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** $(\mathcal{M}, \xrightarrow{\ell})$.

Example: **Paxos**, taken from the official TLA+ examples

States: $(\mathcal{S}, \mathcal{B}, \mathcal{V})$ where:

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

Transitions: for example

$$\frac{b \in \text{BallotNr}}{(\mathcal{S}, \mathcal{B}, \mathcal{V}) \to (\mathcal{S} \cup \{\texttt{msg1a}(b)\}, \mathcal{B}, \mathcal{V})}$$

Finite traces

A **model trace** of $\mathcal{M}$ is a *non-empty* finite sequence μ of the form

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

We write $\mu \overset{\ell}{::} \delta$ for snoc, and define $\mathrm{first}(\mu)$ and $\mathrm{last}(\mu)$.

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

$$\frac{(e_k, \sigma) \rightarrow_h (e_k', \sigma', \vec{e_f})}{([e_1, \ldots, e_k, \ldots, e_n], \sigma) \xrightarrow{\zeta} ([e_1, \ldots, e_k', \ldots, e_n, \vec{e_f}], \sigma')}$$

where the **locale** ζ identifies the thread e_k in a language-specific way. We write τ such program traces.

Trillium relates **model traces** and **program traces** with a **simulation**.

The parameters of Trillium

A **model** $\mathcal{M}$;

An **Iris predicate** $\mathsf{StateInterp}(\mu, \tau)$ which denote the authoritative ownership of the "current" model and program traces μ and τ;

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

$$\mu \rightsquigarrow \mu \overset{\zeta}{::} c' \qquad\qquad \tau \rightsquigarrow \tau \overset{\ell}{::} \delta'$$

is deemed **valid**.

Example: The Aneris instantiation

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

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

$$\mathsf{StateInterp}(\mu, \tau) \coloneqq \mathsf{AnerisStateInterp}(\mu) * \mathsf{OwnAuthModel}(\mathsf{last}(\tau)).$$

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

$$\mathsf{ValidEvolution}(\mu, \tau, \zeta, \ell, \mathsf{c}', \delta') \ \coloneqq \ \mathsf{last}(\tau) = \delta' \vee \mathsf{last}(\tau) \to \delta'.$$

Trillium's notion of refinement

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

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

as the greatest-fixpoint of:

$$\begin{aligned}
&\mathsf{Ref}_\xi(\mu,\tau) \Leftrightarrow \\
&\qquad \xi(\mu,\tau) \;\; \wedge \;\; \forall c', \zeta.\; \mathsf{last}(\mu) \xrightarrow{\zeta} c' \;\Rightarrow \\
&\qquad\; (\exists \delta', \ell.\ \mathsf{ValidEvolution}(\mu,\tau,\zeta,\ell,c',\delta') \wedge \mathsf{Ref}_\xi(\mu \overset{\zeta}{::} c', \tau \overset{\ell}{::} \delta'))
\end{aligned}$$

Trillium's notion of refinement

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

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

as the greatest-fixpoint of:

$$\begin{aligned}
&\mathsf{Ref}_\xi(\mu,\tau) \Leftrightarrow \\
&\qquad \xi(\mu,\tau) \ \wedge \ \forall c',\zeta.\ \mathsf{last}(\mu) \xrightarrow{\zeta} c' \ \Rightarrow \\
&\qquad (\exists \delta',\ell.\ \mathsf{ValidEvolution}(\mu,\tau,\zeta,\ell,c',\delta') \wedge \mathsf{Ref}_\xi(\mu \overset{\zeta}{::} c', \tau \overset{\ell}{::} \delta'))
\end{aligned}$$

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

Ref_ξ is the **largest simulation included in** $\bar{\xi}$.

The adequacy theorem

Let e be a **program**, σ a **program state**, and Φ an **Iris predicate** on values. Let δ be a **model state** and $\xi \subseteq \mathsf{ProgTrace} \times \mathsf{ModelTrace}$.

Suppose the user-defined relation ValidEvolution is **ξ-finitary**. If

$$\vdash_{\mathit{Iris}} \; \Rrightarrow_{\top} \mathsf{StateInterp}([([e], \sigma)], [\delta]) * \mathsf{wp}_{\top}\, e @ \zeta \left\{\Phi\right\} * \mathsf{AlwaysHolds}(\xi)$$

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

The adequacy theorem

Let e be a **program**, σ a **program state**, and Φ an **Iris predicate** on values. Let δ be a **model state** and $\xi \subseteq \mathsf{ProgTrace} \times \mathsf{ModelTrace}$.

