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 ν-calculus is a paradigmatic total language for studying the problem of contextual equivalence in higher-order languages with name generation. Models for the ν-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 ν-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.

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 ν-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 ν-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$^+$04, Tze12].

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.

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 generativity:

$$(\texttt{let}\; x \Leftarrow \texttt{new}\; \texttt{in}\; e) = e, \text{ provided } x \text{ is not free in } e. \qquad \text{(Drop)}$$
$$(\texttt{let}\; x \Leftarrow \texttt{new}\; \texttt{in}\; \texttt{let}\; y \Leftarrow \texttt{new}\; \texttt{in}\; e) = (\texttt{let}\; y \Leftarrow \texttt{new}\; \texttt{in}\; \texttt{let}\; x \Leftarrow \texttt{new}\; \texttt{in}\; e) \qquad \text{(Swap)}.$$

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

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

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

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

Section 1 sketches the language with which we will be working, and a naive 'raw' domain-theoretic semantics for it. This semantics does not validate interesting equivalences, but is adequate. By constructing a realizability relation between it and the more abstract semantics we subsequently

$$\frac{}{\Gamma, x : \tau \vdash_v x : \tau} \qquad \frac{}{\Gamma \vdash_v b : \texttt{bool}} \qquad \frac{}{\Gamma \vdash_v i : \texttt{int}} \qquad \frac{\Gamma, f : \tau \to \tau', x : \tau \vdash_c e : \tau'}{\Gamma \vdash_v \texttt{rec}\ f\ x = e : \tau \to \tau'}$$

$$\frac{\Gamma \vdash_v v : \texttt{int} \qquad \Gamma \vdash_v v' : \texttt{int}}{\Gamma \vdash_v v + v' : \texttt{int}} \qquad \frac{\Gamma \vdash_v v : \tau \qquad \Gamma \vdash_v v' : \tau \qquad \tau \in \{\texttt{int}, \texttt{name}\}}{\Gamma \vdash_v v = v' : \texttt{bool}} \qquad \frac{\Gamma \vdash_v v : \tau}{\Gamma \vdash_c v : \tau}$$

$$\frac{}{\Gamma \vdash_c \texttt{new} : \texttt{name}} \qquad \frac{\Gamma \vdash_c e : \tau \qquad \Gamma, x : \tau \vdash_c e' : \tau'}{\Gamma \vdash_c \texttt{let}\ x \Leftarrow e\ \texttt{in}\ e' : \tau'} \qquad \frac{\Gamma \vdash_v v : \tau \to \tau' \qquad \Gamma \vdash_v v' : \tau}{\Gamma \vdash_c v\, v' : \tau'}$$

$$\frac{\Gamma \vdash_v v : \texttt{bool} \qquad \Gamma \vdash_c e : \tau \qquad \Gamma \vdash_c e' : \tau}{\Gamma \vdash_c \texttt{if}\ v\ \texttt{then}\ e\ \texttt{else}\ e' : \tau}$$

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

introduce, we will be able to show adequacy of the more abstract semantics. In Section 2 we introduce our category of setoids; these are predomains where there is a (possibly-empty) set of ‘proofs’ witnessing the equality of each pair of elements. We then describe pullback-preserving functors from the category of worlds **W** into the category of setoids. Such functors will interpret types of our language in the more abstract semantics, with morphisms between them interpreting terms. The interesting construction here is that of a dynamic allocation monad over the category of pullback-preserving functors. Section 7 shows how the abstract semantics is defined and related to the more concrete one. Section 8 then shows how the semantics may be used to establish basic equivalences involving name generation. Section 9 describes how proof-relevant parametric functors can validate a more subtle equivalence involving encapsulation of new names.

1. Syntax and Semantics

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

$$\begin{array}{rcl}
\tau & := & \texttt{int} \mid \texttt{bool} \mid \texttt{name} \mid \tau \to \tau' \\
v & := & x \mid b \mid i \mid \texttt{rec}\ f\ x = e \mid v + v' \mid v = v' \\
e & := & v \mid \texttt{new} \mid \texttt{let}\ x \Leftarrow e\ \texttt{in}\ e' \mid v\, v' \mid \texttt{if}\ v\ \texttt{then}\ e\ \texttt{else}\ e' \\
\Gamma & := & x_1 : \tau_1, \ldots, x_n : \tau_n
\end{array}$$

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

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

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

We define a simple-minded concrete denotational semantics $\lceil\!\lceil \cdot \rceil\!\rceil$ for this language using predomains (ω-cpos) and continuous maps. For types we take

$$\begin{gathered}\lceil\!\lceil \mathtt{int} \rceil\!\rceil = \mathbb{Z} \qquad \lceil\!\lceil \mathtt{bool} \rceil\!\rceil = \mathbb{B} \qquad \lceil\!\lceil \mathtt{name} \rceil\!\rceil = \mathbb{N} \\ \lceil\!\lceil \tau \to \tau' \rceil\!\rceil = \lceil\!\lceil \tau \rceil\!\rceil \to (\mathbb{N} \to \mathbb{N} \times \lceil\!\lceil \tau' \rceil\!\rceil)_\perp \\ \lceil\!\lceil x_1 : \tau_1, \ldots, x_n : \tau_n \rceil\!\rceil = \lceil\!\lceil \tau_1 \rceil\!\rceil \times \cdots \times \lceil\!\lceil \tau_n \rceil\!\rceil\end{gathered}$$

and there are then conventional clauses defining

$$\lceil\!\lceil \Gamma \vdash_v v : \tau \rceil\!\rceil : \lceil\!\lceil \Gamma \rceil\!\rceil \to \lceil\!\lceil \tau \rceil\!\rceil \qquad \text{and} \qquad \lceil\!\lceil \Gamma \vdash_c e : \tau \rceil\!\rceil : \lceil\!\lceil \Gamma \rceil\!\rceil \to (\mathbb{N} \to \mathbb{N} \times \lceil\!\lceil \tau \rceil\!\rceil)_\perp$$

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

$$\lceil\!\lceil \Gamma \vdash_c \mathtt{new} \ : \mathtt{name} \rceil\!\rceil(\eta) = [\lambda n.(n+1, n)]$$

returning the next free name and incrementing the name supply. This semantics validates no interesting equivalences involving names, but is adequate for the obvious operational semantics. Our more abstract semantics, $[\![\cdot]\!]$, will be related to $\lceil\!\lceil \cdot \rceil\!\rceil$ in order to establish *its* adequacy.

2. Setoids

We define the *category of setoids*, *Std*, as the exact completion of the category of predomains, see [CFS87, BCRS98]. We give here an elementary description of this category using the language of dependent types. A *setoid* A consists of a predomain $|A|$ and for any two $x, y \in |A|$ a set $A(x, y)$ of "proofs" (that x and y are equal). The set of triples $\{(x, y, p) \mid p \in A(x, y)\}$ must itself be a predomain and the first and second projections must be continuous. Furthermore, there are continuous functions $r_A : \Pi x \in |A|.A(x, x)$ and $s_A : \Pi x, y \in |A|.A(x, y) \to A(y, x)$ and $t_A : \Pi x, y, z.A(x, y) \times A(y, z) \to A(x, z)$, witnessing reflexivity, symmetry and transitivity; note that, unlike the case of *groupoids*, no equations involving r, s and t are imposed.

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

The following is folklore, see also [BCRS98].

Proposition 2.1. The category of setoids is cartesian-closed. Cartesian product is given pointwise. The function space $A \Rightarrow B$ of setoids A and B is given as follows: the underlying cpo $|A \Rightarrow B|$ comprises pairs (f_0, f_1) which are *representatives* of morphisms from A to B. That is, $f_0 : |A| \to |B|$ and $f_1 : \Pi x, y \in |A|.A(x, y) \to B(f_0(x), f_0(y))$ are continuous functions with the pointwise ordering.

