# Interactive Verification of Concurrent Systems

Philipp Rümmer

ph_r@gmx.net

University of Karlsruhe

Institute for Logic, Complexity and Deduction Systems

D-76128 Karlsruhe, Germany

## Overview

- Introduction to CSP:

## Overview

- Introduction to CSP:
  - Concept, basic operators

## Overview

- Introduction to CSP:
  - Concept, basic operators
- Introduction to JCSP:

## Overview

- Introduction to CSP:
  - Concept, basic operators
- Introduction to JCSP:
  - Basic classes

## Overview

- Introduction to CSP:
  - Concept, basic operators
- Introduction to JCSP:
  - Basic classes
- Approach for verification of JCSP systems:

## Overview

- Introduction to CSP:
  - Concept, basic operators
- Introduction to JCSP:
  - Basic classes
- Approach for verification of JCSP systems:
  - Representation of systems

## Overview

- Introduction to CSP:
  - Concept, basic operators
- Introduction to JCSP:
  - Basic classes
- Approach for verification of JCSP systems:
  - Representation of systems
  - Concept of a calculus

## Communicating Sequential Processes

- *Process Algebra*, originally devised by Tony Hoare (1978)

## Communicating Sequential Processes

- *Process Algebra*, originally devised by Tony Hoare (1978)
- Formalism to design/describe interacting systems

## Communicating Sequential Processes

- *Process Algebra*, originally devised by Tony Hoare (1978)
- Formalism to design/describe interacting systems
- Today widely used to model protocols or hardware

## Communicating Sequential Processes

- *Process Algebra*, originally devised by Tony Hoare (1978)
- Formalism to design/describe interacting systems
- Today widely used to model protocols or hardware
- Analysis (mostly) through model checking (e.g. the FDR model checker)

## Introduction to CSP: Processes

- Processes pose central entity of CSP concept

## Introduction to CSP: Processes

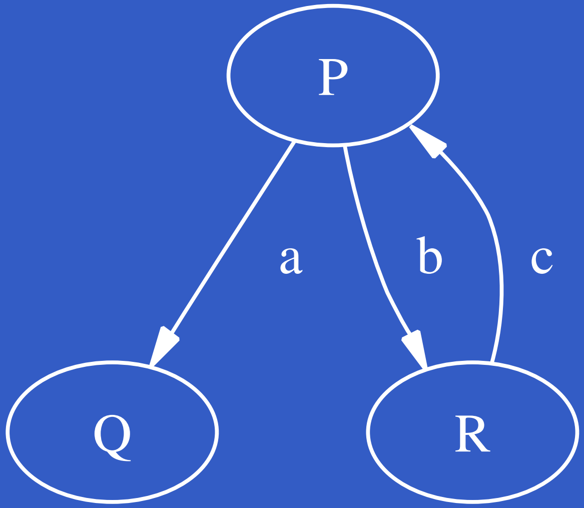
- Processes pose central entity of CSP concept
- A process is described uniquely by its potential communication with an environment

## Introduction to CSP: Processes

- Processes pose central entity of CSP concept
- A process is described uniquely by its potential communication with an environment
- Communication is a sequence of atomic *events* (e.g. Processes $\triangleq$ Languages)

## Automata as Processes

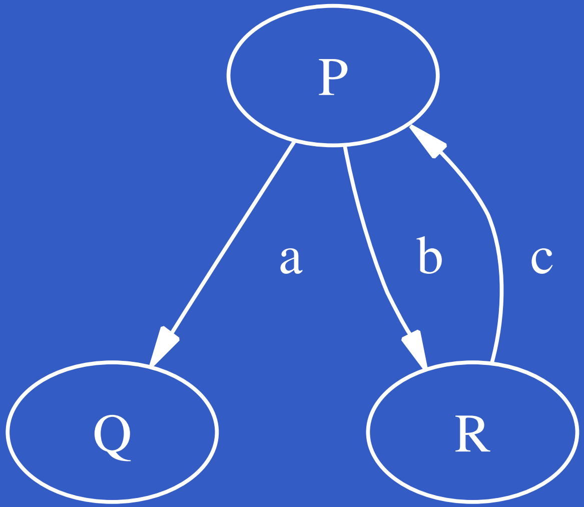
- States of automata can be regarded as processes:



$$\begin{tikzcd}
 & P \arrow[dl, "a"] \arrow[dr, bend right=10, "b"] & \\
Q & & R \arrow[ul, bend right=40, "c"']
\end{tikzcd}$$

## Automata as Processes

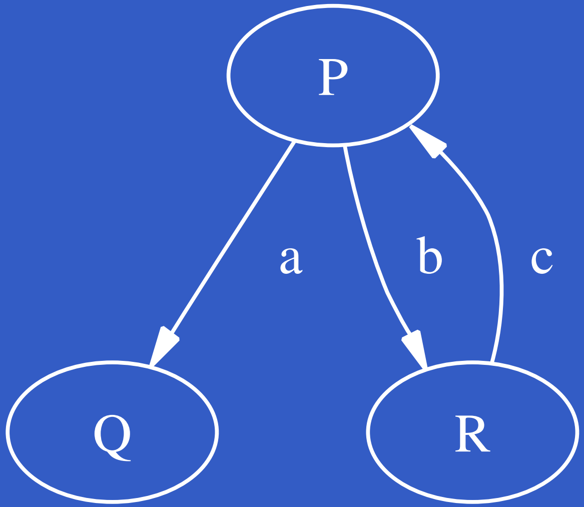
- States of automata can be regarded as processes:



$$\mathit{traces}\,[\![Q]\!] = \{\epsilon\}$$

## Automata as Processes

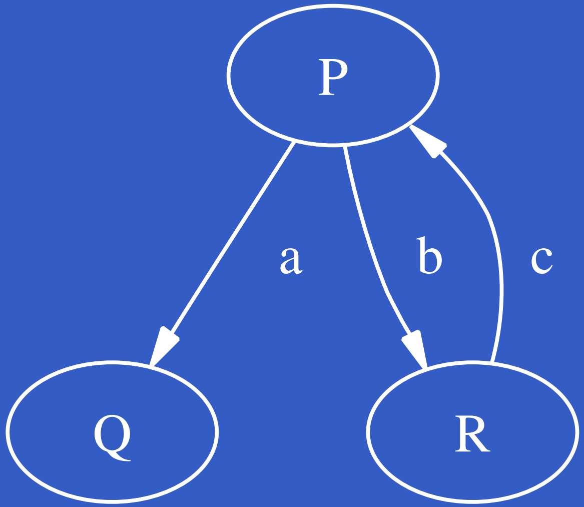
- States of automata can be regarded as processes:



$$\mathit{traces}\,[\![Q]\!] = \{\epsilon\}$$
$$\mathit{traces}\,[\![P]\!] = \{a, bca, \ldots\}$$

