TaDA Live: Compositional Reasoning for Termination of Fine-grained Concurrent Programs

Emanuele D’Osualdo
MPI-SWS

Julian Sutherland
Imperial College / Nethermind

Azadeh Farzan
University of Toronto

Philippa Gardner
Imperial College London

Iris Workshop’22
Goal

Compositional verification of total correctness:
- for fine-grained concurrent programs
- with blocking behaviour
- under fair scheduling

Approach

TaDA Live: a novel concurrent separation logic
A beefy 84-page paper!

- In-depth motivation of design
- Formalisation of the model
- Full proof system with illustrative examples
- Several realistic case studies
- Soundness argument
- More related work
Busy-waiting

//...  
[x] := 1
//...

//...

do {
  d := [x]
} while (d ≠ 1)
//...

Scope:

- First-order imperative code — think java.util.concurrent (no step-indexing)
- Sequential consistency semantics
- Pen & Paper logic (more on this later)
Busy-waiting

Program features of interest:

- **Fine-grained concurrency**
  Synchronization through custom busy-waiting patterns
Busy-waiting

Program features of interest:

- **Fine-grained concurrency**
  Synchronization through custom busy-waiting patterns

- **Blocking behaviour**
  Termination of a thread requires cooperation of the others
Busy-waiting

Program features of interest:

- **Fine-grained concurrency**
  Synchronization through custom busy-waiting patterns

- **Blocking behaviour**
  Termination of a thread requires cooperation of the others

- **Fairness assumption**
  Necessary for termination with blocking
Busy-waiting

Properties of interest:

- Functional correctness
- Termination guarantees
The challenge

Compositionality is the main challenge:

- Thread-local (scalability)
- Module-local (reuse)

What should specifications look like?
The challenge

\[
\{x \mapsto 0\} \quad \quad /\ldots
\]

/\ldots

/\ldots

\[\begin{align*}
[x] &:= 1 \\
/\ldots
\end{align*}\]

/\ldots

\{True\}

/\ldots

/\ldots

\{x \mapsto 0\} \quad \quad /\ldots

/\ldots

/\ldots

\begin{align*}
\text{do } & \{ \\
& d := [x] \\
\text{while} & (d \neq 1) \\
\text{while} & (d \neq 1) \\
\text{while} & (d \neq 1)
\end{align*}

Total Hoare triples:

\[
\vdash \{P\} \triangleright \{Q\} \text{ total}
\]

\[
\triangleright \text{ run from a state satisfying } P \text{ always terminates} \text{ in a state satisfying } Q.
\]
The challenge

Total Hoare triples:

\[ \vdash \{ P \} \mathcal{C} \{ Q \} \text{ total} \]

\[ \vdash \{ x \mapsto 1 \} \text{ do } \{ d := [x] \} \text{ while}(d \neq 1) \{ \text{True} \} \text{ total} \]

\[ \mathcal{C} \text{ run from a state satisfying } P \textbf{ always terminates} \text{ in a state satisfying } Q. \]
The challenge

\[
\{x \mapsto 0\} \quad //... \\
//... \\
//... \\
[x] := 1 \\
//... \\
//... \\
\{\text{True}\}
\]

Total Hoare triples:

\[\vdash \{P\} \triangleright \{Q\} \text{ total}\]

\[\vdash \{\text{share}(x)\} \text{ do } \{d := [x]\} \text{ while}(d \neq 1) \{\text{True}\} \text{ total}\]

\[\triangleright \text{ run from a state satisfying } P \textbf{ always terminates} \text{ in a state satisfying } Q.\]
The challenge

Total Hoare triples:

\[ \vdash \{ P \} \subseteq \{ Q \} \text{ total} \]

\[ \vdash \{ \text{share}(x) \} \text{ do } \{ d := [x] \} \text{ while } (d \neq 1) \{ \text{True} \} \text{ total} \]

The loop is **blocking**: it cannot promise to always terminate…
The challenge

\[ \{ x \mapsto 0 \} \]

//...

//...

//...

\[ \text{do } \{ \]
\[ \text{d := [x]} \]
\[ \} \text{ while( d \neq 1 )} \]

//...

\[ \{ \text{True} \} \]

I terminate if: my environment sets \( x \) to 1 eventually.

