Sessions and Separation

Jonas Kastberg Hinrichsen, IT University of Copenhagen

Under supervision of
Jesper Bengtson, IT University of Copenhagen
Robbert Krebbers, Radboud University

11. June 2021
IT University of Copenhagen

Combining

Combining

Session Types

Combining
Session Types with *Separation* Logic

Combining

Session Types with *Separation* Logic

to ensure correctness

Combining

Session Types with *Separation* Logic

to ensure correctness

of concurrent programs

Combining

Session Types with *Separation* Logic

to ensure correctness

of concurrent programs

(that use message passing with other concurrency mechanisms)

Key observation:
Concurrency is important

Key observation: Concurrency is important

Concurrency: Working together to complete a shared task

Key observation: Concurrency is important

Concurrency: Working together to complete a shared task

- A way of increasing productivity

Key observation: Concurrency is important

Concurrency: Working together to complete a shared task

- A way of increasing productivity

Concurrency is everywhere

Key observation: Concurrency is important

Concurrency: Working together to complete a shared task

- A way of increasing productivity

Concurrency is everywhere

- Real world: Cooks in a kitchen

Key observation: Concurrency is important

Concurrency: Working together to complete a shared task

- A way of increasing productivity

Concurrency is everywhere

- Real world: Cooks in a kitchen
- Between computers: Server farms

Key observation: Concurrency is important

Concurrency: Working together to complete a shared task

- A way of increasing productivity

Concurrency is everywhere

- Real world: Cooks in a kitchen
- Between computers: Server farms
- Within computers: Multi-core processors

Key observation: Concurrency is important

Concurrency: Working together to complete a shared task

- A way of increasing productivity

Concurrency is everywhere

- Real world: Cooks in a kitchen
- Between computers: Server farms
- Within computers: Multi-core processors
  - A core is like a cook

Key observation: Concurrency is important

Concurrency: Working together to complete a shared task

- A way of increasing productivity

Concurrency is everywhere

- Real world: Cooks in a kitchen
- Between computers: Server farms
- Within computers: Multi-core processors
  - A core is like a cook

Concurrent programs: Instructions on how the cores should work together

Problem:

Concurrency is difficult

Problem: Concurrency is difficult

Coordinating a concurrent effort is notoriously difficult

Problem: Concurrency is difficult

Coordinating a concurrent effort is notoriously difficult

- Real world: “Too many cooks spoil the broth”

Problem: Concurrency is difficult

Coordinating a concurrent effort is notoriously difficult

- Real world: "Too many cooks spoil the broth"
- Between computers: Miscommunication

Problem: Concurrency is difficult

Coordinating a concurrent effort is notoriously difficult

- Real world: "Too many cooks spoil the broth"
- Between computers: Miscommunication
- Within computers: Data races

A problem has been detected and Windows has been shut down to prevent damage to your computer.

PFN_LIST_CORRUPT

If this is the first time you've seen this Stop error screen, restart your computer. If this screen appears again, follow these steps:

Check to make sure any new hardware or software is properly installed. If this is a new installation, ask your hardware or software manufacturer for any Windows updates you might need.

If problems continue, disable or remove any newly installed hardware or software. Disable BIOS memory options such as caching or shadowing. If you need to use Safe Mode to remove or disable components, restart your computer, press F8 to select Advanced Startup Options, and then select Safe Mode.

Technical information:

*** STOP: 0x0000004e (0x00000099, 0x00900009, 0x00000900, 0x00000900)

Your computer restarted because of a problem. Press a key or wait a few seconds to continue starting up.

Votre ordinateur a redémarré en raison d’un problème. Pour poursuivre le redémarrage, appuyez sur une touche ou patientez quelques secondes.

El ordenador se ha reiniciado debido a un problema. Para continuar con el arranque, pulse cualquier tecla o espere unos segundos.

Ihr Computer wurde aufgrund eines Problems neu gestartet. Drücken Sie zum Fortfahren eine Taste oder warten Sie einige Sekunden.

問題が起きたためコンピュータを再起動しました。このまま起動する場合は、いずれかのキーを押すか、数秒間そのままお待ちください。

电脑因出现问题而重新启动。请按一下按键，或等几秒钟以继续启动。

Goal:

Ensure Correctness

of Concurrent Programs

Goal:

Ensure Correctness

of Concurrent Programs

(*i.e.*, That they do not crash, and produce the expected results)

Goal: Ensure Correctness of Concurrent Programs

Program testing

Goal: Ensure Correctness of Concurrent Programs

Program testing

- Running the program with various input, and checking the output

Goal: Ensure Correctness of Concurrent Programs

Program testing

- Running the program with various input, and checking the output
- Problem: Hard to guarantee full code coverage

Goal: Ensure Correctness of Concurrent Programs

Program testing

- Running the program with various input, and checking the output
- Problem: Hard to guarantee full code coverage
  - Especially for concurrent programs

Goal: Ensure Correctness of Concurrent Programs

Program testing

- Running the program with various input, and checking the output
- Problem: Hard to guarantee full code coverage
  - Especially for concurrent programs: Execution order can change

Goal: Ensure Correctness of Concurrent Programs

Program testing

- Running the program with various input, and checking the output
- Problem: Hard to guarantee full code coverage
  - Especially for concurrent programs: Execution order can change
    - One chef adds soy sauce, then another salts to taste ☑

Goal: Ensure Correctness of Concurrent Programs

Program testing

- Running the program with various input, and checking the output
- Problem: Hard to guarantee full code coverage
  - Especially for concurrent programs: Execution order can change
    - One chef adds soy sauce, then another salts to taste ☑
    - One chef salts to taste, then another adds soy sauce ☒

Goal: Ensure Correctness of Concurrent Programs

Program testing

- Running the program with various input, and checking the output
- Problem: Hard to guarantee full code coverage
  - Especially for concurrent programs: Execution order can change
    - One chef adds soy sauce, then another salts to taste ☑
    - One chef salts to taste, then another adds soy sauce ☒

Formal verification

Goal: Ensure Correctness of Concurrent Programs

Program testing

- Running the program with various input, and checking the output
- Problem: Hard to guarantee full code coverage
  - Especially for concurrent programs: Execution order can change
    - One chef adds soy sauce, then another salts to taste ☑
    - One chef salts to taste, then another adds soy sauce ☒

Formal verification

- *Prove* that *any* execution of the program is correct

Goal: Ensure Correctness of Concurrent Programs

Program testing

- Running the program with various input, and checking the output
- Problem: Hard to guarantee full code coverage
  - Especially for concurrent programs: Execution order can change
    - One chef adds soy sauce, then another salts to taste ☑
    - One chef salts to taste, then another adds soy sauce ☒

Formal verification

- *Prove* that *any* execution of the program is correct
  - Guarantees full code coverage

Goal: Ensure Correctness of Concurrent Programs

Program testing

- Running the program with various input, and checking the output
- Problem: Hard to guarantee full code coverage
  - Especially for concurrent programs: Execution order can change
    - One chef adds soy sauce, then another salts to taste ☑
    - One chef salts to taste, then another adds soy sauce ☒

Formal verification

- *Prove* that *any* execution of the program is correct
  - Guarantees full code coverage
    - Also for concurrent programs

Goal: Ensure Correctness of Concurrent Programs

Program testing

- Running the program with various input, and checking the output
- Problem: Hard to guarantee full code coverage
  - Especially for concurrent programs: Execution order can change
    - One chef adds soy sauce, then another salts to taste ☑
    - One chef salts to taste, then another adds soy sauce ☒

Formal verification

- *Prove* that *any* execution of the program is correct
  - Guarantees full code coverage
    - Also for concurrent programs
- Statically: Without running the program

Math!

Math!

(Board Games!)

Define a mathematical model

Define a mathematical model (*e.g.*, separation logic)

Define a mathematical model (*e.g.*, separation logic)

- Like designing a board game!

Formal Verification

Define a mathematical model (*e.g.*, separation logic)

- Like designing a board game!

Specify programs and expected results

Define a mathematical model (*e.g.*, separation logic)

- Like designing a board game!

Specify programs and expected results (*e.g.*, $\{\mathsf{True}\}\,\texttt{sort}\ \vec{v}\,\{\vec{w}.\,\texttt{sorted\_of}\ \vec{w}\ \vec{v}\}$)

Define a mathematical model (*e.g.*, separation logic)

- Like designing a board game!

Specify programs and expected results (*e.g.*, $\{\text{True}\}\ \texttt{sort}\ \vec{v}\ \{\vec{w}.\ \texttt{sorted\_of}\ \vec{w}\ \vec{v}\}$)

- Like a scenario in the board game!

Formal Verification

Define a mathematical model (*e.g.*, separation logic)

- Like designing a board game!

Specify programs and expected results (*e.g.*, $\{\mathsf{True}\}\,\mathtt{sort}\ \vec{v}\,\{\vec{w}.\,\mathtt{sorted\_of}\ \vec{w}\ \vec{v}\}$)

- Like a scenario in the board game!

Carry out derivations

Formal Verification

Define a mathematical model (*e.g.*, separation logic)

- Like designing a board game!

Specify programs and expected results (*e.g.*, $\{\text{True}\}\ \texttt{sort}\ \vec{v}\ \{\vec{w}.\ \texttt{sorted\_of}\ \vec{w}\ \vec{v}\}$)

- Like a scenario in the board game!

Carry out derivations

- Playing the board game, one rule at a time

Formal Verification

Define a mathematical model (*e.g.*, separation logic)

- Like designing a board game!

Specify programs and expected results (*e.g.*, $\{\text{True}\}\,\texttt{sort}\ \vec{v}\,\{\vec{w}.\,\texttt{sorted\_of}\ \vec{w}\ \vec{v}\}$)

- Like a scenario in the board game!

Carry out derivations

- Playing the board game, one rule at a time

Adequacy

Formal Verification

Define a mathematical model (*e.g.*, separation logic)

- Like designing a board game!

Specify programs and expected results (*e.g.*, $\{\text{True}\}\,\texttt{sort}\ \vec{v}\,\{\vec{w}.\,\texttt{sorted\_of}\ \vec{w}\ \vec{v}\}$)

- Like a scenario in the board game!

Carry out derivations

- Playing the board game, one rule at a time

Adequacy (*e.g.*, $\texttt{if}\ \{\text{True}\}\,e\,\{v.\,\varphi\ v\}\ \texttt{then}\ \text{correct}\ (e, \varphi)$)

Formal Verification

Define a mathematical model (*e.g.*, separation logic)

- Like designing a board game!

Specify programs and expected results (*e.g.*, $\{\mathsf{True}\}\,\texttt{sort}\ \vec{v}\,\{\vec{w}.\,\texttt{sorted\_of}\ \vec{w}\ \vec{v}\}$)

- Like a scenario in the board game!

Carry out derivations

- Playing the board game, one rule at a time

Adequacy (*e.g.*, $\texttt{if}\ \{\mathsf{True}\}\,e\,\{v.\,\varphi\ v\}\ \texttt{then}\ \mathsf{correct}\ (e, \varphi)$)

- Winning the board game ensures certain properties (such as correctness)

Formal Verification

Define a mathematical model (*e.g.*, separation logic)

- Like designing a board game!

Specify programs and expected results (*e.g.*, $\{\mathrm{True}\}\,\texttt{sort}\ \vec{v}\,\{\vec{w}.\,\texttt{sorted\_of}\ \vec{w}\ \vec{v}\}$)

- Like a scenario in the board game!

Carry out derivations

- Playing the board game, one rule at a time

Adequacy (*e.g.*, $\texttt{if}\ \{\mathrm{True}\}\,e\,\{v.\,\varphi\ v\}\ \texttt{then}\ \mathrm{correct}\ (e, \varphi)$)

- Winning the board game ensures certain properties (such as correctness)

Just create and play a board game!

Formal Verification

Define a mathematical model (*e.g.*, separation logic)

- Like designing a board game!

Specify programs and expected results (*e.g.*, $\{\text{True}\}\, \texttt{sort}\ \vec{v}\, \{\vec{w}.\, \texttt{sorted\_of}\ \vec{w}\ \vec{v}\}$)

- Like a scenario in the board game!

Carry out derivations

- Playing the board game, one rule at a time

Adequacy (*e.g.*, $\texttt{if}\ \{\text{True}\}\, e\, \{v.\, \varphi\ v\}\ \texttt{then}\ \text{correct}\ (e, \varphi)$)

- Winning the board game ensures certain properties (such as correctness)

Just create and play a board game!

- That ensures correctness of concurrent programs

Goal:
Board game

Goal:

Board game

that ensures correctness

Goal:

Board game

that ensures correctness

of concurrent programs

Goal: Board game that ensures correctness of concurrent programs

First: Settle on the programming language to verify

Goal: Board game that ensures correctness of concurrent programs

First: Settle on the programming language to verify (Syntax and semantics)

Goal: Board game that ensures correctness of concurrent programs

First: Settle on the programming language to verify (Syntax and semantics)

- Like the theme of the board game

Goal: Board game that ensures correctness of concurrent programs

First: Settle on the programming language to verify (Syntax and semantics)

- Like the theme of the board game
- Settle on concurrency mechanisms

Goal: Board game that ensures correctness of concurrent programs

First: Settle on the programming language to verify (Syntax and semantics)

- Like the theme of the board game
- Settle on concurrency mechanisms: Tools to describe collaboration

Goal: Board game that ensures correctness of concurrent programs

First: Settle on the programming language to verify (Syntax and semantics)

- Like the theme of the board game
- Settle on concurrency mechanisms: Tools to describe collaboration
  - Shared memory

Goal: Board game that ensures correctness of concurrent programs

First: Settle on the programming language to verify (Syntax and semantics)

- Like the theme of the board game
- Settle on concurrency mechanisms: Tools to describe collaboration
  - Shared memory
    - Cooks collaborate on a shared dish

Goal: Board game that ensures correctness of concurrent programs

First: Settle on the programming language to verify (Syntax and semantics)

- Like the theme of the board game
- Settle on concurrency mechanisms: Tools to describe collaboration
  - Shared memory
    - Cooks collaborate on a shared dish
    - Cooks take turns adding to the shared dish

Goal: Board game that ensures correctness of concurrent programs

First: Settle on the programming language to verify (Syntax and semantics)

- Like the theme of the board game
- Settle on concurrency mechanisms: Tools to describe collaboration
  - Shared memory
    - Cooks collaborate on a shared dish
    - Cooks take turns adding to the shared dish
  - Message passing

Goal: Board game that ensures correctness of concurrent programs

First: Settle on the programming language to verify (Syntax and semantics)

- Like the theme of the board game
- Settle on concurrency mechanisms: Tools to describe collaboration
  - Shared memory
    - Cooks collaborate on a shared dish
    - Cooks take turns adding to the shared dish
  - Message passing
    - Cooks work separately on different parts of the dish

Goal: Board game that ensures correctness of concurrent programs

First: Settle on the programming language to verify (Syntax and semantics)

- Like the theme of the board game
- Settle on concurrency mechanisms: Tools to describe collaboration
  - Shared memory
    - Cooks collaborate on a shared dish
    - Cooks take turns adding to the shared dish
  - Message passing
    - Cooks work separately on different parts of the dish
    - Cooks send finished parts to head chef

Goal: Board game that ensures correctness of concurrent programs

First: Settle on the programming language to verify (Syntax and semantics)

- Like the theme of the board game
- Settle on concurrency mechanisms: Tools to describe collaboration
  - Shared memory
    - Cooks collaborate on a shared dish
    - Cooks take turns adding to the shared dish
  - Message passing
    - Cooks work separately on different parts of the dish
    - Cooks send finished parts to head chef
    - Head chef finishes the dish

Goal: Board game that ensures correctness of concurrent programs

First: Settle on the programming language to verify (Syntax and semantics)

