The Next 700 Relational Program Logics

Kenji Maillard,
Cătălin Hrițcu, Exequiel Rivas, Antoine Van Muylde

Inria Paris

Paper conditionally accepted at POPL'20:
https://arxiv.org/abs/1907.05244
two (different) computational monads

\[ \text{M}_1, \text{M}_2 \]

\[ \text{M}_1 \text{A} = \text{S}_1 \rightarrow \text{A} \times \text{S}_1 \]
\[ \text{M}_2 \text{A} = \text{S}_2 \rightarrow \text{A} \times \text{S}_2 \]

relational monad morphism

\[ \theta_{\text{rel}} \]

\[ \text{W}_{\text{rel}} \]

relational specification monad

\[ \text{W}_{\text{rel}} \text{A}_1 \text{A}_2 = \left( (\text{A}_1 \times \text{S}_1) \times (\text{A}_2 \times \text{S}_2) \rightarrow \text{P} \right) \rightarrow \text{S}_1 \times \text{S}_2 \rightarrow \text{P} \]

\[ \theta_{\text{rel}}(c_1, c_2) = \lambda \text{post } s_1 \ s_2. \ \text{post } (c_1 \ s_1, \ c_2 \ s_2) \]

explore syntactic similarity between \( c_1 \) and \( c_2 \)

\( c_1 \) and \( c_2 \) run independently, not something SMT solvable

Solution: define relational program logics, using \( \theta_{\text{rel}} \) for the semantics:

\[ \models c_1 \sim c_2 \{ w \} = \theta_{\text{rel}}(c_1, c_2) \leq w \]

Rules defined using general recipe, \( \forall \text{M}_1, \text{M}_2, \theta_{\text{rel}}, \text{W}_{\text{rel}} \)
General recipe, 3 kinds of rules:

1. Rules from ambient dependent type theory

2. Rules for monadic constructs (sound for all)

3. Rules for effect-specific actions

Recipe for algebraic operations (soundness guaranteed): unfold get and ret then apply $\theta_{rel}$ to them to obtain $w$

This works: state, nondet, IO, RHL (state+loops), RHTT
1st extension (work in progress)

needed for **probabilities**, nondet refinement, ...

\[ W_{rel} A_1 A_2 = ((A_1 \times A_2) \to [0, 1]) \to [0, 1] \]

\[ p, q : [0, 1] \quad r \sim (B_p, B_q) \]

\[ \vdash \text{flip } p \sim \text{flip } q \left\{ \lambda \text{post. } \sum_{b_1, b_2} r(b_1, b_2) \cdot \text{post}(b_1, b_2) \right\} \]

\[ \theta_{rel}(d_1, d_2) = \lambda \text{post. } \inf_{r \sim (d_1, d_2)} \sum_{b_1, b_2} r(b_1, b_2) \cdot \text{post}(b_1, b_2) \]

**Lax relational monad morphism:**

\[ \theta_{rel}(\text{bind}^{M_1} m_1 f_1, \text{bind}^{M_2} m_2 f_2) \leq \text{bind}^{W_{rel}} (\theta_{rel}(m_1, m_2)) (\theta_{rel} \circ (f_1, f_2)) \]
2\textsuperscript{nd} extension (for exceptions)

\[ W_{\text{rel}}^{\text{Exc}}(A_1, A_2) = ((A_1 + E_1) \times (A_2 + E_2) \to \mathbb{P}) \to \mathbb{P} \]

\[ \vdash m_1 \sim m_2 \{ w^m \} \quad \forall a_1, a_2 \vdash f_1 a_1 \sim f_2 a_2 \{ w^f(a_1, a_2) \} \]

\[ \vdash \text{bind}^{M_1} m_1 f_1 \sim \text{bind}^{M_2} m_2 f_2 \{ \text{bind}^{W_{\text{rel}}} w^m w^f \} \]

\textbf{let} \ \text{bind}^{W_{\text{rel}}}^{\text{Exc}} w_m (w_{f_1} : A_1 \to ((B_1 + E_1) \to \mathbb{P}) \to \mathbb{P}) \to \mathbb{P}) \ w_f \ \varphi =

\[ w_m (\lambda x : (A_1 + E_1) \times (A_2 + E_2). \]

\textbf{match} \ x \ \textbf{with}

\[ | \text{Inl} \ a_1, \text{Inl} \ a_2 \to w_f \ a_1 \ a_2 \ \varphi \]

\[ | \text{Inr} \ e_1, \text{Inr} \ e_2 \to \varphi (\text{Inr} \ e_1, \text{Inr} \ e_2) \]

\[ | \text{Inl} \ a_1, \text{Inr} \ e_2 \to w_{f_1} \ a_1 (\lambda be. \ \varphi \ be (\text{Inr} \ e_2)) \]

\[ | \text{Inr} \ e_1, \text{Inl} \ a_2 \to w_{f_2} \ a_2 (\lambda be. \ \varphi (\text{Inr} \ e_1) \ be) \) \]
2\textsuperscript{nd} extension is complex!

\[ W_{\text{rel}}^{\text{Exc}}(A_1, A_2) = ((A_1 + E_1) \times (A_2 + E_2) \rightarrow \mathcal{P}) \rightarrow \mathcal{P} \]

\[ \Gamma \models c_1 \{w_1\} \sim c_2 \{w_2\} | w_{\text{rel}} = \left( \begin{array}{c}
\forall \gamma_1 : \Gamma_1, \theta_1(c_1 \gamma_1) \leq w_1 \gamma_1, \\
\forall \gamma_2 : \Gamma_2, \theta_2(c_2 \gamma_2) \leq w_2 \gamma_2, \\
\forall (\gamma_1, \gamma_2) : \Gamma_1 \times \Gamma_2, \theta_{\text{rel}}(c_1 \gamma_1, c_2 \gamma_2) \leq w_{\text{rel}}(\gamma_1, \gamma_2) \end{array} \right) \]

We tame some of the complexity by switching to a \textit{relational} dependent type theory (embedded in Coq)

The first relational program logic for catchable exceptions
Conclusions

Once we're completely done with the theory ... ... and **work out some more examples** ... **this could be a good fit for F*!**

**EasyCrypt-style relational verification**

– for an **actual programming language** with **dependent types** and tons of other goodies
– for **arbitrary** effects, relational specification monads, and relational monad morphisms

**Verify your crypto proofs entirely in F*!**