Expressive modular verification of termination for busy-waiting programs

Work in progress

Justus Fasse and Bart Jacobs (KU Leuven)

Presented at the Iris Workshop 2023

1. Verifying deadlock-freedom
2. Verifying absence of infinite recursion
3. Verifying termination of busy-waiting programs
4. Modular specifications
    A. Logically atomic triples
    B. Total correctness logically atomic triples with liveness assumption
5. Conclusion

1. **Verifying deadlock-freedom**
2. Verifying absence of infinite recursion
3. Verifying termination of busy-waiting programs
4. Modular specifications
   A. Logically atomic triples
   B. Total correctness logically atomic triples with liveness assumption
5. Conclusion

Deadlock-freedom

```
let s = CreateSignal in
  Await s  ‖  SetSignal s
```

Deadlock-freedom

No one to wait on

```
let s = CreateSignal in
  Await s  ‖  ✗
```

Deadlock-freedom

Signals and obligations

```
let s = CreateSignal in
```

$\{\mathsf{obs}(\{s\}) * \mathsf{sig}(s,\ \ \mathit{false})\}$

$\boxed{\exists b.\, \mathsf{sig}(s,\ \ b)}$

$\{\mathsf{obs}(\emptyset)\}$	$\Vert$	$\{\mathsf{obs}(\{s\})\}$
`Await s`	$\Vert$	`SetSignal s`
$\{\mathsf{obs}(\emptyset)\}$	$\Vert$	$\{\mathsf{obs}(\emptyset)\}$

Deadlock-freedom

Circular dependencies

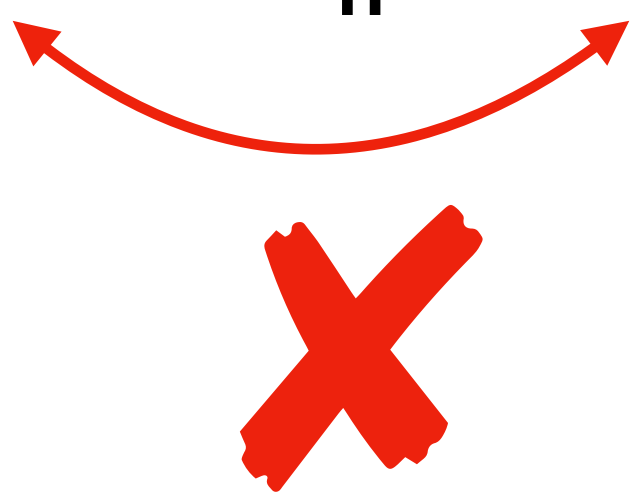
```
let s = CreateSignal in
let t = CreateSignal in
```

`Await s`	‖	`Await t`
`SetSignal t`	‖	`SetSignal s`

Deadlock-freedom

Circular dependencies

```
let s = CreateSignal in
let t = CreateSignal in
Await s       ‖ Await t
SetSignal t   ‖ SetSignal s
```



Deadlock-freedom

```
let s = CreateSignal () in
```

$\{\mathsf{obs}(\{s\}) * \mathsf{sig}(s, \ \ \mathit{false})\}$

$\boxed{\exists b.\, \mathsf{sig}(s, \ \ b)}$

$\{\mathsf{obs}(\emptyset)\}$	‖	$\{\mathsf{obs}(\{s\})\}$
`Await s`	‖	`SetSignal s`
$\{\mathsf{obs}(\emptyset)\}$	‖	$\{\mathsf{obs}(\emptyset)\}$

Deadlock-freedom

Levels

```
let s = CreateSignal ɭ in
```

$\{\mathsf{obs}(\{s\}) * \mathsf{sig}(s, \mathfrak{l}, \mathit{false})\}$

$\boxed{\exists b.\, \mathsf{sig}(s, \mathfrak{l}, b)}$

$\{\mathsf{obs}(\emptyset)\}$	$\Vert$	$\{\mathsf{obs}(\{s\})\}$
`Await s`	$\Vert$	`SetSignal s`
$\{\mathsf{obs}(\emptyset)\}$	$\Vert$	$\{\mathsf{obs}(\emptyset)\}$

1. Verifying deadlock-freedom
2. **Verifying absence of infinite recursion**
3. Verifying termination of busy-waiting programs
4. Modular specifications
   A. Logically atomic triples
   B. Total correctness logically atomic triples with liveness assumption
5. Conclusion

Absence of infinite recursion

```
(μ loop n.
  if n = 0 then ()
  else
   loop (n-1)) 10
```

Absence of infinite recursion

Call permissions

```
(μ loop n.
  if n = 0 then ()
  else

   burn δ in

   loop (n-1)) 10
```

Absence of infinite recursion

Call permissions

$\{n \cdot \mathsf{cp}(\delta)\}$

```
(μ loop n.
  if n = 0 then ()
  else
```

$\{n \cdot \mathsf{cp}(\delta)\}$

```
   burn δ in
```

$\{(n-1) \cdot \mathsf{cp}(\delta)\}$

```
   loop (n-1)) 10
```

Absence of infinite recursion

Call permissions

```
(μ loop n.
  if n = 0 then ()
  else

   burn δn receive δn-1 in

   loop (n-1)) 10
```

Absence of infinite recursion

Call permissions

```
{cp(δn)}
(μ loop n.
  if n = 0 then ()
  else
   {cp(δn)}
   burn δn receive δn-1 in
   {cp(δn-1)}
   loop (n-1)) 10
```

1. Verifying deadlock-freedom
2. Verifying absence of infinite recursion
3. **Verifying termination of busy-waiting programs**
4. Modular specifications
   A. Logically atomic triples
   B. Total correctness logically atomic triples with liveness assumption
5. Conclusion

$\{\mathsf{obs}(\emptyset)\}$

```
let s = CreateSignal ι in
```

$\{\mathsf{obs}(\{s\}) * \mathsf{sig}(s, \iota, \mathit{false})\}$

$\boxed{\exists b.\, \mathsf{sig}(s, \iota, b)}$

$\{\mathsf{obs}(\emptyset)\}$	‖	$\{\mathsf{obs}(\{s\})\}$
`Await s`	‖	`SetSignal s`
$\{\mathsf{obs}(\emptyset)\}$	‖	$\{\mathsf{obs}(\emptyset)\}$

Busy-waiting

$\{\mathsf{obs}(\emptyset)\}$

```
let s = CreateSignal ɩ in
```

$\{\mathsf{obs}(\{s\}) * \mathsf{sig}(s, \iota, \mathit{false})\}$

$\boxed{\exists b.\, \mathsf{sig}(s, \iota, b)}$

$\{\mathsf{obs}(\emptyset)\}$

```
(μ loop ().
  if !s then ()
  else
    loop ()) ()
```

