Trillium: Unifying Refinement and Higher-Order Distributed Separation Logic

Léo Stefanesco – MPI-SWS

j.w.w. Lars Birkedal, Léon Gondelman, Abel Nieto, Simon Oddershede Gregersen, and Amin Timany

Aarhus University
A more *intensional* adequacy theorem

“Axiomatic” logics such as Iris are justified by their adequacy theorems.

The adequacy for the *standard Hoare triple* essentially guarantees a proved closed program will not crash.

Therefore, *module* specifications are justified by the fact that there are *client program* which are safe.

We want a more *direct* way to argue that a module implements *Paxos*, for example.
Models

We represent the specification as a labeled transition system \((M, \xrightarrow{\ell})\).

Example: Paxos, taken from the official TLA+ examples

States: \((S, B, V)\) where:

- \(S \in \mathcal{P}(\text{PaxosMessage})\) are the sent messages,
- \(B : \text{Acceptor} \rightarrow \text{BallotNr}_\bot\) is the greatest ballot promise;
- \(V : \text{Acceptor} \rightarrow (\text{BallotNr} \times \text{Value})_\bot\) is the last accepted value;
- and \(\text{BallotNr} := \mathbb{N}\).

Transitions: for example

\[
\frac{b \in \text{BallotNr}}{(S, B, V) \rightarrow (S \cup \{\text{msg1a}(b)\}, B, V)}
\]
Finite traces

A model trace of $M$ is a non-empty finite sequence $\mu$ of the form

$$\delta_1 \xrightarrow{\ell_1} \delta_2 \xrightarrow{\ell_2} \cdots \xrightarrow{\ell_{n-1}} \delta_n$$

We write $\mu \downharpoonright\ell \delta$ for snoc, and define first($\mu$) and last($\mu$).

Program traces have configurations $c = (tp, \sigma)$ where $\sigma$ is the state and $tp = [e_1, \ldots, e_n]$ is the thread-pool. The transitions are lifted from a thread-local reduction relation:

$$\left((e_k, \sigma) \rightarrow_h (e'_k, \sigma', \vec{e_f})\right)$$

$$\left(\left[e_1, \ldots, e_k, \ldots, e_n\right], \sigma\right) \xrightarrow{\zeta} \left(\left[e_1, \ldots, e'_k, \ldots, e_n, \vec{e_f}\right], \sigma'\right)$$

where the locale $\zeta$ identifies the thread $e_k$ in a language-specific way. We write $\tau$ such program traces.

Trillium relates model traces and program traces with a simulation.
The parameters of Trillium

A **model** \( \mathcal{M} \);

An **Iris predicate** \( \text{StateInterp}(\mu, \tau) \) which denote the authoritative ownership of the “current” model and program traces \( \mu \) and \( \tau \);

A **Coq predicate** \( \text{ValidEvolution}(\mu, \tau, \zeta, \ell, c', \delta') \) which defines when the joint evolution

\[
\mu \rightsquigarrow \mu \zeta :: c' \quad \tau \rightsquigarrow \tau \ell :: \delta'
\]

is deemed **valid**.
Example: The Aneris instantiation

**Aneris models** are not labeled but have an initial state. This defines a map AnerisModel $\rightarrow$ Model.

The **Aneris state interpretation** is of the form

$$\text{StateInterp}(\mu, \tau) := \text{AnerisStateInterp}(\mu) \ast \text{OwnAuthModel}(\text{last}(\tau)).$$

**Valid transitions** are given by the reflexive closure of the transition relation of the model

$$\text{ValidEvolution}(\mu, \tau, \zeta, \ell, c', \delta') := \text{last}(\tau) = \delta' \lor \text{last}(\tau) \rightarrow \delta'.$$
Trillium’s notion of refinement

Given a relation \( \xi \subseteq \text{ProgTrace} \times \text{ModelTrace} \), we define

\[
\text{Ref}_\xi \subseteq \text{ProgTrace} \times \text{ModelTrace}
\]

as the greatest-fixpoint of:

\[
\text{Ref}_\xi (\mu, \tau) \iff \\
\xi(\mu, \tau) \land \forall c', \zeta. \text{last}(\mu) \xrightarrow{\zeta} c' \Rightarrow \\
(\exists \delta', \ell. \text{ValidEvolution}(\mu, \tau, \zeta, \ell, c', \delta') \land \text{Ref}_\xi (\mu :: c', \tau :: \delta'))
\]
Trillium’s notion of refinement

