Proof Engineering for Program Logics in Isabelle/HOL

Lecture 4: Rely-Guarantee Reasoning in Isabelle

Chelsea Edmonds
University of Western Australia
chelsea.edmonds@uwa.edu.au

ANU Logic Summer School 2025

Course Overview

Lectures:

- Basic reasoning on programs in Isabelle/HOL
- Program Logics: Hoare and Rely-Guarantee
- A side quest: Intro to Coinduction in Isabelle/HOL
- **Formally defining Rely-guarantee reasoning**
- Modular proofs in Isabelle/HOL and more.

Mix of theory and Isabelle/HOL implementations/proofs.

Lecture 4 Overview

- Recall Rely-Guarantee Logic
- A trace-based Isabelle implementation
- An inductive based Isabelle implementation
- A coinductive based Isabelle implementation
- Comparing soundness proofs.
- A note on equivalence.

Acknowledgement: This lecture reflects recent joint work with Andrei Popescu, Jamie Wright, and John Derrick at the University of Sheffield.

Rely-Guarantee Semantics Approaches

Recall our four ways of defining $\models \{P, R\} c \{G, Q\}$:

- **Trace-Based Approach**
  (Xu et al. 1997 [5] , formalised in Isabelle by Nieto 2003 [3])
- Reachability Approach (Coleman & Jones, 2007 [1])
- **Inductive Approach**
  (Vafeiadis & Parkinson, 2007 [4], formalised in Isabelle by Jackson et al. 2024 [2])
- **A Coinductive Approach?** (New work!)

Let's now formally define these.

Trace-Based Approach

Working with Traces

Recall our slightly more formal way of explaining RG logic from lecture 2.

Consider a command whose execution trace has environment steps $\epsilon(\sigma_i, \sigma_{i+1})$ and program steps $\tau(\sigma_i, \sigma_{i+1})$, where σ_i represents the state after i steps:

$$\sigma_0 \ldots \tau(\sigma_i, \sigma_{i+1}) \ldots \epsilon(\sigma_j, \sigma_{j+1}) \ldots \sigma_f$$

$\{P, R\}\, C\, \{G, Q\}$ holds means:

- $P\ \sigma_0$ holds
- $Q\ \sigma_f$ holds if the command terminates
- Every environment step ϵ satisfies the rely condition, i.e. $R\ \sigma_j\ \sigma_{j+1}$
- Every program step τ satisfies the guarantee condition, i.e. $G\ \sigma_i\ \sigma_{i+1}$

How do we state this formally (i.e. in a Isabelle translatable way!)?

Working with Traces

We extend our configurations to label steps as either *environment* or *command* (i.e. program) steps: (l, c, s).

A trace is a series of configurations

$$tr = [(l^i, c^i, s^1), \ldots, (l^n, c^n, s^n)]$$

where:

- $l^{i+1} = C \Rightarrow (c^i, s^i) \Rightarrow (c^{i+1}, s^{i+1})$
- $l^{i+1} = E \Rightarrow c^i = c^{i+1}$

Trace-Based Rely-Guarantee Conditions

And formally defines our RG principles as:

- $tr \models_{pre} P \iff P\, s^1$
- $tr \models_{rely} R \iff \forall i \in \{1..n-1\}.\, l^{(i+1)} = E \Rightarrow R\, s^i\, s^{(i+1)}$
- $tr \models_{guar} G \iff \forall i \in \{1..n-1\}.\, l^{(i+1)} = C \Rightarrow G\, s^i\, s^{(i+1)}$
- $tr \models_{post} Q \iff Q\, s^n$

Our RG clause is valid, $\models \{P, R\} c \{G, Q\}$, iff for all valid traces tr:

$$tr \models_{pre} P \wedge tr \models_{rely} R \Longrightarrow tr \models_{guar} G \wedge \mathrm{final}(c^n) \longrightarrow tr \models_{post} Q$$

Traces in Isabelle

What is required to implement this in Isabelle?

- Introducing labelled configurations as a datatype, with an easy way to still access the core configuration (i.e. functions and basic lemmas).
- A trace definition using the built-in list type.
- Lots and lots of basic lemmas to reason about this trace definition.
- Definitions for each of the RG predicates on traces.
- The final RG semantics definition.
- ... and a lot more lemmas on both of the above!

