Relational Views Framework for Proving Linearizability

(of concurrent libraries)

Artem Khyzha   Alexey Gotsman   Matthew Parkinson

Concurrent libraries

- Encapsulate efficient concurrent algorithms
  - Java: java.util.concurrent
  - C++: Intel Threading Building Blocks
  - C#: System.Collections.Concurrent
- Implement stacks, queues, skip lists, hash tables, etc
- It is not easy to understand why they are correct

Concurrent library specification

- A standard correctness criterion — linearizability

Implementation

concrete library

$\sqsubseteq$

Specification

abstract library

Concurrent library specification

Non-blocking stack

```
struct Node {
  Node *next; int val;
} *Top;

void push(int v) {
  Node *t, *x;
  x = new Node;
  x->val = v;
  do {
    t = Top;
    x->next = t;
  } while(!CAS(&Top,t,x)); }
```

$$\sqsubseteq$$

Atomic stack ADT

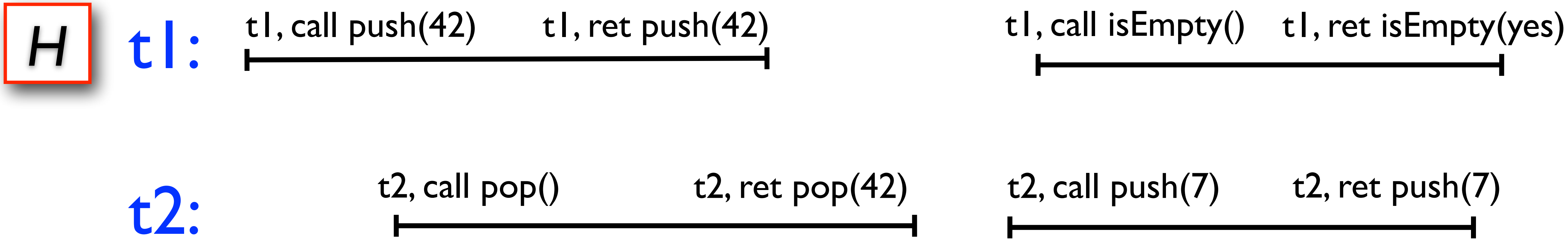
```
Sequence S;

void push(int v) {
  atomic { S = v :: S; }
}
```

Contextual refinement

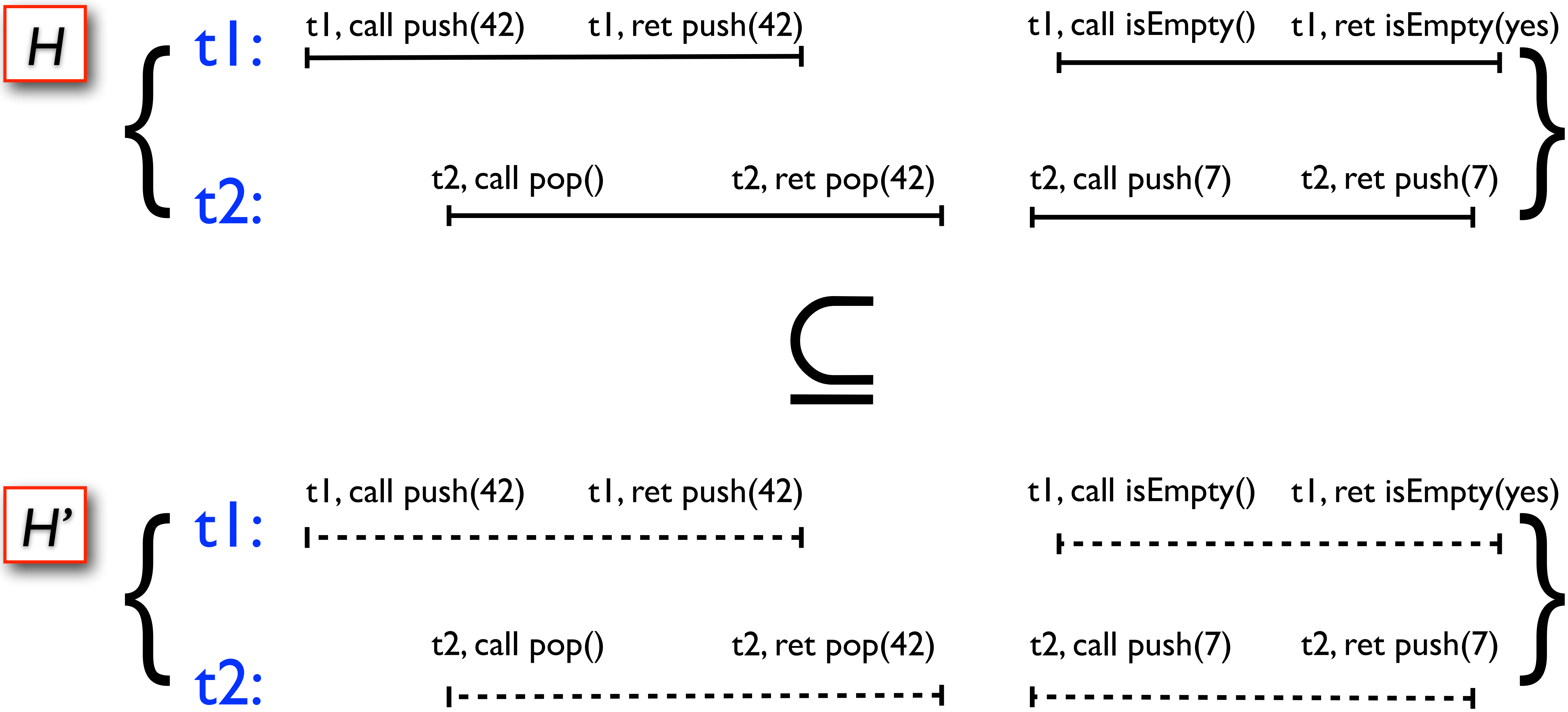
- If $L \sqsubseteq L'$, then $C(L')$ can recreate executions of $C(L)$.
- It is sound to replace a library with its specification in reasoning about its client.
- Proofs about $C(L')$ work for $C(L)$.

