Verification of Chase-Lev work-stealing deque

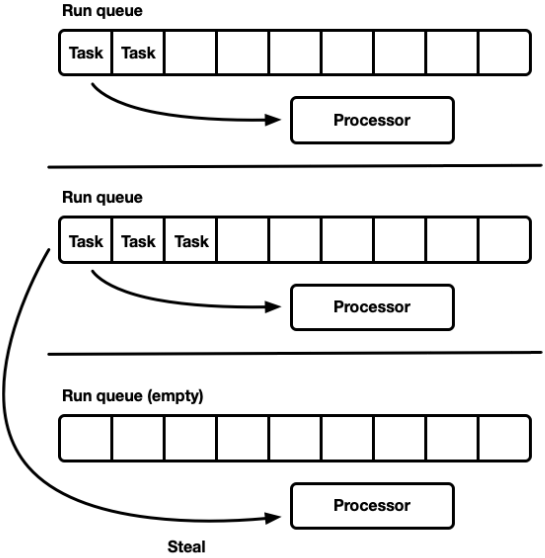
Clément Allain
François Pottier

May 5, 2023

Verification of a scheduler

```
let rec fib pool n =
  if n < 2 then 1 else
  let r1 = async pool (fun () -> fib_par (n - 1)) in
  let r2 = async pool (fun () -> fib_par (n - 2)) in
  await pool r1 + await pool r2
```

Work-stealing



Work-stealing algorithms

1. Frigo, Leiserson & Randall (1998)
    - at the core of Cilk 5
    - lock
2. Arora, Blumofe & Plaxton (2001)
    - no lock
    - one fixed size array (not circular), can overflow
3. Hendler, Lev & Shavit (2004)
    - no lock
    - list of small size arrays, no overflow
    - memory leak?
4. Chase & Lev (2005)
    - no lock
    - circular arrays, no overflow

Why is it interesting?

- demonstration of Iris on a (simplified) real-life concurrent data structure
- rich ghost state to enforce a subtle protocol
    - logical state ≠ physical state
    - external future-dependent linearization point
- use of prophecy variables (with memory)

The rest of this talk

- specification using logically atomic triples
- rough idea of how the data structure works
- why we need prophecy variables (with memory)

Specification — `chaselev_make`

$$\begin{array}{c} \{\,\mathrm{True}\,\} \\ \hline \texttt{chaselev\_make ()} \\ \hline \left\{\, \lambda\, t \cdot\ \text{chaselev-inv}\ t\ \iota\ \ast\ \text{chaselev-model}\ t\ []\ \ast\ \text{chaselev-owner}\ t \,\right\} \end{array}$$

Specification — `chaselev_make`

$$\begin{array}{c} \{\,\mathrm{True}\,\} \\ \hline \texttt{chaselev\_make ()} \\ \hline \left\{\,\lambda\, t \cdot\ \text{chaselev-inv}\ t\ \iota\ \ast\ \text{chaselev-model}\ t\ []\ \ast\ \text{chaselev-owner}\ t\,\right\} \end{array}$$

enforces a protocol (using an Iris invariant)

Specification — `chaselev_make`

$$\begin{array}{c} \{\,\text{True}\,\} \\ \hline \texttt{chaselev\_make ()} \\ \hline \left\{\, \lambda\, t \cdot\ \text{chaselev-inv}\ t\ \iota\ \ast\ \text{chaselev-model}\ t\ []\ \ast\ \text{chaselev-owner}\ t \,\right\} \end{array}$$

asserts the list of values that the deque (logically) contains

Specification — `chaselev_make`

$$\begin{array}{c} \{\, \text{True} \,\} \\ \hline \texttt{chaselev\_make ()} \\ \hline \left\{ \lambda\, t \cdot\ \text{chaselev-inv}\ t\ \iota\ \ast\ \text{chaselev-model}\ t\ []\ \ast\ \text{chaselev-owner}\ t \right\} \end{array}$$

gives the owner exclusive access to his end of the deque

Specification — `chaselev_push`

$$\begin{array}{c}
\left\{\ \text{chaselev-inv}\ t\ \iota\ \ *\ \ \text{chaselev-owner}\ t\ \right\} \\
\hline
\left\langle\ \forall\, vs \cdot\ \ \text{chaselev-model}\ t\ vs\ \right\rangle \\
\hline
\texttt{chaselev\_push}\ t\ v,\ \uparrow \iota \\
\hline
\left\langle\ \exists \cdot\ \text{chaselev-model}\ t\ (vs \mathbin{+\!\!+} [v])\ \right\rangle \\
\hline
\left\{\ \lambda\,\texttt{()} \cdot\ \ \text{chaselev-owner}\ t\ \right\}
\end{array}$$

