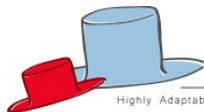


A Relational Trace Logic for Simple Hierarchical Actor-Based Component Systems

Ilham W. Kurnia, Arnd Poetzsch-Heffter
{ilham,poetzsch}@cs.uni-kl.de

AGERE!
Tucson, Arizona, USA

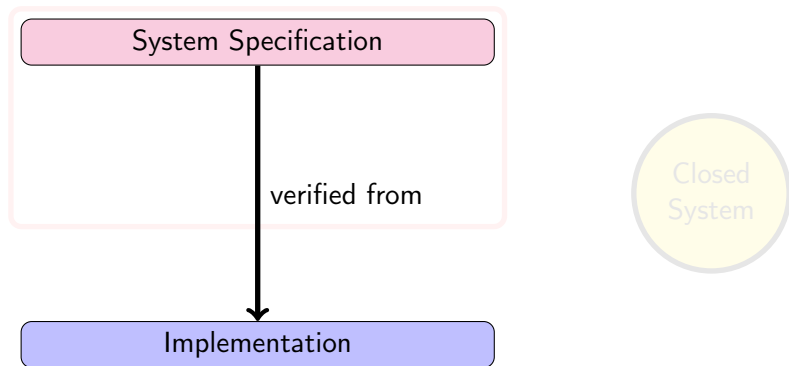
October 21–22, 2012



HATS

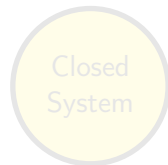
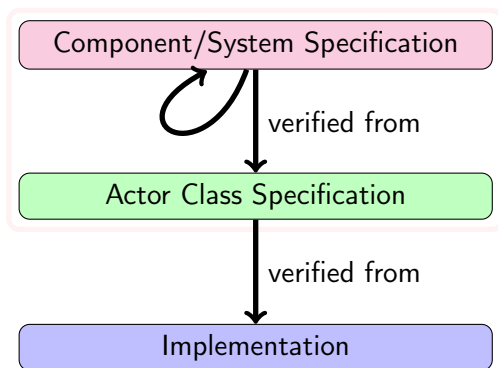
Highly Adaptable and Trustworthy Software using Formal Models

Motivation



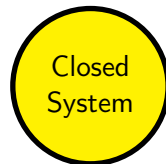
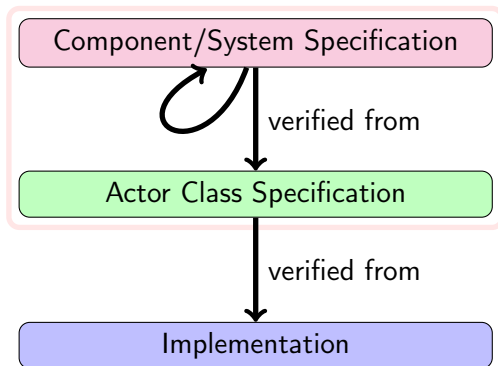
[ABS] See <http://hats-project.eu>, in particular <http://tools.hats-project.eu>

Motivation



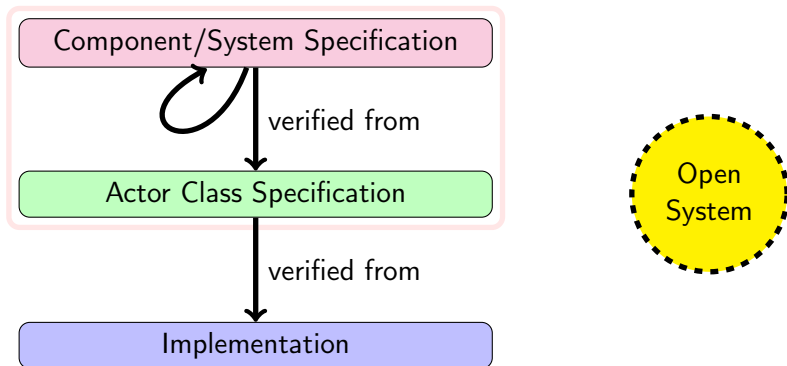
[ABS] See <http://hats-project.eu>, in particular <http://tools.hats-project.eu>

Motivation



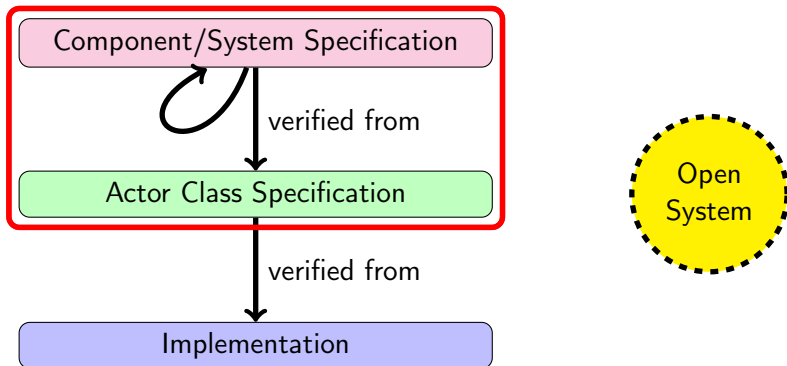
[ABS] See <http://hats-project.eu>, in particular <http://tools.hats-project.eu>

Motivation



[ABS] See <http://hats-project.eu>, in particular <http://tools.hats-project.eu>

Motivation



[ABS] See <http://hats-project.eu>, in particular <http://tools.hats-project.eu>

Example [Arts and Dam, 1999]

- ▶ Server performs a database query for client by delegating it to workers.
- ▶ Server triggers merging of partial results.
- ▶ If the query is large, workers delegate part of it to other worker.
- ▶ Workers merge their computation with partial result and pass it on.

Client

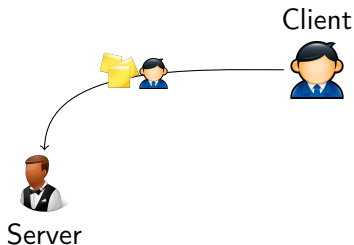


Server

[1999] T. Arts, M. Dam: Verifying a Distributed Database Lookup Manager Written in Erlang. World Congress on Formal Methods 1999: 682–700

Example [Arts and Dam, 1999]

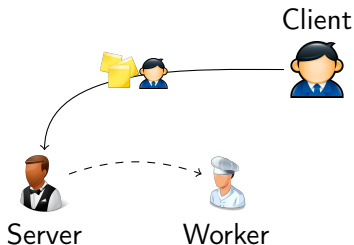
- ▶ Server performs a database query for client by delegating it to workers.
- ▶ Server triggers merging of partial results.
- ▶ If the query is large, workers delegate part of it to other worker.
- ▶ Workers merge their computation with partial result and pass it on.



[1999] T. Arts, M. Dam: Verifying a Distributed Database Lookup Manager Written in Erlang. World Congress on Formal Methods 1999: 682–700

Example [Arts and Dam, 1999]

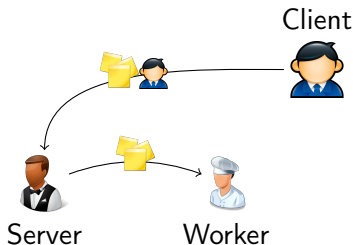
- ▶ Server performs a database query for client by delegating it to workers.
- ▶ Server triggers merging of partial results.
- ▶ If the query is large, workers delegate part of it to other worker.
- ▶ Workers merge their computation with partial result and pass it on.



[1999] T. Arts, M. Dam: Verifying a Distributed Database Lookup Manager Written in Erlang. World Congress on Formal Methods 1999: 682–700

Example [Arts and Dam, 1999]

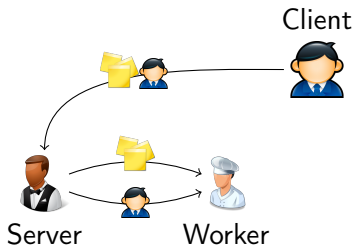
- ▶ Server performs a database query for client by delegating it to workers.
- ▶ Server triggers merging of partial results.
- ▶ If the query is large, workers delegate part of it to other worker.
- ▶ Workers merge their computation with partial result and pass it on.



[1999] T. Arts, M. Dam: Verifying a Distributed Database Lookup Manager Written in Erlang. World Congress on Formal Methods 1999: 682–700

Example [Arts and Dam, 1999]

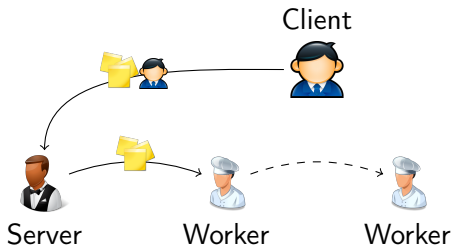
- ▶ Server performs a database query for client by delegating it to workers.
- ▶ Server triggers merging of partial results.
- ▶ If the query is large, workers delegate part of it to other worker.
- ▶ Workers merge their computation with partial result and pass it on.



[1999] T. Arts, M. Dam: Verifying a Distributed Database Lookup Manager Written in Erlang. World Congress on Formal Methods 1999: 682–700

Example [Arts and Dam, 1999]

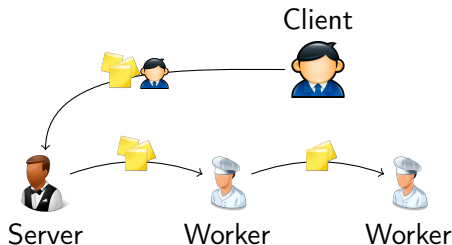
- ▶ Server performs a database query for client by delegating it to workers.
- ▶ Server triggers merging of partial results.
- ▶ If the query is large, workers delegate part of it to other worker.
- ▶ Workers merge their computation with partial result and pass it on.



[1999] T. Arts, M. Dam: Verifying a Distributed Database Lookup Manager Written in Erlang. World Congress on Formal Methods 1999: 682–700

Example [Arts and Dam, 1999]

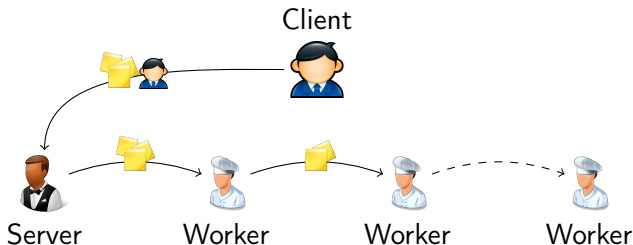
- ▶ Server performs a database query for client by delegating it to workers.
- ▶ Server triggers merging of partial results.
- ▶ If the query is large, workers delegate part of it to other worker.
- ▶ Workers merge their computation with partial result and pass it on.



[1999] T. Arts, M. Dam: Verifying a Distributed Database Lookup Manager Written in Erlang. World Congress on Formal Methods 1999: 682–700

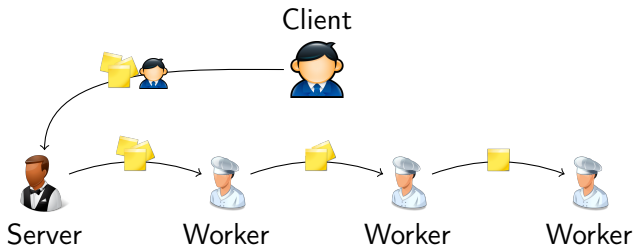
Example [Arts and Dam, 1999]

- ▶ Server performs a database query for client by delegating it to workers.
- ▶ Server triggers merging of partial results.
- ▶ If the query is large, workers delegate part of it to other worker.
- ▶ Workers merge their computation with partial result and pass it on.



