

# A nominal relational model for local variables

Rasmus Ejlers Møgelberg <sup>1</sup>

*IT University of Copenhagen*

---

## Abstract

This paper introduces a type theory for state with local variable allocation and constructs two models. The first models types as nominal sets using known techniques. The second is a new relational variant of the first, modelling types as equivariant relations on the types of the first model. These relations can be thought of as approximations of contextual equivalence and implement a relational reasoning principle for local state, which in earlier work has been implemented using families of relations indexed by relations on state. The work of this paper reduces these families of relations to simply relations, and also show how they constitute a model in their own right. Hopefully this means that they can be used to construct better models. The relational model also validates a parametricity principle for the operation allocating local state, which implies the relational reasoning principle for local state.

*Keywords:* local state, denotational semantics, parametric polymorphism, nominal sets

---

## 1 Introduction

Most programming languages contain a construction for declaring local state, i.e., variables that can only be accessed by a specified piece of the program. This restriction of access provides an important information hiding principle: changes to an implementations internal use of local state should not affect the observable behaviour of the program. When constructing tools for reasoning about local state a challenging issue is to express the information hiding principle. One idea that has proved useful is relational reasoning: if two programs are implemented using local state, and we can show that there exists a relation between the local states preserved by the programs, then the programs should be contextually equivalent.

Traditionally, denotational models of local state have used presheaf categories [14,19,8,15], but recently an alternative approach using a continuation monad on the category of nominal sets [6] has been suggested [21]. The advantage of this new approach is that the technical development is simpler, mainly because the exponentials in the category of nominal sets have a simpler description than the Kripke style exponents of presheaf categories. In fact the two approaches are related,

---

<sup>1</sup> This work was supported by the Danish Agency for Science, Technology and Innovation

as the category of nominal sets is equivalent to a full subcategory of a presheaf topos [6].

Existing approaches to relational reasoning in the nominal sets models [2,3] construct, for each type, a family of relations on the denotations of the type, indexed by relations on state. The construction of the relations uses Kripke semantics as one has to take into account that any program can be called at a later time where the store contains more variables. One then shows that the relations corresponding to identity relations on state are contained in contextual equivalence. This technique has been successful in creating a useful tool for showing contextual equivalence.

In this paper we investigate two questions. First we ask if the parametrised relations on types constitute a model in their own right. At the moment these relations seem like a trick for obtaining better equational theories on a given model, but it is often useful to have a notion of model based on relations of the same form as the original model. For example, in relational parametricity constructing a model based on relations on types is the first step of the parametric completion process [20], which can be thought of as a way of constructing better models. We hope that something similar can be done here.

We emphasize that the relational model construction should be similar to the construction of the first model, avoiding the Kripke style exponents using nominal sets. This was the technical advantage of the nominal sets model, and we should not need to reintroduce the Kripke style exponents at the relational level.

Similar developments have in fact been done for the presheaf models [13,12,5]. There the model of local state used presheaves indexed over state shapes and the relational model used relations on types indexed over relations on state shapes. These models show a connection between local variables and relational parametricity: a procedure can be called in any extension of the state that it is defined in, and it is relationally parametric in this state extension.

This leads to the second question investigated in this paper: can the relational reasoning principle for local state be explained as a consequence of relational parametricity for the operation allocating local state? In [9], Alex Simpson and the author have shown how, in type theories for computational effects, algebraic operations [17] giving rise to effects satisfy a relational parametricity principle. In a type theory for local state there should be an algebraic operation for allocating local state, and we should be able to apply the general theory here.

To answer these questions we consider a type theory for local state with two levels of types: value types and computation types. Essentially, values have value types and computations have computation types. Using this distinction, the operations accessing the store can be expressed as functions taking computations as input and producing new computations. This approach has been chosen to relate to the earlier work of [9]. The type theory is described in Section 2.

We present two models of the type theory. The first is a model based on nominal sets using the techniques of [21,2], presented in Section 3. One difference between the model in [2] and the present is that we use sorted nominal set theory: each atom represents a cell in the store, and each of these cells have a sort representing the type of values that may be stored in that cell. The type used for the domain of the operation allocating local state is modelled using atom abstraction.

Sections 4 and 5 present a relational model, where types are interpreted as relations on the types of the first model. In the relational model nominal sets sorted by relations on state are used. Crucial for the interpretation of types is a relational interpretation of the atom abstraction type, taking relations on the sort of the atom abstracted as input. This is one of the main technical contributions of this paper. Section 5 also discusses relational parametricity for variable allocation using the relational model.

The relational model gives rise to a natural language for relational reasoning for local state. The language is related to System R [1]. Section 6 sketches this language and gives examples of its use.

## 2 A type theory for computations with local state

The type theory we consider is a two-level type theory for effects, by which we mean that it has two notions of types: value types and computation types. Two level type theories for effects [10] are a refinement of Moggi’s computational meta-language [11] based on adjunctions rather than monads. The author has in earlier work with Alex Simpson [9,10] shown how these can be useful for reasoning about effectful languages, e.g. for studying the combination of parametric polymorphism and computational effects, and Paul Levy has shown how the related call-by-push-value paradigm [8] can be seen as a generalisation of call-by-name and call-by-value. Here, however, we study a simpler calculus than those mentioned above: compared to [9,10] there is no stoup and no linear function space in the present calculus, and compared to [8] there is only one judgement. We have chosen this simplification because these constructions are not needed for the investigations of this paper, but the model allows for them to be added.

We assume that we are given some set of types  $\Sigma$  that allocated variables can have. The elements of  $\Sigma$  are referred to as sorts. We consider only ground store in this paper, and will later assume that sorts are interpreted as sets, unlike general types which are interpreted as nominal sets. Emphasizing the dependence on  $\Sigma$  we name the type theory  $\text{TLS}(\Sigma)$ .

Following the notational convention of underlining computation types, types are given by the grammar

$$\begin{aligned} \underline{A} &::= A \rightarrow \underline{B} \mid !A \mid [\mathbf{b}]\underline{A} \\ A &::= \mathbf{b} \mid \text{ref } \mathbf{b} \mid A \rightarrow B \mid !A \mid [\mathbf{b}]A. \end{aligned}$$

So computation types constitute a subset of the value types. Note that all sorts are value types. Most of the type constructors are standard. For example  $!$  is similar to Moggi’s monadic type constructor, and  $\text{ref } \mathbf{b}$  is a type of references to storage cells of type  $\mathbf{b}$ . The type  $[\mathbf{b}]A$  is a type of elements of  $A$  with an abstracted name of a storage cell of type  $\mathbf{b}$ . In the nominal sets model,  $[\mathbf{b}]A$  will be modelled using atom abstraction.

Terms are given by the grammar

$$t ::= x \mid \lambda x : A. t \mid t u \mid \langle \langle a : \mathbf{b} \rangle \rangle t \mid !t \mid \text{let } !x \text{ be } t \text{ in } t' \mid \text{lup}_{\mathbf{b}, \underline{B}} \mid \text{upd}_{\mathbf{b}, \underline{B}} \mid \text{new}_{\mathbf{b}, \underline{B}}$$

$$\begin{array}{c}
\frac{}{\Theta \mid \Gamma, x: A, \Gamma' \vdash x: A} \quad \frac{\Theta \mid \Gamma, x: A, \Gamma' \vdash t: B}{\Theta \mid \Gamma, \Gamma' \vdash \lambda x: A. t: A \rightarrow B} \\
\frac{\Theta \mid \Gamma \vdash t: A \rightarrow B \quad \Theta \mid \Gamma \vdash u: A}{\Theta \mid \Gamma \vdash tu: B} \\
\frac{}{\Theta, a: \mathbf{b}, \Theta' \mid \Gamma \vdash a: \text{ref } \mathbf{b}} \quad \frac{\Theta, a: \mathbf{b}, \Theta' \mid \Gamma \vdash t: A}{\Theta, \Theta' \mid \Gamma \vdash \langle\langle a: \mathbf{b} \rangle\rangle t: [\mathbf{b}]A} \\
\frac{\Theta \mid \Gamma \vdash t: !A \quad \Theta \mid \Gamma, x: A, \Gamma' \vdash u: \underline{B}}{\Theta \mid \Gamma \vdash \text{let } !x \text{ be } t \text{ in } u: \underline{B}} \quad \frac{\Theta \mid \Gamma \vdash t: A}{\Theta \mid \Gamma \vdash !t: !A}
\end{array}$$

Fig. 1. Typing rules

Typing rules can be found in Figure 1. Open terms have two contexts: a context of variables  $x: A$  in the usual sense — we use  $\Gamma$  as metavariable for that — and a context of names  $a: \mathbf{b}$  for which we use  $\Theta$  as a metavariable. The names should be thought of as references to cells in the store. Just like variables may be abstracted in terms using  $\lambda$ -abstraction, names may be abstracted in terms to construct terms with bound names. This is the construction  $\langle\langle a: \mathbf{b} \rangle\rangle t$ .

The operations for accessing the state are functions that take computations as input and produce new computations. For example, following the Plotkin-Power axiomatisation of state effects [17], writing to and reading from store is given by the constants

$$\begin{aligned}
\text{upd}_{\mathbf{b}, \underline{B}}: \underline{B} \rightarrow \text{ref } \mathbf{b} \rightarrow \mathbf{b} \rightarrow \underline{B} \\
\text{lup}_{\mathbf{b}, \underline{B}}: (\mathbf{b} \rightarrow \underline{B}) \rightarrow \text{ref } \mathbf{b} \rightarrow \underline{B}
\end{aligned}$$

Intuitively, the first of these (update) takes a computation, a reference to a storage cell of type  $\mathbf{b}$ , a value of type  $\mathbf{b}$ , and returns the computation that first updates the storage cell with the given value and then continues the original computation. The lookup operation  $\text{lup}$  takes a family of computations indexed by values of type  $\mathbf{b}$  and a reference to a storage cell, and returns the computation that looks up the value in the storage cell and continues with the corresponding computation.

To these operations, we add an operation

$$\text{new}_{\mathbf{b}, \underline{B}}: [\mathbf{b}]\underline{B} \rightarrow \mathbf{b} \rightarrow \underline{B}$$

for allocating a fresh storage cell. It takes a computation of type  $\underline{B}$  with an abstracted name of a storage cell of type  $\mathbf{b}$ , a value of type  $\mathbf{b}$  and returns the computation which allocates a fresh cell of type  $\mathbf{b}$ , initializes it to the given value of type  $\mathbf{b}$  and continues with the given computation with the abstracted name bound to the allocated cell.

I will not present an equational theory for the terms as we will be interested in the theories induced by the models to be defined below.

**Lemma 2.1** *If  $\Theta \mid \Gamma \vdash t: A$  is a valid typing judgement, so is  $\Theta' \mid \Gamma' \vdash t: A$ , for any contexts  $\Theta', \Gamma'$  extending  $\Theta, \Gamma$ .*

### 3 A model in nominal sets

We construct an interpretation of the type theory  $\text{TLS}(\Sigma)$  using nominal sets [6]. We assume that we are given for each  $\mathbf{b} \in \Sigma$  an interpretation of  $\mathbf{b}$  as a set  $\llbracket \mathbf{b} \rrbracket$ . Moreover we assume that we are given a *sorted collection of atoms* by which we shall mean a map of sets  $\mathbb{A} \rightarrow \Sigma$ , such that for each sort  $\mathbf{b}$  there are infinitely many atoms mapping to  $\mathbf{b}$  by the sorting function. We shall think of a sorted collection of atoms as a signature for a typed store. The set  $\mathbb{A}$  is the set of cells and the sorting maps a cell to the type of the content stored in that cell. The set of atoms with sort  $\mathbf{b}$  is denoted  $\mathbb{A}_{\mathbf{b}}$ , and we often write  $\mathbf{a}_{\mathbf{b}}$  to indicate that  $\mathbf{a}$  is an atom of sort  $\mathbf{b}$ .

Given a sorted collection of atoms  $\mathbb{A} \rightarrow \Sigma$  we can form the category of nominal sets, which we denote  $\mathbf{Nom}$  or sometimes  $\mathbf{Nom}_{[\mathbb{A} \rightarrow \Sigma]}$  when the atom sorting is not clear from context, as usual. First we consider the group  $\mathbf{Perm}(\mathbb{A} \rightarrow \Sigma)$  of finite permutations (i.e. permutations fixing all but a finite number of elements) of  $\mathbb{A}$  respecting sortings. The category  $\mathbf{Nom}$  has as objects sets with a left  $\mathbf{Perm}(\mathbb{A} \rightarrow \Sigma)$ -action, i.e., a map  $\cdot : \mathbf{Perm}(\mathbb{A} \rightarrow \Sigma) \times X \rightarrow X$  such that  $(\pi \circ \pi') \cdot x = \pi \cdot (\pi' \cdot x)$  and  $id \cdot x = x$  such that each  $x$  in  $X$  has a finite support, i.e. a finite set  $A$  of atoms such that if  $\pi$  fixes all elements of  $A$  then  $\pi \cdot x = x$ . It is well known that  $\mathbf{Nom}$  is a cartesian closed category with exponent given by the set of finitely supported functions with respect to the action defined by  $(\pi \cdot f)(x) = \pi \cdot (f(\pi^{-1} \cdot x))$ , see [6]. We write  $A \times B$  and  $A \rightarrow_{\text{fs}} B$  or  $B^A$  for the cartesian closed structure on the category of nominal sets and reserve  $A \rightarrow B$  for the set of functions with empty support between a pair of nominal sets. Finally,  $[\mathbb{A}_{\mathbf{b}}]A$  is used for atom abstraction and we write  $x \in X$  for  $X$  a nominal set if  $x$  is an element in the underlying set of  $X$ .

