Separation Logic 4/4

Jean-Marie Madiot

Inria Paris

February 24, 2026

Reminders about ghost state

What are invariants lacking?

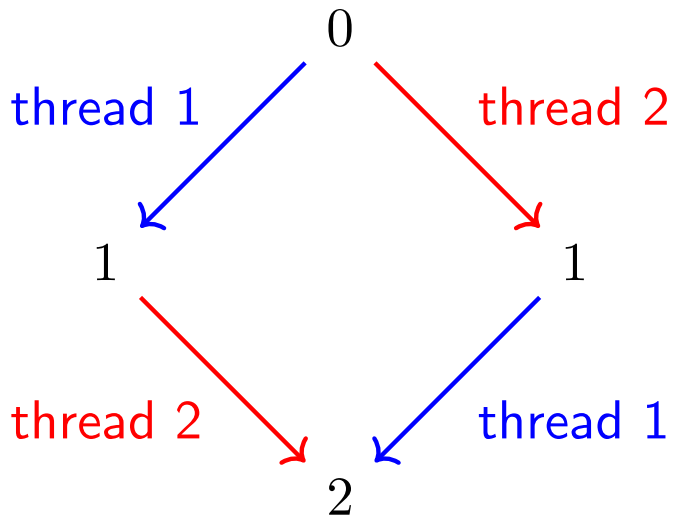
At high-level the program has two interleavings:

```
    let r = ref 0 in
    let l = newlock () in
acquire l;    || acquire l;
r := !r + 1; || r := !r + 1;
release  l    || release l
    acquire l;
    assert (!r = 2)
```

What are invariants lacking?

At high-level the program has two interleavings:

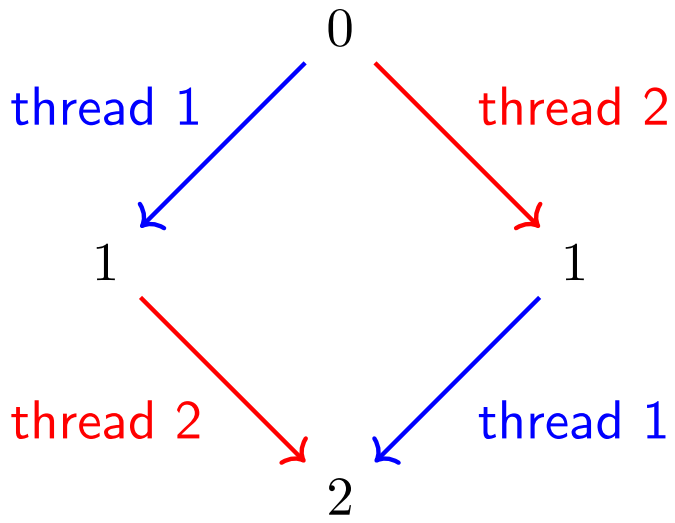
```
    let r = ref 0 in
    let l = newlock () in
acquire l;    || acquire l;
r := !r + 1; || r := !r + 1;
release  l    || release l
    acquire l;
    assert (!r = 2)
```



What are invariants lacking?

At high-level the program has two interleavings:

```
    let r = ref 0 in
    let l = newlock () in
acquire l;    || acquire l;
r := !r + 1; || r := !r + 1;
release  l    || release l
    acquire l;
    assert (!r = 2)
```

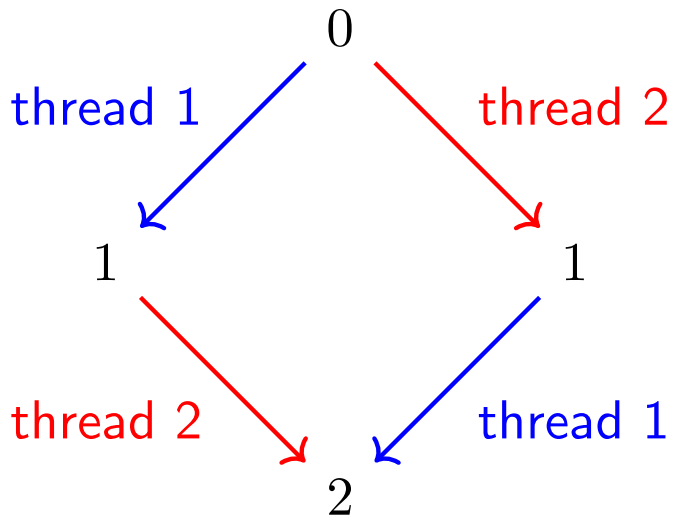


A proof would need need to reflect this somehow.

What are invariants lacking?

At high-level the program has two interleavings:

```
    let r = ref 0 in
    let l = newlock () in
acquire l;    || acquire l;
r := !r + 1; || r := !r + 1;
release  l    || release l
    acquire l;
    assert (!r = 2)
```



A proof would need need to reflect this somehow.

- some notion of *state* embedded into the separation logic
- *splitting* (and *combining*) states into parts for each thread
- all possibles orders: combining is *commutative* and *associative* (same reason as for $*$)

One way to add state: auxiliary variables

```
let r = ref 0
let r1 = ref 0
let r2 = ref 0
let p = newlock()
```

```
acquire p;                 ||    acquire p;
r := !r + 1;               ||    r := !r + 1;
r1 := !r1 + 1;             ||    r2 := !r2 + 1;
release p;                 ||    release p;
```

```
acquire p;
assert (!r == 2);
```

Ghost state

What is ghost state?

In parallel programs $p = e_1||\ldots||e_n$, reductions $(p, h) \rightarrow (p', h')$ can be from any of the e_i, so $\rightarrow$ is non-deterministic.

What is ghost state?

In parallel programs $p = e_1||\ldots||e_n$, reductions $(p, h) \rightarrow (p', h')$ can be from any of the e_i, so $\rightarrow$ is non-deterministic.

Correctness of non-deterministic programs must consider **all** possible physical steps $(p, h) \rightarrow (p', h')$.

What is ghost state?

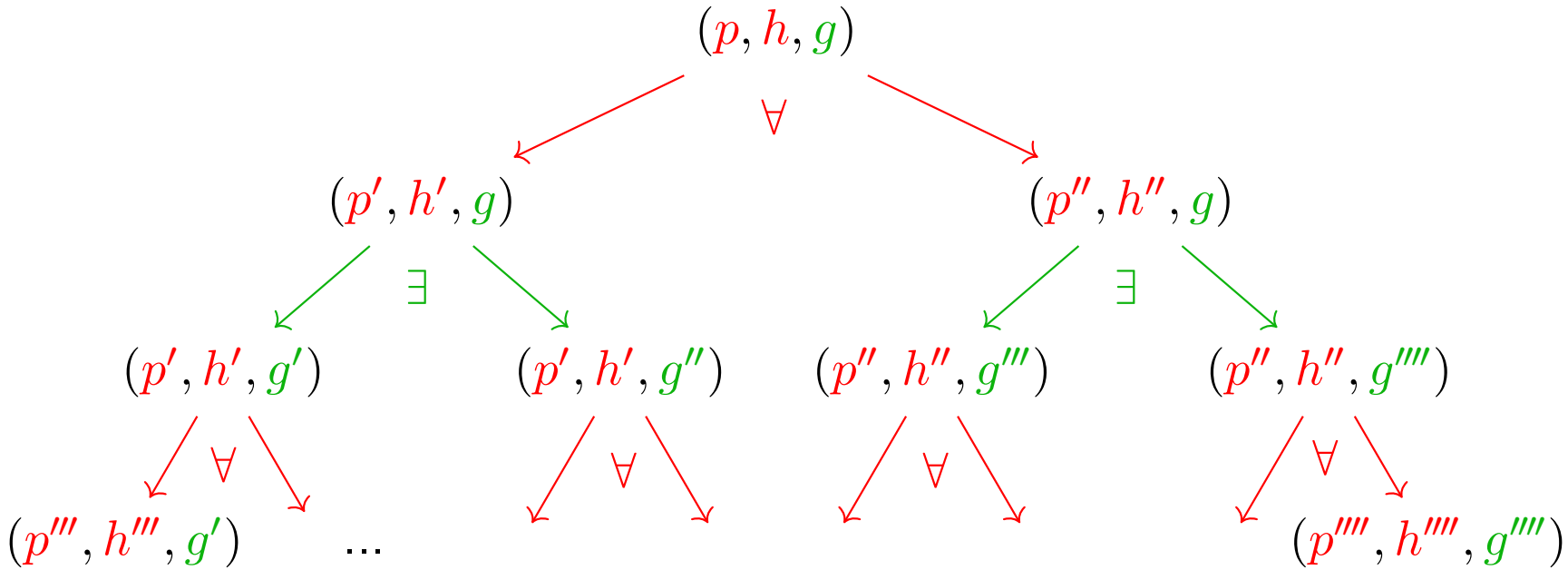
In parallel programs $p = e_1||\ldots||e_n$, reductions $(p, h) \to (p', h')$ can be from any of the e_i, so $\to$ is non-deterministic.

Correctness of non-deterministic programs must consider **all** possible physical steps $(p, h) \to (p', h')$. For each, we get to **choose** a ghost update $g \to g'$:

What is ghost state?

In parallel programs $p = e_1||\ldots||e_n$, reductions $(p, h) \to (p', h')$ can be from any of the e_i, so $\to$ is non-deterministic.

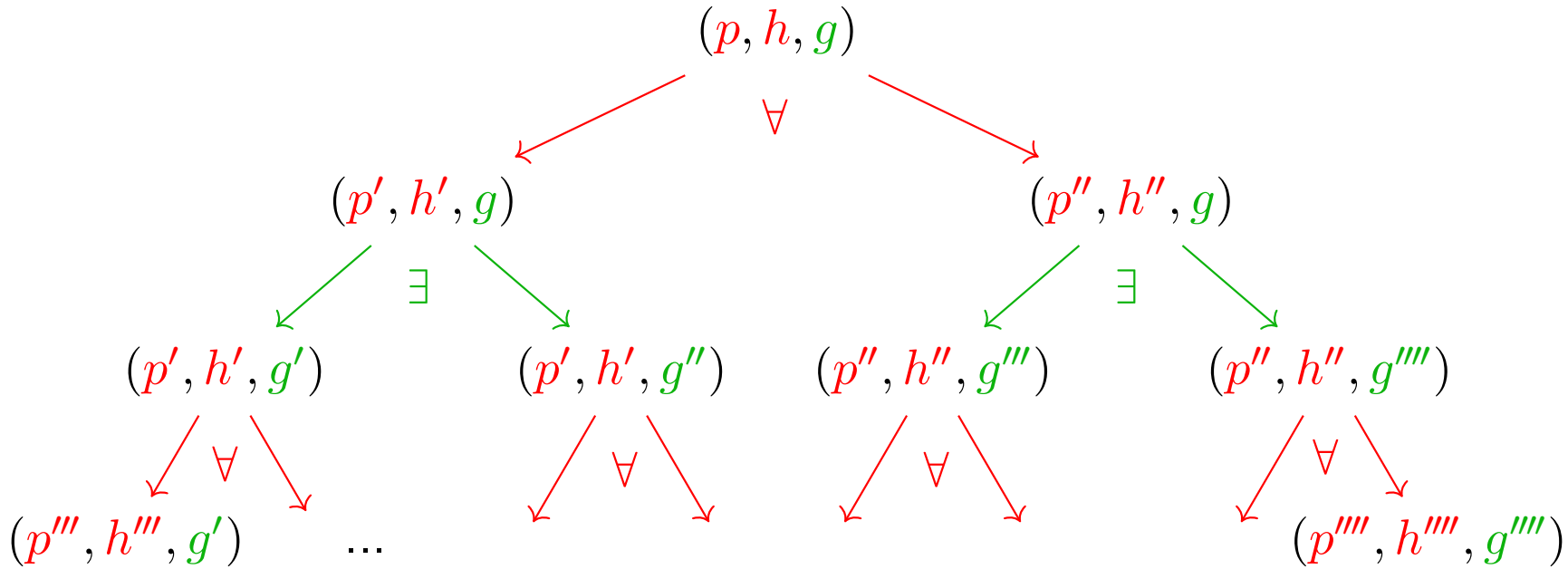
Correctness of non-deterministic programs must consider **all** possible physical steps $(p, h) \to (p', h')$. For each, we get to **choose** a ghost update $g \to g'$:



What is ghost state?

In parallel programs $p = e_1 || \ldots || e_n$, reductions $(p, h) \to (p', h')$ can be from any of the e_i, so $\to$ is non-deterministic.

Correctness of non-deterministic programs must consider **all** possible physical steps $(p, h) \to (p', h')$. For each, we get to **choose** a ghost update $g \to g'$:



Ghost state also needs to be split $g = g_1 \cdot \ldots \cdot g_n$, one part for each e_i.

Ghost state, ghost assertions

Let $(\mathcal{M}, \cdot)$ be a commutative *semigroup* (or "monoid"), i.e. $\forall abc \in \mathcal{M}$:

$$a \cdot (b \cdot c) = (a \cdot b) \cdot c \qquad\qquad a \cdot b = b \cdot a$$

Ghost state, ghost assertions

Let $(\mathcal{M}, \cdot)$ be a commutative *semigroup* (or "monoid"), i.e. $\forall abc \in \mathcal{M}$:

$$a \cdot (b \cdot c) = (a \cdot b) \cdot c \qquad\qquad a \cdot b = b \cdot a$$

$a \in \mathcal{M}$ is called a *resource*. At any given time:

- each thread t_i "owns" a resource $a_i \in \mathcal{M}$,

Ghost state, ghost assertions

Let $(\mathcal{M}, \cdot)$ be a commutative *semigroup* (or "monoid"), i.e. $\forall abc \in \mathcal{M}$:

$$a \cdot (b \cdot c) = (a \cdot b) \cdot c \qquad\qquad a \cdot b = b \cdot a$$

$a \in \mathcal{M}$ is called a *resource*. At any given time:

- each thread t_i "owns" a resource $a_i \in \mathcal{M}$,
- each unopened lock "owns" a resource $r_j \in \mathcal{M}$ that satisfies its R_j

Ghost state, ghost assertions

Let $(\mathcal{M}, \cdot)$ be a commutative *semigroup* (or "monoid"), i.e. $\forall abc \in \mathcal{M}$:

$$a \cdot (b \cdot c) = (a \cdot b) \cdot c \qquad\qquad a \cdot b = b \cdot a$$

$a \in \mathcal{M}$ is called a *resource*. At any given time:

- each thread t_i "owns" a resource $a_i \in \mathcal{M}$,
- each unopened lock "owns" a resource $r_j \in \mathcal{M}$ that satisfies its R_j
- the global resource $a_1 \cdot \ldots \cdot a_n \cdot r_1 \cdot \ldots \cdot r_k$ is the global ghost state

Ghost state, ghost assertions

Let $(\mathcal{M}, \cdot)$ be a commutative *semigroup* (or "monoid"), i.e. $\forall abc \in \mathcal{M}$:

$$a \cdot (b \cdot c) = (a \cdot b) \cdot c \qquad\qquad a \cdot b = b \cdot a$$

$a \in \mathcal{M}$ is called a *resource*. At any given time:

- each thread t_i "owns" a resource $a_i \in \mathcal{M}$,
- each unopened lock "owns" a resource $r_j \in \mathcal{M}$ that satisfies its R_j
- the global resource $a_1 \cdot \ldots \cdot a_n \cdot r_1 \cdot \ldots \cdot r_k$ is the global ghost state

Ownership of a is written $\boxed{a}$. Composition maps to separation:

$$\boxed{a \cdot b} = \boxed{a} * \boxed{b}$$

Ghost updates

The prover can perform some "ghost updates" inside each thread:

$\boxed{a} \Rrightarrow \boxed{a'}$ $\quad \approx$ changing $a \cdot a_2 \cdot a_3 \cdot \ldots$ to $a' \cdot a_2 \cdot a_3 \cdot \ldots$

Ghost updates

The prover can perform some "ghost updates" inside each thread:

$$\boxed{a} \Rrightarrow \boxed{a'} \qquad \approx \text{changing } a \cdot a_2 \cdot a_3 \cdot \ldots \text{ to } a' \cdot a_2 \cdot a_3 \cdot \ldots$$

The ghost state can be updated before or after physical steps:

$$\textsc{Ghost-update}\quad \frac{P \Rrightarrow P' \qquad \{P'\}\, c\, \{\lambda x.\, Q'\} \qquad \forall x.\, Q'\, x \Rrightarrow Q\, x}{\{P\}\, c\, \{\lambda x.\, Q\}}$$

but $P \Rrightarrow P'$ does **not** imply $P \rhd P'$

Ghost updates

The prover can perform some "ghost updates" inside each thread:

$$\boxed{a} \Rrightarrow \boxed{a'} \qquad \approx \text{changing } a \cdot a_2 \cdot a_3 \cdot \ldots \text{ to } a' \cdot a_2 \cdot a_3 \cdot \ldots$$

The ghost state can be updated before or after physical steps:

$$\textsc{Ghost-update}\quad \frac{P \Rrightarrow P' \qquad \{P'\}\, c\, \{\lambda x.\, Q'\} \qquad \forall x.\, Q'\, x \Rrightarrow Q\, x}{\{P\}\, c\, \{\lambda x.\, Q\}}$$

but $P \Rrightarrow P'$ does **not** imply $P \rhd P'$

But to be useful, we must deduce *something* from $\boxed{a}$, knowing that other threads can update their own resources.

any idea how?

