

Verification of Chase-Lev work-stealing deque (work in progress)

Clément Allain
François Pottier

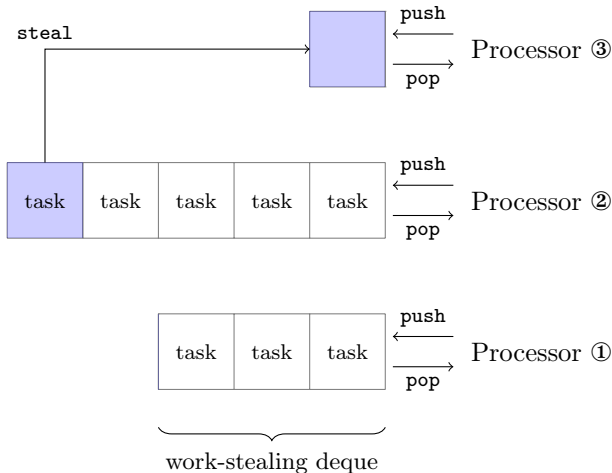
Inria Paris

May 22, 2023

Context: scheduler for task-based parallelism

- ▶ Cilk (C, C++)
- ▶ Threading Building Blocks (C++)
- ▶ Taskflow (C++)
- ▶ Tokio (RUST)
- ▶ Goroutines (Go)
- ▶ Domainslib (OCAML 5)

Work-stealing



Chase-Lev work-stealing deque

1. *The Implementation of the Cilk-5 Multithreaded Language.*
Frigo, Leiserson & Randall (1998).
 - ▶ lock
2. *Thread Scheduling for Multiprogrammed Multiprocessors.*
Arora, Blumofe & Plaxton (1998).
 - ▶ non-blocking
 - ▶ one fixed size array, potential overflow
3. *A dynamic-sized nonblocking work stealing deque.*
Hendler, Lev, Moir, & Shavit (2004).
 - ▶ non-blocking
 - ▶ list of small arrays, no overflow
4. *Dynamic circular work-stealing deque.*
Chase & Lev (2005).
 - ▶ non-blocking
 - ▶ circular arrays, no overflow

The rest of this talk

- ▶ Specification in IRIS (logically atomic triples).
- ▶ Sketch of the proof for a *simplified* version with one infinite array (as opposed to multiple circular arrays).
 - ▶ physical state
 - ▶ logical state
 - ▶ external future-dependent linearization point
- ▶ Why we need prophecy variables.

Specification

Physical state

Logical state

Prophecy variables

Specification — chaselev_make

$$\frac{\{ \text{True} \}}{\text{chaselev_make } ()}$$
$$\left\{ \lambda t. \text{ chaselev-inv } t \iota * \text{ chaselev-model } t \square * \text{ chaselev-owner } t \right\}$$

Specification — chaselev_make

$$\frac{\{ \text{True} \}}{\text{chaselev_make } ()}$$
$$\left\{ \lambda t. \text{chaselev-inv } t \iota * \text{chaselev-model } t [] * \text{chaselev-owner } t \right\}$$

t is an instance of Chase-Lev deque.
Enforces a protocol (using an Iris invariant).

Specification — chaselev_make

$$\frac{\{ \text{True} \}}{\text{chaselev_make } ()}$$
$$\left\{ \lambda t. \text{ chaselev-inv } t \iota * \text{ chaselev-model } t [] * \text{ chaselev-owner } t \right\}$$

Asserts the list of values that t logically contains.

Specification — chaselev_make

$$\frac{\{ \text{True} \}}{\text{chaselev_make } ()}$$
$$\left\{ \lambda t. \text{ chaselev-inv } t \iota * \text{ chaselev-model } t [] * \text{ chaselev-owner } t \right\}$$

Gives the owner exclusive access to his end of t .

