Logical Relations in Iris

Amin Timany
imec-Distrinet, KU Leuven, Belgium
amin.timany@cs.kuleuven.be

Robbert Krebbers
Aarhus University, Denmark
mail@robbertkrebbers.nl

Lars Birkedal
Aarhus University, Denmark
birkedal@cs.au.dk

Abstract

We present a formalization of logical relations for the language $\mathsf{F}_{\mu,\mathit{ref},\mathit{conc}}$: a call-by-value higher-order language with impredicative polymorphism, recursive types, general references, and concurrency. The logical relation interpretation is defined in Iris, a state-of-the-art higher-order concurrent separation logic, which in turn is formalized in Coq. The proof effort is made simpler by the use of the novel interactive proof mode for Iris, called IPM [4].

1. Introduction

It is well-known that it is challenging to define logical relations for higher-order programming languages with genereral references and concurrency [1, 2]. One of the main challenges is the so-called type-world circularity [1]. Here we side-step that challenge because we define the logical relation in Iris, a state-of-the-art higher-order concurrent separation logic. Iris is a logic of resources, with built-in support for defining guarded recursive predicates and invariants. We present both a unary and a binary logical relation interpretation of the types of $\mathsf{F}_{\mu,\mathit{ref},\mathit{conc}}$.

The Coq source code for our development can be found at `https://bitbucket.org/logsem/iris-logrel`. The material presented in this abstract is briefly described in [4] as a case study.

2. The language $\mathsf{F}_{\mu,\mathit{ref},\mathit{conc}}$

Iris is not tied to a fixed programming language, but can be instantiated with different programming languages. Here we instantiate it to $\mathsf{F}_{\mu,\mathit{ref},\mathit{conc}}$, which we formalize in Coq using the Autosubst library for De Bruijn terms [5]. De Bruijn terms make it easier to deal with substitution on open terms, which is needed for our proofs.

The terms and types of $\mathsf{F}_{\mu,\mathit{ref},\mathit{conc}}$ can be found in Figure 1. Terms are untyped, so type-level abstraction is written as $\Lambda\, e$ and type application as $e\ \_$. The operational semantics is split into two parts: thread-local head reduction $\to_{\mathsf{h}}$ and thread-pool reduction $\to_{\mathsf{tp}}$, see Figure 2. Both are defined using standard call-by-value evaluation contexts K, whose definition is omitted. Thread-pool reduction is defined on configurations $\rho = (\vec{e}, \sigma)$ consisting of a state σ (a finite partial map from locations to values) and a thread-pool $\vec{e}$ (a list of expressions corresponding to the threads). The thread-pool reduction is defined by interleaving, *i.e.*, by picking a thread and executing it, thread-locally, for one step. The only special case is $\mathtt{fork}\ \{e\}$, which spawns a new thread e, and reduces itself to the unit value $()$.

Typing judgments take the form $\Xi \mid \Gamma \vdash e : \tau$, where Ξ is a context of type variables, and Γ is a context assigning types to program variables. The inference rules for the typing judgment are mostly standard and hence omitted.

3. Unary logical relation

The unary logical relation for $\mathsf{F}_{\mu,\mathit{ref},\mathit{conc}}$ is presented in Figure 3. The logical relation is defined by two relations, indexed over types τ,

$$\begin{aligned}
e &::= x \mid \ell \mid \mathtt{rec}\, f(x) = e \mid \Lambda\, e \mid \mathtt{fold}\, e \mid \mathtt{unfold}\, e \mid e\ e \\
&\quad\ \mid e\ _ \mid \mathtt{fork}\ \{e\} \mid \mathtt{ref}(e) \mid\ !\, e \mid e \leftarrow e \mid \mathtt{CAS}(e, e, e) \\
v &::= n \mid \ell \mid \mathtt{rec}\, f(x) = e \mid \Lambda\, e \mid \mathtt{fold}\, v \\
\tau &::= X \mid \mathbb{N} \mid \tau \to \tau \mid \forall X.\, \tau \mid \mu X.\, \tau \mid \mathtt{ref}(\tau)
\end{aligned}$$

Figure 1. The syntax of $\mathsf{F}_{\mu,\mathit{ref},\mathit{conc}}$ (sums and products omitted).

