Interaction, Concurrency, Nondeterminism, Time, Composition, Distribution, Abstraction

An Interface Centric Approach

Practical and Theoretical Consequences

Manfred Broy

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

Topics of Concurrency

Theoretical

- Nondeterminism, concurrency
  - Parallel operators (parallel or)
  - Ambiguity
- State machines
- Computability
  - Algorithms
  - Models of computability
  - Time
  - Infinite computations
  - Unbounded nondeterminism
- Denotational semantics
- Fixpoint theory

Practical

- Nondeterminism and ambiguity
- Abstraction: Interface behavior
- Modularity
  - Encapsulation, information hiding, interface behavior
- Real time
- Graphical models
- Specification
- Distribution and architecture
  - Composition
- Verification
- Missing programming languages

The Two Basic Models

State based models of concurrency

- Influenced by von Neumann architecture: shared state
- Interleaving concurrency
  - implicit
  - nondeterminism
  - deadlock
- State based assertion techniques
  - ghost variables,
  - stuttering
  - prophecy variables
- Composition
  - fairness
  - intensional

History based models of concurrency

- Data Flow
- Infinite computations
  - streams and histories
- Explicit Concurrency
- Safety and liveness
- Composition
  - compositionality
  - extensionality principle
- Distribution
- Abstraction: modularity
  - information hiding/encapsulation
- Components

General Observations

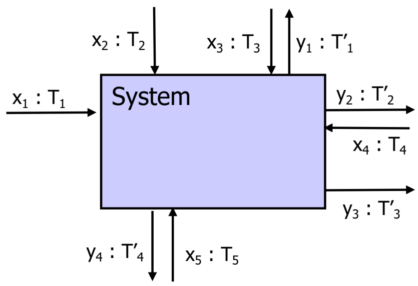
- Numerous models
  - Petri Nets, Data Flow, TLA , CSP, CCS, B, Unity, Rely/Guarantee, State Charts, Esterel, …
- Missing studies of the sufficient comparisons of different approaches
- Theoretical consequences not sufficiently investigated
  - How does the notion of algorithm generalize to concurrency and vice versa
  - What about computability when considering nondeterminism, concurrency and/or time
- Practice versus theory
  - In theoretical approaches practical consequences often not sufficiently taken care of
  - In practical approaches theoretical consequences often not sufficiently taken care of
- Programming languages based on the Neumann architectures
  - Shared state
- A lot of concepts on low level implementation issues
  - Operating systems, scheduling, bus systems

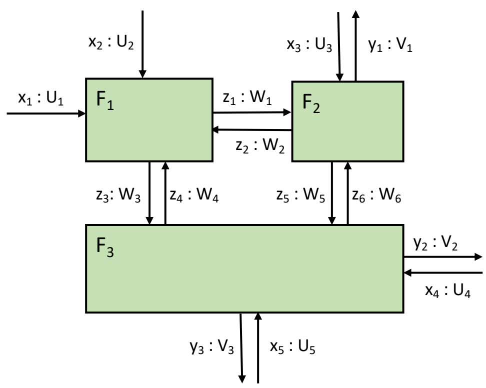
Practical challenges

- System specification
  - At what abstraction level?
  - Specifying concurrent algorithms or functional behavior of distributed systems
- System composition
  - Composition of system specifications
  - Compositionality
  - Modularity
  - Compositional verification
- Cyber physical systems
  - Modeling physical devices
- Real time
  - Time out
  - Delay
  - Urgency
- Levels of abstraction
  - Platform independent models of concurrent systems
  - Platform specific models of concurrent systems
- Distribution
- Safety and liveness
  - Fairness
- Design
  - Architecture
- Interface specification
  - Multiservice systems
  - Feature interaction between services
  - Assumption/commitment
  - Provided and required services

Interface Based Modelling Theory

- Interface Model
  - Syntactic
  - Behavioral
- Architecture Model
  - Composition
  - Feedback
- Expressive power
  - Data flow
  - Time flow
- System specification
- System composition
- System Verification
- Operational models