Validity

Idea: pick an invariant on the global resource, *validity*: $\mathcal{V} : \mathcal{M} \to \texttt{Prop}$

$$\mathcal{V}(a_1 \cdot \ldots \cdot a_n \cdot r_1 \cdot \ldots \cdot r_k) \text{ at all times} \qquad \text{and} \qquad \frac{\mathcal{V}(a \cdot b)}{\mathcal{V}(a)}$$

Validity

Idea: pick an invariant on the global resource, *validity*: $\mathcal{V} : \mathcal{M} \to \texttt{Prop}$

$$\mathcal{V}(a_1 \cdot \ldots \cdot a_n \cdot r_1 \cdot \ldots \cdot r_k) \text{ at all times} \qquad \text{and} \qquad \frac{\mathcal{V}(a \cdot b)}{\mathcal{V}(a)}$$

Follows rely-guarantee-style protocol:

Validity

Idea: pick an invariant on the global resource, *validity*: $\mathcal{V} : \mathcal{M} \to \texttt{Prop}$

$$\mathcal{V}(a_1 \cdot \ldots \cdot a_n \cdot r_1 \cdot \ldots \cdot r_k) \text{ at all times} \qquad \text{and} \qquad \frac{\mathcal{V}(a \cdot b)}{\mathcal{V}(a)}$$

Follows rely-guarantee-style protocol:

- ownership $\boxed{a}$ **provides** validity of the "local" resource:

$$\overline{\boxed{a} \Rightarrow \mathcal{V}(a)}$$

Validity

Idea: pick an invariant on the global resource, *validity*: $\mathcal{V} : \mathcal{M} \to \texttt{Prop}$

$$\mathcal{V}(a_1 \cdot \ldots \cdot a_n \cdot r_1 \cdot \ldots \cdot r_k) \text{ at all times} \qquad \text{and} \qquad \frac{\mathcal{V}(a \cdot b)}{\mathcal{V}(a)}$$

Follows rely-guarantee-style protocol:

- ownership $\boxed{a}$ **provides** validity of the "local" resource:

$$\frac{}{\boxed{a} \Rightarrow \mathcal{V}(a)}$$

- updating ghost $\boxed{a} \Rrightarrow \boxed{a'}$ **requires** preservation of global validity in all contexts:

$$\frac{a \rightsquigarrow a'}{\boxed{a} \Rrightarrow \boxed{a'}} \qquad a \rightsquigarrow a' \equiv \begin{cases} \mathcal{V}(a) \Rightarrow \mathcal{V}(a') \\ \forall c \in \mathcal{M} \quad \mathcal{V}(a \cdot c) \Rightarrow \mathcal{V}(a' \cdot c) \end{cases}$$

Validity

Idea: pick an invariant on the global resource, *validity*: $\mathcal{V} : \mathcal{M} \to \texttt{Prop}$

$$\mathcal{V}(a_1 \cdot \ldots \cdot a_n \cdot r_1 \cdot \ldots \cdot r_k) \text{ at all times} \qquad \text{and} \qquad \frac{\mathcal{V}(a \cdot b)}{\mathcal{V}(a)}$$

Follows rely-guarantee-style protocol:

- ownership $\boxed{a}$ **provides** validity of the "local" resource:

$$\frac{}{\boxed{a} \Rightarrow \mathcal{V}(a)}$$

- updating ghost $\boxed{a} \Rrightarrow \boxed{a'}$ **requires** preservation of global validity in all contexts:

$$\frac{a \rightsquigarrow a'}{\boxed{a} \Rrightarrow \boxed{a'}} \qquad a \rightsquigarrow a' \equiv \begin{cases} \mathcal{V}(a) \Rightarrow \mathcal{V}(a') \\ \forall c \in \mathcal{M} \quad \mathcal{V}(a \cdot c) \Rightarrow \mathcal{V}(a' \cdot c) \end{cases}$$

$a \rightsquigarrow a'$ is called a *frame-preserving update*.

Resource algebra

A *resource algebra* is a triple $(\mathcal{M}, \cdot, \mathcal{V})$ such that $(\mathcal{M}, \cdot)$ is a commutative semigroup with a downward-closed predicate $\mathcal{V} \subseteq \mathcal{M}$, i.e. satisfying $\forall a\ b\ c$:

$$a \cdot (b \cdot c) = (a \cdot b) \cdot c \qquad a \cdot b = b \cdot a \qquad \frac{\mathcal{V}(a \cdot b)}{\mathcal{V}(a)}$$

Last rule also written $\dfrac{\mathcal{V}(c) \qquad a \preccurlyeq c}{\mathcal{V}(a)}$ where $a \preccurlyeq c \ \equiv\ \exists b.\ c = a \cdot b$

Resource algebra

A *resource algebra* is a triple $(\mathcal{M}, \cdot, \mathcal{V})$ such that $(\mathcal{M}, \cdot)$ is a commutative semigroup with a downward-closed predicate $\mathcal{V} \subseteq \mathcal{M}$, i.e. satisfying $\forall a\ b\ c$:

$$a \cdot (b \cdot c) = (a \cdot b) \cdot c \qquad a \cdot b = b \cdot a \qquad \frac{\mathcal{V}(a \cdot b)}{\mathcal{V}(a)}$$

Last rule also written $\dfrac{\mathcal{V}(c) \qquad a \preccurlyeq c}{\mathcal{V}(a)}$ where $a \preccurlyeq c \ \equiv\ \exists b.\ c = a \cdot b$

Exercise: Give a resource algebra $\mathcal{M}_t$ with $t \in \mathcal{M}_t$ s.t.

$$\mathcal{V}(t) \qquad \neg\mathcal{V}(t \cdot t)$$

t is sometimes called a *token*.

Resource algebra exercise: $\mathcal{M}_{\mathsf{sf}}$

Exercise: (ra_SF.v) Give a resource algebra $\mathcal{M}_{\mathsf{sf}}$ with $S, F \in \mathcal{M}_{\mathsf{sf}}$ s.t.

$$\mathcal{V}(S) \qquad \neg\mathcal{V}(S \cdot F) \qquad F = F \cdot F \qquad S \rightsquigarrow F$$

Recall:

$$a \cdot (b \cdot c) = (a \cdot b) \cdot c \qquad a \cdot b = b \cdot a \qquad \frac{\mathcal{V}(a \cdot b)}{\mathcal{V}(a)}$$

$$a \rightsquigarrow a' \;\equiv\; \begin{cases} \forall b \in \mathcal{M} \;\; \mathcal{V}(a \cdot b) \Rightarrow \mathcal{V}(a' \cdot b) \\ \mathcal{V}(a) \Rightarrow \mathcal{V}(a') \end{cases}$$

For once, the Rocq proof is easier `https://madiot.fr/progproofs/ra_SF.v`

Resource algebra $\mathcal{M}_{\mathsf{sf}}$

$$a \cdot (b \cdot c) = (a \cdot b) \cdot c \qquad a \cdot b = b \cdot a \qquad \frac{\mathcal{V}(a \cdot b)}{\mathcal{V}(a)}\ \mathcal{V}{\downarrow}$$

$$1 :\ \mathcal{V}(S) \qquad 2 :\ \neg\mathcal{V}(S \cdot F) \qquad 3 :\ F = F \cdot F \qquad 4 :\ S \rightsquigarrow F$$

Operations and validity:

$\cdot$	S	F	
S			
F			
$\mathcal{V}(-)$			

Resource algebra $\mathcal{M}_{\mathsf{sf}}$

$$a \cdot (b \cdot c) = (a \cdot b) \cdot c \qquad a \cdot b = b \cdot a \qquad \frac{\mathcal{V}(a \cdot b)}{\mathcal{V}(a)}\ \mathcal{V}{\downarrow}$$

$$1 :\ \mathcal{V}(S) \qquad 2 :\ \neg\mathcal{V}(S \cdot F) \qquad 3 :\ F = F \cdot F \qquad 4 :\ S \rightsquigarrow F$$

Operations and validity:

$\cdot$	S	F	
S			
F		F	
$\mathcal{V}(-)$	True		

by 1,3;

Resource algebra $\mathcal{M}_{\mathsf{sf}}$

$$a \cdot (b \cdot c) = (a \cdot b) \cdot c \qquad a \cdot b = b \cdot a \qquad \frac{\mathcal{V}(a \cdot b)}{\mathcal{V}(a)}\ \mathcal{V}{\downarrow}$$

$$1 :\ \mathcal{V}(S) \qquad 2 :\ \neg\mathcal{V}(S \cdot F) \qquad 3 :\ F = F \cdot F \qquad 4 :\ S \rightsquigarrow F$$

Operations and validity:

$\cdot$	S	F	
S			
F		F	
$\mathcal{V}(-)$	True	True	

by 1,3; by 1,4;

Resource algebra $\mathcal{M}_{\mathsf{sf}}$

$$a \cdot (b \cdot c) = (a \cdot b) \cdot c \qquad a \cdot b = b \cdot a \qquad \frac{\mathcal{V}(a \cdot b)}{\mathcal{V}(a)}\ \mathcal{V}{\downarrow}$$

$$1 :\ \mathcal{V}(S) \qquad 2 :\ \neg\mathcal{V}(S \cdot F) \qquad 3 :\ F = F \cdot F \qquad 4 :\ S \rightsquigarrow F$$

Operations and validity:

$\cdot$	S	F	↯
S		↯	
F		F	
↯			

$\mathcal{V}(-)$	True	True	False

by 1,3; by 1,4; by 2 (↯: notation for some invalid);

Resource algebra $\mathcal{M}_{\mathsf{sf}}$

$$a \cdot (b \cdot c) = (a \cdot b) \cdot c \qquad a \cdot b = b \cdot a \qquad \frac{\mathcal{V}(a \cdot b)}{\mathcal{V}(a)}\ \mathcal{V}{\downarrow}$$

$$1:\ \mathcal{V}(S) \qquad 2:\ \neg\mathcal{V}(S \cdot F) \qquad 3:\ F = F \cdot F \qquad 4:\ S \rightsquigarrow F$$

Operations and validity:

$\cdot$	S	F	$\lightning$
S		$\lightning$	
F	$\lightning$	F	
$\lightning$			

$\mathcal{V}(-)$	True	True	False

by 1,3; by 1,4; by 2 ($\lightning$: notation for some invalid); by comm.;

Resource algebra $\mathcal{M}_{\mathsf{sf}}$

$$a \cdot (b \cdot c) = (a \cdot b) \cdot c \qquad a \cdot b = b \cdot a \qquad \frac{\mathcal{V}(a \cdot b)}{\mathcal{V}(a)}\ \mathcal{V}{\downarrow}$$

$$1:\ \mathcal{V}(S) \qquad 2:\ \neg\mathcal{V}(S \cdot F) \qquad 3:\ F = F \cdot F \qquad 4:\ S \rightsquigarrow F$$

Operations and validity:

$\cdot$	S	F	$\lightning$
S		$\lightning$	$\lightning$
F	$\lightning$	F	$\lightning$
$\lightning$	$\lightning$	$\lightning$	$\lightning$

$\mathcal{V}(-)$	True	True	False

by 1,3; by 1,4; by 2 ($\lightning$: notation for some invalid); by comm.; by $\mathcal{V}{\downarrow}$;

Resource algebra $\mathcal{M}_{\mathsf{sf}}$

$$a \cdot (b \cdot c) = (a \cdot b) \cdot c \qquad a \cdot b = b \cdot a \qquad \frac{\mathcal{V}(a \cdot b)}{\mathcal{V}(a)}\ \mathcal{V}{\downarrow}$$

$$1:\ \mathcal{V}(S) \qquad 2:\ \neg\mathcal{V}(S \cdot F) \qquad 3:\ F = F \cdot F \qquad 4:\ S \rightsquigarrow F$$

Operations and validity:

$\cdot$	S	F	$\lightning$
S	$\lightning$	$\lightning$	$\lightning$
F	$\lightning$	F	$\lightning$
$\lightning$	$\lightning$	$\lightning$	$\lightning$

$\mathcal{V}(-)$	True	True	False

by 1,3; by 1,4; by 2 ($\lightning$: notation for some invalid); by comm.; by $\mathcal{V}{\downarrow}$; by 4.

Resource algebra $\mathcal{M}_{\mathsf{sf}}$

$$a \cdot (b \cdot c) = (a \cdot b) \cdot c \qquad a \cdot b = b \cdot a \qquad \frac{\mathcal{V}(a \cdot b)}{\mathcal{V}(a)}\ \mathcal{V}{\downarrow}$$

$$1:\ \mathcal{V}(S) \qquad 2:\ \neg\mathcal{V}(S \cdot F) \qquad 3:\ F = F \cdot F \qquad 4:\ S \rightsquigarrow F$$

Operations and validity:

$\cdot$	S	F	$\lightning$
S	$\lightning$	$\lightning$	$\lightning$
F	$\lightning$	F	$\lightning$
$\lightning$	$\lightning$	$\lightning$	$\lightning$

$\mathcal{V}(-)$	True	True	False

by 1,3; by 1,4; by 2 ($\lightning$: notation for some invalid); by comm.; by $\mathcal{V}{\downarrow}$; by 4.

Still need to check that laws hold (1, 2, 3, 4, assoc, sym, $\mathcal{V}{\downarrow}$) (exercise!)

Usage of $\mathcal{M}_{\mathsf{sf}}$ in a proof

We will use $\mathcal{M}_{\mathsf{sf}}$ to prove a simpler program:

```
    let r = ref 0
    let l = newlock ()
acquire l; || acquire l;
r := !r;   || r := 1;
release l  || release l
    acquire l;
    assert (!r = 1)
```

Usage of $\mathcal{M}_{\mathsf{sf}}$ in a proof

We will use $\mathcal{M}_{\mathsf{sf}}$ to prove a simpler program:

```
    let r = ref 0
    let l = newlock ()
acquire l; || acquire l;
r := !r;   || r := 1;
release l  || release l
    acquire l;
    assert (!r = 1)
```

Consequences of the properties on S, F:

$\mathcal{V}(S)$	ghost state can start at S
$\neg\mathcal{V}(S \cdot F)$	$S * F = S \cdot F \Rightarrow \ulcorner\mathsf{False}\urcorner$
$F = F \cdot F$	$F = F \cdot F = F * F$
$S \rightsquigarrow F$	$S \Rrightarrow F$

Exercise: Give a lock invariant using $\mathcal{M}_{\mathsf{sf}}$ to prove the program above.

Proof of `r := !r || r := 1`

$\{\boxed{S}\}$

```
let r = ref 0
```

$\{\boxed{S} * r \mapsto 0\}$

Proof of `r := !r || r := 1`

$\{\boxed{S}\}$

```
let r = ref 0
```

$\{\boxed{S} * r \mapsto 0\} \Rightarrow \{R\}$ with $R = r \mapsto 0 * \boxed{S} \lor r \mapsto 1 * \boxed{F}$

Proof of `r := !r || r := 1`

$\{\boxed{S}\}$
`let r = ref 0`
$\{\boxed{S} * r \mapsto 0\} \Rightarrow \{R\}$ with $R = r \mapsto 0 * \boxed{S} \lor r \mapsto 1 * \boxed{F}$
`let l = newlock ()`
$\{l \rightsquigarrow \mathsf{Lock}\, R\}$

Proof of `r := !r || r := 1`

$\{\boxed{S}\}$
`let r = ref 0`
$\{\boxed{S} * r \mapsto 0\} \Rightarrow \{R\}$ with $R = r \mapsto 0 * \boxed{S} \lor r \mapsto 1 * \boxed{F}$
`let l = newlock ()`
$\{l \rightsquigarrow \mathsf{Lock}\, R\}$
`acquire l;`
$\{r \mapsto 0 * \boxed{S} \lor r \mapsto 1 * \boxed{F}\}$

Proof of `r := !r || r := 1`

$\{\boxed{S}\}$
`let r = ref 0`
$\{\boxed{S} * r \mapsto 0\} \Rrightarrow \{R\}$ with $R = r \mapsto 0 * \boxed{S} \Vdash r \mapsto 1 * \boxed{F}$
`let l = newlock ()`
$\{l \rightsquigarrow \mathsf{Lock}\, R\}$
`acquire l;`
$\{r \mapsto 0 * \boxed{S} \Vdash r \mapsto 1 * \boxed{F}\}$

$\{r \mapsto 0 * \boxed{S}\}$	$\{r \mapsto 1 * \boxed{F}\}$
`r := !r;`	`r := !r;`
$\{r \mapsto 0 * \boxed{S}\}$	$\{r \mapsto 1 * \boxed{F}\}$
$\{R\}$	$\{R\}$

Proof of `r := !r || r := 1`

$\{\boxed{S}\}$

`let r = ref 0`

$\{\boxed{S} * r \mapsto 0\} \Rightarrow \{R\}$ with $R = r \mapsto 0 * \boxed{S} \vee r \mapsto 1 * \boxed{F}$

`let l = newlock ()`

$\{l \rightsquigarrow \mathsf{Lock}\, R\}$

`acquire l;`

$\{r \mapsto 0 * \boxed{S} \vee r \mapsto 1 * \boxed{F}\}$

$\{r \mapsto 0 * \boxed{S}\}$	$\{r \mapsto 1 * \boxed{F}\}$
`r := !r;`	`r := !r;`
$\{r \mapsto 0 * \boxed{S}\}$	$\{r \mapsto 1 * \boxed{F}\}$
$\{R\}$	$\{R\}$

$\{R\}$

`release l`

$\{\top\}$

Proof of `r := !r || r := 1`

$\{\boxed{S}\}$
`let r = ref 0`
$\{\boxed{S} * r \mapsto 0\} \Rightarrow \{R\}$ with $R = r \mapsto 0 * \boxed{S} \vee r \mapsto 1 * \boxed{F}$
`let l = newlock ()`
$\{l \rightsquigarrow \mathsf{Lock}\, R\}$

`acquire l;`		`acquire l;`
$\{r \mapsto 0 * \boxed{S} \vee r \mapsto 1 * \boxed{F}\}$		
$\{r \mapsto 0 * \boxed{S}\}$	$\{r \mapsto 1 * \boxed{F}\}$	
`r := !r;`	`r := !r;`	
$\{r \mapsto 0 * \boxed{S}\}$	$\{r \mapsto 1 * \boxed{F}\}$	
$\{R\}$	$\{R\}$	
$\{R\}$		
`release l`		
$\{\top\}$		

Proof of `r := !r || r := 1`

$\{\boxed{S}\}$
`let r = ref 0`
$\{\boxed{S} * r \mapsto 0\} \Rightarrow \{R\}$ with $R = r \mapsto 0 * \boxed{S} \vee r \mapsto 1 * \boxed{F}$
`let l = newlock ()`
$\{l \rightsquigarrow \mathsf{Lock}\, R\}$