Linearizability



- Client observes events t, call m(a)
  and t, ret m(r)
- A history — a trace of events

Linearizability



H

t1:
t1, call push(42) t1, ret push(42)
t1, call isEmpty() t1, ret isEmpty(yes)

t2:
t2, call pop() t2, ret pop(42)
t2, call push(7) t2, ret push(7)

$\subseteq$

H’

t1:
t1, call push(42) t1, ret push(42)
t1, call isEmpty() t1, ret isEmpty(yes)

t2:
t2, call pop() t2, ret pop(42)
t2, call push(7) t2, ret push(7)

Relational Views Framework

- Recipe:
    - linearization points-based proof method
    - Views Framework for thread-modular reasoning about concurrency

A simple and generic logic for proving linearizability

Views Framework

- The Views Framework is a generalisation of Hoare-style program logics for concurrency.
- Framework can be instantiated into existing logics by adjusting parameters.

Views Framework

Rely-Guarantee

Owicki-Gries

Local Rely-Guarantee

Deny-Guarantee

RGSep

Separation logic

Concurrent Abstract Predicates

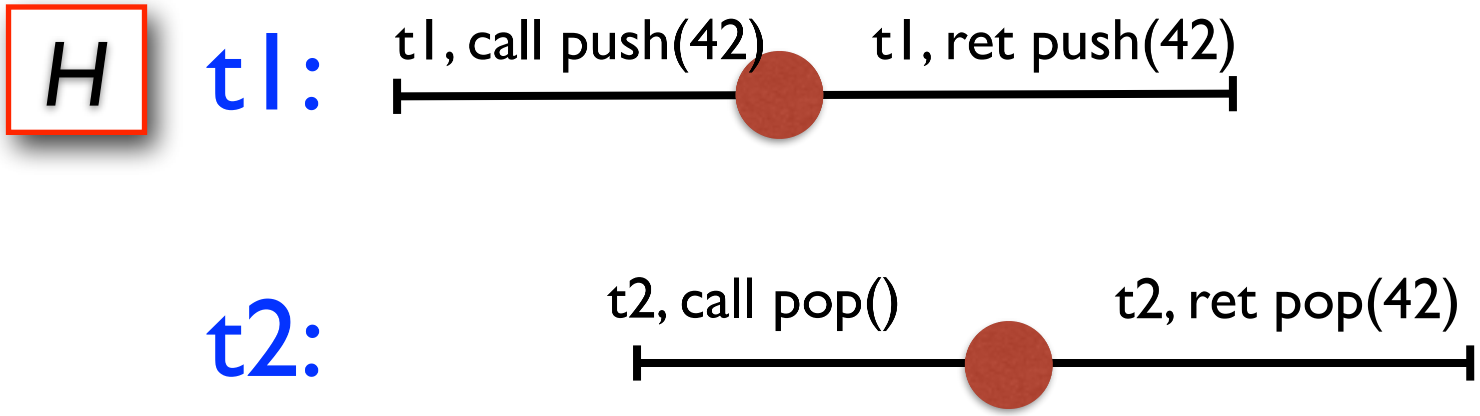
Views Framework

- The Views Framework is a generalisation of Hoare-style program logics for concurrency.
- Framework can be instantiated into existing logics by adjusting parameters.