Isabelle Demo

Counting-Based (Inductive) Approach

The Inductive Safety Definition

We introduce the concept of *safety* of a configuration (c, s).

$$\mathrm{safe}_{(R,G,Q)}\, n\, (c, s)$$

This states that executing n steps of c from a state s is *safe* with respect to our rely, guarantee, and post conditions.

We call (R, G, Q) a *reduced RG clause*. Think of n as a “counter” on the number of interactive program steps.

The Inductive Safety Definition

Formally, safety can be defined as an inductive definition:

$$\mathrm{safe}_{(R,G,Q)}\,0\,(c,s)\quad\text{(Base)}$$

$$\frac{\begin{array}{ll}1. & \forall s'.\ R\ s\ s' \implies \mathrm{safe}_{(R,G,Q)}\,n\,(c,s') \\ 2. & \mathrm{final}\,(c,s) \implies Q\ s \\ 3. & \forall c', s'.\ ((c,s) \Rightarrow (c',s')) \implies G\ s\ s' \land \mathrm{safe}_{(R,G,Q)}\,n\,(c',s')\end{array}}{\mathrm{safe}_{(R,G,Q)}\,(n+1)\,(c,s)}\text{(Step)}$$

Figure 1: The inductive-safety predicate safe

Inductive Safety: Base Case

$$\text{safe}_{(R,G,Q)}\,0\,(c, s) \quad \text{(Base)}$$

Every command in every state is safe after 0 steps... as nothing has happened yet!

This can seem a little redundant... keep that in mind!

Inductive Safety: Inductive Step

$$\frac{\begin{array}{lll} \text{Rely} & 1. & \forall s'.\ R\ s\ s' \implies \text{safe}_{(R,G,Q)}\ n\ (c, s') \\ \text{Post} & 2. & \text{final}\,(c, s) \implies Q\ s \\ \text{Guarantee} + \text{Step} & 3. & \forall c', s'.\ ((c, s) \Rightarrow (c', s')) \implies G\ s\ s' \wedge \text{safe}_{(R,G,Q)}\ n\ (c', s') \end{array}}{\text{safe}_{(R,G,Q)}\ (n+1)\ (c, s)}\ (\text{Step})$$

Intuitively, executing $n+1$ steps of c from state s is *safe* based on the following hypotheses:

1. An environment step to s' that is compliant with the rely-condition $(R\ s\ s')$ produces an n-safe configuration (c, s').
2. If (c, s) is final, then the post-condition holds $(Q\ s)$.
3. Any computation step $((c, s) \Rightarrow (c', s'))$ produces an n-safe configuration (c', s') and a guarantee-compliant state change $(G\ s\ s')$.

Semantics via Safety

The inductive counting-based (VP) validity of an RG clause can now be defined:

$$\models_{\mathsf{VP}} \{P, R\}\ c\ \{G, Q\} \iff \forall s \in \mathsf{State}, \forall n \in \mathbb{N}.\ P\ s \implies \mathsf{safe}_{(R,G,Q)}\, n\, (c, s)$$

Inductive Counting-Based Approach in Isabelle

What do we need to implement this in Isabelle?

- An inductive definition of the safety predicate.
- Some helper lemmas on the safety predicate.
- The RG validity definition, and some helper lemmas to break it down.

Inductive Counting-Based Approach in Isabelle

Recall from last lecture what an inductive definition in Isabelle provides:

- Introduction rules from the base and step cases.
- A cases rule:

$$\forall n, c, s.\ \left(n = 0\ \vee\ \begin{pmatrix} \exists m.\ n = m + 1 \\ \wedge\ \left(\forall s'.\ R\,s\,s' \longrightarrow \mathsf{safe}_{(R,G,Q)}\,m\,(c, s')\right) \\ \wedge\ \left(\mathsf{final}(c, s) \longrightarrow Q\,s\right) \\ \wedge\ \left(\forall c'\,s'.\ (c, s) \Rightarrow (c', s') \longrightarrow G\,s\,s' \wedge \mathsf{safe}_{(R,G,Q)}\,m\,(c', s')\right) \end{pmatrix} \right)$$

$$\Longrightarrow\ \mathsf{safe}_{(R,G,Q)}\,n\,(c, s)$$

Inductive Counting-Based Approach in Isabelle

And lastly:

- Induction rule

$$\frac{\forall c\, s.\ K\, 0\, (c, s) \qquad \mathsf{safe}_{(R,G,Q)}\, n\, (c, s) \qquad \forall m\, c\, s.\ \mathsf{safe}_{(R,G,Q)}\, m\, (c, s) \wedge K\, m\, (c, s) \rightarrow K\, (m + 1)\, (c, s)}{K\, n\, (c, s)}\,(\text{induct})$$

These are essential for working with the definition in a formal environment!

