A Logical Relation for Monadic Encapsulation of State Proving contextual equivalences in the presence of runST

Haskell is often considered a *pure* functional programming language because effectful compuations are encapsulated using monads. To preserve purity, values usually cannot escape from those monads. One notable exception is the ST monad, introduced by Launchbury and Peyton Jones in 1994 (). The ST monad comes equipped with a function `runST` : $(\forall \rho, \mathsf{ST}\ \rho\ \tau) \to \tau$ that allows a value to escape from the monad: `runST` runs a stateful computation of the monadic type $\mathsf{ST}\ \rho\ \tau$ and then returns the resulting value of type τ. In the original paper (), the authors argued informally that the ST monad is "safe", in the sense that stateful computations are properly encapsulated and therefore the purity of the functional language is preserved.

In this paper we present a logical relations model of STLang, a higher-order functional programming language with impredicative polymorphism, recursive types, and a Haskell-style ST monad type with `runST`. In constrast to earlier work, we give an operational semantics of STLang with a global mutable heap, capturing how the language would be implemented in reality. We use our logical relations model to show for the first time that `runST` provides proper encapsulation of state. Concretely, we state a number of contextual approximations and equivalences that are expected to hold for pure computations and we then use our logical relations model to prove that they indeed hold for STLang, i.e., in the presence of stateful computations encapsulated using `runST`.

In STLang, values of any type can be stored in the heap, and thus it is an example of a language with so-called higher-order store. It is well-known that it is challenging to construct logical relations for languages with higher-order store. Here we define our logical relations model in Iris, a state-of-the-art higher-order separation logic (). Iris's base logic () comes equipped with certain modalities which we use to simplify the construction of the logical relations. Logical relations for other type systems have been defined in Iris before (). But to make our logical relations model powerful enough to prove the contextual equivalences for purity we use a new approach to defining logical relations in Iris, which involves several new technical innovations.

Additional Key Words and Phrases: ST Monad, Logical Relations, Functional Programming Languages, Theory of Programming Languages, Program Logics, Iris

ACM Reference format:
. 2017. A Logical Relation for Monadic Encapsulation of State. *PACM Progr. Lang.* 1, 1, Article 1 (January 2017), 20 pages.
DOI: 10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

The ST monad, as described in Launchbury and Peyton Jones (1994) and implemented in the standard Haskell library, is actually a family ST ρ of monads, where ρ ranges over types, which satisfy the following interface. The first two functions

```
return :: α → ST ρ α
(>>=)  :: ST ρ α → (α → ST ρ β) → ST ρ β
```

are the standard Kleisly arrow interface of monads in Haskell (>>= is called "bind"). The next three functions

```
newSTRef   :: α → ST ρ (STRef ρ α)
readSTRef  :: STRef ρ α → ST ρ α
writeSTRef :: STRef ρ α → α → ST ρ ()
```

2017. 2475-1421/2017/1-ART1 $15.00
DOI: 10.1145/nnnnnnn.nnnnnnn

```
fibST :: Integer → Integer
fibST n =
  let fibST' 0 x _ = readSTRef x
      fibST' n x y = do
          x' <- readSTRef x
          y' <- readSTRef y
          writeSTRef x y'
          writeSTRef y (x'+y')
          fibST' (n-1) x y
  in
  if n < 2 then n else
    runST $ do
      x <- newSTRef 0
      y <- newSTRef 1
      fibST' n x y
```

```

let fibST : ℕ -> ℕ =
  let rec fibST' n x y =
    if n = 0 then !x
    else bind !x in λ x' ->
           bind !y in λ y' ->
             bind x := y' in λ () ->
               bind y := (x' + y') in λ () ->
                  fibST' (n - 1) x y
  in
  if n < 2 then n else
    runST {
      bind ref 0 in λ x ->
        bind ref 1 in λ y ->
          fibST' n x y }
```

Fig. 1. Computing Fibonacci numbers using the ST monad in Haskell (left) and in STLang (right). Haskell code adapted from https://wiki.haskell.org/Monad/ST

are used to *create*, *read from* and *write into* references, respectively. Notice that the reference type STRef ρ τ, contains, as is usual, the type of the contents of the reference celss, τ, but also another type parameter, ρ, which, intuitively, indicates which (logical) region of the heap this reference belongs to. The interesting part of the interface is the interaction of this type parameter with following function

$$\texttt{runST} :: (\forall\ \rho.\ \texttt{ST}\ \rho\ \alpha) \rightarrow \alpha$$

The runST function runs effectful computations and extracts the result from the ST monad. Notice the impredicative quantification of the type variable of runST. (Recall that in Haskell, free type variables are implicitly universally quantified).

Finally, equality on references is decidable:

$$(==) :: \texttt{STRef}\ \rho\ \alpha \rightarrow \texttt{STRef}\ \rho\ \alpha \rightarrow \texttt{bool}$$

Notice that equality is an ordinary function, since it returns directly a boolean value, not a value of type ST ρ bool.

The semantics of these operations is intended to be the same as for ML-style references. In particular, it should be possible to use an operational semantics with a global heap with in-place update for the stateful operations. The ingenious idea of Launchbury and Peyton Jones () is that the parametric polymorphism in runST should still ensure that stateful computations are properly encapsulated and thus that ordinary functions remain pure.

To show that formally, we define a higher-order functional programming language, called STLang, with impredicative polymorphism, recursive types, and a Haskell-style ST monad type with runST. The operational semantics of STLang uses a global mutable heap for stateful operations.

Figure 1 shows how to compute the *n*-th term of the Fibonacci sequence in Haskell using the ST monad, and in STLang. Haskell programmers will notice that the STLang program on the right is essentially the same as the one on the left after the do-notation has been expanded. The inner function fibST' can be typed as follows:

$$\texttt{fibST'} :: \texttt{Integer} \rightarrow \texttt{ST}\ \rho\ \texttt{Integer} \rightarrow \texttt{ST}\ \rho\ \texttt{Integer} \rightarrow \texttt{ST}\ \rho\ \texttt{Integer}$$

Hence, the argument of runST has type $(\forall\ \rho.\ \texttt{ST}\ \rho\texttt{Integer})$ and thus fibST indeed has return type **Integer**.

1.1 The ST Monad, intuitively

The idea behind the ST monad is that its first parameter, that we have denoted by the type variable γ, denotes a region of the heap. In that mental model, the heap is made of disjoint regions, whose names are types, and the inhabitants of ST ρ τ are computations that only read, write and allocate in the region named ρ, and that produce a value of type τ.

Recall that, for a given type τ, `runST` has the following type:

```
runST :: (∀ γ. ST γ τ) → τ
```

In that model, this type tells us two things. First, and most importantly, a computation e can only be run in *any* region, in particular from a new empty region. And, since Haskell's polymorphism is parametric, it means that e does not depend on, nor touch any part of the heap that has not been allocated by e itself. Second, because γ cannot be free in τ, and because the types of references contain their region, references cannot escape from `runST`.

Launchbury and Peyton Jones (1994) state that the result of a computation e of type $\forall\gamma, \mathrm{ST}\ \gamma\ \tau$ (with $\gamma \notin \mathsf{FV}(\tau)$) in an initial heap h is the same as its result if run in the empty heap, in the presence of any environment, assuming that:

- we know e will only allocate in a set ϕ of locations,
- all locations in ϕ are *not* allocated in h,
- the environment does not write in ϕ.

As corollary, they conclude that the two properties stated above hold.

However, it is not clear that this statement means that Haskell remains *pure* when extended with the ST monad. More practically, we would like to give the programmer a more direct way to reason on their program, without reasoning on the semantics, in particular we would like to know that equational reasoning extends to Haskell with ST. Moreover, Launchbury and Peyton Jones (1994) state that "*A full proof lies well outside the scope of this paper*".

This is the goal of this work: characterize what it means for a language to be pure; and prove that adding the ST monad to a pure higher-order language preserve that purity.

1.2 What is purity?

Of course, the ST monad, as implemented in Haskell, is not "pure" in its strictest meaning, since it can mutate the heap. Moreover, one could implement the ST monad using something like the (strictly pure) State monad. This tells us that, on the one hand, we would like a definition of purity that is independent of the semantics, and on the other hand that how the ST monad is implemented is crucial.

Since our goal is to give evidence that Haskell's ST monad satisfies some properties, we have to give STLang a semantics close to that of the Haskell implementation. This language and its operational semantics is discussed more formally in §2, but, in short, there is a single untyped heap, and all the side effects of an effectful computation `e` are done when the `runST e` is reduced.

One of the main reasons in favor of Haskell's purity is that it makes reasoning about programs easier, because it allows for equational reasoning: by disallowing imperative features (or hiding them), some equations hold. For example, one can duplicate computations:

```
let x = e in (x, x)   =   (e, e),                                        (1)
```

of course, this equation would not hold in the presence of unrestricted side effects like ML: if `e` had been `y := !y + 1`, which increments the reference `y`, the reference would be incremented by 2 on the left hand-side of (1) and by 1 on the right.

$$
\begin{aligned}
\circledcirc &::= + \mid - \mid * \mid = \mid < \\
e &::= x \mid () \mid \mathtt{true} \mid \mathtt{false} \mid n \mid \ell \mid (e, e) \mid \mathtt{inj}_i\, e \mid \mathtt{rec}\, f(x) = e \mid \Lambda\, e \mid \mathtt{fold}\, e \mid \mathtt{unfold}\, e \mid e\ e \\
&\quad \mid e\ \_ \mid \pi_i\, e \mid \mathtt{match}\, e\, \mathtt{with}\, \mathtt{inj}_i\, x \Rightarrow e_i\, \mathtt{end} \mid \mathtt{if}\, e\, \mathtt{then}\, e\, \mathtt{else}\, e \mid e \circledcirc e \\
&\quad \mid \mathtt{ref}(e) \mid\ !\, e \mid e \leftarrow e \mid e == e \mid \mathtt{bind}\, e\, \mathtt{in}\, e \mid \mathtt{return}\, e \mid \mathtt{runST}\ \{e\} \\
v &::= () \mid \mathtt{true} \mid \mathtt{false} \mid n \mid \ell \mid (v, v) \mid \mathtt{inj}_i\, v \\
&\quad \mid \mathtt{rec}\, f(x) = e \mid \Lambda\, e \mid \mathtt{fold}\, v \mid \mathtt{ref}(v) \mid\ !\, v \mid v \leftarrow v \mid \mathtt{bind}\, v\, \mathtt{in}\, v \mid \mathtt{return}\, v \\
\tau &::= X \mid \rho \mid \mathbf{1} \mid \mathbb{B} \mid \mathbb{N} \mid \tau \times \tau \mid \tau + \tau \mid \tau \to \tau \mid \forall X.\, \tau \mid \mu X.\, \tau \mid \mathsf{ref}(\tau) \mid \mathsf{ST}\ \rho\ \tau
\end{aligned}
$$

Fig. 2. The syntax of STLang.

