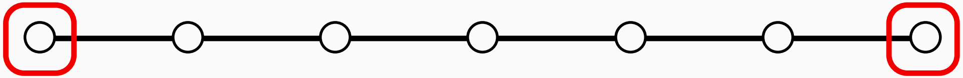
Context-aware Trace Contracts

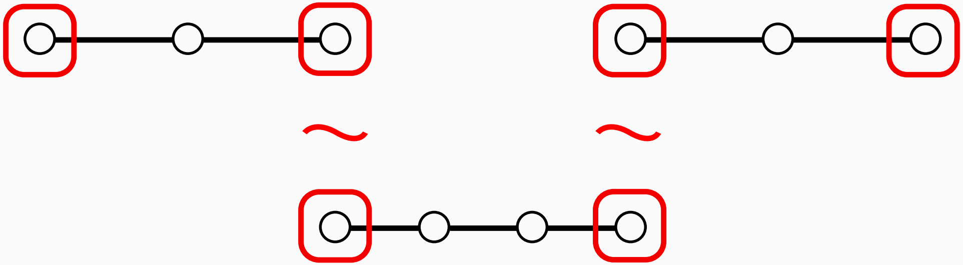
Eduard Kamburjan[1]
Reiner Hähnle[2]
Marco Scaletta[2]

[1]University of Oslo
[2]TU Darmstadt
ABS Workshop, 04.10.23

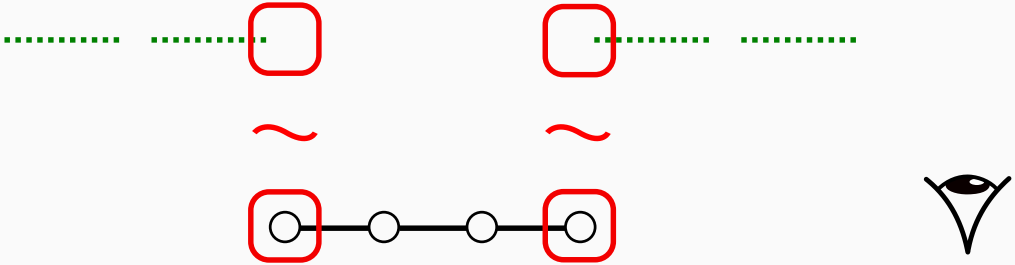
Context beyond States, Calls beyond Synchronicity



Context beyond States, Calls beyond Synchronicity

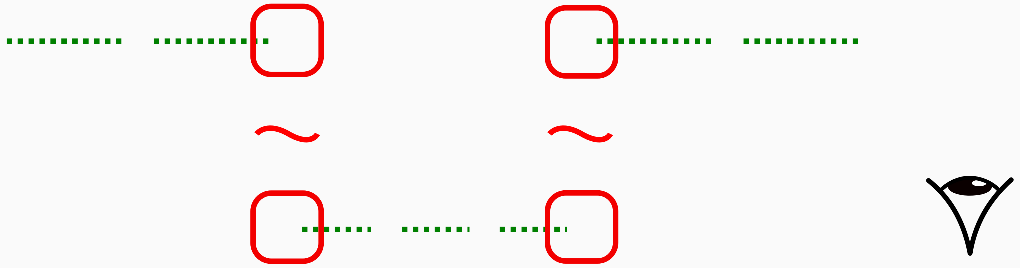


Context beyond States, Calls beyond Synchronicity



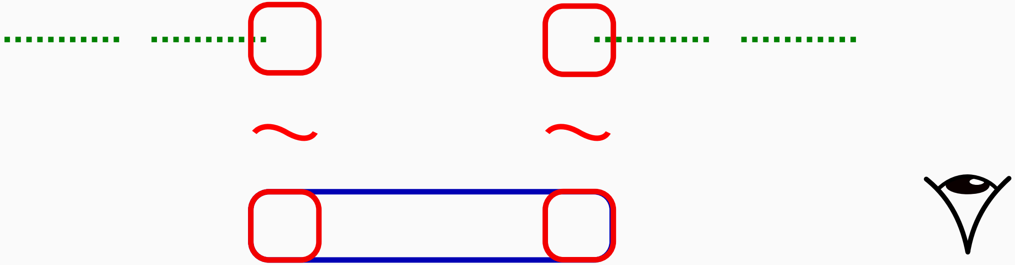
- Contracts abstract the call context

Context beyond States, Calls beyond Synchronicity



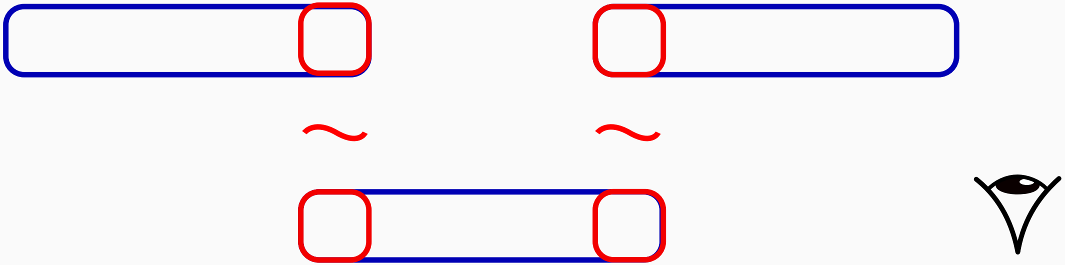
- Contracts abstract the call context
- All context encoded in *state* predicates

Context beyond States, Calls beyond Synchronicity



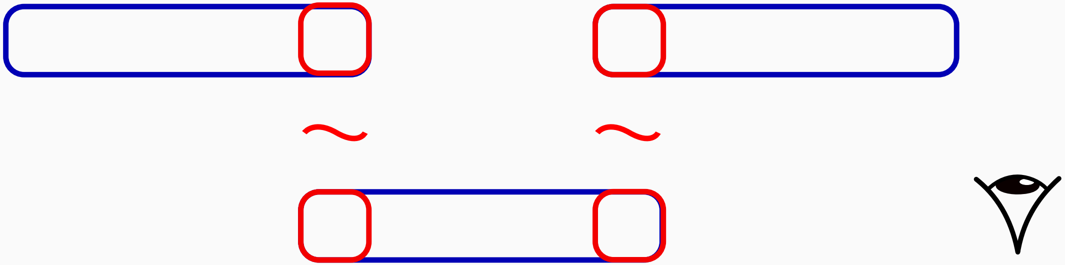
- Contracts abstract the call context
- All context encoded in *state* predicates