- Like the theme of the board game
- Settle on concurrency mechanisms: Tools to describe collaboration
  - Shared memory
    - Cooks collaborate on a shared dish
    - Cooks take turns adding to the shared dish
  - Message passing
    - Cooks work separately on different parts of the dish
    - Cooks send finished parts to head chef
    - Head chef finishes the dish
- It is common to combine multiple concurrency mechanisms

Goal: Board game that ensures correctness of concurrent programs

First: Settle on the programming language to verify (Syntax and semantics)

- Like the theme of the board game
- Settle on concurrency mechanisms: Tools to describe collaboration
  - Shared memory
    - Cooks collaborate on a shared dish
    - Cooks take turns adding to the shared dish
  - Message passing
    - Cooks work separately on different parts of the dish
    - Cooks send finished parts to head chef
    - Head chef finishes the dish
- It is common to combine multiple concurrency mechanisms

Then: Settle on the properties to guarantee

Goal: Board game that ensures correctness of concurrent programs

First: Settle on the programming language to verify (Syntax and semantics)

- Like the theme of the board game
- Settle on concurrency mechanisms: Tools to describe collaboration
  - Shared memory
    - Cooks collaborate on a shared dish
    - Cooks take turns adding to the shared dish
  - Message passing
    - Cooks work separately on different parts of the dish
    - Cooks send finished parts to head chef
    - Head chef finishes the dish
- It is common to combine multiple concurrency mechanisms

Then: Settle on the properties to guarantee

- Crash-freedom (safety)

Goal: Board game that ensures correctness of concurrent programs

First: Settle on the programming language to verify (Syntax and semantics)

- Like the theme of the board game
- Settle on concurrency mechanisms: Tools to describe collaboration
  - Shared memory
    - Cooks collaborate on a shared dish
    - Cooks take turns adding to the shared dish
  - Message passing
    - Cooks work separately on different parts of the dish
    - Cooks send finished parts to head chef
    - Head chef finishes the dish
- It is common to combine multiple concurrency mechanisms

Then: Settle on the properties to guarantee

- Crash-freedom (safety)
- Terminating programs produce the expected results (functional correctness)

Goal: Board game that ensures correctness of concurrent programs

First: Settle on the programming language to verify (Syntax and semantics)

- Like the theme of the board game
- Settle on concurrency mechanisms: Tools to describe collaboration
  - Shared memory
    - Cooks collaborate on a shared dish
    - Cooks take turns adding to the shared dish
  - Message passing
    - Cooks work separately on different parts of the dish
    - Cooks send finished parts to head chef
    - Head chef finishes the dish
- It is common to combine multiple concurrency mechanisms

Then: Settle on the properties to guarantee

- Crash-freedom (safety)
- Terminating programs produce the expected results (functional correctness)

Finally: Settle on the rules, and prove adequacy

Two existing solutions:

Session Types and Separation Logic

Existing solution: Separation Logic

Mathematical model for analysing programs with shared memory

Existing solution: Separation Logic

Mathematical model for analysing programs with shared memory

- Actively being researched since year 2000

Existing solution: Separation Logic

Mathematical model for analysing programs with shared memory

- Actively being researched since year 2000
- Pioneered by Peter O'hearn and John C. Reynolds

Existing solution: Separation Logic

Mathematical model for analysing programs with shared memory

- Actively being researched since year 2000
- Pioneered by Peter O'hearn and John C. Reynolds

Guarantees crash-freedom and functional correctness

Existing solution: Separation Logic

Mathematical model for analysing programs with shared memory

- Actively being researched since year 2000
- Pioneered by Peter O'hearn and John C. Reynolds

Guarantees crash-freedom and functional correctness

Complicated board game

Existing solution: Separation Logic

Mathematical model for analysing programs with shared memory

- Actively being researched since year 2000
- Pioneered by Peter O'hearn and John C. Reynolds

Guarantees crash-freedom and functional correctness

Complicated board game

- Not automatically solvable by a computer

Existing solution: Separation Logic

Mathematical model for analysing programs with shared memory

- Actively being researched since year 2000
- Pioneered by Peter O'hearn and John C. Reynolds

Guarantees crash-freedom and functional correctness

Complicated board game

- Not automatically solvable by a computer
- Playing and winning requires interactive help

Existing solution: Separation Logic

Mathematical model for analysing programs with shared memory

- Actively being researched since year 2000
- Pioneered by Peter O'hearn and John C. Reynolds

Guarantees crash-freedom and functional correctness

Complicated board game

- Not automatically solvable by a computer
- Playing and winning requires interactive help
- Important to have simple rules

Existing solution: Separation Logic

Mathematical model for analysing programs with shared memory

- Actively being researched since year 2000
- Pioneered by Peter O'hearn and John C. Reynolds

Guarantees crash-freedom and functional correctness

Complicated board game

- Not automatically solvable by a computer
- Playing and winning requires interactive help
- Important to have simple rules (like in chess)

Existing solution: Separation Logic

Mathematical model for analysing programs with shared memory

- Actively being researched since year 2000
- Pioneered by Peter O'hearn and John C. Reynolds

Guarantees crash-freedom and functional correctness

Complicated board game

- Not automatically solvable by a computer
- Playing and winning requires interactive help
- Important to have simple rules (like in chess)

The Iris separation logic Iris

Iris logo: https://iris-project.org/

Existing solution: Separation Logic

Mathematical model for analysing programs with shared memory

- Actively being researched since year 2000
- Pioneered by Peter O'hearn and John C. Reynolds

Guarantees crash-freedom and functional correctness

Complicated board game

- Not automatically solvable by a computer
- Playing and winning requires interactive help
- Important to have simple rules (like in chess)

The Iris separation logic Iris

- Simple rules for shared memory, and other concurrency mechanisms

Iris logo: https://iris-project.org/

Existing solution: Separation Logic

Mathematical model for analysing programs with shared memory

- Actively being researched since year 2000
- Pioneered by Peter O'hearn and John C. Reynolds

Guarantees crash-freedom and functional correctness

Complicated board game

- Not automatically solvable by a computer
- Playing and winning requires interactive help
- Important to have simple rules (like in chess)

The Iris separation logic

- Simple rules for shared memory, and other concurrency mechanisms
- Problem: Lack of simple rules for message passing

Iris logo: https://iris-project.org/

Existing solution: Session types

Mathematical model for analysing message-passing programs

Existing solution: Session types

Mathematical model for analysing message-passing programs

- Actively being researched since the 90s

Existing solution: Session types

Mathematical model for analysing message-passing programs

- Actively being researched since the 90s
- Pioneered by Kohei Honda

Existing solution: Session types

Mathematical model for analysing message-passing programs

- Actively being researched since the 90s
- Pioneered by Kohei Honda

Guarantees that programs are crash-free (and deadlock-free)

Existing solution: Session types

Mathematical model for analysing message-passing programs

- Actively being researched since the 90s
- Pioneered by Kohei Honda

Guarantees that programs are crash-free (and deadlock-free)

- Problem: Does not generally guarantee functional correctness

Existing solution: Session types

Mathematical model for analysing message-passing programs

- Actively being researched since the 90s
- Pioneered by Kohei Honda

Guarantees that programs are crash-free (and deadlock-free)

- Problem: Does not generally guarantee functional correctness

Less complicated board game

Existing solution: Session types

Mathematical model for analysing message-passing programs

- Actively being researched since the 90s
- Pioneered by Kohei Honda

Guarantees that programs are crash-free (and deadlock-free)

- Problem: Does not generally guarantee functional correctness

Less complicated board game

- Automatically solvable by a computer

Existing solution: Session types

Mathematical model for analysing message-passing programs

- Actively being researched since the 90s
- Pioneered by Kohei Honda

Guarantees that programs are crash-free (and deadlock-free)

- Problem: Does not generally guarantee functional correctness

Less complicated board game

- Automatically solvable by a computer
- Intuitive rules for message passing

Existing solution: Session types

Mathematical model for analysing message-passing programs

- Actively being researched since the 90s
- Pioneered by Kohei Honda

Guarantees that programs are crash-free (and deadlock-free)

- Problem: Does not generally guarantee functional correctness

Less complicated board game

- Automatically solvable by a computer
- Intuitive rules for message passing

Many variants of session types exists

Existing solution: Session types

Mathematical model for analysing message-passing programs

- Actively being researched since the 90s
- Pioneered by Kohei Honda

Guarantees that programs are crash-free (and deadlock-free)

- Problem: Does not generally guarantee functional correctness

Less complicated board game

- Automatically solvable by a computer
- Intuitive rules for message passing

Many variants of session types exists

- We consider: Binary session types

Existing solution: Session types

Mathematical model for analysing message-passing programs

- Actively being researched since the 90s
- Pioneered by Kohei Honda

Guarantees that programs are crash-free (and deadlock-free)

- Problem: Does not generally guarantee functional correctness

Less complicated board game

- Automatically solvable by a computer
- Intuitive rules for message passing

Many variants of session types exists

- We consider: Binary session types
  - Binary: Communication is between two parties

Key idea:

Combine

(Binary) Session Types and

(the Iris) Separation Logic

Key idea:

Combine

(Binary) Session Types and

(the Iris) Separation Logic

to ensure correctness

Key idea:

Combine

(Binary) Session Types and

(the Iris) Separation Logic

to ensure correctness

of concurrent programs

Key idea:

Combine

(Binary) Session Types and

(the Iris) Separation Logic

to ensure correctness

of concurrent programs

(that use message passing with other concurrency mechanisms)

Contribution 1:
Actris

Contribution 1:

Actris: A separation logic

Contribution 1:

Actris: A separation logic with a session type-based mechanism

Contribution 1:

Actris: A separation logic with a session type-based mechanism for ensuring correctness

Contribution 1 of my Ph.D. thesis

Contribution 1:

Actris: A separation logic with a session type-based mechanism for ensuring correctness of concurrent programs

Contribution 1:

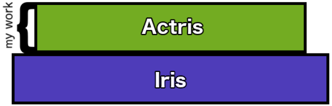
Actris: A separation logic with a session type-based mechanism for ensuring correctness of concurrent programs that combine binary message passing with other concurrency mechanisms

Contribution 1 of my Ph.D. thesis

Contribution 1:

Actris: A separation logic with a session type-based mechanism for ensuring correctness of concurrent programs that combine binary message passing with other concurrency mechanisms

- Built on top of Iris

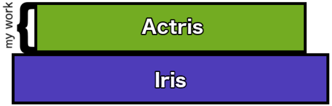


Contribution 1:

Actris: A separation logic with a session type-based mechanism for ensuring correctness of concurrent programs that combine binary message passing with other concurrency mechanisms

- Built on top of Iris

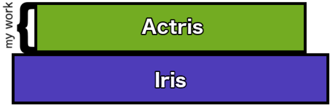
	Session Types	**Iris**	**Actris**
Shared memory	✗	✓	✓



Contribution 1:

Actris: A separation logic with a session type-based mechanism for ensuring correctness of concurrent programs that combine binary message passing with other concurrency mechanisms

- Built on top of Iris

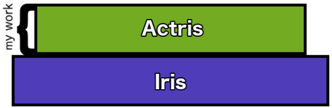


	Session Types	**Iris**	**Actris**
Shared memory	✗	✓	✓
Other concurrency	✗	✓	✓

Contribution 1:

Actris: A separation logic with a session type-based mechanism for ensuring correctness of concurrent programs that combine binary message passing with other concurrency mechanisms

- Built on top of Iris

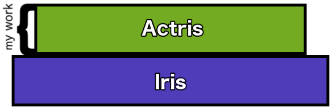


	Session Types	**Iris**	**Actris**
Shared memory	✗	✓	✓
Other concurrency	✗	✓	✓
Crash-freedom	✓	✓	✓

Contribution 1:

Actris: A separation logic with a session type-based mechanism for ensuring correctness of concurrent programs that combine binary message passing with other concurrency mechanisms

- Built on top of Iris

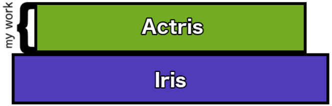


	Session Types	**Iris**	**Actris**
Shared memory	✗	✓	✓
Other concurrency	✗	✓	✓
Crash-freedom	✓	✓	✓
Functional correctness	✗	✓	✓

Contribution 1:

Actris: A separation logic with a session type-based mechanism for ensuring correctness of concurrent programs that combine binary message passing with other concurrency mechanisms

- Built on top of Iris

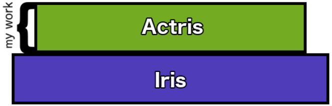


	Session Types	**Iris**	**Actris**
Shared memory	✗	✓	✓
Other concurrency	✗	✓	✓
Crash-freedom	✓	✓	✓
Functional correctness	✗	✓	✓
Message passing	✓	✗	✓

Contribution 1:

Actris: A separation logic with a session type-based mechanism for ensuring correctness of concurrent programs that combine binary message passing with other concurrency mechanisms

- Built on top of Iris

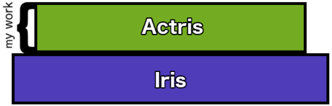


	Session Types	**Iris**	**Actris**
Shared memory	✗	✓	✓
Other concurrency	✗	✓	✓
Crash-freedom	✓	✓	✓
Functional correctness	✗	✓	✓
Message passing	✓	✗	✓
Deadlock-freedom	✓	✗	✗

Contribution 1:

Actris: A separation logic with a session type-based mechanism for ensuring correctness of concurrent programs that combine binary message passing with other concurrency mechanisms

- Built on top of Iris



	Session Types	Iris	Actris
Shared memory	✗	✓	✓
Other concurrency	✗	✓	✓
Crash-freedom	✓	✓	✓
Functional correctness	✗	✓	✓
Message passing	✓	✗	✓
Deadlock-freedom	✓	✗	✗
Automatically solvable	✓	✗	✗

Problem:

Bugs and cheating in the board game

Problem: Bugs and cheating in the board game

Bugs and Cheating

Problem: Bugs and cheating in the board game

Bugs and Cheating

- Bugs: Contradictory rules

Problem: Bugs and cheating in the board game

Bugs and Cheating

- Bugs: Contradictory rules
  - Like drawing infinite cards

Problem: Bugs and cheating in the board game

Bugs and Cheating

- Bugs: Contradictory rules
  - Like drawing infinite cards (or obtaining a paradox)

Problem: Bugs and cheating in the board game

Bugs and Cheating

- Bugs: Contradictory rules
  - Like drawing infinite cards (or obtaining a paradox)
- Cheating: Not following the rules of the board game

Problem: Bugs and cheating in the board game

Bugs and Cheating

- Bugs: Contradictory rules
  - Like drawing infinite cards (or obtaining a paradox)
- Cheating: Not following the rules of the board game

Bugs or cheating = All bets are off

Problem: Bugs and cheating in the board game

Bugs and Cheating

- Bugs: Contradictory rules
    - Like drawing infinite cards (or obtaining a paradox)
- Cheating: Not following the rules of the board game

Bugs or cheating = All bets are off

- No guaranteed properties from winning

Problem: Bugs and cheating in the board game

Bugs and Cheating

- Bugs: Contradictory rules
  - Like drawing infinite cards (or obtaining a paradox)
- Cheating: Not following the rules of the board game

Bugs or cheating = All bets are off

- No guaranteed properties from winning

These are complicated board games

Problem: Bugs and cheating in the board game

Bugs and Cheating

- Bugs: Contradictory rules
  - Like drawing infinite cards (or obtaining a paradox)
- Cheating: Not following the rules of the board game

Bugs or cheating = All bets are off

- No guaranteed properties from winning

These are complicated board games

- Difficult to avoid bugs

Problem: Bugs and cheating in the board game

Bugs and Cheating

- Bugs: Contradictory rules
  - Like drawing infinite cards (or obtaining a paradox)
- Cheating: Not following the rules of the board game

Bugs or cheating = All bets are off

- No guaranteed properties from winning

These are complicated board games

- Difficult to avoid bugs
- Cheating can happen by accident

Solution:

Mechanisation!

Turning the board game

Solution: Mechanisation!