Given a relation $\xi \subseteq \text{ProgTrace} \times \text{ModelTrace}$, we define

$$\text{Ref}_\xi \subseteq \text{ProgTrace} \times \text{ModelTrace}$$

as the greatest-fixpoint of:

$$\text{Ref}_\xi(\mu, \tau) \iff \xi(\mu, \tau) \land \forall c', \zeta. \text{last}(\mu) \xrightarrow{\zeta} c' \Rightarrow$$

$$(\exists \delta', \ell. \text{ValidEvolution}(\mu, \tau, \zeta, \ell, c', \delta') \land \text{Ref}_\xi(\mu :: c', \tau :: \delta'))$$

When $\text{ValidEvolution}(\mu, \tau, \zeta, \ell, c', \delta') = \text{last}(\tau) \xrightarrow{\zeta} \delta'$ and $\xi(\mu, \tau) \iff \bar{\xi}(\text{last}(\mu), \text{last}(\tau))$ for some $\bar{\xi}$,

$\text{Ref}_\xi$ is the largest simulation included in $\bar{\xi}$. 
The adequacy theorem

Let $e$ be a program, $\sigma$ a program state, and $\Phi$ an Iris predicate on values. Let $\delta$ be a model state and $\xi \subseteq \text{ProgTrace} \times \text{ModelTrace}$.

Suppose the user-defined relation $\text{ValidEvolution}$ is $\xi$-finitary. If

$$\vdash_{\text{Iris}} \Rightarrow_{\top} \text{StateInterp}([[e, \sigma]), [\delta]) \ast \text{wp}_{\top} e@\zeta \{\Phi\} \ast \text{AlwaysHolds}(\xi)$$

then $\text{Ref}_{\xi}([[e, \sigma]), [\delta]) \land \text{safe}(e, \sigma)$ holds in the meta-logic.
The adequacy theorem

Let $e$ be a program, $\sigma$ a program state, and $\Phi$ an Iris predicate on values. Let $\delta$ be a model state and $\xi \subseteq \text{ProgTrace} \times \text{ModelTrace}$.

Suppose the user-defined relation $\text{ValidEvolution}$ is $\xi$-finitary. If

$$\vdash_{\text{Iris}} \models_{\top} \text{StateInterp}([([e], \sigma]), [\delta]) \ast \text{wp}_{\top} e@\zeta \{\Phi\} \ast \text{AlwaysHolds}(\xi)$$

then $\text{Ref}_{\xi}([([e], \sigma]), [\delta]) \land \text{safe}(e, \sigma)$ holds in the meta-logic.

Where $\text{AlwaysHolds}(\xi)$ is the Iris predicate

$$\forall \mu, \tau. \left( \text{StateInterp}(\mu, \tau) \ast \left( \begin{array}{c} \text{first}(\mu) = ([e], \sigma) \ast \text{first}(\tau) = \delta \ast \left( \begin{array}{c} \forall \mu', \tau', c', \delta', \zeta, \ell. \mu = \mu' :: c' \land \tau = \tau' :: \delta' \Rightarrow \\ \text{ValidEvolution}(\mu', \tau', \zeta, \ell, c', \delta') \land \xi(\mu', \tau') \end{array} \right) \right) \ast \end{array} \right) \equiv_{\ast}^{\top} \xi(\mu, \tau)$$
The weakest precondition

The Iris predicate \( \wp_{\mathcal{E}} e \oplus \zeta \{ \Phi \} \) is defined as the guarded fixpoint:

\[
\begin{align*}
\Rightarrow_{\mathcal{E}} \Phi(e) & \quad \text{if } e \in \text{Val} \\
\forall \mu, \tau, K. & \quad \text{otherwise} \\
thread(\zeta, \text{last}(\mu)) = K[e] & \quad \text{ValidExec}(\mu) \\
\text{ValidEvolution}(\mu) & \\
\text{StateInterp}(\mu, \tau) & \equiv_{\mathcal{E}} \emptyset \\
\text{reducible}(e, \text{state}(\text{last}(\mu))) & \\
\forall e', \sigma', \vec{e}_f. & \\
(e, \text{state}(\text{last}(\mu))) \rightarrow (e', \sigma', \vec{e}_f) & \quad \emptyset \equiv_{\mathcal{E}} \emptyset \\
\exists \delta', \ell. & \quad \text{StateInterp}(\mu \gg update(i, K[e'], \sigma', \text{last}(\mu)), \tau \gg \delta') \\
\text{ValidEvolution}(\mu, \tau, \zeta, \ell, update(i, K[e'], \sigma', \text{last}(\mu)), \delta) & \\
\wp_{\mathcal{E}} e' \oplus \zeta \{ \Phi \} & \quad \star \quad \wp_{\top} e_j \oplus \zeta_j \{ \text{forkpost} \} \\
\end{align*}
\]

\(1 \leq j \leq k\)
Finiteness requirements

In proving the adequacy theorem, we need to prove **Coq existentials** from the corresponding **Iris existentials** over \( \delta', \ell \).

