SoftTeX
Software Technology Group

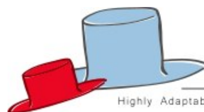
TECHNISCHE UNIVERSITÄT
KAISERSLAUTERN

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

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

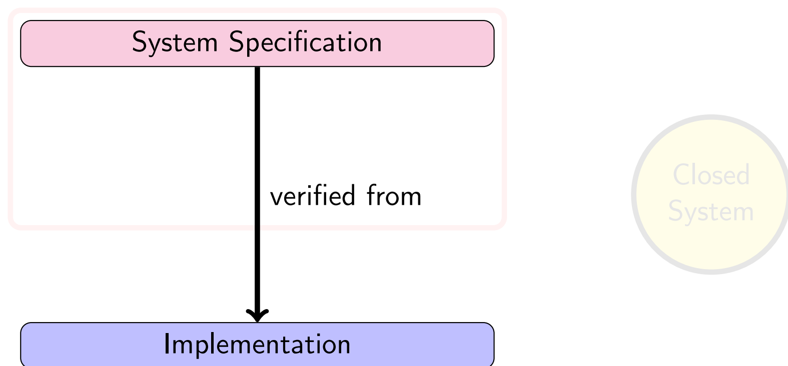
AGERE!
Tucson, Arizona, USA

October 21–22, 2012



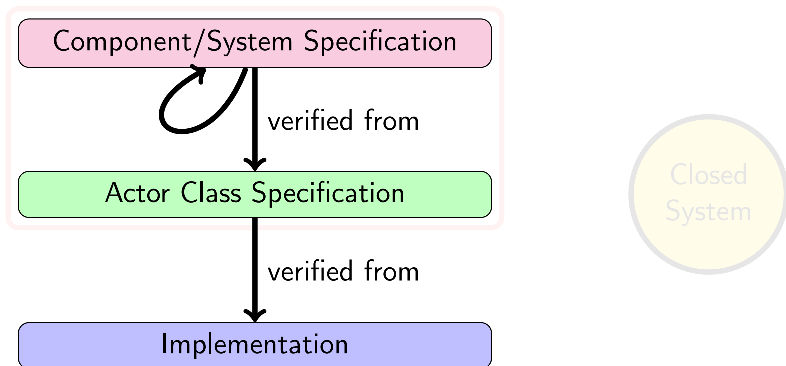
HATS
Highly Adaptable and Trustworthy Software using Formal Models

Motivation



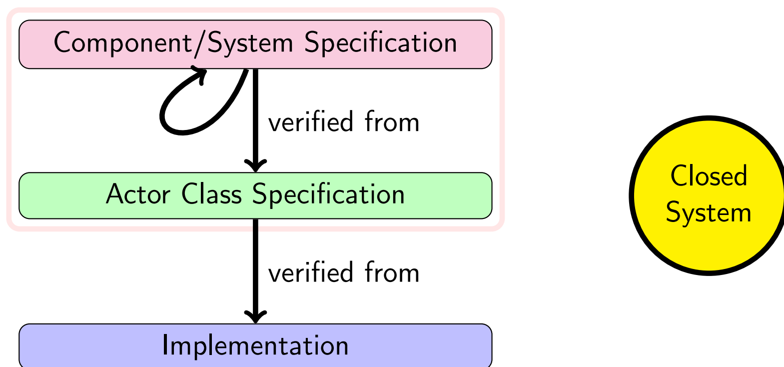
[ABS] See http://hats-project.eu, in particular http://tools.hats-project.eu

Motivation



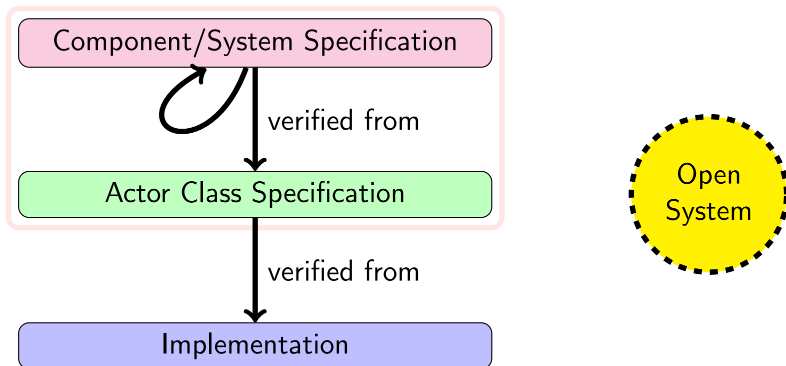
[ABS] See http://hats-project.eu, in particular http://tools.hats-project.eu

Motivation



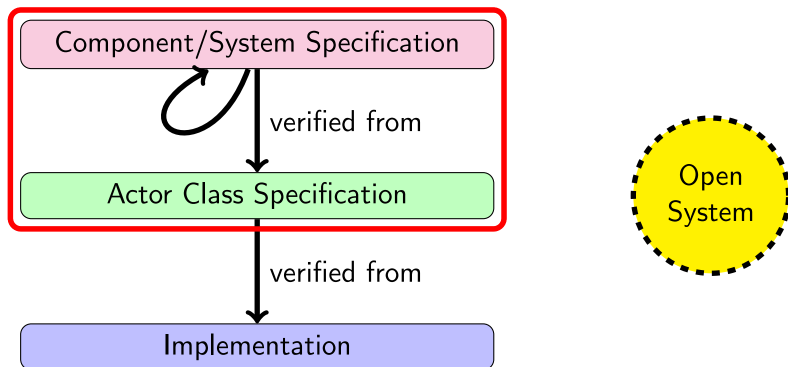
[ABS] See http://hats-project.eu, in particular http://tools.hats-project.eu

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.

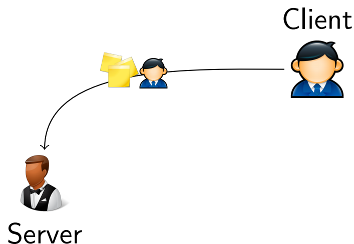
Client

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]

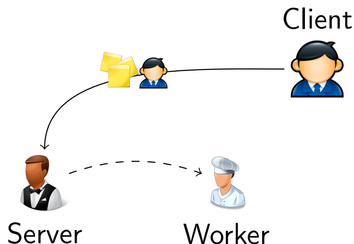
- Server performs a database query for client by delegating it to workers.
- Server triggers merging of partial results.
- If the query is large, workers delegate part of it to other worker.
- Workers merge their computation with partial result and pass it on.



[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]

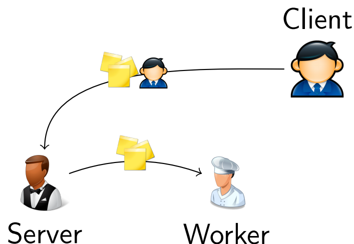
- Server performs a database query for client by delegating it to workers.
- Server triggers merging of partial results.
- If the query is large, workers delegate part of it to other worker.
- Workers merge their computation with partial result and pass it on.



[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]

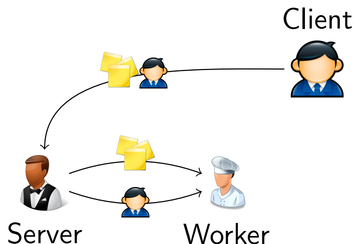
- Server performs a database query for client by delegating it to workers.
- Server triggers merging of partial results.
- If the query is large, workers delegate part of it to other worker.
- Workers merge their computation with partial result and pass it on.