Discrete systems: the modeling theory in a nutshell

Sets of typed channels

$X = \{x_1 : T_1, x_2 : T_2, \dots \}$

$Y = \{y_1 : S_1, y_2 : S_2, \dots \}$

syntactic interface

$(X \blacktriangleright Y)$

data stream of type T

$\text{STREAM } T = \{ \mathbb{N}\setminus\{0\} \rightarrow T^* \}$

valuation of channel set X

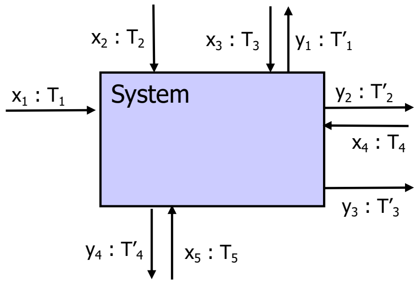
$\vec{X} = \{X \rightarrow \text{STREAM}[T]\}$

interface behavior for syn. interface $(X \blacktriangleright Y)$

interface predicates

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

represented by interface assertions:
logical formula with channel names
as variables for streams



Forms of models
- mathematical
- logical
- graphical

Example: Interface Specification - Data flow

$\text{MIX} = (x, z: \text{Tstr } M \blacktriangleright y: \text{Tstr } M): \forall m \in M: m\#x+m\#z = m\#y$

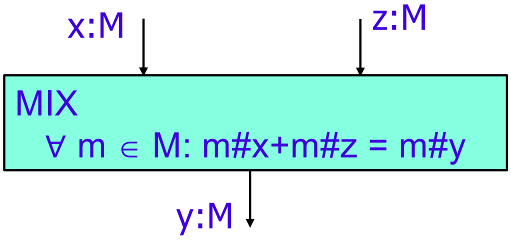
textual

MIX

in x, z: TSTR M
out y: TSTR M

$\forall m \in M: m\#x+m\#z = m\#y$

by tableau



graphical

Streams

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

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

Timed Streams: Illustration: Time Flow and Data Flow

$\tilde{x}$	3	1	0	2	3	1	0	3	3	1	0	3
x	a b b	c		a a	b c c	a		a a a	b c b	b		c c c
time	1	2	3	4	5	6	7	8	9	10	11	12

Timed stream $x = \langle\langle a\ b\ b\rangle\ \langle c\rangle\ \langle\rangle\ \langle a\ a\rangle\ \langle b\ c\ c\rangle\ \langle a\rangle\ \langle\rangle\ \langle a\ a\ a\rangle\ \langle b\ c\ b\rangle\ \langle b\rangle\ \langle\rangle\ \langle c\ c\ c\rangle\ \ldots\rangle$

Time abstraction $\bar{x} = \langle a\ b\ b\ c\ a\ a\ b\ c\ c\ a\ a\ a\ a\ b\ c\ b\ b\ c\ c\ c\ \ldots\rangle$

Timing $\tilde{x} = \langle 3\ 1\ 0\ 2\ 3\ 1\ 0\ 3\ 3\ 1\ 0\ 3\ \ldots\rangle$

Elements at time $@x = \langle 1\ 1\ 1\ 2\ 4\ 4\ 5\ 5\ 5\ 6\ 8\ 8\ 8\ 9\ 9\ 9\ 10\ 12\ 12\ 12\ \ldots\rangle$

$\tilde{x}(t) = \#x(t)$ timing of x by the stream $\tilde{x}: \mathbb{N}_+ \rightarrow \mathbb{N}$

$n@x$ time of nth element in x

Timed Streams

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

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

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

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

$\#x$ number of elements in x

$M\#x$ number of elements in x that are in set M

$m\#x = \{m\}\#x$

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:

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

Finite timed histories

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

Stream histories

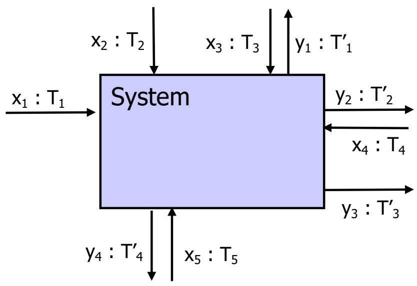
$\overline{X} = (X \rightarrow M^{*|\omega})$

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

Syntactic interfaces

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

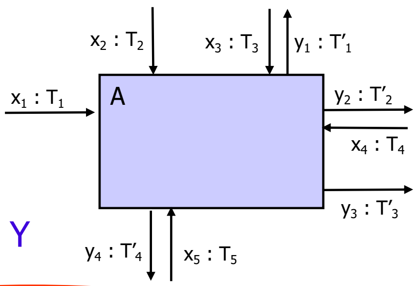
(X▸Y)



