Abstraction in Concurrent Systems

Beyond Monotonicity

Towards a Modular Development

Manfred Broy

Technische Universität München
Institut für Informatik
D-80290 Munich, Germany

Concurrent Interactive Systems: Topics

- Generalized Moore machines are chosen as computational model
  - ◊ Concurrent composition
  - ◊ Compositionality
- Moore machines are a perfect model for explicit concurrency
  - ◊ Crystal clear notion of internal (encapsulation, information hiding) and external (interface) behavior
  - ◊ Built in notion of time
- Abstractions for the behavior of Moore machines
  - ◊ Interface abstraction: timed and nontimed
  - ◊ Time abstraction
- Interface specifications and their concurrent composition
- Timed based reasoning for untimed system specifications
- Modularity
- Modelling cyber-physical systems

abstraction

Model	Forms of descriptions
Untimed interface behaviors	untimed interface specifications, composition, channel hiding
time abstraction	
Timed interface behaviors	timed interface specifications, composition, channel hiding
interface abstraction	
Operational semantics Moore machines	state transitions with I/O, composition, channel hiding

Syntactic interfaces

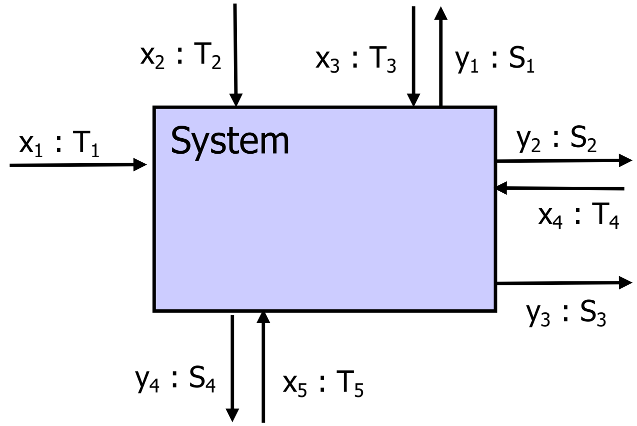
Given channel sets X and Y, a syntactic interface is denoted by

$$(X \blacktriangleright Y)$$

Sets of typed channels (names of communication lines)

Input $\quad X = \{x_1 : T_1, x_2 : T_2, \ldots\}$

Output $\quad Y = \{y_1 : S_1, y_2 : S_2, \ldots\}$



Moore machines

For syntactic interface $(X \blacktriangleright Y)$, a generalized nondeterministic (total) Moore machine with state space Σ is a pair (Δ, Λ) where Δ is a *total state transition function*

$$\Delta: (\Sigma \times \overline{X}_{fin}) \to \wp(\Sigma \times \overline{Y}_{fin})\backslash\{\emptyset\}$$

and $\Lambda \subseteq \Sigma$ is a *nonempty set of initial states* and for $a \in \overline{X}_{fin}$, $b \in \overline{Y}_{fin}$, $\sigma, \sigma' \in \Sigma$

$$(\sigma', b) \in \Delta(\sigma, a)$$

where the output b does not depend on the input a but only on the state σ.

Formally defined, there exists an output function:

$$\Xi: \Sigma \to \wp(\overline{Y}_{fin})\backslash\{\emptyset\}$$

indicating that the output depends only on the state such that

$$\forall\ \sigma \in \Sigma, a \in \overline{X}_{fin}: \Xi(\sigma) = \{b \in \overline{Y}_{fin}: \exists\ \sigma' \in \Sigma: (\sigma', b) \in \Delta(\sigma, a)\}$$

Interaction: Input and Output via Syntactic Interfaces (X ▶ Y)

Set of inputs for a syntactic interface in one step of the system

$$\overline{X}_{fin} = (X \rightarrow M^*)$$

For input $z \in \overline{X}_{fin}$ for each channel x_k the sequence of values $z(x_k)$ is of type T_k

The same holds for output $y \in \overline{Y}_{fin}$

$$\overline{Y}_{fin} = (Y \rightarrow M^*)$$

Timed Streams

Timed streams $(M^*)^\omega = \mathbb{N}_+ \rightarrow M^*$

Finite timed streams $(M^*)^* = \cup_{n \in \mathbb{N}} (\{m \in \mathbb{N}_+ : m \leq n\} \rightarrow M^*)$

For $x \in (M^*)^\omega$:

$$x{\downarrow}t : \{n \in \mathbb{N} : 1 \leq n \leq t\} \rightarrow M^*$$

$$1 \leq n \leq t \Rightarrow (x{\downarrow}t)(n) = x(n)$$

#x number of elements in stream x

S#x number of elements in x that are in set S

m#x = {m}#x

Type of all timed streams: Tstr M

Histories of Timed and Untimed Streams

Given a set of typed channel names

$$X = \{c_1{:}T_1, \ldots, c_m{:}T_m\}$$

by $\vec{X}$ we denote channel histories given by families of timed streams, one timed stream for each of the channels:

Timed histories

$$\vec{X} = (X \rightarrow (M^*)^\omega)$$

Finite timed histories

$$\vec{X}_{fin} = (X \rightarrow (M^*)^*)$$

Computations of Moore machines and interface abstraction

Given a Moore machine (Δ, Λ) for syntactic interface $(X \blacktriangleright Y)$ with state space Σ a computation is given by

- an infinite stream of states $\{\sigma_i \in \Sigma: i \in \mathbb{N}\}$
- an input history of $x \in \vec{X}$
- an output history $y \in \vec{Y}$

such that $\sigma_0 \in \Lambda$ and

$$\forall\, i \in \mathbb{N}: (\sigma_{i+1}, y(i+1)) \in \Delta(\sigma_i, x(i+1))$$

This way a Moore machine defines an timed interface predicate :

$$[\![\Delta, \Lambda]\!] \;:\; \vec{X} \times \vec{Y} \rightarrow \mathbb{B}$$

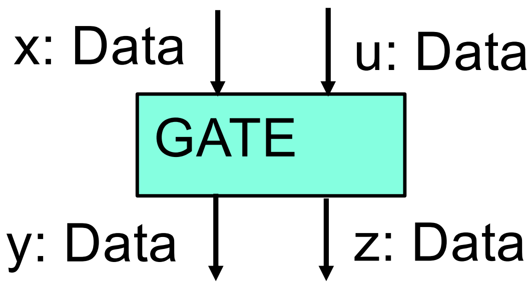
the result of interface abstraction: information hiding of states

Example of interface specification for a system: predicate GATE

We specify a system by an interface predicate GATE on timed streams $x, u, y, z \in (Data^*)^\omega$ of data as follows

GATE = (x: Data, u: Data ▶ y: Data, z: Data):

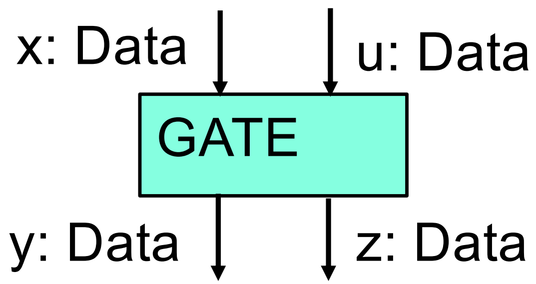
$$\forall d \in Data: d\#x+d\#u = d\#y+d\#z \wedge (d\#x+d\#u = \infty \Rightarrow (d\#y = \infty \wedge d\#z = \infty))$$



Interface specification: interface predicates and assertions

$$\mathrm{GATE}: \vec{X} \times \vec{Y} \rightarrow \mathbb{B}$$

$$\mathrm{GATE} = (X \blacktriangleright Y): A$$



syntactic interface

where A is an assertion with free identifiers from X and Y

GATE = (x: Data, u: Data ▶ y: Data, z: Data):

$\forall n \in \mathrm{Data}: d\#x+d\#u = d\#y+d\#z \wedge (d\#x+d\#u = \infty \Rightarrow (d\#y = \infty \wedge d\#z = \infty))$

predicate

assertion A

We write $Q::(X \blacktriangleright Y)$ to express that Q is an interface predicate for the syntactic interface $(X \blacktriangleright Y)$; we write Q^A for the assertion on streams defined by Q

Parts of interface specifications

An interface specification is given by

- a name (such as GATE)
- a syntactic interface

  GATE::(x: Data, u: Data ▸ y: Data, z: Data)

- an interface assertion for the involved streams

  $GATE^A$ =

  $(\forall\ d \in \text{Data: } d\#x+d\#u = d\#y+d\#z \wedge (d\#x+d\#u = \infty \Rightarrow (d\#y = \infty \wedge d\#z = \infty)))$

An interface specification defines a functional specification of an interface predicate GATE::(x: Data, u: Data ▸ y: Data, z: Data) on histories/streams