As in [21,2] we use a cps style semantics, because it greatly simplifies the interpretation of new. So we assume given a nominal set  $R$  of results, which has trivial permutation action. The set  $\mathbb{A}$  is a nominal set with the obvious action and recalling that each sort  $\mathbf{b}$  has an interpretation as a set, we can consider each of these a nominal set with trivial permutation action.

We define the nominal set of states as

$$\mathbb{S} = \prod_{\mathbf{b} \in \Sigma} \mathbb{A}_{\mathbf{b}} \rightarrow_{\text{fs}} \llbracket \mathbf{b} \rrbracket.$$

Note that by this product we mean the product in  $\mathbf{Nom}$ , i.e., we only consider elements  $f$  with a finite set of atoms which supports all of the  $f_{\mathbf{b}} : \mathbb{A}_{\mathbf{b}} \rightarrow_{\text{fs}} \llbracket \mathbf{b} \rrbracket$ .

**Remark 3.1** We usually assume that all the sets  $\llbracket \mathbf{b} \rrbracket$  are non-empty as otherwise  $\mathbb{S}$  becomes empty. However the model constructed in this section and the theorems here do not depend on this assumption.

A continuation is a map that takes a state and produces a result in  $R$ , however, by our definition of  $\mathbb{S}$ , states are infinite and intuitively, computable continuations can only look at finite subsets of the state. The restriction to finitely supported functions is not sufficient to ensure this restriction, as, e.g., maps counting the number of cells holding a specific value are finitely supported (in fact equivariant). Instead we define our collection of continuations to be the nominal set

$$\mathbb{K} = \{k : R^{\mathbb{S}} \mid \exists A \subseteq_{\text{fin}} \mathbb{A}. \forall s, s' \in \mathbb{S}. (\forall \mathbf{a} \in A. s(\mathbf{a}) = s'(\mathbf{a})) \implies k(s) = k(s')\}.$$

$$\begin{array}{ll}
\llbracket A \rightarrow B \rrbracket_{\mathcal{V}} = \llbracket A \rrbracket_{\mathcal{V}} \rightarrow_{\text{fs}} \llbracket B \rrbracket_{\mathcal{V}} & \llbracket A \rightarrow B \rrbracket_{\mathcal{C}} = \llbracket B \rrbracket_{\mathcal{C}} \times \llbracket A \rrbracket_{\mathcal{V}} \\
\llbracket !A \rrbracket_{\mathcal{V}} = \mathbb{K}^{\llbracket A \rrbracket_{\mathcal{V}}} & \llbracket !A \rrbracket_{\mathcal{C}} = \mathbb{K}^{\llbracket A \rrbracket_{\mathcal{V}}} \\
\llbracket [\mathbf{b}]A \rrbracket_{\mathcal{V}} = \llbracket A_{\mathbf{b}} \rrbracket_{\mathcal{V}} & \llbracket [\mathbf{b}]A \rrbracket_{\mathcal{C}} = \llbracket \mathbf{b} \rrbracket \times \llbracket A_{\mathbf{b}} \rrbracket_{\mathcal{C}} \\
\llbracket \text{ref } \mathbf{b} \rrbracket_{\mathcal{V}} = A_{\mathbf{b}} &
\end{array}$$

Fig. 2. Interpretation of types

This restriction has important consequences for the model and is also crucial for the construction of the relational model.

**Lemma 3.2** *If  $k \in \mathbb{K}$  and  $s, s' \in \mathbb{S}$  agree on the support of  $k$ , then  $k(s) = k(s')$ .*

Two-level type theories are modelled using adjunctions [10], and the model constructed in this section is based on the adjunction below.

$$\begin{array}{ccc}
& \mathbf{Nom}^{\text{op}} & \\
\mathbb{K}(-) \uparrow & \dashv & \downarrow \mathbb{K}(-) \\
& \mathbf{Nom} & 
\end{array} \tag{1}$$

This means that all computation types  $\underline{A}$  are interpreted as objects  $\llbracket \underline{A} \rrbracket_{\mathcal{C}}$  of  $\mathbf{Nom}^{\text{op}}$  and all value types  $\underline{B}$  as objects  $\llbracket \underline{B} \rrbracket_{\mathcal{V}}$  of  $\mathbf{Nom}$ . The interpretation of types is defined in Figure 5. Since computation types are special value types these are given two interpretations, but these are related as the next proposition shows.

**Proposition 3.3** *There is an isomorphism of nominal sets*

$$\llbracket \underline{B} \rrbracket_{\mathcal{V}} \cong \mathbb{K}^{\llbracket \underline{B} \rrbracket_{\mathcal{C}}}$$

for any computation type  $\underline{B}$ .

The isomorphism is constructed by induction on the structure of  $\underline{B}$ , but we omit the construction for reasons of space. In fact, we shall take the interpretation of a computation type to be a triple  $(\llbracket \underline{B} \rrbracket_{\mathcal{V}}, \llbracket \underline{B} \rrbracket_{\mathcal{C}}, i_{\underline{B}})$  where  $i_{\underline{B}}$  is the isomorphism of Proposition 3.3. The isomorphism is used in the interpretation of the operations.

Before we describe the interpretation of terms we recall the tensor product on  $\mathbf{Nom}$ :

$$A \otimes B = \{(x, y) \mid x \sharp y\}$$

where  $x \sharp y$  means that  $x, y$  have disjoint support. In particular  $\otimes_{\mathbf{b}_i \in \bar{\mathbf{b}}} A_{\mathbf{b}_i}$  is the set of tuples  $(\mathbf{a}^1, \dots, \mathbf{a}^n)$  of distinct elements with  $\mathbf{a}^i \in A_{\mathbf{b}_i}$ .

Terms are interpreted as maps of nominal sets. More precisely, if

$$a_1 : \mathbf{b}_1, \dots, a_m : \mathbf{b}_m \mid x_1 : A_1, \dots, x_n : A_n \vdash t : B$$

then  $t$  is interpreted as an equivariant map from  $\otimes A_{\mathbf{b}_i} \times \prod_i \llbracket A_i \rrbracket_{\mathcal{V}}$  to  $\llbracket B \rrbracket_{\mathcal{V}}$ . The interpretation of terms is detailed in Figure 3. For simplicity of notation, in the

$$\begin{aligned}
\llbracket x \rrbracket_\rho &= \rho(x) \\
\llbracket a \rrbracket_\rho &= \rho(a) \\
\llbracket \lambda x : \mathbf{A}. t \rrbracket_\rho &= a \mapsto \llbracket t \rrbracket_{\rho[x \mapsto a]} \\
\llbracket t u \rrbracket_\rho &= \llbracket t \rrbracket_\rho(\llbracket u \rrbracket_\rho) \\
\llbracket \langle a : \mathbf{b} \rangle t \rrbracket_\rho &= \text{fresh } \mathbf{a}_b. (\mathbf{a}_b. \llbracket t \rrbracket_{\rho[a \mapsto \mathbf{a}_b]}) \\
\llbracket ! t \rrbracket_\rho &= \eta(\llbracket t \rrbracket_\rho) \\
\llbracket \text{let } ! x \text{ be } t \text{ in } u \rrbracket_\rho &= \mathbb{K}^{\lambda v. \widehat{\llbracket u \rrbracket_{\rho[x \mapsto v]}}}(\llbracket t \rrbracket_\rho) \\
\llbracket \text{lup}_{\mathbf{b}, \underline{\mathbf{B}}}(t) \rrbracket_\rho(\mathbf{a}_b, s, x) &= \llbracket t \rrbracket_\rho(s(\mathbf{a}_b))(s, x) \\
\llbracket \text{upd}_{\mathbf{b}, \underline{\mathbf{B}}}(t) \rrbracket_\rho(n)(\mathbf{a}_b, s, x) &= \llbracket t \rrbracket_\rho(s[\mathbf{a}_b \mapsto n], x) \\
\llbracket \text{new}_{\mathbf{b}, \underline{\mathbf{B}}}(t) \rrbracket_\rho(n)(s, x) &= \text{fresh } \mathbf{a}_b. (\llbracket t \rrbracket_{\rho @ \mathbf{a}_b})(s[\mathbf{a}_b \mapsto n], x)
\end{aligned}$$

Fig. 3. Interpretation of terms

figure we use environments. A  $\Theta, \Gamma$ -environment is a map  $\rho$  from the set of variables in  $\Theta, \Gamma$ , such that  $\rho(a_i) \in \mathbb{A}_{\mathbf{b}_i}$  such that  $\rho(a_i) \neq \rho(a_j)$  for  $i \neq j$ , and  $\rho(x_i) \in \llbracket \mathbf{A}_i \rrbracket_\nu$ . We write  $\llbracket t \rrbracket_\rho$  for the value  $\llbracket t \rrbracket(\rho(a_1), \dots, \rho(a_m), \rho(x_1), \dots, \rho(x_n))$ . We use  $\eta$  for the unit of the monad  $\mathbb{K}^{\mathbb{K}^{(-)}}$ . Standard notation from nominal set theory is used in Figure 3. For example  $x @ \mathbf{a}_b$  is used for the concretion of  $x$  at  $\mathbf{a}_b$ , i.e., the unique  $z$  such that  $x = (\mathbf{a}_b. z)$ .

The interpretation of  $\text{let } ! x \text{ be } t \text{ in } u$  deserves a little explanation. Using Proposition 3.3 the interpretation of  $u$  gives rise to a map  $\lambda v. \llbracket u \rrbracket_{\rho[x \mapsto v]} : \llbracket \mathbf{A} \rrbracket_\nu \rightarrow_{\text{fs}} \mathbb{K}^{\llbracket \mathbf{B} \rrbracket c}$  which corresponds to a map  $\lambda v. \widehat{\llbracket u \rrbracket_{\rho[x \mapsto v]}} : \llbracket \mathbf{B} \rrbracket c \rightarrow_{\text{fs}} \mathbb{K}^{\llbracket \mathbf{A} \rrbracket_\nu}$  by the adjunction (1). We can apply the composite map

$$\llbracket ! \mathbf{A} \rrbracket_\nu = \mathbb{K}^{\mathbb{K}^{\llbracket \mathbf{A} \rrbracket_\nu}} \xrightarrow{\mathbb{K}^{(\lambda v. \widehat{\llbracket u \rrbracket_{\rho[x \mapsto v]})}} \rightarrow \mathbb{K}^{\llbracket \mathbf{B} \rrbracket c} \xrightarrow{\cong} \llbracket \mathbf{B} \rrbracket_\nu$$

to  $\llbracket t \rrbracket_\rho$ .

In the figure the notation  $s[\mathbf{a}_b \mapsto n]$  is used for updating  $s$  at location  $\mathbf{a}_b$ , where  $s \in \mathbb{S}$ , i.e.,  $s[\mathbf{a}_b \mapsto n](\mathbf{a}'_b)$  is  $n$  if  $\mathbf{a}'_b = \mathbf{a}_b$  and  $s(\mathbf{a}'_b)$  otherwise.

**Proposition 3.4** *For any term  $t$  the interpretation  $\llbracket t \rrbracket$  of  $t$  is a well defined equivariant map.*

The well definedness of the interpretation of  $\text{new}$  is proved as in [21], which leaves only the issue of showing that all continuations defined are in  $\mathbb{K}$ . The proof is routine.

**Proposition 3.5** *The model validates the Plotkin-Power axioms for local state [15] - Figure 4 - and the monadic rules for let-binding [11].*

The last axiom of Figure 4 (garbage collection) does not hold in the model of [2], but can only be verified up to relational reasoning. The reason it holds in this model is the restriction to continuations in  $\mathbb{K}$ .