We claim that a set of such equations characterize purity. But first, it is unclear what the equal sign means in (1). In this work we have chosen *contextual equality*, intuitively, two programs are contextually equivalent, if no program can distinguish them. First, we (somewhat informally) define contextual refinement.

Definition 1.1 (Contextual refinements). Given two programs e and e' of type τ under some environment, ie. $\Xi \mid \Gamma \vdash e : \tau$ and $\Xi \mid \Gamma \vdash e' : \tau$, we say that e contextually refines e' in environment $\Xi|\Gamma$, which we write

$$\Xi \mid \Gamma \vDash e \preceq_{\mathsf{ctx}} e' : \tau$$

if, for all context C with a hole of type τ, and for all starting heaps h and h',

$$(C[e], h)\ \text{terminates without errors} \quad \Longrightarrow \quad (C[e'], h')\ \text{terminates without errors.}$$

We say that e and e' are contextually equivalent, which we write $\Xi \mid \Gamma \vDash e \approx_{\mathsf{ctx}} e' : \tau$ if both $\Xi \mid \Gamma \vDash e \preceq_{\mathsf{ctx}} e' : \tau$ and $\Xi \mid \Gamma \vDash e' \preceq_{\mathsf{ctx}} e : \tau$.

Notice that this notion of contextual refinement is stronger that usual, since the starting heaps of e and e' do not have to be related!

The refinements and equivalences that we will prove are the following:

$$
\begin{aligned}
\mathtt{let}\, x = e_2\, \mathtt{in}\, (e_1, x) &\approx_{\mathsf{ctx}} (e_1, e_2) && (2)\\
\mathtt{let}\, x = e'\, \mathtt{in}\, (x, x) &\approx_{\mathsf{ctx}} (e, e) && (3)\\
\mathtt{let}\, y = e_1\, \mathtt{in}\, \mathtt{rec}\, f(x) = e_2 &\preceq_{\mathsf{ctx}} \mathtt{rec}\, f(x) = \mathtt{let}\, y = e_1\, \mathtt{in}\, e_2 && (4)\\
\mathtt{let}\, y = e_1\, \mathtt{in}\, \Lambda\, e_2 &\preceq_{\mathsf{ctx}} \Lambda\, (\mathtt{let}\, y = e_1\, \mathtt{in}\, e_2) && (5)\\
e &\preceq_{\mathsf{ctx}} \mathtt{rec}\, f(x) = (e\ x) && (6)\\
e &\approx_{\mathsf{ctx}} \Lambda\, (e\ \_) && (7)\\
(\lambda\, x.\, e_1)\ e_2 &\preceq_{\mathsf{ctx}} e_1[e_2/x] && (8)\\
(\mathtt{rec}\, f(x) = e_1)\ e_2 &\preceq_{\mathsf{ctx}} e_1[e_2, (\mathtt{rec}\, f(x) = e_1)/x, f] && (9)
\end{aligned}
$$

moreover we prove the monad laws for the ST monad:

$$\ldots \qquad (10)$$

2 THE LANGUAGE: STLang

3 PROPERTIES THAT JUSTIFY PURITY

Lemma 3.1 (Commutativity). *If* $\Xi \mid \Gamma \vDash e_1 \preceq_{\mathsf{log}} e_1' : \tau$ *and* $\Xi \mid \Gamma \vDash e_2 \preceq_{\mathsf{log}} e_2' : \tau'$ *then*

$\boxed{\Xi \mid \Gamma \vdash e : \tau}$

TVar
$$\Xi \mid \Gamma, x : \tau \vdash x : \tau$$

TRec
$$\frac{\Xi \mid \Gamma, x : \tau_1, f : \tau_1 \to \tau_2 \vdash e : \tau_2}{\Xi \mid \Gamma \vdash \texttt{rec}\, f(x) = e : \tau_1 \to \tau_2}$$

TAbs
$$\frac{\Xi, X \mid \Gamma \vdash e : \tau}{\Xi \mid \Gamma \vdash \Lambda\, e : \forall X.\, \tau}$$

TFold
$$\frac{\Xi \mid \Gamma \vdash e : \tau[\mu X.\, \tau / X]}{\Xi \mid \Gamma \vdash \texttt{fold}\, e : \mu X.\, \tau}$$

TInst
$$\frac{\Xi \mid \Gamma \vdash e : \forall X.\, \tau \qquad \Xi \vdash \tau'}{\Xi \mid \Gamma \vdash e\ \_ : \tau[\tau'/X]}$$

TNew
$$\frac{\Xi \mid \Gamma \vdash e : \tau \qquad \Xi \vdash \rho}{\Xi \mid \Gamma \vdash \texttt{ref}(e) : \mathsf{ST}\ \rho\ (\mathsf{STRef}\ \rho\ \tau)}$$

TDeref
$$\frac{\Xi \mid \Gamma \vdash e : \mathsf{STRef}\ \rho\ \tau}{\Xi \mid \Gamma \vdash\ !e : \mathsf{ST}\ \rho\ \tau}$$

TGets
$$\frac{\Xi \mid \Gamma \vdash e : \mathsf{STRef}\ \rho\ \tau \qquad \Xi \mid \Gamma \vdash e' : \tau}{\Xi \mid \Gamma \vdash e \leftarrow e' : \mathsf{ST}\ \rho\ \mathbf{1}}$$

TRefEq
$$\frac{\Xi \mid \Gamma \vdash e : \mathsf{STRef}\ \rho\ \tau \qquad \Xi \mid \Gamma \vdash e' : \mathsf{STRef}\ \rho\ \tau}{\Xi \mid \Gamma \vdash e == e' : \mathbb{B}}$$

TBind
$$\frac{\Xi \mid \Gamma \vdash e : \mathsf{ST}\ \rho\ \tau \qquad \Xi \mid \Gamma \vdash e' : \tau \to (\mathsf{ST}\ \rho\ \tau')}{\Xi \mid \Gamma \vdash \texttt{bind}\, e\, \texttt{in}\, e' : \mathsf{ST}\ \rho\ \tau'}$$

TReturn
$$\frac{\Xi \mid \Gamma \vdash e : \tau \qquad \Xi \vdash \rho}{\Xi \mid \Gamma \vdash \texttt{return}\, e : \mathsf{ST}\ \rho\ \tau}$$

TRunST
$$\frac{\Xi, X \mid \Gamma \vdash e : \mathsf{ST}\ X\ \tau \qquad \Xi \vdash \tau}{\Xi \mid \Gamma \vdash \texttt{runST}\ \{e\} : \tau}$$

Fig. 3. An excerpt of the typing rules for STLang.

(a) $\Xi \mid \Gamma \vDash \texttt{let}\, x = e_2\, \texttt{in}\, (e_1, x) \preceq_{\log} (e_1', e_2') : \tau \times \tau'$
(b) $\Xi \mid \Gamma \vDash (e_1, e_2) \preceq_{\log} \texttt{let}\, x = e_2'\, \texttt{in}\, (e_1', x) : \tau \times \tau'$

Lemma 3.2 (Idempotency). *If* $\Xi \mid \Gamma \vDash e \preceq_{\log} e' : \tau$ *then*

(a) $\Xi \mid \Gamma \vDash (e, e) \preceq_{\log} \texttt{let}\, x = e'\, \texttt{in}\, (x, x) : \tau \times \tau'$
(b) $\Xi \mid \Gamma \vDash \texttt{let}\, x = e\, \texttt{in}\, (x, x) \preceq_{\log} (e', e') : \tau \times \tau'$

Lemma 3.3 (rec hoisting). *If* $\Xi \mid \Gamma \vDash e_1 \preceq_{\log} e_1' : \tau$, $\Xi \mid \Gamma, y : \tau, x : \tau', f : \tau' \to \tau'' \vDash e_2 \preceq_{\log} e_2' : \tau''$, $\Xi \mid \Gamma \vdash e_1' : \tau$ *and* $\Xi \mid \Gamma, y : \tau, x : \tau', f : \tau' \to \tau'' \vdash e_2' : \tau''$ *then*

$$\Xi \mid \Gamma \vDash \texttt{let}\, y = e_1\, \texttt{in}\, \texttt{rec}\, f(x) = e_2 \preceq_{\text{ctx}} \texttt{rec}\, f(x) = \texttt{let}\, y = e_1'\, \texttt{in}\, e_2' : \tau' \to \tau''$$

Lemma 3.4 (Λ hoisting). *If* $\Xi \mid \Gamma \vDash e_1 \preceq_{\log} e_1' : \tau$ *and* $\Xi, X \mid \Gamma, y : \tau \vDash e_2 \preceq_{\log} e_2' : \tau'$ *then*

$$\Xi \mid \Gamma \vDash \texttt{let}\, y = e_1\, \texttt{in}\, \Lambda\, e_2 \preceq_{\text{ctx}} \Lambda\, (\texttt{let}\, y = e_1'\, \texttt{in}\, e_2') : \forall X.\, \tau'$$

Lemma 3.5 (η expansion for rec). *If* $\Xi \mid \Gamma \vDash e \preceq_{\log} e' : \tau \to \tau'$ *and* $\Xi \mid \Gamma \vdash e' : \tau \to \tau'$ *then*

$$\Xi \mid \Gamma \vDash e \preceq_{\text{ctx}} \texttt{rec}\, f(x) = (e'\ x) : \tau \to \tau'$$

Lemma 3.6 (η expansion for Λ). *If* $\Xi \mid \Gamma \vDash e \preceq_{\log} e' : \forall X.\, \tau$ *then*

$$\Xi \mid \Gamma \vDash e \preceq_{\text{ctx}} \Lambda\, (e'\ \_) : \forall X.\, \tau$$

Lemma 3.7 (β reduction for λ). *If* $\Xi \mid \Gamma \vDash \lambda\, x.\, e_1 \preceq_{\log} \lambda\, x.\, e_1' : \tau \to \tau'$, $\Xi \mid \Gamma \vDash e_2 \preceq_{\log} e_2' : \tau$, $\Xi \mid \Gamma, x : \tau \vdash e_1' : \tau'$ *and* $\Xi \mid \Gamma \vdash e_2' : \tau$, *then*

$$\Xi \mid \Gamma \vDash (\lambda\, x.\, e_1)\ e_2 \preceq_{\text{ctx}} e_1'[e_2'/x] : \tau'$$