Thread-local CBV head-reduction (omitted): $(e, \sigma) \to_{\mathsf{h}} (e', \sigma')$

Thread-pool reduction: $(\vec{e}, \sigma) \to_{\mathsf{tp}} (\vec{e}', \sigma')$

$$\frac{(e, \sigma) \to_{\mathsf{h}} (e', \sigma')}{(\vec{e}_1\ K[\, e\,]\ \vec{e}_2, \sigma) \to_{\mathsf{tp}} (\vec{e}_1\ K[\, e'\,]\ \vec{e}_2, \sigma')}$$

$$(\vec{e}_1\ K[\,\mathtt{fork}\ \{e\}\,]\ \vec{e}_2, \sigma) \to_{\mathsf{tp}} (\vec{e}_1\ K[\,()\,]\ \vec{e}_2\ e, \sigma)$$

Figure 2. Operational semantics of $\mathsf{F}_{\mu,\mathit{ref},\mathit{conc}}$.

$$\begin{aligned}
[\![X]\!]_\Delta(v) &\triangleq \Delta(X)(v) \\
[\![\tau_1 \to \tau_2]\!]_\Delta(v) &\triangleq \Box(\forall w.\, [\![\tau_1]\!]_\Delta(w) \to [\![\tau_2]\!]^{\mathcal{E}}_\Delta(v\ w)) \\
[\![\forall X.\, \tau]\!]_\Delta(v) &\triangleq \forall f.\ \Box [\![\tau]\!]^{\mathcal{E}}_{\Delta[X \mapsto f]}(v\ _) \\
[\![\mu X.\, \tau]\!]_\Delta(v) &\triangleq \mu f.\, \exists w.\, v = \mathtt{fold}\, w \wedge \triangleright [\![\tau]\!]_{\Delta[X \mapsto f]}(w) \\
[\![\mathtt{ref}(\tau)]\!]_\Delta(v) &\triangleq \exists \ell.\, v = \ell \wedge \boxed{\exists w.\, \ell \mapsto w * [\![\tau]\!]_\Delta(w)}^{\mathcal{N}.\ell} \\
[\![\tau]\!]^{\mathcal{E}}_\Delta(e) &\triangleq \mathsf{wp}\ e\ \{v.\, [\![\tau]\!]_\Delta(v)\}
\end{aligned}$$

Figure 3. The unary logical relation for $\mathsf{F}_{\mu,\mathit{ref},\mathit{conc}}$.

namely a value interpretation $[\![\tau]\!]_\Delta : \mathit{Val} \to \mathit{iProp}$ and an expression interpretation $[\![\tau]\!]^{\mathcal{E}}_\Delta : \mathit{Expr} \to \mathit{iProp}$. Note that these relations are Iris relations; *iProp* is the type of Iris propositions. Formally, these logical relations are defined on types τ in context Ξ, but we omit the Ξ here for notational simplicity. We furthermore omit product, sum and base types, but these are present in the Coq formalization. The Δ is a mapping from type variables to value interpretations, *i.e.*, $\Delta : \mathit{Tvar} \to \mathit{Val} \to \mathit{iProp}$.

Experts on logical relations will recognize that this definition is extremely compact; this is because we express the relations in Iris. The case for function types expresses the usual requirement that a value is in the interpretation if it maps a value in the argument type to an expression in the result type. The $\Box$ modality (here and elsewhere) is used to ensure that the relation is persistent, which means that it does not depend on any resources. We use this modality, because Iris is a logic of resources, which means that its propositions (*iProp*) generally express ownership of (ghost) resources. The logical relation must be persistent since typing is intuitionistic (consider for example that the context Γ is copied in the usual typing rule for products). The definition for recursive types is given using a recursively defined predicate (the second μ is a fixpoint combinator); this is well-defined in Iris since the recursion variable occurs under the $\triangleright$ guard[1].