- We extend the original framework to support proving properties of pairs of programs

---

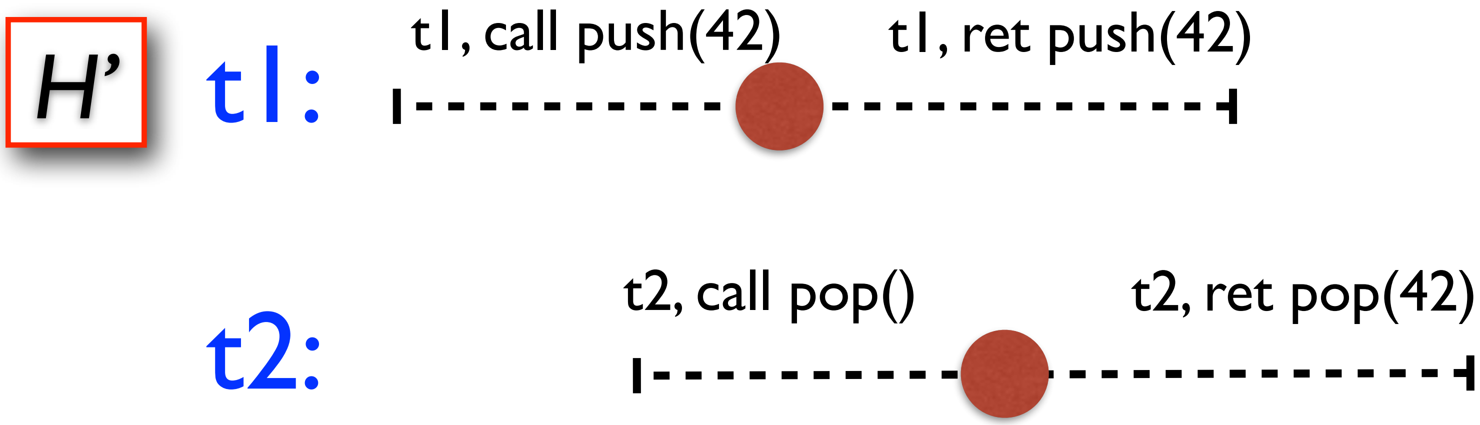
Linearization points



```
struct Node {
  Node *next; int val;
} *Top;

void push(int v) {
  Node *t, *x;
  x = new Node;
  x->val = v;
  do {
    t = Top;
    x->next = t;
  } while(!CAS(&Top,t,x)); }
```

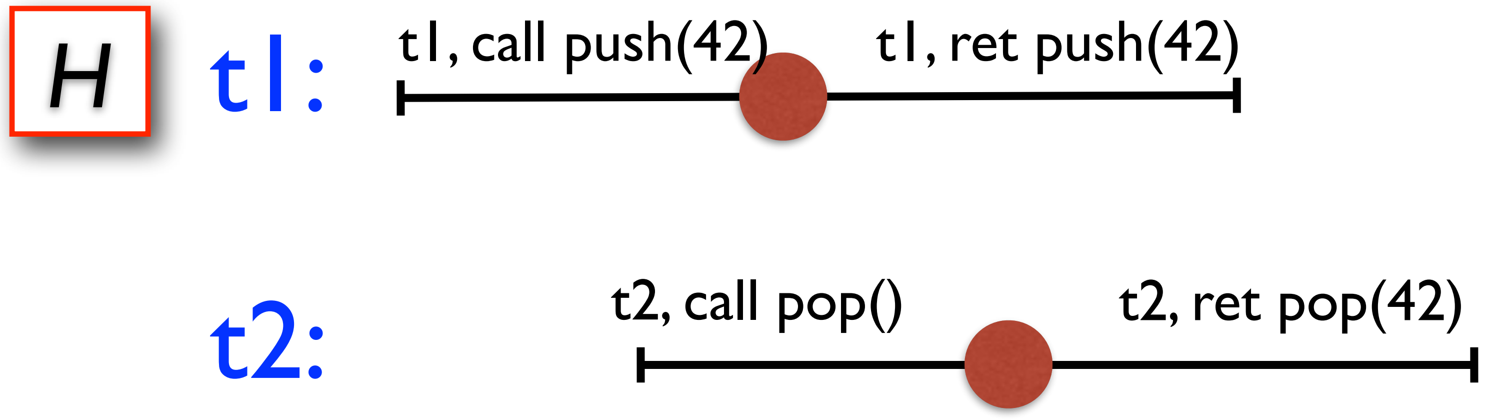
- implemented methods take effect instantly



```
Sequence S;

void push(int v) {
  atomic { S = v·S; }
}
```

