Iris: Higher-Order Concurrent Separation Logic

Robust Safety

Amin Timany

Aarhus University, Denmark

August 14, 2026

Robust Safety

Our modular reasoning principles imply that we can combine programs that are proven, e.g., the HT-PAR rule.

Question: What can we say about combining a proven correct program with an arbitrary, adversarial program?

Important insight:

— We can express limitations of arbitrary programs in terms Iris propositions and Hoare triples.

— Modular reasoning: we can combine these Hoare triples with those of proven correct programs.

— We obtain proofs about the result of linking a proven correct program with an arbitrary, adversarial program.

Robust Safety

Notice: We consider arbitrary programs which may crash.

Hence, we use a weaker, non-progressive variant of Hoare triples which allow the program to get stuck:

$$\{P\}_{\not\downarrow}\ e\ \{x.\ Q\}$$

Just like ordinary Hoare triples the non-progressive version also enforces invariants and does not allow assertion failures.

$$\{P\}\ e\ \{x.\ Q\} \vdash \{P\}_{\not\downarrow}\ e\ \{x.\ Q\}$$

All the modular reasoning principles of ordinary Hoare triples, *e.g.*,, HOARE-FRAME, HOARE-PAR, *etc.*, also hold for the non-progressive variant.

Robust Safety

Similar to Correct_φ we define NonProg_φ which

— does not guarantee progress (programs may get stuck)

— requires no assertion failures

— if the program terminates to a value v, $\varphi(v)$ must hold

Theorem (Non-progressive Adequacy)

If we prove

$$\vdash \{\mathit{True}\}_{\not\lightning}\ e\ \{x.\ \varphi(x)\}$$

in Iris *for a suitable* φ, *then* $\text{NonProg}_\varphi(e)$

Robust Safety, an Example

The following *even_counter* module returns two functions, one to read the value and one to increment it by two.

$$
\begin{aligned}
\mathit{even\_counter} \triangleq\ & \mathsf{let}\ c := \mathsf{ref}\,(0)\ \mathsf{in} \\
& \mathsf{let}\ \mathit{incr}\ \_ := \mathsf{FAA}\ c2\ \mathsf{in} \\
& \mathsf{let}\ \ \mathit{read}\ \_ := \mathsf{let}\ x := !\,c\ \mathsf{in}\ \mathsf{assert}(x\ \%\ 2 = 0);\ x\ \mathsf{in} \\
& (\mathit{incr}, \mathit{read})
\end{aligned}
$$

Question: can we prove $\mathsf{NonProg}_{\mathit{isEven}}(\mathit{prog})$?

$$
\mathit{prog} = \mathsf{let}\ (\mathit{incr}, \mathit{read}) := \mathit{even\_counter}\ \mathsf{in}\ \mathit{adversary};\ \mathit{read}\ ()
$$

where adversary is a program with no hard-coded locations or assertions.

Robust Safety, an Example

The following *even_counter* module returns two functions, one to read the value and one to increment it by two.

$$
\begin{aligned}
\mathit{even\_counter} \triangleq\ & \mathsf{let}\ c := \mathsf{ref}\,(0)\ \mathsf{in} \\
& \mathsf{let}\ \mathit{incr}\ \_ := \mathsf{FAA}\ c2\ \mathsf{in} \\
& \mathsf{let}\ \ \mathit{read}\ \_ := \mathsf{let}\ x := !\,c\ \mathsf{in}\ \mathsf{assert}(x\ \%\ 2 = 0);\ x\ \mathsf{in} \\
& (\mathit{incr}, \mathit{read})
\end{aligned}
$$

Question: can we prove $\mathsf{NonProg}_{\mathit{isEven}}(\mathit{prog})$?

$$
\mathit{prog} = \mathsf{let}\ (\mathit{incr}, \mathit{read}) := \mathit{even\_counter}\ \mathsf{in}\ \mathit{adversary};\ \mathit{read}\ ()
$$

where adversary is a program with no hard-coded locations or assertions.

Hint: the language does not support pointer arithmetic; the only way to get a pointer is if the program allocates it or if it is passed to it from another part of the program.

Robust Safety, an Example

Question: is our assumption of no pointer arithmetic reasonable?

Yes, this property holds for our high-level programming language. Hence, we can indeed prove $\mathsf{NonProg}_{isEven}(prog)$ from the previous slide.

Question: But more realistically, programs can be linked to adversary programs written in more low-level languages, e.g., directly in assembly. Surely, those can perform pointer arithmetic.

Yes, however, some modern hardware architectures (still mostly in research labs) feature so-called *capabilities* which essentially restrict pointer arithmetic which can be used to enable the guarantee that we have assumed about pointers.