Context beyond States, Calls beyond Synchronicity



- Contracts abstract the call context
- All context encoded in *state* predicates

Context beyond States, Calls beyond Synchronicity



- Contracts abstract the call context
- All context encoded in *state* predicates

- Removing the need for ghost histories in states
- Enabling simpler asynchronous method contracts

Specification

Synchronous Language

Sync

- Imperative language with procedures (`m(){s; return}`)
- Synchronous calls (`m();`), file operations (`open(f);`, `write(f);`, `close(f);`)
- All variables global, no parameters, no return values

Example:

```
do() { open(f); operate(); closeF(); return; }
operate() { write(f); return; }
closeF() { close(f); return; }
```

Traces and Trace Logic

Traces

A trace is a sequence of states σ and events

$$\mathsf{invoc}(\mathtt{m}, i), \mathsf{start}(\mathtt{m}, i), \mathsf{ret}(i), \mathsf{write}(e), \ldots$$

Trace Logic

Let ϕ be a state formula. A trace formula θ has traces as models and is defined by

$$\theta \ ::= \ \theta \wedge \theta \mid \lceil \phi \rceil \mid \mathsf{ev}(\overline{e}) \mid \theta \ast\ast \theta \mid \theta \cdot \theta \mid \ \ldots$$

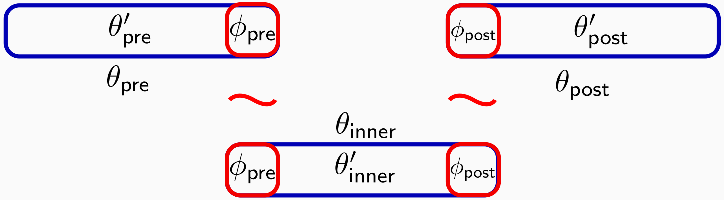
- Important shortcut: $\overset{\overline{\mathsf{ev}}}{\cdot\cdot}$ is any trace that does not contain any event from $\overline{\mathsf{ev}}$
- Special case: $\cdot\cdot$ is any trace
- Simplification for talk: no variables, only constants (=read-only variables)

Contracts

Trace Contract

A trace contract C_m for procedure m is a triple $\langle\theta_\mathsf{pre}|\theta_\mathsf{inner}|\theta_\mathsf{post}\rangle$ with

$$\theta_\mathsf{pre} = \theta'_\mathsf{pre} \cdot \lceil\phi_\mathsf{pre}\rceil \qquad \theta_\mathsf{inner} = \lceil\phi_\mathsf{pre}\rceil \cdot \theta'_\mathsf{inner} \cdot \lceil\phi_\mathsf{post}\rceil \qquad \theta_\mathsf{post} = \lceil\phi_\mathsf{post}\rceil \cdot \theta'_\mathsf{post}$$



Contracts (Ex.)

The contract for `operate` is

$$C_{\texttt{operate}} = \left\langle \cdot\cdot\ \mathsf{open(f)}\ \overset{\mathsf{close(f)}}{\cdot\cdot}\ \lceil\mathbf{true}\rceil \middle| \lceil\mathbf{true}\rceil\ \overset{\mathsf{close(f),open(f)}}{\cdot\cdot}\ \lceil\mathbf{true}\rceil \middle| \lceil\mathbf{true}\rceil\ \cdot\cdot\ \mathsf{close(f)}\ \cdot\cdot \right\rangle$$

Contracts (Ex.)

The contract for `operate` is

$$C_{\texttt{operate}} = \left\langle \cdot\cdot \; \mathsf{open(f)} \; \overset{\mathsf{close(f)}}{\cdot\cdot} \middle| \overset{\mathsf{close(f),open(f)}}{\cdot\cdot} \middle| \cdot\cdot \; \mathsf{close(f)} \; \cdot\cdot \right\rangle$$

Contracts (Ex.)

The contract for `operate` is

$$C_{\texttt{operate}} = \left\langle \cdot\cdot\ \mathsf{open(f)}\ \overset{\mathsf{close(f)}}{\cdot\cdot} \middle| \overset{\mathsf{close(f),open(f)}}{\cdot\cdot} \middle| \cdot\cdot\ \mathsf{close(f)}\ \cdot\cdot \right\rangle$$

The contract for `closeF` is

$$C_{\texttt{closeF}} = \left\langle \cdot\cdot\ \mathsf{open(f)}\ \overset{\mathsf{close(f)}}{\cdot\cdot} \middle| \cdot\cdot\ \mathsf{close(f)}\ \cdot\cdot \middle| \cdot\cdot \right\rangle$$

- No extra state "*isOpen(f)*"
- No FO history "$\forall i < |\mathit{history}|.\ \mathit{history}[i] \neq \mathsf{open(f)}$"
- What to do with θ_{post}?

Verification

Verification

- The semantics of programs ($[\![s]\!]_\tau$), trace updates ($[\![\mathcal{U}]\!]_\sigma$) and formulas ($[\![\Phi]\!]$) are sets of traces (prefixed with τ or σ)
- Symbolic execution idea: reduce program to trace update, have a special solver for relating trace updates and trace formulas

Judgments

- $\{\mathcal{U}\} : \Phi$ – All traces described by $\mathcal{U}$ are described by Φ

$$\sigma \models \{\mathcal{U}\} : \Phi \iff [\![\mathcal{U}]\!]_\sigma \subseteq [\![\Phi]\!]$$

- $\{\mathcal{U}\}s : \Phi$ – All traces described by first $\mathcal{U}$ and then s are described by Φ

$$\sigma \models \{\mathcal{U}\}s : \Phi \iff \bigcup_{\tau \in [\![\mathcal{U}]\!]_\sigma} [\![\mathcal{U}]\!]_\sigma ** [\![s]\!]_\tau \subseteq [\![\Phi]\!]$$

