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:

```ocaml
let (c, c’) = new chan () in
fork {let x = recv c’ in send c’ (x + 2)};
send c 40; recv c
```

Session types:

```
c : !N.?N.end and c’ : ?N.!N.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: \( \{x \mapsto a \ast y \mapsto b\} \) swap \( x \) \( y \) \( \{x \mapsto b \ast y \mapsto a\} \)
  - Establish functional correctness through interactive or semi-automated proofs
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:

! (x : N) ⟨x⟩{10 < x}. ? ⟨x + 2⟩{True}. end

Session types:

Example:

!N.?N.end
Dependent separation protocols

**Dependent separation protocols:**

Example:

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

Protocols:

\[
\text{prot} \triangleq ! \vec{x} : \vec{\tau} \langle v \rangle \{ P \} . \text{prot}
\]

\[
\mid ? \vec{x} : \vec{\tau} \langle v \rangle \{ P \} . \text{prot}
\]

\[
\mid \text{end}
\]

**Session types:**

Example:

\( ! N . ? N . \text{end} \)

Protocols:

\[
\text{st} \triangleq ! T . \text{st}
\]

\[
\mid ? T . \text{st}
\]

\[
\mid \text{end}
\]
Dependent separation protocols

Example:

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

Protocols:

$$prot \triangleq ! \vec{x} : \vec{\tau} \langle v \rangle \{ P \}. \ prot$$

$$| \ ? \vec{x} : \vec{\tau} \langle v \rangle \{ P \}. \ prot$$

$$| \ \text{end}$$

Duality:

$$! \vec{x} : \vec{\tau} \langle v \rangle \{ P \}. \ prot \triangleq \ ? \vec{x} : \vec{\tau} \langle v \rangle \{ P \}. \overline{prot}$$

$$? \vec{x} : \vec{\tau} \langle v \rangle \{ P \}. \overline{prot} \triangleq ! \vec{x} : \vec{\tau} \langle v \rangle \{ P \}. \overline{prot}$$

$$\overline{\text{end}} = \text{end}$$

Session types:

Example:

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

Protocols:

$$st \triangleq ! T. \overline{st}$$

$$| \ ? T. \overline{st}$$

$$| \ \text{end}$$

Duality:

$$! T. \overline{st} \triangleq ? T. \overline{st}$$

$$? T. \overline{st} \triangleq ! T. \overline{st}$$

$$\overline{\text{end}} = \text{end}$$
Proof rules for dependent separation protocols

**Dependent separation protocols:**

\[ \{ \text{True} \} \]
\[ \text{new-chan} () \]
\[ \{ (c, c'). c \mapsto \text{prot} \land c' \mapsto \text{prot} \} \]

\[ \{ c \mapsto ! x: \bar{r} \langle v \rangle \{ P \}. \text{prot} \land P[t/x] \} \]
\[ \text{send} \ c \ (v[t/x]) \]
\[ \{ c \mapsto \text{prot}[t/x] \} \]

\[ \{ c \mapsto ? x: \bar{r} \langle v \rangle \{ P \}. \text{prot} \} \]
\[ \text{recv} \ c \]
\[ \{ w. \exists t. (w = v[t/x]) \land c \mapsto \text{prot}[t/x] \land P[t/x] \} \]

**Session types:**

\[ \text{newchan} () : st \otimes \bar{st} \]

\[ \text{send} : (! T.st \otimes T) \to st \]

\[ \text{recv} : ? T.st \to (T \otimes st) \]
Example – Dependency between messages

Example program:

```ocaml
{True}
let (c, c') = new_chan () in
fork {let x = recv c' in send c' (x + 2)};
send c 40; recv c
{w. w = 42}
```
Example — Dependency between messages

Example program:

```
{True}
  let (c, c') = new_chan () in
  fork {let x = recv c' in send c' (x + 2)};
  send c 40; recv c

{w. w = 42}
```

Dependent separation protocols:

```
c  \mapsto  ! (x : N) \langle x \rangle \{True\}. ? \langle x + 2 \rangle \{True\}. end

\n
c'  \mapsto  ? (x : N) \langle x \rangle \{True\}. ! \langle x + 2 \rangle \{True\}. end
```
Example – Dependency between messages

Example program:

```
{True}
let (c, c') = new_chan () in
fork {let x = recv c' in send c' (x + 2)};
send c 40; recv c
{w. w = 42}
```

Dependent separation protocols:

```
c  ─→ ! (x : N) ⟨x⟩{True}. ? ⟨x + 2⟩{True}. end

c' ─→ ? (x : N) ⟨x⟩{True}. ! ⟨x + 2⟩{True}. end
```

Properties obtained:

- ✓ Type safety / session fidelity
- ✓ Functional correctness
Example – References

Example program:

```
{True}
  let (c, c') = new_chan () in
  fork {let x = recv c' in x ← (!x + 2); send c' ()};
  let y = ref (40) in send c y; recv c; !y
{w. w = 42}
```
Example – References

Example program:

```
{True}
   let (c, c') = new_chan () in
   fork {let x = recv c' in x ← (!x + 2); send c' ()};
   let y = ref (40) in send c y; recv c; !y

{w. w = 42}
```

Dependent separation protocols:

```
c  ↦! (ℓ : Loc)(x : N) ⟨ℓ⟩{ℓ ↦ n}. ? ⟨()⟩{ℓ ↦ (x + 2)}. end

c'  ↦? (ℓ : Loc)(x : N) ⟨ℓ⟩{ℓ ↦ n}. ! ⟨()⟩{ℓ ↦ (x + 2)}. end
```
Example – References

Example program:

\[
\begin{align*}
\{\text{True}\} \\
\text{let } (c, c') = \text{new\_chan}() \text{ in} \\
\text{fork } \{\text{let } x = \text{recv } c' \text{ in } x \leftarrow (!x + 2); \text{send } c'()\}; \\
\text{let } y = \text{ref} (40) \text{ in } \text{send } c \ y \text{; recv } c; !y
\end{align*}
\]

\{w. \ w = 42\}

Dependent separation protocols:

\[c \leadsto ! (\ell: \text{Loc})(x: \text{N}) \langle \ell \rangle \{\ell \mapsto n\}. ? \langle()\rangle \{\ell \mapsto (x + 2)\}. \text{end}\]

\[c' \leadsto ? (\ell: \text{Loc})(x: \text{N}) \langle \ell \rangle \{\ell \mapsto n\}. ! \langle()\rangle \{\ell \mapsto (x + 2)\}. \text{end}\]

Properties obtained:

✓ Type safety / session fidelity

✓ Functional correctness
Soundness of Actris

If \{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 \{\text{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
A powerful, general, language-independent, framework for modeling your own domain specific higher-order separation logics with powerful tactics in Coq
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
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
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 $c \leftrightarrow prot$ 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 $c \leftrightarrow prot$, 450 lines for Coq tactics specific for message passing)
Implementation and model of Actris in Iris

**Approach:**
- Implement `new_chan`, `send`, and `recv` as a library using lock-protected buffers
- Define $c \mapsto prot$ 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 $c \mapsto prot$, 450 lines for Coq tactics specific for message passing)
Demo in Coq