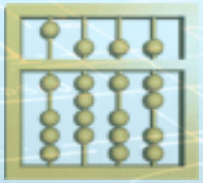
On Architecture Specification

Assertion Logic for Distributed Architectures

Manfred Broy

Technische Universität
München
Institut for Informatics



Architecture

Architecture is the structuring of (software) systems

- (functionality, functional features)
  into a set of services (functional service/feature architecture)
- (solution)
  into a set of sub-systems (sub-system architecture)
  - ◊ platform independent
  - ◊ platform specific

But

- how well is our methodology to design and specify such architectures?

Architecture

Architecture is determined by a set of architectural elements

- these elements have interfaces
  - ◊ over which they are connected
  - ◊ and exchange data
- an architecture is determined
  - ◊ by the set of its elements
  - ◊ the structure of their connections
  - ◊ the data flow between its elements

Structuring Architecture – Future Reference Architecture

- Functional service architecture – multi-feature architecture:
  - ◊ all functional features/services offered by software systems
  - ◊ structure services into an architecture of services
  - ◊ describe both their interface behavior as well as their
  - ◊ mutual relationships defining their functional dependencies, often called feature interactions.
- Platform-independent component/sub-system architecture:
  - ◊ decomposing systems into a set of components/sub-systems
  - ◊ describing by their roles, captured by the services they offer in terms of their interfaces and their interface behavior.
  - ◊ specifications of interfaces might also rely on assumptions about their context.
- Platform-dependent software and hardware architecture
  - ◊ deployment, scheduling, networking

Structuring Architecture – Future Reference Architecture

- Functional service architecture – multi-feature architecture:
  - all functional features/services offered by software systems
  - structure services into an architecture of services
  - describe both their interface behavior as well as their
  - mutual relationships defining their functional dependencies, often called feature interactions.
- Platform-independent component/sub-system architecture:
  - decomposing systems into a set of components/sub-systems
  - describing by their roles, captured by the services they offer in terms of their interfaces and their interface behavior.
  - specifications of interfaces might also rely on assumptions about their context.
- Platform-dependent software and hardware architecture
  - deployment, scheduling, networking

Architecture Meta-models

An architectural type is defined by

- the concept of system/component (model)
  - ◊ the type of syntactic and
  - ◊ semantic interface
- by the concepts to form architectural structures
  - ◊ composition
    - syntactic
    - semantic

Essence: Architecture Specification

Architecture is not what is represented and finally implemented in code but

- a description of architectural structures and rules which are required by the design for implementations leading to code that is correct w.r.t. the specified architecture.
- The rules, structure, and therefore the principles of architecture usually cannot be reengineered from the code but provide an additional design frame that is documented in the architecture specification.
- An architecture design consists of the specification of the system's structures, rules, and principles.
- Implemented systems realize architectures.
- Architectures define the overall structure and logics of systems.

Interfaces Everywhere

- Interfaces are used to describe
  - ◊ the behavior of the functional features of systems
  - ◊ the behavior if components of systems.
- Interfaces occur everywhere,
  - ◊ in the functional service architecture,
  - ◊ in the logical sub-systems/components architecture, and
  - ◊ in the technical architecture as well.
- Specifying interfaces can be done by interface assertions.
  - ◊ Assertions can be formulated quite formally or rather informally
  - ◊ We refer to a fully formalized notion of interface and interface behavior

Interfaces, their Structures and Properties

Interfaces are determined

- by the chosen system boundary
- which separates a system from its operational context and
- determines the flow of data/actions/events between the system and its context

For interfaces, we distinguish between

- the *syntactic interface* of a system
  - ◊ that describes which actions may be executed at the interface and which kind of information is exchanged by these actions across the system border,
- the *semantic interface* (also called *interface behavior*)
  - ◊ which describes the behavior evolving over the system border in terms of the specific information exchanged in the process of interaction by actions according to the syntactic interface.

Syntactic Interfaces

An interface defines the way a system interacts with its context.

We use a specific concept of interface:

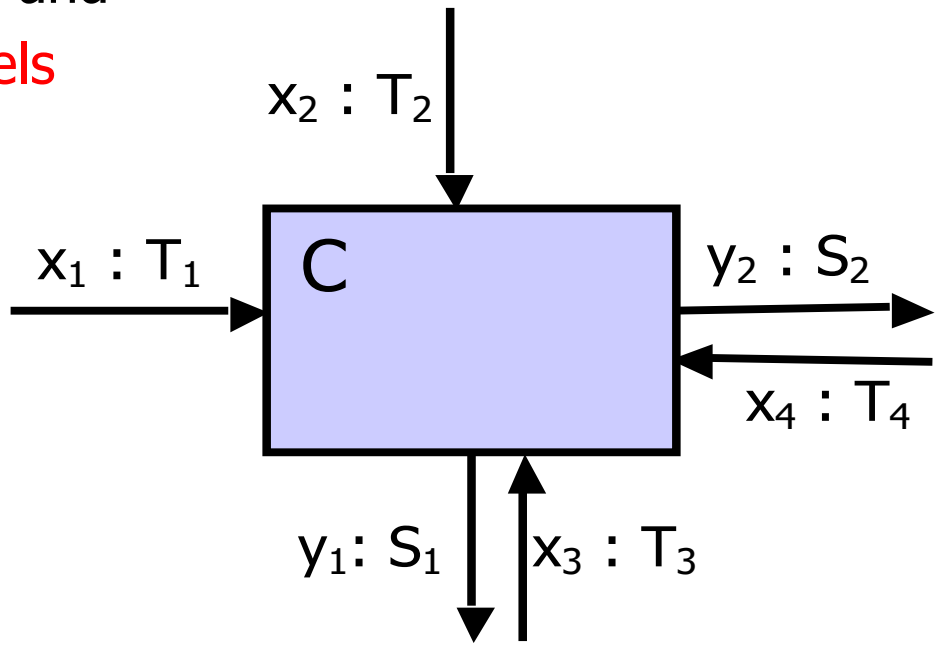
- Syntactically an interface is specified by a set of channels where each channel
  - ◊ has a data type assigned that defines
  - ◊ the set of messages, events, or signals transmitted over that channel.
- The set of channels is divided into a set of input and a set of output channels

Property Oriented Specification of Interfaces

- Components and systems, in general, run in parallel
  - ◊ are distributed
  - ◊ communicate and interact being part of networks,
  - ◊ act in real time
  - ◊ the interface concept has to be chosen appropriately.
- We denote a syntactic interface by (I ▸ O) where
  - ◊ I denotes the set of input channels and
  - ◊ O denotes the set of output channels
  - ◊ channels are typed
  - ◊ we write c : T to indicate that channel c has type T