Sequent Calculus

$$\textbf{(Assign)}\ \frac{\Gamma \vdash \{\mathcal{U}\}\{\texttt{v} \ := \ \texttt{e}\}\ \texttt{s} : \Phi}{\Gamma \vdash \{\mathcal{U}\}\texttt{v = e; s} : \Phi}$$

$$\textbf{(If)}\ \frac{\begin{array}{c}\Gamma, \{\mathcal{U}\} \cdot\cdot \lceil \texttt{e} \rceil \vdash \{\mathcal{U}\}\texttt{s1; s2} : \Phi \\ \Gamma, \{\mathcal{U}\} \cdot\cdot \lceil \texttt{!e} \rceil \vdash \{\mathcal{U}\}\texttt{s2} : \Phi\end{array}}{\Gamma \vdash \{\mathcal{U}\}\texttt{if(e)s1; s2} : \Phi}$$

$$\textbf{(Write)}\ \frac{\begin{array}{c}\Gamma \vdash \{\mathcal{U}\} \cdot\cdot \mathtt{open(f)} \overset{\texttt{close(f)}}{\cdot\cdot} \\ \Gamma \vdash \{\mathcal{U}\}\{\mathtt{write(f)}\}\ \texttt{s} : \Phi\end{array}}{\Gamma \vdash \{\mathcal{U}\}\texttt{write(f); s} : \Phi}$$

Synchronous Contract Rule

$$\textbf{(Call)}\ \frac{\vdash \qquad\qquad\qquad\qquad :}{\Gamma \vdash \{\mathcal{U}\}\mathtt{m}();\ \mathtt{s} : \Phi ** \theta ** \Psi}$$

- Split specification into pre-trace, inner trace and post-trace

Synchronous Contract Rule

$$\textbf{(Call)}\ \frac{\begin{array}{c} \Gamma \vdash \{\mathcal{U}\} : (\Phi \wedge \theta^{\mathtt{m}}_{\mathsf{pre}}) \\ \Gamma, \{\mathcal{U}\} \qquad\qquad : (\Phi \wedge \theta^{\mathtt{m}}_{\mathsf{pre}}) \\ \vdash \{\mathcal{U}\} \qquad\qquad : (\Phi \wedge \theta^{\mathtt{m}}_{\mathsf{pre}}) \end{array}}{\Gamma \vdash \{\mathcal{U}\}\mathtt{m}();\ \mathtt{s} : \Phi \ast\ast \theta \ast\ast \Psi}$$

- Split specification into pre-trace, inner trace and post-trace
- Standard pre-condition

Synchronous Contract Rule

$$\textbf{(Call)}\ \frac{[\![\theta^{\mathtt{m}}_{\mathsf{inner}}]\!] \subseteq [\![\theta]\!] \qquad \Gamma \vdash \{\mathcal{U}\} : (\Phi \wedge \theta^{\mathtt{m}}_{\mathsf{pre}}) \qquad \begin{array}{l}\Gamma, \{\mathcal{U}\}\{\mathsf{run}(\mathsf{m}, i)\} : (\Phi \wedge \theta^{\mathtt{m}}_{\mathsf{pre}}) ** \theta^{\mathtt{m}}_{\mathsf{inner}} \\ \quad \vdash \{\mathcal{U}\}\{\mathsf{run}(\mathsf{m}, i)\} \quad : (\Phi \wedge \theta^{\mathtt{m}}_{\mathsf{pre}}) ** \theta^{\mathtt{m}}_{\mathsf{inner}}\end{array}}{\Gamma \vdash \{\mathcal{U}\}\mathtt{m}();\ \mathtt{s} : \Phi ** \theta ** \Psi}$$

- Split specification into pre-trace, inner trace and post-trace
- Standard pre-condition
- Abstract inner trace with its contract

Synchronous Contract Rule

$$\textbf{(Call)}\ \frac{[\![\theta^{\mathtt{m}}_{\mathsf{inner}}]\!] \subseteq [\![\theta]\!] \qquad \Gamma \vdash \{\mathcal{U}\} : (\Phi \land \theta^{\mathtt{m}}_{\mathsf{pre}}) \qquad \begin{array}{l}\Gamma, \{\mathcal{U}\}\{\mathsf{run}(\mathsf{m}, i)\} : (\Phi \land \theta^{\mathtt{m}}_{\mathsf{pre}}) ** \theta^{\mathtt{m}}_{\mathsf{inner}} \\ \qquad \vdash \{\mathcal{U}\}\{\mathsf{run}(\mathsf{m}, i)\}\ \mathtt{s} : (\Phi \land \theta^{\mathtt{m}}_{\mathsf{pre}}) ** \theta^{\mathtt{m}}_{\mathsf{inner}} ** \Psi \land \theta^{\mathtt{m}}_{\mathsf{post}}\end{array}}{\Gamma \vdash \{\mathcal{U}\}\mathtt{m}();\ \mathtt{s} : \Phi ** \theta ** \Psi}$$

- Split specification into pre-trace, inner trace and post-trace
- Standard pre-condition
- Abstract inner trace with its contract
- Additional post-condition

Proof Obligations

Given a contract of procedure `m`

$$\Big\langle \qquad \Big| \qquad \Big| \qquad \Big\rangle$$

For each procedure we need to prove the following (slightly simplified)

$$\vdash \qquad :$$

Proof Obligations

Given a contract of procedure $\mathtt{m}$

$$\left\langle \qquad \left| \lceil \phi^{\mathtt{m}}_{\mathsf{pre}} \rceil \cdot \theta'^{\mathtt{m}}_{\mathsf{inner}} \cdot \lceil \phi^{\mathtt{m}}_{\mathsf{post}} \rceil \right| \qquad \right\rangle$$

For each procedure we need to prove the following (slightly simplified)

$$\vdash \qquad \mathtt{s}_{\mathtt{m}} : \qquad \theta'^{\mathtt{m}}_{\mathsf{inner}} \cdot \lceil \phi^{\mathtt{m}}_{\mathsf{post}} \rceil$$

Proof Obligations

Given a contract of procedure `m`

$$\left\langle \theta'^{\mathtt{m}}_{\mathsf{pre}} \cdot \lceil \phi^{\mathtt{m}}_{\mathsf{pre}} \rceil \middle| \lceil \phi^{\mathtt{m}}_{\mathsf{pre}} \rceil \cdot \theta'^{\mathtt{m}}_{\mathsf{inner}} \cdot \lceil \phi^{\mathtt{m}}_{\mathsf{post}} \rceil \middle| \qquad\qquad \right\rangle$$