[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]

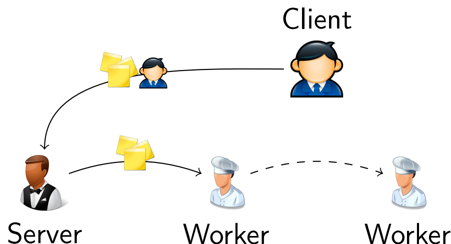
- Server performs a database query for client by delegating it to workers.
- Server triggers merging of partial results.
- If the query is large, workers delegate part of it to other worker.
- Workers merge their computation with partial result and pass it on.



[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]

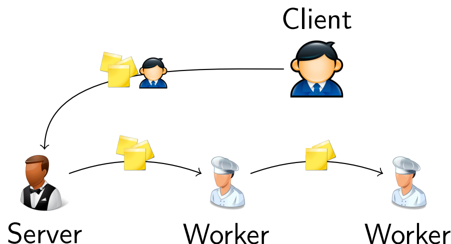
- Server performs a database query for client by delegating it to workers.
- Server triggers merging of partial results.
- If the query is large, workers delegate part of it to other worker.
- Workers merge their computation with partial result and pass it on.



[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]

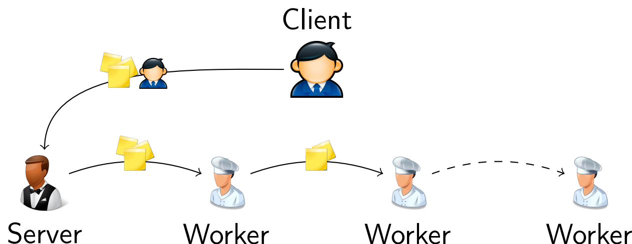
- Server performs a database query for client by delegating it to workers.
- Server triggers merging of partial results.
- If the query is large, workers delegate part of it to other worker.
- Workers merge their computation with partial result and pass it on.



[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]

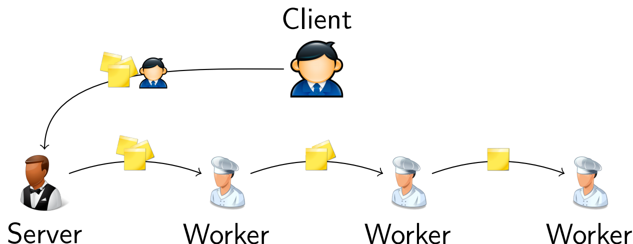
- Server performs a database query for client by delegating it to workers.
- Server triggers merging of partial results.
- If the query is large, workers delegate part of it to other worker.
- Workers merge their computation with partial result and pass it on.



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

Example [Arts and Dam, 1999]

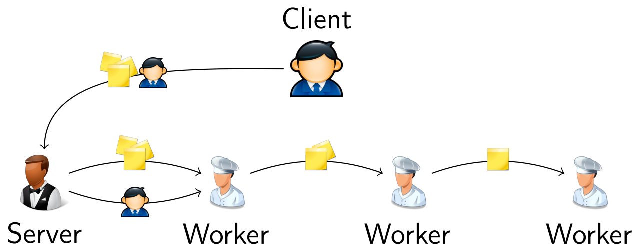
- Server performs a database query for client by delegating it to workers.
- Server triggers merging of partial results.
- If the query is large, workers delegate part of it to other worker.
- Workers merge their computation with partial result and pass it on.



[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]

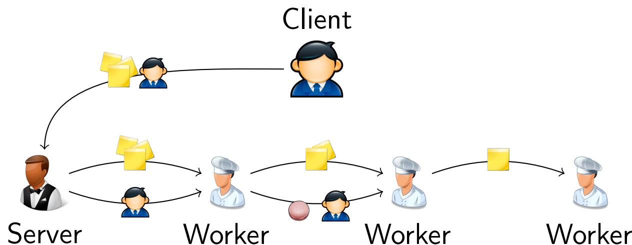
- Server performs a database query for client by delegating it to workers.
- Server triggers merging of partial results.
- If the query is large, workers delegate part of it to other worker.
- Workers merge their computation with partial result and pass it on.



[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]

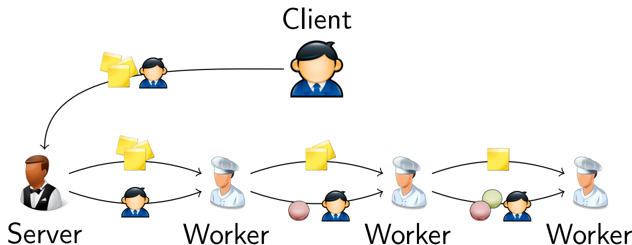
- Server performs a database query for client by delegating it to workers.
- Server triggers merging of partial results.
- If the query is large, workers delegate part of it to other worker.
- Workers merge their computation with partial result and pass it on.



[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]

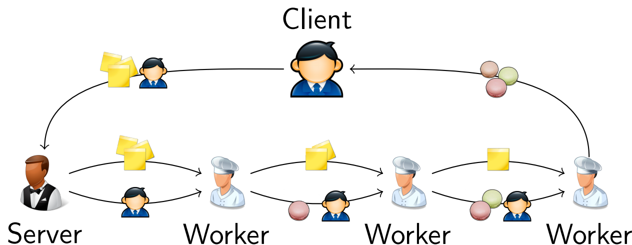
- Server performs a database query for client by delegating it to workers.
- Server triggers merging of partial results.
- If the query is large, workers delegate part of it to other worker.
- Workers merge their computation with partial result and pass it on.



[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]

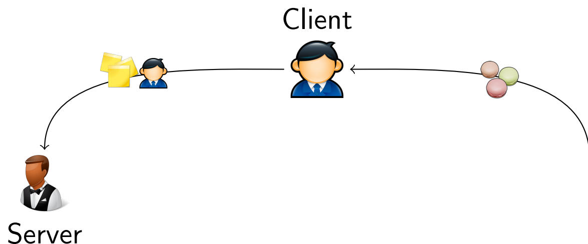
- Server performs a database query for client by delegating it to workers.
- Server triggers merging of partial results.
- If the query is large, workers delegate part of it to other worker.
- Workers merge their computation with partial result and pass it on.



[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

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 ($a \rightarrow b := \textbf{new}\ C$)
  - Message send ($a \rightarrow b.msg(\overline{p})$)
- Traces abstract from internal representation
- Traces are used as semantic foundation for specifications

Example

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

Events and Traces

- Trace: A sequence of *observable* events

  Observable events:
  - Creation of new actors ($a \rightarrow b := \textbf{new}\ C$)
  - Message send ($a \rightarrow b.msg(\overline{p})$)
- Traces abstract from internal representation
- Traces are used as semantic foundation for specifications

Example

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

Events and Traces

- Trace: A sequence of *observable* events

  Observable events:
  - Creation of new actors ($a \rightarrow b := \textbf{new}\ C$)
  - Message send ($a \rightarrow b.msg(\overline{p})$)
- Traces abstract from internal representation
- Traces are used as semantic foundation for specifications

Example

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

Events and Traces

- Trace: A sequence of *observable* events

  Observable events:

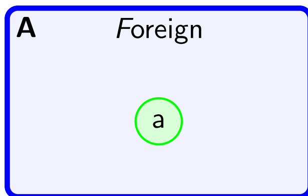
  - Creation of new actors ($a \rightarrow b := \textbf{new}\ C$)
  - Message send ($a \rightarrow b.msg(\overline{p})$)

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

Example

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

Trace Semantics (1)