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) qua morphisms, *i.e.*, continuous functions $\mu : \Pi a \in |A|.B(f_0(a), g_0(a))$.

Proof. The evaluation morphism $(A \Rightarrow B) \times A \longrightarrow B$ sends (f_0, f_1) and a to $f_0(a)$. If $h : C \times A \longrightarrow B$ is a morphism represented by (h_0, h_1) then the morphism $\lambda(h) : C \longrightarrow A \Rightarrow B$ may 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 $f_1(a, a', p) = h_1((c, a), (c, a'), (r(c), p))$. Likewise, $\lambda(h)_1(c, c', p) = \mu$ where $\mu(a) = h_1((c, a), (c', a), (p, r(a)))$. The remaining verifications are left to the reader. □

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

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

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

$$\begin{array}{rcll}
f(Y(f)) & = & Y(f) & \text{(Fixpoint)} \\
f(Y(g \circ f)) & = & Y(f \circ g) & \text{(Dinaturality)} \\
f(Y(g)) & = & Y(h) \text{ if } f \text{ is strict and } fg = hf & \text{(Uniformity)} \\
Y(f^n) & = & Y(f) & \text{(Power)} \\
Y(\lambda x.f(x, x)) & = & Y(\lambda x.Y(\lambda y.f(x, y))) & \text{(Diagonal)} \\
Y(\lambda \vec{x}.\vec{t}(\vec{x})) & = & \langle Y(s), \dots, Y(s) \rangle & \text{(Amalgamation)} \\
\multicolumn{4}{l}{\text{when } t_i(y, \dots, y) = s(y) \text{ for } i = 1, \dots, n \text{ and } \vec{t} = \langle t_1, \dots, t_n \rangle}
\end{array}$$

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 define $d_i \in |D|$ by by $d_0 = \perp$ and $d_{i+1} = f_0(d_i)$. We then put $Y(f) = \sup_i d_i$.

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

Equations "Diagonal" and "Dinaturality" follow directly from the validity of these properties for the least fixpoint combinator for cpos. For the sake of completeness we prove the second one. 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 $f_0(e_i)) \leq d_{i+1}$. It follows that $Y(fg)$ and $Y(gf)$ are actually equal. Equation "Fixpoint" is a direct consequence of dinaturality (take $g = \mathrm{id}$).

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

Equation "Power", finally, can be deduced from amalgamation and dinaturality or alternatively inherited directly from the least fixpoint combinator. □

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

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

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

3. Finite sets and injections

Pullback squares are a central notion in our framework. As it will become clear later, they are the "proof-relevant" component of logical relations. Recall that a morphism u in a category is a monomorphism if $ux = ux'$ implies $x = x'$ for all morphisms x, x'. A commuting square $xu = x'u'$ of morphisms is a *pullback* if whenever $xv = x'v'$ there is unique t such that $v = ut$ and $v' = u't$. This can be visualized as follows:

$$\begin{tikzcd}
 & \overline{\mathsf{w}} & \\
\mathsf{w} \arrow[ur, "x"] & & \mathsf{w}' \arrow[ul, "x'"'] \\
 & \underline{\mathsf{w}} \arrow[ul, "u"] \arrow[ur, "u'"'] & \\
 & \cdot \arrow[u, dotted, "t"'] \arrow[uul, bend left, "v"] \arrow[uur, bend right, "v'"'] &
\end{tikzcd}$$

We write ${}^{x}_{u}\diamond^{x'}_{u'}$ or $\mathsf{w}{}^{x}_{u}\diamond^{x'}_{u'}\mathsf{w}'$ (when $\mathsf{w}^{(\prime)} = \mathrm{dom}(x^{(\prime)})$) for such a pullback square. We call the common codomain of x and x' the *apex* of the pullback, written $\overline{\mathsf{w}}$, while the common domain of u, u' is the *low point* of the square, written $\underline{\mathsf{w}}$. A pullback square $\mathsf{w}{}^{x}_{u}\diamond^{x'}_{u'}\mathsf{w}'$ with apex $\overline{\mathsf{w}}$ is *minimal* if whenever there is another pullback $\mathsf{w}{}^{x_1}_{u}\diamond^{x'_1}_{u'}\mathsf{w}'$ over the same span and with apex $\overline{\mathsf{w}_1}$, then there is a unique morphism $t : \overline{\mathsf{w}} \to \overline{\mathsf{w}_1}$ such that $x_1 = tx$ and $x'_1 = tx'$.

Lemma 3.1. If a category $\mathbf{C}$ has pullbacks and $\mathsf{w}{}^{x}_{u}\diamond^{x'}_{u'}\mathsf{w}'$ with apex $\overline{\mathsf{w}}$ is a *minimal* pullback then the morphisms x and x' are *jointly epic*, i.e. for any $f, g : \overline{\mathsf{w}} \to \mathsf{w}_1$, if $fx = gx$ and $fx' = gx'$, then $f = g$.

Proof. Let h be the equalizer of f, g. Both x and x' factor through h so write $x = hy$, $x' = hy'$. Now ${}^{y}_{u}\diamond^{y'}_{u'}$ is also a pullback and minimality shows that h is an isomorphism so $f = g$. $\square$

We are going to work with (pullback-preserving) functors from a category of worlds into the category of setoids. In our treatment of proof-relevant logical relations for reasoning about stateful computation [BHN14], we build a category of worlds from PERs on heaps. It is also possible to work more axiomatically, defining a category of worlds to be a category with pullbacks in which every span can be completed to a minimal pullback square, and all morphisms are monomorphisms. For the simple setting of this paper, however, we fix one particular instance:

Definition 3.2 (Category of worlds)**.** The *category of worlds* $\mathbf{W}$ has finite sets of natural numbers as objects and injective functions for morphisms.

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

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

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

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

$$\begin{tikzcd}
 & X'_1 \cup X'_2 & \\
X_1 \cong X'_1 \arrow[ur, "x"] & & X_2 \cong X'_2 \arrow[ul, "x'"'] \\
 & X'_1 \cap X'_2 \arrow[ul, "u"] \arrow[ur, "u'"'] &
\end{tikzcd}$$

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

4. Setoid-valued functors

A functor A from the category of worlds $\mathbf{W}$ to the category of setoids comprises as usual for each $\mathsf{w} \in \mathbf{W}$ a setoid $A\mathsf{w}$ and for each $u : \mathsf{w} \to \mathsf{w}'$ a morphism of setoids $Au : A\mathsf{w} \to A\mathsf{w}'$ preserving identities and composition. This means that there exist continuous functions of type $\Pi a.A\mathsf{w}(a, (Aid)\, a)$; and for any two morphisms $u : \mathsf{w} \to \mathsf{w}_1$ and $v : \mathsf{w}_1 \to \mathsf{w}_2$ a continuous function of type $\Pi a.A\mathsf{w}_2(Av(Au\, a), A(vu)\, a)$.

If $u : \mathsf{w} \to \mathsf{w}'$ and $a \in A\mathsf{w}$ we may write $u.a$ or even ua for $Au(a)$ and likewise for proofs in $A\mathsf{w}$. Note that there is a proof of equality of $(uv).a$ and $u.(v.a)$. In the sequel, we shall abbreviate 'setoid-valued functor(s)' as 's.v.f.'.

Intuitively, s.v.f. will become the denotations of value and computation types. Thus, an element of $A\mathsf{w}$ is a value involving at most the names in w. If $u : \mathsf{w} \to \mathsf{w}_1$ then $A\mathsf{w} \ni a \mapsto u.a \in A\mathsf{w}_1$ represents renaming and possible weakening by names not "actually" occurring in a. Note that due to the restriction to injective functions identification of names ("contraction") is precluded. This is in line with Stark's use of set-valued functors on the category $\mathbf{W}$ to model fresh names.