For each procedure we need to prove the following (slightly simplified)

$$\mathcal{U}\{\mathsf{start}(\mathtt{m}, i)\} : \theta'^{\mathtt{m}}_{\mathsf{pre}} \cdot \lceil \phi^{\mathtt{m}}_{\mathsf{pre}} \rceil \vdash \mathcal{U}\{\mathsf{start}(\mathtt{m}, i)\} \mathtt{s}_{\mathtt{m}} : \theta'^{\mathtt{m}}_{\mathsf{pre}} \cdot \lceil \phi^{\mathtt{m}}_{\mathsf{pre}} \rceil \cdot \theta'^{\mathtt{m}}_{\mathsf{inner}} \cdot \lceil \phi^{\mathtt{m}}_{\mathsf{post}} \rceil$$

Proof Obligations

Given a contract of procedure `m`

$$\left\langle \theta'^{\mathtt{m}}_{\text{pre}} \cdot \lceil \phi^{\mathtt{m}}_{\text{pre}} \rceil \middle| \lceil \phi^{\mathtt{m}}_{\text{pre}} \rceil \cdot \theta'^{\mathtt{m}}_{\text{inner}} \cdot \lceil \phi^{\mathtt{m}}_{\text{post}} \rceil \middle| \lceil \phi^{\mathtt{m}}_{\text{post}} \rceil \cdot \theta'^{\mathtt{m}}_{\text{post}} \right\rangle$$

For each procedure we need to prove the following (slightly simplified)

$$\mathcal{U}\{\text{start}(\mathtt{m}, i)\} : \theta'^{\mathtt{m}}_{\text{pre}} \cdot \lceil \phi^{\mathtt{m}}_{\text{pre}} \rceil \vdash \mathcal{U}\{\text{start}(\mathtt{m}, i)\} \mathtt{s}_{\mathtt{m}} : \theta'^{\mathtt{m}}_{\text{pre}} \cdot \lceil \phi^{\mathtt{m}}_{\text{pre}} \rceil \cdot \theta'^{\mathtt{m}}_{\text{inner}} \cdot \lceil \phi^{\mathtt{m}}_{\text{post}} \rceil$$

- **The post-trace $\theta'^{\mathtt{m}}_{\text{post}}$ is not part of the proof obligation**
- Two-layered soundness: If all proof obligations can be closed, then all procedures fulfill their contract

Example

$\mathcal{U}\{\mathsf{start}(\mathtt{do}, i)\} \Rightarrow \vdash \mathcal{U}\{\mathsf{start}(\mathtt{do}, i)\}\mathtt{open(f);\ operate();\ s} \Rightarrow$

Example

$$\frac{\mathcal{U}\{\mathsf{start}(\mathtt{do}, i)\} \;{:}\!\cdot\!\cdot\!\vdash \mathcal{U}\{\mathsf{start}(\mathtt{do}, i)\}\{\mathsf{open}(\mathsf{f})\}\texttt{operate(); s} \;{:}\!\cdot\!\cdot\; \ast\!\!\ast \;\cdot\cdot\; \ast\!\!\ast \;\cdot\cdot}{\mathcal{U}\{\mathsf{start}(\mathtt{do}, i)\} \;{:}\!\cdot\!\cdot\!\vdash \mathcal{U}\{\mathsf{start}(\mathtt{do}, i)\}\texttt{\textbf{open}(f); operate(); s} \;{:}\!\cdot\!\cdot}$$

Example

$$\dfrac{\dfrac{(\mathit{pre}) \qquad (\mathit{inner}) \qquad (\mathit{post})}{\mathcal{U}\{\mathsf{start}(\mathtt{do}, i)\} \mathrel{:\!\cdot\!\cdot\!\vdash} \mathcal{U}\{\mathsf{start}(\mathtt{do}, i)\}\{\mathsf{open}(\mathsf{f})\}\mathtt{operate();\ s} \mathrel{:\!\cdot\!\cdot} \ast\!\!\ast \cdot\cdot \ast\!\!\ast \cdot\cdot}}{\mathcal{U}\{\mathsf{start}(\mathtt{do}, i)\} \mathrel{:\!\cdot\!\cdot\!\vdash} \mathcal{U}\{\mathsf{start}(\mathtt{do}, i)\}\mathtt{\textbf{open}(f);\ operate();\ s} \mathrel{:\!\cdot\!\cdot}}$$

Example

$$\dfrac{\mathcal{U}\{\mathsf{start}(\mathtt{do}, i)\} :\cdot\cdot \vdash \mathcal{U}\{\mathsf{start}(\mathtt{do}, i)\}\{\mathsf{open}(\mathsf{f})\} :\cdot\cdot\ \mathsf{open}(\mathsf{f}) \overset{\mathsf{close}(\mathsf{f})}{\cdot\cdot} \wedge \cdot\cdot}{(\mathit{pre})}$$

$$\dfrac{\dfrac{(\mathit{pre}) \qquad (\mathit{inner}) \qquad (\mathit{post})}{\mathcal{U}\{\mathsf{start}(\mathtt{do}, i)\} :\cdot\cdot \vdash \mathcal{U}\{\mathsf{start}(\mathtt{do}, i)\}\{\mathsf{open}(\mathsf{f})\}\mathtt{operate();\ s} :\cdot\cdot \circledast \cdot\cdot \circledast \cdot\cdot}}{\mathcal{U}\{\mathsf{start}(\mathtt{do}, i)\} :\cdot\cdot \vdash \mathcal{U}\{\mathsf{start}(\mathtt{do}, i)\}\mathtt{open(f);\ operate();\ s} :\cdot\cdot}$$