Isabelle Demo

Coinductive Approach

Re-examining our Inductive Definition

$$\mathsf{safe}_{(R,G,Q)}\, 0\, (c, s) \quad \text{(Base)}$$

$$\frac{\begin{array}{ll} 1. & \forall s'.\ R\ s\ s' \implies \mathsf{safe}_{(R,G,Q)}\, n\, (c, s') \\ 2. & \mathsf{final}\, (c, s) \implies Q\ s \\ 3. & \forall c', s'.\ ((c, s) \Rightarrow (c', s')) \implies G\ s\ s' \wedge \mathsf{safe}_{(R,G,Q)}\, n\, (c', s') \end{array}}{\mathsf{safe}_{(R,G,Q)}\, (n+1)\, (c, s)}\ \text{(Step)}$$

Observe that:

- The base case is *always true*, i.e. a little redundant.
- When intuitively thinking about program safety (which is over any number of steps), carrying around a step counter (n) is a little unnatural.

A Coinductive Safety Definition

$$\begin{array}{c}
\begin{array}{ll}
1. & \forall s'.\ R\ s\ s' \implies \mathsf{safeC}_{(R,G,Q)}\,(c, s') \\
2. & \mathsf{final}\,(c, s) \implies Q\ s \\
3. & \forall c', s'.\ ((c, s) \Rightarrow (c', s')) \implies G\ s\ s' \land \mathsf{safeC}_{(R,G,Q)}\,(c', s')
\end{array} \\
\hline\hline
\mathsf{safeC}_{(R,G,Q)}\,(c, s)
\end{array}\ (\mathsf{StepC})$$

Figure 2: The coinductive-safety predicate safeC

The implicit coiterative nature of coinduction means we no longer need to explicitly count the number of steps!

We use the double-line rule to indicate it is a coinductive defintion.

A Coinductive Safety Definition

$$\dfrac{\begin{array}{ll} 1. & \forall s'.\ R\ s\ s' \implies \mathsf{safeC}_{(R,G,Q)}\,(c, s') \\ 2. & \mathsf{final}\,(c, s) \implies Q\ s \\ 3. & \forall c', s'.\ ((c, s) \Rightarrow (c', s')) \implies G\ s\ s' \wedge \mathsf{safeC}_{(R,G,Q)}\,(c', s') \end{array}}{\mathsf{safeC}_{(R,G,Q)}\,(c, s)}\ (\mathsf{StepC})$$

The intuition behind this definition is practically identical to our inductive approach. A configuration (c, s) is *safe* with respect to a reduced RG clause (R, G, Q) if:

1. An environment step to s' that is compliant with the rely-condition ($R\ s\ s'$) produces a safe configuration (c, s').
2. If (c, s) is final, then the post-condition holds $(Q\ s)$.
3. Any computation step $((c, s) \Rightarrow (c', s'))$ produces another safe configuration (c', s') and a guarantee-compliant state change $(G\ s\ s')$.

Semantics via Coinductive Safety

The coinductive validity of an RG clause can now be defined:

$$\models_{\mathsf{C}} \{P, R\}\ c\ \{G, Q\} \iff \forall s \in \mathsf{State}.\ P\ s \implies \mathsf{safeC}_{(R,G,Q)}\,(c, s)$$

i.e. safety for any number of steps.

Coinductive Approach in Isabelle

What do we need to implement this in Isabelle?

- A coinductive definition of the safety predicate.
- Some helper lemmas on the safety predicate.
- The RG validity definition, and some helper lemmas to break it down.

In other words, the implementation is very similar to the inductive approach!

