Iris: Higher-Order Concurrent Separation Logic

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

Lars Birkedal

Aarhus University, Denmark

April 5, 2021

Overview

Today:

- Course Introduction
- Operational Semantics of $\lambda_{\mathrm{ref,conc}}$

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 assisistant)
- 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:
  - Verifying Hash Tables in Iris, M.Sc. thesis by Esben Clausen, 2017.
  - 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.

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 $\lambda_{\mathrm{ref,conc}}$.
- Now: syntax and operational semantics of $\lambda_{\mathrm{ref,conc}}$.

Syntax, I

$$
\begin{array}{lrcl}
 & x, y, f & \in & \mathit{Var} \\
 & \ell & \in & \mathit{Loc} \\
 & n & \in & \mathbb{Z} \\
 & \circledcirc & ::= & + \mid - \mid * \mid = \mid < \mid \cdots \\
\mathit{Val} & v & ::= & () \mid \mathsf{true} \mid \mathsf{false} \mid n \mid \ell \mid (v, v) \mid \mathsf{inj}_1\, v \mid \mathsf{inj}_2\, v \mid \mathsf{rec}\, f(x) = e \\
\mathit{Exp} & e & ::= & x \mid n \mid e \circledcirc e \mid () \mid \mathsf{true} \mid \mathsf{false} \mid \mathsf{if}\, e\, \mathsf{then}\, e\, \mathsf{else}\, e \mid \ell \\
 & & \mid & (e, e) \mid \pi_1\, e \mid \pi_2\, e \mid \mathsf{inj}_1\, e \mid \mathsf{inj}_2\, e \\
 & & \mid & \mathsf{match}\, e\, \mathsf{with}\, \mathsf{inj}_1\, x \Rightarrow e \mid \mathsf{inj}_2\, y \Rightarrow e\, \mathsf{end} \\
 & & \mid & \mathsf{rec}\, f(x) = e \mid e\ e \\
 & & \mid & \mathsf{ref}(e) \mid\ !\, e \mid e \leftarrow e \mid \mathsf{cas}(e, e, e) \mid \mathsf{fork}\ \{e\}
\end{array}
$$

Syntax, II

$$
\begin{array}{llll}
\mathit{ECtx} & E & ::= & - \mid E \circledcirc e \mid v \circledcirc E \mid \mathsf{if}\ E\ \mathsf{then}\ e\ \mathsf{else}\ e \mid (E, e) \mid (v, E) \mid \pi_1\ E \mid \pi_2\ E \\
 & & \mid & \mathsf{inj}_1\ E \mid \mathsf{inj}_2\ E \mid \mathsf{match}\ E\ \mathsf{with}\ \mathsf{inj}_1\ x \Rightarrow e \mid \mathsf{inj}_2\ y \Rightarrow e\ \mathsf{end} \\
 & & \mid & E\ e \mid v\ E \mid \mathsf{ref}(E) \mid\ !\ E \mid E \leftarrow e \mid v \leftarrow E \\
 & & \mid & \mathsf{cas}(E, e, e') \mid \mathsf{cas}(v, E, e) \mid \mathsf{cas}(v, v', E) \\
\mathit{Heap} & h & \in & \mathit{Loc} \xrightarrow{\text{fin}} \mathit{Val} \\
\mathit{TPool} & \mathcal{E} & \in & \mathbb{N} \xrightarrow{\text{fin}} \mathit{Exp} \\
\mathit{Config} & \varsigma & ::= & (h, \mathcal{E})
\end{array}
$$

Syntactic Sugar

- We write $\lambda x.e$ for the term $\mathsf{rec}\, f(x) = e$ 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 $\mathsf{let}\, x = e_1 \,\mathsf{in}\, e_2$ for the term $(\lambda x.e_2)e_1$.
- We write $e_1; e_2$ for the term $\mathsf{let}\, x = e_1 \,\mathsf{in}\, e_2$ where x is some fresh variable not appearing in e_2.

Pure reduction

$$
\begin{aligned}
v \circledcirc v' &\overset{\text{pure}}{\rightsquigarrow} v'' && \text{if } v'' = v \circledcirc v' \\
\textsf{if true then } e_1 \textsf{ else } e_2 &\overset{\text{pure}}{\rightsquigarrow} e_1 \\
\textsf{if false then } e_1 \textsf{ else } e_2 &\overset{\text{pure}}{\rightsquigarrow} e_2 \\
\pi_i\,(v_1, v_2) &\overset{\text{pure}}{\rightsquigarrow} v_i \\
\textsf{match inj}_i\, v \textsf{ with inj}_1\, x_1 \Rightarrow e_1 \mid \textsf{inj}_2\, x_2 \Rightarrow e_2 \textsf{ end} &\overset{\text{pure}}{\rightsquigarrow} e_i[v/x_i] \\
(\textsf{rec}\, f(x) = e)\, v &\overset{\text{pure}}{\rightsquigarrow} e[(\textsf{rec}\, f(x) = e)/f, v/x]
\end{aligned}
$$