Specification — chaselev_push

Specification of a concurrent operation (≃ transaction): standard triple + logically atomic triple

$$\begin{array}{c} \{\, P \,\} \\ \hline \langle\, \forall\, \overline{x} \cdot P_{\text{lin}} \,\rangle \\ \hline e,\ \mathcal{E} \\ \hline \langle\, \exists\, \overline{y} \cdot Q_{\text{lin}} \,\rangle \\ \hline \{\, \lambda\, res \cdot Q \,\} \end{array}$$

P : private precondition

Q : private postcondition

P_{lin} : public precondition

Q_{lin} : public postcondition

Specification — `chaselev_push`

For a concurrent data structure:

$$\begin{array}{c} \{\, \text{???-inv} \cdots * P \,\} \\ \hline \langle\, \forall\, \overline{x} \cdot \text{???-model} \cdots \,\rangle \\ \hline e,\ \mathcal{E} \\ \hline \langle\, \exists\, \overline{y} \cdot \text{???-model} \cdots \,\rangle \\ \hline \{\, \lambda\, res \cdot Q \,\} \end{array}$$

Specification — `chaselev_push`

$$\begin{array}{c}
\left\{\ \text{chaselev-inv}\ t\ \iota\ \ast\ \text{chaselev-owner}\ t\ \right\} \\ \hline
\left\langle\ \forall\, vs \cdot\ \ \text{chaselev-model}\ t\ vs\ \right\rangle \\ \hline
\texttt{chaselev\_push}\ t\ v,\ \uparrow\iota \\ \hline
\left\langle\ \exists \cdot\ \ \text{chaselev-model}\ t\ (vs \mathbin{+\!\!+} [v])\ \right\rangle \\ \hline
\left\{\ \lambda\,\texttt{()} \cdot\ \ \text{chaselev-owner}\ t\ \right\}
\end{array}$$

owner's operation

Specification — `chaselev_push`

$$\begin{array}{c}
\left\{\ \text{chaselev-inv}\ t\ \iota\ \ast\ \text{chaselev-owner}\ t\ \right\} \\ \hline
\left\langle\ \forall\, vs \cdot\ \text{chaselev-model}\ t\ vs\ \right\rangle \\ \hline
\texttt{chaselev\_push}\ t\ v,\ \uparrow\iota \\ \hline
\left\langle\ \exists \cdot\ \text{chaselev-model}\ t\ (vs \mathbin{+\!\!+} [v])\ \right\rangle \\ \hline
\left\{\ \lambda\,\texttt{()} \cdot\ \text{chaselev-owner}\ t\ \right\}
\end{array}$$

v is atomically pushed at the owner's end

Specification — `chaselev_pop`

$$\begin{array}{c}
\left\{ \text{chaselev-inv } t\ \iota \ \ast\ \text{chaselev-owner } t \right\} \\ \hline
\left\langle \forall\, vs \cdot \ \text{chaselev-model } t\ vs \right\rangle \\ \hline
\texttt{chaselev\_pop } t,\ \uparrow \iota \\ \hline
\left\langle \exists\, o \cdot \ \bigvee \left[\begin{array}{l} vs = [] \ast o = \texttt{NONE} \ast \text{chaselev-model } t\ [] \\ \exists\, v, vs' \cdot vs = vs' +\!\!+ [v] \ast o = \texttt{SOME } v \ast \text{chaselev-model } t\ vs' \end{array} \right] \right\rangle \\ \hline
\left\{ \lambda\, o \cdot \ \text{chaselev-owner } t \right\}
\end{array}$$

Specification — `chaselev_pop`

$$
\begin{array}{c}
\left\{\ \text{chaselev-inv}\ t\ \iota\ *\ \text{chaselev-owner}\ t\ \right\} \\
\hline
\left\langle\ \forall\, vs \cdot\ \text{chaselev-model}\ t\ vs\ \right\rangle \\
\hline
\texttt{chaselev\_pop}\ t,\ \uparrow \iota \\
\hline
\left\langle\ \exists\, o \cdot\ \bigvee \begin{bmatrix} vs = [] * o = \texttt{NONE} * \text{chaselev-model}\ t\ [] \\ \exists\, v, vs' \cdot vs = vs' \mathbin{+\!\!+} [v] * o = \texttt{SOME}\ v * \text{chaselev-model}\ t\ vs' \end{bmatrix}\ \right\rangle \\
\hline
\left\{\ \lambda\, o \cdot\ \text{chaselev-owner}\ t\ \right\}
\end{array}
$$