See our work on studying the assembly language capability machines in Iris (the Cerise paper) for more details.

How Do We Prove Robust Safety?

Question: how do we prove $\mathsf{NonProg}_{isEven}(prog)$?

$$prog = \mathsf{let}\ (incr, read) := even\_counter\ \mathsf{in}\ adversary;\ read\ ()$$

where adversary is a program with no hard-coded locations or assertions.

How Do We Prove Robust Safety?

Question: how do we prove $\mathsf{NonProg}_{isEven}(prog)$?

$$prog = \mathsf{let}\ (incr, read) := even\_counter\ \mathsf{in}\ adversary;\ read\ ()$$

where adversary is a program with no hard-coded locations or assertions.

Modular Reasoning!

How Do We Prove Robust Safety?

We can easily show the following specs for *even_counter*:

$$\{\mathsf{True}\}_{\lightning}$$
$$\quad \mathit{even\_counter}$$
$$\left\{\begin{array}{l} x.\ \exists f, g.\ x = (f, g) \wedge \\ \quad (\forall v.\ \{\mathsf{True}\}_{\lightning}\ f\ v\ \{y.\ y = ()\}) \wedge \\ \quad (\forall v.\ \{\mathsf{True}\}_{\lightning}\ g\ v\ \{y.\ \mathit{isEven}(y)\}) \end{array}\right\}$$

The proof is very similar to the examples we have seen before. We simply use an invariant that asserts the location is always even.

If we somehow had a non-progressive Hoare triple for the adversary program, we could just compose it with the spec above; because modularity!

Logical Relations for Establishing Robust Safety

We define a logical relations model for our language:

- We define relations capturing good values and good expressions
- Similar to the logical relations for typed languages, except relations are not indexed by types
  - Our language has not typed
  - Arbitrary adversarial programs may not be well-typed even if had types
- We show that all adversarial programs (no hard-coded locations or assertions) are good

Logical Relations for Establishing Robust Safety

We define the $good_{val}$ and $good_{exp}$ relations as follows:

$$good_{exp}(e) \triangleq \{\mathsf{True}\}_{\lightning}\ e\ \{x.\ good_{val}(x)\}$$

where $good_{val}(v)$ is inductively as follows:[1]

$$\begin{aligned}
good_{val}(n) &\triangleq \mathsf{True} && \text{if } n \in \mathbb{Z}\\
good_{val}(b) &\triangleq \mathsf{True} && \text{if } b \in \{\mathsf{true}, \mathsf{false}\}\\
good_{val}(()) &\triangleq \mathsf{True}\\
good_{val}(v_1, v_2) &\triangleq \triangleright\, good_{val}(v_1) \wedge \triangleright\, good_{val}(v_2)\\
good_{val}(\mathsf{rec}\ f\ x := e) &\triangleq \forall v.\ \{\triangleright\, good_{val}(v)\}_{\lightning}\ (\mathsf{rec}\ f\ x := e)\ v\ \{x.\ good_{val}(x)\}\\
&\vdots\\
good_{val}(\ell) &\triangleq \boxed{\exists v.\ \ell \mapsto v * good_{val}(v)}
\end{aligned}$$

[1] By induction on the form of values instead of types.

Logical Relations for Establishing Robust Safety

We define the $good_{val}$ and $good_{exp}$ relations as follows:

$$good_{exp}(e) \triangleq \{\mathsf{True}\}_{\not\downarrow}\ e\ \{x.\ good_{val}(x)\}$$

where $good_{val}(v)$ is inductively as follows:[1]

$$\begin{aligned}
good_{val}(n) &\triangleq \mathsf{True} && \text{if } n \in \mathbb{Z}\\
good_{val}(b) &\triangleq \mathsf{True} && \text{if } b \in \{\mathsf{true}, \mathsf{false}\}\\
good_{val}(()) &\triangleq \mathsf{True}\\
good_{val}(v_1, v_2) &\triangleq \triangleright\, good_{val}(v_1) \wedge \triangleright\, good_{val}(v_2)\\
good_{val}(\mathsf{rec}\ f\ x := e) &\triangleq \forall v.\ \{\triangleright\, good_{val}(v)\}_{\not\downarrow}\ (\mathsf{rec}\ f\ x := e)\ v\ \{x.\ good_{val}(x)\}\\
&\vdots\\
good_{val}(\ell) &\triangleq \boxed{\exists v.\ \ell \mapsto v * good_{val}(v)}
\end{aligned}$$

Note the laters! Also, these relations are persistent; why?

[1] By induction on the form of values instead of types.

