Iris: Higher-Order Concurrent Separation Logic

Lecture 1: Introduction and Operational Semantics of $\lambda_{\text{ref,conc}}$

Lars Birkedal

Aarhus University, Denmark

November 10, 2017
Overview

Today:

- Course Introduction
- Operational Semantics of $\lambda_{\text{ref,conc}}$
Introduction: goals of this course

- Formal verification of programs written in realistic programming languages
  - verification can mean many things, depending on which properties we try to verify
  - the properties we focus on include full functional correctness, so properties are rich / deep
- We focus on techniques that scale to concurrent higher-order imperative programs
  - important in practise
  - hard to reason about, especially modularly
Applications

- Verification of challenging concurrent libraries whose correctness is critical (interactively, in the Coq proof assistant)
- Foundation for semi-automated tools, such as Caper
- Framework for expressing and proving invariants captured by type systems.
  - ML types, runST, type-and-effect systems, Rust, …
Projects

- After this course, you can do projects related to above applications, e.g., using our Coq implementation of Iris.
A framework for higher-order concurrent separation logic

Applicable to many different programming languages (see http://iris-project.org for examples)

In this course: we fix a particular higher-order concurrent imperative programming language, called $\lambda_{\text{ref,conc}}$.

Now: syntax and operational semantics of $\lambda_{\text{ref,conc}}$. 
Syntax, 1

\[ x, y, f \in \text{Var} \]
\[ \ell \in \text{Loc} \]
\[ n \in \mathbb{Z} \]
\[ \odot ::= + | - | * | = | < | \cdots \]
\[ \text{Val} \]
\[ v ::= () | \text{true} | \text{false} | n | \ell | (v, v) | \text{inj}_1 v | \text{inj}_2 v | \text{rec } f(x) = e \]
\[ \text{Exp} \]
\[ e ::= x | n | e \odot e | () | \text{true} | \text{false} | \text{if } e \text{ then } e \text{ else } e | \ell \]
\[ | (e, e) | \pi_1 e | \pi_2 e | \text{inj}_1 e | \text{inj}_2 e \]
\[ | \text{match } e \text{ with } \text{inj}_1 x \Rightarrow e | \text{inj}_2 y \Rightarrow e \text{ end} \]
\[ | \text{rec } f(x) = e | e \ e \]
\[ | \text{ref}(e) | ! e | e \leftarrow e | \text{cas}(e, e, e) | \text{fork } \{ e \} \]
\[
ECtx \quad E ::= \quad \mathop{\_} \mid E \odot e \mid v \odot E \mid \text{if } E \text{ then } e \text{ else } e \mid (E, e) \mid (v, E) \mid \pi_1 E \mid \pi_2 E \\
\quad \mid \text{inj}_1 E \mid \text{inj}_2 E \mid \text{match } E \text{ with } \text{inj}_1 x \Rightarrow e \mid \text{inj}_2 y \Rightarrow e \text{ end} \\
\quad \mid E e \mid v E \mid \text{ref}(E) \mid ! E \mid E \leftarrow e \mid v \leftarrow E \\
\quad \mid \text{cas}(E, e, e') \mid \text{cas}(v, E, e) \mid \text{cas}(v, v', E)
\]

\[
\text{Heap} \quad h \in \text{Loc} \xrightarrow{\text{fin}} \text{Val} \\
\text{TPool} \quad \mathcal{E} \in \mathbb{N} \xrightarrow{\text{fin}} \text{Exp} \\
\text{Config} \quad \varsigma ::= (h, \mathcal{E})
\]
Pure reduction

\[ v ⊗ v' \xrightarrow{\text{pure}} v'' \]

if \( v'' = v ⊗ v' \)

if true then \( e_1 \) else \( e_2 \) \xrightarrow{\text{pure}} \( e_1 \)

if false then \( e_1 \) else \( e_2 \) \xrightarrow{\text{pure}} \( e_2 \)

\[ \pi_i (v_1, v_2) \xrightarrow{\text{pure}} v_i \]

match \( \text{inj}_i \) \( v \) with \( \text{inj}_1 x_1 \Rightarrow e_1 \mid \text{inj}_2 x_2 \Rightarrow e_2 \) end \xrightarrow{\text{pure}} \( e_i[v/x_i] \)

\[ (\text{rec } f(x) = e) v \xrightarrow{\text{pure}} e[(\text{rec } f(x) = e)/f, v/x] \]
Per-thread one-step reduction

\[(h, e) \leadsto (h, e')\]
\[(h, \text{ref}(v)) \leadsto (h[\ell \mapsto v], \ell)\]
\[(h, ! \ell) \leadsto (h, h(\ell))\]
\[(h, \ell \leftarrow v) \leadsto (h[\ell \mapsto v],())\]
\[(h, \text{cas}(\ell, v_1, v_2)) \leadsto (h[\ell \mapsto v_2], \text{true})\]
\[(h, \text{cas}(\ell, v_1, v_2)) \leadsto (h, \text{false})\]

if \(e^{\text{pure}} \leadsto e'\)
if \(\ell \not\in \text{dom}(h)\)
if \(\ell \in \text{dom}(h)\)
if \(\ell \in \text{dom}(h)\)
if \(h(\ell) = v_1\)
if \(h(\ell) \neq v_1\)
Configuration reduction

\[(h, e) \rightsquigarrow (h', e')\]
\[(h, \mathcal{E}[i \mapsto E[e]]) \rightarrow (h', \mathcal{E}[i \mapsto E[e']])\]

\[j \notin \text{dom}(\mathcal{E}) \cup \{i\}\]
\[(h, \mathcal{E}[i \mapsto E[\text{fork } \{e\}]]) \rightarrow (h, \mathcal{E}[i \mapsto E()[j \mapsto e]])\]