owner's operation

Specification — `chaselev_pop`

$$\begin{array}{c} \left\{ \text{chaselev-inv}\ t\ \iota \ast \text{chaselev-owner}\ t \right\} \\ \hline \left\langle \forall\, vs \cdot\ \text{chaselev-model}\ t\ vs \right\rangle \\ \hline \texttt{chaselev\_pop}\ t,\ \uparrow \iota \\ \hline \left\langle \exists\, o \cdot\ \bigvee \left[\begin{array}{l} vs = [] \ast o = \texttt{NONE} \ast \text{chaselev-model}\ t\ [] \\ \exists\, v, vs' \cdot vs = vs' +\!\!+ [v] \ast o = \texttt{SOME}\ v \ast \text{chaselev-model}\ t\ vs' \end{array} \right] \right\rangle \\ \hline \left\{ \lambda\, o \cdot\ \text{chaselev-owner}\ t \right\} \end{array}$$

either 1) the deque is seen empty
or 2) some value v is atomically popped at the owner's end

Specification — `chaselev_steal`

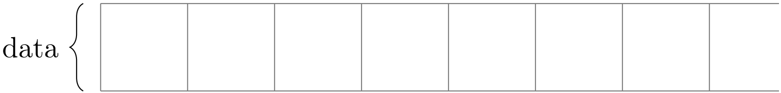
$$
\begin{array}{c}
\left\{\ \text{chaselev-inv}\ t\ \iota\ \right\} \\ \hline
\left\langle \forall\, vs \cdot\ \ \text{chaselev-model}\ t\ vs \right\rangle \\ \hline
\texttt{chaselev\_steal}\ t,\ \uparrow \iota \\ \hline
\left\langle \exists\, o \cdot\ \ \bigvee \left[\begin{array}{l} vs = [] * o = \texttt{NONE} * \text{chaselev-model}\ t\ [] \\ \exists\, v, vs' \cdot vs = v :: vs' * o = \texttt{SOME}\ v * \text{chaselev-model}\ t\ vs' \end{array} \right] \right\rangle \\ \hline
\{\, \lambda\, o \cdot\ \text{True} \,\}
\end{array}
$$

Specification — `chaselev_steal`

$$
\begin{array}{c}
\left\{\ \text{chaselev-inv}\ t\ \iota\ \right\} \\
\hline
\left\langle\ \forall\, vs \cdot\ \ \text{chaselev-model}\ t\ vs\ \right\rangle \\
\hline
\texttt{chaselev\_steal}\ t,\ \uparrow \iota \\
\hline
\left\langle\ \exists\, o \cdot\ \ \bigvee \left[\ \begin{array}{l} vs = [\,] * o = \texttt{NONE} * \text{chaselev-model}\ t\ [\,] \\ \exists\, v, vs' \cdot vs = v :: vs' * o = \texttt{SOME}\ v * \text{chaselev-model}\ t\ vs' \end{array}\ \right]\ \right\rangle \\
\hline
\{\, \lambda\, o \cdot \text{True}\,\}
\end{array}
$$

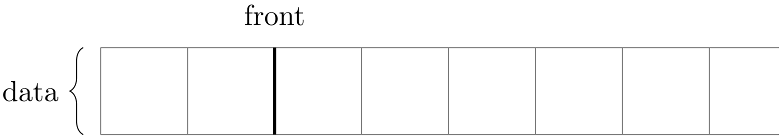
either 1) the deque is seen empty
or 2) some value v is atomically popped at the thieves' end

Physical state



data: infinite array storing all values

Physical state



data: infinite array storing all values

front: *monotone* index for thieves' end

Physical state

data {

data: in

front: m

CHASELEVFRONTLBGET

$$\frac{\boxed{\bullet\ \mathit{front}}^{\gamma.\mathrm{front}}}{\boxed{\circ\ \mathit{front}}^{\gamma.\mathrm{front}}}$$

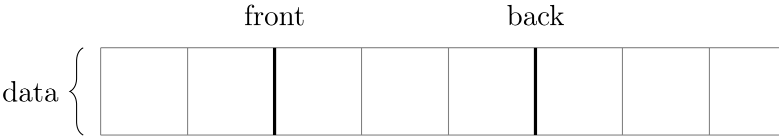
CHASELEVFRONTVALID

$$\frac{\boxed{\bullet\ \mathit{front}_1}^{\gamma.\mathrm{front}} \qquad \boxed{\circ\ \mathit{front}_2}^{\gamma.\mathrm{front}}}{\mathit{front}_2 \leqslant \mathit{front}_1}$$

CHASELEVFRONTUPDATE

$$\frac{\mathit{front} \leqslant \mathit{front}' \qquad \boxed{\bullet\ \mathit{front}}^{\gamma.\mathrm{front}}}{\boxed{\bullet\ \mathit{front}'}^{\gamma.\mathrm{front}}}$$

