket$  and  $f_2 : \llbracket \Gamma, x:\tau_1 \rrbracket \rightarrow T\llbracket \tau_2 \rrbracket$  are the denotations of  $\Gamma \vdash e_1 : \tau_1$  and  
 395  $\Gamma, x:\tau_2 \vdash e_2 : \tau_2$  then the interpretation of  $\text{let } x \leftarrow e_1 \text{ in } e_2$  is the morphism  $f : \llbracket \Gamma \rrbracket \rightarrow T\llbracket \tau_2 \rrbracket$  given  
 396 by

$$f = \mu \circ T f_2 \circ \sigma \circ \langle \text{id}_\Gamma, f_1 \rangle$$

397 where  $\mu$  is the monad multiplication,  $\sigma$  is the monad strength and where we have made the sim-  
 398 plifying assumption that  $\llbracket \Gamma, x:\tau \rrbracket = \llbracket \Gamma \rrbracket \times \llbracket \tau \rrbracket$ . Assuming that  $f_1$  and  $f_2$  now stand for the first  
 399 components of concrete representatives of these morphisms, one particular concrete representative  
 400 of this morphism (now also denoted  $f$ ) satisfies:

$$f\mathbf{w}(\gamma) = f_2(i.\gamma, a), \text{ where } i \text{ is the inclusion } \mathbf{w} \hookrightarrow \mathbf{w}_1 \text{ and } f_1\mathbf{w}(\gamma) = (\mathbf{w}_1, a).$$

401 Our aim is now to relate these morphisms to the computational interpretation  $\llbracket e \rrbracket$ .

402 **Definition 7.1.** For each type  $\tau$  and world  $\mathbf{w}$  we define two relations; the relation  $\Vdash_{\mathbf{w}}^\tau \subseteq \llbracket \tau \rrbracket \times \llbracket \tau \rrbracket \mathbf{w}$   
 403 and  $\Vdash_{\mathbf{w}}^{T\tau} \subseteq (\mathbb{N} \rightarrow (\mathbb{N} \times \llbracket \tau \rrbracket)_\perp) \times T\llbracket \tau \rrbracket \mathbf{w}$  by the following clauses.

$$b \Vdash_{\mathbf{w}}^{\text{bool}} \mathbf{b} \iff b = \mathbf{b}$$

$$i \Vdash_{\mathbf{w}}^{\text{int}} i \iff i = i$$

$$l \Vdash_{\mathbf{w}}^{\text{name}} k \iff l = k$$

$$f \Vdash_{\mathbf{w}}^{\tau \rightarrow \tau'} g \iff \forall \mathbf{w}_1 \supseteq \mathbf{w}. \forall v. \forall v'. v \Vdash_{\mathbf{w}_1}^\tau v' \Rightarrow f(v) \Vdash_{\mathbf{w}_1}^{T\tau'} g_0(\mathbf{w} \hookrightarrow \mathbf{w}_1, v)$$

$$c \Vdash_{\mathbf{w}}^{T\tau} \mathbf{c} \iff [c(\max(\mathbf{w}) + 1) = \perp \Leftrightarrow \mathbf{c} = \perp] \wedge [c(\max(\mathbf{w}) + 1) = (n_1, v) \wedge \mathbf{c} = (\mathbf{w}_1, v) \Rightarrow n_1 = \max(\mathbf{w}_1) + 1 \wedge v \Vdash_{\mathbf{w}_1}^\tau v)].$$

404 Notice that  $T\tau$  is not part of the syntax, but  $T$  is a marker to distinguish the two relations defined  
 405 above.

406 The following lemma states that the realizability relation is stable with respect to enlargement  
 407 of worlds. It is needed for the “fundamental lemma” 7.3.

408 **Lemma 7.2.** Let  $\tau$  be a type. If  $u : \mathbf{w} \hookrightarrow \mathbf{w}_1$  is an inclusion as indicated and  $v \Vdash_{\mathbf{w}}^\tau u.v$ ,  
 409 too.

410 The proof is by a straightforward induction on types. Note, however, that the restriction to  
 411 inclusions is important for the cases of function type and the type name. We extend  $\Vdash$  to typing  
 412 contexts by putting

$$\eta \Vdash_{\mathbf{w}}^\Gamma \gamma \iff \forall x \in \text{dom}(\Gamma). \eta(x) \Vdash_{\mathbf{w}}^{\Gamma(x)} \gamma(x)$$

413 for  $\eta \in \llbracket \Gamma \rrbracket$  and  $\gamma \in \llbracket \Gamma \rrbracket$ .

414 **Theorem 7.3 (Fundamental lemma).** Let  $\Gamma \vdash e : \tau$  be a well typed term. There exists a representa-  
 415 tive  $(\mathbf{c}, \_)$  of the equivalence class  $\llbracket e \rrbracket$  at world  $\mathbf{w}$  such that if  $\eta \Vdash_{\mathbf{w}}^\Gamma \gamma$  then  $\llbracket e \rrbracket \eta \Vdash_{\mathbf{w}}^{T\tau} \mathbf{c}(\gamma)$ .

416 *Proof.* By induction on typing rules. We always chose for the representative the one given as witness  
 417 in the definition of  $\llbracket e \rrbracket$ . Most of the cases are straightforward. For illustration we show **new** and  
 418 **let** : As for **new**, we pick the representative  $\mathbf{c}$  that at world  $\mathbf{w}$  returns  $(\mathbf{w} \cup \{\max(\mathbf{w}) + 1\}, \max(\mathbf{w}))$ .  
 419 Now, with  $c = \llbracket \text{new} \rrbracket$ , we have  $c(\max(\mathbf{w})) = (\max(\mathbf{w}) + 1, \max(\mathbf{w}))$  and  $c \Vdash_{\mathbf{w}}^{T\tau} \mathbf{c}$  holds, since  
 420  $\max(\mathbf{w} \cup \{\max(\mathbf{w}) + 1\}) = \max(\mathbf{w}) + 1$ .

