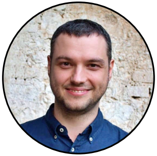
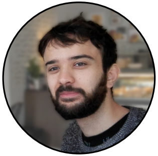
TaDA Live: Compositional Reasoning for Termination of Fine-grained Concurrent Programs



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

The TOPLAS paper

D'Osualdo, Sutherland, Farzan, Gardner
TaDA Live: Compositional Reasoning for Termination of Fine-grained Concurrent Programs
TOPLAS 2021 — `https://doi.org/10.1145/3477082`

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

`//...`
`[x] := 1`
`//...`
	`//...`
	`do {`
	`  d := [x]`
	`} while(d ≠ 1)`
	`//...`

Program features of interest:

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

Busy-waiting

	‖	`//...`
`//...`	‖	**do** {
$[x] := 1$	‖	$\quad d := [x]$
`//...`	‖	} **while**$(d \neq 1)$
	‖	`//...`

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

```
//...                 ||   //...
[x] := 1              ||   do {
//...                 ||     d := [x]
                      ||   } while(d ≠ 1)
                      ||   //...
```

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

`//...`
`[x] := 1`
`//...`
	`//...`
	`do` `{`
	`  d := [x]`
	`}` **`while`**`(d ≠ 1)`
	`//...`

Properties of interest:

- Functional correctness
- Termination guarantees

```
            ||  //...
//...       ||  do {
[x] := 1    ||    d := [x]
//...       ||  } while(d ≠ 1)
            ||  //...
```

Compositionality is the main challenge:

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

What should specifications look like?

The challenge

$\{\mathsf{x} \mapsto 0\}$

```
//...                //...
[x] := 1             do {
//...          ||      d := [x]
                     } while(d ≠ 1)
                     //...
```

$\{\mathsf{True}\}$

Total Hoare triples:

$\vdash \{P\}\ \mathbb{C}\ \{Q\}$ total

$\mathbb{C}$ run from a state satisfying P **always terminates** in a state satisfying Q.

The challenge

$\{x \mapsto 0\}$

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

```
//...
do {
  d := [x]
} while(d ≠ 1)
//...
```

$\{\mathrm{True}\}$

Total Hoare triples:

$\vdash \{P\}\ \mathbb{C}\ \{Q\}$ total

$\mathbb{C}$ run from a state satisfying P **always terminates** in a state satisfying Q.

$\vdash \{\ x \mapsto 1\ \}$ `do {d := [x]} while(d ≠ 1)` $\{\mathrm{True}\}$ total

The challenge

$\{x \mapsto 0\}$

```
//...                 //...
[x] := 1              do {
//...                   d := [x]
                      } while(d ≠ 1)
                      //...
```

$\{\mathsf{True}\}$

Total Hoare triples:

$\vdash \{P\}\ \mathbb{C}\ \{Q\}\ \mathsf{total}$

$\mathbb{C}$ run from a state satisfying P **always terminates** in a state satisfying Q.

$\vdash \{\mathrm{share}(x)\}$ `do {d := [x]} while(d ≠ 1)` $\{\mathsf{True}\}\ \mathsf{total}$

The challenge

$\{\mathsf{x} \mapsto 0\}$

```
//...              //...
[x] := 1           do {
//...                d := [x]
                   } while(d ≠ 1)
                   //...
```

$\{\mathsf{True}\}$

Total Hoare triples:

$\vdash \{P\}\ \mathbb{C}\ \{Q\}\ \mathsf{total}$

$\mathbb{C}$ run from a state satisfying P **always terminates** in a state satisfying Q.

$\cancel{\vdash \{\mathsf{share}(\mathsf{x})\}\ \mathtt{do}\ \{\mathsf{d} := [\mathsf{x}]\}\ \mathtt{while}(\mathsf{d} \neq 1)\ \{\mathsf{True}\}\ \mathsf{total}}$

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

The challenge

$\{x \mapsto 0\}$

```
//...                  //...
[x] := 1           ||  do {
//...              ||    d := [x]
                       } while(d ≠ 1)
                       //...
```

$\{\text{True}\}$

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

Total Hoare triples:

$\vdash \{P\}\ \mathbb{C}\ \{Q\}$ total

$\mathbb{C}$ run from a state satisfying P **always terminates** in a state satisfying Q.