Suppose the user-defined relation ValidEvolution is **ξ-finitary**. If

$$\vdash_{\mathit{Iris}} \;\; \mid\!\Rrightarrow_{\top} \mathsf{StateInterp}([([e], \sigma)], [\delta]) * \mathsf{wp}_{\top}\, e @ \zeta \left\{ \Phi \right\} * \mathsf{AlwaysHolds}(\xi)$$

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

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

$$\forall \mu, \tau. \; \left(\begin{array}{l} \mathsf{first}(\mu) = ([e], \sigma) * \mathsf{first}(\tau) = \delta \, * \\ \left(\begin{array}{l} \forall \mu', \tau', c', \delta', \zeta, \ell.\; \mu = \mu' \overset{\zeta}{::} c' \land \tau = \tau' \overset{\ell}{::} \delta' \Rightarrow \\ \mathsf{ValidEvolution}(\mu', \tau', \zeta, \ell, c', \delta') \land \xi(\mu', \tau') \end{array} \right) * \\ \mathsf{StateInterp}(\mu, \tau) \, * \\ \left(\begin{array}{l} \forall e_1, \ldots, e_n, \sigma.\; \mathsf{last}(\mu) = (e_1, \ldots, e_n; \sigma) \Rightarrow \\ (\forall 1 \leq i \leq n.\; e_i \in \mathsf{Val} \lor \mathsf{reducible}(e_i, \sigma)) \land \\ (e_1 \in \mathsf{Val} \Rightarrow \Phi(e_1)) \end{array} \right) \end{array} \right) \; {}^{\top}{\equiv\!\!*}^{\emptyset} \;\; \xi(\mu, \tau)$$

The weakest precondition

The Iris predicate $\mathrm{wp}_{\mathcal{E}}\ e @ \zeta \ \{\Phi\}$ is defined as the **guarded fixpoint**:

$$
\begin{cases}
\Rrightarrow_{\mathcal{E}} \Phi(e) & \textit{if } e \in \mathsf{Val} \\
\begin{array}{l}
\forall \mu, \tau, K. \\
\quad \mathit{thread}(\zeta, \mathsf{last}(\mu)) = K[e] \twoheadrightarrow \\
\quad \mathit{ValidExec}(\mu) \twoheadrightarrow \\
\quad \mathsf{StateInterp}(\mu, \tau) \ {}^{\mathcal{E}}\!\!\Rrightarrow^{\emptyset} \\
\quad \mathsf{reducible}(e, \mathit{state}(\mathsf{last}(\mu))) * \\
\quad \forall e', \sigma', \vec{e_f}. \\
\qquad (e, \mathit{state}(\mathsf{last}(\mu))) \to (e', \sigma', \vec{e_f}) \twoheadrightarrow \triangleright \ {}^{\emptyset}\!\!\Rrightarrow^{\mathcal{E}} \\
\qquad \exists \delta', \ell.\ \mathsf{StateInterp}(\mu \overset{\zeta}{::} \mathit{update}(i, K[e'], \sigma', \mathsf{last}(\mu)), \tau \overset{\ell}{::} \delta')) * \\
\qquad\quad \mathsf{ValidEvolution}(\mu, \tau, \zeta, \ell, \mathit{update}(i, K[e'], \sigma', \mathsf{last}(\mu)), \delta)) * \\
\qquad\quad \mathrm{wp}_{\mathcal{E}}\ e' @ \zeta \ \{\Phi\} * \mathop{\Large\ast}_{1 \le j \le k} \mathrm{wp}_{\top}\ e_{f_j} @ \zeta_j \ \{\mathsf{forkpost}\}
\end{array} & \textit{otherwise}
\end{cases}
$$

Finiteness requirements

In proving the adequacy theorem, we need to prove **Coq existentials** from the corresponding **Iris existentials** over δ', ℓ.

The semantics of Iris existentials is basically

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

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

$$\begin{aligned} a \vDash \exists x : A, P &\Leftrightarrow \forall i, (a, i) \in [\![\exists x : A, P]\!] \\ &\Rightarrow \exists v \in [\![A]\!], a \vDash P[x := v] \end{aligned}$$

Finiteness requirements

In proving the adequacy theorem, we need to prove **Coq existentials** from the corresponding **Iris existentials** over δ', ℓ.

The semantics of Iris existentials is basically

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

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

$$\begin{aligned} a \vDash \exists x : A, P \Leftrightarrow \forall i, (a, i) \in [\![\exists x : A, P]\!] \\ \Rightarrow \exists v \in [\![A]\!], a \vDash P[x := v] \end{aligned}$$

ValidEvolution is ξ-**finitary** if the following set if finite:

$$\{(\delta', \ell) \mid \text{ValidEvolution}(\mu, \tau, \zeta, \ell, c', \ell) \wedge \xi(\mu \overset{\zeta}{::} c', \tau \overset{\ell}{::} \delta')\}$$

for every μ, τ, ζ, c'.