`acquire l;`	`acquire l;`
$\{r \mapsto 0 * \boxed{S} \vee r \mapsto 1 * \boxed{F}\}$	$\{r \mapsto 0 * \boxed{S} \vee r \mapsto 1 * \boxed{F}\}$
$\{r \mapsto 0 * \boxed{S}\}$ | $\{r \mapsto 1 * \boxed{F}\}$	
`r := !r;` | `r := !r;`	
$\{r \mapsto 0 * \boxed{S}\}$ | $\{r \mapsto 1 * \boxed{F}\}$	
$\{R\}$ | $\{R\}$	
$\{R\}$	
`release l`	
$\{\top\}$	

Proof of `r := !r || r := 1`

$\{\boxed{S}\}$
`let r = ref 0`
$\{\boxed{S} * r \mapsto 0\} \Rightarrow \{R\}$ with $R = r \mapsto 0 * \boxed{S} \vee r \mapsto 1 * \boxed{F}$
`let l = newlock ()`
$\{l \rightsquigarrow \mathsf{Lock}\, R\}$

`acquire l;`	`acquire l;`
$\{r \mapsto 0 * \boxed{S} \vee r \mapsto 1 * \boxed{F}\}$	$\{r \mapsto 0 * \boxed{S} \vee r \mapsto 1 * \boxed{F}\}$
$\{r \mapsto 0 * \boxed{S}\}$ | $\{r \mapsto 1 * \boxed{F}\}$	$\{r \mapsto 0 * \boxed{S}\}$
`r := !r;` | `r := !r;`	`r := 1;`
$\{r \mapsto 0 * \boxed{S}\}$ | $\{r \mapsto 1 * \boxed{F}\}$	$\{r \mapsto 1 * \boxed{S}\}$
$\{R\}$ | $\{R\}$	
$\{R\}$	
`release l`	
$\{\top\}$	

Proof of `r := !r || r := 1`

$\{\boxed{S}\}$
`let r = ref 0`
$\{\boxed{S} * r \mapsto 0\} \Rrightarrow \{R\}$ with $R = r \mapsto 0 * \boxed{S} \vee r \mapsto 1 * \boxed{F}$
`let l = newlock ()`
$\{l \rightsquigarrow \mathsf{Lock}\, R\}$

`acquire l;`	`acquire l;`
$\{r \mapsto 0 * \boxed{S} \vee r \mapsto 1 * \boxed{F}\}$	$\{r \mapsto 0 * \boxed{S} \vee r \mapsto 1 * \boxed{F}\}$
$\{r \mapsto 0 * \boxed{S}\}$	$\{r \mapsto 1 * \boxed{F}\}$	$\{r \mapsto 0 * \boxed{S}\}$
`r := !r;`	`r := !r;`	`r := 1;`
$\{r \mapsto 0 * \boxed{S}\}$	$\{r \mapsto 1 * \boxed{F}\}$	$\{r \mapsto 1 * \boxed{S}\} \Rrightarrow$
$\{R\}$	$\{R\}$	$\{r \mapsto 1 * \boxed{F}\}$
$\{R\}$		
`release l`		
$\{\top\}$		

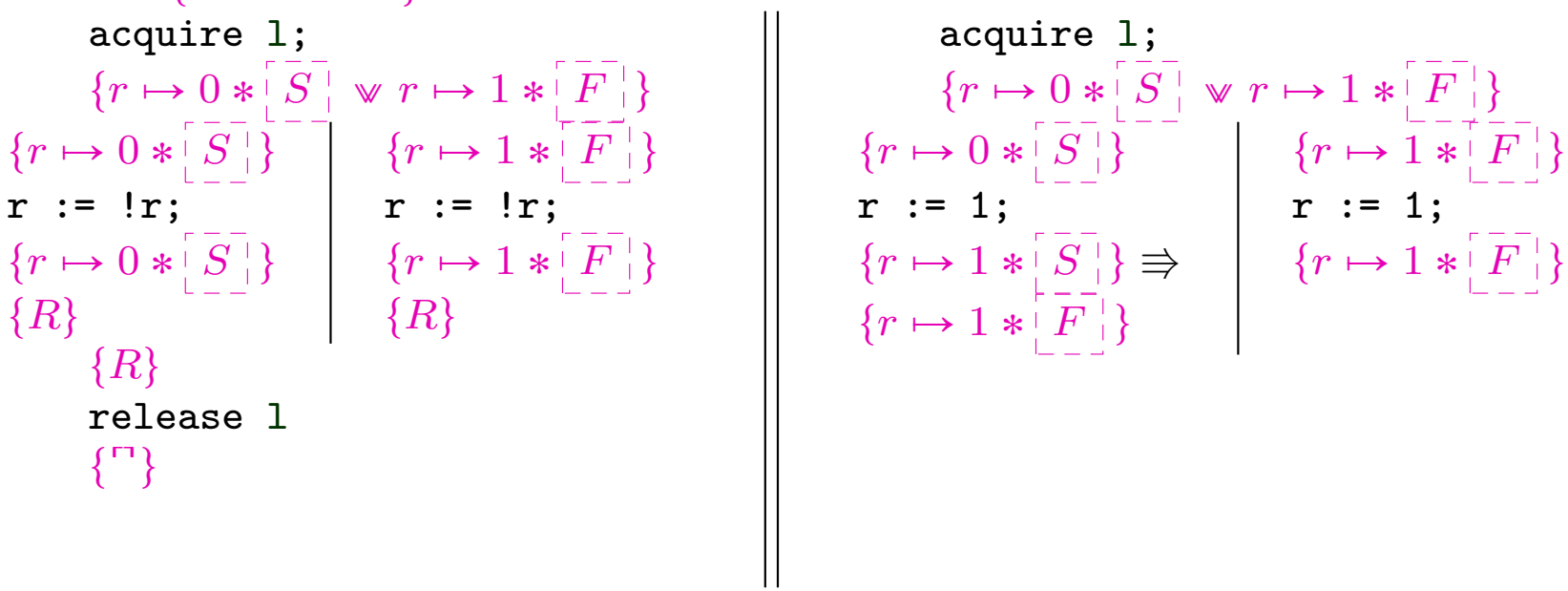
Proof of `r := !r || r := 1`

$\{\boxed{S}\}$
`let r = ref 0`
$\{\boxed{S} * r \mapsto 0\} \Rrightarrow \{R\}$ with $R = r \mapsto 0 * \boxed{S} \vee r \mapsto 1 * \boxed{F}$
`let l = newlock ()`
$\{l \rightsquigarrow \mathsf{Lock}\, R\}$

`acquire l;`	`acquire l;`
$\{r \mapsto 0 * \boxed{S} \vee r \mapsto 1 * \boxed{F}\}$	$\{r \mapsto 0 * \boxed{S} \vee r \mapsto 1 * \boxed{F}\}$
$\{r \mapsto 0 * \boxed{S}\}$	$\{r \mapsto 1 * \boxed{F}\}$
`r := !r;`	`r := 1;`
$\{r \mapsto 0 * \boxed{S}\}$	$\{r \mapsto 1 * \boxed{S}\} \Rrightarrow$
$\{R\}$	$\{r \mapsto 1 * \boxed{F}\}$
$\{R\}$	
`release l`	
$\{\top\}$	

Proof of `r := !r || r := 1`

$\{\boxed{S}\}$
`let r = ref 0`
$\{\boxed{S} * r \mapsto 0\} \Rrightarrow \{R\}$ with $R = r \mapsto 0 * \boxed{S} \vee r \mapsto 1 * \boxed{F}$
`let l = newlock ()`
$\{l \rightsquigarrow \mathsf{Lock}\, R\}$



Proof of `r := !r || r := 1`

$\{\boxed{S}\}$
`let r = ref 0`
$\{\boxed{S} * r \mapsto 0\} \Rrightarrow \{R\}$ with $R = r \mapsto 0 * \boxed{S} \vee r \mapsto 1 * \boxed{F}$
`let l = newlock ()`
$\{l \rightsquigarrow \mathsf{Lock}\, R\}$

`acquire l;`
$\{r \mapsto 0 * \boxed{S} \vee r \mapsto 1 * \boxed{F}\}$

$\{r \mapsto 0 * \boxed{S}\}$	$\{r \mapsto 1 * \boxed{F}\}$
`r := !r;`	`r := !r;`
$\{r \mapsto 0 * \boxed{S}\}$	$\{r \mapsto 1 * \boxed{F}\}$
$\{R\}$	$\{R\}$

$\{R\}$
`release l`
$\{\top\}$

`acquire l;`
$\{r \mapsto 0 * \boxed{S} \vee r \mapsto 1 * \boxed{F}\}$

$\{r \mapsto 0 * \boxed{S}\}$	$\{r \mapsto 1 * \boxed{F}\}$
`r := 1;`	`r := 1;`
$\{r \mapsto 1 * \boxed{S}\} \Rrightarrow$	$\{r \mapsto 1 * \boxed{F}\}$
$\{r \mapsto 1 * \boxed{F}\}$	

$\{r \mapsto 1 * \boxed{F}\}$

Proof of `r := !r || r := 1`

$\{\boxed{S}\}$
`let r = ref 0`
$\{\boxed{S} * r \mapsto 0\} \Rightarrow \{R\}$ with $R = r \mapsto 0 * \boxed{S} \vee r \mapsto 1 * \boxed{F}$
`let l = newlock ()`
$\{l \rightsquigarrow \mathsf{Lock}\, R\}$

`acquire l;`	`acquire l;`
$\{r \mapsto 0 * \boxed{S} \vee r \mapsto 1 * \boxed{F}\}$	$\{r \mapsto 0 * \boxed{S} \vee r \mapsto 1 * \boxed{F}\}$
$\{r \mapsto 0 * \boxed{S}\}$	$\{r \mapsto 1 * \boxed{F}\}$
`r := !r;`	`r := 1;`
$\{r \mapsto 0 * \boxed{S}\}$	$\{r \mapsto 1 * \boxed{F}\}$
$\{R\}$	$\{R\}$
$\{R\}$	
`release l`	
$\{\urcorner\}$	

$\{r \mapsto 1 * \boxed{S}\} \Rrightarrow \{r \mapsto 1 * \boxed{F}\}$

$\{r \mapsto 1 * \boxed{F}\} \Rightarrow \{r \mapsto 1 * \boxed{F} * \boxed{F}\}$

Proof of `r := !r || r := 1`

$\{\boxed{S}\}$
`let r = ref 0`
$\{\boxed{S} * r \mapsto 0\} \Rightarrow \{R\}$ with $R = r \mapsto 0 * \boxed{S} \vee r \mapsto 1 * \boxed{F}$
`let l = newlock ()`
$\{l \rightsquigarrow \mathsf{Lock}\, R\}$

`acquire l;`
$\{r \mapsto 0 * \boxed{S} \vee r \mapsto 1 * \boxed{F}\}$

$\{r \mapsto 0 * \boxed{S}\}$	$\{r \mapsto 1 * \boxed{F}\}$
`r := !r;`	`r := !r;`
$\{r \mapsto 0 * \boxed{S}\}$	$\{r \mapsto 1 * \boxed{F}\}$
$\{R\}$	$\{R\}$

$\{R\}$
`release l`
$\{\boxed{\ }\}$

`acquire l;`
$\{r \mapsto 0 * \boxed{S} \vee r \mapsto 1 * \boxed{F}\}$

$\{r \mapsto 0 * \boxed{S}\}$	$\{r \mapsto 1 * \boxed{F}\}$
`r := 1;`	`r := 1;`
$\{r \mapsto 1 * \boxed{S}\} \Rrightarrow$	$\{r \mapsto 1 * \boxed{F}\}$
$\{r \mapsto 1 * \boxed{F}\}$	

$\{r \mapsto 1 * \boxed{F}\} \Rightarrow \{r \mapsto 1 * \boxed{F} * \boxed{F}\}$
$\{R * \boxed{F}\}$
`release l`
$\{\boxed{F}\}$

Proof of `r := !r || r := 1`

$\{\boxed{S}\}$

`let r = ref 0`

$\{\boxed{S} * r \mapsto 0\} \Rightarrow \{R\}$ with $R = r \mapsto 0 * \boxed{S} \vee r \mapsto 1 * \boxed{F}$

`let l = newlock ()`

$\{l \rightsquigarrow \mathsf{Lock}\, R\}$

`acquire l;`

$\{r \mapsto 0 * \boxed{S} \vee r \mapsto 1 * \boxed{F}\}$

$\{r \mapsto 0 * \boxed{S}\}$	$\{r \mapsto 1 * \boxed{F}\}$
`r := !r;`	`r := !r;`
$\{r \mapsto 0 * \boxed{S}\}$	$\{r \mapsto 1 * \boxed{F}\}$
$\{R\}$	$\{R\}$

$\{R\}$

`release l`

$\{\boxed{\ }\}$

`acquire l;`

$\{r \mapsto 0 * \boxed{S} \vee r \mapsto 1 * \boxed{F}\}$

$\{r \mapsto 0 * \boxed{S}\}$	$\{r \mapsto 1 * \boxed{F}\}$
`r := 1;`	`r := 1;`
$\{r \mapsto 1 * \boxed{S}\} \Rrightarrow$	$\{r \mapsto 1 * \boxed{F}\}$
$\{r \mapsto 1 * \boxed{F}\}$	

$\{r \mapsto 1 * \boxed{F}\} \Rightarrow \{r \mapsto 1 * \boxed{F} * \boxed{F}\}$

$\{R * \boxed{F}\}$

`release l`

$\{\boxed{F}\}$

$\{\boxed{F}\}$

`acquire l;`

$\{\boxed{F} * (r \mapsto 0 * \boxed{S} \vee r \mapsto 1 * \boxed{F})\}$

Proof of `r := !r || r := 1`

$\{\boxed{S}\}$

`let r = ref 0`

$\{\boxed{S} * r \mapsto 0\} \Rrightarrow \{R\}$ with $R = r \mapsto 0 * \boxed{S} \vee r \mapsto 1 * \boxed{F}$

`let l = newlock ()`

$\{l \rightsquigarrow \mathsf{Lock}\, R\}$

`acquire l;`

$\{r \mapsto 0 * \boxed{S} \vee r \mapsto 1 * \boxed{F}\}$

$\{r \mapsto 0 * \boxed{S}\}$	$\{r \mapsto 1 * \boxed{F}\}$
`r := !r;`	`r := !r;`
$\{r \mapsto 0 * \boxed{S}\}$	$\{r \mapsto 1 * \boxed{F}\}$
$\{R\}$	$\{R\}$

$\{R\}$

`release l`

$\{\lceil\rceil\}$

`acquire l;`

$\{r \mapsto 0 * \boxed{S} \vee r \mapsto 1 * \boxed{F}\}$

$\{r \mapsto 0 * \boxed{S}\}$	$\{r \mapsto 1 * \boxed{F}\}$
`r := 1;`	`r := 1;`
$\{r \mapsto 1 * \boxed{S}\} \Rrightarrow$	$\{r \mapsto 1 * \boxed{F}\}$
$\{r \mapsto 1 * \boxed{F}\}$	

$\{r \mapsto 1 * \boxed{F}\} \Rrightarrow \{r \mapsto 1 * \boxed{F} * \boxed{F}\}$

$\{R * \boxed{F}\}$

`release l`

$\{\boxed{F}\}$

$\{\boxed{F}\}$

`acquire l;`

$\{\boxed{F} * (r \mapsto 0 * \boxed{S} \vee r \mapsto 1 * \boxed{F})\}$

$\{\boxed{F} * r \mapsto 1 * \boxed{F}\}$ as $\boxed{S} * \boxed{F} \rhd \ulcorner\mathsf{False}\urcorner$

`assert (!r = 1)`

Standard resource algebra constructions

Resource algebra product

The **product** of two RA $(A, \cdot_A, \mathcal{V}_A)$ and $(B, \cdot_B, \mathcal{V}_B)$ is defined as $(A \times B, \cdot_{A\times B}, \mathcal{V}_{A\times B})$ where

$$(a, b) \cdot_{A\times B} (a', b') \triangleq (a \cdot_A a', b \cdot_B b')$$
$$\mathcal{V}_{A\times B}((a, b)) \triangleq \mathcal{V}_A(a) \wedge \mathcal{V}_B(b)$$

Resource algebra product

The **product** of two RA A and B is defined as $A \times B$ with

$$(a, b) \cdot (a', b') \triangleq (a \cdot a', b \cdot b')$$
$$\mathcal{V}((a, b)) \triangleq \mathcal{V}(a) \wedge \mathcal{V}(b)$$

Resource algebra product

The **product** of two RA A and B is defined as $A \times B$ with

$$(a, b) \cdot (a', b') \triangleq (a \cdot a', b \cdot b')$$
$$\mathcal{V}((a, b)) \triangleq \mathcal{V}(a) \wedge \mathcal{V}(b)$$

Property: frame-preserving update is pointwise:

$$\frac{a \rightsquigarrow a' \qquad b \rightsquigarrow b'}{(a, b) \rightsquigarrow (a', b')}$$

Option resource algebra

Option of an resource algebra A, with carrier:

```
option A := None | Some of A
```

and operations:

$- \cdot -$	None	Some(a)
None	None	Some(a)
Some(b)	Some(b)	Some$(a \cdot b)$

$\mathcal{V}(-)$	True	$\mathcal{V}(a)$

Option resource algebra

Option of an resource algebra A, with carrier:

```
option A := None | Some of A
```

and operations:

$- \cdot -$	None	Some(a)
None	None	Some(a)
Some(b)	Some(b)	Some$(a \cdot b)$

$\mathcal{V}(-)$	True	$\mathcal{V}(a)$

Properties of frame-preserving update:

$$\frac{a \rightsquigarrow b}{\texttt{Some}(a) \rightsquigarrow \texttt{Some}(b)} \qquad \frac{}{\texttt{Some}(a) \rightsquigarrow \texttt{None}}$$

Option resource algebra

Option of an resource algebra A, with carrier:

```
option A := None | Some of A
```

and operations:

$- \cdot -$	None	Some(a)
None	None	Some(a)
Some(b)	Some(b)	Some$(a \cdot b)$

$\mathcal{V}(-)$	True	$\mathcal{V}(a)$

Properties of frame-preserving update:

$$\frac{a \rightsquigarrow b}{\texttt{Some}(a) \rightsquigarrow \texttt{Some}(b)} \qquad \frac{}{\texttt{Some}(a) \rightsquigarrow \texttt{None}}$$

When clear from context we write ϵ for the unit None and a for Some(a).