**Proof.** The monadic rules hold because let-binding is defined the usual way using

$$\begin{aligned}
& \text{lup}_{\mathbf{b}, \underline{\mathbf{B}}}(\lambda v. \text{upd}_{\mathbf{b}, \underline{\mathbf{B}}} t a v) a = t \\
& \text{lup}_{\mathbf{b}, \underline{\mathbf{B}}}(\lambda v'. \text{lup}_{\mathbf{b}, \underline{\mathbf{B}}}(\lambda v. t v v') a) a = \text{lup}_{\mathbf{b}, \underline{\mathbf{B}}}(\lambda v. t v v) a \\
& \text{upd}_{\mathbf{b}, \underline{\mathbf{B}}}(\text{upd}_{\mathbf{b}, \underline{\mathbf{B}}} t a v) a v' = \text{upd}_{\mathbf{b}, \underline{\mathbf{B}}} t a v \\
& \text{upd}_{\mathbf{b}, \underline{\mathbf{B}}}(\text{lup}_{\mathbf{b}, \underline{\mathbf{B}}} t a) a v = \text{upd}_{\mathbf{b}, \underline{\mathbf{B}}}(t v) a v \\
& \text{lup}_{\mathbf{b}, \underline{\mathbf{B}}}(\lambda v. (\text{lup}_{\mathbf{b}', \underline{\mathbf{B}}}(\lambda v'. t) a')) a = \text{lup}_{\mathbf{b}', \underline{\mathbf{B}}}(\lambda v'. (\text{lup}_{\mathbf{b}, \underline{\mathbf{B}}}(\lambda v. t) a)) a' \\
& \text{upd}_{\mathbf{b}, \underline{\mathbf{B}}}(\text{upd}_{\mathbf{b}', \underline{\mathbf{B}}} t a' v') a v = \text{upd}_{\mathbf{b}', \underline{\mathbf{B}}}(\text{upd}_{\mathbf{b}, \underline{\mathbf{B}}} t a v) a' v' \\
& \text{upd}_{\mathbf{b}, \underline{\mathbf{B}}}(\text{lup}_{\mathbf{b}', \underline{\mathbf{B}}}(\lambda v'. t) a') a v = \text{lup}_{\mathbf{b}', \underline{\mathbf{B}}}(\lambda v'. (\text{upd}_{\mathbf{b}, \underline{\mathbf{B}}} t a v)) a' \\
& \text{new}_{\mathbf{b}, \underline{\mathbf{B}}}(\langle\langle a : \mathbf{b} \rangle\rangle(\text{upd}_{\mathbf{b}, \underline{\mathbf{B}}} t a v)) v' = \text{new}_{\mathbf{b}, \underline{\mathbf{B}}}(\langle\langle a : \mathbf{b} \rangle\rangle(t)) v \\
& \text{new}_{\mathbf{b}, \underline{\mathbf{B}}}(\langle\langle a : \mathbf{b} \rangle\rangle(\text{lup}_{\mathbf{b}, \underline{\mathbf{B}}} t a)) v = \text{new}_{\mathbf{b}, \underline{\mathbf{B}}}(\langle\langle a : \mathbf{b} \rangle\rangle(t(v))) v \\
& \text{new}_{\mathbf{b}, \underline{\mathbf{B}}}(\langle\langle a : \mathbf{b} \rangle\rangle(\text{new}_{\mathbf{b}', \underline{\mathbf{B}}}(\langle\langle a' : \mathbf{b}' \rangle\rangle t) v')) v = \text{new}_{\mathbf{b}', \underline{\mathbf{B}}}(\langle\langle a' : \mathbf{b}' \rangle\rangle(\text{new}_{\mathbf{b}, \underline{\mathbf{B}}}(\langle\langle a : \mathbf{b} \rangle\rangle t) v)) v' \\
& \text{new}_{\mathbf{b}, \underline{\mathbf{B}}}(\langle\langle a : \mathbf{b} \rangle\rangle(\text{upd}_{\mathbf{b}', \underline{\mathbf{B}}} t a' v')) v = \text{upd}_{\mathbf{b}', \underline{\mathbf{B}}}(\text{new}_{\mathbf{b}, \underline{\mathbf{B}}}(\langle\langle a : \mathbf{b} \rangle\rangle t v) a' v') \\
& \text{new}_{\mathbf{b}, \underline{\mathbf{B}}}(\langle\langle a : \mathbf{b} \rangle\rangle(\text{lup}_{\mathbf{b}', \underline{\mathbf{B}}} t a')) v = \text{lup}_{\mathbf{b}', \underline{\mathbf{B}}}(\text{new}_{\mathbf{b}, \mathbf{b}' \rightarrow \underline{\mathbf{B}}}(\langle\langle a : \mathbf{b} \rangle\rangle t) v) a' \\
& \text{new}_{\mathbf{b}, \underline{\mathbf{B}}}(\langle\langle a : \mathbf{b} \rangle\rangle t)(v) = t, \quad \text{if } a \notin \text{FN}(t)
\end{aligned}$$

Fig. 4. Plotkin-Power axioms for local state [15]. The axioms above assume  $a \neq a'$  and in the side condition of the last rule  $\text{FN}(t)$  stands for the set of free name variables in  $t$ .

the monad  $\mathbb{K}^{\mathbb{K}^{(-)}}$ . Verifying the axioms for local state is for most cases routine. In fact, it is well known that any object of the form  $A^{\mathbb{S}}$  is a model of the axioms for global state (the first seven axioms), which is really what is used here. We only show the interesting case of the last axiom.

For the garbage collection axiom we compute

$$\begin{aligned}
\llbracket \text{new}_{\mathbf{b}, \underline{\mathbf{B}}}(\langle\langle a : \mathbf{b} \rangle\rangle t)(v) \rrbracket_{\rho}(s, x) &= \text{fresh } \mathbf{a}_{\mathbf{b}}. \llbracket t \rrbracket_{\rho[a \mapsto \mathbf{a}_{\mathbf{b}}]}(s[\mathbf{a}_{\mathbf{b}} \mapsto \llbracket v \rrbracket_{\rho}], x) \\
&= \text{fresh } \mathbf{a}_{\mathbf{b}}. \llbracket t \rrbracket_{\rho}(s[\mathbf{a}_{\mathbf{b}} \mapsto \llbracket v \rrbracket_{\rho}], x)
\end{aligned}$$

using a weakening lemma. Now,  $\mathbf{a}_{\mathbf{b}}$  is fresh for the continuation  $\llbracket t \rrbracket_{\rho}(-, x)$ , and so by Lemma 3.2, we get

$$\begin{aligned}
\text{fresh } \mathbf{a}_{\mathbf{b}}. \llbracket t \rrbracket_{\rho}(s[\mathbf{a}_{\mathbf{b}} \mapsto \llbracket v \rrbracket_{\rho}], x) &= \text{fresh } \mathbf{a}_{\mathbf{b}}. \llbracket t \rrbracket_{\rho}(s, x) \\
&= \llbracket t \rrbracket_{\rho}(s, x).
\end{aligned}$$

□

## 4 A relational model construction

The type theory  $\text{TLS}(\Sigma)$  as defined above is parametric in the collection of storable types  $\Sigma$ , and the interpretation given above is parametric in the interpretations of the storable types and in the sorted collection of atoms  $\mathbb{A} \rightarrow \Sigma$ . In this section we study how relations between this data induces relations between the corresponding languages and their interpretations. The theorems obtained here will be specialised in Section 5 and used to construct the relational model.

Any map  $f: \Sigma \rightarrow \Sigma'$  induces a translation  $\text{TLS}(\Sigma) \rightarrow \text{TLS}(\Sigma')$  obtained by substituting each  $\mathbf{b}$  by  $f(\mathbf{b})$ . If we had term constants in the language we would need a little more data to get such a translation. The translation is described in detail in Appendix A. In fact, in the appendix we have described something slightly more general, as we have described how any map  $f: \Sigma \rightarrow (\Sigma')^*$  (the codomain being the set of lists of elements in  $\Sigma'$ ) induces a translation  $\text{TLS}(\Sigma) \rightarrow \text{TLS}(\Sigma')$  translating each type of  $\text{TLS}(\Sigma)$  to a list of types of  $\text{TLS}(\Sigma')$  and each term to a list of terms. We need this generality later on.

Relations between sorted collections of atoms will be given by maps (and spans of maps) of the form

$$\begin{array}{ccc}
\mathbb{A} & \xrightarrow{c} & \mathbb{A}' \\
\downarrow & & \downarrow \\
\Sigma & \xrightarrow{d} & \Sigma'
\end{array} \tag{2}$$

where  $c$  is injective. Given the intuition of sorted collections of atoms being signatures for stores, such a map corresponds to a signature for a map mapping stores of one type to stores of another (supplying dummy values to the cells not in the image of  $c$ ). Consider, for example, the case of  $d$  being the identity, then (2) corresponds to adding more cells to a store. Another special case is  $d$  being an inclusion and (2) a pullback. This corresponds to extending a store with more cell types.

We shall generalise slightly from maps of the form (2), as we want to include signatures for maps that distribute information stored in one cell in the first store to information stored in multiple cells in the second store. For example each cell in the first store of type  $\text{Int} \times \text{String}$  may be mapped to two cells of type  $\text{Int}$  and  $\text{String}$  respectively.

**Definition 4.1** For any atom collection  $\mathbb{A} \rightarrow \Sigma$  let  $\Sigma^*$  be the set of lists of elements of  $\Sigma$  and  $\mathbb{A}^{\#*}$  be the set of lists  $\bar{\mathbf{a}}$  of elements of  $\mathbb{A}$  such that  $\mathbf{a}_i = \mathbf{a}_j$  implies  $i = j$ . A *list inclusion of atom collections* is a pair of atom collections  $\mathbb{A} \rightarrow \Sigma$  and  $\mathbb{A}' \rightarrow \Sigma'$  and a pair of maps  $c, d$ , such that

$$\begin{array}{ccc}
\mathbb{A} & \xrightarrow{c} & (\mathbb{A}')^{\#*} \\
\downarrow & & \downarrow \\
\Sigma & \xrightarrow{d} & (\Sigma')^*
\end{array}$$

commutes, and such that for all  $\mathbf{a}, \mathbf{a}'$  if  $c(\mathbf{a}) \cap c(\mathbf{a}') \neq \emptyset$  then  $\mathbf{a} = \mathbf{a}'$ .

In the definition we have used the notation  $c(\mathbf{a}) \cap c(\mathbf{a}')$  to denote the intersection of the sets of elements in the two lists  $c(\mathbf{a})$  and  $c(\mathbf{a}')$ . Below we shall often apply set theoretic notation to lists by which we always mean the notation applied to the sets of elements of the lists.

The strong injectivity condition in Definition 4.1 implies that  $a, b$  induce a homomorphism  $(c, d)_* : \mathbf{Perm}(\mathbb{A} \rightarrow \Sigma) \rightarrow \mathbf{Perm}(\mathbb{A}' \rightarrow \Sigma')$  defined as

$$((c, d)_*\pi)(\mathbf{a}) = \begin{cases} c(\pi(\mathbf{a}'))_i & \text{if } c(\mathbf{a}')_i = \mathbf{a} \\ \mathbf{a} & \text{else} \end{cases}$$

This homomorphism can be used to define a functor

$$(c, d)^* : \mathbf{Nom}_{[\mathbb{A}' \rightarrow \Sigma']} \rightarrow \mathbf{Nom}_{[\mathbb{A} \rightarrow \Sigma]}$$

mapping a nominal set  $X$  to the nominal set with the same underlying set  $X$  and action defined as  $(\pi, x) \mapsto ((c, d)_*\pi) \cdot x$  for  $\pi \in \mathbf{Perm}(\mathbb{A} \rightarrow \Sigma)$ . So, for example, the action of a transposition is defined to be  $(\mathbf{a} \mathbf{a}') x = (c(\mathbf{a})_1 c(\mathbf{a}')_1) \dots (c(\mathbf{a})_n c(\mathbf{a}')_n) x$ . The support of an element can be computed as

$$\text{supp}_{(c, d)^* X}(x) = c^{-1}(\text{supp}_X(x)) = \{\mathbf{a} \mid c(\mathbf{a}) \cap \text{supp}_X(x) \neq \emptyset\}$$

for any  $x \in X$ .

**Proposition 4.2** *The functor  $(c, d)^*$  is well defined and product preserving.*

We remark that  $(c, d)^*$  needs not preserve the cartesian closed structure.

Since the image  $d(\mathbf{b})$  of a sort  $\mathbf{b}$  in general is a list of sorts it is convenient to introduce some list notation. For  $\bar{\mathbf{a}} = \mathbf{a}_1, \dots, \mathbf{a}_n$  and  $\bar{\mathbf{b}} = \mathbf{b}_1, \dots, \mathbf{b}_m$  we write  $[\mathbb{A}_{\bar{\mathbf{b}}}]A$  for  $[\mathbb{A}_{\mathbf{b}_1}] \dots [\mathbb{A}_{\mathbf{b}_m}]A$ , write  $x @ \bar{\mathbf{a}}$  for  $x @ \mathbf{a}_1 \dots @ \mathbf{a}_n$  and write  $(\bar{\mathbf{a}}.x)$  for  $(\mathbf{a}_1 \dots (\mathbf{a}_n.x))$ . This notation is extended to lists  $\bar{A}$  and  $\bar{x}$  by applying the constructions to each element in the list.

**Proposition 4.3** *The functor  $(c, d)^*$  preserves atom abstractions in the sense that*

$$(c, d)^*([\mathbb{A}'_{d(\mathbf{b})}]X) \cong [\mathbb{A}_{\mathbf{b}}]((c, d)^*X)$$

**Proof.** One direction of the isomorphism maps  $x$  in  $[\mathbb{A}'_{d(\mathbf{b})}]X$  to fresh  $\mathbf{a}_{\mathbf{b}}$ .  $(\mathbf{a}_{\mathbf{b}}.x @ c(\mathbf{a}_{\mathbf{b}}))$ . The other maps  $x \in [\mathbb{A}_{\mathbf{b}}]((c, d)^*X)$  to fresh  $\mathbf{a}_{\mathbf{b}}$ .  $(c(\mathbf{a}_{\mathbf{b}}).x @ \mathbf{a}_{\mathbf{b}})$ . We omit the verification that these in fact define isomorphisms.  $\square$

We now show how a span of atom inclusions

$$\begin{array}{ccccc} (\mathbb{A}^0)^{\#*} & \xleftarrow{c_0} & \mathbb{A} & \xrightarrow{c_1} & (\mathbb{A}^1)^{\#*} \\ \downarrow & & \downarrow & & \downarrow \\ (\Sigma^0)^* & \xleftarrow{d_0} & \Sigma & \xrightarrow{d_1} & (\Sigma^1)^* \end{array} \quad (3)$$

gives rise to a relational interpretation of  $\text{TLS}(\Sigma)$  with types interpreted as relations. First note that the span induces a cospan of categories and functors

$$\mathbf{Nom}_{[\mathbb{A}^0 \rightarrow \Sigma^0]} \xrightarrow{(c_0, d_0)^*} \mathbf{Nom}_{[\mathbb{A} \rightarrow \Sigma]} \xleftarrow{(c_1, d_1)^*} \mathbf{Nom}_{[\mathbb{A}^1 \rightarrow \Sigma^1]}$$

and we can use this cospan to define the category **Rel** as

**Objects:** Triples  $(\bar{X}, \bar{Y}, Q)$  where  $\bar{X}, \bar{Y}$  are lists of objects of  $\mathbf{Nom}_{[\mathbb{A}^0 \rightarrow \Sigma^0]}$  and  $\mathbf{Nom}_{[\mathbb{A}^1 \rightarrow \Sigma^1]}$  respectively and  $Q$  is an equivariant (in the sense of  $\mathbf{Nom}_{[\mathbb{A} \rightarrow \Sigma]}$ ) subset of  $\prod_i (c_0, d_0)^* X_i \times \prod_j (c_1, d_1)^* Y_j$

**Morphisms:** A morphism from  $(\bar{X}, \bar{Y}, Q)$  to  $(\bar{X}', \bar{Y}', Q')$  is a pair of maps  $f: \prod_i X_i \rightarrow \prod_i X'_i$  and  $g: \prod_j Y_j \rightarrow \prod_j Y'_j$  in  $\mathbf{Nom}_{[\mathbb{A}^0 \rightarrow \Sigma^0]}$  and  $\mathbf{Nom}_{[\mathbb{A}^1 \rightarrow \Sigma^1]}$  respectively, such that  $((c_0, d_0)^* f, (c_1, d_1)^* g)$  map pairs related in  $Q$  to pairs related in  $Q'$ .