Specification — chaselev_push

$$\frac{\left\{ \text{chaselev-inv } t \iota * \text{chaselev-owner } t \right\}}{\frac{\left\langle \forall vs. \text{ chaselev-model } t \text{ } vs \right\rangle}{\text{chaselev_push } t \text{ } v, \uparrow \iota} \left\langle \exists. \text{ chaselev-model } t \text{ } (vs \uplus [v]) \right\rangle} \left\{ (). \text{ chaselev-owner } t \right\}$$

Specification — chaselev_push

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

$$\frac{\frac{\frac{\{P\}}{\langle \forall \overline{x}. P_{\text{lin}} \rangle}}{e, \mathcal{E}}}{\langle \exists \overline{y}. Q_{\text{lin}} \rangle} \frac{}{\{res. Q\}}$$

P : private precondition

Q : private postcondition

P_{lin} : public precondition

Q_{lin} : public postcondition

Specification — chaselev_push

For a concurrent data structure:

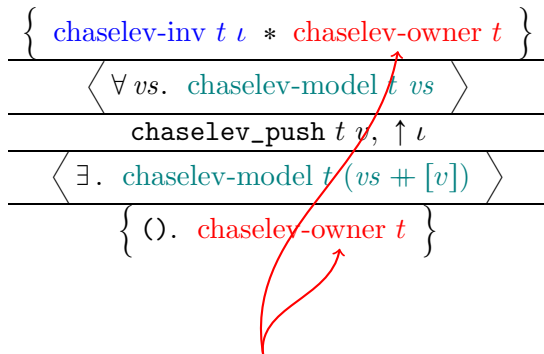
$$\frac{\frac{\frac{\{ ???\text{-inv} \dots * P \}}{\langle \forall \overline{x}. ???\text{-model} \dots \rangle}}{e, \mathcal{E}}}{\frac{\langle \exists \overline{y}. ???\text{-model} \dots \rangle}{\{ res. Q \}}}$$

Specification — chaselev_push

$$\frac{\left\{ \text{chaselev-inv } t \iota * \text{chaselev-owner } t \right\}}{\frac{\left\langle \forall vs. \text{ chaselev-model } t \text{ } vs \right\rangle}{\frac{\text{chaselev_push } t \text{ } v, \uparrow \iota}{\left\langle \exists. \text{ chaselev-model } t (vs \uplus [v]) \right\rangle}}}$$

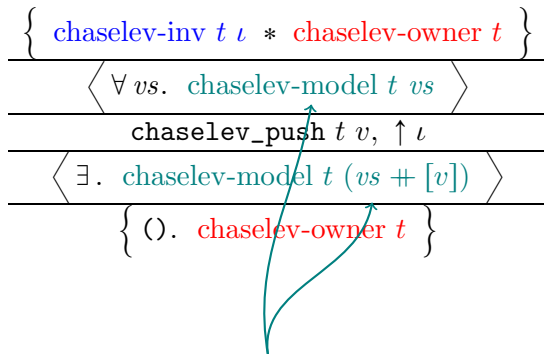
t is an instance of Chase-Lev deque.

Specification — chaselev_push



This operation is reserved to the owner of t .

Specification — chaselev_push

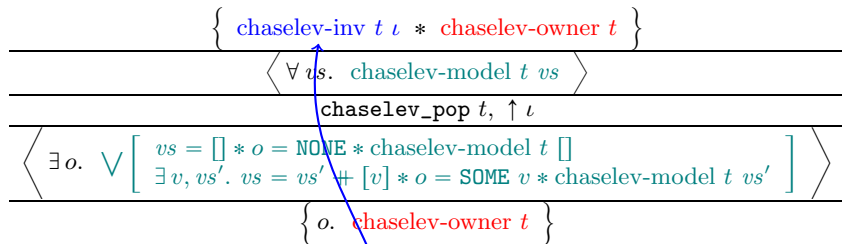


v is atomically pushed at the owner's end of t .

Specification — chaselev_pop

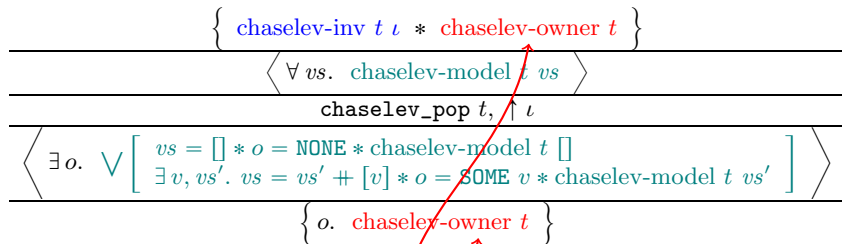
$$\frac{\left\{ \text{chaselev-inv } t \ \iota * \text{chaselev-owner } t \right\}}{\frac{\left\langle \forall \text{vs. } \text{chaselev-model } t \ \text{vs} \right\rangle}{\text{chaselev_pop } t, \uparrow \iota} \left\langle \exists o. \bigvee \left[\begin{array}{l} \text{vs} = [] * o = \text{NONE} * \text{chaselev-model } t \ [] \\ \exists v, \text{vs}'. \text{vs} = \text{vs}' \uplus [v] * o = \text{SOME } v * \text{chaselev-model } t \ \text{vs}' \end{array} \right] \right\rangle} \left\{ o. \text{chaselev-owner } t \right\}$$