~~$\vdash \{\text{share}(x)\}$ `do {d := [x]} while(d ≠ 1)` $\{\text{True}\}$ total~~

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

Liveness invariants

$\{x \mapsto 0\}$

```
//...                //...
[x] := 1             do {
//...                  d := [x]
                     } while(d ≠ 1)
                     //...
```

$\{\mathsf{True}\}$

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

TaDA Live's starting observation:

Blocking = termination conditional on *liveness invariants*

$\{x \mapsto 0\}$

```
//...            //...
[x] := 1         do {
//...              d := [x]
                 } while(d ≠ 1)
                 //...
```

$\{\mathrm{True}\}$

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

TaDA Live's starting observation:

Blocking = termination conditional on *liveness invariants*

$\square\,(\Diamond$ "good state")

Always *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

$$\{x \mapsto 0\}$$

$$\{\quad\quad\quad * \quad\quad\quad\}$$

```
[x] := 1    ||    do {
                    d := [x]
                  } while(d ≠ 1)
```

$$\{\quad\quad\quad * \quad\quad\quad\}$$

$$\{\mathrm{True}\}$$

Rely/Guarantee in TaDA

$$\{x \mapsto 0\}$$

$$\{\ \mathbf{sh}(x, 0) * \lceil w \rceil \quad * \quad \exists v.\, \mathbf{sh}(x, v)\ \}$$

$[x] := 1 \quad \Big\| \quad$ **do** { d := [x] } **while**(d $\neq$ 1)

$$\{\ \mathbf{sh}(x, 1) * \lceil w \rceil \quad * \quad \exists v.\, \mathbf{sh}(x, v)\ \}$$

$$\{\mathrm{True}\}$$

Protocol:

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

Allowed updates of **sh**:

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

(w is write permission)

Rely/Guarantee in TaDA

$$\{x \mapsto 0\}$$

$$\{\ \mathbf{sh}(x, 0) * \lceil w \rceil \quad * \quad \exists v.\, \mathbf{sh}(x, v)\ \}$$

$$[x] := 1 \quad \Big\| \quad \texttt{do}\ \{\ d := [x]\ \}\ \texttt{while}(d \neq 1)$$

$$\{\ \mathbf{sh}(x, 1) * \lceil w \rceil \quad * \quad \exists v.\, \mathbf{sh}(x, v)\ \}$$

$$\{\mathsf{True}\}$$

Protocol:

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

Allowed updates of **sh**:

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

(w is write permission)

The standard (total) while rule:

Loop invariant

$$\frac{\forall \beta.\ \vdash \{P(\beta) \wedge \mathbb{B}\}\ \mathbb{C}\ \{\exists \beta'. P(\beta') \wedge \beta' < \beta\}}{\vdash \{P(\_)\}\ \texttt{while}(\mathbb{B})\{\mathbb{C}\}\ \{P(\_) \wedge \neg \mathbb{B}\}}$$

Rely/Guarantee in TaDA

$$\{x \mapsto 0\}$$

$$\left\{\ \mathbf{sh}(x,0) * \lceil w \rceil\ \ * \quad \exists v.\,\mathbf{sh}(x,v)\ \right\}$$

```
[x] := 1    ||    do {
            ||      d := [x]
            ||    } while(d ≠ 1)
```

$$\left\{\ \mathbf{sh}(x,1) * \lceil w \rceil\ \ * \quad \exists v.\,\mathbf{sh}(x,v)\ \right\}$$

$$\{\mathrm{True}\}$$

Protocol:

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

Allowed updates of **sh**:

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

(w is write permission)

The standard (total) while rule:

Variant
Upper bound on amount of work to do

Well founded
Amount of work decreases

$$\frac{\forall \beta.\ \vdash \{P(\beta) \land \mathbb{B}\}\ \mathbb{C}\ \{\exists \beta'. P(\beta') \land \beta' < \beta\}}{\vdash \{P(\_)\}\ \texttt{while}(\mathbb{B})\{\mathbb{C}\}\ \{P(\_) \land \neg\mathbb{B}\}}$$

Rely/Guarantee in TaDA

$$\{x \mapsto 0\}$$

$$\{\ \mathbf{sh}(x,0) * \lceil w \rceil \quad * \quad \exists v.\, \mathbf{sh}(x,v)\ \}$$

```
[x] := 1     ||   do {
             ||     d := [x]
             ||   } while(d ≠ 1)
```

$$\{\ \mathbf{sh}(x,1) * \lceil w \rceil \quad * \quad \exists v.\, \mathbf{sh}(x,v)\ \}$$

$$\{\mathrm{True}\}$$

Protocol:

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

Allowed updates of **sh**:

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

(w is write permission)

TaDA Live's while rule:

$$\frac{\forall \beta.\ \ \vdash \{P(\beta) \qquad \land \mathbb{B}\}\ \mathbb{C}\ \{\exists \beta'.\, P(\beta') \land \beta' \leq \beta\}}{\vdash \{P(\_)\}\ \texttt{while}(\mathbb{B})\{\mathbb{C}\}\ \{P(\_) \land \neg\mathbb{B}\}}$$

Rely/Guarantee in TaDA

$$\{x \mapsto 0\}$$

$$\{\ \mathbf{sh}(x,0) * \lceil w \rceil \quad * \quad \exists v.\, \mathbf{sh}(x,v)\ \}$$