Example: Paxos

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

$$\xi(\mu, \tau) := \exists \mathcal{S},\ \mathsf{last}(\mu) = (\mathcal{S}, \_, \_) \wedge \mathsf{Messages}(\mathsf{last}(\tau)) \sim \mathcal{S}$$

Note that ξ-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 σ, if $(e, \emptyset) \rightarrow^* (\mathsf{tp}, \sigma)$ and both $\mathsf{ChosenI}(\mathsf{Messages}(\sigma), v_1)$ and $\mathsf{ChosenI}(\mathsf{Messages}(\sigma), v_2)$ hold then

$$v_1 = v_2.$$

Reasoning rules

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

$$\frac{\delta \to \delta' \quad \mathsf{Atomic}(e) \quad e \notin \mathsf{Val}}{\mathsf{OwnFragModel}(\delta) * \mathsf{wp}_{\mathcal{E}}\ e\left\{x.\ \mathsf{OwnFragModel}(\delta') \mathrel{-\!\!*} \Phi(x)\right\} \vdash \mathsf{wp}_{\mathcal{E}}\ e\left\{x.\ \Phi(x)\right\}}$$

If there exists a certain label $\ell_{stutter}$ such that

$$\mathsf{ValidEvolution}(\mu, \tau, \zeta, \ell_{stutter}, c', \mathsf{last}(\tau))$$

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 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 μ 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**:

$$\begin{tikzcd} \mu \arrow[r, bend left, "\text{fairness}"] \arrow[r, phantom, "\precsim"] & \bar{\tau} \arrow[l, bend left, "\text{termination}"] \end{tikzcd}$$

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 liveroles(s) of **live** roles for each state $s \in \mathcal{F}$;
5. a **fuel bound** 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 Live model construction (1)

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

$$\begin{tikzcd} \mu \arrow[r, bend left, "\text{fair}"] \arrow[r, phantom, "\precsim_f"] & \tau \arrow[l, bend left, "\text{finite}"] \arrow[r, bend left, "\text{fair}"] \arrow[r, phantom, "\precsim_s"] & \bar{\tau} \arrow[l, bend left, "\text{finite}"] \end{tikzcd}$$

where

- μ is a trace of the program;
- $\bar{\tau}$ is a trace of $\mathcal{F}$;
- τ is a trace of a **Trillium model** Live($\mathcal{F}$);
- $\precsim_f$ is induced by $\mathrm{Ref}_{\xi_{fair}}$;
- $\precsim_s$ corresponds to removing **finite stutter**.

The Live model construction (2)

Given a **fairness model** $\mathcal{F}$, we define a model 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 ρ 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 thead 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**:

$$(s, F, T) \xrightarrow{\mathsf{Step}(\rho,\zeta)} (s', F', T')$$

with the (slightly simplified) conditions:

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

$$(s, F, T) \xrightarrow{\mathsf{Stutter}(\zeta)} (s, F', T')$$

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

Trillium instantiation

$$\mathsf{ValidEvolution}(\mu, \tau, \zeta, \ell, \mathsf{c}', \delta') := \mathsf{localeof}(\ell) = \zeta \ \wedge\ \mathsf{last}(\tau) \xrightarrow{\ell} \delta'.$$

The three components of the states of $\mathsf{Live}(\mathcal{F})$ have corresponding **Iris resources**:

- $\mathsf{ModelIs}(s)$ for the underlying state of $\mathcal{F}$;
- $\mathsf{FuelsAre}(\zeta, \mathit{fs})$ where fs is a finite partial map from **roles** to **fuels**.

Theorem Given a program e, a finitely branching fairness model $\mathcal{F}$, a state $\delta_0 \in \mathcal{F}$, if

$$\mathsf{ModelIs}(s_0) \ -\!\!*\ \mathsf{FuelsAre}(\zeta_0, \mathit{fs}_0) \ -\!\!*\ {}^{\top}\!\Rrightarrow^{\top} \mathsf{wp}_{\top}\ e \mathbin{@} \zeta_0 \left\{\mathsf{FuelsAre}(\zeta_0, \emptyset)\right\}$$

holds in Iris for any fs_0, and if $\mathcal{F}$ is fairly terminating, then e is fairly terminating.

Back to the example

We relate it with the following **fairness model**:

$$\begin{tikzcd} \cdots \arrow[r, "\text{No}"] & m,1 \arrow[loop above, "\text{No}"] \arrow[r, "\text{Yes}"] & m,0 \arrow[loop above, "\text{Yes}"] \arrow[r, "\text{No}"] & m-1,1 \arrow[loop above, "\text{No}"] \arrow[r, "\text{Yes}"] & \cdots \end{tikzcd}$$

It is **fairly terminating** because we 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?