Lemma 3.8 (β reduction for Λ). *If* $\Xi \mid \Gamma \vDash \Lambda\, e \preceq_{\log} \Lambda\, e' : \forall X.\, \tau$ *then*

$$\Xi \mid \Gamma \vDash (\Lambda\, e)\ \_ \preceq_{\text{ctx}} e' : \tau'$$

Reduction: $\boxed{\langle h, e\rangle \rightarrow \langle h', e'\rangle}$ and head step: $\boxed{\langle h, e\rangle \rightarrow_h \langle h', e'\rangle}$
Evaluation contexts:

$$\begin{aligned} K ::= & [] \mid (K, e) \mid (v, K) \mid \mathtt{inj}_i\, K \mid \mathtt{fold}\, K \mid \mathtt{unfold}\, K \mid K\ e \mid v\ K \mid K\ \_ \\ & \mid \pi_i\, K \mid \mathtt{match}\, K\, \mathtt{with}\, \mathtt{inj}_i\, x \Rightarrow e_i\, \mathtt{end} \mid \mathtt{if}\, K\, \mathtt{then}\, e\, \mathtt{else}\, e \\ & \mid K \circledcirc e \mid v \circledcirc K \mid \mathtt{ref}(K) \mid\, !\, K \mid K \leftarrow e \mid v \leftarrow K \\ & \mid K == e \mid v == K \mid \mathtt{bind}\, K\, \mathtt{in}\, e \mid \mathtt{bind}\, v\, \mathtt{in}\, K \mid \mathtt{return}\, K \mid \mathtt{runST}\ \{K\} \end{aligned}$$

$$\frac{\langle h, e\rangle \rightarrow_h \langle h', e'\rangle}{\langle h, K[e]\rangle \rightarrow \langle h', K[e']\rangle} \qquad \langle h, \mathtt{unfold}\,(\mathtt{fold}\, v)\rangle \rightarrow_h \langle h, v\rangle$$

$$\langle h, (\mathtt{rec}\, f(x) = e)\ v\rangle \rightarrow_h \langle h, e[v, \mathtt{rec}\, f(x) = e/x, f]\rangle \qquad \langle h, (\Lambda\, e)\ \_\rangle \rightarrow_h \langle h, e\rangle$$

$$\langle h, \mathtt{match}\, \mathtt{inj}_i\, v\, \mathtt{with}\, \mathtt{inj}_i\, x \Rightarrow e_i\, \mathtt{end}\rangle \rightarrow_h \langle h, e_i[v/x]\rangle \qquad \frac{\ell = \ell'}{\langle h, \ell == \ell'\rangle \rightarrow_h \langle h, \mathtt{true}\rangle}$$

$$\frac{\ell \neq \ell'}{\langle h, \ell == \ell'\rangle \rightarrow_h \langle h, \mathtt{false}\rangle} \qquad \frac{\langle h, v\rangle \rightsquigarrow \langle h', e\rangle}{\langle h, \mathtt{runST}\ \{v\}\rangle \rightarrow_h \langle h', \mathtt{runST}\ \{e\}\rangle} \qquad \langle h, \mathtt{runST}\ \{\mathtt{return}\, v\}\rangle \rightarrow_h \langle h, v\rangle$$

Effectful reduction: $\boxed{\langle h, v\rangle \rightsquigarrow \langle h', e\rangle}$ and effectful head step: $\boxed{\langle h, v\rangle \rightsquigarrow_h \langle h', e\rangle}$ Effectful evaluation contexts: $\mathbb{K} ::= [] \mid \mathtt{bind}\, \mathbb{K}\, \mathtt{in}\, v$

$$\frac{\langle h, v\rangle \rightsquigarrow_h \langle h', e\rangle}{\langle h, \mathbb{K}[v]\rangle \rightsquigarrow \langle h', \mathbb{K}[e]\rangle} \qquad \langle h, \mathtt{bind}\,(\mathtt{return}\, v)\, \mathtt{in}\, v'\rangle \rightsquigarrow_h \langle h, v'\ v\rangle$$

Alloc

$$\frac{\ell \notin \mathrm{dom}(h)}{\langle h, \mathtt{ref}(v)\rangle \rightsquigarrow_h \langle h \uplus \{\ell \mapsto v\}, \mathtt{return}\, \ell\rangle} \qquad \langle h \uplus \{\ell \mapsto v\}, !\,\ell\rangle \rightsquigarrow_h \langle h \uplus \{\ell \mapsto v\}, \mathtt{return}\, v\rangle$$

$$\langle h \uplus \{\ell \mapsto v'\}, \ell \leftarrow v\rangle \rightsquigarrow_h \langle h \uplus \{\ell \mapsto v\}, \mathtt{return}\, ()\rangle$$

If $\rightharpoonup$ is a relation, we note $\rightharpoonup^n$ its iterated self-composition and $\rightharpoonup^*$ its reflexive and transitive closure.

Fig. 4. An excerpt of the dynamics of STLang, a CBV small-step operational semantics.

Lemma 3.9 (β reduction for rec). *If* $\Xi \mid \Gamma \vDash \mathtt{rec}\, f(x) = e_1 \preceq_{\log} \mathtt{rec}\, f(x) = e_1' : \tau \rightarrow \tau'$, $\Xi \mid \Gamma \vDash e_2 \preceq_{\log} e_2' : \tau$, $\Xi \mid \Gamma, x : \tau, f : \tau \rightarrow \tau' \vdash e_1' : \tau'$ *and* $\Xi \mid \Gamma \vdash e_2' : \tau$, *then*

$$\Xi \mid \Gamma \vDash (\mathtt{rec}\, f(x) = e_1)\ e_2 \preceq_{\mathrm{ctx}} e_1'[e_2', (\mathtt{rec}\, f(x) = e_1')/x, f] : \tau'$$

4 IRIS AND EXTENDING IT

In this work, we use a very restricted subset of Iris (Krebbers et al. 2017a). Understanding Iris in particular and higher-order separation logic in general is not in any way crucial for understanding the logical relation presented in this paper. A higher-level understanding of these topics as is explained in this section should suffice.

$$\begin{array}{rll}
\kappa ::= & \mathbf{1} \mid \kappa \longrightarrow \kappa \mid \mathit{Expr} \mid \mathit{Val} \mid \mathbb{N} \mid \mathbb{B} \mid \kappa \xrightarrow{\text{fin}} \kappa \mid \mathsf{finset}(\kappa) \mid & \text{Iris types} \\
& \mathit{Names} \mid \mathit{Monoid} \mid \mathit{iProp} \mid \kappa \times \kappa \mid \ \ldots & \\
\gamma : & \mathit{Names} & \text{Instance names for ownership} \\
a, b, g, h : & \kappa & \\
M ::= & \bullet\, a \mid \circ\, a \mid \mathsf{ag}(a) \mid \mathsf{ex}(a) \mid \bot_M & \text{Monoids} \\
\Phi : & \kappa \longrightarrow \mathit{iProp} & \text{Predicates} \\
P ::= & \top \mid \bot \mid P * P \mid P \mathrel{-\!\!*} P \mid P \wedge P \mid P \Rightarrow P \mid P \vee P \mid \forall x : \kappa.\, \Phi \mid & \text{Iris propositions (Prop)} \\
& \exists x : \kappa.\, \Phi \mid \checkmark(a) \mid \triangleright P \mid \ \Box P \mid \mathrel{|}\!\!\Rrightarrow \mid \boxed{M}^{\gamma} \mid \gamma \Mapsto \Phi &
\end{array}$$

Fig. 5. The fragment of Iris we use.

4.1 Iris in a nutshell

The syntax of the fragment of Iris that is used in this paper is depicted in Figure 5. This subset includes the connectives of higher-order separation logic ($\top$, $\bot$, $\wedge$, $\vee$, $\Rightarrow$, $*$, $\mathrel{-\!\!*}$, $\forall$ and $\exists$). In addition to those, Iris also includes the "later" modality ($\triangleright$). This modality is used as guard for taking *guarded* fixpoints. In Iris it is used to take the fixpoint for the definition of weakest preconditions or to guard impredicative invariants (to avoid self-referential paradoxes). Here we do not use either of these constructs. However we use the later modality to take the guarded fixpoint corresponding to the interpretation of recursive types. For any proposition P, $P \vdash \triangleright P$. In other words, if P holds, then it so does *guarded* P. The later modality also commutes with all of the connectives of higher-order separation logic including quantifiers.

The other modality of the Iris logic that we use is the "always" modality ($\Box$). Intuitively, $\Box P$ holds whenever P holds and is a duplicable assertion. In particular we have $(\Box P) * (\Box P) \dashv\vdash \Box P$ where $\dashv\vdash$ is the logical equivalence of formulas. We use the term *persistence* for duplicable proposition. In particular, we require that relatedness of programs be duplicable as programs can be used multiple times, i.e., the type system is not linear or affine. Notice that the always modality is idempotent, $\Box P \vdash \Box\Box P$, and also we have $\Box P \vdash P$. The latter simply says that if P holds and is duplicable, it simply holds. The always modality also commutes with all of the connectives of higher-order separation logic including quantifiers.

Perhaps the most important modality in Iris is the "update" modality[1] ($\mathrel{|}\!\!\Rrightarrow$). Intuitively, the proposition $\mathrel{|}\!\!\Rrightarrow P$ holds if the resources (as in higher-order separation logic resources, for instance ownership of a heap chunk) can be updated (allocated, deallocated, or changed) such that proposition P holds. We write $P \Rrightarrow Q$ as a shorthand for $P \mathrel{-\!\!*} \mathrel{|}\!\!\Rrightarrow Q$. Notice that the update modality is idempotent.

$$\mathrel{|}\!\!\Rrightarrow(\mathrel{|}\!\!\Rrightarrow P) \dashv\vdash \mathrel{|}\!\!\Rrightarrow P$$

In other words, if we can update resources such that after updating resources P holds, then we can update resources such that P holds.

For all of the modalities of Iris we have the following properties:

$$\begin{array}{c}\textsc{Mod-intro}\\ P \vdash \bowtie P\end{array} \qquad\qquad \begin{array}{c}\textsc{Mod-mono}\\ \dfrac{P \vdash Q}{\bowtie P \vdash \bowtie Q}\end{array}$$

for any modality modality $\bowtie$.