$\{\mathsf{obs}(\emptyset)\}$

‖

$\{\mathsf{obs}(\{s\})\}$

```
SetSignal s
```

$\{\mathsf{obs}(\emptyset)\}$

Busy-waiting

$\{\mathsf{obs}(\emptyset)\}$

```
let s = CreateSignal ɩ in
```

$\{\mathsf{obs}(\{s\}) * \mathsf{sig}(s, \iota, \mathit{false})\}$

$\boxed{\exists b.\, \mathsf{sig}(s, \iota, b)}$

$\{\mathsf{obs}(\emptyset)\}$

```
(μ loop ().
  if !s then ()
  else
  ✗loop ()) ()
```

$\{\mathsf{obs}(\emptyset)\}$

‖

$\{\mathsf{obs}(\{s\})\}$

```
SetSignal s
```

$\{\mathsf{obs}(\emptyset)\}$

Busy-waiting

$\{\mathsf{obs}(\emptyset)\}$

```
let s = CreateSignal ℓ in
```

$\{\mathsf{obs}(\{s\}) * \mathsf{sig}(s, \ell, \mathit{false})\}$

$\boxed{\exists b.\, \mathsf{sig}(s, \ell, b)}$

$\{\mathsf{obs}(\emptyset)\}$

```
(μ loop ().
  if !s then ()
  else
```

$\{\mathsf{obs}(\emptyset) * \mathsf{cp}(\delta_0)\}$

```
    burn δ₀ in loop ()) ()
```

$\{\mathsf{obs}(\emptyset)\}$

||

$\{\mathsf{obs}(\{s\})\}$

```
SetSignal s
```

$\{\mathsf{obs}(\emptyset)\}$

Busy-waiting

$\{\mathsf{obs}(\emptyset)\}$

```
let s = CreateSignal ɩ in
```

$\{\mathsf{obs}(\{s\}) * \mathsf{sig}(s, \iota, \mathit{false})\}$

$\boxed{\exists b.\, \mathsf{sig}(s, \iota, b)}$

$\{\mathsf{obs}(\emptyset)\}$

```
(μ loop ().
  if !s then ()
  else
```

✘ $\{\mathsf{obs}(\emptyset) * \mathsf{cp}(\delta_0)\}$

```
    burn δ0 in loop ()) ()
```

$\{\mathsf{obs}(\emptyset)\}$

||

$\{\mathsf{obs}(\{s\})\}$

```
SetSignal s
```

$\{\mathsf{obs}(\emptyset)\}$

$\{\mathsf{obs}(\emptyset)\}$

```
let s = CreateSignal ɩ in
```

$\{\mathsf{obs}(\{s\}) * \mathsf{sig}(s, \iota, \mathit{false})\}$

$\exists b.\, \mathsf{sig}(s, \iota, b)$

$\{\mathsf{obs}(\emptyset)\}$

```
(μ loop ().
  if !s
        then ()
  else
```

$\{\mathsf{obs}(\emptyset) * \mathsf{cp}(\delta_0)\}$

```
    burn δ0 in loop ()) ()
```

$\{\mathsf{obs}(\emptyset)\}$

||

$\{\mathsf{obs}(\{s\})\}$

```
SetSignal s
```

$\{\mathsf{obs}(\emptyset)\}$

$\{\mathsf{cp}(\delta_1) * \mathsf{obs}(\emptyset)\}$

```
let s = CreateSignal ɩ in
CreateWaitPerm s δ1 δ0;
```

$\{\mathsf{obs}(\{s\}) * \mathsf{sig}(s, \iota, \mathit{false}) * \mathsf{waitp}(s, \delta_0)\}$

$\boxed{\exists b.\, \mathsf{sig}(s, \iota, b)}$

$\{\mathsf{obs}(\emptyset)\}$

```
(μ loop ().
  if !s
       then ()
  else
```

$\{\mathsf{obs}(\emptyset) * \mathsf{cp}(\delta_0)\}$

```
    burn δ0 in loop ()) ()
```

$\{\mathsf{obs}(\emptyset)\}$

||

$\{\mathsf{obs}(\{s\})\}$

```
SetSignal s
```

$\{\mathsf{obs}(\emptyset)\}$

$\{\mathsf{cp}(\delta_1) * \mathsf{obs}(\emptyset)\}$

```
let s = CreateSignal ɩ in
CreateWaitPerm s δ₁ δ₀;
```

$\{\mathsf{obs}(\{s\}) * \mathsf{sig}(s, \iota, \mathit{false}) * \mathsf{waitp}(s, \delta_0)\}$

$\boxed{\exists b.\, \mathsf{sig}(s, \iota, b)}$

$\{\mathsf{obs}(\emptyset) * \mathsf{waitp}(s, \delta_0)\}$

```
(μ loop ().
  if ⟨if !s then () else wait s δ₀;
      !s⟩ then ()
  else
```

$\{\mathsf{obs}(\emptyset) * \mathsf{waitp}(s, \delta_0) * \mathsf{cp}(\delta_0)\}$

```
    burn δ₀ in loop ()) ()
```

$\{\mathsf{obs}(\emptyset)\}$

||

$\{\mathsf{obs}(\{s\})\}$

```
SetSignal s
```

$\{\mathsf{obs}(\emptyset)\}$

$$\{\mathsf{cp}(\delta_1) * \mathsf{obs}(\emptyset)\}$$

```
let f = ref 41 in
let s = CreateSignal 𝔩 in
CreateWaitPerm s δ₁ δ₀;
```

$$\{\mathsf{obs}(\{s\}) * \mathsf{sig}(s, \mathfrak{l}, \mathit{false}) * \mathsf{waitp}(s, \delta_0) * f \mapsto 41\}$$

$$\boxed{\exists b, n.\, \mathsf{sig}(s, \mathfrak{l}, b) * f \mapsto n * n \neq 42 \to b = \mathit{false}}$$

$$\{\mathsf{obs}(\emptyset) * \mathsf{waitp}(s, \delta_0)\}$$

```
(μ loop ().
  if ⟨if !f = 42 then () else wait s δ₀;
      !f = 42⟩ then ()
  else
```

$$\{\mathsf{obs}(\emptyset) * \mathsf{waitp}(s, \delta_0) * \mathsf{cp}(\delta_0)\}$$

```
    burn δ₀ in loop ()) ()
```

$$\{\mathsf{obs}(\emptyset)\}$$

∥

$$\{\mathsf{obs}(\{s\})\}$$

```
⟨f := 42; SetSignal s⟩
```

$$\{\mathsf{obs}(\emptyset)\}$$

$\{\mathsf{cp}(\delta_1) * \mathsf{obs}(\emptyset)\}$

```
let f = ref 41 in
let s = CreateSignal ɩ in
CreateWaitPerm s δ₁ δ₀;
```

$\{\mathsf{obs}(\{s\}) * \mathsf{sig}(s, \iota, \mathit{false}) * \mathsf{waitp}(s, \delta_0) * f \mapsto 41\}$