Definition 4.1. We call an s.v.f., A, *pullback-preserving* if for every pullback square $\mathsf{w}{}^{x}_{u}\diamond^{x'}_{u'}\mathsf{w}'$ with apex $\overline{\mathsf{w}}$ and low point $\underline{\mathsf{w}}$ the diagram $A\mathsf{w}{}^{Ax}_{Au}\diamond^{Ax'}_{Au'}A\mathsf{w}'$ is a pullback in *Std*. This means that there is a continuous function of type

$$\Pi a \in A\mathsf{w}.\Pi a' \in A\mathsf{w}'.A\overline{\mathsf{w}}(x.a, x'.a') \to \Sigma \underline{a} \in A\underline{\mathsf{w}}.A\mathsf{w}(u.\underline{a}, a) \times A\mathsf{w}'(u'.\underline{a}, a')$$

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

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

Lemma 4.2. If A is a s.v.f., $u : \mathsf{w} \to \mathsf{w}'$ and $a, a' \in A\mathsf{w}$, there is a continuous function $A\mathsf{w}'(u.a, u.a') \to A\mathsf{w}(a, a')$. Moreover, the "common ancestor" $\underline{a}$ of a and a' is unique up to $\sim$.

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

Definition 4.3 (Morphism of functors)**.** If A, B are s.v.f., a morphism from A to B is a pair $e = (e_0, e_1)$ of continuous functions where $e_0 : \Pi\mathsf{w}.A\mathsf{w} \to B\mathsf{w}$ and $e_1 : \Pi\mathsf{w}.\Pi\mathsf{w}'.\Pi x : \mathsf{w} \to \mathsf{w}'.\Pi a \in A\mathsf{w}.\Pi a' \in A\mathsf{w}'.A\mathsf{w}'(x.a, a') \to B\mathsf{w}'(x.e_0(a), e_0(a'))$. A proof that morphisms e, e' are equal is given by a continuous function $\mu : \Pi\mathsf{w}.\Pi a \in A\mathsf{w}.B\mathsf{w}(e(a), e'(a))$.

These morphisms compose in the obvious way and so the s.v.f. and morphisms between them form a category.

5. Instances of setoid-valued functors

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

5.1. Base types.

For each predomain D we can define a constant s.v.f., denoted D as well, with $D\mathsf{w}$ defined as the discrete setoid over D and Du as the identity. These constant s.v.f. serve as denotation of base types like booleans or integers.

The s.v.f. N of names is given by $N\mathsf{w} = \mathsf{w}$ where w on the right hand side stands for the discrete setoid over the discrete cpo of names in w, and $Nu = u$. Thus, e.g. $N\{1, 2, 3\} = \{1, 2, 3\}$.

5.2. Cartesian closure.

Let A and B be s.v.f. The product $A \times B$ is given by taking a pointwise product of setoids. For the sake of completeness, we note that $(A \times B)\mathsf{w} = A\mathsf{w} \times B\mathsf{w}$ (product predomain) and $(A \times B)\mathsf{w}((a, b), (a', b')) = A\mathsf{w}(a, a') \times B\mathsf{w}(b, b')$. This defines a cartesian product on the category of s.v.f. 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 empty indexed product and () for the only element of $1\mathsf{w}$. Note that 1 is the terminal object in the category of s.v.f.

The function space $A \Rightarrow B$ is the s.v.f. given as follows. $|(A \Rightarrow B)\mathsf{w}|$ contains pairs (f_0, f_1) where $f_0(u) \in |A\mathsf{w}_1 \Rightarrow B\mathsf{w}_1|$ for each w_1 and $u : \mathsf{w} \to \mathsf{w}_1$. If $u : \mathsf{w} \to \mathsf{w}_1$ and $v : \mathsf{w}_1 \to \mathsf{w}_2$ then

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

where

$$[Av \Rightarrow B\mathsf{w}_2] : (A\mathsf{w}_2 \Rightarrow B\mathsf{w}_2) \to (A\mathsf{w}_1 \Rightarrow B\mathsf{w}_2)$$
$$[A\mathsf{w}_1 \Rightarrow Bv] : (A\mathsf{w}_1 \Rightarrow B\mathsf{w}_1) \to (A\mathsf{w}_1 \Rightarrow B\mathsf{w}_2)$$

are the obvious composition morphisms.

A proof in $(A \Rightarrow B)\mathsf{w}((f_0, f_1), (f_0', f_1'))$ is a function g that for each $u : \mathsf{w} \to \mathsf{w}_1$ yields a proof $g(u) \in (A\mathsf{w}_1 \Rightarrow B\mathsf{w}_1)(f_0(u), f_0'(u))$.

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

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

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

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

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

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

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

The laws from Theorem 2.3 can be directly inherited. $\square$

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

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

6. Dynamic Allocation Monad

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

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

6.1. Definition of the monad.

Let A be a s.v.f., then we put

$$|TA\mathsf{w}| = \{(\mathsf{w}_1, a) \mid \mathsf{w} \subseteq \mathsf{w}_1 \wedge a \in A\mathsf{w}_1\}_\perp$$

Thus, an element of $TA\mathsf{w}$ consists of an extension of w together with an element of A taken at that extension. Note that the extension is not existentially quantified but part of the element.

The ordering is given by $(\mathsf{w}_1, a) \leq (\mathsf{w}'_1, a')$ if $\mathsf{w}_1 = \mathsf{w}'_1$ and $a \leq a'$ in $A\mathsf{w}_1$ and of course, $\perp$ is the least element of $TA\mathsf{w}$.

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

$$\begin{tikzcd}
 & \overline{\mathsf{w}} & \\
\mathsf{w}_1 \arrow[ur, "x"] & & \mathsf{w}'_1 \arrow[ul, "x'"'] \\
 & \mathsf{w} \arrow[ul, hook, "u"] \arrow[ur, hook', "u'"'] &
\end{tikzcd}$$

with $\overline{\mathsf{w}} = \mathrm{cod}(x) = \mathrm{cod}(x')$. The third component p then is a proof that a and a' are equal when transported to $\overline{\mathsf{w}}$, formally, $p \in A\overline{\mathsf{w}}(x.a, x'.a')$. The ordering is again discrete in x, x' and inherited from A in p. Formally, $((\mathsf{w}_1, a), (\mathsf{w}'_1, a'), (x, x', p)) \le ((\mathsf{w}_1, b), (\mathsf{w}'_1, b'), (x, x', q))$ when $(x.a, x'.a', p) \le (x.b, x'.b, q)$ in $A\mathrm{cod}(x)$ and of course $(\bot, \bot, \bot)$ is the least element. No $\le$-relation exists between triples with different mediating co-span.

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

Next, we define the morphism part. Assume that $u : \mathsf{w} \to \mathsf{q}$ is a morphism in $\mathbf{W}$. We want to construct a morphism $Au : TA\mathsf{w} \to TA\mathsf{q}$ in *Std*. So let $(\mathsf{w}_1, a) \in TA\mathsf{w}$ and $i : \mathsf{w} \hookrightarrow \mathsf{w}_1$ be the inclusion. We complete the span i, u to a minimal pullback

$$\begin{tikzcd}
\mathsf{w}_1 \arrow[r, "u_1"] & \mathsf{q}_1 \\
\mathsf{w} \arrow[u, hook, "i"] \arrow[r, "u"'] & \mathsf{q} \arrow[u, hook, "j"']
\end{tikzcd}$$

with j an inclusion as indicated. We then define $TAu(\mathsf{w}_1, a) = (\mathsf{q}_1, u_1.a)$. It is important that the square above is a pullback; minimality is not strictly necessary, but simplifies subsequent definitions. We assume a function that returns such completions to minimal pullbacks in some chosen way. The particular choice is unimportant.