- The semantics of actor *a* of class *C* is a trace set
  - Run for all contexts ⇝ produces the traces
  - Project the traces to events "relevant" to *a*
  - Collect all the traces

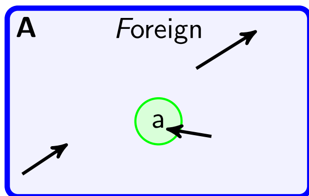
Remark: validity/well-formedness of the trace sets

Example

$$b \to s := \mathtt{new}\ \mathtt{Server} \cdot c \to s.\mathtt{serve}(c, query) \cdot s \to w := \mathtt{new}\ \mathtt{Worker} \cdot s \to w.\mathtt{do}(query) \cdot s \to w.\mathtt{propagate}(\mathtt{null}, c) \in Traces(\mathtt{Server})$$

where s is the actor of class `Server`

Trace Semantics (1)



- The semantics of actor 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

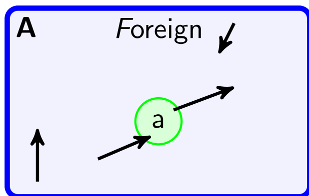
Remark: validity/well-formedness of the trace sets

Example

$$b \to s := \texttt{new Server} \cdot c \to s.\texttt{serve}(c, query) \cdot s \to w := \texttt{new Worker} \cdot$$
$$s \to w.\texttt{do}(query) \cdot s \to w.\texttt{propagate}(\texttt{null}, c) \qquad \in Traces(\texttt{Server})$$

where s is the actor of class `Server`

Trace Semantics (1)



- The semantics of actor 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

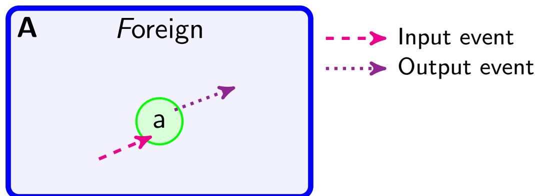
Remark: validity/well-formedness of the trace sets

Example

$b \rightarrow s := \texttt{new Server} \cdot c \rightarrow s.\texttt{serve}(c, query) \cdot s \rightarrow w := \texttt{new Worker} \cdot$
$s \rightarrow w.\texttt{do}(query) \cdot s \rightarrow w.\texttt{propagate}(\texttt{null}, c) \quad \in Traces(\texttt{Server})$

where s is the actor of class $\texttt{Server}$

Trace Semantics (1)



- The semantics of actor 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

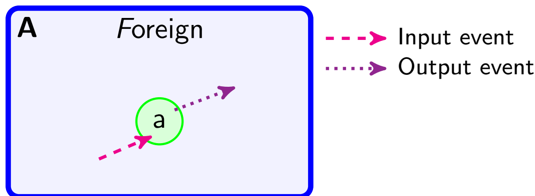
Remark: validity/well-formedness of the trace sets

Example

$$b \rightarrow s := \texttt{new Server} \cdot c \rightarrow s.\texttt{serve}(c, query) \cdot s \rightarrow w := \texttt{new Worker} \cdot$$
$$s \rightarrow w.\texttt{do}(query) \cdot s \rightarrow w.\texttt{propagate}(\texttt{null}, c) \qquad \in Traces(\texttt{Server})$$

where s is the actor of class $\texttt{Server}$

Trace Semantics (1)



- The semantics of actor 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

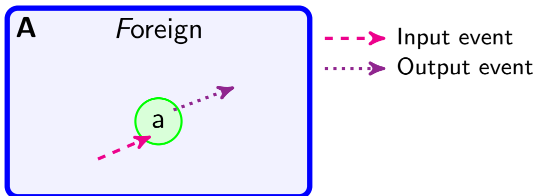
Remark: validity/well-formedness of the trace sets

Example

$$b \rightarrow s := \mathbf{new}\ \texttt{Server} \cdot c \rightarrow s.\texttt{serve}(c, query) \cdot s \rightarrow w := \mathbf{new}\ \texttt{Worker} \cdot s \rightarrow w.\texttt{do}(query) \cdot s \rightarrow w.\texttt{propagate}(\mathbf{null}, c) \quad \in Traces(\texttt{Server})$$

where s is the actor of class $\texttt{Server}$

Trace Semantics (1)



- The semantics of actor 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

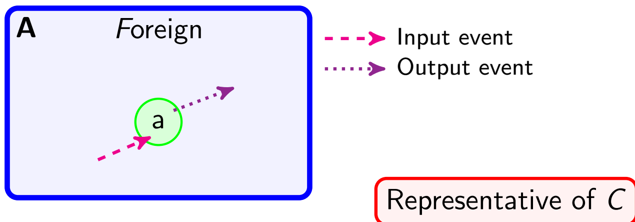
Remark: validity/well-formedness of the trace sets

Example

$b \rightarrow s := \texttt{new Server} \cdot c \rightarrow s.\texttt{serve}(c, query) \cdot s \rightarrow w := \texttt{new Worker} \cdot$
$s \rightarrow w.\texttt{do}(query) \cdot s \rightarrow w.\texttt{propagate}(\texttt{null}, c) \quad \in Traces(\texttt{Server})$

where s is the actor of class `Server`

Trace Semantics (1)



- The semantics of actor 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

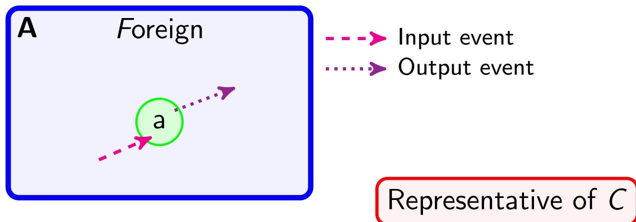
Remark: validity/well-formedness of the trace sets

Example

$b \rightarrow s := \texttt{new Server} \cdot c \rightarrow s.\texttt{serve}(c, query) \cdot s \rightarrow w := \texttt{new Worker} \cdot$
$s \rightarrow w.\texttt{do}(query) \cdot s \rightarrow w.\texttt{propagate}(\texttt{null}, c) \quad \in Traces(\texttt{Server})$

where s is the actor of class $\texttt{Server}$

Trace Semantics (1)



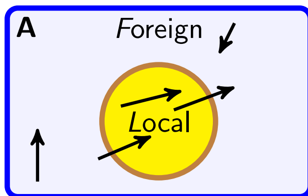
- The semantics of actor a of class C is a trace set $Traces(C)$
  - Run for all contexts $\rightsquigarrow$ produces the traces
  - Project the traces to events "relevant" to a
  - Collect all the traces

Remark: validity/well-formedness of the trace sets

Example

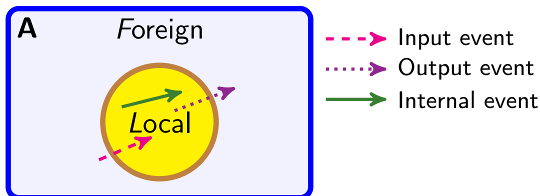
$b \rightarrow s := \mathbf{new}\ \texttt{Server} \cdot c \rightarrow s.\texttt{serve}(c, query) \cdot s \rightarrow w := \mathbf{new}\ \texttt{Worker} \cdot$
$s \rightarrow w.\texttt{do}(query) \cdot s \rightarrow w.\texttt{propagate}(\mathbf{null}, c) \quad \in Traces(\texttt{Server})$
where s is the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

Trace Semantics (2)



- The semantics of a set of local actor L is a trace set
  - Run for all contexts ⇝ produces the traces
  - Project the traces to events "relevant" to L
  - Collect all the traces

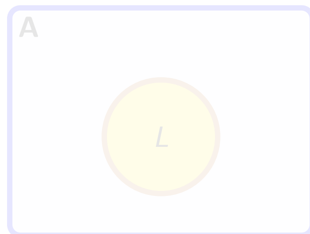
Trace Semantics (2)



- 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

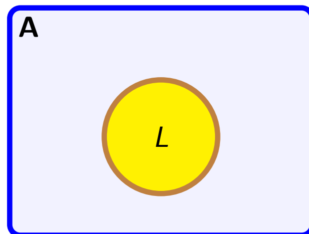


- L is a component if *no* other actors outside L are created by an actor in L.
- L can be obtained from the actor creation tree.