$\boxed{\exists b, n.\, \mathsf{sig}(s, \iota, b) * f \mapsto n * n \neq 42 \rightarrow b = \mathit{false}}$

$\{\mathsf{obs}(\emptyset) * \mathsf{waitp}(s, \delta_0)\}$

```
(μ loop ().
  if ⟨if !f = 42 then () else wait s δ₀;
      !f = 42⟩ then ()
  else
```

$\{\mathsf{obs}(\emptyset) * \mathsf{waitp}(s, \delta_0) * \mathsf{cp}(\delta_0)\}$

```
    burn δ₀ in loop ()) ()
```

$\{\mathsf{obs}(\emptyset)\}$

∥

$\{\mathsf{obs}(\{s\})\}$

```
⟨f := 42; SetSignal s⟩
```

$\{\mathsf{obs}(\emptyset)\}$

Busy-waiting

```
let f = ref 41 in
```

```
(μ loop ().
  if !f = 42 then ()
  else
    loop ()) ()
```

‖

```
f := 42
```

1. Verifying deadlock-freedom
2. Verifying absence of infinite recursion
3. Verifying termination of busy-waiting programs
4. **Modular specifications**
    A. Logically atomic triples
    B. Total correctness logically atomic triples with liveness assumption
5. Conclusion

“Classic” spec for terminating spinlock

"Classic" spec for terminating spinlock

$\{R * 0 \leq n\}$ `create` $\mathfrak{l}\ n$ $\{lk. \exists \gamma.\ \mathit{is\_lock}(\gamma, lk, \mathfrak{l}, n, R)\}$

$\mathit{is\_lock}(\gamma, lk, \mathfrak{l}, n_1 + n_2, R) \iff \mathit{is\_lock}(\gamma, lk, \mathfrak{l}, n_1, R) * \mathit{is\_lock}(\gamma, lk, \mathfrak{l}, n_2, R)$

$\{\mathit{is\_lock}(\gamma, lk, \mathfrak{l}, 1, R) * \mathsf{obs}(O) * \mathfrak{l} \prec O\}$ `acquire` lk $\{\exists s.\ \mathit{locked}(\gamma, s) * R * \mathsf{obs}(O \cup \{s\})\}$

$\{\mathit{is\_lock}(\gamma, lk, \mathfrak{l}, 0, R) * \mathit{locked}(\gamma, s) * \mathsf{obs}(O)\}$ `release` lk $\{\mathsf{obs}(O \setminus \{s\})\}$

"Classic" spec for terminating spinlock

$\{R * 0 \leq n\}$ `create` $\mathfrak{l}\ n$ $\{lk. \exists \gamma.\ is\_lock(\gamma, lk, \mathfrak{l}, n, R)\}$

$is\_lock(\gamma, lk, \mathfrak{l}, n_1 + n_2, R) \iff is\_lock(\gamma, lk, \mathfrak{l}, n_1, R) * is\_lock(\gamma, lk, \mathfrak{l}, n_2, R)$

$\{is\_lock(\gamma, lk, \mathfrak{l}, 1, R) * \mathsf{obs}(O) * \mathfrak{l} \prec O\}$ `acquire` lk $\{\exists s.\ locked(\gamma, s) * R * \mathsf{obs}(O \cup \{s\})\}$

$\{is\_lock(\gamma, lk, \mathfrak{l}, 0, R) * locked(\gamma, s) * \mathsf{obs}(O)\}$ `release` lk $\{\mathsf{obs}(O \setminus \{s\})\}$

Locking patterns

```
acquire lk      ║  acquire lk
// …            ║  // …
release lk      ║  release lk
```

✅

Locking patterns

`acquire lk`
`// …`
`release lk`
	‖	`acquire lk`
		`// …`
		`release lk`
			✅	`acquire lk`
				`//no release`
					‖	`// no acquire` ❌

Locking patterns

```
acquire lk      ||  acquire lk    ✅
// …            ||  // …
release lk      ||  release lk
```

```
acquire lk      ||  // no acquire   ❌
//no release    ||
```

```
// acquire/release in different threads
let x = ref false in
acquire lk      ||  // busy wait for x    ❌
x := true       ||  release lk
```

Locking patterns

```
acquire lk      ‖  acquire lk
// …            ‖  // …
release lk      ‖  release lk
```

✅

```
acquir
//
```

ire ❌

```
// acq
lo
```

t threads

```
‖ // busy wait for x
‖ release lk
```

❌

Module controls termination reasoning

1. Verifying deadlock-freedom
2. Verifying absence of infinite recursion
3. Verifying termination of busy-waiting programs
4. **Modular specifications**
   - A. **Logically atomic triples**
   - B. Total correctness logically atomic triples with liveness assumption
5. Conclusion

Atomic triple for a lock

$$\langle b.\ \mathit{lock\_state}(\mathit{lk}, b)\rangle\ \texttt{acquire}\ \mathit{lk}\ \langle \mathit{lock\_state}(\mathit{lk}, \mathit{true}) * b = \mathit{false}\rangle$$

Proving atomic triples

$$\langle b.\, \mathit{lock\_state}(\mathit{lk}, b) \rangle\ \texttt{acquire}\ \mathit{lk}\ \langle \mathit{lock\_state}(\mathit{lk}, \mathit{true}) * b = \mathit{false} \rangle \triangleq$$

$$\forall \Phi.\, \langle b.\, \mathit{lock\_state}(\mathit{lk}, b) \mid \mathit{lock\_state}(\mathit{lk}, \mathit{true}) * b = \mathit{false} \Rrightarrow \Phi \rangle \twoheadrightarrow \mathsf{wp}\ \texttt{acquire}\ \mathit{lk}\ \{\Phi\}$$

Proving atomic triples

$$\langle b.\, \mathit{lock\_state}(lk, b) \rangle\ \texttt{acquire}\ lk\ \langle \mathit{lock\_state}(lk, \mathit{true}) * b = \mathit{false} \rangle \triangleq$$
$$\forall \Phi.\ \langle b.\, \mathit{lock\_state}(lk, b) \mid \mathit{lock\_state}(lk, \mathit{true}) * b = \mathit{false} \Rrightarrow \Phi \rangle \mathrel{-\!\!*} \mathsf{wp}\ \texttt{acquire}\ lk\ \{\Phi\}$$

```
// …

CAS(lk,false,true)



// …
```

Proving atomic triples

$$\langle b.\, \mathit{lock\_state}(\mathit{lk}, b)\rangle\ \texttt{acquire}\ \mathit{lk}\ \langle \mathit{lock\_state}(\mathit{lk}, \mathit{true}) * b = \mathit{false}\rangle \triangleq$$
$$\forall \Phi.\, \langle b.\, \mathit{lock\_state}(\mathit{lk}, b) \mid \mathit{lock\_state}(\mathit{lk}, \mathit{true}) * b = \mathit{false} \Rrightarrow \Phi\rangle \mathrel{-\!\!*} \mathsf{wp}\ \texttt{acquire}\ \mathit{lk}\ \{\Phi\}$$

```
// …
```

