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

Circularity

Granularity abstraction

Applications and conclusion
Modular reasoning about libraries

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

![Diagram showing clients, spec, and libraries]

- Set
- Multiset
- Bin. search tree
- Red-black
- AVL
- Splay
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:
  
  \[ x \mapsto 4, \quad \text{lst}(x, 1 :: 2 :: \varepsilon), \quad \ldots \]

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

- Imagine a hash-map resource, $\text{hash}(x, f)$, that asserts
  - that the current value of key $k \in \text{dom}(f)$ is $f(k)$
  - and the **exclusive** right to modify keys in $\text{dom}(f)$

- $\text{hash}$ supports "key" **local** reasoning about hash-maps:

  \[
  \begin{align*}
  \{ \text{hash}(x, f) \star k \in \text{dom}(f) \} & \quad \text{x.Get}(k) \quad \{ r. \text{hash}(x, f) \star r = f(k) \} \\
  \{ \text{hash}(x, f) \star k \in \text{dom}(f) \} & \quad \text{x.Set}(k, v) \quad \{ \text{hash}(x, f[k \mapsto v]) \}
  \end{align*}
  \]

  \[
  \text{hash}(x, f \oplus g) \iff \text{hash}(x, f) \star \text{hash}(x, g)
  \]
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.
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

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.

<table>
<thead>
<tr>
<th>Libraries</th>
<th>Flex. ownership</th>
<th>Granularity abs.</th>
</tr>
</thead>
<tbody>
<tr>
<td>HOSL</td>
<td>✓</td>
<td>×</td>
</tr>
<tr>
<td>CAP</td>
<td>×</td>
<td>✓</td>
</tr>
<tr>
<td>iCAP</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>

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

Modular reasoning

Circularity

Granularity abstraction

Applications and conclusion
Circularity

Example: A lock specification

$\exists \text{isLock}, \text{locked} : \text{Val} \times \text{Prop} \rightarrow \text{Prop} \quad \forall R : \text{Prop}.$

\[
\{ \text{stable}(R) \ast R \} \quad \text{new Lock()} \quad \{ \text{isLock}(\text{ret}, R) \}
\]

\[
\{ \text{isLock}(x, R) \} \quad x.\text{Acquire()} \quad \{ \text{locked}(x, R) \ast R \}
\]

\[
\{ \text{locked}(x, R) \ast R \} \quad x.\text{Release()} \quad \{ \text{isLock}(x, R) \}
\]

$\forall x : \text{Val}. \text{isLock}(x, R) \iff \text{isLock}(x, R) \ast \text{isLock}(x, R)$
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:

\[ I(x, R)(s) = \begin{cases} 
  x.\text{locked} \mapsto \text{true} & \text{if } s = L \\
  x.\text{locked} \mapsto \text{false} \ast R \ast \ldots & \text{if } s = U 
\end{cases} \]
Circularity

Example: A spinlock resource

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

\[
\text{isLock}(x, R) = \exists n : RId. ...
\]

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

\[ Prop = \mathcal{P}(\text{Heap}) \]

iCAP assertions are predicates over heaps and protocols:

\[ Prop \approx \mathcal{P}(\text{Heap} \times (\text{RId} \rightarrow_{\text{fin}} (\text{SId} \times \text{Protocol}))) \]

Protocols consist of an LTS and an interp. function:

\[ \text{Protocol} = \text{LTS} \times (\text{SId} \rightarrow \text{Prop}) \]
Outline

Modular reasoning

Circularity

Granularity abstraction

Applications and conclusion
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:

\[
\{ \text{stack}(\text{this}, \alpha) \} \text{Stack.Push}(x) \{ \text{stack}(\text{this}, x :: \alpha) \}
\]

We want to reason about the atomic instructions that cause the abstract state to change.
What does it mean for an operation to appear atomic?

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

\[
\{ \text{stack}(\text{this}, \alpha) \} \text{Stack.Push}(x) \{ \text{stack}(\text{this}, x :: \alpha) \}
\]

We want to reason about the atomic instructions that cause the abstract state to change.
Granularity abstraction for “free”

- What does it mean for an operation to appear atomic?

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

\[
\{ \text{stack}(\text{this}, \alpha) \} \text{Stack.Push}(x) \{ \text{stack}(\text{this}, x :: \alpha) \}
\]

- 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

Modular reasoning

Circularity

Granularity abstraction

Applications and conclusion
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.