[1]In the (Krebbers et al. 2017a) paper this modality is called the *basic* update modality.

$$\textsc{Ghost-Alloc}\quad \frac{\checkmark a}{\Rrightarrow \exists \gamma.\, \boxed{a}^{\gamma}} \qquad \textsc{Validity}\quad \frac{a \neq \bot_M}{\checkmark a} \qquad \textsc{Own-Valid}\quad \boxed{a}^{\gamma} \vdash \checkmark(a) \qquad \textsc{Sharing}\quad \boxed{a}^{\gamma} * \boxed{b}^{\gamma} \dashv\vdash \boxed{a \cdot b}^{\gamma} \qquad \textsc{Agree}\quad \mathrm{ag}(a) \cdot \mathrm{ag}(a) = \mathrm{ag}(a)$$

$$\textsc{Disagree}\quad \frac{a \neq b}{\mathrm{ag}(a) \cdot \mathrm{ag}(b) = \bot_M} \qquad \textsc{Exclusive}\quad \mathrm{ex}(a) \cdot b = \bot_M \qquad \textsc{Auth-Included}\quad \bullet\, a \cdot \circ\, b \vdash a \subseteq b \qquad \textsc{Frag-distributes}\quad \circ\, a \cdot \circ\, b = \circ\,(a \cdot b) \qquad \textsc{Full-Exclusive}\quad \bullet\, a \cdot \bullet\, b = \bot_M$$

$$\textsc{Auth-Alloc-Finset}\quad \frac{h \cap a = \emptyset}{\boxed{\bullet\, h}^{\gamma} \Rrightarrow \boxed{\bullet\,(h \uplus a) \cdot \circ\, a}^{\gamma}} \qquad \textsc{Auth-Alloc-Fpfn}\quad \frac{\mathrm{dom}(h) \cap \mathrm{dom}(a) = \emptyset}{\boxed{\bullet\, h}^{\gamma} \Rrightarrow \boxed{\bullet\,(h \uplus a) \cdot \circ\, a}^{\gamma}}$$

$$\textsc{Fpfn-operation-success}\quad \frac{\forall x \in \mathrm{dom}(a) \cap \mathrm{dom}(b).\ \checkmark(a(x) \cdot b(x))}{a \cdot b = \begin{cases} a(x) & \text{if } x \in \mathrm{dom}(a) \wedge x \notin \mathrm{dom}(b) \\ a(x) \cdot b(x) & \text{if } x \in \mathrm{dom}(a) \cap \mathrm{dom}(b) \\ b(x) & \text{if } x \in \mathrm{dom}(b) \wedge x \notin \mathrm{dom}(a) \end{cases}}$$

$$\textsc{Fpfn-operation-fail}\quad \frac{a(x) \cdot b(x) = \bot_M \qquad x \in \mathrm{dom}(a) \cap \mathrm{dom}(b)}{a \cdot b = \bot_M}$$

$$\textsc{Auth-update-Fpfn}\quad \boxed{\bullet\,(h \uplus (\ell \mapsto \mathrm{ex}(v_1))) \cdot \circ\, \ell \mapsto \mathrm{ex}(v_1)}^{\gamma} \Rrightarrow \boxed{\bullet\,(h \uplus (\ell \mapsto \mathrm{ex}(v_2))) \cdot \circ\, \ell \mapsto \mathrm{ex}(v_2)}^{\gamma}$$

Fig. 6. Rules for ghost resources in Iris

Ownership in Iris. In Iris, resources are elements of partial monoids, seen as a subset of a (full) monoid $\mathcal{M}$ characterized by the *validity* predicate $\checkmark : \mathcal{M} \longrightarrow iProp$. For example the heap can be seen as the monoid of finite partial maps $Loc \rightharpoonup_{\mathrm{fin}} Val$, with disjoint union for product. The partiality comes from the fact that disjoint union is partial. We write $\boxed{M}^{\gamma}$ to asserts that the monoid instance named γ has contents M. This is the way Iris represents ghost resources. All the relevant rules regarding ghost ownership are depicted in Figure 6. This figure shows the rule Alloc used for allocating new instances of monoids (resources). It intuitively says that we can always allocate any monoid element a so long as it is *valid* ($\checkmark a$). It also depicts the rules necessary for allocating and updating finite set $\mathrm{finset}(A)$ and finite partial function $A \rightharpoonup_{\mathrm{fin}} M$ monoids. These are the only monoids that we use. In these monoids, the monoid operation, $\cdot$, is *disjoint* union. The notation $a \mapsto b : A \rightharpoonup_{\mathrm{fin}} B \triangleq \{(a, b)\}$ is a singleton finite partial function.

Some useful monoids. The constructs $\bullet$ and $\circ$ are constructors of the so-called authoritative monoid $\textsc{Auth}(M)$. We read $\bullet\, a$ as *full a* and $\circ\, a$ as *fragment a*. The intuitive idea is a division of sorts. We want to distribute the ownership information but keep track of everything in a central "location". This central location is the full part of the resource (see rule Auth-Included). The fragment part can be shared (rule **??**) while the full part (the central location) should always remain unique (rule Full-Exclusive).

Apart from the authoritative monoids, we also use the agreement monoid $\textsc{Ag}(M)$ and exclusive monoid $\textsc{Ex}(M)$. As the name suggests, the operation of the agreement monoid guarantee that $\mathrm{ag}(a) \cdot \mathrm{ag}(b)$ is invalid whenever $a \neq b$ (and otherwise it is idempotent; see rules Agree and Disagree). From the rule Agree it follows that the ownership of elements of $\textsc{Ag}(M)$ is persistent.

$$\boxed{\mathrm{ag}(a)}^{\gamma} \dashv\vdash \boxed{\mathrm{ag}(a) \cdot \mathrm{ag}(a)}^{\gamma} \dashv\vdash \boxed{\mathrm{ag}(a)}^{\gamma} * \boxed{\mathrm{ag}(a)}^{\gamma}$$

The operation of exclusive monoid never results in a valid element enforcing that there can only be one instance of it owned.

Notice that the monoids can be nested. The update rule for finite partial functions (Auth-update-Fpfn) in fact is defined for a monoid of the form $A \rightharpoonup_{\text{fin}} \text{Ex}(M)$. The reason for this is that we can update both the full part and the fragment part and know that there is no other part of the fragment is not updated. As another interesting fact about ownership in Iris notice that a rule similar to Auth-update-Fpfn for a monoid of the form $A \rightharpoonup_{\text{fin}} \text{Ag}(M)$ is unsound, i.e., we can prove $\bot$.

Saved predicates. The last Iris proposition that we use is called saved propositions $\gamma \Mapsto \Phi$. This is simply a mechanism for assigning a name γ to a predicate Φ. There are only three rules governing the use of this kind of Iris proposition. We can allocate them (rule SavedPred-Alloc), they are persistent (rule SavedPred-Persistent) and that they are just a "dictionary", if you will, mapping names to predicates (rule SavedPred-Equiv).

SavedPred-Alloc
$$\mid\!\Rrightarrow \exists \gamma.\ \gamma \Mapsto \Phi$$

SavedPred-Persistent
$$\gamma \Mapsto \Phi \dashv\vdash \gamma \Mapsto \Phi * \gamma \Mapsto \Phi$$

SavedPred-Equiv
$$\frac{\gamma \Mapsto \Phi * \gamma \Mapsto \Psi}{\triangleright \Phi(a) \vdash \triangleright \Psi(a)}$$

As usual, the later modality is added as a guard to avoid self referential paradoxes. After all, we are storing a predicate (something of type $A \rightarrow iProp$) inside a proposition (something of type $iProp$).

4.2 Future modality, if-convergent and plainness

Defining logical relations for programming languages (Ahmed 2004) featuring higher-order mutable state requires arguments step-indexes and Kripke worlds (also known as possible worlds). By defining the logical relation in Iris we do not have to worry about constructing or directly dealing with either step-indexes or possible worlds. These are all taken care of by Iris and buried under layers of abstraction as the underlying model of Iris itself involves step-indexes and Kripke worlds.

Logical relations based on Kripke worlds are parameterized by a world (each world has a step index). Roughly speaking two expressions are related in a world W if whenever the left expression reduces to a value in a number steps, n, less than the W's step-index then the right hand side also reduces to a value and there is a world W' with $W \sqsubseteq W'$ with n step-indexes less such that the values are related in W'. The relation $W \sqsubseteq W'$ is read as W' is a future world of W.

In this section we describe a modality, $\mid\!\equiv\!\{\}\!\!\Rrightarrow\!\triangleright$, named the future modality, defined in Iris. The intuitive reading of $\mid\!\equiv\!\{n\}\!\!\Rrightarrow\!\triangleright P$ is that there is a "world" (although we do not have the concept of a world in the logic of Iris) where P holds. In other works defining logical relations in Iris (Krebbers et al. 2017b; Krogh-Jespersen et al. 2017a) the authors use the weakest precondition construction of Iris to express logical relatedness. See §4.4 for more details on this. There we explain in more details how and why the future modality resembles future Kripke worlds. We also give arguments why using Iris' weakest precondition mechanism is not suitable for encoding of the logical relation for STLang that is strong enough to establish the results that we do in this paper. Notice however that understanding the relation between the future modality and future worlds and other arguments presented in §4.4 is in no way crucial for understanding and/or appreciating the logical relation that we define or the theorems that are proven using this logical relation.

Future modality. We define the future modality $\mid\!\equiv\!\{\}\!\!\Rrightarrow\!\triangleright$ as follows:

$$\mid\!\equiv\!\{n\}\!\!\Rrightarrow\!\triangleright P \triangleq (\mid\!\Rrightarrow \triangleright)^n \mid\!\Rrightarrow P$$

where $(\mid\!\Rrightarrow \triangleright)^n$ is n times repetition of $\mid\!\Rrightarrow \triangleright$. Similar to the update modality, we write $P \equiv\!\{n\}\!\!\Rrightarrow\!\triangleright Q$ as a shorthand for $P -\!\!* \mid\!\equiv\!\{n\}\!\!\Rrightarrow\!\triangleright Q$.