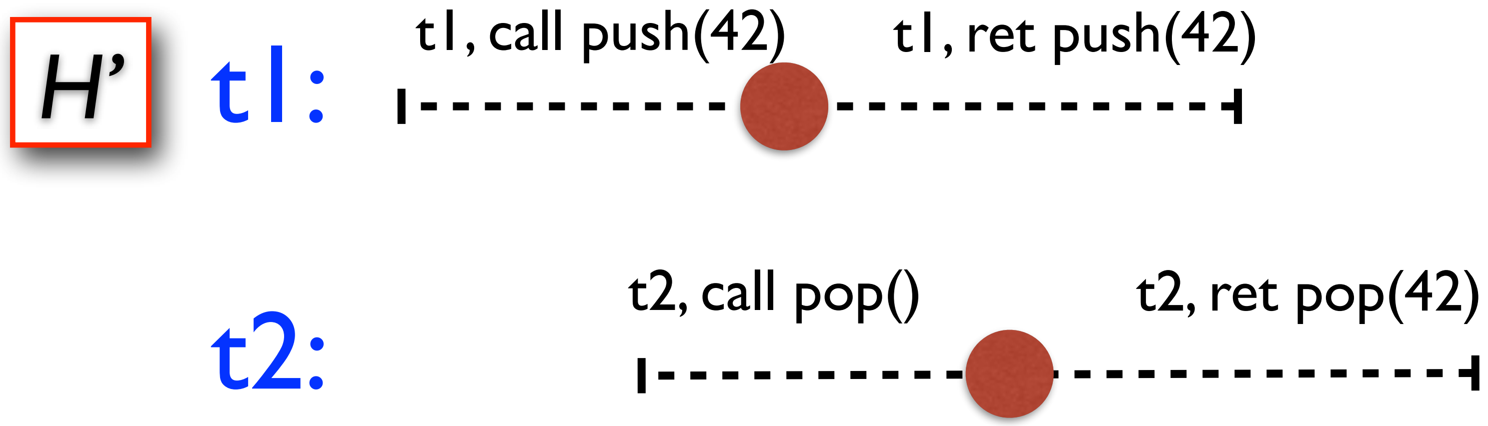
Linearizability



```
struct Node {
  Node *next; int val;
} *Top;

void push(int v) {
  Node *t, *x;
  x = new Node;
  x->val = v;
  do {
    t = Top;
    x->next = t;
  } while(!CAS(&Top,t,x)); }
```

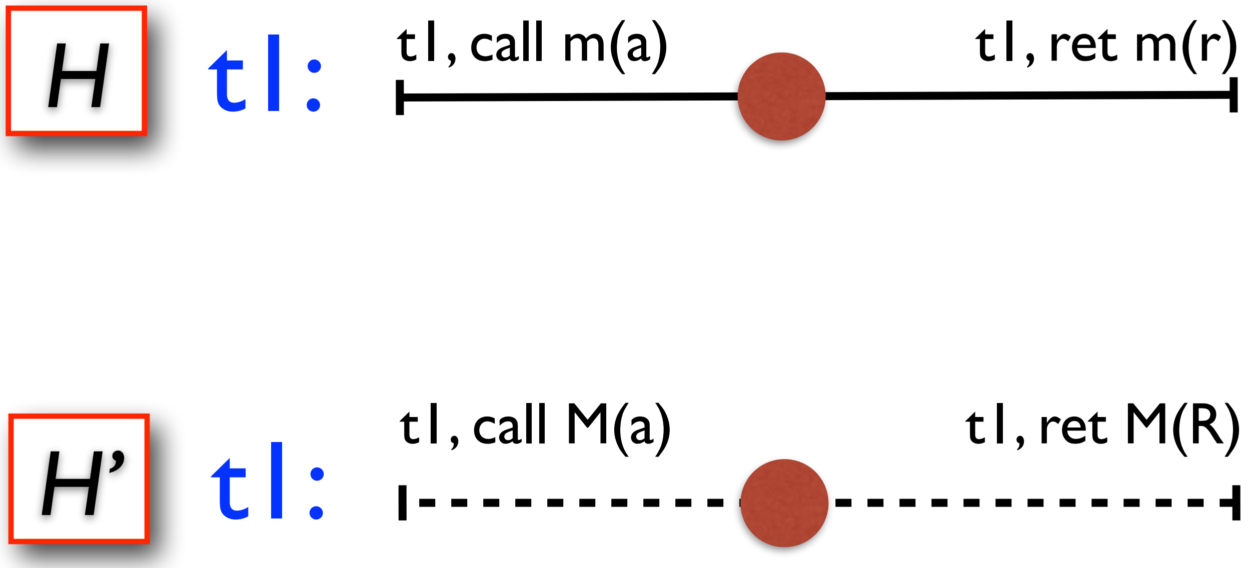
Linearization point



```
Sequence S;

void push(int v) {
  atomic { S = v·S; }
}
```