`[x] := 1`	‖	`do {`
		`  d := [x]`
		`} while(d ≠ 1)`

$$\{\ \mathbf{sh}(x,1) * \lceil w \rceil \quad * \quad \exists v.\, \mathbf{sh}(x,v)\ \}$$

$$\{\mathrm{True}\}$$

Protocol:

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

Allowed updates of **sh**:

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

(w is write permission)

Target state T:

$\mathbf{sh}(x,1)$

TaDA Live's while rule:

$$\frac{\begin{array}{l}\forall\beta.\ \vdash \{P(\beta) * T \land \mathbb{B}\}\ \mathbb{C}\ \{\exists\beta'.\, P(\beta') \land \beta' < \beta\} \\ \forall\beta.\ \vdash \{P(\beta) \land \mathbb{B}\}\ \mathbb{C}\ \{\exists\beta'.\, P(\beta') \land \beta' \leq \beta\}\end{array}}{\vdash \{P(\_)\}\ \mathtt{while}(\mathbb{B})\{\mathbb{C}\}\ \{P(\_) \land \neg\mathbb{B}\}}$$

Rely/Guarantee in TaDA

$$\{x \mapsto 0\}$$

$$\{\ \mathbf{sh}(x,0) * \lceil w \rceil\ \ *\ \ \exists v.\,\mathbf{sh}(x,v)\ \}$$

```
[x] := 1    ||    do {
                    d := [x]
                  } while(d ≠ 1)
```

$$\{\ \mathbf{sh}(x,1) * \lceil w \rceil\ \ *\ \ \exists v.\,\mathbf{sh}(x,v)\ \}$$

$$\{\mathrm{True}\}$$

Protocol:

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

Allowed updates of **sh**:

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

(w is write permission)

Target state T:

$\mathbf{sh}(x,1)$

TaDA Live's while rule:

Env. during the loop: $\Diamond(\Box T)$

$$\frac{\begin{array}{l} \forall \beta.\ \vdash \{P(\beta) * T \land \mathbb{B}\}\ \mathbb{C}\ \{\exists \beta'.\,P(\beta') \land \beta' < \beta\} \\ \forall \beta.\ \vdash \{P(\beta) \land \mathbb{B}\}\ \mathbb{C}\ \{\exists \beta'.\,P(\beta') \land \beta' \leq \beta\} \end{array}}{\vdash \{P(\_)\}\ \mathtt{while}(\mathbb{B})\{\mathbb{C}\}\ \{P(\_) \land \neg\mathbb{B}\}}$$

Rely/Guarantee in TaDA

$$\{\mathsf{x} \mapsto 0\}$$

$$\{\ \mathbf{sh}(\mathsf{x}, 0) * \lceil \mathsf{w} \rceil\ \ * \quad \exists v.\, \mathbf{sh}(\mathsf{x}, v)\ \}$$

```
[x] := 1    ||    do {
                    d := [x]
                  } while(d ≠ 1)
```

$$\{\ \mathbf{sh}(\mathsf{x}, 1) * \lceil \mathsf{w} \rceil\ \ * \quad \exists v.\, \mathbf{sh}(\mathsf{x}, v)\ \}$$

$$\{\mathsf{True}\}$$

Protocol:

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

Allowed updates of **sh**:

$\mathsf{w} : (\mathsf{x}, 0) \rightsquigarrow (\mathsf{x}, 1)$

(w is write permission)

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

TaDA Live's while rule:

Env. during the loop

$\forall \beta. \quad \vdash \{P(\beta) * T$

$\forall \beta. \quad \vdash \{P(\beta)\}$

$\vdash \{P(\_)\}$ wh

$\Diamond (\Box T)$

Env. Liveness Condition

Here as LTL *only for illustration*

Liveness Invariants

Proving $\Diamond\,(\Box\,\mathbf{sh}(\mathrm{x}, 1))$

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

Protocol:

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

Allowed updates of **sh**:

$\mathrm{w} : (\mathrm{x}, 0) \leadsto (\mathrm{x}, 1)$

(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:

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

- Subjective assertions:

$$\lfloor \mathsf{u} \rfloor^{\mathsf{L}} = \text{'I own } \mathsf{u}\text{'} \qquad \lfloor \mathsf{u} \rfloor^{\mathsf{E}} = \text{'I know env. owns } \mathsf{u}\text{'}$$

- $\vdash \{P\}\ \mathbb{C}\ \{Q\} \approx$ **If** $\Box(\lfloor \mathsf{u} \rfloor^{\mathsf{E}} \Rightarrow \Diamond(\mathsf{u}\ \text{fulfilled}))$ **then** $\mathbb{C}$ terminates