Turning the board game into a video game!

Solution: Mechanisation!

Turning the board game into a video game!

- More restrictive design environment

Solution: Mechanisation!

Turning the board game into a video game!

- More restrictive design environment = Less chance of contradictory rules

Solution: Mechanisation!

Turning the board game into a video game!

- More restrictive design environment $=$ Less chance of contradictory rules
  - Interactive theorem prover

Solution: Mechanisation!

Turning the board game into a video game!

- More restrictive design environment = Less chance of contradictory rules
  - Interactive theorem prover (Coq)

Coq image: https://ilyasergey.net/pnp/

Solution: Mechanisation!

Turning the board game into a video game!

- More restrictive design environment = Less chance of contradictory rules
  - Interactive theorem prover (Coq)
    - Like a very strict game engine

Coq image: https://ilyasergey.net/pnp/

Solution: Mechanisation!

Turning the board game into a video game!

- More restrictive design environment = Less chance of contradictory rules
  - Interactive theorem prover (Coq)
    - Like a very strict game engine
- Strict referee

Coq image: https://ilyasergey.net/pnp/

Solution: Mechanisation!

Turning the board game into a video game!

- More restrictive design environment $=$ Less chance of contradictory rules
  - Interactive theorem prover (Coq)
    - Like a very strict game engine
- Strict referee
  - No accidental cheating

Coq image: https://ilyasergey.net/pnp/

Solution: Mechanisation!

Turning the board game into a video game!

- More restrictive design environment = Less chance of contradictory rules
  - Interactive theorem prover (Coq)
    - Like a very strict game engine
- Strict referee
  - No accidental cheating

Mechanisation takes time

Coq image: https://ilyasergey.net/pnp/

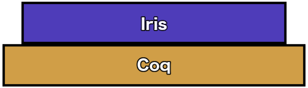
Solution: Mechanisation!

Turning the board game into a video game!

- More restrictive design environment = Less chance of contradictory rules
  - Interactive theorem prover (Coq)
    - Like a very strict game engine
- Strict referee
  - No accidental cheating

Mechanisation takes time

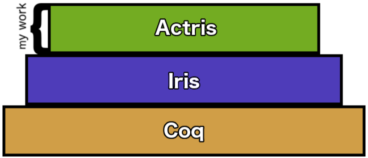
- Iris has already been fully mechanised in Coq



Coq image: https://ilyasergey.net/pnp/

Contribution 2:

Full mechanisation of Actris on top of Iris in Coq

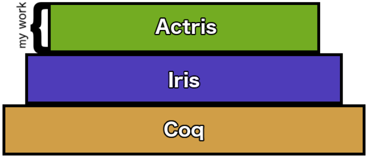


Contribution 2 of my Ph.D. thesis

Contribution 2:

Full mechanisation of Actris on top of Iris in Coq

- With verified program examples (*e.g.*, a variant of the map-reduce algorithm)



Observation:

Ongoing effort on mechanising Session Types

Problem:

No mechanisation of session type systems that combine advanced features

Problem:

No mechanisation of session type systems that combine advanced features

(That we know of)

Solution:
Semantic Typing

Solution:

Semantic Typing

(Board game inception)

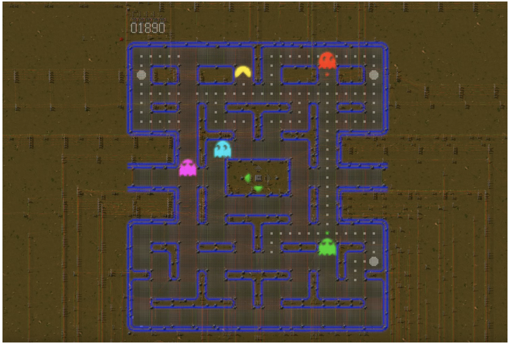
Solution: Semantic Typing (Board game inception)

Playing a board game inside another board game

Solution: Semantic Typing (Board game inception)

Playing a board game inside another board game

- Like Pacman in Factorio



Pacman image: https://www.youtube.com/watch?v=_VR_b9YwqH8

Solution: Semantic Typing (Board game inception)

Playing a board game inside another board game

- Like Pacman in Factorio

Defining a session type system within Actris

Solution: Semantic Typing (Board game inception)

Playing a board game inside another board game

- Like Pacman in Factorio

Defining a session type system within Actris

- Using the session-type based mechanism to model session types

Solution: Semantic Typing (Board game inception)

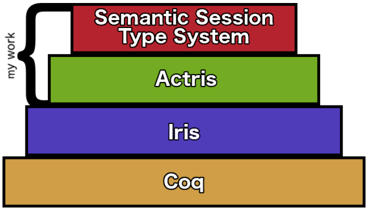
Playing a board game inside another board game

- Like Pacman in Factorio

Defining a session type system within Actris

- Using the session-type based mechanism to model session types

Inherit the properties of Actris



Solution: Semantic Typing (Board game inception)

Playing a board game inside another board game

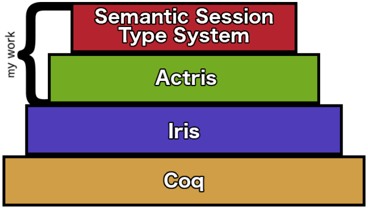
- Like Pacman in Factorio

Defining a session type system within Actris

- Using the session-type based mechanism to model session types

Inherit the properties of Actris

- The mechanisation of Actris



Solution: Semantic Typing (Board game inception)

Playing a board game inside another board game

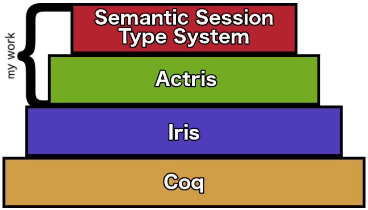
- Like Pacman in Factorio

Defining a session type system within Actris

- Using the session-type based mechanism to model session types

Inherit the properties of Actris

- The mechanisation of Actris
- The session type-based features of Actris



Solution: Semantic Typing (Board game inception)

Playing a board game inside another board game

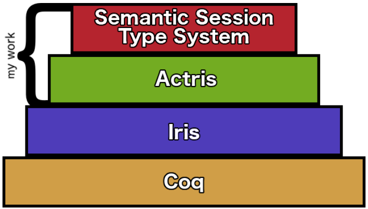
- Like Pacman in Factorio

Defining a session type system within Actris

- Using the session-type based mechanism to model session types

Inherit the properties of Actris

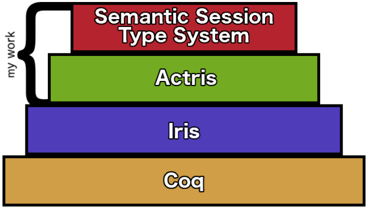
- The mechanisation of Actris
- The session type-based features of Actris
- The other concurrency mechanisms of Iris



Contribution 3 of my PhD thesis

Contribution 3:

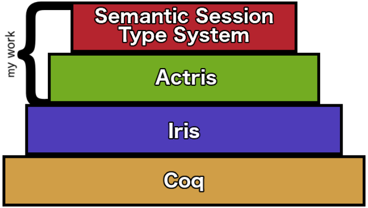
Defining and mechanising a Semantic Session Type System on top of Actris on top of Iris in Coq



Contribution 3:

Defining and mechanising a Semantic Session Type System on top of Actris on top of Iris in Coq

- With verified program examples (*e.g.*, a message-passing-based producer-consumer)



Contributions of my PhD thesis

Contribution 1:

Actris: A separation logic with a session type-based mechanism for ensuring correctness of concurrent programs that combine binary message passing with other concurrency mechanisms

- Built on top of Iris

Contribution 2:

Full mechanisation of Actris in Coq

- With verified program examples (*e.g.*, a variant of the map-reduce algorithm)

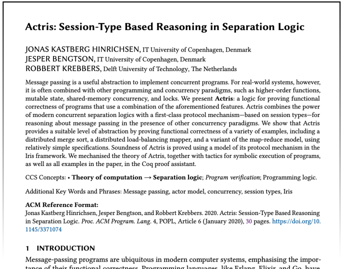
Contribution 3:

Defining and mechanising a Semantic Session Type System on top of Actris

- With verified program examples (*e.g.*, a message-passing-based producer-consumer)

Actris: Session-Type Based Reasoning in Separation Logic

- ACM SIGPLAN Symposium on Principles of Programming Languages 2020 [POPL'20]



Actris: Session-Type Based Reasoning in Separation Logic

JONAS KASTBERG HINRICHSEN, IT University of Copenhagen, Denmark
JESPER BENGTSON, IT University of Copenhagen, Denmark
ROBBERT KREBBERS, Delft University of Technology, The Netherlands

Message passing is a useful abstraction to implement concurrent programs. For real-world systems, however, it is often combined with other programming and concurrency paradigms, such as higher-order functions, mutable state, shared-memory concurrency, and locks. We present **Actris**: a logic for proving functional correctness of programs that use a combination of the aforementioned features. Actris combines the power of modern concurrent separation logics with a first-class protocol mechanism—based on session types—for reasoning about message passing in the presence of other concurrency paradigms. We show that Actris provides a suitable level of abstraction by proving functional correctness of a variety of examples, including a distributed merge sort, a distributed load-balancing mapper, and a variant of the map-reduce model, using relatively simple specifications. Soundness of Actris is proved using a model of its protocol mechanism in the Iris framework. We mechanised the theory of Actris, together with tactics for symbolic execution of programs, as well as all examples in the paper, in the Coq proof assistant.

CCS Concepts: • **Theory of computation → Separation logic**; *Program verification*; Programming logic.

Additional Key Words and Phrases: Message passing, actor model, concurrency, session types, Iris

ACM Reference Format:
Jonas Kastberg Hinrichsen, Jesper Bengtson, and Robbert Krebbers. 2020. Actris: Session-Type Based Reasoning in Separation Logic. *Proc. ACM Program. Lang.* 4, POPL, Article 6 (January 2020), 30 pages. https://doi.org/10.1145/3371074

1 INTRODUCTION

Message-passing programs are ubiquitous in modern computer systems, emphasising the impor-

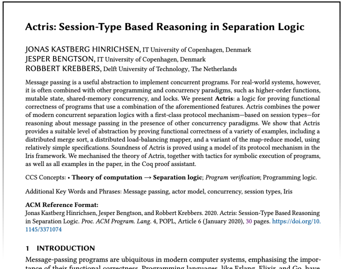
Publications

Actris: Session-Type Based Reasoning in Separation Logic

- ACM SIGPLAN Symposium on Principles of Programming Languages 2020 [POPL'20]

Actris 2.0: Asynchronous Session-Type Based Reasoning in Separation Logic

- Journal of Logical Methods in Computer Science [LMCS] (Conditionally accepted)



Actris: Session-Type Based Reasoning in Separation Logic

JONAS KASTBERG HINRICHSEN, IT University of Copenhagen, Denmark
JESPER BENGTSON, IT University of Copenhagen, Denmark
ROBBERT KREBBERS, Delft University of Technology, The Netherlands

Message passing is a useful abstraction to implement concurrent programs. For real-world systems, however, it is often combined with other programming and concurrency paradigms, such as higher-order functions, mutable state, shared-memory concurrency, and locks. We present **Actris**: a logic for proving functional correctness of programs that use a combination of the aforementioned features. Actris combines the power of modern concurrent separation logics with a first-class protocol mechanism—based on session types—for reasoning about message passing in the presence of other concurrency paradigms. We show that Actris provides a suitable level of abstraction by proving functional correctness of a variety of examples, including a distributed merge sort, a distributed load-balancing mapper, and a variant of the map-reduce model, using relatively simple specifications. Soundness of Actris is proved using a model of its protocol mechanism in the Iris framework. We mechanised the theory of Actris, together with tactics for symbolic execution of programs, as well as all examples in the paper, in the Coq proof assistant.

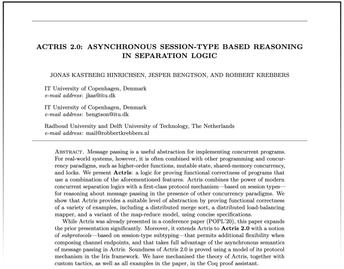
CCS Concepts: • **Theory of computation** → **Separation logic**; *Program verification*; Programming logic.

Additional Key Words and Phrases: Message passing, actor model, concurrency, session types, Iris

ACM Reference Format:
Jonas Kastberg Hinrichsen, Jesper Bengtson, and Robbert Krebbers. 2020. Actris: Session-Type Based Reasoning in Separation Logic. *Proc. ACM Program. Lang.* 4, POPL, Article 6 (January 2020), 30 pages. https://doi.org/10.1145/3371074

1 INTRODUCTION

Message-passing programs are ubiquitous in modern computer systems, emphasising the impor-



ACTRIS 2.0: ASYNCHRONOUS SESSION-TYPE BASED REASONING IN SEPARATION LOGIC

JONAS KASTBERG HINRICHSEN, JESPER BENGTSON, AND ROBBERT KREBBERS

IT University of Copenhagen, Denmark
e-mail address: jkas@itu.dk

IT University of Copenhagen, Denmark
e-mail address: bengtson@itu.dk

Radboud University and Delft University of Technology, The Netherlands
e-mail address: mail@robbertkrebbers.nl

Abstract. Message passing is a useful abstraction for implementing concurrent programs. For real-world systems, however, it is often combined with other programming and concurrency paradigms, such as higher-order functions, mutable state, shared-memory concurrency, and locks. We present **Actris**: a logic for proving functional correctness of programs that use a combination of the aforementioned features. Actris combines the power of modern concurrent separation logics with a first-class protocol mechanism—based on session types—for reasoning about message passing in the presence of other concurrency paradigms. We show that Actris provides a suitable level of abstraction by proving functional correctness of a variety of examples, including a distributed merge sort, a distributed load-balancing mapper, and a variant of the map-reduce model, using concise specifications.

While Actris was already presented in a conference paper (POPL'20), this paper expands the prior presentation significantly. Moreover, it extends Actris to **Actris 2.0** with a notion of *subprotocols*—based on session-type subtyping—that permits additional flexibility when composing channel endpoints, and that takes full advantage of the asynchronous semantics of message passing in Actris. Soundness of Actris 2.0 is proved using a model of its protocol mechanism in the Iris framework. We have mechanised the theory of Actris, together with custom tactics, as well as all examples in the paper, in the Coq proof assistant.

Publications

Actris: Session-Type Based Reasoning in Separation Logic

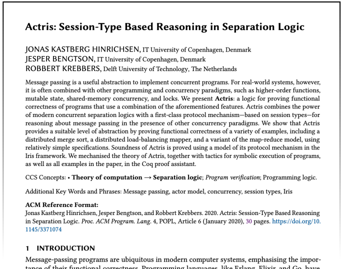
- ACM SIGPLAN Symposium on Principles of Programming Languages 2020 [POPL'20]

Actris 2.0: Asynchronous Session-Type Based Reasoning in Separation Logic

- Journal of Logical Methods in Computer Science [LMCS] (Conditionally accepted)

Machine-Checked Semantic Session Typing

- Certified Programs and Proofs Conference 2021 [CPP'21] (Distinguished Paper Award)



Actris: Session-Type Based Reasoning in Separation Logic

JONAS KASTBERG HINRICHSEN, IT University of Copenhagen, Denmark
JESPER BENGTSON, IT University of Copenhagen, Denmark
ROBBERT KREBBERS, Delft University of Technology, The Netherlands

Message passing is a useful abstraction to implement concurrent programs. For real-world systems, however, it is often combined with other programming and concurrency paradigms, such as higher-order functions, mutable state, shared-memory concurrency, and locks. We present **Actris**: a logic for proving functional correctness of programs that use a combination of the aforementioned features. Actris combines the power of modern concurrent separation logics with a first-class protocol mechanism—based on session types—for reasoning about message passing in the presence of other concurrency paradigms. We show that Actris provides a suitable level of abstraction by proving functional correctness of a variety of examples, including a distributed merge sort, a distributed load-balancing mapper, and a variant of the map-reduce model, using relatively simple specifications. Soundness of Actris is proved using a model of its protocol mechanism in the Iris framework. We mechanised the theory of Actris, together with tactics for symbolic execution of programs, as well as all examples in the paper, in the Coq proof assistant.