The semantics of Iris existentials is basically

\[
(a, i) \in \llbracket \exists x : A, P \rrbracket \iff \exists v_i \in [A], (a, i) \in [P[x := v_i]]
\]

If the \( v_i \) must belong to some **finite set**, by the **pigeon hole principle**,

\[
a \models \exists x : A, P \iff \forall i, (a, i) \in \llbracket \exists x : A, P \rrbracket
\]

\[
\Rightarrow \exists v \in [A], a \models P[x := v]
\]
Finiteness requirements

In proving the adequacy theorem, we need to prove Coq existentials from the corresponding Iris existentials over $\delta', \ell$.

The semantics of Iris existentials is basically

$$(a, i) \in \llbracket \exists x: A, P \rrbracket \iff \exists v_i \in \llbracket A \rrbracket, (a, i) \in \llbracket P[x := v_i] \rrbracket$$

If the $v_i$ must belong to some finite set, by the pigeon hole principle,

$$a \models \exists x: A, P \iff \forall i, (a, i) \in \llbracket \exists x: A, P \rrbracket$$
$$\Rightarrow \exists v \in \llbracket A \rrbracket, a \models P[x := v]$$

ValidEvolution is $\xi$-finitary if the following set if finite:

$$\{(\delta', \ell) \mid \text{ValidEvolution}(\mu, \tau, \zeta, \ell, c', \ell') \land \xi(\mu' \vdash c', \tau \vdash \delta')\}$$

for every $\mu, \tau, \zeta, c'$. 
Example: Paxos

Since Paxos is a distributed algorithm, our aim is to show that the model and the program **send the same messages**:

\[
\xi(\mu, \tau) := \exists S, \text{last}(\mu) = (S, _, _) \land \text{Messages(last}(\tau)) \sim S
\]

Note that \(\xi\)-finiteness is automatic.

We are able to transport the correctness of the model to any execution of the module:

Let \(e\) be a distributed system obtained by composing \(n\) proposers, \(m\) acceptors, and \(k\) learners.

For any thread-pool \(tp\) and state \(\sigma\), if \((e, \emptyset) \rightarrow^* (tp, \sigma)\) and both \(\text{ChosenI} (\text{Messages}(\sigma), \nu_1)\) and \(\text{ChosenI} (\text{Messages}(\sigma), \nu_2)\) hold then

\[
\nu_1 = \nu_2.
\]
Reasoning rules

The **inference rule** to update the **model state** in an instantiation such as Aneris is:

\[
\delta \rightarrow \delta' \quad \text{Atomic(e)} \quad e \notin \text{Val} \\
\text{OwnFragModel}(\delta) \ast \text{wp}_e \ e \{x. \text{OwnFragModel}(\delta') \rightarrow \Phi(x)\} \vdash \text{wp}_e \ e \{x. \Phi(x)\}
\]

If there exists a certain label \(\ell_{\text{stutter}}\) such that

\[
\text{ValidEvolution}(\mu, \tau, \zeta, \ell_{\text{stutter}}, c', \text{last}(\tau))
\]

always holds, then **all the inference rules** of the usual Iris weakest precondition hold in Trillium.
Other examples

**Hanoi towers** and **Incrementing loop**: enforcing memory to go through a succession of values. Uses **events** to detect memory allocations.

**Two-phase commit**: very similar to the Paxos example

**Eventual consistency of CRDTs**: proves that if, eventually, no operations are performed, the state of every nodes converge. Makes assumptions on the network and well as the program.

**Fair termination of concurrent programs**: the rest of this presentation
Fair termination

Termination of every execution is too strong a notion for most concurrent programs.

Running example:

```
let rec yes b n = if cas b 1 0 then n := !n-1;
    if n > 0 then yes b n

let rec no b m = if cas b 0 1 then m := !m-1;
    if m > 0 then no b m

let start k = let b = ref 0 in
    (yes b (ref k) || no b (ref k))
```

This program terminates when the scheduler is fair.
Fairness

A program trace $\mu$ is **fair** if it is finite, or if it is infinite and every reducible thread eventually takes a step.

A program $e$ is **fairly terminating** if all its fair traces are finite.

