Relational Reasoning for Recursive Types and References

Nina Bohr and Lars Birkedal

IT University of Copenhagen (ITU)
{ninab,birkedal}@itu.dk

Abstract. We present a local relational reasoning method for reasoning about contextual equivalence of expressions in a λ-calculus with recursive types and general references. Our development builds on the work of Benton and Leperchey, who devised a nominal semantics and a local relational reasoning method for a language with simple types and simple references. Their method uses a parameterized logical relation. Here we extend their approach to recursive types and general references. For the extension, we build upon Pitts' and Shinwell's work on relational reasoning about recursive types (but no references) in nominal semantics. The extension is non-trivial because of general references (higher-order store) and makes use of some new ideas for proving the existence of the parameterized logical relation and for the choice of parameters.

1 Introduction

Proving equivalence of programs is important for verifying the correctness of compiler optimizations and other program transformations. Program equivalence is typically defined in terms of *contextual equivalence*, which expresses that two program expressions are equivalent if they have the same observable behaviour when placed in any program context C. It is generally quite hard to show directly that two program expressions are contextually equivalent because of the universal quantification over all contexts. Thus there has been an extensive research effort to find reasoning methods that are easier to use for establishing contextual equivalence, in particular to reduce the set of contexts one has to consider, see, e.g., [7, 3, 1, 6] and the references therein. For programming languages with references, it is not enough to restrict attention to fewer contexts, since one also needs to be able to reason about equivalence under *related* stores. To address this challenge, methods based on logical relations and bisimulations have been proposed, see, e.g., [8, 2, 13]. The approaches based on logical relations have so far been restricted to deal only with simple integer references (or references to such). To extend the method to general references in typed languages, one also needs to extend the method to work in the presence of recursive types. The latter is a challenge on its own, since one cannot easily establish the existence of logical relations by induction in the presence of recursive types. Thus a number of research papers have focused on relational reasoning methods for recursive types without references, e.g., [3, 1]. Recently, the bisimulation approach has

been simplified and extended to work for untyped languages with general references [5, 4]. For effectiveness of the reasoning method, we seek *local* reasoning methods, which only require that we consider the accessible part of a store and which works in the presence of a separated (non-interfering) invariant that is preserved by the context. In [2], Benton and Leperchey developed a relational reasoning method for a language with simple references that does allow for local reasoning. Their approach is inspired by related work on separation logic [10, 9]. In particular, an important feature of the state relations of Benton and Leperchey is that they depend on only part of the store: that allows us to reason that related states are still related if we update them in parts on which the relation does not depend. In this paper we extend the work of Benton and Leperchey to relational reasoning about contextual equivalence of expressions in a typed programming language with general recursive types *and* general references (thus with higher-order store). We arrive at a useful reasoning method. In particular, we have used it to verify all the examples of [5]. We believe that the method is simple to use, but more work remains to compare the strengths and weaknesses of the method we present here with that of *loc.cit.*

Before giving an overview of the technical development, we now present two examples of pairs of programs that can easily be shown contextually equivalent with the method we develop. The examples are essentially equivalent to (or perhaps slightly more involved than) examples in [5]. Section 5 contains the proofs of contextual equivalence.

The programs M and N shown below both take a function as argument and returns two functions, set and get. In M, there is one hidden reference y, which set can use to store a function. The get function returns the contents of y. The program N uses three local references y_0, y_1 and p. The p reference holds a integer value. The set function updates p and depending on the value of p it stores its argument in either y_0 or y_1. The get function returns the contents of y_0 or y_1, depending on the value of p. Note that the programs store functions in the store. Intuitively, the programs M and N are contextually equivalent because they use *local storage*. The proof method we develop allows us to prove that they are contextually equivalent via local reasoning.

$$
\begin{array}{l}
\mathrm{M} = \mathrm{rec}\ f\ (g\colon \tau \to T\tau')\colon T(((\tau \to T\tau') \to T\mathrm{unit}) \times (\mathrm{unit} \to T(\tau \to T\tau'))) = \\
\qquad \mathrm{let}\ y \Leftarrow \mathrm{ref}\ g\ \mathrm{in} \\
\qquad \mathrm{let}\ \mathrm{set} \Leftarrow \mathrm{val}\ (\mathrm{rec}\ f_{1M}(g_1 : \tau \to T\tau') : T\mathrm{unit} = y := g_1)\ \mathrm{in} \\
\qquad \mathrm{let}\ \mathrm{get} \Leftarrow \mathrm{val}\ (\mathrm{rec}\ f_{2M}(x : \mathrm{unit}) : T(\tau \to T\tau') = !y)\ \mathrm{in} \\
\qquad\quad (\mathrm{set},\mathrm{get})
\end{array}
$$

$$
\begin{array}{l}
\mathrm{N} = \mathrm{rec}\ f\ (g\colon \tau \to T\tau')\colon T(((\tau \to T\tau') \to T\mathrm{unit}) \times (\mathrm{unit} \to T(\tau \to T\tau'))) = \\
\qquad \mathrm{let}\ y_0 \Leftarrow \mathrm{ref}\ g\ \mathrm{in} \\
\qquad \mathrm{let}\ y_1 \Leftarrow \mathrm{ref}\ g\ \mathrm{in} \\
\qquad \mathrm{let}\ p \Leftarrow \mathrm{ref}\ 0\ \mathrm{in} \\
\qquad \mathrm{let}\ \mathrm{set} \Leftarrow \mathrm{val}\ (\mathrm{rec}\ f_{1N}(g_1 : \tau \to T\tau') : T\mathrm{unit} = \\
\qquad\qquad\qquad\qquad \mathrm{if}\ \mathrm{iszero}(!p)\ \mathrm{then} \\
\qquad\qquad\qquad\qquad\quad (p := 1;\ y_1 := g_1)
\end{array}
$$

$$
\begin{array}{l}
\qquad\qquad\qquad\quad \text{else} \\
\qquad\qquad\qquad\qquad (p := 0;\, y_0 := g_1)) \text{ in} \\
\text{let get} \Leftarrow \text{val (rec } f_{2N}(x : \text{unit}) : (\tau \to T\tau') = \\
\qquad\qquad\qquad\qquad \text{if iszero}(!p) \text{ then } !y_0 \text{ else } !y_1) \text{ in} \\
\quad (\text{set},\text{get})
\end{array}
$$

Next consider the programs M' and N' below. They both have a free variable g of function type. In M', g is applied to a function that just returns unit and then M' returns the constant unit function. In N', g is applied to a function that updates a reference local to N', maintaining the invariant that the value of the local reference is always greater than zero. After the call to g, N' returns the constant unit function if the value of the local reference is greater than zero; otherwise it diverges (Ω stands for a diverging term). Intuitively, it is clear that M' and N' are contextually equivalent, since the local reference in N' initially is greater than zero and g can only update the local reference via the function it is given as argument and, indeed, we can use our method to prove formally that M' and N' are contextually equivalent via local reasoning.

$$
\begin{array}{l}
M' = \text{let } f \Leftarrow \text{val (rec } f'(a : \text{unit}) : T\text{unit} = \text{val } ()) \text{ in} \\
\qquad\quad \text{let } w \Leftarrow gf \text{ in} \\
\qquad\qquad \text{val } f
\end{array}
$$

$$
\begin{array}{l}
N' = \text{let } x \Leftarrow \text{ref } 1 \text{ in} \\
\qquad\quad \text{let } f \Leftarrow \text{val (rec } f'(a : \text{unit}) : T\text{unit}) = x := !x + 1) \text{ in} \\
\qquad\quad \text{let } w \Leftarrow gf \text{ in} \\
\qquad\qquad \text{let } z \Leftarrow \text{if iszero}(!x) \text{ then } \Omega \text{ else (rec } f'(a : \text{unit}) : T\text{unit} = \text{val } ()) \text{ in} \\
\qquad\qquad\quad \text{val z}
\end{array}
$$

We now give an overview of the technical development, which makes use of a couple of new ideas for proving the existence of the parameterized logical relation and for the choice of parameters.

In Section 2 we first present the language and in Section 3 we give a denotational semantics in the category of FM-cpo's. Adapting methods developed by Pitts [7] and Shinwell [11, 12] we prove the existence of a recursive domain in $(\text{FM-Cpo}_{\perp})^4$, $\mathbb{D} = (\mathbb{V}, \mathbb{K}, \mathbb{M}, \mathbb{S})$, such that $i : F(\mathbb{D}, \mathbb{D}) \cong \mathbb{D}$ where F is our domain constructor. The 4-tuple of domains $\mathbb{D}$ has the minimal invariant property, that is, $id_{\mathbb{D}}$ is the least fixed point of $\delta : (\mathbb{D} \to \mathbb{D}) \to (\mathbb{D} \to \mathbb{D})$ where $\delta(e) = i \circ F(e, e) \circ i^{-1}$. Denotations of values are given in $\mathbb{V}$, continuations in $\mathbb{K}$, computations in $\mathbb{M}$ and stores in $\mathbb{S}$. We show adequacy via a logical relation, the existence of which is established much as in [11].