Specification — chaselev_pop



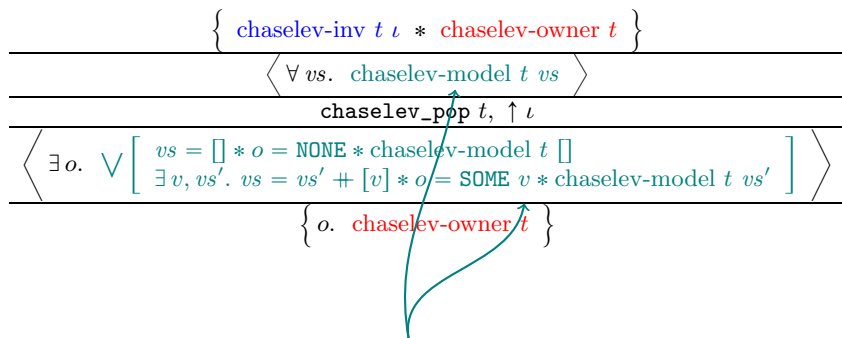
t is an instance of Chase-Lev deque.

Specification — chaselev_pop



This operation is reserved to the owner of t .

Specification — chaselev_pop

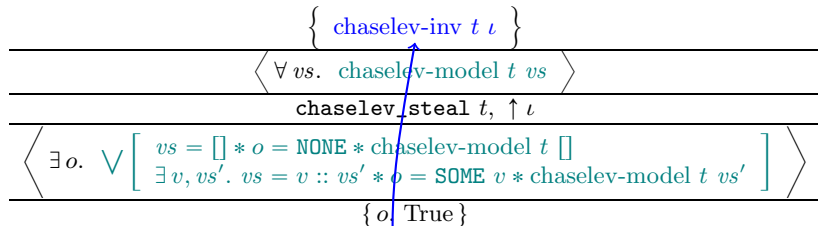


Either 1) t is seen empty
 or 2) some value v is atomically popped at the owner's end of t .

Specification — chaselev_steal

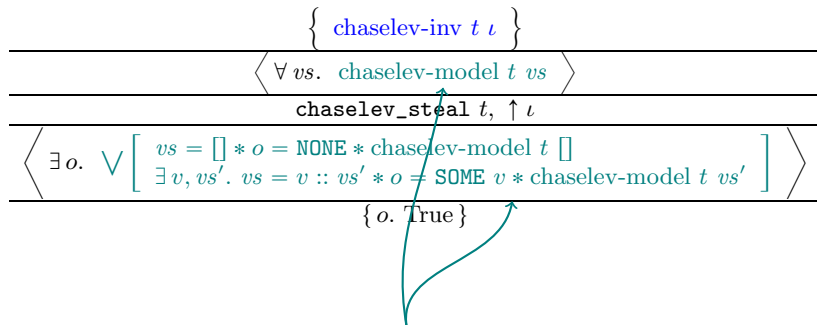
$$\frac{\left\{ \text{chaselev-inv } t \ \iota \right\}}{\frac{\left\langle \forall \text{ vs. } \text{chaselev-model } t \ \text{vs} \right\rangle}{\text{chaselev_steal } t, \uparrow \iota} \left\langle \exists o. \bigvee \left[\begin{array}{l} \text{vs} = [] * o = \text{NONE} * \text{chaselev-model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

Specification — chaselev_steal



t is an instance of Chase-Lev deque.

Specification — chaselev_steal



Either 1) t is seen empty
 or 2) some value v is atomically popped at the thieves' end of t .