Resource algebra exercise: $\mathcal{H}_{\{0,1\}}$

Exercise: (ra_H.v) Give a RA $(\mathcal{H}_{\{0,1\}}, \cdot, \mathcal{V})$ with $\mathsf{h}_0, \mathsf{h}_1 \in \mathcal{H}_{\{0,1\}}$ s.t.

$$\mathcal{V}(\mathsf{h}_0 \cdot \mathsf{h}_0) \qquad \mathsf{h}_0 \cdot \mathsf{h}_0 \rightsquigarrow \mathsf{h}_1 \cdot \mathsf{h}_1 \qquad \forall i, j \in \{0,1\} \;\; \mathcal{V}(\mathsf{h}_i \cdot \mathsf{h}_j) \Rightarrow i = j$$

Resource algebra exercise: $\mathcal{H}_{\{0,1\}}$

$$a \cdot (b \cdot c) = (a \cdot b) \cdot c \qquad a \cdot b = b \cdot a \qquad \frac{\mathcal{V}(a \cdot b)}{\mathcal{V}(a)}\ \mathcal{V}{\downarrow}$$

$$1: \ \mathcal{V}(\mathsf{h}_0 \cdot \mathsf{h}_0) \qquad 2: \ \mathsf{h}_0 \cdot \mathsf{h}_0 \rightsquigarrow \mathsf{h}_1 \cdot \mathsf{h}_1 \qquad 3: \ \mathcal{V}(\mathsf{h}_i \cdot \mathsf{h}_j) \Rightarrow i = j$$

$\cdot$	h_0	h_1		
h_0				
h_1				
$\mathcal{V}(-)$				

Resource algebra exercise: $\mathcal{H}_{\{0,1\}}$

$$a \cdot (b \cdot c) = (a \cdot b) \cdot c \qquad a \cdot b = b \cdot a \qquad \frac{\mathcal{V}(a \cdot b)}{\mathcal{V}(a)}\ \mathcal{V}{\downarrow}$$

$$1 :\ \mathcal{V}(\mathsf{h}_0 \cdot \mathsf{h}_0) \qquad 2 :\ \mathsf{h}_0 \cdot \mathsf{h}_0 \rightsquigarrow \mathsf{h}_1 \cdot \mathsf{h}_1 \qquad 3 :\ \mathcal{V}(\mathsf{h}_i \cdot \mathsf{h}_j) \Rightarrow i = j$$

$\cdot$	h_0	h_1		
h_0				
h_1				
$\mathcal{V}(-)$	True			

by 1,$\mathcal{V}{\downarrow}$;

Resource algebra exercise: $\mathcal{H}_{\{0,1\}}$

$$a \cdot (b \cdot c) = (a \cdot b) \cdot c \qquad a \cdot b = b \cdot a \qquad \frac{\mathcal{V}(a \cdot b)}{\mathcal{V}(a)}\ \mathcal{V}{\downarrow}$$

$$1: \ \mathcal{V}(\mathsf{h}_0 \cdot \mathsf{h}_0) \qquad 2: \ \mathsf{h}_0 \cdot \mathsf{h}_0 \rightsquigarrow \mathsf{h}_1 \cdot \mathsf{h}_1 \qquad 3: \ \mathcal{V}(\mathsf{h}_i \cdot \mathsf{h}_j) \Rightarrow i = j$$

$\cdot$	h_0	h_1		
h_0				
h_1				
$\mathcal{V}(-)$	True	True		

by 1,$\mathcal{V}{\downarrow}$; by 1,2,$\mathcal{V}{\downarrow}$;

Resource algebra exercise: $\mathcal{H}_{\{0,1\}}$

$$a \cdot (b \cdot c) = (a \cdot b) \cdot c \qquad a \cdot b = b \cdot a \qquad \frac{\mathcal{V}(a \cdot b)}{\mathcal{V}(a)}\ \mathcal{V}{\downarrow}$$

$$1: \ \mathcal{V}(\mathsf{h}_0 \cdot \mathsf{h}_0) \qquad 2: \ \mathsf{h}_0 \cdot \mathsf{h}_0 \rightsquigarrow \mathsf{h}_1 \cdot \mathsf{h}_1 \qquad 3: \ \mathcal{V}(\mathsf{h}_i \cdot \mathsf{h}_j) \Rightarrow i = j$$

$\cdot$	h_0	h_1	↯	
h_0		↯		
h_1	↯			
↯				
$\mathcal{V}(-)$	True	True	False	

by 1,$\mathcal{V}{\downarrow}$; by 1,2,$\mathcal{V}{\downarrow}$; by 3rd;

Resource algebra exercise: $\mathcal{H}_{\{0,1\}}$

$$a \cdot (b \cdot c) = (a \cdot b) \cdot c \qquad a \cdot b = b \cdot a \qquad \frac{\mathcal{V}(a \cdot b)}{\mathcal{V}(a)}\ \mathcal{V}{\downarrow}$$

$$1: \ \mathcal{V}(\mathsf{h}_0 \cdot \mathsf{h}_0) \qquad 2: \ \mathsf{h}_0 \cdot \mathsf{h}_0 \rightsquigarrow \mathsf{h}_1 \cdot \mathsf{h}_1 \qquad 3: \ \mathcal{V}(\mathsf{h}_i \cdot \mathsf{h}_j) \Rightarrow i = j$$

$\cdot$	h_0	h_1	↯	
h_0		↯	↯	
h_1	↯		↯	
↯	↯	↯	↯	↯
			↯	
$\mathcal{V}(-)$	True	True	False	

by 1,$\mathcal{V}{\downarrow}$; by 1,2,$\mathcal{V}{\downarrow}$; by 3rd; by $\mathcal{V}{\downarrow}$; Choice for $\mathsf{h}_0 \cdot \mathsf{h}_0$?

Resource algebra exercise: $\mathcal{H}_{\{0,1\}}$

$$a \cdot (b \cdot c) = (a \cdot b) \cdot c \qquad a \cdot b = b \cdot a \qquad \frac{\mathcal{V}(a \cdot b)}{\mathcal{V}(a)}\ \mathcal{V}{\downarrow}$$

$$1: \ \mathcal{V}(\mathsf{h}_0 \cdot \mathsf{h}_0) \qquad 2: \ \mathsf{h}_0 \cdot \mathsf{h}_0 \rightsquigarrow \mathsf{h}_1 \cdot \mathsf{h}_1 \qquad 3: \ \mathcal{V}(\mathsf{h}_i \cdot \mathsf{h}_j) \Rightarrow i = j$$

$\cdot$	h_0	h_1	↯	
h_0		↯	↯	
h_1	↯		↯	
↯	↯	↯	↯	↯
			↯	
$\mathcal{V}(-)$	True	True	False	

by 1,$\mathcal{V}{\downarrow}$; by 1,2,$\mathcal{V}{\downarrow}$; by 3rd; by $\mathcal{V}{\downarrow}$; Choice for $\mathsf{h}_0 \cdot \mathsf{h}_0$? not h_0 otherwise
$\mathsf{h}_0 \cdot \mathsf{h}_0 = \mathsf{h}_0 \cdot \mathsf{h}_0 \cdot \mathsf{h}_0 \rightsquigarrow \mathsf{h}_1 \cdot \mathsf{h}_1 \cdot \mathsf{h}_0 \succcurlyeq \mathsf{h}_1 \cdot \mathsf{h}_0 =$ ↯ are valid

Resource algebra exercise: $\mathcal{H}_{\{0,1\}}$

$$a \cdot (b \cdot c) = (a \cdot b) \cdot c \qquad a \cdot b = b \cdot a \qquad \frac{\mathcal{V}(a \cdot b)}{\mathcal{V}(a)}\ \mathcal{V}{\downarrow}$$

$$1: \ \mathcal{V}(\mathsf{h}_0 \cdot \mathsf{h}_0) \qquad 2: \ \mathsf{h}_0 \cdot \mathsf{h}_0 \rightsquigarrow \mathsf{h}_1 \cdot \mathsf{h}_1 \qquad 3: \ \mathcal{V}(\mathsf{h}_i \cdot \mathsf{h}_j) \Rightarrow i = j$$

$\cdot$	h_0	h_1	↯	
h_0		↯	↯	
h_1	↯		↯	
↯	↯	↯	↯	↯
			↯	
$\mathcal{V}(-)$	True	True	False	

by 1,$\mathcal{V}{\downarrow}$; by 1,2,$\mathcal{V}{\downarrow}$; by 3rd; by $\mathcal{V}{\downarrow}$; Choice for $\mathsf{h}_0 \cdot \mathsf{h}_0$? not h_0 otherwise
$\mathsf{h}_0 \cdot \mathsf{h}_0 = \mathsf{h}_0 \cdot \mathsf{h}_0 \cdot \mathsf{h}_0 \rightsquigarrow \mathsf{h}_1 \cdot \mathsf{h}_1 \cdot \mathsf{h}_0 \geqslant \mathsf{h}_1 \cdot \mathsf{h}_0 =$ ↯ are valid
Not h_1 otherwise $\mathsf{h}_1 \cdot \mathsf{h}_1 = \mathsf{h}_0 \cdot \mathsf{h}_0 \cdot \mathsf{h}_1$ is valid;

Resource algebra exercise: $\mathcal{H}_{\{0,1\}}$

$$a \cdot (b \cdot c) = (a \cdot b) \cdot c \qquad a \cdot b = b \cdot a \qquad \frac{\mathcal{V}(a \cdot b)}{\mathcal{V}(a)}\ \mathcal{V}{\downarrow}$$

$$1: \ \mathcal{V}(\mathsf{h}_0 \cdot \mathsf{h}_0) \qquad 2: \ \mathsf{h}_0 \cdot \mathsf{h}_0 \rightsquigarrow \mathsf{h}_1 \cdot \mathsf{h}_1 \qquad 3: \ \mathcal{V}(\mathsf{h}_i \cdot \mathsf{h}_j) \Rightarrow i = j$$

$\cdot$	h_0	h_1	↯	ok
h_0	ok	↯	↯	
h_1	↯		↯	
↯	↯	↯	↯	↯
ok			↯	

$\mathcal{V}(-)$	True	True	False	True

by 1,$\mathcal{V}{\downarrow}$; by 1,2,$\mathcal{V}{\downarrow}$; by 3rd; by $\mathcal{V}{\downarrow}$; Choice for $\mathsf{h}_0 \cdot \mathsf{h}_0$? not h_0 otherwise
$\mathsf{h}_0 \cdot \mathsf{h}_0 = \mathsf{h}_0 \cdot \mathsf{h}_0 \cdot \mathsf{h}_0 \rightsquigarrow \mathsf{h}_1 \cdot \mathsf{h}_1 \cdot \mathsf{h}_0 \succcurlyeq \mathsf{h}_1 \cdot \mathsf{h}_0 =$ ↯ are valid
Not h_1 otherwise $\mathsf{h}_1 \cdot \mathsf{h}_1 = \mathsf{h}_0 \cdot \mathsf{h}_0 \cdot \mathsf{h}_1$ is valid;

Resource algebra exercise: $\mathcal{H}_{\{0,1\}}$

$$a \cdot (b \cdot c) = (a \cdot b) \cdot c \qquad a \cdot b = b \cdot a \qquad \frac{\mathcal{V}(a \cdot b)}{\mathcal{V}(a)}\ \mathcal{V}{\downarrow}$$

$$1: \ \mathcal{V}(\mathsf{h}_0 \cdot \mathsf{h}_0) \qquad 2: \ \mathsf{h}_0 \cdot \mathsf{h}_0 \rightsquigarrow \mathsf{h}_1 \cdot \mathsf{h}_1 \qquad 3: \ \mathcal{V}(\mathsf{h}_i \cdot \mathsf{h}_j) \Rightarrow i = j$$

$\cdot$	h_0	h_1	↯	ok
h_0	ok	↯	↯	↯
h_1	↯		↯	↯
↯	↯	↯	↯	↯
ok	↯	↯	↯	↯

$\mathcal{V}(-)$	True	True	False	True

by 1,$\mathcal{V}{\downarrow}$; by 1,2,$\mathcal{V}{\downarrow}$; by 3rd; by $\mathcal{V}{\downarrow}$; Choice for $\mathsf{h}_0 \cdot \mathsf{h}_0$? not h_0 otherwise
$\mathsf{h}_0 \cdot \mathsf{h}_0 = \mathsf{h}_0 \cdot \mathsf{h}_0 \cdot \mathsf{h}_0 \rightsquigarrow \mathsf{h}_1 \cdot \mathsf{h}_1 \cdot \mathsf{h}_0 \geqslant \mathsf{h}_1 \cdot \mathsf{h}_0 =$ ↯ are valid
Not h_1 otherwise $\mathsf{h}_1 \cdot \mathsf{h}_1 = \mathsf{h}_0 \cdot \mathsf{h}_0 \cdot \mathsf{h}_1$ is valid; similarly for $ok \cdot -$

Resource algebra exercise: $\mathcal{H}_{\{0,1\}}$

$$a \cdot (b \cdot c) = (a \cdot b) \cdot c \qquad a \cdot b = b \cdot a \qquad \frac{\mathcal{V}(a \cdot b)}{\mathcal{V}(a)}\ \mathcal{V}{\downarrow}$$

$$1: \ \mathcal{V}(\mathsf{h}_0 \cdot \mathsf{h}_0) \qquad 2: \ \mathsf{h}_0 \cdot \mathsf{h}_0 \rightsquigarrow \mathsf{h}_1 \cdot \mathsf{h}_1 \qquad 3: \ \mathcal{V}(\mathsf{h}_i \cdot \mathsf{h}_j) \Rightarrow i = j$$

$\cdot$	h_0	h_1	$\lightning$	ok
h_0	ok	$\lightning$	$\lightning$	$\lightning$
h_1	$\lightning$	ok (or h_1)	$\lightning$	$\lightning$
$\lightning$	$\lightning$	$\lightning$	$\lightning$	$\lightning$
ok	$\lightning$	$\lightning$	$\lightning$	$\lightning$
$\mathcal{V}(-)$	True	True	False	True

by 1,$\mathcal{V}{\downarrow}$; by 1,2,$\mathcal{V}{\downarrow}$; by 3rd; by $\mathcal{V}{\downarrow}$; Choice for $\mathsf{h}_0 \cdot \mathsf{h}_0$? not h_0 otherwise
$\mathsf{h}_0 \cdot \mathsf{h}_0 = \mathsf{h}_0 \cdot \mathsf{h}_0 \cdot \mathsf{h}_0 \rightsquigarrow \mathsf{h}_1 \cdot \mathsf{h}_1 \cdot \mathsf{h}_0 \geqslant \mathsf{h}_1 \cdot \mathsf{h}_0 = \lightning$ are valid
Not h_1 otherwise $\mathsf{h}_1 \cdot \mathsf{h}_1 = \mathsf{h}_0 \cdot \mathsf{h}_0 \cdot \mathsf{h}_1$ is valid; similarly for $ok \cdot -$

Generalization

More generally $\mathcal{H}_X$ defined for any set X:

$$\mathcal{H}_X ::= (\mathsf{h}_x)_{x \in X} \mid \lightning \mid ok$$

If $x \neq y$:

$\cdot$	h_x	h_y	$\lightning$	ok
h_x	ok	$\lightning$	$\lightning$	$\lightning$
h_y	$\lightning$	ok	$\lightning$	$\lightning$
$\lightning$	$\lightning$	$\lightning$	$\lightning$	$\lightning$
ok	$\lightning$	$\lightning$	$\lightning$	$\lightning$

$\mathcal{V}(-)$	True	True	False	True

Similar laws

$$\mathcal{V}(\mathsf{h}_x \cdot \mathsf{h}_x) \qquad \mathsf{h}_x \cdot \mathsf{h}_x \rightsquigarrow \mathsf{h}_y \cdot \mathsf{h}_y \qquad \mathcal{V}(\mathsf{h}_x \cdot \mathsf{h}_y) \Rightarrow x = y$$

Resource algebra for `incr r || incr r`

$\texttt{option}\ \mathcal{H}_{\{0,1\}} \times \texttt{option}\ \mathcal{H}_{\{0,1\}}$

$\mathcal{V}((\mathsf{h}_0 \cdot \mathsf{h}_0, \mathsf{h}_0 \cdot \mathsf{h}_0))$ $\quad$ $\mathcal{V}((\mathsf{h}_x \cdot \mathsf{h}_y, \epsilon)) \Rightarrow x = y$ $\quad$ $(\mathsf{h}_x, \mathsf{h}_y) = (\mathsf{h}_x, \epsilon) \cdot (\epsilon, \mathsf{h}_y)$

$(\mathsf{h}_0 \cdot \mathsf{h}_0, \epsilon) \rightsquigarrow (\mathsf{h}_1 \cdot \mathsf{h}_1, \epsilon)$ $\quad$ $(\epsilon, \mathsf{h}_0 \cdot \mathsf{h}_0) \rightsquigarrow (\epsilon, \mathsf{h}_1 \cdot \mathsf{h}_1)$

Exercise: Give a lock invariant sufficient to prove the program below:

```
    let r = ref 0
    let l = newlock ()
acquire l;    || acquire l;
r := !r + 1; || r := !r + 1;
release l     || release l
    acquire l;
    assert (!r = 2)
```

Resource algebra for `incr r || incr r`

$\texttt{option}\ \mathcal{H}_{\{0,1\}} \times \texttt{option}\ \mathcal{H}_{\{0,1\}}$

$\mathcal{V}((\mathsf{h}_0 \cdot \mathsf{h}_0, \mathsf{h}_0 \cdot \mathsf{h}_0))$ $\quad$ $\mathcal{V}((\mathsf{h}_x \cdot \mathsf{h}_y, \epsilon)) \Rightarrow x = y$ $\quad$ $(\mathsf{h}_x, \mathsf{h}_y) = (\mathsf{h}_x, \epsilon) \cdot (\epsilon, \mathsf{h}_y)$

$(\mathsf{h}_0 \cdot \mathsf{h}_0, \epsilon) \rightsquigarrow (\mathsf{h}_1 \cdot \mathsf{h}_1, \epsilon)$ $\quad$ $(\epsilon, \mathsf{h}_0 \cdot \mathsf{h}_0) \rightsquigarrow (\epsilon, \mathsf{h}_1 \cdot \mathsf{h}_1)$

Exercise: Give a lock invariant sufficient to prove the program below:

```
    let r = ref 0
    let l = newlock ()
acquire l;    || acquire l;
r := !r + 1; || r := !r + 1;
release l     || release l
    acquire l;
    assert (!r = 2)
```

$$R \equiv \exists ij.\, r \mapsto i + j * \boxed{(\mathsf{h}_i, \mathsf{h}_j)}$$

Modularity?