[1999] T. Arts, M. Dam: Verifying a Distributed Database Lookup Manager Written in Erlang. World Congress on Formal Methods 1999: 682–700

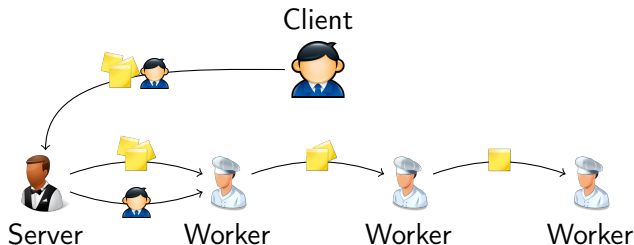
- ▶ Server performs a database query for client by delegating it to workers.
- ▶ Server triggers merging of partial results.
- ▶ If the query is large, workers delegate part of it to other worker.
- ▶ Workers merge their computation with partial result and pass it on.



[1999] T. Arts, M. Dam: Verifying a Distributed Database Lookup Manager Written in Erlang. World Congress on Formal Methods 1999: 682–700

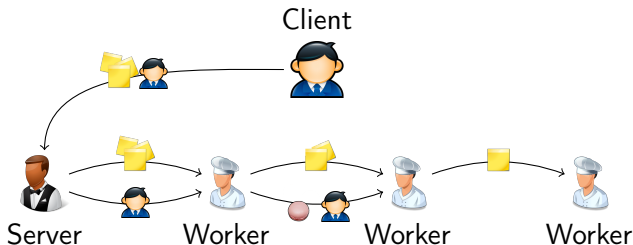
Example [Arts and Dam, 1999]

- ▶ Server performs a database query for client by delegating it to workers.
- ▶ Server triggers merging of partial results.
- ▶ If the query is large, workers delegate part of it to other worker.
- ▶ Workers merge their computation with partial result and pass it on.



[1999] T. Arts, M. Dam: Verifying a Distributed Database Lookup Manager Written in Erlang. World Congress on Formal Methods 1999: 682–700

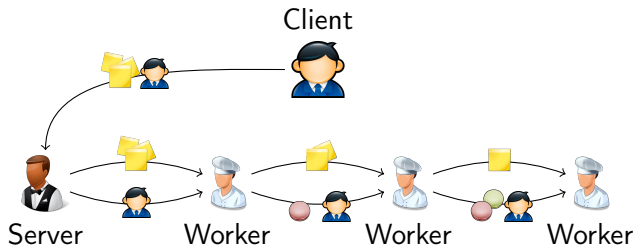
- ▶ Server performs a database query for client by delegating it to workers.
- ▶ Server triggers merging of partial results.
- ▶ If the query is large, workers delegate part of it to other worker.
- ▶ Workers merge their computation with partial result and pass it on.



[1999] T. Arts, M. Dam: Verifying a Distributed Database Lookup Manager Written in Erlang. World Congress on Formal Methods 1999: 682–700

Example [Arts and Dam, 1999]

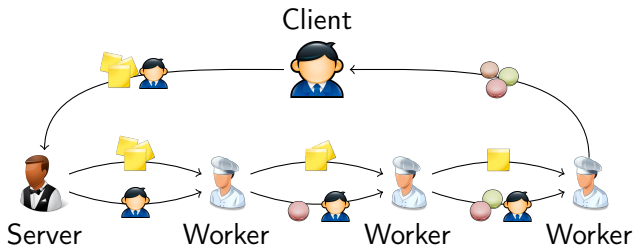
- ▶ Server performs a database query for client by delegating it to workers.
- ▶ Server triggers merging of partial results.
- ▶ If the query is large, workers delegate part of it to other worker.
- ▶ Workers merge their computation with partial result and pass it on.



[1999] T. Arts, M. Dam: Verifying a Distributed Database Lookup Manager Written in Erlang. World Congress on Formal Methods 1999: 682–700

Example [Arts and Dam, 1999]

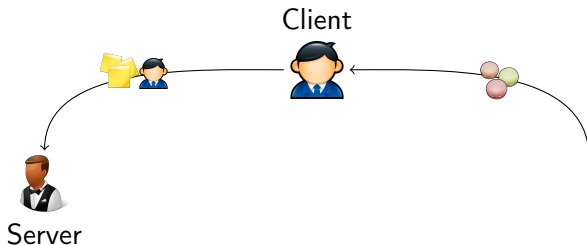
- ▶ Server performs a database query for client by delegating it to workers.
- ▶ Server triggers merging of partial results.
- ▶ If the query is large, workers delegate part of it to other worker.
- ▶ Workers merge their computation with partial result and pass it on.



[1999] T. Arts, M. Dam: Verifying a Distributed Database Lookup Manager Written in Erlang. World Congress on Formal Methods 1999: 682–700

Example [Arts and Dam, 1999]

- ▶ Server performs a database query for client by delegating it to workers.
- ▶ Server triggers merging of partial results.
- ▶ If the query is large, workers delegate part of it to other worker.
- ▶ Workers merge their computation with partial result and pass it on.



[1999] T. Arts, M. Dam: Verifying a Distributed Database Lookup Manager Written in Erlang. World Congress on Formal Methods 1999: 682–700

Is It True That

whenever a server receives a request, there will be a correct response?

How can we **formally** prove that it holds in all environments?

Goal and Overview

Goal

Demonstrate an approach that handles open systems

- 1 Trace semantics for actor-based components
- 2 Interface specifications for actor classes and components that interact with their environment
- 3 Logic for hierarchical verification

Overview

- ▶ Semantics of actor systems
- ▶ Specification of classes and components
- ▶ Verification and proof rules

Goal and Overview

Goal

Demonstrate an approach that handles open systems

- 1 Trace semantics for actor-based components
- 2 Interface specifications for actor classes and components that interact with their environment
- 3 Logic for hierarchical verification

Overview

- ▶ Semantics of actor systems
- ▶ Specification of classes and components
- ▶ Verification and proof rules

Events and Traces

- ▶ Trace: A sequence of *observable* events

Observable events:

- Creation of new actors ($a \rightarrow b := \text{new } C$)
- Message send ($a \rightarrow b.\text{msg}(\bar{p})$)

- ▶ Traces abstract from internal representation
- ▶ Traces are used as semantic foundation for specifications

Example

- ▶ $b \rightarrow s := \text{new Server} \cdot c \rightarrow s.\text{serve}(c, \text{query})$
- ▶ $s \rightarrow w := \text{new Worker} \cdot s \rightarrow w.\text{do}(\text{query}) \cdot s \rightarrow w.\text{propagate}(\text{null}, c) \cdot w \rightarrow c.\text{response}(\text{compute}(\text{query}))$

Events and Traces

- ▶ Trace: A sequence of *observable* events

Observable events:

- Creation of new actors ($a \rightarrow b := \text{new } C$)
- Message send ($a \rightarrow b.\text{msg}(\bar{p})$)

- ▶ Traces abstract from internal representation

- ▶ Traces are used as semantic foundation for specifications

Example

- ▶ $b \rightarrow s := \text{new Server} \cdot c \rightarrow s.\text{serve}(c, \text{query})$
- ▶ $s \rightarrow w := \text{new Worker} \cdot s \rightarrow w.\text{do}(\text{query}) \cdot s \rightarrow w.\text{propagate}(\text{null}, c) \cdot w \rightarrow c.\text{response}(\text{compute}(\text{query}))$

Events and Traces

- ▶ Trace: A sequence of *observable* events

Observable events:

- Creation of new actors ($a \rightarrow b := \text{new } C$)
- Message send ($a \rightarrow b.\text{msg}(\bar{p})$)

- ▶ Traces abstract from internal representation
- ▶ Traces are used as semantic foundation for specifications

Example

- ▶ $b \rightarrow s := \text{new Server} \cdot c \rightarrow s.\text{serve}(c, \text{query})$
- ▶ $s \rightarrow w := \text{new Worker} \cdot s \rightarrow w.\text{do}(\text{query}) \cdot s \rightarrow w.\text{propagate}(\text{null}, c) \cdot w \rightarrow c.\text{response}(\text{compute}(\text{query}))$

Events and Traces

- ▶ Trace: A sequence of *observable* events

Observable events:

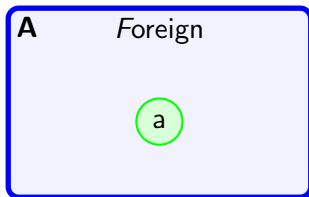
- Creation of new actors ($a \rightarrow b := \text{new } C$)
- Message send ($a \rightarrow b.\text{msg}(\bar{p})$)

- ▶ Traces abstract from internal representation
- ▶ Traces are used as semantic foundation for specifications

Example

- ▶ $b \rightarrow s := \text{new Server} \cdot c \rightarrow s.\text{serve}(c, \text{query})$
- ▶ $s \rightarrow w := \text{new Worker} \cdot s \rightarrow w.\text{do}(\text{query}) \cdot s \rightarrow w.\text{propagate}(\text{null}, c) \cdot w \rightarrow c.\text{response}(\text{compute}(\text{query}))$

Trace Semantics (1)



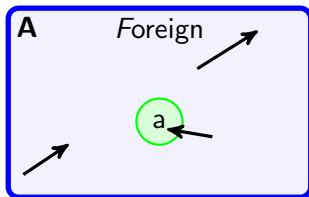
- ▶ The semantics of a of class C is a **trace set**
 - Run for all contexts $\rightsquigarrow$ produces the traces
 - Project the traces to events “relevant” to a
 - Collect all the traces

Remark: validity/well-formedness of the trace sets

Example

$b \rightarrow s := \text{new Server} \cdot c \rightarrow s.\text{serve}(c, \text{query}) \cdot s \rightarrow w := \text{new Worker} \cdot$
 $s \rightarrow w.\text{do}(\text{query}) \cdot s \rightarrow w.\text{propagate}(\text{null}, c) \in \text{Traces}(\text{Server})$
 where s is the actor of class **Server**

Trace Semantics (1)



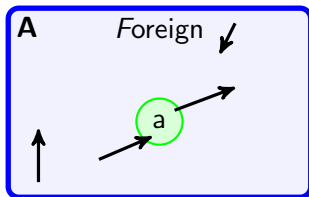
- ▶ The semantics of a of class C is a trace set
 - Run for all contexts $\rightsquigarrow$ produces the traces
 - Project the traces to events “relevant” to a
 - Collect all the traces

Remark: validity/well-formedness of the trace sets

Example

$b \rightarrow s := \text{new Server} \cdot c \rightarrow s.\text{serve}(c, \text{query}) \cdot s \rightarrow w := \text{new Worker} \cdot$
 $s \rightarrow w.\text{do}(\text{query}) \cdot s \rightarrow w.\text{propagate}(\text{null}, c) \in \text{Traces}(\text{Server})$
 where s is the actor of class **Server**

Trace Semantics (1)



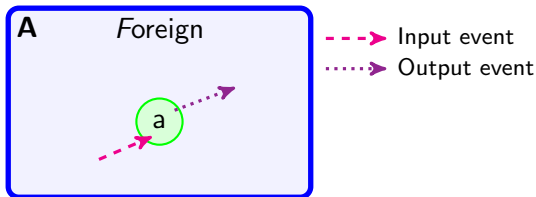
- ▶ The semantics of a of class C is a trace set
 - Run for all contexts $\rightsquigarrow$ produces the traces
 - Project the traces to events “relevant” to a
 - Collect all the traces