Per-thread one-step reduction

$$
\begin{aligned}
(h, e) &\rightsquigarrow (h, e') && \text{if } e \overset{\text{pure}}{\rightsquigarrow} e' \\
(h, \mathsf{ref}(v)) &\rightsquigarrow (h[\ell \mapsto v], \ell) && \text{if } \ell \notin \mathrm{dom}(h) \\
(h, !\,\ell) &\rightsquigarrow (h, h(\ell)) && \text{if } \ell \in \mathrm{dom}(h) \\
(h, \ell \leftarrow v) &\rightsquigarrow (h[\ell \mapsto v], ()) && \text{if } \ell \in \mathrm{dom}(h) \\
(h, \mathsf{cas}(\ell, v_1, v_2)) &\rightsquigarrow (h[\ell \mapsto v_2], \mathsf{true}) && \text{if } h(\ell) = v_1 \\
(h, \mathsf{cas}(\ell, v_1, v_2)) &\rightsquigarrow (h, \mathsf{false}) && \text{if } h(\ell) \neq v_1
\end{aligned}
$$

Configuration reduction

$$\frac{(h, e) \leadsto (h', e')}{(h, \mathcal{E}[i \mapsto E[e]]) \to (h', \mathcal{E}[i \mapsto E[e']])}$$

$$\frac{j \notin \mathrm{dom}(\mathcal{E}) \cup \{i\}}{(h, \mathcal{E}[i \mapsto E[\mathsf{fork}\ \{e\}]]) \to (h, \mathcal{E}[i \mapsto E[()]][j \mapsto e])}$$

Example: factorial

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

$$
\begin{aligned}
([], [0 \mapsto v(2)]) &\rightsquigarrow ([], [0 \mapsto \textbf{if}\; 2 = 0 \;\textbf{then}\; 1 \;\textbf{else}\; 2 * v(2-1)]) \\
&\rightsquigarrow ([], [0 \mapsto \textbf{if false then}\; 1 \;\textbf{else}\; 2 * v(2-1)]) \\
&\rightsquigarrow ([], [0 \mapsto 2 * v(2-1)]) \\
&\rightsquigarrow ([], [0 \mapsto 2 * v(1)]) \\
&\rightsquigarrow ([], [0 \mapsto 2 * \textbf{if}\; 1 = 0 \;\textbf{then}\; 1 \;\textbf{else}\; 1 * v(1-1)]) \\
&\rightsquigarrow ([], [0 \mapsto 2 * \textbf{if false then}\; 1 \;\textbf{else}\; 1 * v(1-1)]) \\
&\rightsquigarrow ([], [0 \mapsto 2 * 1 * v(1-1)]) \\
&\rightsquigarrow ([], [0 \mapsto 2 * 1 * v(0)]) \\
&\rightsquigarrow ([], [0 \mapsto 2 * 1 * \textbf{if}\; 0 = 0 \;\textbf{then}\; 1 \;\textbf{else}\; 0 * v(0-1)]) \\
&\rightsquigarrow ([], [0 \mapsto 2 * 1 * \textbf{if true then}\; 1 \;\textbf{else}\; 0 * v(0-1)]) \\
&\rightsquigarrow ([], [0 \mapsto 2 * 1 * 1]) \\
&\rightsquigarrow ([], [0 \mapsto 2 * 1]) \\
&\rightsquigarrow ([], [0 \mapsto 2])
\end{aligned}
$$

Example: functional lists

Let $v = \mathsf{rec}\,\mathsf{inc}(xs) = \mathsf{match}\ xs\ \mathsf{with}\ \mathsf{inj}_1\, x_1 \Rightarrow xs \mid \mathsf{inj}_2\, x_2 \Rightarrow \mathsf{inj}_2\,(1 + \pi_1\, x_2, \mathsf{inc}(\pi_2\, x_2))\ \mathsf{end}$.
We consider the evaluation of $v(\mathsf{inj}_2\,(7, \mathsf{inj}_1\,()))$ (v applied to the list with one element, the value 7).

