Iris: Higher-Order Concurrent Separation Logic

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

Lars Birkedal

Aarhus University, Denmark

April 5, 2021
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 and 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.

- Selected Example Projects:
  - Formalizing Concurrent Stacks With Helping: A Case Study In Iris, project by Daniel Gratzer and Mathias Høier, 2017.
    [A version of this is now a chapter in the lecture notes.]
  - Modular Verification of the Ticket Lock, project by Marit Ohlenbusch, 2018.
    [A version of this is now a chapter in the lecture notes.]
  - The Array-Based Queueing Lock, project by Simon Vindum and Emil Gjørup, 2019.
    [A version of this is now a chapter in the lecture notes.]
  - The CHL Lock and Logical Relations in Iris, project by Zongyaun Liu, 2020.
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

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

\[ ECtx \quad E \ ::= \quad \cdot \mid E \odot e \mid v \odot E \mid \text{if } E \text{ then } e \text{ else } e \mid (E, e) \mid (v, E) \mid \pi_1 E \mid \pi_2 E \]
\[ \quad \mid \text{inj}_1 E \mid \text{inj}_2 E \mid \text{match } E \text{ with } \text{inj}_1 x \Rightarrow e \mid \text{inj}_2 y \Rightarrow e \text{ end} \]
\[ \quad \mid E e \mid v E \mid \text{ref}(E) \mid !E \mid E \leftarrow e \mid v \leftarrow E \]
\[ \quad \mid \text{cas}(E, e, e') \mid \text{cas}(v, E, e) \mid \text{cas}(v, v', E) \]

\[ \text{Heap} \quad h \in \text{Loc} \xrightarrow{\text{fin}} \text{Val} \]
\[ \text{TPool} \quad \mathcal{E} \in \mathbb{N} \xrightarrow{\text{fin}} \text{Exp} \]
\[ \text{Config} \quad \varsigma \ ::= \quad (h, \mathcal{E}) \]
We write \( \lambda x. e \) for the term \( \text{rec } f(x) = e \) where \( f \) is some fresh variable not appearing in \( e \). Thus \( \lambda x. e \) is a non-recursive function with argument \( x \) and body \( e \).

We write \( \text{let } x = e_1 \text{ in } e_2 \) for the term \( (\lambda x. e_2)e_1 \).

We write \( e_1; e_2 \) for the term \( \text{let } x = e_1 \text{ in } e_2 \) where \( x \) is some fresh variable not appearing in \( e_2 \).
Pure reduction

\[ v \odot v' \xrightarrow{\text{pure}} v'' \quad \text{if } v'' = v \odot v' \]

\[
\begin{align*}
\text{if true then } e_1 \text{ else } e_2 & \xrightarrow{\text{pure}} e_1 \\
\text{if false then } e_1 \text{ else } e_2 & \xrightarrow{\text{pure}} e_2 \\
\pi_i (v_1, v_2) & \xrightarrow{\text{pure}} v_i \\
\text{match inj}_i v \text{ with inj}_1 x_1 \Rightarrow e_1 | \text{ inj}_2 x_2 \Rightarrow e_2 \text{ end} & \xrightarrow{\text{pure}} e_i[v/x_i] \\
(rec \ f(x) = e) v & \xrightarrow{\text{pure}} e[(rec \ f(x) = e)/f, v/x]
\end{align*}
\]
Per-thread one-step reduction

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

if \(e \rightsquigarrow e'\)
if \(\ell \notin \text{dom}(h)\)
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

\[
(h, e) \rightsquigarrow (h', e')
\]

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

\[
j \notin \text{dom}(\mathcal{E}) \cup \{i\}
\]

\[
(h, \mathcal{E}[i \mapsto E[\text{fork } \{e\}]]) \rightarrow (h, \mathcal{E}[i \mapsto E[(])][j \mapsto e])
\]
Example: factorial