## Automata as Processes

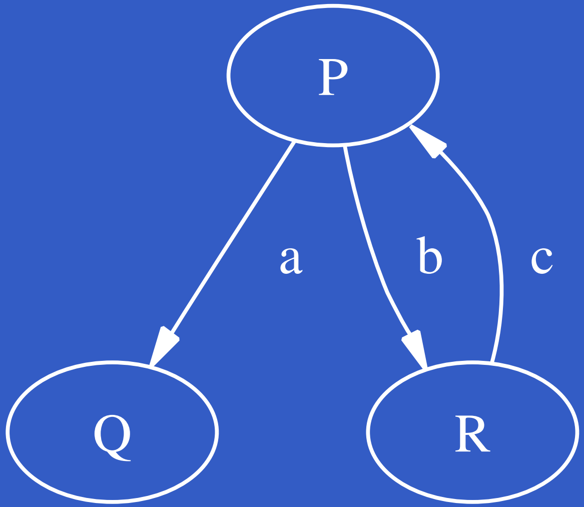
- States of automata can be regarded as processes:



$$\begin{aligned} traces\,[\![Q]\!] &= \{\epsilon\} \\ traces\,[\![P]\!] &= \{a, bca, \ldots\} \\ traces\,[\![R]\!] &= \{ca, cbca, \ldots\} \end{aligned}$$

## Automata as Processes

- States of automata can be regarded as processes:



$$\begin{aligned} \mathit{traces}\,[\![Q]\!] &= \{\epsilon\} \\ \mathit{traces}\,[\![P]\!] &= \{a, bca, \ldots\} \\ \mathit{traces}\,[\![R]\!] &= \{ca, cbca, \ldots\} \end{aligned}$$

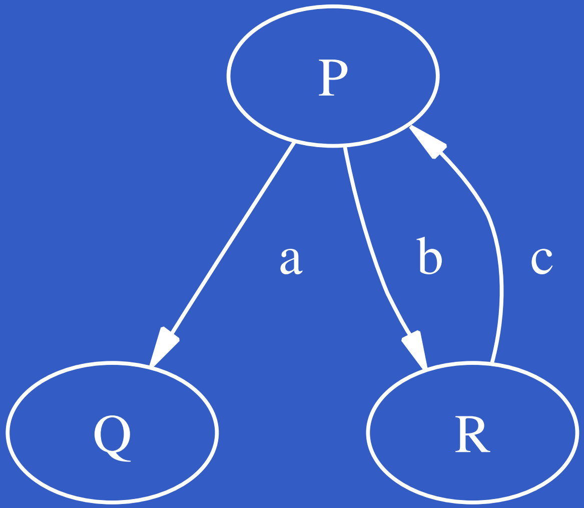
- Generalised as *labelled transition systems* (LTS), usually infinite

## CSP Terms

- Processes are usually described through CSP terms

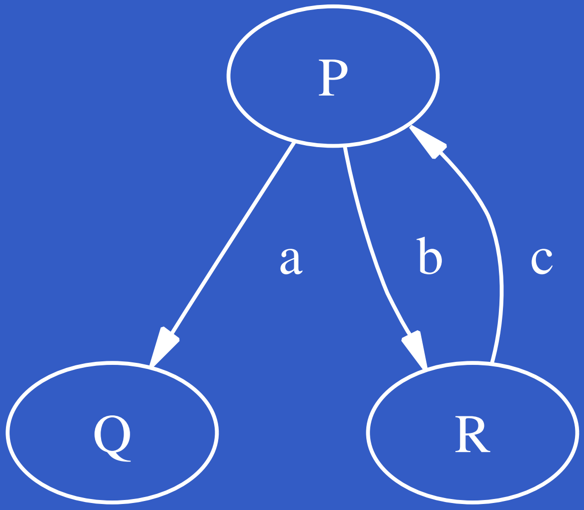
## CSP Terms

- Processes are usually described through CSP terms
- Definition of automaton using terms:



## CSP Terms

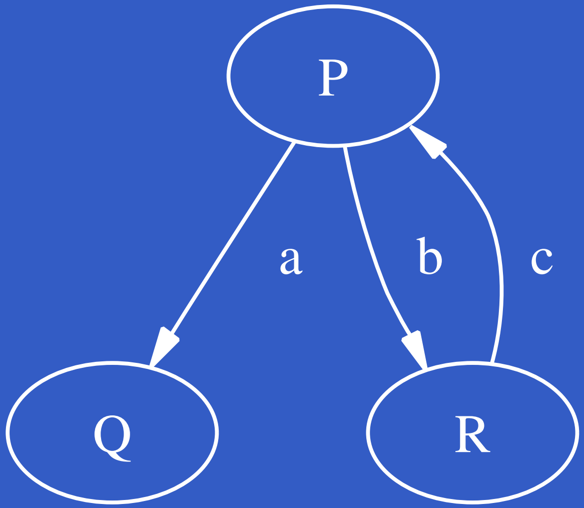
- Processes are usually described through CSP terms
- Definition of automaton using terms:



$$Q = \mathit{stop}$$

## CSP Terms

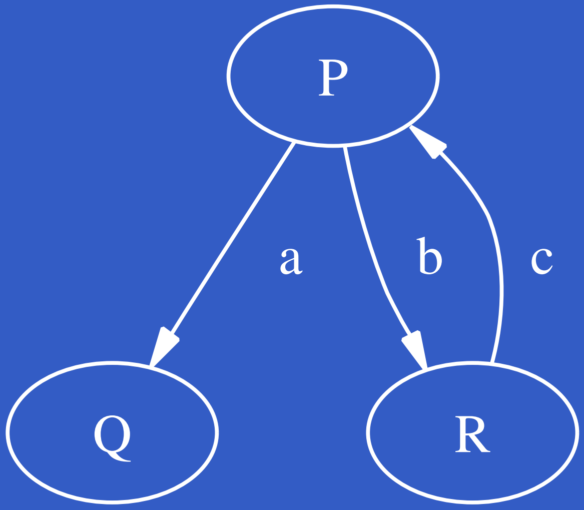
- Processes are usually described through CSP terms
- Definition of automaton using terms:



$$\begin{aligned} Q &= \mathit{stop} \\ R &= c \longrightarrow P \end{aligned}$$

## CSP Terms

- Processes are usually described through CSP terms
- Definition of automaton using terms:



$$\begin{aligned} Q &= \mathit{stop} \\ R &= c \longrightarrow P \\ P &= (a \longrightarrow Q) \,\Box\, (b \longrightarrow R) \end{aligned}$$

## Basic CSP Operators: Locked Process

- Term notation:

$$\mathit{stop}$$

## Basic CSP Operators: Locked Process

- Term notation:

$$\mathit{stop}$$

- LTS appearance:

$$\boxed{\mathit{stop}}$$

## Basic CSP Operators: Prefixing

- Term notation:

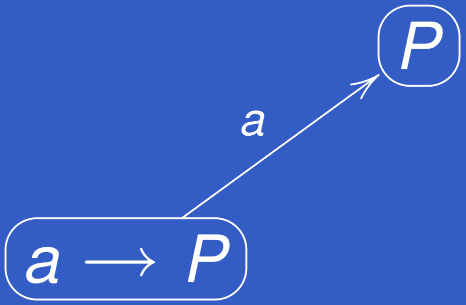
$$a \longrightarrow P$$

## Basic CSP Operators: Prefixing

- Term notation:

$$a \longrightarrow P$$

- LTS appearance:



## Basic CSP Operators: Choice

- Term notation:

$$P \Box Q$$

## Basic CSP Operators: Choice

- Term notation:

$$P \,\square\, Q$$

- LTS appearance:

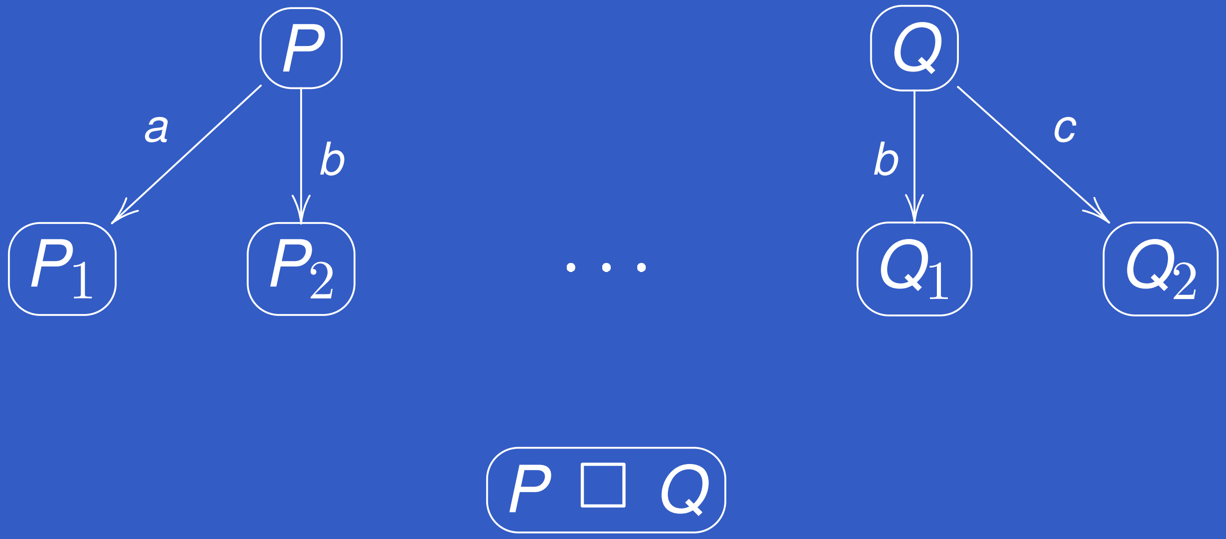
$$\begin{tikzcd}
 & \boxed{P} \arrow[dl, "a"'] \arrow[d, "b"] & & \boxed{Q} \arrow[d, "b"'] \arrow[dr, "c"] & \\
\boxed{P_1} & \boxed{P_2} & \cdots & \boxed{Q_1} & \boxed{Q_2}
\end{tikzcd}$$

## Basic CSP Operators: Choice

- Term notation:

$$P \,\square\, Q$$

- LTS appearance:

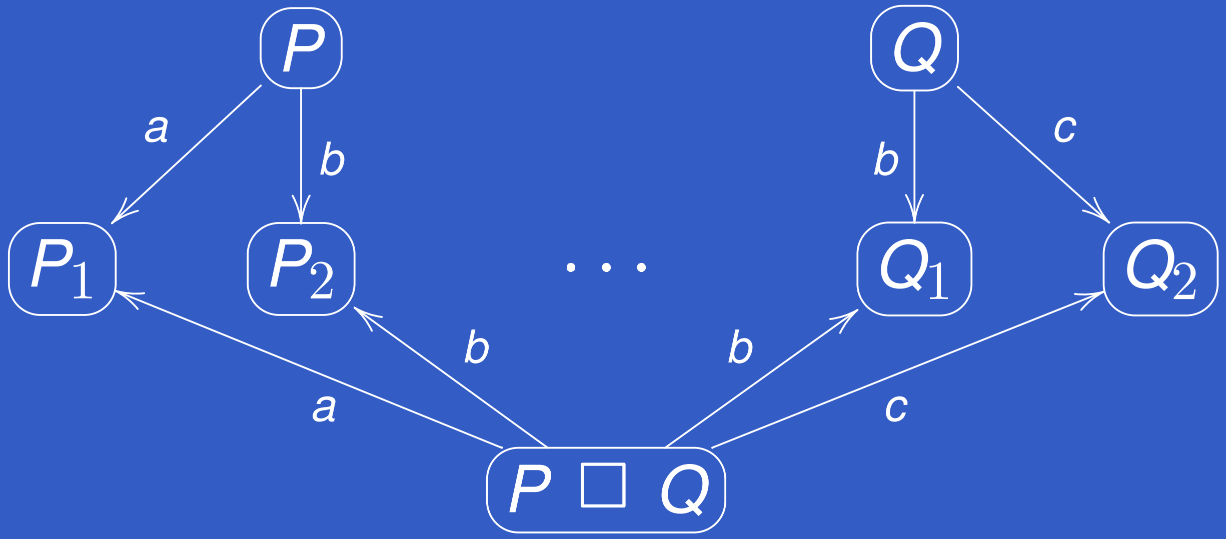


## Basic CSP Operators: Choice

- Term notation:

$$P \,\Box\, Q$$

- LTS appearance:


$$\begin{tikzcd}
 & P \arrow[dl, "a"'] \arrow[d, "b"] & & Q \arrow[d, "b"'] \arrow[dr, "c"] & \\
P_1 & P_2 & \cdots & Q_1 & Q_2 \\
 & & P \,\Box\, Q \arrow[ull, "a"] \arrow[ul, "b"'] \arrow[ur, "b"] \arrow[urr, "c"'] & &
\end{tikzcd}$$

## Basic CSP Operators: Parallelism

- Term notation: ($X$: the *interface* set)

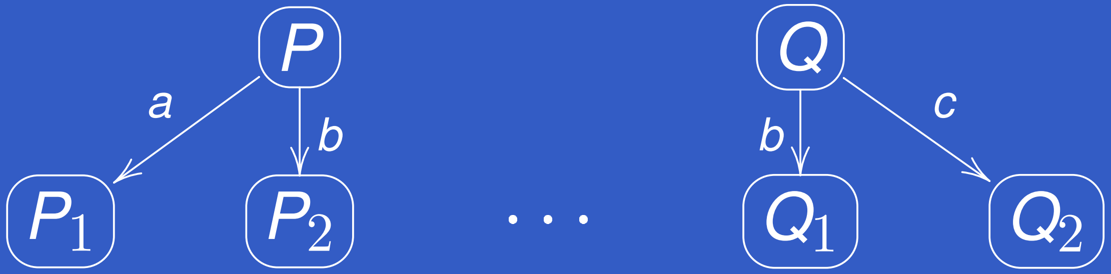
$$P \mathrel{[\![} X \mathrel{]\!]} Q$$

## Basic CSP Operators: Parallelism

- Term notation: ($X$: the *interface* set)

$$P \mathbin{[\![ X ]\!]} Q$$

- LTS appearance (for $X = \{b\}$):


$$\begin{tikzcd} & \boxed{P} \arrow[dl, "a"'] \arrow[d, "b"] & & & \boxed{Q} \arrow[d, "b"'] \arrow[dr, "c"] & \\ \boxed{P_1} & \boxed{P_2} & \cdots & & \boxed{Q_1} & \boxed{Q_2} \end{tikzcd}$$