Strong Causality

Strongly Causal Interface Predicates

$Q :: (X \blacktriangleright Y)$

is strongly causal if for all $x, x' \in \vec{X}, \ y \in \vec{Y}, t \in \mathbb{N}$

$$x{\downarrow}t = x'{\downarrow}t \wedge Q(x, y) \Rightarrow \exists\, y' \in \vec{Y}: Q(x', y') \wedge y{\downarrow}t+1 = y'{\downarrow}t+1$$

For every interface predicate $Q::(X \blacktriangleright Y)$
there exists a weakest refinement $Q^{©}$ of Q which is strongly causal

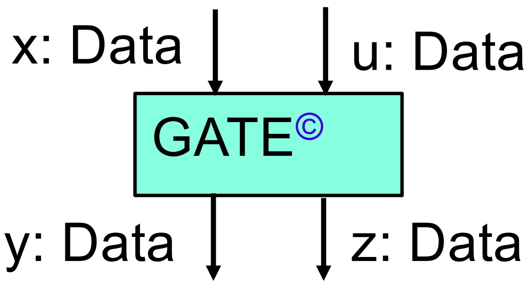
Note: If $Q(x, y) = false$ for all $x \in \vec{X}, y \in \vec{Y}$ then Q is strongly causal
Then does not exist a Moore machine with interface behavior that fulfils Q

Strongly causal refinement of GATE

$GATE^{©} = (x: Data, u: Data ▸ y: Data, z: Data):$

$\forall n \in Data: d\#x+d\#u = d\#y+d\#z \wedge (d\#x+d\#u = \infty \Rightarrow (d\#y = \infty \wedge d\#z = \infty))$

$\wedge \forall t \in \mathbb{N}: d\#x{\downarrow}t+d\#u{\downarrow}t \geq d\#y{\downarrow}(t+1)+d\#z{\downarrow}(t+1)$

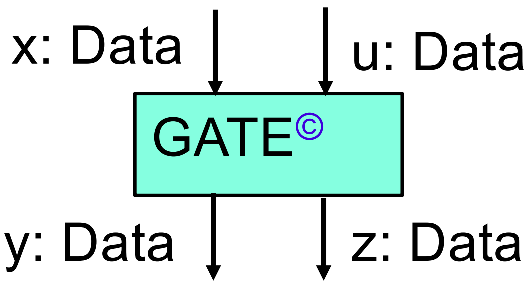


Strongly causal refinement of GATE

$\text{GATE}^{©} = (x\text{: Data}, u\text{: Data} \blacktriangleright y\text{: Data}, z\text{: Data})$:

$$\forall\, d \in \text{Data}: d\#x+d\#u \le d\#y+d\#z \wedge (d\#x+d\#u = \infty \Rightarrow (d\#y = \infty \wedge d\#z = \infty))$$

$$\wedge\ \forall\, t \in \mathbb{N}: d\#x{\downarrow}t+d\#u{\downarrow}t \ge d\#y{\downarrow}(t+1)+d\#z{\downarrow}(t+1))$$



Liveness

Strong causality, safety

Full Realizability

Strongly Causal Functions

$$f: \vec{X} \to \vec{Y}$$

is strongly causal if for all $x, z \in \vec{X}$, $t \in \mathbb{N}$

$$x{\downarrow}t = z{\downarrow}t \Rightarrow f(x){\downarrow}t{+}1 = f(z){\downarrow}t{+}1$$

Then we write SC[f]

Every strongly causal f has a unique fixpoint (Proof: Banach's Fixpoint Theorem)

Fully Realizable Predicates

Given $Q::(X \blacktriangleright Y)$

$$\text{Real}[Q] = \{f \in \vec{X} \rightarrow \vec{Y}: SC[f] \wedge \forall x \in \vec{X}: Q(x, f(x))\}$$

Real[Q] denotes the set of realizations of Q

Q is realizable if $\exists f \in \text{Real}[Q]$

Q is fully realizable if Q is realizable and

$Q(x, y) = \exists f \in \text{Real}[Q]: y = f(x)$

Every realization $f \in \text{Real}[Q]$ defines a strategy to compute $y = f(x)$ given x such that $Q(x, y)$ holds

Fully Realizable Predicates

For every predicate $Q::(X \blacktriangleright Y)$ there exists a weakest refinement $Q^{®}$ of Q

$$Q^{®}(x, y) = \exists f \in \text{Real}[Q]: y = f(x)$$

$Q^{®}$ is fully realizable if Q is realizable

$Q^{®} = \text{false}$ iff Q is not realizable

The interface behavior

$$[\![\Delta, \Lambda]\!]$$

of Moore machines (Δ, Λ) is strongly causal and fully realizable

Theorem: Properties of timed interface behavior of Moore machines

For the interface behavior

$[\![\Delta, \Lambda]\!] :: (\Delta, \Lambda)$

of Moore machines (Δ, Λ) with syntactic interface $(X \blacktriangleright Y)$ we get:

$[\![\Delta, \Lambda]\!]$ is strongly causal

$[\![\Delta, \Lambda]\!]$ is fully realizable

Theorem

For every fully realizable interface predicate $Q::(X \blacktriangleright Y)$ there exists a Moore machine (Δ, Λ) with

$$Q = [\![\Delta, \Lambda]\!]$$

From theory to practice

The introduced theory has a strong relationship to practical interactive computing

- Realizations of interface specifications
  - ◊ Have unique fixpoints (Banach, see also later)
  - ◊ Represent computations
- The interface Moore machines corresponds to fully realizable interface behaviors
- Causality models the relationship between input and output as found for practical systems
- Fully realizable interface behaviors form a semantic model for the interface behavior of Moore machines
  - ◊ Realizability introduces a strategy to guarantee certain liveness conditions
- Timed interface behaviors

Concurrent Composition

Compositionality of Syntactic Interfaces

Two syntactic interfaces $(X_k \blacktriangleright Y_k)$ for $k = 1, 2$, are called composable if

$$X_1 \cap X_2 = \emptyset$$

$$Y_1 \cap Y_2 = \emptyset$$

and the channels both in $X_1 \cup X_2$ and $Y_1 \cup Y_2$ have consistent types.

These channels in $(X_1 \cup X_2) \cap (Y_1 \cup Y_2)$ are (called) feedback channels.

Moore machines and interface predicates are called composable, if their syntactic interfaces are composable.

Concurrent composition of Moore machines

Moore machines $(\Delta_k,\Lambda_k)::(X_k \blacktriangleright Y_k)$, $k = 1, 2$, with composable syntactic interfaces, are composed concurrently to a Moore machine, $X = (X_1 \cup X_2)\backslash Y$, $Y = Y_1 \cup Y_2$,

$$((\Delta_1, \Lambda_1) \times (\Delta_2, \Lambda_2)::(X \blacktriangleright Y))$$

defined by

$$(\Delta, \Lambda) = ((\Delta_1, \Lambda_1) \times (\Delta_2, \Lambda_2))$$

where for

$$\Sigma = (\Sigma_1 \times \Sigma_2)$$

$$\Lambda = \{(\sigma_1, \sigma_2): \sigma_1 \in \Sigma_1 \wedge \sigma_2 \in \Sigma_2\}$$

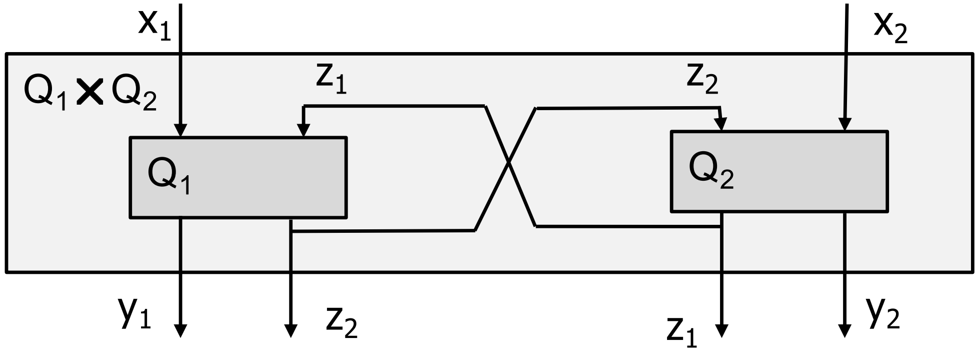
$$\Delta((\sigma_1, \sigma_2), x) = \{((\tau_1, \tau_2), y): (\tau_1, y|Y_1) \in \Delta_1(\sigma_1, x|X_1) \wedge (\tau_2, y|Y_2) \in \Delta_2(\sigma_2, x|X_2)\}$$