421 Next, assume that  $\Gamma \vdash \text{let } x \Leftarrow e_1 \text{ in } e_2 : \tau_2$ , where  $\Gamma \vdash e_1 : \tau_1$  and  $\Gamma, x : \tau_1 \vdash e_2 : \tau_2$ .  
 422 Choose, according to the induction hypothesis appropriate representatives  $c_1$  of  $\llbracket e_1 \rrbracket$  and  $c_2$  of  $\llbracket e_2 \rrbracket$ .  
 423 If  $\eta \Vdash_{\mathbf{w}}^{\Gamma} \gamma$  for some initial world  $\mathbf{w}$  then we have (H1)  $\llbracket e_1 \rrbracket \eta \Vdash_{\mathbf{w}}^{T\tau_1} c_1(\gamma)$ . If  $\llbracket e_1 \rrbracket \eta(\max(\mathbf{w}) + 1) = \perp$   
 424 then  $c_1(\gamma) = \perp$ , too, and the same goes for the interpretation of the entire let-construct. So suppose  
 425 that  $\llbracket e_1 \rrbracket \eta(\max(\mathbf{w}) + 1) = (n_1, \nu)$ . By (H1), we must then have  $c_1(\gamma) = (\mathbf{w}_1, \nu)$  where  $\mathbf{w} \subseteq \mathbf{w}_1$  and  
 426  $n_1 = \max(\mathbf{w}_1) + 1$  and  $\nu \Vdash_{\mathbf{w}_1}^{\tau_1} \nu$ .

427 By Lemma 7.2 we then have  $\eta \Vdash_{\mathbf{w}_1}^{\Gamma} i.\gamma$  where  $i : \mathbf{w} \hookrightarrow \mathbf{w}_1$ . Thus, by the induction hypothesis,  
 428 we get (H2)  $\llbracket e_2 \rrbracket (\eta[x \mapsto \nu]) \Vdash_{\mathbf{w}_1}^{T\tau_2} c_2(i.\gamma[x \mapsto \nu])$ . Thus, putting  $c(\gamma) = c_2(i.\gamma, \nu)$  furnishes the required  
 429 representative of  $\llbracket \text{let } x \Leftarrow e_1 \text{ in } e_2 \rrbracket(\mathbf{w})$   $\square$

430 **Remark 7.4.** Note that the particular choice of representative matters here. For example, if  $c_0 \mathbf{w} =$   
 431  $(\mathbf{w} \uplus \{\max(\mathbf{w}) + 1, \max(\mathbf{w}) + 2\}, \max(\mathbf{w}) + 1)$  then there exists  $c_1$  such that  $(c_0, c_1) : 1 \rightarrow TN$  and  
 432  $(c_0, c_1)$  and  $\llbracket \text{new} \rrbracket$  are equal qua morphisms of s.v.f. Yet,  $\llbracket \text{new} \rrbracket \not\approx_{\mathbf{w}}^{TN} c_0$ .

433 It would have been an option to refrain from the identification of  $\sim$ -related morphisms. The  
 434 formulation of the Fundamental Lemma would then have become slightly easier as we would have  
 435 defined  $\llbracket e \rrbracket$  so as to yield the required witnesses directly. On the other hand, the equational proper-  
 436 ties of the so obtained category would be quite weak and in particular cartesian closure, monad laws,  
 437 functor laws, etc would only hold up to  $\sim$ . This again would not really be a problem but prevent the  
 438 use of standard category-theoretic terminology.

### 439 7.1. Observational Equivalence.

440 **Definition 7.5.** Let  $\tau$  be a type. We define an *observation of type  $\tau$*  as a closed term  $\vdash o : \tau \rightarrow \text{bool}$ .  
 441 Two values  $\nu, \nu' \in \llbracket \tau \rrbracket$  are *observationally equivalent at type  $\tau$*  if for all observations  $o$  of type  
 442  $\tau$  one has that  $\llbracket o \rrbracket(\nu)(0)$  is defined iff  $\llbracket o \rrbracket(\nu')(0)$  is defined and when  $\llbracket o \rrbracket(\nu)(0) = (n_1, \nu_1)$  and  
 443  $\llbracket o \rrbracket(\nu')(0) = (n'_1, \nu'_1)$  then  $\nu_1 = \nu'_1$ .

444 Note that observational equivalence is a congruence since an observation can be extended by  
 445 any englobing context. We also note that observational equivalence is the coarsest reasonable con-  
 446 gruence.

447 We now show how the proof-relevant semantics can be used to deduce observational equiva-  
 448 lences.

449 **Theorem 7.6** (Observational equivalence). If  $\tau$  is a type and  $\nu \Vdash_{\emptyset}^{\tau} e$  and  $\nu' \Vdash_{\emptyset}^{\tau} e'$  with  $e \sim e'$  in  $\llbracket \tau \rrbracket \emptyset$   
 450 then  $\nu$  and  $\nu'$  are observationally equivalent at type  $\tau$ .

451 *Proof.* Let  $o$  be an observation at type  $\tau$ . By the Fundamental Lemma (Theorem 7.3) we have  
 452  $\llbracket o \rrbracket \Vdash_{\emptyset}^{\tau \rightarrow \text{bool}} \llbracket o \rrbracket$ .

453 Now, since  $e \sim e'$  we also have  $\llbracket o \rrbracket(e) \sim \llbracket o \rrbracket(e')$  and, of course,  $\llbracket o \rrbracket(\nu) \Vdash_{\emptyset}^{T\text{bool}} \llbracket o \rrbracket(e)$  and  
 454  $\llbracket o \rrbracket(\nu') \Vdash_{\emptyset}^{T\text{bool}} \llbracket o \rrbracket(e')$ .

