iCAP: Impredicative Concurrent Abstract Predicates

Kasper Svendsen, Lars Birkedal
Aarhus University

ESOP, Grenoble
April 8, 2014

Introduction

Goal

- Modular reasoning about libraries with shared state.

This talk

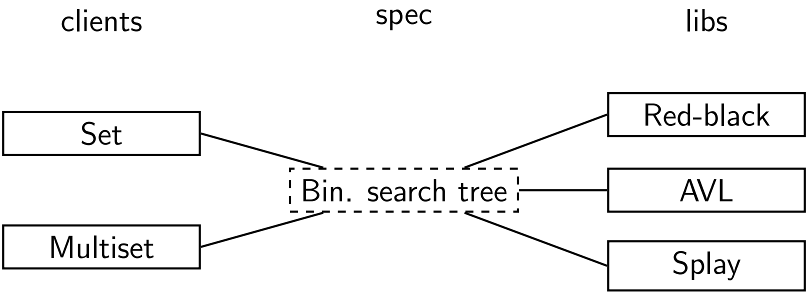
- HOSL supports modular reasoning about libraries.
- CAP supports modular reasoning about sharing.
- Neither supports granularity abstraction.

- HOSL + CAP is more than the sum of its parts:
  - Introduces a non-trivial circularity.
  - Granularity abstraction is **definable**.

Outline

Modular reasoning about libraries

- Verify libraries **independently** through **abstract** specs.



Modular reasoning about sharing

- Separation logic (SL) supports modular reasoning about state through the notion of **ownership**.
- Classic separation logic supports resources with ownership expressed in terms of primitive heap cells:

$$x \mapsto 4, \quad \mathit{lst}(x, 1 :: 2 :: \varepsilon), \quad \ldots$$

- **Ownership expressed in terms of ADT concepts** supports modular reasoning about ADTs.

Modular reasoning about sharing

- Imagine a hash-map resource, $\mathit{hash}(x, f)$, that asserts
  - that the current value of key $k \in \mathit{dom}(f)$ is $f(k)$
  - and the **exclusive** right to modify keys in $\mathit{dom}(f)$
- *hash* supports **"key" local** reasoning about hash-maps:

$$\{\mathit{hash}(x, f) * k \in \mathit{dom}(f)\} \quad \texttt{x.Get(k)} \quad \{r.\ \mathit{hash}(x, f) * r = f(k)\}$$
$$\{\mathit{hash}(x, f) * k \in \mathit{dom}(f)\} \quad \texttt{x.Set(k,v)} \quad \{\mathit{hash}(x, f[k \mapsto v])\}$$

$$\mathit{hash}(x, f \uplus g) \Leftrightarrow \mathit{hash}(x, f) * \mathit{hash}(x, g)$$

Modular reasoning about sharing

- What if a client wants to share ownership of a key?
  - Hash spec says nothing about atomicity of `Get` or `Set`
- Imagine `Get` and `Set` both **appear** to be atomic.
  - Then clients do not have to worry about interleavings.
- **Granularity abstraction** supports modular reasoning.

Modular reasoning about

```
void Set(k, v) {
  lock(this.l);
  ... // update the map
  unlock(this.l);
}
```

- What if a client wants to s
  - Hash spec says nothing about atomicity of `Get` or `Set`
- Imagine `Get` and `Set` both **appear** to be atomic.
  - Then clients do not have to worry about interleavings.
- **Granularity abstraction** supports modular reasoning.

Modular reasoning

Wishlist

- Ability to verify clients and libraries independently.
- A more abstract, user-definable notion of ownership.
- Granularity abstraction.

Modular reasoning

Wishlist

- Ability to verify clients and libraries independently.
- A more abstract, user-definable notion of ownership.
- Granularity abstraction.

	Libraries	Flex. ownership	Granularity abs.
HOSL	✓	✗	✗
CAP	✗	✓	✗
iCAP	✓	✓	✓

Biering et al., ESOP 2005; Dinsdale-Young et al., ECOOP 2010

Outline

Circularity

Example: A lock specification

$$\exists \mathsf{isLock}, \mathsf{locked} : \mathsf{Val} \times \mathsf{Prop} \to \mathsf{Prop}.\ \forall \mathsf{R} : \mathsf{Prop}.$$

$$\begin{array}{rcl} \{\mathsf{stable}(\mathsf{R}) * \mathsf{R}\} & \texttt{new Lock()} & \{\mathsf{isLock}(\mathsf{ret}, \mathsf{R})\} \\ \{\mathsf{isLock}(\mathsf{x}, \mathsf{R})\} & \texttt{x.Acquire()} & \{\mathsf{locked}(\mathsf{x}, \mathsf{R}) * \mathsf{R}\} \\ \{\mathsf{locked}(\mathsf{x}, \mathsf{R}) * \mathsf{R}\} & \texttt{x.Release()} & \{\mathsf{isLock}(\mathsf{x}, \mathsf{R})\} \end{array}$$

$$\forall \mathsf{x} : \mathsf{Val}.\ \mathsf{isLock}(\mathsf{x}, \mathsf{R}) \Leftrightarrow \mathsf{isLock}(\mathsf{x}, \mathsf{R}) * \mathsf{isLock}(\mathsf{x}, \mathsf{R})$$

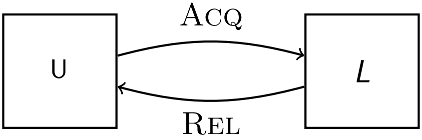
Circularity

- iCAP extends SL with **shared regions** and **protocols**.
- A protocol consists of
  - a labelled transition system describing the abstract states and operations of a shared region
  - an interpretation function describing the resources owned by the shared region in each abstract state
- Accesses to shared resources must be atomic and obey relevant protocols.