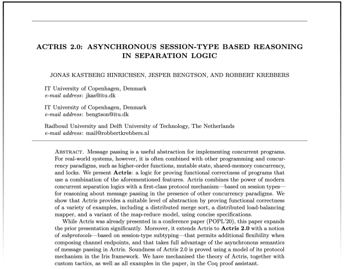
CCS Concepts: • **Theory of computation** → **Separation logic**; *Program verification*; Programming logic.

Additional Key Words and Phrases: Message passing, actor model, concurrency, session types, Iris

ACM Reference Format:
Jonas Kastberg Hinrichsen, Jesper Bengtson, and Robbert Krebbers. 2020. Actris: Session-Type Based Reasoning in Separation Logic. *Proc. ACM Program. Lang.* 4, POPL, Article 6 (January 2020), 30 pages. https://doi.org/10.1145/3371074

1 INTRODUCTION

Message-passing programs are ubiquitous in modern computer systems, emphasising the impor-



ACTRIS 2.0: ASYNCHRONOUS SESSION-TYPE BASED REASONING IN SEPARATION LOGIC

JONAS KASTBERG HINRICHSEN, JESPER BENGTSON, AND ROBBERT KREBBERS

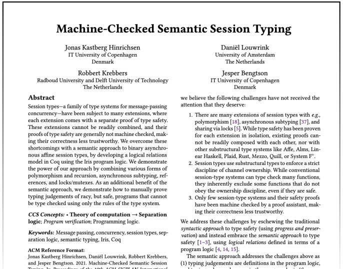
IT University of Copenhagen, Denmark
e-mail address: jkas@itu.dk

IT University of Copenhagen, Denmark
e-mail address: bengtson@itu.dk

Radboud University and Delft University of Technology, The Netherlands
e-mail address: mail@robbertkrebbers.nl

Abstract. Message passing is a useful abstraction for implementing concurrent programs. For real-world systems, however, it is often combined with other programming and concurrency paradigms, such as higher-order functions, mutable state, shared-memory concurrency, and locks. We present **Actris**: a logic for proving functional correctness of programs that use a combination of the aforementioned features. Actris combines the power of modern concurrent separation logics with a first-class protocol mechanism—based on session types—for reasoning about message passing in the presence of other concurrency paradigms. We show that Actris provides a suitable level of abstraction by proving functional correctness of a variety of examples, including a distributed merge sort, a distributed load-balancing mapper, and a variant of the map-reduce model, using concise specifications.

While Actris was already presented in a conference paper (POPL'20), this paper expands the prior presentation significantly. Moreover, it extends Actris to **Actris 2.0** with a notion of *subprotocols*—based on session-type subtyping—that permits additional flexibility when composing channel endpoints, and that takes full advantage of the asynchronous semantics of message passing in Actris. Soundness of Actris 2.0 is proved using a model of its protocol mechanism in the Iris framework. We have mechanised the theory of Actris, together with custom tactics, as well as all examples in the paper, in the Coq proof assistant.



Machine-Checked Semantic Session Typing

Jonas Kastberg Hinrichsen
IT University of Copenhagen
Denmark

Daniël Louwrink
University of Amsterdam
The Netherlands

Robbert Krebbers
Radboud University and Delft University of Technology
The Netherlands

Jesper Bengtson
IT University of Copenhagen
Denmark

Abstract

Session types—a family of type systems for message-passing concurrency—have been subject to many extensions, where each extension comes with a separate proof of type safety. These extensions cannot be readily combined, and their proofs of type safety are generally not machine checked, making their correctness less trustworthy. We overcome these shortcomings with a semantic approach to binary asynchronous affine session types, by developing a logical relations model in Coq using the Iris program logic. We demonstrate the power of our approach by combining various forms of polymorphism and recursion, asynchronous subtyping, references, and locks/mutexes. As an additional benefit of the semantic approach, we demonstrate how to manually prove typing judgements of racy, but safe, programs that cannot be type checked using only the rules of the type system.

CCS Concepts: • **Theory of computation** → **Separation logic**; *Program verification*; Programming logic.

Keywords: Message passing, concurrency, session types, separation logic, semantic typing, Iris, Coq

ACM Reference Format:
Jonas Kastberg Hinrichsen, Daniël Louwrink, Robbert Krebbers, and Jesper Bengtson. 2021. Machine-Checked Semantic Session

we believe the following challenges have not received the attention that they deserve:

1. There are many extensions of session types with *e.g.*, polymorphism [18], asynchronous subtyping [37], and sharing via locks [5]. While type safety has been proven for each extension in isolation, existing proofs cannot be readily composed with each other, nor with other substructural type systems like Affe, Alms, Linear Haskell, Plaid, Rust, Mezzo, Quill, or System F°.
2. Session types use substructural types to enforce a strict discipline of channel ownership. While conventional session-type systems can type check many functions, they inherently exclude some functions that do not obey the ownership discipline, even if they are safe.
3. Only few session-type systems and their safety proofs have been machine checked by a proof assistant, making their correctness less trustworthy.

We address these challenges by eschewing the traditional *syntactic approach* to type safety (using *progress and preservation*) and instead embrace the *semantic approach* to type safety [1–3], using *logical relations* defined in terms of a program logic [4, 14, 15].

The semantic approach addresses the challenges above as (1) typing judgements are definitions in the program logic,

Actris and Actris 2.0

Papers: POPL'20 and LMCS

Thesis: Chapter 3

joint work with

Jesper Bengtson, IT University of Copenhagen
Robbert Krebbers, Radboud University

Operational Semantics

Operational semantics: A mathematical model of a programming language

Operational semantics: A mathematical model of a programming language
Programming Language: Representative language

Operational Semantics

Operational semantics: A mathematical model of a programming language
Programming Language: Representative language with

- Higher-order functions

Operational semantics: A mathematical model of a programming language
Programming Language: Representative language with

- Higher-order functions
- Higher-order mutable references

Operational semantics: A mathematical model of a programming language
Programming Language: Representative language with

- Higher-order functions
- Higher-order mutable references
- Fork-based concurrency

Operational Semantics

Operational semantics: A mathematical model of a programming language
Programming Language: Representative language with

- Higher-order functions
- Higher-order mutable references
- Fork-based concurrency

$$v \in \mathsf{Val} ::= () \mid i \mid b \mid \ell \mid \texttt{rec}\, f\ x := e \mid \ldots \qquad (i \in \mathbb{Z}, b \in \mathbb{B}, \ell \in \mathsf{Loc})$$
$$e \in \mathsf{Expr} ::= v \mid x \mid e_1(e_2) \mid \texttt{ref}\ (e) \mid\ !\, e \mid e_1 \leftarrow e_2 \mid \texttt{fork}\ \{e\} \mid \ldots$$

Operational Semantics

Operational semantics: A mathematical model of a programming language
Programming Language: Representative language with

- Higher-order functions
- Higher-order mutable references
- Fork-based concurrency

$$v \in \mathsf{Val} ::= () \mid i \mid b \mid \ell \mid \texttt{rec}\, f\ x := e \mid \ldots \qquad (i \in \mathbb{Z}, b \in \mathbb{B}, \ell \in \mathsf{Loc})$$
$$e \in \mathsf{Expr} ::= v \mid x \mid e_1(e_2) \mid \texttt{ref}\ (e) \mid !\, e \mid e_1 \leftarrow e_2 \mid \texttt{fork}\ \{e\} \mid \ldots$$

HeapLang: Language shipped with Iris

Operational Semantics

Operational semantics: A mathematical model of a programming language
Programming Language: Representative language with

- Higher-order functions
- Higher-order mutable references
- Fork-based concurrency

$$v \in \mathsf{Val} ::= () \mid i \mid b \mid \ell \mid \mathtt{rec}\, f\ x := e \mid \ldots \qquad (i \in \mathbb{Z}, b \in \mathbb{B}, \ell \in \mathsf{Loc})$$
$$e \in \mathsf{Expr} ::= v \mid x \mid e_1(e_2) \mid \mathtt{ref}\ (e) \mid {!}\, e \mid e_1 \leftarrow e_2 \mid \mathtt{fork}\ \{e\} \mid \ldots$$

HeapLang: Language shipped with Iris

- Includes many state-of-the-art features

Operational Semantics

Operational semantics: A mathematical model of a programming language

Programming Language: Representative language with

- Higher-order functions
- Higher-order mutable references
- Fork-based concurrency

$$v \in \mathsf{Val} ::= () \mid i \mid b \mid \ell \mid \texttt{rec}\, f\ x := e \mid \ldots \qquad (i \in \mathbb{Z}, b \in \mathbb{B}, \ell \in \mathsf{Loc})$$
$$e \in \mathsf{Expr} ::= v \mid x \mid e_1(e_2) \mid \texttt{ref}\,(e) \mid !\,e \mid e_1 \leftarrow e_2 \mid \texttt{fork}\,\{e\} \mid \ldots$$

HeapLang: Language shipped with Iris

- Includes many state-of-the-art features
- Integrated with the Iris separation logic

Operational Semantics

Operational semantics: A mathematical model of a programming language
Programming Language: Representative language with

- Higher-order functions
- Higher-order mutable references
- Fork-based concurrency

$$
\begin{aligned}
v \in \mathsf{Val} &::= () \mid i \mid b \mid \ell \mid \texttt{rec}\, f\ x := e \mid \ldots \qquad (i \in \mathbb{Z}, b \in \mathbb{B}, \ell \in \mathsf{Loc}) \\
e \in \mathsf{Expr} &::= v \mid x \mid e_1(e_2) \mid \texttt{ref}\,(e) \mid !\,e \mid e_1 \leftarrow e_2 \mid \texttt{fork}\,\{e\} \mid \ldots
\end{aligned}
$$

HeapLang: Language shipped with Iris

- Includes many state-of-the-art features
- Integrated with the Iris separation logic
- Already mechanised, with tactic support

Implementation of message-passing primitives

Extend HeapLang with message passing

Implementation of message-passing primitives

Extend HeapLang with message passing

- As a straightforward implementation using lock-protected buffers

Implementation of message-passing primitives

Extend HeapLang with message passing

- As a straightforward implementation using lock-protected buffers

Message-passing primitives

`new_chan` (): Allocate channel and return two channel endpoints

`send` c v : Send the value *v* over the channel endpoint *c*

`recv` c : Await and return the first value over channel endpoint *c*

Implementation of message-passing primitives

Extend HeapLang with message passing

- As a straightforward implementation using lock-protected buffers

Message-passing primitives

$\texttt{new\_chan}\ ()$: Allocate channel and return two channel endpoints

$\texttt{send}\ \mathrm{c}\ \mathrm{v}$: Send the value v over the channel endpoint c

$\texttt{recv}\ \mathrm{c}$: Await and return the first value over channel endpoint c

Example:

$$\begin{array}{ll}
\texttt{let}\,(c, c') := \texttt{new_chan}\ ()\ \texttt{in} & \\
\texttt{fork}\ \{\texttt{let}\, x := \texttt{recv}\ c'\ \texttt{in}\, \texttt{send}\ c'\ (x+2)\}\,; & \text{// Service thread} \\
\texttt{send}\ c\ 40;\ \texttt{recv}\ c & \text{// Client thread}
\end{array}$$

Implementation of message-passing primitives

Extend HeapLang with message passing

- As a straightforward implementation using lock-protected buffers

Message-passing primitives

`new_chan` (): Allocate channel and return two channel endpoints

`send` c v : Send the value v over the channel endpoint c

`recv` c : Await and return the first value over channel endpoint c

Example:

$$
\begin{array}{ll}
\texttt{let}\,(c, c') := \texttt{new\_chan}\;()\;\texttt{in} & \\
\texttt{fork}\;\{\texttt{let}\,x := \texttt{recv}\;c'\;\texttt{in}\,\texttt{send}\;c'\;(x+2)\}\,; & \text{// Service thread} \\
\texttt{send}\;c\;40;\;\texttt{recv}\;c & \text{// Client thread}
\end{array}
$$

Many variants of message passing exist

Ours is: binary, asynchronous, order-preserving and reliable

Implementation of message-passing primitives

Extend HeapLang with message passing

- As a straightforward implementation using lock-protected buffers

Message-passing primitives

$\texttt{new\_chan}\ ()$: Allocate channel and return two channel endpoints

$\texttt{send}\ \mathrm{c}\ \mathrm{v}$: Send the value v over the channel endpoint c

$\texttt{recv}\ \mathrm{c}$: Await and return the first value over channel endpoint c

Example:

$$\begin{array}{ll}
\texttt{let}\,(c, c') := \texttt{new_chan}\ ()\ \texttt{in} & \\
\texttt{fork}\ \{\texttt{let}\, x := \texttt{recv}\ c'\ \texttt{in}\,\texttt{send}\ c'\ (x+2)\}\,; & \textit{// Service thread} \\
\texttt{send}\ c\ 40;\ \texttt{recv}\ c & \textit{// Client thread}
\end{array}$$

Many variants of message passing exist

Ours is: binary, asynchronous, order-preserving and reliable

To simulate state-of-the-art message passing (like in the Go language)

Goal

Example program:

$$
\begin{array}{ll}
\texttt{let}\,(c, c') := \texttt{new\_chan}\,()\,\texttt{in} & \\
\texttt{fork}\,\{\texttt{let}\,x := \texttt{recv}\ c'\,\texttt{in}\,\texttt{send}\ c'\ (x+2)\}\,; & \text{// Service thread} \\
\texttt{send}\ c\ 40;\ \texttt{recv}\ c & \text{// Client thread}
\end{array}
$$

Goal

Example program:

$$\begin{array}{ll} \texttt{let}\,(c, c') := \texttt{new\_chan}\;()\;\texttt{in} & \\ \texttt{fork}\;\{\texttt{let}\,x := \texttt{recv}\;c'\;\texttt{in}\,\texttt{send}\;c'\;(x+2)\}\,; & \text{// Service thread} \\ \texttt{send}\;c\;40;\;\texttt{recv}\;c & \text{// Client thread} \end{array}$$

Show that:

Program does not crash

Program is correct (returns 42)

Session types (recap)

Symbols

$$
\begin{aligned}
A ::= {} & \mathsf{Z} \mid \mathsf{B} \mid 1 \mid \\
& \texttt{chan}\ S \mid \ldots
\end{aligned}
$$

Session types (recap)

Symbols

$$
\begin{aligned}
A ::= {} & \mathbb{Z} \mid \mathbb{B} \mid 1 \mid \\
& \texttt{chan}\ S \mid \ldots
\end{aligned}
$$

$$
\begin{aligned}
S ::= {} & \mathbf{!}A.\, S && \mid \\
& \mathbf{?}A.\, S && \mid \\
& \texttt{end} && \mid \ldots
\end{aligned}
$$

Session types (recap)

Symbols

$$
\begin{aligned}
A ::= {} & \mathbb{Z} \mid \mathbb{B} \mid 1 \mid \\
& \texttt{chan}\ S \mid \ldots
\end{aligned}
$$

$$
\begin{aligned}
S ::= {} & \mathbf{!}A.\, S && \mid \\
& \mathbf{?}A.\, S && \mid \\
& \texttt{end} && \mid \ldots
\end{aligned}
$$

Example

$$\mathbf{!}\mathbb{Z}.\, \mathbf{?}\mathbb{Z}.\, \texttt{end}$$

Session types (recap)

Symbols

$$
\begin{aligned}
A ::= {} & \mathrm{Z} \mid \mathrm{B} \mid 1 \mid \\
& \texttt{chan}\ S \mid \ldots
\end{aligned}
$$

$$
\begin{aligned}
S ::= {} & \mathbf{!}A.\,S & \mid \\
& \mathbf{?}A.\,S & \mid \\
& \texttt{end} & \mid \ldots
\end{aligned}
$$