Remark: validity/well-formedness of the trace sets

Example

$b \rightarrow s := \text{new Server} \cdot c \rightarrow s.\text{serve}(c, \text{query}) \cdot s \rightarrow w := \text{new Worker} \cdot$
 $s \rightarrow w.\text{do}(\text{query}) \cdot s \rightarrow w.\text{propagate}(\text{null}, c) \in \text{Traces}(\text{Server})$
 where s is the actor of class `Server`

Trace Semantics (1)



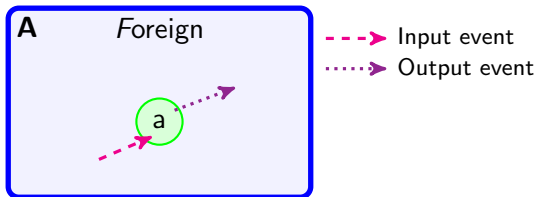
- ▶ The semantics of a of class C is a trace set
 - Run for all contexts $\rightsquigarrow$ produces the traces
 - Project the traces to events “relevant” to a
 - Collect all the traces

Remark: validity/well-formedness of the trace sets

Example

$b \rightarrow s := \text{new Server} \cdot c \rightarrow s.\text{serve}(c, \text{query}) \cdot s \rightarrow w := \text{new Worker} \cdot$
 $s \rightarrow w.\text{do}(\text{query}) \cdot s \rightarrow w.\text{propagate}(\text{null}, c) \in \text{Traces}(\text{Server})$
 where s is the actor of class **Server**

Trace Semantics (1)



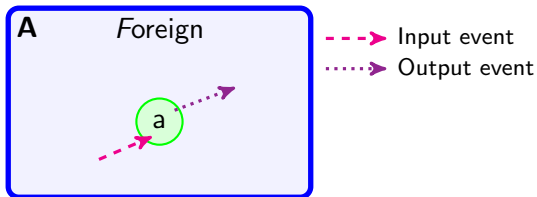
- ▶ The semantics of a of class C is a trace set
 - Run for all contexts $\rightsquigarrow$ produces the traces
 - Project the traces to events “relevant” to a
 - Collect all the traces

Remark: validity/well-formedness of the trace sets

Example

$b \rightarrow s := \text{new Server} \cdot c \rightarrow s.\text{serve}(c, \text{query}) \cdot s \rightarrow w := \text{new Worker} \cdot$
 $s \rightarrow w.\text{do}(\text{query}) \cdot s \rightarrow w.\text{propagate}(\text{null}, c) \in \text{Traces}(\text{Server})$
 where s is the actor of class **Server**

Trace Semantics (1)



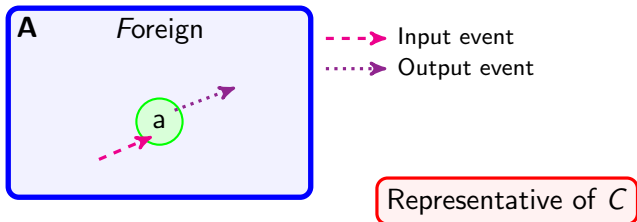
- ▶ The semantics of a of class C is a trace set
 - Run for all contexts $\rightsquigarrow$ produces the traces
 - Project the traces to events “relevant” to a
 - Collect all the traces

Remark: validity/well-formedness of the trace sets

Example

$b \rightarrow s := \text{new Server} \cdot c \rightarrow s.\text{serve}(c, \text{query}) \cdot s \rightarrow w := \text{new Worker} \cdot$
 $s \rightarrow w.\text{do}(\text{query}) \cdot s \rightarrow w.\text{propagate}(\text{null}, c) \in \text{Traces}(\text{Server})$
 where s is the actor of class **Server**

Trace Semantics (1)



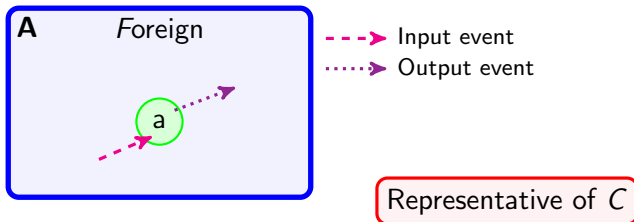
- ▶ The semantics of a of class C is a trace set $Traces(C)$
 - Run for **all contexts** $\rightsquigarrow$ produces the traces
 - Project the traces to events “relevant” to a
 - Collect all the traces

Remark: validity/well-formedness of the trace sets

Example

$b \rightarrow s := \text{new Server} \cdot c \rightarrow s.\text{serve}(c, \text{query}) \cdot s \rightarrow w := \text{new Worker} \cdot$
 $s \rightarrow w.\text{do}(\text{query}) \cdot s \rightarrow w.\text{propagate}(\text{null}, c) \in Traces(\text{Server})$
 where s is the actor of class **Server**

Trace Semantics (1)



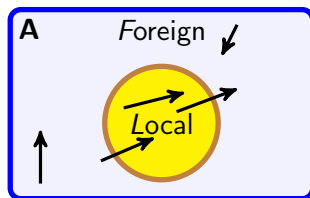
- ▶ The semantics of a of class C is a trace set $Traces(C)$
 - Run for all contexts $\rightsquigarrow$ produces the traces
 - Project the traces to events “relevant” to a
 - Collect all the traces

Remark: validity/well-formedness of the trace sets

Example

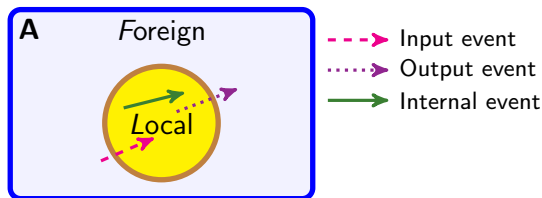
$b \rightarrow s := \text{new Server} \cdot c \rightarrow s.\text{serve}(c, \text{query}) \cdot s \rightarrow w := \text{new Worker} \cdot$
 $s \rightarrow w.\text{do}(\text{query}) \cdot s \rightarrow w.\text{propagate}(\text{null}, c) \in Traces(\text{Server})$
 where s is the actor of class **Server**

Trace Semantics (2)



- ▶ The semantics of a set of local actor L is a trace set
 - Run for all contexts $\rightsquigarrow$ produces the traces
 - Project the traces to events “relevant” to L
 - Collect all the traces

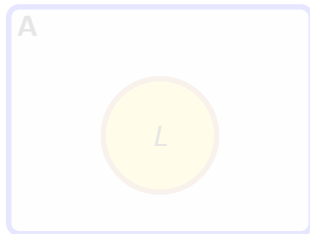
Trace Semantics (2)



- ▶ The semantics of a set of local actor L is a trace set
 - Run for all contexts $\rightsquigarrow$ produces the traces
 - Project the traces to events “relevant” to L
 - Collect all the traces

Components

- ▶ Need an abstraction between single classes and the whole system
- ▶ Hierarchical reasoning:
 - from classes to small components
 - from small components to larger components
 - hiding internal behavior



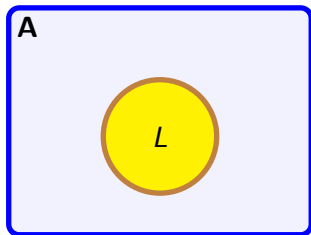
- ▶ L is a **component** if *no* other actors outside L are created by an actor in L .
- ▶ L can be obtained from the actor creation tree.

Example



Components

- ▶ Need an abstraction between single classes and the whole system
- ▶ Hierarchical reasoning:
 - from classes to small components
 - from small components to larger components
 - hiding internal behavior



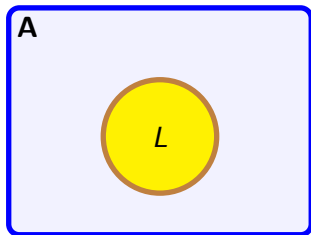
- ▶ L is a **component** if *no* other actors outside L are created by an actor in L .
- ▶ L can be obtained from the actor creation tree.

Example



Components

- ▶ Need an abstraction between single classes and the whole system
- ▶ Hierarchical reasoning:
 - from classes to small components
 - from small components to larger components
 - hiding internal behavior



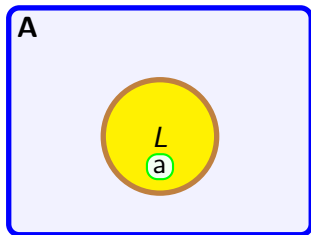
- ▶ L is a **component** if *no* other actors outside L are created by an actor in L .
- ▶ L can be obtained from the actor creation tree.

Example



Components

- ▶ Need an abstraction between single classes and the whole system
- ▶ Hierarchical reasoning:
 - from classes to small components
 - from small components to larger components
 - hiding internal behavior



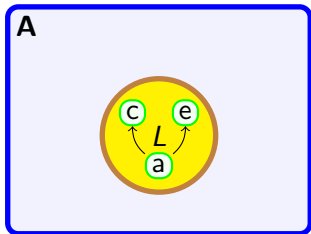
- ▶ L is a **component** if *no* other actors outside L are created by an actor in L .
- ▶ L can be obtained from the actor creation tree.

Example



Components

- ▶ Need an abstraction between single classes and the whole system
- ▶ Hierarchical reasoning:
 - from classes to small components
 - from small components to larger components
 - hiding internal behavior



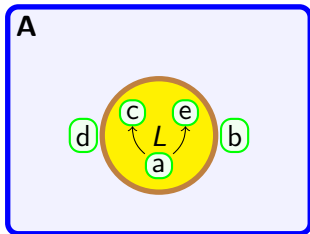
- ▶ L is a **component** if *no* other actors outside L are created by an actor in L .
- ▶ L can be obtained from the actor creation tree.

Example



Components

- ▶ Need an abstraction between single classes and the whole system
- ▶ Hierarchical reasoning:
 - from classes to small components
 - from small components to larger components
 - hiding internal behavior



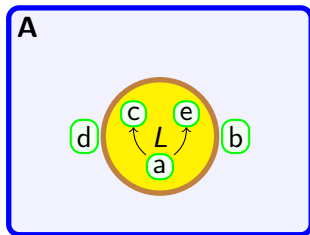
- ▶ L is a **component** if *no* other actors outside L are created by an actor in L .
- ▶ L can be obtained from the actor creation tree.

Example



Components

- ▶ Need an abstraction between single classes and the whole system
- ▶ Hierarchical reasoning:
 - from classes to small components
 - from small components to larger components
 - hiding internal behavior



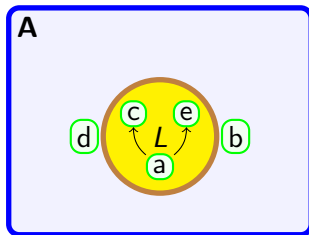
- ▶ L is a **component** if *no* other actors outside L are created by an actor in L .
- ▶ L can be obtained from the actor creation tree.

Example



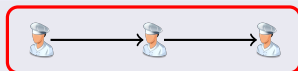
Components

- ▶ Need an abstraction between single classes and the whole system
- ▶ Hierarchical reasoning:
 - from classes to small components
 - from small components to larger components
 - hiding internal behavior



- ▶ L is a **component** if *no* other actors outside L are created by an actor in L .
- ▶ L can be obtained from the actor creation tree.

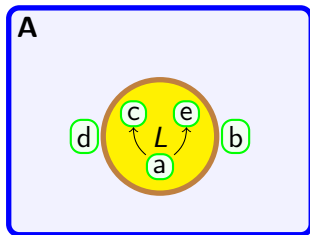
Example



$$L = \{w_1, w_2, w_3\}$$

Components

- ▶ Need an abstraction between single classes and the whole system
- ▶ Hierarchical reasoning:
 - from classes to small components
 - from small components to larger components
 - hiding internal behavior



- ▶ L is a **component** if *no* other actors outside L are created by an actor in L .
- ▶ L can be obtained from the actor creation tree.