Total Hoare triples:

\[ \vdash \{ P \} \mathbin{\text{C}} \{ Q \} \text{ total} \]

\[ \vdash \{ \text{share}(x) \} \text{ do } \{ \text{d := [x]} \} \text{ while( d \neq 1 )} \{ \text{True} \} \text{ total} \]

The loop is blocking: it cannot promise to always terminate...
Liveness invariants

TaDA Live's starting observation:

Blocking = termination conditional on *liveness invariants*

I terminate if: my environment sets x to 1 eventually.
Liveness invariants

\[
\{ x \mapsto 0 \}
\]

//...

\[ [x] := 1 \]

//...

\[
\{ \text{True} \}
\]

Liveness invariants

I terminate if: my environment sets \( x \) to 1 \textit{eventually}.

**TaDA Live's starting observation:**

Blocking = termination conditional on \textit{liveness invariants}

\[ \square(\Diamond \text{“good state”}) \]

\textit{Always} \quad \textit{Eventually}
TaDA Live’s contributions

TaDA Live’s innovations:

1. **Subjective Obligations**
   Thread-local reasoning with liveness invariants

2. **Obligation layers**
   Compositional deadlock-freedom

3. **Logical atomicity for blocking code**
   Enabling modular reasoning
TaDA Live’s contributions

TaDA Live’s innovations:

1. **Subjective Obligations**
   Thread-local reasoning with liveness invariants

2. **Obligation layers**
   Compositional deadlock-freedom

3. **Logical atomicity for blocking code**
   Enabling modular reasoning
Rely/Guarantee in TaDA

\[
\{ x \mapsto 0 \} \\
\{ \ast \} \\
\text{[x] := 1} \\
\{ \ast \} \\
\{ \text{True} \}
\]

\[
\{ \ast \} \\
\begin{array}{c}
\begin{array}{c}
\text{do} \\
\text{d := [x]}
\end{array} \\
\{ \text{while (d }\neq 1) \}
\end{array}
\]
Rely/Guarantee in TaDA

\[
\begin{align*}
&\{ x \mapsto 0 \} \\
&\{ \text{sh}(x, 0) \star [w] \star \exists v. \text{sh}(x, v) \} \\
&\{ \text{do} \{ \\
&\quad [x] := 1 \\
&\quad d := [x] \\
&\quad \} \text{ while}(d \neq 1) \} \\
&\{ \text{sh}(x, 1) \star [w] \star \exists v. \text{sh}(x, v) \} \\
&\{ \text{True} \}
\end{align*}
\]

Protocol:
\[ \mathcal{I}(\text{sh}(x, v)) \triangleq (x \mapsto v) \]

Allowed updates of \text{sh}:
\[ w : (x, 0) \rightsquigarrow (x, 1) \]

(w is write permission)
Rely/Guarantee in TaDA

\[ \{ x \mapsto 0 \} \]
\[ \{ \text{sh}(x, 0) * [w] * \exists v. \text{sh}(x, v) \} \]
\[ [x] := 1 \]
\[ \{ \text{sh}(x, 1) * [w] * \exists v. \text{sh}(x, v) \} \]
\[ \{ \text{True} \} \]

Protocol:
\[ J(\text{sh}(x, v)) \triangleq (x \mapsto v) \]

Allowed updates of sh:
\[ w : (x, 0) \leadsto (x, 1) \]
(w is write permission)

The standard (total) while rule:

\[ \forall \beta. \vdash \{ P(\beta) \land B \} \subseteq \{ \exists \beta'. P(\beta') \land \beta' < \beta \} \]

\[ \vdash \{ P(\_\_\_) \} \text{ while}(B) \{ C \} \{ P(\_\_) \land \neg B \} \]
\[
\begin{aligned}
&\{ x \mapsto 0 \} \\
&\{ \text{sh}(x, 0) \times [w] \times \exists v. \text{sh}(x, v) \} \\
&\hspace{1cm} \text{do} \{ \\
&\hspace{2cm} d := [x] \\
&\hspace{1cm} \} \text{ while}(d \neq 1) \\
&\{ \text{sh}(x, 1) \times [w] \times \exists v. \text{sh}(x, v) \} \\
&\{ \text{True} \}
\end{aligned}
\]

Protocol:
\[\mathcal{I}(\text{sh}(x, v)) \triangleq (x \mapsto v)\]

Allowed updates of \text{sh}:
\[w : (x, 0) \leadsto (x, 1)\]

(\(w\) is write permission)

The standard (total) while rule:

\[\forall \beta. \vdash \{ P(\beta) \land B \} \subseteq \{ \exists \beta'. P(\beta') \land \beta' < \beta \}\]

\[\vdash \{ P(\_ \_ \_ ) \} \text{ while}(B)\{C \} \{ P(\_ \_ \_ ) \land \neg B \}\]