Example

$$\frac{\mathcal{U}\{\mathsf{start}(\texttt{do}, i)\} \mathrel{:\cdot\cdot}\vdash \mathcal{U}\{\mathsf{start}(\texttt{do}, i)\}\{\mathsf{open}(\mathsf{f})\} \mathrel{:\cdot\cdot} \mathsf{open}(\mathsf{f})\ \overset{\mathsf{close(f)}}{\cdot\cdot} \wedge \cdot\cdot}{(\mathit{pre})}$$

$$\frac{[\![\overset{\mathsf{close(f),open(f)}}{\cdot\cdot}]\!] \subseteq [\![\cdot\cdot]\!]}{(\mathit{inner})}$$

$$\cfrac{\cfrac{(\mathit{pre}) \qquad (\mathit{inner}) \qquad (\mathit{post})}{\mathcal{U}\{\mathsf{start}(\texttt{do}, i)\} \mathrel{:\cdot\cdot}\vdash \mathcal{U}\{\mathsf{start}(\texttt{do}, i)\}\{\mathsf{open}(\mathsf{f})\}\texttt{operate(); s} \mathrel{:\cdot\cdot} \ast\!\ast \cdot\cdot \ast\!\ast \cdot\cdot}}{\mathcal{U}\{\mathsf{start}(\texttt{do}, i)\} \mathrel{:\cdot\cdot}\vdash \mathcal{U}\{\mathsf{start}(\texttt{do}, i)\}\textbf{\texttt{open}}\texttt{(f); operate(); s} \mathrel{:\cdot\cdot}}$$

Example

$$\frac{\mathcal{U}\{\mathsf{start}(\mathsf{do}, i)\} :\cdot\cdot\vdash \mathcal{U}\{\mathsf{start}(\mathsf{do}, i)\}\{\mathsf{open}(\mathsf{f})\} :\cdot\cdot\ \mathsf{open}(\mathsf{f})\ \overset{\mathsf{close}(\mathsf{f})}{\cdot\cdot} \wedge \cdot\cdot}{(\textit{pre})}$$

$$\frac{\llbracket \overset{\mathsf{close}(\mathsf{f}),\mathsf{open}(\mathsf{f})}{\cdot\cdot} \rrbracket \subseteq \llbracket \cdot\cdot \rrbracket}{(\textit{inner})}$$

$$\frac{\begin{array}{c}\mathcal{U}\{\mathsf{start}(\mathsf{do}, i)\} :\cdot\cdot, \mathcal{U}\{\mathsf{start}(\mathsf{do}, i)\}\{\mathsf{open}(\mathsf{f})\}\{\mathsf{run}(\mathtt{operate}, 1)\} :\cdot\cdot\ \mathsf{open}(\mathsf{f})\ \overset{\mathsf{close}(\mathsf{f})}{\cdot\cdot} \ast\!\ast \overset{\mathsf{close}(\mathsf{f}),\mathsf{open}(\mathsf{f})}{\cdot\cdot} \\ \vdash \mathcal{U}\{\mathsf{start}(\mathsf{do}, i)\}\{\mathsf{open}(\mathsf{f})\}\{\mathsf{run}(\mathtt{operate}, 1)\}\mathtt{s} :\cdot\cdot\ \mathsf{open}(\mathsf{f})\ \overset{\mathsf{close}(\mathsf{f})}{\cdot\cdot} \ast\!\ast \overset{\mathsf{close}(\mathsf{f}),\mathsf{open}(\mathsf{f})}{\cdot\cdot} \ast\!\ast \cdot\cdot\ \mathsf{close}(\mathsf{f})\ \cdot\cdot\end{array}}{(\textit{post})}$$

$$\frac{\dfrac{(\textit{pre}) \qquad (\textit{inner}) \qquad (\textit{post})}{\mathcal{U}\{\mathsf{start}(\mathsf{do}, i)\} :\cdot\cdot\vdash \mathcal{U}\{\mathsf{start}(\mathsf{do}, i)\}\{\mathsf{open}(\mathsf{f})\}\mathtt{operate();\ s} :\cdot\cdot \ast\!\ast \cdot\cdot \ast\!\ast \cdot\cdot}}{\mathcal{U}\{\mathsf{start}(\mathsf{do}, i)\} :\cdot\cdot\vdash \mathcal{U}\{\mathsf{start}(\mathsf{do}, i)\}\mathtt{open(f);\ operate();\ s} :\cdot\cdot}$$

Asynchronous Communication

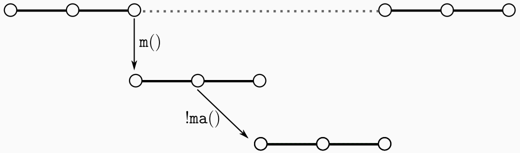
Asynchronous Language

Asynchronous Calls

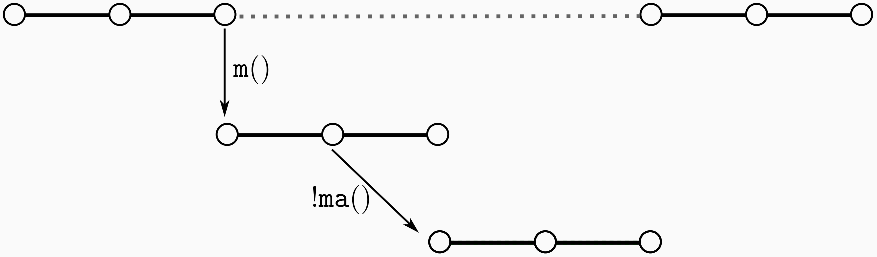
- Syntax: `!m()`
- Semantics: $[\![s]\!]^G_\tau$ are all traces produced by s, including its asynchronous calls

Tree-Like Asynchronous Communication

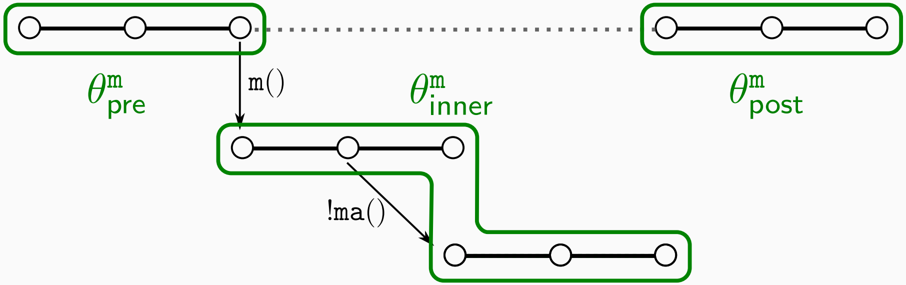
- All processes P_i invoked by P are run *directly* after P terminates.
- From the perspective of the caller of P, all P_i are invisible.
- **The specification θ_{inner} includes the asynchronously called processes**