Specification

Physical state

Logical state

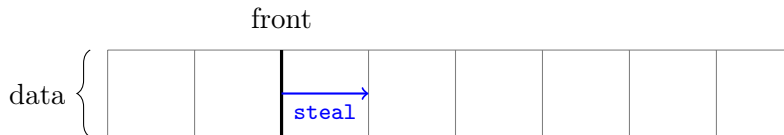
Prophecy variables

Physical state



data: infinite array storing all values

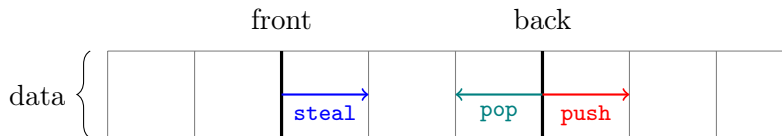
Physical state



data: infinite array storing all values

front: *monotone* index for thieves' end

Physical state

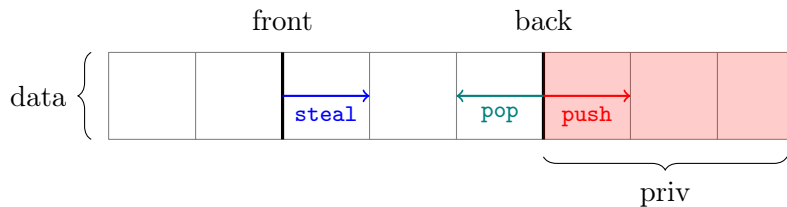


data: infinite array storing all values

front: *monotone* index for thieves' end

back: index for owner's end

Physical state



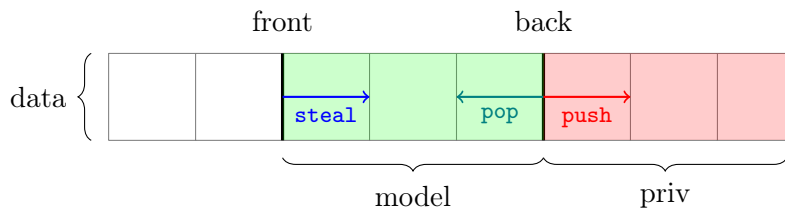
data: infinite array storing all values

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

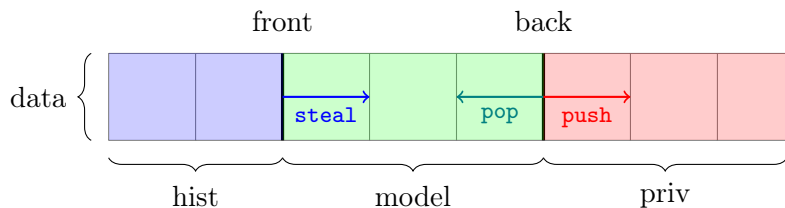
front: *monotone* index for thieves' end

back: index for owner's end

priv: list of private values (controlled by owner)

model: list of contained values

Physical state



data: infinite array storing all values

front: *monotone* index for thieves' end

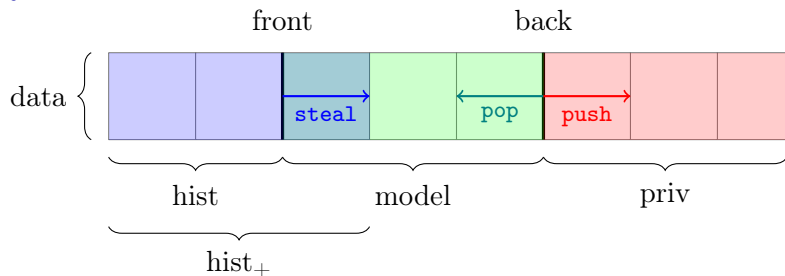
back: index for owner's end

priv: list of private values (controlled by owner)

model: list of contained values

hist: *monotone* list of 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)