Physical state

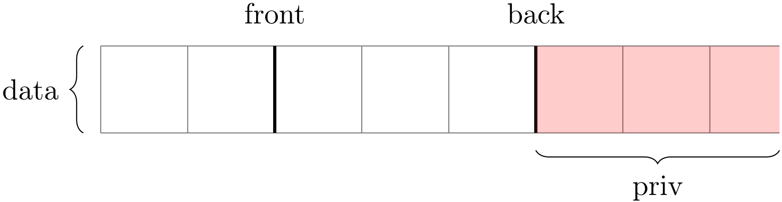


data: infinite array storing all values

front: *monotone* index for thieves’ end

back: index for owner’s end

Physical state



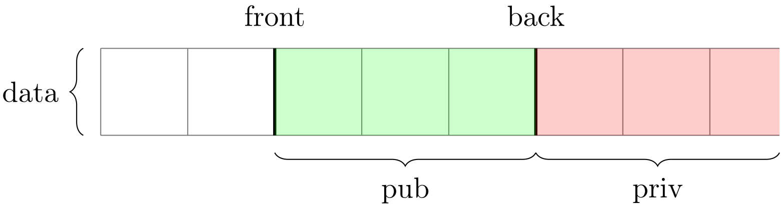
data: infinite array storing all values

front: *monotone* index for thieves' end

back: index for owner's end

priv: list of private values (controlled by owner)

Physical state



data: infinite array storing all values

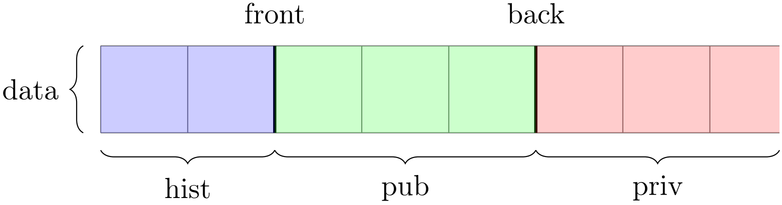
front: *monotone* index for thieves' end

back: index for owner's end

priv: list of private values (controlled by owner)

pub: list of public values (= model)

Physical state



data: infinite array storing all values

front: *monotone* index for thieves' end

back: index for owner's end

priv: list of private values (controlled by owner)

pub: list of public values (= model)

hist: *monotone* list of history values

Physical state

data

data: in

front: m

back: in

priv: lis

pub: lis

hist: m

ChaselevHistLbGet

$$\frac{\boxed{\bullet\ \mathit{hist}}^{\gamma.\mathrm{hist}}}{\boxed{\circ\ \mathit{hist}}^{\gamma.\mathrm{hist}}}$$

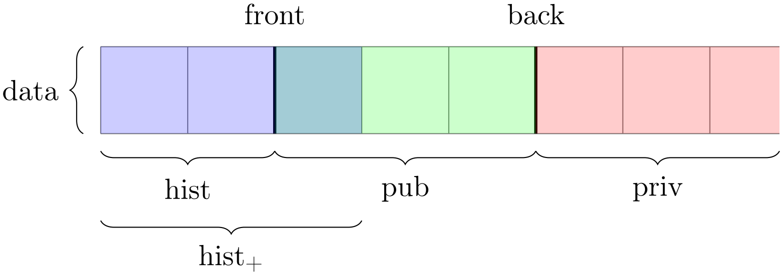
ChaselevHistValid

$$\frac{\boxed{\bullet\ \mathit{hist}_1}^{\gamma.\mathrm{hist}} \qquad \boxed{\circ\ \mathit{hist}_2}^{\gamma.\mathrm{hist}}}{\mathit{hist}_2 \sqsubseteq_{\mathrm{prefix}} \mathit{hist}_1}$$

ChaselevHistUpdate

$$\frac{\boxed{\bullet\ \mathit{hist}}^{\gamma.\mathrm{hist}}}{\boxed{\bullet\ (\mathit{hist} +\!\!+ [v])}^{\gamma.\mathrm{hist}}}$$

Physical state



data: infinite array storing all values

front: *monotone* index for thieves' end

back: index for owner's end

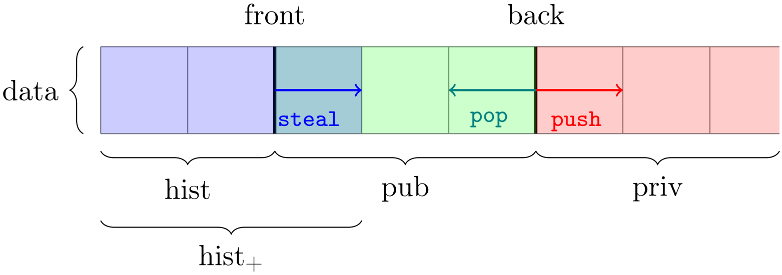
priv: list of private values (controlled by owner)

pub: list of public values (= model)

hist: *monotone* list of history values

hist$_{+}$: *monotone* list of extended history values

Physical state



data: infinite array storing all values

front: *monotone* index for thieves' end

back: index for owner's end

priv: list of private values (controlled by owner)

