Iris: Higher-Order Concurrent Separation Logic

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

Lars Birkedal

Aarhus University, Denmark

November 10, 2017

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.

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, I

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

Syntax, II

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

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

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

Configuation reduction

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

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