**Proposition 4.4** *The category **Rel** is cartesian closed and the pair of projection functors*

$$\mathbf{Nom}_{[\mathbb{A}^0 \rightarrow \Sigma^0]} \longleftarrow \mathbf{Rel} \longrightarrow \mathbf{Nom}_{[\mathbb{A}^1 \rightarrow \Sigma^1]}$$

mapping  $(\bar{X}, \bar{Y}, Q)$  to  $\prod_i X_i$  and  $\prod_j Y_j$  respectively, preserve the cartesian closed structure.

**Proof.** We just describe the exponent:  $(\bar{X}, \bar{Y}, Q) \rightarrow (\bar{X}', \bar{Y}', Q')$  is the triple  $(\prod_i X_i \rightarrow_{\text{fs}} \prod_i X'_i, \prod_i Y_i \rightarrow_{\text{fs}} \prod_i Y'_i, Q'^Q)$  where the first two components are constructed using products and exponents in  $\mathbf{Nom}_{[\mathbb{A}^0 \rightarrow \Sigma^0]}$  and  $\mathbf{Nom}_{[\mathbb{A}^1 \rightarrow \Sigma^1]}$  respectively, and  $Q'^Q(f, g)$  holds if  $Q(\bar{x}, \bar{y})$  implies  $Q'(f(\bar{x}), g(\bar{y}))$ .  $\square$

Just as in Section 3 we assume that we are given an interpretation of the sorts, so we assume that we are given for each sort  $\mathbf{b}$  in  $\Sigma^0$  a set denoted  $\llbracket \mathbf{b} \rrbracket$  and likewise for each  $\mathbf{b}$  in  $\Sigma^1$ , and we assume that we are given, for each  $\mathbf{r}$  in  $\Sigma$  a relation  $Q_{\mathbf{r}}^{\mathcal{V}}: \text{Rel}(\llbracket \text{TLS}(d_0)(\mathbf{r}) \rrbracket, \llbracket \text{TLS}(d_1)(\mathbf{r}) \rrbracket)$ . Here, since each  $\text{TLS}(d_i)(\mathbf{r})$  is a list of types,  $\llbracket \text{TLS}(d_i)(\mathbf{r}) \rrbracket$  is a list of sets and we write  $\text{Rel}(\bar{X}, \bar{Y})$  for lists of sets  $\bar{X}, \bar{Y}$  for the set of subsets of  $\prod_i X_i \times \prod_j Y_j$ . Finally, we assume that we are given a result set  $R$  such that the interpretations of  $\text{TLS}(\Sigma^0)$  and  $\text{TLS}(\Sigma^1)$  can be defined exactly as in Section 3.

The relational interpretation is defined much as the interpretation of Section 3, and we start by defining a continuation object in **Rel** as follows. First consider the nominal sets  $\mathbb{K}_0, \mathbb{K}_1$  defined as in Section 3 for the atom sortings  $\mathbb{A}^0 \rightarrow \Sigma^0$  and  $\mathbb{A}^1 \rightarrow \Sigma^1$  respectively. Define the relation

$$\mathbb{K}_{\text{Rel}} = \{(k_0, k_1) \in \mathbb{K}_0 \times \mathbb{K}_1 \mid \exists A \subseteq_{\text{fin}} \mathbb{A}. \forall s_0, s_1. (\forall \mathbf{a}_{\mathbf{r}} \in A. Q_{\mathbf{r}}^{\mathcal{V}}(s_0(c_0(\mathbf{a}_{\mathbf{r}})), s_1(c_1(\mathbf{a}_{\mathbf{r}})))) \implies k_0(s_0) = k_1(s_1)\}$$

**Lemma 4.5** *If all  $Q_{\mathbf{r}}^{\mathcal{V}}$  are non empty then  $(k_0, k_1) \in \mathbb{K}_{\text{Rel}}$  iff for all  $s_0, s_1$ ,*

$$\forall \mathbf{a}_{\mathbf{r}} \in c_0^{-1}(\text{supp}(k_0)) \cup c_1^{-1}(\text{supp}(k_1)). Q_{\mathbf{r}}^{\mathcal{V}}(s_0(c_0(\mathbf{a}_{\mathbf{r}})), s_1(c_1(\mathbf{a}_{\mathbf{r}})))$$

*implies  $k_0(s_0) = k_1(s_1)$ .*

**Remark 4.6** If one of the  $Q_{\mathbf{r}}^{\mathcal{V}}$  is empty, then  $\mathbb{K}_{\text{Rel}}(k_0, k_1)$  will hold for all continuations  $k_0, k_1$ , and so we shall usually assume that all these relations are non-empty. One can avoid this assumption if one uses  $c_0^{-1}(\text{supp}(k_0)) \cup c_1^{-1}(\text{supp}(k_1))$  instead of  $A$  in the definition of  $\mathbb{K}_{\text{Rel}}$ .

**Lemma 4.7** *The triple  $(\mathbb{K}_0, \mathbb{K}_1, \mathbb{K}_{\text{Rel}})$  defines an object of **Rel**.*

$$\begin{aligned}
Q_{\mathbf{A} \rightarrow \mathbf{B}}^{\mathcal{V}} &= (Q_{\mathbf{B}}^{\mathcal{V}})^{Q_{\mathbf{A}}^{\mathcal{V}}} & Q_{\mathbf{A} \rightarrow \mathbf{B}}^{\mathcal{C}} &= Q_{\mathbf{B}}^{\mathcal{C}} \times Q_{\mathbf{A}}^{\mathcal{V}} \\
Q_{! \mathbf{A}}^{\mathcal{V}} &= \mathbb{K}_{\mathbf{Rel}}^{\mathbb{K}_{\mathbf{Rel}}^{Q_{\mathbf{A}}^{\mathcal{V}}}} & Q_{! \mathbf{A}}^{\mathcal{C}} &= \mathbb{K}_{\mathbf{Rel}}^{Q_{\mathbf{A}}^{\mathcal{V}}} \\
Q_{[\mathbf{r}] \mathbf{A}}^{\mathcal{V}} &= [\mathbf{A}_{\mathbf{r}}] Q_{\mathbf{A}}^{\mathcal{V}} & Q_{[\mathbf{r}] \mathbf{B}}^{\mathcal{C}} &= Q_{\mathbf{r}}^{\mathcal{V}} \times [\mathbf{A}_{\mathbf{r}}] Q_{\mathbf{B}}^{\mathcal{V}} \\
Q_{\text{ref } \mathbf{b}}^{\mathcal{V}} &= (\mathbb{A}_{c_0(\mathbf{b})}^0, \mathbb{A}_{c_1(\mathbf{b})}^1, \mathbb{A}_{\mathbf{b}})
\end{aligned}$$

Fig. 5. Interpretation of types

The model of  $\text{TLS}(\Sigma)$  is based on the adjunction

$$\begin{array}{ccc}
& \mathbf{Rel}^{\text{op}} & \\
\mathbb{K}_{\mathbf{Rel}}^{(-)} \uparrow & \dashv & \downarrow \mathbb{K}_{\mathbf{Rel}}^{(-)} \\
& \mathbf{Rel} & 
\end{array}$$

where the exponents and products refer to the cartesian closed structure of  $\mathbf{Rel}$ .

More precisely, as in Section 3 we define for each value type  $\mathbf{A}$  an object  $Q_{\mathbf{A}}^{\mathcal{V}}$  of  $\mathbf{Rel}$  and for each computation type  $\mathbf{B}$  an object  $Q_{\mathbf{B}}^{\mathcal{C}}$  such that

$$Q_{\mathbf{B}}^{\mathcal{V}} \cong \mathbb{K}_{\mathbf{Rel}}^{Q_{\mathbf{B}}^{\mathcal{C}}}.$$

The definition is much as in Section 3 and is presented in Figure 5. We explain the new constructions used in the figure.

The relation  $Q_{\text{ref } \mathbf{b}}^{\mathcal{V}}$  is given by the span

$$\mathbb{A}_{d_0(\mathbf{b})}^0 \xleftarrow{c_0} \mathbb{A}_{\mathbf{b}} \xrightarrow{c_1} \mathbb{A}_{d_1(\mathbf{b})}^1$$

where for a list  $\bar{\mathbf{b}} = \mathbf{b}_1, \dots, \mathbf{b}_n$  we write  $\mathbb{A}_{\bar{\mathbf{b}}}$  for  $(\mathbb{A}_{\mathbf{b}_1}, \dots, \mathbb{A}_{\mathbf{b}_n})$ .

The crucial ingredient in the relational model defined here is an atom abstraction for objects of  $\mathbf{Rel}$ . For  $(\bar{A}, \bar{B}, Q)$  we define  $[\mathbf{A}_{\mathbf{r}}](\bar{A}, \bar{B}, Q)$  to be the relation  $[\mathbf{A}_{\mathbf{r}}]Q: \text{Rel}([\mathbb{A}_{d_0(\mathbf{r})}^0] \bar{A}, [\mathbb{A}_{d_1(\mathbf{r})}^1] \bar{B})$  relating  $(\bar{x}, \bar{y})$  iff (leaving the isomorphisms of Proposition 4.3 implicit)  $Q(\bar{x} @ \mathbf{a}_{\mathbf{r}}, \bar{y} @ \mathbf{a}_{\mathbf{r}})$  for some / any  $\mathbf{a}_{\mathbf{r}}$  fresh for  $\bar{x}, \bar{y}$ . (This is independent of choice of  $\mathbf{a}_{\mathbf{r}}$  as  $Q$  is assumed equivariant.)

**Lemma 4.8** *Each  $Q_{\mathbf{A}}^{\mathcal{V}}$  and  $Q_{\mathbf{B}}^{\mathcal{C}}$  defines an object of  $\mathbf{Rel}$*

**Proof.** The relations are shown to be equivariant by induction on the type structure. The base cases of sorts are hypotheses of the theorem and equivariance of  $\mathbb{K}_{\mathbf{Rel}}$  is Lemma 4.7. For the constructions for function spaces and the free computation type constructor note that these are given using the cartesian closed structure of  $\mathbf{Rel}$  and that the objects of this category are equivariant relations. For the case of atom abstraction, assume  $Q_{[\mathbf{r}] \mathbf{A}}^{\mathcal{V}}(\bar{x}, \bar{y})$ . Given atoms  $\mathbf{a}_{\mathbf{r}}, \mathbf{a}'_{\mathbf{r}}$  we need to show that

$$Q_{\mathbf{A}}^{\mathcal{V}}(((\mathbf{a}_{\mathbf{r}} \mathbf{a}'_{\mathbf{r}}) \bar{x}) @ \mathbf{a}''_{\mathbf{r}}, ((\mathbf{a}_{\mathbf{r}} \mathbf{a}'_{\mathbf{r}}) \bar{y}) @ \mathbf{a}''_{\mathbf{r}})$$

for some / any fresh  $\mathbf{a}_r''$ . We may assume  $\mathbf{a}_r''$  fresh for  $\mathbf{a}_r, \mathbf{a}_r'$  and then

$$((\mathbf{a}_r \mathbf{a}_r') \bar{x}) @ \mathbf{a}_r'' = (\mathbf{a}_r \mathbf{a}_r') (\bar{x} @ \mathbf{a}_r'')$$

and likewise for  $\bar{y}$ . The result now follows as by induction  $Q_A^\mathcal{V}$  is equivariant.  $\square$

**Lemma 4.9** *For all computation types  $\underline{B}$  the **Rel** objects  $Q_{\underline{B}}^\mathcal{V}$  and  $\mathbb{K}_{\text{Rel}}^{Q_{\underline{B}}^\mathcal{C}}$  are isomorphic.*

Before we interpret terms in the model, we define the interpretation of contexts. We write  $Q_\Gamma^\mathcal{V}$  for  $\prod_i Q_{A_i}^\mathcal{V}$  if  $\Gamma$  is the context  $x_1 : A_1, \dots, x_n : A_n$ . For the name contexts  $\Theta = a_1 : \mathbf{b}_1, \dots, a_n : \mathbf{b}_n$  note first that for each  $i$  the span

$$\mathbb{A}_{d_0(\mathbf{b}_i)}^0 \xleftarrow{c_0} \mathbb{A}_{\mathbf{b}_i} \xrightarrow{c_1} \mathbb{A}_{d_1(\mathbf{b}_i)}^1 \quad (4)$$

defines a relation from  $\otimes_j \mathbb{A}_{d_0(\mathbf{b}_i)_j}^0$  to  $\otimes_j \mathbb{A}_{d_1(\mathbf{b}_i)_j}^1$ . We define  $Q_\Theta^\mathcal{V}$  as the span

$$\otimes_i \otimes_j \mathbb{A}_{d_0(\mathbf{b}_i)}^0 \xleftarrow{\otimes_i c_0} \otimes_i \mathbb{A}_{\mathbf{b}_i} \xrightarrow{\otimes_i c_1} \otimes_i \otimes_j \mathbb{A}_{d_1(\mathbf{b}_i)}^1$$

**Theorem 4.10** *For any term  $\Theta \mid \Gamma \vdash t : A$  in  $\text{TLS}(\Sigma)$  the interpretations of  $\text{TLS}(d_0)(t)$  and  $\text{TLS}(d_1)(t)$  define a map*

$$([\text{TLS}(d_0)(t)], [\text{TLS}(d_1)(t)]): Q_\Theta^\mathcal{V} \times Q_\Gamma^\mathcal{V} \rightarrow Q_A^\mathcal{V}$$

of **Rel**.