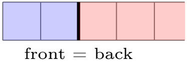
pub: list of public values (= model)

hist: *monotone* list of history values

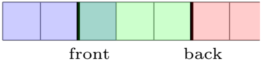
hist$_+$: *monotone* list of extended history values

Logical state

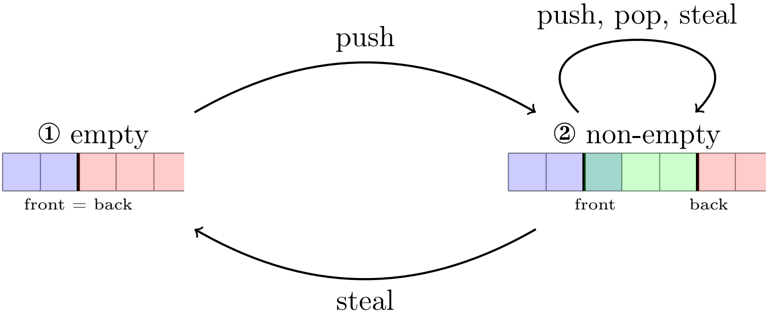
① empty



② non-empty

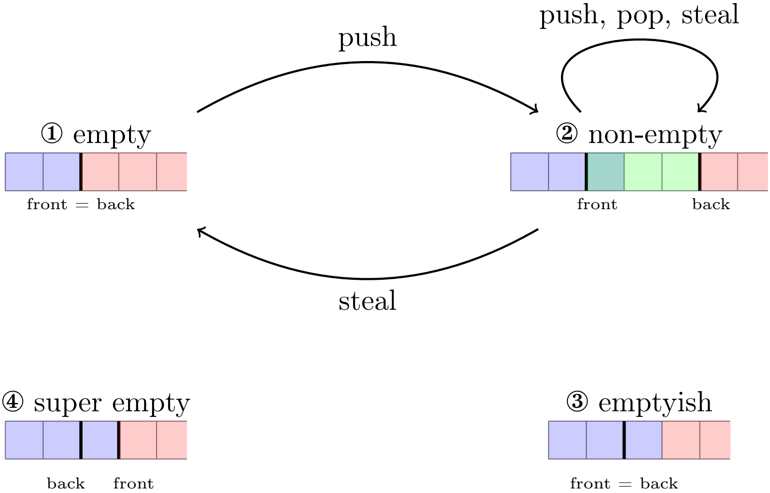


Logical state



⟶ linearization

Logical state



push

push, pop, steal

① empty

front = back

② non-empty

front

back

steal

④ super empty

back

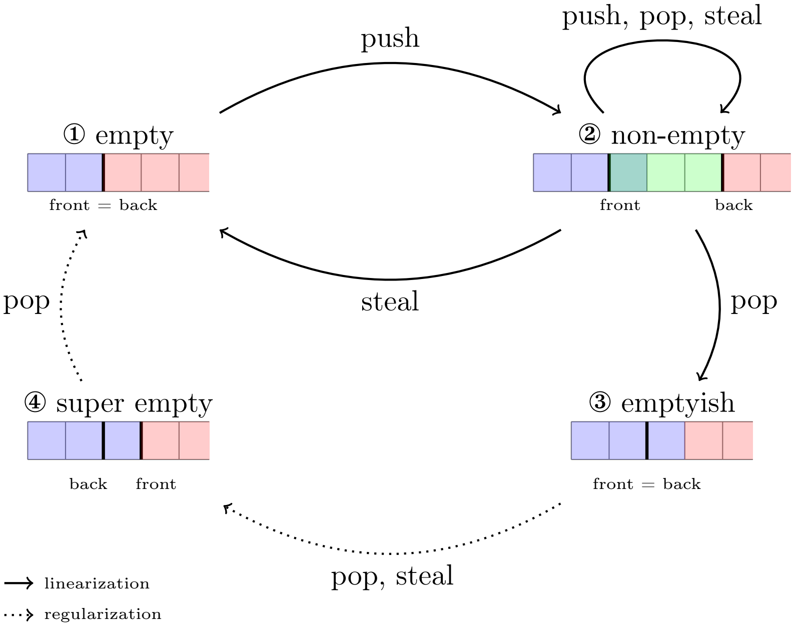
front

③ emptyish

front = back

⟶ linearization

Logical state



push

push, pop, steal

① empty

front = back

② non-empty

front

back

steal

pop

pop

④ super empty

back

front

③ emptyish

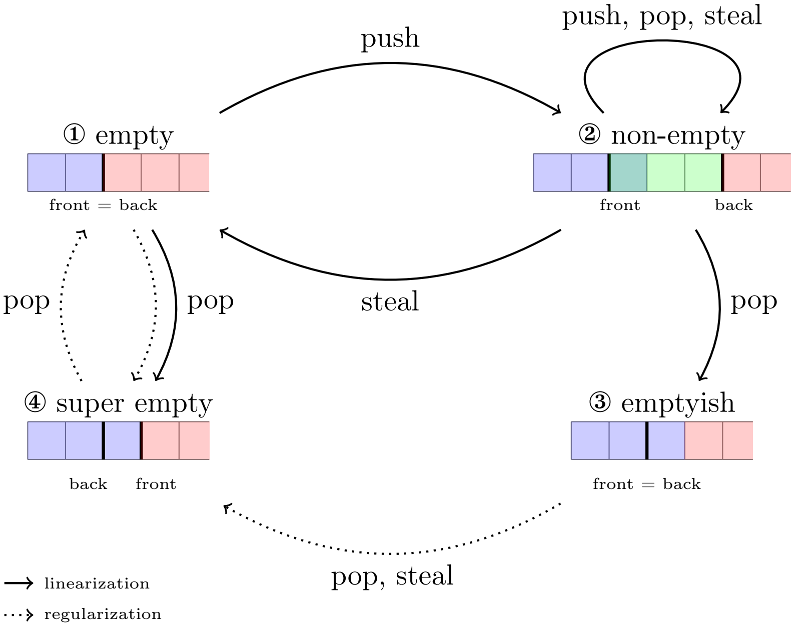
front = back

pop, steal

⟶ linearization

⋯› regularization

Logical state



Prophecy variable

$$\{\,\text{True}\,\}\ \texttt{NewProph}\ \{\,\lambda\, p \cdot \exists\, \mathit{prophs} \cdot \text{proph}\ p\ \mathit{prophs}\,\}$$

$$\frac{\begin{array}{c}\text{atomic}\ e \\ \text{proph}\ p\ \mathit{prophs} \\ \text{WP}\ e\ \left\{\begin{array}{l} \lambda w \cdot \forall\, \mathit{prophs}' \cdot \\ \mathit{prophs} = (w, v) :: \mathit{prophs}' \mathrel{-\!\!*} \\ \text{proph}\ p\ \mathit{prophs}' \mathrel{-\!\!*} \\ \Phi\ w \end{array}\right\}\end{array}}{\text{WP}\ \texttt{Resolve}\ e\ p\ v\ \{\,\Phi\,\}}$$

Prophecy variable in RDCSS

```
let rdcss rm rn m1 n1 n2 =
  let p = NewProph in