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

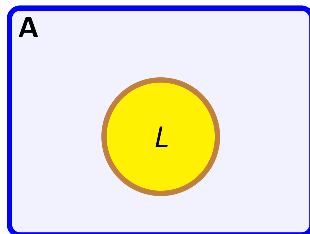


- L is a component if *no* other actors outside L are created by an actor in L.
- L can be obtained from the actor creation tree.

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

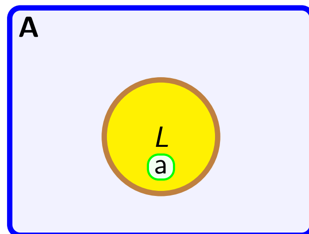


- L is a component if *no* other actors outside L are created by an actor in L.
- L can be obtained from the actor creation tree.

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

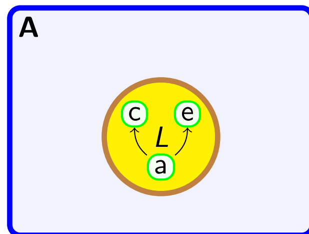


- L is a component if *no* other actors outside L are created by an actor in L.
- L can be obtained from the actor creation tree.

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

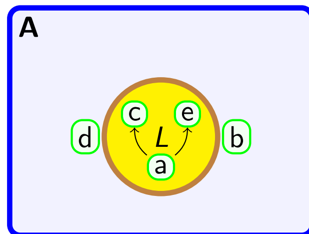


- *L* is a component if *no* other actors outside *L* are created by an actor in *L*.
- *L* can be obtained from the actor creation tree.

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

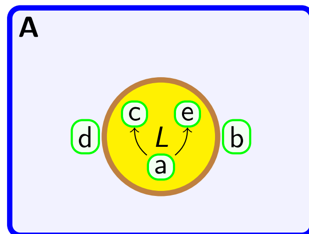


- L is a component if *no* other actors outside L are created by an actor in L.
- L can be obtained from the actor creation tree.

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

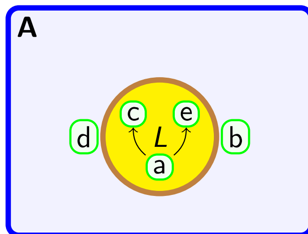


- L is a component if *no* other actors outside L are created by an actor in L.
- L can be obtained from the actor creation tree.

Example

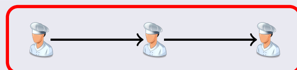
Components

- 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.

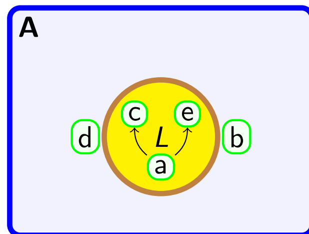
Example



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

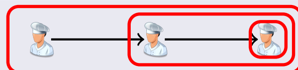
Components

- 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.

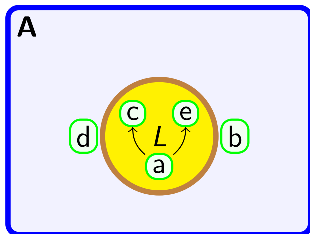
Example



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

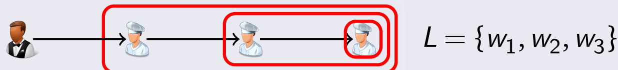
Components

- 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.

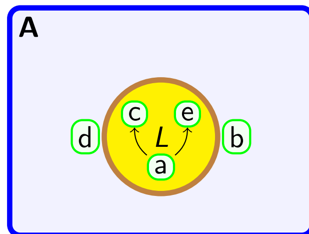
Example



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

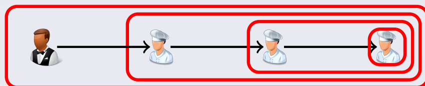
Components

- 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.

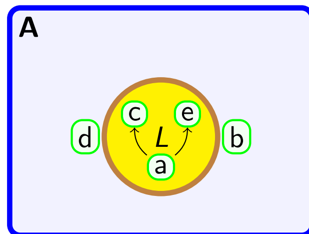
Example



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

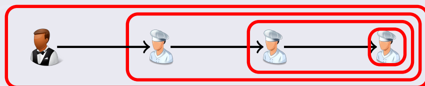
Components

- 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



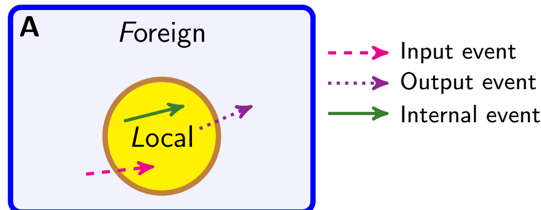
- L is a component if *no* other actors outside L are created by an actor in L.
- L can be obtained from the actor creation tree.

Example



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

Boxing a Component

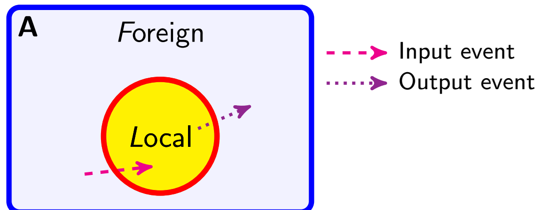


- Hide internal events: boxing
- Each trace is projected to the foreign actors
- Because actors of the same class behave the same, the component can be represented by the *initial* actor class: $[C]$.

Example

$b \to s := \mathtt{new}\ \mathtt{Server} \cdot c \to s.\mathtt{serve}(c, q) \cdot w \to c.\mathtt{response}(\mathrm{compute}(q))$
$\in \mathit{Traces}([\mathtt{Server}])$
where s and w are `Server` and `Worker` actors, respectively.

Boxing a Component

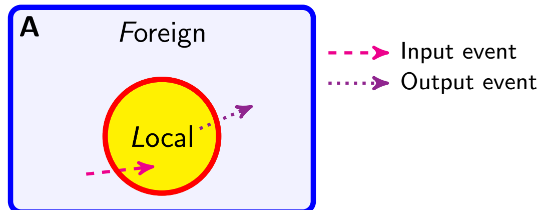


- Hide internal events: boxing
- Each trace is projected to the foreign actors
- Because actors of the same class behave the same, the component can be represented by the *initial* actor class: $[C]$.

Example

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

Boxing a Component

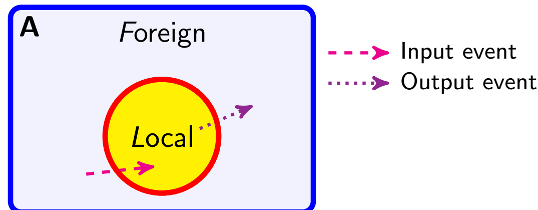


- Hide internal events: boxing
- Each trace is projected to the foreign actors
- Because actors of the same class behave the same, the component can be represented by the *initial* actor class: $[C]$.

