Verifying functional correctness of message-passing programs in Coq

Robbert Krebbers, TU Delft

Joint work with
Jonas Kastberg Hinrichsen, ITU
Jesper Bengtson, ITU

4 June 2020 @ VEST Workshop, Online

Type checking message-passing programs using session types

Example program:

$$\texttt{let}\,(c, c') = \texttt{new\_chan}\,()\,\texttt{in}$$
$$\texttt{fork}\,\{\texttt{let}\,x = \texttt{recv}\,c'\,\texttt{in}\,\texttt{send}\,c'\,(x+2)\}\,;$$
$$\texttt{send}\,c\;40; \texttt{recv}\,c$$

Session types:

$$c : \mathbf{!}\mathsf{N}.\mathbf{?}\mathsf{N}.\mathsf{end} \quad \texttt{and} \quad c' : \mathbf{?}\mathsf{N}.\mathbf{!}\mathsf{N}.\mathsf{end}$$

Properties obtained:

- ☑ Type safety / session fidelity
- ☒ Functional correctness

How to prove functional correctness of message-passing programs

Combine

- **Session Types** [Honda et al., ESOP'98]
  - Type system for channels
  - Example: **!**N.**?**N.end
  - Ensures safety automatically through static type checking
- **Concurrent Separation Logic** [O'Hearn & Brooks, CONCUR'04]
  - Logic for reasoning about concurrent programs with mutable state.
  - Example: $\{x \mapsto a * y \mapsto b\}\ \texttt{swap}\ x\ y\ \{x \mapsto b * y \mapsto a\}$
  - Establish functional correctness through interactive or semi-automated proofs

Actris [Kastberg Hinrichsen, Bengtson, Krebbers; POPL'20]

A concurrent separation logic for proving functional correctness of programs that combine message passing with other programming and concurrency paradigms

- New notion of dependent separation protocols for reasoning about message passing in separation logic
- Integration with Iris and its existing concurrency mechanisms
- Verification of feature-complete programs including a variant of map-reduce
- A full mechanization of all of the above in Coq with tactics for interactive program proofs

Dependent separation protocols

Dependent separation protocols:

Example:

$$\mathbf{!}\,(x : \mathsf{N})\,\langle x\rangle\{10 < x\}.\,\mathbf{?}\,\langle x + 2\rangle\{\mathsf{True}\}.\,\mathsf{end}$$

Session types:

Example:

$$\mathbf{!}\mathsf{N}.\mathbf{?}\mathsf{N}.\mathsf{end}$$

Dependent separation protocols

Dependent separation protocols:

Example:

$$\mathbf{!}\,(x : \mathsf{N})\,\langle x\rangle\{10 < x\}.\,\mathbf{?}\,\langle x + 2\rangle\{\mathsf{True}\}.\,\mathsf{end}$$

Protocols:

$$\begin{aligned} \mathit{prot} \triangleq\ & \mathbf{!}\,\vec{x}{:}\vec{\tau}\,\langle v\rangle\{P\}.\,\mathit{prot} \\ & \mid \mathbf{?}\,\vec{x}{:}\vec{\tau}\,\langle v\rangle\{P\}.\,\mathit{prot} \\ & \mid \mathsf{end} \end{aligned}$$

Session types:

Example:

$$\mathbf{!}\mathsf{N}.\mathbf{?}\mathsf{N}.\mathsf{end}$$

Protocols:

$$\begin{aligned} \mathit{st} \triangleq\ & \mathbf{!}\,T.\mathit{st} \\ & \mid \mathbf{?}\,T.\mathit{st} \\ & \mid \mathsf{end} \end{aligned}$$

Dependent separation protocols

Dependent separation protocols:

Example:

$$!\,(x : \mathrm{N})\,\langle x\rangle\{10 < x\}.\,?\,\langle x+2\rangle\{\mathrm{True}\}.\,\mathrm{end}$$

Protocols:

$$\begin{aligned} prot \triangleq\ & !\,\vec{x}{:}\vec{\tau}\,\langle v\rangle\{P\}.\,prot \\ |\ & ?\,\vec{x}{:}\vec{\tau}\,\langle v\rangle\{P\}.\,prot \\ |\ & \mathrm{end} \end{aligned}$$