Contracts

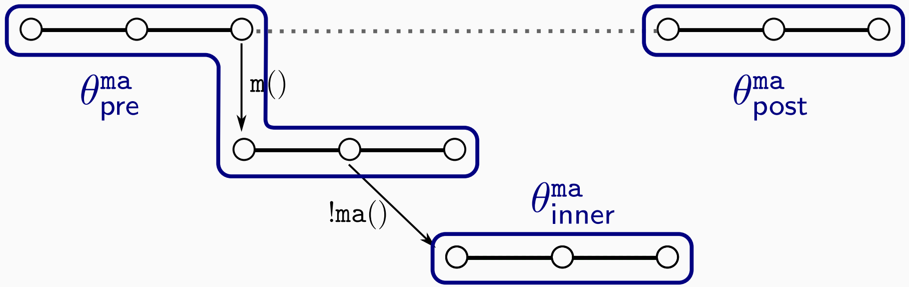


Contracts



$\theta^{\mathtt{m}}_{\mathsf{pre}}$
$\mathtt{m}()$
$\theta^{\mathtt{m}}_{\mathsf{inner}}$
$\theta^{\mathtt{m}}_{\mathsf{post}}$
$!\mathtt{ma}()$

Contracts


m()
!ma()
$\theta^{\mathtt{ma}}_{\mathsf{pre}}$
$\theta^{\mathtt{ma}}_{\mathsf{post}}$
$\theta^{\mathtt{ma}}_{\mathsf{inner}}$

- Decoupled pre-state and pre-trace
- Who is obliged to ensure θ_{post}?

Rules

- New judgment: $\{U\} :_G \Phi$ describes global (=including async. calls) traces

$$\sigma \models \{\mathcal{U}\} :_G \Phi \iff [\![\mathcal{U}]\!]^G_\sigma \subseteq [\![\Phi]\!]$$

- $\{\mathcal{U}\}s :_G \Phi$ is analogous
- schedule($\mathcal{U}$) returns the set of invocation events which are not resolved yet

$$\textbf{(async)}\ \frac{\Gamma \vdash \{\mathcal{U}\}\{\mathsf{invoc}(\mathtt{m}, i)\}!\mathtt{m}();\ \mathtt{s} :_G \Phi}{\Gamma \vdash \{\mathcal{U}\}!\mathtt{m}();\ \mathtt{s} :_G \Phi}\ i \text{ fresh} \quad \textbf{(return)}\ \frac{\Gamma \vdash \{\mathcal{U}\}\{\mathsf{ret}(\mathsf{id})\} :_G \Phi}{\Gamma \vdash \{\mathcal{U}\}\mathtt{return} :_G \Phi}$$

$$\textbf{(finish)}\ \frac{\mathsf{schedule}(\mathcal{U}) = \emptyset \qquad \Gamma \vdash \{\mathcal{U}\} : \Phi}{\Gamma \vdash \{\mathcal{U}\} :_G \Phi}$$

Asynchronous Calls

$$\text{schedule}(\mathcal{U}) = \{\text{invoc}(\mathtt{m}, i)\}$$

$$\textbf{(ScheduleD)}\ \frac{\vdash \qquad\qquad\qquad\qquad :_G}{\Gamma \vdash \{\mathcal{U}\} :_G \Phi ** \theta ** \Psi}$$

Asynchronous Calls

$$\mathrm{schedule}(\mathcal{U}) = \{\mathrm{invoc}(\mathtt{m}, i)\}$$

$$\Gamma \vdash \{\mathcal{U}\} : (\Phi \wedge \theta^{\mathtt{m}}_{\mathsf{pre}})$$

$$\Gamma, \{\mathcal{U}\} \qquad (\Phi \wedge \theta^{\mathtt{m}}_{\mathsf{pre}})$$

$$\textbf{(ScheduleD)}\ \frac{\vdash \{\mathcal{U}\} \qquad :_G (\Phi \wedge \theta^{\mathtt{m}}_{\mathsf{pre}})}{\Gamma \vdash \{\mathcal{U}\} :_G \Phi ** \theta ** \Psi}$$

Asynchronous Calls

$$\textbf{(ScheduleD)}\ \frac{\begin{array}{c}\mathsf{schedule}(\mathcal{U}) = \{\mathsf{invoc}(\mathtt{m}, i)\} \\ [\![\theta^{\mathtt{m}}_{\mathsf{inner}}]\!] \subseteq [\![\theta]\!] \qquad \Gamma \vdash \{\mathcal{U}\} : (\Phi \wedge \theta^{\mathtt{m}}_{\mathsf{pre}}) \\ \Gamma, \{\mathcal{U}\}\{\mathsf{run}(\mathsf{m}, i)\} : (\Phi \wedge \theta^{\mathtt{m}}_{\mathsf{pre}}) ** \theta^{\mathtt{m}}_{\mathsf{inner}} \\ \vdash \{\mathcal{U}\}\{\mathsf{run}(\mathsf{m}, i)\} \ :_G (\Phi \wedge \theta^{\mathtt{m}}_{\mathsf{pre}}) ** \theta^{\mathtt{m}}_{\mathsf{inner}}\end{array}}{\Gamma \vdash \{\mathcal{U}\} :_G \Phi ** \theta ** \Psi}$$

Asynchronous Calls

$$\textbf{(ScheduleD)}\ \frac{\begin{array}{c}\mathrm{schedule}(\mathcal{U}) = \{\mathrm{invoc}(\mathtt{m}, i)\} \\ [\![\theta^{\mathtt{m}}_{\mathsf{inner}}]\!] \subseteq [\![\theta]\!] \qquad \Gamma \vdash \{\mathcal{U}\} : (\Phi \land \theta^{\mathtt{m}}_{\mathsf{pre}}) \\ \Gamma, \{\mathcal{U}\}\{\mathrm{run}(\mathrm{m}, i)\} : (\Phi \land \theta^{\mathtt{m}}_{\mathsf{pre}}) ** \theta^{\mathtt{m}}_{\mathsf{inner}} \\ \vdash \{\mathcal{U}\}\{\mathrm{run}(\mathrm{m}, i)\} \ :_G (\Phi \land \theta^{\mathtt{m}}_{\mathsf{pre}}) ** \theta^{\mathtt{m}}_{\mathsf{inner}} ** \Psi \land \theta^{\mathtt{m}}_{\mathsf{post}}\end{array}}{\Gamma \vdash \{\mathcal{U}\} :_G \Phi ** \theta ** \Psi}$$