Fixing the resource algebra once lacks modularity, making it tedious to:

- handle more threads e.g. $(\epsilon, \epsilon, h_0)$
- compose sequentially/reuse proofs $(h_1, h_1, h_1, \text{start of next proof})$
- allow different resource algebras, etc.

Modularity?

Fixing the resource algebra once lacks modularity, making it tedious to:

- handle more threads e.g. $(\epsilon, \epsilon, \mathsf{h}_0)$
- compose sequentially/reuse proofs $(\mathsf{h}_1, \mathsf{h}_1, \mathsf{h}_1, \text{start of next proof})$
- allow different resource algebras, etc.

We need several instances of a given resource algebra, names for those instances, to allow of several different resource algebras, ...

Modularity?

Fixing the resource algebra once lacks modularity, making it tedious to:

- handle more threads e.g. $(\epsilon, \epsilon, \mathsf{h}_0)$
- compose sequentially/reuse proofs $(\mathsf{h}_1, \mathsf{h}_1, \mathsf{h}_1, \text{start of next proof})$
- allow different resource algebras, etc.

We need several instances of a given resource algebra, names for those instances, to allow of several different resource algebras, ...

Answer: package all of that into one resource algebra.

Resource algebra of functions

If $\mathcal{M}$ is a RA and X is any set, then $X \to \mathcal{M}$ (or $\mathcal{M}^X$) is a RA:

$$(f \cdot g)(x) \triangleq \lambda x.f(x) \cdot g(x) \qquad \mathcal{V}(f) \triangleq \forall x.\mathcal{V}(f(x)) \qquad \frac{f(x) \rightsquigarrow a}{f \rightsquigarrow f[x := a]}$$

and so is the set of partial functions $X \rightharpoonup \mathcal{M}$.

Resource algebra of functions

If $\mathcal{M}$ is a RA and X is any set, then $X \to \mathcal{M}$ (or $\mathcal{M}^X$) is a RA:

$$(f \cdot g)(x) \triangleq \lambda x. f(x) \cdot g(x) \qquad \mathcal{V}(f) \triangleq \forall x. \mathcal{V}(f(x)) \qquad \frac{f(x) \rightsquigarrow a}{f \rightsquigarrow f[x := a]}$$

and so is the set of partial functions $X \rightharpoonup \mathcal{M}$. This lets us talk about the singleton partial function even if $\mathcal{M}$ has no unit:

$$\boxed{a}^{\gamma} \triangleq \boxed{\{(\gamma, a)\}}$$

$$\begin{aligned} \boxed{a}^{\gamma} * \boxed{b}^{\gamma} &\Leftrightarrow \boxed{\{(\gamma, a)\} \cdot \{(\gamma, b)\}} \\ &= \boxed{\{(\gamma, a \cdot b)\}} \\ &= \boxed{a \cdot b}^{\gamma} \end{aligned}$$

This is the solution used to avoid conflict between resources used for different purposes; γ is called a *ghost name*.

Resource algebra of functions

If $\mathcal{M}$ is a RA and X is any set, then $X \to \mathcal{M}$ (or $\mathcal{M}^X$) is a RA:

$$(f \cdot g)(x) \triangleq \lambda x. f(x) \cdot g(x) \qquad \mathcal{V}(f) \triangleq \forall x. \mathcal{V}(f(x)) \qquad \frac{f(x) \rightsquigarrow a}{f \rightsquigarrow f[x := a]}$$

and so is the set of partial functions $X \rightharpoonup \mathcal{M}$. This lets us talk about the singleton partial function even if $\mathcal{M}$ has no unit:

$$\boxed{a}^{\gamma} \triangleq \boxed{\{(\gamma, a)\}}$$

$$\begin{aligned} \boxed{a}^{\gamma} * \boxed{b}^{\gamma} &\Leftrightarrow \boxed{\{(\gamma, a)\} \cdot \{(\gamma, b)\}} \\ &= \boxed{\{(\gamma, a \cdot b)\}} \\ &= \boxed{a \cdot b}^{\gamma} \end{aligned}$$

This is the solution used to avoid conflict between resources used for different purposes; γ is called a *ghost name*.

In Iris $X = \mathbb{N}^*$ is called `gname` and $\{(x, a)\}$ is written `{[ x := a ]}` and the "global" unnamed ghost $\boxed{f}$ is noted Own(f).

Frame preservation vs. ghost creation

In order to create any new ghost resource $\boxed{g}^{\gamma}$, we would need:

$$\lceil\rceil \Rrightarrow \boxed{g}^{\gamma} \qquad \text{i.e.} \qquad \varnothing \rightsquigarrow \{(\gamma, g)\}$$

Frame preservation vs. ghost creation

In order to create any new ghost resource $\boxed{g}^{\gamma}$, we would need:

$$\ulcorner \urcorner \Rrightarrow \boxed{g}^{\gamma} \qquad \text{i.e.} \qquad \varnothing \rightsquigarrow \{(\gamma, g)\}$$

but it is impossible.

Exercise: In the RA of partial functions $\mathbb{N} \rightharpoonup \mathcal{M}_{\mathsf{sf}}$ show $\forall \gamma.\ \neg(\varnothing \rightsquigarrow \{(\gamma, S)\})$.

Frame preservation vs. ghost creation

In order to create any new ghost resource $\boxed{g}^{\gamma}$, we would need:

$$\ulcorner\urcorner \Rrightarrow \boxed{g}^{\gamma} \qquad \text{i.e.} \qquad \varnothing \rightsquigarrow \{(\gamma, g)\}$$

but it is impossible.

Exercise: In the RA of partial functions $\mathbb{N} \rightharpoonup \mathcal{M}_{\mathsf{sf}}$ show $\forall \gamma.\ \neg(\varnothing \rightsquigarrow \{(\gamma, S)\})$.

If, by contradiction $\varnothing \rightsquigarrow \{(\gamma, S)\}$ i.e. $\forall f.\ \mathcal{V}(f \cdot \varnothing) \Rightarrow \mathcal{V}(f \cdot \{(\gamma, S)\})$, then in particular with $f = \{(\gamma, S)\}$ we have:

- $\mathcal{V}(f \cdot \varnothing) = \mathcal{V}(\{(\gamma, S)\}) = \mathcal{V}(S) =$ True
- $\mathcal{V}(f \cdot \{(\gamma, S)\}) = \mathcal{V}(\{(\gamma, S \cdot S)\}) = \mathcal{V}(S \cdot S) = \mathcal{V}(\lightning) =$ False

True $\Rightarrow$ False, contradiction.

Frame preservation and ghost allocation

$a \rightsquigarrow b$ is generalized to $a \rightsquigarrow B$ where B is a set $B \subseteq \mathcal{M}$:

$$a \rightsquigarrow B \triangleq \forall c^? \in \mathcal{M}^? \ \mathcal{V}(a \cdot c^?) \Rightarrow \exists b \in B \ \mathcal{V}(b \cdot c^?)$$

$$\mathcal{M}^? \triangleq \bot \uplus \mathcal{M} \qquad a \cdot \bot \triangleq a$$

$a \rightsquigarrow b$ is redefined as simply $a \rightsquigarrow \{b\}$.

Frame preservation and ghost allocation

$a \rightsquigarrow b$ is generalized to $a \rightsquigarrow B$ where B is a set $B \subseteq \mathcal{M}$:

$$a \rightsquigarrow B \triangleq \forall c^? \in \mathcal{M}^? \ \mathcal{V}(a \cdot c^?) \Rightarrow \exists b \in B \ \mathcal{V}(b \cdot c^?)$$

$$\mathcal{M}^? \triangleq \bot \uplus \mathcal{M} \qquad a \cdot \bot \triangleq a$$

$a \rightsquigarrow b$ is redefined as simply $a \rightsquigarrow \{b\}$.

If B is infinite and all possible frames $c^?$ are finite, then there is necessarily an element in B that is not in the frame.

Frame preservation and ghost allocation

$a \rightsquigarrow b$ is generalized to $a \rightsquigarrow B$ where B is a set $B \subseteq \mathcal{M}$:

$$a \rightsquigarrow B \triangleq \forall c^? \in \mathcal{M}^?\ \mathcal{V}(a \cdot c^?) \Rightarrow \exists b \in B\ \mathcal{V}(b \cdot c^?)$$

$$\mathcal{M}^? \triangleq \bot \uplus \mathcal{M} \qquad a \cdot \bot \triangleq a$$

$a \rightsquigarrow b$ is redefined as simply $a \rightsquigarrow \{b\}$.

If B is infinite and all possible frames $c^?$ are finite, then there is necessarily an element in B that is not in the frame.

Choosing **finite** partial functions $\mathbb{N} \xrightarrow{\text{fin}} \mathcal{M}$, we finally have:

$$\frac{\mathcal{V}(a)}{\varnothing \rightsquigarrow \{\{(\gamma, a)\} \mid \gamma \in \mathbb{N}\}} \qquad \frac{\mathcal{V}(a)}{\mathsf{True} \Rrightarrow \exists \gamma\ \boxed{a}^{\gamma}}\ \textsc{ghost-alloc}$$

Called *allocation* because we need to find a unused ghost name γ.

Proof of `incr r || incr r` with ghost allocation

$\{\ulcorner\urcorner\}$; by GHOST-ALLOC, SINCE $\mathsf{h}_0 \cdot \mathsf{h}_0$ IS VALID:

Proof of `incr r || incr r` with ghost allocation

$\{\ulcorner\urcorner\}$; by GHOST-ALLOC, SINCE $\mathsf{h}_0 \cdot \mathsf{h}_0$ IS VALID:
$\{\exists\gamma.\ \boxed{\mathsf{h}_0 \cdot \mathsf{h}_0}^{\gamma}\}$, we prove for all γ_1:

Proof of `incr r || incr r` with ghost allocation

$\{\ulcorner\urcorner\}$; by GHOST-ALLOC, SINCE $\mathsf{h}_0 \cdot \mathsf{h}_0$ IS VALID:

$\{\exists\gamma.\ \boxed{\mathsf{h}_0 \cdot \mathsf{h}_0}^{\gamma}\}$, we prove for all γ_1:

$\{\boxed{\mathsf{h}_0 \cdot \mathsf{h}_0}^{\gamma_1}\}$; by GHOST-ALLOC:

Proof of `incr r || incr r` with ghost allocation

$\{\ulcorner\urcorner\}$; by GHOST-ALLOC, SINCE $\mathsf{h}_0 \cdot \mathsf{h}_0$ IS VALID:
$\{\exists\gamma.\ \boxed{\mathsf{h}_0 \cdot \mathsf{h}_0}^{\gamma}\}$, we prove for all γ_1:
$\{\boxed{\mathsf{h}_0 \cdot \mathsf{h}_0}^{\gamma_1}\}$; by GHOST-ALLOC:
$\{\boxed{\mathsf{h}_0 \cdot \mathsf{h}_0}^{\gamma_1} * \exists\gamma.\ \boxed{\mathsf{h}_0 \cdot \mathsf{h}_0}^{\gamma}\}$; we prove for all γ_2:

Proof of `incr r || incr r` with ghost allocation

$\{\ulcorner\urcorner\}$; by GHOST-ALLOC, SINCE $\mathsf{h}_0 \cdot \mathsf{h}_0$ IS VALID:

$\{\exists\gamma.\ \boxed{\mathsf{h}_0 \cdot \mathsf{h}_0}^{\gamma}\}$, we prove for all γ_1:

$\{\boxed{\mathsf{h}_0 \cdot \mathsf{h}_0}^{\gamma_1}\}$; by GHOST-ALLOC:

$\{\boxed{\mathsf{h}_0 \cdot \mathsf{h}_0}^{\gamma_1} * \exists\gamma.\ \boxed{\mathsf{h}_0 \cdot \mathsf{h}_0}^{\gamma}\}$; we prove for all γ_2:

$\{\boxed{\mathsf{h}_0 \cdot \mathsf{h}_0}^{\gamma_1} * \boxed{\mathsf{h}_0 \cdot \mathsf{h}_0}^{\gamma_2}\}$

Proof of `incr r || incr r` with ghost allocation

$\{\ulcorner\urcorner\}$; by GHOST-ALLOC, SINCE $\mathsf{h}_0 \cdot \mathsf{h}_0$ IS VALID:

$\{\exists\gamma.\ \boxed{\mathsf{h}_0 \cdot \mathsf{h}_0}^{\gamma}\}$, we prove for all γ_1:

$\{\boxed{\mathsf{h}_0 \cdot \mathsf{h}_0}^{\gamma_1}\}$; by GHOST-ALLOC:

$\{\boxed{\mathsf{h}_0 \cdot \mathsf{h}_0}^{\gamma_1} * \exists\gamma.\ \boxed{\mathsf{h}_0 \cdot \mathsf{h}_0}^{\gamma}\}$; we prove for all γ_2:

$\{\boxed{\mathsf{h}_0 \cdot \mathsf{h}_0}^{\gamma_1} * \boxed{\mathsf{h}_0 \cdot \mathsf{h}_0}^{\gamma_2}\}$

```
let r = ref 0
```

$\{\boxed{\mathsf{h}_0}^{\gamma_1} * \boxed{\mathsf{h}_0}^{\gamma_1} * \boxed{\mathsf{h}_0}^{\gamma_2} * \boxed{\mathsf{h}_0}^{\gamma_2} * r \mapsto 0\}$

Proof of `incr r || incr r` with ghost allocation

$\{\ulcorner\urcorner\}$; by GHOST-ALLOC, SINCE $\mathsf{h}_0 \cdot \mathsf{h}_0$ IS VALID:

$\{\exists\gamma.\ \boxed{\mathsf{h}_0 \cdot \mathsf{h}_0}^{\gamma}\}$, we prove for all γ_1:

$\{\boxed{\mathsf{h}_0 \cdot \mathsf{h}_0}^{\gamma_1}\}$; by GHOST-ALLOC:

$\{\boxed{\mathsf{h}_0 \cdot \mathsf{h}_0}^{\gamma_1} * \exists\gamma.\ \boxed{\mathsf{h}_0 \cdot \mathsf{h}_0}^{\gamma}\}$; we prove for all γ_2:

$\{\boxed{\mathsf{h}_0 \cdot \mathsf{h}_0}^{\gamma_1} * \boxed{\mathsf{h}_0 \cdot \mathsf{h}_0}^{\gamma_2}\}$

```
let r = ref 0
```

$\{\boxed{\mathsf{h}_0}^{\gamma_1} * \boxed{\mathsf{h}_0}^{\gamma_1} * \boxed{\mathsf{h}_0}^{\gamma_2} * \boxed{\mathsf{h}_0}^{\gamma_2} * r \mapsto 0\}$

we choose $R \triangleq$

Proof of `incr r || incr r` with ghost allocation

$\{\ulcorner\urcorner\}$; by GHOST-ALLOC, SINCE $\mathsf{h}_0 \cdot \mathsf{h}_0$ IS VALID:

$\{\exists\gamma.\ \boxed{\mathsf{h}_0 \cdot \mathsf{h}_0}^{\gamma}\}$, we prove for all γ_1:

$\{\boxed{\mathsf{h}_0 \cdot \mathsf{h}_0}^{\gamma_1}\}$; by GHOST-ALLOC:

$\{\boxed{\mathsf{h}_0 \cdot \mathsf{h}_0}^{\gamma_1} * \exists\gamma.\ \boxed{\mathsf{h}_0 \cdot \mathsf{h}_0}^{\gamma}\}$; we prove for all γ_2:

$\{\boxed{\mathsf{h}_0 \cdot \mathsf{h}_0}^{\gamma_1} * \boxed{\mathsf{h}_0 \cdot \mathsf{h}_0}^{\gamma_2}\}$

```
let r = ref 0
```

$\{\boxed{\mathsf{h}_0}^{\gamma_1} * \boxed{\mathsf{h}_0}^{\gamma_1} * \boxed{\mathsf{h}_0}^{\gamma_2} * \boxed{\mathsf{h}_0}^{\gamma_2} * r \mapsto 0\}$

we choose $R \triangleq \exists ij.\ r \mapsto i + j * \boxed{\mathsf{h}_i}^{\gamma_1} * \boxed{\mathsf{h}_j}^{\gamma_2}$

$\{\boxed{\mathsf{h}_0}^{\gamma_1} * \boxed{\mathsf{h}_0}^{\gamma_2} * R\}$

Proof of `incr r || incr r` with ghost allocation

$\{\ulcorner\urcorner\}$; by GHOST-ALLOC, SINCE $\mathsf{h}_0 \cdot \mathsf{h}_0$ IS VALID:

$\{\exists\gamma.\ \boxed{\mathsf{h}_0 \cdot \mathsf{h}_0}^{\gamma}\}$, we prove for all γ_1:

$\{\boxed{\mathsf{h}_0 \cdot \mathsf{h}_0}^{\gamma_1}\}$; by GHOST-ALLOC:

$\{\boxed{\mathsf{h}_0 \cdot \mathsf{h}_0}^{\gamma_1} * \exists\gamma.\ \boxed{\mathsf{h}_0 \cdot \mathsf{h}_0}^{\gamma}\}$; we prove for all γ_2:

$\{\boxed{\mathsf{h}_0 \cdot \mathsf{h}_0}^{\gamma_1} * \boxed{\mathsf{h}_0 \cdot \mathsf{h}_0}^{\gamma_2}\}$

```
let r = ref 0
```