$I = \{a : T_1, \ldots, x_4 : T_4\}$

$O = \{b : S_2, \ldots, x_4 : S_4\}$



System Interaction: Timed Data Streams

- Let $\mathbb{N}$ denote the natural numbers (including 0) and $\mathbb{N}^+$ denote the strictly positive natural numbers.
- The system model is based on the concept of a global clock.
  - ◊ The system model is time synchronous and message asynchronous.
  - ◊ We use the natural numbers $\mathbb{N}^+$ to number the time intervals.
- Given a message set $M \subseteq \mathbb{M}$ of data elements of type T we represent a *timed stream* s of type T by a function

$$s: \mathbb{N}^+ \rightarrow M^*$$

System Interaction: Timed Data Streams

- In a timed stream s a sequence of messages $s(t)$ is given for each time interval $t \in \mathbb{N}^+$;
  - ◊ $s(t) = \varepsilon$ indicates that in time interval t no message is communicated.
- By $(M^*)^\infty$ we denote the set of infinite timed streams.
- By $(M^*)^t$ we denote the set of finite timed streams of length t

Channel valuations

- For a set of typed channels C we denote a channel valuation by

$$z: C \to (M^*)^\infty$$

  where for each channel $c : T$ the stream $z(c)$ contains only messages of the type T of channel c

- For a the set of channels C we denote the set of channel valuations by

$$\vec{C}$$

- For $t \in \mathbb{N}$ we denote by $z{\downarrow}t$ the finite channel valuation

$$z{\downarrow}t : C \to (M^*)^n$$

  consisting for every channel $c \in C$ of the first t sequences in $z(c)$

System interface specification: interface assertions

- For a syntactic interface $(I \blacktriangleright O)$ we describe the interface behavior by an interface assertion Q
- An interface assertion is
  - ◊ a formula Q in predicate logic that uses
  - ◊ the channels from $(I \blacktriangleright O)$ as free logical variables standing for streams
  - ◊ we write

$$Q : (I \blacktriangleright O)$$

- Q defines a predicate

$$q: \vec{I} \times \vec{O} \to \mathbb{B}$$

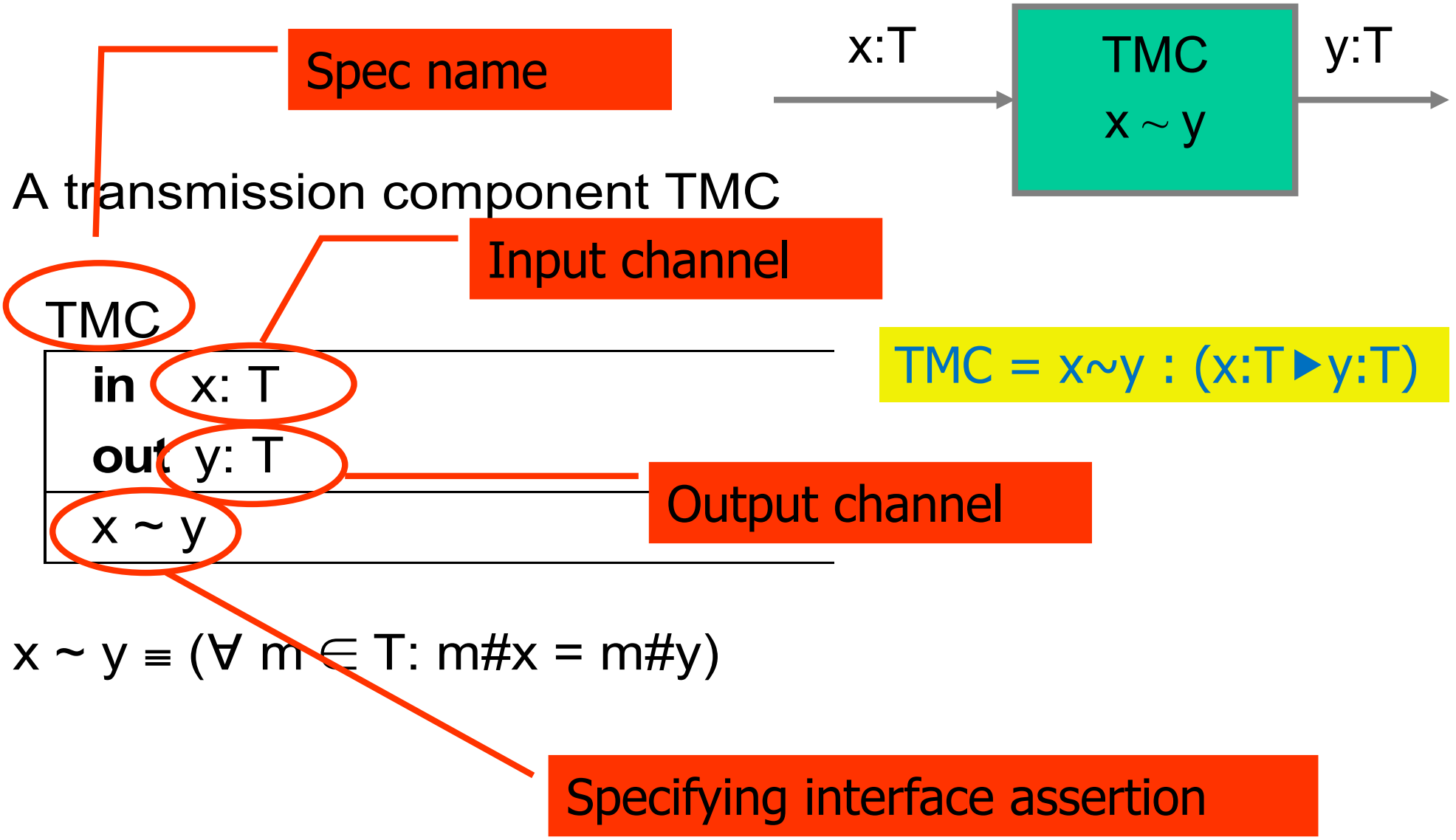
on channel valuations $x \in \vec{I}$ and $y \in \vec{O}$ as follows

$$q(x, y) = Q[x(A)/A, \ldots , y(b_1)/b_1, \ldots]$$

where $I = \{A, \ldots \}$, $O = \{b_1, \ldots \}$

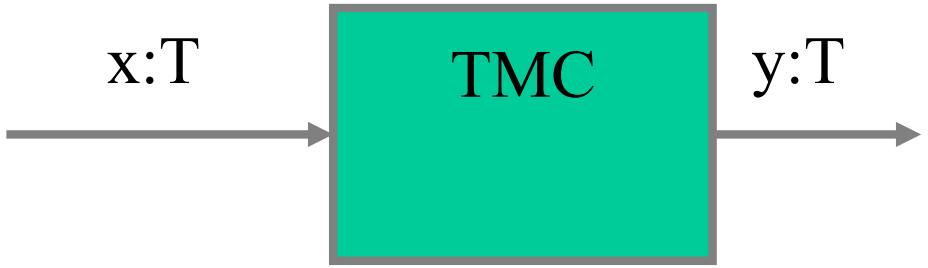
- q is called the predicate associated with assertion Q