Variant
Upper bound on amount of work to do

Well founded
Amount of work decreases
Rely/Guarantee in TaDA

\[
\begin{align*}
\{ x \mapsto 0 \} \\
\{ \text{sh}(x, 0) \ast [w] \ast \exists v. \text{sh}(x, v) \} \\
\{ \text{do} \{ \\
\quad d := [x] \\
\quad \} \text{ while}(d \neq 1) \} \\
\{ \text{sh}(x, 1) \ast [w] \ast \exists v. \text{sh}(x, v) \} \\
\{ \text{True} \}
\end{align*}
\]

Protocol:
\[ \mathcal{I}(\text{sh}(x, v)) \triangleq (x \mapsto v) \]

Allowed updates of \text{sh}:
\[ w : (x, 0) \leadsto (x, 1) \]
\((w \text{ is write permission})\)

TaDA Live’s while rule:

\[
\forall \beta. \vdash \{P(\beta) \land B\} \implies \{ \exists \beta'. P(\beta') \land \beta' \leq \beta \}
\]

\[
\vdash \{ P(\_\_\_) \} \text{ while}(B)\{C\} \{ P(\_\_\_) \land \neg B \}
\]
Rely/Guarantee in TaDA

\(\{ x \mapsto 0 \} \)

\(\{ \text{sh}(x, 0) * [w] * \exists v. \text{sh}(x, v) \} \)

[do]

[\( [x] := 1 \)]

\(\{ \text{sh}(x, 1) * [w] * \exists v. \text{sh}(x, v) \} \)

\(\{ \text{True} \} \)

\(\{ x \mapsto v \} = I(\text{sh}(x, v)) \)

Protocol:

\(\{ x \mapsto v \} \)

Allowed updates of \(\text{sh}\):

\(w : (x, 0) \leadsto (x, 1) \)

\((w \text{ is write permission})\)

Target state \(T\):

\(\text{sh}(x, 1)\)

TaDA Live’s while rule:

\(\forall \beta. \vdash \{ P(\beta) \ast T \land B \} \subseteq \{ \exists \beta'. P(\beta') \land \beta' < \beta \}\)

\(\forall \beta. \vdash \{ P(\beta) \land B \} \subseteq \{ \exists \beta'. P(\beta') \land \beta' \leq \beta \}\)

\(\vdash \{ P(\_\_) \} \text{ while}(B)\{C\} \{ P(\_\_) \land \neg B \}\)

\(\}\)
Rely/Guarantee in TaDA

\begin{align*}
\{ x \mapsto 0 \} \\
\{ \text{sh}(x, 0) \ast [w] \ast \exists v. \text{sh}(x, v) \}
\end{align*}

\begin{align*}
[x] := 1 \\
\{ \text{sh}(x, 1) \ast [w] \ast \exists v. \text{sh}(x, v) \}
\end{align*}

\{ \text{True} \}

Target state $T$: $\text{sh}(x, 1)$

TaDA Live’s while rule:

\begin{align*}
\forall \beta. & \vdash \{ P(\beta) \ast T \land B \} \subseteq \{ \exists \beta'. P(\beta') \land \beta' < \beta \} \\
\forall \beta. & \vdash \{ P(\beta) \land B \} \subseteq \{ \exists \beta'. P(\beta') \land \beta' \leq \beta \}
\end{align*}

\begin{align*}
\vdash \{ P(\_\_) \} \text{while}(B)\{C\} \{ P(\_\_) \land \neg B \}
\end{align*}

Protocol:

$\mathcal{I}(\text{sh}(x, v)) \triangleq (x \mapsto v)$

Allowed updates of sh:

$w : (x, 0) \leadsto (x, 1)$

($w$ is write permission)
Rely/Guarantee in TaDA

\[ \{ x \mapsto 0 \} \]
\[ \{ \text{sh}(x, 0) \ast [w] \ast \exists v. \text{sh}(x, v) \} \]
\[ [x] := 1 \]
\[ \{ \text{sh}(x, 1) \ast [w] \ast \exists v. \text{sh}(x, v) \} \]
\[ \{ \text{True} \} \]

Target state \( T \):
\( \text{sh}(x, 1) \)

TaDA Live’s while rule:

\[ \text{Protocol:} \]
\[ I(\text{sh}(x, v)) \triangleq (x \mapsto v) \]

Allowed updates of \( \text{sh} \):
\( w : (x, 0) \leadsto (x, 1) \)
\((w \text{ is write permission})\)

\[ \forall \beta. \vdash \{ P(\beta) \ast T \} \]
\[ \forall \beta. \vdash \{ P(\beta) \} \]
\[ \vdash \{ P(\_\_) \} \]