Proving linearizability



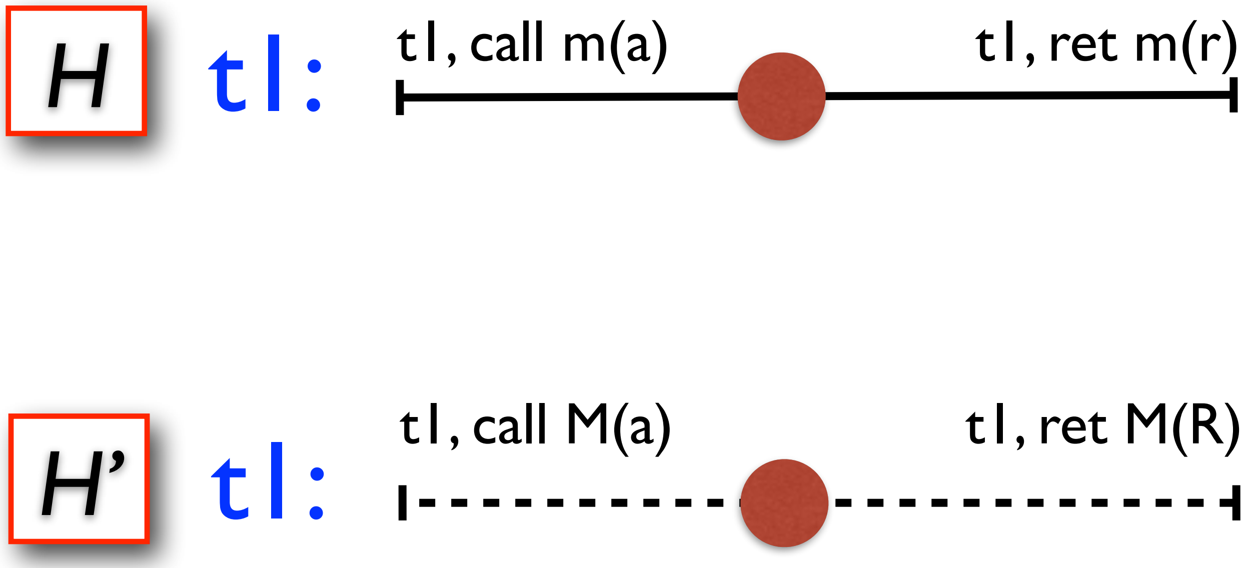
H t1: t1, call m(a) t1, ret m(r)

H' t1: t1, call M(a) t1, ret M(R)

Facts to prove:

- there is an LP in m
- it is passed once
- return values of m and M are the same

Proving linearizability



Two forms of auxiliary state:

- a state of the spec
- one-time permission to pass a LP — a **token**

Facts to prove:

- there is an LP in m
- it is passed once
- return values of m and M are the same

Specifying methods

$$\boxed{\vdash_t \{\ \mathrm{I}([\mathrm{todo}(M)]_t)\ \}\ m\ \{\ \mathrm{I}([\mathrm{done}(M)]_t)\ \}}$$

- m starts in $(s, \sigma, \mathit{tokens})$ sat. $\mathrm{I}([\mathrm{todo}(M)]_t)$,
- uses $[\mathrm{todo}(M)]_t$ as a permission to pass an LP
- finishes in $(s', \sigma', \mathit{tokens}')$ sat. $\mathrm{I}([\mathrm{done}(M)]_t)$

Relational views

- A set of *views* — assertions in a logic

  $I([todo(M)]_t)$, $I([done(M)]_t)$

- *Reification* — interpretation of views

  (s, σ, tokens) sat. $I([todo(M)]_t)$

- A *composition* operation $*$
  - encodes thread-modular reasoning method

Relational views

(inspired by separation logic)

- *Views* are sets of partial states w\ tokens
- Views = sets of (s, σ, tokens)
- States = heap configurations (Loc —> Val)
- Tokens = sets of [todo **inc**(3)]$_t$ and [done **inc**(7)]$_t$

state of the impl.	abstract state	tokens
$c \mapsto 42$	$C \Mapsto 42$	[todo **inc**(7))]$_t$

Relational views

(inspired by separation logic)