Interface specification predicates and assertions

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

Q = (X▸Y): A



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

MIX = (x, z: Tstr M▸ y: Tstr M): ∀ m ∈ M: m#x+m#z = m#y

MIX(x, z, y) = ∀ m ∈ M: m#x+m#z = m#y

predicate (let M be a nonempty set/type) assertion

We write Q::(X▸Y) to express that Q is an interface predicate for the syntactic interface (X▸Y)

Delay and time out

FOW = (y: Tstr M ▶ z: Tstr M): $\forall$ m $\in$ M: m#z = m#y

Delay: d $\in \mathbb{N}$: d ≥ 1

FOWD = (y: Tstr M ▶ z: Tstr M): $\forall$ m $\in$ M: m#z = m#y
$\wedge \forall$ t $\in \mathbb{N}$: m#(y↓t) ≥ m#(z↓t+d)

Time out: u $\in \mathbb{N}$: u ≥ 1

FOWTO = (y: Tstr M ▶ z: Tstr M): $\forall$ m $\in$ M: m#z = m#y
$\wedge \forall$ t $\in \mathbb{N}$: m#(z↓t+u) ≥ m#(y↓t))

Delay and time out:

FOWD = (y: Tstr M ▶ z: Tstr M): $\forall$ m $\in$ M: m#z = m#y
$\wedge \forall$ t $\in \mathbb{N}$: m#(z↓t+u) ≥ m#(y↓t) ≥ m#(z↓t+d)

Refinement of interface predicates

An interface predicate Q'::(X ▶ Y)
is called refinement of an interface predicate Q::(X ▶ Y) if

$$Q' \Rightarrow Q$$

Hiding

Hiding

Given a specification

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

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

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

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

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

Causality

Strongly Causal Interface Predicates

$Q::(X \blacktriangleright Y)$

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

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

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

Note: If $Q(x, y) = false$ for all $x \in \vec{X}$, $y \in \vec{Y}$ then Q is strongly causal

Example: Interface Specification: Strong Causality

TRA = (x: Tstr M ► y: Tstr M):

$\forall m \in M: (m\#x = 0 \Rightarrow m\#y = 0) \wedge (m\#x = \infty \Rightarrow m\#y > 0)$ nucleus

TRA(x, y)

$\Rightarrow$ logical reasoning

$m\#x = \infty \Rightarrow \exists t \in \mathbb{N}: m\#y{\downarrow}t > 0$

TRA©(x, y)

$\Rightarrow$ strong causality

$\forall t \in \mathbb{N}: m\#(x{\downarrow}t) = 0 \Rightarrow m\#(y{\downarrow}t+1) = 0$

$\Rightarrow$ logical reasoning

$m\#x = \infty \Rightarrow \exists t \in \mathbb{N}: m\#(x{\downarrow}t) > 0 \wedge m\#(y{\downarrow}t) = 0 \wedge m\#(y{\downarrow}t+1) > 0$

Specification nuclei

In a specification we may give just a nucleus

MIX = (x, z: Tstr M ► y: Tstr M): $\forall m \in M: m\#x+m\#z = m\#y$

This is an assertion that gives the key characteristic from which further properties are deduced in refinement steps typically be the step to adding strong causality –

going from MIX to MIX©.

Example: Interface Specification: Strong Causality

MIX = (x, z: Tstr M ▸ y: Tstr M): $\forall m \in M: m\#x+m\#z = m\#y$

nucleus

MIX©(x, y) = $\forall m \in M: m\#x+m\#z = m\#y$

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

FOW = (y: Tstr M ▸ x: Tstr M): $\forall m \in M: m\#x = m\#y$

nucleus

FOW©(x, y) = $\forall m \in M: m\#x = m\#y \wedge \forall t \in \mathbb{N}: m\#(y{\downarrow}t) \geq m\#(x{\downarrow}t+1)$

Input enabledness

If Q::(X ▸ Y) ≠ false is strongly causal then Q is input enabled

since there exists $z \in \vec{X}$ and $y \in \vec{Y}$ such that Q(x, y)
for all $x \in \vec{X}$