## Basic CSP Operators: Parallelism

- Term notation: ($X$: the *interface* set)

$$P \mathbin{\|[} X \mathbin{]\|} Q$$

- LTS appearance (for $X = \{b\}$):

$$\begin{tikzcd}
 & P \arrow[dl, "a"'] \arrow[d, "b"] & & Q \arrow[d, "b"'] \arrow[dr, "c"] & \\
P_1 & P_2 & \cdots & Q_1 & Q_2
\end{tikzcd}$$

$$\boxed{P \mathbin{\|[} X \mathbin{]\|} Q}$$

# Basic CSP Operators: Parallelism

- Term notation: ($X$: the *interface* set)

$$P \mathrel{|[} X \mathrel{]|} Q$$

- LTS appearance (for $X = \{b\}$):

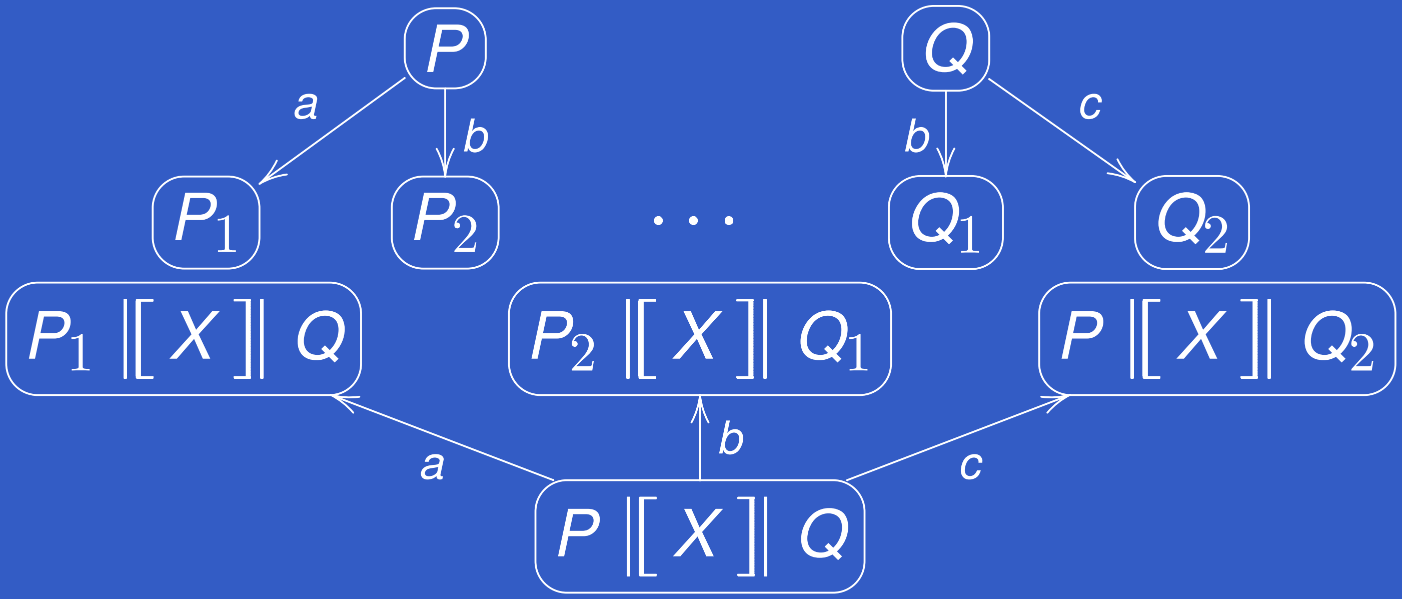
$$\begin{tikzcd}
 & P \arrow[dl, "a"'] \arrow[d, "b"] & & Q \arrow[d, "b"'] \arrow[dr, "c"] & \\
P_1 & P_2 & \cdots & Q_1 & Q_2 \\
P_1 \mathrel{|[} X \mathrel{]|} Q & & & & P \mathrel{|[} X \mathrel{]|} Q_2 \\
 & & P \mathrel{|[} X \mathrel{]|} Q \arrow[ull, "a"] \arrow[urr, "c"'] & &
\end{tikzcd}$$

## Basic CSP Operators: Parallelism

- Term notation: ($X$: the *interface* set)

$$P \; [\![ X ]\!] \; Q$$

- LTS appearance (for $X = \{b\}$):



$$\begin{tikzcd} & P \arrow[dl, "a"'] \arrow[d, "b"] & & Q \arrow[d, "b"'] \arrow[dr, "c"] & \\ P_1 & P_2 & \cdots & Q_1 & Q_2 \\ P_1 \; [\![ X ]\!] \; Q & & P_2 \; [\![ X ]\!] \; Q_1 & & P \; [\![ X ]\!] \; Q_2 \\ & & P \; [\![ X ]\!] \; Q \arrow[ull, "a"] \arrow[u, "b"'] \arrow[urr, "c"'] & & \end{tikzcd}$$

## Basic CSP Operators: Parallelism

- Parallelism $\Longrightarrow$ Product of LTSs

## Basic CSP Operators: Parallelism

- Parallelism $\Longrightarrow$ Product of LTSs
- Full synchronisation $P \mathbin{[\![} A/ \mathbin{]\!]} Q \Longrightarrow$ Intersection of languages

## Basic CSP Operators: Parallelism

- Parallelism $\Longrightarrow$ Product of LTSs
- Full synchronisation $P \,[\![\, A I \,]\!]\, Q \Longrightarrow$ Intersection of languages
- Shorter notation for interleaving:

$$P \,|\!|\!|\, Q \quad := \quad P \,[\![\, \emptyset \,]\!]\, Q$$

## Basic CSP Operators: Messages

- Transmission of values as events

## Basic CSP Operators: Messages

- Transmission of values as events
- Sending message $v$:

$$x \longrightarrow P \quad = \quad !x \longrightarrow P$$

## Basic CSP Operators: Messages

- Transmission of values as events
- Sending message $v$:

$$x \longrightarrow P \quad = \quad !x \longrightarrow P$$

- Reading of messages: Generalised Choice

$$?x : A \longrightarrow P(x) :=$$
$$(v_1 \longrightarrow P(v_1)) \; \Box \; (v_2 \longrightarrow P(v_2)) \; \Box \; \cdots$$

(where $A = \{v_1, v_2, \ldots\}$)

## Example

- A process computing successors:

$$S = ?n \longrightarrow !(n+1) \longrightarrow S \qquad (n \in \mathbb{N})$$

## Example

- A process computing successors:

$$S = ?n \rightarrow !(n+1) \rightarrow S \qquad (n \in \mathbb{N})$$

- LTS view:

$$\begin{tikzcd}
 & S \arrow[dl, bend right, "0"'] \arrow[dr, bend right, "n"'] & & \\
S_0 \arrow[ur, bend right, "1"'] & \cdots & S_n \arrow[ul, bend right, "n+1"'] & \cdots
\end{tikzcd}$$

## Example (2)

- Communication between processes:

$$A = \,!41 \longrightarrow \,?n \longrightarrow A(n)$$

## Example (2)

- Communication between processes:

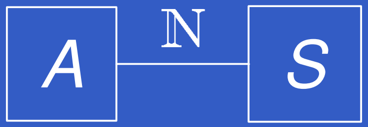
$$A = \,!41 \longrightarrow \,?n \longrightarrow A(n)$$
$$B = A \,\|[\, \mathbb{N} \,]\|\, S$$

## Example (2)

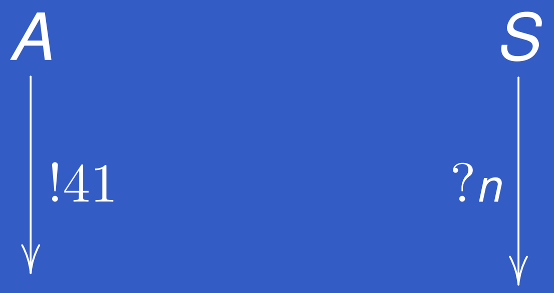
- Communication between processes:

$$A = \,!41 \longrightarrow \,?n \longrightarrow A(n)$$
$$B = A \,\|[\, \mathbb{N} \,]\|\, S$$

-



## Example (3)



$$S = ?n \longrightarrow !(n+1) \longrightarrow S$$
$$A = !41 \longrightarrow ?n \longrightarrow A(n)$$

## Example (3)

$$\begin{tikzcd}
A \arrow[d, dash] & & S \arrow[d, dash] \\
41 \arrow[rr, dotted] \arrow[d] & & 41 \arrow[d] \\
?n \longrightarrow A(n) \arrow[d, "?n"] & & !42 \longrightarrow S \arrow[d, "!42"'] \\
{} & & {}
\end{tikzcd}$$

$$S = ?n \longrightarrow !(n+1) \longrightarrow S$$
$$A = !41 \longrightarrow ?n \longrightarrow A(n)$$

## Example (3)

$$\begin{tikzcd}
A \arrow[dd] & & & S \arrow[dd] \\
& 41 \arrow[r, dotted] & 41 & \\
?n \longrightarrow A(n) \arrow[dd] & & & !42 \longrightarrow S \arrow[dd] \\
& 42 & 42 \arrow[l, dotted] & \\
A(42) \arrow[d, dotted] & & & S \arrow[d, dotted] \\
{} & & & {}
\end{tikzcd}$$

$$S = ?n \longrightarrow !(n+1) \longrightarrow S$$
$$A = !41 \longrightarrow ?n \longrightarrow A(n)$$

## CSP for Java (JCSP)

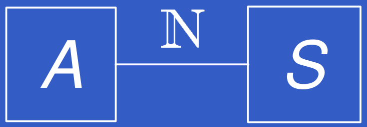
- Implementation of CSP process model in Java by P. D. Austin and P. H. Welch

## CSP for Java (JCSP)

- Implementation of CSP process model in Java by P. D. Austin and P. H. Welch
- Very similar to the Occam language

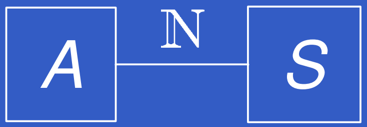
## JCSP Example

- CSP example: $A \,[\![\, \mathbb{N} \,]\!]\, S$

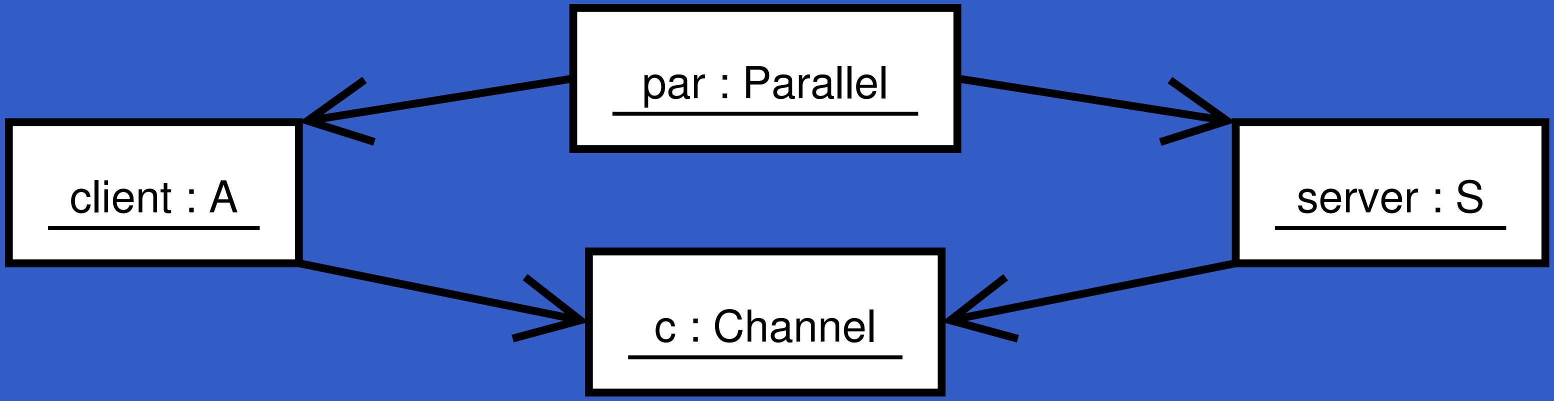


## JCSP Example

- CSP example: $A \, [\![ \, \mathbb{N} \, ]\!] \, S$

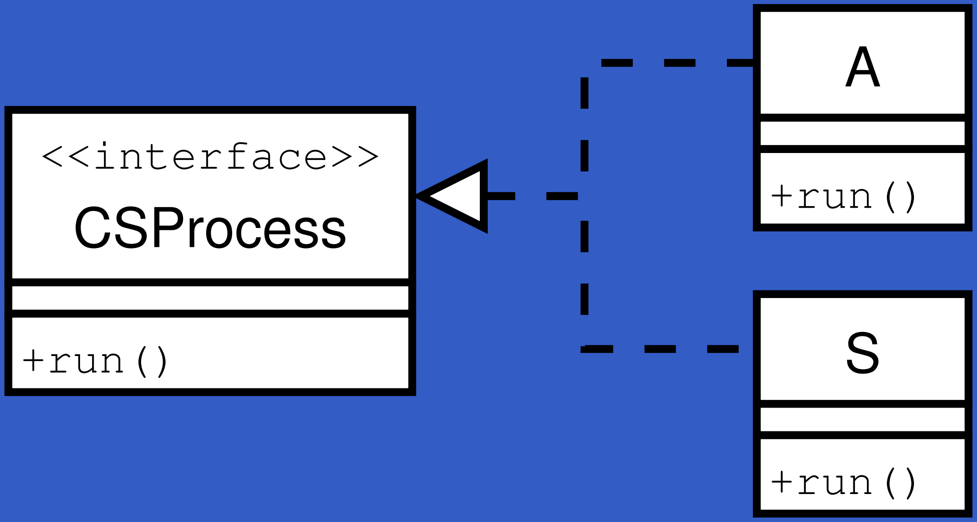


- Corresponding JCSP object diagram:



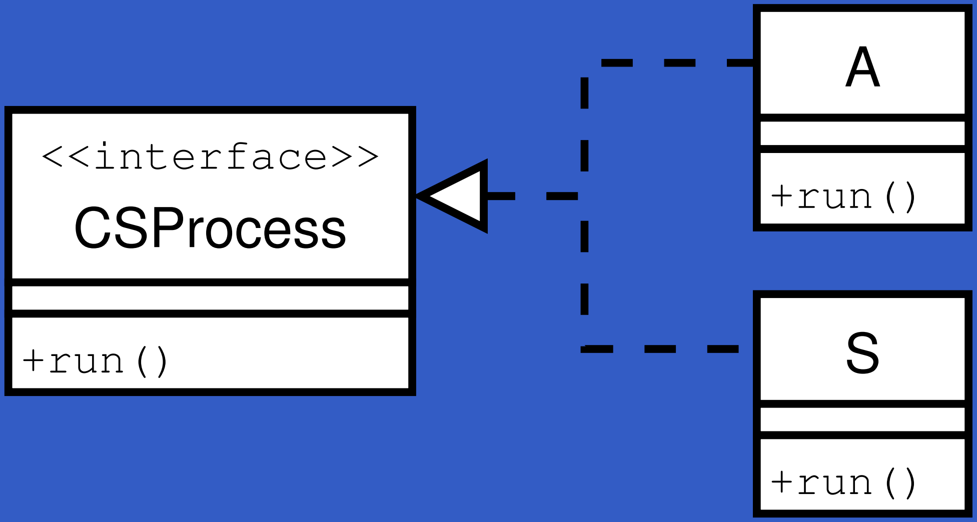
## Introduction to JCSP

- Processes represented by interface



## Introduction to JCSP

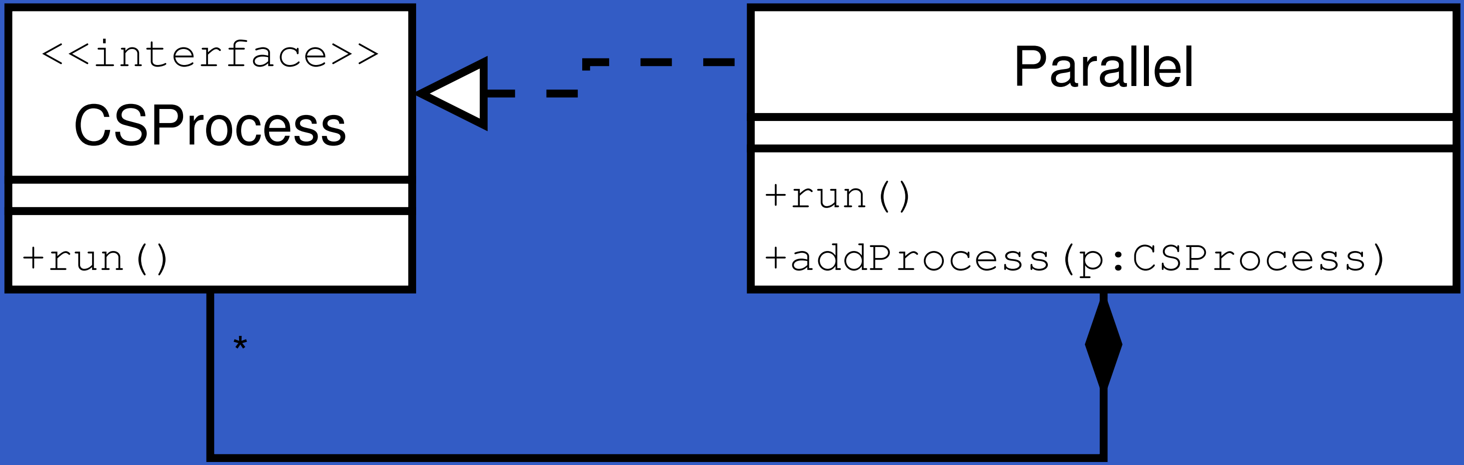
- Processes represented by interface



- In JCSP processes have identities

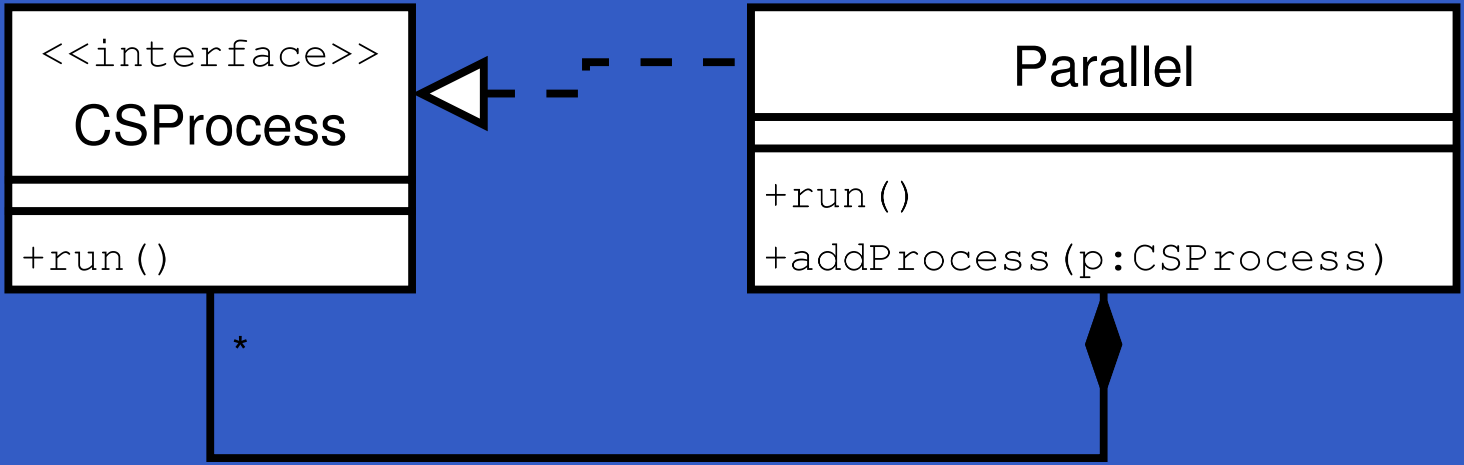
## Introduction to JCSP (2)