Theorems

$$[\![(\Delta_1, \Lambda_1) \times (\Delta_2, \Lambda_2)]\!] = ([\![\Delta_1, \Lambda_1]\!] \wedge [\![\Delta_2, \Lambda_2]\!])$$

$$(Q_1 \Rightarrow [\![(\Delta_1, \Lambda_1]\!]) \wedge (Q_2 \Rightarrow [\![\Delta_2, \Lambda_2]\!]) \Rightarrow ((Q_1 \wedge Q_2) \Rightarrow [\![(\Delta_1, \Lambda_1) \times (\Delta_2, \Lambda_2)]\!])$$

Concurrent Composition of Interface Specifications



Interface predicates Q_k::(X_k ► Y_k), k = 1, 2, with composable syntactic interfaces are composed concurrently to an interface predicate

$$(Q_1 \times Q_2)::(X \blacktriangleright Y)$$

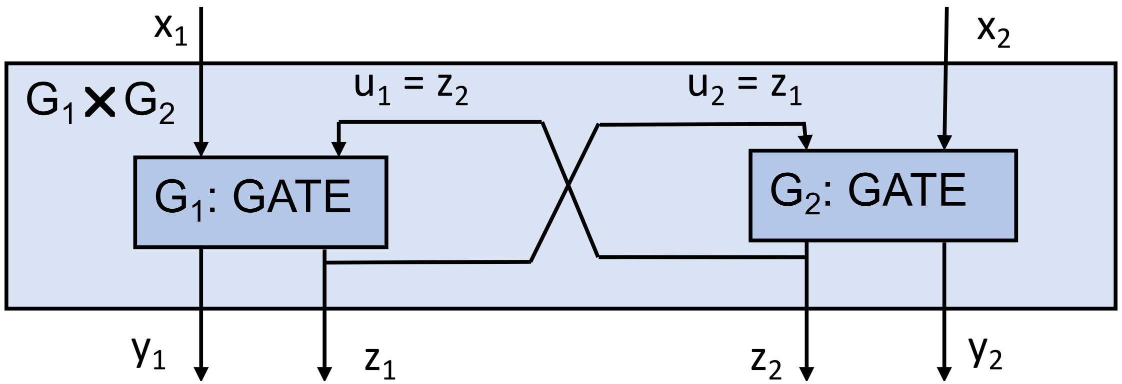
where $X = (X_1 \cup X_2) \backslash Y$, $Y = Y_1 \cup Y_2$ and its interface assertion is defined by

$$(Q_1 \times Q_2)^A = ((Q_1^{®})^A \wedge (Q_2^{®})^A)$$

Composition of two Gates

TG = (x_1: Data, x_2: Data ▶ y_1: Data, z_1: Data, y_2: Data, z_2: Data):

$\text{GATE}^{®}(x_1, z_2, y_1, z_1) \wedge \text{GATE}^{®}(x_2, z_1, y_2, z_2)$



Theorems

$(Q_1 \times Q_2)$ is strongly causal if Q_1 and Q_2 are strongly causal

$(Q_1 \times Q_2)$ is fully realizable if Q_1 and Q_2 are fully realizable

$$(Q_1 \times Q_2)^A \Rightarrow (Q_1{}^A \wedge Q_2{}^A)$$

Note

In most cases, for practical useful interface predicates, we have Q::(X ▸ Y)

$$Q^{©} = Q^{®}$$

Given an interface predicates Q::(X ▸ Y)

- There exists a Moore machine that implements Q iff Q is realizable
- There exists a Moore machine with an interface behavior equal to Q iff Q is fully realizable
- If $Q^{®}$ = false then there does not exist a Moore machine that implements Q

Hiding

Hiding of Output Streams

Given a specification

$$Q = (X \blacktriangleright Y) \colon A$$

where A is an interface assertion with free identifiers from X and Y and $Y' \subseteq Y$

$$(\text{Hide } Y' \colon Q) \colon\colon (X \blacktriangleright Y \backslash Y')$$

for $x \in \vec{X}$, $y' \in \overrightarrow{Y \backslash Y'}$

$$(\text{Hide } Y' \colon Q)(x, y') = \exists\, y \in \vec{Y} \colon Q(x, y) \wedge y' = y|(Y \backslash Y')$$

Abstraction of Time

Untimed streams and histories

$$M^{*|\omega} = M^* \cup M^\omega$$

Finite Streams: $M^* = \cup_{n \in \mathbb{N}} \{t \in \mathbb{N}: 1 \le t \le n\} \rightarrow M$

Infinite Streams: $M^\omega = \mathbb{N} \rightarrow M$

Data type of streams over set M: Str M

Untimed histories

$$\overline{X} = (X \rightarrow M^{*|\omega})$$

$$\overline{X}_{fin} = (X \rightarrow M^*)$$

Timed and untimed interface behavior specifications

- An interface predicate

$$Q: \vec{X} \times \vec{Y} \rightarrow \mathbb{B}$$

  is called a timed specification and we write Q :: (X ▶ Y)

- An interface predicate

$$R: \overline{X} \times \overline{Y} \rightarrow \mathbb{B}$$

  is called a untimed specification and we write R :: (X▷Y)

From timed to untimed histories and vice versa

Given a timed stream $x \in (M^*)^\omega$ we define an untimed stream $\bar{x} \in M^{*|\omega}$ by

$$\bar{x} = x(1)\hat{}x(2)\hat{}x(3)\hat{}\ldots$$

The same notation is used for histories.

Given a timed interface predicate $Q :: (X \blacktriangleright Y)$ we define an untimed interface predicate $\bar{Q} :: (X \triangleright Y)$ by

$$\bar{Q}(x', y') = \exists\, x \in \vec{X}, y \in \vec{Y}: Q(x, y) \wedge x' = \bar{x} \wedge y' = \bar{y}$$

For an untimed interface predicate $R :: (X \triangleright Y)$ define timed interface predicates $R^{>}, \vec{R} :: (X \blacktriangleright Y)$ by

$$R^{>}(x, y) = R(\bar{x}, \bar{y})$$

$$\vec{R} = (R^{>})^{\circledR}$$

Theorems

$$\overline{\overrightarrow{R}} \Rightarrow R$$

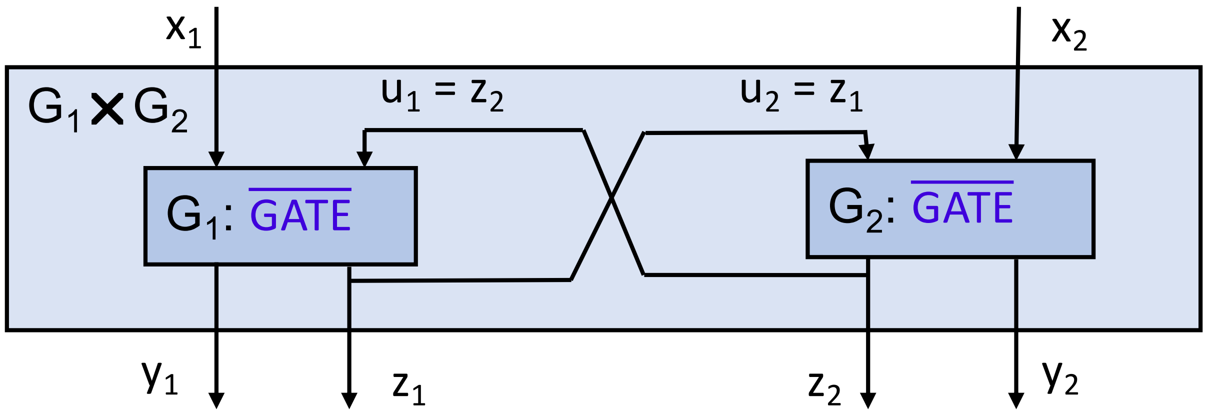
$$Q^{\circledR} \Rightarrow \overrightarrow{\overline{Q}}$$

Concurrent composition of untimed interface predicates

Composition of two untimed Gates

UTG = (x_1: Data, x_2: Data ▸ y_1: Data, z_1: Data, y_2: Data, z_2: Data):

$$\overline{\text{GATE}}(x_1, z_2, y_1, z_1) \wedge \overline{\text{GATE}}(x_2, z_1, y_2, z_2)$$



Reasoning about $\overline{\text{GATE}}(x_1, z_2, y_1, z_1) \wedge \overline{\text{GATE}}(x_2, z_1, y_2, z_2)$

Looking for solutions for these feedback equations for streams z_1 and z_2, consider as a simple example the following input streams

$$x_1 = \langle 1\ 1\rangle \wedge x_2 = \langle 2\ 2\rangle$$

This input leads (for all $d \in \text{Data}$) to the assertions

$$d\#\langle 1\ 1\rangle + d\#z_2 = d\#y_1 + d\#z_1 \wedge ((d\#\langle 1\ 1\rangle + d\#z_2) = \infty \Rightarrow (d\#y_1 = \infty \wedge d\#z_1 = \infty))$$

$$d\#\langle 2\ 2\rangle + d\#z_1 = d\#y_2 + d\#z_2 \wedge ((d\#\langle 2\ 2\rangle + d\#z_1) = \infty \Rightarrow (d\#y_2 = \infty \wedge d\#z_2 = \infty))$$