Circularity

Example: A spinlock protocol

- A lock can be in one of two abstract states:



- For a spinlock x, we interpret these states as follows:

$$I(\mathsf{x}, \mathsf{R})(s) = \begin{cases} \mathsf{x.locked} \mapsto \textbf{true} & \text{if } s = \mathsf{L} \\ \mathsf{x.locked} \mapsto \textbf{false} * \mathsf{R} * \ldots & \text{if } s = \mathsf{U} \end{cases}$$

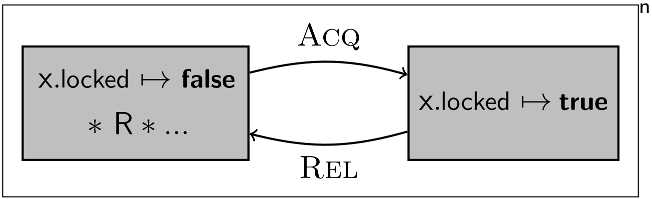
Circularity

Example: A spinlock resource

- The spinlock resource **asserts** the existence of a shared region with a spinlock protocol:

$$\mathsf{isLock}(x, R) = \exists n : \mathsf{RId}.\ \ldots$$

$*$



- iCAP assertions are predicates over heaps and **protocols**

Circularity

- HOSL assertions are predicates over heaps:

$$Prop = \mathcal{P}(Heap)$$

- iCAP assertions are predicates over heaps and **protocols**:

$$Prop \approx \mathcal{P}(Heap \times (RId \rightharpoonup_{fin} (SId \times Protocol)))$$

- Protocols consists of an LTS and an interp. function:

$$Protocol = LTS \times (SId \to Prop)$$

Outline

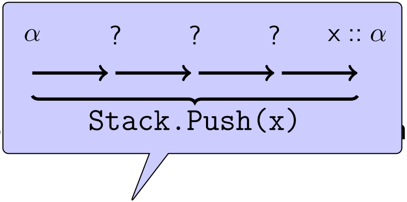
Granularity abstraction for "free"

- What does it mean for an operation to **appear** atomic?
- A standard HOSL specification relates the initial and terminal abstract state:

$$\{stack(this, \alpha)\}\texttt{Stack.Push(x)}\{stack(this, \mathsf{x} :: \alpha)\}$$

- We want to reason about the **atomic instructions** that cause the abstract state to **change**.

Granularity abstraction for "free"



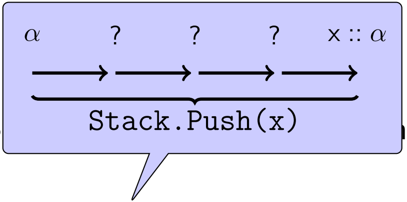
- What does it mean fo … ic?

- A standard HOSL specification relates the initial and terminal abstract state:

$$\{\mathit{stack}(\mathit{this}, \alpha)\}\texttt{Stack.Push(x)}\{\mathit{stack}(\mathit{this}, \mathsf{x} :: \alpha)\}$$

- We want to reason about the **atomic instructions** that cause the abstract state to **change**.

Granularity abstraction for "free"

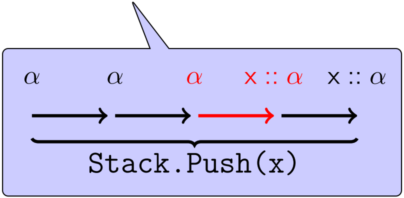


- What does it mean fo ic?

- A standard HOSL specification relates the initial and terminal abstract state:

$$\{stack(this, \alpha)\}\texttt{Stack.Push(x)}\{stack(this, \mathsf{x} :: \alpha)\}$$

- We want to reason about the **atomic instructions** that cause the abstract state to **change**.



Granularity abstraction for “free”

- We can use HOSL + CAP + phantom state to reason about atomic update of abstract state.
  - Store abstract ADT state in a phantom field.
  - Let clients reason about update of abstract state **inside** ADT method using higher-order quantification.
  - Let clients share phantom field using shared regions.

Outline

Selected applications

We have used iCAP to verify

- synchronization primitives
  - spin-locks, ticket lock, seq-lock, r/w lock, barrier
- fine-grained concurrent data structures
  - treiber's stack (with helping), michael-scott queue
- higher-order reentrant concurrent event driven code
  - joins library

Ongoing work

iCAP-TSO

- iCAP for a high-level lang. with a TSO memory model
- Two interconnected logics:
  - a high-level logic for SC reasoning
  - a low-level logic for TSO reasoning
- **Granularity abstraction for free!**

Conclusion

HOSL + CAP is more than the sum of its parts

- Introduces a non-trivial circularity.
- Solving the circularity gets us:
  - granularity abstraction
  - modular reasoning about reentrancy

  almost free of charge.

iCAP

- A logic for modular reasoning about partial correctness of concurrent, higher-order, reentrant, imperative code.