Example

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

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

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

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

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

$\forall t \in Traces(D) \bullet split(t, L) = (input, output)$

- 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:

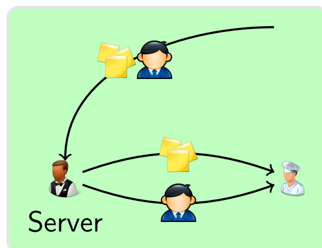
$$\{\ p\ \}\ D\ \{\ q\ \}$$

Partial specification

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

- Informal semantics: $\forall t \in \mathit{Traces}(D) \bullet \mathit{split}(t, L) = (\mathit{input}, \mathit{output})$
  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)

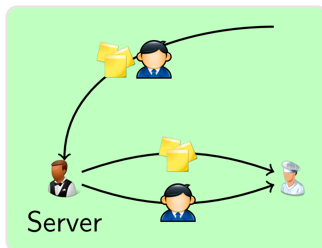
$\{\ \$ = \langle \underline{this} := \textbf{new}\ \texttt{Server} \cdot \underline{this}.\texttt{serve}(\underline{c}, \underline{q})\rangle\ \}$

$\texttt{Server}$

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

- Instead of program, we have a class.

Example – Server Class



$ is the trace constant

Example (Server class specification)

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

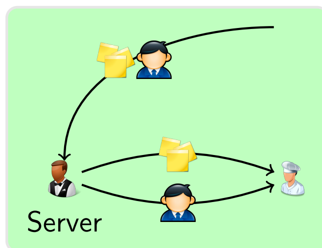
$\quad \texttt{Server}$

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

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)

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

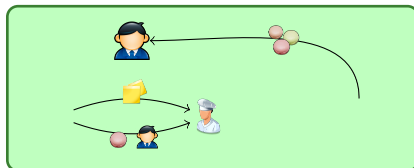
$\texttt{Server}$

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

this, c, q, w are logical variables

- Instead of program, we have a class.

Example – Server and Worker Components



Example (Worker component specification)

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

$[\texttt{Worker}]$

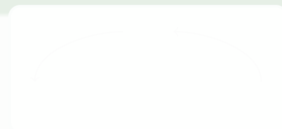
$\{\ \$ = \langle \underline{c}.\texttt{response}(\texttt{merge}(\underline{v}, \texttt{compute}(\underline{q})))\rangle\}$

Example (Server component specification)

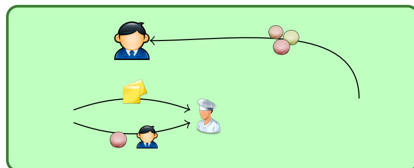
$\{\ \$ = \langle \underline{\text{this}} := \textbf{new}\ \texttt{Server} \cdot \underline{\text{this}}.\texttt{serve}(\underline{c}, \underline{q})\rangle\}$

$[\texttt{Server}]$

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



Example – Server and Worker Components



Example (Worker component specification)

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

$[\texttt{Worker}]$

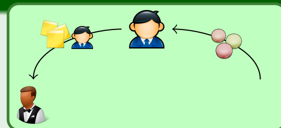
$\{\ \$ = \langle \underline{c}.\texttt{response}(\texttt{merge}(\underline{v}, \texttt{compute}(\underline{q})))\rangle\}$

Example (Server component specification)

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

$[\texttt{Server}]$

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



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

$$\frac{p \implies p_1 \quad \{p_1\}\ D\ \{q_1\} \quad q_1 \implies q}{\{p\}\ D\ \{q\}}$$

INVARIANCE

$$\frac{\{p\}\ D\ \{q\}}{\{p \wedge r\}\ D\ \{q \wedge r\}}$$

where $consFree(r)$

SUBSTITUTION

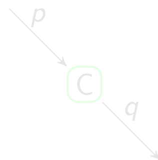
$$\frac{\{p\}\ D\ \{q\}}{\{p[\underline{x}/r]\}\ D\ \{q[\underline{x}/r]\}}$$

where $x \in free(p) \cup free(q)$ and $consFree(r)$

Rules (2)

Boxing

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



BoxedComposition

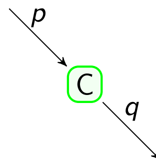
$$\frac{\{p \wedge \underline{i} = \$\}\ C\ \{q \wedge noSelfExposure(\underline{i}, \$)\} \quad \{q'\}\ D\ \{r \wedge nonCr(\$)\} \qquad match(q, q', D)}{\{p\}\ [C]\ \{r\}}$$

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

Rules (2)

Boxing

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



BoxedComposition

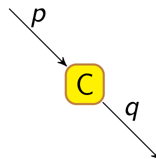
$$\frac{\{p \wedge \underline{i} = \$\}\ C\ \{q \wedge noSelfExposure(\underline{i}, \$)\} \quad \{q'\}\ D\ \{r \wedge nonCr(\$)\} \quad match(q, q', D)}{\{p\}\ [C]\ \{r\}}$$

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

Rules (2)

Boxing

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



BoxedComposition

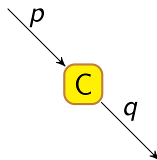
$$\frac{\{p \wedge \underline{i} = \$\}\ C\ \{q \wedge noSelfExposure(\underline{i}, \$)\} \quad \{q'\}\ D\ \{r \wedge nonCr(\$)\} \quad match(q, q', D)}{\{p\}\ [C]\ \{r\}}$$

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

Rules (2)

Boxing

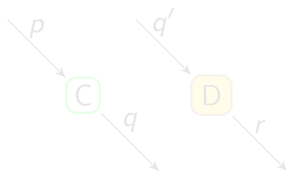
$$\frac{\{p\}\ C\ \{q \wedge nonCreational(\$)\}}{\{p\}\ [C]\ \{q\}}$$



BoxedComposition

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

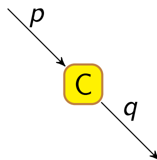
where $\underline{i} \notin free(p) \cup free(q)$



Rules (2)

BOXING

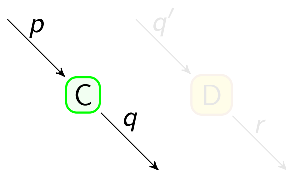
$$\frac{\{p\}\ C\ \{q \wedge nonCreational(\$)\}}{\{p\}\ [C]\ \{q\}}$$



BOXEDCOMPOSITION

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

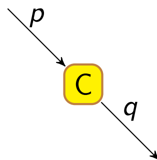
where $\underline{i} \notin free(p) \cup free(q)$



Rules (2)

Boxing

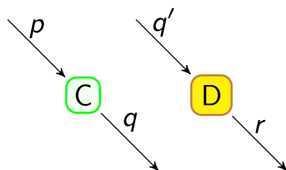
$$\frac{\{p\}\ C\ \{q \wedge nonCreational(\$)\}}{\{p\}\ [C]\ \{q\}}$$



BoxedComposition

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

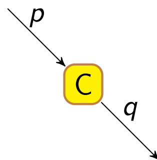
where $\underline{i} \notin free(p) \cup free(q)$



Rules (2)

Boxing

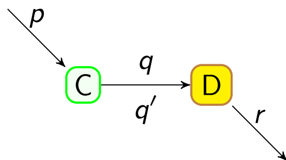
$$\frac{\{p\}\ C\ \{q \wedge nonCreational(\$)\}}{\{p\}\ [C]\ \{q\}}$$