Solutions for the defining equations:

$$y_1 = \langle 1\ 1\rangle \wedge z_1 = \langle\rangle \wedge y_2 = \langle 2\ 2\rangle \wedge z_2 = \langle\rangle$$

$$y_1 = \langle 1\ 1\rangle \wedge z_1 = \langle 1\ 2\rangle \wedge y_2 = \langle 2\ 2\rangle \wedge z_2 = \langle 1\ 2\rangle$$

$$y_1 = \langle 1\ 2\rangle \wedge z_1 = \langle 1\ 2\ 1\rangle \wedge y_2 = \langle 2\ 1\rangle \wedge z_2 = \langle 2\ 1\ 2\rangle$$

$y_1 = \langle 1\ 1\rangle \wedge z_1 = \langle 3\rangle \wedge y_2 = \langle 2\ 2\rangle \wedge z_2 = \langle 3\rangle$ here z_1 and z_2 correspond to noncausal fixpoints

Concurrent composition of untimed interface predicates

Concurrent composition of composable untimed interface predicates $R_k::(X_k \triangleright Y_k)$, for $k = 1, 2$, leads to an untimed interface predicate

$$R :: (X \triangleright Y)$$

where $X = (X_1 \cup X_2) \backslash Y$, $Y = Y_1 \cup Y_2$, defined by (excluding noncausal fixpoints)

$$R = \overline{\vec{R}_1 \wedge \vec{R}_2}$$

We write

$$R_1 \times R_2 = \overline{\vec{R}_1 \wedge \vec{R}_2}$$

Refinement of Untimed Interface Predicates

The untimed interface predicate $\overline{\overrightarrow{R_k}}$ is a refinement of untimed interface predicate R_k where $\overrightarrow{R_k}$ is the weakest time dependent interface predicate of a Moore machine the time abstraction of which is a refinement of R_k.

For composable untimed interface predicates $R_k :: (X_k \triangleright Y_k)$ we conclude

$$\overrightarrow{R_k} \Rightarrow (R_k^{>})^{©}$$

$$\overline{\overrightarrow{R_k}} \Rightarrow R_k$$

$$\overline{(\overrightarrow{R_1} \times \overrightarrow{R_2})}^{A} \Rightarrow R_1^{A} \wedge R_2^{A}$$

$$\overrightarrow{R_1} \times \overrightarrow{R_2} \Rightarrow (R_1^{>})^{©} \wedge (R_2^{>})^{©}$$

Composition of untimed GATE

$$G_1 = GATE(x_1, z_2, y_1, z_1)$$
$$G_2 = GATE(x_2, z_1, y_2, z_2)$$

$$UTG = \overline{G_1} \times \overline{G_2}$$

$$UTG = \overline{\overrightarrow{\overline{G_1}} \times \overrightarrow{\overline{G_2}}}$$

The effect of adding strong causality

$$(\overline{G_1}^{>})^{©} \wedge (\overline{G_2}^{>})^{©}$$

implies the assertions $\forall$ d $\in$ Data, t $\in \mathbb{N}$:

$$d\#x_1{\downarrow}t+d\#z_2{\downarrow}t \geq d\#y_1{\downarrow}(t+1)+d\#z_1{\downarrow}(t+1) \wedge d\#x_2{\downarrow}t+d\#z_1{\downarrow}t \geq d\#y_2{\downarrow}(t+1)+d\#z_2{\downarrow}(t+1)$$

By induction on t we prove from this equation (for timed and nontimed streams)

$$d\#x_1 = 0 \wedge d\#x_2 = 0 \Rightarrow d\#z_1 = 0 \wedge d\#z_2 = 0$$

which excludes for input streams $x_1 = \langle 1\ 1 \rangle \wedge x_2 = \langle 2\ 2 \rangle$ the feedback according to the noncausal fixpoint

$$y_1 = \langle 1\ 1 \rangle \wedge z_1 = \langle 3 \rangle \wedge y_2 = \langle 2\ 2 \rangle \wedge z_2 = \langle 3 \rangle$$

Theorem

Concurrent composition of composable fully realizable untimed interface predicates $R_k::(X_k \triangleright Y_k)$, for $k = 1, 2$, leads to an interface predicate

$$R_1 \times R_2 :: (X \triangleright Y)$$

where $X = (X_1 \cup X_2) \backslash Y$, $Y = Y_1 \cup Y_2$, which is fully realizable!

Moreover

$$(R_1 \times R_2)^A \Rightarrow R_1^A \wedge R_2^A$$

What we prove from $R_1^A \wedge R_2^A$ holds for $(R_1 \times R_2)^A$

What did we do?

For an untimed interface predicates $R_k::(X_k \triangleright Y_k)$ the interface predicate:

$$(X \triangleright Y): R_1^A \wedge R_2^A$$

is in general too weak to identify causal fixpoints. Therefore, we instead consider

$$(X \triangleright Y): (\overline{\overrightarrow{R}_1 \wedge \overrightarrow{R}_2})^A$$

Deriving the exact specification for concurrent composition of untimed interface predicates, we

- consider their timed versions, complete them by full realizability (resulting in false, if they are not realizable),
- compose the result by concurrent composition and
- go back to the untimed version (by time abstraction, which is either false or fully realizable).

What justifies this idea?

Every untimed system is executed

- by a timed implementation (a Moore machine);
- therefore we can use this construct to define the precise result of concurrent composition of untimed system specifications.

Observations

- The time abstraction for the interface predicate of a Moore machine

$$\overline{[\![\Delta, \Lambda]\!]}$$

  is well defined and fully realizable
- There are Moore machines with different timed interface behaviors with identical untimed interface behaviors ("abstraction")
- For every untimed interface behavior of a Moore machine there is a most general Moore machine with this untimed interface behavior.

Abstractions

- A Moore (Δ, Λ) machine for syntactic interface $(X \blacktriangleright Y)$ can be abstracted to its fully realizable timed interface behavior

$$[\![\Delta, \Lambda]\!] :: (X \blacktriangleright Y)$$

- A fully realizable interface behavior $Q :: (X \blacktriangleright Y)$ can be abstracted
  - ◇ into a timed interface specification $P :: (X \blacktriangleright Y)$ (a nucleus) such that

$$Q = P^{®}$$

  - ◇ into an untimed interface behavior

$$\overline{Q} :: (X \triangleright Y)$$

Untimed systems: from theory to practice

- We get a model for untimed interactive computations
- The notions worked out for timed systems carry over to untimed systems
  - ◊ The critical task of concurrent composition of nondeterministic untimed interface behaviors (represented by predicates) which requires the identification of “least” fixpoints for feedback loops is solved by using results from timed systems
- Fully realizable untimed systems form a semantic model for the untimed interface behavior of Moore machines (abstracting away the time steps)