Coinductive Approach in Isabelle

Recall from last lecture what a coinductive definition in Isabelle provides:

- Introduction and cases rule work as per inductive case (minus the n variable/base case!)
- A coinduction rule:

$$\dfrac{\forall c\, s.\ K(c,s) \longrightarrow \left(\begin{array}{l} (\forall s'.\ R\ s\ s' \longrightarrow K(c,s') \vee \mathrm{safeC}_{R,G,Q}(c,s')) \\ \wedge\ \ (\mathrm{final}\ (c,s) \longrightarrow Q\ s) \\ \wedge\ \ (\forall c'\, s'.\ (c,s) \Rightarrow (c',s') \longrightarrow \\ \qquad\qquad G\ s\ s' \wedge (K(c',s') \vee \mathrm{safeC}_{R,G,Q}(c,s'))) \end{array}\right)}{\mathrm{safeC}_{R,G,Q}(c,s)}$$

Isabelle Demo

Soundness and Proof Engineering

Soundness of RG Proof System vs Semantics

As with Hoare Logic, we want to show that our RG proof system (from lecture 2) is sound with respect to our RG semantics definitions and operational semantics.

$$\vdash \{P, R\}\ c\ \{G, Q\} \Longrightarrow \models \{P, R\}\ c\ \{G, Q\}$$

Our choice of validity definition has a significant impact on the effort required!

Soundness Proof Strategies

For a given command syntax c, the proof strategy changes as follows.

- $\models_{\text{XRH}}$ quantifies over multiple execution steps via explicit traces, hence we must:
  - First characterize the traces that could start with c, our “inversion lemmas”
  - Then induct over those explicit traces
- $\models_{\text{VP}}$ quantifies over a single step.
  - “inversion lemmas” come automatically from the cases rule of our inductive definition.
  - Apply natural number induction rule over n.
- $\models_{\text{C}}$ quantifies over a single step.
  - “inversion lemmas” come automatically from the cases rule of our coinductive definition.
  - Apply coinductive rule from our coinductive definition.

Isabelle Demo

An aside: equivalence

Our research has shown that all three approaches (as well as the reachability approach) are also equivalent.

So why bother with doing the soundness proof for each approach?

- Understanding the benefits and disadvantages from a proof engineering perspective.
- Gain a deeper understanding of each approach from an RG reasoning perspective.
- Uncover new results about the relationship between inductive and coinductive approaches.
- *Use proof engineering results to inform future work* (e.g. soundness of a different semantics, variations on the program logics etc).

The point of using a proof assistant isn't always just to verify a set result!

Next Time

Next Lecture:

- Locales and Type-classes
- Introducing Modularity and Abstractness via Locales.

Isabelle exercises/extended work

- This is part of some recent work to appear in a paper shortly (see project website here: `https://covert-project.github.io/site/publications.html`)
- Try out a trace proof for comparison!

References i

Joey W. Coleman and Cliff B. Jones.
A structural proof of the soundness of rely/guarantee rules.
J. Log. Comput., 17(4):807–841, 2007.

Vincent Jackson, Toby Murray, and Christine Rizkallah.
A generalised union of rely-guarantee and separation logic using permission algebras.
In Yves Bertot, Temur Kutsia, and Michael Norrish, editors, *15th International Conference on Interactive Theorem Proving, ITP 2024, September 9-14, 2024, Tbilisi, Georgia*, volume 309 of *LIPIcs*, pages 23:1–23:16. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2024.

References ii

Leonor Prensa Nieto.
The Rely-Guarantee method in Isabelle/HOL.
In Pierpaolo Degano, editor, *Programming Languages and Systems, 12th European Symposium on Programming, ESOP 2003, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2003, Warsaw, Poland, April 7-11, 2003, Proceedings*, volume 2618 of *Lecture Notes in Computer Science*, pages 348–362. Springer, 2003.

References iii

- Viktor Vafeiadis and Matthew J. Parkinson.
  A marriage of rely/guarantee and separation logic.
  In Luís Caires and Vasco Thudichum Vasconcelos, editors, *CONCUR 2007 - Concurrency Theory, 18th International Conference, CONCUR 2007, Lisbon, Portugal, September 3-8, 2007, Proceedings*, volume 4703 of *Lecture Notes in Computer Science*, pages 256–271. Springer, 2007.

- Qiwen Xu, Willem P. de Roever, and Jifeng He.
  The rely-guarantee method for verifying shared variable concurrent programs.
  Formal Aspects Comput., 9(2):149–174, 1997.