BoxedComposition

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

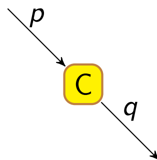
where $\underline{i} \notin free(p) \cup free(q)$



Rules (2)

Boxing

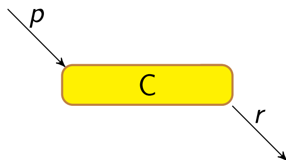
$$\frac{\{p\}\ C\ \{q \wedge nonCreational(\$)\}}{\{p\}\ [C]\ \{q\}}$$



BoxedComposition

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

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



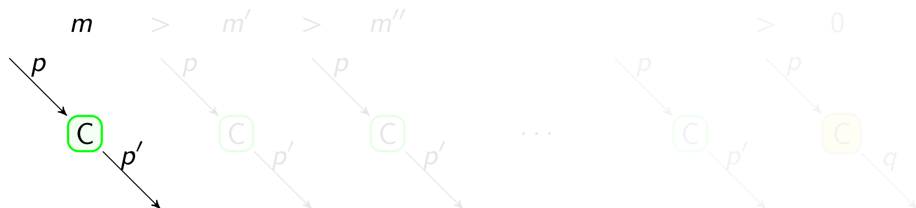
Rule(3)

INDUCTION

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

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

Rule(3)

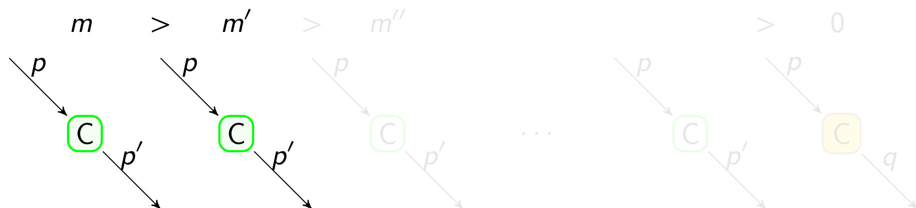


INDUCTION

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

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

Rule(3)

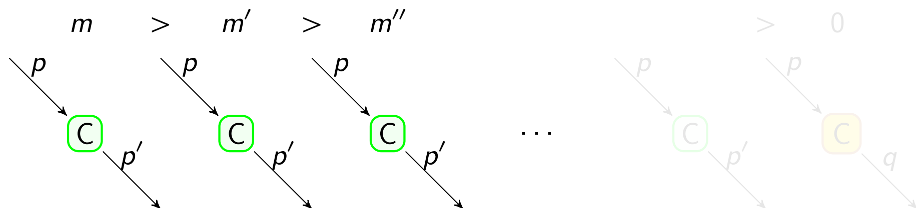


Induction

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

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

Rule(3)

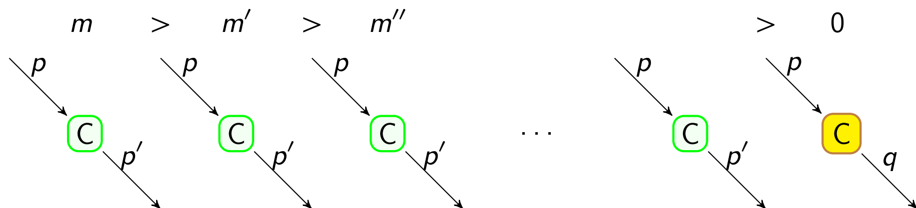


Induction

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

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

Rule(3)

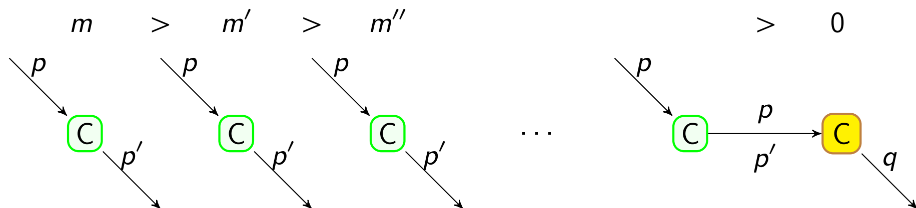


INDUCTION

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

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

Rule(3)

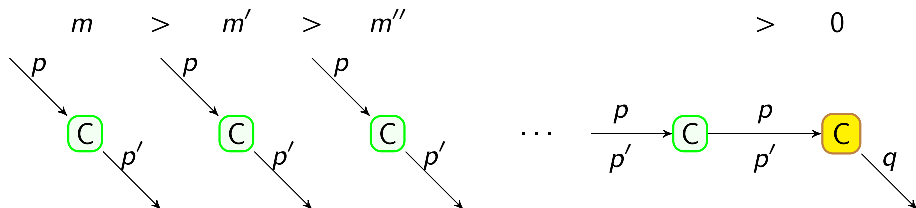


INDUCTION

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

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

Rule(3)

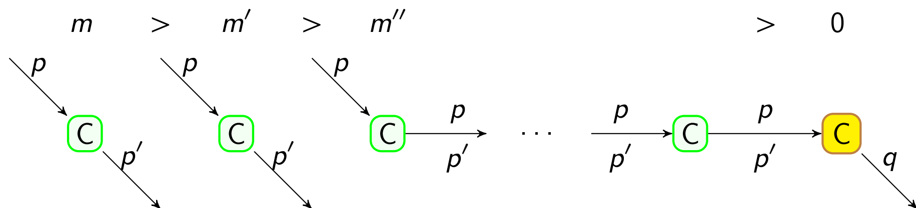


INDUCTION

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

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

Rule(3)

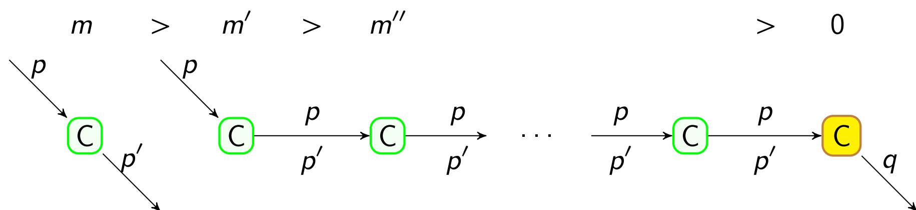


INDUCTION

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

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

Rule(3)

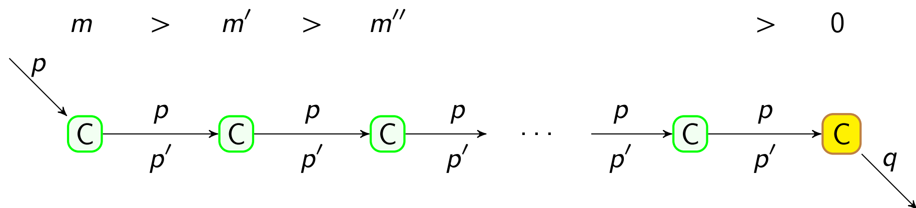


Induction

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

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

Rule(3)

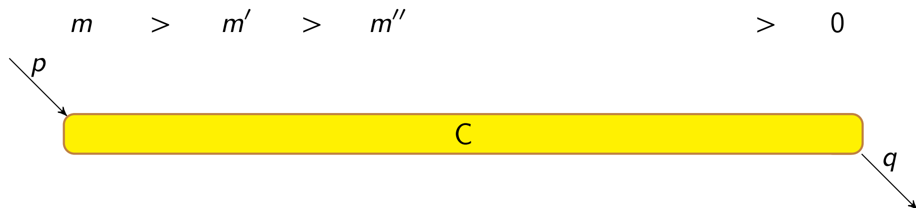


INDUCTION

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

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

Rule(3)