- Non-deterministic version explores all possible next scheduling decisions

Example (Spec. and Code)

```
do() { open(f); !closeF(); operate(); return; }
operate() { write(f); return; }
closeF() { close(f); return; }
```

$$C_{\mathsf{operate}} = \left\langle \cdot\cdot\ \mathsf{open(f)}\ \overset{\mathsf{close(f)}}{\cdot\cdot} \middle| \overset{\mathsf{close(f),open(f)}}{\cdot\cdot} \middle| \cdot\cdot\ \mathsf{close(f)}\ \cdot\cdot \right\rangle$$

Example (Proof Sketch)

$\Gamma \vdash \{U\}$`open(f); !closeF(); operate(); return;` $:_G \cdots$

Example (Proof Sketch)

$$\dfrac{\begin{array}{c}\dfrac{}{\Gamma' \vdash \{U\}\{\mathsf{open(f)}\}\{\mathsf{invoc(}\texttt{closeF()}\mathsf{)}\}\texttt{operate(); }\textbf{\texttt{return}}\texttt{;} :_G \cdot\cdot} \\ \vdots \end{array}}{\Gamma \vdash \{U\}\textbf{\texttt{open}}\texttt{(f); !closeF(); operate(); }\textbf{\texttt{return}}\texttt{;} :_G \cdot\cdot}$$

Example (Proof Sketch)

$$\begin{array}{c}
\hline
\Gamma'' \vdash \{U\}\{\mathsf{open}(\mathsf{f})\}\{\mathsf{invoc}(\texttt{closeF}(),1)\}\{\mathsf{run}(\texttt{operate},2)\}\textbf{return};\ :_G \Phi ** \cdot\cdot\ \mathsf{close}(\mathsf{f})\ \cdot\cdot \\
\hline
\vdots \\
\hline
\Gamma' \vdash \{U\}\{\mathsf{open}(\mathsf{f})\}\{\mathsf{invoc}(\texttt{closeF}())\}\texttt{operate();}\ \textbf{return};\ :_G \cdot\cdot \\
\hline
\vdots \\
\hline
\Gamma \vdash \{U\}\textbf{open}\texttt{(f); !closeF(); operate();}\ \textbf{return};\ :_G \cdot\cdot
\end{array}$$

Example (Proof Sketch)

$$\dfrac{\dfrac{\dfrac{\dfrac{}{\Gamma''' \vdash \{U\}\{\mathrm{open}(\mathrm{f})\}\{\mathrm{invoc}(\mathtt{closeF}(),1)\}\{\mathrm{run}(\mathtt{operate},2)\}\{\mathrm{ret}(0)\} :_G \Phi ** \cdot\cdot\ \mathrm{closeF}\ \cdot\cdot}}{\vdots}}{\Gamma'' \vdash \{U\}\{\mathrm{open}(\mathrm{f})\}\{\mathrm{invoc}(\mathtt{closeF}(),1)\}\{\mathrm{run}(\mathtt{operate},2)\}\mathtt{\textbf{return}};\ :_G \Phi ** \cdot\cdot\ \mathrm{close}(\mathrm{f})\ \cdot\cdot}}{\vdots}$$

$$\dfrac{\dfrac{\vdots}{\Gamma' \vdash \{U\}\{\mathrm{open}(\mathrm{f})\}\{\mathrm{invoc}(\mathtt{closeF}())\}\mathtt{operate();}\ \mathtt{\textbf{return}};\ :_G \cdot\cdot}}{\vdots}$$

$$\dfrac{\vdots}{\Gamma \vdash \{U\}\mathtt{\textbf{open}(f);\ !closeF();\ operate();}\ \mathtt{\textbf{return}};\ :_G \cdot\cdot}$$

Example (Proof Sketch)

$$\begin{array}{c}
\hline
\Gamma'''' \vdash \{U\}\{\mathsf{open}(\mathsf{f})\}\{\mathsf{invoc}(\texttt{closeF}(), 1)\}\{\mathsf{run}(\texttt{operate}, 2)\}\{\mathsf{ret}(0)\}\{\mathsf{run}(\texttt{closeF}, 1)\} :_G \Phi ** \cdot\cdot\ \mathsf{closeF}\ \cdot\cdot \\
\hline
\vdots \\
\hline
\Gamma''' \vdash \{U\}\{\mathsf{open}(\mathsf{f})\}\{\mathsf{invoc}(\texttt{closeF}(), 1)\}\{\mathsf{run}(\texttt{operate}, 2)\}\{\mathsf{ret}(0)\} :_G \Phi ** \cdot\cdot\ \mathsf{closeF}\ \cdot\cdot \\
\hline
\vdots \\
\hline
\Gamma'' \vdash \{U\}\{\mathsf{open}(\mathsf{f})\}\{\mathsf{invoc}(\texttt{closeF}(), 1)\}\{\mathsf{run}(\texttt{operate}, 2)\}\texttt{\textbf{return}};\ :_G \Phi ** \cdot\cdot\ \mathsf{close}(\mathsf{f})\ \cdot\cdot \\
\hline
\vdots \\
\hline
\Gamma' \vdash \{U\}\{\mathsf{open}(\mathsf{f})\}\{\mathsf{invoc}(\texttt{closeF}())\}\texttt{operate();}\ \texttt{\textbf{return}};\ :_G \cdot\cdot \\
\hline
\vdots \\
\hline
\Gamma \vdash \{U\}\texttt{\textbf{open}(f);}\ \texttt{!closeF();}\ \texttt{operate();}\ \texttt{\textbf{return}};\ :_G \cdot\cdot
\end{array}$$