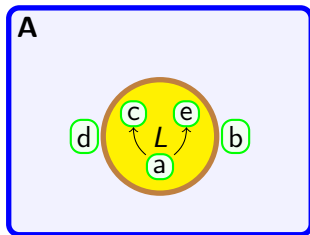
Example



$$L = \{w_1, w_2, w_3\}$$

Components

- ▶ Need an abstraction between single classes and the whole system
- ▶ Hierarchical reasoning:
 - from classes to small components
 - from small components to larger components
 - hiding internal behavior



- ▶ L is a **component** if *no* other actors outside L are created by an actor in L .
- ▶ L can be obtained from the actor creation tree.

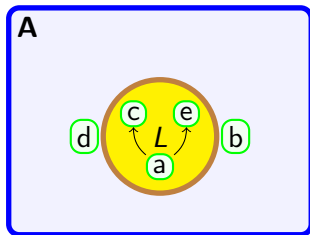
Example



$$L = \{w_1, w_2, w_3\}$$

Components

- ▶ Need an abstraction between single classes and the whole system
- ▶ Hierarchical reasoning:
 - from classes to small components
 - from small components to larger components
 - hiding internal behavior



- ▶ L is a **component** if *no* other actors outside L are created by an actor in L .
- ▶ L can be obtained from the actor creation tree.

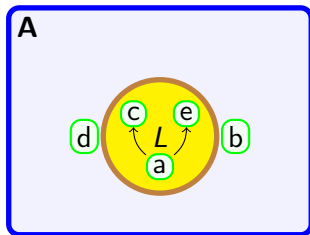
Example



$$L = \{s, w_1, w_2, w_3\}$$

Components

- ▶ Need an abstraction between single classes and the whole system
- ▶ Hierarchical reasoning:
 - from classes to small components
 - from small components to larger components
 - hiding internal behavior



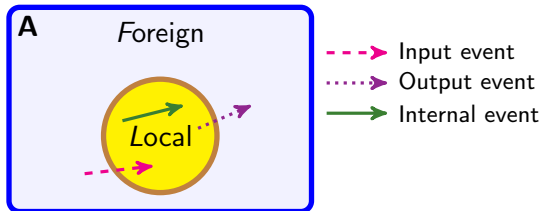
- ▶ L is a **component** if *no* other actors outside L are created by an actor in L .
- ▶ L can be obtained from the actor creation tree.

Example



$$L = \{s, w_1, w_2, w_3, \dots\}$$

Boxing a Component



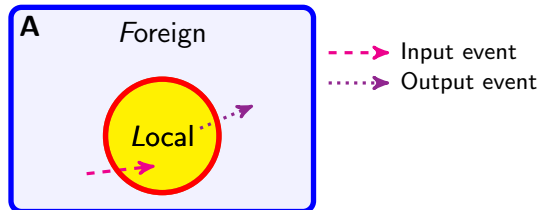
- ▶ **Hide** internal events: **boxing**
- ▶ Each trace is projected to the foreign actors
- ▶ Because actors of the same class behave the same, the component can be represented by the *initial* actor class: $[C]$.

Example

$b \rightarrow s := \text{new Server} \cdot c \rightarrow s.\text{serve}(c, q) \cdot w \rightarrow c.\text{response}(\text{compute}(q))$
 $\in \text{Traces}([\text{Server}])$

where s and w are **Server** and **Worker** actors, respectively.

Boxing a Component



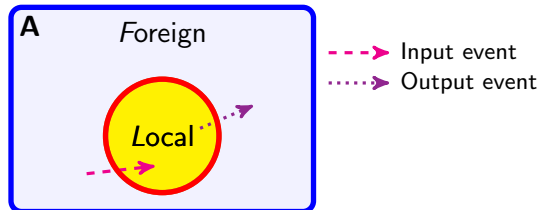
- ▶ **Hide** internal events: **boxing**
- ▶ Each trace is projected to the foreign actors
- ▶ Because actors of the same class behave the same, the component can be represented by the *initial* actor class: $[C]$.

Example

$b \rightarrow s := \text{new Server} \cdot c \rightarrow s.\text{serve}(c, q) \cdot w \rightarrow c.\text{response}(\text{compute}(q))$
 $\in \text{Traces}([\text{Server}])$

where s and w are **Server** and **Worker** actors, respectively.

Boxing a Component

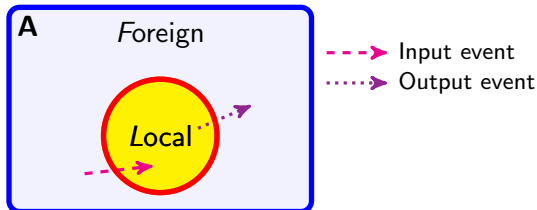


- ▶ Hide internal events: **boxing**
- ▶ Each trace is projected to the foreign actors
- ▶ Because actors of the same class behave the same, the component can be represented by the *initial* actor class: $[C]$.

Example

$b \rightarrow s := \text{new Server} \cdot c \rightarrow s.\text{serve}(c, q) \cdot w \rightarrow c.\text{response}(\text{compute}(q))$
 $\in \text{Traces}([\text{Server}])$
 where s and w are **Server** and **Worker** actors, respectively.

Boxing a Component



- ▶ **Hide** internal events: **boxing**
- ▶ Each trace is projected to the foreign actors
- ▶ Because actors of the same class behave the same, the component can be represented by the *initial* actor class: $[C]$.

Example

$$b \rightarrow s := \text{new Server} \cdot c \rightarrow s.\text{serve}(c, q) \cdot w \rightarrow c.\text{response}(\text{compute}(q))$$

$$\in \text{Traces}([\text{Server}])$$

where s and w are **Server** and **Worker** actors, respectively.

Issues to Handle

- ▶ How to specify class properties
- ▶ How to specify component properties
- ▶ How to use the specifications of smaller components to verify specifications of larger components

Specification Approach

Basic ideas:

- ▶ Specify classes and components in a similar way
- ▶ Generalize specifications of sequential procedures/methods:
 - In sequential case: relate pre- to poststates
 - In generalized case: relate traces of input events to output events
- ▶ Specification format similar to Hoare-logic:

$$\{ p \} D \{ q \}$$

where p and q are so-called trace assertions
and D is a class or component

- ▶ Informal semantics:
- ▶ If an input trace of D satisfies p , the output trace will satisfy q

Specification Approach

Basic ideas:

- ▶ Specify classes and components in a similar way
- ▶ Generalize specifications of sequential procedures/methods:
 - In sequential case: relate pre- to poststates
 - In generalized case: relate traces of input events to output events
- ▶ Specification format similar to Hoare-logic:

$$\{ p \} D \{ q \}$$

where p and q are so-called trace assertions and D is a class or component

First-order logic

- ▶ Informal semantics:
If an input trace of D satisfies p , the output trace will satisfy q
- ▶ Senders are **suppressed** in input and output traces

Specification Approach

Basic ideas:

- ▶ Specify classes and components in a similar way
- ▶ Generalize specifications of sequential procedures/methods:
 - In sequential case: relate pre- to poststates
 - In generalized case: relate traces of input events to output events
- ▶ Specification format similar to Hoare-logic:

$$\{ p \} D \{ q \}$$

Partial specification

where p and q are so-called trace assertions
and D is a class or component

→ First-order logic

- ▶ Informal semantics: $\forall t \in \text{Traces}(D) \bullet \text{split}(t, L) = (\text{input}, \text{output})$
If an input trace of D satisfies p , the output trace will satisfy q
- ▶ Senders are **suppressed** in input and output traces

Specification Approach

Basic ideas:

- ▶ Specify classes and components in a similar way
- ▶ Generalize specifications of sequential procedures/methods:
 - In sequential case: relate pre- to poststates
 - In generalized case: relate traces of input events to output events
- ▶ Specification format similar to Hoare-logic:

$$\{ p \} D \{ q \}$$

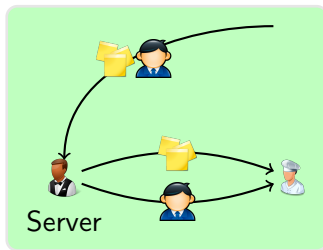
Partial specification

where p and q are so-called trace assertions
and D is a class or component

→ First-order logic

- ▶ Informal semantics: $\forall t \in \text{Traces}(D) \bullet \text{split}(t, L) = (\text{input}, \text{output})$
If an input trace of D satisfies p , the output trace will satisfy q
- ▶ Senders are **suppressed** in input and output traces

Example – Server Class



Example (Server class specification)

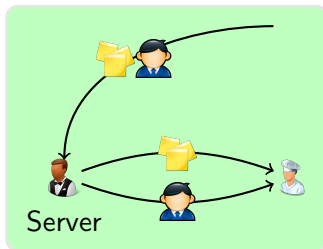
$$\{ \$ = \langle \underline{\text{this}} := \text{new Server} \cdot \underline{\text{this}}.\text{serve}(\underline{c}, \underline{q}) \rangle \}$$

Server

$$\{ \exists \underline{w} \bullet \$ = \langle \underline{w} := \text{new Worker} \cdot \underline{w}.\text{do}(\underline{q}) \cdot \underline{w}.\text{propagate}(\text{null}, \underline{c}) \rangle \}$$

► Instead of program, we have a class.

Example – Server Class



\$ is the trace constant

Example (Server class specification)

$$\{ \$ = \langle \underline{\text{this}} := \text{new Server} \cdot \underline{\text{this}}.\text{serve}(\underline{c}, \underline{q}) \rangle \}$$

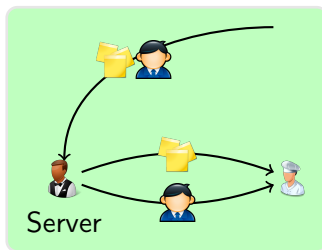
Server

$$\{ \exists \underline{w} \bullet \$ = \langle \underline{w} := \text{new Worker} \cdot \underline{w}.\text{do}(\underline{q}) \cdot \underline{w}.\text{propagate}(\text{null}, \underline{c}) \rangle \}$$

this, c, q, w are
logical variables

► Instead of program, we have a class.

Example – Server Class



\$ is the trace constant

Example (Server class specification)

$$\{ \$ = \langle \underline{\text{this}} := \text{new Server} \cdot \underline{\text{this}}.\text{serve}(\underline{c}, \underline{q}) \rangle \}$$

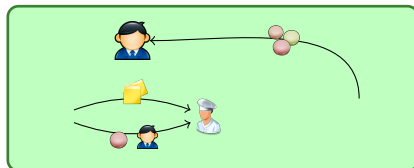
Server

$$\{ \exists \underline{w} \bullet \$ = \langle \underline{w} := \text{new Worker} \cdot \underline{w}.\text{do}(\underline{q}) \cdot \underline{w}.\text{propagate}(\text{null}, \underline{c}) \rangle \}$$

this, c, q, w are logical variables

- Instead of program, we have a class.