INDUCTION

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

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

Example: Verifying Server Component

Use

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

and

$$\{\ \$ = \langle \underline{\text{this}} := \textbf{new}\ \texttt{Worker} \cdot \underline{\text{this}}.\texttt{do}(\underline{q}) \cdot \underline{\text{this}}.\texttt{propagate}(\underline{v}, \underline{c})\rangle\}$$
$$[\texttt{Worker}]$$
$$\{\ \$ = \langle \underline{c}.\texttt{response}(\texttt{merge}(\underline{v}, \texttt{compute}(\underline{q})))\rangle\}$$

to prove

$$\{\ \$ = \langle \underline{\text{this}} := \textbf{new}\ \texttt{Server} \cdot \underline{\text{this}}.\texttt{serve}(\underline{c}, \underline{q})\rangle\}$$
$$[\texttt{Server}]$$
$$\{\ \$ = \langle \underline{c}.\texttt{response}(\texttt{compute}(\underline{q}))\rangle\}$$

BoxedComposition

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

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

Server Component Verification (1)

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

Proof.

$$\dfrac{\dfrac{\dfrac{}{\{\mathrm{InSrv}\}\ \mathtt{Server}\ \{\exists \underline{w} \bullet \mathrm{OutSrv}\}}\ \textsc{ClassSpec}}{\{\mathrm{InSrv} \wedge \underline{i} = cse(\mathrm{InSrv})\}\ \mathtt{Server}\ \{\exists \underline{w} \bullet \mathrm{OutSrv} \wedge \underline{i} = cse(\mathrm{InSrv})\}}\ \textsc{Inv} \quad \mathrm{InSrv} \wedge \underline{i} = \$ \implies \mathrm{InSrv} \wedge \underline{i} = cse(\mathrm{InSrv}) \quad \exists \underline{w} \bullet \mathrm{OutSrv} \wedge \underline{i} = cse(\mathrm{InSrv}) \implies \exists \underline{w} \bullet \mathrm{OutSrv} \wedge noSelfExp(\underline{i}, \$)}{\{\mathrm{InSrv} \wedge \underline{i} = \$\}\ \mathtt{Server}\ \{\exists \underline{w} \bullet \mathrm{OutSrv} \wedge noSelfExp(\underline{i}, \$)\}}\ \textsc{Cons}$$

BoxedComposition

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

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

Server Component Verification (1)

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

Proof.

$$\textsc{Inv}\ \dfrac{\textsc{ClassSpec}\ \dfrac{}{\{\mathsf{InSrv}\}\ \mathtt{Server}\ \{\exists \underline{\mathsf{w}} \bullet \mathsf{OutSrv}\}}}{\{\mathsf{InSrv} \wedge \underline{\mathsf{i}} = \mathit{cse}(\mathsf{InSrv})\}\ \mathtt{Server}\ \{\exists \underline{\mathsf{w}} \bullet \mathsf{OutSrv} \wedge \underline{\mathsf{i}} = \mathit{cse}(\mathsf{InSrv})\}}$$

$$\mathsf{InSrv} \wedge \underline{\mathsf{i}} = \$ \implies \mathsf{InSrv} \wedge \underline{\mathsf{i}} = \mathit{cse}(\mathsf{InSrv})$$

$$\exists \underline{\mathsf{w}} \bullet \mathsf{OutSrv} \wedge \underline{\mathsf{i}} = \mathit{cse}(\mathsf{InSrv}) \implies \exists \underline{\mathsf{w}} \bullet \mathsf{OutSrv} \wedge \mathit{noSelfExp}(\underline{\mathsf{i}}, \$)$$

$$\textsc{Cons}\ \dfrac{\cdots}{\{\mathsf{InSrv} \wedge \underline{\mathsf{i}} = \$\}\ \mathtt{Server}\ \{\exists \underline{\mathsf{w}} \bullet \mathsf{OutSrv} \wedge \mathit{noSelfExp}(\underline{\mathsf{i}}, \$)\}}$$

BoxedComposition

$$\dfrac{\{p \wedge \underline{\mathsf{i}} = \$\}\ C\ \{q \wedge \mathit{noSelfExp}(\underline{\mathsf{i}}, \$)\} \quad \{q'\}\ D\ \{r \wedge \mathit{nonCr}(\$)\} \quad \mathit{match}(q, q', D)}{\{p\}\ [C]\ \{r\}}$$

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

Server Component Verification (1)

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

Proof.

$$\dfrac{\dfrac{\dfrac{}{\{\mathsf{InSrv}\}\ \mathtt{Server}\ \{\exists \underline{w} \bullet \mathsf{OutSrv}\}}\ \textsc{ClassSpec}}{\{\mathsf{InSrv} \land \underline{i} = \mathit{cse}(\mathsf{InSrv})\}\ \mathtt{Server}\ \{\exists \underline{w} \bullet \mathsf{OutSrv} \land \underline{i} = \mathit{cse}(\mathsf{InSrv})\}}\ \textsc{Inv} \qquad \begin{array}{c}\mathsf{InSrv} \land \underline{i} = \$ \implies \mathsf{InSrv} \land \underline{i} = \mathit{cse}(\mathsf{InSrv}) \\ \exists \underline{w} \bullet \mathsf{OutSrv} \land \underline{i} = \mathit{cse}(\mathsf{InSrv}) \implies \exists \underline{w} \bullet \mathsf{OutSrv} \land \mathit{noSelfExp}(\underline{i}, \$)\end{array}}{\{\mathsf{InSrv} \land \underline{i} = \$\}\ \mathtt{Server}\ \{\exists \underline{w} \bullet \mathsf{OutSrv} \land \mathit{noSelfExp}(\underline{i}, \$)\}}\ \textsc{Cons}$$

$\square$

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

Server Component Verification (1)

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

Proof.

$$\textsc{Cons}\ \dfrac{\textsc{Inv}\ \dfrac{\textsc{ClassSpec}\ \dfrac{}{\{\mathsf{InSrv}\}\ \mathtt{Server}\ \{\exists \underline{\mathsf{w}} \bullet \mathsf{OutSrv}\}}}{\{\mathsf{InSrv} \land \underline{\mathsf{i}} = cse(\mathsf{InSrv})\}\ \mathtt{Server}\ \{\exists \underline{\mathsf{w}} \bullet \mathsf{OutSrv} \land \underline{\mathsf{i}} = cse(\mathsf{InSrv})\}} \quad \begin{array}{c} \mathsf{InSrv} \land \underline{\mathsf{i}} = \$ \implies \mathsf{InSrv} \land \underline{\mathsf{i}} = cse(\mathsf{InSrv}) \\ \exists \underline{\mathsf{w}} \bullet \mathsf{OutSrv} \land \underline{\mathsf{i}} = cse(\mathsf{InSrv}) \implies \exists \underline{\mathsf{w}} \bullet \mathsf{OutSrv} \land noSelfExp(\underline{\mathsf{i}}, \$) \end{array}}{\{\mathsf{InSrv} \land \underline{\mathsf{i}} = \$\}\ \mathtt{Server}\ \{\exists \underline{\mathsf{w}} \bullet \mathsf{OutSrv} \land noSelfExp(\underline{\mathsf{i}}, \$)\}}$$

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

Server Component Verification (2)

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

Proof.