Duality:

$$\begin{aligned} \overline{!\,\vec{x}{:}\vec{\tau}\,\langle v\rangle\{P\}.\,prot} &= ?\,\vec{x}{:}\vec{\tau}\,\langle v\rangle\{P\}.\,\overline{prot} \\ \overline{?\,\vec{x}{:}\vec{\tau}\,\langle v\rangle\{P\}.\,prot} &= !\,\vec{x}{:}\vec{\tau}\,\langle v\rangle\{P\}.\,\overline{prot} \\ \overline{\mathrm{end}} &= \mathrm{end} \end{aligned}$$

Session types:

Example:

$$!\mathrm{N}.?\mathrm{N}.\mathrm{end}$$

Protocols:

$$\begin{aligned} st \triangleq\ & !\,T.st \\ |\ & ?\,T.st \\ |\ & \mathrm{end} \end{aligned}$$

Duality:

$$\begin{aligned} \overline{!\,T.st} &= ?\,T.\overline{st} \\ \overline{?\,T.st} &= !\,T.\overline{st} \\ \overline{\mathrm{end}} &= \mathrm{end} \end{aligned}$$

Proof rules for dependent separation protocols

Dependent separation protocols:

$\{\mathsf{True}\}$
  `new_chan ()`
$\{(c, c').\ c \rightarrowtail \mathit{prot} * c' \rightarrowtail \overline{\mathit{prot}}\}$

$\{c \rightarrowtail \mathbf{!}\,\vec{x}{:}\vec{\tau}\,\langle v\rangle\{P\}.\ \mathit{prot} * P[\vec{t}/\vec{x}]\}$
  `send` c $(v[\vec{t}/\vec{x}])$
$\{c \rightarrowtail \mathit{prot}[\vec{t}/\vec{x}]\}$

$\{c \rightarrowtail \mathbf{?}\,\vec{x}{:}\vec{\tau}\,\langle v\rangle\{P\}.\ \mathit{prot}\}$
  `recv` c
$\{w.\ \exists\vec{t}.\ (w = v[\vec{t}/\vec{x}]) * c \rightarrowtail \mathit{prot}[\vec{t}/\vec{x}] * P[\vec{t}/\vec{x}]\}$

Session types:

`newchan` $() : \mathit{st} \otimes \overline{\mathit{st}}$

`send` : $(\mathbf{!}\,T.\mathit{st} \otimes T) \multimap \mathit{st}$

`recv` : $\mathbf{?}\,T.\mathit{st} \multimap (T \otimes \mathit{st})$

Example – Dependency between messages

Example program:

$$
\begin{array}{l}
\{\mathsf{True}\} \\
\quad \texttt{let}\ (c, c') = \texttt{new\_chan}\ ()\ \texttt{in} \\
\quad \texttt{fork}\ \{\texttt{let}\ x = \texttt{recv}\ c'\ \texttt{in}\ \texttt{send}\ c'\ (x + 2)\}\,; \\
\quad \texttt{send}\ c\ 40; \texttt{recv}\ c \\
\{w.\ w = 42\}
\end{array}
$$

Example – Dependency between messages

Example program:

$$
\begin{array}{l}
\{\mathsf{True}\} \\
\quad \texttt{let}\ (c, c') = \texttt{new\_chan}\ ()\ \texttt{in} \\
\quad \texttt{fork}\ \{\texttt{let}\ x = \texttt{recv}\ c'\ \texttt{in}\ \texttt{send}\ c'\ (x + 2)\}\,; \\
\quad \texttt{send}\ c\ 40; \texttt{recv}\ c \\
\{w.\ w = 42\}
\end{array}
$$

Dependent separation protocols:

$$
\begin{aligned}
c &\rightarrowtail \mathbf{!}\,(x : \mathsf{N})\,\langle x \rangle \{\mathsf{True}\}.\,\mathbf{?}\,\langle x + 2 \rangle \{\mathsf{True}\}.\,\mathsf{end} \\
c' &\rightarrowtail \mathbf{?}\,(x : \mathsf{N})\,\langle x \rangle \{\mathsf{True}\}.\,\mathbf{!}\,\langle x + 2 \rangle \{\mathsf{True}\}.\,\mathsf{end}
\end{aligned}
$$