Example – Server and Worker Components



Example (Worker component specification)

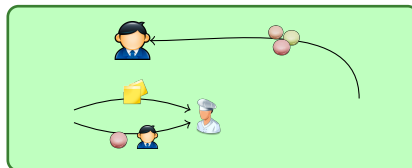
```
{ $ = <this := new Worker · this.do(q) · this.propagate(v, c)>>
  [Worker]
{ $ = <c.response(merge(v, compute(q)))>>}
```

Example (Server component specification)

```
{ $ = <this := new Server · this.serve(c, q)>>
  [Server]
{ $ = <c.response(compute(q))>>}
```



Example – Server and Worker Components

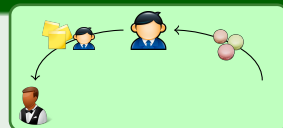


Example (Worker component specification)

```
{ $ = <this := new Worker · this.do(q) · this.propagate(v, c)>>
  [Worker]
{ $ = <c.response(merge(v, compute(q)))>>}
```

Example (Server component specification)

```
{ $ = <this := new Server · this.serve(c, q)>>
  [Server]
{ $ = <c.response(compute(q))>>}
```



Our Verification Approach

Here, we present a logic that supports a simple form of composition:

- ▶ An actor can create other actors and send messages to the newly created actor, but never expose
 - its own name to the newly created actors
 - the name of the newly created actor to other actors

I.e., actor chains of arbitrary length can be constructed, and communication flows in one direction

- ▶ Actors outside of the chain can be freely passed

The proof system uses (verified) class specs as axioms and has 3 standard rules and 3 new proof rules

Remark

See Ahrendt and Dylla [2012] and Din et al. [2012] to verify whether an implementation satisfies the class specification

- [2012] W. Ahrendt, M. Dylla: A system for compositional verification of asynchronous objects. *Sci. Comput. Program.* 77(12): 1289–1309 (2012)
- [2012] C. C. Din, J. Dovland, E. B. Johnsen, O. Owe: Observable behavior of distributed systems: Component reasoning for concurrent objects. *J. Log. Algebr. Program.* 81(3): 227–256 (2012)

Our Verification Approach

Here, we present a logic that supports a simple form of composition:

- ▶ An actor can create other actors and send messages to the newly created actor, but never expose
 - its own name to the newly created actors
 - the name of the newly created actor to other actors

I.e., actor chains of arbitrary length can be constructed, and communication flows in one direction

- ▶ Actors outside of the chain can be freely passed

The proof system uses (verified) class specs as axioms and has 3 standard rules and 3 new proof rules

Remark

See Ahrendt and Dylla [2012] and Din et al. [2012] to verify whether an implementation satisfies the class specification

- [2012] W. Ahrendt, M. Dylla: A system for compositional verification of asynchronous objects. *Sci. Comput. Program.* 77(12): 1289–1309 (2012)
- [2012] C. C. Din, J. Dovland, E. B. Johnsen, O. Owe: Observable behavior of distributed systems: Component reasoning for concurrent objects. *J. Log. Algebr. Program.* 81(3): 227–256 (2012)

Rules (1)

CONSEQUENCE

$$\frac{\begin{array}{l} p \implies p_1 \\ \{p_1\} D \{q_1\} \\ q_1 \implies q \end{array}}{\{p\} D \{q\}}$$

INVARIANCE

$$\frac{\{p\} D \{q\}}{\{p \wedge r\} D \{q \wedge r\}} \\ \text{where } \text{consFree}(r)$$

SUBSTITUTION

$$\frac{\{p\} D \{q\}}{\{p[\underline{x}/r]\} D \{q[\underline{x}/r]\}} \\ \text{where } x \in \text{free}(p) \cup \text{free}(q) \\ \text{and } \text{consFree}(r)$$

Rules (2)

BOXING

$$\frac{\{p\} C \{q \wedge \text{nonCreational}(\$)\}}{\{p\} [C] \{q\}}$$

BOXEDCOMPOSITION

$$\frac{\{p \wedge \underline{i} = \$\} C \{q \wedge \text{noSelfExposure}(\underline{i}, \$)\} \quad \{q'\} D \{r \wedge \text{nonCr}(\$)\} \quad \text{match}(q, q', D)}{\{p\} [C] \{r\}}$$

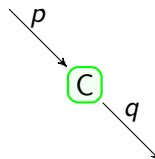
where $\underline{i} \notin \text{free}(p) \cup \text{free}(q)$



Rules (2)

BOXING

$$\frac{\{p\} C \{q \wedge \text{nonCreational}(\$)\}}{\{p\} [C] \{q\}}$$



BOXEDCOMPOSITION

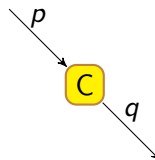
$$\frac{\begin{array}{l} \{p \wedge \underline{i} = \$\} C \{q \wedge \text{noSelfExposure}(\underline{i}, \$)\} \\ \{q'\} D \{r \wedge \text{nonCr}(\$)\} \quad \text{match}(q, q', D) \end{array}}{\begin{array}{l} \{p\} [C] \{r\} \\ \text{where } \underline{i} \notin \text{free}(p) \cup \text{free}(q) \end{array}}$$



Rules (2)

BOXING

$$\frac{\{p\} C \{q \wedge \text{nonCreational}(\$)\}}{\{p\} [C] \{q\}}$$



BOXEDCOMPOSITION

$$\frac{\{p \wedge \underline{i} = \$\} C \{q \wedge \text{noSelfExposure}(\underline{i}, \$)\} \quad \{q'\} D \{r \wedge \text{nonCr}(\$)\} \quad \text{match}(q, q', D)}{\{p\} [C] \{r\}}$$

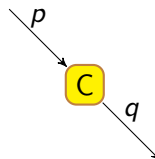
where $\underline{i} \notin \text{free}(p) \cup \text{free}(q)$



Rules (2)

BOXING

$$\frac{\{p\} C \{q \wedge \text{nonCreational}(\$)\}}{\{p\} [C] \{q\}}$$



BOXEDCOMPOSITION

$$\frac{\{p \wedge \underline{i} = \$\} C \{q \wedge \text{noSelfExposure}(\underline{i}, \$)\} \quad \{q'\} D \{r \wedge \text{nonCr}(\$)\} \quad \text{match}(q, q', D)}{\{p\} [C] \{r\}}$$

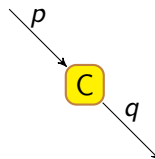
where $\underline{i} \notin \text{free}(p) \cup \text{free}(q)$



Rules (2)

BOXING

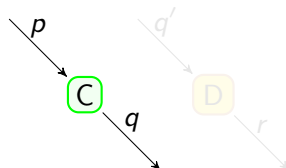
$$\frac{\{p\} C \{q \wedge \text{nonCreational}(\$)\}}{\{p\} [C] \{q\}}$$



BOXEDCOMPOSITION

$$\frac{\{p \wedge \underline{i} = \$\} C \{q \wedge \text{noSelfExposure}(\underline{i}, \$)\} \quad \{q'\} D \{r \wedge \text{nonCr}(\$)\} \quad \text{match}(q, q', D)}{\{p\} [C] \{r\}}$$

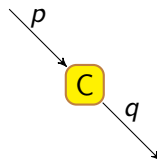
where $\underline{i} \notin \text{free}(p) \cup \text{free}(q)$



Rules (2)

BOXING

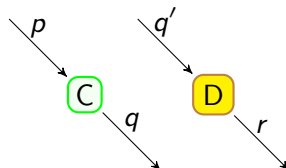
$$\frac{\{p\} C \{q \wedge \text{nonCreational}(\$)\}}{\{p\} [C] \{q\}}$$



BOXEDCOMPOSITION

$$\frac{\{p \wedge \underline{i} = \$\} C \{q \wedge \text{noSelfExposure}(\underline{i}, \$)\} \quad \{q'\} D \{r \wedge \text{nonCr}(\$)\} \quad \text{match}(q, q', D)}{\{p\} [C] \{r\}}$$

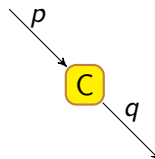
where $\underline{i} \notin \text{free}(p) \cup \text{free}(q)$



Rules (2)

BOXING

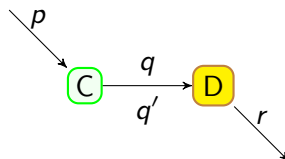
$$\frac{\{p\} C \{q \wedge \text{nonCreational}(\$)\}}{\{p\} [C] \{q\}}$$



BOXEDCOMPOSITION

$$\frac{\{p \wedge \underline{i} = \$\} C \{q \wedge \text{noSelfExposure}(\underline{i}, \$)\} \quad \{q'\} D \{r \wedge \text{nonCr}(\$)\} \quad \text{match}(q, q', D)}{\{p\} [C] \{r\}}$$

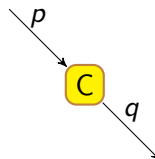
where $\underline{i} \notin \text{free}(p) \cup \text{free}(q)$



Rules (2)

BOXING

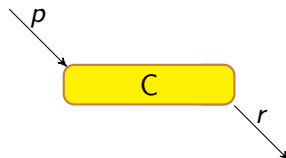
$$\frac{\{p\} C \{q \wedge \text{nonCreational}(\$)\}}{\{p\} [C] \{q\}}$$



BOXEDCOMPOSITION

$$\frac{\{p \wedge \underline{i} = \$\} C \{q \wedge \text{noSelfExposure}(\underline{i}, \$)\} \quad \{q'\} D \{r \wedge \text{nonCr}(\$)\} \quad \text{match}(q, q', D)}{\{p\} [C] \{r\}}$$

where $\underline{i} \notin \text{free}(p) \cup \text{free}(q)$



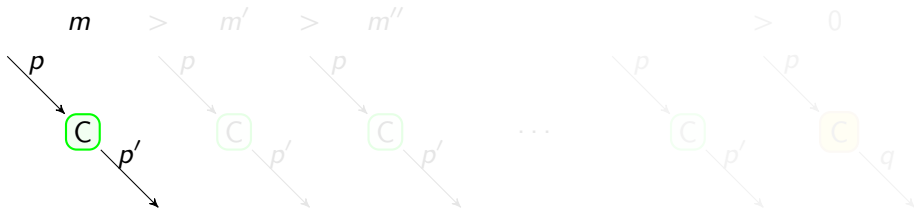
Rule(3)



INDUCTION

$$\frac{
 \begin{array}{l}
 \{p \wedge m = \underline{z} \wedge m > 0 \wedge \underline{i} = \$\} \ C \ \{p' \wedge m < \underline{z} \wedge \text{noSelfExp}(\underline{i}, \$)\} \\
 \{p \wedge m = 0\} \ [C] \ \{q\} \quad \text{match}(p', p, C)
 \end{array}
 }{
 \begin{array}{l}
 \{p\} \ [C] \ \{q\} \\
 \text{where } \underline{i} \notin \text{free}(p) \cup \text{free}(q)
 \end{array}
 }$$

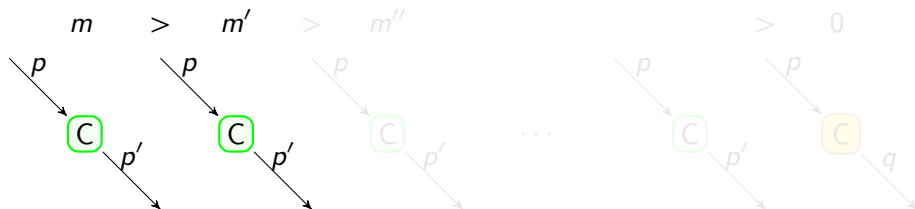
Rule(3)



INDUCTION

$$\frac{
 \begin{array}{l}
 \{p \wedge m = \underline{z} \wedge m > 0 \wedge \underline{i} = \$\} \ C \ \{p' \wedge m < \underline{z} \wedge \text{noSelfExp}(\underline{i}, \$)\} \\
 \{p \wedge m = 0\} \ [C] \ \{q\} \quad \text{match}(p', p, C)
 \end{array}
 }{
 \begin{array}{l}
 \{p\} \ [C] \ \{q\} \\
 \text{where } \underline{i} \notin \text{free}(p) \cup \text{free}(q)
 \end{array}
 }$$