The denotational semantics can be used to establish simple forms of contextual equivalence qua adequacy. For stronger proofs of contextual equivalences we define a parameterized relation between pairs of denotations of values, pairs of denotations of continuations, pairs of denotations of computations, pairs of denotations of stores. We can express contextual equivalence for two computations by requiring that they have the same terminaton behaviour when placed in the same arbitrary closing contexts.

Since our denotations belong to a recursive domain, the existence of the parameterized logical relation again involves a separate proof. The proof requires that the relations are preserved under approximations. On the other hand we want the parameters to express invariants for hidden local areas of related stores, and such properties of stores will not be preserved under approximations. Therefore our relations are really given by 4-tuples, which we think of as two pairs: the 4-tuples have the form (d_1', d_1, d_2', d_2), where $d_1' \sqsubseteq d_1$ and $d_2' \sqsubseteq d_2$. We can now let the approximation be carried out over the primed domain elements d_1', d_2', and preserve the invariant on the non-primed elements d_1, d_2. Correspondingly, relatedness of computations is stated as a two-sided termination approximation. Termination of application of an approximated computation m_1' to an approximated continuation k_1' and an approximated store S_1' implies termination in the other side of the non-approximated elements, $m_1' k_1' S_1' = \top \Longrightarrow m_2 k_2 S_2 = \top$, and similarly for the other direction. With this separation of approximation from the local properties that the parameters express, we can prove that the relation exists. We can then extract a binary relation, defined via reference to the 4-ary relation, such that the binary relation implies contextual equivalence.

A parameter expresses properties of two related stores; and computations are related under a parameter if they have equivalent termination behaviour when executed in stores, which preserve at least the invariants expressed by the parameter. Our parameters are designed to express relatedness of pairs in the presence of higher-order store and therefore they are somewhat more complex than the parameters used by Benton and Leperchey [2]. As we have seen in the examples above, we can prove contextual equivalence of two functions, which allocate local store in different ways, and then return functions set and get that access the hidden local storage. These local locations can be updated later by application of the exported set-functions to related arguments. In between the return of the functions and the application of the returned set-functions, there might have been built up additional local store invariants. Thus functions stored by a later call to the returned set-function may require further properties of stores in order to have equivalent behaviour, than was the case when our set and get functions were returned. To handle this possibility our parameters include pairs of locations; two stores are then related wrt. such pairs of locations if the pair of locations contain values that are related relative to the invariants that hold for the two stores.

In more detail, a parameter has the form $\Delta\{r_1, \ldots, r_n\}$. Here Δ is a store type that types a finite set of locations; these are intuitively our "visible locations." The $r_1, \ldots, r_n$ are local parameters. A local parameter r_i has its own finite area of store in each side, disjoint from the visible area and from all the other local parameters' store areas. A local parameter r_i has the form $(P_1, LL_1) \vee \cdots \vee (P_m, LL_m)$. The Ps express properties of two stores and the LLs are lists of location pairs. It is possible to decide if two states fulfill the properties expressed by the Ps by only considering the contents of r_is private areas of store. At least one P must hold and the corresponding LL must hold values related relative to the invariants that hold for the two stores (we can also think of this as related

at the given time in computation). Using FM domain theory makes it posible for us to express the parameters directly by location names.

We present the definition of our relation, state its existence and the theorem that relatedness implies contextual equivalence in Section 4. In the following Section 5 we show how we prove contextual equivalence of our example programs. We hope that the proofs will convince the reader that our logical relations proof method is fairly straightforward to apply; in particular the choice of parameters is very natural. We conclude in Section 6.

For reasons of space most proofs have been omitted from this extended abstract.

2 Language

The language we consider is a call-by-value, monadically-typed λ-calculus with recursion, general recursive types, and general dynamically allocated references. Types are either *value types* τ or *computation types* $T\tau$. Values of any closed value type can be stored in the store.

$$\begin{aligned}\tau &::= \alpha \mid \mathrm{unit} \mid \mathrm{int} \mid \tau \times \tau \mid \tau + \tau \mid \tau\mathrm{ref} \mid \tau \to T\tau \mid \mu\alpha.\tau \\ \gamma &::= \tau \mid T\tau\end{aligned}$$

Typing contexts, Γ, are finite maps from variables to closed value types. We assume infinite sets of variables, ranged over by x, type variables, ranged over by α, and locations, ranged over by l. We let $\mathbb{L}$ denote the set of locations. Store types Δ are finite maps from locations to value types. Terms G are either *values* V or *computations* M:

$$\begin{aligned}V &::= x \mid \underline{n} \mid \underline{l} \mid () \mid (V, V') \mid \mathrm{in}_i V \mid \mathrm{rec}\ f(x:\tau) = M \mid \mathrm{fold}\ V \\ M &::= VV' \mid \mathrm{let}\ x \Leftarrow M\ \mathrm{in}\ M' \mid \mathrm{val}\ V \mid \pi_i V \mid \mathrm{ref}\ V \mid !V \mid \\ &\quad V := V' \mid \mathrm{case}\ V\ \mathrm{of}\ \mathrm{in}_{1x1} \Rightarrow M_1; \mathrm{in}_{2x2} \Rightarrow M_2 \mid \\ &\quad V = V' \mid V + V' \mid \mathrm{iszero}\ V \mid \mathrm{unfold}\ V \\ G &::= M \mid V.\end{aligned}$$

Continuations K take the following form:

$$K ::= \mathrm{val}\ x \mid \mathrm{let}\ y \Leftarrow M\ \mathrm{in}\ K$$

The typing judgments take the form

$$\Delta; \Gamma \vdash V : \tau \qquad \Delta; \Gamma \vdash M : T\tau \qquad \Delta; \vdash K : (x:\tau)^{\top}$$

The typing rules for values and terms are as in [2] extended with rules for recursive types, except that the type for references is not restricted. Here we just include the following three selected rules:

$$\frac{\Delta; \Gamma \vdash V : \tau}{\Delta; \Gamma \vdash refV : T(\tau\mathrm{ref})}$$

$$\frac{\Delta; \Gamma \vdash V : \tau[\mu\alpha.\tau/\alpha]}{\Delta; \Gamma \vdash fold\ V : \mu\alpha.\tau} \qquad \frac{\Delta; \Gamma \vdash V : \mu\alpha.\tau}{\Delta; \Gamma \vdash unfold\ V : T(\tau[\mu\alpha.\tau/\alpha])}$$

Stores Σ are finite maps from locations to closed values. A store Σ has store type Δ, written $\Sigma : \Delta$, if, for all l in the domain of Δ, $\Delta; \vdash \Sigma(l) : \Delta(l)$.

The operational semantics is defined via a termination judgment $\Sigma, \text{let } x \Leftarrow M \text{ in } K \downarrow$, where M is closed and K is a *continuation term in* x. Typed continuation terms are defined by:

$$\frac{}{\Delta; \vdash val\ x : (x : \tau)^\top} \qquad \frac{\Delta; x : \tau \vdash M : T\tau' \quad \Delta; \vdash K : (y : \tau')^\top}{\Delta; \vdash let\ y \Leftarrow M\ in\ K : (x : \tau)^\top}$$

The defining rules for the termination judgment $\Sigma, \text{let } x \Leftarrow M \text{ in } K \downarrow$ are standard given that the language is call-by-value, with left-to-right evaluation order. We just include one rule as an example:

$$\frac{\Sigma, \text{let } x \Leftarrow \text{val } V \text{ in } K \downarrow}{\Sigma, \text{let } x \Leftarrow \text{unfold}(\text{fold } V) \text{ in } K \downarrow}$$

A *context* is a computation term with a hole, and we write $C[.] : (\Delta; \Gamma \vdash \gamma) \Rightarrow (\Delta; - \vdash T\tau)$ to mean that whenever $\Delta; \Gamma \vdash G : \gamma$ then $\Delta; - \vdash C[G] : T\tau$.

The definition of contextual equivalence is standard and as in [2].

Definition 1. *If* $\Delta; \Gamma \vdash G_i : \gamma$*, for* $i = 1, 2$ *then* G_1 *and* G_2 *are* contextually equivalent*, written*

$$\Delta; \Gamma \vdash G_1 =_{ctx} G_2,$$

if, for all types τ*, for all contexts* $C[.] : (\Delta; \Gamma \vdash \gamma) \Rightarrow (\Delta; - \vdash T\tau)$ *and for all stores* $\Sigma : \Delta$*,*

$$\Sigma, \text{let } x \Leftarrow C[G_1] \text{ in val } x \downarrow \iff \Sigma, \text{let } x \Leftarrow C[G_2] \text{ in val } x \downarrow .$$

3 Denotational Semantics

We define a denotational semantics of the language from the previous section and show that the semantics is *adequate*. The denotational semantics is defined using FM-domains [11]. The semantics and the adequacy proof, in particular the existence proof of the logical relation used to prove adequacy, builds on Shinwell's work on semantics of recursive types in FM-domains [11]. Our approach is slightly different from that of Shinwell since we make use of universal domains to model the fact that any type of value can be stored in the store, but technically it is a minor difference.