Example: System interface specification



Specification of Timing Properties

Example: TMC with Timing Restrictions

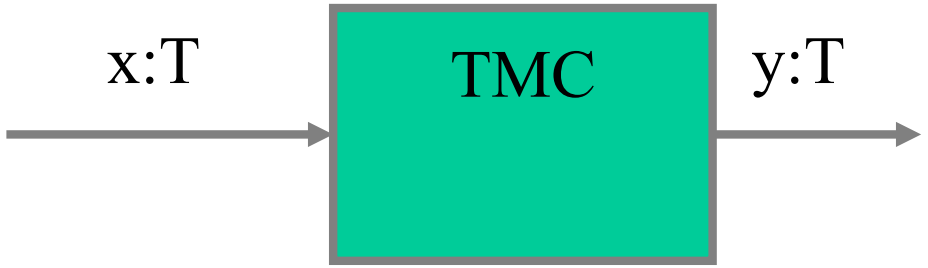


TMC

in x: T **out** y: T
$\forall\ t \in IN: \forall\ m \in T:$ $m\#(y{\downarrow}t+delay) \leq m\#(x{\downarrow}t) \leq m\#(y{\downarrow}t+delay+deadline)$

Specification of Probabilities

Example:
TMC with Probability Restrictions



TMC

in x: T **out** y: T
$\forall\, t \in IN: \forall\, m \in T:$ $\mathbf{P}(m\#(x{\downarrow}t) \leq m\#(y{\downarrow}t+delay+deadline)) \geq 0.8$

Expressiveness

This interface theory includes the

- specification of real time properties and
- can be extended to a probabilistic view.

It supports the description of probabilistic interface properties.

Causality

- P : (I ▶ O) with associated predicate p is called causal iff for all valuations $x, x' \in \vec{I}$, $y \in \vec{O}$

$$x{\downarrow}t = x'{\downarrow}t \Rightarrow (p(x, y{\downarrow}t) \Leftrightarrow p(x', y{\downarrow}t))$$

- and strongly causal iff for all $x \in \vec{I}$, $y \in \vec{O}$

$$x{\downarrow}t = x'{\downarrow}t \Rightarrow (p(x, y{\downarrow}t+1) \Leftrightarrow p(x', y{\downarrow}t+1))$$

where

$$p(x, y{\downarrow}t) = \exists\, y' : y'{\downarrow}t = y{\downarrow}t \wedge p(x, y')$$

Adding Causality to an Assertion

To an interface assertion $P : (I \blacktriangleright O)$ with associated predicate p we add causality

- resulting in interface assertion $P^{©}$
- with associated (strongly) causal predicate $p^{©}$
- being the weakest predicate where

$$p^{©}(x, y) \Rightarrow p(x, y)$$
$$p^{©}(x, y) \wedge x{\downarrow}t = x'{\downarrow}t \Rightarrow p^{©}(x', y{\downarrow}t(+1))$$

Proofs by Causality - Example

$p^{©}(x, y) = y \lesssim x$

$$y \lesssim x \Rightarrow x \sim y$$

$$y \lesssim x \wedge x{\downarrow}t = x'{\downarrow}t \Rightarrow \exists\, y' \colon y'{\downarrow}t+1 = y{\downarrow}t+1 \wedge y' \lesssim x'$$

Theorem: $y \lesssim x \Rightarrow d\#y{\downarrow}t+1 \leq d\#x{\downarrow}t$

Proof: Assume $y \lesssim x$; then $x \sim y$

Choose $x' = (x{\downarrow}t)\hat{}x''$ with $d\#x'' = 0$; then $x{\downarrow}t = x'{\downarrow}t$

There exists y' such that $d\#y{\downarrow}t+1 = d\#y'{\downarrow}t+1 \wedge x' \lesssim y'$

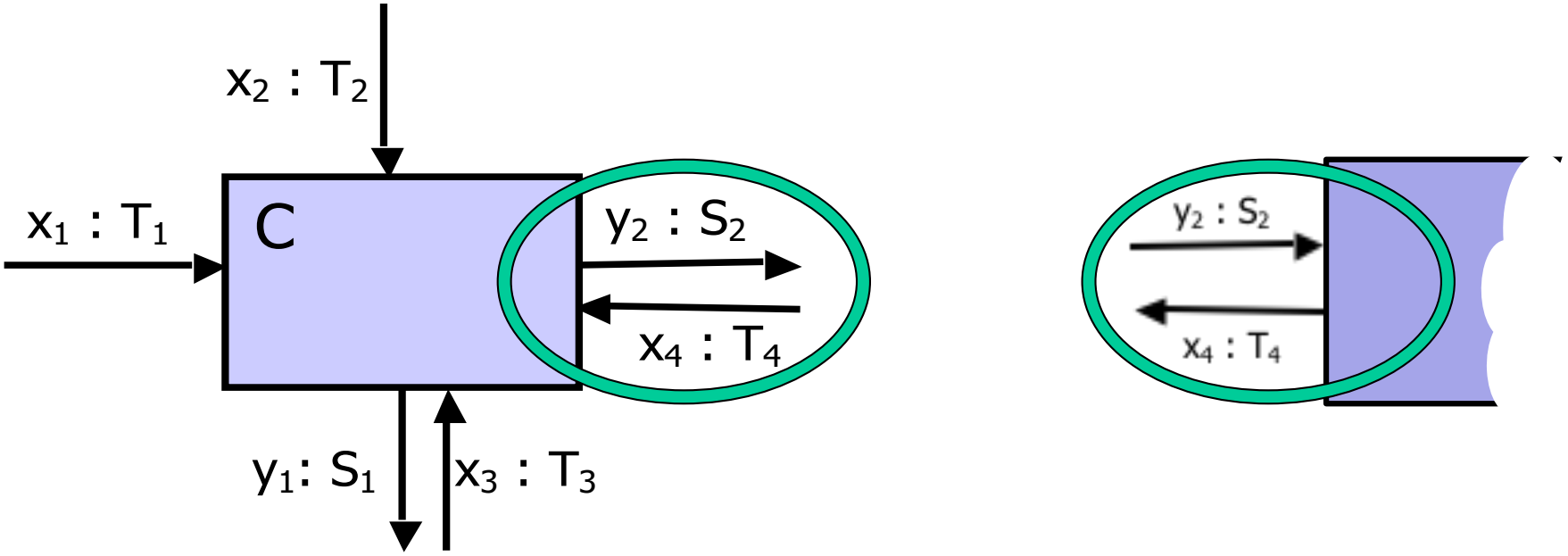
then $x' \sim y'$ holds from which we get $d\#y' = d\#x' = d\#x{\downarrow}t$

and thus

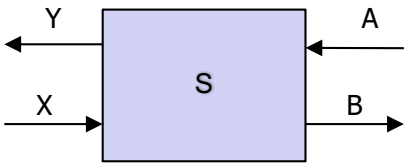
$$d\#y{\downarrow}t+1 = d\#y'{\downarrow}t+1 \leq d\#x' = d\#x{\downarrow}t$$