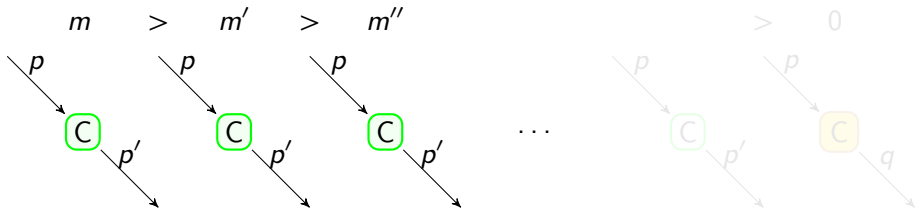
Rule(3)



INDUCTION

$$\begin{array}{c}
 \{p \wedge m = \underline{z} \wedge m > 0 \wedge \underline{i} = \$\} \ C \ \{p' \wedge m < \underline{z} \wedge \text{noSelfExp}(\underline{i}, \$)\} \\
 \{p \wedge m = 0\} \ [C] \ \{q\} \quad \text{match}(p', p, C) \\
 \hline
 \{p\} \ [C] \ \{q\} \\
 \text{where } \underline{i} \notin \text{free}(p) \cup \text{free}(q)
 \end{array}$$

Rule(3)

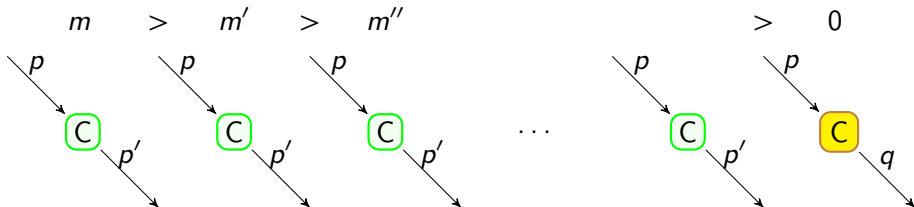


INDUCTION

$$\frac{
 \begin{array}{l}
 \{p \wedge m = \underline{z} \wedge m > 0 \wedge \underline{i} = \$\} \ C \ \{p' \wedge m < \underline{z} \wedge \text{noSelfExp}(\underline{i}, \$)\} \\
 \{p \wedge m = 0\} \ [C] \ \{q\} \quad \text{match}(p', p, C)
 \end{array}
 }{
 \{p\} \ [C] \ \{q\}
 }$$

where $\underline{i} \notin \text{free}(p) \cup \text{free}(q)$

Rule(3)

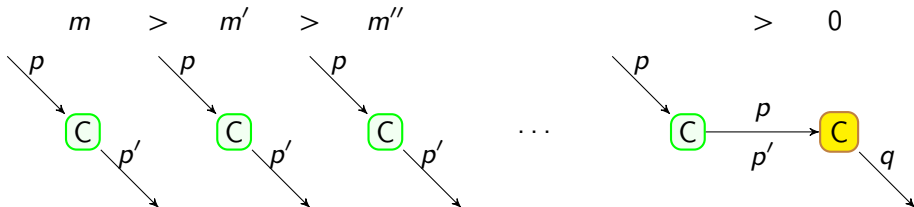


INDUCTION

$$\frac{
 \begin{array}{l}
 \{p \wedge m = \underline{z} \wedge m > 0 \wedge \underline{i} = \$\} \ C \ \{p' \wedge m < \underline{z} \wedge \text{noSelfExp}(\underline{i}, \$)\} \\
 \{p \wedge m = 0\} \ [C] \ \{q\} \quad \text{match}(p', p, C)
 \end{array}
 }{
 \{p\} \ [C] \ \{q\}
 }$$

where $\underline{i} \notin \text{free}(p) \cup \text{free}(q)$

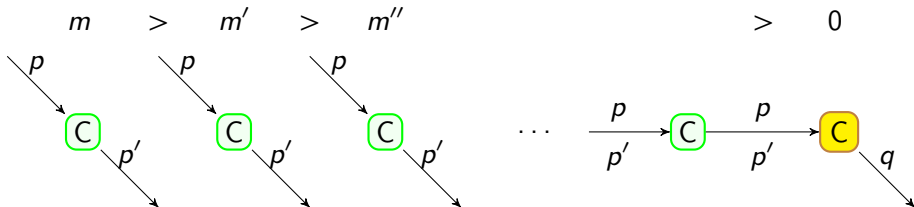
Rule(3)



INDUCTION

$$\frac{
 \begin{array}{l}
 \{p \wedge m = \underline{z} \wedge m > 0 \wedge \underline{i} = \$\} \ C \ \{p' \wedge m < \underline{z} \wedge \text{noSelfExp}(\underline{i}, \$)\} \\
 \{p \wedge m = 0\} \ [C] \ \{q\} \quad \text{match}(p', p, C)
 \end{array}
 }{
 \begin{array}{l}
 \{p\} \ [C] \ \{q\} \\
 \text{where } \underline{i} \notin \text{free}(p) \cup \text{free}(q)
 \end{array}
 }$$

Rule(3)

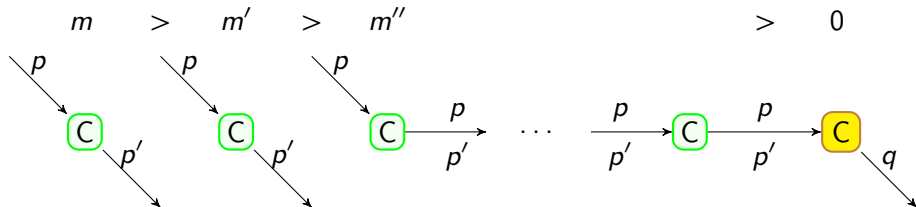


INDUCTION

$$\frac{
 \begin{array}{l}
 \{p \wedge m = \underline{z} \wedge m > 0 \wedge \underline{i} = \$\} \ C \ \{p' \wedge m < \underline{z} \wedge \text{noSelfExp}(\underline{i}, \$)\} \\
 \{p \wedge m = 0\} \ [C] \ \{q\} \quad \text{match}(p', p, C)
 \end{array}
 }{
 \{p\} \ [C] \ \{q\}
 }$$

where $\underline{i} \notin \text{free}(p) \cup \text{free}(q)$

Rule(3)

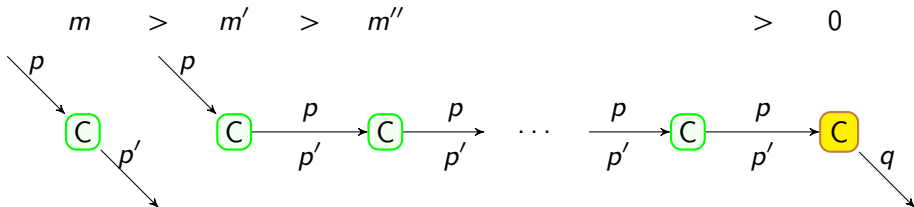


INDUCTION

$$\frac{
 \begin{array}{l}
 \{p \wedge m = \underline{z} \wedge m > 0 \wedge \underline{i} = \$\} \ C \ \{p' \wedge m < \underline{z} \wedge \text{noSelfExp}(\underline{i}, \$)\} \\
 \{p \wedge m = 0\} \ [C] \ \{q\} \quad \text{match}(p', p, C)
 \end{array}
 }{
 \{p\} \ [C] \ \{q\}
 }$$

where $\underline{i} \notin \text{free}(p) \cup \text{free}(q)$

Rule(3)

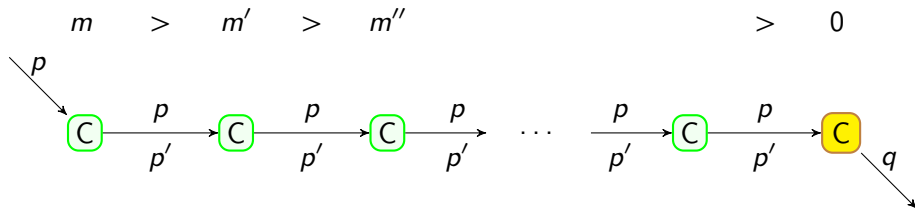


INDUCTION

$$\frac{
 \{p \wedge m = \underline{z} \wedge m > 0 \wedge \underline{i} = \$\} \ C \ \{p' \wedge m < \underline{z} \wedge \text{noSelfExp}(\underline{i}, \$)\} \\
 \{p \wedge m = 0\} \ [C] \ \{q\} \quad \text{match}(p', p, C)
 }{
 \{p\} \ [C] \ \{q\}
 }$$

where $\underline{i} \notin \text{free}(p) \cup \text{free}(q)$

Rule(3)

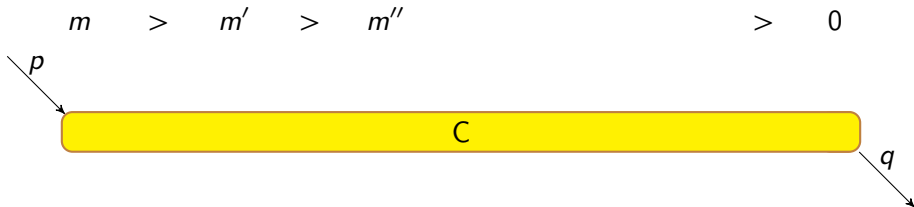


INDUCTION

$$\frac{
 \begin{array}{l}
 \{p \wedge m = \underline{z} \wedge m > 0 \wedge \underline{i} = \$\} \ C \ \{p' \wedge m < \underline{z} \wedge \text{noSelfExp}(\underline{i}, \$)\} \\
 \{p \wedge m = 0\} \ [C] \ \{q\} \quad \text{match}(p', p, C)
 \end{array}
 }{
 \{p\} \ [C] \ \{q\}
 }$$

where $\underline{i} \notin \text{free}(p) \cup \text{free}(q)$

Rule(3)



INDUCTION

$$\frac{
 \begin{array}{l}
 \{p \wedge m = \underline{z} \wedge m > 0 \wedge \underline{i} = \$\} \ C \ \{p' \wedge m < \underline{z} \wedge \text{noSelfExp}(\underline{i}, \$)\} \\
 \{p \wedge m = 0\} \ [C] \ \{q\} \quad \text{match}(p', p, C)
 \end{array}
 }{
 \{p\} \ [C] \ \{q\}
 }$$

where $\underline{i} \notin \text{free}(p) \cup \text{free}(q)$

Example: Verifying Server Component

Use

$$\{ \$ = \langle \underline{\text{this}} := \text{new Server} \cdot \underline{\text{this}}.\text{serve}(\underline{c}, \underline{q}) \rangle \}$$

Server

$$\{ \exists \underline{w} \bullet \$ = \langle \underline{w} := \text{new Worker} \cdot \underline{w}.\text{do}(\underline{q}) \cdot \underline{w}.\text{propagate}(\text{null}, \underline{c}) \rangle \}$$

and

$$\{ \$ = \langle \underline{\text{this}} := \text{new Worker} \cdot \underline{\text{this}}.\text{do}(\underline{q}) \cdot \underline{\text{this}}.\text{propagate}(\underline{v}, \underline{c}) \rangle \}$$

[Worker]

$$\{ \$ = \langle \underline{c}.\text{response}(\text{merge}(\underline{v}, \text{compute}(\underline{q}))) \rangle \}$$

to prove

$$\{ \$ = \langle \underline{\text{this}} := \text{new Server} \cdot \underline{\text{this}}.\text{serve}(\underline{c}, \underline{q}) \rangle \}$$

[Server]

$$\{ \$ = \langle \underline{c}.\text{response}(\text{compute}(\underline{q})) \rangle \}$$

BOXED COMPOSITION

$$\{p \wedge i = \$\} \quad C \{q \wedge \text{noSelfExposure}(i, \$)\}$$

$$\{q'\} \quad D \{r \wedge \text{nonCr}(\$)\}$$

$$\text{match}(q, q', D)$$

$$\{p\} \quad [C] \quad \{r\}$$

where $i \notin \text{free}(p) \cup \text{free}(q)$

Server Component Verification (1)

- ▶ $\text{InSrv} \stackrel{\text{def}}{=} \$ = \langle \underline{\text{this}} := \text{new Server} \cdot \underline{\text{this}}.\text{serve}(\underline{c}, \underline{q}) \rangle$
- ▶ $\text{OutSrv} \stackrel{\text{def}}{=} \$ = \langle \underline{w} := \text{new Worker} \cdot \underline{w}.\text{do}(\underline{q}) \cdot \underline{w}.\text{propagate}(\text{null}, \underline{c}) \rangle$

Proof.