We begin by calling to mind some basic facts about FM-domains; see [11] for more details. Fix a countable set of atoms, which in our case will be the locations, $\mathbb{L}$. A *permutation* is a bijective function $\pi \in (\mathbb{L} \to \mathbb{L})$ such that the set $\{l \mid \pi(l) \neq l\}$ is finite. An FM-set X is a set equipped with a permutation action: an operation $\pi \bullet - : perms(\mathbb{L}) \times X \to X$ that preserves composition and identity, and such that each element $x \in X$ is finitely supported: there is a finite set $L \subset \mathbb{L}$ such that whenever π fixes each element of L, the action

of π fixes x: $\pi \bullet x = x$. There is a smallest such set, which we write $supp(x)$. A morphism of FM-sets is a function $f : D \to D'$ between the underlying sets that is equivariant: $\forall x.\pi \bullet (fx) = f(\pi \bullet x)$. An FM-cpo is an FM-set with an equivariant partial order relation $\sqsubseteq$ and least upper bounds of all finitely-supported ω-chains. A morphism of FM-cpos is a morphism of their underlying FM-sets that is monotone and preserves lubs of finitely-supported chains. We only require the existence and preservation of lubs of finitely-supported chains, so an FM-cpo may not be a cpo in the usual sense. The sets $\mathbb{Z}$, $\mathbb{N}$, etc., are discrete FM-cpos with the trivial action. The set of locations, $\mathbb{L}$, is a discrete FM-cpo with the action $\pi \bullet l = \pi(l)$. The category of FM-cpos is bicartesian closed: we write 1 and $\times$ for the finite products, $D \Rightarrow D'$ for the internal hom and 0,+ for the coproducts. The action on products is pointwise, and on functions is given by conjugation: $\pi \bullet f = \lambda x.\pi \bullet (f(\pi^{-}1 \bullet x))$. The category is not well-pointed: morphisms $1 \to D$ correspond to elements of $1 \Rightarrow D$ with empty support. The lift monad, $(-)_L$, is defined as usual with the obvious action. The Kleisli category FM-$\mathrm{Cpo}_\perp$ is the category of pointed FM-cpos (FM-cppos) and strict continuous maps, which is symmetric monoidal closed, with smash product $\otimes$ and strict function space $\multimap$. If D is a pointed FM-cpo then $\mathit{fix} : (D \Rightarrow D) \multimap D$ is defined by the lub of an ascending chain in the usual way. We write $\mathbb{O}$ for the discrete FM-cpo with elements $\perp$ and $\top$, ordered by $\perp \sqsubseteq \top$.

As detailed in [11], one may solve recursive domain equations in FM-$\mathrm{Cpo}_\perp$. For the denotational semantics, we use minimal invariant recursive domains:

$$\begin{aligned}
\mathbb{V} &\cong 1_\perp \oplus \mathbb{Z}_\perp \oplus \mathbb{L}_\perp \oplus (\mathbb{V} \oplus \mathbb{V}) \oplus (\mathbb{V} \otimes \mathbb{V}) \oplus (\mathbb{V} \multimap \mathbb{M})_\perp \oplus \mathbb{V} \\
\mathbb{K} &\cong (\mathbb{S} \multimap (\mathbb{V} \multimap \mathbb{O})) \\
\mathbb{M} &\cong (\mathbb{K} \multimap (\mathbb{S} \multimap \mathbb{O})) \\
\mathbb{S} &\cong \mathbb{L}_\perp \multimap \mathbb{V}.
\end{aligned}$$

Formally, these are obtained as the minimal invariant solution to a locally FM-continuous functor $F : (\text{FM-}\mathrm{Cpo}^4_\perp)^{\mathrm{op}} \times \text{FM-}\mathrm{Cpo}^4_\perp \to \text{FM-}\mathrm{Cpo}^4_\perp$. We write $\mathbb{D}$ for $(\mathbb{V}, \mathbb{K}, \mathbb{M}, \mathbb{S})$ and i for the isomorphism $i : F(\mathbb{D}, \mathbb{D}) \cong \mathbb{D}$. We will often omit the isomorphism i and the injections into the sum writing, e.g., simply (v_1, v_2) for an element of $\mathbb{V}$.

Types, τ are interpreted by $[\![\tau]\!] = \mathbb{V}$, computation types $T\tau$ are interpreted by $[\![T\tau]\!] = \mathbb{M}$, continuation types $(x : \tau)^\top$ are interpreted by $[\![(x : \tau)^\top]\!] = \mathbb{K}$, and store types Δ are interpreted by $[\![\Delta]\!] = \mathbb{S}$. Type environments $\Gamma = x_1 : \tau_1, \ldots, x_n : \tau_n$ are interpreted by $\mathbb{V}^n$.

Typing judgments are interpreted as follows:

- $[\![\Delta; \Gamma \vdash V : \tau]\!] \in ([\![\Gamma]\!] \multimap [\![\tau]\!])$
- $[\![\Delta; \Gamma \vdash M : T\tau]\!] \in ([\![\Gamma]\!] \multimap [\![T\tau]\!])$
- $[\![\Delta; \vdash K : (x : \tau)^\top]\!] \in \mathbb{K}$

The actual definition of the interpretations is quite standard, except for allocation which makes use of the properties of FM-cpo's:

$$\begin{aligned}
&[\![\Delta; \Gamma \vdash \mathrm{ref} V : T(\tau\mathrm{ref})]\!]\,\rho = \lambda k.\lambda S. \\
&\quad k(S([l \mapsto [\![\Delta; \Gamma \vdash V : \tau]\!]\,\rho])l \\
&\text{for some/any } l \notin supp(\lambda l'.k(S[l' \mapsto [\![\Delta; \Gamma \vdash V : \tau]\!]\,\rho])l')
\end{aligned}$$

The definition is much as in [2]. The use of FM-cpo's ensure that it is a good definition. As in [2], we use the monad T to combine state with continuations to get a good control over what the new location has to be fresh for.

We only include two additional cases of the semantic definition, namely the one for unfold and the one for continuations:

$$\begin{array}{l}[\![\Gamma \vdash \text{unfold } V : T(\tau[\mu\alpha.\tau/\alpha])]\!]\,\rho = \lambda k.\lambda S.\\ \mathit{case}\ \ [\![\Delta; \Gamma \vdash V : \mu\alpha.\tau]\!]\,\rho\ \mathit{of}\ \ i_1 \circ \mathit{in}_\mu(d)\ \mathit{then}\ kSd;\ \ \mathit{else}\bot,\end{array}$$

where in_μ is the appropriate injection of $\mathbb{V}$ into $1_\bot \oplus \mathbb{Z}_\bot \oplus \mathbb{L}_\bot \oplus (\mathbb{V} \oplus \mathbb{V}) \oplus (\mathbb{V} \otimes \mathbb{V}) \oplus (\mathbb{V} \multimap \mathbb{M})_\bot \oplus \mathbb{V}$ and i_1 is the isomorphism from this sum into $\mathbb{V}$.

$$\begin{array}{l}[\![\Delta; \vdash K : (x:\tau)^\top]\!] = \lambda S.\lambda d.\\ \quad [\![\Delta; x:\tau \vdash K : T\tau']\!]\{x \mapsto d\}(\lambda S'.(\lambda d'.\top)_\bot)_\bot S\end{array}$$

Theorem 1 (Soundness and Adequacy). *If* $\Delta; \vdash M : T\tau$, $\Delta; \vdash K : (x:\tau)^\top$, $\Sigma : \Delta$ *and* $S \in [\![\Sigma : \Delta]\!]$ *then*

$$\Sigma, \text{let } x \Leftarrow M \text{ in } K \downarrow \quad \mathit{iff} \quad [\![\Delta; \vdash M : T\tau]\!] * [\![\Delta; \vdash K : (x:\tau)^\top]\!]\, S = \top.$$

Soundness is proved by induction and to show adequacy one defines a formal approximation relation between the denotational and the operational semantics. The existence proof of the relation is non-trivial because of the recursive types, but follows from a fairly straightforward adaptation of Shinwell's existence proof in [11] (Shinwell shows adequacy for a language with recursive types, but without references).

Corollary 1. $[\![\Delta; \Gamma \vdash G_1 : \gamma]\!] = [\![\Delta; \Gamma \vdash G_2 : \gamma]\!]$ *implies* $\Delta; \Gamma \vdash G_1 =_{ctx} G_2$.

4 A Parameterized Logical Relation

In this section we define a parameterized logical relation on $\mathbb{D}$ and $F(\mathbb{D}, \mathbb{D})$, which we can use to prove contextual equivalence. (In the following we will sometimes omit the isomorphism i, i^{-1} between $F(\mathbb{D}, \mathbb{D})$ and $\mathbb{D}$).

4.1 Accessibility maps, simple state relations and parameters

Intuitively, the parameters express properties of two related states by expressing requirements of disjoint areas of states. There is a "visible" area and a finite number of "hidden invariants." In the logical relation, computations are related under a parameter if they have corresponding termination behaviour under the assumption that they are executed in states satisfying the properties expressed by the parameter.