About interfaces

- Structuring interfaces
  - ◊ causality basically says that the output produced till time t does only depend on input received before time t.
- Given syntactic interface (I ▸ O) we call the syntactic interface (O ▸ I) then *inverse* interface.
  - ◊ It is denoted by $(I \blacktriangleright O)^{-1}$. It can also be called matching interface



Decomposition into Sub-interfaces



$Q : (X \cup A \blacktriangleright Y \cup B)$

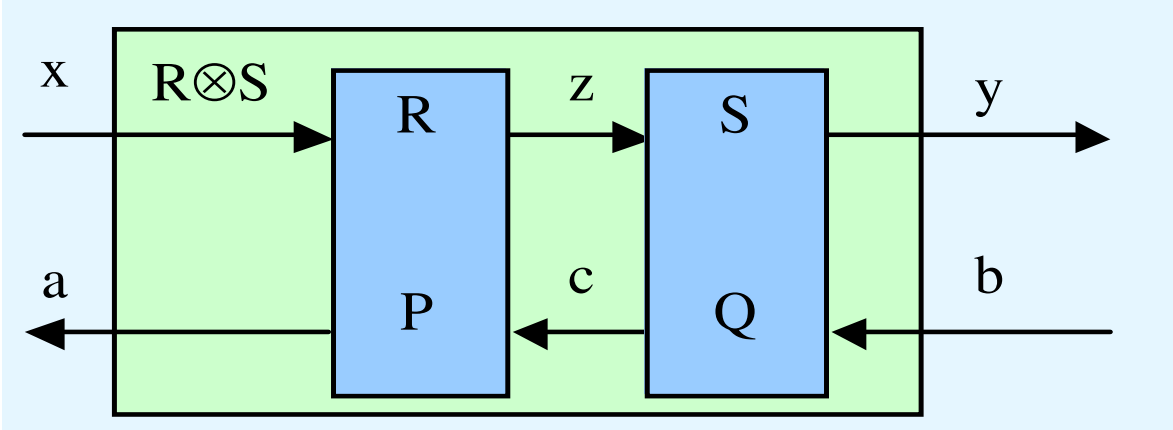
Assume disjoint channel sets A, B, X, Y

consistency: $\forall$ A, X: $\exists$ B, Y: Q

There are two sub-interfaces $(X \blacktriangleright Y)$ and $(A \blacktriangleright B)$

- Independence: $Q = (\exists A, B: Q) \wedge (\exists X, Y: Q)$
- $(X \blacktriangleright Y)$ independent of $(A \blacktriangleright B)$: $(\exists A, B: Q) = \exists B: Q$
- Pipeline element: $Y = A = \varnothing$

Modularity: Rules of compositions for interface specs



R

in x, z: T
out a, b: T
P

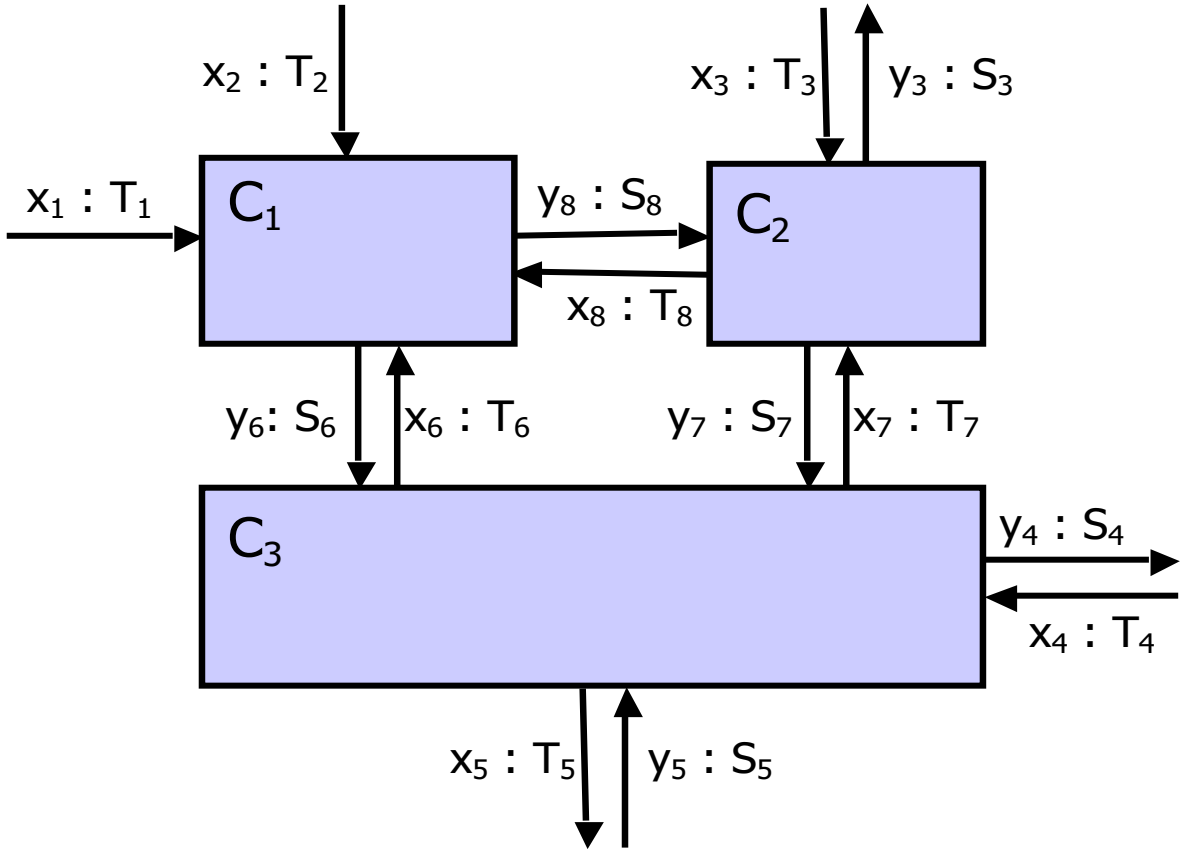
S

in b, z: T
out c, y: T
Q

R⊗S

in x, b: T
out a, y: T
$\exists$ c, z: R $\wedge$ Q

Composition



System Composition: Hiding Channels

- Given two systems S and R
  - ◊ with syntactic interfaces (I ▶ O) and (J ▶ W) and interface specifications Q and R,
  - ◊ where $H = (I \cap W) \cup (J \cap O)$ denotes the internal channels that connect the two systems and
  - ◊ we get the interface assertion for the composite system hiding internal channels

$$(\exists\, H: Q \wedge R) : ((I \cup J) \setminus H \blacktriangleright (O \cup W) \setminus H)$$