Env. Liveness Condition
Here as LTL only for illustration
Liveness Invariants

Proving $\Diamond (\Box sh(x, 1))$

- Protocol says what is **allowed** (safety)
- Need to know what **will** happen (liveness)

Protocol:

$I (sh(x, v)) \triangleq (x \mapsto v)$

Allowed updates of $sh$:

$w : (x, 0) \leadsto (x, 1)$

($w$ is write permission)
Liveness Invariants

TaDA Live’s Obligations:

- An obligation is an exclusive token $u$
- $u$ is *fulfilled* = ‘Nobody holds $u$’
- Implicitly, obligations encode a liveness invariant:

  $\Box(\Diamond u \text{ fulfilled})$

- Subjective assertions:

  $[u]^L = \text{‘I own } u\text{’}$  \quad $[u]^E = \text{‘I know env. owns } u\text{’}$

- $\vdash \{P\} \subseteq \{Q\} \approx \text{If } \Box([u]^E \implies \Diamond(u \text{ fulfilled})) \text{ then } C \text{ terminates}$
Liveness Invariants

\[
\begin{align*}
\{ \text{sh}(x, 0) \ast [w] \} & \quad \{ x \mapsto 0 \} \\
[\{x\}] := 1 \\
\{ \text{sh}(x, 1) \ast [w] \} & \quad \{ \text{True} \}
\end{align*}
\]

\[
\begin{align*}
\{ \text{sh}(x, 0) \} & \quad \{ \text{sh}(x, 0) \lor \text{sh}(x, 1) \} \\
\text{do} \quad \{ \\
\quad d := [x] \\
\} \text{ while}(d \neq 1) \\
\{ \exists v. \text{sh}(x, v) \}
\end{align*}
\]

Protocol:
\[
\mathcal{I} (\text{sh}(x, v)) \triangleq (x \mapsto v)
\]

Allowed updates of \text{sh}:
\[
\mathcal{W} : (x, 0) \sim (x, 1)
\]

(\mathcal{W} \text{ is write permission})
Liveness Invariants

\[
\begin{align*}
\{ \text{sh}(x, 0) \ast [w] \ast \lfloor u \rfloor \} & \quad \{ x \mapsto 0 \} \\
\{ \text{sh}(x, 1) \ast [w] \} & \quad \{ \text{True} \}
\end{align*}
\]

\[
\{ \text{sh}(x, 0) \ast \lfloor u \rfloor^E \} \quad \lor \quad \text{sh}(x, 1)
\]

\[
\text{do} \{ \\
\quad d := [x] \\
\} \quad \text{while} (d \neq 1)
\]

\[
\exists v. \text{sh}(x, v)
\]

TaDA Live Protocol:

* u obligation

Allowed updates of \( \text{sh} \):

* \( w : (x, 0), u \leadsto (x, 1) \)

The update fulfils \( u \)
Liveness Invariants

\[
\begin{align*}
\{ \text{sh}(x, 0) * [w] \} & \quad \{ x \mapsto 0 \} \\
\{ \text{sh}(x, 0) * [u]^E \} & \quad \{ \text{sh}(x, 0) * [u]^L \} \\
\{ \text{sh}(x, 1) * [w] \} & \quad \{ \text{True} \}
\end{align*}
\]

\[
\begin{aligned}
[x] := 1 \\
\text{do } \{ \\
\quad d := [x] \\
\} \text{ while}(d \neq 1)
\end{aligned}
\]

\[
\{ \exists \nu. \text{sh}(x, \nu) \}
\]

\begin{itemize}
\item \( \Box L \Rightarrow \Diamond (\Box T) \)
\item \( \forall \beta. \vdash \{ P(\beta) * T \land B \} \subseteq \{ \exists \beta'. P(\beta') \land \beta' < \beta \} \)
\item \( \forall \beta. \vdash \{ P(\beta) \land B \} \subseteq \{ \exists \beta'. P(\beta') \land \beta' \leq \beta \} \)
\item \( \vdash \{ P(\_)*L \} \text{ while}(B)\{C\} \{ P(\_)*L \land \neg B \} \)
\end{itemize}

TaDA Live Protocol:

\( u \) obligation

Allowed updates of \( \text{sh} \):
\( w : (x, 0), u \mapsto (x, 1) \)