  let descr = ref (rm, m1, n1, n2, p) in
  ...
```

```
let complete descr rn =
  let (rm, m1, n1, n2, p) = !descr in
  let id = NewId in
  let m = !rm in
  let n_new = if m = m1 then n2 else n1 in
  Resolve (CmpXchg rn (inr descr) (inl n_new)) p id ;
  ()
```

Prophecy variable with memory

$$\{\,\mathrm{True}\,\}\ \texttt{NewProph}\ \{\,\lambda\, p \cdot \exists\, \gamma, \mathit{prophs} \cdot \mathrm{proph}\ p\ \gamma\ []\ \mathit{prophs}\,\}$$

$$\frac{\begin{array}{c}\text{atomic } e \\ \mathrm{proph}\ p\ \gamma\ \mathit{past}\ \mathit{prophs} \\ \mathrm{WP}\ e\ \left\{\begin{array}{l} \lambda w \cdot \forall\, \mathit{prophs}' \cdot \\ \mathit{prophs} = (w, v) :: \mathit{prophs}' \mathrel{-\!\!*} \\ \mathrm{proph}\ p\ \gamma\ (\mathit{past} +\!\!+ [(w, v)])\ \mathit{prophs}' \mathrel{-\!\!*} \\ \Phi\ w \end{array}\right\}\end{array}}{\mathrm{WP}\ \texttt{Resolve}\ e\ p\ v\ \{\,\Phi\,\}}$$

Prophecy variable with memory

$$\textsc{ProphecyLbGet}\quad \frac{\text{proph } p\ \gamma\ \mathit{past}\ \mathit{prophs}}{\text{proph-lb } \gamma\ \mathit{prophs}}$$

$$\textsc{ProphecyValid}\quad \frac{\text{proph } p\ \gamma\ \mathit{past}\ \mathit{prophs}_1 \qquad \text{proph-lb } \gamma\ \mathit{prophs}_2}{\exists\, \mathit{past}_1, \mathit{past}_2 \cdot \bigwedge \left[\begin{array}{l} \mathit{past} = \mathit{past}_1 \mathbin{+\!\!+} \mathit{past}_2 \\ \mathit{past}_2 \mathbin{+\!\!+} \mathit{prophs}_1 = \mathit{prophs}_2 \end{array}\right.}$$

Thank you for your attention!