Let \( \nu = \text{rec fac}(n) = \text{if } n = 0 \text{ then } 1 \text{ else } n \times \text{fac}(n - 1) \). We wish to consider the evaluation of \( \nu(2) \). For each step, think about what the evaluation context is.

\[
\begin{align*}
(\emptyset, [0 \mapsto \nu(2)]) & \Rightarrow (\emptyset, [0 \mapsto \text{if } 2 = 0 \text{ then } 1 \text{ else } 2 \times \nu(2 - 1)]) \\
& \Rightarrow (\emptyset, [0 \mapsto \text{if false then } 1 \text{ else } 2 \times \nu(2 - 1)]) \\
& \Rightarrow (\emptyset, [0 \mapsto 2 \times \nu(2 - 1)]) \\
& \Rightarrow (\emptyset, [0 \mapsto 2 \times \nu(1)]) \\
& \Rightarrow (\emptyset, [0 \mapsto 2 \times \text{if } 1 = 0 \text{ then } 1 \text{ else } 1 \times \nu(1 - 1)]) \\
& \Rightarrow (\emptyset, [0 \mapsto 2 \times \text{if false then } 1 \text{ else } 1 \times \nu(1 - 1)]) \\
& \Rightarrow (\emptyset, [0 \mapsto 2 \times 1 \times \nu(1 - 1)]) \\
& \Rightarrow (\emptyset, [0 \mapsto 2 \times 1 \times \nu(0)]) \\
& \Rightarrow (\emptyset, [0 \mapsto 2 \times 1 \times \text{if } 0 = 0 \text{ then } 1 \text{ else } 0 \times \nu(0 - 1)]) \\
& \Rightarrow (\emptyset, [0 \mapsto 2 \times 1 \times \text{if true then } 1 \text{ else } 0 \times \nu(0 - 1)]) \\
& \Rightarrow (\emptyset, [0 \mapsto 2 \times 1 \times 1]) \\
& \Rightarrow (\emptyset, [0 \mapsto 2 \times 1]) \\
& \Rightarrow (\emptyset, [0 \mapsto 2])
\end{align*}
\]
Example: functional lists

Let \( \nu = \text{rec inc}(xs) = \text{match } xs \text{ with } \text{inj}_1 x_1 \Rightarrow xs \mid \text{inj}_2 x_2 \Rightarrow \text{inj}_2 (1 + \pi_1 x_2, \text{inc}(\pi_2 x_2)) \text{ end} \).

We consider the evaluation of \( \nu(\text{inj}_2 (7, \text{inj}_1 ())) \) (\( \nu \) applied to the list with one element, the value 7).

\[
([], [0 \mapsto \nu(\text{inj}_2 (7, \text{inj}_1 ()))] )
\sim (\[], [0 \mapsto \text{match } \text{inj}_2 (7, \text{inj}_1 ()) \text{ with } \text{inj}_1 x_1 \Rightarrow \text{inj}_2 (7, \text{inj}_1 ()) \mid \text{inj}_2 x_2 \Rightarrow \text{inj}_2 (1 + \pi_1 x_2, \nu(\pi_2 x_2)) \text{ end}])
\sim (\[], [0 \mapsto \text{inj}_2 (1 + \pi_1 (7, \text{inj}_1 ()), \nu(\pi_2 (7, \text{inj}_1 ()))))]])
\sim (\[], [0 \mapsto \text{inj}_2 (1 + 7, \nu(\pi_2 (7, \text{inj}_1 ()))))]])
\sim (\[], [0 \mapsto \text{inj}_2 (8, \nu(\pi_2 (7, \text{inj}_1 ()))))]])
\sim (\[], [0 \mapsto \text{inj}_2 (8, \nu(\text{inj}_1 ()))))]
\sim (\[], [0 \mapsto \text{inj}_2 (8, \text{match } \text{inj}_1 () \text{ with } \text{inj}_1 x_1 \Rightarrow \text{inj}_1 () \mid \text{inj}_2 x_2 \Rightarrow \text{inj}_2 (1 + \pi_1 x_2, \nu(\pi_2 x_2)) \text{ end}]))
\sim (\[], [0 \mapsto \text{inj}_2 (8, \text{inj}_1 ())])
\]
Example: references