$x{\downarrow}0 = z{\downarrow}0 \wedge Q(z, y) \Rightarrow \exists y' \in \vec{Y}: Q(x, y') \wedge y{\downarrow}1 = y'{\downarrow}1$

Realizability

Strongly Causal Functions

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

is strongly causal if for $t \in \mathbb{N}$

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

Then we write SC[f]

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

Fully Realizable Predicates

$Q::(X \blacktriangleright Y)$

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

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® = false if Q is not realizable

Example: Interface Specification: Strong Realizability

INF = (x: Tstr M ▸ y: Tstr M): #x = ∞ ⇒ #y = 0

nucleus

INF is not strongly causal, realizable, but not fully realizable

INF®(x, y)

⇒

full realizability

#y = 0

Example: Interface Specification: Strong Realizability

REP = (x: Tstr M ▸ y: Tstr M):

(#x < ∞ ⇒ #y = 0) ∧ (#x = ∞ ⇒ #y = ∞)

nucleus

REP is strongly causal

REP®(x, y)

⇒

realizability

false

Example: Interface Specification: full realizability

NoID = ({x: Tstr M} ▶ {y: Tstr M}): x ≠ y

nucleus

NoID is strongly causal

strong causality

NoID®(x, y)

⇒

full realizability

false

A specification that is realizable, strongly causal but not fully realizable

DEMO = (x: Tstr M ▶ y: Tstr M): x ≠ y ∨ y = ε where ε is the stream with #ε = 0

DEMO is strongly causal

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

DEMO is realizable

f(x) = ε

but not fully realizable: ε is the only fixpoint of DEMO:

DEMO(ε, y) holds for all y with y ≠ ε

There is no realization f with f(ε) ≠ ε since f has a unique fixpoint z where DEMO(z, f(z)) and z = f(z). Since DEMO(z, z) implies z = ε we get f(ε) = ε

Assumptions and Commitments

Assumption/Commitment

How to deal with specifications, that are not realizable

Example: An interactive queue

Queue

in x: Str Data | {req}
out y: Str Data
Data©y ⊑ Data©x
#y = req#x

Here M©x is the sub-stream of x consisting of the elements in set M

x ⊑ y stands for stream x is prefix of stream y

Inconsistence with strong causality: not input enabled

However, if we require

$$\text{Data}©y \sqsubseteq \text{Data}©x$$

$$\#y = \text{req}\#x$$

then there exist input streams x such that there does not exist some output y such that Queue(x, y) - Example: $x = \langle \text{req} \rangle$

We define the assertion Asu(x) that has to hold for x:

$$\text{Asu}(x) = \forall\, z \in \text{Str Data} \mid \{\text{req}\}: z \sqsubseteq x \Rightarrow \text{req}\#z \leq \text{Data}\#z$$

$$\text{QueueAC}(x, y) = (\text{Asu}(x) \Rightarrow \text{Queue}(x, y))$$

Interfaces with assumptions

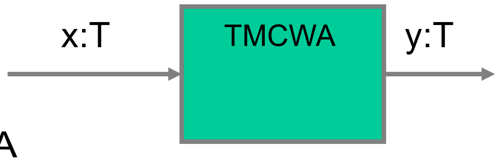
For the syntactic interface (X ▶ Y) we may include

- an assumption Asu(y, x) which is a specification of the *inverse* interface (Y ▶ X) and defines properties of the context
- a commitment Com(x, y) which is a specification of the behavior the syntactic interface (X ▶ Y) as long as the assumption is fulfilled.

this leads to the specification

$$\text{Asu}(y, x) \Rightarrow \text{Com}(x, y)$$

Example: System interface specification



A transmission component TMCWA

TMCWA

in x: Tstr M

out y: Tstr M

assume $\forall t \in \mathbb{N}: \#x{\downarrow}t \leq 1+\#y{\downarrow}t$

commit $\forall m \in M: m\#x = m\#y$

Operational Semantics: Moore machines

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*

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

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

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

the output b does not depend on the input a but only on the state σ.
Formally defined, there exists an output function:

$$\Xi: \Sigma \rightarrow \wp(\overline{Y}_{fin})\backslash\{\varnothing\}$$

such that

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

Moore machines compute interface behavior

We write $(\Delta, \Lambda)::(X \blacktriangleright Y)$ to express that (Δ, Λ) is the Moore machine that operates over the syntactic interface $(X \blacktriangleright Y)$.