From timed to untimed interface predicates and vice versa

Time Abstraction and Timed Representation

- Abstraction

$$\text{Abs: } \vec{Y} \to \overline{Y}$$

$$\text{Abs}(y) = \overline{y}$$

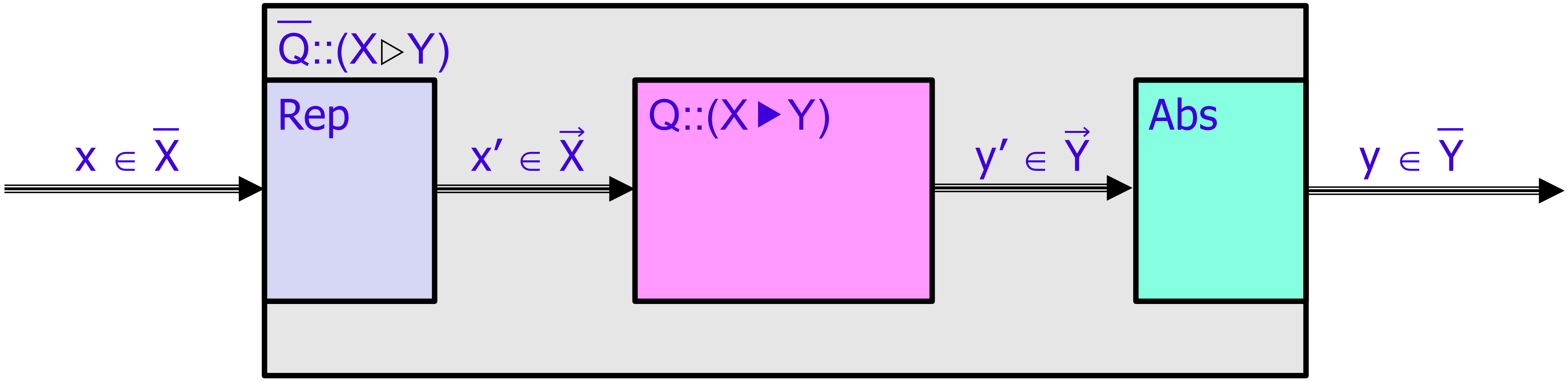
- Representation

$$\text{Rep: } \overline{X} \to (\vec{X})$$

$$\text{Rep}(x) = \{z \in \vec{X}\text{: } \overline{z} = x\}$$

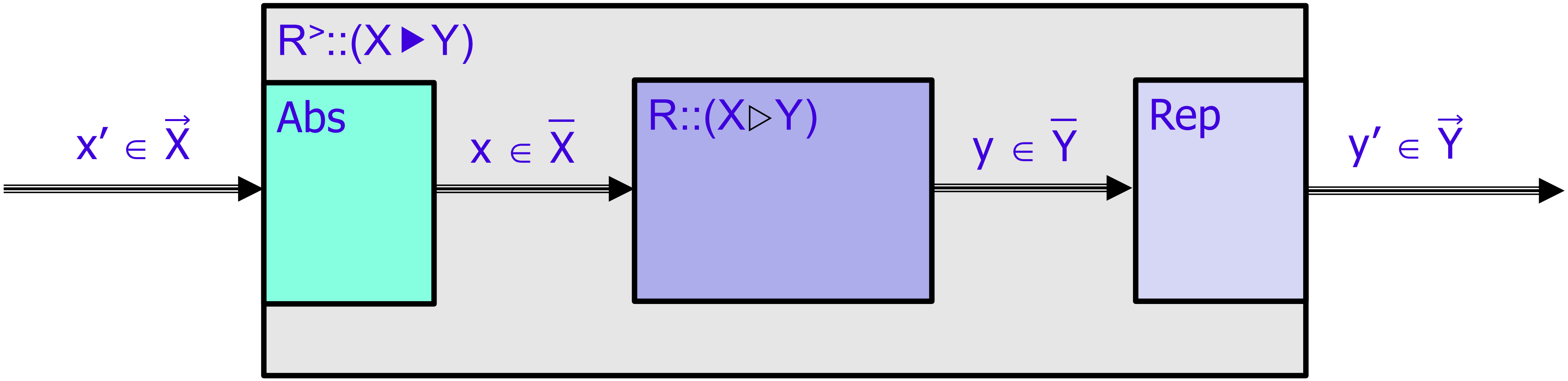
Time Abstraction

$\overline{Q} = \text{Rep} \circ Q \circ \text{Abs}$



Time Introduction: From Untimed to Timed Interface Predicates

$$R^{>} = Abs \circ R \circ Rep$$



Fully realizable untimed interface predicates

An untimed interface predicate R :: (X ▶ Y) is called fully realizable if

$\overrightarrow{R}$ is realizable

and

$$\overline{\overrightarrow{R}} = R$$

Time insensitive timed system

A fully realizable timed interface predicate Q :: (X ▶ Y) is called time insensitive if

$$\bar{z} = \bar{x} \Rightarrow \{\bar{y} \in \bar{Y} : Q(x, y)\} = \{\bar{y} \in \bar{Y} : Q(z, y)\}$$

- GATE, TCG and TCBG are time insensitive, TCIG is not time insensitive

Predicate Q is called fully time insensitive if Q is time insensitive and

$$\overrightarrow{\overline{Q}} = Q$$

- Then timing of input x to Q influences the timing of the output y but all data outputs $\bar{y}$ possible for any input z such that $\bar{z} = \bar{x}$ are also possible for input x.
- The timing of output is only restricted by the causality!
- If Q is not time insensitive then there exists input x and z where $\{\bar{y} \in \bar{Y} : Q(x, y)\} \neq \{\bar{y} \in \bar{Y} : Q(z, y)\}$; the data output may depend on the timing.

Theorem

- If a specification is time insensitive we can reason about its data flow independent of its timing
  - ◊ The data output depends only on the data input
  - ◊ Of course the timing of the output may depend on the timing of the input
  - ◊ Time abstraction maintains the relation between untimed input and untimed output
- A specification is not time insensitive if its data output is not independent of the timing of its input

Properties of fully realizable untimed interface predicates: beyond prefix monotonicity

Fully realizable untimed interface predicates of Moore machines are not prefix monotonic, in general: Example: GATE – $\overline{\text{GATE}}$ is fully realizable!

$$\overline{\text{GATE}}(1^\infty, \langle\rangle, y, z) \Rightarrow y = 1^\infty \wedge z = 1^\infty$$

$$\overline{\text{GATE}}(1^\infty, 2^\infty, y', z') \Rightarrow 1\#y' = \infty \wedge 2\#y' = \infty \wedge 1\#z' = \infty \wedge 2\#z' = \infty$$

$\overline{\text{GATE}}$ is not prefix monotonic, since

$$1^\infty \sqsubseteq 1^\infty \wedge \langle\rangle \sqsubseteq 2^\infty$$

but

$$\overline{\text{GATE}}(1^\infty, \langle\rangle, y, z) \wedge \overline{\text{GATE}}(1^\infty, 2^\infty, y', z')$$

$$\Rightarrow$$

$$\neg(y \sqsubseteq y') \wedge \neg(z \sqsubseteq z')$$

Practical Impacts to
Software and Systems Engineering

Topics

- Refinement
- Modularity
  - Compositionality
- Verification calculus
  - Soundness and relative completeness
- Architecture
  - Layered architectures
  - Distribution
- Assumption/commitment specifications
- Feature interactions
- Explicit parallelism
- Real time
- Cyber-physical systems

Modularity

A formal system specification and implementation framework is modular, if

- there is an implementation and specification calculus such that for system S and specification Q (of the same syntactic interface) we write and deduce

$$S \vdash Q$$

  which means system S fulfils specification Q

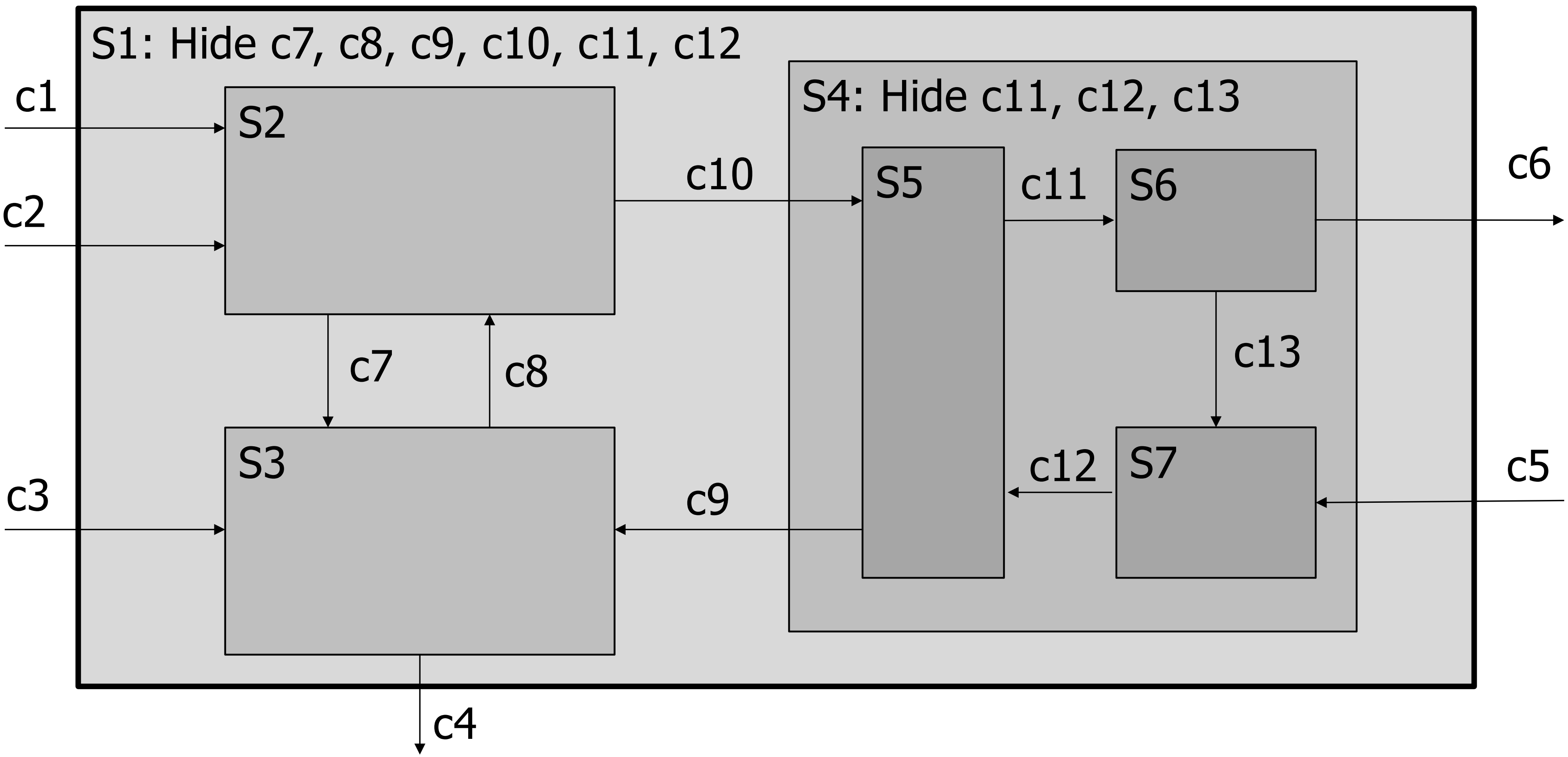
- For every composition operator $\times$ for systems S_1 and S_2 there is a composition operator $\times$ for specifications Q_1 and Q_2, such that

$$(S_1 \vdash Q_1 \wedge S_2 \vdash Q_2) \Rightarrow S_1 \times S_2 \vdash Q_1 \times Q_2$$