$$
\begin{aligned}
&([], [0 \mapsto v(\mathsf{inj}_2\,(7, \mathsf{inj}_1\,()))])\\
&\rightsquigarrow ([], [0 \mapsto \mathsf{match}\ \mathsf{inj}_2\,(7, \mathsf{inj}_1\,())\ \mathsf{with}\ \mathsf{inj}_1\, x_1 \Rightarrow \mathsf{inj}_2\,(7, \mathsf{inj}_1\,()) \mid \mathsf{inj}_2\, x_2 \Rightarrow \mathsf{inj}_2\,(1 + \pi_1\, x_2, v(\pi_2\, x_2))\ \mathsf{end}])\\
&\rightsquigarrow ([], [0 \mapsto \mathsf{inj}_2\,(1 + \pi_1\,(7, \mathsf{inj}_1\,()), v(\pi_2\,(7, \mathsf{inj}_1\,())))])\\
&\rightsquigarrow ([], [0 \mapsto \mathsf{inj}_2\,(1 + 7, v(\pi_2\,(7, \mathsf{inj}_1\,())))])\\
&\rightsquigarrow ([], [0 \mapsto \mathsf{inj}_2\,(8, v(\pi_2\,(7, \mathsf{inj}_1\,())))])\\
&\rightsquigarrow ([], [0 \mapsto \mathsf{inj}_2\,(8, v(\mathsf{inj}_1\,()))])\\
&\rightsquigarrow ([], [0 \mapsto \mathsf{inj}_2\,(8, \mathsf{match}\ \mathsf{inj}_1\,()\ \mathsf{with}\ \mathsf{inj}_1\, x_1 \Rightarrow \mathsf{inj}_1\,() \mid \mathsf{inj}_2\, x_2 \Rightarrow \mathsf{inj}_2\,(1 + \pi_1\, x_2, v(\pi_2\, x_2))\ \mathsf{end})])\\
&\rightsquigarrow ([], [0 \mapsto \mathsf{inj}_2\,(8, \mathsf{inj}_1\,())])
\end{aligned}
$$

Example: references

Let $v = \mathsf{rec}\,\mathsf{swap}(p) = \mathsf{let}\, z = !(\pi_1\, p)\, \mathsf{in}\, \pi_1\, p \leftarrow !(\pi_2\, p); \pi_2\, p \leftarrow z$.
We consider the evaluation of $v(\mathsf{ref}(2), \mathsf{ref}(3))$.

$$([], [0 \mapsto v(\mathsf{ref}(2), \mathsf{ref}(3))])$$
$$([l_1 \mapsto 2], [0 \mapsto v(l_1, \mathsf{ref}(3))])$$
$$([l_1 \mapsto 2, l_{18} \mapsto 3], [0 \mapsto v(l_1, l_{18})])$$
$$([l_1 \mapsto 2, l_{18} \mapsto 3], [0 \mapsto \mathsf{let}\, z = !(\pi_1\,(l_1, l_{18}))\, \mathsf{in}\, \pi_1\,(l_1, l_{18}) \leftarrow !(\pi_2\,(l_1, l_{18})); \pi_2\,(l_1, l_{18}) \leftarrow z])$$
$$([l_1 \mapsto 2, l_{18} \mapsto 3], [0 \mapsto \mathsf{let}\, z = !\, l_1\, \mathsf{in}\, \pi_1\,(l_1, l_{18}) \leftarrow !(\pi_2\,(l_1, l_{18})); \pi_2\,(l_1, l_{18}) \leftarrow z])$$
$$([l_1 \mapsto 2, l_{18} \mapsto 3], [0 \mapsto \mathsf{let}\, z = 2\, \mathsf{in}\, \pi_1\,(l_1, l_{18}) \leftarrow !(\pi_2\,(l_1, l_{18})); \pi_2\,(l_1, l_{18}) \leftarrow z])$$
$$([l_1 \mapsto 2, l_{18} \mapsto 3], [0 \mapsto \pi_1\,(l_1, l_{18}) \leftarrow !(\pi_2\,(l_1, l_{18})); \pi_2\,(l_1, l_{18}) \leftarrow 2])$$
$$([l_1 \mapsto 2, l_{18} \mapsto 3], [0 \mapsto l_1 \leftarrow !(\pi_2\,(l_1, l_{18})); \pi_2\,(l_1, l_{18}) \leftarrow 2])$$
$$([l_1 \mapsto 2, l_{18} \mapsto 3], [0 \mapsto l_1 \leftarrow !\, l_{18}; \pi_2\,(l_1, l_{18}) \leftarrow 2])$$
$$([l_1 \mapsto 2, l_{18} \mapsto 3], [0 \mapsto l_1 \leftarrow 3; \pi_2\,(l_1, l_{18}) \leftarrow 2])$$
$$([l_1 \mapsto 3, l_{18} \mapsto 3], [0 \mapsto \pi_2\,(l_1, l_{18}) \leftarrow 2])$$
$$([l_1 \mapsto 3, l_{18} \mapsto 3], [0 \mapsto l_{18} \leftarrow 2])$$
$$([l_1 \mapsto 3, l_{18} \mapsto 2], [0 \mapsto ()])$$

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.

$$
\begin{aligned}
&([], [0 \mapsto e]) \\
&([], [0 \mapsto (); (4+5)+6, 1 \mapsto (1+2)+3]) \\
&([], [0 \mapsto (4+5)+6, 1 \mapsto (1+2)+3]) \\
&([], [0 \mapsto 9+6, 1 \mapsto (1+2)+3]) \\
&([], [0 \mapsto 9+6, 1 \mapsto 3+3]) \\
&([], [0 \mapsto 9+6, 1 \mapsto 6]) \\
&([], [0 \mapsto 15, 1 \mapsto 6])
\end{aligned}
$$