The update fulfils \( u \)
Liveness Invariants

\[
\begin{align*}
\{ \text{sh}(x, 0) \ast [w] \ast [u]^L \} & \quad \{ x \rightarrow 0 \} \\
{x} & \leftarrow 1 \\
\{ \text{sh}(x, 1) \ast [w] \} & \quad \{ \text{True} \}
\end{align*}
\]

\[\text{TaDA Live Protocol:} \]
\[\begin{align*}
&\; \text{w obligation} \\
&\; \text{Allowed updates of \text{sh}:} \\
&\; w : (x, 0), u \sim (x, 1) \\
&\; \text{The update fulfils } u
\end{align*}\]

\[\text{Target state } T: \]
\[\text{sh}(x, 1)\]

\[\Box L \Rightarrow \Diamond (\Box T)\]

\[\forall \beta. \vdash \{ P(\beta) \ast T \land B \} \supset \{ \exists \beta’. P(\beta’) \land \beta’ < \beta \}\]

\[\forall \beta. \vdash \{ P(\beta) \land B \} \supset \{ \exists \beta’. P(\beta’) \land \beta’ \leq \beta \}\]

\[\vdash \{ P(_) \ast L \} \text{ while}(B)\{C\} \{ P(_) \ast L \land \neg B \}\]
Liveness Invariants

\[
\begin{align*}
\{ & \text{sh}(x, 0) \ast [w] \} \\
\{ & \text{sh}(x, 0) \ast [u]^E \} \\
\{ & \text{sh}(x, 1) \ast [w] \} \\
\{ & \text{True} \}
\end{align*}
\]

\[\{ x \mapsto 0 \}\]

\[\{ \text{sh}(x, 1) \ast [w] \}\]

\[\{ \text{do } \{ \\
\text{d := } [x] \\
\} \text{ while(} \text{d} \neq 1) \}
\]

\[\{ \exists \upsilon. \text{sh}(x, \upsilon) \}\]

\[\{ \text{True} \}
\]

During the loop \(L\):
\[(\text{sh}(x, 0) \ast [u]^E) \lor \text{sh}(x, 1)\]

Target state \(T\):
\[
\text{sh}(x, 1)
\]

TaDA Live Protocol:

\(u\) obligation

Allowed updates of \(\text{sh}\):
\(w : (x, 0), u \leadsto (x, 1)\)

The update fulfils \(u\)

\[\square L \Rightarrow \Diamond (\square T)\]

\[\forall \beta. \vdash \{ P(\beta) \ast T \land B \} \supset \{ \exists \beta'. P(\beta') \land \beta' < \beta \}\]

\[\forall \beta. \vdash \{ P(\beta) \land B \} \supset \{ \exists \beta'. P(\beta') \land \beta' \leq \beta \}\]

\[\vdash \{ P(\_ \_ ) \ast L \} \text{ while}(B)\{C\} \{ P(\_\_ ) \ast L \land \neg B \}\]
Environment liveness condition

\[ \square L \Rightarrow \Diamond (\square sh(x, 1)) \]
Environment liveness condition

\[ \square L \Rightarrow \square (\Diamond \text{sh}(x, 1)) \]

\[ \square L \Rightarrow \Diamond (\Box \text{sh}(x, 1)) \]

\[ \text{sh}(x, 1) \Rightarrow \Box \text{sh}(x, 1) \]
Environment liveness condition

\( \square L \Rightarrow \square (\Diamond \text{sh}(x, 1)) \)

\( \square L \Rightarrow \Diamond (\square \text{sh}(x, 1)) \)

**Stability**

Protocol asserts:

\((x, 1) \not\Rightarrow (x, 0)\)

\(\text{sh}(x, 1) \Rightarrow \square \text{sh}(x, 1)\)
Environment liveness condition

**Environ. liveness**

If $L$ then either:

- $\text{sh}(x, 1)$, or
- $\lfloor u \rfloor^E$ and fulfilling $u$
  takes us to target

\[ \Box L \Rightarrow \Box (\Diamond \text{sh}(x, 1)) \]

\[ \Box L \Rightarrow \Diamond (\Box \text{sh}(x, 1)) \]

**Stability**

Protocol asserts:

$(x, 1) \not\Rightarrow (x, 0)$

$\text{sh}(x, 1) \Rightarrow \Box \text{sh}(x, 1)$
$L \triangleq (sh(x, 0) \ast \lfloor u \rfloor^E) \lor sh(x, 1)$

$w : (x, 0), u \leadsto (x, 1)$

**Environ. liveness**

If $L$ then either:

- $sh(x, 1)$, or
- $\lfloor u \rfloor^E$ and fulfilling $u$ takes us to target

**Stability**

Protocol asserts:

$(x, 1) \not\leadsto (x, 0)$

$sh(x, 1) \Rightarrow \Box sh(x, 1)$

$\Box L \Rightarrow \Box (\Diamond sh(x, 1))$
\[ L \triangleq (\text{sh}(x, 0) \ast [u]^E) \lor \text{sh}(x, 1) \]

\[ \mathbf{w} : (x, 0), u \leadsto (x, 1) \]

### Environ. liveness

If \( L \) then either:

- \( \text{sh}(x, 1) \), or
- \([u]^E\) and fulfilling \( u \)
  takes us to target

\[ \Box (\Diamond u \text{ fulfilled}) \]

\[ \Box L \Rightarrow \Box (\Diamond \text{sh}(x, 1)) \]

### Stability

Protocol asserts:

\( (x, 1) \not\Rightarrow (x, 0) \)

\( \text{sh}(x, 1) \Rightarrow \Box \text{sh}(x, 1) \)

\[ \Box L \Rightarrow \Diamond (\Box \text{sh}(x, 1)) \]
Environ. liveness

If $L$ then either:

- $T$ holds, or
- $\lceil u \rceil^E$ and fulfilling $u$ takes us to $T$

Stability

$T \Rightarrow \Box T$

\[
\forall \beta. \vdash \{ P(\beta) \land T \land B \} \subseteq \{ \exists \beta'. P(\beta') \land \beta' < \beta \}
\]

\[
\forall \beta. \vdash \{ P(\beta) \land B \} \subseteq \{ \exists \beta'. P(\beta') \land \beta' \leq \beta \}
\]

\[
\vdash \{ P(\_ \land L) \} \text{ while}(B)\{C\} \{ P(\_ \land L \land \neg B) \}
\]
How can a thread “fulfil” an obligation?

- To fulfil $u = \text{To go from owning, to not owning } [u]^L$
Obligation fulfilment

How can a thread “fulfil” an obligation?

- To fulfil $u$ = To go from owning, to not owning $[u]^L$
- $[u]^L$ cannot be affine: $[u]^L \nRightarrow \text{emp}$
Obligation fulfilment

How can a thread “fulfil” an obligation?

- To fulfil $u = \text{To go from owning, to not owning } [u]_L$
- $[u]_L$ cannot be affine: $[u]_L \not\Rightarrow \text{emp}$
- Issue with non-affine resources and invariants:

$$[u]_L \Rightarrow [\overline{u}]_L \Rightarrow \text{emp}$$
How can a thread “fulfil” an obligation?

- To fulfil $u = \text{To go from owning, to not owning } [u]^L$
- $[u]^L$ cannot be affine: $[u]^L \not\Rightarrow \text{emp}$
- Issue with non-affine resources and invariants:

$$[u]^L \Rightarrow [u]^L \Rightarrow \text{emp}$$

- In TaDA Live: that’s not a bug, it’s a feature!
How can a thread “fulfil” an obligation?

- To fulfil $u = \text{To go from owning, to not owning } [u]^L$
- $[u]^L$ cannot be affine: $[u]^L \not\Rightarrow \text{emp}$
- Issue with non-affine resources and invariants:
  \[ [u]^L \Rightarrow [u]^L \Rightarrow \text{emp} \]
- In TaDA Live: that’s not a bug, it’s a feature!
  - Obligations “belong” to an invariant (named $r$): $[u]^L_r$
Obligation fulfilment

How can a thread “fulfil” an obligation?

- To fulfil $u = \text{To go from owning, to not owning } [u]^L$
- $[u]^L$ cannot be affine: $[u]^L \not\Rightarrow \text{emp}$
- Issue with non-affine resources and invariants:
  \[
  [u]^L \Rightarrow [u]^L \Rightarrow \text{emp}
  \]

- In TaDA Live: that’s not a bug, it’s a feature!
  - Obligations “belong” to an invariant (named $r$): $[u]_r^L$
  - Invariants can only own obligations that belong to them:
  \[
  I (\text{sh}_r(x, v)) \triangleq x \mapsto v \ast (v = 1 \Rightarrow [u]_r^L)
  \]
Obligation fulfilment

How can a thread “fulfil” an obligation?