Liveness Invariants

$\{x \mapsto 0\}$

$\left\{\mathbf{sh}(x,0) * \lceil w \rceil\right\}$

```
[x] := 1
```

$\{\mathbf{sh}(x,1) * \lceil w \rceil\}$

$\left\{\begin{array}{l}\mathbf{sh}(x,0)\\ \quad \lor \mathbf{sh}(x,1)\end{array}\right\}$

```
do {
  d := [x]
} while(d ≠ 1)
```

$\{\exists v.\, \mathbf{sh}(x,v)\}$

$\{\mathrm{True}\}$

Protocol:

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

Allowed updates of **sh**:

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

(w is write permission)

Liveness Invariants

$$\{x \mapsto 0\}$$

$$\left\{\begin{array}{l}\mathbf{sh}(x,0) * \lceil w \rceil \\ \quad * \lfloor u \rfloor^{L}\end{array}\right\} \qquad \left\{\begin{array}{l}\mathbf{sh}(x,0) * \lfloor u \rfloor^{E} \\ \quad \vee\ \mathbf{sh}(x,1)\end{array}\right\}$$

```
[x] := 1          ||   do {
                  ||     d := [x]
                  ||   } while(d ≠ 1)
```

$$\{\mathbf{sh}(x,1) * \lceil w \rceil\} \qquad \{\exists v.\, \mathbf{sh}(x,v)\}$$

$$\{\mathrm{True}\}$$

TaDA Live Protocol:

u obligation

Allowed updates of **sh**:

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

The update fulfils u

Liveness Invariants

$$\{x \mapsto 0\}$$

$$\left\{\begin{array}{l}\mathbf{sh}(x,0) * \lceil w \rceil \\ \quad * \lfloor u \rfloor^{L}\end{array}\right\} \quad \Bigg\| \quad \left\{\begin{array}{l}\mathbf{sh}(x,0) * \lfloor u \rfloor^{E} \\ \quad \vee\ \mathbf{sh}(x,1)\end{array}\right\}$$

```
[x] := 1        ||   do {
                ||     d := [x]
                ||   } while(d ≠ 1)
```

$$\{\mathbf{sh}(x,1) * \lceil w \rceil\} \quad \Bigg\| \quad \{\exists v.\ \mathbf{sh}(x,v)\}$$

$$\{\mathrm{True}\}$$

TaDA Live Protocol:

u obligation

Allowed updates of **sh**:
$w : (x,0), u \rightsquigarrow (x,1)$

The update fulfils u

$$\square L \Rightarrow \lozenge(\square T)$$

$$\frac{\begin{array}{l}\forall\beta.\ \vdash \{P(\beta) * T \wedge \mathbb{B}\}\ \mathbb{C}\ \{\exists\beta'.\ P(\beta') \wedge \beta' < \beta\} \\ \forall\beta.\ \vdash \{P(\beta) \wedge \mathbb{B}\}\ \mathbb{C}\ \{\exists\beta'.\ P(\beta') \wedge \beta' \leq \beta\}\end{array}}{\vdash \{P(\_) * L\}\ \mathtt{while}(\mathbb{B})\{\mathbb{C}\}\ \{P(\_) * L \wedge \neg\mathbb{B}\}}$$

Liveness Invariants

$$\{x \mapsto 0\}$$

$$\left\{\begin{array}{l}\mathbf{sh}(x,0) * \lceil w \rceil \\ \quad * \lfloor u \rfloor^{L}\end{array}\right\} \quad \Bigg\| \quad \left\{\begin{array}{l}\mathbf{sh}(x,0) * \lfloor u \rfloor^{E} \\ \quad \vee\ \mathbf{sh}(x,1)\end{array}\right\}$$

```
[x] := 1          ||   do {
                  ||     d := [x]
                  ||   } while(d ≠ 1)
```

$$\{\mathbf{sh}(x,1) * \lceil w \rceil\} \quad \Big\| \quad \{\exists v.\ \mathbf{sh}(x,v)\}$$

$$\{\mathrm{True}\}$$

TaDA Live Protocol:

u obligation

Allowed updates of **sh**:

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

The update fulfils u

Target state T:

$\mathbf{sh}(x,1)$

$$\Box L \Rightarrow \Diamond(\Box T)$$

$$\frac{\begin{array}{l}\forall \beta.\ \vdash \{P(\beta) * T \wedge \mathbb{B}\}\ \mathbb{C}\ \{\exists \beta'.\, P(\beta') \wedge \beta' < \beta\} \\ \forall \beta.\ \vdash \{P(\beta) \wedge \mathbb{B}\}\ \mathbb{C}\ \{\exists \beta'.\, P(\beta') \wedge \beta' \leq \beta\}\end{array}}{\vdash \{P(\_) * L\}\ \mathtt{while}(\mathbb{B})\{\mathbb{C}\}\ \{P(\_) * L \wedge \neg\mathbb{B}\}}$$