  - ◊ a modular composition of systems over corresponding sub-interfaces.
  - ◊ Without hiding we get

$$Q \wedge R : ((I \cup J) \setminus H \blacktriangleright O \cup W)$$

If both (I ▶ O): Q and (J ▶ W): R are strongly causal so is

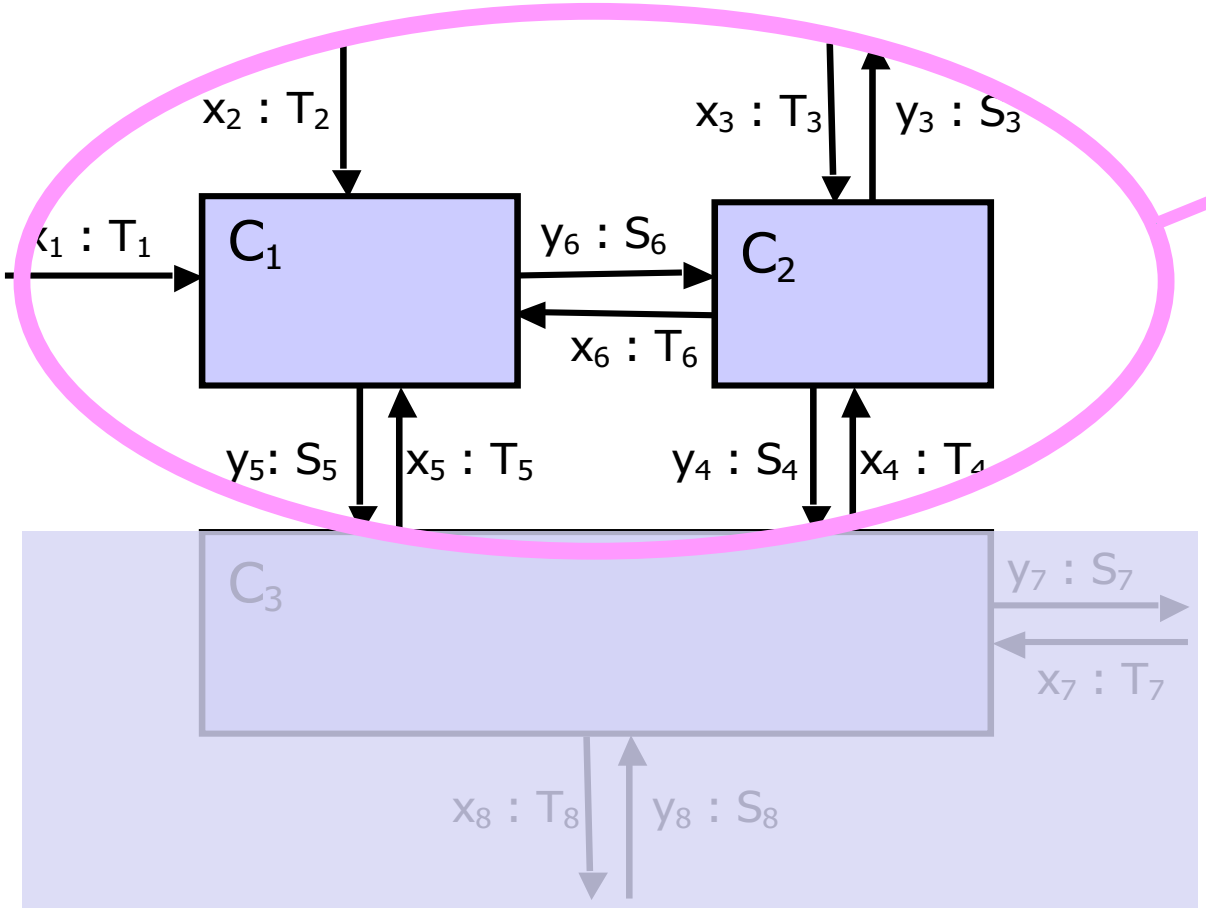
$$Q \wedge R : ((I \cup J) \setminus H \blacktriangleright O \cup W)$$

System Composition: Hiding Channels

- Given two systems S and R
  - ◊ with syntactic interfaces (I ▶ O) and (J ▶ W) and interface specifications Q and R,
  - ◊ where $H = (I \cap W) \cup (J \cap O)$ denotes the internal channels that connect the two systems and
- We specify the property of the connector of the two channels
  - ◊ $G = (I \cup W \cup J \cup O) \backslash H$ denotes the set of external channels of the composite system;

    $\exists G: Q \wedge R$
- Btw: Can be used for watch dogs.