- To fulfil $u = \text{To go from owning, to not owning } [u]^L$
- $[u]^L$ cannot be affine: $[u]^L \not\Rightarrow \text{emp}$
- Issue with non-affine resources and invariants:
  \[ [u]^L \Rightarrow [u]^L \Rightarrow \text{emp} \]

- In TaDA Live: that’s not a bug, it’s a feature!
  - Obligations “belong” to an invariant (named $r$): $[u]^L_r$
  - Invariants can only own obligations that belong to them:
    \[ \mathcal{I} (\text{sh}_r(x,v)) \triangleq x \mapsto v \ast (v = 1 \Rightarrow [u]^L_r) \]
  - Meaning of “fulfilment” strictly controlled by the protocol:
    \[ w : (x,0), u \leadsto (x,1) \text{ is the only way to fulfil } u. \]
Liveness Invariants

\[ \{ x \mapsto 0 \} \]

\[ \{ \text{sh}(x, 0) \ast [w] \ast [u]^L \} \]

\[ \{ \text{sh}(x, 1) \ast [w] \} \]

\[ \{ \text{True} \} \]

\[ \{ (\text{sh}(x, 0) \ast [u]^E) \lor \text{sh}(x, 1) \} \]

\[ \text{do} \{ \]
\[ \quad d := [x] \]
\[ \} \text{ while}(\neg d) \]

\[ \exists v. \text{sh}(x, v) \]
Rule for parallel composition checks obligations are fulfilled.
TaDA Live’s contributions

TaDA Live’s innovations:

1. **Subjective Obligations**
   Thread-local reasoning with liveness invariants

2. **Obligation layers**
   Compositional deadlock-freedom

3. **Logical atomicity for blocking code**
   Enabling modular reasoning
Obligation layers

⚠️ Deadlock

\[
\begin{align*}
\text{do} & \ {\{ \\
& \quad d_1 := [y] \\
& \quad \text{while}(d_1 \neq 1) \\
& \quad [x] := 1
\}} \ \text{while}(d_1 \neq 1) \\
& \quad \text{do} & \ {\{ \\
& \quad d_2 := [x] \\
& \quad \text{while}(d_2 \neq 1) \\
& \quad [y] := 1
\}}
\end{align*}
\]
Obligation layers

⚠️ Deadlock

```plaintext
do {
    d_1 := [y]
} while(d_1 ≠ 1)
[x] := 1
```

```plaintext
do {
    d_2 := [x]
} while(d_2 ≠ 1)
[y] := 1
```

Attempt at a proof:
- $u_x$ obligation to set $x$ to 1
- $u_y$ obligation to set $y$ to 1
Obligation layers

Deadlock

\[\begin{align*}
\text{do} & \{ \\
& \quad d_1 := [y] \\
& \quad \text{while}(d_1 \neq 1) \\
& \quad [x] := 1 \\
\} & \quad \text{while}(d_2 \neq 1) \\
& \quad [y] := 1
\end{align*}\]

Attempt at a proof:

- \(u_x\) obligation to set \(x\) to 1
- \(u_y\) obligation to set \(y\) to 1
Obligation layers

⚠️ Deadlock $\Rightarrow$ unsound circular reasoning

Assumes $\square \Diamond u_y$
while holding $u_x$
continuously

```
\begin{align*}
\text{do } & \{ \\
& \hspace{1em} d_1 := [y] \\
& \hspace{1em} [x] := 1 \\
\} \text{ while}(d_1 \neq 1)
\end{align*}
```

```
\begin{align*}
\text{do } & \{ \\
& \hspace{1em} d_2 := [x] \\
& \hspace{1em} [y] := 1 \\
\} \text{ while}(d_2 \neq 1)
\end{align*}
```

Assumes $\square \Diamond u_x$
while holding $u_y$
continuously
Obligation layers

⚠️ Deadlock

Assumes □◇u_y while holding u_x continuously

\[
\begin{align*}
\text{do} & \{ \\
d_1 & := [y] \\
\} \text{ while}(d_1 \neq 1) \\
[x] & := 1
\end{align*}
\]

Assumes □◇u_x while holding u_y continuously

\[
\begin{align*}
\text{do} & \{ \\
d_2 & := [x] \\
\} \text{ while}(d_2 \neq 1) \\
y & := 1
\end{align*}
\]

TaDA Live’s solution:

- lay(u) ∈ L well-founded order
- lay(u_1) < lay(u_2) means fulfilling u_2 may depend on liveness invariant of u_1
Obligation layers

⚠️ Deadlock

Assumes □◇u_y while holding u_x continuously

\[ \text{lay}(u_y) < \text{lay}(u_x) \]

TaDA Live’s solution:

- \( \text{lay}(u) \in \mathcal{L} \) well-founded order
- \( \text{lay}(u_1) < \text{lay}(u_2) \) means fulfilling \( u_2 \) may depend on liveness invariant of \( u_1 \)
Environment liveness condition