Let \( v = \text{rec swap}(p) = \text{let } z = !(\pi_1 p) \text{ in } \pi_1 p \leftarrow !(\pi_2 p); \pi_2 p \leftarrow z. \)

We consider the evaluation of \( v(\text{ref}(2), \text{ref}(3)). \)

\[
\begin{align*}
&([], [0 \mapsto v(\text{ref}(2), \text{ref}(3))]) \\
&([l_1 \mapsto 2], [0 \mapsto v(l_1, \text{ref}(3))]) \\
&([l_1 \mapsto 2, l_{18} \mapsto 3], [0 \mapsto v(l_1, l_{18})]) \\
&([l_1 \mapsto 2, l_{18} \mapsto 3], [0 \mapsto \text{let } z = !(\pi_1 (l_1, l_{18})) \text{ in } \pi_1 (l_1, l_{18}) \leftarrow !(\pi_2 (l_1, l_{18}); \pi_2 (l_1, l_{18}) \leftarrow z]) \\
&([l_1 \mapsto 2, l_{18} \mapsto 3], [0 \mapsto \text{let } z = !l_1 \text{ in } \pi_1 (l_1, l_{18}) \leftarrow !(\pi_2 (l_1, l_{18}); \pi_2 (l_1, l_{18}) \leftarrow z]) \\
&([l_1 \mapsto 2, l_{18} \mapsto 3], [0 \mapsto \pi_1 (l_1, l_{18}) \leftarrow !(\pi_2 (l_1, l_{18}); \pi_2 (l_1, l_{18}) \leftarrow 2]) \\
&([l_1 \mapsto 2, l_{18} \mapsto 3], [0 \mapsto l_1 \leftarrow !(\pi_2 (l_1, l_{18}); \pi_2 (l_1, l_{18}) \leftarrow 2]) \\
&([l_1 \mapsto 2, l_{18} \mapsto 3], [0 \mapsto l_1 \leftarrow !l_{18}; \pi_2 (l_1, l_{18}) \leftarrow 2]) \\
&([l_1 \mapsto 2, l_{18} \mapsto 3], [0 \mapsto l_1 \leftarrow 3; \pi_2 (l_1, l_{18}) \leftarrow 2]) \\
&([l_1 \mapsto 3, l_{18} \mapsto 3], [0 \mapsto \pi_2 (l_1, l_{18}) \leftarrow 2]) \\
&([l_1 \mapsto 3, l_{18} \mapsto 3], [0 \mapsto l_{18} \leftarrow 2]) \\
&([l_1 \mapsto 3, l_{18} \mapsto 2], [0 \mapsto ()])
\end{align*}
\]
Example: concurrency

Let $e = \text{fork } \{(1 + 2) + 3\}; (4 + 5) + 6$.

We consider the evaluation of $e$, and just show one possible reduction sequence (more than one possible). Notice the interleaving of reductions in the two threads.

([], [0 $\mapsto$ $e$])

([], [0 $\mapsto$ (); (4 + 5) + 6, 1 $\mapsto$ (1 + 2) + 3])

([], [0 $\mapsto$ (4 + 5) + 6, 1 $\mapsto$ (1 + 2) + 3])

([], [0 $\mapsto$ 9 + 6, 1 $\mapsto$ (1 + 2) + 3])

([], [0 $\mapsto$ 9 + 6, 1 $\mapsto$ 3 + 3])

([], [0 $\mapsto$ 9 + 6, 1 $\mapsto$ 6])

([], [0 $\mapsto$ 15, 1 $\mapsto$ 6])