Example

$\mathbf{!}\mathrm{Z}.\,\mathbf{?}\mathrm{Z}.\,\texttt{end}$

Duality

$$
\begin{aligned}
\overline{\mathbf{!}A.\,S} &= \mathbf{?}A.\,\overline{S} \\
\overline{\mathbf{?}A.\,S} &= \mathbf{!}A.\,\overline{S} \\
\overline{\texttt{end}} &= \texttt{end}
\end{aligned}
$$

Session types (recap)

Symbols

$$A ::= \mathsf{Z} \mid \mathsf{B} \mid 1 \mid \texttt{chan}\ S \mid \ldots$$

$$S ::= \mathbf{!}A.\,S \mid \mathbf{?}A.\,S \mid \texttt{end} \mid \ldots$$

Example

$\mathbf{!}\mathsf{Z}.\,\mathbf{?}\mathsf{Z}.\,\texttt{end}$

Duality

$$\begin{aligned} \overline{\mathbf{!}A.\,S} &= \mathbf{?}A.\,\overline{S} \\ \overline{\mathbf{?}A.\,S} &= \mathbf{!}A.\,\overline{S} \\ \overline{\texttt{end}} &= \texttt{end} \end{aligned}$$

Usage

$c : \texttt{chan}\ S$

Session types (recap)

Symbols

$$A ::= \mathsf{Z} \mid \mathsf{B} \mid 1 \mid \texttt{chan}\ S \mid \ldots$$

$$S ::= \mathbf{!}A.\,S \mid \mathbf{?}A.\,S \mid \texttt{end} \mid \ldots$$

Example

$$\mathbf{!}\mathsf{Z}.\,\mathbf{?}\mathsf{Z}.\,\texttt{end}$$

Duality

$$\begin{aligned} \overline{\mathbf{!}A.\,S} &= \mathbf{?}A.\,\overline{S} \\ \overline{\mathbf{?}A.\,S} &= \mathbf{!}A.\,\overline{S} \\ \overline{\texttt{end}} &= \texttt{end} \end{aligned}$$

Usage

$$c : \texttt{chan}\ S$$

Rules

$$\texttt{new\_chan} : 1 \multimap \texttt{chan}\ S \times \texttt{chan}\ \overline{S}$$

$$\texttt{send} : (\texttt{chan}\ (\mathbf{!}A.\,S) \times A) \multimap \texttt{chan}\ S$$

$$\texttt{recv} : \texttt{chan}\ (\mathbf{?}A.\,S) \multimap (A \times \texttt{chan}\ S)$$

Example program - via session types

Example program:

$$\begin{array}{ll}\texttt{let}\,(c, c') := \texttt{new\_chan}\;()\;\texttt{in} & \\ \texttt{fork}\;\{\texttt{let}\,x := \texttt{recv}\;c'\;\texttt{in}\,\texttt{send}\;c'\;(x+2)\}\,; & // \text{ Service thread} \\ \texttt{send}\;c\;40;\;\texttt{recv}\;c & // \text{ Client thread}\end{array}$$

Example program - via session types

Example program:

$$\begin{array}{ll}\texttt{let}\,(c, c') := \texttt{new\_chan}\;()\;\texttt{in} & \\ \texttt{fork}\;\{\texttt{let}\,x := \texttt{recv}\;c'\;\texttt{in}\,\texttt{send}\;c'\;(x+2)\}\,; & \text{// Service thread} \\ \texttt{send}\;c\;40;\;\texttt{recv}\;c & \text{// Client thread}\end{array}$$

Session types:

$$\begin{array}{lll} c & : & \texttt{chan}\,(\mathbf{!}\mathbb{Z}.\,\mathbf{?}\mathbb{Z}.\,\texttt{end}) \qquad \texttt{and} \\ c' & : & \texttt{chan}\,(\mathbf{?}\mathbb{Z}.\,\mathbf{!}\mathbb{Z}.\,\texttt{end}) \end{array}$$

Example program - via session types

Example program:

$$
\begin{array}{ll}
\texttt{let}\,(c, c') := \texttt{new\_chan}\ ()\ \texttt{in} & \\
\texttt{fork}\ \{\texttt{let}\, x := \texttt{recv}\ c'\ \texttt{in}\, \texttt{send}\ c'\ (x+2)\}\,; & \text{// Service thread} \\
\texttt{send}\ c\ 40;\ \texttt{recv}\ c & \text{// Client thread}
\end{array}
$$

Session types:

$$
\begin{array}{lcl}
c & : & \texttt{chan}\ (\mathbf{!}\mathbb{Z}.\,\mathbf{?}\mathbb{Z}.\,\texttt{end}) \qquad \texttt{and} \\
c' & : & \texttt{chan}\ (\mathbf{?}\mathbb{Z}.\,\mathbf{!}\mathbb{Z}.\,\texttt{end})
\end{array}
$$

Properties obtained:

- ☑ Program does not crash
- ☒ Program is correct (returns 42)

Actris

Actris:
Dependent separation protocols

Actris:
Dependent separation protocols

(Like logical session types)

Dependent separation protocols - Definitions

	Dependent separation protocols	Session types
Symbols	$prot ::= \,!\,\vec{x}:\vec{\tau}\,\langle v\rangle\{P\}.\,prot \;\mid$ $\quad\quad\;\; ?\vec{x}:\vec{\tau}\,\langle v\rangle\{P\}.\,prot \;\mid$ $\quad\quad\;\; \mathsf{end}$	$S ::= \,!A.\,S \;\mid$ $\quad\;\; ?A.\,S \;\mid$ $\quad\;\; \mathtt{end} \;\mid \ldots$

	Dependent separation protocols	**Session types**
Symbols	$prot ::= \mathbf{!}\, \vec{x} : \vec{\tau}\, \langle v \rangle \{P\}.\, prot \quad \vert$ $\mathbf{?} \vec{x} : \vec{\tau}\, \langle v \rangle \{P\}.\, prot \quad \vert$ end	$S ::= \mathbf{!} A.\, S \quad \vert$ $\mathbf{?} A.\, S \quad \vert$ $\mathtt{end} \quad \vert \ldots$
Example	$\mathbf{!}\, (x : \mathbb{Z})\, \langle x \rangle \{\mathsf{True}\}.\, \mathbf{?} (y : \mathbb{Z})\, \langle y \rangle \{y = (x + 2)\}.\, \mathsf{end}$	$\mathbf{!}\mathsf{Z}.\, \mathbf{?}\mathsf{Z}.\, \mathtt{end}$

Dependent separation protocols - Definitions

	Dependent separation protocols	**Session types**
Symbols	$\begin{aligned} \mathit{prot} ::= {} & \mathbf{!}\, \vec{x}:\vec{\tau}\, \langle v \rangle \{P\}.\, \mathit{prot} \quad \mid \\ & \mathbf{?}\vec{x}:\vec{\tau}\, \langle v \rangle \{P\}.\, \mathit{prot} \quad \mid \\ & \mathsf{end} \end{aligned}$	$\begin{aligned} S ::= {} & \mathbf{!}A.\, S \quad \mid \\ & \mathbf{?}A.\, S \quad \mid \\ & \mathtt{end} \quad \mid \ldots \end{aligned}$
Example	$\mathbf{!}\,(x:\mathbb{Z})\,\langle x \rangle \{\mathsf{True}\}.\, \mathbf{?}(y:\mathbb{Z})\,\langle y \rangle \{y = (x+2)\}.\, \mathsf{end}$	$\mathbf{!}\mathrm{Z}.\, \mathbf{?}\mathrm{Z}.\, \mathtt{end}$
Duality	$\begin{aligned} \overline{\mathbf{!}\, \vec{x}:\vec{\tau}\, \langle v \rangle \{P\}.\, \mathit{prot}} &= \mathbf{?}\vec{x}:\vec{\tau}\, \langle v \rangle \{P\}.\, \overline{\mathit{prot}} \\ \overline{\mathbf{?}\vec{x}:\vec{\tau}\, \langle v \rangle \{P\}.\, \mathit{prot}} &= \mathbf{!}\, \vec{x}:\vec{\tau}\, \langle v \rangle \{P\}.\, \overline{\mathit{prot}} \\ \overline{\mathsf{end}} &= \mathsf{end} \end{aligned}$	$\begin{aligned} \overline{\mathbf{!}A.\, S} &= \mathbf{?}A.\, \overline{S} \\ \overline{\mathbf{?}A.\, S} &= \mathbf{!}A.\, \overline{S} \\ \overline{\mathtt{end}} &= \mathtt{end} \end{aligned}$

Dependent separation protocols - Definitions

	Dependent separation protocols	Session types
Symbols	$prot ::= \mathbf{!}\,\vec{x}:\vec{\tau}\,\langle v\rangle\{P\}.\,prot \mid \mathbf{?}\vec{x}:\vec{\tau}\,\langle v\rangle\{P\}.\,prot \mid \mathsf{end}$	$S ::= \mathbf{!}A.\,S \mid \mathbf{?}A.\,S \mid \mathtt{end} \mid \ldots$
Example	$\mathbf{!}\,(x:\mathbb{Z})\,\langle x\rangle\{\mathrm{True}\}.\,\mathbf{?}(y:\mathbb{Z})\,\langle y\rangle\{y=(x+2)\}.\,\mathsf{end}$	$\mathbf{!}\mathrm{Z}.\,\mathbf{?}\mathrm{Z}.\,\mathtt{end}$
Duality	$\overline{\mathbf{!}\,\vec{x}:\vec{\tau}\,\langle v\rangle\{P\}.\,prot} = \mathbf{?}\vec{x}:\vec{\tau}\,\langle v\rangle\{P\}.\,\overline{prot}$ $\overline{\mathbf{?}\vec{x}:\vec{\tau}\,\langle v\rangle\{P\}.\,prot} = \mathbf{!}\,\vec{x}:\vec{\tau}\,\langle v\rangle\{P\}.\,\overline{prot}$ $\overline{\mathsf{end}} = \mathsf{end}$	$\overline{\mathbf{!}A.\,S} = \mathbf{?}A.\,\overline{S}$ $\overline{\mathbf{?}A.\,S} = \mathbf{!}A.\,\overline{S}$ $\overline{\mathtt{end}} = \mathtt{end}$
Usage	$c \rightarrowtail prot$	$c : \mathtt{chan}\ S$

Dependent separation protocols - Rules

	Dependent separation protocols	Session types
New	$\{\mathsf{True}\}$ $\quad \texttt{new\_chan}\ ()$ $\{(c, c').\ c \rightarrowtail \mathit{prot} * c' \rightarrowtail \overline{\mathit{prot}}\}$	$\texttt{new\_chan} : 1 \multimap \texttt{chan}\ S \times \texttt{chan}\ \overline{S}$

Dependent separation protocols - Rules

	Dependent separation protocols	Session types
New	$\{\mathsf{True}\}$ $\quad$ `new_chan` $()$ $\{(c, c').\, c \rightarrowtail \mathit{prot} * c' \rightarrowtail \overline{\mathit{prot}}\}$	`new_chan` $: 1 \multimap \texttt{chan}\ S \times \texttt{chan}\ \overline{S}$
Send	$\{c \rightarrowtail \mathbf{!}\, \vec{x}{:}\vec{\tau}\, \langle v \rangle \{P\}.\, \mathit{prot} * P[\vec{t}/\vec{x}]\}$ $\quad$ `send` $c\ (v[\vec{t}/\vec{x}])$ $\{c \rightarrowtail \mathit{prot}[\vec{t}/\vec{x}]\}$	`send` $: (\texttt{chan}\ (\mathbf{!}A.\, S) \times A) \multimap \texttt{chan}\ S$

Dependent separation protocols - Rules

	Dependent separation protocols	**Session types**
New	$\{\text{True}\}$ $\quad\texttt{new\_chan}\ ()$ $\{(c, c').\ c \rightarrowtail \mathit{prot} * c' \rightarrowtail \overline{\mathit{prot}}\}$	$\texttt{new\_chan} : 1 \multimap \texttt{chan}\ S \times \texttt{chan}\ \overline{S}$
Send	$\{c \rightarrowtail\ !\,\vec{x}{:}\vec{\tau}\,\langle v\rangle\{P\}.\ \mathit{prot} * P[\vec{t}/\vec{x}]\}$ $\quad\texttt{send}\ c\ (v[\vec{t}/\vec{x}])$ $\{c \rightarrowtail \mathit{prot}[\vec{t}/\vec{x}]\}$	$\texttt{send} : (\texttt{chan}\ (!A.\ S) \times A) \multimap \texttt{chan}\ S$
Recv	$\{c \rightarrowtail ?\vec{x}{:}\vec{\tau}\,\langle v\rangle\{P\}.\ \mathit{prot}\}$ $\quad\texttt{recv}\ c$ $\{w.\ \exists(\vec{y} : \vec{\tau}).\ (w = v[\vec{y}/\vec{x}]) *$ $\qquad P[\vec{y}/\vec{x}] * c \rightarrowtail \mathit{prot}[\vec{y}/\vec{x}]\}$	$\texttt{recv} :\ \texttt{chan}\ (?A.\ S) \multimap (A \times \texttt{chan}\ S)$

Example program - via dependent separation protocols

Example program:

$$\begin{array}{ll}
\texttt{let}\,(c, c') := \texttt{new_chan}\;()\;\texttt{in} & \\
\texttt{fork}\;\{\texttt{let}\,x := \texttt{recv}\;c'\;\texttt{in}\,\texttt{send}\;c'\;(x+2)\}\,; & \text{// Service thread} \\
\texttt{send}\;c\;40;\;\texttt{recv}\;c & \text{// Client thread}
\end{array}$$

Example program - via dependent separation protocols

Example program:

$$\begin{array}{ll}\texttt{let}\,(c, c') := \texttt{new\_chan}\;()\;\texttt{in} & \\ \texttt{fork}\;\{\texttt{let}\,x := \texttt{recv}\;c'\;\texttt{in}\,\texttt{send}\;c'\;(x+2)\}\,; & \text{// Service thread} \\ \texttt{send}\;c\;40;\;\texttt{recv}\;c & \text{// Client thread}\end{array}$$

Dependent separation protocols:

$$\begin{array}{rl}c \rightarrowtail \mathbf{!}\,(x:\mathbb{Z})\,\langle x\rangle\{\mathsf{True}\}.\,\mathbf{?}(y:\mathbb{Z})\,\langle y\rangle\{y=(x+2)\}.\,\mathsf{end} & \quad \texttt{and} \\ c' \rightarrowtail \mathbf{?}(x:\mathbb{Z})\,\langle x\rangle\{\mathsf{True}\}.\,\mathbf{!}\,(y:\mathbb{Z})\,\langle y\rangle\{y=(x+2)\}.\,\mathsf{end} & \end{array}$$

Example program - via dependent separation protocols

Example program:

$$
\begin{array}{ll}
\texttt{let}\,(c, c') := \texttt{new\_chan}\;()\;\texttt{in} & \\
\texttt{fork}\;\{\texttt{let}\,x := \texttt{recv}\;c'\;\texttt{in}\,\texttt{send}\;c'\;(x+2)\}\,; & \text{// Service thread} \\
\texttt{send}\;c\;40;\;\texttt{recv}\;c & \text{// Client thread}
\end{array}
$$

Dependent separation protocols:

$$
\begin{array}{rl}
c \rightarrowtail & \mathbf{!}\,(x:\mathbb{Z})\,\langle x\rangle\{\mathsf{True}\}.\,\mathbf{?}(y:\mathbb{Z})\,\langle y\rangle\{y = (x+2)\}.\,\mathsf{end} \qquad \texttt{and} \\
c' \rightarrowtail & \mathbf{?}(x:\mathbb{Z})\,\langle x\rangle\{\mathsf{True}\}.\,\mathbf{!}\,(y:\mathbb{Z})\,\langle y\rangle\{y = (x+2)\}.\,\mathsf{end}
\end{array}
$$