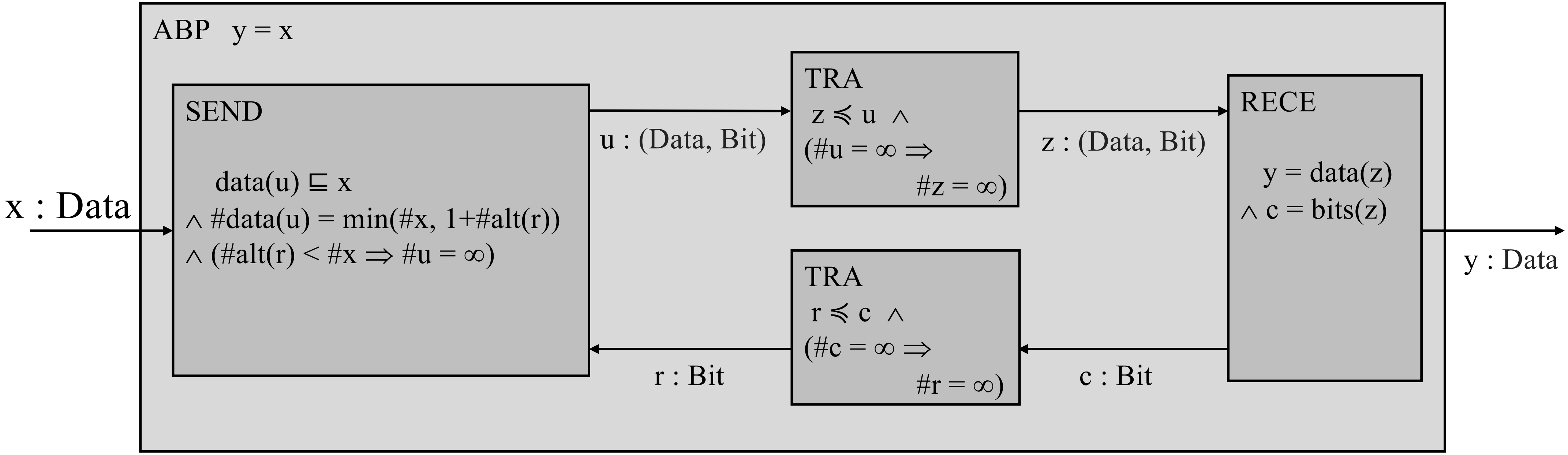
- Note that in our case the system model is a subset of the specification model

Architecture

Hierarchical Architecture: Graphical Representation



Architecture: Alternating Bit

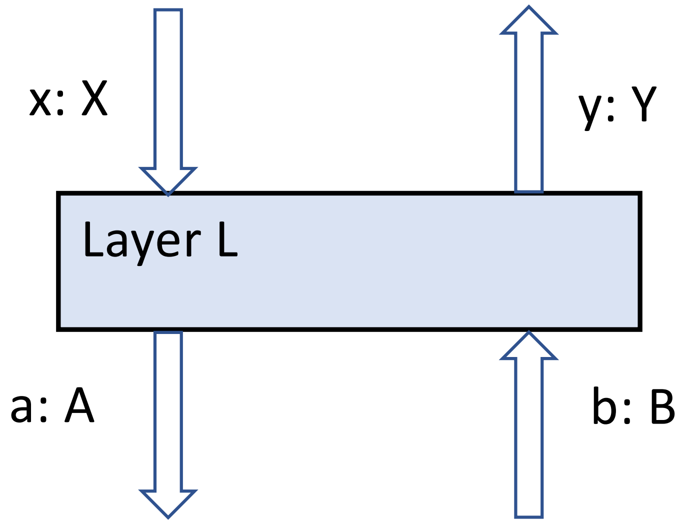


Layered Architectures

Layers in Layered Architectures

- Layered architectures have many advantages.
- In many applications, therefore layered architectures are applied.

$L = (x: \vec{X}, b: \vec{B} \blacktriangleright y: \vec{Y}, a: \vec{A}): R(a, b) \Rightarrow Q(x, y)$



Let the interface behavior

$$S = (x: \vec{X} \blacktriangleright y: \vec{Y}): Q(x, y)$$

denote the provided service and

$$W = (a: \vec{A} \blacktriangleright b: \vec{B}): R(a, b)$$

denote the required service.

Forming Layered Architectures

We have two layers (k = 1, 2)

$$L_k = (x_k: \vec{X}_k, b_k: \vec{B}_k \blacktriangleright y_k: \vec{Y}_k, a_k: \vec{A}_k): R_k(a_k, b_k) \Rightarrow Q_k(x_k, y_k)$$

that fit syntactically together, if

$X_1 = A_2$ and $Y_1 = B_2$,

and semantically if the provided service

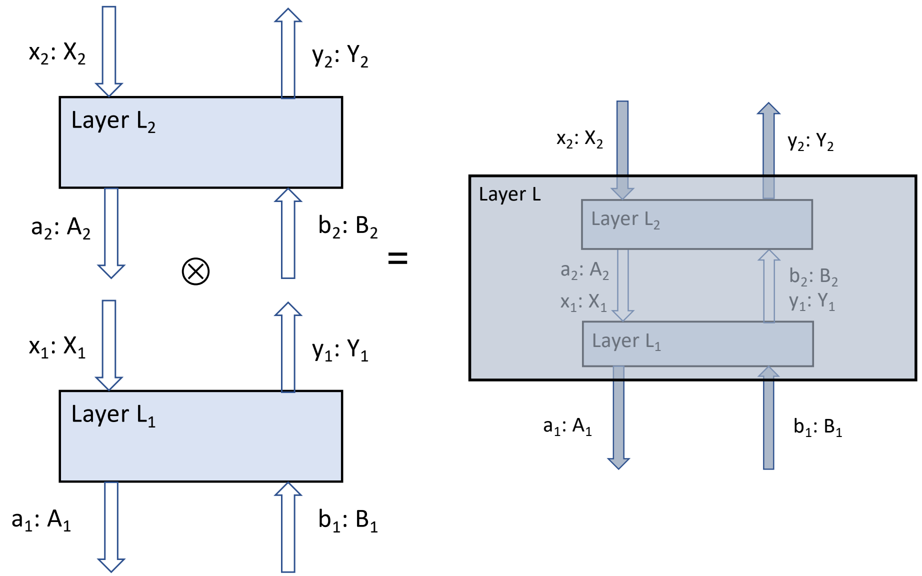
$$S_1 = (x_1: \vec{X}_1 \blacktriangleright y_1: \vec{Y}_1): Q_1(x_1, y_1)$$

of the lower layer L_1 is a refinement of the requested service

$$W_2 = (a_2: \vec{A}_2 \blacktriangleright b_2: \vec{B}_2): R_2(a_2, b_2)$$

of the upper layer L_2 which means (note that $X_1 = B_2$ and $Y_1 = A_2$)

$$Q_1(x_1, y_1) \Rightarrow R_2(x_1, y_1)$$



Proof

We compose the two layers to a system L

$$L$$

$$= \text{Hide } x_1 \in : \vec{X}_1,\ y_1: \vec{Y}_1: L_1 \times L_2$$

$$= (x_2: \vec{X}_2,\ b_1: \vec{B}_1 \blacktriangleright y_2: \vec{Y}_2,\ a_1: \vec{A}_1): \exists\, x_1 \in : \vec{X}_1,\ y_1: \vec{Y}_1:$$

$$(R_1(a_1, b_1) \Rightarrow Q_1(x_1, y_1)) \wedge (R_2(x_1, y_1) \Rightarrow Q_2(x_2, y_2))$$

If $Q_1(x_1, y_1) \Rightarrow R_2(x_1, y_1)$ holds we conclude

$$L = (x_2: \vec{X}_2,\ b_1: \vec{B}_1 \blacktriangleright y_2: \vec{Y}_2,\ a_1: \vec{A}_1): (R_1(a_1, b_1) \Rightarrow Q_2(x_2, y_2))$$

System L which is the result of composing the two layers is a layer again with the provided service of layer L_2 and the requested service of layer L_1.