Example – Dependency between messages

Example program:

$$
\begin{array}{l}
\{\mathsf{True}\} \\
\quad \texttt{let}\ (c, c') = \texttt{new\_chan}\ ()\ \texttt{in} \\
\quad \texttt{fork}\ \{\texttt{let}\ x = \texttt{recv}\ c'\ \texttt{in}\ \texttt{send}\ c'\ (x+2)\}\,; \\
\quad \texttt{send}\ c\ 40; \texttt{recv}\ c \\
\{w.\ w = 42\}
\end{array}
$$

Dependent separation protocols:

$$c \rightarrowtail \mathbf{!}\,(x : \mathsf{N})\,\langle x \rangle \{\mathsf{True}\}.\,\mathbf{?}\,\langle x+2 \rangle \{\mathsf{True}\}.\,\mathsf{end}$$

$$c' \rightarrowtail \mathbf{?}\,(x : \mathsf{N})\,\langle x \rangle \{\mathsf{True}\}.\,\mathbf{!}\,\langle x+2 \rangle \{\mathsf{True}\}.\,\mathsf{end}$$

Properties obtained:

- ☑ Type safety / session fidelity
- ☑ Functional correctness

Example – References

Example program:

$$
\begin{array}{l}
\{\mathsf{True}\} \\
\quad \texttt{let}\,(c, c') = \texttt{new\_chan}\,()\,\texttt{in} \\
\quad \texttt{fork}\,\{\texttt{let}\,x = \texttt{recv}\,c'\,\texttt{in}\,x \leftarrow (!\,x + 2); \texttt{send}\,c'\,()\}\,; \\
\quad \texttt{let}\,y = \texttt{ref}\,(40)\,\texttt{in}\,\texttt{send}\,c\,y; \texttt{recv}\,c; !\,y \\
\{w.\, w = 42\}
\end{array}
$$

Example – References

Example program:

$$
\begin{array}{l}
\{\mathsf{True}\} \\
\quad \texttt{let}\,(c, c') = \texttt{new\_chan}\,()\,\texttt{in} \\
\quad \texttt{fork}\,\{\texttt{let}\,x = \texttt{recv}\,c'\,\texttt{in}\,x \leftarrow (!\,x + 2); \texttt{send}\,c'\,()\}\,; \\
\quad \texttt{let}\,y = \texttt{ref}\,(40)\,\texttt{in}\,\texttt{send}\,c\,y; \texttt{recv}\,c; !\,y \\
\{w.\, w = 42\}
\end{array}
$$

Dependent separation protocols:

$$c \rightarrowtail \mathbf{!}\,(\ell : \mathsf{Loc})(x : \mathsf{N})\,\langle \ell \rangle \{\ell \mapsto n\}.\,\mathbf{?}\,\langle () \rangle \{\ell \mapsto (x + 2)\}.\,\mathsf{end}$$

$$c' \rightarrowtail \mathbf{?}\,(\ell : \mathsf{Loc})(x : \mathsf{N})\,\langle \ell \rangle \{\ell \mapsto n\}.\,\mathbf{!}\,\langle () \rangle \{\ell \mapsto (x + 2)\}.\,\mathsf{end}$$

Example – References

Example program:

$$
\begin{array}{l}
\{\mathsf{True}\} \\
\quad \mathtt{let}\,(c, c') = \mathtt{new\_chan}\,()\,\mathtt{in} \\
\quad \mathtt{fork}\,\{\mathtt{let}\,x = \mathtt{recv}\,c'\,\mathtt{in}\,x \leftarrow (!\,x + 2); \mathtt{send}\,c'\,()\}\,; \\
\quad \mathtt{let}\,y = \mathtt{ref}\,(40)\,\mathtt{in}\,\mathtt{send}\,c\,y; \mathtt{recv}\,c; !\,y \\
\{w.\, w = 42\}
\end{array}
$$

Dependent separation protocols:

$$c \rightarrowtail \mathbf{!}\,(\ell : \mathsf{Loc})(x : \mathsf{N})\,\langle \ell \rangle \{\ell \mapsto n\}.\,\mathbf{?}\,\langle () \rangle \{\ell \mapsto (x + 2)\}.\,\mathsf{end}$$

$$c' \rightarrowtail \mathbf{?}\,(\ell : \mathsf{Loc})(x : \mathsf{N})\,\langle \ell \rangle \{\ell \mapsto n\}.\,\mathbf{!}\,\langle () \rangle \{\ell \mapsto (x + 2)\}.\,\mathsf{end}$$