CLASSSPEC $\frac{}{\{\text{InSrv}\} \text{Server} \{\exists \underline{w} \bullet \text{OutSrv}\}}$

INV $\frac{}{\{\text{InSrv} \wedge \underline{i} = \text{cse}(\text{InSrv})\} \text{Server} \{\exists \underline{w} \bullet \text{OutSrv} \wedge \underline{i} = \text{cse}(\text{InSrv})\}}$
 $\text{InSrv} \wedge \underline{i} = \$ \implies \text{InSrv} \wedge \underline{i} = \text{cse}(\text{InSrv})$

CONS $\frac{\exists \underline{w} \bullet \text{OutSrv} \wedge \underline{i} = \text{cse}(\text{InSrv}) \implies \exists \underline{w} \bullet \text{OutSrv} \wedge \text{noSelfExp}(\underline{i}, \$)}{\{\text{InSrv} \wedge \underline{i} = \$\} \text{Server} \{\exists \underline{w} \bullet \text{OutSrv} \wedge \text{noSelfExp}(\underline{i}, \$)\}}$

BOXEDCOMPOSITION

$\{p \wedge \underline{i} = \$\} \ C \ \{q \wedge \text{noSelfExp}(\underline{i}, \$)\}$
 $\{q'\} \ D \ \{r \wedge \text{nonCr}(\$)\}$
 $\text{match}(q, q', D)$

 $\{p\} \ [C] \ \{r\}$
 where $\underline{i} \notin \text{free}(p) \cup \text{free}(q)$

Server Component Verification (1)

- ▶ $\text{InSrv} \stackrel{\text{def}}{=} \$ = \langle \underline{\text{this}} := \text{new Server} \cdot \underline{\text{this}}.\text{serve}(\underline{c}, \underline{q}) \rangle$
- ▶ $\text{OutSrv} \stackrel{\text{def}}{=} \$ = \langle \underline{w} := \text{new Worker} \cdot \underline{w}.\text{do}(\underline{q}) \cdot \underline{w}.\text{propagate}(\text{null}, \underline{c}) \rangle$

Proof.

$$\begin{array}{c}
 \text{CLASSSPEC} \frac{}{\{\text{InSrv}\} \text{ Server } \{\exists \underline{w} \bullet \text{OutSrv}\}} \\
 \text{INV} \frac{}{\{\text{InSrv} \wedge \underline{i} = \text{cse}(\text{InSrv})\} \text{ Server } \{\exists \underline{w} \bullet \text{OutSrv} \wedge \underline{i} = \text{cse}(\text{InSrv})\}} \\
 \text{InSrv} \wedge \underline{i} = \$ \implies \text{InSrv} \wedge \underline{i} = \text{cse}(\text{InSrv}) \\
 \exists \underline{w} \bullet \text{OutSrv} \wedge \underline{i} = \text{cse}(\text{InSrv}) \implies \exists \underline{w} \bullet \text{OutSrv} \wedge \text{noSelfExp}(\underline{i}, \$) \\
 \text{CONS} \frac{}{\{\text{InSrv} \wedge \underline{i} = \$\} \text{ Server } \{\exists \underline{w} \bullet \text{OutSrv} \wedge \text{noSelfExp}(\underline{i}, \$)\}}
 \end{array}$$

BOXED COMPOSITION

$$\begin{array}{c}
 \{p \wedge \underline{i} = \$\} \ C \ \{q \wedge \text{noSelfExp}(\underline{i}, \$)\} \\
 \{q'\} \ D \ \{r \wedge \text{nonCr}(\$)\} \\
 \text{match}(q, q', D) \\
 \hline
 \{p\} \ [C] \ \{r\} \\
 \text{where } \underline{i} \notin \text{free}(p) \cup \text{free}(q)
 \end{array}$$

□

Server Component Verification (1)

- ▶ $\text{InSrv} \stackrel{\text{def}}{=} \$ = \langle \text{this} := \text{new Server} \cdot \text{this.serve}(\underline{c}, \underline{q}) \rangle$
- ▶ $\text{OutSrv} \stackrel{\text{def}}{=} \$ = \langle \underline{w} := \text{new Worker} \cdot \underline{w}.\text{do}(\underline{q}) \cdot \underline{w}.\text{propagate}(\text{null}, \underline{c}) \rangle$

Proof.

$$\begin{array}{c}
 \text{CLASSSPEC} \frac{}{\{\text{InSrv}\} \text{ Server } \{\exists \underline{w} \bullet \text{OutSrv}\}} \\
 \text{INV} \frac{}{\{\text{InSrv} \wedge \underline{i} = \text{cse}(\text{InSrv})\} \text{ Server } \{\exists \underline{w} \bullet \text{OutSrv} \wedge \underline{i} = \text{cse}(\text{InSrv})\}} \\
 \text{InSrv} \wedge \underline{i} = \$ \implies \text{InSrv} \wedge \underline{i} = \text{cse}(\text{InSrv}) \\
 \exists \underline{w} \bullet \text{OutSrv} \wedge \underline{i} = \text{cse}(\text{InSrv}) \implies \exists \underline{w} \bullet \text{OutSrv} \wedge \text{noSelfExp}(\underline{i}, \$) \\
 \text{CONS} \frac{}{\{\text{InSrv} \wedge \underline{i} = \$\} \text{ Server } \{\exists \underline{w} \bullet \text{OutSrv} \wedge \text{noSelfExp}(\underline{i}, \$)\}}
 \end{array}$$

BOXED COMPOSITION

$$\begin{array}{c}
 \{p \wedge \underline{i} = \$\} \ C \ \{q \wedge \text{noSelfExp}(\underline{i}, \$)\} \\
 \{q'\} \ D \ \{r \wedge \text{nonCr}(\$)\} \\
 \text{match}(q, q', D) \\
 \hline
 \{p\} \ [C] \ \{r\} \\
 \text{where } \underline{i} \notin \text{free}(p) \cup \text{free}(q)
 \end{array}$$

□

Server Component Verification (1)

- ▶ $\text{InSrv} \stackrel{\text{def}}{=} \$ = \langle \underline{\text{this}} := \text{new Server} \cdot \underline{\text{this}}.\text{serve}(\underline{c}, \underline{q}) \rangle$
- ▶ $\text{OutSrv} \stackrel{\text{def}}{=} \$ = \langle \underline{w} := \text{new Worker} \cdot \underline{w}.\text{do}(\underline{q}) \cdot \underline{w}.\text{propagate}(\text{null}, \underline{c}) \rangle$

Proof.

$$\begin{array}{c}
 \text{CLASSSPEC} \frac{}{\{\text{InSrv}\} \text{ Server } \{\exists \underline{w} \bullet \text{OutSrv}\}} \\
 \text{INV} \frac{\{\text{InSrv} \wedge \underline{i} = \text{cse}(\text{InSrv})\} \text{ Server } \{\exists \underline{w} \bullet \text{OutSrv} \wedge \underline{i} = \text{cse}(\text{InSrv})\}}{\text{InSrv} \wedge \underline{i} = \$ \implies \text{InSrv} \wedge \underline{i} = \text{cse}(\text{InSrv})} \\
 \text{CONS} \frac{\exists \underline{w} \bullet \text{OutSrv} \wedge \underline{i} = \text{cse}(\text{InSrv}) \implies \exists \underline{w} \bullet \text{OutSrv} \wedge \text{noSelfExp}(\underline{i}, \$)}{\{\text{InSrv} \wedge \underline{i} = \$\} \text{ Server } \{\exists \underline{w} \bullet \text{OutSrv} \wedge \text{noSelfExp}(\underline{i}, \$)\}}
 \end{array}$$

BOXED COMPOSITION

$$\begin{array}{c}
 \{p \wedge \underline{i} = \$\} \ C \ \{q \wedge \text{noSelfExp}(\underline{i}, \$)\} \\
 \{q'\} \ D \ \{r \wedge \text{nonCr}(\$)\} \\
 \text{match}(q, q', D) \\
 \hline
 \{p\} \ [C] \ \{r\} \\
 \text{where } \underline{i} \notin \text{free}(p) \cup \text{free}(q)
 \end{array}$$



Server Component Verification (2)

- ▶ $\text{InWrk} \stackrel{\text{def}}{=} \$ = \langle \underline{\text{this}} := \text{new Worker} \cdot \underline{\text{this}}.\text{do}(\underline{q}) \cdot \underline{\text{this}}.\text{propagate}(\underline{v}, \underline{c}) \rangle$
- ▶ $\text{OutWrk} \stackrel{\text{def}}{=} \$ = \langle \underline{c}.\text{response}(\text{merge}(\underline{v}, \text{compute}(\underline{q}))) \rangle$
- ▶ $\text{OutSrvC} \stackrel{\text{def}}{=} \$ = \langle \underline{c}.\text{response}(\text{compute}(\underline{q})) \rangle$

Proof.

$$\begin{array}{c}
 \text{SUB} \frac{\{\text{InWrk}\} [\text{Worker}] \{\text{OutWrk}\}}{\{\text{InWrk}[\underline{v}/\underline{\text{null}}]\} [\text{Worker}] \{\text{OutWrk}[\underline{v}/\underline{\text{null}}]\} \\
 \text{OutWrk}[\underline{v}/\underline{\text{null}}] \implies \text{OutSrvC} \wedge \text{nonCr}(\$)} \\
 \text{CONS} \frac{}{\{\text{InWrk}[\underline{v}/\underline{\text{null}}]\} [\text{Worker}] \{\text{OutSrvC} \wedge \text{nonCr}(\$)\}}
 \end{array}$$

BOXED COMPOSITION

$$\frac{\{p \wedge i = \$\} \ C \ \{q \wedge \text{noSelfExp}(i, \$)\} \quad \{q'\} \ D \ \{r \wedge \text{nonCr}(\$)\} \quad \text{match}(q, q', D)}{\{p\} \ [C] \ \{r\}}$$

where $i \notin \text{free}(p) \cup \text{free}(q)$



Server Component Verification (2)

- ▶ $\text{InWrk} \stackrel{\text{def}}{=} \$ = \langle \underline{\text{this}} := \text{new Worker} \cdot \underline{\text{this}}.\text{do}(q) \cdot \underline{\text{this}}.\text{propagate}(\underline{v}, \underline{c}) \rangle$
- ▶ $\text{OutWrk} \stackrel{\text{def}}{=} \$ = \langle \underline{c}.\text{response}(\text{merge}(\underline{v}, \text{compute}(q))) \rangle$
- ▶ $\text{OutSrvC} \stackrel{\text{def}}{=} \$ = \langle \underline{c}.\text{response}(\text{compute}(q)) \rangle$