The goal is to use **Trillium** to prove **fair termination** of programs by constructing a fairness-preserving and termination-preserving refinement with an **abstract model**:
Fairness model

To have a notion of fairness for model traces, the models need a surrogate for the notion of thread: roles.

A fairness model $\mathcal{F}$ is:
1. a set $\mathcal{F}$ of states;
2. a set $R$ of roles;
3. a labeled transition relation $\rightarrow \subseteq \mathcal{F} \times R \times \mathcal{F}$;
4. a finite set $\text{liveroles}(s)$ of live roles for each state $s \in \mathcal{F}$;
5. a fuel bound $\text{fuelmax}(s) \in \mathbb{N}$ for each state;
6. satisfying certain conditions.

A trace of a fairness model is fair if it is finite, or if it is infinite and every role which is live at some point, eventually it takes a step or stops being live.
The \textbf{Live model construction (1)}

The relation $\lesssim$ is defined as the relational composition of:

$$\mu \lesssim_f \lesssim_s \tau \lesssim_s \bar{\tau}$$

where

- $\mu$ is a trace of the program;
- $\bar{\tau}$ is a trace of $\mathcal{F}$;
- $\tau$ is a trace of a \textbf{Trillium model} \text{Live}(\mathcal{F});
- $\lesssim_f$ is induced by $\text{Ref}_{\xi_{\text{fair}}}$;
- $\lesssim_s$ corresponds to removing \textbf{finite stutter}. 


The Live model construction (2)

Given a fairness model $\mathcal{F}$, we define a model $\text{Live}(\mathcal{F})$.

Its states are triples $(s, F, T)$ of

- a state $s \in \mathcal{F}$;
- a map $F : \text{liveroles}(s) \to \mathbb{N}$ associating each live role with its fuel;
- a map $T : \text{liveroles}(s) \to \text{Locale}$ which associates each live role with a locale.

Its labels are $\text{Step}(\rho, \zeta) \mid \text{Stutter}(\zeta)$, where $\zeta \in \text{Locale}$ and $\rho$ is a role.

The idea behind the transitions is that each thread must share its resources fairly between the roles it handles. Each step the thread takes without a step in one of its roles decreases this role’s fuel.
The Live model construction (3)

There are two kinds of transitions:

\[(s, F, T) \xrightarrow{\text{Step}(\rho, \zeta)} (s', F', T')\]

with the (slightly simplified) conditions:

1. \(s \xrightarrow{\rho} s'\) in \(\mathcal{F}\);
2. \(T(\rho) = \zeta\);
3. \(F'(\rho) \leq \text{fuelmax}(s')\);
4. \(\forall \rho' \in R^{-1}(\zeta) \setminus \{\rho\}, \; F'(\rho') < F(\rho')\).

\[(s, F, T) \xrightarrow{\text{Stutter}(\zeta)} (s, F', T')\]

when \(\forall \rho' \in R^{-1}(\zeta), \; F'(\rho') < F(\rho')\).
Trillium instantiation

ValidEvolution(µ, τ, ζ, ℓ, c', δ') := localeof(ℓ) = ζ ∧ last(τ) → δ'.

The three components of the states of Live(ℱ) have corresponding Iris resources:

- Modells(s) for the underlying state of ℱ;
- FuelsAre(ζ, fs) where fs is a finite partial map from roles to fuels.

**Theorem**  Given a program e, a finitely branching fairness model ℱ, a state δ₀ ∈ ℱ, if

Modells(s₀) → FuelsAre(ζ₀, fs₀) →wp T e @ δ₀ {FuelsAre(ζ₀, ¯∅)}

holds in Iris for any fs₀, and if ℱ is fairly terminating, then e is fairly terminating.
Back to the example

We relate it with the following fairness model:

\[
\begin{array}{c}
\cdots \quad \text{No} \quad \rightarrow \quad m, 1 \quad \text{Yes} \quad \rightarrow \quad m, 0 \quad \text{No} \quad \rightarrow \quad m-1, 1 \quad \text{Yes} \quad \rightarrow \quad \cdots
\end{array}
\]

It is fairly terminating because there is a well-founded order such that, for each state, there exists a role which makes this order decrease.

Therefore, the program start is fairly terminating.
Future work

**Compositionality of the models**: lots of techniques we could adopt.

**Liveness of distributed systems**: adapt the logic for fair termination of concurrent programs to prove liveness properties of distributed systems.
Future work

**Compositionality of the models**: lots of techniques we could adopt.

**Liveness of distributed systems**: adapt the logic for fair termination of concurrent programs to prove liveness properties of distributed systems.

Thank you!
Questions?