Definition 2. *A function* $A : \mathbb{S} \to \mathcal{P}_{\text{fin}}(\mathbb{L})$ *from* $\mathbb{S}$ *to the set of finite subsets of* $\mathbb{L}$ *is an* accessibility map *if*

$$\forall S_1, S_2.\ (\forall l \in A(S_1).\ S_1 l = S_2 l) \Rightarrow A(S_1) = A(S_2)$$

We let $A_\emptyset$ denote the accessibility map defined by $\forall S.A_\emptyset(S) = \emptyset$, and we let $A_{\{l_1,\ldots,l_k\}}$ denote the accessibility map defined by $\forall S.A_{\{l_1,\ldots,l_k\}}(S) = \{l_1,\ldots,l_k\}$.

Definition 3. *A* simple state relation P *is a triple* $(\hat{p}, A_{p1}, A_{p2})$ *satisfying that* A_{p1} *and* A_{p2} *are accessibility maps and* $\hat{p}$ *is a relation on* $\mathbb{S}$ *satisfying, for all states* $S_1, S_2, S_1', S_2' \in \mathbb{S}$*,*

$$\big(\forall l_1 \in A_{p1}(S_1).S_1 l_1 = S_1' l_1 \ \wedge\ \forall l_2 \in A_{p2}(S_2).S_2 l_2 = S_2' l_2\big) \\ \Rightarrow \big((S_1, S_2) \in \hat{p} \Leftrightarrow (S_1', S_2') \in \hat{p}\big).$$

Note that a simple state relation is essentially a relation on states for which it can be decided whether a pair of states belong to the relation only on the basis of some parts of the states, defined by a pair of accessibility maps.

We denote the "always true" simple state relation $(\mathbb{S} \times \mathbb{S}, A_\emptyset, A_\emptyset)$ by T.

We now define the notion of a local parameter, which we will later use to express hidden invariants of two related states. Intuitively, a local parameter has its own private areas of the states. These areas are used for testing conditions and for storing related values. The testing condition is a disjunction of simple state relations, where to each disjunct there is an associated list of pairs of locations from the two related states. At least one condition must be satisfied and the corresponding list of locations hold related values.

Definition 4. *A* local parameter r *is a finite non-empty set of pairs* $\{(P_1, LL_1), .., (P_m, LL_m)\}$*, where each* P_i *is a simple state relation* $P_i = (\hat{p}_i, A_{pi1}, A_{pi2})$ *and*
each LL_i *is a finite set of location pairs and closed value types* $LL_i = \{\ (l_{i11}, l_{i12}, \tau_{i1}), \ldots, (l_{in_i1}, l_{in_i2}, \tau_{n_i})\ \}$*.* $(n_i \geq 0)$*.*

We often write a local parameter as $r = ((P_1, LL_1) \vee \ldots \vee (P_m, LL_m))$. For a location list LL, we write L_1 resp. L_2 for the set of locations that occur as first resp. second components in the location list LL. For a local parameter r, there are associated accessibility maps A_{r1} and A_{r2} given by $\forall S.\ A_{r1}(S) = \bigcup_i A_{pi1}(S) \cup L_1$ and $\forall S.\ A_{r2}(S) = \bigcup_i A_{pi2}(S) \cup L_2$.

We denote the "always true" local parameter $\{(T, \emptyset)\}$ also simply by T. It has the associated accessibility maps $A_\emptyset, A_\emptyset$.

As explained in the introduction we have included the LL-list to be used for storing related values which may later be updated by exported updating functions. The updated values may require more invariants to hold for the stores in order to have equivalent behaviour. This interpretation of the local parameter is expressed in the definition of our invariant relation $F(\nabla, \nabla)$ below.

Definition 5. *A* parameter Δr *is a pair* (Δ, r)*, with* Δ *a store type, and* $r = \{r_1, .., r_n\}$ *a finite set of local parameters such that* $T \in r$*.*

For a parameter Δr we associate accessibility maps A_{r1} and A_{r2}, given by $\forall S.\ A_{r1}(S) = \bigcup A_{r_i1}(S)$ and $\forall S.\ A_{r2}(S) = \bigcup A_{r_i2}(S)$.

For each store type Δ we have a special the "always true" parameter $\Delta id_\emptyset = \Delta\{T\}$.

Definition 6. *For parameters $\Delta' r'$ and Δr define*
$\Delta' r' \rhd \Delta r \overset{def}{\iff} \Delta' \supseteq \Delta$ *and* $r' \supseteq r$.

The ordering relation $\rhd$ is reflexive, transitive and antisymmetric. For all parameters Δr it holds that there are only finitely many parameters $\Delta_0 r_0$ such that $\Delta r \rhd \Delta_0 r_0$. For convenience we sometimes write $\Delta r \lhd \Delta' r'$ for $\Delta' r' \rhd \Delta r$.

4.2 Parameterized relations and contextual equivalence

In this section we will define a parameterized logical relation on $\mathbb{D}$ and $F(\mathbb{D},\mathbb{D})$. Let $D = (D_V, D_K, D_M, D_S) \in \{\mathbb{D}, F(\mathbb{D},\mathbb{D})\}$. We define the set of relations $\mathcal{R}(D)$ on D as follows.
$\mathcal{R}(D) = \hat{R}_V \times \hat{R}_K \times \hat{R}_M \times \hat{R}_S$ where

$\hat{R}_V$ = all subsets of
$D_V^4 \times \{\tau \mid \tau \text{ is a closed value type}\} \times \{\text{parameter}\}$ that include
$\{(\bot, v_1, \bot, v_2, \tau, \Delta r) \mid v_1, v_2 \in D_V,\ \tau$ closed value type, Δr parameter$\}$
$\hat{R}_K$ = all subsets of
$D_K^4 \times \{(x:\tau)^\top \mid (x:\tau)^\top \text{ is a continuation type}\} \times \{\text{parameter}\}$ that include
$\{(\bot, k_1, \bot, k_2, (x:\tau)^\top, \Delta r) \mid$
$k_1, k_2 \in D_K, (x:\tau)^\top$ continuation type, Δr parameter$\}$
$\hat{R}_M$ = all subsets of
$D_M^4 \times \{T\tau \mid T\tau \text{ is a closed computation type}\} \times \{\text{parameter}\}$ that include
$\{(\bot, m_1, \bot, m_2, T\tau, \Delta r) \mid$
$m_1, m_2 \in D_M, T\tau$ closed computation type, Δr parameter$\}$
$\hat{R}_S$ = all subsets of $D_S^4 \times \{\text{parameter}\}$ that include
$\{(\bot, S_1, \bot, S_2, \Delta r) \mid S_1, S_2 \in D_S, \Delta r$ parameter$\}$

A relation $(R_1, R_2, R_3, R_4) \in \mathcal{R}(D)$ is *admissible* if,
for each i, R_i is closed under least upper bounds of finitely supported chains of the form $(d_1^i, d_1, d_2^i, d_2, (type), \Delta r)_{i\in\omega}$ where $d_1, d_2, type, \Delta r$ are constant. We let $\mathcal{R}_{adm}(D)$ denote the admissible relations over D.

Theorem 2. *There exists a relational lifting of the functor F to $(\mathcal{R}(\mathbb{D})^{\mathrm{op}} \times \mathcal{R}(\mathbb{D})) \to \mathcal{R}(F(\mathbb{D},\mathbb{D}))$ and an* admissible *relation $\nabla = (\nabla_V, \nabla_K, \nabla_M, \nabla_S) \in \mathcal{R}_{adm}(\mathbb{D})$ satisfying the equations in Figure 1 and $(i,i) : F(\nabla,\nabla) \subset \nabla \ \wedge\ (i^{-1}, i^{-1}) : \nabla \subset F(\nabla,\nabla)$.*

Proof (Theorem 2, existence of an invariant relation ∇). The proof makes use of the ideas mentioned in the Introduction in combination with a proof method inspired from Pitts [7]. We have defined a relational structure on the domains $\mathbb{D}$ and $F(\mathbb{D},\mathbb{D}) \in \text{FM-Cpo}_\bot^4$ as products of relations on each of their four domain-projections. Each of these relations is a 4-ary relation with elements $(d_1', d_1, d_2', d_2, (type), \Delta r)$ where $d_1' = d_2' = \bot$ relates to everything.