Liveness Invariants

$$\{x \mapsto 0\}$$

$$\left\{\begin{array}{r}\mathbf{sh}(x,0) * [w] \\ * \lfloor u \rfloor^{L}\end{array}\right\} \quad \Bigg\| \quad \left\{\begin{array}{c}\mathbf{sh}(x,0) * \lfloor u \rfloor^{E} \\ \lor\ \mathbf{sh}(x,1)\end{array}\right\}$$

```
[x] := 1        ||   do {
                ||     d := [x]
                ||   } while(d ≠ 1)
```

$$\{\mathbf{sh}(x,1) * [w]\} \quad \| \quad \{\exists v.\ \mathbf{sh}(x,v)\}$$

$$\{\mathrm{True}\}$$

TaDA Live Protocol:

u obligation

Allowed updates of **sh**:

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

The update fulfils u

During the loop L:

$(\mathbf{sh}(x,0) * \lfloor u \rfloor^{E}) \lor \mathbf{sh}(x,1)$

Target state T:

$\mathbf{sh}(x,1)$

$$\Box L \Rightarrow \Diamond (\Box T)$$

$$\frac{\begin{array}{l}\forall \beta.\ \vdash \{P(\beta) * T \land \mathbb{B}\}\ \mathbb{C}\ \{\exists \beta'.\ P(\beta') \land \beta' < \beta\} \\ \forall \beta.\ \vdash \{P(\beta) \land \mathbb{B}\}\ \mathbb{C}\ \{\exists \beta'.\ P(\beta') \land \beta' \leq \beta\}\end{array}}{\vdash \{P(\_) * L\}\ \mathtt{while}(\mathbb{B})\{\mathbb{C}\}\ \{P(\_) * L \land \neg \mathbb{B}\}}$$

Environment liveness condition

$$\Box L \Rightarrow \Diamond (\Box \mathbf{sh}(\mathsf{x}, 1))$$

Environment liveness condition

$$\dfrac{\dfrac{}{\Box\, L \Rightarrow \Box\,(\Diamond\, \mathbf{sh}(\mathsf{x}, 1))} \qquad \dfrac{}{\mathbf{sh}(\mathsf{x}, 1) \Rightarrow \Box\, \mathbf{sh}(\mathsf{x}, 1)}}{\Box\, L \Rightarrow \Diamond\,(\Box\, \mathbf{sh}(\mathsf{x}, 1))}$$

Environment liveness condition

Stability

Protocol asserts:
$(\mathsf{x}, 1) \not\rightsquigarrow (\mathsf{x}, 0)$

$$\dfrac{\dfrac{}{\Box\, L \Rightarrow \Box\,(\Diamond\, \mathbf{sh}(\mathsf{x}, 1))} \qquad \dfrac{}{\mathbf{sh}(\mathsf{x}, 1) \Rightarrow \Box\, \mathbf{sh}(\mathsf{x}, 1)}}{\Box\, L \Rightarrow \Diamond\,(\Box\, \mathbf{sh}(\mathsf{x}, 1))}$$

Environment liveness condition

Environ. liveness

If L then either:

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

Stability

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

$$\dfrac{\dfrac{\text{Environ. liveness}}{\Box L \Rightarrow \Box(\Diamond \mathbf{sh}(x, 1))} \qquad \dfrac{\text{Stability}}{\mathbf{sh}(x, 1) \Rightarrow \Box \mathbf{sh}(x, 1)}}{\Box L \Rightarrow \Diamond(\Box \mathbf{sh}(x, 1))}$$

Environment liveness condition

$$L \triangleq (\mathbf{sh}(x, 0) * \lfloor u \rfloor^{E}) \vee \mathbf{sh}(x, 1)$$

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

Environ. liveness

If L then either:

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

Stability

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

$$\dfrac{\dfrac{\text{Environ. liveness}}{\Box L \Rightarrow \Box(\Diamond \mathbf{sh}(x, 1))} \qquad \dfrac{\text{Stability}}{\mathbf{sh}(x, 1) \Rightarrow \Box \mathbf{sh}(x, 1)}}{\Box L \Rightarrow \Diamond(\Box \mathbf{sh}(x, 1))}$$

Environment liveness condition

$$L \triangleq (\mathbf{sh}(\mathrm{x}, 0) * \lfloor \mathsf{u} \rfloor^{\mathsf{E}}) \vee \mathbf{sh}(\mathrm{x}, 1)$$

$$\mathsf{w} : (\mathrm{x}, 0), \mathsf{u} \rightsquigarrow (\mathrm{x}, 1)$$

Environ. liveness

If L then either:

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

Stability

Protocol asserts: $(\mathrm{x}, 1) \not\rightsquigarrow (\mathrm{x}, 0)$

$$\dfrac{\dfrac{\text{Environ. liveness} \qquad \Box(\Diamond\, \mathsf{u}\ \text{fulfilled})}{\Box L \Rightarrow \Box(\Diamond\, \mathbf{sh}(\mathrm{x}, 1))} \qquad \dfrac{\text{Stability}}{\mathbf{sh}(\mathrm{x}, 1) \Rightarrow \Box\, \mathbf{sh}(\mathrm{x}, 1)}}{\Box L \Rightarrow \Diamond(\Box\, \mathbf{sh}(\mathrm{x}, 1))}$$