## 5 Approximating contextual equivalence

In this section we assume that we are given a set of sorts and an interpretation  $\llbracket \mathbf{b} \rrbracket$  of each sort as above. Using the model construction of Section 4 we will show how to construct a family of relations  $\llbracket A \rrbracket_{\mathcal{R}_\mathcal{V}} : \text{Rel}(\llbracket A \rrbracket_{\mathcal{V}}, \llbracket A \rrbracket_{\mathcal{V}})$  on the interpretations  $\llbracket A \rrbracket_{\mathcal{V}}$  of the types of  $\text{TLS}(\Sigma)$  as constructed in Section 3. These relations should be thought of as approximations of contextual equivalence, and giving a larger (in the sense of equating more terms) equality theory on  $\text{TLS}(\Sigma)$ , than the one induced by the model of Section 3.

Strictly speaking, we cannot talk about contextual equivalence in the type theory  $\text{TLS}(\Sigma)$  in the usual sense, as there is no operational semantics. But the intended use of  $\text{TLS}(\Sigma)$  for operational semantics is as a target language for translations, and we show conditions on the relations which ensure that if the translation is adequate, then the relations in the target language lift to approximations of contextual equivalence for the source language.

First consider the set of non-empty relations on sorts

$$\Sigma^{\text{Rel}} = \{(\bar{\mathbf{b}}, \bar{\mathbf{b}'}, R) \mid \bar{\mathbf{b}}, \bar{\mathbf{b}'} \in \Sigma^*, R : \text{Rel}(\llbracket \bar{\mathbf{b}} \rrbracket, \llbracket \bar{\mathbf{b}'} \rrbracket), R \text{ non-empty}\}.$$

The restriction to *non-empty* relations is necessary to get an interesting model, although there are ways around it, see Remark 4.6.

By *the relational language* we shall mean  $\text{TLS}(\Sigma^{\text{Rel}})$ . There is a reflexive graph of sets

$$\Sigma^* \begin{array}{c} \xleftarrow{\text{cod}} \\ \xrightarrow{\Delta} \\ \xleftarrow{\text{dom}} \end{array} \Sigma^{\text{Rel}}$$

where  $\text{dom}$  maps a relation  $(\bar{\mathbf{b}}, \bar{\mathbf{b}}', R)$  to its domain  $\bar{\mathbf{b}}$  and  $\text{cod}$  maps it to its codomain  $\bar{\mathbf{b}}'$  and  $\Delta$  maps a list of sorts to the identity relation on the sorts. Moreover, there is an operation  $-^{\text{op}}: \Sigma^{\text{Rel}} \rightarrow \Sigma^{\text{Rel}}$  mapping  $(\bar{\mathbf{b}}, \bar{\mathbf{b}}', R)$  to  $(\bar{\mathbf{b}}', \bar{\mathbf{b}}, R^{\text{op}})$ . We use  $\mathbf{r}$  as metavariable for elements of  $\Sigma^{\text{Rel}}$ .

Suppose we are given an infinite set of atoms  $\mathbb{A}$  (this time just a set, no sorting) (so  $\mathbb{A}$  could just be the set of natural numbers). Consider the sets

$$\begin{aligned} \mathbb{A}^{\text{dom}} &= \coprod_{(\bar{\mathbf{b}}, \bar{\mathbf{b}}', R) \in \Sigma^{\text{Rel}}} \coprod_{0 \leq i \leq |\bar{\mathbf{b}}|} \mathbb{A} \\ \mathbb{A}^{\text{Rel}} &= \mathbb{A} \times \Sigma^{\text{Rel}} \end{aligned}$$

The second projection defines an atom sorting  $\mathbb{A}^{\text{Rel}} \rightarrow \Sigma^{\text{Rel}}$ , and so does the map  $\mathbb{A}^{\text{dom}} \rightarrow \Sigma$  mapping  $((\bar{\mathbf{b}}, \bar{\mathbf{b}}', R), i, \mathbf{a})$  to  $\mathbf{b}_i$ . There is a span of list inclusions of atom collections

$$\begin{array}{ccccc} (\mathbb{A}^{\text{dom}})^{\#*} & \xleftarrow{\text{dom}} & \mathbb{A}^{\text{Rel}} & \xrightarrow{\mathbb{A} \times (-)^{\text{op}}} & \mathbb{A}^{\text{Rel}} & \xrightarrow{\text{dom}} & (\mathbb{A}^{\text{dom}})^{\#*} \\ \downarrow & & \downarrow & & \downarrow & & \downarrow \\ \Sigma^* & \xleftarrow{\text{dom}} & \Sigma^{\text{Rel}} & \xrightarrow{\text{cod}} & \Sigma^* & & \Sigma^* \end{array} \quad (5)$$

where  $\text{dom}(\mathbf{a}, \mathbf{r}) = ((\mathbf{r}, 1, \mathbf{a}), \dots, (\mathbf{r}, n, \mathbf{a}))$  for  $n$  the length of the domain of  $\mathbf{r}$ .

**Definition 5.1** The *relational model* of  $\text{TLS}(\Sigma^{\text{Rel}})$  is the model obtained by applying the construction of Section 4 to (5). For any type  $\mathbf{R}$  of  $\text{TLS}(\Sigma^{\text{Rel}})$  we write  $\llbracket \mathbf{R} \rrbracket_{\mathcal{R}_V}$  for the relation obtained by interpreting the type in the relational model.

**Notation 5.2** In the following we shall often write simply  $\Delta$  for  $\text{TLS}(\Delta)$ .

**Definition 5.3** For  $t, t'$  closed terms of the same type  $\mathbf{A}$  in  $\text{TLS}(\Sigma)$  we write  $t \sim_{\mathbf{A}} t'$  for  $\llbracket \Delta(\mathbf{A}) \rrbracket_{\mathcal{R}_V}(\llbracket t \rrbracket, \llbracket t' \rrbracket)$ . For open terms we write  $\Theta \mid \Gamma \vdash t \sim_{\mathbf{A}} t'$  if both  $t, t'$  have type  $\mathbf{A}$  in context  $\Theta, \Gamma$  and

$$(\llbracket t \rrbracket, \llbracket t' \rrbracket): \llbracket \Delta(\Theta) \rrbracket_{\mathcal{R}_V} \times \llbracket \Delta(\Gamma) \rrbracket_{\mathcal{R}_V} \rightarrow \llbracket \Delta(\mathbf{A}) \rrbracket_{\mathcal{R}_V}.$$

**Lemma 5.4** Suppose  $t, t': \mathbf{A}$  in  $\text{TLS}(\Sigma)$  and there exists  $t''$  of type  $\Delta(\mathbf{A})$  in  $\text{TLS}(\Sigma^{\text{Rel}})$  such that  $\text{TLS}(\text{dom})(t'') = t$  and  $\text{TLS}(\text{cod})(t'') = t'$ . Then  $t \sim_{\mathbf{A}} t'$ .

**Proof.** By Theorem 4.10  $Q_{\Delta(\mathbf{A})}^{\text{Rel}}(\text{TLS}(\text{dom})(t''), \text{TLS}(\text{cod})(t''))$ , and so the lemma follows by definition of  $\llbracket \Delta(\mathbf{A}) \rrbracket_{\mathcal{R}_V}$ .  $\square$

The next theorem summarizes the essential properties of the relations defined above.

**Theorem 5.5** (i) For each value type  $A$  of  $\text{TLS}(\Sigma)$  the relation  $\llbracket \Delta(A) \rrbracket_{\mathcal{R}_V}$  is symmetric and zigzag-closed, i.e.,

$$\llbracket \Delta(A) \rrbracket_{\mathcal{R}_V}(x, y) \wedge \llbracket \Delta(A) \rrbracket_{\mathcal{R}_V}(z, y) \wedge \llbracket \Delta(A) \rrbracket_{\mathcal{R}_V}(z, w) \implies \llbracket \Delta(A) \rrbracket_{\mathcal{R}_V}(x, w)$$

(ii) For each value type  $A$  of  $\text{TLS}(\Sigma)$  the relation  $\sim_A$  is an equivalence relation.

(iii) The union of all the  $\sim_A$  relations is a congruence relation on terms.

**Proof.** The proof of (i) is by induction on the structure of  $A$ , but we omit it for reasons of space. By (i),  $\sim_A$  is symmetric and zigzag-closed, and by Lemma 5.4 it is reflexive. These properties imply transitivity, and so we conclude that it is an equivalence relation.

For the proof of (iii), suppose  $t \sim_A t'$  and suppose  $C[-]$  is a context such that whenever  $u$  is a term of type  $A$  then  $C[u]$  has type  $B$ . We must show that  $C[t] \sim_B C[t']$ . Applying  $\Delta$  to  $C[-]$  we get a context that preserves relations, so applying the context  $\llbracket \Delta(C[-]) \rrbracket$  to  $(\llbracket t \rrbracket, \llbracket t' \rrbracket)$  gives a pair of related values. But clearly  $\llbracket \Delta(C[-]) \rrbracket(\llbracket t \rrbracket, \llbracket t' \rrbracket)$  is  $(\llbracket C[t] \rrbracket, \llbracket C[t'] \rrbracket)$ .  $\square$

### 5.1 Relational parametricity for new

If  $A$  is a type of  $\text{TLS}(\Sigma)$  and  $Q: \text{Rel}(\llbracket \mathbf{b} \rrbracket, \llbracket \mathbf{b}' \rrbracket)$  then writing  $\mathbf{r}$  for  $(\mathbf{b}, \mathbf{b}', Q)$ , we obtain a  $\text{TLS}(\Sigma^{\text{Rel}})$  type  $[\mathbf{r}]\Delta(A)$ . In the model, this type is interpreted as a relation from  $[\mathbf{b}]A$  to  $[\mathbf{b}']A$  and so we can think of the mapping  $\mathbf{r} \mapsto [\mathbf{r}]\Delta(A)$  as a *relational interpretation* of the type constructor  $[-]A$ , similarly to the relational interpretations of the type constructors of second-order lambda calculus used in the theory of parametric polymorphism [18,16]. By Theorem 4.10 the model verifies the relation preservation statement  $(\text{new}_{\mathbf{b}, \underline{\mathbf{B}}}, \text{new}_{\mathbf{b}', \underline{\mathbf{B}}}) : [\mathbf{r}](\Delta(\underline{\mathbf{B}})) \rightarrow \mathbf{r} \rightarrow \Delta(\underline{\mathbf{B}})$ . We can formulate this principle logically as a parametricity statement

$$\forall \mathbf{b}, \mathbf{b}', \mathbf{r} : \text{Rel}(\mathbf{b}, \mathbf{b}'). [\mathbf{r}](\Delta(\underline{\mathbf{B}}))(x, y) \wedge \mathbf{r}(n, n') \implies \Delta(\underline{\mathbf{B}})(\text{new}_{\mathbf{b}, \underline{\mathbf{B}}}(x)(n), \text{new}_{\mathbf{b}', \underline{\mathbf{B}}}(y)(n'))$$

Note that this is equivalent to

$$\forall \mathbf{b}, \mathbf{b}'. (\exists \mathbf{r} : \text{Rel}(\mathbf{b}, \mathbf{b}'). [\mathbf{r}](\Delta(\underline{\mathbf{B}}))(x, y) \wedge \mathbf{r}(n, n')) \implies \Delta(\underline{\mathbf{B}})(\text{new}_{\mathbf{b}, \underline{\mathbf{B}}}(x)(n), \text{new}_{\mathbf{b}', \underline{\mathbf{B}}}(y)(n')). \quad (6)$$

Given our intuition for the atom abstraction types  $[\mathbf{b}]\underline{\mathbf{B}}$  that the elements of these are computations with an abstracted cell name, we should read  $[\mathbf{r}](\Delta(\underline{\mathbf{B}}))(x, y)$  as meaning that  $x, y$  preserve the relation  $\mathbf{r}$  between the abstracted cells. The formula (6) should be read as the relational reasoning principle for local state mentioned in the introduction, and we see that in this model, the relational reasoning principle follows from a parametricity principle for new.