$$\dfrac{\dfrac{\{\text{InWrk}\}\ [\texttt{Worker}]\ \{\text{OutWrk}\}}{\{\text{InWrk}[\underline{v}/\texttt{null}]\}\ [\texttt{Worker}]\ \{\text{OutWrk}[\underline{v}/\texttt{null}]\}}\ \textsc{Sub} \qquad \text{OutWrk}[\underline{v}/\texttt{null}] \implies \text{OutSrvC} \wedge \mathit{nonCr}(\$)}{\{\text{InWrk}[\underline{v}/\texttt{null}]\}\ [\texttt{Worker}]\ \{\text{OutSrvC} \wedge \mathit{nonCr}(\$)\}}\ \textsc{Cons}$$

$$\textsc{BoxedComposition}\quad \dfrac{\{p \wedge \underline{i} = \$\}\ C\ \{q \wedge \mathit{noSelfExp}(\underline{i}, \$)\} \quad \{q'\}\ D\ \{r \wedge \mathit{nonCr}(\$)\} \quad \mathit{match}(q, q', D)}{\{p\}\ [C]\ \{r\}}$$
where $\underline{i} \notin \mathit{free}(p) \cup \mathit{free}(q)$

Server Component Verification (2)

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

Proof.

$$\dfrac{\dfrac{\{\mathsf{InWrk}\}\ [\mathtt{Worker}]\ \{\mathsf{OutWrk}\}}{\{\mathsf{InWrk}[\underline{\mathtt{v}}/\mathtt{null}]\}\ [\mathtt{Worker}]\ \{\mathsf{OutWrk}[\underline{\mathtt{v}}/\mathtt{null}]\}}\ \textsc{Sub} \qquad \mathsf{OutWrk}[\underline{\mathtt{v}}/\mathtt{null}] \implies \mathsf{OutSrvC} \wedge nonCr(\$)}{\{\mathsf{InWrk}[\underline{\mathtt{v}}/\mathtt{null}]\}\ [\mathtt{Worker}]\ \{\mathsf{OutSrvC} \wedge nonCr(\$)\}}\ \textsc{Cons}$$

BoxedComposition

$$\dfrac{\{p \wedge \underline{\mathrm{i}} = \$\}\ C\ \{q \wedge noSelfExp(\underline{\mathrm{i}}, \$)\} \quad \{q'\}\ D\ \{r \wedge nonCr(\$)\} \quad match(q, q', D)}{\{p\}\ [C]\ \{r\}}$$
where $\underline{\mathrm{i}} \notin free(p) \cup free(q)$

□

Server Component Verification (2)

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

Proof.

$$\textsc{Cons}\ \dfrac{\textsc{Sub}\ \dfrac{\{\mathsf{InWrk}\}\ [\mathtt{Worker}]\ \{\mathsf{OutWrk}\}}{\{\mathsf{InWrk}[\underline{\mathtt{v}}/\mathbf{null}]\}\ [\mathtt{Worker}]\ \{\mathsf{OutWrk}[\underline{\mathtt{v}}/\mathbf{null}]\}} \quad \mathsf{OutWrk}[\underline{\mathtt{v}}/\mathbf{null}] \implies \mathsf{OutSrvC} \land nonCr(\$)}{\{\mathsf{InWrk}[\underline{\mathtt{v}}/\mathbf{null}]\}\ [\mathtt{Worker}]\ \{\mathsf{OutSrvC} \land nonCr(\$)\}}$$

$$\textsc{BoxedComposition}\ \dfrac{\{p \land \underline{\mathtt{i}} = \$\}\ C\ \{q \land noSelfExp(\underline{\mathtt{i}}, \$)\} \quad \{q'\}\ D\ \{r \land nonCr(\$)\} \quad match(q, q', D)}{\{p\}\ [C]\ \{r\}}$$
where $\underline{\mathtt{i}} \notin free(p) \cup free(q)$

Server Component Verification (2)

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

Proof.

$$\textsc{Cons}\ \dfrac{\textsc{Sub}\ \dfrac{\{\mathsf{InWrk}\}\ [\mathtt{Worker}]\ \{\mathsf{OutWrk}\}}{\{\mathsf{InWrk}[\underline{\mathtt{v}}/\mathbf{null}]\}\ [\mathtt{Worker}]\ \{\mathsf{OutWrk}[\underline{\mathtt{v}}/\mathbf{null}]\}} \quad \mathsf{OutWrk}[\underline{\mathtt{v}}/\mathbf{null}] \Longrightarrow \mathsf{OutSrvC} \wedge nonCr(\$)}{\{\mathsf{InWrk}[\underline{\mathtt{v}}/\mathbf{null}]\}\ [\mathtt{Worker}]\ \{\mathsf{OutSrvC} \wedge nonCr(\$)\}}$$

$$\textsc{BoxedComposition}\ \dfrac{\{p \wedge \underline{\mathtt{i}} = \$\}\ C\ \{q \wedge noSelfExp(\underline{\mathtt{i}}, \$)\} \quad \{q'\}\ D\ \{r \wedge nonCr(\$)\} \quad match(q, q', D)}{\{p\}\ [C]\ \{r\}}$$

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

Server Component Verification (3)

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

Proof.

$$\textsc{BoxCmp}\ \frac{\begin{array}{c}\{\mathsf{InSrv} \wedge \underline{i} = \$\}\ \mathtt{Server}\ \{\exists \underline{w} \bullet \mathsf{OutSrv} \wedge \mathit{noSelfExp}(\underline{i}, \$)\} \\ \{\mathsf{InWrk}[\underline{v}/\mathtt{\textbf{null}}]\}\ [\mathtt{Worker}]\ \{\mathsf{OutSrvC} \wedge \mathit{nonCr}(\$)\} \\ \mathit{match}(\exists \underline{w} \bullet \mathsf{OutSrv}, \mathsf{InWrk}[\underline{v}/\mathtt{\textbf{null}}], [\mathtt{Worker}])\end{array}}{\{\mathsf{InSrvC}\}\ [\mathtt{Server}]\ \{\mathsf{OutSrvC}\}}$$

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

Worker component is verified using BOXING and INDUCTION.

BOXEDCOMPOSITION

$$\frac{\begin{array}{c}\{p \wedge \underline{i} = \$\}\ C\ \{q \wedge \mathit{noSelfExp}(\underline{i}, \$)\} \\ \{q'\}\ D\ \{r \wedge \mathit{nonCr}(\$)\} \\ \mathit{match}(q, q', D)\end{array}}{\{p\}\ [C]\ \{r\}}$$

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

Server Component Verification (3)

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

Proof.

$$\textsc{BoxCmp}\ \frac{\begin{array}{c}\{\mathsf{InSrv} \wedge \underline{\mathsf{i}} = \$\}\ \mathtt{Server}\ \{\exists \underline{\mathsf{w}} \bullet \mathsf{OutSrv} \wedge \mathit{noSelfExp}(\underline{\mathsf{i}}, \$)\} \\ \{\mathsf{InWrk}[\underline{\mathsf{v}}/\mathbf{null}]\}\ [\mathtt{Worker}]\ \{\mathsf{OutSrvC} \wedge \mathit{nonCr}(\$)\} \\ \mathit{match}(\exists \underline{\mathsf{w}} \bullet \mathsf{OutSrv}, \mathsf{InWrk}[\underline{\mathsf{v}}/\mathbf{null}], [\mathtt{Worker}])\end{array}}{\{\mathsf{InSrvC}\}\ [\mathtt{Server}]\ \{\mathsf{OutSrvC}\}}$$

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

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

Worker component is verified using Boxing and Induction.

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!