Lemma 4.1 (Properties of the future modality). *The future modality, $\mid\!\equiv\!\{n\}\!\!\Rrightarrow\!\triangleright$, has the following properties:*

(1) $|\!\!\!\equiv\!\{n\}\!\!\equiv\!\!\triangleright Q \dashv\vdash \mathrel{|\!\!\!\Rrightarrow}(|\!\!\!\equiv\!\{n\}\!\!\equiv\!\!\triangleright Q)$
(2) $|\!\!\!\equiv\!\{n\}\!\!\equiv\!\!\triangleright Q \dashv\vdash |\!\!\!\equiv\!\{n\}\!\!\equiv\!\!\triangleright(\mathrel{|\!\!\!\Rrightarrow} Q)$
(3) $(|\!\!\!\equiv\!\{n\}\!\!\equiv\!\!\triangleright Q) * (|\!\!\!\equiv\!\{n\}\!\!\equiv\!\!\triangleright Q') \vdash |\!\!\!\equiv\!\{n\}\!\!\equiv\!\!\triangleright(Q * Q')$
(4) $\triangleright^{n} P \vdash |\!\!\!\equiv\!\{n\}\!\!\equiv\!\!\triangleright P$
(5) *If* $(|\!\!\!\equiv\!\{m-n\}\!\!\equiv\!\!\triangleright P) * Q \vdash Q'$ *then* $(|\!\!\!\equiv\!\{m\}\!\!\equiv\!\!\triangleright P) * (|\!\!\!\equiv\!\{n\}\!\!\equiv\!\!\triangleright Q) \vdash (|\!\!\!\equiv\!\{n\}\!\!\equiv\!\!\triangleright Q')$

If Convergent (IC). The purpose of the future modality is to play the role of referring to future worlds in Iris despite the fact that in the logic of Iris there is no *direct* way to refer to or talk about the underlying worlds or step-indexes. Recall that we want say that two expressions *expr* and e' are related if whenever e reduces in n number of steps then e' reduces to a value and then the values are related at a future world n steps away. Here we define the *If Convergent (IC)* construct in Iris. We define IC as follows:

$$\mathsf{IC}^{\gamma}\ e\ \{\!|v.\ Q|\!\} \triangleq \forall h_1, h_2, v, n.\ \langle h_1, e\rangle \to^{n} \langle h_2, v\rangle * \boxed{\bullet\,\mathsf{excl}(h_1)}^{\gamma} \equiv\!\{n\}\!\!\equiv\!\!\triangleright \boxed{\bullet\,\mathsf{excl}(h_2)}^{\gamma} * Q$$

where v may appear in Q.[2] The function excl is mapping from heap to the monoid $(Loc \xrightarrow{\text{fin}} \mathrm{Ex}(Val))$ representing the heap:

$$\mathsf{excl}(h) = \{(\ell, \mathrm{ex}(v)) \mid (\ell, v) \in h\}$$

This definition basically spells out what we said earlier about IC. Akin to Hoare triples being defined using the weakest precondition, we define IC triples as follows:

$$\{\!|P|\!\}\ e\ \{\!|v.\ Q|\!\}_{\gamma} \triangleq \Box(P \twoheadrightarrow \mathsf{IC}^{\gamma}\ e\ \{\!|v.\ Q|\!\})$$

The name γ here is used to keep track of the contents of the heap. This allows us to abstract the heap away in our arguments.[3] Perhaps the best way to see how IC in proofs is to show the following lemma about the properties of IC.

Lemma 4.2 (Properties of IC). *The IC predicate satisfies the following properties:*[4]

(1) $IC^{\gamma}\ e\ \{\!|v.\ Q|\!\} * (\forall w.\ (Q\ w) \twoheadrightarrow IC^{\gamma}\ K[w]\ \{\!|v.\ Q'\ v|\!\}) \vdash IC^{\gamma}\ K[e]\ \{\!|v.\ Q'|\!\}$
(2) $\mathrel{|\!\!\!\Rrightarrow}(Q\ w) \vdash IC^{\gamma}\ w\ \{\!|v.\ Q|\!\}$
(3) $(\forall v.\ (P\ v) \Rrightarrow (Q\ v)) * IC^{\gamma}\ e\ \{\!|v.\ P|\!\} \vdash IC^{\gamma}\ e\ \{\!|v.\ Q|\!\}$
(4) $\mathrel{|\!\!\!\Rrightarrow} IC^{\gamma}\ e\ \{\!|v.\ Q|\!\} \vdash IC^{\gamma}\ e\ \{\!|v.\ Q|\!\}$
(5) $IC^{\gamma}\ e\ \{\!|v.\ \mathrel{|\!\!\!\Rrightarrow} Q|\!\} \vdash IC^{\gamma}\ e\ \{\!|v.\ Q|\!\}$
(6) $(\forall h.\ \langle h, e\rangle \to \langle h, e'\rangle) * \triangleright IC^{\gamma}\ e'\ \{\!|v.\ Q\ v|\!\} \vdash IC^{\gamma}\ e\ \{\!|v.\ Q|\!\}$
(7) $\triangleright(\forall \ell.\ \boxed{\circ\,\ell \mapsto ex(v)}^{\gamma} \Rrightarrow Q\ \ell) \vdash IC^{\gamma}\ \mathtt{runST}\ \{\mathtt{ref}(v)\}\ \{\!|w.\ Q|\!\}$
(8) $\triangleright \boxed{\circ\,\ell \mapsto ex(v)}^{\gamma} * \triangleright(\boxed{\circ\,\ell \mapsto ex(v)}^{\gamma} \Rrightarrow Q\ v) \vdash IC^{\gamma}\ \mathtt{runST}\ \{!\,\ell\}\ \{\!|w.\ Q|\!\}$
(9) $\triangleright \boxed{\circ\,\ell \mapsto ex(v')}^{\gamma} * \triangleright(\boxed{\circ\,\ell \mapsto ex(v)}^{\gamma} \Rrightarrow Q\ ()) \vdash IC^{\gamma}\ \mathtt{runST}\ \{\ell \leftarrow v\}\ \{\!|w.\ Q|\!\}$
(10) $IC^{\gamma}\ \mathtt{runST}\ \{e\}\ \{\!|v.\ Q|\!\} * \Big(\forall w.\ (Q\ w) \twoheadrightarrow$
$IC^{\gamma}\ \mathtt{runST}\ \{\mathbb{K}[\mathtt{return}\, w]\}\ \{\!|v.\ Q'\ w|\!\}\Big) \vdash IC^{\gamma}\ \mathtt{runST}\ \{\mathbb{K}[e]\}\ \{\!|v.\ Q'|\!\}$

The cases (1) and (2) above show that IC is a monad in the same way that weakest precondition (aka Dijkstra monad) is a monad. The properties stated in this Lemma 4.2 are the only properties that we need to know in order to prove the compatibility lemmas. It is only for parts of the lemmas justifying purity that we have to unfold the definition IC. Even then, it is only the properties of the future modality in Lemma 4.1 and Lemma 4.7 are used and the future modality is never unfolded.

[2]In general the number steps, n, can also appear in Q but here we only present this slightly simpler version.
[3]This is very similar to the way the definition of weakest preconditions in Iris hides the state.
[4]Notice that in the following lemma the statements are not always the strongest provable (e.g., the case 7 could have 2 $\triangleright$ modalities). They are however enough for our purposes.

Notice that IC is parameterized by the name γ of the monoid representing the heap. This is not an arbitrary choice. Intuitively, this allows us to talk about hypothetical executions of the program by simply allocating a fresh instance the heap. For this reason, quantifying over this instance name can have interesting (at least at the first sight) consequences.

Lemma 4.3 (Heap is untouched). *If we have $\forall \gamma.\ IC^{\gamma}\ e\ \{\!|\top|\!\}$ then we can prove:*

$$\forall h_1, h_2, v, n.\ \ \langle h_1, e\rangle \to^{n} \langle h_2, v\rangle \equiv\!\{n\}\!\!\Rrightarrow\!\triangleright h_1 \subseteq h_2$$

Proof. Let us assume we have h_1, h_2, v and n such that $\langle h_1, e\rangle \to^{n} \langle h_2, v\rangle$. We have to show $\models\!\{n\}\!\!\Rrightarrow\!\triangleright h_1 \subseteq h_2$. Since the goal allows us to update we can use the rule Ghost-Alloc to allocate $\boxed{\bullet\, \mathsf{excl}(h_1) \cdot \circ\, \mathsf{excl}(h_1)}^{\gamma_h}$ which gives us $\boxed{\bullet\, \mathsf{excl}(h_1)}^{\gamma_h} * \boxed{\circ\, \mathsf{excl}(h_1)}^{\gamma_h}$. Now we can use $\boxed{\bullet\, \mathsf{excl}(h_1)}^{\gamma_h}$ in $\forall \gamma.\ \mathsf{IC}^{\gamma}\ e\ \{\!|\top|\!\}$ and use monotonicity of IC we get $\boxed{\bullet\, \mathsf{excl}(h_2)}^{\gamma_h}$. We now have to show $h_1 \subseteq h_2$ but we have $\boxed{\bullet\, \mathsf{excl}(h_2)}^{\gamma_h} * \boxed{\circ\, \mathsf{excl}(h_1)}^{\gamma_h}$ and from it, using the rules of ownership in Figure 6 we can prove $h_1 \subseteq h_2$. □

Plainness. We define the modality *naught* as follows

$$(\rtimes P)\ n\ x \triangleq P\ n\ \varepsilon$$

where ε is the empty resource. This definition says that $\rtimes P$ holds for n steps for resource x if P holds n steps for the empty resource. In prose, P does not depend on any resources. This definition is very similar to the definition of the always modality ($\Box$).

Lemma 4.4 (Basic properties of the naught modality). *The naught modality satisfies the following properties.*

(1) $\rtimes P \vdash P$
(2) $\rtimes P \vdash \rtimes \rtimes P$
(3) $\rtimes P \vdash \Box P$
(4) $\Box \rtimes P \dashv\vdash \rtimes \Box P$ *– similarly for* $\triangleright$
(5) $(\forall x, \rtimes \Phi) \dashv\vdash \rtimes(\forall x, \Phi)$ *– similarly for* $\exists$
(6) $\rtimes(P \wedge Q) \vdash (\rtimes P) \wedge (\rtimes Q)$ *– similarly for* $\vee$ *and* $*$

Plainness is a concept defined based on the naught modality in similar fashion as the concept of persistent is defined based on the always modality.

$$\mathsf{plain}(P) \triangleq P \vdash \rtimes P$$

Plain properties are those that are independent of any resources. A prime example here is reducibility:

$$\mathsf{plain}(\langle h_1, e_1\rangle \to^{*} \langle h_2, e_2\rangle)$$

Plainness commutes with all connectives of the logic and most modalities.

Lemma 4.5 (Properties of plainness). *Plainness satisfies the following properties.*

(1) plain$(\rtimes P)$
(2) If plain(P) *and plain*(Q) *then plain*$(P \wedge Q)$ *– similarly for* $\vee$ *and* $*$
(3) If for all $(x : A)$ *plain*$(\Phi\ x)$ *then plain*$(\forall x : A.\ \Phi)$ *– similarly for* $\exists$
(4) If plain(P) *then plain*$(\triangleright P)$ *– similarly for* $\Box$
(5) If plain(P) *then persistent*(P)
(6) If plain(Q) *and* $P \vdash Q$ *then* $P \vdash Q * P$

The most interesting aspect of plain propositions is how they interact with the update and future modalities.