Properties obtained:

- ☑ Program does not crash (safety)
- ☑ Program is correct (returns 42)

Example program:

$$\texttt{let}\,(c, c') := \texttt{new\_chan}\;()\;\texttt{in}$$
$$\texttt{fork}\;\{\texttt{let}\,\ell := \texttt{recv}\;c'\;\texttt{in}\,\ell \leftarrow (!\,\ell + 2);\;\texttt{send}\;c'\;()\};\quad \text{// Service thread}$$
$$\texttt{let}\,\ell := \texttt{ref}\;40\;\texttt{in}\,\texttt{send}\;c\;\ell;\;\texttt{recv}\;c;\;\;!\,\ell\qquad \text{// Client thread}$$

Example program - References

Example program:

$$
\begin{array}{ll}
\texttt{let}\,(c, c') := \texttt{new\_chan}\;()\;\texttt{in} & \\
\texttt{fork}\;\{\texttt{let}\,\ell := \texttt{recv}\;c'\;\texttt{in}\,\ell \leftarrow (!\,\ell + 2);\;\texttt{send}\;c'\;()\}\,; & \text{// Service thread} \\
\texttt{let}\,\ell := \texttt{ref}\;40\;\texttt{in}\,\texttt{send}\;c\;\ell;\;\texttt{recv}\;c;\;\;!\,\ell & \text{// Client thread}
\end{array}
$$

Dependent separation protocols:

$$
\begin{array}{rl}
c \rightarrowtail \mathbf{!}\,(\ell : \mathsf{Loc})\;(x : \mathbb{Z})\,\langle \ell \rangle \{\ell \mapsto x\}.\,\mathbf{?}\langle () \rangle \{\ell \mapsto (x+2)\}.\,\mathsf{end} & \qquad \texttt{and} \\
c' \rightarrowtail \mathbf{?}(\ell : \mathsf{Loc})\;(x : \mathbb{Z})\,\langle \ell \rangle \{\ell \mapsto x\}.\,\mathbf{!}\,\langle () \rangle \{\ell \mapsto (x+2)\}.\,\mathsf{end} &
\end{array}
$$

Example program - References

Example program:

$$\texttt{let}\,(c, c') := \texttt{new\_chan}\;()\;\texttt{in}$$
$$\texttt{fork}\;\{\texttt{let}\,\ell := \texttt{recv}\;c'\;\texttt{in}\,\ell \leftarrow (!\,\ell + 2);\;\texttt{send}\;c'\;()\}\,; \quad \text{// Service thread}$$
$$\texttt{let}\,\ell := \texttt{ref}\;40\;\texttt{in}\;\texttt{send}\;c\;\ell;\;\texttt{recv}\;c;\;\;!\,\ell \quad \text{// Client thread}$$

$$\boxed{\{\mathsf{True}\}\,\texttt{ref}\;v\,\{\ell.\,\ell \mapsto v\}}$$

Dependent separation protocols:

$$c \rightarrowtail \mathbf{!}\,(\ell : \mathsf{Loc})\;(x : \mathbb{Z})\,\langle \ell \rangle \{\ell \mapsto x\}.\,\mathbf{?}\langle () \rangle \{\ell \mapsto (x + 2)\}.\,\mathsf{end} \qquad \texttt{and}$$
$$c' \rightarrowtail \mathbf{?}(\ell : \mathsf{Loc})\;(x : \mathbb{Z})\,\langle \ell \rangle \{\ell \mapsto x\}.\,\mathbf{!}\,\langle () \rangle \{\ell \mapsto (x + 2)\}.\,\mathsf{end}$$

Example program:

$$\texttt{let}\,(c, c') := \texttt{new\_chan}\ ()\ \texttt{in}$$
$$\texttt{fork}\ \{\texttt{let}\,\ell := \texttt{recv}\ c'\ \texttt{in}\,\ell \leftarrow (!\,\ell + 2);\ \texttt{send}\ c'\ ()\}\,; \quad \text{// Service thread}$$
$$\texttt{let}\,\ell := \texttt{ref}\ 40\ \texttt{in}\ \texttt{send}\ c\ \ell;\ \texttt{recv}\ c;\ \ !\,\ell \quad \text{// Client thread}$$

$\{\mathrm{True}\}\,\texttt{ref}\ v\,\{\ell.\,\ell \mapsto v\}$

$\{\ell \mapsto v\}\ !\,\ell\,\{w.\,w = v \wedge \ell \mapsto v\}$

Dependent separation protocols:

$$c \rightarrowtail \mathbf{!}\,(\ell : \mathrm{Loc})\ (x : \mathbb{Z})\,\langle \ell \rangle \{\ell \mapsto x\}.\ \mathbf{?}\langle () \rangle \{\ell \mapsto (x + 2)\}.\,\mathrm{end} \qquad \texttt{and}$$
$$c' \rightarrowtail \mathbf{?}(\ell : \mathrm{Loc})\ (x : \mathbb{Z})\,\langle \ell \rangle \{\ell \mapsto x\}.\ \mathbf{!}\,\langle () \rangle \{\ell \mapsto (x + 2)\}.\,\mathrm{end}$$

Example program - References

Example program:

$$\{\ell \mapsto v\}\ \ell \leftarrow w\ \{\ell \mapsto w\}$$

$$\begin{array}{ll}
\texttt{let}\,(c, c') := \texttt{new_chan}\ ()\ \texttt{in} & \\
\texttt{fork}\ \{\texttt{let}\,\ell := \texttt{recv}\ c'\ \texttt{in}\,\ell \leftarrow (!\,\ell + 2);\ \texttt{send}\ c'\ ()\}\,; & \text{// Service thread} \\
\texttt{let}\,\ell := \texttt{ref}\ 40\ \texttt{in}\,\texttt{send}\ c\ \ell;\ \texttt{recv}\ c;\ \ !\,\ell & \text{// Client thread}
\end{array}$$

$$\{\mathsf{True}\}\ \texttt{ref}\ v\ \{\ell.\, \ell \mapsto v\}$$

$$\{\ell \mapsto v\}\ !\,\ell\ \{w.\, w = v \wedge \ell \mapsto v\}$$

Dependent separation protocols:

$$\begin{array}{rl}
c \rightarrowtail \mathbf{!}\,(\ell : \mathrm{Loc})\ (x : \mathbb{Z})\,\langle \ell \rangle \{\ell \mapsto x\}.\, \mathbf{?}\langle () \rangle \{\ell \mapsto (x + 2)\}.\, \mathrm{end} & \quad \texttt{and} \\
c' \rightarrowtail \mathbf{?}(\ell : \mathrm{Loc})\ (x : \mathbb{Z})\,\langle \ell \rangle \{\ell \mapsto x\}.\, \mathbf{!}\,\langle () \rangle \{\ell \mapsto (x + 2)\}.\, \mathrm{end} &
\end{array}$$

Example program - Recursion

Example program:

$$
\begin{array}{ll}
\texttt{let}\,(c, c') := \texttt{new\_chan}\;()\;\texttt{in} & \\
\texttt{fork}\;\{\text{loop}\;\{\texttt{let}\,x := \texttt{recv}\;c'\;\texttt{in}\,\texttt{send}\;c'\;(x+2)\}\}\,; & \text{// Service thread} \\
\texttt{send}\;c\;18;\;\texttt{let}\,x := \texttt{recv}\;c\;\texttt{in} & \text{// Client thread} \\
\texttt{send}\;c\;20;\;\texttt{let}\,y := \texttt{recv}\;c\;\texttt{in}\,x + y &
\end{array}
$$

Example program - Recursion

Example program:

$$\texttt{let}\,(c, c') := \texttt{new\_chan}\,()\;\texttt{in}$$
$$\texttt{fork}\;\{\text{loop}\;\{\texttt{let}\;x := \texttt{recv}\;c'\;\texttt{in}\;\texttt{send}\;c'\;(x+2)\}\}\,; \qquad \text{// Service thread}$$
$$\texttt{send}\;c\;18;\;\texttt{let}\;x := \texttt{recv}\;c\;\texttt{in} \qquad \text{// Client thread}$$
$$\texttt{send}\;c\;20;\;\texttt{let}\;y := \texttt{recv}\;c\;\texttt{in}\;x + y$$

Dependent separation protocols:

$$c \rightarrowtail \mu rec.\,\mathbf{!}\,(x : \mathbb{Z})\,\langle x\rangle\{\text{True}\}.\,\mathbf{?}(y : \mathbb{Z})\,\langle y\rangle\{y = (x+2)\}.\,rec \qquad \texttt{and}$$
$$c' \rightarrowtail \mu rec.\,\mathbf{?}(x : \mathbb{Z})\,\langle x\rangle\{\text{True}\}.\,\mathbf{!}\,(y : \mathbb{Z})\,\langle y\rangle\{y = (x+2)\}.\,rec$$

Example program - Recursion

Example program:

$$
\begin{array}{ll}
\texttt{let}\,(c, c') := \texttt{new\_chan}\,()\,\texttt{in} & \\
\texttt{fork}\,\{\mathrm{loop}\,\{\texttt{let}\,x := \texttt{recv}\ c'\,\texttt{in}\,\texttt{send}\ c'\ (x+2)\}\}\,; & \text{// Service thread} \\
\texttt{send}\ c\ 18;\ \texttt{let}\,x := \texttt{recv}\ c\,\texttt{in} & \text{// Client thread} \\
\texttt{send}\ c\ 20;\ \texttt{let}\,y := \texttt{recv}\ c\,\texttt{in}\,x + y &
\end{array}
$$

Dependent separation protocols:

$$
\begin{array}{rl}
c \rightarrowtail \mu rec.\,!\,(x:\mathbb{Z})\,\langle x\rangle\{\mathrm{True}\}.\,?(y:\mathbb{Z})\,\langle y\rangle\{y = (x+2)\}.\,rec & \quad \texttt{and} \\
c' \rightarrowtail \mu rec.\,?(x:\mathbb{Z})\,\langle x\rangle\{\mathrm{True}\}.\,!\,(y:\mathbb{Z})\,\langle y\rangle\{y = (x+2)\}.\,rec &
\end{array}
$$

Proof:

- Client thread: follows immediately from Actris's rules
- Service thread: follows immediately using Löb induction

Example program - Subprotocols (Actris 2.0)

Example program:

$$
\begin{array}{ll}
\texttt{let}\,(c, c') := \texttt{new\_chan}\ ()\ \texttt{in} & \\
\texttt{fork}\ \{\text{loop}\ \{\texttt{let}\,x := \texttt{recv}\ c'\ \texttt{in}\,\texttt{send}\ c'\ (x+2)\}\}\,; & \text{// Service thread} \\
\texttt{send}\ c\ 18;\ \texttt{let}\,x := \texttt{recv}\ c\ \texttt{in} & \text{// Client thread} \\
\texttt{send}\ c\ 20;\ \texttt{let}\,y := \texttt{recv}\ c\,\texttt{in}\,x + y &
\end{array}
$$

Example program - Subprotocols (Actris 2.0)

Example program:

$$
\begin{array}{ll}
\texttt{let}\,(c, c') := \texttt{new\_chan}\ ()\ \texttt{in} & \\
\texttt{fork}\ \{\text{loop}\ \{\texttt{let}\, x := \texttt{recv}\ c'\ \texttt{in}\, \texttt{send}\ c'\ (x + 2)\}\}\,; & \text{// Service thread} \\
\texttt{send}\ c\ 18;\ \texttt{let}\, x := \texttt{recv}\ c\ \texttt{in} & \text{// Client thread} \\
\texttt{send}\ c\ 20;\ \texttt{let}\, y := \texttt{recv}\ c\ \texttt{in}\, x + y &
\end{array}
$$

Example program - Subprotocols (Actris 2.0)

Example program:

$$\begin{array}{ll}
\texttt{let}\,(c, c') := \texttt{new_chan}\;()\;\texttt{in} & \\
\texttt{fork}\;\{\text{loop}\;\{\texttt{let}\,x := \texttt{recv}\;c'\;\texttt{in}\,\texttt{send}\;c'\;(x+2)\}\}\,; & //\ \text{Service thread} \\
\texttt{send}\;c\;18;\;\texttt{send}\;c\;20; & //\ \text{Client thread} \\
\texttt{let}\,x := \texttt{recv}\;c\;\texttt{in}\;\texttt{let}\,y := \texttt{recv}\;c\,\texttt{in}\,x+y &
\end{array}$$

Example program - Subprotocols (Actris 2.0)

Example program:

$$
\begin{array}{ll}
\texttt{let}\,(c, c') := \texttt{new\_chan}\,()\,\texttt{in} & \\
\texttt{fork}\,\{\text{loop}\,\{\texttt{let}\,x := \texttt{recv}\,c'\,\texttt{in}\,\texttt{send}\,c'\,(x+2)\}\}\,; & \text{// Service thread} \\
\texttt{send}\,c\,18;\ \texttt{send}\,c\,20; & \text{// Client thread} \\
\texttt{let}\,x := \texttt{recv}\,c\,\texttt{in}\,\texttt{let}\,y := \texttt{recv}\,c\,\texttt{in}\,x+y &
\end{array}
$$

Dependent separation protocols:

$$
\begin{array}{ll}
c \rightarrowtail \mu rec.\,\mathbf{!}\,(x:\mathbb{Z})\,\langle x\rangle\{\text{True}\}.\,\mathbf{?}(y:\mathbb{Z})\,\langle y\rangle\{y=(x+2)\}.\,rec & \quad \texttt{and} \\
c' \rightarrowtail \mu rec.\,\mathbf{?}(x:\mathbb{Z})\,\langle x\rangle\{\text{True}\}.\,\mathbf{!}\,(y:\mathbb{Z})\,\langle y\rangle\{y=(x+2)\}.\,rec &
\end{array}
$$

Example program - Subprotocols (Actris 2.0)

Example program:

$$
\begin{array}{ll}
\texttt{let}\,(c, c') := \texttt{new\_chan}\;()\;\texttt{in} & \\
\texttt{fork}\;\{\text{loop}\;\{\texttt{let}\,x := \texttt{recv}\;c'\;\texttt{in}\,\texttt{send}\;c'\;(x+2)\}\}\,; & //\ \text{Service thread} \\
\texttt{send}\;c\;18;\;\texttt{send}\;c\;20; & //\ \text{Client thread} \\
\texttt{let}\,x := \texttt{recv}\;c\;\texttt{in}\;\texttt{let}\,y := \texttt{recv}\;c\,\texttt{in}\,x+y &
\end{array}
$$

Dependent separation protocols:

$$
\begin{array}{rl}
c \rightarrowtail \mu rec.\,!\,(x:\mathbb{Z})\,\langle x\rangle\{\text{True}\}.\,?(y:\mathbb{Z})\,\langle y\rangle\{y=(x+2)\}.\,rec & \quad \texttt{and} \\
c' \rightarrowtail \mu rec.\,?(x:\mathbb{Z})\,\langle x\rangle\{\text{True}\}.\,!\,(y:\mathbb{Z})\,\langle y\rangle\{y=(x+2)\}.\,rec &
\end{array}
$$

Subprotocol relation ($\sqsubseteq$)

Example program - Subprotocols (Actris 2.0)

Example program:

$\texttt{let}\,(c, c') := \texttt{new\_chan}\,()\,\texttt{in}$
$\texttt{fork}\,\{\text{loop}\,\{\texttt{let}\,x := \texttt{recv}\;c'\;\texttt{in}\,\texttt{send}\;c'\;(x+2)\}\}\,;$ // Service thread
$\texttt{send}\;c\;18;\;\texttt{send}\;c\;20;$ // Client thread
$\texttt{let}\,x := \texttt{recv}\;c\;\texttt{in}\;\texttt{let}\,y := \texttt{recv}\;c\;\texttt{in}\;x+y$