455 From  $\llbracket o \rrbracket(e) \sim \llbracket o \rrbracket(e')$ <sup>1</sup> we conclude that either  $\llbracket o \rrbracket(e)(0)$  and  $\llbracket o \rrbracket(e')(0)$  both diverge in which  
 456 case the same is true for  $\llbracket o \rrbracket(\nu)(0)$  and  $\llbracket o \rrbracket(\nu')(0)$  by definition of  $\Vdash_{\emptyset}^{T\text{bool}}$ . Secondly, if  $\llbracket o \rrbracket(e)(0) =$   
 457  $(-, \rightarrow, b, -)$  and  $\llbracket o \rrbracket(e')(0) = (-, \rightarrow, b', -)$  for booleans  $b, b'$  then, by definition of  $\sim$  at  $T\llbracket \text{bool} \rrbracket$  we get  
 458  $b = b'$  and, again by definition of  $\Vdash_{\emptyset}^{T\text{bool}}$  this then implies that  $\llbracket o \rrbracket(\nu)(0) = (-, b)$  and  $\llbracket o \rrbracket(\nu')(0) =$   
 459  $(-, b')$  with  $b = b'$ , hence the claim.  $\square$

<sup>1</sup>More precisely, we are using the representative of the equivalence class given by Theorem 7.3.

460

## 8. DIRECT-STYLE PROOFS

 461 We now have enough machinery to provide direct-style proofs for equivalences involving name  
 462 generation.

 463 If  $\Gamma \vdash e : \tau$  and  $\Gamma \vdash e' : \tau$ , we say the equation  $\Gamma \vdash e = e' : \tau$  is *semantically sound* if  
 464  $\llbracket e \rrbracket = \llbracket e' \rrbracket$  are equal morphisms from  $\llbracket \Gamma \rrbracket$  to  $\llbracket \tau \rrbracket$ . If  $v = v'$  can be derived by sound equations and  
 465 congruence rules, then  $\llbracket v \rrbracket$  and  $\llbracket v' \rrbracket$  are equivalent by Theorem 7.6. We omit the formal definition  
 466 of such derivations using an equational theory. We refer to [BHN14] for details on how this could  
 467 be set up.

 468 From the categorical properties of setoids the soundness of  $\beta, \eta$ , fixpoint unrolling and simi-  
 469 lar equations is obvious. We now demonstrate the soundness of the more interesting equations  
 470 involving name generation.

 471 **8.1. Drop equation.** We start with the following equation, which eliminates a dummy allocation:

$$c := (\text{let } x \leftarrow \text{new in } e) = e =: c', \text{ provided } x \text{ is not free in } e.$$

 472 Formally we have  $\Gamma \vdash e : \tau$  and the equation reads  $\Gamma \vdash c = c' : \tau$ . We have  $\llbracket c' \rrbracket w(\gamma) = (w_1, v)$  for  
 473 some extension  $w_1$  of  $w$  and  $v : \llbracket \tau \rrbracket w_1$  and  $\llbracket c \rrbracket w(\gamma) = (w_2, i.v)$  where  $w_2 = w_1 \cup \{\max(w_1) + 1\}$  and  
 474  $i : w_1 \hookrightarrow w_2$ .

 475 Now it remains to construct a proof of  $(w_1, v) \sim (w_2, v) \in TAW$ , which should depend contin-  
 476 uously on  $\gamma$ . To that end, we consider the following pullback square, where the annotations above  
 477 and below the square are just to illustrate in which world the semantic values are:

$$\begin{array}{ccc}
 \llbracket c' \rrbracket \gamma & & v \\
 & \nearrow & \\
 & w_1 & \xrightarrow{i} \\
 w & \nearrow & w_2 \\
 & \searrow & \nearrow \\
 & w_2 & \xrightarrow{id} \\
 \llbracket c \rrbracket \gamma & & i.v
 \end{array}$$

 478 Clearly we have  $i.v \sim id.i.v$  and therefore the pullback above is a proof that  $(w_1, v) \sim (w_2, v) \in TAW$ .

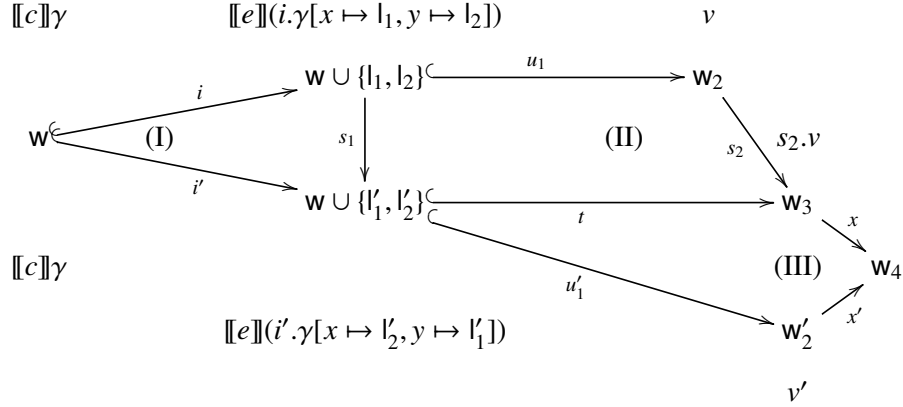
 479 **8.2. Swap equation.** Let us now consider the following equivalence where the order in which the  
 480 names are generated is switched:

$$c := (\text{let } x \leftarrow \text{new in let } y \leftarrow \text{new in } e) = (\text{let } y \leftarrow \text{new in let } x \leftarrow \text{new in } e) =: c'.$$

 481 Let  $l_1, l_2, l'_1, l'_2$  be the concrete locations allocated by the left-hand-side and right-hand-side of the  
 482 equation. In fact,  $l_1 = \max(w) + 1, l_2 = l_1 + 1$  and  $l'_2 = l_1$  and  $l'_1 = l_2$ . We have  $\llbracket c \rrbracket \gamma = (w_2, v)$ , where  
 483  $\llbracket e \rrbracket(i.\gamma[x \mapsto l_1, y \mapsto l_2]) = (w_2, v)$ . We also have  $\llbracket c' \rrbracket \gamma$ , where  $\llbracket e \rrbracket(i'.\gamma[x \mapsto l'_2, y \mapsto l'_1]) = (w'_2, v')$ .