$\{\exists b.\, \mathit{lock\_state}(\mathit{lk}, b)\}$

```
CAS(lk,false,true)
```

```
// …
```

Proving atomic triples

$$\langle b.\ \mathit{lock\_state}(lk, b)\rangle\ \texttt{acquire}\ lk\ \langle \mathit{lock\_state}(lk, \mathit{true}) * b = \mathit{false}\rangle \triangleq$$
$$\forall \Phi.\ \langle b.\ \mathit{lock\_state}(lk, b) \mid \mathit{lock\_state}(lk, \mathit{true}) * b = \mathit{false} \Rrightarrow \Phi\rangle \twoheadrightarrow \mathsf{wp}\ \texttt{acquire}\ lk\ \{\Phi\}$$

```
// ...
```

$\{\exists b.\ \mathit{lock\_state}(lk, b)\}$

```
CAS(lk,false,true)
```

$$\left\{\begin{array}{ll} \text{if CAS unsuccessful} & \langle b.\ \mathit{lock\_state}(lk, b) \mid \mathit{lock\_state}(lk, \mathit{true}) * b = \mathit{false} \Rrightarrow \Phi\rangle \\ \text{if CAS successful} & \Phi \end{array}\right\}$$

```
// ...
```

1. Verifying deadlock-freedom
2. Verifying absence of infinite recursion
3. Verifying termination of busy-waiting programs
4. **Modular specifications**
   A. Logically atomic triples
   B. **Total correctness logically atomic triples with liveness assumption**
5. Conclusion

Total correctness atomic triple for (unfair) lock

$$\langle b \qquad . \; lock\_state(lk, b)\rangle \; \texttt{acquire} \; lk \; \langle lock\_state(lk, true) * b = false\rangle$$

Total correctness atomic triple for (unfair) lock

$$\langle b \twoheadrightarrow \{false\} \,.\, lock\_state(lk, b) \rangle \; \texttt{acquire} \; lk \; \langle lock\_state(lk, true) * b = false \rangle^{\mathfrak{t}}$$

Total correctness logically atomic triples

$$\langle \vec{x} \quad . \, P \rangle \; e \; \langle \vec{v}.Q \rangle_{\mathcal{E}} \triangleq$$

$$\forall \Phi \qquad . \qquad \langle \vec{x} \quad . \, P \mid \vec{v}.Q \Rrightarrow \Phi \rangle_{\top \setminus \mathcal{E}} \twoheadrightarrow \mathsf{wp}\; e \quad \{\Phi\}$$

Total correctness logically atomic triples

$$\langle \vec{x} \quad . \, P \rangle \; e \; \langle \vec{v}.Q \rangle_{\mathcal{E}} \triangleq$$

$$\forall \Phi \qquad . \qquad \langle \vec{x} \quad . \, P \mid \vec{v}.Q \Rrightarrow \Phi \rangle_{\top \setminus \mathcal{E}} \twoheadrightarrow \textsf{wp}\; e \qquad \{\Phi\}$$

$$\langle \vec{x} \quad . \, P \mid \vec{v}.Q \Rrightarrow \Phi \rangle_{\mathcal{E}} \;\; \vdash \; {}_{\mathcal{E}}{\models\!\!\Rrightarrow}_{\emptyset} \exists \vec{x}. \, P \, *$$

$$\Big(\big(\qquad P \;{}_{\emptyset}{\Rrightarrow\!\!*}_{\mathcal{E}} \langle \vec{x} \quad . \, P \mid \vec{v}.Q \Rrightarrow \Phi \rangle_{\mathcal{E}} \big) \wedge$$

$$\big(\qquad \forall \vec{v}. \, Q \;{}_{\emptyset}{\Rrightarrow\!\!*}_{\mathcal{E}} \Phi \big) \Big)$$

Total correctness logically atomic triples

$$\langle \vec{x} \twoheadrightarrow X.\, P \rangle \; e \; \langle \vec{v}.Q \rangle^{\wr}_{\mathcal{E}} \triangleq$$

$$\forall \Phi \quad . \quad \langle \vec{x} \twoheadrightarrow X.\, P \mid \vec{v}.Q \Rrightarrow \Phi \rangle^{\wr}_{\top \setminus \mathcal{E}} \twoheadrightarrow \mathsf{wp}\; e \quad \{\Phi\}$$

$$\langle \vec{x} \twoheadrightarrow X.\, P \mid \vec{v}.Q \Rrightarrow \Phi \rangle^{\wr}_{\mathcal{E}} \quad \vdash {}_{\mathcal{E}}{\Rrightarrow}_{\emptyset} \exists \vec{x}.\, P \ast$$

$$\Bigg(\Big(\quad P \;{}_{\emptyset}{\Rrightarrow\!\!\ast}_{\mathcal{E}}\; \langle \vec{x} \twoheadrightarrow X.\, P \mid \vec{v}.Q \Rrightarrow \Phi \rangle^{\wr}_{\mathcal{E}} \quad \Big) \wedge$$

$$\Big(\quad \forall \vec{v}.\, Q \;{}_{\emptyset}{\Rrightarrow\!\!\ast}_{\mathcal{E}}\; \Phi \quad \Big) \Bigg)$$

Total correctness logically atomic triples

$$\langle \vec{x} \twoheadrightarrow X.\, P \rangle\ e\ \langle \vec{v}.Q \rangle^{\lightning}_{\mathcal{E}} \triangleq$$

$$\forall \Phi \qquad , \beta. \qquad \langle \vec{x} \overset{\beta}{\twoheadrightarrow} X.\, P \mid \vec{v}.Q \Rrightarrow \Phi \rangle^{\lightning}_{\top \setminus \mathcal{E}} \mathrel{-\!\!*} \mathsf{wp}\ e \quad \beta\ \{\Phi\}$$

$$\langle \vec{x} \overset{\beta}{\twoheadrightarrow} X.\, P \mid \vec{v}.Q \Rrightarrow \Phi \rangle^{\lightning}_{\mathcal{E}} \ \vdash\ {}_{\mathcal{E}}{\Rrightarrow}_{\emptyset} \exists \vec{x}.\, P *$$

$$\Bigg(\Big(\qquad \mathsf{wp}^{\Downarrow}_{\emptyset}\ \beta \left\{ \mathsf{cp}(\delta_e) * \qquad (P\ {}_{\emptyset}{\Rrightarrow\!\!*}_{\mathcal{E}}\ \langle \vec{x} \overset{\beta}{\twoheadrightarrow} X.\, P \mid \vec{v}.Q \Rrightarrow \Phi \rangle^{\lightning}_{\mathcal{E}}\ ) \right\} \Big) \wedge$$

$$\Big(\qquad \forall \vec{v}.\, Q\ {}_{\emptyset}{\Rrightarrow\!\!*}_{\mathcal{E}}\ \Phi\ \Big) \Bigg)$$