$\{\boxed{\mathsf{h}_0}^{\gamma_1} * \boxed{\mathsf{h}_0}^{\gamma_1} * \boxed{\mathsf{h}_0}^{\gamma_2} * \boxed{\mathsf{h}_0}^{\gamma_2} * r \mapsto 0\}$

we choose $R \triangleq \exists ij.\ r \mapsto i + j * \boxed{\mathsf{h}_i}^{\gamma_1} * \boxed{\mathsf{h}_j}^{\gamma_2}$

$\{\boxed{\mathsf{h}_0}^{\gamma_1} * \boxed{\mathsf{h}_0}^{\gamma_2} * R\}$

```
let l = newlock ()
```

$\{\boxed{\mathsf{h}_0}^{\gamma_1} * \boxed{\mathsf{h}_0}^{\gamma_2} * l \rightsquigarrow \mathsf{Lock}\, R\}$

Proof of `incr r || incr r` with ghost allocation

$\{\ulcorner\urcorner\}$; by GHOST-ALLOC, SINCE $\mathsf{h}_0 \cdot \mathsf{h}_0$ IS VALID:

$\{\exists\gamma.\ \boxed{\mathsf{h}_0 \cdot \mathsf{h}_0}^{\gamma}\}$, we prove for all γ_1:

$\{\boxed{\mathsf{h}_0 \cdot \mathsf{h}_0}^{\gamma_1}\}$; by GHOST-ALLOC:

$\{\boxed{\mathsf{h}_0 \cdot \mathsf{h}_0}^{\gamma_1} * \exists\gamma.\ \boxed{\mathsf{h}_0 \cdot \mathsf{h}_0}^{\gamma}\}$; we prove for all γ_2:

$\{\boxed{\mathsf{h}_0 \cdot \mathsf{h}_0}^{\gamma_1} * \boxed{\mathsf{h}_0 \cdot \mathsf{h}_0}^{\gamma_2}\}$

```
let r = ref 0
```

$\{\boxed{\mathsf{h}_0}^{\gamma_1} * \boxed{\mathsf{h}_0}^{\gamma_1} * \boxed{\mathsf{h}_0}^{\gamma_2} * \boxed{\mathsf{h}_0}^{\gamma_2} * r \mapsto 0\}$

we choose $R \triangleq \exists ij.\ r \mapsto i + j * \boxed{\mathsf{h}_i}^{\gamma_1} * \boxed{\mathsf{h}_j}^{\gamma_2}$

$\{\boxed{\mathsf{h}_0}^{\gamma_1} * \boxed{\mathsf{h}_0}^{\gamma_2} * R\}$

```
let l = newlock ()
```

$\{\boxed{\mathsf{h}_0}^{\gamma_1} * \boxed{\mathsf{h}_0}^{\gamma_2} * l \rightsquigarrow \mathsf{Lock}\, R\}$

$\{\boxed{\mathsf{h}_0}^{\gamma_1}\}$	$\{\boxed{\mathsf{h}_0}^{\gamma_2}\}$
`acquire l; r := !r + 1; release l`	`acquire l; r := !r + 1; release l`
$\{\boxed{\mathsf{h}_1}^{\gamma_1}\}$	$\{\boxed{\mathsf{h}_1}^{\gamma_2}\}$

Proof of `incr r || incr r` with ghost allocation

$\{\ulcorner\urcorner\}$; by GHOST-ALLOC, SINCE $\mathsf{h}_0 \cdot \mathsf{h}_0$ IS VALID:

$\{\exists\gamma.\ \boxed{\mathsf{h}_0 \cdot \mathsf{h}_0}^{\gamma}\}$, we prove for all γ_1:

$\{\boxed{\mathsf{h}_0 \cdot \mathsf{h}_0}^{\gamma_1}\}$; by GHOST-ALLOC:

$\{\boxed{\mathsf{h}_0 \cdot \mathsf{h}_0}^{\gamma_1} * \exists\gamma.\ \boxed{\mathsf{h}_0 \cdot \mathsf{h}_0}^{\gamma}\}$; we prove for all γ_2:

$\{\boxed{\mathsf{h}_0 \cdot \mathsf{h}_0}^{\gamma_1} * \boxed{\mathsf{h}_0 \cdot \mathsf{h}_0}^{\gamma_2}\}$

```
let r = ref 0
```

$\{\boxed{\mathsf{h}_0}^{\gamma_1} * \boxed{\mathsf{h}_0}^{\gamma_1} * \boxed{\mathsf{h}_0}^{\gamma_2} * \boxed{\mathsf{h}_0}^{\gamma_2} * r \mapsto 0\}$

we choose $R \triangleq \exists ij.\ r \mapsto i + j * \boxed{\mathsf{h}_i}^{\gamma_1} * \boxed{\mathsf{h}_j}^{\gamma_2}$

$\{\boxed{\mathsf{h}_0}^{\gamma_1} * \boxed{\mathsf{h}_0}^{\gamma_2} * R\}$

```
let l = newlock ()
```

$\{\boxed{\mathsf{h}_0}^{\gamma_1} * \boxed{\mathsf{h}_0}^{\gamma_2} * l \rightsquigarrow \mathsf{Lock}\, R\}$

$\{\boxed{\mathsf{h}_0}^{\gamma_1}\}$	$\{\boxed{\mathsf{h}_0}^{\gamma_2}\}$
`acquire l; r := !r + 1; release l`	`acquire l; r := !r + 1; release l`
$\{\boxed{\mathsf{h}_1}^{\gamma_1}\}$	$\{\boxed{\mathsf{h}_1}^{\gamma_2}\}$

$\{\boxed{\mathsf{h}_1}^{\gamma_1} * \boxed{\mathsf{h}_1}^{\gamma_2}\}$

```
acquire l;
```

$\{\boxed{\mathsf{h}_1}^{\gamma_1} * \boxed{\mathsf{h}_1}^{\gamma_2} * R\}$

$\{\boxed{\mathsf{h}_1}^{\gamma_1} * \boxed{\mathsf{h}_1}^{\gamma_2} * r \mapsto 2\}$

```
assert (!r = 2)
```

Several types of resource algebra

The dependent product of finite partial functions to each $\mathcal{M}_i$ is an RA:

$$\prod_{i \in I} \mathbb{N} \xrightarrow{\text{fin}} \mathcal{M}_i$$

$$\boxed{a : \mathcal{M}_i}^{\gamma} \triangleq \boxed{\lambda j. \begin{cases} \{(\gamma, a)\} & \text{if } i = j \\ \varnothing & \text{otherwise} \end{cases}}$$

Several types of resource algebra

The dependent product of finite partial functions to each $\mathcal{M}_i$ is an RA:

$$\prod_{i \in I} \mathbb{N} \xrightarrow{\text{fin}} \mathcal{M}_i \qquad \boxed{a : \mathcal{M}_i}^{\gamma} \triangleq \boxed{\lambda j. \begin{cases} \{(\gamma, a)\} & \text{if } i = j \\ \varnothing & \text{otherwise} \end{cases}}$$

The Σ of "`iProp` Σ" does the bookkeeping between i's and $\mathcal{M}_i$'s, e.g.:

`Context` $`\{$`inG` $\Sigma\ \mathcal{M}\}.$	$\approx$	$\mathcal{M} \in \Sigma$
`Context` $`\{$`mylibG` $\Sigma\}.$	$\approx$	for all $\mathcal{M}$ used in `mylib`, $\mathcal{M} \in \Sigma$

Modalities

Persistent ressources

"Persistent" is a little stronger than "duplicable":

$$\frac{\mathsf{persistent}(P)}{P \rhd P * P} \qquad \frac{\mathsf{persistent}(P)}{P \mathbin{\mathbb{\wedge}} Q \rhd P * Q}$$

Persistent ressources

"Persistent" is a little stronger than "duplicable":

$$\frac{\mathsf{persistent}(P)}{P \rhd P * P} \qquad \frac{\mathsf{persistent}(P)}{P \mathbin{\mathbb{\&}} Q \rhd P * Q}$$

Some persistent resources:

$\ulcorner P \urcorner$ $\quad l \rightsquigarrow \mathsf{Lock}\ R$ $\quad \boxed{F}^{\gamma}$ $\quad \{P\}\ e\ \{Q\}$ $\quad p.\mathsf{length} \mapsto n$ (maybe)

Persistent ressources

"Persistent" is a little stronger than "duplicable":

$$\frac{\mathsf{persistent}(P)}{P \rhd P * P} \qquad \frac{\mathsf{persistent}(P)}{P \mathbin{\mathbb{W}} Q \rhd P * Q}$$

Some persistent resources:

$\ulcorner P \urcorner$ $\quad l \rightsquigarrow \mathsf{Lock}\ R$ $\quad \boxed{F}^{\gamma}$ $\quad \{P\}\ e\ \{Q\}$ $\quad p.\mathsf{length} \mapsto n$ (maybe)

Some **non-persistent** resources:

$l \mapsto v$ $\quad l \overset{1/4}{\mapsto} v$ $\quad \boxed{S}^{\gamma}$ $\quad \boxed{\mathsf{h}_0}^{\gamma}$ $\quad \mathsf{wp}\,(l := v)\,(\lambda\_.\ l \mapsto \_)$

Persistent ressources

"Persistent" is a little stronger than "duplicable":

$$\frac{\mathsf{persistent}(P)}{P \rhd P * P} \qquad \frac{\mathsf{persistent}(P)}{P \mathbin{\mathbb{W}} Q \rhd P * Q}$$

Some persistent resources:

$\ulcorner P \urcorner$ $\quad l \rightsquigarrow \mathsf{Lock}\ R$ $\quad \boxed{F}^{\gamma}$ $\quad \{P\}\ e\ \{Q\}$ $\quad p.\mathsf{length} \mapsto n$ (maybe)

Some **non-persistent** resources:

$l \mapsto v$ $\quad l \overset{1/4}{\mapsto} v$ $\quad \boxed{S}^{\gamma}$ $\quad \boxed{\mathsf{h}_0}^{\gamma}$ $\quad \mathsf{wp}\,(l := v)\,(\lambda\_.\ l \mapsto \_)$

Persistent resources:

- can be shared between threads:

$$\frac{\{P_1 * P\}\ e_1\ \{Q_1\} \qquad \{P_2 * P\}\ e_2\ \{Q_2\} \qquad \mathsf{persistent}(P)}{\{P_1 * P_2 * P\}\ e_1 || e_2\ \{Q_1 * Q_2 * P\}}$$

- can be saved for later using the frame rule, to they are true **forever** (whereas non-persistent resources are consumed)
- can be seen as *knowledge* (e.g. to indicate a task is done)

Persistently modality $\square P$

$\square P \approx (P * \ulcorner P \text{ is persistent} \urcorner)$ pronounced "persistently P" or "forever P"

Persistently modality $\Box P$

$\Box P \approx (P * \ulcorner P \text{ is persistent} \urcorner)$ pronounced "persistently P" or "forever P"

Helpful for concise rules, definitions, properties:

$$\{P\}\, e\, \{Q\} \;\approx\; \Box(P \mathrel{-\!\!*} \mathsf{wp}\, e\, Q) \qquad P \Rrightarrow Q \;\approx\; \Box(P \mathrel{-\!\!*} \Rrightarrow Q)$$

$\Box$-E
$$\frac{}{\Box P \rhd P}$$

$\Box$-IDEMP
$$\frac{}{\Box P \rhd \Box\Box P}$$

$\Box$-INTRO
$$\frac{\Box P \rhd Q}{\Box P \rhd \Box Q}$$

$\Box$-MONO
$$\frac{P \rhd Q}{\Box P \rhd \Box Q}$$

$\Box$-SEP
$$\frac{S \rhd \Box P \wedge Q}{S \rhd \Box P * Q}$$

$$\frac{\{P_1 * \Box P\}\, e_1\, \{Q_1\} \qquad \{P_2 * \Box P\}\, e_2\, \{Q_2\}}{\{P_1 * P_2 * \Box P\}\, e_1 || e_2\, \{Q_1 * Q_2 * P\}}$$

Later modality $\triangleright$

$\triangleright P$ ("later P") can be thought as "P holds after one reduction step".

careful: $\triangleright$ is a modality ($\triangleright P$), $\rhd$ is a binary operation ($P \rhd Q$)

Later modality $\triangleright$

$\triangleright P$ ("later P") can be thought as "P holds after one reduction step".

careful: $\triangleright$ is a modality ($\triangleright P$), $\rhd$ is a binary operation ($P \rhd Q$)

Step reductions "consume" laters:

$$\frac{\{P\}\ e'\ \{Q\} \qquad e, \sigma \to e', \sigma}{\{\triangleright P\}\ e\ \{Q\}}$$

Later modality $\triangleright$

$\triangleright P$ ("later P") can be thought as "P holds after one reduction step".

careful: $\triangleright$ is a modality ($\triangleright P$), $\rhd$ is a binary operation ($P \rhd Q$)

Step reductions "consume" laters:

$$\frac{\{P\}\ e'\ \{Q\} \qquad e, \sigma \to e', \sigma}{\{\triangleright P\}\ e\ \{Q\}}$$

$$P \vdash \triangleright P \qquad \frac{P \vdash Q}{\triangleright P \vdash \triangleright Q} \qquad \triangleright(P * Q) \dashv\vdash \triangleright P * \triangleright Q \qquad \text{etc}$$

Later modality $\triangleright$

$\triangleright P$ ("later P") can be thought as "P holds after one reduction step".

careful: $\triangleright$ is a modality ($\triangleright P$), $\rhd$ is a binary operation ($P \rhd Q$)

Step reductions "consume" laters:

$$\frac{\{P\}\ e'\ \{Q\} \qquad e, \sigma \to e', \sigma}{\{\triangleright P\}\ e\ \{Q\}}$$

$$P \vdash \triangleright P \qquad \frac{P \vdash Q}{\triangleright P \vdash \triangleright Q} \qquad \triangleright(P * Q) \dashv\vdash \triangleright P * \triangleright Q \qquad \text{etc}$$

$\triangleright$ and $\square$ are modalities in Iris, where "`iProp`" are not "`heap -> Prop`" and have features such as step indexing, and resources are not only heaps.

Later modality $\triangleright$

$\triangleright P$ ("later P") can be thought as "P holds after one reduction step".

careful: $\triangleright$ is a modality ($\triangleright P$), $\rhd$ is a binary operation ($P \rhd Q$)

Step reductions "consume" laters:

$$\frac{\{P\}\ e'\ \{Q\} \qquad e, \sigma \to e', \sigma}{\{\triangleright P\}\ e\ \{Q\}}$$

$$P \vdash \triangleright P \qquad \frac{P \vdash Q}{\triangleright P \vdash \triangleright Q} \qquad \triangleright(P * Q) \dashv\vdash \triangleright P * \triangleright Q \qquad \text{etc}$$

$\triangleright$ and $\square$ are modalities in Iris, where "`iProp`" are not "`heap -> Prop`" and have features such as step indexing, and resources are not only heaps.

$\triangleright^n \mathsf{False} \vdash P$ iff P holds for n steps

Later modality $\triangleright$

$\triangleright P$ ("later P") can be thought as "P holds after one reduction step".

careful: $\triangleright$ is a modality ($\triangleright P$), $\rhd$ is a binary operation ($P \rhd Q$)

Step reductions "consume" laters:

$$\frac{\{P\}\ e'\ \{Q\} \qquad e, \sigma \to e', \sigma}{\{\triangleright P\}\ e\ \{Q\}}$$

$$P \vdash \triangleright P \qquad \frac{P \vdash Q}{\triangleright P \vdash \triangleright Q} \qquad \triangleright(P * Q) \dashv\vdash \triangleright P * \triangleright Q \qquad \text{etc}$$

$\triangleright$ and $\square$ are modalities in Iris, where "`iProp`" are not "`heap -> Prop`" and have features such as step indexing, and resources are not only heaps.

$\triangleright^n \mathsf{False} \vdash P$ iff P holds for n steps $\qquad \vdash \exists k.\ \triangleright^k \mathsf{False}$

Löb rule

The **Löb rule** is very convenient for partial correctness:

$$\frac{Q \land \triangleright P \vdash P}{Q \vdash P}\ \textsc{Löb}$$

Invariants

Invariants

Iris invariants (not the ones for loops / data structures / locks):

- invariants are **allocated**:

$$\textsc{Inv-alloc}\quad R \Rrightarrow \boxed{R}^{\mathcal{N}} \qquad \text{where} \qquad \boxed{R}^{\mathcal{N}} \triangleq \exists \iota \in \mathcal{N}.\, \boxed{R}^{\iota}$$

where $\mathcal{N}$ is an infinite set of names called "namespace"

Invariants

Iris invariants (not the ones for loops / data structures / locks):

- invariants are **allocated**:

$$\textsc{Inv-alloc}\quad R \Rrightarrow \boxed{R}^{\mathcal{N}} \qquad \text{where} \qquad \boxed{R}^{\mathcal{N}} \triangleq \exists \iota \in \mathcal{N}.\, \boxed{R}^{\iota}$$

  where $\mathcal{N}$ is an infinite set of names called "namespace"

- they are **persistent** so can be shared: $\boxed{R}^{\mathcal{N}} = \boxed{R}^{\mathcal{N}} * \boxed{R}^{\mathcal{N}}$

Invariants

Iris invariants (not the ones for loops / data structures / locks):

- invariants are **allocated**:

$$\textsc{Inv-alloc}\quad R \Rrightarrow \boxed{R}^{\mathcal{N}} \qquad \text{where} \qquad \boxed{R}^{\mathcal{N}} \triangleq \exists \iota \in \mathcal{N}.\, \boxed{R}^{\iota}$$

  where $\mathcal{N}$ is an infinite set of names called "namespace"

- they are **persistent** so can be shared: $\boxed{R}^{\mathcal{N}} = \boxed{R}^{\mathcal{N}} * \boxed{R}^{\mathcal{N}}$
- before any step, any thread can **open** $\boxed{R}^{\mathcal{N}}$ to get R for a very short time: the thread must give R back before any other thread is run. They can be open across **atomic** steps.

