Abstract

We present a formalization of logical relations for the language $F_{\mu, ref, conc}$: 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 \[3\].

1. Introduction

It is well-known that it is challenging to define logical relations for higher-order programming languages with general references and concurrency \[1\]. 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 type-level abstraction \[4\]. Iris is not tied to a fixed programming language, but can be instantiated to another one \[5\].

The Coq source code for our development can be found at https://bitbucket.org/logarem/iris-logrel. The material presented in this abstract is briefly described in \[1\] as a case study.

2. The language $F_{\mu, ref, conc}$

Iris is not tied to a fixed programming language, but can be instantiated with different programming languages. Here we instantiate it to $F_{\mu, ref, conc}$, which we formalize in Coq using the Absubst 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 $F_{\mu, ref, conc}$ can be found in Figure 1. Terms are untyped, so type-level abstraction is written as $\Lambda \varepsilon$ and type application as $\varepsilon \tau$.

3. Unary logical relation

The unary logical relation for $F_{\mu, ref, conc}$ is presented in Figure 3. The logical relation is defined by two relations, indexed over types $\tau$.

\begin{align*}
e & := x | \ell | \text{rec}(e) = e | \Lambda e | \text{fold} e | \text{unfold} e | e e \\
& | e . \text{fork} \{ e \} | \text{ref}(e) | ! e | e \leftarrow e | \text{CAS}(e, e, e) \\
v & := n | \ell | \text{rec}(x) = e | \Lambda e | \text{fold} v \\
\tau & := X | N | \tau \rightarrow \tau | \forall X. \tau | \mu X. \tau | \text{ref}(\tau)
\end{align*}

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

\begin{align*}
\text{Thread-local CBV head-reduction (omitted): } (e, \sigma) \rightarrow _h (e', \sigma') \\
\text{Thread-pool reduction: } (e', \sigma) \rightarrow _t (e'', \sigma'') \\
& (e, \sigma) \rightarrow _t (e'', \sigma'') \\
& (\varepsilon_1 K[e] \varepsilon_2, \sigma) \rightarrow _t (\varepsilon_1 K[e'] \varepsilon_2, \sigma') \\
& (\varepsilon_1 K[\text{fork} \{ e \} \varepsilon_2], \sigma) \rightarrow _t (\varepsilon_1 K[() \varepsilon_2], \sigma)
\end{align*}

Figure 2. Operational semantics of $F_{\mu, ref, conc}$.

\begin{align*}
[X]_\Delta(v) & \triangleq \Delta(X)(v) \\
[\tau_1 \rightarrow \tau_2]_\Delta(v) & \triangleq \square(v \rightarrow [\tau_1]_\Delta(w) \rightarrow [\tau_2]_\Delta(w)) \\
[\forall X. \tau]_\Delta(v) & \triangleq \forall f. \square[\tau]_\Delta(\forall \rightarrow f(v)) \\
[\mu X. \tau]_\Delta(v) & \triangleq \mu f. 3w. v = \text{fold} w \land \square[\tau]_\Delta(\forall \rightarrow f(w)) \\
[\text{ref}(\tau)]_\Delta(v) & \triangleq \exists f. v = \ell \Lambda X. [\varepsilon w. \ell \rightarrow w * [\tau]_\Delta(w)]^N,\ell \\
& [\tau]_\Delta(e) = w. e \{ v. [\tau]_\Delta(v) \}
\end{align*}

Figure 3. The unary logical relation for $F_{\mu, ref, conc}$.
intuitionistic (consider for example that the context \( \Gamma \) is copied in the usual typing rule for products). The definition for recursive types is given using a recursively defined predicate (the second \( \mu \) is a fixpoint combinator); this is well-defined in Iris since the recursion variable occurs under the \( \forall \) guard.

We use the Iris invariant
\[
\frac{\exists w, \ell \to w \vdash \tau}[\exists (\Delta \ell)]
\]

We use the Iris invariant
\[
\frac{\exists w, \ell \to w \vdash [\tau.html]_\ell}{\exists (\Delta \ell)}
\]

We use the Iris invariant
\[
\frac{\exists w, \ell \to w \vdash [\tau.html]_\ell}{\exists (\Delta \ell)}
\]

The box is (on paper) Iris notation for invariants and the superscript \((N, \ell)\) 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 \( \text{ref} \) states that a value of this type is a location \( \ell \) and, invariantly, the location \( \ell \) contains a value \( w \) in memory that is in the interpretation of \( \tau \). 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 \( [\tau.html]_\Delta \) uses Iris weakest preconditions to say that \( e \) is in the semantic interpretation of \( \tau \), if it is a computation whose possibly resulting value \( v \) is in the semantic interpretation of \( \tau \). Using the expression relation, we can define the semantic interpretation of types as:

\[
\Xi | \Gamma \vdash e : \tau \equiv \forall \Delta \forall v. \left( \bigwedge \{ [\sigma.html]_\Delta(v_i) \} \vdash [\tau.html]_\Delta(e[v_i/v]) \right)
\]

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

For this logical relation, we can now prove:

1. The fundamental theorem of logical relation[7]

   \textbf{Theorem fundamental} \( \Gamma \vdash e : \tau \Rightarrow \Gamma \vdash e : \tau \).

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

   \textbf{Corollary type soundness} \( e : \tau \to \text{tp} \sigma \sigma' : [\lambda i. e] : \tau \to \text{rtc} \text{step} \{i\} ) \Rightarrow \text{isSome} (\text{to_val} e) \lor \text{reducible} e^i \sigma^i' .

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 \( F_{\mu, \text{ref}, \text{conc}} \) in the way you 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 \( F_{\mu, \text{ref}, \text{conc}} \) 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 \textit{relation} between two expressions \( e \) and \( e' \) using weakest precondition predicates, which are unary. This can be done as follows:

\[
[\tau.html]_\Delta(e, e') \equiv \forall j. j \Rightarrow K[e'] \rightarrow wp \ e \ (v. \exists w. j \Rightarrow K[w] \cdot [\tau.html]_\Delta(v, w))
\]

Here \( j \Rightarrow K[e'] \) 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 \( j \Rightarrow K[w] \). 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 \( \tau \).

Logical relatedness of \( e \) and \( e' \) is then defined as:

\[
\Xi | \Gamma \vdash e \leq_{\text{log}} e' : \tau \equiv \forall \Delta \forall v \forall \sigma^i \forall \sigma'^i. \Delta(v_i) \vdash [\tau.html]_\Delta(e[v_i/v], [\sigma.html]_\Delta(e[v_i/v], [\sigma'^i/v^i])
\]

where \( \Gamma = x_1 : \sigma_1, \ldots, x_n : \sigma_n \) and the environments \( \Delta : \text{Tvar} \to \text{Val} \to \text{iProp} \) map into persistent interpretations. Here \( [\Psi.html]_{\text{prop}} \) 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