Total correctness logically atomic triples

$$\langle \vec{x} \twoheadrightarrow X.\, P \rangle \; e \; \langle \vec{v}.Q \rangle^{\ell}_{\mathcal{E}} \triangleq$$

$$\forall \Phi, O, \quad , \beta.\, \mathsf{obs}(O) \mathrel{-\!\!*} \langle \vec{x} \overset{\beta}{\twoheadrightarrow} X.\, P \mid \vec{v}.Q \Rrightarrow \Phi \rangle^{\ell;O}_{\top \setminus \mathcal{E}} \mathrel{-\!\!*} \mathsf{wp}\; e \quad \beta \, \{\Phi\}$$

$$\langle \vec{x} \overset{\beta}{\twoheadrightarrow} X.\, P \mid \vec{v}.Q \Rrightarrow \Phi \rangle^{\ell;O}_{\mathcal{E}} \vdash {}_{\mathcal{E}}{\Rrightarrow}_{\emptyset} \exists \vec{x}.\, P * \lceil \ell \rceil \prec O *$$

$$\Bigg(\Big(\forall \mathcal{O}.\, \ell \prec \mathcal{O} * \qquad \mathsf{obs}(\mathcal{O}) \mathrel{-\!\!*}$$

$$\mathsf{wp}^{\Downarrow}_{\emptyset} \; \beta \left\{ \mathsf{cp}(\delta_e) * \mathsf{obs}(\mathcal{O}) * \left(P \; {}_{\emptyset}{\Rrightarrow\!\!*}_{\mathcal{E}} \; \langle \vec{x} \overset{\beta}{\twoheadrightarrow} X.\, P \mid \vec{v}.Q \Rrightarrow \Phi \rangle^{\ell;O}_{\mathcal{E}} \right) \right\} \Big) \land$$

$$\Big(\qquad \forall \vec{v}.\, Q \; {}_{\emptyset}{\Rrightarrow\!\!*}_{\mathcal{E}} \; \Phi \quad \Big) \Bigg)$$

Total correctness logically atomic triples

$$\langle \vec{x} \twoheadrightarrow X.\, P \rangle \; e \; \langle \vec{v}.Q \rangle^{\text{⌊}}_{\mathcal{E}} \triangleq$$

$$\forall \Phi, O, \quad , \beta.\, \mathsf{obs}(O) \twoheadrightarrow \langle \vec{x} \overset{\beta}{\twoheadrightarrow} X.\, P \mid \vec{v}.Q \Rrightarrow \Phi \rangle^{\text{⌊};O}_{\top \setminus \mathcal{E}} \twoheadrightarrow \mathsf{wp}\; e \quad \beta\, \{\Phi\}$$

$$\langle \vec{x} \overset{\beta}{\twoheadrightarrow} X.\, P \mid \vec{v}.Q \Rrightarrow \Phi \rangle^{\text{⌊};O}_{\mathcal{E}} \vdash {}_{\mathcal{E}}{\Rrightarrow}_{\emptyset} \exists \vec{x}.\, P * \lceil \text{⌊} \rceil \prec O *$$

$$\Big((\forall \mathcal{O}.\, \text{⌊} \prec \mathcal{O} * \vec{x} \notin X * \mathsf{obs}(\mathcal{O}) \twoheadrightarrow$$

$$\mathsf{wp}^{\Downarrow}_{\emptyset}\; \beta \left\{ \mathsf{cp}(\delta_e) * \mathsf{obs}(\mathcal{O}) * (P \;{}_{\emptyset}{\Rrightarrow}\!\!*_{\mathcal{E}}\; \langle \vec{x} \overset{\beta}{\twoheadrightarrow} X.\, P \mid \vec{v}.Q \Rrightarrow \Phi \rangle^{\text{⌊};O}_{\mathcal{E}}) \right\}) \wedge$$

$$(\qquad \forall \vec{v}.\, Q \;{}_{\emptyset}{\Rrightarrow}\!\!*_{\mathcal{E}}\; \Phi \;) \Big)$$

Total correctness logically atomic triples

$$\langle b \twoheadrightarrow \{false\}.\ lock\_state(lk, b)\rangle \ \texttt{acquire}\ lk \ \langle lock\_state(lk, true) * b = false\rangle^{\lceil}$$

$$\langle \vec{x} \twoheadrightarrow X.\ P\rangle\ e\ \langle \vec{v}.Q\rangle^{\lceil}_{\mathcal{E}} \triangleq$$

$$\forall \Phi, O, \ , \beta.\ \mathsf{obs}(O) \twoheadrightarrow \langle \vec{x} \overset{\beta}{\twoheadrightarrow} X.\ P \mid \vec{v}.Q \Rrightarrow \Phi\rangle^{\lceil;O}_{\top\setminus\mathcal{E}} \twoheadrightarrow \mathsf{wp}\ e \quad \beta\ \{\Phi\}$$

$$\langle \vec{x} \overset{\beta}{\twoheadrightarrow} X.\ P \mid \vec{v}.Q \Rrightarrow \Phi\rangle^{\lceil;O}_{\mathcal{E}} \vdash {}_{\mathcal{E}}{\Rrightarrow}_{\emptyset} \exists \vec{x}.\ P * \lceil \lceil \rceil \prec O * \ldots$$

$$\Big(\big(\forall \mathcal{O}.\ \lceil \prec \mathcal{O} * \vec{x} \notin X * \mathsf{obs}(\mathcal{O}) \twoheadrightarrow$$

$$\mathsf{wp}^{\Downarrow}_{\emptyset}\ \beta \left\{\mathsf{cp}(\delta_e) * \mathsf{obs}(\mathcal{O}) * (P\ {}_{\emptyset}{\Rrightarrow}\!\!*_{\mathcal{E}}\ \langle \vec{x} \overset{\beta}{\twoheadrightarrow} X.\ P \mid \vec{v}.Q \Rrightarrow \Phi\rangle^{\lceil;O}_{\mathcal{E}})\right\}\big) \wedge$$

$$\big(\qquad \forall \vec{v}.\ Q\ {}_{\emptyset}{\Rrightarrow}\!\!*_{\mathcal{E}}\ \Phi \ \big)\Big)$$

```
⟨let b = CAS(lk,false,true) in
  (if b then () else β); b⟩
```