We use the Iris invariant $\boxed{\exists w.\, \ell \mapsto w * [\![\tau]\!]_\Delta(w)}^{\mathcal{N}.\ell}$ to express that a value is in the interpretation of a reference type $\mathtt{ref}(\tau)$. The box is (on paper) Iris notation for invariants and the superscript $(\mathcal{N}.\ell)$ is the name of the invariant. Iris uses names to make sure invariants are not used more than once (which would be unsound). This semantics of the type $\mathtt{ref}(\tau)$ states that a value of this type is a location ℓ and, invariantly, the location ℓ contains a value w in memory that is in the interpretation of τ. This use of Iris invariants dispels the need for explicit possible worlds and explicit treatment of the type-world circularity, which is otherwise typical for logical relation for reference types [1, 2].

Finally, the expression relation $[\![\tau]\!]^{\mathcal{E}}_\Delta$ uses Iris weakest preconditions to say that e is in the semantic interpretation of τ, if it is a computation whose possibly resulting value v is in the semantic interpretation of τ. Using the expression relation, we can define the semantic interpretation of types as:

$$\Xi \mid \Gamma \vDash e : \tau \triangleq \forall \Delta \vec{v}.\ \left(\bigwedge\nolimits_i [\![\sigma_i]\!]_\Delta(v_i)\right) \vdash [\![\tau]\!]^{\mathcal{E}}_\Delta(e[\vec{v}/\vec{x}])$$

where $\Gamma = x_1 : \sigma_1, \ldots, x_n : \sigma_n$ and the environments $\Delta : \mathit{Tvar} \to \mathit{Val} \to \mathit{iProp}$ map into persistent interpretations.

For this logical relation, we can now prove:

1. The fundamental theorem of logical relation[2]:

```
Theorem fundamental Γ e τ:Γ ⊢t e : τ → Γ ⊨ e : τ.
```

2. Type soundness, *i.e.*, that reduction of any well-typed expression can never get stuck:

```
Corollary type_soundness e τ e' thp σ σ' :
  [] ⊢t e : τ → rtc step ([e], σ) (e'::thp, σ') →
  is_Some (to_val e') ∨ reducible e' σ'.
```

The first result is proven in Iris using IPM. The latter result is formalized in plain Coq and relies on the fundamental theorem and the adequacy result for Iris, which formalizes that the weakest precondition predicate of Iris really is connected to the operational semantics of $\mathsf{F}_{\mu,ref,conc}$ in the way you would expect.

Note that the corollary `type_soundness` shows the true power of using a proof assistant instead of a standalone tool: we can compose a proof in Iris with the adequacy result of Iris into a corollary that only mentions the typing judgment and the operational semantics. So, one no longer has to trust Iris or IPM!

4. Binary logical relation

We have also defined a binary logical relation for $\mathsf{F}_{\mu,ref,conc}$ and proven that logical relatedness implies contextual approximation. It is not too hard to generalize the unary logical *value* interpretation to a binary relation, but to generalize the *expression* interpretation from the unary logical relation to the binary logical relation, one needs to find some way of expressing a *relation* between two expressions e and e' using weakest precondition predicates, which are unary. This can be done as follows:

$$[\![\tau]\!]^{\mathcal{E}}_\Delta(e, e') \triangleq \forall j K.\, j \Rrightarrow K[\,e'\,] \to \mathsf{wp}\, e \left\{v.\, \exists w.\, j \Rrightarrow K[\,w\,] * [\![\tau]\!]_\Delta(v, w)\right\}$$

Here $j \Rrightarrow K[\,e'\,]$ is a predicate on ghost state, which expresses that the specification side e' is in some evaluation context K for some thread j before we run e. In the post-condition, we have $j \Rrightarrow K[\,w\,]$. Together with an appropriate invariant on ghost state, these predicates ensure that we really are relating the execution of e, which results in a value v to an execution of e', which results in a value w, and that those values are related at the type τ.