model: list of contained values

hist: *monotone* list of history values

hist₊: *monotone* list of extended history values

Specification

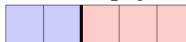
Physical state

Logical state

Prophecy variables

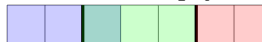
Logical state

① empty



front = back

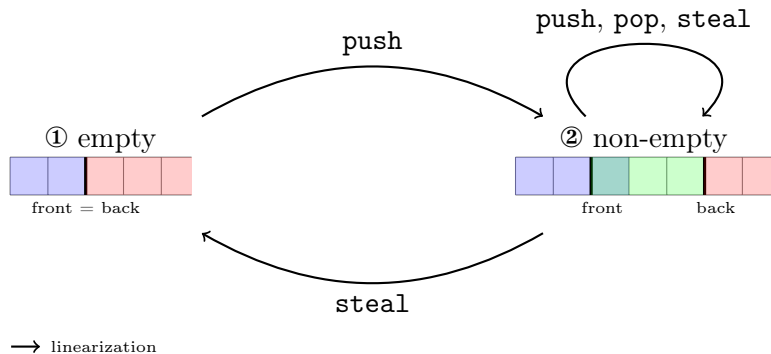
② non-empty



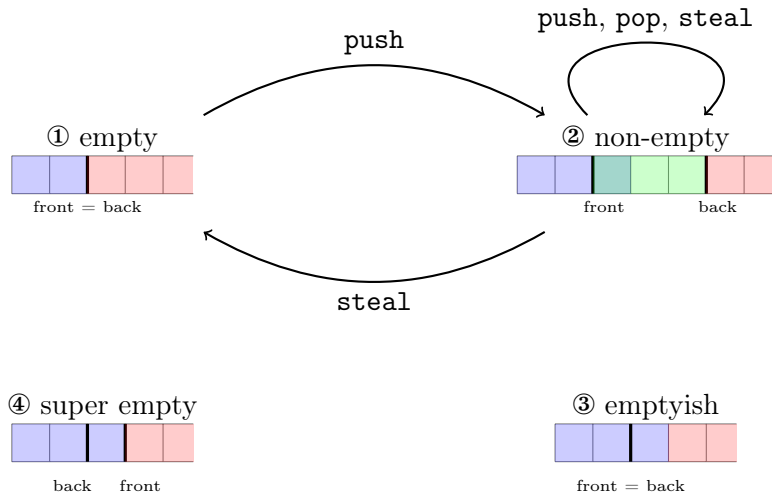
front

back

Logical state

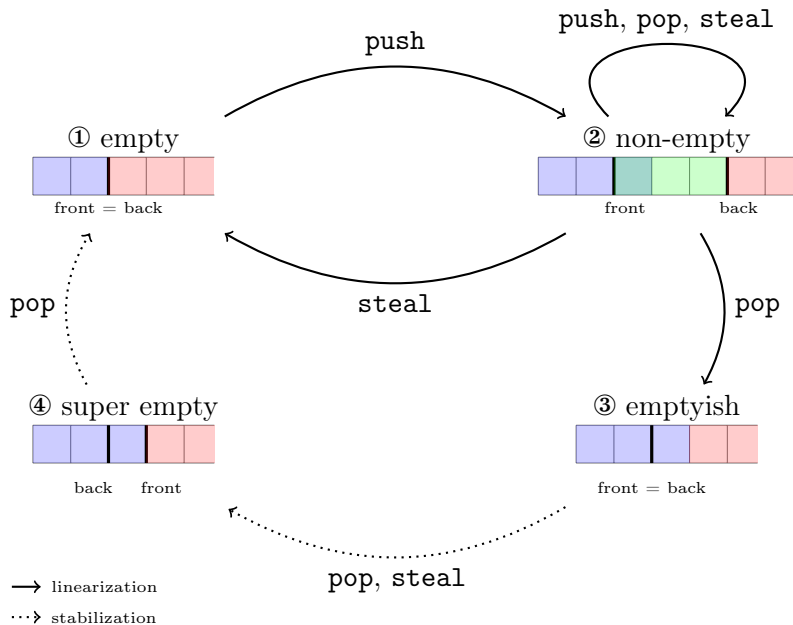


Logical state

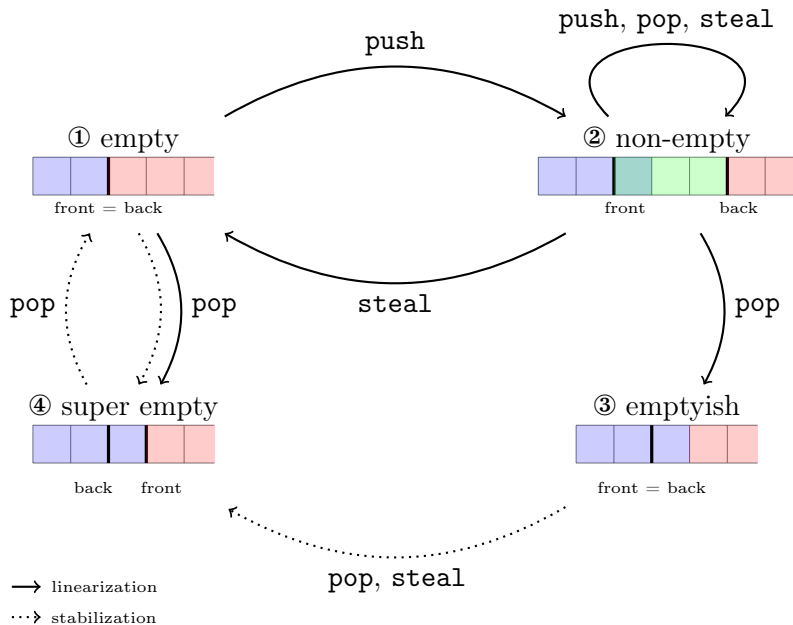


→ linearization

Logical state



Logical state



Specification

Physical state

Logical state

Prophecy variables

Prophecy variables

The future is ours: prophecy variables in separation logic.

Jung, Lepigre, Parthasarathy, Rapoport, Timany, Dreyer & Jacobs (2020).

$\{ \text{True} \} \text{NewProph } \{ \lambda p. \exists \text{prophs}. \text{proph } p \text{ prophs} \}$

$$\frac{\text{atomic } e \quad \text{proph } p \text{ prophs} \quad \left\{ \begin{array}{l} \lambda w. \forall \text{prophs}'. \\ \text{prophs} = (w, v) :: \text{prophs}' \multimap \\ \text{proph } p \text{ prophs}' \multimap \\ \Phi \ w \end{array} \right\}}{\text{WP Resolve } e \ p \ v \ \{ \Phi \}}$$

Prophecy variables with memory

$\{ \text{True} \} \text{ NewProph } \{ \lambda p. \exists \gamma, \text{prophs}. \text{proph } p \ \gamma \ \square \ \text{prophs} \}$


$$\frac{\text{atomic } e \quad \text{proph } p \ \gamma \ \text{past } \text{prophs} \quad \left\{ \begin{array}{l} \lambda w. \forall \text{prophs}'. \\ \text{prophs} = (w, v) :: \text{prophs}' \multimap \\ \text{proph } p \ \gamma \ (\text{past} \# [(w, v)]) \ \text{prophs}' \multimap \\ \Phi \ w \end{array} \right\}}{\text{WP Resolve } e \ p \ v \ \{ \Phi \}}$$

Prophecy variables with memory