Total correctness logically atomic triples

$$\langle \vec{x} \twoheadrightarrow X.\, P \rangle\ e\ \langle \vec{v}.Q \rangle^{\iota}_{\mathcal{E}} \triangleq$$

$$\forall \Phi, O, \quad , \beta.\, \mathsf{obs}(O) \mathrel{-\!\!*} \langle \vec{x} \xrightarrow{\beta} X.\, P \mid \vec{v}.Q \Rrightarrow \Phi \rangle^{\iota;O}_{\top\setminus\mathcal{E}} \mathrel{-\!\!*} \mathsf{wp}\ e \quad \beta\ \{\Phi\}$$

$$\langle \vec{x} \xrightarrow{\beta} X.\, P \mid \vec{v}.Q \Rrightarrow \Phi \rangle^{\iota;O}_{\mathcal{E}} \vdash {}_{\mathcal{E}}{\models\!\!\Rrightarrow}_{\emptyset} \exists \vec{x}.\, P * \lceil \iota \rceil \prec O *$$

$$\Big((\forall \mathcal{O}.\, \iota \prec \mathcal{O} * \vec{x} \notin X * \mathsf{obs}(\mathcal{O}) \mathrel{-\!\!*}$$

$$\mathsf{wp}^{\Downarrow}_{\emptyset}\ \beta\ \left\{ \mathsf{cp}(\delta_e) * \mathsf{obs}(\mathcal{O}) * (P\ {}_{\emptyset}{\Rrightarrow\!\!*}_{\mathcal{E}}\ \langle \vec{x} \xrightarrow{\beta} X.\, P \mid \vec{v}.Q \Rrightarrow \Phi \rangle^{\iota;O}_{\mathcal{E}}) \right\}) \wedge$$

$$(\qquad \forall \vec{v}.\, Q\ {}_{\emptyset}{\Rrightarrow\!\!*}_{\mathcal{E}}\ \Phi \quad) \Big)$$

Total correctness logically atomic triples

$$\langle \vec{x} \twoheadrightarrow X.\, P \rangle \; e \; \langle \vec{v}.Q \rangle^{\ell}_{\mathcal{E}} \triangleq$$

$$\forall \Phi, O, \alpha, \beta.\, \mathsf{obs}(O) \twoheadrightarrow \langle \vec{x} \overset{\beta}{\twoheadrightarrow} X.\, P \underset{\alpha}{\mid} \vec{v}.Q \Rrightarrow \Phi \rangle^{\ell;O}_{\top \setminus \mathcal{E}} \twoheadrightarrow \mathsf{wp}\; e \; \alpha \; \beta \; \{\Phi\}$$

$$\langle \vec{x} \overset{\beta}{\twoheadrightarrow} X.\, P \underset{\alpha}{\mid} \vec{v}.Q \Rrightarrow \Phi \rangle^{\ell;O}_{\mathcal{E}} \vdash {}_{\mathcal{E}}\!\Rrightarrow_{\emptyset} \exists \vec{x}.\, P * \lceil \ell \rceil \prec O \, *$$

$$\Big(\big(\forall \mathcal{O}.\, \ell \prec \mathcal{O} * \vec{x} \notin X * \mathsf{obs}(\mathcal{O}) \twoheadrightarrow$$

$$\mathsf{wp}^{\Downarrow}_{\emptyset} \; \beta \; \Big\{ \mathsf{cp}(\delta_e) * \mathsf{obs}(\mathcal{O}) * (P \; {}_{\emptyset}\!\Rrightarrow\!\!*_{\mathcal{E}} \langle \vec{x} \overset{\beta}{\twoheadrightarrow} X.\, P \underset{\alpha}{\mid} \vec{v}.Q \Rrightarrow \Phi \rangle^{\ell;O}_{\mathcal{E}}) \Big\}\big) \wedge$$

$$\big(\mathsf{obs}(O) \twoheadrightarrow \mathsf{wp}^{\Downarrow}_{\emptyset} \; \alpha \; \{\forall \vec{v}.\, Q \; {}_{\emptyset}\!\Rrightarrow\!\!*_{\mathcal{E}} \Phi\}\big) \Big)$$

Total correctness logically atomic triples

$\langle b \twoheadrightarrow \{\mathit{false}\}\,.\, \mathit{lock\_state}(lk, b)\rangle$ `acquire` lk $\langle \mathit{lock\_state}(lk, \mathit{true}) * b = \mathit{false}\rangle^{\iota}$

$$\langle \vec{x} \twoheadrightarrow X.\, P\rangle\; e\; \langle \vec{v}.Q\rangle^{\iota}_{\mathcal{E}} \triangleq$$

$$\forall \Phi, O, \alpha, \beta.\, \mathsf{obs}(O) \twoheadrightarrow * \langle \vec{x} \overset{\beta}{\twoheadrightarrow} X.\, P \underset{\alpha}{|} \vec{v}.Q \Rrightarrow \Phi\rangle^{\iota;O}_{\top\setminus\mathcal{E}} \twoheadrightarrow * \;\mathsf{wp}\; e\; \alpha\; \beta\; \{\Phi\}$$

$$\langle \vec{x} \overset{\beta}{\twoheadrightarrow} X.\, P \underset{\alpha}{|} \vec{v}.Q \Rrightarrow \Phi\rangle^{\iota;O}_{\mathcal{E}} \vdash {}_{\mathcal{E}}{\Rrightarrow}_{\emptyset}\, \exists \vec{x}.\, P * \lceil \iota \ldots$$

$$\Big(\big(\forall \mathcal{O}.\, \iota \prec \mathcal{O} * \vec{x} \notin X * \mathsf{obs}(\mathcal{O}) \twoheadrightarrow *$$

$$\mathsf{wp}^{\Downarrow}_{\emptyset}\; \beta \left\{ \mathsf{cp}(\delta_e) * \mathsf{obs}(\mathcal{O}) * (P\; {}_{\emptyset}{\Rrightarrow}\!\!*_{\mathcal{E}}\; \langle \vec{x} \overset{\beta}{\twoheadrightarrow} X.\, P \underset{\alpha}{|} \vec{v}.Q \Rrightarrow \Phi\rangle^{\iota;O}_{\mathcal{E}}) \right\}\big) \wedge$$

$$\big(\mathsf{obs}(O) \twoheadrightarrow * \;\mathsf{wp}^{\Downarrow}_{\emptyset}\; \alpha\; \{\forall \vec{v}.\, Q\; {}_{\emptyset}{\Rrightarrow}\!\!*_{\mathcal{E}}\; \Phi\}\big)\Big)$$

```
⟨let b = CAS(lk,false,true) in
  (if b then α else β); b⟩
```