 484 Define  $s(l_1) = l'_2, s(l_2) = l'_1$  and  $s \upharpoonright w = id$ . Naturality of  $\llbracket e \rrbracket$ , i.e.,  $\llbracket e \rrbracket \circ \llbracket \Gamma, x, y \rrbracket s \sim T \llbracket t \rrbracket s \circ \llbracket e \rrbracket$   
 485 furnishes a co-span  $x, x'$  so that  $x.s_2.v \sim x'.v'$  and  $xt = x'u'_1$  (III). Here  $s_2, t$  is the completion of the

486 span  $s_1, u_1$  to a minimal pullback as contained in the definition of  $T\llbracket\tau\rrbracket s$ .



487 Notice that the square (I) commutes by definition of  $s_1$ ; the square (II) commutes because it is  
 488 a minimal pullback. As a result the entire diagram commutes.  $x s_2 u_1$  and  $x' u'_1$  is the proof that  
 489  $\llbracket c \rrbracket \gamma \sim \llbracket c' \rrbracket \gamma$ .

490 This is essentially the proof as given in the Stark's thesis [Sta94], but now it also works  
 491 in the presence of recursion.

## 492 9. PROOF-RELEVANT PARAMETRIC FUNCTORS

493 The following equation (Stark's "Equivalence 12") cannot be validated in Stark's functor category  
 494 model and neither is it valid in the category of s.v.f.

$$(let\ n \leftarrow new\ in\ fun\ x.x = n) = (fun\ x.false). \quad (9.1)$$

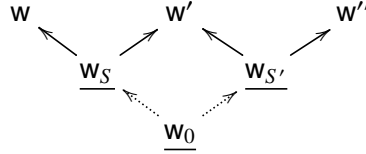
495 The above is, nevertheless, a valid contextual equivalence. The intuition is that the name  $n$  generated  
 496 in the left-hand side is never revealed to the context and is therefore distinct from any name that the  
 497 context might pass in as argument to the function; hence, the function will always return `false`. To  
 498 justify this equivalence, Stark constructs a model based on the more traditional Kripke logical rela-  
 499 tions. He also gives a category-theoretic version of that logical relation using so-called parametric  
 500 functors. In this section, we construct a proof-relevant version of these parametric functors which  
 501 will allow us to justify the above equivalence in the presence of recursion and in direct style. In fact,  
 502 this seems to be the first time that this equivalence has been established in this setting; we are not  
 503 aware of an earlier extension of parametric functors to recursion.

504 We also show that the transition to proof relevance makes the induced logical relation transitive,  
 505 something that is apparently not possible with traditional Kripke logical relations.

506 **9.1. Spans of Worlds.** We use capital letters  $S, S', \dots$  for spans of worlds. If  $S$  is the span  $w \xleftarrow{u}$   
 507  $\underline{w} \xrightarrow{u'} w'$  then we use the notations  $S : w \leftrightarrow w'$  and  $w = \text{dom}(S)$  (left domain),  $w' = \text{dom}'(S)$  (right  
 508 domain),  $\underline{w} = \text{lop}(S)$  (low point),  $u = S.u$ ,  $u' = S.u'$ . For world  $w$  we denote  $r(w) : w \leftrightarrow w$  the  
 509 identity span  $w \xleftarrow{id} w \xrightarrow{id} w$ . If  $S : w \leftrightarrow w'$  then  $s(S) : w' \leftrightarrow w$  is given by  $w' \xleftarrow{S.u'} \text{lop}(S) \xrightarrow{S.u} w$ . If  
 510  $S : w \leftrightarrow w'$  and  $S' : w' \leftrightarrow w''$  then we define  $t(S, S') : w \leftrightarrow w''$  as  $w \xleftarrow{S.u} x \xrightarrow{S'.u' x'} w''$  where  $x, x'$

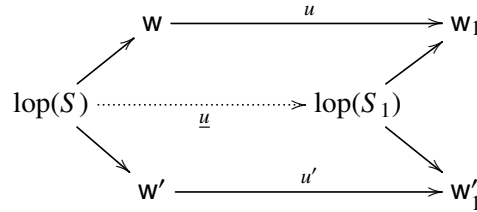


511 complete  $S.u'$  and  $S'.u$  to a pullback square.



512 We assume a fixed choice of such completions to pullback squares. We do not assume that the  
 513  $t$ -operation is associative or satisfies any other laws.

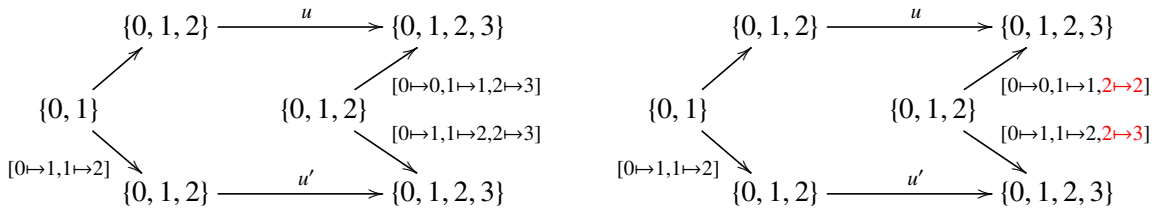
514 **Definition 9.1.** A *parametric square* consists of two spans  $S : w \leftrightarrow w'$  and  $S_1 : w_1 \leftrightarrow w'_1$  and  
 515 two morphisms  $u : w \rightarrow w_1$  and  $u' : w' \rightarrow w'_1$  such that there exists a morphism  $\underline{u}$  making the two  
 516 squares in the following diagram pullbacks (thus in particular commute).



517 We use the notation  $(u, u') : S \rightarrow S_1$  in this situation.

518 Note that the witnessing morphism  $\underline{u}$  is uniquely determined since we can complete  $S_1$  to a  
 519 pullback in which case  $\underline{u}$  is the unique mediating morphism given by universal property of the latter  
 520 pullback.

521 The reader is invited to check that under the interpretation of spans as partial bijections the  
 522 presence of a parametric square  $(u, u') : S \rightarrow S'$  asserts that  $S'$  is obtained from  $S$  by consistent  
 523 renaming followed by the addition of links and “garbage”. In the following diagram the left diagram  
 524 is parametric and the right one is not. In particular, the value 2 is mapped to 2 and 3 in the diagram  
 525 to the right (as illustrated in red).