We define the action of $F(-,+)$ on relations $R^-, R^+ \in \mathbb{D}$ such that it holds that $d_1' \sqsubseteq d_1$ and $d_2' \sqsubseteq d_2$ in elements $(d_1', d_1, d_2', d_2, (type), \Delta r)$ of $F(R^-, R^+)_n$,

$$
\begin{array}{lll}
F(\nabla,\nabla)_V & = & \{(\bot,\ v_1,\ \bot,\ v_2,\ \tau,\ \Delta r)\ \} \cup \\
& & \{(v_1',\ v_1,\ v_2',\ v_2, \tau, \Delta r)\ | \\
& & \quad v_1' \sqsubseteq v_1 \neq \bot\ \wedge\ v_2' \sqsubseteq v_2 \neq \bot\ \wedge \\
& & \quad (v_1',\ v_1,\ v_2',\ v_2, \tau, \Delta r) \in \Diamond\ \} \\
& & \text{where} \\
\Diamond & = & \{(in_1 *,\ in_1 *,\ in_1 *,\ in_1 *,\ \text{unit},\ \Delta r)\ \} \cup \\
& & \{(in_{\mathbb{Z}} n,\ in_{\mathbb{Z}} n,\ in_{\mathbb{Z}} n,\ in_{\mathbb{Z}} n,\ \text{int},\ \Delta r)\ |\ n \in \mathbb{Z}\ \} \cup \\
& & \{(in_{\mathbb{L}} l,\ in_{\mathbb{L}} l,\ in_{\mathbb{L}} l,\ in_{\mathbb{L}} l,\ (\Delta l)\text{ref},\ \Delta r)\ |\ l \in dom(\Delta)\ \} \cup \\
& & \{(in_{\oplus} in_i d_1',\ in_{\oplus} in_i d_1,\ in_{\oplus} in_i d_2',\ in_{\oplus} in_i d_2,\ \tau_1 + \tau_2,\ \Delta r)\ | \\
& & \quad \exists \Delta_0 r_0 \lhd \Delta r.\ (d_1',\ d_1,\ d_2,\ d_2,\ \tau_i,\ \Delta_0 r_0) \in \nabla_V,\ i \in \{1,2\}\ \} \cup \\
& & \{(in_{\otimes}(d_{1a}', d_{1b}'),\ in_{\otimes}(d_{1a}, d_{1b}),\ in_{\otimes}(d_{2a}', d_{2b}'),\ in_{\otimes}(d_{2a}, d_{2b}), \\
& & \quad \tau_a \times \tau_b,\ \Delta r)\ | \\
& & \quad \exists \Delta_0 r_0 \lhd \Delta r.\ (d_{1a}',\ d_{1a},\ d_{2a}',\ d_{2a}',\ \tau_a,\ \Delta_0 r_0) \in \nabla_V \text{ and} \\
& & \qquad (d_{1b}',\ d_{1b},\ d_{2b}',\ d_{2b},\ \tau_b,\ \Delta_0 r_0) \in \nabla_V\ \} \cup \\
& & \{(in_{\multimap} d_1',\ in_{\multimap} d_1,\ in_{\multimap} d_2'\ in_{\multimap} d_2,\ \tau \to \mathrm{T}\tau',\ \Delta r)\ | \\
& & \quad \forall \Delta' r' \rhd \Delta r,\ (v_1'\ , v_1,\ v_2',\ v_2,\ \tau,\ \Delta' r') \in \nabla_V. \\
& & \qquad (d_1' v_1',\ d_1 v_1,\ d_2' v_2',\ d_2 v_2,\ \mathrm{T}\tau',\ \Delta' r') \in \nabla_M\ \} \cup \\
& & \{(in_{\mu} d_1',\ in_{\mu} d_1,\ in_{\mu} d_2',\ in_{\mu} d_2,\ \mu\alpha.\tau, \Delta r)\ | \\
& & \quad \exists \Delta_0 r_0 \lhd \Delta r.\ (d_1',\ d_1,\ d_2',\ d_2,\ \tau[\mu\alpha.\tau/\alpha],\ \Delta_0 r_0) \in \nabla_V\ \}
\end{array}
$$

$$
\begin{array}{lll}
F(\nabla,\nabla)_K & = & \{(k_1',\ k_1,\ k_2',\ k_2,\ (x:\tau)^{\top},\ \Delta r)\ | \\
& & \quad k_1' \sqsubseteq k_1\ \wedge\ k_2' \sqsubseteq k_2\ \wedge\ \forall \Delta' r' \rhd \Delta r. \\
& & \quad \forall (S_1',\ S_1,\ S_2',\ S_2,\ \Delta' r') \in \nabla_S. \\
& & \quad \forall (v_1',\ v_1,\ v_2',\ v_2,\ \tau,\ \Delta' r') \in \nabla_V. \\
& & \qquad (k_1' S_1' v_1' = \top \Rightarrow k_2 S_2 v_2 = \top) \wedge \\
& & \qquad (k_2' S_2' v_2' = \top \Rightarrow k_1 S_1 v_1 = \top)\ \}
\end{array}
$$

$$
\begin{array}{lll}
F(\nabla,\nabla)_M & = & \{(m_1',\ m_1,\ m_2',\ m_2,\ T\tau,\ \Delta r)\ | \\
& & \quad m_1' \sqsubseteq m_1\ \wedge\ m_2' \sqsubseteq m_2\ \wedge\ \forall \Delta' r' \rhd \Delta r. \\
& & \quad \forall (k_1',\ k_1,\ k_2',\ k_2,\ (x:\tau)^{\top},\ \Delta' r') \in \nabla_K. \\
& & \quad \forall (S_1',\ S_1,\ S_2',\ S_2,\ \Delta' r') \in \nabla_S\ . \\
& & \qquad (m_1' k_1' S_1' = \top \Rightarrow m_2 k_2 S_2 = \top) \wedge \\
& & \qquad (m_2' k_2' S_2' = \top \Rightarrow m_1 k_1 S_1 = \top)\ \}
\end{array}
$$

$$
\begin{array}{lll}
F(\nabla,\nabla)_S & = & \{(\bot,\ S_1,\ \bot,\ S_2,\ \Delta r)\ \} \cup \\
& & \{(S_1',\ S_1,\ S_2',\ S_2,\ \Delta r)\ |\ r = \{r_1, \ldots, r_n\}\ \wedge \\
& & \quad S_1' \sqsubseteq S_1 \neq \bot\ \ \wedge\ S_2' \sqsubseteq S_2 \neq \bot\ \wedge\ \forall i \neq j,\ i,j \in 1,\ldots,n. \\
& & \quad A_{ri1}(S_1) \cap A_{rj1}(S_1) = \emptyset\ \wedge\ A_{ri2}(S_2) \cap A_{rj2}(S_2) = \emptyset\ \wedge \\
& & \quad dom(\Delta) \cap A_{r1}(S_1) = \emptyset\ \wedge\ dom(\Delta) \cap A_{r2}(S_2) = \emptyset\ \wedge \\
& & \quad \forall l \in dom(\Delta).(S_1' l,\ S_1 l,\ S_2' l,\ S_2 l,\ \Delta l, \Delta r) \in \nabla_V\ \wedge \\
& & \quad \forall r_a \in r. \exists (P_b, LL_b) \in r_a.\ (S_1, S_2) \in \hat{p}_b\ \wedge \\
& & \qquad \forall (l_1, l_2, \tau) \in LL_b.(S_1' l_1, S_1 l_1, S_2' l_2, S_2 l_2, \tau, \Delta r) \in \nabla_V
\end{array}
$$

Fig. 1. Invariant Relation ∇

$n \in \{V, K, M, S\}$. In the definition of $F(R^-, R^+)_S \in \mathcal{R}(i^{-1}\mathbb{S})$ the accessibility maps and the simple state relations mentioned in a parameter Δr are only used on the non-primed elements s_1, s_2 from $(s'_1, s_1, s'_2, s_2, \Delta r)$. As explained, approximation will be carried out on the primed domain elements. Therefore, we define application of a pair of functions (f, j) to a relation only for $f \sqsubseteq j$ with j an isomorphism $j \in \{i, i^{-1}, id_{\mathbb{D}}, id_{F(\mathbb{D},\mathbb{D})}\}$. In an application $(f, j)R$ we apply f to the elements in the primed positions, and j to the elements of the non-primed positions. Then we define $(f, j) : R \subset S$ to mean that set theoretically $(f, j)R \subseteq S$. It holds that $F(R^-, R^+)$ preserves admissibility of R^+. It also holds that $R^-, R^+, S^-, S^+ \in \mathcal{R}(\mathbb{D})$ with $(f^-, id_{\mathbb{D}}) : S^- \subset R^-$ and $(f^+, id_{\mathbb{D}}) : R^+ \subset S^+$ implies $(F(f^-, f^+), id_{F(\mathbb{D},\mathbb{D})}) : F(R^-, R^+) \subset F(S^-, S^+)$. These properties are essential for the proof of existence of the invariant relation ∇.

Proposition 1 (Weakening). *For all $\Delta' r' \rhd \Delta r$,*

- $(v'_1, v_1, v'_2, v_2, \tau, \Delta r) \in \nabla_V \Rightarrow (v'_1, v_1, v'_2, v_2, \tau, \Delta' r') \in \nabla_V$,
- $(k'_1, k_1, k'_2, k_2, (x : \tau)^\top, \Delta r) \in \nabla_K \Rightarrow (k'_1, k_1, k'_2, k_2, (x : \tau)^\top, \Delta' r') \in \nabla_K$,
- $(m'_1, m_1, m'_2, m_2, T\tau, \Delta r) \in \nabla_M \Rightarrow (m'_1, m_1, m'_2, m_2, T\tau, \Delta' r') \in \nabla_M$.

Below we define a binary relation between denotations of typing judgement conclusions. This relation will be used as basis for proofs of contextual equivalence. The relation is defined by reference to the 4-ary relations from ∇. For two closed terms, two continuations, or two states the binary relation requires that their denotations d_1, d_2 are related as two pairs $(d_1, d_1, d_2, d_2, (type), parameter) \in \nabla_j$. The denotations of open value-terms with n free variables belong to $\mathbb{V}^n \multimap \mathbb{V}$, denotations of open computation terms to$\mathbb{V}^n \multimap \mathbb{M}$. They must give related elements in ∇ whenever they are applied to n-tuples of ∇-related elements form $\mathbb{V}$.