Implementation — chaselev_make

```
let chaselev_make _ =
  let t = AllocN 4 () in
  t.front <- 0 ;
  t.back <- 0 ;
  t.data <- inf_array_make () ;
  t.prophecy <- NewProph ;
  t
```

Implementation — chaselev_push

```
let chaselev_push t v =
  let back = !t.back in
  inf_array_set !t.data back v ;
  t.back <- back + 1
```

Implementation — chaselev_steal

```
let rec chaselev_steal t =
  let id = NewId in
  let front = !t.front in
  let back = !t.back in
  if front < back then (
    if Snd (
      Resolve (
        CmpXchg t.front front (front + 1)
       ) !t.prophecy (front, id)
    ) then (
      SOME (inf_array_get !t.data front)
    ) else (
      chaselev_steal t
    )
  ) else (
    NONE
  )
```

Implementation — chaselev_pop

```
let chaselev_pop t =
  let id = NewId in
  let back = !t.back - 1 in
  t.back <- back ;
  let front = !t.front in
  if back < front then (
    t.back <- front
  ) else (
    if front < back then (
      SOME (inf_array_get !t.data back)
    ) else (
      if Snd (
        Resolve (
          CmpXchg t.front front (front + 1)
        ) !t.prophecy (front, id)
      ) then (
        t.back <- front + 1 ;
        SOME (inf_array_get !t.data back)
      ) else (
        t.back <- front + 1 ;
        NONE
      )
```

Infinite array

$$\begin{array}{c} \{\,\text{True}\,\} \\ \hline \texttt{inf\_array\_make}\ v \\ \hline \{\,\lambda\ arr \cdot \text{inf-array-model}\ arr\ (\lambda\_ \cdot v)\,\} \end{array}$$

$$\begin{array}{c} \langle\,\forall\, vs \cdot \text{inf-array-model}\ arr\ vs * 0 \leqslant i\,\rangle \\ \hline \texttt{inf\_array\_get}\ arr\ i \\ \hline \langle\,\exists \cdot \lambda\,(vsi) \cdot \text{inf-array-model}\ arr\ vs\,\rangle \end{array}$$

$$\begin{array}{c} \langle\,\forall\, vs \cdot \text{inf-array-model}\ arr\ vs * 0 \leqslant i\,\rangle \\ \hline \texttt{inf\_array\_set}\ arr\ i\ v \\ \hline \langle\,\exists \cdot \lambda\ \_ \cdot \text{inf-array-model}\ arr\ vs[i \mapsto v]\,\rangle \end{array}$$

Invariant

$$
\begin{array}{l}
\text{chaselev-inv } t\ \iota \triangleq \\
\quad \exists\, \ell, \gamma, \mathit{data}, p \cdot \\
\quad \mathop{\huge *} \left[\begin{array}{l}
t = \ell * \text{meta } \ell\ \gamma \\
\ell.\text{data} \mapsto_{\square} \mathit{data} * \ell.\text{prophecy} \mapsto_{\square} p \\
\boxed{\text{chaselev-inv-inner } \ell\ \gamma\ \iota\ \mathit{data}\ p}^{\iota}
\end{array} \right.
\end{array}
$$

Invariant

$$
\begin{array}{l}
\text{chaselev-inv-inner}\ \ell\ \gamma\ \iota\ \mathit{data}\ p \triangleq \\
\quad \exists\, \mathit{front}, \mathit{back}, \mathit{hist}, \mathit{pub}, \mathit{priv}, \mathit{past}, \mathit{prophs} \cdot \\
\quad \ast \left[\begin{array}{l}
\ell.\text{front} \mapsto \mathit{front} \ast \ell.\text{back} \mapsto \mathit{back} \\
\boxed{\bullet\,(\mathit{back}, \mathit{priv})}^{\gamma.\text{ctl}} \\
\boxed{\bullet\,\mathit{front}}^{\gamma.\text{front}} \\
\text{inf-array-model}\ \mathit{data}\ (\mathit{hist} +\!\!+\ \mathit{pub})\ \mathit{priv} \\
\boxed{\bullet\,\mathit{pub}}^{\gamma.\text{pub}} \quad \ast\ |\mathit{pub}| = (\mathit{back} - \mathit{front})_{+} \\
\text{wise-prophet-model}\ p\ \gamma.\text{prophet}\ \mathit{past}\ \mathit{prophs} \\
\forall (\mathit{front}', \_) \in \mathit{past} \cdot \mathit{front}' < \mathit{front} \\
\text{chaselev-state}\ \gamma\ \iota\ \mathit{front}\ \mathit{back}\ \mathit{hist}\ \mathit{pub}\ \mathit{prophs}
\end{array}\right.
\end{array}
$$