526 We also note that if  $S, S' : w \leftrightarrow w'$  then  $(id, id) : S \rightarrow S'$  is a parametric square if and only if there  
 527 exists an isomorphism  $t : \text{lop}(S) \rightarrow \text{lop}(S')$  such that  $S'.u t = S.u$  and  $S'.u' t = S.u'$ . In this case,  
 528 we call  $S$  and  $S'$  isomorphic spans and write  $S \cong S'$ . Notice that for any span  $S : w \leftrightarrow w'$  we have  
 529  $t(S, r(w')) \cong S \cong t(r(w), w')$  as well as other properties such as associativity of  $t(\cdot, \cdot)$  up to  $\cong$ .

530 **Definition 9.2.** A *parametric functor* is a set-valued functor on the category of worlds (a set  $Aw$  for  
 531 each world  $w$  and functorial transition functions  $Au : Aw \rightarrow Aw'$  when  $u : w \rightarrow w'$ ) together with a  
 532 relation  $AS \subseteq Aw \times Aw'$  for each span  $S : w \leftrightarrow w'$ . It is required that  $Ar(w)$  is the equality relation  
 533 on  $Aw$ .

534 **9.2. Parametric Set-Valued Functors.** Our aim is now to define a proof-relevant version of para-  
 535 metric functors: parametric s.v.f. It would be possible to do so by taking a s.v.f. and adding a  
 536 relational component which, however, would also have to proof-relevant and compatible with the  
 537 proof-relevant equality that is already present. Instead, we use the following more economical ap-  
 538 proach which also makes the action on spans (the “relational component”) not only reflexive as  
 539 already required in Definition 9.2 but also symmetric and transitive.

540 So, just like an s.v.f., a parametric s.v.f.  $A$ , contains a predomain  $Aw$  for each  $w$  and for each  
 541  $u : w \rightarrow w'$  a continuous function  $Au : Aw \rightarrow Aw'$ . This time, however, we have a “heterogeneous  
 542 equality” allowing one to compare elements of two different worlds without the need of transporting  
 543 them to a larger common world as done in s.v.f.. Thus, a parametric s.v.f. has for each span  
 544  $S : w \leftrightarrow w'$  and elements  $a \in Aw, a' \in Aw'$ , a set of “proofs”  $AS(a, a')$  asserting equality of  
 545 these elements. As in the case of s.v.f., the set of tuples  $(S, a, a', p)$  with  $p \in AS(a, a')$  must  
 546 carry a predomain structure. We also require this semantic equality to be reflexive, symmetric  
 547 and transitive in an heterogeneous sense, thus employing the  $r, s, t$  operations on spans defined  
 548 above. Furthermore, the transition functions should behave functorially as in s.v.f., this time in the  
 549 sense of the “heterogeneous equality”. Every s.v.f. gives rise to parametric s.v.f. by instantiating  
 550 the heterogeneous equality to the larger world, but not all parametric s.v.f.s. are of this form (see  
 551 Example 9.6).

552 **Definition 9.3.** A parametric s.v.f.  $A$  consists of the following data.

- 553 (1) For each world  $w$  a predomain  $Aw$ .  
 554 (2) For each  $u : w \rightarrow w'$  a continuous function  $Au : Aw \rightarrow Aw'$ . We use the notation  $u.a =$   
 555  $Au(a)$ .  
 556 (3) For each span  $S : w \leftrightarrow w'$  and  $a \in Aw$  and  $a' \in Aw'$  a set  $AS(a, a')$  such that the set of  
 557 quadruples  $(S, a, a', p)$  with  $p \in AS(a, a')$  is a predomain with continuous second and third  
 558 projections and discrete ordering in the first component.  
 559 (4) For each parametric square  $(u, u') : S \rightarrow S'$  a continuous function

$$A(u, u') : \Pi a \in \text{Adom}(S). \Pi a' \in \text{Adom}'(S). AS(a, a') \rightarrow AS'(u.a, u'.a')$$

- 560 (5) For each parametric square  $(id, id) : S \rightarrow S'$  a continuous function

$$A(S, S') : \Pi a \in \text{Adom}(S). \Pi a' \in \text{Adom}'(S). AS(a, a') \rightarrow AS'(a, a')$$

- 561 (6) Continuous functions of the following types, witnessing reflexivity, symmetry and transi-  
 562 tivity in the “heterogeneous sense”:

$$\begin{aligned} & \Pi w. \Pi a \in Aw. Ar(w)(a, a) \\ & \Pi S. \Pi a \in \text{Adom}(S). \Pi a' \in \text{Adom}'(S). AS(a, a') \rightarrow As(S)(a', a) \\ & \Pi w \ w' \ w''. \Pi S : w \leftrightarrow w'. \Pi S' : w' \leftrightarrow w''. \Pi a \in Aw. \Pi a' \in Aw'. \Pi a'' \in Aw''. \\ & AS(a, a') \times AS'(a', a'') \rightarrow At(S, S')(a, a'') \end{aligned}$$

- 563 (7) Continuous functions of the following types, witnessing the functorial laws:

$$\begin{aligned} & \Pi w. \Pi a \in Aw. Ar(w)(a, id.a) \\ & \Pi w \ w_1 \ w_2. \Pi u : w \rightarrow w_1. \Pi v : w_1 \rightarrow w_2. Ar(w_2)(v.u.a, (vu).a) \end{aligned}$$

564 Suppose that  $S, S' : w \leftrightarrow w'$  are isomorphic spans between  $w$  and  $w'$  in the sense that  $(id, id) :$   
 565  $S \rightarrow S'$  where  $t$  is the isomorphism associated to  $(id, id)$ . The purpose of axiom (5) is so that if (we  
 566 have an element of)  $AS(a, a')$  then  $A(id, id)(a, a') \in AS'(id.a, id.a')$  by axiom (4) which is almost  
 567 but not quite as good as  $A(S, S')(a, a') \in AS'(a, a')$  which we get by axiom (5). Indeed, from (an  
 568 element of)  $AS'(id.a, id.a)$  we even get (an element of)  $At(r(w), t(S', s(r(w))))(a, a')$  using Axioms  
 569 (6) and (7), but without explicitly postulating axiom (5) as we do or making extra assumptions on