## 6 Verifying term equivalences

We introduce a language useful for reasoning about the relational model. Typing judgements are of the form

$$\frac{\overline{\mathbf{a}_1} : \overline{\mathbf{b}_1} \quad \overline{\mathbf{a}_n} : \overline{\mathbf{b}_n}}{\overline{\mathbf{r}_1} : \overline{\mathbf{b}'_1}, \dots, \overline{\mathbf{r}_n} : \overline{\mathbf{b}'_n}} \mid \frac{\overline{x_1} : \overline{A_1} \quad \overline{x_m} : \overline{A_m}}{\overline{x'_1} : \overline{A'_1}, \dots, \overline{x'_m} : \overline{A'_m}} \quad \frac{\overline{t} : \overline{\mathbf{B}}}{\overline{t'} : \overline{\mathbf{B}'}} \vdash \frac{}{\mathbf{S}}$$

Such a judgement is valid, if the upper third of the judgement and the lower third both are valid typing judgements in  $\text{TLS}(\Sigma)$ , each  $\mathbf{r}_i \in \Sigma^{\text{Rel}}$  and each  $\mathbf{R}_i, \mathbf{S}$  are types of  $\text{TLS}(\Sigma^{\text{Rel}})$  and the interpretation of the upper and lower third of the judgement define a map in the relational model, i.e.,

$$(\llbracket t \rrbracket, \llbracket t' \rrbracket): \bigotimes_i \llbracket \mathbf{r}_i \rrbracket_{\mathcal{R}_V} \times \prod_j \llbracket \mathbf{R}_j \rrbracket_{\mathcal{R}_V} \rightarrow \llbracket \mathbf{S} \rrbracket_{\mathcal{R}_V}.$$

From soundness of the model for  $\text{TLS}(\Sigma^{\text{Rel}})$  we obtain a typing rule for this language for each typing rule of Figure 1, for example

$$\frac{\Theta, \frac{\bar{\mathbf{a}}: \bar{\mathbf{b}}}{\mathbf{a}': \bar{\mathbf{b}}}, \Theta' \mid \Gamma \vdash \frac{\bar{t}: \bar{\mathbf{A}}}{\mathbf{r}: \bar{\mathbf{R}}}}{\Theta, \Theta' \mid \Gamma \vdash \frac{\langle \langle \bar{\mathbf{a}}: \bar{\mathbf{b}} \rangle \bar{t}: \bar{\mathbf{b}} \bar{\mathbf{A}} \rangle}{\langle \langle \bar{\mathbf{a}}': \bar{\mathbf{b}}' \rangle \bar{t}': \bar{\mathbf{b}} \bar{\mathbf{A}}' \rangle}}}{\Theta, \Theta' \mid \Gamma \vdash \frac{\langle \langle \bar{\mathbf{a}}: \bar{\mathbf{b}} \rangle \bar{t}: \bar{\mathbf{b}} \bar{\mathbf{A}} \rangle}{\langle \langle \bar{\mathbf{a}}': \bar{\mathbf{b}}' \rangle \bar{t}': \bar{\mathbf{b}} \bar{\mathbf{A}}' \rangle}} \quad (7)}$$

and similarly, weakening of the judgement follows from Lemma 2.1. Note that in judgement (7) the lists  $\bar{\mathbf{b}}, \bar{\mathbf{b}}'$  need not be of the same length, they may even be empty. The language obtained this way is very similar to System R [1] (see also [4]).

The statement of an equivalence of terms

$$\mathbf{a}_1: \mathbf{b}_1, \dots, \mathbf{a}_n: \mathbf{b}_n \mid x_1: \mathbf{A}_1, \dots, x_m: \mathbf{A}_m \vdash t \sim_{\mathbf{B}} t'$$

is the same as the statement

$$\frac{\mathbf{a}_1: \mathbf{b}_1, \dots, \mathbf{a}_n: \mathbf{b}_n \mid \Delta(\mathbf{b}_1), \dots, \Delta(\mathbf{b}_n)}{\mathbf{a}_1: \mathbf{b}_1, \dots, \mathbf{a}_n: \mathbf{b}_n} \mid \frac{x_1: \mathbf{A}_1, \dots, x_m: \mathbf{A}_m \vdash t: \mathbf{B}}{x_1: \mathbf{A}_1, \dots, x_m: \mathbf{A}_m \vdash t': \mathbf{B}}$$

and we shall see in the following examples how to use the language to reason about equivalence of terms.

**Example 6.1** In this example we show equivalence of two implementations of a counter. We will write both implementations of the counter in the type theory  $\text{TLS}(\{\text{Int}\})$ . Both counters are implemented with a local variable which in both cases is initialized to 0. The first counter takes an integer  $n$ , increases the value of the local variable by  $n$  and returns the new value. The second counter decreases the value of the local variable by  $n$  and returns the negative of the resulting value. The counters  $\text{counter}_1, \text{counter}_2$  are defined as

$$\text{counter}_i = \text{new}_{\text{Int},!}(\text{Int} \rightarrow !(\text{Int}))(\langle \langle a: \text{Int} \rangle \rangle !c_{i,a})(0)$$

where  $c_{i,a}$  are the function that in an imperative language would have been written as something like

$$\begin{aligned} c_{1,a} &= \lambda x: \text{Int}. a := !a + x; [!a] \\ c_{2,a} &= \lambda x: \text{Int}. a := !a - x; [-!a] \end{aligned}$$

but in our type theory are written as

$$\begin{aligned} c_{1,a} &= \lambda x: \text{Int}. \text{lup}_{\text{Int}, \text{Int} \rightarrow !(\text{Int})}(\lambda n: \text{Int}. \text{upd}_{\text{Int}, !(\text{Int})}(!n + x) a (n + x)) a \\ c_{2,a} &= \lambda x: \text{Int}. \text{lup}_{\text{Int}, \text{Int} \rightarrow !(\text{Int})}(\lambda n: \text{Int}. \text{upd}_{\text{Int}, !(\text{Int})}(!x - n) a (n - x)) a \end{aligned}$$

Both counters have type  $!(\text{Int} \rightarrow !(\text{Int}))$ . They are computations returning functions of type  $\text{Int} \rightarrow !(\text{Int})$  - and not just functions - because they allocate a local variable and initialize it to 0, before they return a function.

Consider the relation  $\mathbf{r}: \text{Rel}(\text{Int}, \text{Int})$  defined by  $\mathbf{r} = \{(n, -n) \mid n \in \mathbb{N}\}$ . One may easily check that

$$\begin{array}{c} a: \text{Int} \\ \mathbf{r} \\ a: \text{Int} \end{array} \mid - \vdash \begin{array}{c} c_{1,a}: \text{Int} \rightarrow !(\text{Int}) \\ \Delta(\text{Int}) \rightarrow !(\Delta \text{Int}) \\ c_{2,a}: \text{Int} \rightarrow !(\text{Int}) \end{array}$$

and so by (7)

$$- \mid - \vdash \begin{array}{c} \langle\langle a: \text{Int} \rangle\rangle! c_{1,a}: [\mathbf{b}]!(\text{Int} \rightarrow !(\text{Int})) \\ [\mathbf{r}]!(\Delta \text{Int} \rightarrow !(\Delta \text{Int})) \\ \langle\langle a: \text{Int} \rangle\rangle! c_{2,a}: [\mathbf{b}]!(\text{Int} \rightarrow !(\text{Int})) \end{array}$$

and since  $(0, 0) \in \mathbf{r}$  we conclude

$$- \mid - \vdash \text{counter}_1 \sim_{!(\text{Int} \rightarrow !(\text{Int}))} \text{counter}_2.$$

**Example 6.2** The use of relations between lists of types in this paper is in order to reason about equivalence of programs using a different number of local variables as illustrated in this simple example. Suppose the collection of sorts is closed under products, and suppose the term  $t$  uses a single local variable cell  $a$  of type  $\mathbf{b} \times \mathbf{b}'$ , and that term  $t'$  behaves exactly as  $t$ , except that it uses two local cells  $a, a'$  of type  $\mathbf{b}, \mathbf{b}'$  respectively to store the needed information, and suppose that these are defined in the same context. In such a case we would like to show that

$$\text{new}_{\mathbf{b} \times \mathbf{b}', \underline{\mathbf{B}}}(\langle\langle a: \mathbf{b} \times \mathbf{b}' \rangle\rangle t)(\langle\langle v, v' \rangle\rangle) \sim_{\underline{\mathbf{B}}} \text{new}_{\mathbf{b}', \underline{\mathbf{B}}}(\text{new}_{\mathbf{b}, [\mathbf{b}'] \underline{\mathbf{B}}}(\langle\langle a: \mathbf{b} \rangle\rangle \langle\langle a': \mathbf{b}' \rangle\rangle t') v) v'$$

Precisely, we will assume that

$$\Delta(\Theta), \begin{array}{c} a: \mathbf{b} \times \mathbf{b}' \\ \mathbf{r} \\ a: \mathbf{b}, a': \mathbf{b}' \end{array} \mid \Delta(\Gamma) \vdash \begin{array}{c} t \\ \Delta(\underline{\mathbf{B}}) \\ t' \end{array}$$

where  $\mathbf{r}: \text{Rel}([\mathbf{b} \times \mathbf{b}'], ([\mathbf{b}], [\mathbf{b}']))$  is the relation  $\{(\langle\langle n, n' \rangle\rangle, n, n') \mid n \in [\mathbf{b}], n' \in [\mathbf{b}']\}$ . By (7)

$$\Delta(\Theta) \mid \Delta(\Gamma) \vdash \begin{array}{c} \langle\langle a: \mathbf{b} \times \mathbf{b}' \rangle\rangle t \\ [\mathbf{r}] \Delta(\underline{\mathbf{B}}) \\ \langle\langle a: \mathbf{b} \rangle\rangle \langle\langle a': \mathbf{b}' \rangle\rangle t' \end{array}$$

and so by relational parametricity of  $\text{new}$  and  $\mathbf{r}(\langle\langle v, v' \rangle\rangle, v, v')$  we get

$$\Delta(\Theta) \mid \Delta(\Gamma) \vdash \begin{array}{c} \text{new}_{\mathbf{b} \times \mathbf{b}', \underline{\mathbf{B}}}(\langle\langle a: \mathbf{b} \times \mathbf{b}' \rangle\rangle t)(\langle\langle v, v' \rangle\rangle) \\ \Delta(\underline{\mathbf{B}}) \\ \text{new}_{\mathbf{b}', \underline{\mathbf{B}}}(\text{new}_{\mathbf{b}, [\mathbf{b}'] \underline{\mathbf{B}}}(\langle\langle a: \mathbf{b} \rangle\rangle \langle\langle a': \mathbf{b}' \rangle\rangle t') v) v' \end{array}$$

as desired.

## 7 Conclusions

We have answered the first question posed in the introduction to the positive by showing how the techniques used to model local state using nominal sets can be

used to construct a relational model. This reduces families of relations indexed by relations on state to simply relations, and the construction can hopefully be used to construct better models of local state. Comparing with earlier work on relational reasoning for nominal set models of local state [2,3], we treat a simple notion of state, and further work is needed to treat references to references or higher order state using the techniques presented here.

This paper also stated a parametricity principle for new which implies a useful relational reasoning principle for local state. Comparing this to ordinary relational parametricity however, the relational interpretation of types (here the mapping of  $A$  to  $\text{TLS}(\Delta)(A)$ ) does not satisfy an identity extension theorem such as the one known from second-order lambda-calculus [18].

## Acknowledgement

The author had helpful discussions with Lars Birkedal, Alex Simpson and Ian Stark. Paul Taylors diagram macros were used.

## A List translations of TLS

A list translation is a translation from one calculus to another translating types as lists of types and terms as lists of terms. The length of the list resulting as a translation of a type can vary from type to type and may even be zero. In this appendix we see how a map  $f: \Sigma \rightarrow (\Sigma')^*$  induces a list translation  $\text{TLS}(f): \text{TLS}(\Sigma) \rightarrow \text{TLS}(\Sigma')$ , which we in this section, to keep notation as short as possible, simply denote  $(-)^*$ .

A term context  $\Gamma = x_1: A_1, \dots, x_m: A_m$  is translated as

$$\Gamma^* = x_{1,1}: A_{1,1}^*, \dots, x_{1,p_1}: A_{1,p_1}^*, \dots, x_{m,1}: A_{m,1}^*, \dots, x_{m,p_m}: A_{m,p_m}^*$$

where  $A_i^* = (A_{i,1}^*, \dots, A_{i,p_i}^*)$  and the name contexts are translated similarly. If  $B^* = (B_1^*, \dots, B_n^*)$  then a term judgement  $\Theta \mid \Gamma \vdash t: B$  is translated as a list  $(t_1^*, \dots, t_n^*)$  such that

$$\Theta^* \mid \Gamma^* \vdash t_i^*: B_i^*$$

is a valid term judgement.