State

$$
\begin{array}{l}
\text{chaselev-state}\ \gamma\ \iota\ \mathit{front}\ \mathit{back}\ \mathit{hist}\ \mathit{pub}\ \mathit{prophs} \triangleq \\
\bigvee \left[\begin{array}{l}
\text{chaselev-state}_1\ \gamma\ \mathit{front}\ \mathit{back}\ \mathit{hist} \\
\text{chaselev-state}_2\ \gamma\ \iota\ \mathit{front}\ \mathit{back}\ \mathit{hist}\ \mathit{pub}\ \mathit{prophs} \\
\text{chaselev-lock}\ \gamma * \bigvee \left[\begin{array}{l}
\text{chaselev-state}_3\ \gamma\ \mathit{front}\ \mathit{back}\ \mathit{hist}\ \mathit{prophs} \\
\text{chaselev-state}_4\ \gamma\ \mathit{front}\ \mathit{back}\ \mathit{hist}
\end{array} \right.
\end{array} \right.
\end{array}
$$

State 1 (empty)

$$\text{chaselev-state}_1\ \gamma\ \mathit{front}\ \mathit{back}\ \mathit{hist} \triangleq$$

$$\ast \left[\begin{array}{l} \mathit{front} = \mathit{back} \\ \boxed{\bullet\, \mathit{hist}}^{\gamma.\text{hist}} \quad \ast\ |\mathit{hist}| = \mathit{front} \\ \boxed{\bullet\ -\cdot\circ\ -}^{\gamma.\text{winner}} \end{array}\right.$$

State 2 (non-empty)

$$\text{chaselev-state}_2\ \gamma\ \iota\ \mathit{front}\ \mathit{back}\ \mathit{hist}\ \mathit{pub}\ \mathit{prophs} \triangleq$$

$$\mathop{\ast}\left[\begin{array}{l}
\mathit{front} < \mathit{back} \\
\boxed{\bullet\,(\mathit{hist} +\!\!+ [\mathit{pub}[0]])}^{\gamma.\text{hist}} \ast |\mathit{hist}| = \mathit{front} \\
\quad \textbf{match}\ \text{filter}\ (\lambda(\mathit{front}', \_) \cdot \mathit{front}' = \mathit{front})\ \mathit{prophs}\ \textbf{with} \\
\quad |\ [] \Rightarrow \boxed{\bullet\ - \cdot \circ\ -}^{\gamma.\text{winner}} \\
\quad |\ (\_, \mathit{id}) :: \_ \Rightarrow \\
\qquad \bigvee \left[\begin{array}{l}
\boxed{\bullet\ - \cdot \circ\ -}^{\gamma.\text{winner}} \\
\text{identifier}\ \mathit{id} \ast \exists\, \Phi \cdot \boxed{\bullet\,(\mathit{front}, \Phi)}^{\gamma.\text{winner}} \ast \text{chaselev-au}\ \gamma\ \iota\ \Phi
\end{array}\right.
\end{array}\right.$$

State 3 (emptyish)

$$\text{chaselev-state}_3\ \gamma\ \mathit{front}\ \mathit{back}\ \mathit{hist}\ \mathit{prophs} \triangleq$$

$$\mathop{\ast}\left[\begin{array}{l} \mathit{front} = \mathit{back} \\ \boxed{\bullet\, \mathit{hist}}^{\gamma.\mathrm{hist}} \ast |\mathit{hist}| = \mathit{front} + 1 \\ \quad \textbf{match}\ \text{filter}\ (\lambda(\mathit{front}', \_) \cdot \mathit{front}' = \mathit{front})\ \mathit{prophs}\ \textbf{with} \\ \quad |\ [\,] \Rightarrow \boxed{\circ\,(\mathit{front}, -)}^{\gamma.\mathrm{winner}} \\ \quad |\ \_ \Rightarrow \exists\,\Phi \cdot \boxed{\bullet\,(\mathit{front},\Phi)}^{\gamma.\mathrm{winner}} \ast \Phi(\texttt{SOME}\ \mathit{hist}[\mathit{front}]) \end{array}\right.$$

State 4 (super empty)

$$\text{chaselev-state}_4\ \gamma\ \mathit{front}\ \mathit{back}\ \mathit{hist} \triangleq$$

$$\mathop{\ast}\left[\begin{array}{l} \mathit{front} = \mathit{back} + 1 \\ \boxed{\bullet\, \mathit{hist}}^{\gamma.\text{hist}} \quad \ast\ |\mathit{hist}| = \mathit{front} \\ \boxed{\bullet\ - \cdot \circ\ -}^{\gamma.\text{winner}} \end{array}\right.$$