$$\frac{\text{PROPHECYLBGET} \quad \text{proph } p \ \gamma \ \text{past} \ \text{prophs}}{\text{proph-lb } \gamma \ \text{prophs}}$$

$$\frac{\text{PROPHECYVALID} \quad \text{proph } p \ \gamma \ \text{past} \ \text{prophs}_1 \quad \text{proph-lb } \gamma \ \text{prophs}_2}{\exists \text{past}_1, \text{past}_2. \bigwedge \left[\begin{array}{l} \text{past} = \text{past}_1 \uplus \text{past}_2 \\ \text{past}_2 \uplus \text{prophs}_1 = \text{prophs}_2 \end{array} \right]}$$

Conclusion

- ▶ Coq mechanization is available on `github` :
`https://github.com/clef-men/caml5`
- ▶ Simplified Chase-Lev deque (one infinite array) ✓
Real-life Chase-Lev deque (multiple circular arrays) 
- ▶ Proof looks more complex than the sketch. In particular, transitions between logical states are not really formalized.
- ▶ We plan to verify more primitives (Domainslib, Taskflow) based on Chase-Lev deque. This is thanks to modularity of IRIS specifications.

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

$$\frac{\frac{\{ \text{True} \}}{\text{inf_array_make } v}}{\{ \lambda \text{ arr}. \exists \gamma. \text{inf-array-inv } \text{arr } \gamma \iota * \text{inf-array-model } \text{arr } \gamma (\lambda _ . v) \}}$$

$$\frac{\frac{\frac{\{ \text{inf-array-inv } \text{arr } \gamma \iota * 0 \leq i \}}{\langle \forall \text{ vs}. \text{inf-array-model } \text{arr } \gamma \text{ vs} \rangle}}{\text{inf_array_get } \text{arr } i, \uparrow \iota}}{\langle \exists . \text{inf-array-model } \text{arr } \gamma \text{ vs} \rangle}}{\{ \text{vs } i. \text{True} \}}$$

$$\frac{\frac{\frac{\{ \text{inf-array-inv } \text{arr } \gamma \iota * 0 \leq i \}}{\langle \forall \text{ vs}. \text{inf-array-model } \text{arr } \gamma \text{ vs} \rangle}}{\text{inf_array_set } \text{arr } i \text{ } v, \uparrow \iota}}{\langle \exists . \text{inf-array-model } \text{arr } \gamma \text{ vs} [i \mapsto v] \rangle}}{\{ (). \text{True} \}}$$