Logical Relations for Establishing Robust Safety

Theorem (Fundamental Theorem of Robust Safety (FTRS))

Let e be any expression with no hard-coded locations or assertions.
Furthermore, let $\vec{vs}$ be values which are all good.
The following holds:

$$good_{exp}(e[\vec{vs}/\vec{xs}])$$

where $\vec{xs}$ are variables (as many as $\vec{vs}$).

Proof.

By Löb induction and then case analysis on e. □

Robust Safety, an Example (proof)

$$\{\mathsf{True}\}_{\lightning}$$
$$\quad \textit{even\_counter}$$
$$\left\{\begin{array}{l} x.\ \exists f, g.\ x = (f, g) \wedge \\ \quad (\forall v.\ \{\mathsf{True}\}_{\lightning}\ f\ v\ \{y.\ y = ()\}) \wedge \\ \quad (\forall v.\ \{\mathsf{True}\}_{\lightning}\ g\ v\ \{y.\ \mathit{isEven}(y)\}) \end{array}\right\}$$

$$\{\mathsf{True}\}_{\lightning}$$
$$\quad \text{let } (\mathit{incr}, \mathit{read}) := \mathit{even\_counter} \text{ in}$$
$$\quad \mathit{adversary};$$
$$\quad \mathit{read}\ ()$$
$$\{x.\ \mathit{isEven}(x)\}$$

Theorem (Fundamental Theorem of Robust Safety (FTRS))

Let e be any expression with no hard-coded locations or assertions.
Furthermore, let $\vec{vs}$ be values which are all good.
The following holds:

$$\mathit{good}_{exp}(e[\vec{vs}/\vec{xs}])$$

where $\vec{xs}$ are variables (as many as $\vec{vs}$).

Robust Safety, an Example (proof)

$$\{\mathsf{True}\}_{\lightning}$$
$$\left\{\begin{array}{l}(\forall v.\ \{\mathsf{True}\}_{\lightning}\ f\ v\ \{y.\ y = ()\}) \wedge \\ (\forall v.\ \{\mathsf{True}\}_{\lightning}\ g\ v\ \{y.\ \mathit{isEven}(y)\})\end{array}\right\}_{\lightning}$$
$$\quad \text{let } (\mathit{incr}, \mathit{read}) := (f, g) \text{ in}$$
$$\quad \mathit{adversary};$$
$$\quad \mathit{read}\ ()$$
$$\{x.\ \mathit{isEven}(x)\}$$
$$\{x.\ \mathit{isEven}(x)\}$$

$$\{\mathsf{True}\}_{\lightning}$$
$$\quad \mathit{even\_counter}$$
$$\left\{\begin{array}{l}x.\ \exists f, g.\ x = (f, g) \wedge \\ \quad (\forall v.\ \{\mathsf{True}\}_{\lightning}\ f\ v\ \{y.\ y = ()\}) \wedge \\ \quad (\forall v.\ \{\mathsf{True}\}_{\lightning}\ g\ v\ \{y.\ \mathit{isEven}(y)\})\end{array}\right\}$$

Theorem (Fundamental Theorem of Robust Safety (FTRS))

Let e be any expression with no hard-coded locations or assertions.
Furthermore, let $\vec{vs}$ be values which are all good.
The following holds:

$$\mathit{good}_{exp}(e[\vec{vs}/\vec{xs}])$$

where $\vec{xs}$ are variables (as many as $\vec{vs}$).

Robust Safety, an Example (proof)

$$\{\mathsf{True}\}_{\lightning}$$
$$\left\{\begin{array}{l}(\forall v.\ \{\mathsf{True}\}_{\lightning}\ f\ v\ \{y.\ y = ()\}) \wedge \\ (\forall v.\ \{\mathsf{True}\}_{\lightning}\ g\ v\ \{y.\ \mathit{isEven}(y)\})\end{array}\right\}_{\lightning}$$
$$\mathit{adversary}[f, g/\mathit{incr}, \mathit{read}];$$
$$g\ ()$$
$$\{x.\ \mathit{isEven}(x)\}$$
$$\{x.\ \mathit{isEven}(x)\}$$

$$\{\mathsf{True}\}_{\lightning}$$
$$\mathit{even\_counter}$$
$$\left\{\begin{array}{l}x.\ \exists f, g.\ x = (f, g) \wedge \\ \quad (\forall v.\ \{\mathsf{True}\}_{\lightning}\ f\ v\ \{y.\ y = ()\}) \wedge \\ \quad (\forall v.\ \{\mathsf{True}\}_{\lightning}\ g\ v\ \{y.\ \mathit{isEven}(y)\})\end{array}\right\}$$

Theorem (Fundamental Theorem of Robust Safety (FTRS))

Let e be any expression with no hard-coded locations or assertions.
Furthermore, let $\vec{vs}$ be values which are all good.
The following holds:

$$\mathit{good}_{exp}(e[\vec{vs}/\vec{xs}])$$

where $\vec{xs}$ are variables (as many as $\vec{vs}$).