The translation is defined in Figure A.1. In the figure we have used short notation  $\text{upd}_{f(\mathbf{b}), \underline{B}_i^*} t_i^*$  which means

$$\text{upd}_{f(\mathbf{b})_1, \text{ref}(f(\mathbf{b})_2) \rightarrow f(\mathbf{b})_2 \rightarrow \dots \rightarrow f(\mathbf{b})_n \rightarrow \underline{B}_i^*} (\dots (\text{upd}_{f(\mathbf{b})_n, \underline{B}_i^*} (t_i^*)) \dots).$$

Similar notation is used for new and lup, although these cases are slightly more complicated, because these operations have non-trivial coarity. For example new  $f(\mathbf{b}), \underline{B}_i^*(t)$  is

$$\lambda x_1: f(\mathbf{b})_1 \dots \lambda x_n: f(\mathbf{b})_n. (\text{new}_{f(\mathbf{b})_n, \underline{B}_i^*} (\dots \text{new}_{f(\mathbf{b})_1, [f(\mathbf{b})_2] \dots [f(\mathbf{b})_n] \underline{B}_i^*} (t)(x_1)) \dots (x_n))$$

**Proposition A.1** *The translation  $\text{TLS}(f)$  takes computation types to lists of computation types and preserves validity of type judgements.*

$$\begin{aligned}
(\mathbf{A} \rightarrow \mathbf{B})^* &= (\mathbf{A}_1^* \rightarrow \dots \rightarrow \mathbf{A}_n^* \rightarrow \mathbf{B}_1^*, \dots, \mathbf{A}_1^* \rightarrow \dots \rightarrow \mathbf{A}_n^* \rightarrow \mathbf{B}_m^*) \\
([\mathbf{b}]\mathbf{A})^* &= ([f(\mathbf{b})]_1 \dots [f(\mathbf{b})_k] \mathbf{A}_1^*, \dots, [f(\mathbf{b})]_1 \dots [f(\mathbf{b})_k] \mathbf{A}_n^*) \\
(\text{ref } \mathbf{b})^* &= (\text{ref } f(\mathbf{b})_1, \dots, \text{ref } f(\mathbf{b})_k) \\
(!\mathbf{A})^* &= (!\mathbf{A}_1^*, \dots, !\mathbf{A}_n^*) \\
\\
x_i^* &= (x_{i,1}, \dots, x_{i,p_i}) \\
(\lambda x : \mathbf{A}. t)^* &= (\lambda x_1 : \mathbf{A}_1^* \dots \lambda x_n : \mathbf{A}_n^*. t_1^*, \dots, \lambda x_1 : \mathbf{A}_1^* \dots \lambda x_n : \mathbf{A}_n^*. t_m^*) \\
(t u)^* &= (t_1^* u_1^* \dots u_n^*, \dots, t_m^* u_1^* \dots u_n^*) \\
(\langle\langle a : \mathbf{b} \rangle\rangle t)^* &= (\langle\langle \bar{a}_1 : f(\mathbf{b})_1 \rangle\rangle \dots \langle\langle \bar{a}_k : f(\mathbf{b})_k \rangle\rangle t_1^*, \dots, \langle\langle \bar{a}_1 : f(\mathbf{b})_1 \rangle\rangle \dots \langle\langle \bar{a}_k : f(\mathbf{b})_k \rangle\rangle t_m^*) \\
!t^* &= (!t_1^*, \dots, !t_m^*) \\
(\text{let } !x \text{ be } u \text{ in } t)^* &= (\text{let } !x_1^* \text{ be } u_1^* \text{ in } (\dots (\text{let } !x_n^* \text{ be } u_n^* \text{ in } t_1^*) \dots), \dots, \\
&\quad \text{let } !x_1^* \text{ be } u_1^* \text{ in } (\dots (\text{let } !x_n^* \text{ be } u_n^* \text{ in } t_m^*) \dots)) \\
(\text{lup}_{\mathbf{b}, \underline{\mathbf{B}}} t)^* &= (\text{lup}_{f(\mathbf{b}), \underline{\mathbf{B}}_1^*} t_1^*, \dots, \text{lup}_{f(\mathbf{b}), \underline{\mathbf{B}}_m^*} t_m^*) \\
(\text{upd}_{\mathbf{b}, \underline{\mathbf{B}}} t)^* &= (\text{upd}_{f(\mathbf{b}), \underline{\mathbf{B}}_1^*} t_1^*, \dots, \text{upd}_{f(\mathbf{b}), \underline{\mathbf{B}}_m^*} t_m^*) \\
(\text{new}_{\mathbf{b}, \underline{\mathbf{B}}} t)^* &= (\text{new}_{f(\mathbf{b}), \underline{\mathbf{B}}_1^*} t_1^*, \dots, \text{new}_{f(\mathbf{b}), \underline{\mathbf{B}}_m^*} t_m^*) \\
\\
\text{Assuming } \mathbf{A}^* &= (\mathbf{A}_1^*, \dots, \mathbf{A}_n^*), \mathbf{B}^* = (\mathbf{B}_1^*, \dots, \mathbf{B}_m^*), f(\mathbf{b}) = (f(\mathbf{b})_1, \dots, f(\mathbf{b})_k), \\
x^* &= (x_1, \dots, x_n), t^* = (t_1^*, \dots, t_m^*) \text{ and } u^* = (u_1^*, \dots, u_n^*).
\end{aligned}$$

Fig. A.1. List translation  $(-)^* = \text{TLS}(f) : \text{TLS}(\Sigma) \rightarrow \text{TLS}(\Sigma')$ .

## References

- [1] Martín Abadi, Luca Cardelli, and Pierre-Louis Curien. Formal parametric polymorphism. *Theoretical Computer Science*, 121(1–2):9–58, December 1993.
- [2] Nick Benton and Benjamin Leperchey. Relational reasoning in a nominal semantics for storage. In *Typed Lambda Calculi and Applications, TLCA 2005*, volume 3461 of *Lecture Notes in Computer Science*, pages 86–101. Springer, 2005.
- [3] N. Bohr and L. Birkedal. Relational reasoning for recursive types and references. In Naoki Kobayashi, editor, *APLAS*, volume 4279 of *Lecture Notes in Computer Science*, pages 79–96. Springer, 2006.
- [4] B.P. Dunphy. *Parametricity as a notion of uniformity in reflexive graphs*. PhD thesis, 2004.
- [5] Brian Dunphy and Uday S. Reddy. Parametric limits. In *Proceedings of the 19th IEEE Symposium on Logic in Computer Science (LICS-04)*, pages 242–251, 2004.
- [6] M. J. Gabbay and A. M. Pitts. A new approach to abstract syntax with variable binding. *Formal Aspects of Computing*, 13:341–363, 2001.
- [7] Murdoch J. Gabbay. *A Theory of Inductive Definitions with  $\alpha$ -Equivalence*. PhD thesis, Cambridge University, UK, 2001.
- [8] Paul Blain Levy. *Call By Push Value, a Functional/ Imperative Synthesis*. Kluwer, December 2003.
- [9] Rasmus Ejlers Møgelberg and Alex Simpson. Relational parametricity for computational effects. In *LICS*, pages 346–355. IEEE Computer Society, 2007.
- [10] R.E. Møgelberg and A. Simpson. An enriched calculus for linearly used effects. Submitted, 2009.
- [11] E. Moggi. Notions of computation and monads. *Information and Computation*, 93:55–92, 1991.
- [12] P. W. O’Hearn and J. C. Reynolds. From algol to polymorphic linear lambda-calculus. *Jrnl. A.C.M.*, 47(1):167–223, January 2000.
- [13] P. W. O’Hearn and R. D. Tennent. Parametricity and local variables. *Journal of the ACM*, 42(3):658–709, May 1995.

- [14] F. J. Oles. *A category-theoretic approach to the semantics of programming languages*. PhD thesis, Syracuse University, Syracuse, N.Y., 1982.
- [15] Plotkin and Power. Notions of computation determine monads. In *FOSSACS: International Conference on Foundations of Software Science and Computation Structures*. LNCS, 2002.
- [16] Gordon Plotkin and Martín Abadi. A logic for parametric polymorphism. In *Typed lambda calculi and applications*, volume 664 of *Lecture Notes in Comput. Sci.*, pages 361–375. Springer, 1993.
- [17] Gordon D. Plotkin and A. John Power. Computational effects and operations: An overview. *Electr. Notes Theor. Comput. Sci.*, 73:149–163, 2004.
- [18] J.C. Reynolds. Types, abstraction, and parametric polymorphism. *Information Processing*, 83:513–523, 1983.
- [19] John C. Reynolds. The essence of Algol. In *Proceedings of the 1981 International Symposium on Algorithmic Languages*, pages 345–372. North-Holland, 1981.
- [20] E.P. Robinson and G. Rosolini. Reflexive graphs and parametric polymorphism. In S. Abramsky, editor, *Proc. 9th Symposium in Logic in Computer Science*, pages 364–371. I.E.E.E. Computer Society, 1994.
- [21] Mark R. Shinwell and Andrew M. Pitts. On a monadic semantics for freshness. *Theor. Comput. Sci.*, 342(1):28–55, 2005.

## B Proofs

This part of the appendix contains proofs of theorems stated in the main text. These are provided for use in the refereeing process and are not intended as part of the submission.

### B.1 Proof of Proposition 3.3

The following lemma is proved in [7].

**Lemma B.1** *The following isomorphisms hold in  $\mathbf{Nom}$ .*

$$\begin{aligned} [\mathbb{A}_{\mathbf{b}}](A \times B) &\cong [\mathbb{A}_{\mathbf{b}}]A \times [\mathbb{A}_{\mathbf{b}}]B \\ [\mathbb{A}_{\mathbf{b}}](A \rightarrow_{\text{fs}} B) &\cong [\mathbb{A}_{\mathbf{b}}]A \rightarrow_{\text{fs}} [\mathbb{A}_{\mathbf{b}}]B \\ [\mathbb{A}_{\mathbf{b}}]\mathbb{A} &\cong \mathbb{A} + 1 \end{aligned}$$

Moreover, if any  $\mathbf{a}_{\mathbf{b}}$  is fresh for any  $x$  in  $A$  (e.g. if  $A$  has trivial permutation action) then  $[\mathbb{A}_{\mathbf{b}}]A \cong A$ .

As a corollary to the lemma we immediately get

**Corollary B.2** *The nominal sets  $[\mathbb{A}_{\mathbf{b}}]\mathbb{K}$  and  $\mathbb{K}^{[\mathbf{b}]}$  are isomorphic*

**Proof.** First note that

$$\begin{aligned} [\mathbb{A}_{\mathbf{b}}]\mathbb{S} &\cong [\mathbb{A}_{\mathbf{b}}](\prod_{\mathbf{b} \in \Sigma} \mathbb{A}_{\mathbf{b}} \rightarrow_{\text{fs}} [\mathbf{b}]) \\ &\cong ([\mathbb{A}_{\mathbf{b}}](\mathbb{A}_{\mathbf{b}} \rightarrow_{\text{fs}} [\mathbf{b}])) \times [\mathbb{A}_{\mathbf{b}}](\prod_{\mathbf{b}' \neq \mathbf{b}} \mathbb{A}_{\mathbf{b}'} \rightarrow_{\text{fs}} [\mathbf{b}']) \\ &\cong ((\mathbb{A}_{\mathbf{b}} + 1) \rightarrow_{\text{fs}} [\mathbf{b}]) \times \prod_{\mathbf{b}' \neq \mathbf{b}} \mathbb{A}_{\mathbf{b}'} \rightarrow_{\text{fs}} [\mathbf{b}'] \\ &\cong [\mathbf{b}] \times \mathbb{S} \end{aligned}$$

So, by Lemma B.1,  $[\mathbb{A}_{\mathbf{b}}](R^{\mathbb{S}}) \cong R^{\mathbb{S} \times [\mathbf{b}]}$ . The left to right isomorphism maps  $k$  in  $[\mathbb{A}_{\mathbf{b}}](R^{\mathbb{S}})$  to  $\lambda \langle s, x \rangle. \text{fresh } \mathbf{a}_{\mathbf{b}}. k @ \mathbf{a}_{\mathbf{b}}(s[\mathbf{a}_{\mathbf{b}} \mapsto x])$  and the other direction maps  $h$  to

fresh  $\mathbf{a}_b$ . ( $\mathbf{a}_b. \lambda s. h(s, s(\mathbf{a}_b))$ ). We leave it to the reader to check that these restrict to an isomorphism between  $[\mathbb{A}_b]\mathbb{K}$  and  $\mathbb{K}[\mathbb{b}]$ .  $\square$

**Proof.** [Proof of Proposition 3.3] The proof is by induction on the structure of  $\underline{\mathbb{B}}$ . We prove the interesting case of atom abstraction.

$$\begin{aligned} [[\mathbb{b}]\underline{\mathbb{B}}]_{\mathcal{V}} &= [\mathbb{A}_b][\underline{\mathbb{B}}]_{\mathcal{V}} \\ &\cong [\mathbb{A}_b](\mathbb{K}[\underline{\mathbb{B}}]_c) \\ &\cong \mathbb{K}[\mathbb{b}] \times [\mathbb{A}_b][\underline{\mathbb{B}}]_c \\ &= \mathbb{K}[[\mathbb{b}]\underline{\mathbb{B}}]_c \end{aligned}$$

$\square$

Proposition 4.9 is proved similarly.

### B.2 Proof of Theorem 4.10

**Proof.** As in definition of the interpretation of terms in Section 3 we work with environments. A  $\Gamma, \Theta$ -environment  $\rho$  is a map mapping each variable  $x_i: \mathbf{A}_i$  in  $\Gamma$  to a pair of lists such that  $(\rho_0(x_i), \rho_1(x_i)) \in Q_{\mathbf{A}_i}^{\mathcal{V}}$  and each  $a_j: \mathbf{b}_j$  in  $\Theta$  to a pair  $(\rho_0(a_j), \rho_1(a_j))$  in the relation (4) and we prove by induction on  $t$  that for any such environment we get

$$Q_{\mathbf{A}}^{\mathcal{V}}(\llbracket \text{TLS}(d_0)(t) \rrbracket_{\rho_0}, \llbracket \text{TLS}(d_1)(t) \rrbracket_{\rho_1})$$

The case of variable introduction is trivial as usual. The cases of  $\lambda$ -abstraction and function application follow from the fact that the projections

$$\mathbf{Nom}_{[\mathbb{A}^0 \rightarrow \Sigma^0]} \leftarrow \mathbf{Rel} \rightarrow \mathbf{Nom}_{[\mathbb{A}^1 \rightarrow \Sigma^1]}$$

preserve the cartesian closed structure. The case of let-expressions and thunks can be proved likewise, since these are defined using the adjunction structure, which in turn in all three categories are defined using the cartesian closed structure.