Invariants

Iris invariants (not the ones for loops / data structures / locks):

- invariants are **allocated**:

$$\textsc{Inv-alloc}\quad R \Rrightarrow \boxed{R}^{\mathcal{N}} \qquad \text{where} \qquad \boxed{R}^{\mathcal{N}} \triangleq \exists \iota \in \mathcal{N}.\, \boxed{R}^{\iota}$$

  where $\mathcal{N}$ is an infinite set of names called "namespace"

- they are **persistent** so can be shared: $\boxed{R}^{\mathcal{N}} = \boxed{R}^{\mathcal{N}} * \boxed{R}^{\mathcal{N}}$
- before any step, any thread can **open** $\boxed{R}^{\mathcal{N}}$ to get R for a very short time: the thread must give R back before any other thread is run. They can be open across **atomic** steps.
- can be encoded using ghost state + step indexing + modifying the definition of Hoare triples

Invariant opening

Invariant are opened and closed across atomic steps:

$$\frac{\{P * \triangleright R\}\ e\ \{Q * \triangleright R\}_{\mathcal{E}\setminus\mathcal{N}} \qquad \mathsf{atomic}(e) \qquad \mathcal{N} \subseteq \mathcal{E}}{\{P * \boxed{R}^{\mathcal{N}}\}\ e\ \{Q * \boxed{R}^{\mathcal{N}}\}_{\mathcal{E}}}\ \textsc{Hoare-inv}$$

– $\triangleright R$ is for step indexing
– $\mathcal{E}$ says which invariants we can still open (prevents nested openings)

Deriving lock invariants

Compare-and-set

Somehow simplified operational semantics of CAS (Compare-and-set):

$$\textsc{cas-failure}\ \frac{\sigma(l) \neq x}{\langle \sigma, \texttt{CAS}\, l\, x\, y \rangle \to \langle \sigma, \texttt{false} \rangle}$$

$$\textsc{cas-success}\ \frac{\sigma(l) = x}{\langle \sigma, \texttt{CAS}\, l\, x\, y \rangle \to \langle \sigma[\ell \mapsto y], \texttt{true} \rangle}$$

Compare-and-set

Somehow simplified operational semantics of CAS (Compare-and-set):

$$\textsc{cas-failure}\ \frac{\sigma(l) \neq x}{\langle \sigma, \texttt{CAS}\, l\, x\, y \rangle \to \langle \sigma, \texttt{false} \rangle} \qquad \textsc{cas-success}\ \frac{\sigma(l) = x}{\langle \sigma, \texttt{CAS}\, l\, x\, y \rangle \to \langle \sigma[\ell \mapsto y], \texttt{true} \rangle}$$

Proof rules:

$$\textsc{cas-failure}\quad \{l \mapsto z * \ulcorner z \neq x \urcorner\}\ \texttt{CAS}\, l\, x\, y\ \{\lambda b. l \mapsto z * \ulcorner b = \texttt{false} \urcorner\}$$

$$\textsc{cas-success}\quad \{l \mapsto x\}\ \texttt{CAS}\, l\, x\, y\ \{\lambda b. l \mapsto y * \ulcorner b = \texttt{true} \urcorner\}$$

Other atomic operations:
Compare-and-swap, Fetch-and-add (FAA), Test-and-set, ...

Deriving lock rules

One possible lock implementation: the **spin-lock**, using CAS:

```
let newlock () = ref 0
let acquire l = while not (CAS l 0 1) do ()
let release l = l := 0
```

Deriving lock rules

One possible lock implementation: the **spin-lock**, using CAS:

```
let newlock () = ref 0
let acquire l = while not (CAS l 0 1) do ()
let release l = l := 0
```

We want to recover the specifications for locks:

$$\forall l R.\quad l \rightsquigarrow \mathsf{Lock} R \;\vdash\; \triangleright\; l \rightsquigarrow \mathsf{Lock} R * l \rightsquigarrow \mathsf{Lock} R$$

$$\forall R.\quad \{R\}\ \texttt{newlock}\ ()\ \{\lambda l.\ l \rightsquigarrow \mathsf{Lock}\, R\}$$

$$\forall l R.\quad \{l \rightsquigarrow \mathsf{Lock}\, R\}\ \texttt{acquire l}\ \{\lambda \_.\ R * l \rightsquigarrow \mathsf{Lock}\, R\}$$

$$\forall l R.\ \{R * l \rightsquigarrow \mathsf{Lock}\, R\}\ \texttt{release l}\ \{\lambda \_.\ l \rightsquigarrow \mathsf{Lock}\, R\}$$

Deriving lock rules

One possible lock implementation: the **spin-lock**, using CAS:

```
let newlock () = ref 0
let acquire l = while not (CAS l 0 1) do ()
let release l = l := 0
```

We want to recover the specifications for locks:

$$\forall l R.\quad l \rightsquigarrow \mathsf{Lock} R \;\vdash\; \triangleright\; l \rightsquigarrow \mathsf{Lock} R * l \rightsquigarrow \mathsf{Lock} R$$

$$\forall R.\quad \{R\}\ \texttt{newlock}\ ()\ \{\lambda l.\ l \rightsquigarrow \mathsf{Lock}\, R\}$$

$$\forall l R.\quad \{l \rightsquigarrow \mathsf{Lock}\, R\}\ \texttt{acquire l}\ \{\lambda \_.\ R * l \rightsquigarrow \mathsf{Lock}\, R\}$$

$$\forall l R.\ \{R * l \rightsquigarrow \mathsf{Lock}\, R\}\ \texttt{release l}\ \{\lambda \_.\ l \rightsquigarrow \mathsf{Lock}\, R\}$$

Exercise: Give a definition for $l \rightsquigarrow \mathsf{Lock}\, R$ and prove the rules above

Deriving lock rules

One possible lock implementation: the **spin-lock**, using CAS:

```
let newlock () = ref 0
let acquire l = while not (CAS l 0 1) do ()
let release l = l := 0
```

We want to recover the specifications for locks:

$$\forall l R.\quad l \rightsquigarrow \mathsf{Lock} R \;\vdash\; \rhd\; l \rightsquigarrow \mathsf{Lock} R * l \rightsquigarrow \mathsf{Lock} R$$

$$\forall R.\quad \{R\}\ \texttt{newlock}\ ()\ \{\lambda l.\ l \rightsquigarrow \mathsf{Lock}\, R\}$$

$$\forall l R.\quad \{l \rightsquigarrow \mathsf{Lock}\, R\}\ \texttt{acquire l}\ \{\lambda \_.\ R * l \rightsquigarrow \mathsf{Lock}\, R\}$$

$$\forall l R.\ \{R * l \rightsquigarrow \mathsf{Lock}\, R\}\ \texttt{release l}\ \{\lambda \_.\ l \rightsquigarrow \mathsf{Lock}\, R\}$$

Exercise: Give a definition for $l \rightsquigarrow \mathsf{Lock}\, R$ and prove the rules above

$$l \rightsquigarrow \mathsf{Lock}\, R \;\triangleq\; \boxed{l \mapsto 1 \;\veebar\; l \mapsto 0 * R}^{\mathcal{N}_{\mathsf{lock}}}$$

Authoritative resource algebra

Authoritative RA

The authoritative RA over an RA $\mathcal{M}$ is, wheree $a, b \in \mathcal{M}$,

$$\mathrm{Auth}(\mathcal{M}) ::= \bullet a \mid \circ b \mid \bullet\circ(a, b) \mid \lightning$$

Intuition:

- One unique *global authority* $\bullet a$, or authoritative view you need it to update

Authoritative RA

The authoritative RA over an RA $\mathcal{M}$ is, wheree $a, b \in \mathcal{M}$,

$$\mathrm{Auth}(\mathcal{M}) ::= \bullet a \mid \circ b \mid \bullet\circ(a, b) \mid \lightning$$

Intuition:

- One unique *global authority* $\bullet a$, or authoritative view
  you need it to update
- Several *fragments* $\circ b$, or fragmental views
  use them to record independent contributions

Authoritative RA

The authoritative RA over an RA $\mathcal{M}$ is, wheree $a, b \in \mathcal{M}$,

$$\mathrm{Auth}(\mathcal{M}) ::= \bullet a \mid \circ b \mid \bullet\circ(a, b) \mid \lightning$$

Intuition:

- One unique *global authority* $\bullet a$, or authoritative view
  you need it to update
- Several *fragments* $\circ b$, or fragmental views
  use them to record independent contributions

Main properties:

$$\frac{\mathcal{V}(a \cdot c)}{\bullet a \cdot \circ b \rightsquigarrow \bullet(a \cdot c) \cdot \circ(b \cdot c)}$$

$$\mathcal{V}(\bullet a \cdot \circ b) \Rightarrow b \preccurlyeq a$$

Authoritative RA

Operations:

$\cdot$	$\bullet a$	$\circ b$	$\bullet\circ(a,b)$	$\lightning$
$\bullet a'$	$\lightning$	$\bullet\circ(a',b)$	$\lightning$	$\lightning$
$\circ b'$	$\bullet\circ(a,b')$	$\circ(b\cdot b')$	$\bullet\circ(a,b\cdot b')$	$\lightning$
$\bullet\circ(a',b')$	$\lightning$	$\bullet\circ(a',b\cdot b')$	$\lightning$	$\lightning$
$\lightning$	$\lightning$	$\lightning$	$\lightning$	$\lightning$
$\mathcal{V}(-)$	$\mathcal{V}(a)$	$\mathcal{V}(b)$	$\mathcal{V}(a)\wedge b \preccurlyeq a$	False

Example usage of $\mathrm{Auth}(\mathcal{M})$

Using $\mathrm{Auth}((\mathbb{N}, +))$ we can prove that any k threads doing:

$$e_{incr} \triangleq \texttt{acquire l; incr r; release l}$$

will increment `r` *at least* k times.

Example usage of $\mathrm{Auth}(\mathcal{M})$

Using $\mathrm{Auth}((\mathbb{N}, +))$ we can prove that any k threads doing:

$$e_{incr} \triangleq \texttt{acquire l; incr r; release l}$$

will increment `r` *at least* k times.

Using lock invariant $R = \exists n.\, r \mapsto n * \boxed{\bullet n}^{\gamma}$, we can prove:

$$l \rightsquigarrow \mathsf{Lock}\, R \vdash \{\boxed{\circ 0}^{\gamma}\}\ e_{incr}\ \{\boxed{\circ 1}^{\gamma}\}$$
$$l \rightsquigarrow \mathsf{Lock}\, R \vdash \{\boxed{\circ 0}^{\gamma}\}\ (e_{incr} \mid\mid e_{incr} \mid\mid \ldots \mid\mid e_{incr})\ \{\boxed{\circ k}^{\gamma}\}$$
$$R \vdash \{\boxed{\circ k}^{\gamma}\}\ \texttt{!r}\ \{n.n \geqslant k\}$$

Example usage of $\mathrm{Auth}(\mathcal{M})$

Using $\mathrm{Auth}((\mathbb{N}, +))$ we can prove that any k threads doing:

$$e_{incr} \triangleq \texttt{acquire l; incr r; release l}$$

will increment `r` *at least* k times.

Using lock invariant $R = \exists n.\, r \mapsto n * \boxed{\bullet n}^{\gamma}$, we can prove:

$$l \rightsquigarrow \mathsf{Lock}\, R \vdash \{\boxed{\circ 0}^{\gamma}\}\ e_{incr}\ \{\boxed{\circ 1}^{\gamma}\}$$
$$l \rightsquigarrow \mathsf{Lock}\, R \vdash \{\boxed{\circ 0}^{\gamma}\}\ (e_{incr} \mid\mid e_{incr} \mid\mid \ldots \mid\mid e_{incr})\ \{\boxed{\circ k}^{\gamma}\}$$
$$R \vdash \{\boxed{\circ k}^{\gamma}\}\ \texttt{!r}\ \{n.n \geqslant k\}$$

Indeed with $\boxed{\bullet n}^{\gamma} * \boxed{\circ 4}^{\gamma}$ we can only prove $4 \preccurlyeq_{(\mathbb{N},+)} n$ i.e. $4 \leqslant n$

Intuitively $\circ k$ does not prevent "other" $\circ 1$'s from contributing to $\bullet n$

Checking counter monotonicity using $\mathrm{Auth}(\mathbb{N}_{max})$

```
let r = Atomic.make 0
let read () = Atomic.get r
let incr () =
  Atomic.fetch_and_add r 1

let check () =
  let x = read () in
  let y = read () in
  assert (y >= x)

let rec loop f () =
  f (); loop f ()

let () =
  let open Domain in
  let d1 = spawn (loop incr) in
  let d2 = spawn (loop check) in
  join d1; join d2
```

Checking counter monotonicity using $\mathrm{Auth}(\mathbb{N}_{max})$

Let $\mathbb{N}_{max} = (\mathbb{N}, max)$

```
let r = Atomic.make 0
let read () = Atomic.get r
let incr () =
  Atomic.fetch_and_add r 1

let check () =
  let x = read () in
  let y = read () in
  assert (y >= x)

let rec loop f () =
  f (); loop f ()

let () =
  let open Domain in
  let d1 = spawn (loop incr) in
  let d2 = spawn (loop check) in
  join d1; join d2
```

Checking counter monotonicity using $\mathrm{Auth}(\mathbb{N}_{max})$

```
let r = Atomic.make 0
let read () = Atomic.get r
let incr () =
  Atomic.fetch_and_add r 1

let check () =
  let x = read () in
  let y = read () in
  assert (y >= x)

let rec loop f () =
  f (); loop f ()

let () =
  let open Domain in
  let d1 = spawn (loop incr) in
  let d2 = spawn (loop check) in
  join d1; join d2
```

Let $\mathbb{N}_{max} = (\mathbb{N}, max)$

Exercise: (mon_counter.v) Invariant, specs, proof sketch

Checking counter monotonicity using $\mathrm{Auth}(\mathbb{N}_{max})$

```
let r = Atomic.make 0
let read () = Atomic.get r
let incr () =
  Atomic.fetch_and_add r 1

let check () =
  let x = read () in
  let y = read () in
  assert (y >= x)

let rec loop f () =
  f (); loop f ()

let () =
  let open Domain in
  let d1 = spawn (loop incr) in
  let d2 = spawn (loop check) in
  join d1; join d2
```

Let $\mathbb{N}_{max} = (\mathbb{N}, max)$

Exercise: (mon_counter.v) Invariant, specs, proof sketch

$$\boxed{\exists n\ r \mapsto n * \boxed{\bullet(n : \mathbb{N}_{max})}^{\gamma}}^{\iota}$$

Checking counter monotonicity using $\mathrm{Auth}(\mathbb{N}_{max})$

```
let r = Atomic.make 0
let read () = Atomic.get r
let incr () =
  Atomic.fetch_and_add r 1

let check () =
  let x = read () in
  let y = read () in
  assert (y >= x)

let rec loop f () =
  f (); loop f ()

let () =
  let open Domain in
  let d1 = spawn (loop incr) in
  let d2 = spawn (loop check) in
  join d1; join d2
```

Let $\mathbb{N}_{max} = (\mathbb{N}, max)$

Exercise: (mon_counter.v) Invariant, specs, proof sketch

$$\boxed{\exists n\ r \mapsto n * \boxed{\bullet(n : \mathbb{N}_{max})}^{\gamma}}^{\iota}$$

$$\{\boxed{\circ n}^{\gamma}\}\ \texttt{read}\ ()\ \{k.\ulcorner k \geqslant n\urcorner * \boxed{\circ k}^{\gamma}\}$$

Checking counter monotonicity using $\mathrm{Auth}(\mathbb{N}_{max})$

```
let r = Atomic.make 0
let read () = Atomic.get r
let incr () =
  Atomic.fetch_and_add r 1

let check () =
  let x = read () in
  let y = read () in
  assert (y >= x)

let rec loop f () =
  f (); loop f ()

let () =
  let open Domain in
  let d1 = spawn (loop incr) in
  let d2 = spawn (loop check) in
  join d1; join d2
```

Let $\mathbb{N}_{max} = (\mathbb{N}, max)$

Exercise: (mon_counter.v) Invariant, specs, proof sketch

$$\boxed{\exists n\ r \mapsto n * \boxed{\bullet(n : \mathbb{N}_{max})}^{\gamma}}^{\iota}$$

$$\{\boxed{\circ n}^{\gamma}\}\ \texttt{read}\ ()\ \{k.\ulcorner k \geqslant n\urcorner * \boxed{\circ k}^{\gamma}\}$$

$$\{\boxed{\circ n}^{\gamma}\}\ \texttt{incr}\ ()\ \{\boxed{\circ(n+1)}^{\gamma}\}$$

Checking counter monotonicity using $\mathrm{Auth}(\mathbb{N}_{max})$

```
let r = Atomic.make 0
let read () = Atomic.get r
let incr () =
  Atomic.fetch_and_add r 1

let check () =
  let x = read () in
  let y = read () in
  assert (y >= x)

let rec loop f () =
  f (); loop f ()

let () =
  let open Domain in
  let d1 = spawn (loop incr) in
  let d2 = spawn (loop check) in
  join d1; join d2
```

Let $\mathbb{N}_{max} = (\mathbb{N}, max)$

Exercise: (mon_counter.v) Invariant, specs, proof sketch

$$\boxed{\exists n\ r \mapsto n * \boxed{\bullet(n : \mathbb{N}_{max})}^{\gamma}}^{\iota}$$

$$\{\boxed{\circ n}^{\gamma}\}\ \texttt{read}\ ()\ \{k.\ulcorner k \geqslant n\urcorner * \boxed{\circ k}^{\gamma}\}$$

$$\{\boxed{\circ n}^{\gamma}\}\ \texttt{incr}\ ()\ \{\boxed{\circ(n+1)}^{\gamma}\}$$

Proof of `check`:

$$\{\boxed{\circ 0}^{\gamma}\}\quad \texttt{let x = read ()}$$
$$\{x \geqslant 0 * \boxed{\circ x}^{\gamma}\}\quad \texttt{let y = read ()}$$
$$\{y \geqslant x * \boxed{\circ y}^{\gamma}\}\quad \texttt{assert (y >= x)}$$

Fractional RA

Intuitive definition:

$$\mathrm{Frac} \triangleq (0,1] \cap \mathbb{Q} \mid \lightning \qquad \mathcal{V}(q) \triangleq q \neq \lightning \qquad q \cdot q' \triangleq \begin{cases} q + q' & \text{if } q + q' \leqslant 1 \\ \lightning & \text{otherwise} \end{cases}$$