Invariant

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

Invariant

$\text{chaselev-inv-inner } \ell \ \gamma \ \iota \ \text{data } p \triangleq$

$\exists \text{ front, back, hist, model, priv, past, prophs.}$

$\left[\begin{array}{l} \ell.\text{front} \mapsto \text{front} * \ell.\text{back} \mapsto \text{back} \\ \text{---} \gamma.\text{ctl} \\ \bullet (\text{back}, \text{priv}) \\ \text{---} \gamma.\text{front} \\ \bullet \text{front} \\ \text{---} \\ * \text{inf-array-model } \text{data } \gamma.\text{data } (\text{hist} \# \text{model}) \text{ priv} \\ \text{---} \gamma.\text{model} \\ \bullet \text{model} \quad * |\text{model}| = (\text{back} - \text{front})_+ \\ \text{---} \\ \text{wise-prophet-model } p \ \gamma.\text{prophet } \text{past } \text{prophs} \\ \forall (\text{front}', _) \in \text{past}. \text{front}' < \text{front} \\ \text{chaselev-state } \gamma \ \iota \ \text{front } \text{back } \text{hist } \text{model } \text{prophs} \end{array} \right.$

State

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

State 1 (empty)

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

$$* \left[\begin{array}{l} \text{front} = \text{back} \\ \begin{array}{|c|} \hline \bullet \text{ hist} \\ \hline \end{array} * |\text{hist}| = \text{front} \\ \begin{array}{|c|} \hline \bullet \text{ --- } \cdot \text{ } \circ \text{ ---} \\ \hline \end{array} \end{array} \right] \gamma.\text{winner}$$

State 2 (non-empty)

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

$$\begin{array}{l}
 \left[\begin{array}{l}
 \text{front} < \text{back} \\
 \boxed{\bullet (\text{hist} \uplus [\text{model}[0]])}^{\gamma.\text{hist}} * |\text{hist}| = \text{front} \\
 \\
 \mathbf{match} \text{ filter } (\lambda(\text{front}', _). \text{front}' = \text{front}) \text{ prophs } \mathbf{with} \\
 | \square \Rightarrow \boxed{\bullet _ \cdot \circ _}^{\gamma.\text{winner}} \\
 | (_, id) :: _ \Rightarrow \\
 \bigvee \left[\begin{array}{l}
 \boxed{\bullet _ \cdot \circ _}^{\gamma.\text{winner}} \\
 \text{identifier } id * \exists \Phi. \boxed{\bullet (\text{front}, \Phi)}^{\gamma.\text{winner}} * \text{chaselev-au } \gamma \iota \Phi
 \end{array} \right.
 \end{array} \right]
 \end{array}$$

State 3 (emptyish)

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

$$* \left[\begin{array}{l} \text{front} = \text{back} \\ \boxed{\bullet \text{ hist}}^{\gamma.\text{hist}} * |\text{hist}| = \text{front} + 1 \\ \text{match filter } (\lambda(\text{front}', _). \text{front}' = \text{front}) \text{ prophs with} \\ | \square \Rightarrow \boxed{\circ (\text{front}, _)}^{\gamma.\text{winner}} \\ | _ \Rightarrow \exists \Phi. \boxed{\bullet (\text{front}, \Phi)}^{\gamma.\text{winner}} * \Phi \text{ (SOME hist}[\text{front}]) \end{array} \right.$$

State 4 (super empty)

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

$$* \left[\begin{array}{l} \text{front} = \text{back} + 1 \\ \begin{array}{|l} \text{---} \gamma.\text{hist} \\ \bullet \text{ hist} \end{array} * |\text{hist}| = \text{front} \\ \begin{array}{|l} \text{---} \gamma.\text{winner} \\ \bullet \text{ ---} \circ \text{ ---} \end{array} \end{array} \right]$$