- CSP operator of parallelism (actually interleaving) captured by



## Introduction to JCSP (2)

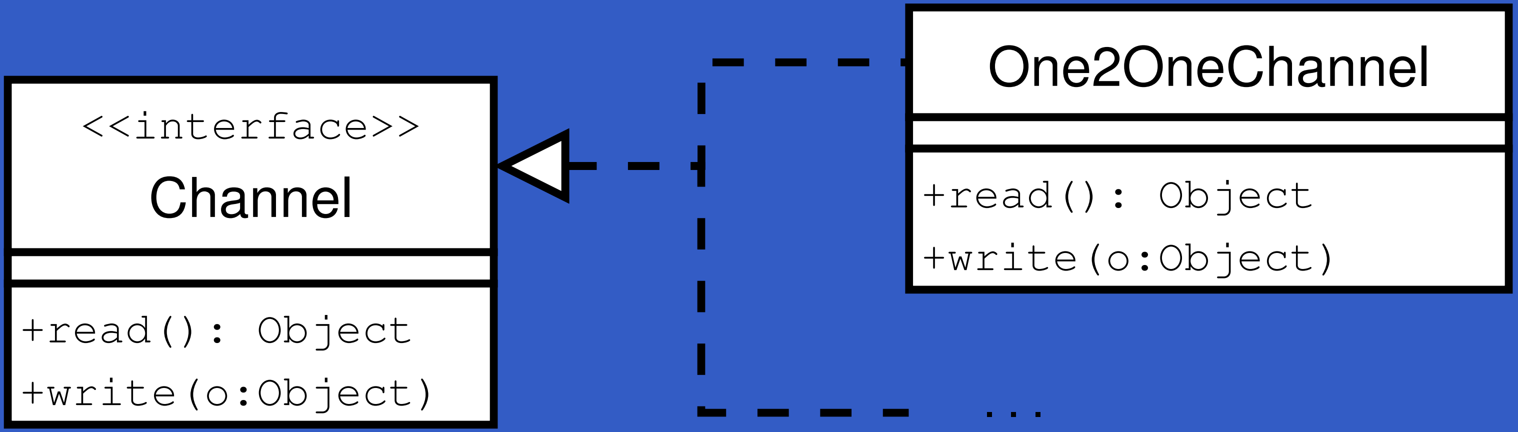
- CSP operator of parallelism (actually interleaving) captured by



- Each process is executed in its own thread

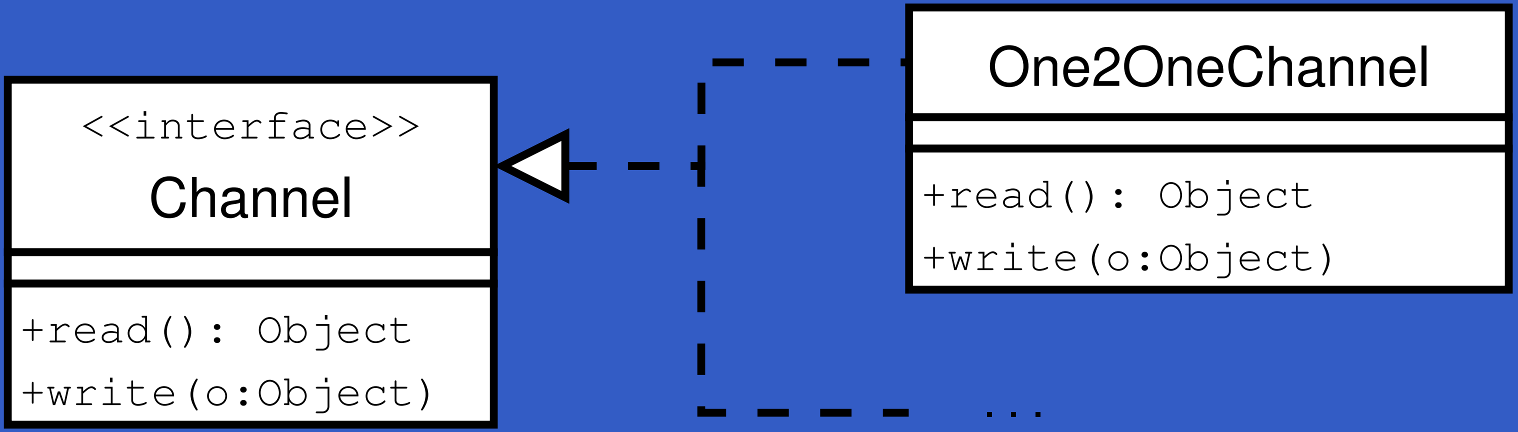
## Introduction to JCSP (3)

- Messages are sent through *channels*



## Introduction to JCSP (3)

- Messages are sent through *channels*



- No “unbound” events as in CSP

## Implementation of $S$ in JCSP

```
import jcsp.lang.*;
public class S implements CSProcess {
    private final Channel c;
    public S (Channel c) { this.c = c; }

    public void run () {
        while ( true ) {
            final Integer i = (Integer)c.read();
            c.write(new Integer(i.intValue() + 1));
        }
    }
}
```

## Building Systems from Components

- Interface of a component is a tuple of channels + a protocol

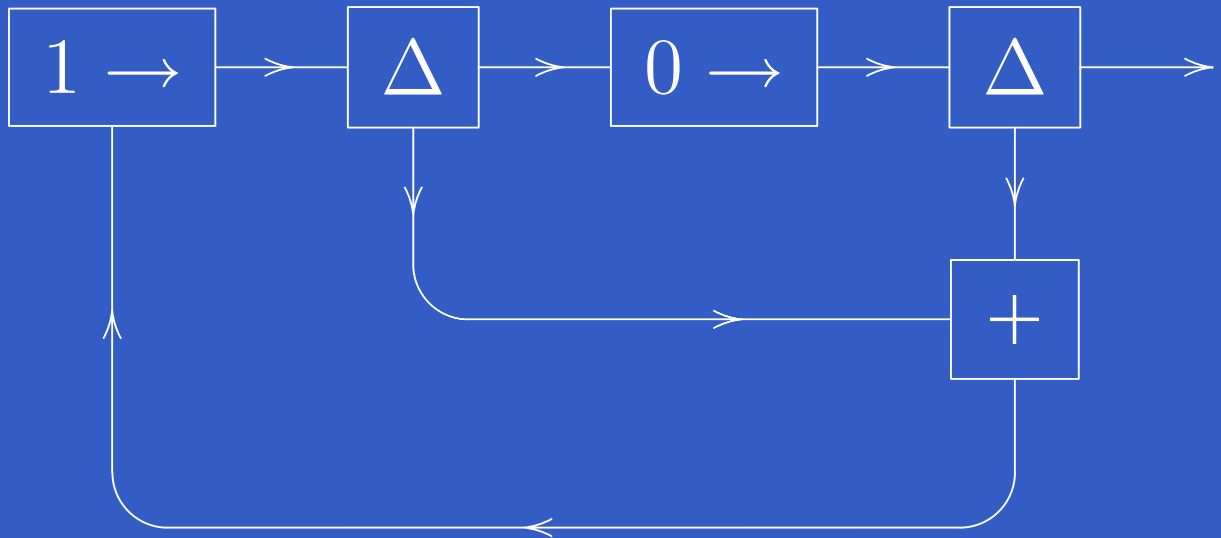
## Building Systems from Components

- Interface of a component is a tuple of channels + a protocol
- Systems are assembled from simpler components

## Building Systems from Components

- Interface of a component is a tuple of channels + a protocol
- Systems are assembled from simpler components
- Contrary to normal instances of classes, components are active

## Building Systems from Components



## A Proof System for JCSP

- Basic concept:

## A Proof System for JCSP

- Basic concept:
  - Represent JCSP systems as CSP terms

## A Proof System for JCSP

- Basic concept:
  - Represent JCSP systems as CSP terms
  - Symbolically execute CSP terms $\longrightarrow$ LTS