While rule

Environ. liveness

If L then either:

- T holds, or
- $\lfloor \mathsf{u} \rfloor^{\mathsf{E}}$ and fulfilling u takes us to T

Stability

$$T \Rightarrow \Box T$$

$$\frac{\begin{array}{l}\forall \beta.\ \vdash \{P(\beta) * T \wedge \mathbb{B}\}\ \mathbb{C}\ \{\exists \beta'.\, P(\beta') \wedge \beta' < \beta\} \\ \forall \beta.\ \vdash \{P(\beta) \wedge \mathbb{B}\}\ \mathbb{C}\ \{\exists \beta'.\, P(\beta') \wedge \beta' \leq \beta\}\end{array}}{\vdash \{P(\_) * L\}\ \mathtt{while}(\mathbb{B})\{\mathbb{C}\}\ \{P(\_) * L \wedge \neg \mathbb{B}\}}$$

Obligation fulfilment

How can a thread "fulfil" an obligation?

- To fulfil $\mathtt{u}$ = To go from owning, to not owning $\lfloor \mathtt{u} \rfloor^{\mathsf{L}}$

Obligation fulfilment

How can a thread "fulfil" an obligation?

- To fulfil U = To go from owning, to not owning $\lfloor \mathsf{U} \rfloor^{\mathsf{L}}$
- $\lfloor \mathsf{U} \rfloor^{\mathsf{L}}$ cannot be affine: $\lfloor \mathsf{U} \rfloor^{\mathsf{L}} \not\Rightarrow \mathrm{emp}$

Obligation fulfilment

How can a thread "fulfil" an obligation?

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

$$\lfloor u \rfloor^{L} \Rrightarrow \boxed{\lfloor u \rfloor^{L}} \Rrightarrow \text{emp}$$

Obligation fulfilment

How can a thread "fulfil" an obligation?

- To fulfil U = To go from owning, to not owning $\lfloor \mathsf{U} \rfloor^{\mathsf{L}}$
- $\lfloor \mathsf{U} \rfloor^{\mathsf{L}}$ cannot be affine: $\lfloor \mathsf{U} \rfloor^{\mathsf{L}} \not\Rightarrow \mathsf{emp}$
- Issue with non-affine resources and invariants:

$$\lfloor \mathsf{U} \rfloor^{\mathsf{L}} \Rrightarrow \boxed{\lfloor \mathsf{U} \rfloor^{\mathsf{L}}} \Rightarrow \mathsf{emp}$$

- In TaDA Live: that's not a bug, it's a feature!

Obligation fulfilment

How can a thread "fulfil" an obligation?

- To fulfil U = To go from owning, to not owning $\lfloor \mathsf{U} \rfloor^{\mathsf{L}}$
- $\lfloor \mathsf{U} \rfloor^{\mathsf{L}}$ cannot be affine: $\lfloor \mathsf{U} \rfloor^{\mathsf{L}} \not\Rightarrow \mathsf{emp}$
- Issue with non-affine resources and invariants:

$$\lfloor \mathsf{U} \rfloor^{\mathsf{L}} \Rrightarrow \boxed{\lfloor \mathsf{U} \rfloor^{\mathsf{L}}} \Rightarrow \mathsf{emp}$$

- In TaDA Live: that's not a bug, it's a feature!
  - Obligations "belong" to an invariant (named r): $\lfloor \mathsf{U} \rfloor^{\mathsf{L}}_{r}$

Obligation fulfilment

How can a thread "fulfil" an obligation?

- To fulfil U = To go from owning, to not owning $\lfloor \mathsf{U} \rfloor^{\mathsf{L}}$
- $\lfloor \mathsf{U} \rfloor^{\mathsf{L}}$ cannot be affine: $\lfloor \mathsf{U} \rfloor^{\mathsf{L}} \not\Rightarrow \mathrm{emp}$
- Issue with non-affine resources and invariants:

$$\lfloor \mathsf{U} \rfloor^{\mathsf{L}} \Rrightarrow \boxed{\lfloor \mathsf{U} \rfloor^{\mathsf{L}}} \Rightarrow \mathrm{emp}$$

- In TaDA Live: that's not a bug, it's a feature!
  - Obligations "belong" to an invariant (named r): $\lfloor \mathsf{U} \rfloor^{\mathsf{L}}_{r}$
  - Invariants can only own obligations that belong to them:

$$\mathcal{I}(\mathbf{sh}_{r}(\mathsf{x}, v)) \triangleq \mathsf{x} \mapsto v * (v = 1 \mathrel{\dot{\Rightarrow}} \lfloor \mathsf{U} \rfloor^{\mathsf{L}}_{r})$$

Obligation fulfilment

How can a thread “fulfil” an obligation?