![Diagram with logical statements and conditions]

- If $L$ then either:
  - $\text{sh}(x, 1)$, or
  - $[u]^E$ and fulfilling $u$ takes us to target

Layers control which liveness invariants are available here

- Protocol asserts: $(x, 1) \not\Rightarrow (x, 0)$

- $\square \Rightarrow \diamond (\square \text{sh}(x, 1))$

- $\square L \Rightarrow \square (\diamond \text{sh}(x, 1))$

- $\text{sh}(x, 1) \Rightarrow \square \text{sh}(x, 1)$
Obligation layers

\[ x := 1 \]
\[ \text{do } \{ \quad d_1 := y \}
\[ \text{while}(d_1 \neq 1) \]
\[ \text{do } \{ \quad d_2 := x \]
\[ \text{while}(d_2 \neq 1) \]
\[ y := 1 \]

Assumes $\Box \Diamond u_x$
while holding $u_y$
continuously

\[ \downarrow \]

\[ \text{lay}(u_x) < \text{lay}(u_y) \]

TaDA Live’s solution:
- $\text{lay}(u) \in L$ well-founded order
- $\text{lay}(u_1) < \text{lay}(u_2)$ means
  fulfilling $u_2$ may depend on liveness
  invariant of $u_1$
TaDA Live’s contributions

TaDA Live’s innovations:

1. **Subjective Obligations**
   Thread-local reasoning with liveness invariants

2. **Obligation layers**
   Compositional deadlock-freedom

3. **Logical atomicity for blocking code**
   Enabling modular reasoning
Logical atomicity:

- specs for code that behaves as if atomic (~linearizable)
- enables modularity without losing precision
- TaDA Live first logic with total logical atomic specs for blocking code
Example: specification of a lock

\[ \vdash \forall v \in \{0, 1\}. \langle L(x, v) \rangle \text{lock}(x) \langle L(x, 1) \land v = 0 \rangle \]
Example: specification of a lock

\[ \vdash \forall v \in \{0, 1\}. \langle L(x, v) \rangle \text{lock}(x) \langle L(x, 1) \land v = 0 \rangle \]
Example: specification of a lock

\[ \Gamma \vdash \forall v \in \{0, 1\}. \langle L(x, v) \rangle \text{lock}(x) \langle L(x, 1) \land v = 0 \rangle \quad \text{total?!?} \]
Example: specification of a lock

\[ \vdash \forall v \in \{0, 1\} \rightarrow \{0\}. \langle L(x, v) \rangle \text{lock}(x) \langle L(x, 1) \land v = 0 \rangle \]

Liveness invariant \( \square \diamond v=0 \)
(responsibility of client)
Example: specification of a lock

\[ \vdash \forall v \in \{0, 1\} \rightarrow \{0\}. \langle L(x, v) \rangle \ \text{lock}(x) \ \langle L(x, 1) \land v = 0 \rangle \]

TaDA Live can:

- verify fine-grained implementations against the spec
  - the implementation proof can *make use* of the liveness invariant to establish termination
- use the spec to verify strong specs of clients
  - the client can use client-side obligations to *discharge* the liveness invariant
Related work

- **Total TaDA (non-blocking only)**
  [da Rocha Pinto, Dinsdale-Young, Gardner, Sutherland’16]

- **Built-in blocking primitives (no busy-waiting):**
  [Kobayashi’06] [Boström, Müller’15] [Leino, Müller, Smans’10]
  [Hamin, Jacobs’18 & ’19] [Jacobs, Bosnacki, Kuiper’18]

- **LiLi** [Liang, Feng’16 & ’18]
  - Logic to prove linearizability by *progress-preserving* contextual refinement
  - No client reasoning within the logic, no rule for parallel
  - Atomic operations might be specified using non-atomic code

- [Reinhard, Jacobs’21] concurrent independent work
  (restricted form of busy-waiting, no logical atomicity)
Future work

Iris can already prove termination for:

- Some first-order *non-blocking* programs

Mechanization challenges:

- Step indexing vs liveness
  - Transfinite Iris
  - Higher-order patterns unexplored
- Non-affine obligations
  - Iron-style trackable resources?
There is so much more in the 84-page paper!

- In-depth motivation of design
- Formalisation of the model
- Full proof system with illustrative examples
- Several realistic case studies
- Soundness argument
- More related work

D’Osualdo, Sutherland, Farzan, Gardner

TaDA Live: Compositional Reasoning for Termination of Fine-grained Concurrent Programs

TOPLAS 2021 — https://doi.org/10.1145/3477082
Thank you!