Fractional RA

Intuitive definition:

$$\mathrm{Frac} \triangleq (0,1] \cap \mathbb{Q} \mid \lightning \qquad \mathcal{V}(q) \triangleq q \neq \lightning \qquad q \cdot q' \triangleq \begin{cases} q + q' & \text{if } q + q' \leqslant 1 \\ \lightning & \text{otherwise} \end{cases}$$

Easier-to-use definition:

$$\mathrm{Frac} \triangleq \mathbb{Q}^{+*} \qquad \mathcal{V}(q) \triangleq q \leqslant 1 \qquad q \cdot q' \triangleq q + q'$$

Fractional RA

Intuitive definition:

$$\mathrm{Frac} \triangleq (0,1] \cap \mathbb{Q} \mid \lightning \qquad \mathcal{V}(q) \triangleq q \neq \lightning \qquad q \cdot q' \triangleq \begin{cases} q + q' & \text{if } q + q' \leqslant 1 \\ \lightning & \text{otherwise} \end{cases}$$

Easier-to-use definition:

$$\mathrm{Frac} \triangleq \mathbb{Q}^{+*} \qquad \mathcal{V}(q) \triangleq q \leqslant 1 \qquad q \cdot q' \triangleq q + q'$$

You still have to be a bit careful, here is a wrong definition:

$$\mathrm{Frac} \triangleq \mathbb{Q} \qquad \mathcal{V}(q) \triangleq 0 < q \leqslant 1 \qquad q \cdot q' \triangleq q + q'$$

Authoritative fractional RA

Derived construction: $\mathrm{FracAuth}(M) \triangleq \mathrm{Auth}((\mathrm{Frac} \times \mathcal{M})^?)$, noting:

$$\bullet a \triangleq \bullet(1, a) \qquad \circ_q\, b \triangleq \circ(q, b)$$

Properties:

$$\circ_{q+q'}(b \cdot b') \equiv \circ_q b \cdot \circ_{q'} b' \qquad \frac{\mathcal{V}(a \cdot c)}{\bullet a \cdot \circ_q b \rightsquigarrow \bullet(a \cdot c) \cdot \circ_q(b \cdot c)}$$

$$\mathcal{V}(\bullet a \cdot \circ_q b) \;\Rightarrow\; b \preccurlyeq a \qquad \mathcal{V}(\bullet a \cdot \circ_1 b) \;\Rightarrow\; b = a \qquad \frac{\mathcal{V}(a')}{\bullet a \cdot \circ_1 b \rightsquigarrow \bullet a' \cdot \circ_1 a'}$$

Example usage of $\mathrm{FracAuth}(\mathcal{M})$

Using $\mathrm{FracAuth}((\mathbb{N}, +))$ we can finally prove modularly that k threads

$$e_{incr} \triangleq \texttt{acquire l; incr r; release l}$$

will increment `r` at *exactly* k times.

Example usage of $\mathrm{FracAuth}(\mathcal{M})$

Using $\mathrm{FracAuth}((\mathbb{N}, +))$ we can finally prove modularly that k threads

$$e_{incr} \triangleq \texttt{acquire l; incr r; release l}$$

will increment `r` at *exactly* k times.

Using lock the invariant $R = \exists n\ r \mapsto n * \boxed{\bullet n}^{\gamma}$, we can prove

Example usage of $\mathrm{FracAuth}(\mathcal{M})$

Using $\mathrm{FracAuth}((\mathbb{N}, +))$ we can finally prove modularly that k threads

$$e_{incr} \triangleq \texttt{acquire l; incr r; release l}$$

will increment `r` at *exactly* k times.

Using lock the invariant $R = \exists n\ r \mapsto n * \boxed{\bullet n}^{\gamma}$, we can prove

$$\begin{aligned}
\mathsf{True} &\Rrightarrow \exists \gamma\ \boxed{\bullet 0}^{\gamma} * \boxed{\circ_1 0}^{\gamma} \\
&\Rightarrow \exists \gamma\ \boxed{\bullet 0}^{\gamma} * \mathop{\Large *}_{i=1}^{k} \boxed{\circ_{1/k} 0}^{\gamma} \\
l \rightsquigarrow \mathsf{Lock}\, R \vdash \{\boxed{\circ_{1/k} 0}^{\gamma}\}\ & e_{incr}\ \{\boxed{\circ_{1/k} 1}^{\gamma}\} \\
l \rightsquigarrow \mathsf{Lock}\, R \vdash \{\boxed{\circ_1 0}^{\gamma}\}\ & (e_{incr} \mid\mid e_{incr} \mid\mid \ldots \mid\mid e_{incr})\ \{\boxed{\circ_1 k}^{\gamma}\} \\
R \vdash \{\boxed{\circ_1 k}^{\gamma}\}\ & \texttt{!r}\ \{n.\ulcorner n = k \urcorner\}
\end{aligned}$$

Other common uses of Auth

When Loc and Val are any set (not necessarily RAs), this is a useful RA:

$$\mathrm{Auth}(Loc \xrightarrow{\mathsf{fin}} \mathrm{Excl}(Val))$$

Other common uses of Auth

When Loc and Val are any set (not necessarily RAs), this is a useful RA:

$$\mathrm{Auth}(Loc \xrightarrow{\mathsf{fin}} \mathrm{Excl}(Val)) \qquad\qquad \ell \mapsto v \triangleq \boxed{\circ[\ell := v]}^{\gamma_{heap}}$$

Other common uses of Auth : heaps

When Loc and Val are any set (not necessarily RAs), this is a useful RA:

$$\mathrm{Auth}(Loc \xrightarrow{\mathsf{fin}} \mathrm{Excl}(Val)) \qquad\qquad \ell \mapsto v \triangleq \boxed{\circ[\ell := v]}^{\gamma_{heap}}$$

$\ell \mapsto v$ is derived; threads and invariants own the fragmental view; the wp ties the authoritative view $\boxed{\bullet\sigma}^{\gamma_{heap}}$ to the actual physical steps.

Other common uses of Auth : heaps

When Loc and Val are any set (not necessarily RAs), this is a useful RA:

$$\mathrm{Auth}(Loc \xrightarrow{\text{fin}} \mathrm{Excl}(Val)) \qquad \ell \mapsto v \triangleq \boxed{\circ[\ell := v]}^{\gamma_{heap}}$$

$\ell \mapsto v$ is derived; threads and invariants own the fragmental view; the wp ties the authoritative view $\boxed{\bullet\sigma}^{\gamma_{heap}}$ to the actual physical steps.

For fractional permissions, uses $\mathrm{View}(A, B)$ which generalizes $\mathrm{Auth}(A)$ to two algebras with a extra binary validity $holds : A \to B \to Prop$:

$$\mathrm{View}(Loc \to Val, Loc \to \mathrm{Frac} \times Val) \qquad \ell \mapsto_q v \triangleq \boxed{\circ[\ell := (q, v)]}^{\gamma_{heap}}$$

Other common uses of Auth : heaps

When Loc and Val are any set (not necessarily RAs), this is a useful RA:

$$\mathrm{Auth}(Loc \xrightarrow{\text{fin}} \mathrm{Excl}(Val)) \qquad \ell \mapsto v \triangleq \boxed{\circ[\ell := v]}^{\gamma_{heap}}$$

$\ell \mapsto v$ is derived; threads and invariants own the fragmental view; the wp ties the authoritative view $\boxed{\bullet\sigma}^{\gamma_{heap}}$ to the actual physical steps.

For fractional permissions, uses $\mathrm{View}(A, B)$ which generalizes $\mathrm{Auth}(A)$ to two algebras with a extra binary validity $holds : A \to B \to Prop$:

$$\mathrm{View}(Loc \to Val, Loc \to \mathrm{Frac} \times Val) \qquad \ell \mapsto_q v \triangleq \boxed{\circ[\ell := (q, v)]}^{\gamma_{heap}}$$

Singleton type class mechanism `gen_heapGS` not to write γ_{heap}

Weakest preconditions

Presentation with weakest preconditions

WP are generally preferred to triples, in practice:

- more primitive: triples are decomposed into $-\!\!*$ and wp (and $\Box$)

$$\{P\}\ e\ \{Q\} \triangleq \Box(P -\!\!* \mathsf{wp}\, e\, Q)$$

- preconditions become *spatial* hypotheses, more easily managed

Presentation with weakest preconditions

WP are generally preferred to triples, in practice:

- more primitive: triples are decomposed into $-\!\!*$ and wp (and $\Box$)

$$\{P\}\ e\ \{Q\} \ \triangleq\ \Box(P -\!\!* \mathsf{wp}\, e\, Q)$$

- preconditions become *spatial* hypotheses, more easily managed
- maybe a little less intuitive than triples, so we keep triples too.
  e.g. the frame rule becomes:

$$(P * \mathsf{wp}\, e\, Q) \ \rhd\ \mathsf{wp}\, e\, (P * Q)$$

Presentation with weakest preconditions

WP are generally preferred to triples, in practice:

- more primitive: triples are decomposed into $-\!\!*$ and wp (and $\Box$)

$$\{P\}\ e\ \{Q\} \ \triangleq\ \Box(P -\!\!* \mathsf{wp}\, e\, Q)$$

- preconditions become *spatial* hypotheses, more easily managed
- maybe a little less intuitive than triples, so we keep triples too.
  e.g. the frame rule becomes:

$$(P * \mathsf{wp}\, e\, Q) \ \rhd\ \mathsf{wp}\, e\, (P * Q)$$

the rule for ghost update becomes:

$$\frac{\Rrightarrow \mathsf{wp}\, e\, (\Rrightarrow Q)}{\mathsf{wp}\, e\, Q}$$

where $\Rrightarrow$ is a unary ghost update, again more primitive:
$(P \Rrightarrow Q) \triangleq \Box(P -\!\!* \Rrightarrow Q)$

Builtin consequence rule in postconditions

$(\mathsf{wp}\, e\, -)$ is a monotone predicate transformer, so $\mathsf{wp}\, e\, Q$ is equivalent to

$$\mathsf{wp}\, e\, Q \quad \Leftrightarrow \quad \big(\forall R.\ (\forall v.\ Qv \mathrel{-\!\!*} Rv) \mathrel{-\!\!*} \mathsf{wp}\, e\, R\big)$$

Builtin consequence rule in postconditions

$(\mathsf{wp}\, e\, -)$ is a monotone predicate transformer, so $\mathsf{wp}\, e\, Q$ is equivalent to

$$\mathsf{wp}\, e\, Q \quad \Leftrightarrow \quad \big(\forall R.\ (\forall v.\ Qv \mathrel{-\!\!*} Rv) \mathrel{-\!\!*} \mathsf{wp}\, e\, R\big)$$

In Iris, instead of:

$$\{P\}\ e\ \{Q\} \ \triangleq\ \Box(P \mathrel{-\!\!*} \mathsf{wp}\, e\, Q)$$

the following is equivalent

$$\{P\}\ e\ \{Q\} \ \triangleq\ \Box(\forall R\ P \mathrel{-\!\!*} (\forall v\ Qv \mathrel{-\!\!*} Rv) \mathrel{-\!\!*} \mathsf{wp}\, e\, R)$$

Builtin consequence rule in postconditions

$(\mathsf{wp}\, e\, -)$ is a monotone predicate transformer, so $\mathsf{wp}\, e\, Q$ is equivalent to

$$\mathsf{wp}\, e\, Q \quad \Leftrightarrow \quad (\forall R.\ (\forall v.\ Qv \mathrel{-\!\!*} Rv) \mathrel{-\!\!*} \mathsf{wp}\, e\, R)$$

In Iris, instead of:

$$\{P\}\ e\ \{Q\} \ \triangleq\ \Box(P \mathrel{-\!\!*} \mathsf{wp}\, e\, Q)$$

the following is equivalent

$$\{P\}\ e\ \{Q\} \ \triangleq\ \Box(\forall R\ P \mathrel{-\!\!*} (\forall v\ Qv \mathrel{-\!\!*} Rv) \mathrel{-\!\!*} \mathsf{wp}\, e\, R)$$

but is easier to apply since R is universally quantified.
This is the reason for
`iIntros (R) "Hprecondition Hpostcondition_cont".`

(Note that separating implication $\mathrel{-\!\!*}$ has no easy "heap entailment" $\rhd$ counterpart here, $\mathrel{-\!\!*}$ is more expressive.)

Simplified rules for load, store, alloc

Exercise: Complete the following implications:

$$\begin{array}{lll}
\forall \ell v Q & \ldots\ldots\ldots & -\!\!* \left(\ldots\ldots\ldots\ldots\ldots\ldots\right) \quad -\!\!* \, \mathsf{wp}\,(!\ell)\,Q \\
\forall \ell v v' Q & \ldots\ldots\ldots & -\!\!* \left(\ldots\ldots\ldots\ldots\ldots\ldots\right) \quad -\!\!* \, \mathsf{wp}\,(\ell := v)\,Q \\
\forall v Q & \ldots\ldots\ldots & -\!\!* \left(\ldots\ldots\ldots\ldots\ldots\ldots\right) \quad -\!\!* \, \mathsf{wp}\,(\texttt{ref}\, v)\,Q
\end{array}$$

Simplified rules for load, store, alloc

Exercise: Complete the following implications:

$$
\begin{array}{llll}
\forall \ell v Q & \ell \mapsto v & \mathrel{-\!\!*} (\ell \mapsto v \mathrel{-\!\!*} Q v) & \mathrel{-\!\!*} \mathsf{wp}\,(!\ell)\,Q \\
\forall \ell v v' Q & \ell \mapsto v' & \mathrel{-\!\!*} (\ell \mapsto v \mathrel{-\!\!*} Q()) & \mathrel{-\!\!*} \mathsf{wp}\,(\ell := v)\,Q \\
\forall v Q & \top & \mathrel{-\!\!*} (\forall \ell \;\; \ell \mapsto v \mathrel{-\!\!*} Q \ell) & \mathrel{-\!\!*} \mathsf{wp}\,(\texttt{ref}\; v)\,Q
\end{array}
$$

Variants/instances of Iris

Relaxed memory

- Invariants such as $\boxed{lock \mapsto 0 \lor lock \mapsto 1 * \exists n\ r \mapsto n}^{\iota}$ only make sense if there is an instantaneous view of the memory, which is not true in relaxed memory
- for now, axiomatic memory models do not fit Iris, but view-based operational memory models (for e.g. for release-acquire synchronisation) can be made to fit
- single-location invariants $\boxed{\ell \mid I}$ which can provide knowledge + special mechanisms (escrows) to transmit non-persistent resources

Linearizability

Under sequential consistency linearizability can be reasoned about using logically atomic triples:

$$\langle P \rangle\, e \,\langle Q \rangle$$

means: "at the linearization point in the execution of e, the resources in P are atomically consumed to produce the resources in Q"

Liveness?

- Transfinite Iris: ordinal step indices for the *existential property* and termination

	Iris	Transfinite Iris
if $\models \exists x\ P$ then for some x $\models P$	✗	✓
$\triangleright(\exists x\, P) \Leftrightarrow \exists x \triangleright P$	✓	✗
$\triangleright(P * Q) \Leftrightarrow \triangleright P * \triangleright Q$	✓	✗

- Nola: "no later" at invariant opening, replaced with restricted formulas

IRIS
$$\frac{\{P * \triangleright R\}\ e\ \{Q * \triangleright R\}}{\{P * \boxed{R}^{\iota}\}\ e\ \{Q\}}$$

NOLA
$$\frac{[P * [\![F]\!]]\ e\ [Q * [\![F]\!]] \qquad F \in Fml}{[P * \boxed{F}]\ e\ [Q]}$$

Conclusion

Separation Logic

- SL well-suited for data structures with pointers,
- abstract internal representation with higher-order logic
- can reason about other kinds of resources e.g. for time/space complexity, memory leaks
- extends well to concurrency

Iris in particular

- very expressive ghost state
- proof mode almost necessary in practice
- many extensions:
  - relaxed/weak memory models, effect handlers, distributed systems, cryptographic reasoning, probabilities, ...
- not only for safety: type soundness, e.g. rustbelt, relational separation logics, liveness, linearizability

More exercises

(1) design a resource algebra such that:

$$\mathcal{V}(r(0)) \qquad \forall n \in \mathbb{N}\ r(n) \equiv t(n) \cdot r(n+1) \qquad \neg\mathcal{V}(t(n) \cdot t(n))$$

motivation: allocate once $\Rrightarrow \exists \gamma\ \boxed{r(0)}^{\gamma}$ to generate an infinitely many tokens $\boxed{t(i)}^{\gamma}$, each will be used to transfer resources through single-location invariants at iteration i of a loop.

More exercises

(1) design a resource algebra such that:

$$\mathcal{V}(r(0)) \qquad \forall n \in \mathbb{N}\ r(n) \equiv t(n) \cdot r(n+1) \qquad \neg\mathcal{V}(t(n) \cdot t(n))$$

motivation: allocate once $\Rrightarrow \exists \gamma\ \boxed{r(0)}^{\gamma}$ to generate an infinitely many tokens $\boxed{t(i)}^{\gamma}$, each will be used to transfer resources through single-location invariants at iteration i of a loop.

(2) How to remove the last `acquire` in examples? How to recover resources back from an invariant? — in general how to make *cancellable invariants*?

More exercises

(1) design a resource algebra such that:

$$\mathcal{V}(r(0)) \qquad \forall n \in \mathbb{N}\ r(n) \equiv t(n) \cdot r(n+1) \qquad \neg\mathcal{V}(t(n) \cdot t(n))$$

motivation: allocate once $\Rightarrow\!\exists\gamma\ \boxed{r(0)}^{\gamma}$ to generate an infinitely many tokens $\boxed{t(i)}^{\gamma}$, each will be used to transfer resources through single-location invariants at iteration i of a loop.

(2) How to remove the last `acquire` in examples? How to recover resources back from an invariant? — in general how to make *cancellable invariants*?

(3) For using Iris, five exercises here:
https://gitlab.mpi-sws.org/iris/tutorial-popl21

Iris programming session?