Lemma 4.6 (Update modality and plainness). *If P is plain and $Q \vdash P$ then*

$$\models\!\!\Rrightarrow Q \vdash (\models\!\!\Rrightarrow Q) * P$$

This lemma basically says that if we can update resources to have Q which implies a plain proposition P then we can have P without actually updating resources. A very informal proof is the following. We pretend that the resources are updated. We know that under this pretense P holds but if holds under this fictional update it should have held under the resources that we began with. This lemma, the definition of the naught modality and some of the cases of Lemma 4.4 are the only places in this development that we have to define/prove things directly inside the model of Iris.

From Lemma 4.6 and properties of plainness we can derive the following lemma pertaining the interaction between the future modality and plainness.

Lemma 4.7 (Future modality and plainness). *If P is plain and $Q \vdash P$ then*

$$(\models\!\{n\}\!\models\!\triangleright Q) \vdash (\models\!\{n\}\!\models\!\triangleright Q) * \triangleright^n P$$

This lemma intuitively says that if we know that some plain fact holds in the "future", that is in n steps by updating the resources then we know that it holds in n steps regardless of updating resources. The best way to appreciate what this lemma brings to the table so to speak is to see how we can use it in action. Let us consider the following lemma which proves something along the lines of idempotency.

Lemma 4.8 (A very basic variant of idempotency). *If $\square(\forall \gamma, h_1'.\ IC^{\gamma}\ e \ \{\!\{v.\ \exists h_2', v'.\ \langle h_1', e' \rangle \to^* \langle h_2', v' \rangle\}\!\})$ then*

$$\forall \gamma, h_1'.\ IC^{\gamma}\ e \ \{\!\{v.\ \exists h_2', v', w'.\ \langle h_1', (e', e') \rangle \to^* \langle h_2', (v', w') \rangle\}\!\}$$

Proof. Let assume that we have

(1) h_1', γ_h (2) h_1, h_2, v, n (3) $\boxed{\bullet\, \mathsf{excl}(h_1)}^{\gamma_h}$ (4) $\langle h_1, e \rangle \to^n \langle h_2, v \rangle$

So we have to show $\models\!\{n\}\!\models\!\triangleright \boxed{\bullet\, \mathsf{excl}(h_2)}^{\gamma_h} * \exists h_2', v', w'.\ \langle h_1', (e', e') \rangle \to^* \langle h_2', (v', w') \rangle$. We can allocate a fresh instance $\boxed{\bullet\, \mathsf{excl}(h_1)}^{\gamma'}$ and subsequently duplicate and instantiate (with h_1', this fresh allocation and (4)) our assumption to get

(5) $\models\!\{n\}\!\models\!\triangleright \boxed{\bullet\, \mathsf{excl}(h_2)}^{\gamma'} * \exists h_2', v'.\ \langle h_1', e' \rangle \to^* \langle h_2', v' \rangle$

Now using Lemma 4.7 on (5) we get $\triangleright^n \exists h_2', v'.\ \langle h_1', e' \rangle \to^* \langle h_2', v' \rangle$ and since later and quantifiers commute we get that there exists $h^\dagger$ and $v^\dagger$ such that $\triangleright^n \langle h_1', e' \rangle \to^* \langle h^\dagger, v^\dagger \rangle$ and the latter fact (using Lemma 4.1 case 4) gives us

(6) $\models\!\{n\}\!\models\!\triangleright \langle h_1', e' \rangle \to^* \langle h^\dagger, v^\dagger \rangle$

Now we can instantiate our assumption by $h^\dagger$ (3) (4) to get

(7) $\models\!\{n\}\!\models\!\triangleright \boxed{\bullet\, \mathsf{excl}(h_2)}^{\gamma_h} * \exists h_2', v'.\ \langle h^\dagger, e' \rangle \to^* \langle h_2', v' \rangle$

Now we can use Lemma 4.1 case 3 in (6) and (8) to get

(8) $\models\!\{n\}\!\models\!\triangleright \langle h_1', e' \rangle \to^* \langle h^\dagger, v^\dagger \rangle * \boxed{\bullet\, \mathsf{excl}(h_2)}^{\gamma_h} * \exists h_2', v'.\ \langle h^\dagger, e' \rangle \to^* \langle h_2', v' \rangle$

By monotonicity of the future modality we get

(9) $\langle h_1', e' \rangle \to^* \langle h^\dagger, v^\dagger \rangle$
(10) $\boxed{\bullet\, \mathsf{excl}(h_2)}^{\gamma_h}$
(11) h_2', v'
(12) $\langle h^\dagger, e' \rangle \to^* \langle h_2', v' \rangle$

and we have to show

$$\boxed{\bullet\, \mathsf{excl}(h_2)}^{\gamma_h} * \exists h_2', v', w'.\ \langle h_1', (e', e') \rangle \to^* \langle h_2', (v', w') \rangle$$

which should be trivial now. □

Although the relation between e and e' in the assumption of this lemma is not the exact definition that we will use for our notion relatedness it is close enough that the actual proof of the idempotency lemma is very similar. Indeed the proof presented above can be considered as a rough proof sketch for that lemma.

4.3 Weakest preconditions and Invariants unsuitable

Previous works on representing logical relations in Iris (Krebbers et al. 2017b; Krogh-Jespersen et al. 2017b) have used Iris's weakest preconditions and invariants for representing their logical relations. In this subsection we discuss why we opted not to use these constructions. Understanding this subsection is in no way necessary for understanding the main results of the paper.

4.4 Iris and Kripke worlds

In this subsection we describe the relation between the future modality as defined in the earlier part of this section in Iris and future worlds in logical relations based on Kripke worlds. We also argue why this is not possible to use weakest preconditions to define the logical relation as is usual practice when defining logical relations in Iris. Understanding this subsection is in *no way* necessary or helpful for understanding and appreciating the main results of this paper, i.e., the logical relation presented and the proof of theorems that justify purity of STLang.

Roughly speaking, the logical relatedness of two expressions $Rel(e, e')$ usually states something along the lines that whenever e reduces to a value v then so does e' reduce to a value v' and v and v' are suitably related, $VRel(v, v')$. Notice the similarity between logical relatedness and contextual refinement.

For logical relations based on Kripke worlds the relation is parameterized by a *world*. Each world W has a step index $W.idx$ and a relation $W.Rel$ that is mapping from the set of worlds with smaller step indexes to interpretations of values for each pair of memory locations related by W. The logical relatedness $Rel(W, e, e')$ would then more or less be stating that whenever e reduces in n steps, and the step-index of W, $W.idx$, is bigger than n then there is a *future* world, $W \sqsubseteq W'$, such that $W.idx - W'.idx = n$ and $VRel(W', v, v')$. The future relation is defined as follows:

$$W \sqsubseteq W' \triangleq W'.idx \leq W.idx \land \exists R_1, R_2.\ W'.Rel = R_1 \uplus R_2 \land W.Rel\ |_{W'.idx} = R_1$$

It basically says that a world W' is a future world of W if it has a less step-index and the relation of W' can be divided in two disjoint parts one which is the same as the relation of W when the latter is restricted to the step index of W'.

Working in Iris, we do not need to worry about step-indexes or worlds. These are all taken care of by Iris and buried under layers of abstraction.

In fact the underlying model of Iris involves step-indexes and Kripke worlds. However, these constructs are hidden deep under layers of abstraction and the user needs not to know or care about them. The *only* way to logically refer to step-indexes is the later modality. As to the contents of the worlds, the update modality $\Rrightarrow P$ allows us to talk about the fact that there are *extensions* of the underlying worlds that P is satisfied.

More formally, a proposition $P : iProp$ in Iris is in fact internally defined as $\widehat{P} : \mathbb{N} \longrightarrow iRes \longrightarrow \mathrm{Prop}$ where $iRes$ is the type of Iris resources (the parallel of worlds in Iris) and Prop is the type of propositions of the meta logic. Each Iris proposition basically says at each step-index which worlds it satisfy it. Notice that in Iris (Jung et al. 2016), resources themselves, much like Kripke worlds, are step-indexed. The basic connectives of Iris are defined in the model the way one would expect.[5]

$$\begin{aligned}
\widehat{P \wedge Q} &\triangleq \lambda n\, r.\; \widehat{P}\; n\; r \wedge \widehat{Q}\; n\; r \\
\widehat{P * Q} &\triangleq \lambda n\, r.\; \exists r_1, r_2.\; r = r_1 \hat{\cdot} r_2 \wedge \widehat{P}\; n\; r_1 \wedge \widehat{Q}\; n\; r_2 \\
\widehat{\forall x : A.\; Q} &\triangleq \lambda n\, r.\; \forall x : A.\; \widehat{(P\; x)}\; n\; r \\
&\vdots
\end{aligned}$$

Here $\hat{\cdot}$ is an operation that combines to worlds similar to disjoint union but is defined based on the operations of the monoids that are used for ghost states.[6] In particular the later modality and update modality are defined as follows:

$$\begin{aligned}
\widehat{\triangleright P} &\triangleq \lambda n\, r.\; \begin{cases} \widehat{P}\,(n-1)\; r & \text{if } n > 1 \\ \top & n = 0 \end{cases} \\
\widehat{\Rrightarrow P} &\triangleq \lambda n\, r.\; \forall r_f.\; \checkmark_n(r \hat{\cdot} r_f) \Rightarrow \exists r'.\; \checkmark_n(r' \hat{\cdot} r_f) \wedge \widehat{P}\; n\; r'
\end{aligned}$$

Here $\checkmark_n(r)$ is validity up-to n steps for a step-indexed resource r. This is defined similar to $\hat{\cdot}$ based on the validity of the monoids that are used for the ghost states. In plain words, $\triangleright P$ at step index 0 accepts any world and at each step-index $n > 0$ accepts any world at step-index n that P would accept at step-index P. In other words, it only accepts worlds that are (1 step) in the future (without any extensions) of the worlds that P would accept. The proposition $\Rrightarrow P$ on the other hand, accepts worlds that can be extended so that they satisfy P. When put together $\Rrightarrow \triangleright P$ exactly spells out the condition for (1 step) future worlds that satisfy P.

$$\widehat{\Rrightarrow \triangleright P} \triangleq \lambda n\, r.\; \begin{cases} \forall r_f.\; \checkmark_n(r \hat{\cdot} r_f) \Rightarrow \exists r'.\; \checkmark_n(r' \hat{\cdot} r_f) \wedge \widehat{P}\,(n-1)\; r' & \text{if } n > 1 \\ \forall r_f.\; \checkmark_n(r \hat{\cdot} r_f) \Rightarrow \exists r'.\; \checkmark_n(r' \hat{\cdot} r_f) & n = 0 \end{cases}$$

Hence the future modality $\models\!\{n\}\!\!\Rrightarrow\! P$ is basically satisfied by worlds for which there is a future state n steps away that satisfies P.