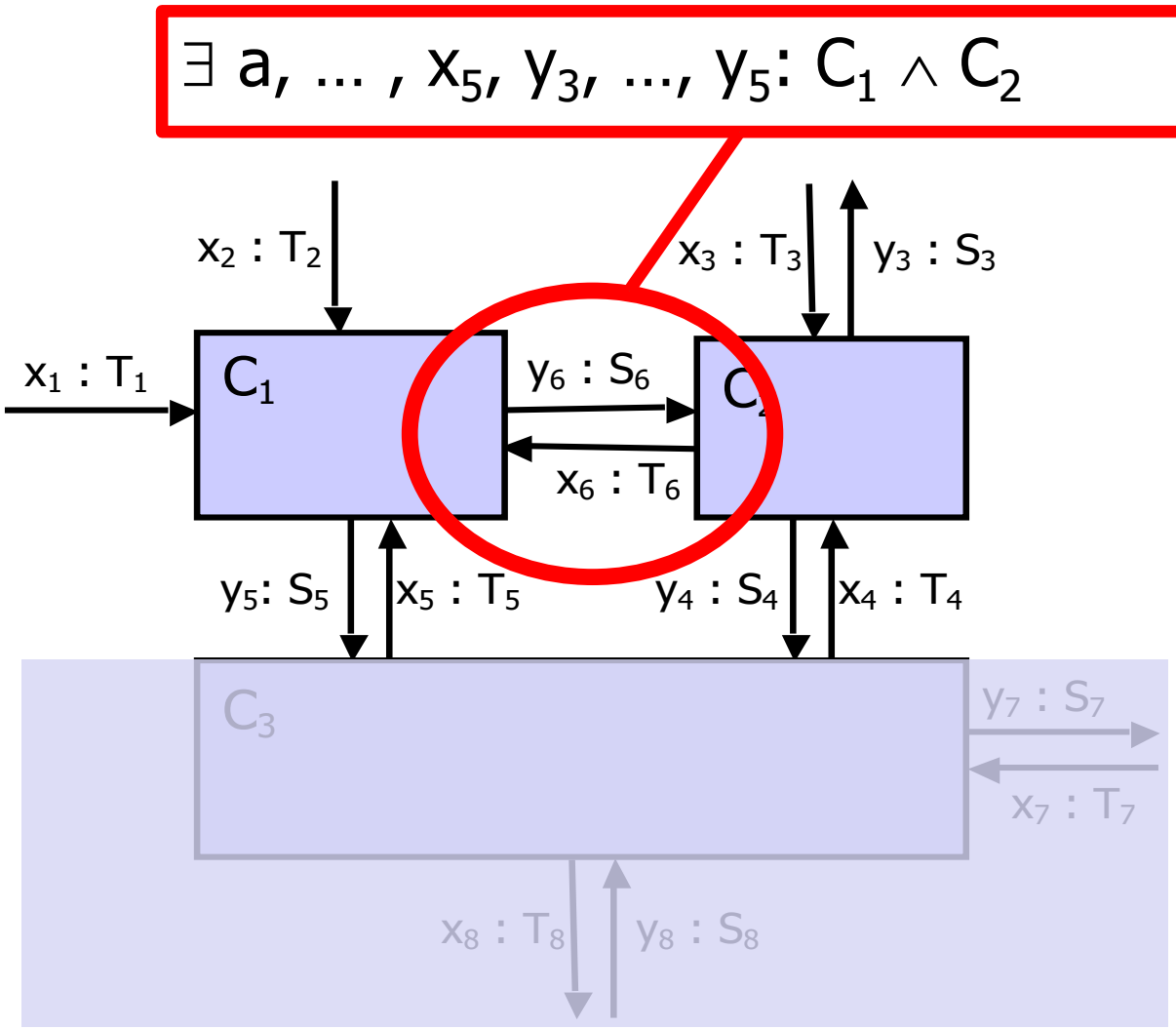
Composition and Interface Abstraction



$\exists\ x_6, y_6: C_1 \wedge C_2$

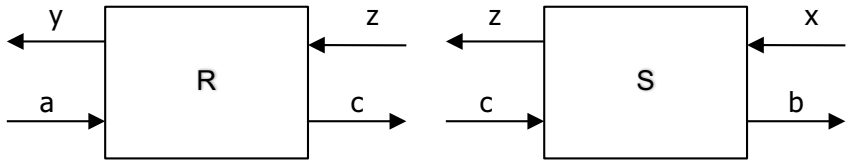
Interface specification of the external behavior of the composite system

Composition and Interface Abstraction



Interaction specification of the internal channels of the composite system

Example: Composition



We specify system

$$R \equiv P : (a{:}D, z{:}\{req\} \blacktriangleright c{:}D, y{:}\{req\})$$

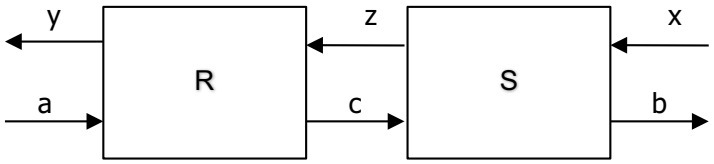
$$P \equiv [\#c = \min(\#a, \#z) \wedge \forall\, d \in D{:}\ d\#c \leq d\#a \wedge y = z]$$

We specify system

$$S \equiv Q : (c{:}D, x{:}\{req\} \blacktriangleright b{:}D, z{:}\{req\})$$

$$Q \equiv [\#b = \min(\#c, \#x) \wedge \forall\, d \in D{:}\ d\#b \leq d\#c \wedge z = x]$$

Interface of the Composed System



Composing the two systems results in interface assertion

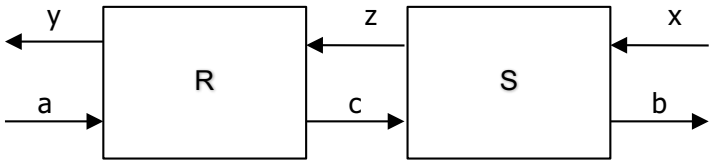
$$\#c = \min(\#a, \#z)$$
$$\wedge\ \forall\ d \in D: d\#c \leq d\#a$$
$$\wedge\ \ \#b = \min(\#c, \#x)$$
$$\wedge\ \forall\ d \in D: d\#b \leq d\#c$$
$$\wedge\ y = z \wedge z = x$$

Hiding c and z by existential quantification we get

$$\#b = \min(\#a, \#x) \wedge \forall\ d: d\#b \leq d\#x \wedge y = x$$

- In this case, the composed system fulfills the same assertion as the two sub-systems.

Properties of the hidden channels



For the hidden channels, we get the assertions

$$\exists\, a, b: \#c = \min(\#a, \#z)$$
$$\wedge\ \forall\, d \in D: d\#c \leq d\#a$$
$$\wedge\ \#b = \min(\#c, \#z)$$
$$\wedge\ \forall\, d \in D: d\#b \leq d\#c$$

which reduces to

$$\#c \leq \#z$$

This assertion characterizes the property of the internal channels of this little architecture.

Interfaces with assumptions

Often, in an interface specification for the syntactic interface (I ▶ O) we include

- an assumption asu(y, x) which is a specification of the *inverse* interface (O ▶ I) – a specification for the matching interface –

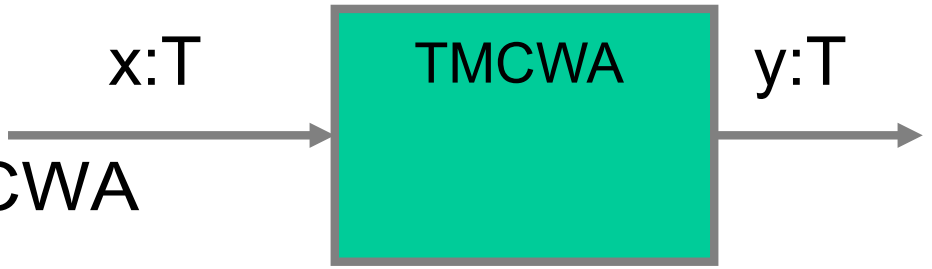
- a commitment cmt(x, y) which is a specification of the behavior the syntactic interface (I ▶ O) as long as the assumption is fulfilled.