570 the  $t(-, -)$  or  $id.-$  operations it seems impossible to reach  $AS'(a, a')$ . The following lemma is an  
 571 instance of Axiom (5):

572 **Lemma 9.4.** If  $S \cong S'$  are isomorphic spans over  $w, w'$ , there is a continuous function of type:

$$\Pi a \in Aw. \Pi a' \in Aw'. AS(a, a') \rightarrow AS'(a, a')$$

573 **Lemma 9.5.** Let  $A$  be a parametric s.v.f. We have a continuous function of type:

$$\Pi a \in Aw. \Pi w_1. \Pi u : w \rightarrow w_1. AS(a, u.a)$$

574 where  $S$  is  $w \xleftarrow{id} w \xrightarrow{u} w_1$ .

575 *Proof.* We use the parametric square  $(id, u) : S_0 \rightarrow S$  where  $S_0$  is  $w \xleftarrow{id} w \xrightarrow{id} w$ . □

576 Every parametric s.v.f. also is a plain s.v.f. where we just define  $Aw(a, a') = Ar(w)(a, a')$  and  
 577 quotient the transition maps  $Au$  by pointwise  $\sim$ -equivalence.

578 But also every s.v.f.  $A$  can be extended to a parametric s.v.f.: first fix a particular choice of  
 579 transition functions  $Au : Aw \rightarrow Aw'$  when  $u : w \rightarrow w'$ . Now define  $AS(a, a') = A\bar{w}(x.a, x'.a')$   
 580 where  $\bar{w}$  is the apex of a completion of  $S$  to a minimal pullback and  $x : \text{dom}(S) \rightarrow \bar{w}$ ,  $x' : \text{dom}'(S) \rightarrow \bar{w}$   
 581 are the corresponding maps.

582 However, this correspondence is not one-to-one. For a concrete counterexample, consider the  
 583 following example which also lies at the heart of the justification of “Equivalence 12” with para-  
 584 metric functors.

585 **Example 9.6.** The parametric s.v.f.  $[N \Rightarrow B]$  is defined by  $[N \Rightarrow B]w = 2^w$  (functions from  $w$  to  
 586  $\{\text{true}, \text{false}\}$ ) and

$$[N \Rightarrow B]S(f, f') = \begin{cases} \{\star\} & \text{if } \forall n \in \text{lop}(S). f(S.u(n)) = f'(S.u'(n)) \\ \emptyset & \text{otherwise} \end{cases}$$

587 Now, let  $S$  be  $\{0\} \leftarrow \emptyset \rightarrow \emptyset$  and put  $f(x) = “x=0”$  and  $f'(x) = \text{false}$ . We have  $[N \Rightarrow B]S(f, f')$ , i.e.,  
 588  $[N \Rightarrow B]S(f, f') = \{\star\}$ , thus  $f$  and  $f'$  are considered equal above span  $S$ . On the other hand, if we  
 589 complete  $S$  to a minimal pullback by  $\{0\} \xrightarrow{1} \{0\} \xleftarrow{x'} \emptyset$  then  $[N \Rightarrow B]r(\{0\})(f, x'.f) = \emptyset$ , i.e.,  $f$  and  $f'$   
 590 are not equal when regarded over the least common world, namely  $\{0\}$ .

591 **Definition 9.7.** A parametric natural transformation,  $f$ , from parametric s.v.f.  $A$  to  $B$  consists of  
 592 two continuous functions

$$f_0 : \Pi w. Aw \rightarrow Bw$$

$$f_1 : \Pi S. \Pi a \in \text{dom}(S). \Pi a' : \text{dom}'(S). AS(a, a') \rightarrow BS(f_0 \text{dom}(S)(a), f_0 \text{dom}'(S)(a'))$$

593 As usual we refer both  $f_0$  and  $f_1$  as  $f$ . Two parametric natural transformations  $f, f' : A \rightarrow B$  are  
 594 identified if there is a continuous function of type

$$\Pi w. \Pi a \in Aw. Br(w)(f w(a), f' w(a))$$

595 The identification of “pointwise equal” parametric natural transformations is meaningful as  
 596 follows:

597 **Lemma 9.8.** Let  $f$  and  $f'$  be representatives of the same parametric natural transformation  $A \rightarrow B$ .  
 598 There then is a continuous function of the following type:

$$\Pi S. \Pi a \in \text{dom}(S). \Pi a' \in \text{dom}'(S). AS(a, a') \rightarrow BS(f \text{dom}(S)(a), f' \text{dom}'(S)(a'))$$

599 *Proof.* Given  $S$ ,  $a$ ,  $a'$ , and  $p \in AS(a, a')$  we obtain  $BS(f \text{dom}(S)(a), f \text{dom}'(S)(a'))$  and we also  
 600 obtain  $Br(\text{dom}'(S))(f \text{dom}'(S)(a'), f \text{dom}'(S)(a'))$  since  $f$  and  $f'$  are pointwise equal. We conclude  
 601 by transitivity and Lemma 9.4. □

602 **Lemma 9.9.** If  $f : A \rightarrow B$  is a parametric natural transformation then there are continuous functions  
 603 of the following types

$$\begin{aligned} & \Pi w \mathbf{w}_1. \Pi u : w \rightarrow \mathbf{w}_1. \Pi a \in AW. Br(\mathbf{w}_1)(u.fw(a), fw_1(u.a)) \\ & \Pi w. \Pi a \ a' \in AW. Ar(w)(a, a') \rightarrow Br(w)(fw(a), fw(a')) \end{aligned}$$