5 THE LOGICAL RELATION

In this section we define a binary logical relation that can portray contextual approximation. Formally, we define a family of value relations $\mathcal{V}[\![\Xi \vdash \tau]\!]_\Delta$ and a family of expression relations $\mathcal{E}[\![\phi]\!]_\Delta$. The value relation is an Iris relation of type $(Val \times Val) \longrightarrow iProp$ that identifies values of type τ and is defined by induction on τ. The expression relation is defined independently of the value relation and take a value relation and extends it to expressions. The expression relation has the following type:

$$\mathcal{E}[\![]\!]_\Delta : ((Val \times Val) \longrightarrow iProp) \longrightarrow (Expr \times Expr) \longrightarrow iProp$$

The type system is polymorphic, thus to keep track of the semantic interpretations of instantiated types, both relations are indexed by the map:

$$\Delta : Tvar \longrightarrow (((Val \times Val) \longrightarrow iProp) \times \{\tau \in Types \mid \mathsf{FV}(\tau) = \emptyset\})$$

from type variables to interpretation of types and closed types.

[5]Many of the internal representations presented here are simplified versions what is actually defined in the actual model of Iris.

[6]Iris is a general framework can be instantiated by the user picking the monoids that the user needs for representing the ghost state for their particular application.

The expression relation specifies that e approximates e', if a step taken in e can be simulated by zero or more steps in e'. We think of e as the implementation and e' as the specification. Computing the specification program e' is expressed as an Iris assertion:

$$(h, e) \Downarrow_{\Phi}^{\gamma} \triangleq \exists h', v.\ \langle h, e\rangle \longrightarrow_d^* \langle h', v\rangle * \boxed{\bullet\,\mathsf{excl}(h')}^{\gamma} * \Phi(v)$$

which says, that there exists a deterministic reduction from (h, e) to (h', v), the resulting heap h' is owned and the value is in a value relation. The deterministic reduction relations, $\longrightarrow_d$ and $\rightsquigarrow_d$, are defined by the same inference rules as $\longrightarrow$ and $\rightsquigarrow$, except that the only non-deterministic rule, Alloc, is replaced by a deterministic one:

$$\textsc{det-Alloc}\quad \frac{\ell = min(Loc \setminus \mathrm{dom}(h))}{\langle h, \mathtt{ref}(v)\rangle \rightsquigarrow_h \langle h \uplus \{\ell \mapsto v\}, \mathtt{return}\,\ell\rangle}$$

A simulation between e and e' is expressed as the following hoare-like triple:

$$\left\{\!\!\left| \boxed{\bullet\,\mathsf{excl}(h_1')}^{\gamma_h'} * \mathsf{regions} \right|\!\!\right\}\ e\ \left\{\!\!\left| w.\ (h_1', e') \Downarrow_{\mathcal{V}[\![\Xi \vdash \tau]\!]_{\Delta}(w,\cdot)}^{\gamma_h'} * \mathsf{regions} \right|\!\!\right\}_{\gamma_h}$$

The hoare triple reads, when given full ownership of a heap $\boxed{\bullet\,\mathsf{excl}(h_1')}^{\gamma_h'}$ and an assertion regions that keep track of all allocated regions, e terminates with value w. Notice, that the heap is universally quantified and not restricted in any way. Furthermore, a reduction on the specification side exists, such that the terminating values are related. Before discussing how regions are maintained, we describe how we relate values.

For ground types, $\mathcal{V}[\![\Xi \vdash \tau]\!]_{\Delta}$ is simply the identity relation. We say that pairs are related if they are related point-wise. Functions are related if they map related arguments to related results.

The interesting part is how values of types `STRef` ρ *typ* and `ST` $\rho\ \tau$ are interpreted. Ordinarily, we say that two references are related if they contain related values, normally enforced by having the value-relation owning the points-to predicates for the implementation and specification side. However, for logical relation to express PURITY we would like references:

(1) to be simulationally agnostik - specifically, it cannot own any resources tied to any heap.
(2) to assert that the any invariant/relation imposed on a pair of locations are guaranteed to hold in all simulations.
(3) to have a bijection between a location on the left and right to allow for location-comparison.

To solve (1) we use regions as an abstraction. Regions are disjoint related sections respected by all simulations. The ensure agreement agreement on locations and relations between a region and locations for that region we use a monoid of type Reg : $\textsc{Auth}(Type \xrightarrow{\text{fin}} \textsc{Ag}(Names \times Names))$:

$$\boxed{\circ\,\rho \mapsto \mathsf{ag}(\gamma_{bij}, \gamma_{rel}) : \mathsf{Reg}}^{\gamma_{reg}} * \boxed{\circ\,\rho \mapsto \mathsf{ag}(\gamma_{bij}', \gamma_{rel}') : \mathsf{Reg}}^{\gamma_{reg}} \vdash \gamma_{bij} = \gamma_{bij}' \wedge \gamma_{rel} = \gamma_{rel}' \tag{11}$$

The above property says, that regions identified by ρ always agrees on two monoids; a monoid identifying bijections and a monoid that holds invariants for locations on the region. This property and all others in this section are easy derivable from the monoid rules in SECTION MONOIDS. We can of course allocate new such agreements, for new regions, as the property describes below.

$$\boxed{\bullet\,rs : \mathsf{Reg}}^{\gamma_{reg}} \wedge r \notin rs \vdash \boxed{\bullet\,rs : \mathsf{Reg}}^{\gamma_{reg}} * \boxed{\circ\,\rho \mapsto \mathsf{ag}(\gamma_{bij}, \gamma_{rel}) : \mathsf{Reg}}^{\gamma_{reg}} \tag{12}$$

To let an invariant be imposed on a pair of related locations without violating (1), we demand that each region is responsible for enforcing all such invariants. Again, we will use a monoid to facilitate agreement, however, we cannot store arbitrary assertions of type *iProp*. We solve this by having a monoid of type $\mathsf{Rel} : \textsc{Auth}(Loc \times Loc) \xrightarrow{\text{fin}} (\textsc{Ag}(Names))$:

$$\boxed{\circ\,(l, l') \mapsto \mathsf{ag}(\gamma_{pred}) : \mathsf{Rel}}^{\gamma_{reg}} * \boxed{\circ\,(l, l') \mapsto \mathsf{ag}(\gamma'_{pred}) : \mathsf{Rel}}^{\gamma_{reg}} \vdash \gamma_{pred} = \gamma'_{pred} \tag{13}$$

$$\begin{aligned} Hrel &\triangleq (Loc \times Loc) \xrightarrow{\text{fin}} (\textsc{Ag}(Names)) \\ Hbij &\triangleq \mathcal{P}(Loc \times Loc) \end{aligned}$$

However, to show stronger To enable re-ordering of expressions we would like to:

we have a different encoding that assert the existence of

Here we

As indicated by from the introduction one way to interpret ST monads is as expressions with disjoint heaps. This however, is an oversimplification. Consider the two monoids:

sdsds

$$\begin{aligned} \mathsf{region}(\rho, \gamma_h, \gamma'_h) \triangleq \exists r : Hrel, \gamma_{bij}, \gamma_{rel}.\ & \boxed{\circ\,\rho \mapsto \mathsf{ag}(\gamma_{bij}, \gamma_{rel})}^{\gamma_{reg}} * \boxed{\bullet\,\mathsf{excl}(r)}^{\gamma_{rel}} * \\ \mathop{\Huge *}_{(\ell, \ell') \mapsto \mathsf{ag}(\gamma_{pred}) \in r} & \Big(\exists P : (Val \times Val) \to iProp), v, v'.\ \boxed{\circ\,\ell \mapsto \mathsf{ex}(v)}^{\gamma_h} * \\ & \quad \boxed{\circ\,\ell' \mapsto \mathsf{ex}(v')}^{\gamma'_h} * \gamma_{pred} \Mapsto P * \triangleright P(v, v')\Big) \end{aligned}$$

The region predicate says, that for any heap identified by ρ, with an implementation heap γ_h and γ'_h

The working heap

γ_{reg} is a fixed "global" resource name

We define the logical relation in two stages. We define a value relation $\mathcal{V}[\![\tau]\!]_\Delta$ and an expression relation $\mathcal{E}[\![\tau]\!]_\Delta : (Expr \times Expr) \to iProp$.

Where:

$$\mathsf{close}(\Delta, \rho) \triangleq \rho[(\Delta\vec{X}).2/\vec{X}]$$

closes the (open) type ρ using the closed types stored in Δ,

$$bij(g) \triangleq \text{“}g \textit{ is the graph of a bijection.”}$$

We now define the regions and the $\mathsf{region}(\rho, \gamma_h, \gamma'_h)$ predicates. They make use of the two following resource algebras to keep track of the locations associated to each heap regions:

$$\begin{aligned} \mathsf{region}(\rho, \gamma_h, \gamma'_h) \triangleq \exists r : Hrel, \gamma_{bij}, \gamma_{rel}.\ & \\ \boxed{\circ\,\rho \mapsto \mathsf{ag}(\gamma_{bij}, \gamma_{rel})}^{\gamma_{reg}} * \boxed{\bullet\,r}^{\gamma_{rel}} * & \\ \mathop{\Huge *}_{(\ell, \ell') \mapsto \mathsf{ag}(\gamma_{pred}) \in r} & \Big(\exists P : (Val \times Val) \to iProp), v, v'.\ \boxed{\circ\,\ell \mapsto \mathsf{ex}(v)}^{\gamma_h} * \\ & \quad \boxed{\circ\,\ell' \mapsto \mathsf{ex}(v')}^{\gamma'_h} * \gamma_{pred} \Mapsto P * \triangleright P(v, v')\Big) \end{aligned}$$

$$\begin{aligned} \mathsf{regions} \triangleq \exists M : \{\tau \in Types \mid \mathsf{FV}(\tau) = \emptyset\} \xrightarrow{\text{fin}} \textsc{Ag}(Names \times Names).\ & \boxed{\bullet\,M}^{\gamma_{reg}} * \\ \mathop{\Huge *}_{\rho \mapsto \mathsf{ag}(\gamma_{bij}, \gamma_{rel}) \in M} \Big(\exists g : Hbij, r : Hrel.\ \mathsf{finite}(g) \wedge \boxed{\bullet\,g}^{\gamma_{bij}} * bij(g) * \boxed{\circ\,r}^{\gamma_{rel}} * g \subseteq \mathsf{dom}(r)\Big) & \end{aligned}$$