- Views are sets of partial states w\ tokens
    - represent an ownership of a part of a heap
- *Composition* forces views to describe disjoint parts of a heap

$$c \mapsto 42 * C \Mapsto 42 * [\mathrm{todo}(\mathbf{inc}(3))]t$$

$$p * p' = \{ (s \uplus s', \sigma \uplus \sigma', \tau \cup \tau') \mid (s, \sigma, \tau) \in p \textbf{ and } (s', \sigma', \tau') \in p' \}$$

Relational views

(inspired by separation logic)

- *Reification* function:
  - complements partial heap configurations arbitrarily

$$\lfloor p \rfloor = \{ (s \uplus s', \sigma \uplus \sigma', \tau) \mid (s, \sigma, \tau) \in p \}$$

Relational views: rely/guarantee

- Views are triples of (assertion, rely, guarantee)
    - assertion is a set of (state, state, tokens)
    - guarantee — thread's own transitions
    - rely — transitions by other threads
- $\vdash_t \{(P, R, G)\}\ C\ \{(Q, R, G)\}$

Relational views: rely/guarantee

- Views are triples of (assertion, rely, guarantee)
    - assertion is a set of (state, state, tokens)

- Reification erases all extra information:
    - $\lfloor (\text{assertion}, \text{rely}, \text{guarantee}) \rfloor = \text{assertion}$

Relational views: rely/guarantee

- Views are triples of (assertion, rely, guarantee)
  - assertion is a set of (state, state, tokens)

- $(P, R, G) * (P', R', G') = (P \cup P', R \cap R', G \cup G')$,
  when $G' \subseteq R, G \subseteq R'$

- Encodes a parallel composition rule

Proof rules

$$\frac{\Vdash_t \{p\}\ \alpha\ \{q\}:}{\vdash_t \{\mathcal{P}\}\ \alpha\ \{\mathcal{Q}\}}$$

$$\frac{\vdash_t \{\mathcal{P}\}\ C_1\ \{\mathcal{P}'\} \quad \vdash_t \{\mathcal{P}'\}\ C_2\ \{\mathcal{Q}\}}{\vdash_t \{\mathcal{P}\}\ C_1\ ;\ C_2\ \{\mathcal{Q}\}}$$

$$\frac{\vdash_t \{\mathcal{P}\}\ C\ \{\mathcal{Q}\}}{\vdash_t \{\mathcal{P} * \mathcal{R}\}\ C\ \{\mathcal{Q} * \mathcal{R}\}}$$

$$\frac{\vdash_t \{\mathcal{P}\}\ C_1\ \{\mathcal{Q}\} \quad \vdash_t \{\mathcal{P}\}\ C_2\ \{\mathcal{Q}\}}{\vdash_t \{\mathcal{P}\}\ C_1 + C_2\ \{\mathcal{Q}\}}$$

$$\frac{\vdash_t \{\mathcal{P}\}\ C\ \{\mathcal{Q}\}}{\vdash_t \{\exists X.\, \mathcal{P}\}\ C\ \{\exists X.\, \mathcal{Q}\}}$$

$$\frac{\mathcal{P}' \Rrightarrow \mathcal{P} \quad \vdash_t \{\mathcal{P}\}\ C\ \{\mathcal{Q}\} \quad \mathcal{Q} \Rrightarrow \mathcal{Q}'}{\vdash_t \{\mathcal{P}'\}\ C\ \{\mathcal{Q}'\}}$$

$$\frac{\vdash_t \{\mathcal{P}_1\}\ C\ \{\mathcal{Q}_1\} \quad \vdash_t \{\mathcal{P}_2\}\ C\ \{\mathcal{Q}_2\}}{\vdash_t \{\mathcal{P}_1 \vee \mathcal{P}_2\}\ C\ \{\mathcal{Q}_1 \vee \mathcal{Q}_2\}}$$

$$\frac{\vdash_t \{\mathcal{P}\}\ C\ \{\mathcal{P}\}}{\vdash_t \{\mathcal{P}\}\ C^{\star}\ \{\mathcal{P}\}}$$

Proof rules

$$\frac{\Vdash_t \{p\}\ \alpha\ \{q\}:}{\vdash_t \{\mathcal{P}\}\ \alpha\ \{\mathcal{Q}\}}$$

$$\frac{\vdash_t \{\mathcal{P}\}\ C_1\ \{\mathcal{P}'\} \qquad \vdash_t \{\mathcal{P}'\}\ C_2\ \{\mathcal{Q}\}}{\vdash_t \{\mathcal{P}\}\ C_1\ ;\ C_2\ \{\mathcal{Q}\}}$$

Rule for atomic commands

AND

linearization points

$$\frac{\vdash_t \{\mathcal{P}\}\ C\ \{\mathcal{Q}\}}{\vdash_t \{\mathcal{P} * \mathcal{R}\}\ C\ \{\mathcal{Q} * \mathcal{R}\}}$$