Forming Layered Architectures

If the layers fit together, we get a layered architecture

$L_k = (x_k:\vec{X}_k,\ b_k:\vec{B}_k \blacktriangleright y_k:\vec{Y}_k,\ a_k:\vec{A}_k): R_k(a_k, b_k) \Rightarrow Q_k(x_k, y_k)$

that fit syntactically together, if

$X_k = A_{k+1}$ and $Y_k = B_{k+1}$,

and semantically if the provided service

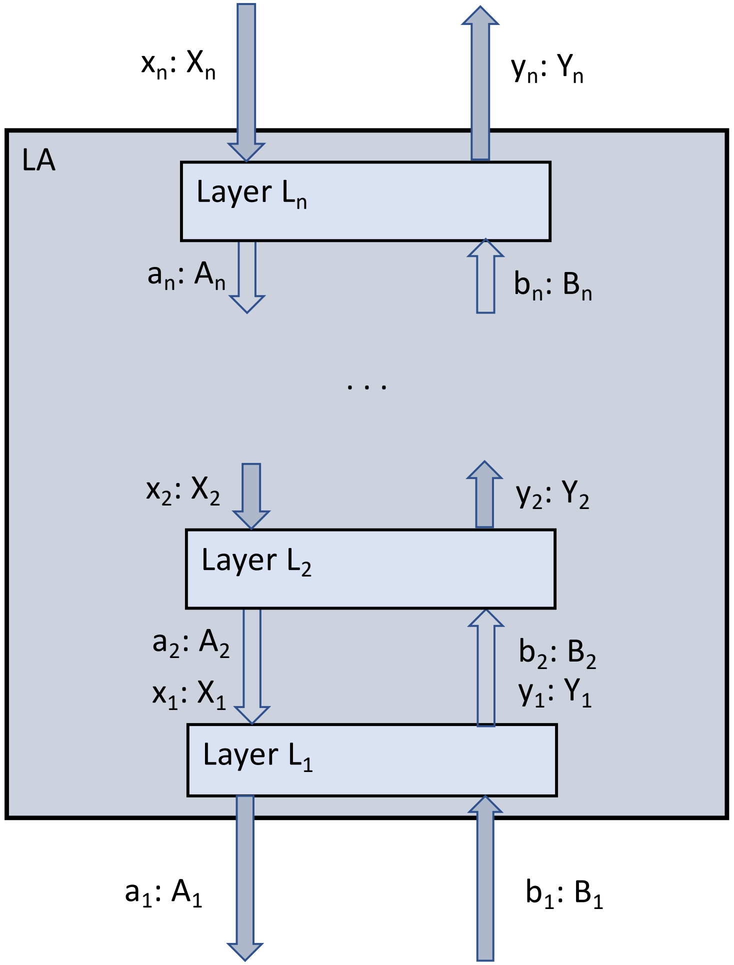
$S_k = (x_k: \vec{X}_k \blacktriangleright y_k: \vec{Y}_k): Q_k(x_k, y_k)$

of lower layer L_k is a refinement of the requested service

$W_{k+1} = (a_{k+1}: \vec{A}_{k+1} \blacktriangleright b_{k+1}: \vec{B}_{k+1}): R_2(a_{k+1}, b_{k+1})$

of the upper layer L_2 which means

$Q_k(x_k, y_k) \Rightarrow R_{k+1}(x_k, y_k)$



Feature Interaction

Projection

Given a specification

$$(X \blacktriangleright Y): Q$$

where $X' \subseteq X$, $Y' \subseteq Y$

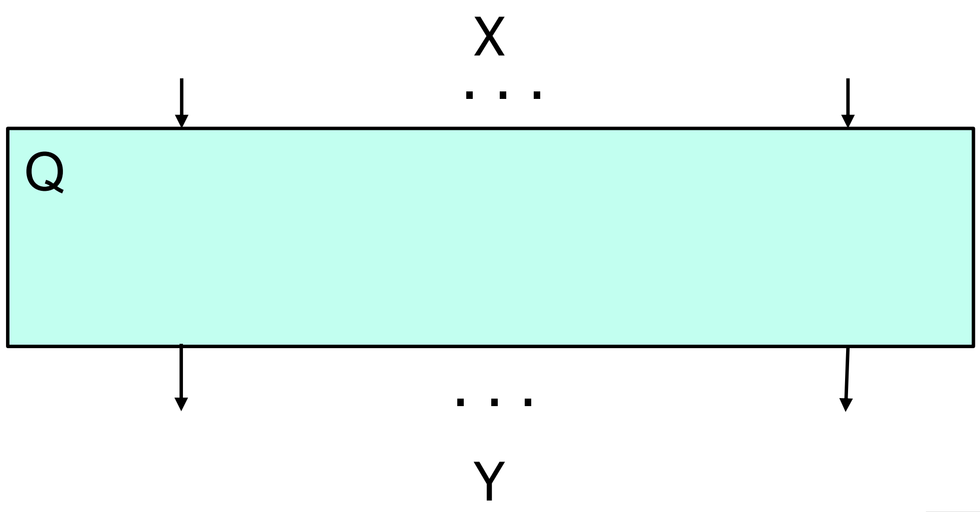
a subservice $Q\dagger(X' \blacktriangleright Y')$ is defined

by projection

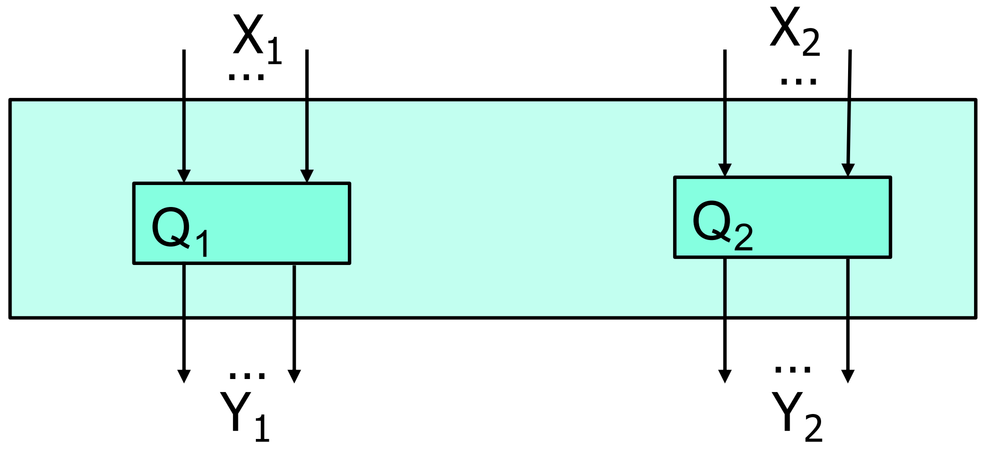
$$(Q\dagger(X' \blacktriangleright Y'))(x', y') = \exists\, x \in \vec{X}, y \in \vec{Y}: Q(x, y) \wedge x' = x|X' \wedge y' = y|Y'$$

Feature interaction

Can we decompose a system



into

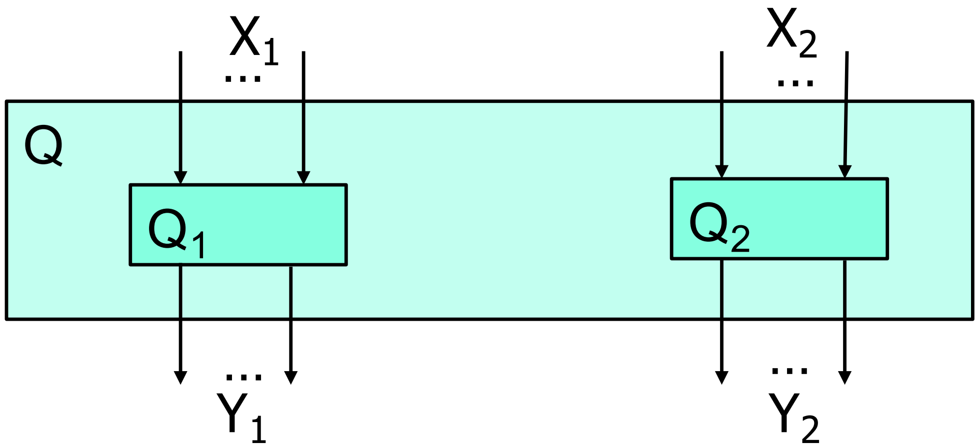


Feature Interaction

Let $X = X_1 \cup X_2$, $Y = Y_1 \cup Y_2$, where the sets X_1, X_2, Y_1, and Y_2 are pairwise disjoint