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

$$\begin{tikzcd}
\mathsf{w}_1 = \{0,1,2\} \arrow[r, "0\mapsto1{,}1\mapsto2{,}2\mapsto3"] & \{0,1,2,3\} = \mathsf{q}_1 \\
\mathsf{w} = \{0\} \arrow[u, hook, "i"] \arrow[r, "0\mapsto1"'] & \{0,1\} = \mathsf{q} \arrow[u, hook, "j"']
\end{tikzcd}$$

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

$$\begin{tikzcd}
\mathsf{w}_1 = \{0,1,2\} \arrow[r, "0\mapsto1{,}1\mapsto0{,}2\mapsto3"] & \{0,1,2,3\} \\
\mathsf{w} = \{0\} \arrow[u, hook, "i"] \arrow[r, "0\mapsto1"'] & \{0,1\} = \mathsf{q} \arrow[u, hook]
\end{tikzcd}$$

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

$$\begin{tikzcd}
\mathsf{w}_1 = \{0,1,2\} \arrow[r, "0\mapsto 1, 1\mapsto 2, 2\mapsto 3"] & \{0,1,2,3,4,5\} \\
\mathsf{w} = \{0\} \arrow[u, hook, "i"] \arrow[r, "0\mapsto 1"'] & \{0,1\} = \mathsf{q} \arrow[u, hook]
\end{tikzcd}$$

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

$$\begin{tikzcd}
& \overline{\mathsf{w}} \arrow[rr, "\overline{u}"] & & \overline{\mathsf{q}} & \\
& & \mathsf{w}'_1 \arrow[ul, "x'"'] \arrow[rr, "u'_1"] & & \mathsf{q}'_1 \arrow[ul, "y'"'] \\
\mathsf{w}_1 \arrow[uur, "x"] \arrow[rrr, "u_1"'] & & & \mathsf{q}_1 \arrow[uu, "y"'] & \\
\mathsf{w} \arrow[u, hook, "i"] \arrow[uurr, hook, "i'"] \arrow[rrr, "u"'] & & & \mathsf{q} \arrow[u, hook, "j"] \arrow[uur, hook, "j'"'] &
\end{tikzcd}$$

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

Proposition 6.1. T is a strong monad on the category of s.v.f. The unit sends $\mathsf{v} \in A\mathsf{w}$ to $(\mathsf{w}, \mathsf{v}) \in (TA)\mathsf{w}$. The multiplication sends $(\mathsf{w}_1, (\mathsf{w}_2, v)) \in (TTA)\mathsf{w}$ to $(\mathsf{w}_2, v) \in TA\mathsf{w}$. The strength map sends $(a, (\mathsf{w}_1, b)) \in (A \times TB)\mathsf{w}$ to $(\mathsf{w}_1, (a.i, b))$ where $i : \mathsf{w} \hookrightarrow \mathsf{w}_1$.

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

Given a cpo-valued functor A one may construct a poset-valued functor $TA\mathsf{w}$ which has for underlying set equivalence classes of pairs (w_1, a) as in Stark's definition above. As for the ordering, the only reasonable choice is to decree that on representatives $(\mathsf{w}_1, a) \leq (\mathsf{w}'_1, a')$ if $x.a \leq x'.a'$ for some co-span x, x' with $xi = x'i'$ where i, i' are the inclusions as above. However, while this defines a partial order it is not clear why it should have suprema of ascending chains because there is no reason why these witnessing spans should match up so that they can be pasted to a witnessing span for the limit of the chain. Indeed, Shinwell's thesis [Shi04] contains a concrete counterexample.

The transition to proof relevance that we have made allows us to define the order on representatives as we have done and thus to bypass these difficulties.

7. Observational Equivalence and Fundamental Lemma

We now construct the machinery that connects the concrete language with the denotational machinery introduced in Section 1. In particular, we define the semantics of types, written using $[\![\cdot]\!]$, as s.v.f. inductively as follows:

- For basic types $[\![\tau]\!]$ is the corresponding discrete s.v.f..
- $[\![\tau \to \tau']\!]$ is defined as the function space $[\![\tau]\!] \to T[\![\tau]\!]$, where T is the dynamic allocation monad.
- For typing context Γ we define $[\![\Gamma]\!]$ as the indexed product of s.v.f. $\prod_{x\in\mathrm{dom}(\Gamma)}[\![\Gamma(x)]\!]$.

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

$$[\![\mathtt{new}\,]\!]\mathsf{w} = (\mathsf{w} \cup \{n+1\}, n+1)$$

where $n = \max(\mathsf{w})$, *i.e.*, the greatest number in the world w.

If $f_1 : [\![\Gamma]\!] \to T[\![\tau_1]\!]$ and $f_2 : [\![\Gamma, x{:}\tau_1]\!] \to T[\![\tau_2]\!]$ are the denotations of $\Gamma \vdash e_1 : \tau_1$ and $\Gamma, x{:}\tau_2 \vdash e_2 : \tau_2$ then the interpretation of $\mathtt{let}\, x {\Leftarrow} e_1 \,\mathtt{in}\, e_2$ is the morphism $f : [\![\Gamma]\!] \to T[\![\tau_2]\!]$ given by

$$f = \mu \circ T f_2 \circ \sigma \circ \langle id_\Gamma, f_1\rangle$$

where μ is the monad multiplication, σ is the monad strength and where we have made the simplifying assumption that $[\![\Gamma, x{:}\tau]\!] = [\![\Gamma]\!] \times [\![\tau]\!]$. Assuming that f_1 and f_2 now stand for the first components of concrete representatives of these morphisms one particular concrete representative of this morphism (now also denoted f) satisfies:

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

Our aim is now to relate these morphisms to the computational interpretation $\lceil\!\lceil e \rceil\!\rceil$.

Definition 7.1. For each type τ and world w we define a relation $\Vdash^\tau_{\mathsf{w}} \subseteq \lceil\!\lceil \tau \rceil\!\rceil \times [\![\tau]\!]\mathsf{w}$ by the following clauses. We write $\max(\mathsf{w}) = \max(\{n \mid n \in \mathsf{w}\})$ so that, in particular, $\max(\mathsf{w}) + 1 \notin \mathsf{w}$.

$$\begin{array}{l}
b \Vdash^{\mathtt{bool}}_{\mathsf{w}} \mathsf{b} \iff b = \mathsf{b} \\
i \Vdash^{\mathtt{int}}_{\mathsf{w}} \mathsf{i} \iff i = \mathsf{i} \\
l \Vdash^{\mathtt{name}}_{\mathsf{w}} k \iff l = k \\
f \Vdash^{\tau\to\tau'}_{\mathsf{w}} g \iff \forall \mathsf{w}_1 \supseteq \mathsf{w}.\forall v.\forall \mathsf{v}. v \Vdash^\tau_{\mathsf{w}_1} \mathsf{v} \Rightarrow f(v) \Vdash^{\tau'}_{\mathsf{w}_1} g_0(\mathsf{w} \hookrightarrow \mathsf{w}_1, \mathsf{v}) \\
c \Vdash^{T\tau}_{\mathsf{w}} \mathsf{c} \iff \begin{array}{l} [c(\max(\mathsf{w})+1) = \bot \Leftrightarrow \mathsf{c} = \bot] \wedge \\ [c(\max(\mathsf{w})+1) = (n_1, v) \wedge \mathsf{c} = (\mathsf{w}_1, \mathsf{v}) \Rightarrow n_1 = \max(\mathsf{w}_1) + 1 \wedge v \Vdash^\tau_{\mathsf{w}_1} \mathsf{v}))]. \end{array}
\end{array}$$

The following lemma states that the realizability relation is stable with respect to enlargement of worlds for value types. It is needed for the "fundamental lemma" 7.3. The proof is a straightforward induction on types.

Lemma 7.2. Let τ be a value type, *i.e.*, $\mathtt{bool}$, $\mathtt{int}$, or function type, $\tau_1 \to \tau_2$. If $u : \mathsf{w} \hookrightarrow \mathsf{w}_1$ is an inclusion as indicated and $v \Vdash^\tau_{\mathsf{w}} \mathsf{v}$ then $v \Vdash^\tau_{\mathsf{w}_1} u.\mathsf{v}$, too.