$\vdash_t$ $\mathcal{Q}\}$

$$\frac{\vdash_t \{\mathcal{P}\}\ C\ \{\mathcal{Q}\}}{\vdash_t \{\exists X.\mathcal{P}\}\ C\ \{\exists X.\mathcal{Q}\}}$$

$$\frac{\mathcal{P}' \Rrightarrow \mathcal{P} \qquad \vdash_t \{\mathcal{P}\}\ C\ \{\mathcal{Q}\} \qquad \mathcal{Q} \Rrightarrow \mathcal{Q}'}{\vdash_t \{\mathcal{P}'\}\ C\ \{\mathcal{Q}'\}}$$

$$\frac{\vdash_t \{\mathcal{P}_1\}\ C\ \{\mathcal{Q}_1\} \qquad \vdash_t \{\mathcal{P}_2\}\ C\ \{\mathcal{Q}_2\}}{\vdash_t \{\mathcal{P}_1 \vee \mathcal{P}_2\}\ C\ \{\mathcal{Q}_1 \vee \mathcal{Q}_2\}}$$

$$\frac{\vdash_t \{\mathcal{P}\}\ C\ \{\mathcal{P}\}}{\vdash_t \{\mathcal{P}\}\ C^{\star}\ \{\mathcal{P}\}}$$

Rule for atomic commands

$\Vdash_{t} \{p\}\ \alpha\ \{q\}$:

requirements to reasoning techniques

Rely-Guarantee

Local Rely-Guarantee

Deny-Guarantee

RGSep

Separation logic

Concurrent Abstract Predicates

Rule for atomic commands

$\Vdash_t \{p\}\ \alpha\ \{q\}$:

- follows a concrete semantics of α;

✓ `{ x -> 42 } ++x { x -> 43 }`

✗ `{ x -> 42 } --x { x -> 43 }`

Rule for atomic commands

$\Vdash_t \{p\}\ \alpha\ \{q\}$:

- follows a concrete semantics of α;
- requires simulation by an (optional) update to the abstract state;

✓ `{ x -> 42 } ++x { x -> 43 }`

✗ `{ x -> 42 } --x { x -> 43 }`

Rule for atomic commands

$$\begin{tikzcd}
s \arrow[rr, dash, "{(s, \sigma, \tau) \in \lfloor p \rfloor}"] \arrow[d, "\alpha"'] & & \sigma \arrow[d, dashed, bend right, "A"'] \arrow[d, dashed, bend left, "\mathrm{nop}"] \\
s' \arrow[rr, dash, "{(s', \sigma', \tau') \in \lfloor q \rfloor}"'] & & \sigma'
\end{tikzcd}$$

$[\mathrm{todo}(A)] \in \tau$

$[\mathrm{done}(A)] \in \tau'$

Rule for atomic commands

$$\begin{tikzcd}
s \arrow[r, dash, "(s, \sigma, \tau) \in \lfloor p * r \rfloor"] \arrow[d, "\alpha"'] & \sigma \arrow[d, dashed, bend right, "A"'] \arrow[d, dashed, bend left, "\mathrm{nop}"] \\
s' \arrow[r, dash, "(s', \sigma', \tau') \in \lfloor q * r \rfloor"'] & \sigma'
\end{tikzcd}$$

$[\mathrm{todo}(A)] \in \tau$

$[\mathrm{done}(A)] \in \tau'$

Rule for atomic commands

$$\dfrac{\begin{array}{l}
\text{for all } (s,\ \sigma,\ \tau) \in \lfloor p * r \rfloor \\
\quad \text{for all state updates } s \xrightarrow{\alpha} s' \\
\qquad \text{there exists } \sigma \xrightarrow{A} \sigma' \text{ such that} \\
\qquad\qquad\qquad \text{in } \tau\text{, [todo}(A)\text{] changes to [done}(A)\text{]} \\
\qquad\qquad\qquad \text{or } A = \mathrm{nop},\ \sigma = \sigma' \\
\qquad\qquad \text{and } (s',\ \sigma',\ \tau') \in \lfloor q * r \rfloor
\end{array}}{\vdash \{p\}\ \alpha\ \{q\}}$$

Specifying methods

$$\boxed{\vdash_t \{\ I([\mathrm{todo}(M)]_t)\ \}\ m\ \{\ I([\mathrm{done}(M)]_t)\ \}}$$