Definition 7 (Relating denotations of open expressions).

- *For all $\Gamma = x_1 : \tau_1, \ldots, x_n : \tau_n$ and $\Delta; \Gamma \vdash V_1 : \tau$ and $\Delta; \Gamma \vdash V_2 : \tau$ let $v_1 = [\![\Delta; \Gamma \vdash V_1 : \tau]\!]$ and $v_2 = [\![\Delta; \Gamma \vdash V_2 : \tau]\!]$, and define*

$$\begin{array}{l}(v_1, v_2, \tau, \Delta r) \in \nabla_V^\Gamma \stackrel{def}{\Longleftrightarrow} \\ \quad \forall \Delta' r' \rhd \Delta r. \forall i \in \{1, \ldots, n\}. \forall (v'_{1i}, v_{1i}, v'_{2i}, v_{2i}, \tau_i, \Delta' r') \in \nabla_V. \\ \quad\quad (v_1(\overline{v'_{1i}}), v_1(\overline{v_{1i}}), v_2(\overline{v'_{2i}}), v_2(\overline{v_{2i}}), \tau, \Delta' r') \in \nabla_V.\end{array}$$

- *For all $\Gamma = x_1 : \tau_1, \ldots, x_n : \tau_n$, $\Delta; \Gamma \vdash M_1 : T\tau$ and $\Delta; \Gamma \vdash M_2 : T\tau$, let $m_1 = [\![\Delta; \Gamma \vdash M_1 : T\tau]\!]$ and $m_2 = [\![\Delta; \Gamma \vdash M_2 : T\tau]\!]$, and define*

$$\begin{array}{l}(m_1, m_2, T\tau, \Delta r) \in \nabla_M^\Gamma \stackrel{def}{\Longleftrightarrow} \\ \quad \forall \Delta' r' \rhd \Delta r. \forall i \in \{1, \ldots, n\}. \forall (v'_{1i}, v_{1i}, v'_{2i}, v_{2i}, \tau_i, \Delta' r') \in \nabla_V. \\ \quad\quad (m_1(\overline{v'_{1i}}), m_1(\overline{v_{1i}}), m_2(\overline{v'_{2i}}), m_2(\overline{v_{2i}}), T\tau, \Delta' r') \in \nabla_M.\end{array}$$

- *For all $\Delta; \vdash K_1 : (x : \tau)^\top$ and $\Delta; \vdash K_2 : (x : \tau)^\top$, let $k_1 = [\![\Delta; \vdash K_1 : (x : \tau)^\top]\!]$ and $k_2 = [\![\Delta; \vdash K_2 : (x : \tau)^\top]\!]$, and define*

$$(k_1, k_2, (x : \tau)^\top, \Delta r) \in \nabla_K^\emptyset \stackrel{def}{\Longleftrightarrow} (k_1, k_1, k_2, k_2, (x : \tau)^\top, \Delta r) \in \nabla_K.$$

– *For all* $\Sigma_1 : \Delta$, $\Sigma_2 : \Delta$, *let* $S_1 \in [\![\Sigma_1 : \Delta]\!]$ *and* $S_2 \in [\![\Sigma_2 : \Delta]\!]$, *and define*

$$(S_1, S_2, \Delta r) \in \nabla_S^{\emptyset} \stackrel{def}{\Longleftrightarrow} (S_1, S_1, S_2, S_2, \Delta r) \in \nabla_S.$$

Lemma 1.

1. *Suppose* $(m_1, m_2, T\tau, \Delta r) \in \nabla_M^{\Gamma}$. *We then have that*

$$\begin{array}{l} \forall \Delta' r' \rhd \Delta r. \forall (v_{1j}, v_{2j}, \tau_j, \Delta' r') \in \nabla_V^{\emptyset}. \forall j \in \{1, \ldots, n\}. \\ \forall (k_1, k_2, (x:\tau)^{\top}, \Delta' r') \in \nabla_K^{\emptyset}. \forall (S_1, S_2, \Delta' r') \in \nabla_S^{\emptyset}. \\ \quad (i^{-1}(m_1(\overline{v_{1j}})))k_1 S_1 = \top \Longleftrightarrow (i^{-1}(m_2(\overline{v_{2j}})))k_2 S_2 = \top. \end{array}$$

Theorem 3 (Fundamental Theorem). *For all parameters* Δr *it holds that*

– *if* $\Delta; \Gamma \vdash V : \tau$ *then* $([\![\Delta; \Gamma \vdash V : \tau]\!], [\![\Delta; \Gamma \vdash V : \tau]\!], \tau, \Delta r) \in \nabla_V^{\Gamma}$,
– *if* $\Delta; \Gamma \vdash M : T\tau$ *then* $([\![\Delta; \Gamma \vdash M : T\tau]\!], [\![\Delta; \Gamma \vdash M : T\tau]\!], T\tau, \Delta r) \in \nabla_M^{\Gamma}$.

The Fundamental Theorem is proved in the standard way by showing that all the typing rules preserve relatedness in ∇^{Γ}; weakening (Proposition 1) is used in several proof cases.

Lemma 2.

– $\forall r.\ ([\![\Delta; \vdash val\ x : (x:\tau)^{\top}]\!], [\![\Delta; \vdash val\ x : (x:\tau)^{\top}]\!], (x:\tau)^{\top}, \Delta r) \in \nabla_K^{\emptyset}$,
– *if* $S \in [\![\Delta]\!]$ *then* $(S, S, \Delta id_{\emptyset}) \in \nabla_S^{\emptyset}$.

The following theorem expresses that we can show two computations or two values to be contextually equivalent by showing that they are related in ∇^{Γ} under a parameter $\Delta id_{\emptyset}$, which does not require that any hidden invariants hold for states. The computations may themselves be able to build up local state invariants and a proof of relatedness will often require one to express these invariants; see the examples in the next section.

Theorem 4 (Contextual Equivalence). *Let* $C[\_\,] : (\Delta; \Gamma \vdash \gamma) \Rightarrow (\Delta; \vdash T\tau)$ *be a context. If* $\Delta; \Gamma \vdash G_1 : \gamma$ *and* $\Delta; \Gamma \vdash G_2 : \gamma$ *and*

$$([\![\Delta; \Gamma \vdash G_1 : \gamma]\!], [\![\Delta; \Gamma \vdash G_2 : \gamma]\!], \gamma, \Delta id_{\emptyset}) \in \nabla_j^{\Gamma},\ j \in \{V, M\}$$

then

$$\forall \Sigma : \Delta.\ (\Sigma, let\ x \Leftarrow C[G_1]\ in\ val\ x \downarrow \Longleftrightarrow \Sigma, let\ x \Leftarrow C[G_2]\ in\ val\ x \downarrow).$$

5 Examples

Before presenting our examples, we will first sketch how a typical proof of contextual equivalence proceeds. Thus, suppose we wish to show that two computations m_1 and m_2 are contextually equivalent. We then need to show that they are related in a parameter $\Delta id_{\emptyset}$ or, equivalently, in Δr, for any r. This requires us to

show, for any extended parameter $\Delta^1 r^1$, any pair[1] of continuations k_1 and k_2 related in $\Delta^1 r^1$, and any pair of states S_1 and S_2 related in $\Delta^1 r^1$, $m_1 k_1 S_1$ and $m_2 k_2 S_2$ have the same termination behaviour. The latter amounts to showing that $k_1(S_1[\ldots])v_1$ and $k_2(S_2[\ldots])v_2$ have the same termination behaviour, where $S_1[\ldots]$ and $S_2[\ldots]$ are potentially updated versions of S_1 and S_2; and v_1 and v_2 are values. Since k_1 and k_2 are assumed related in $\Delta^1 r^1$, it suffices to define a parameter $\Delta^2 r^2$ extending $\Delta^1 r^1$ and show that $S_1[\ldots]$ and $S_2[\ldots]$ are related in $\Delta^2 r^2$ and that v_1 and v_2 are related in $\Delta^2 r^2$. Typically, the definition of the parameter $\Delta^2 r^2$ essentially consists of defining one or more local parameters, which capture the intuition for why the computations are related.

In the first example below we prove that M and N from the Introduction are contextually equivalent. In this case, the only local parameter we have to define is $\tilde{r}^3 = ((P_1, LL_1) \vee (P_2, LL_2))$, where

$$\begin{array}{ll} P_1 = (\{(S_1, S_2) \mid S_2 l_p = 0\}, A_\emptyset, A_{\{l_p\}}), & LL_1 = \{(l_y, l_{y0})\}, \\ P_2 = (\{(S_1, S_2) \mid S_2 l_p = n \neq 0\}, A_\emptyset, A_{\{l_p\}}), & LL_2 = \{(l_y, l_{y1})\}. \end{array}$$

This local parameter expresses that, depending on the value of $S_2(l_p)$, either the locations (l_y, l_{y0}) or the locations (l_y, l_{y1}) contain related values.

In the first subsection below we present the proof of contextual equivalence of M and N in detail. Formally, there are several cases to consider, but do note that the proof follows the outline given above and is almost automatic except for the definition of the local parameter shown above.

5.1 Example 1

Consider the programs M and N from the Introduction.