this leads to the specification

$$asu(y, x) \Rightarrow cmt(x, y)$$

Example: System interface specification

A transmission component TMCWA

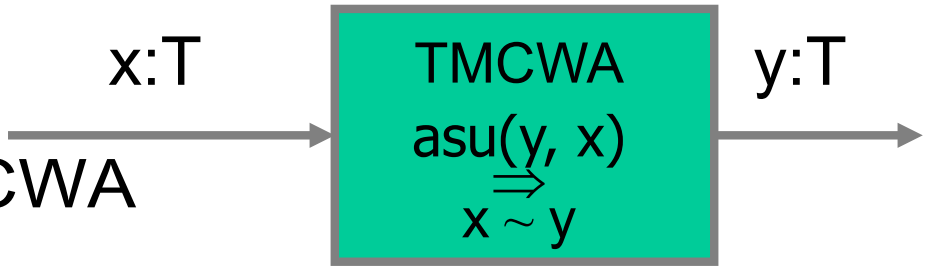


TMCWA

in x: T **out** y: T
assume $\forall\ t \in \mathbb{N}: \#x{\downarrow}t \leq 1+\#y{\downarrow}t$ **commit** $\forall\ m \in T: m\#x = m\#y$

Example: System interface specification

A transmission component TMCWA



TMCWA

in x: T **out** y: T
asu(y, x) ⇒ x ~ y

$x \sim y \equiv (\forall\ m \in T: m\#x = m\#y)$

$asu(y, x) \equiv (\forall\ t \in \mathbb{N}: \#x{\downarrow}t \leq 1+\#y{\downarrow}t)$

We speak of a contract

Fulfilling inner assumptions

Given two interface specifications (that fit together)

- One with assertion L
- One with assertion $(A \Rightarrow C)$ with assumption A and commitment C

If

$$L \Rightarrow A$$

holds, then the assertion for the composed system is

$$(A \Rightarrow C) \wedge L$$

which since L implies assumption A can be simplified to

$$C \wedge L$$

Fulfilling inner assumptions with the help of outer assumptions

Given two interface specifications (that fit together)

- One with assertion L
- One with assertion $(A \Rightarrow C)$ with assumption A and commitment
- Let B be an assumption for the composed system

If

$$B \wedge L \Rightarrow A$$

holds, then an assertion for the composed system is

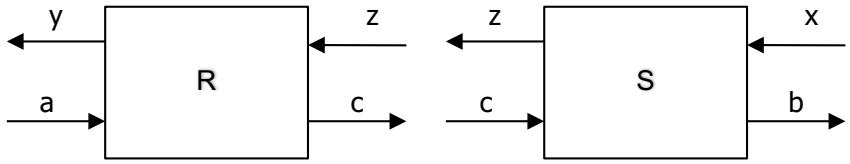
$$B \Rightarrow ((A \Rightarrow C) \wedge L)$$

which since $B \wedge L$ implies assumption A can be simplified to

$$B \Rightarrow C \wedge L$$

Of course, we are interested in the weakest assumption B that does the job.

Example: Assumption / Commitment Specifications



We introduce an additional assumption A and commitment C for the second component S

$$A \equiv \forall\, t: \#(c{\downarrow}t) \leq \#(z{\downarrow}t)$$

$$C \equiv \forall\, t: \#(z{\downarrow}t) \leq \#(c{\downarrow}t)+1$$

and add A to P:

$$P' \equiv A \wedge P \qquad Q' \equiv (A \Rightarrow C) \wedge Q$$

We get obviously

$$P' \Rightarrow A$$

and by composition a component that fulfills the specification

$$P \wedge Q \wedge A \wedge C$$

Import of services: Requested and Provided Interfaces

If a system S with syntactic interface (all sets I, J, O, W disjoint)

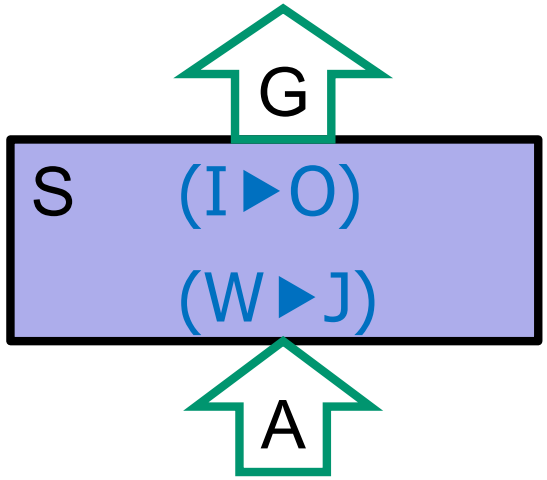
$$(I \cup J \blacktriangleright O \cup W)$$

and sub-interface

$$(I \blacktriangleright O)$$

requires a certain sub-interface

$$(W \blacktriangleright J)$$



to be able to fulfill its task this is expressed by

- Given a specification of the requested interface $A : (W \blacktriangleright J)$
- Given a specification of the provided interface $G : (I \blacktriangleright O)$ we define

$$S = (A \Rightarrow G):(I \cup J \blacktriangleright O \cup W)$$

A is called *assumed interface*.

- S is called a layer

Import: Satisfying Assumptions

- Given

$$R = B : (W \blacktriangleright J)$$

with

$$B \Rightarrow A$$

We get the interface specification of the composed system

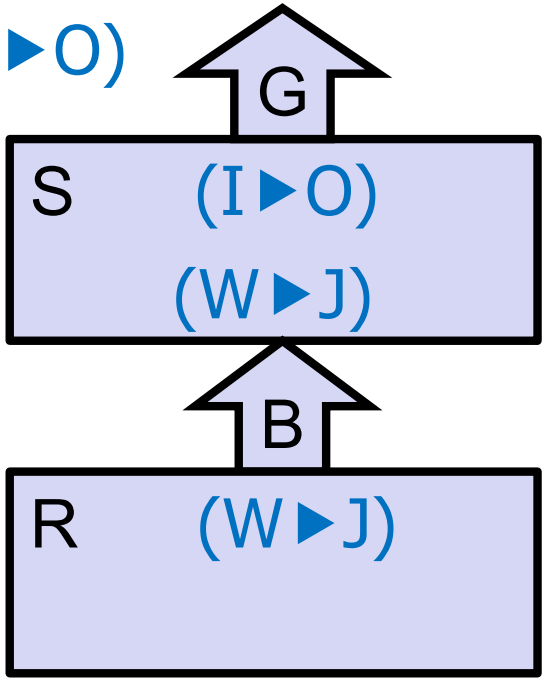
$$S \otimes R = \exists\, J, W:\ (A \Rightarrow G) \wedge B : (I \blacktriangleright O)$$