The subservices $Q_1 = Q|(X_1 \blacktriangleright Y_1)$ and $Q_2 = Q|(X_2 \blacktriangleright Y_2)$ of service Q are free of feature interactions if

$$Q(x, y) = (Q_1(x|X_1, y|Y_1) \wedge Q_2(x|X_2, y|Y_2))$$



Modelling Physical Devices

- Control Theory (Regelungstechnik)
  - ◊ Control theory deals with the control of dynamical systems in engineered processes and machines.
  - ◊ The objective is to develop a model or algorithm governing the application of system inputs to drive the system to a desired state, while minimizing any delay, overshoot, or steady-state error and ensuring a level of control stability;
  - ◊ often with the aim to achieve a degree of optimality.
- Control Theory works with continuous functions over the parameter time, with differential and integral equations and the notion of stability

Controller and System as Relations on Streams

- Controller:

Control Layer

in x, b: Stream Data
out y, a: Stream Data

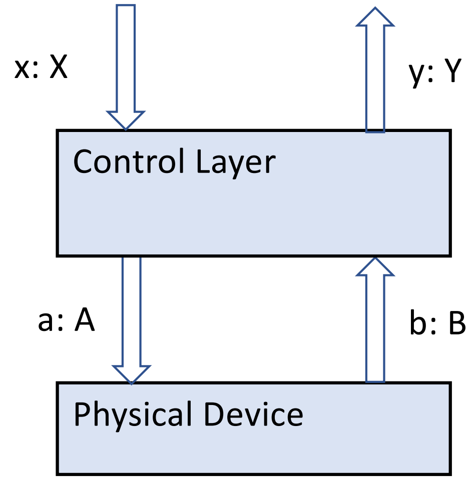
Interface Assertion

- System

Physical Device

in a: Stream Data
out b: Stream Data

Interface Assertion



Specifying the Physical Device: Simple Example Automatic Window

- The state space is used to model the state of the window.
- A state consists of two attributes:

  mode: {stopped, goin_up, goin_down, alarm}

  p: [0:100]

- Here p stands for position and represents the position of the window.
- The position p = 100 holds if the window is closed, p = 0 holds if the window is open.
- The mode indicates the actual movement of the window, the position indicates how far the window is closed.
- The state mode = goin_up, position = 50 models the state of the window moving up in a situation where it is half closed.

Input and Output

- The control input to the system and its output are given by the following two sets:

Input = {open, close, stop}

Output = {open, closed, stopped, alarm, mov_up, mov_down}

State Transition Function

- The state transition function is defined as follows
  - (we write for any set M the set $M_+ = M \cup \{\varepsilon\}$ where ε stands for *no message*):

$$\Delta: \text{State} \times \text{Input}_+ \rightarrow \wp(\text{State} \times \text{Output}_+)$$

Table: State Transitions

- The attributes of the next state are represented by mode’ and p’.

mode	p	input	mode’	p’	output
≠ alarm		stop	stopped	= p	stopped
stopped		ε	stopped	= p	stopped
goin_down | stopped		close	goin_up	= p	mov_up
goin_up | stopped		open	goin_down	= p	mov_down
goin_up	= 100	ε | close	stopped	= 100	closed
goin_up	< 100	ε | close	goin_up	> p	mov_up
goin_up			alarm	= p	alarm
goin_down	= 0	ε | open	stopped	= 0	open
goin_down	> 0	ε | open	goin_down	< p	mov_down
alarm	> 0		alarm	< p	alarm
alarm	= 0		stopped	= 0	open

Interpretion of the table

The table Tab. 1 defines the state transition relation by a disjunctive formula. Every line in the table defines an assertion. For instance, the following line

goin_up	= 100	ε | close	stopped	= 100	closed

represents the conjunctive formula:

$$mode = goin\_up \wedge p = 100 \wedge (input = \varepsilon \vee input = close)$$

$$\wedge\ mode' = stopped \wedge p' = 100 \wedge output = closed$$

These conjunctive formulas represented by the lines of the table are connected by disjunction to deduce the formula that specifies the state transition from data.

The Behavior of the Physical System as a Relation on Streams

The specification of a relation representing the function

$$\varphi\text{: State} \rightarrow (\ (\text{Stream Input} \times \text{Stream Output}) \rightarrow \text{IB}\ )$$

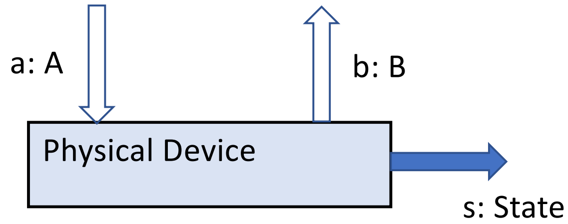
that describes the specification of the behavior of the physical device is derived from the state transition function as follows

$$\varphi(\sigma)(\langle e\rangle\hat{}a, \langle r\rangle\hat{}b) = \exists\ \sigma' \in \text{State:}\ \ (\sigma', r) \in \Delta(\sigma, e) \wedge \varphi(\sigma')(a, b)$$

State Observations

The behavior of the physical system is defined

- By a state machine with input and output
- The states model the state of the physical systems
- The state machine defines a relation φ between
  - the input stream(s) – the stream of actuator signals – and
  - the output stream(s) – the stream of sensor signals
- The relation φ can be extended to a behavior ϕ of the physical systems in terms of its states



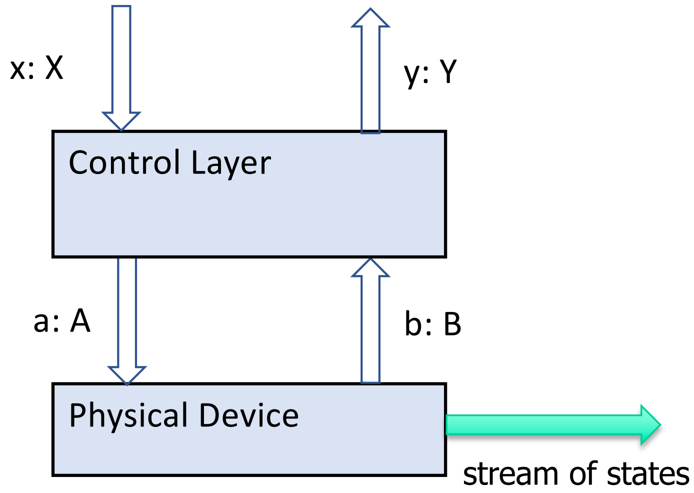
$$\phi\text{: State} \rightarrow (\ (\text{Stream Input} \times \text{Stream Output} \times \text{Stream State}) \rightarrow \text{IB}\ )$$

$$\phi(\sigma)(\langle e\rangle\hat{}a, \langle r\rangle\hat{}b, \langle\sigma'\rangle\hat{}s) = ((\sigma', r) \in \Delta(\sigma, e) \wedge \phi(\sigma')(a, b, s))$$

Conclusion

- We compose the physical device specified by the interface assertion $\phi(\sigma_0)(a, b, s)$ where σ_0 is the initial state of the physical device with the
  - ◊ control layer specified by the interface assertion $CL(x, b, y, a)$ and get the interface assertion of the composed system

$$\exists\ a, b: CL(x, b, y, a) \ \wedge\ \phi(\sigma_0)(a, b, s)$$



Design Framework

Semantic driven system development

- Encapsulation
  - ◊ Form architectural elements with interfaces that encapsulate the access by interfaces
- Information hiding
  - ◊ Hide implementation details not needed to understand the effect on the context
- Functional abstraction: Model the interface including interface behavior
- Composition
  - ◊ Define the interface behavior of composed systems from the interface behavior of the components
- Interface refinement
  - ◊ Make specifications more detailed
- Modularity (generalization of Liskov‘s substitution principle)
  - ◊ Guarantee that refinement of specifications of components leads to refinement of specifications of composed systems

Concluding Remarks

- Expressive power and flexibility
  - ◊ In principle all kinds of behavior can be specified
  - ◊ Specifications can be noncausal, weakly or strongly causal, realizable or fully realizable
- Specification, composition, verification and refinement by a calculus that is
  - ◊ Sound
  - ◊ Relatively complete
  - ◊ Making specification f.r. (often s.c. is enough) is sufficient for all proofs
- Methodological extensions
  - ◊ Assumption/Commitment specifications
  - ◊ Time free specifications
- Architecture design by specifications
  - ◊ Distributed concurrent systems
- Further Extensions
  - ◊ Infinite networks (recursive definitions of networks)
  - ◊ Dynamic systems
  - ◊ Probability

Topics for future research

- A tool for proving in the calculus
- A programming language for implementation
- Probabilities for interface behavior
- A time free version for non-time-sensitive interface specifications
    - ◊ Ambiguous operators