We want to show that M and N are related in any parameter Δr, that is $\forall \Delta r.\ ([\![\emptyset; \vdash M : \sigma]\!], [\![\emptyset; \vdash N : \sigma]\!]), \sigma, \Delta r) \in \nabla_V^\emptyset$. Here $\sigma = (\tau \to T\tau') \to T(\sigma_1 \times \sigma_2)$, and $\sigma_1 = (\tau \to T\tau') \to T\mathrm{unit}$ and $\sigma_2 = \mathrm{unit} \to (\tau \to T\tau')$). As M and N are values of function type, their denotations have the forms $in_{\multimap} d_M$ and $in_{\multimap} d_N$. We need to show $\forall \Delta^1 r^1 \rhd \Delta r. \forall (v_1', v_1, v_2', v_2, \tau \to T\tau', \Delta^1 r^1) \in \nabla_V.\ (d_M v_1', d_M v_1, d_N v_2', d_N v_2, T(\sigma_1 \times \sigma_2), \Delta^1 r^1) \in \nabla_M$.

It suffices to show that $\forall \Delta^2 r^2 \rhd \Delta^1 r^1. \forall (k_1', k_1, k_2', k_2, (x : \sigma_1 \times \sigma_2)^\top, \Delta^2 r^2) \in \nabla_K.$ $\forall (S_1', S_1, S_2', S_2, \Delta^2 r^2) \in \nabla_S$ it holds that $(d_M v_1') k_1' S_1' = \top \Longrightarrow (d_N v_2) k_2 S_2 = \top$ and $(d_N v_2') k_2' S_2' = \top \Longrightarrow (d_M v_1) k_1 S_1 = \top$.

$$\text{Now,} (d_M v_1) k_1 S_1 = k_1(S_1[l_y \mapsto v_1])([\![\emptyset; y \vdash recf_{1M}]\!](y \mapsto l_y), [\![\emptyset; y \vdash recf_{2M}]\!](y \mapsto l_y)),$$

where l_y is a location that is fresh wrt. the store S_1 in combination with the parameter $\Delta^2 r^2$, i.e.,

$$l_y \notin dom(\Delta^2) \cup A_{r^2 1}(S_1). \tag{1}$$

The value of $(d_M v_1') k_1' S_1'$ is similar.
Moreover,

$$\begin{array}{l} (d_N v_2) k_2 S_2 = k_2\,(S_2[l_p \mapsto in_{\mathbb{Z}} 0, l_{y0} \mapsto v_2, l_{y1} \mapsto v_2]) \\ \qquad\qquad ([\![\emptyset; p, y_0, y_1 \vdash recf_{1N}]\!](p \mapsto l_p, y_0 \mapsto l_{y0}, y_1 \mapsto l_{y1}), \\ \qquad\qquad\ [\![\emptyset; p, y_0, y_1 \vdash recf_{2N}]\!](p \mapsto l_p, y_0 \mapsto l_{y0}, y_1 \mapsto l_{y1})), \end{array}$$

[1] Formally, we consider 4-tuples.

where l_p, l_{y0}, l_{y1} are locations that are fresh wrt. the store S_2 in combination with the parameter $\Delta^2 r^2$, i.e

$$l_p, l_{y0}, l_{y1} \notin dom(\Delta^2) \cup A_{r^2 2}(S_2). \tag{2}$$

The value of $(d_N v_2')k_2' S_2'$ is similar.

Since the continuations are related in the parameter $\Delta^2 r^2$ it suffices to show that, if $S_1' \neq \bot \vee S_2' \neq \bot$ then we can give an extended parameter $\Delta^3 r^3 \rhd \Delta^2 r^2$ such that the updated states and the values (pairs of (set,get)) are related in the extended parameter $\Delta^3 r^3$.

We let $\Delta^3 r^3 = \Delta^2(r^2 \cup \{\tilde{r}^3\})$, where $\tilde{r}^3 = ((P_1, LL_1) \vee (P_2, LL_2))$, and

$$\begin{array}{ll} P_1 = (\{(S_1, S_2) \mid S_2 l_p = 0\}, A_\emptyset, A_{\{l_p\}}), & LL_1 = \{(l_y, l_{y0}, \tau \to T\tau')\}, \\ P_2 = (\{(S_1, S_2) \mid S_2 l_p = n \neq 0\}, A_\emptyset, A_{\{l_p\}}), & LL_2 = \{(l_y, l_{y1}, \tau \to T\tau')\}. \end{array}$$

Recall $\forall S.\ A_\emptyset(S) = \emptyset \ \wedge\ \forall S.\ A_{\{l_p\}}(S) = \{l_p\}$.
Then it holds that the accessibility maps associated with the local parameter $\tilde{r}^3$, are given by $\forall S. A_{\tilde{r}^3 1}(S) = \{l_y\}$ and $\forall S. A_{\tilde{r}^3 2}(S) = \{l_p, l_{y0}, l_{y1}\}$.

We now verify that

$$\begin{array}{l} (\, S_1'[l_y \mapsto v_1'], S_1[l_y \mapsto v_1], S_2'[l_p \mapsto in_{\mathbb{Z}} 0, l_{y0} \mapsto v_2', l_{y1} \mapsto v_2'], \\ \ S_2[l_p \mapsto in_{\mathbb{Z}} 0, l_{y0} \mapsto v_2, l_{y1} \mapsto v_2], \Delta^3 r^3) \in \nabla_S. \end{array} \tag{3}$$

By (1) and (2), all locations viewed by the local parameter $\tilde{r}^3$ are disjoint from $dom(\Delta^2)$ and from all local areas viewed by r^2. The stores have only been changed in locations viewed by $\tilde{r}^3$. Since values related in a parameter are also related in any extending parameter (weakening) every requirement from $\Delta^2 r^2$ still holds. Finally, since $S_2[l_p \mapsto in_{\mathbb{Z}} 0, l_{y0} \mapsto v_2, l_{y1} \mapsto v_2](l_p) = 0$ and the values stored in locations l_y and l_{y0} in the updated stores, namely v_1', v_1, v_2', v_2, are related in $\Delta^1 r^1$ and then by weakening also in $\Delta^3 r^3$, the first disjunct of $\tilde{r}^3$ is satisfied, and hence (3) holds.

It remains to show

A: $(\llbracket \emptyset; y \vdash recf_{1M} \rrbracket(y \mapsto l_y), \llbracket \emptyset; y \vdash recf_{1M} \rrbracket(y \mapsto l_y), \llbracket \emptyset; p, y_0, y_1 \vdash f_{1N} \rrbracket(p \mapsto l_p, y_0 \mapsto l_{y0}, y_1 \mapsto l_{y1}), \llbracket \emptyset; p, y_0, y_1 \vdash recf_{1N} \rrbracket(p \mapsto l_p, y_0 \mapsto l_{y0}, y_1 \mapsto l_{y1}), (\tau \to T\tau') \to T\mathrm{unit}, \Delta^3 r^3) \in \nabla_V$ and

B: $(\llbracket \emptyset; y \vdash recf_{2M} \rrbracket(y \mapsto l_y), \llbracket \emptyset; y \vdash recf_{2M} \rrbracket(y \mapsto l_y), \llbracket \emptyset; p, y_0, y_1 \vdash recf_{2N} \rrbracket(p \mapsto l_p, y_0 \mapsto l_{y0}, y_1 \mapsto l_{y1}) \llbracket \emptyset; p, y_0, y_1 \vdash recf_{2N} \rrbracket(p \mapsto l_p, y_0 \mapsto l_{y0}, y_1 \mapsto l_{y1}), (\tau \to T\tau') \to T\mathrm{unit}, \Delta^3 r^3) \in \nabla_V$

Now let $\Delta^4 r^4 \rhd \Delta^3 r^3,\ (w_1', w_1, w_2', w_2, \tau \to T\tau', \Delta^4 r^4) \in \nabla_V$, and
let $\Delta^5 r^5 \rhd \Delta^4 r^4, (K_1', K_1, K_2', K_2, (x : \tau \to T\tau')^\top \Delta^5 r^5) \in \nabla_K, (S_1', S_1, S_2', S_2, \Delta^5 r^5) \in \nabla_S,\ (c_1', c_1, c_2', c_2, (x : \mathrm{unit})^\top, \Delta^5 r^5) \in \nabla_K$

We have denotations $\llbracket \emptyset; y \vdash recf_{1M} \rrbracket(y \mapsto l_y) = in_{\multimap} d_{M1}$, $\llbracket \emptyset; p, y_0, y_1 \vdash recf_{1N} \rrbracket(p \mapsto l_p, y_0 \mapsto l_{y0}, y_1 \mapsto l_{y1}) = in_{\multimap} d_{N1}$, $\llbracket \emptyset; y \vdash recf_{2M} \rrbracket(y \mapsto l_y) = in_{\multimap} d_{M2}$, $\llbracket \emptyset; p, y_0, y_1 \vdash recf_{2N} \rrbracket(p \mapsto l_p, y_0 \mapsto l_{y0}, y_1 \mapsto l_{y1}) = in_{\multimap} d_{N2}$.

A: Now we want to show relatedness of the setters. As before if $w_1' = w_2' = \bot$ or $S_1' = S_2' = \bot$ we are done. Otherwise we reason as follows.