Proof.

$$\begin{array}{c}
 \text{SUB} \frac{\{\text{InWrk}\} [\text{Worker}] \{\text{OutWrk}\}}{\{\text{InWrk}[\underline{v}/\text{null}]\} [\text{Worker}] \{\text{OutWrk}[\underline{v}/\text{null}]\} \\
 \text{OutWrk}[\underline{v}/\text{null}] \implies \text{OutSrvC} \wedge \text{nonCr}(\$)} \\
 \text{CONS} \frac{}{\{\text{InWrk}[\underline{v}/\text{null}]\} [\text{Worker}] \{\text{OutSrvC} \wedge \text{nonCr}(\$)\}}
 \end{array}$$

BOXED COMPOSITION

$$\begin{array}{c}
 \{p \wedge \underline{i} = \$\} \ C \ \{q \wedge \text{noSelfExp}(\underline{i}, \$)\} \\
 \{q'\} \ D \ \{r \wedge \text{nonCr}(\$)\} \\
 \text{match}(q, q', D) \\
 \hline
 \{p\} \ [C] \ \{r\} \\
 \text{where } \underline{i} \notin \text{free}(p) \cup \text{free}(q)
 \end{array}$$



Server Component Verification (2)

- ▶ $\text{InWrk} \stackrel{\text{def}}{=} \$ = \langle \underline{\text{this}} := \text{new Worker} \cdot \underline{\text{this}}.\text{do}(\underline{q}) \cdot \underline{\text{this}}.\text{propagate}(\underline{v}, \underline{c}) \rangle$
- ▶ $\text{OutWrk} \stackrel{\text{def}}{=} \$ = \langle \underline{c}.\text{response}(\text{merge}(\underline{v}, \text{compute}(\underline{q}))) \rangle$
- ▶ $\text{OutSrvC} \stackrel{\text{def}}{=} \$ = \langle \underline{c}.\text{response}(\text{compute}(\underline{q})) \rangle$

Proof.

$$\begin{array}{c}
 \text{SUB} \frac{\{\text{InWrk}\} [\text{Worker}] \{\text{OutWrk}\}}{\{\text{InWrk}[\underline{v}/\text{null}]\} [\text{Worker}] \{\text{OutWrk}[\underline{v}/\text{null}]\}} \\
 \text{CONS} \frac{\text{OutWrk}[\underline{v}/\text{null}] \implies \text{OutSrvC} \wedge \text{nonCr}(\$)}{\{\text{InWrk}[\underline{v}/\text{null}]\} [\text{Worker}] \{\text{OutSrvC} \wedge \text{nonCr}(\$)\}}
 \end{array}$$

BOXED COMPOSITION

$$\frac{\begin{array}{l} \{p \wedge \underline{i} = \$\} \ C \ \{q \wedge \text{noSelfExp}(\underline{i}, \$)\} \\ \{q'\} \ D \ \{r \wedge \text{nonCr}(\$)\} \\ \text{match}(q, q', D) \end{array}}{\begin{array}{l} \{p\} \ [C] \ \{r\} \\ \text{where } \underline{i} \notin \text{free}(p) \cup \text{free}(q) \end{array}}$$



Server Component Verification (2)

- ▶ $\text{InWrk} \stackrel{\text{def}}{=} \$ = \langle \underline{\text{this}} := \text{new Worker} \cdot \underline{\text{this}}.\text{do}(\underline{q}) \cdot \underline{\text{this}}.\text{propagate}(\underline{v}, \underline{c}) \rangle$
- ▶ $\text{OutWrk} \stackrel{\text{def}}{=} \$ = \langle \underline{c}.\text{response}(\text{merge}(\underline{v}, \text{compute}(\underline{q}))) \rangle$
- ▶ $\text{OutSrvC} \stackrel{\text{def}}{=} \$ = \langle \underline{c}.\text{response}(\text{compute}(\underline{q})) \rangle$

Proof.

$$\begin{array}{c}
 \text{SUB} \frac{\{\text{InWrk}\} [\text{Worker}] \{\text{OutWrk}\}}{\{\text{InWrk}[\underline{v}/\underline{\text{null}}]\} [\text{Worker}] \{\text{OutWrk}[\underline{v}/\underline{\text{null}}]\} \\
 \text{OutWrk}[\underline{v}/\underline{\text{null}}] \implies \text{OutSrvC} \wedge \text{nonCr}(\$)} \\
 \text{CONS} \frac{}{\{\text{InWrk}[\underline{v}/\underline{\text{null}}]\} [\text{Worker}] \{\text{OutSrvC} \wedge \text{nonCr}(\$)\}}
 \end{array}$$

BOXED COMPOSITION

$$\frac{\{p \wedge \underline{i} = \$\} \ C \ \{q \wedge \text{noSelfExp}(\underline{i}, \$)\} \ \{q'\} \ D \ \{r \wedge \text{nonCr}(\$)\} \ \text{match}(q, q', D)}{\{p\} \ [C] \ \{r\}}$$

where $\underline{i} \notin \text{free}(p) \cup \text{free}(q)$



Server Component Verification (3)

- ▶ $\text{InSrv} = \text{InSrvC} \stackrel{\text{def}}{=} \$ = \langle \underline{\text{this}} := \text{new Server} \cdot \underline{\text{this}}.\text{serve}(\underline{c}, \underline{q}) \rangle$
- ▶ $\text{OutSrvC} \stackrel{\text{def}}{=} \$ = \langle \underline{c}.\text{response}(\text{compute}(\underline{q})) \rangle$
- ▶ $\text{OutSrv} \stackrel{\text{def}}{=} \$ = \langle \underline{w} := \text{new Worker} \cdot \underline{w}.\text{do}(\underline{q}) \cdot \underline{w}.\text{propagate}(\text{null}, \underline{c}) \rangle$
- ▶ $\text{InWrk} \stackrel{\text{def}}{=} \$ = \langle \underline{\text{this}} := \text{new Worker} \cdot \underline{\text{this}}.\text{do}(\underline{q}) \cdot \underline{\text{this}}.\text{propagate}(\underline{v}, \underline{c}) \rangle$
- ▶ $\text{OutWrk} \stackrel{\text{def}}{=} \$ = \langle \underline{c}.\text{response}(\text{merge}(\underline{v}, \text{compute}(\underline{q}))) \rangle$

Proof.

$$\begin{aligned} & \{ \text{InSrv} \wedge \underline{i} = \$ \} \text{Server} \{ \exists \underline{w} \bullet \text{OutSrv} \wedge \text{noSelfExp}(\underline{i}, \$) \} \\ & \{ \text{InWrk}[\underline{v}/\text{null}] \} [\text{Worker}] \{ \text{OutSrvC} \wedge \text{nonCr}(\$) \} \\ & \text{match}(\exists \underline{w} \bullet \text{OutSrv}, \text{InWrk}[\underline{v}/\text{null}], [\text{Worker}]) \end{aligned}$$

BoxCmp

$$\frac{}{\{ \text{InSrvC} \} [\text{Server}] \{ \text{OutSrvC} \}}$$

BOXEDCOMPOSITION

$$\begin{aligned} & \{ p \wedge \underline{i} = \$ \} \ C \ \{ q \wedge \text{noSelfExp}(\underline{i}, \$) \} \\ & \{ q' \} \ D \ \{ r \wedge \text{nonCr}(\$) \} \\ & \text{match}(q, q', D) \\ \hline & \{ p \} \ [C] \ \{ r \} \\ & \text{where } \underline{i} \notin \text{free}(p) \cup \text{free}(q) \end{aligned}$$

Thus, the server component satisfies its specification
if the worker component satisfies its specification.

Worker component is verified using BOXING and INDUCTION.

Server Component Verification (3)

- ▶ $\text{InSrv} = \text{InSrvC} \stackrel{\text{def}}{=} \$ = \langle \underline{\text{this}} := \text{new Server} \cdot \underline{\text{this}}.\text{serve}(\underline{c}, \underline{q}) \rangle$
- ▶ $\text{OutSrvC} \stackrel{\text{def}}{=} \$ = \langle \underline{c}.\text{response}(\text{compute}(\underline{q})) \rangle$
- ▶ $\text{OutSrv} \stackrel{\text{def}}{=} \$ = \langle \underline{w} := \text{new Worker} \cdot \underline{w}.\text{do}(\underline{q}) \cdot \underline{w}.\text{propagate}(\text{null}, \underline{c}) \rangle$
- ▶ $\text{InWrk} \stackrel{\text{def}}{=} \$ = \langle \underline{\text{this}} := \text{new Worker} \cdot \underline{\text{this}}.\text{do}(\underline{q}) \cdot \underline{\text{this}}.\text{propagate}(\underline{v}, \underline{c}) \rangle$
- ▶ $\text{OutWrk} \stackrel{\text{def}}{=} \$ = \langle \underline{c}.\text{response}(\text{merge}(\underline{v}, \text{compute}(\underline{q}))) \rangle$

Proof.

$$\text{BoxCMP} \frac{\begin{array}{l} \{\text{InSrv} \wedge \underline{i} = \$\} \text{Server} \{\exists \underline{w} \bullet \text{OutSrv} \wedge \text{noSelfExp}(\underline{i}, \$)\} \\ \{\text{InWrk}[\underline{v}/\text{null}]\} [\text{Worker}] \{\text{OutSrvC} \wedge \text{nonCr}(\$)\} \\ \text{match}(\exists \underline{w} \bullet \text{OutSrv}, \text{InWrk}[\underline{v}/\text{null}], [\text{Worker}]) \end{array}}{\{\text{InSrvC}\} [\text{Server}] \{\text{OutSrvC}\}}$$

Thus, the server component satisfies its specification
if the worker component satisfies its specification.

Worker component is verified using **BOXING** and **INDUCTION**.

BOXED COMPOSITION

$$\frac{\begin{array}{l} \{p \wedge \underline{i} = \$\} \ C \ \{q \wedge \text{noSelfExp}(\underline{i}, \$)\} \\ \{q'\} \ D \ \{r \wedge \text{nonCr}(\$)\} \\ \text{match}(q, q', D) \end{array}}{\begin{array}{l} \{p\} \ [C] \ \{r\} \\ \text{where } \underline{i} \notin \text{free}(p) \cup \text{free}(q) \end{array}}$$

Soundness Discussion

Proving Soundness

- ▶ **BOXING**: follows from the component definition
- ▶ **BOXEDCOMPOSITION**: proof by cases
- ▶ **INDUCTION**: proof by induction on the number new actors created

Obtaining a **sound** logic was difficult

- ▶ Trace semantics must reflect *all* contexts
- ▶ Soundness proof needs to take into account *all* contexts

Soundness Discussion

Proving Soundness

- ▶ **BOXING**: follows from the component definition
- ▶ **BOXEDCOMPOSITION**: proof by cases
- ▶ **INDUCTION**: proof by induction on the number new actors created

Obtaining a **sound** logic was difficult

- ▶ Trace semantics must reflect *all* contexts
- ▶ Soundness proof needs to take into account *all* contexts

Conclusion

- ▶ Trace semantics for open actor-based systems
- ▶ Hoare-like specification technique for actor-based component systems
 - Trace-based
 - Hiding internal behavior
- ▶ A sound logic, strong enough to handle the server example
 - Works for *open systems*

Future work

- ▶ Enrich the logic to support more composition patterns (e.g. trees)
- ▶ Support more complex communication constructs, such as futures
- ▶ Build the connection between core operational semantics and the trace semantics (in particular, well-formedness of trace sets)

Thank You!