604 *Proof.* Fix  $u, a$  and  $w$ . Lemma 9.5 furnishes an element of  $AS(a, u.a)$  where  $S$  is  $w \xleftarrow{1} w \xrightarrow{u} \mathbf{w}_1$ .  
 605 Since  $f$  is a parametric natural transformation, we then get an element of  $BS(fw(a), fw_1(u.a))$ . We  
 606 then get the desired element of  $Br(\mathbf{w}_1)(u.fw(a), fw_1(u.a))$  by applying parametricity of  $B$  to the  
 607 parametric square  $(u, 1) : S \rightarrow r(\mathbf{w}_1)$ .  $\square$

608 **Theorem 9.10.** The parametric s.v.f. with parametric natural transformations form a cartesian  
 609 closed category with fixpoint operator obeying the laws from Theorems 2.3 & 5.3. There is a  
 610 strong monad  $T$  on this category where

$$\begin{aligned} TAW &= \{(w_1, a) \mid w \subseteq w_1 \wedge a \in AW_1\}_\perp \\ TAS((w_1, a), (w'_1, a')) &= \{(S_1 : w_1 \leftrightarrow w'_1, p) \mid (w \hookrightarrow w_1, w' \hookrightarrow w'_1) : S \rightarrow S_1 \wedge p \in AS'(a, a')\} \end{aligned}$$

611 *Proof.* The interesting bit is the proof of transitivity for the monad for it seems to rely essentially  
 612 on proof relevance. So, suppose that  $S : w \leftrightarrow w'$  and  $S' : w' \leftrightarrow w''$  and that  $(w_1, a) \in TAW$  and  
 613  $(w'_1, a') \in TAW'$  and  $(w''_1, a'') \in TAW''$ . Furthermore, suppose that  $(S_1, p) \in TAS((w_1, a), (w'_1, a'))$   
 614 and  $(S'_1, p') \in TAS'((w'_1, a'), (w''_1, a''))$ .

615 Now, by definition, we have  $S_1 : w_1 \leftrightarrow w'_1$  and  $S'_1 : w'_1 \leftrightarrow w''_1$  and also  $p \in AS_1(a, a')$  and  
 616  $p' \in AS'_1(a', a'')$ . We thus obtain (an element of)  $At(S_1, S'_1)(a, a'')$  and this, together with  $t(S_1, S'_1)$   
 617 furnishes the required proof.  $\square$

618 **Remark 9.11.** Notice that if the extensions  $w_1$  were existentially quantified, as in Stark's original  
 619 definition, then the transitivity construction in the above proof would not have been possible because  
 620 we would have no guarantee that the existential witnesses used in the two assumptions are the same.

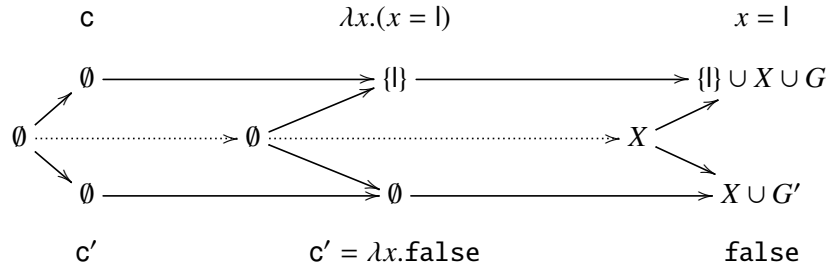
621 Indeed, we see the above observation as a possible contribution to addressing the often-vexing  
 622 problem that ‘logical relations don’t compose’ [PPST00].

623 The parametric s.v.f. and their natural transformations thus also interpret our language. Using a  
 624 realization relation  $\Vdash$  defined analogously one can then use this to deduce observational equivalence  
 625 as well. It can then be shown that the parametric interpretation validates all the equivalences from  
 626 Section 8 and in addition “Equivalence 12” above. This is because the two functions in question  
 627 will be equal above the appropriate span as explained in Example 9.6.

628 **9.3. Private Name Equation.** We now return to our motivating equivalence, illustrating that a  
 629 function value may encapsulate a freshly generated name without revealing it to the context:

$$c = (\text{let } n \leftarrow \text{new in } \lambda x.(x = n)) = (\lambda x.\text{false}) = c'$$

630 The equivalence proof is based on the following diagram:



631 We show that  $\mathbf{c}$  and  $\mathbf{c}'$  are equivalent in the trivial span to the left. For the generation of the fresh  
 632 value in  $\mathbf{c}$ , we choose the extension of the worlds with the fresh value  $l$ , the second span shown in the  
 633 diagram. Now it remains to prove that  $\lambda x.x = l$  and  $\mathbf{c}'$  are equivalent above the latter. This means that  
 634 for any extension of worlds,  $x = l$  and  $\mathbf{false}$  should be related. Consider the extension of worlds  
 635 in the right-most span in the diagram above. The names in  $X$  denote the common names, while  $G$   
 636 and  $G'$  the spurious names created. Notice that  $l$  is not in the low point of the third span because  
 637 the squares with vertices  $\emptyset, X, \{l\} \cup X \cup G, \{l\}$  and  $\emptyset, X, X \cup G', \emptyset$  are pullbacks as by Definition 9.1.  
 638 Thus, the value of  $x$  cannot be  $l$  and  $x = l$  is indeed equal to  $\mathbf{false}$ .

639

## 10. DISCUSSION

640 We have introduced proof-relevant logical relations and shown how they may be used to model and  
 641 reason about simple equivalences in a higher-order language with recursion and name generation.  
 642 A key innovation compared with previous functor category models is the use of functors valued in  
 643 setoids (which are here also built on predomains), rather than plain sets. One payoff is that we can  
 644 work with a direct style model rather than one based on continuations (which, in the absence of  
 645 control operators in the language, is less abstract).