## A Proof System for JCSP

- Basic concept:
  - Represent JCSP systems as CSP terms
  - Symbolically execute CSP terms $\longrightarrow$ LTS
  - Specify/Prove LTS properties using a temporal/modal logic

## A Proof System for JCSP

- Basic concept:
  - Represent JCSP systems as CSP terms
  - Symbolically execute CSP terms $\longrightarrow$ LTS
  - Specify/Prove LTS properties using a temporal/modal logic

| | |
|---|---|
| JavaCard(DL) | CSP model of JCSP |
| CSP calculus | |
| Modal logic/calculus | |

## A Proof System for JCSP (2)

- A sequential Java program $\alpha$ is regarded as a CSP process $T(\alpha)$

## A Proof System for JCSP (2)

- A sequential Java program $\alpha$ is regarded as a CSP process $\mathsf{T}(\alpha)$
- Communication of $\mathsf{T}(\alpha)$ is caused (only) by JCSP primitives

## A Proof System for JCSP (2)

- A sequential Java program $\alpha$ is regarded as a CSP process $T(\alpha)$
- Communication of $T(\alpha)$ is caused (only) by JCSP primitives
  - $\Longrightarrow$ no shared memory

## A Proof System for JCSP (2)

- A sequential Java program $\alpha$ is regarded as a CSP process $T(\alpha)$
- Communication of $T(\alpha)$ is caused (only) by JCSP primitives
  - $\Longrightarrow$ no shared memory
- JCSP primitives are modelled using CSP operators

## Modelling JCSP Parallelism

- Class Parallel is represented by interleaving:

```
Parallel par = new Parallel();
par.addProcess(s1);
par.addProcess(s2); ...
par.run();
```

## Modelling JCSP Parallelism

- Class Parallel is represented by interleaving:

```
  Parallel par = new Parallel();
  par.addProcess(s1);
  par.addProcess(s2); ...
  par.run();
```

- Corresponding process term:

$$T(\text{.. par.run(); \ldots}) =$$
$$T(\text{.. s1.run(); \ldots}) \;|||\; T(\text{.. s2.run(); \ldots}) \;|||\; \cdots$$

## Modelling JCSP Channels

- Each JCSP channel object is represented by a routing process

## Modelling JCSP Channels

- Each JCSP channel object is represented by a routing process
- Channel constructors add routers:

$$T(\text{.. new One2OneChannel(); } \ldots) = \\ \text{O2ORouter} \mathrel{[\![} \text{O2OEvents} \mathrel{]\!]} T(\text{.. } \ldots)$$

## Modelling JCSP Channels

- Each JCSP channel object is represented by a routing process
- Channel constructors add routers:

$$T(\text{.. new One2OneChannel(); ...}) = $$
$$\text{O2ORouter} \,[\![\, \text{O2OEvents} \,]\!]\, T(\text{.. ...})$$

- Execution of Channel.read(), Channel.write() raises events

## Example

System after execution of

```
Channel c = new One2OneChannel();
Parallel par = new Parallel();
par.addProcess(new P1 (c));
par.addProcess(new P2 (c));
par.run();
```

## Example

System after execution of

```
Channel c = new One2OneChannel();
Parallel par = new Parallel();
par.addProcess(new P1 (c));
par.addProcess(new P2 (c));
par.run();
```

is described by

$$\text{O2ORouter} \;[\![\, \text{O2OEvents} \,]\!]\; \big( \mathcal{T}(\text{p1.run();}) \;|||\; \mathcal{T}(\text{p2.run();}) \big)$$

## Message Transmission through $c$

$$\text{O2ORouter} \,|[\, \text{O2OEvents} \,]|\, \big( T(\text{.. c.write(o); \ldots}) \;|||\; T(\text{.. a=c.read(); \ldots}) \big)$$

## Message Transmission through *c*

$$\text{O2ORouter} \;[\![\, \text{O2OEvents} ]\!]\; \big( T(\text{.. c.write(o); ...}) \;|||\; T(\text{.. a=c.read(); ...}) \big)$$

$$\rightsquigarrow \quad \cdots [\![ \cdots ]\!] \big( \,!o \longrightarrow T(\text{.. ...}) \;|||\; ?a \longrightarrow T(\text{.. ...}) \big)$$

## Message Transmission through $c$

$$\text{O2ORouter} \;[\![\, \text{O2OEvents} ]\!]\; \Big( T(\text{.. c.write(o); ...}) \;|||\; T(\text{.. a=c.read(); ...}) \Big)$$

$$\rightsquigarrow \quad \cdots [\![ \cdots ]\!] \Big( \,!o \longrightarrow T(\text{.. ...}) \;|||\; ?a \longrightarrow T(\text{.. ...}) \Big)$$

Further execution is performed by CSP calculus:

$$\rightsquigarrow \quad \cdots$$

$$\rightsquigarrow \quad !o \longrightarrow \Big( \text{O2ORouter} \;[\![ \cdots ]\!]\; \big( T(\text{.. ...}) \;|||\; T(\text{.. ...}) \big) \Big)$$

## Symbolic Analysis of JCSP Systems

- Possible logics for expressing properties of processes:

## Symbolic Analysis of JCSP Systems

- Possible logics for expressing properties of processes:
  - $\mu$-Calculus

## Symbolic Analysis of JCSP Systems

- Possible logics for expressing properties of processes:
  - $\mu$-Calculus
  - LTL, CTL, etc.

## Symbolic Analysis of JCSP Systems

- Possible logics for expressing properties of processes:
  - $\mu$-Calculus
  - LTL, CTL, etc.
- Possible CSP calculi:

## Symbolic Analysis of JCSP Systems

- Possible logics for expressing properties of processes:
  - $\mu$-Calculus
  - LTL, CTL, etc.
- Possible CSP calculi:
  - Based on operational semantics of CSP

## Symbolic Analysis of JCSP Systems

- Possible logics for expressing properties of processes:
  - $\mu$-Calculus
  - LTL, CTL, etc.
- Possible CSP calculi:
  - Based on operational semantics of CSP
  - Rewriting system based on algebraic laws

## Symbolic Analysis of JCSP Systems

- Possible logics for expressing properties of processes:
  - $\mu$-Calculus
  - LTL, CTL, etc.
- Possible CSP calculi:
  - Based on operational semantics of CSP
  - Rewriting system based on algebraic laws
  - Rewriting system based on partial order extension of CSP

## Summary: Modelling JCSP

- Representation of simple channels finished

## Summary: Modelling JCSP

- Representation of simple channels finished
- Incomplete: More complex communication (e.g. buffered channels, barriers, sending of complex data structures)

## Summary: CSP calculi

- Most encouraging results with partial order approach

## Summary: CSP calculi

- Most encouraging results with partial order approach
- Further investigation needed for:

## Summary: CSP calculi

- Most encouraging results with partial order approach
- Further investigation needed for:
  - Interface to modal logic

## Summary: CSP calculi

- Most encouraging results with partial order approach
- Further investigation needed for:
  - Interface to modal logic
  - Interaction with user

## Summary: CSP calculi

- Most encouraging results with partial order approach
- Further investigation needed for:
  - Interface to modal logic
  - Interaction with user
  - Treatment of proving techniques (postponed): Induction, compositional proving