Observe that $(d_{M1} w_1) c_1 S_1 = c_1(S_1[l_y \mapsto w_1]) in_1 *$ and similarly $(d_{M1} w_1) c_1' S_1' = c_1'(S_1'[l_y \mapsto w_1']) in_1 *$. Also, $(d_{N1} w_2) c_2 S_2 = c_2(S_2[l_p \mapsto in_{\mathbb{Z}} 0, l_{y0} \mapsto w_2]) in_1 *$, if $S_2 l_P \neq 0$, and $(d_{N1} w_2) c_2 S_2 = c_2(S_2[l_p \mapsto in_{\mathbb{Z}} 1, l_{y1} \mapsto w_2]) in_1 *$, if $S_2 l_P = 0$. Similarly for the approximation $(d_{N1} w_2') c_2' S_2'$.

Since the states are related in $\Delta^5 r^5$ which is an extension of $\Delta^3 r^3$ we know that the content of $S_2 l_p$ is $in_{\mathbb{Z}} n$ for some n. We know that the continuations c_1', c_1, c_2', c_2 are related in $\Delta^5 r^5$. $(in_1 *, in_1 *, in_1 *, in_1 *, \text{unit}, \Delta^5 r^5)$ since they are related in any parameter. So if we can show that the updated states are related in $\Delta^5 r^5$ we are done.

The states S_1', S_1, S_2', S_2 are related in $\Delta^5 r^5$. All changes are only within the store areas belonging to $\tilde{r}^3$ and the changes preserve the invariant for $\tilde{r}^3$, hence the updated states are still related in $\Delta^5 r^5$. We conclude that the setters are related in $\nabla^3 r^3$.

B: Now we want to show relatedness of the getters. As before, if the denotations are applied to related unit type values where the approximations are $\bot$ or if $S_1' = S_2' = \bot$ we are done. Otherwise we reason as follows. Note that $(d_{M2} in_1 *) K_1 S_1 = K_1 S_1 (S_1 l_y)$ and similarly $(d_{M2} in_1 *) K_1' S_1' = K_1' S_1' (S_1' l_y)$. Since the states are not $\bot$ and are related in $\Delta^5 r^5$ which is an extension of $\Delta^3 r^3$ we know that the content of $S_2 l_p$ is $in_{\mathbb{Z}} n$ for some n. We have that $(d_{N2} in_1 *) K_2 S_2 = K_2 S_2 (S_2 l_{y0})$, if $n = 0$, and $(d_{N2} in_1 *) K_2 S_2 = K_2 S_2 (S_2 l_{y1})$, if $n \neq 0$. Similarly for the approximation $(d_{N2} in_1 *) K_2' S_2'$.

We know that the continuations K_1', K_1, K_2', K_2 and the states $S_1' S_1, S_2', S_2$ are related in $\Delta^5 r^5$. So if we can show that the retrieved values are related in $\Delta^5 r^5$ we are done.

Since the states $S_1' S_1, S_2', S_2$ are related in $\Delta^5 r^5$ they satisfy the invariant of $\tilde{r}^3$. So the content of $S_2 l_p$ is $in_{\mathbb{Z}} n$ for some n. If $n = 0$ then $S_1' l_y, S_1 l_y, S_2' l_{y0}, S_2 l_{y0}$ are related in $\Delta^5 r^r$, and if $n \neq 0$ then $S_1' l_y, S_1 l_y, S_2' l_{y1}, S_2 l_{y1}$ are related in $\Delta^5 r^r$, again by the requirement from $\tilde{r}^3$. This is what we need for the retrieved values to be related. We conclude that the getters are related in $\nabla^3 r^3$.

Then we can conclude that $([\![M]\!], [\![N]\!], \sigma, \Delta r) \in \nabla_V^{\emptyset}$, and as Δr was arbitrary that they are related in any parameter. Hence the programs M and N are contextually equivalent.

5.2 Example 2

Consider the computation terms M' and N' from the Introduction. They both have a free variable g of function type. We want to show that M' and N' are related in any parameter Δr.

We need to show $\forall \Delta^1 r^1 \rhd \Delta r. \forall (g_1', g_1, g_2', g_2, \sigma, \Delta^1 r^1) \in \nabla_V$.
$\forall \Delta^2 r^2 \rhd \Delta^1 r^1. \forall (k_1', k_1, k_2', k_2, (x : \sigma_1)^{\top}, \Delta^2 r^2 \in \nabla_K). \forall (S_1', S_1, S_2', S_2, \Delta^2 r^2) \in \nabla_S$.
$[\![\emptyset; g : \sigma \vdash M' : T\sigma_1]\!](g \mapsto g_1') k_1' S_1' = \top \Longrightarrow [\![\emptyset; g : \sigma \vdash N' : T\sigma_1]\!](g \mapsto g_2) k_2 S_2 = \top$ and
$[\![\emptyset; g : \sigma \vdash N' : T\sigma_1]\!](g \mapsto g_2') k_2' S_2' = \top \Longrightarrow [\![\emptyset; g : \sigma \vdash M' : T\sigma_1]\!](g \mapsto g_1) k_1 S_1 = \top$.
Here $\sigma = \sigma_1 \to T\text{unit}$, and $\sigma_1 = \text{unit} \to T\text{unit}$.

For the proof of this we define a local parameter $\tilde{r}^3 = (P^3, \emptyset)$ for $P^3 = (\{(S_a, S_b) | S_b l_x = in_{\mathbb{Z}} n > 0)\}, A_{\emptyset}, A_{\{l_x\}})$, where l_x is fresh for $dom(\Delta^2) \cup A_{r^2 2}(S_2)$. Then we have a parameter $\Delta^3 r^3$ where $\Delta^3 = \Delta^2$ and $r^3 = r^2 \cup \{\tilde{r}^3\}$ which we use in the proof.

6 Conclusion

We have presented a local relational proof method for establishing contextual equivalence of expressions in a language with recursive types and general references, building on earlier work of Benton and Leperchey [2]. The proof of existence of the logical relation is fairly intricate because of the interplay between

recursive types and local parameters for reasoning about higher-order store. However, the method is easy to use on examples: the only non-trivial steps are to guess the right local parameters — but since the local parameters express the intuitive reason for contextual equivalence, the non-trivial steps are really fairly straightforward. It is possible to extend our method to a language also with impredicative polymorphism; we will report on that on another occasion.

References

1. A. Ahmed. Step-indexed syntactic logical relations for recursive and quantified types. In P. Sestoft, editor, *Programming Languages and Systems. 15th European Symposium on Programming, ESOP 2006*, volume 3924 of *Lecture Notes in Computer Science*, pages 69–83. Springer, 2006.
2. N. Benton and B. Leperchey. Relational reasoning in a nominal semantics for storage. In *Proceedings of the Seventh International Conference on Typed Lambda Calculi and Applications (TLCA'05)*, volume 3461 of *Lecture Notes in Computer Science*. Springer, 2005.
3. L. Birkedal and R. Harper. Constructing interpretations of recursive types in an operational setting. *Information and Computation*, 155:3–63, 1999.
4. V. Koutavas and M. Wand. Bisimulations for untyped imperative objects. In P. Sestoft, editor, *Programming Languages and Systems, 15th European Symposium on Programming, ESOP 2005, Vienna, Austria.*, volume 3924 of *Lecture Notes in Computer Science*. Springer, 2006. to appear.
5. V. Koutavas and M. Wand. Small bisimulations for reasoning about higher-order imperative programs. In *POPL '06: Proceedings of the 33rd ACM SIGPLAN-SIGACT symposium on Principles of Programming Languages*, pages 141–152, New York, NY, USA, 2006. ACM Press.
6. I. A. Mason, S. Smith, and C. L. Talcott. From operational semantics to domain theory. *Information and Computation*, 128(1):26–47, 1996.
7. A. Pitts. Relational properties of domains. *Information and Computation*, 127:66–90, 1996.
8. A. Pitts and I. Stark. Observable properties of higher order functions that dynamically create local names, or: What's new? In *Mathematical Foundations of Computer Science, Proc. 18th Int. Symp., Gdańsk, 1993*, volume 711 of *Lecture Notes in Computer Science*, pages 122–141. Springer-Verlag, Berlin, 1993.
9. U. Reddy and H. Yang. Correctness of data representations involving heap data structures. *Science of Computer Programming*, 50(1-3):129–160, Mar. 2004.
10. J. C. Reynolds. Separation logic: A logic for shared mutable data structures. In *Proc. of the 17th Annual IEEE Symposium on Logic in Computer Science (LICS'02)*, pages 55–74, Copenhagen, Denmark, July 2002. IEEE Press.
11. M. Shinwell. *The Fresh Approach: Functional Programming with Names and Binders*. PhD thesis, Computer Laboratory, Cambridge University, Dec. 2004.
12. M. R. Shinwell and A. M. Pitts. On a monadic semantics for freshness. *Theoretical Computer Science*, 342:28–55, 2005.
13. E. Sumii and B. C. Pierce. A bisimulation for type abstraction and recursion. In *ACM SIGPLAN–SIGACT Symposium on Principles of Programming Languages (POPL), Long Beach, California*, 2005.