Total correctness logically atomic triples

$$\langle \vec{x} \twoheadrightarrow X.\, P \rangle\ e\ \langle \vec{v}.Q \rangle^{\lightning}_{\mathcal{E}} \triangleq$$

$$\forall \Phi, O, \alpha, \beta.\, \mathsf{obs}(O) \mathrel{-\!\!*} \langle \vec{x} \overset{\beta}{\twoheadrightarrow} X.\, P \underset{\alpha}{\mid} \vec{v}.Q \Rrightarrow \Phi \rangle^{\lightning;O}_{\top\setminus\mathcal{E}} \mathrel{-\!\!*} \mathsf{wp}\ e\ \alpha\ \beta\ \{\Phi\}$$

$$\langle \vec{x} \overset{\beta}{\twoheadrightarrow} X.\, P \underset{\alpha}{\mid} \vec{v}.Q \Rrightarrow \Phi \rangle^{\lightning;O}_{\mathcal{E}} \vdash {}_{\mathcal{E}}{\models\!\!\Rrightarrow}_{\emptyset}\, \exists \vec{x}.\, P * \lceil \lightning \rceil \prec O *$$

$$\Big(\big(\forall \mathcal{O}.\, \lightning \prec \mathcal{O} * \vec{x} \notin X * \mathsf{obs}(\mathcal{O}) \mathrel{-\!\!*}$$

$$\mathsf{wp}^{\Downarrow}_{\emptyset}\ \beta \left\{ \mathsf{cp}(\delta_e) * \mathsf{obs}(\mathcal{O}) * \big(P\ {}_{\emptyset}{\Rrightarrow\!\!*}_{\mathcal{E}}\ \langle \vec{x} \overset{\beta}{\twoheadrightarrow} X.\, P \underset{\alpha}{\mid} \vec{v}.Q \Rrightarrow \Phi \rangle^{\lightning;O}_{\mathcal{E}}\big) \right\} \big) \wedge$$

$$\big(\mathsf{obs}(O) \mathrel{-\!\!*} \mathsf{wp}^{\Downarrow}_{\emptyset}\ \alpha\ \{\forall \vec{v}.\, Q\ {}_{\emptyset}{\Rrightarrow\!\!*}_{\mathcal{E}}\ \Phi\}\big) \Big)$$

1. Verifying deadlock-freedom
2. Verifying absence of infinite recursion
3. Verifying termination of busy-waiting programs
4. Modular specifications
   A. Logically atomic triples
   B. Total correctness logically atomic triples with liveness assumption
5. **Conclusion**

Conclusion

We propose:

- Modular specifications for total correctness of busy-waiting concurrent modules

We currently have:

- VeriFast proofs of spinlocks and ticketlocks
- Some building blocks in Coq/Iris
- The belief that the approach scales to cohort locks

Some references

In addition to Iris, this work is heavily influenced by

- D'Osualdo, Emanuele, Julian Sutherland, Azadeh Farzan, and Philippa Gardner. "TaDA Live: Compositional Reasoning for Termination of Fine-Grained Concurrent Programs." ACM Transactions on Programming Languages and Systems 43, no. 4 (December 31, 2021): 1–134. https://doi.org/10.1145/3477082.
- Mulder, Ike, and Robbert Krebbers. "Proof Automation for Linearizability in Separation Logic." Proceedings of the ACM on Programming Languages 7, no. OOPSLA1 (April 6, 2023): 462–91. https://doi.org/10.1145/3586043.
- Leino, K. Rustan M., Peter Müller, and Jan Smans. "Deadlock-Free Channels and Locks." In Programming Languages and Systems, edited by Andrew D. Gordon, 6012:407–26. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010. https://doi.org/10.1007/978-3-642-11957-6_22.
- Kobayashi, Naoki. "A New Type System for Deadlock-Free Processes." In CONCUR 2006 – Concurrency Theory, edited by Christel Baier and Holger Hermanns, 4137:233–47. Berlin, Heidelberg: Springer Berlin Heidelberg, 2006. https://doi.org/10.1007/11817949_16.

Some references

In addition to Iris, this work is heavily influenced by

- Jacobs, Bart, Dragan Bosnacki, and Ruurd Kuiper. “Modular Termination Verification of Single-Threaded and Multithreaded Programs.” ACM Transactions on Programming Languages and Systems 40, no. 3 (September 30, 2018): 1–59. https://doi.org/10.1145/3210258.

- Reinhard, Tobias, and Bart Jacobs. “Ghost Signals: Verifying Termination of Busy Waiting: Verifying Termination of Busy Waiting.” In Computer Aided Verification, edited by Alexandra Silva and K. Rustan M. Leino, 12760:27–50. Cham: Springer International Publishing, 2021. https://doi.org/10.1007/978-3-030-81688-9_2.

- Jacobs, Bart, and Frank Piessens. “Expressive Modular Fine-Grained Concurrency Specification.” In Proceedings of the 38th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, 271–82. Austin Texas USA: ACM, 2011. https://doi.org/10.1145/1926385.1926417.

Backup slides

$\mathsf{wp}^{\Downarrow}$
Definition

$$\mathsf{wp}^{\Downarrow}_{\mathcal{E}}\ e\ \{v.\ P\} \triangleq \forall \sigma, n_s, \vec{\kappa}, n_t.\ S(\sigma, n_s, \vec{\kappa}, n_t) \Rrightarrow\!\!\ast_{\mathcal{E}}$$
$$\exists \sigma', v.\ (e, \sigma \Downarrow v, \sigma') * S(\sigma', n_s, \vec{\kappa}, n_t) * P(v)$$

$\mathsf{wp}^{\Downarrow}$
Lemmas

Big-step-atomic

$$ {}_{\mathcal{E}_1}{\models\!\!\!\Rrightarrow}_{\mathcal{E}_2} \mathsf{wp}^{\Downarrow}_{\mathcal{E}_2}\ e \left\{v.\ {}_{\mathcal{E}_2}{\models\!\!\!\Rrightarrow}_{\mathcal{E}_1} P\right\} \vdash \mathsf{wp}^{\Downarrow}_{\mathcal{E}_1}\ e\,\{v.\,P\} $$

Big-step-bind

$$ \frac{K \text{ is a context}}{\mathsf{wp}^{\Downarrow}_{\mathcal{E}}\ e \left\{v.\,\mathsf{wp}^{\Downarrow}_{\mathcal{E}}\ K[v]\ \{w.\,P\}\right\} \vdash \mathsf{wp}^{\Downarrow}_{\mathcal{E}}\ K[e]\ \{w.\,P\}} $$

Big-step-AtomicBlock

$$ \mathsf{wp}^{\Downarrow}_{\mathcal{E}}\ e\,\{v.\,P\} \vdash \mathsf{wp}_{\mathcal{E}}\ \langle e\rangle\ \{v.\,P\} $$

HeapLang<

Head step rules

- Convention: \ has precedence over ⊎
- θ : thread id
- τ : "thread phase", to prevent self-fueling busy-waiting
- `AtomicBlock` uses big-step evaluation relation that matches the operational semantics but precludes forking

BurnS

$$\frac{\tau' = \sigma.\textsc{Phase}(\theta) \qquad \tau = \max_{\sqsubseteq} \{\tau \mid (\tau, \delta) \in \sigma.\textsc{CallPerms} \wedge \tau \sqsubseteq \tau'\} \qquad (\tau,\delta) \in \sigma.\textsc{CallPerms} \qquad \delta' < \delta \qquad 0 \le n}{\texttt{Burn}(e, \delta, n, \delta'), \sigma \xrightarrow[\theta]{\epsilon}_{\mathsf{h}} e, \sigma : \textsc{CallPerms} \setminus \{(\tau,\delta)\} \uplus (n \cdot (\tau, \delta'))}$$