Properties obtained:

- ☑ Type safety / session fidelity
- ☑ Functional correctness

Soundness of Actris

If $\{\mathrm{True}\}\, e\, \{v.\, \phi(v)\}$ is provable in Actris then:

- ☑ **Type safety/session fidelity:** e will not crash and not send wrong messages
- ☑ **Functional correctness:** If e terminates with v, the postcondition $\phi(v)$ holds

Soundness of Actris

If $\{\mathsf{True}\}\ e\ \{v.\ \phi(v)\}$ is provable in Actris then:

- ☑ **Type safety/session fidelity:** e will not crash and not send wrong messages
- ☑ **Functional correctness:** If e terminates with v, the postcondition $\phi(v)$ holds

Obtained by modeling Actris as an embedded domain-specific logic in **Iris**

Iris [Jung, Krebbers et al.; POPL'15, ICFP'16, ESOP'17, JFP'18]

A powerful, general, language-independent, framework for modeling your own domain specific higher-order separation logics with powerful tactics in Coq

Iris [Jung, Krebbers et al.; POPL'15, ICFP'16, ESOP'17, JFP'18]

A powerful, general, language-independent, framework for modeling your own domain specific higher-order separation logics with powerful tactics in Coq

- **Powerful:** supports reasoning about intricate concurrent programs

Iris [Jung, Krebbers et al.; POPL'15, ICFP'16, ESOP'17, JFP'18]

A powerful, general, language-independent, framework for modeling your own domain specific higher-order separation logics with powerful tactics in Coq

- **Powerful:** supports reasoning about intricate concurrent programs
- **General:** unifies the reasoning principles in many other logics

Iris [Jung, Krebbers et al.; POPL'15, ICFP'16, ESOP'17, JFP'18]

A powerful, general, language-independent, framework for modeling your own domain specific higher-order separation logics with powerful tactics in Coq

- **Powerful:** supports reasoning about intricate concurrent programs
- **General:** unifies the reasoning principles in many other logics
- **Language-independent:** parameterized by the language

Iris [Jung, Krebbers et al.; POPL'15, ICFP'16, ESOP'17, JFP'18]

A powerful, general, language-independent, framework for modeling your own domain specific higher-order separation logics with powerful tactics in Coq

- **Powerful:** supports reasoning about intricate concurrent programs
- **General:** unifies the reasoning principles in many other logics
- **Language-independent:** parameterized by the language
- **Modeling logics:** can be used to model domain-specific logics

Iris [Jung, Krebbers et al.; POPL'15, ICFP'16, ESOP'17, JFP'18]

A powerful, general, language-independent, framework for modeling your own domain specific higher-order separation logics with powerful tactics in Coq

- **Powerful:** supports reasoning about intricate concurrent programs
- **General:** unifies the reasoning principles in many other logics
- **Language-independent:** parameterized by the language
- **Modeling logics:** can be used to model domain-specific logics
- **Tactics in Coq:** for interactive correctness proofs of programs

Implementation and model of Actris in Iris

Approach:

- Implement `new_chan`, `send`, and `recv` as a library using lock-protected buffers
- Define $c \rightarrowtail \mathit{prot}$ using Iris's invariant and ghost state machinery
- Prove Actris's proof rules as lemmas in Iris

Implementation and model of Actris in Iris

Approach:

- Implement `new_chan`, `send`, and `recv` as a library using lock-protected buffers
- Define $c \rightarrowtail \mathit{prot}$ using Iris's invariant and ghost state machinery
- Prove Actris's proof rules as lemmas in Iris

Benefits:

- ☑ Can readily reuse all powerful reasoning mechanisms of Iris
- ☑ Can readily reuse Iris's support for interactive proofs in Coq
- ☑ Actris's soundness result is a corollary of Iris's soundness
- ☑ Very small Coq mechanization
  (200 lines for channel implementation and proofs, 1000 lines for the definition and proof rules of $c \rightarrowtail \mathit{prot}$, 450 lines for Coq tactics specific for message passing)

Demo in Coq