For the case of atom abstraction, by induction we know that for all fresh  $\mathbf{a}_r$

$$Q_{\mathbf{A}}^{\mathcal{V}}(\llbracket \text{TLS}(d_0)t \rrbracket_{\rho_0[\bar{a} \rightarrow c_0(\mathbf{a}_r)]}, \llbracket \text{TLS}(d_1)t \rrbracket_{\rho_1[\bar{a} \rightarrow c_1(\mathbf{a}_r)]})$$

and we must show that

$$Q_{[\mathbf{r}]\mathbf{A}}^{\mathcal{V}}(\llbracket \langle \bar{a}: d_0(\mathbf{r}) \rangle \text{TLS}(d_0)(t) \rrbracket_{\rho_0}, \llbracket \langle \bar{a}: d_1(\mathbf{r}) \rangle \text{TLS}(d_1)(t) \rrbracket_{\rho_1}),$$

i.e., that for any fresh  $\mathbf{a}_r$

$$Q_{\mathbf{A}}^{\mathcal{V}}(\llbracket \langle \bar{a}: d_0(\mathbf{r}) \rangle \text{TLS}(d_0)t \rrbracket_{\rho_0} @ c_0(\mathbf{a}_r), \llbracket \langle \bar{a}: d_1(\mathbf{r}) \rangle \text{TLS}(d_1)t \rrbracket_{\rho_1} @ c_1(\mathbf{a}_r)).$$

But since for each  $i$

$$\llbracket \langle \bar{a}: d_i(\mathbf{r}) \rangle \text{TLS}(d_i)t \rrbracket_{\rho_i} @ a_i(\mathbf{a}_r) = \llbracket \text{TLS}(d_i)t \rrbracket_{\rho_i[\bar{a} \rightarrow a_i(\mathbf{a}_r)]},$$

this follows from the induction hypothesis.

This leaves the cases of the operations for the effects. Consider first the update operation. This is typed as

$$\text{upd}_{\mathbf{r}, \underline{\mathbf{B}}}: \underline{\mathbf{B}} \rightarrow \text{ref } \mathbf{r} \rightarrow \mathbf{r} \rightarrow \underline{\mathbf{B}}$$

and, leaving the isomorphism of Lemma 4.9 implicit, the induction hypothesis tells us that

$$(\llbracket \text{TLS}(d_0)(t) \rrbracket_{\rho_0}, \llbracket \text{TLS}(d_1)(t) \rrbracket_{\rho_1}) \in \mathbb{K}_{\text{Rel}}^{Q_{\underline{\mathbf{B}}}^{\mathcal{C}}},$$

i.e., for any given pair  $(\bar{x}, \bar{y}) \in Q_{\underline{\mathbf{B}}}^{\mathcal{C}}$  there exists a finite  $A \subseteq_{\text{fin}} \mathbb{A}$  such that if

$$Q_{\underline{\mathbf{B}}}^{\mathcal{V}}(s(c_0(\mathbf{a}_{\mathbf{b}})), s'(c_1(\mathbf{a}_{\mathbf{b}}))) \tag{B.1}$$

holds for all  $\mathbf{a}_{\mathbf{b}}$  in  $A$  then

$$\llbracket \text{TLS}(d_0)(t) \rrbracket_{\rho_0}(\bar{x})(s) = \llbracket \text{TLS}(d_1)(t) \rrbracket_{\rho_1}(\bar{y})(s')$$

Given further  $(\bar{n}, \bar{n}') \in Q_{\mathbf{r}}^{\mathcal{V}}$  we must show that

$$(\lambda s. \llbracket \text{TLS}(d_0)(t) \rrbracket_{\rho_0}(\bar{x})(s[c_0(\mathbf{a}_{\mathbf{b}}) \mapsto \bar{n}]), \lambda s'. \llbracket \text{TLS}(d_1)(t) \rrbracket_{\rho_1}(\bar{y})(s'[c_1(\mathbf{a}_{\mathbf{b}}) \mapsto \bar{n}']]) \in \mathbb{K}_{\text{Rel}}.$$

We can use the same  $A$  as above to verify this, because if (B.1) holds for  $(s, s')$  then it also holds for  $(s[c_0(\mathbf{a}_{\mathbf{b}}) \mapsto \bar{n}], s'[c_0(\mathbf{a}_{\mathbf{b}}) \mapsto \bar{n}']])$ . The case of lookup is similar.

For the case of the new operator, suppose  $Q_{\mathbf{r}|\underline{\mathbf{B}}}^{\mathcal{V}}(\llbracket \text{TLS}(d_0)(t) \rrbracket_{\rho_0}, \llbracket \text{TLS}(d_1)(t) \rrbracket_{\rho_1})$ . Leaving the isomorphism of Lemma 4.9 implicit, this means that for any fresh  $\mathbf{a}_{\mathbf{r}}$

$$(\llbracket \text{TLS}(d_0)(t) \rrbracket_{\rho_0} @ \mathbf{a}_{\mathbf{r}}, \llbracket \text{TLS}(d_1)(t) \rrbracket_{\rho_1} @ \mathbf{a}_{\mathbf{r}}) \in \mathbb{K}_{\text{Rel}}^{Q_{\underline{\mathbf{B}}}^{\mathcal{C}}}$$

which, as in the case of update means that if  $(\bar{x}, \bar{y}) \in Q_{\underline{\mathbf{B}}}^{\mathcal{C}}$  then there exists a finite set  $A$  of atoms such that if  $Q_{\underline{\mathbf{B}}}^{\mathcal{V}}(s(c_0(\mathbf{a}'_{\mathbf{b}})), s'(c_1(\mathbf{a}'_{\mathbf{b}})))$  holds for all  $\mathbf{a}'_{\mathbf{b}} \in A$  then

$$\llbracket \text{TLS}(d_0)(t) \rrbracket_{\rho_0} @ \mathbf{a}_{\mathbf{r}}(\bar{x})(s) = \llbracket \text{TLS}(d_1)(t) \rrbracket_{\rho_1} @ \mathbf{a}_{\mathbf{r}}(\bar{y})(s')$$

So as before, if  $(\bar{n}, \bar{n}') \in Q_{\mathbf{r}}^{\mathcal{V}}$ , and  $Q_{\underline{\mathbf{B}}}^{\mathcal{V}}(s(c_0(\mathbf{a}'_{\mathbf{b}})), s'(c_1(\mathbf{a}'_{\mathbf{b}})))$  holds for all  $\mathbf{a}'_{\mathbf{b}} \in A$  then a similar condition holds for  $(s[c_0(\mathbf{a}_{\mathbf{r}}) \mapsto \bar{n}], s'[c_1(\mathbf{a}_{\mathbf{r}}) \mapsto \bar{n}']])$  from which we conclude

$$\llbracket \text{TLS}(d_0)(t) \rrbracket_{\rho_0} @ \mathbf{a}_{\mathbf{r}}(\bar{x})(s[c_0(\mathbf{a}_{\mathbf{r}}) \mapsto \bar{n}]) = \llbracket \text{TLS}(d_1)(t) \rrbracket_{\rho_1} @ \mathbf{a}_{\mathbf{r}}(\bar{y})(s'[c_1(\mathbf{a}_{\mathbf{r}}) \mapsto \bar{n}']])$$

We have shown that  $A$  verifies

$$(\lambda s. \llbracket \text{TLS}(d_0)(t) \rrbracket_{\rho_0} @ \mathbf{a}_{\mathbf{r}}(\bar{x})(s[c_0(\mathbf{a}_{\mathbf{r}}) \mapsto \bar{n}]), \lambda s'. \llbracket \text{TLS}(d_1)(t) \rrbracket_{\rho_1} @ \mathbf{a}_{\mathbf{r}}(\bar{y})(s'[c_1(\mathbf{a}_{\mathbf{r}}) \mapsto \bar{n}']]) \in \mathbb{K}_{\text{Rel}}.$$

□

### B.3 Proof of Theorem 5.5

**Proof.** We show that  $Q_A^\mathcal{V}$  is symmetric by induction on  $A$ . We first show that  $\mathbb{K}_{\text{Rel}}$  is symmetric, so suppose  $(k_0, k_1) \in \mathbb{K}_{\text{Rel}}$  and that this is witnessed by the set  $A \subseteq_{\text{fin}} \mathbb{A}^{\text{Rel}}$ . Recall that in Section 5 a bijection  $(-)^{\text{op}}: \mathbb{A}^{\text{Rel}} \rightarrow \mathbb{A}^{\text{Rel}}$  was defined, and that it maps a name over  $\mathbf{r}$  to a name over  $\mathbf{r}^{\text{op}}$ . We will show that  $A^{\text{op}}$ , which we define to be  $\{\mathbf{a}_{\mathbf{r}^{\text{op}}} \mid \mathbf{a}_{\mathbf{r}} \in A\}$  witnesses  $(k_1, k_0) \in \mathbb{K}_{\text{Rel}}$ . So suppose  $s_0, s_1$  are states such that

$$\forall \mathbf{a}_{\mathbf{r}} \in A^{\text{op}}. Q_{\mathbf{r}}^\mathcal{V}(s_0(\text{dom}(\mathbf{a}_{\mathbf{r}})), s_1(\text{dom}(\mathbf{a}_{\mathbf{r}^{\text{op}})})). \quad (\text{B.2})$$

We must show  $k_1(s_0) = k_0(s_1)$ . But (B.2) is equivalent to

$$\forall \mathbf{a}_{\mathbf{r}} \in A. Q_{\mathbf{r}^{\text{op}}}^\mathcal{V}(s_0(\text{dom}(\mathbf{a}_{\mathbf{r}^{\text{op}})}), s_1(\text{dom}(\mathbf{a}_{\mathbf{r}}))) \quad (\text{B.3})$$

which is equivalent to

$$\forall \mathbf{a}_{\mathbf{r}} \in A. Q_{\mathbf{r}}^\mathcal{V}(s_1(\text{dom}(\mathbf{a}_{\mathbf{r}})), s_0(\text{dom}(\mathbf{a}_{\mathbf{r}^{\text{op}})})) \quad (\text{B.4})$$

which by the assumption that  $A$  witnesses  $(k_0, k_1) \in \mathbb{K}_{\text{Rel}}$  implies  $k_0(s_1) = k_1(s_0)$  as needed.

The case of  $\mathbb{K}_{\text{Rel}}$  together with the obvious fact that  $\llbracket \mathbf{b} \rrbracket_{\mathcal{R}_\mathcal{V}} = \text{eq}_{\llbracket \mathbf{b} \rrbracket}$  gives the base for the induction. In general, if  $\rho, \rho'$  are symmetric so are  $\rho \rightarrow \rho'$  and  $\rho \times \rho'$ , so this proves the induction cases for  $A \rightarrow B$  and  $!A$ .

For the case of atom abstraction we must show that  $Q_{\text{TLS}(\Delta)(\llbracket \mathbf{b} \rrbracket A)}^\mathcal{V}$  is symmetric if  $Q_{\text{TLS}(\Delta)(A)}^\mathcal{V}$  is symmetric. Since

$$\begin{aligned} Q_{\text{TLS}(\Delta)(\llbracket \mathbf{b} \rrbracket A)}^\mathcal{V} &= Q_{[\Delta(\mathbf{b})]\text{TLS}(\Delta)(A)}^\mathcal{V} \\ &= [\mathbb{A}_{\Delta(\mathbf{b})}^{\text{Rel}}]Q_{\text{TLS}(\Delta)(A)}^\mathcal{V} \end{aligned}$$

it suffices to show in general, that if  $\rho$  is symmetric, so is  $[\mathbb{A}_{\Delta(\mathbf{b})}^{\text{Rel}}]\rho$ . By definition  $([\mathbb{A}_{\Delta(\mathbf{b})}^{\text{Rel}}]\rho)(x, y)$  iff

$$\rho(x @ \text{dom}(\mathbf{a}_{\text{eq}_{\llbracket \mathbf{b} \rrbracket}}), y @ \text{dom}(\mathbf{a}_{\text{eq}_{\llbracket \mathbf{b} \rrbracket}}^{\text{op}}))$$

for all fresh  $\mathbf{a}_{\text{eq}_{\llbracket \mathbf{b} \rrbracket}}$ . Since  $\mathbf{a}_{\text{eq}_{\llbracket \mathbf{b} \rrbracket}}^{\text{op}} = \mathbf{a}_{\text{eq}_{\llbracket \mathbf{b} \rrbracket}}$  (this holds for all names over symmetric relations), symmetry of  $\rho$  implies symmetry of  $[\mathbb{A}_{\Delta(\mathbf{b})}^{\text{Rel}}]\rho$  as desired.

Finally we must show that  $Q_{\text{TLS}(\Delta)(\text{ref } \mathbf{b})}^\mathcal{V} = Q_{\text{ref } (\Delta(\mathbf{b}))}^\mathcal{V}$  is symmetric, but  $Q_{\text{ref } (\Delta(\mathbf{b}))}^\mathcal{V}$  is the relation

$$\{(\text{dom}(\mathbf{a}_{\text{eq}_{\llbracket \mathbf{b} \rrbracket}}), \text{dom}(\mathbf{a}_{\text{eq}_{\llbracket \mathbf{b} \rrbracket}}^{\text{op}})) \mid \mathbf{a}_{\text{eq}_{\llbracket \mathbf{b} \rrbracket}} \in \mathbb{A}\},$$

which in fact is the identity relation.  $\square$