CreateSignalS

$$\frac{s \notin \sigma.\textsc{Signals} \qquad \mathfrak{l} \in \mathfrak{L}}{\texttt{CreateSignal}(\mathfrak{l}), \sigma \xrightarrow[\theta]{\epsilon}_{\mathsf{h}} (), \sigma : \textsc{Signals}[s \leftarrow (\mathfrak{l}, \mathit{false})] : \textsc{Obligations}(\theta) \cup \{s\}}$$

SetSignalS

$$\frac{\sigma.\textsc{Signals}(s) = (\mathfrak{l}, \_)}{\texttt{SetSignal}(s), \sigma \xrightarrow[\theta]{\epsilon}_{\mathsf{h}} (), \sigma : \textsc{Signals}[s \leftarrow (\mathfrak{l}, \mathit{true})] : \textsc{Obligations}(\theta) \setminus \{s\}}$$

CreateWaitPermS

$$\frac{\tau' = \sigma.\textsc{Phase}(\theta) \qquad \tau = \max_{\sqsubseteq} \{\tau \mid (\tau, \delta) \in \sigma.\textsc{CallPerms} \wedge \tau \sqsubseteq \tau'\} \qquad (\tau,\delta) \in \sigma.\textsc{CallPerms} \qquad \delta' < \delta}{\texttt{CreateWaitPerm}(s, \delta, \delta') \xrightarrow[\theta]{\epsilon}_{\mathsf{h}} (), \sigma : \textsc{CallPerms} \setminus \{(\tau,\delta)\} : \textsc{WaitPerms} \cup (s, (\tau, \delta'))}$$

WaitS

$$\frac{\tau' = \sigma.\textsc{Phase}(\theta) \qquad \tau = \min_{\sqsubseteq} \{\tau \mid (s, (\tau, \delta)) \in \sigma.\textsc{WaitPerms} \wedge \tau \sqsubseteq \tau'\} \qquad \sigma.\textsc{Signals}(s) = (\mathfrak{l}, \mathit{false}) \qquad \mathfrak{l} \prec \sigma.\textsc{Obligations}(\theta) \qquad (s, (\tau, \delta)) \in \sigma.\textsc{WaitPerms}}{\texttt{Wait}(s, \delta) \xrightarrow[\theta]{\epsilon}_{\mathsf{h}} (), \sigma : \textsc{CallPermissions} \uplus (\tau', \delta)}$$

AssertNoObs

$$\frac{\sigma.\textsc{Obligations}(\theta) = \emptyset}{\texttt{AssertNoObs}, \sigma \xrightarrow[\theta]{\epsilon}_{\mathsf{h}} (), \sigma}$$

ForkS

$$\frac{\tau = \sigma.\textsc{Phase}(\theta) \qquad \mathit{sigs} \text{ set of signals} \qquad \theta' = \min(\mathit{ThreadId} \setminus \mathsf{dom}(\sigma.\textsc{Obligations}))}{\texttt{fork}(e, \mathit{sigs}), \sigma \xrightarrow[\theta]{\epsilon}_{\mathsf{h}} (), \sigma : \textsc{Obligations}(\theta) \setminus \mathit{sigs} : \textsc{Obligations}(\theta') \cup \mathit{sigs} : \textsc{Phase}[\theta \leftarrow \tau.\text{Forker}; \theta' \leftarrow \tau.\text{Forkee}], (e; \texttt{AssertNoObs})}$$

AtomicBlockS

$$\frac{e, \sigma \Downarrow v, \sigma'}{\texttt{AtomicBlock}(e), \sigma \xrightarrow[\theta]{\epsilon}_{\mathsf{h}} v, \sigma'}$$

Total correctness logically atomic triples

With liveness assumption

We reflect "rounds" of waiting with r/R. An example is the current owner of a ticketlock changing. This is our approach to enable waiting based on a module's internal termination argument (e.g. the ticket-based queue).

$$\langle \vec{x} \twoheadrightarrow_r X.\, P \rangle\ e\ \langle \vec{v}.Q \rangle^{\mathfrak{l}}_{\mathcal{E}} \triangleq$$

$$\forall \Phi, \tau, O, R, \alpha, \beta.\ \mathsf{obs}(\tau, O) \twoheadrightarrow \langle \vec{x} \overset{\beta}{\twoheadrightarrow}_r X.\, P \underset{\alpha}{\mid} \vec{v}.Q \Rrightarrow \Phi \rangle^{\mathfrak{l};O}_{\top \setminus \mathcal{E}} \twoheadrightarrow \mathsf{wp}\ e\ \alpha\ \beta\ \{\Phi\}$$

$$\langle \vec{x} \overset{\beta}{\twoheadrightarrow}_r X.\, P \underset{\alpha}{\mid} \vec{v}.Q \Rrightarrow \Phi \rangle^{\mathfrak{l};O}_{\mathcal{E}} \vdash {}_{\mathcal{E}}{\Rrightarrow}_{\emptyset} \exists \vec{x}.\, P * \lceil \mathfrak{l} \rceil \prec O *$$

$$\Big(\big(\forall O'.\, \mathfrak{l} \prec O' * (\exists r_0.\, R(r_0) * (r_0 = r \vee \mathsf{cp}(\tau', \delta'_e) * \vec{x} \notin X * \mathsf{obs}(\tau, O')) \twoheadrightarrow$$

$$\mathsf{wp}^{\Downarrow}_{\emptyset}\ \beta \left\{ \mathsf{cp}(\tau, \delta_e) * R(r) * \mathsf{obs}(\tau, O') * (P\ {}_{\emptyset}{\Rrightarrow}\!\!*_{\mathcal{E}}\ \langle \vec{x} \overset{\beta}{\twoheadrightarrow}_r X.\, P \underset{\alpha}{\mid} \vec{v}.Q \Rrightarrow \Phi \rangle^{\mathfrak{l};O}_{\mathcal{E}}) \right\} \big) \wedge$$

$$\big(R(\_) * \mathsf{obs}(\tau, O) \twoheadrightarrow \mathsf{wp}^{\Downarrow}_{\emptyset}\ \alpha\ \{\forall \vec{v}.\, Q\ {}_{\emptyset}{\Rrightarrow}\!\!*_{\mathcal{E}}\ \Phi\}\big) \Big)$$

“Tricky client”

Unfair spinlock is terminating in the right context

```
let x = ref false in

acquire lk   ||  // busy wait for x          ||  x := true
x := true    ||  (μ loop ().
release lk   ||    acquire(lk);
             ||    let d = !x in
             ||    release(lk);
             ||    if d then ()
             ||    else loop ()) ()
```

Terminating for **fair** locks under fair scheduling

Terminating for **fair and unfair** locks under fair scheduling