- To fulfil u = To go from owning, to not owning $\lfloor \mathsf{u} \rfloor^{\mathsf{L}}$
- $\lfloor \mathsf{u} \rfloor^{\mathsf{L}}$ cannot be affine: $\lfloor \mathsf{u} \rfloor^{\mathsf{L}} \not\Rrightarrow \mathrm{emp}$
- Issue with non-affine resources and invariants:

$$\lfloor \mathsf{u} \rfloor^{\mathsf{L}} \Rrightarrow \boxed{\lfloor \mathsf{u} \rfloor^{\mathsf{L}}} \Rrightarrow \mathrm{emp}$$

- In TaDA Live: that’s not a bug, it’s a feature!
  - Obligations “belong” to an invariant (named r): $\lfloor \mathsf{u} \rfloor^{\mathsf{L}}_{r}$
  - Invariants can only own obligations that belong to them:

$$\mathcal{I}(\mathbf{sh}_r(\mathsf{x}, v)) \triangleq \mathsf{x} \mapsto v * (v = 1 \mathrel{\dot{\Rightarrow}} \lfloor \mathsf{u} \rfloor^{\mathsf{L}}_{r})$$

  - Meaning of “fulfilment” strictly controlled by the protocol:

$\mathsf{w} : (\mathsf{x}, 0), \mathsf{u} \rightsquigarrow (\mathsf{x}, 1)$ is the only way to fulfil u.

Liveness Invariants

$$\{x \mapsto 0\}$$

$$\left\{\mathbf{sh}(x,0) * \lceil w \rceil * \lfloor u \rfloor^{L}\right\} \quad \Big\| \quad \left\{(\mathbf{sh}(x,0) * \lfloor u \rfloor^{E}) \vee \mathbf{sh}(x,1)\right\}$$

```
[x] := 1        ||   do {
                ||     d := [x]
                ||   } while(¬d)
```

$$\left\{\mathbf{sh}(x,1) * \lceil w \rceil\right\} \quad \Big\| \quad \left\{\exists v.\, \mathbf{sh}(x,v)\right\}$$

$$\{\mathrm{True}\}$$

Liveness Invariants

$$\{x \mapsto 0\}$$

$$
\begin{array}{l|l}
\{\mathbf{sh}(x,0) * \lceil w \rceil * \lfloor u \rfloor^{L}\} & \{(\mathbf{sh}(x,0) * \lfloor u \rfloor^{E}) \vee \mathbf{sh}(x,1)\} \\
 & \texttt{do \{} \\
\cancel{[x] := 1}\ \ \texttt{skip} & \quad d := [x] \\
 & \texttt{\} while}(\neg d) \\
\{\mathbf{sh}(x,0) * \lceil w \rceil * \lfloor u \rfloor^{L}\} & \{\exists v.\, \mathbf{sh}(x,v)\}
\end{array}
$$

$$\{\mathrm{True}\}$$

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

```
do {                   ||   do {
  d₁ := [y]            ||     d₂ := [x]
} while(d₁ ≠ 1)        ||   } while(d₂ ≠ 1)
[x] := 1               ||   [y] := 1
```

Obligation layers

⚠ Deadlock

```
do {                    ||    do {
  d1 := [y]             ||      d2 := [x]
} while(d1 ≠ 1)         ||    } while(d2 ≠ 1)
[x] := 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

Assumes $\Box\Diamond u_y$ while holding u_x continuously	do { $d_1 := [y]$ } while($d_1 \neq 1$) $[x] := 1$	‖	do { $d_2 := [x]$ } while($d_2 \neq 1$) $[y] := 1$	Assumes $\Box\Diamond u_x$ while holding u_y continuously

```
do {
  d1 := [y]
} while(d1 ≠ 1)
[x] := 1
```

‖

```
do {
  d2 := [x]
} while(d2 ≠ 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 $\implies$ unsound circular reasoning

Assumes $\Box\Diamond \mathsf{u}_y$ while holding u_x continuously

```
do {
  d₁ := [y]
} while(d₁ ≠ 1)
[x] := 1
```

$\parallel$

```
do {
  d₂ := [x]
} while(d₂ ≠ 1)
[y] := 1
```

Assumes $\Box\Diamond \mathsf{u}_x$ while holding u_y continuously

Obligation layers

⚠ Deadlock

Assumes $\Box\Diamond \mathsf{u}_y$ while holding u_x continuously

```
do {
  d1 := [y]
} while(d1 ≠ 1)
[x] := 1
```

‖

```
do {
  d2 := [x]
} while(d2 ≠ 1)
[y] := 1
```

Assumes $\Box\Diamond \mathsf{u}_x$ while holding u_y continuously

TaDA Live's solution:

- $\mathrm{lay}(\mathsf{u}) \in \mathcal{L}$ well-founded order
- $\mathrm{lay}(\mathsf{u}_1) < \mathrm{lay}(\mathsf{u}_2)$ means fulfilling u_2 may depend on liveness invariant of u_1

Obligation layers

⚠ Deadlock

Assumes $\Box\Diamond \mathsf{u}_y$ while holding u_x continuously

$\Downarrow$

$\mathrm{lay}(\mathsf{u}_y) < \mathrm{lay}(\mathsf{u}_x)$

```
do {
  d1 := [y]
} while(d1 ≠ 1)
[x] := 1
```

$\|$

```
do {
  d2 := [x]
} while(d2 ≠ 1)
[y] := 1
```

Assumes $\Box\Diamond \mathsf{u}_x$ while holding u_y continuously

$\Downarrow$

$\mathrm{lay}(\mathsf{u}_x) < \mathrm{lay}(\mathsf{u}_y)$

$\lightning$

TaDA Live's solution:

- $\mathrm{lay}(\mathsf{u}) \in \mathcal{L}$ well-founded order
- $\mathrm{lay}(\mathsf{u}_1) < \mathrm{lay}(\mathsf{u}_2)$ means fulfilling u_2 may depend on liveness invariant of u_1

Environment liveness condition

Layers control which liveness invariants are available here

Environ. liveness

If L then either:

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

$\square\,(\lozenge\, \mathsf{u}\ \text{fulfilled})$

Stability

Protocol asserts: $(\mathrm{x}, 1) \not\rightsquigarrow (\mathrm{x}, 0)$

$$\dfrac{\dfrac{\text{Environ. liveness} \qquad \square\,(\lozenge\, \mathsf{u}\ \text{fulfilled})}{\square L \Rightarrow \square\,(\lozenge\, \mathbf{sh}(\mathrm{x}, 1))} \qquad \dfrac{\text{Stability}}{\mathbf{sh}(\mathrm{x}, 1) \Rightarrow \square\, \mathbf{sh}(\mathrm{x}, 1)}}{\square L \Rightarrow \lozenge\,(\square\, \mathbf{sh}(\mathrm{x}, 1))}$$

Obligation layers

```
[x] := 1                  ||   do {
do {                      ||     d₂ := [x]
  d₁ := [y]               ||   } while(d₂ ≠ 1)
} while(d₁ ≠ 1)           ||   [y] := 1
```

Assumes $\Box\Diamond \upsilon_x$ while holding υ_y continuously

$$\Downarrow$$

$$\mathrm{lay}(\upsilon_x) < \mathrm{lay}(\upsilon_y)$$

TaDA Live's solution:

- $\mathrm{lay}(\upsilon) \in \mathcal{L}$ well-founded order
- $\mathrm{lay}(\upsilon_1) < \mathrm{lay}(\upsilon_2)$ means fulfilling υ_2 may depend on liveness invariant of υ_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

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

$$\vdash \mathbb{V}v \in \{0,1\}.\left\langle \mathsf{L}(\mathrm{x}, v)\right\rangle \ \texttt{lock(x)}\ \left\langle \mathsf{L}(\mathrm{x}, 1) \land v = 0\right\rangle$$

Logical Atomicity

Example: specification of a lock

$$\vdash \mathbb{V} v \in \{0,1\} . \left\langle \mathsf{L}(\mathrm{x}, v) \right\rangle \ \texttt{lock(x)} \ \left\langle \mathsf{L}(\mathrm{x}, 1) \land v = 0 \right\rangle \qquad \checkmark$$

Example: specification of a lock

$$\vdash \mathbb{\forall} v \in \{0, 1\}.\, \left\langle \mathsf{L}(\mathrm{x}, v) \right\rangle \texttt{lock(x)} \left\langle \mathsf{L}(\mathrm{x}, 1) \land v = 0 \right\rangle \quad \textbf{total?!?}$$

Example: specification of a lock

$$\vdash \mathbb{V} v \in \{0,1\} \twoheadrightarrow \{0\}. \left\langle \mathsf{L}(\mathrm{x}, v) \right\rangle \texttt{lock(x)} \left\langle \mathsf{L}(\mathrm{x}, 1) \land v = 0 \right\rangle$$

Liveness invariant $\Box\Diamond v{=}0$
(responsibility of client)

Logical Atomicity

Example: specification of a lock

$$\vdash \mathbb{V} v \in \{0,1\} \twoheadrightarrow \{0}.\left\langle \mathsf{L}(\mathrm{x}, v) \right\rangle \ \texttt{lock(x)} \ \left\langle \mathsf{L}(\mathrm{x}, 1) \land v = 0 \right\rangle$$

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?

The TOPLAS paper

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!