We extend $\Vdash$ to typing contexts by putting

$$\eta \Vdash^\Gamma_{\mathsf{w}} \gamma \iff \forall x \in \mathrm{dom}(\Gamma).\eta(x) \Vdash^{\Gamma(x)}_{\mathsf{w}} \gamma(x)$$

for $\eta \in \lceil\!\lceil \Gamma \rceil\!\rceil$ and $\gamma \in [\![\Gamma]\!]$.

Theorem 7.3 (Fundamental lemma)**.** Let $\Gamma \vdash e : \tau$ be a well typed term. There exists a representative $(\mathsf{c}, \_)$ of the equivalence class $[\![e]\!]$ at world w such that if $\eta \Vdash^\Gamma_{\mathsf{w}} \gamma$ then $\lceil\!\lceil e \rceil\!\rceil\eta \Vdash^{T\tau}_{\mathsf{w}} \mathsf{c}(\gamma)$.

Proof. By induction on typing rules. We always chose for the representative the one given as witness in the definition of $[\![e]\!]$. Most of the cases are straightforward. For illustration we show `new` and `let` : As for `new` , we pick the representative c that at world w returns $(\mathsf{w} \cup \{\max(\mathsf{w}) + 1\}, \max(\mathsf{w}))$.

Now, with $c = \llbracket \mathtt{new} \rrbracket$, we have $c(\max(\mathsf{w})) = (\max(\mathsf{w}) + 1, \max(\mathsf{w}))$ and $c \Vdash^{TN}_{\mathsf{w}} \mathsf{c}$ holds, since $\max(\mathsf{w} \cup \{\max(\mathsf{w}) + 1\}) = \max(\mathsf{w}) + 1$.

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

By Lemma 7.2 we then have $\eta \Vdash^{\Gamma}_{\mathsf{w}_1} i.\gamma$ where $i : \mathsf{w} \hookrightarrow \mathsf{w}_1$. Thus, by induction hypothesis, we get (H2) $\llbracket e_2 \rrbracket (\eta[x \mapsto v]) \Vdash^{T\tau_2}_{\mathsf{w}_1} \mathsf{c}_2(i.\gamma[x \mapsto \mathsf{v}])$. Thus, putting $\mathsf{c}(\gamma) = \mathsf{c}_2(i.\gamma, \mathsf{v})$ furnishes the required representative of $\llbracket \mathtt{let}\ x \Leftarrow e_1\ \mathtt{in}\ e_2 \rrbracket(\mathsf{w})$ □

Remark 7.4. Note that the particular choice of representative matters here. For example, if $\mathsf{c}_0\mathsf{w} = (\mathsf{w} \uplus \{\max(\mathsf{w}) + 1, \max(\mathsf{w}) + 2\}, \max(\mathsf{w}) + 1)$ then there exists c_1 such that $(\mathsf{c}_0, \mathsf{c}_1) : 1 \to TN$ and $(\mathsf{c}_0, \mathsf{c}_1)$ and $\llbracket \mathtt{new} \rrbracket$ are equal qua morphisms of s.v.f. Yet, $\llbracket \mathtt{new} \rrbracket \nVdash^{TN}_{\mathsf{w}} \mathsf{c}_0$.

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

It is now possible to validate a number of equational rules on the level of the setoid semantics $\llbracket - \rrbracket$ including transitivity, $\beta\eta$, fixpoint unrolling, and congruence rules. We omit the definition of such an equational theory here and refer to [BHN14] for details on how this could be set up. As we now show, equality on the level of the setoid semantics entails observational equivalence on the level of the raw denotational semantics.

7.1. Observational Equivalence.

Definition 7.5. Let τ be a type. We define an *observation of type* τ as a closed term $\vdash o : \tau \to \mathtt{bool}$. Two values $v, v' \in \llbracket \tau \rrbracket$ are *observationally equivalent at type* τ if for all observations o of type τ one has that $\llbracket o \rrbracket(v)(0)$ is defined iff $\llbracket o \rrbracket(v')(0)$ is defined and when $\llbracket o \rrbracket(v)(0) = (n_1, v_1)$ and $\llbracket o \rrbracket(v')(0) = (n'_1, v'_1)$ then $v_1 = v'_1$.

Note that observational equivalence is a congruence since an observation can be extended by any englobing context. We also note that observational equivalence is the coarsest reasonable congruence. Identifying two terms that are not observationally equivalent is clearly unsound in any reasonable sense.

We now show how the proof-relevant semantics can be used to deduce observational equivalences.

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

Proof. Let o be an observation at type τ. By the Fundamental Lemma (Theorem 7.3) we have $\llbracket o \rrbracket \Vdash^{\tau \to \mathtt{bool}}_{\emptyset} \llbracket o \rrbracket$.

Now, since $e \sim e'$ we also have $\llbracket o \rrbracket(e) \sim \llbracket o \rrbracket(e')$ and, of course, $\llbracket o \rrbracket(v) \Vdash^{T\mathtt{bool}}_{\emptyset} \llbracket o \rrbracket(e)$ and $\llbracket o \rrbracket(v') \Vdash^{T\mathtt{bool}}_{\emptyset} \llbracket o \rrbracket(e')$.

From $[\![o]\!](e) \sim [\![o]\!](e')$ we conclude that either $[\![o]\!](e)(0)$ and $[\![o]\!](e')(0)$ both diverge in which case the same is true for $\lceil\!\lceil o \rceil\!\rceil(v)(0)$ and $\lceil\!\lceil o \rceil\!\rceil(v')(0)$ by definition of $\Vdash^{T\tau}$. Secondly, if $[\![o]\!](e)(0)) = (\_, \_, b, \_)$ and $[\![o]\!](e')(0)) = (\_, \_, b', \_)$ for booleans b, b' then, by definition of $\sim$ at $[\![T\tau]\!]$ we get $b = b'$ and, again by definition of $\Vdash^{T\tau}$ this then implies that $\lceil\!\lceil o \rceil\!\rceil(v)(0) = (\_, b)$ and $\lceil\!\lceil o \rceil\!\rceil(v)(0) = (\_, b')$ with $b = b'$, hence the claim. $\square$

8. Direct-Style Proofs

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

8.1. Drop equation.

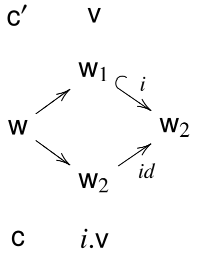
Drop equation. We start with the following equation, which eliminates a dummy allocation:

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

Suppose that $c' \Vdash_{\mathsf{w}}^{\Gamma \vdash TA} \mathsf{c}'$ and that $\mathsf{c}'(\gamma) = (\mathsf{w}_1, \mathsf{v})$. We provide a semantic computation c, such that $c \Vdash_{\mathsf{w}}^{\Gamma \vdash TA} \mathsf{c}$, that is, it is related to the computation that performs a dummy allocation, and we also provide a proof $\mathsf{c}(\gamma) \sim \mathsf{c}'(\gamma) \in TA\mathsf{w}$. From Theorem 7.6, this means that the two computations are observationally equivalent.

Let $\mathsf{c}(\gamma) = (\mathsf{w}_2, i.\mathsf{v})$, where $\mathsf{w}_2 = \mathsf{w}_1 \cup \{\max(\mathsf{w}_1) + 1\}$ and $i : \mathsf{w}_1 \hookrightarrow \mathsf{w}_2$. It is easy to check that indeed $c \Vdash_{\mathsf{w}}^{\Gamma \vdash TA} \mathsf{c}$: the world extension w_2 accounts for the creation of the dummy fresh name, namely, the value $\max(\mathsf{w}_1) + 1$. Moreover, since c does not use the dummy fresh value, the semantic value returned should be $i.\mathsf{v}$.