646 The technical machinery used here is not *entirely* trivial, and the reader might be forgiven for  
 647 thinking it slightly excessive for such a simple language and rudimentary equations. However, our  
 648 aim has not been to present impressive new equivalences, but rather to present an accessible account  
 649 of how the idea of proof relevant logical relations works in a simple setting. The companion paper  
 650 [BHN14] gives significantly more advanced examples of applying the construction to reason about  
 651 equivalences justified by abstract semantic notions of effects and separation, but the way in which  
 652 setoids are used is there potentially obscured by the details of, for example, much more sophisticated  
 653 categories of worlds. Our hope is that this account will bring the idea to a wider audience, make  
 654 the more advanced applications more accessible, and inspire others to investigate the construction  
 655 in their own work.

656 Thanks to Andrew Kennedy for numerous discussions, and to an anonymous referee for sug-  
 657 gesting that we write up the details of how proof-relevance applies to pure name generation.

658

## REFERENCES

- 659 [AGM<sup>+</sup>04] S. Abramsky, D. R. Ghica, A. S. Murawski, C.-H. L. Ong, and I. D. B. Stark. Nominal games and full  
 660 abstraction for the nu-calculus. In *Proc. 19th Annual IEEE Symposium on Logic in Computer Science (LICS*  
 661 *'04)*. IEEE Computer Society, 2004.
- 662 [BB06] N. Bohr and L. Birkedal. Relational reasoning for recursive types and references. In *Proc. Fourth Asian*  
 663 *Symposium on Programming Languages and Systems (APLAS '06)*, volume 4279 of LNCS. Springer, 2006.
- 664 [BCP03] Gilles Barthe, Venanzio Capretta, and Olivier Pons. Setoids in type theory. *J. Funct. Program.*, 13(2):261–  
 665 293, 2003.
- 666 [BCRS98] Lars Birkedal, Aurelio Carboni, Giuseppe Rosolini, and Dana S. Scott. Type theory via exact categories. In  
 667 *Proc. 13th Annual IEEE Symposium on Logic in Computer Science (LICS '98)*. IEEE Computer Society,  
 668 1998.
- 669 [BÉ93] Stephen L. Bloom and Zoltán Ésik. *Iteration Theories - The Equational Logic of Iterative Processes*. EATCS  
 670 Monographs on Theoretical Computer Science. Springer, 1993.
- 671 [BHN14] Nick Benton, Martin Hofmann, and Vivek Nigam. Abstract effects and proof-relevant logical relations. In  
 672 *Proc. 41st ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL '14)*.  
 673 ACM, 2014.
- 674 [BK13] N. Benton and V. Koutavas. A mechanized bisimulation for the nu-calculus. *Higher-Order and Symbolic*  
 675 *Computation*, 2013. To appear.

- 676 [BKBH07] Nick Benton, Andrew Kennedy, Lennart Beringer, and Martin Hofmann. Relational semantics for effect-  
677 based program transformations with dynamic allocation. In *Proc. Ninth International ACM SIGPLAN Sym-*  
678 *posium on Principles and Practice of Declarative Programming (PPDP '07)*. ACM, 2007.
- 679 [BKHB06] Nick Benton, Andrew Kennedy, Martin Hofmann, and Lennart Beringer. Reading, writing and relations:  
680 Towards extensional semantics for effect analyses. In *Proc. Fourth Asian Symposium on Programming Lan-*  
681 *guages and Systems (APLAS '06)*, volume 4279 of *LNCS*. Springer, 2006.
- 682 [BL05] Nick Benton and Benjamin Leperchey. Relational reasoning in a nominal semantics for storage. In *Proc.*  
683 *Seventh International Conference on Typed Lambda Calculi and Applications (TLCA '05)*, volume 3461 of  
684 *LNCS*. Springer, 2005.
- 685 [CFS87] Aurelio Carboni, Peter J. Freyd, and Andre Scedrov. A categorical approach to realizability and polymorphic  
686 types. In *Proc. Third Workshop on Mathematical Foundations of Programming Language Semantics (MFPS*  
687 *'87)*, volume 298 of *LNCS*. Springer, 1987.
- 688 [GP02] Murdoch Gabbay and Andrew M. Pitts. A new approach to abstract syntax with variable binding. *Formal*  
689 *Asp. Comput.*, 13(3-5):341–363, 2002.
- 690 [PPST00] Gordon D. Plotkin, John Power, Donald Sannella, and Robert D. Tennent. Lax logical relations. In *27th*  
691 *International Colloquium on Automata, Languages and Programming (ICALP '00)*, pages 85–102, 2000.
- 692 [PS93] Andrew M. Pitts and Ian D. B. Stark. Observable properties of higher-order functions that dynamically cre-  
693 ate local names, or what's new? In *Proc. 18th International Symposium on Mathematical Foundations of*  
694 *Computer Science (MFCS '93)*, volume 711 of *LNCS*. Springer, 1993.
- 695 [PS98] Andrew Pitts and Ian Stark. Operational reasoning for functions with local state. In *Higher Order Operational*  
696 *Techniques in Semantics*, pages 227–273. Cambridge University Press, 1998.
- 697 [Shi04] Mark R. Shinwell. *The Fresh Approach: functional programming with names and binders*. PhD thesis, Uni-  
698 versity of Cambridge, 2004.
- 699 [SP00] Alex K. Simpson and Gordon D. Plotkin. Complete axioms for categorical fixed-point operators. In *LICS*,  
700 pages 30–41. IEEE Computer Society, 2000.
- 701 [SP05] Mark R. Shinwell and Andrew M. Pitts. On a monadic semantics for freshness. *Theor. Comput. Sci.*,  
702 342(1):28–55, 2005.
- 703 [SPG03] Mark R. Shinwell, Andrew M. Pitts, and Murdoch J. Gabbay. FreshML: Programming with binders made  
704 simple. In *Proc. Eighth ACM SIGPLAN International Conference on Functional programming (ICFP '03)*.  
705 ACM, 2003.
- 706 [Sta94] I. D. B. Stark. *Names and Higher-Order Functions*. PhD thesis, University of Cambridge, Cambridge, UK,  
707 December 1994. Also published as Technical Report 363, University of Cambridge Computer Laboratory.
- 708 [Tze12] N. Tzevelekos. Program equivalence in a simple language with state. *Computer Languages, Systems and*  
709 *Structures*, 38(2), 2012.