Dependent separation protocols:

$$c \rightarrowtail \mu rec.\, \mathbf{!}\,(x:\mathbb{Z})\,\langle x\rangle\{\text{True}\}.\, \mathbf{?}(y:\mathbb{Z})\,\langle y\rangle\{y=(x+2)\}.\, rec \qquad \texttt{and}$$
$$c' \rightarrowtail \mu rec.\, \mathbf{?}(x:\mathbb{Z})\,\langle x\rangle\{\text{True}\}.\, \mathbf{!}\,(y:\mathbb{Z})\,\langle y\rangle\{y=(x+2)\}.\, rec$$

Subprotocol relation ($\sqsubseteq$) (Inspired by asynchronous session subtyping)

Example program - Subprotocols (Actris 2.0)

Example program:

$$
\begin{array}{ll}
\mathtt{let}\,(c, c') := \mathtt{new\_chan}\,()\,\mathtt{in} & \\
\mathtt{fork}\ \{\text{loop}\ \{\mathtt{let}\,x := \mathtt{recv}\ c'\ \mathtt{in}\,\mathtt{send}\ c'\ (x+2)\}\}\,; & \text{// Service thread} \\
\mathtt{send}\ c\ 18;\ \mathtt{send}\ c\ 20; & \text{// Client thread} \\
\mathtt{let}\,x := \mathtt{recv}\ c\ \mathtt{in}\ \mathtt{let}\,y := \mathtt{recv}\ c\,\mathtt{in}\,x + y &
\end{array}
$$

Dependent separation protocols:

$$
\begin{array}{rl}
c \rightarrowtail \mu rec.\,!\,(x:\mathbb{Z})\,\langle x\rangle\{\mathrm{True}\}.\,?(y:\mathbb{Z})\,\langle y\rangle\{y = (x+2)\}.\,rec & \quad \mathtt{and} \\
c' \rightarrowtail \mu rec.\,?(x:\mathbb{Z})\,\langle x\rangle\{\mathrm{True}\}.\,!\,(y:\mathbb{Z})\,\langle y\rangle\{y = (x+2)\}.\,rec &
\end{array}
$$

Subprotocol relation ($\sqsubseteq$) (Inspired by asynchronous session subtyping):

$$
\begin{array}{rl}
& \mu rec.\,!\,(x:\mathbb{Z})\,\langle x\rangle\{\mathrm{True}\}.\,?(y:\mathbb{Z})\,\langle y\rangle\{y = (x+2)\}.\,rec \\
\sqsubseteq & \mu rec.\,!\,(x:\mathbb{Z})\,\langle x\rangle\{\mathrm{True}\}.\,?(y:\mathbb{Z})\,\langle y\rangle\{y = (x+2)\}.\,rec
\end{array}
$$

Example program - Subprotocols (Actris 2.0)

Example program:

$$
\begin{array}{ll}
\texttt{let}\,(c, c') := \texttt{new\_chan}\,()\,\texttt{in} & \\
\texttt{fork}\,\{\text{loop}\,\{\texttt{let}\,x := \texttt{recv}\;c'\;\texttt{in}\,\texttt{send}\;c'\;(x+2)\}\}\,; & \textit{// Service thread} \\
\texttt{send}\;c\;18;\;\texttt{send}\;c\;20; & \textit{// Client thread} \\
\texttt{let}\,x := \texttt{recv}\;c\;\texttt{in}\;\texttt{let}\,y := \texttt{recv}\;c\;\texttt{in}\,x+y &
\end{array}
$$

Dependent separation protocols:

$$
\begin{array}{l}
c \rightarrowtail \mu rec.\,!\,(x:\mathbb{Z})\,\langle x\rangle\{\text{True}\}.\,?(y:\mathbb{Z})\,\langle y\rangle\{y=(x+2)\}.\,rec \qquad \text{and} \\
c' \rightarrowtail \mu rec.\,?(x:\mathbb{Z})\,\langle x\rangle\{\text{True}\}.\,!\,(y:\mathbb{Z})\,\langle y\rangle\{y=(x+2)\}.\,rec
\end{array}
$$

Subprotocol relation ($\sqsubseteq$) (Inspired by asynchronous session subtyping):

$$
\begin{array}{ll}
 & \mu rec.\,!\,(x:\mathbb{Z})\,\langle x\rangle\{\text{True}\}.\,?(y:\mathbb{Z})\,\langle y\rangle\{y=(x+2)\}.\,rec \\
\sqsubseteq & !\,(x_1:\mathbb{Z})\,\langle x_1\rangle\{\text{True}\}.\,?(y_1:\mathbb{Z})\,\langle y_1\rangle\{y_1=(x_1+2)\}. \\
 & \mu rec.\,!\,(x:\mathbb{Z})\,\langle x\rangle\{\text{True}\}.\,?(y:\mathbb{Z})\,\langle y\rangle\{y=(x+2)\}.\,rec
\end{array}
$$

Example program - Subprotocols (Actris 2.0)

Example program:

$$
\begin{array}{ll}
\texttt{let}\,(c, c') := \texttt{new\_chan}\,()\,\texttt{in} & \\
\texttt{fork}\,\{\text{loop}\,\{\texttt{let}\,x := \texttt{recv}\,c'\,\texttt{in}\,\texttt{send}\,c'\,(x+2)\}\}\,; & //\ \text{Service thread} \\
\texttt{send}\,c\,18;\ \texttt{send}\,c\,20; & //\ \text{Client thread} \\
\texttt{let}\,x := \texttt{recv}\,c\,\texttt{in}\,\texttt{let}\,y := \texttt{recv}\,c\,\texttt{in}\,x + y &
\end{array}
$$

Dependent separation protocols:

$$
\begin{array}{l}
c \rightarrowtail \mu rec.\,\mathbf{!}\,(x:\mathbb{Z})\,\langle x\rangle\{\text{True}\}.\,\mathbf{?}(y:\mathbb{Z})\,\langle y\rangle\{y = (x+2)\}.\,rec \qquad \texttt{and} \\
c' \rightarrowtail \mu rec.\,\mathbf{?}(x:\mathbb{Z})\,\langle x\rangle\{\text{True}\}.\,\mathbf{!}\,(y:\mathbb{Z})\,\langle y\rangle\{y = (x+2)\}.\,rec
\end{array}
$$

Subprotocol relation ($\sqsubseteq$) (Inspired by asynchronous session subtyping):

$$
\begin{array}{ll}
 & \mu rec.\,\mathbf{!}\,(x:\mathbb{Z})\,\langle x\rangle\{\text{True}\}.\,\mathbf{?}(y:\mathbb{Z})\,\langle y\rangle\{y = (x+2)\}.\,rec \\
\sqsubseteq & \mathbf{!}\,(x_1:\mathbb{Z})\,\langle x_1\rangle\{\text{True}\}.\,\mathbf{?}(y_1:\mathbb{Z})\,\langle y_1\rangle\{y_1 = (x_1+2)\}. \\
 & \mathbf{!}\,(x_2:\mathbb{Z})\,\langle x_2\rangle\{\text{True}\}.\,\mathbf{?}(y_2:\mathbb{Z})\,\langle y_2\rangle\{y_2 = (x_2+2)\}. \\
 & \mu rec.\,\mathbf{!}\,(x:\mathbb{Z})\,\langle x\rangle\{\text{True}\}.\,\mathbf{?}(y:\mathbb{Z})\,\langle y\rangle\{y = (x+2)\}.\,rec
\end{array}
$$

Example program - Subprotocols (Actris 2.0)

Example program:

```
let (c, c') := new_chan () in
fork {loop {let x := recv c' in send c' (x + 2)}} ;   // Service thread
send c 18; send c 20;                                 // Client thread
let x := recv c in let y := recv c in x + y
```

Dependent separation protocols:

$$c \rightarrowtail \mu rec.\, !\,(x:\mathbb{Z})\,\langle x\rangle\{\mathrm{True}\}.\, ?(y:\mathbb{Z})\,\langle y\rangle\{y = (x+2)\}.\, rec \qquad \texttt{and}$$
$$c' \rightarrowtail \mu rec.\, ?(x:\mathbb{Z})\,\langle x\rangle\{\mathrm{True}\}.\, !\,(y:\mathbb{Z})\,\langle y\rangle\{y = (x+2)\}.\, rec$$

Subprotocol relation ($\sqsubseteq$) (Inspired by asynchronous session subtyping):

$$\begin{aligned} & \mu rec.\, !\,(x:\mathbb{Z})\,\langle x\rangle\{\mathrm{True}\}.\, ?(y:\mathbb{Z})\,\langle y\rangle\{y = (x+2)\}.\, rec \\ \sqsubseteq \quad & !\,(x_1:\mathbb{Z})\,\langle x_1\rangle\{\mathrm{True}\}.\, ?(y_1:\mathbb{Z})\,\langle y_1\rangle\{y_1 = (x_1+2)\}. \\ & !\,(x_2:\mathbb{Z})\,\langle x_2\rangle\{\mathrm{True}\}.\, ?(y_2:\mathbb{Z})\,\langle y_2\rangle\{y_2 = (x_2+2)\}. \\ & \mu rec.\, !\,(x:\mathbb{Z})\,\langle x\rangle\{\mathrm{True}\}.\, ?(y:\mathbb{Z})\,\langle y\rangle\{y = (x+2)\}.\, rec \end{aligned}$$

Example program - Subprotocols (Actris 2.0)

Example program:

$$
\begin{array}{ll}
\texttt{let}\,(c, c') := \texttt{new\_chan}\,()\;\texttt{in} & \\
\texttt{fork}\;\{\text{loop}\;\{\texttt{let}\,x := \texttt{recv}\;c'\;\texttt{in}\,\texttt{send}\;c'\;(x+2)\}\}\,; & // \text{ Service thread} \\
\texttt{send}\;c\;18;\;\texttt{send}\;c\;20; & // \text{ Client thread} \\
\texttt{let}\,x := \texttt{recv}\;c\;\texttt{in}\;\texttt{let}\,y := \texttt{recv}\;c\;\texttt{in}\,x+y &
\end{array}
$$

Dependent separation protocols:

$$
\begin{array}{rl}
c \rightarrowtail \mu rec.\,!\,(x:\mathbb{Z})\,\langle x\rangle\{\text{True}\}.\,?(y:\mathbb{Z})\,\langle y\rangle\{y=(x+2)\}.\,rec & \quad \texttt{and} \\
c' \rightarrowtail \mu rec.\,?(x:\mathbb{Z})\,\langle x\rangle\{\text{True}\}.\,!\,(y:\mathbb{Z})\,\langle y\rangle\{y=(x+2)\}.\,rec &
\end{array}
$$

Subprotocol relation ($\sqsubseteq$) (Inspired by asynchronous session subtyping):

$$
\begin{array}{rl}
& \mu rec.\,!\,(x:\mathbb{Z})\,\langle x\rangle\{\text{True}\}.\,?(y:\mathbb{Z})\,\langle y\rangle\{y=(x+2)\}.\,rec \\
\sqsubseteq & !\,(x_1:\mathbb{Z})\,\langle x_1\rangle\{\text{True}\}.\,!\,(x_2:\mathbb{Z})\,\langle x_2\rangle\{\text{True}\}. \\
& ?(y_1:\mathbb{Z})\,\langle y_1\rangle\{y_1=(x_1+2)\}.\,?(y_2:\mathbb{Z})\,\langle y_2\rangle\{y_2=(x_2+2)\}. \\
& \mu rec.\,!\,(x:\mathbb{Z})\,\langle x\rangle\{\text{True}\}.\,?(y:\mathbb{Z})\,\langle y\rangle\{y=(x+2)\}.\,rec
\end{array}
$$

Adequacy and implementation of Actris

Adequacy of Actris

If $\{\mathsf{True}\}\, e\, \{v.\, \varphi\ v\}$ is provable in Actris then:

- ☑ **Safety:** e will not crash
- ☑ **Functional correctness:** If e terminates with v, the postcondition $\varphi\ v$ holds

Implementation and model of Actris in Iris

Approach:

- Define the type of *prot* using the Iris recursive domain equation solver

Implementation and model of Actris in Iris

Approach:

- Define the type of *prot* using the Iris recursive domain equation solver
- Define operations and relations on *prot*, such as $\overline{prot}$ and $prot_1 \sqsubseteq prot_2$

Implementation and model of Actris in Iris

Approach:

- Define the type of *prot* using the Iris recursive domain equation solver
- Define operations and relations on *prot*, such as $\overline{\mathit{prot}}$ and $\mathit{prot}_1 \sqsubseteq \mathit{prot}_2$
- Implement `new_chan`, `send`, and `recv` on top of HeapLang

Implementation and model of Actris in Iris

Approach:

- Define the type of *prot* using the Iris recursive domain equation solver
- Define operations and relations on *prot*, such as $\overline{prot}$ and $prot_1 \sqsubseteq prot_2$
- Implement `new_chan`, `send`, and `recv` on top of HeapLang
- Define $c \rightarrowtail prot$ using Iris's invariants and ghost state mechanisms

Implementation and model of Actris in Iris

Approach:

- Define the type of *prot* using the Iris recursive domain equation solver
- Define operations and relations on *prot*, such as $\overline{prot}$ and $prot_1 \sqsubseteq prot_2$
- Implement `new_chan`, `send`, and `recv` on top of HeapLang
- Define $c \rightarrowtail prot$ using Iris's invariants and ghost state mechanisms
- Prove Actris's proof rules as lemmas in Iris

Implementation and model of Actris in Iris

Approach:

- Define the type of *prot* using the Iris recursive domain equation solver
- Define operations and relations on *prot*, such as $\overline{prot}$ and $prot_1 \sqsubseteq prot_2$
- Implement `new_chan`, `send`, and `recv` on top of HeapLang
- Define $c \rightarrowtail prot$ using Iris's invariants and ghost state mechanisms
- Prove Actris's proof rules as lemmas in Iris

Benefits:

☑ Actris's adequacy result is a corollary of Iris's adequacy

Implementation and model of Actris in Iris

Approach:

- Define the type of *prot* using the Iris recursive domain equation solver
- Define operations and relations on *prot*, such as $\overline{prot}$ and $prot_1 \sqsubseteq prot_2$
- Implement `new_chan`, `send`, and `recv` on top of HeapLang
- Define $c \rightarrowtail prot$ using Iris's invariants and ghost state mechanisms
- Prove Actris's proof rules as lemmas in Iris

Benefits:

- ☑ Actris's adequacy result is a corollary of Iris's adequacy
- ☑ Readily integrates with other concurrency mechanisms in Iris

Implementation and model of Actris in Iris

Approach:

- Define the type of *prot* using the Iris recursive domain equation solver
- Define operations and relations on *prot*, such as $\overline{prot}$ and $prot_1 \sqsubseteq prot_2$
- Implement `new_chan`, `send`, and `recv` on top of HeapLang
- Define $c \rightarrowtail prot$ using Iris's invariants and ghost state mechanisms
- Prove Actris's proof rules as lemmas in Iris

Benefits:

- ☑ Actris's adequacy result is a corollary of Iris's adequacy
- ☑ Readily integrates with other concurrency mechanisms in Iris
- ☑ Can readily reuse Iris's support for interactive proofs in Coq

Implementation and model of Actris in Iris

Approach:

- Define the type of *prot* using the Iris recursive domain equation solver
- Define operations and relations on *prot*, such as $\overline{prot}$ and $prot_1 \sqsubseteq prot_2$
- Implement `new_chan`, `send`, and `recv` on top of HeapLang
- Define $c \rightarrowtail prot$ using Iris's invariants and ghost state mechanisms
- Prove Actris's proof rules as lemmas in Iris

Benefits:

- ☑ Actris's adequacy result is a corollary of Iris's adequacy
- ☑ Readily integrates with other concurrency mechanisms in Iris
- ☑ Can readily reuse Iris's support for interactive proofs in Coq
- ☑ Small Coq development (~5000 lines in total)