Now it remains to show that $\mathsf{c} \sim \mathsf{c}' \in TA\mathsf{w}$, that is, a proof that the semantic computations are equal. Consider the following pullback square, where the annotations above and below the square are just to illustrate in which world the semantic values are:


$$\begin{tikzcd}
\mathsf{c}' & \mathsf{v} & \\
& \mathsf{w}_1 \arrow[dr, hook, "i"] & \\
\mathsf{w} \arrow[ur] \arrow[dr] & & \mathsf{w}_2 \\
& \mathsf{w}_2 \arrow[ur, "id"'] & \\
\mathsf{c} & i.\mathsf{v} &
\end{tikzcd}$$

Clearly we have $i.\mathsf{v} \sim id.i.\mathsf{v}$ and therefore the pullback above is a proof that $\mathsf{c} \sim \mathsf{c}' \in TA\mathsf{w}$.

8.2. Swap equation.

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

$$c = (\texttt{let}\ x \Leftarrow \texttt{new}\ \ \texttt{in}\ \texttt{let}\ y \Leftarrow \texttt{new}\ \ \texttt{in}\ e) = (\texttt{let}\ y \Leftarrow \texttt{new}\ \ \texttt{in}\ \texttt{let}\ x \Leftarrow \texttt{new}\ \ \texttt{in}\ e) = c'.$$

For showing that these programs are equivalent, we will need to consider world advancements. Assume that we start from an identity pullback square with the empty world $\emptyset$. Moreover, consider the abstract computations $\mathsf{c}_1 = (\{\mathsf{l}_1\}, c_2)$ and $\mathsf{c}_1' = (\{\mathsf{l}_1'\}, c_2')$, where l_1 and l_1' are the first proper concrete locations allocated. Moreover, let $c_2 = (\{\mathsf{l}_1, \mathsf{l}_2\}, c)$ and $c_2' = (\{\mathsf{l}_1', \mathsf{l}_2'\}, c')$, where the second location is allocated. The proof is now the pullback square $\{\mathsf{l}_1, \mathsf{l}_2\}_{u_2 u_1}{}^{id}\!\diamond^{x'}_{u_2' u_1'}\{\mathsf{l}_1', \mathsf{l}_2'\}$, with $\overline{\mathsf{w}} = \{\mathsf{k}_1, \mathsf{k}_2\}$

and where x fixes everything except that it maps l_1 (respectively, l_2') to k_1 and l_2 (respectively, l_1') to k_2, as illustrated by the following diagram.

$$\begin{tikzcd}
 & \mathsf{c} & & & \texttt{let}\, y \Leftarrow \texttt{new in}\, e & & & e & \\
 & \emptyset \arrow[rrr, "u_1"] \arrow[dr] & & & \{\mathsf{l}_1\} \arrow[rrr, "u_2"] \arrow[dr] & & & \{\mathsf{l}_1, \mathsf{l}_2\} \arrow[dr, "x"] & \\
\emptyset \arrow[ur] \arrow[dr] & & \emptyset & \emptyset \arrow[ur] \arrow[dr] & & \{\mathsf{l}_1, \mathsf{l}_1'\} & \emptyset \arrow[ur] \arrow[dr] & & \{\mathsf{k}_1, \mathsf{k}_2\} \\
 & \emptyset \arrow[rrr, "u_1'"'] \arrow[ur] & & & \{\mathsf{l}_1'\} \arrow[rrr, "u_2'"'] \arrow[ur] & & & \{\mathsf{l}_1', \mathsf{l}_2'\} \arrow[ur, "x'"'] & \\
 & \mathsf{c}' & & & \texttt{let}\, x \Leftarrow \texttt{new in}\, e & & & e &
\end{tikzcd}$$

where $\mathsf{k}_1, \mathsf{k}_2$ are new concrete locations. In this way we get that $id.\mathsf{c} \sim x'.\mathsf{c}'$.

9. Proof-relevant parametric functors

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

$$(\texttt{let}\, n \Leftarrow \texttt{new in fun}\, x.x = n) = (\texttt{fun}\, x.\texttt{false}).$$

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

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

It is possible to obtain Stark's parametric functors automatically by interpreting the general categorical definition as internal to the category of setoids and proof-relevant relations. This, however, would lead to some redundancy because the proof-relevant equality is equivalent to relatedness "above" the identity relation (identity extension). We thus give an ad-hoc definition that contains less data. Moreover, the inclusion of reflexivity, symmetry, and transitivity, for relatedness "above" a partial bijection is new.

We use capital letters $S, S', \ldots$ for spans of worlds. If S is the span $\mathsf{w} \xleftarrow{u} \underline{\mathsf{w}} \xrightarrow{u'} \mathsf{w}'$ then we use the notations $S : \mathsf{w} \leftrightarrow \mathsf{w}'$ and $\mathsf{w} = \mathrm{dom}(S)$ (left domain), $\mathsf{w}' = \mathrm{dom}'(S)$ (right domain), $\underline{\mathsf{w}} = \mathrm{lop}(S)$ (low point), $u = S.u$, $u' = S.u'$. For world w we denote $r(\mathsf{w}) : \mathsf{w} \leftrightarrow \mathsf{w}$ the identity span $\mathsf{w} \xleftarrow{1} \mathsf{w} \xrightarrow{1} \mathsf{w}$. If $S : \mathsf{w} \leftrightarrow \mathsf{w}'$ then $s(S) : \mathsf{w}' \leftrightarrow \mathsf{w}$ is given by $\mathsf{w}' \xleftarrow{S.u'} \mathrm{lop}(S) \xrightarrow{S.u} \mathsf{w}$. If $S : \mathsf{w} \leftrightarrow \mathsf{w}'$ and $S' : \mathsf{w}' \leftrightarrow \mathsf{w}''$ then we define $t(S, S') : \mathsf{w} \leftrightarrow \mathsf{w}''$ as $\mathsf{w} \xleftarrow{S.u\ x} . \xrightarrow{S'.u'\ x'} \mathsf{w}''$ where x, x' complete $S.u'$ and $S'.u$ to a pullback square.

$$\begin{tikzcd}
\mathsf{w} & & \mathsf{w}' & & \mathsf{w}'' \\
& \underline{\mathsf{w}_S} \arrow[ul] \arrow[ur] & & \underline{\mathsf{w}_{S'}} \arrow[ul] \arrow[ur] & \\
& & \underline{\mathsf{w}_0} \arrow[ul, dotted] \arrow[ur, dotted] & &
\end{tikzcd}$$

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

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

$$\begin{tikzcd}
& \mathsf{w} \arrow[rr, "u"] & & \mathsf{w}_1 \\
\underline{\mathsf{w}_S} \arrow[ur] \arrow[dr] \arrow[rr, dotted, "\underline{u}"'] & & \underline{\mathsf{w}_{S_1}} \arrow[ur] \arrow[dr] & \\
& \mathsf{w}' \arrow[rr, "u'"] & & \mathsf{w}'_1
\end{tikzcd}$$

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

Note that the witnessing morphism $\underline{u}$ is uniquely determined since it we can complete S' to a pullback in which case $\underline{u}$ is the unique mediating morphism.