which is by $B \Rightarrow A$ under the assumption

$$\exists\, J, W: B$$

equivalent to

$$S \otimes R = G : (I \blacktriangleright O)$$



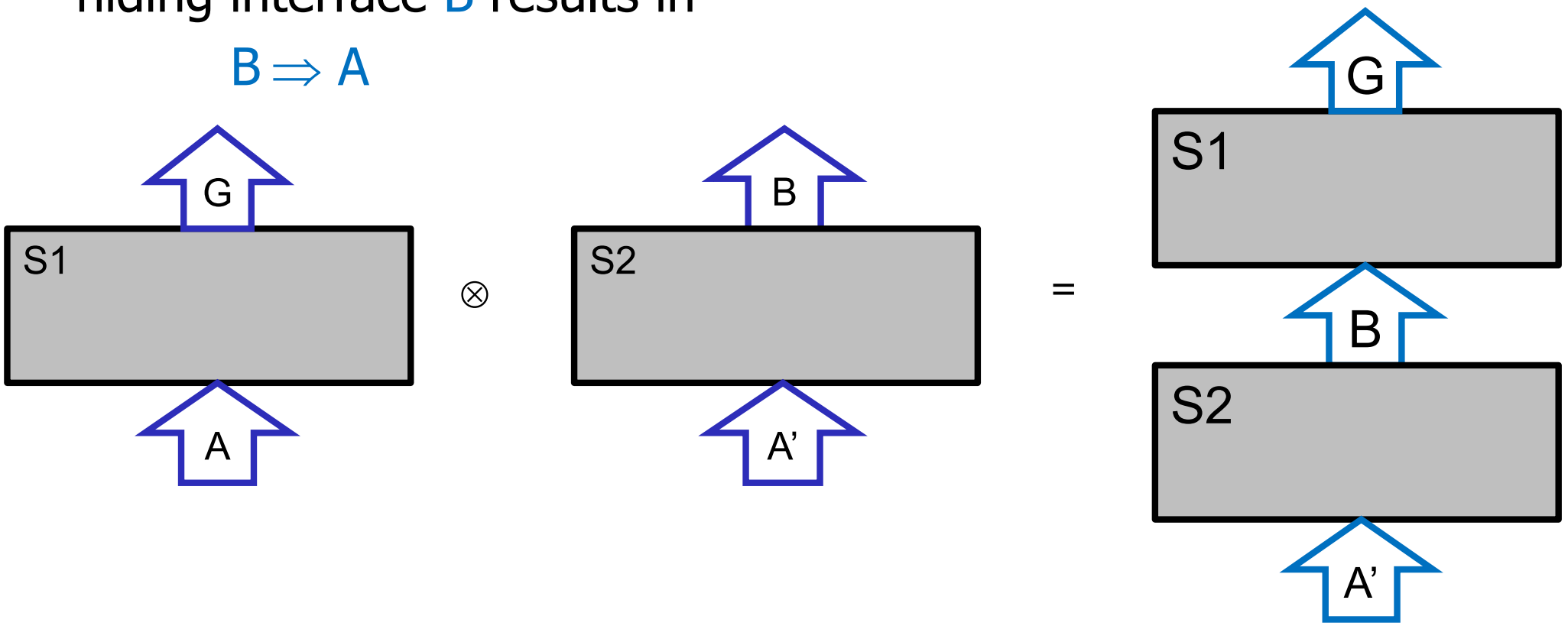
Forming Layered Architectures

Composition of layer S1 and layer S2 via interface B

$$(A \Rightarrow G) \wedge (A' \Rightarrow B)$$

hiding interface B results in

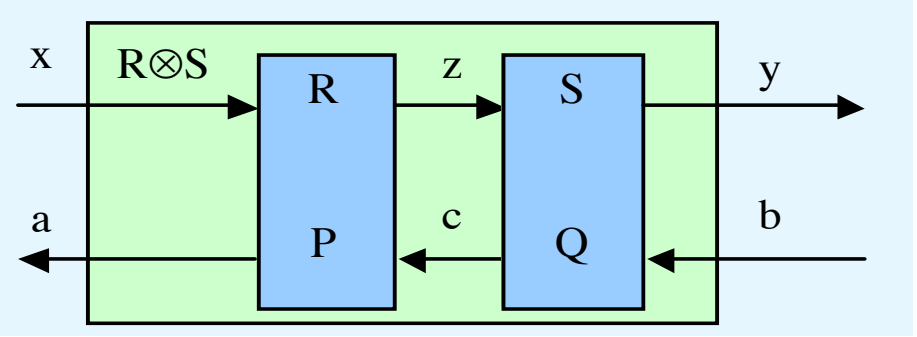
$$B \Rightarrow A$$



Forms of composition: architectural patterns

Pipeline:

$y = c = x = \varnothing$



Layered:

$P = (A \Rightarrow G)$, $Q = (B \Rightarrow H)$

$G:(a \blacktriangleright y)$, $H:(z \blacktriangleright c)$, $A:(c \blacktriangleright z)$, $B:(x \blacktriangleright b)$

$R \otimes S = (B \Rightarrow G):(a \cup x \blacktriangleright y \cup b)$ if $H \Rightarrow A$

Assumption/Commitment: $Q = (A \Rightarrow G)$, $R = (B \Rightarrow H)$

$G:(x \cup c \blacktriangleright a \cup z)$, $H:(z \cup b \blacktriangleright c \cup y)$

$A:(a \cup z \blacktriangleright x \cup c)$, $B:(c \cup y \blacktriangleright z \cup b)$

$R \otimes S = G \wedge H : (x \cup b \blacktriangleright y \cup a)$ if

$(A \Rightarrow G) \Rightarrow B$, $(B \Rightarrow H) \Rightarrow A$

Principles

- Not formal methods but formal foundation
- Occam's Razor:
  - ◊ KISS: keep everything as simple as possible
  - ◊ do not introduce more concepts than needed
- Flexibility and Universality
- System Components as schedulable and deployable units
- Modularity
- Strict Property Orientation - architecture design by specification – everything based on logics
- Covering software as well as cyber-physical system architecture
- Real Time and Probability: Functional Quality Properties

Conclusion: What we got?

A completely logic based theory of service oriented architectures, including

- functional feature architecture
- platform independent component architecture
- platform dependent architecture

Fully formal modular specification

- of properties (including partial properties, abstraction, refinement, interoperability)
- of real time and probability
- can be extended to classes and instances and dynamic architectures
- as a basis for methodology, language, and tool design
- can be applied semi-formally or informally

Open questions

- how well does it scale
- …