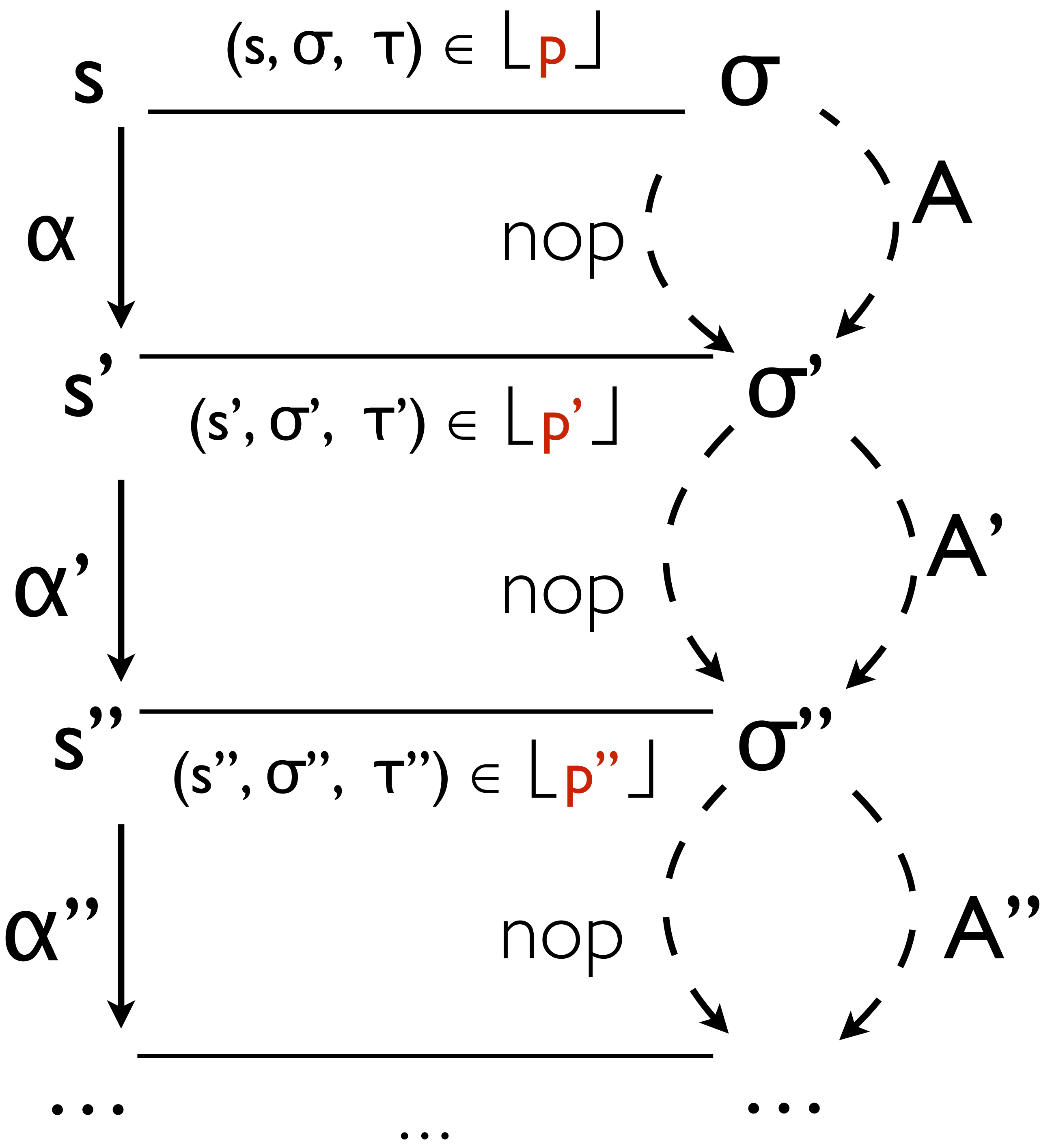
$\Vdash_t \{\ p\ \}\ C\ \{\ q\ \} =$

for all $C \xrightarrow{\alpha} C'$ exists p' .

$\Vdash_t \{\ p\ \}\ \alpha\ \{\ q\ \}$, and

$\Vdash_t \{\ p'\ \}\ C'\ \{\ q\ \}$

Specifying methods



$\Vdash_t \{ p \}\ C\ \{ q \} =$

for all $C \xrightarrow{\alpha} C'$ exists p’ .

$\Vdash_t \{ p \}\ \alpha\ \{ q \}$, and

$\Vdash_t \{ p' \}\ C'\ \{ q \}$

Soundness

- L — library/impl. L’ — library/spec.
- When every method is specified like this:
  - $\vdash_t$ { I([todo(M)]$_t$) } m { I([done(M)]$_t$) }
- histories(L) $\subseteq$ histories(L’)

Work in progress

✓ Simple logic and semantics

✓ Reasoning about LPs and helping

✗ Speculation

Instantiations:

✓ RGsep, RGsim

✗ CaReSL