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

$$\begin{tikzcd}
& \{0,1,2\} \arrow[rr, "u"] & & \{0,1,2,3\} \\
\{0,1\} \arrow[ur] \arrow[dr, "{[0\mapsto 1, 1\mapsto 2]}"'] & & \{0,1,2\} \arrow[ur, "{[0\mapsto 0, 1\mapsto 1, 2\mapsto 3]}"] \arrow[dr, "{[0\mapsto 1, 1\mapsto 2, 2\mapsto 3]}"] & \\
& \{0,1,2\} \arrow[rr, "u'"] & & \{0,1,2,3\}
\end{tikzcd}
\qquad
\begin{tikzcd}
& \{0,1,2\} \arrow[rr, "u"] & & \{0,1,2,3\} \\
\{0,1\} \arrow[ur] \arrow[dr, "{[0\mapsto 1, 1\mapsto 2]}"'] & & \{0,1,2\} \arrow[ur, "{[0\mapsto 0, 1\mapsto 1, 2\mapsto 2]}"] \arrow[dr, "{[0\mapsto 1, 1\mapsto 2, 2\mapsto 3]}"] & \\
& \{0,1,2\} \arrow[rr, "u'"] & & \{0,1,2,3\}
\end{tikzcd}$$

We also note that if $S, S' : \mathsf{w} \leftrightarrow \mathsf{w}'$ then $(1, 1) : S \to S'$ is a parametric square if and only if there exists an isomorphism $t : \mathrm{lop}(S) \to \mathrm{lop}(S)'$ such that $S'.u\,t = S.u$ and $S'.u'\,t = S.u'$.

Definition 9.2. A parametric s.v.f. A consists of the following data.

(1) For each world w a cpo $A\mathsf{w}$.
(2) For each $u : \mathsf{w} \to \mathsf{w}'$ a continuous function $Au : A\mathsf{w} \to A\mathsf{w}'$. We use the notation $u.a = Au(a)$.
(3) For each span $S : \mathsf{w} \leftrightarrow \mathsf{w}'$ and $a \in A\mathsf{w}$ and $a' \in A\mathsf{w}'$ a set $AS(a, a')$ such that the set of quadruples (S, a, a', p) with $p \in AS(a, a')$ is a cpo with continuous second and third projections and discrete ordering in the first component.
(4) For each parametric square $(u, u') : S \to S'$ a continuous function

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

(5) For each parametric square $(1,1) : S \to S'$ a continuous function

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

(6) Continuous functions of the following types:

$$\begin{aligned}&\Pi \mathsf{w}.\Pi a \in A\mathsf{w}.Ar(\mathsf{w})(a,a)\\ &\Pi S.\Pi a \in \mathrm{Adom}(S).\Pi a' \in \mathrm{Adom}(S).AS(a,a') \to As(S)(a',a)\\ &\Pi \mathsf{w}\,\mathsf{w}'\,\mathsf{w}''.\Pi S : \mathsf{w} \leftrightarrow \mathsf{w}'.\Pi S' : \mathsf{w}' \leftrightarrow \mathsf{w}''.\Pi a \in A\mathsf{w}.\Pi a' \in A\mathsf{w}'.\Pi a'' \in A\mathsf{w}''.\\ &\quad AS(a,a') \times AS'(a',a'') \to At(S,S')(a,a'')\end{aligned}$$

(7) Continuous functions of the following types:

$$\begin{aligned}&\Pi \mathsf{w}.\Pi a \in A\mathsf{w}.Ar(\mathsf{w})(a,1.a)\\ &\Pi \mathsf{w}\,\mathsf{w}_1\,\mathsf{w}_2.\Pi u : \mathsf{w} \to \mathsf{w}_1.\Pi v : \mathsf{w}_1 \to \mathsf{w}_2.Ar(\mathsf{w}_2)(v.u.a,(vu).a)\end{aligned}$$

Suppose that $S,S' : \mathsf{w} \leftrightarrow \mathsf{w}'$ are isomorphic spans between w and w' in the sense that $(1,1) : S \to S'$ (see the remark just before Def. 9.2). If (we have an element of) $AS(a,a')$ then $A(1,1)(a,a') \in AS'(1.a,1.a')$ which is almost but not quite as good as $A(S,S')(a,a') \in AS'(a,a')$. Indeed, from $AS'(1.a,1.a)$ we even get $At(r(\mathsf{w}),t(S',s(r(\mathsf{w}))))(a,a')$, but without explicitly postulating it as we do or making extra assumptions on the $t(-,-)$ or $1.-$ operations it seems impossible to reach $AS'(a,a')$.

Lemma 9.3. Let A be a parametric s.v.f. For each $a \in A\mathsf{w}$ and $u : \mathsf{w} \to \mathsf{w}_1$ we have $AS(a,u.a)$ where S is $\mathsf{w} \xleftarrow{1} \mathsf{w} \xrightarrow{u} \mathsf{w}_1$.

Proof. We use the parametric square $(1,u) : S_0 \to S$ where S_0 is $\mathsf{w} \xleftarrow{1} \mathsf{w} \xrightarrow{1} \mathsf{w}$. □

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

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

However, this correspondence is not one-to-one. For a concrete counterexample, consider the following example which also lies at the heart of the justification of "Equivalence 12" with parametric functors.

Example 9.4. The parametric s.v.f. $[N{\Rightarrow}B]$ is defined by $[N{\Rightarrow}B]\mathsf{w} = 2^{\mathsf{w}}$ (functions from w to $\{\mathtt{true},\mathtt{false}\}$) and

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

Now, let S be $\{0\} \leftarrow \emptyset \to \emptyset$ and put $f(x) =$"x=0" and $f'(x) = \mathtt{false}$. We have $[N{\Rightarrow}B]S(f,f')$, *i.e.*, $[N{\Rightarrow}B]S(f,f') = \{\star\}$, thus f and f' are considered equal above span S. On the other hand, if we 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' are not equal when regarded over the least common world, namely $\{0\}$.

Definition 9.5. A parametric natural transformation from parametric s.v.f. A to B consists of two continuous functions

$$\begin{aligned}&f : \Pi \mathsf{w}.A\mathsf{w} \to B\mathsf{w}\\ &f : \Pi S.\Pi a \in \mathrm{dom}(S).\Pi a' : \mathrm{dom}'(S).AS(a,a') \to BS(f\mathrm{dom}(S)(a), f\mathrm{dom}'(S)(a'))\end{aligned}$$

Two parametric natural transformations $f, f' : A \to B$ are identified if there is a continuous function of type

$$\Pi\mathsf{w}.\Pi a \in A\mathsf{w}.Ar(\mathsf{w})(f\mathsf{w}(a), f'\mathsf{w}(a))$$

The identification of "pointwise equal" parametric natural transformations is meaningful as follows:

Lemma 9.6. Let f and f' be representatives of the same parametric natural transformation $A \to B$. There then is a continuous function of the following type:

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

Proof. Given S, a, a', and (an element of) $AS(a, a')$ we obtain $BS(f\mathrm{dom}(S)(a), f\mathrm{dom}'(S)(a'))$ and also $Br(\mathrm{dom}'(S))(f\mathrm{dom}'(S)(a'), f\mathrm{dom}'(S)(a'))$ since f and f' are pointwise equal. We conclude by transitivity. □

Lemma 9.7. If $f : A \to B$ is a parametric natural transformation then there are continuous functions of the following types

$$\Pi\mathsf{w}\,\mathsf{w}_1.\Pi u : \mathsf{w} \to \mathsf{w}_1.\Pi a \in A\mathsf{w}.Br(\mathsf{w}_1)(u.f\mathsf{w}(a), f\mathsf{w}_1(u.a))$$
$$\Pi\mathsf{w}.\Pi a\, a' \in A\mathsf{w}.Ar(\mathsf{w})(a, a') \to Br(\mathsf{w})(f\mathsf{w}(a), f\mathsf{w}(a'))$$

Proof. Fix u a and w. Lemma 9.3 furnishes an element of $AS(a, u.a)$ where S is $\mathsf{w} \xleftarrow{1} \mathsf{w} \xrightarrow{u} \mathsf{w}_1$. Since f is a parametric natural transformation, we then get an element of $BS(f\mathsf{w}(a), f\mathsf{w}_1(u.a))$. We then get the desired element of $Br(\mathsf{w}_1)(u.f\mathsf{w}(a), f\mathsf{w}_1(u.a))$ by applying parametricity of B to the parametric square $(u, 1) : S \to r(\mathsf{w}_1)$. □