$\textsc{Ag}$ is the agreement monoid,.

$$\begin{aligned}
\mathcal{V}\llbracket \Xi \vdash X \rrbracket_\Delta &\triangleq (\Delta(X)).1 \\
\mathcal{V}\llbracket \Xi \vdash \mathbf{1} \rrbracket_\Delta(v, v') &\triangleq v = v' = () \\
\mathcal{V}\llbracket \Xi \vdash \mathbb{B} \rrbracket_\Delta(v, v') &\triangleq v = v' \in \{\texttt{true}, \texttt{false}\} \\
\mathcal{V}\llbracket \Xi \vdash \mathbb{N} \rrbracket_\Delta(v, v') &\triangleq v = v' \in \mathbb{N} \\
\mathcal{V}\llbracket \Xi \vdash \tau \times \tau \rrbracket_\Delta(v, v') &\triangleq \exists w_1, w_2, w_1', w_2'.\ v = (w_1, w_2) \wedge v' = (w_1', w_2') \wedge \\
&\qquad \mathcal{V}\llbracket \Xi \vdash \tau \rrbracket_\Delta(w_1, w_1') \wedge \mathcal{V}\llbracket \Xi \vdash \tau' \rrbracket_\Delta(w_2, w_2') \\
\mathcal{V}\llbracket \Xi \vdash \tau + \tau' \rrbracket_\Delta(v, v') &\triangleq (\exists w, w'.\ v = \texttt{inj}_1\, w \wedge v' = \texttt{inj}_1\, w' \wedge \mathcal{V}\llbracket \Xi \vdash \tau \rrbracket_\Delta(w, w')) \vee \\
&\qquad (\exists w, w'.\ v = \texttt{inj}_2\, w \wedge v' = \texttt{inj}_2\, w' \wedge \mathcal{V}\llbracket \Xi \vdash \tau' \rrbracket_\Delta(w, w')) \\
\mathcal{V}\llbracket \Xi \vdash \tau \to \tau' \rrbracket_\Delta(v, v') &\triangleq \Box \Big(\forall (w, w').\ \mathcal{V}\llbracket \Xi \vdash \tau \rrbracket_\Delta(w, w') \Rightarrow \mathcal{E}\llbracket \Xi \vdash \tau \rrbracket_\Delta(v\ w, v'\ w')\Big) \\
\mathcal{V}\llbracket \Xi \vdash \forall X.\, \tau \rrbracket_\Delta(v, v') &\triangleq \Box \Big(\forall f, \tau'.\ \mathsf{FV}(\tau) = \emptyset \wedge \mathsf{persistent}(f) \Rightarrow \\
&\qquad \mathcal{E}\llbracket \Xi, X \vdash \tau \rrbracket_{\Delta, X \mapsto (f, \tau')}(v\ \_, v'\ \_)\Big) \\
\mathcal{V}\llbracket \Xi \vdash \mu X.\, \tau \rrbracket_\Delta(v, v') &\triangleq \mu f.\ \exists w, w'.\ v = \texttt{fold}\, w \wedge v' = \texttt{fold}\, w' \wedge \\
&\qquad \triangleright \mathcal{V}\llbracket \Xi, X \vdash \tau \rrbracket_{\Delta, X \mapsto (f, \mathsf{close}(\Delta, \mu X.\, \tau))}(w, w') \\
\mathcal{V}\llbracket \Xi \vdash \mathsf{STRef}\ \rho\ \tau \rrbracket_\Delta(v, v') &\triangleq \exists \ell, \ell', \gamma_{bij}, \gamma_{rel}, \gamma_{pred}.\ v = \ell \wedge v' = \ell' \wedge \boxed{\circ\, \mathsf{close}(\Delta, \rho) \mapsto \mathsf{ag}(\gamma_{bij}, \gamma_{rel})}^{\gamma_{reg}} * \\
&\qquad \boxed{\circ\, \mathsf{ag}(\ell, \ell')}^{\gamma_{bij}} * \boxed{\circ\, (\ell, \ell') \mapsto \mathsf{ag}(\gamma_{pred})}^{\gamma_{rel}} * \gamma_{pred} \Mapsto \mathcal{V}\llbracket \Xi \vdash \tau \rrbracket_\Delta \\
\mathcal{V}\llbracket \Xi \vdash \mathsf{ST}\ \rho\ \tau \rrbracket_\Delta(v, v') &\triangleq \forall \gamma_h, \gamma_h', h_1'. \\
&\qquad \{\!\!\{ \boxed{\bullet\, \mathsf{excl}(h_1')}^{\gamma_h'} * \mathsf{regions} * \mathsf{region}(\mathsf{close}(\Delta, \rho), \gamma_h, \gamma_h') \}\!\!\} \\
&\qquad\ \ \texttt{runST}\ \{v\} \\
&\qquad \{\!\!\{ w.\ (h_1', \texttt{runST}\ \{v'\}) \Downarrow^{\gamma_h'}_{\mathcal{V}\llbracket \Xi \vdash \tau \rrbracket_\Delta(w, \cdot)} * \mathsf{regions} * \mathsf{region}(\mathsf{close}(\Delta, \rho), \gamma_h, \gamma_h') \}\!\!\}_{\gamma_h}
\end{aligned}$$

$$\mathcal{E}\llbracket \phi \rrbracket_\Delta(e, e') \triangleq \forall \gamma_h, \gamma_h', h_1'.\ \{\!\!\{ \boxed{\bullet\, \mathsf{excl}(h_1')}^{\gamma_h'} * \mathsf{regions} \}\!\!\}\ e\ \{\!\!\{ w.\ (h_1', e') \Downarrow^{\gamma_h'}_{\phi(w, \cdot)} * \mathsf{regions} \}\!\!\}_{\gamma_h}$$

Fig. 7. Binary logical relation

The semantics of the term environment ($\mathcal{V}\llbracket \Xi \vdash \Gamma \rrbracket_\Delta$) is defined as follows:

$$\begin{aligned}
\mathcal{V}\llbracket \Xi \vdash \cdot \rrbracket_\Delta(\vec{v}, \vec{v'}) &\triangleq \top \\
\mathcal{V}\llbracket \Xi \vdash \Gamma, x : \tau \rrbracket_\Delta(w\vec{v}, w'\vec{v'}) &\triangleq \mathcal{V}\llbracket \Xi \vdash \tau \rrbracket_\Delta(w, w') * \mathcal{V}\llbracket \Xi \vdash \Gamma \rrbracket_\Delta(\vec{v}, \vec{v'})
\end{aligned}$$

The logical relation for open terms is defined as follows:

$$\Xi \mid \Gamma \vDash e \preceq_{\log} e' : \tau \triangleq \forall \Delta, \vec{v}, \vec{v'}.\ \mathcal{V}\llbracket \Xi \vdash \Gamma \rrbracket_\Delta(\vec{v}, \vec{v'}) \Rightarrow \mathcal{E}\llbracket \Xi \vdash \tau \rrbracket_\Delta(e[\vec{v}/\vec{x}], e'[\vec{v'}/\vec{x}])$$

6 THE PROPERTIES JUSTIFYING PURITY DO HOLD

7 RELATED WORK

Recently, interesting models for type-and-effect systems with regions for concurrent languages has been proposed such as (Krogh-Jespersen et al. 2017b) and (Benton et al. 2016). Both works also use regions as an abstraction, modelling disjoint parts of the heap. (Benton et al. 2016) has many of the same interesting properties as we show but for a language with first-order store and with no dynamic allocation. There, regions are expressed as an equivalence relation on fixed parts of the heap. It is not quite clear, even stripping away the concurrency part, how their work could scale to higher-order store with dynamic allocation. (Krogh-Jespersen et al. 2017b) shows no pure-ness results. One high-level explanation for this is because expressions, even if it has no effect-annotation, still can have non-determinism because of effect-masking/hiding. However, even if this was not the case, their model have the same weakness regarding using weakest-pre as discussed when introducing "if convergent" and their encoding forces a strict relatedness between locations in regions at all times, thereby making it extremely difficult to show f.x lambda hoisting.

In (Moggi and Palumbo 1999), the authors improve on the work of (Launchbury and Peyton Jones 1994) by giving full proofs of safety for languages with constructs similar to `runST` (including textttrunST itself), with an instrumented semantics that errors out if a region is accessed without permission. Those languages are given realistic semantics, ie. with a heap and where locations are values, very similar to ours. Unlike our work, they consider call-by-name and call-by-need languages, which we did not do. However, they did not establish the expected equations of a pure language: they write that, "*Indeed substantially more work is needed to establish soundness of equational reasoning with respect to our dynamic semantics (even for something as unsurprising as β-equivalence).*"

8 CONCLUSION

ACKNOWLEDGMENTS

REFERENCES

Amal Ahmed. 2004. *Semantics of Types for Mutable State.* Ph.D. Dissertation. Princeton University.

N. Benton, M. Hofmann, and V. Nigam. 2016. Effect-Dependent Transformations for Concurrent Programs. In *PPDP*.

Ralf Jung, Robbert Krebbers, Lars Birkedal, and Derek Dreyer. 2016. Higher-order ghost state. In *ICFP*. 256–269.

Robbert Krebbers, Ralf Jung, Ale Bizjak, Jacques-Henri Jourdan, Derek Dreyer, and Lars Birkedal. 2017a. The essence of higher-order concurrent separation logic. In *European Symposium on Programming (ESOP)*.

Robbert Krebbers, Amin Timany, and Lars Birkedal. 2017b. Interactive Proofs in Higher-Order Concurrent Separation Logic. In *POPL*.

Morten Krogh-Jespersen, Kasper Svendsen, and Lars Birkedal. 2017a. A Logical Account of a Type-and-Effect System. In *POPL*.

Morten Krogh-Jespersen, Kasper Svendsen, and Lars Birkedal. 2017b. A relational model of types-and-effects in higher-order concurrent separation logic. In *POPL*.

John Launchbury and Simon L. Peyton Jones. 1994. Lazy Functional State Threads. In *Proceedings of the ACM SIGPLAN 1994 Conference on Programming Language Design and Implementation (PLDI '94)*. ACM, New York, NY, USA, 24–35. DOI: http://dx.doi.org/10.1145/178243.178246

E. Moggi and F. Palumbo. 1999. Monadic Encapsulation of Effects: a Revised Approach. *Electronic Notes in Theoretical Computer Science* 26 (1999), 121 –. DOI: http://dx.doi.org/10.1016/S1571-0661(05)80287-7