More on Actris

Features:

- **Higher-order:** sending function closures
- **Delegation:** sending channels over channels
- **Branching:** protocols with choice
- Integration with other concurrency mechanisms of Iris

Case Studies:

- Various channel-based merge sort variants
- Channel-based load-balancing mapper
- A variant of map-reduce

Model:

- **Dependent separation protocols:** prot
- **Channel endpoint ownership:** $c \rightarrowtail \mathit{prot}$
- **Subprotocol relation:** $\mathit{prot}_1 \sqsubseteq \mathit{prot}_2$

<u>In the thesis and associated papers!</u>

Semantic Session Typing

Paper: CPP'21

Thesis: Chapter 4

joint work with

Daniël Louwrink, University of Amsterdam
Jesper Bengtson, IT University of Copenhagen
Robbert Krebbers, Radboud University

Problem

No formal connection between dependent separation protocols and session types

- Protocols merely designed in the style of session types

Problem

No formal connection between dependent separation protocols and session types

- Protocols merely designed in the style of session types

Lack of expressivity of existing session type systems

- Polymorphism, recursion, and subtyping have been studied individually
- No session type system that combines all three

Problem

No formal connection between dependent separation protocols and session types

- Protocols merely designed in the style of session types

Lack of expressivity of existing session type systems

- Polymorphism, recursion, and subtyping have been studied individually
- No session type system that combines all three

Ongoing effort of mechanising adequacy proofs for session type systems

- Results exist for simpler systems
- None exist for more expressive systems

Semantic Typing

Semantic Typing [Milner, Princeton Proof-Carrying Code project, RustBelt Project]

- Types are defined as predicates over values: $\mathsf{Z} \triangleq \lambda w.\, w \in \mathbb{Z}$

Semantic Typing

Semantic Typing [Milner, Princeton Proof-Carrying Code project, RustBelt Project]

- Types are defined as predicates over values: $\mathsf{Z} \triangleq \lambda w.\, w \in \mathbb{Z}$
- Typing judgement are defined as safety-capturing evaluation: $\Gamma \vDash e : A$

Semantic Typing

Semantic Typing [Milner, Princeton Proof-Carrying Code project, RustBelt Project]

- Types are defined as predicates over values: $\mathsf{Z} \triangleq \lambda w.\, w \in \mathbb{Z}$
- Typing judgement are defined as safety-capturing evaluation: $\Gamma \vDash e : A$
- Typing rules are proven as lemmas: $\vDash i : \mathsf{Z}$

Semantic Typing

Semantic Typing [Milner, Princeton Proof-Carrying Code project, RustBelt Project]

- Types are defined as predicates over values: $\mathsf{Z} \triangleq \lambda w.\, w \in \mathbb{Z}$
- Typing judgement are defined as safety-capturing evaluation: $\Gamma \vDash e : A$
- Typing rules are proven as lemmas: $\vDash i : \mathsf{Z} \quad \rightsquigarrow \quad i \in \mathbb{Z}$

Semantic Typing

Semantic Typing [Milner, Princeton Proof-Carrying Code project, RustBelt Project]

- Types are defined as predicates over values: $\mathsf{Z} \triangleq \lambda w.\, w \in \mathbb{Z}$
- Typing judgement are defined as safety-capturing evaluation: $\Gamma \vDash e : A$
- Typing rules are proven as lemmas: $\vDash i : \mathsf{Z} \quad \rightsquigarrow \quad i \in \mathbb{Z}$
- Adequacy is inherited from underlying logic

Semantic Typing using **Iris**

Semantic Typing [Milner, Princeton Proof-Carrying Code project, RustBelt Project]

- Types are defined as predicates over values: $\mathsf{Z} \triangleq \lambda w.\, w \in \mathbb{Z}$
- Typing judgement are defined as safety-capturing evaluation: $\Gamma \vDash e : A$
- Typing rules are proven as lemmas: $\vDash i : \mathsf{Z} \quad \rightsquigarrow \quad i \in \mathbb{Z}$
- Adequacy is inherited from underlying logic

Iris [Iris project]

- Semantic type system for HeapLang
- Mechanised in **Coq**

Semantic Typing using **Iris** and **Actris**

Semantic Typing [Milner, Princeton Proof-Carrying Code project, RustBelt Project]

- Types are defined as predicates over values: $\mathsf{Z} \triangleq \lambda w.\, w \in \mathbb{Z}$
- Typing judgement are defined as safety-capturing evaluation: $\Gamma \vDash e : A$
- Typing rules are proven as lemmas: $\vDash i : \mathsf{Z} \quad \rightsquigarrow \quad i \in \mathbb{Z}$
- Adequacy is inherited from underlying logic

Iris [Iris project]

- Semantic type system for HeapLang
- Mechanised in **Coq**

Actris [Hinrichsen et al., POPL'20]

- **Dependent separation protocols:** Session type-style logical protocols
- Mechanised in **Coq**

Semantic Session Types

Semantic session types are defined as dependent separation protocols:

$$
\begin{aligned}
\mathbf{!}A.\,S &\triangleq \mathbf{!}\,(v : \mathsf{Val})\,\langle v\rangle\{A\,v\}.\,S \qquad \texttt{chan}\ S \triangleq \lambda w.\, w \rightarrowtail S \\
\mathbf{?}A.\,S &\triangleq \mathbf{?}(v : \mathsf{Val})\,\langle v\rangle\{A\,v\}.\,S \\
\texttt{end} &\triangleq \mathsf{end}
\end{aligned}
$$

Semantic Session Types

Semantic session types are defined as dependent separation protocols:

$$
\begin{aligned}
\mathbf{!}A.\,S &\triangleq \mathbf{!}\,(v : \mathsf{Val})\,\langle v\rangle\{A\,v\}.\,S \qquad \texttt{chan}\ S \triangleq \lambda w.\, w \rightarrowtail S\\
\mathbf{?}A.\,S &\triangleq \mathbf{?}(v : \mathsf{Val})\,\langle v\rangle\{A\,v\}.\,S\\
\texttt{end} &\triangleq \mathsf{end}
\end{aligned}
$$

Typing judgement is defined in terms of the Hoare triple

Semantic Session Types

Semantic session types are defined as dependent separation protocols:

$$\begin{aligned} \mathbf{!}A.\,S &\triangleq \mathbf{!}\,(v : \mathsf{Val})\,\langle v\rangle\{A\,v\}.\,S \qquad \texttt{chan}\ S \triangleq \lambda w.\, w \rightarrowtail S \\ \mathbf{?}A.\,S &\triangleq \mathbf{?}(v : \mathsf{Val})\,\langle v\rangle\{A\,v\}.\,S \\ \mathtt{end} &\triangleq \mathtt{end} \end{aligned}$$

Typing judgement is defined in terms of the Hoare triple
Session typing rules are proven using the rules for dependent separation protocols

$$\begin{array}{rll} \Gamma \vDash \texttt{new\_chan}\ () & : \texttt{chan}\ S \times \texttt{chan}\ \overline{S} & \dashv \Gamma \\ \Gamma, c : \texttt{chan}\ (\mathbf{!}A.\,S), x : A \vDash \texttt{send}\ c\ x & : 1 & \dashv \Gamma, c : \texttt{chan}\ S \\ \Gamma, c : \texttt{chan}\ (\mathbf{?}A.\,S) \vDash \texttt{recv}\ c & : A & \dashv \Gamma, c : \texttt{chan}\ S \end{array}$$

Manual typing proofs of racy yet safe programs

Consider the following program and typing judgement:

$$\lambda c.\,(\texttt{recv}\ c \parallel \texttt{recv}\ c) : \texttt{chan}\ (\mathbf{?}\mathsf{Z}.\,\mathbf{?}\mathsf{Z}.\,\texttt{end}) \multimap (\mathsf{Z} \times \mathsf{Z})$$

Manual typing proofs of racy yet safe programs

Consider the following program and typing judgement:

$$\vdash \lambda c.\,(\texttt{recv}\ c \parallel \texttt{recv}\ c) : \mathtt{chan}\ (\mathbf{?}\mathsf{Z}.\,\mathbf{?}\mathsf{Z}.\,\mathtt{end}) \multimap (\mathsf{Z} \times \mathsf{Z}) \quad ✗$$

Manual typing proofs of racy yet safe programs

Consider the following program and typing judgement:

$$\vDash \lambda c.\, (\texttt{recv}\ c \parallel \texttt{recv}\ c) : \texttt{chan}\ (\mathbf{?}\mathsf{Z}.\, \mathbf{?}\mathsf{Z}.\, \texttt{end}) \multimap (\mathsf{Z} \times \mathsf{Z}) \quad ✓$$

Manual typing proofs of racy yet safe programs

Consider the following program and typing judgement:

$$\vDash \lambda c.\,(\texttt{recv}\ c \parallel \texttt{recv}\ c) : \texttt{chan}\ (\mathbf{?}\mathsf{Z}.\,\mathbf{?}\mathsf{Z}.\,\texttt{end}) \multimap (\mathsf{Z} \times \mathsf{Z}) \quad \checkmark$$

The judgement is just another lemma

Manual typing proofs of racy yet safe programs

Consider the following program and typing judgement:

$$\vDash \lambda c. \, (\texttt{recv}\ c \parallel \texttt{recv}\ c) : \texttt{chan}\ (\mathbf{?}\mathsf{Z}.\, \mathbf{?}\mathsf{Z}.\, \mathsf{end}) \multimap (\mathsf{Z} \times \mathsf{Z}) \quad \checkmark$$

The judgement is just another lemma provable by unfolding all type-level definitions

$$\{(c \rightarrowtail \mathbf{?}(v_1 : \mathsf{Val})\, \langle v_1 \rangle \{v_1 \in \mathbb{Z}\}.\, \mathbf{?}(v_2 : \mathsf{Val})\, \langle v_2 \rangle \{v_2 \in \mathbb{Z}\}.\, \mathsf{end})\}$$

$$(\texttt{recv}\ c \parallel \texttt{recv}\ c)$$

$$\{v.\, \exists v_1, v_2.\, (v = (v_1, v_2)) * \triangleright(v_1 \in \mathbb{Z}) * \triangleright(v_2 \in \mathbb{Z})\}$$

Manual typing proofs of racy yet safe programs

Consider the following program and typing judgement:

$$\vDash \lambda c.\, (\texttt{recv}\ c \parallel \texttt{recv}\ c) : \texttt{chan}\ (\mathbf{?}\mathsf{Z}.\, \mathbf{?}\mathsf{Z}.\, \mathsf{end}) \multimap (\mathsf{Z} \times \mathsf{Z}) \quad ✓$$

The judgement is just another lemma provable by unfolding all type-level definitions

$$\{(c \rightarrowtail \mathbf{?}(v_1 : \mathsf{Val})\, \langle v_1 \rangle \{v_1 \in \mathbb{Z}\}.\, \mathbf{?}(v_2 : \mathsf{Val})\, \langle v_2 \rangle \{v_2 \in \mathbb{Z}\}.\, \mathsf{end})\}$$
$$(\texttt{recv}\ c \parallel \texttt{recv}\ c)$$
$$\{v.\, \exists v_1, v_2.\, (v = (v_1, v_2)) * \triangleright(v_1 \in \mathbb{Z}) * \triangleright(v_2 \in \mathbb{Z})\}$$

Manual typing proofs of racy yet safe programs

Consider the following program and typing judgement:

$$\vDash \lambda c.\, (\texttt{recv}\ c \parallel \texttt{recv}\ c) : \texttt{chan}\ (\mathbf{?}\mathsf{Z}.\, \mathbf{?}\mathsf{Z}.\, \mathsf{end}) \multimap (\mathsf{Z} \times \mathsf{Z}) \quad ✓$$

The judgement is just another lemma provable by unfolding all type-level definitions

$$\{(c \rightarrowtail \mathbf{?}(v_1 : \mathsf{Val})\, \langle v_1 \rangle \{v_1 \in \mathbb{Z}\}.\, \mathbf{?}(v_2 : \mathsf{Val})\, \langle v_2 \rangle \{v_2 \in \mathbb{Z}\}.\, \mathsf{end})\}$$

$$(\texttt{recv}\ c \parallel \texttt{recv}\ c)$$

$$\{v.\, \exists v_1, v_2.\, (v = (v_1, v_2)) * \triangleright(v_1 \in \mathbb{Z}) * \triangleright(v_2 \in \mathbb{Z})\}$$

Manual typing proofs of racy yet safe programs

Consider the following program and typing judgement:

$$\vDash \lambda c.\, (\texttt{recv}\ c \parallel \texttt{recv}\ c) : \texttt{chan}\ (\mathbf{?}\mathsf{Z}.\, \mathbf{?}\mathsf{Z}.\, \mathsf{end}) \multimap (\mathsf{Z} \times \mathsf{Z}) \quad \checkmark$$

The judgement is just another lemma provable by unfolding all type-level definitions

$$\{(c \rightarrowtail \mathbf{?}(v_1 : \mathsf{Val})\, \langle v_1 \rangle \{v_1 \in \mathbb{Z}\}.\, \mathbf{?}(v_2 : \mathsf{Val})\, \langle v_2 \rangle \{v_2 \in \mathbb{Z}\}.\, \mathsf{end})\}$$
$$(\texttt{recv}\ c \parallel \texttt{recv}\ c)$$
$$\{v.\, \exists v_1, v_2.\, (v = (v_1, v_2)) * \triangleright(v_1 \in \mathbb{Z}) * \triangleright(v_2 \in \mathbb{Z})\}$$

Using Iris's ghost state machinery!

Manual typing proofs of racy yet safe programs

Consider the following program and typing judgement:

$$\vDash \lambda c.\,(\mathtt{recv}\ c \parallel \mathtt{recv}\ c) : \mathtt{chan}\ (\mathbf{?}\mathsf{Z}.\,\mathbf{?}\mathsf{Z}.\,\mathsf{end}) \multimap (\mathsf{Z} \times \mathsf{Z}) \quad ✓$$

The judgement is just another lemma provable by unfolding all type-level definitions

$$\{(c \rightarrowtail \mathbf{?}(v_1 : \mathsf{Val})\,\langle v_1\rangle\{v_1 \in \mathbb{Z}\}.\,\mathbf{?}(v_2 : \mathsf{Val})\,\langle v_2\rangle\{v_2 \in \mathbb{Z}\}.\,\mathsf{end})\}$$
$$(\mathtt{recv}\ c \parallel \mathtt{recv}\ c)$$
$$\{v.\,\exists v_1, v_2.\,(v = (v_1, v_2)) * \triangleright(v_1 \in \mathbb{Z}) * \triangleright(v_2 \in \mathbb{Z})\}$$

Using Iris's ghost state machinery! Beyond the scope of this presentation

More on the semantic session type system

Features:

- Term and session type equi-recursion
- Term and session type polymorphism
- Term and (asynchronous) session type subtyping
- Unique and shared reference types, copyable types, lock types
- Integration of racy yet safe programs

Case Study:

- Racy yet safe message-passing-based producer-consumer

In the thesis and associated paper!

Future work

Future work

Future Work

- Multi-party communication via multi-party dependent separation protocols (based on [Honda *et al.*, POPL'08])
- Deadlock and resource-leak-freedom (based on ongoing work by Jules Jacobs)
- Proof automation via refinedC-style semantic refinement session types [Sammler *et al.*, PLDI'21]
- Specifications for TCP-based communication in distributed systems based on dependent separation protocols

$$
\begin{aligned}
&\textbf{!}\,\langle\text{“Thank you”}\rangle\{\texttt{ActrisKnowledge}\}.\\
&\mu \mathit{rec}.\,\textbf{?}(q : \texttt{Question})\,\langle q\rangle\{\texttt{AboutActris}\ q\}.\\
&\qquad \textbf{!}\,(a : \texttt{Answer})\,\langle a\rangle\{\texttt{Insightful}\ q\ a\}.\,\mathit{rec}
\end{aligned}
$$