Theorem 9.8. The parametric s.v.f. with parametric natural transformations form a cartesian closed category with fixpoint operator obeying the laws from Theorems 2.3 & 5.3. There is a strong monad T on this category where

$$TA\mathsf{w} = \{(\mathsf{w}_1, a) \mid \mathsf{w} \subseteq \mathsf{w}_1 \wedge a \in A\mathsf{w}_1\}_\perp$$
$$TS((\mathsf{w}_1, a), (\mathsf{w}'_1, a')) = \{(S_1 : \mathsf{w}_1 \leftrightarrow \mathsf{w}'_1, p) | (\mathsf{w} \hookrightarrow \mathsf{w}_1, \mathsf{w}' \hookrightarrow \mathsf{w}'_1) : S \to S_1 \wedge p \in AS'(a, a')\}$$

Proof. The interesting bit is the proof of transitivity for the monad for it seems to rely essentially on proof relevance. So, suppose that $S : \mathsf{w} \leftrightarrow \mathsf{w}'$ and $S' : \mathsf{w}' \leftrightarrow \mathsf{w}''$ and that $(\mathsf{w}_1, a) \in TA\mathsf{w}$ and $(\mathsf{w}'_1, a') \in TA\mathsf{w}'$ and $(\mathsf{w}''_1, a'') \in TA\mathsf{w}''$. Furthermore, suppose that $(S_1, p) \in TAS((\mathsf{w}_1, a), (\mathsf{w}'_1, a'))$ and $(S'_1, p') \in TAS'((\mathsf{w}'_1, a'), (\mathsf{w}''_1, a''))$.

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

Remark 9.9. Notice that if the extensions w_1 were existentially quantified as in Stark's original definition then the transitivity construction in the above proof would not have been possible because we would have no guarantee that the existential witnesses used in the two assumptions are the same.

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

The parametric s.v.f. and their natural transformations thus also interpret our language. Using a realization relation $\Vdash$ defined analogously one can then use this to deduce observational equivalence as well. In can then be shown that the parametric interpretation validates all the equivalences from Section 8 and in addition "Equivalence 12" above. This is because the two functions in question will be equal above the appropriate span as explained in Example 9.4.

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

$$c = (\mathtt{let}\, n \Leftarrow \mathtt{new}\ \mathtt{in}\, \lambda x.(x = n)) = (\lambda x.\mathtt{false}) = c'.$$

The equivalence proof is based on the following diagram:

$$\begin{tikzcd}
& \mathtt{c} & & \lambda x.(x = \mathsf{l}) & & x = \mathsf{l} \\
& \emptyset \arrow[rr] & & \{\mathsf{l}\} \arrow[rr] & & \{\mathsf{l}\} \cup X \cup G \\
\emptyset \arrow[ur] \arrow[dr] \arrow[rr, dotted] & & \emptyset \arrow[ur] \arrow[dr] \arrow[rr, dotted] & & X \arrow[ur] \arrow[dr] & \\
& \emptyset \arrow[rr] & & \emptyset \arrow[rr] & & X \cup G' \\
& \mathtt{c}' & & \mathtt{c}' = \lambda x.\mathtt{false} & & \mathtt{false}
\end{tikzcd}$$

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

10. Discussion

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

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

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

References

[AGM$^+$04] S. Abramsky, D. R. Ghica, A. S. Murawski, C.-H. L. Ong, and I. D. B. Stark. Nominal games and full abstraction for the nu-calculus. In *Proc. 19th Annual IEEE Symposium on Logic in Computer Science (LICS '04)*. IEEE Computer Society, 2004.

[BB06] N. Bohr and L. Birkedal. Relational reasoning for recursive types and references. In *Proc. Fourth Asian Symposium on Programming Languages and Systems (APLAS '06)*, volume 4279 of *LNCS*. Springer, 2006.

[BCP03] Gilles Barthe, Venanzio Capretta, and Olivier Pons. Setoids in type theory. *J. Funct. Program.*, 13(2):261–293, 2003.

[BCRS98] Lars Birkedal, Aurelio Carboni, Giuseppe Rosolini, and Dana S. Scott. Type theory via exact categories. In *Proc. 13th Annual IEEE Symposium on Logic in Computer Science (LICS '98)*. IEEE Computer Society, 1998.

[BÉ93] Stephen L. Bloom and Zoltán Ésik. *Iteration Theories - The Equational Logic of Iterative Processes*. EATCS Monographs on Theoretical Computer Science. Springer, 1993.

[BHN14] Nick Benton, Martin Hofmann, and Vivek Nigam. Abstract effects and proof-relevant logical relations. In *Proc. 41st ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL '14)*. ACM, 2014.

[BK13] N. Benton and V. Koutavas. A mechanized bisimulation for the nu-calculus. *Higher-Order and Symbolic Computation*, 2013. To appear.

[BKBH07] Nick Benton, Andrew Kennedy, Lennart Beringer, and Martin Hofmann. Relational semantics for effect-based program transformations with dynamic allocation. In *Proc. Ninth International ACM SIGPLAN Symposium on Principles and Practice of Declarative Programming (PPDP '07)*. ACM, 2007.

[BKHB06] Nick Benton, Andrew Kennedy, Martin Hofmann, and Lennart Beringer. Reading, writing and relations: Towards extensional semantics for effect analyses. In *Proc. Fourth Asian Symposium on Programming Languages and Systems (APLAS '06)*, volume 4279 of *LNCS*. Springer, 2006.

[BL05] Nick Benton and Benjamin Leperchey. Relational reasoning in a nominal semantics for storage. In *Proc. Seventh International Conference on Typed Lambda Calculi and Applications (TLCA '05)*, volume 3461 of *LNCS*. Springer, 2005.

[CFS87] Aurelio Carboni, Peter J. Freyd, and Andre Scedrov. A categorical approach to realizability and polymorphic types. In *Proc. Third Workshop on Mathematical Foundations of Programming Language Semantics (MFPS '87)*, volume 298 of *LNCS*. Springer, 1987.

[GP02] Murdoch Gabbay and Andrew M. Pitts. A new approach to abstract syntax with variable binding. *Formal Asp. Comput.*, 13(3-5):341–363, 2002.

[PPST00] Gordon D. Plotkin, John Power, Donald Sannella, and Robert D. Tennent. Lax logical relations. In *27th International Colloquium on Automata, Languages and Programming (ICALP '00)*, pages 85–102, 2000.

[PS93] Andrew M. Pitts and Ian D. B. Stark. Observable properties of higher-order functions that dynamically create local names, or what's new? In *Proc. 18th International Symposium on Mathematical Foundations of Computer Science (MFCS '93)*, volume 711 of *LNCS*. Springer, 1993.

[PS98] Andrew Pitts and Ian Stark. Operational reasoning for functions with local state. In *Higher Order Operational Techniques in Semantics*, pages 227–273. Cambridge University Press, 1998.

[Shi04] Mark R. Shinwell. *The Fresh Approach: functional programming with names and binders*. PhD thesis, University of Cambridge, 2004.

[SP00] Alex K. Simpson and Gordon D. Plotkin. Complete axioms for categorical fixed-point operators. In *LICS*, pages 30–41. IEEE Computer Society, 2000.

[SP05] Mark R. Shinwell and Andrew M. Pitts. On a monadic semantics for freshness. *Theor. Comput. Sci.*, 342(1):28–55, 2005.

[SPG03] Mark R. Shinwell, Andrew M. Pitts, and Murdoch J. Gabbay. FreshML: Programming with binders made simple. In *Proc. Eighth ACM SIGPLAN International Conference on Functional programming (ICFP '03)*. ACM, 2003.

[Sta94] I. D. B. Stark. *Names and Higher-Order Functions*. PhD thesis, University of Cambridge, Cambridge, UK, December 1994. Also published as Technical Report 363, University of Cambridge Computer Laboratory.

[Tze12] N. Tzevelekos. Program equivalence in a simple language with state. *Computer Languages, Systems and Structures*, 38(2), 2012.