Robust Safety, an Example (proof)

$$\{\mathsf{True}\}_{\lightning}$$
$$\left\{\begin{array}{l}(\forall v.\ \{\mathsf{True}\}_{\lightning}\ f\ v\ \{y.\ y = ()\}) \wedge \\ (\forall v.\ \{\mathsf{True}\}_{\lightning}\ g\ v\ \{y.\ \mathit{isEven}(y)\})\end{array}\right\}_{\lightning}$$
$$\mathit{adversary}[f, g/\mathit{incr}, \mathit{read}];$$
$$\left\{\begin{array}{l}(\forall v.\ \{\mathsf{True}\}_{\lightning}\ f\ v\ \{y.\ y = ()\}) \wedge \\ (\forall v.\ \{\mathsf{True}\}_{\lightning}\ g\ v\ \{y.\ \mathit{isEven}(y)\})\end{array}\right\}$$
$$g\ ()$$
$$\{x.\ \mathit{isEven}(x)\}$$
$$\{x.\ \mathit{isEven}(x)\}$$

$$\{\mathsf{True}\}_{\lightning}$$
$$\mathit{even\_counter}$$
$$\left\{\begin{array}{l}x.\ \exists f, g.\ x = (f, g) \wedge \\ \quad (\forall v.\ \{\mathsf{True}\}_{\lightning}\ f\ v\ \{y.\ y = ()\}) \wedge \\ \quad (\forall v.\ \{\mathsf{True}\}_{\lightning}\ g\ v\ \{y.\ \mathit{isEven}(y)\})\end{array}\right\}$$

Theorem (Fundamental Theorem of Robust Safety (FTRS))

Let e be any expression with no hard-coded locations or assertions.
Furthermore, let $\vec{vs}$ be values which are all good.
The following holds:

$$\mathit{good}_{\mathit{exp}}(e[\vec{vs}/\vec{xs}])$$

where $\vec{xs}$ are variables (as many as $\vec{vs}$).

Robust Safety, an Example (proof)

$$\{\mathsf{True}\}_{\lightning}$$
$$\left\{\begin{array}{l}(\forall v.\ \{\mathsf{True}\}_{\lightning}\ f\ v\ \{y.\ y = ()\}) \wedge \\ (\forall v.\ \{\mathsf{True}\}_{\lightning}\ g\ v\ \{y.\ \mathit{isEven}(y)\})\end{array}\right\}_{\lightning}$$
$$\mathit{adversary}[f, g/\mathit{incr}, \mathit{read}];$$
$$\left\{\begin{array}{l}(\forall v.\ \{\mathsf{True}\}_{\lightning}\ f\ v\ \{y.\ y = ()\}) \wedge \\ (\forall v.\ \{\mathsf{True}\}_{\lightning}\ g\ v\ \{y.\ \mathit{isEven}(y)\})\end{array}\right\}$$
$$g\ ()$$
$$\{x.\ \mathit{isEven}(x)\}$$
$$\{x.\ \mathit{isEven}(x)\}$$

$$\{\mathsf{True}\}_{\lightning}$$
$$\mathit{even\_counter}$$
$$\left\{\begin{array}{l}x.\ \exists f, g.\ x = (f, g) \wedge \\ \quad (\forall v.\ \{\mathsf{True}\}_{\lightning}\ f\ v\ \{y.\ y = ()\}) \wedge \\ \quad (\forall v.\ \{\mathsf{True}\}_{\lightning}\ g\ v\ \{y.\ \mathit{isEven}(y)\})\end{array}\right\}$$

Theorem (Fundamental Theorem of Robust Safety (FTRS))

Let e be any expression with no hard-coded locations or assertions.
Furthermore, let $\vec{vs}$ be values which are all good.
The following holds:

$$\mathit{good}_{exp}(e[\vec{vs}/\vec{xs}])$$

where $\vec{xs}$ are variables (as many as $\vec{vs}$).

Hence, by applying the adequacy theorem for non-progressive Hoare triples, we get

$$\mathsf{NonProg}_{\mathit{isEven}}(\mathsf{let}\ (\mathit{incr}, \mathit{read}) := \mathit{even\_counter}\ \mathsf{in}\ \mathit{adversary};\ \mathit{read}\ ())$$

as required.

Online resources

I hope these lectures have made you interested in learning more about Iris, separation logic, program verification, *etc.*

See http://iris-project.org

— A software-foundations-like Iris Tutorial in Rocq

— Iris related publications

— PhD theses that include Iris works

— Other manuscripts

See https://cs.au.dk/~timany/talks/plsss26/

— All slides

— Links to other resources