Example (Proof Sketch)

$$
\dfrac{\dfrac{\vdots}{\Gamma'''' \vdash \{U\}\{\mathsf{open}(\mathsf{f})\}\{\mathsf{invoc}(\texttt{closeF}(),1)\}\{\mathsf{run}(\texttt{operate},2)\}\{\mathsf{ret}(0)\}\{\mathsf{run}(\texttt{closeF},1)\} : \Phi ** \cdot\cdot\ \mathsf{closeF}\ \cdot\cdot}}{\Gamma'''' \vdash \{U\}\{\mathsf{open}(\mathsf{f})\}\{\mathsf{invoc}(\texttt{closeF}(),1)\}\{\mathsf{run}(\texttt{operate},2)\}\{\mathsf{ret}(0)\}\{\mathsf{run}(\texttt{closeF},1)\} :_G \Phi ** \cdot\cdot\ \mathsf{closeF}\ \cdot\cdot}
$$

$$
\dfrac{\vdots}{\Gamma''' \vdash \{U\}\{\mathsf{open}(\mathsf{f})\}\{\mathsf{invoc}(\texttt{closeF}(),1)\}\{\mathsf{run}(\texttt{operate},2)\}\{\mathsf{ret}(0)\} :_G \Phi ** \cdot\cdot\ \mathsf{closeF}\ \cdot\cdot}
$$

$$
\dfrac{\vdots}{\Gamma'' \vdash \{U\}\{\mathsf{open}(\mathsf{f})\}\{\mathsf{invoc}(\texttt{closeF}(),1)\}\{\mathsf{run}(\texttt{operate},2)\}\textbf{\texttt{return}}\texttt{;} :_G \Phi ** \cdot\cdot\ \mathsf{close}(\mathsf{f})\ \cdot\cdot}
$$

$$
\dfrac{\vdots}{\Gamma' \vdash \{U\}\{\mathsf{open}(\mathsf{f})\}\{\mathsf{invoc}(\texttt{closeF}())\}\texttt{operate();}\ \textbf{\texttt{return}}\texttt{;} :_G \cdot\cdot}
$$

$$
\dfrac{\vdots}{\Gamma \vdash \{U\}\textbf{\texttt{open}}\texttt{(f);}\ \texttt{!closeF();}\ \texttt{operate();}\ \textbf{\texttt{return}}\texttt{;} :_G \cdot\cdot}
$$

Conclusion

Related Approaches

Typestate

- Typestate is bound to data/objects, not a local view of procedures.

Related Approaches

Typestate

- Typestate is bound to data/objects, not a local view of procedures.

Behavioral Contracts

- Split between parameter-precondition and heap-precondition
- Specify methods that must or may run before a method starts

Related Approaches

Typestate

- Typestate is bound to data/objects, not a local view of procedures.

Behavioral Contracts

- Split between parameter-precondition and heap-precondition
- Specify methods that must or may run before a method starts

First-order Histories and Ghost Variables

- Generally uncompositional for methods, unwieldy specification language

Related Approaches

Typestate

- Typestate is bound to data/objects, not a local view of procedures.

Behavioral Contracts

- Split between parameter-precondition and heap-precondition
- Specify methods that must or may run before a method starts

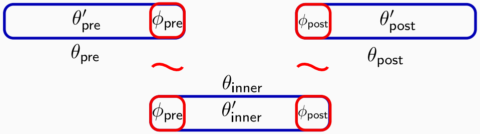
First-order Histories and Ghost Variables

- Generally uncompositional for methods, unwieldy specification language

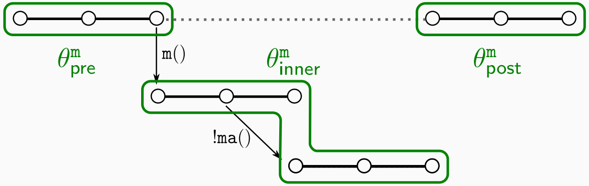
Session Types for Active Objects

- Top-down, not bottom-up, with no context transmitted down
- If context is transmitted, they mirror behavioral contracts

Conclusion

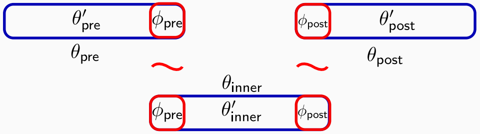


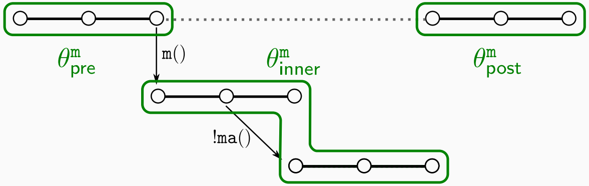
θ'_{pre}
ϕ_{pre}
θ_{pre}
ϕ_{post}
θ'_{post}
θ_{post}
$\sim$
$\sim$
θ_{inner}
ϕ_{pre}
θ'_{inner}
ϕ_{post}



$\theta^{\mathtt{m}}_{\text{pre}}$
m()
$\theta^{\mathtt{m}}_{\text{inner}}$
$\theta^{\mathtt{m}}_{\text{post}}$
!ma()

Conclusion



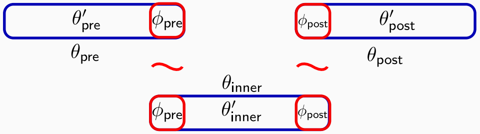


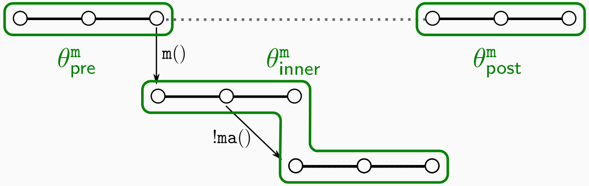
$\theta^{\mathtt{m}}_{\text{pre}}$
m()
$\theta^{\mathtt{m}}_{\text{inner}}$
$\theta^{\mathtt{m}}_{\text{post}}$
!ma()

Context-aware Trace Contracts

- Local specification of global trace context
- Modular, local calculus: 1 PO per procedure, all calls abstracted with contracts
- See paper: Event semantics, call management, observations
- Future work: Support for full Asynchronicity

Conclusion




m()
$\theta^{\mathtt{m}}_{\text{pre}}$
$\theta^{\mathtt{m}}_{\text{inner}}$
$\theta^{\mathtt{m}}_{\text{post}}$
!ma()

Context-aware Trace Contracts

- Local specification of global trace context
- Modular, local calculus: 1 PO per procedure, all calls abstracted with contracts
- See paper: Event semantics, call management, observations
- Future work: Support for full Asynchronicity

Thank you for your attention