Logical relatedness of e and e' is then defined as:

$$\Xi \mid \Gamma \vdash e \leq_{\mathrm{log}} e' : \tau \triangleq \forall \vec{v}\, \vec{v}'\, \Delta\, \rho.\ \boxed{\mathsf{heap}}^{\mathcal{N}} * \boxed{\Psi(\rho)}^{\mathcal{N}'} * \left(\bigwedge\nolimits_i [\![\sigma_i]\!]_\Delta(v_i, v_i')\right) \vdash [\![\tau]\!]^{\mathcal{E}}_\Delta(e[\vec{v}/\vec{x}], e[\vec{v}'/\vec{x}])$$

where $\Gamma = x_1 : \sigma_1, \ldots, x_n : \sigma_n$ and the environments $\Delta : \mathit{Tvar} \to \mathit{Val} \to \mathit{iProp}$ map into persistent interpretations. Here $\boxed{\Psi(\rho)}^{\mathcal{N}'}$ is the invariant on ghost states mentioned above.

The fact that logical relatedness implies contextual approximation is shown by a series of congruence lemmas, corresponding to each of the typing rules (each on average about 10 lines of code). These lemmas are proved in Iris using IPM.

5. Proving logical refinements

We have used the binary logical relation to prove that two fine-grained concurrent implementations of modules contextually refines their coarse-grained counterparts. The first example is a counter module and the second is a stack module. The fine-grained implementations use optimistic concurrency and no locks, whereas the coarse-grained implementations use a spin lock (implemented using a `CAS` loop) to lock the data structure of the module before and after an operation is performed on the data structure.

For the counter, we use an Iris invariant relating the reference cells in the two implementations and the lock used in the coarse-grained implementation. The stack example comes from [7], where it was proved on paper using an invariant formulated using state transition system. State transition systems can be encoded in Iris [3], but in our experience, it is often easier to use direct monoid constructions when working in Coq.

References

[1] A. Ahmed. *Semantics of Types for Mutable State*. PhD thesis, Princeton University, 2004.

[2] L. Birkedal, B. Reus, J. Schwinghammer, K. Støvring, J. Thamsborg, and H. Yang. Step-indexed Kripke models over recursive worlds. In *POPL*, 2011.

[3] R. Jung, D. Swasey, F. Sieczkowski, K. Svendsen, A. Turon, L. Birkedal, and D. Dreyer. Iris: Monoids and invariants as an orthogonal basis for concurrent reasoning. In *POPL*, pages 637–650, 2015.

[4] R. Krebbers, A. Timany, and L. Birkedal. Interactive Proofs in Higher-Order Concurrent Separation Logic, 2016. To appear in POPL'17.

[5] S. Schäfer, T. Tebbi, and G. Smolka. Autosubst: Reasoning with de Bruijn Terms and Parallel Substitutions. In *ITP*, volume 9236 of *LNCS*, pages 359–374, 2015.

[6] M. Sozeau. A new look at generalized rewriting in type theory. *JFR*, 2(1):41–62, 2009.

[7] A. Turon, D. Dreyer, and L. Birkedal. Unifying refinement and Hoare-style reasoning in a logic for higher-order concurrency. In *ICFP*, pages 377–390, 2013.

[1] Formally, one has to show that the value interpretation of types is suitably *non-expansive*, which relies on all the Iris connectives being non-expansive. In the Iris formalization, non-expansiveness is encoded using Coq's setoid machinery [6], which makes it possible to hide most details about non-expansiveness from the end-user. Using this machinery, non-expansiveness of the logical relation is proved automatically.

[2] Contexts Ξ do not appear in the Coq code since we use De Bruijn indices.