$(\Delta, \Lambda)::(X \blacktriangleright Y)$ is called *deterministic* if the for all states $\sigma \in \Sigma$, histories $a \in \overline{X}_{fin}$ the sets Λ and $\Delta(\sigma, a)$ are one-element.

$(\Delta, \Lambda)::(X \blacktriangleright Y)$ *calculates* for an input history $x \in \vec{X}$ an output history $y \in \vec{Y}$, if there exist states $\sigma_0 \in \Lambda$ and $\sigma_t \in \Sigma$ for all $t \in \mathbb{N}$ and

$$(\sigma_{t+1}, y(t)) \in \Delta(\sigma_t, x(t))$$

Then the pair (x, y) of histories is called a *behavioral instance* of $(\Delta, \Lambda)::(X \blacktriangleright Y)$

States are considered as local, as hidden, while input and output is observable.

Moore machines compute infertace behavior

For each history $x \in \vec{X}$ a Moore machine $(\Delta, \Lambda)::(X \blacktriangleright Y)$ computes an interface predicate

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

defined by

$$[[\Delta, \Lambda]](x, y) = \exists\, \sigma \in (\mathbb{N} \to \Sigma): \sigma_0 \in \Lambda \wedge \forall\, t \in \mathbb{N}: (\sigma_{t+1}, y(t)) \in \Delta(\sigma_t, x(t))$$

$(\Delta', \Lambda')::(X \blacktriangleright Y)$ is called (extensional) *refinement* of Moore machine $(\Delta, \Lambda)::(X \blacktriangleright Y)$ if

$$[[\Delta', \Lambda']] \Rightarrow [[\Delta, \Lambda]]$$

Functional Moore machines

For every Moore machine $(\Delta, \Lambda)::(X \blacktriangleright Y)$ its associated interface predicate

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

is fully realizable and thus also strongly causal.

Every strongly causal function $f: \vec{X} \to \vec{Y}$ defines a deterministic Moore machine

$(\Delta_{(X \blacktriangleright Y)}, \{f\})::(X \blacktriangleright Y)$

where $\Sigma_{(X \blacktriangleright Y)}$ is the set of strongly causal functions in $\vec{X} \to \vec{Y}$ and

$\Delta_{(X \blacktriangleright Y)}: (\Sigma_{(X \blacktriangleright Y)} \times \overline{X}_{fin}) \to \wp(\Sigma_{(X \blacktriangleright Y)} \times \overline{Y}_{fin}) \backslash \{\emptyset\}$

is defined for histories $x \in \overline{X}$, $y \in \overline{Y}$ and strongly causal functions $f, f': \vec{X} \to \vec{Y}$ by

$\Delta(X \blacktriangleright Y)(f, a) = \{(f', b)\}$ where for all $x \in \vec{X}$: $f(\langle a \rangle\hat{\ }x) = \langle b \rangle\hat{\ }f'(x)$

Define for Moore machine $(\Delta, \Lambda)::(X \blacktriangleright Y)$ by $DET(\Delta, \Lambda)$ the set of deterministic Moore machines that are refinements of (Δ, Λ); then

$$[[\Delta, \Lambda]](x, y) = \exists (\Delta', \Lambda') \in DET(\Delta, \Lambda): [[\Delta', \Lambda']] (x, y)$$

For every Moore machine $(\Delta, \Lambda)::(X \blacktriangleright Y)$ its set of realizations $f: \vec{X} \to \vec{Y}$ of $[[\Delta, \Lambda]]$ is equal to the set of strongly causal functions

$$\{f': \vec{X} \to \vec{Y} : \exists (\Delta', \Lambda') \in DET(\Delta, \Lambda): \forall x: [[\Delta, \Lambda]](x, f(x))\}$$

For every Moore machine $(\Delta, \Lambda)::(X \blacktriangleright Y)$ its interface predicate $[[\Delta, \Lambda]]::(X \blacktriangleright Y)$ is the disjunction of the associated interface predicates of all its deterministic refinements.

Feature Interaction

Projection

Given a specification

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

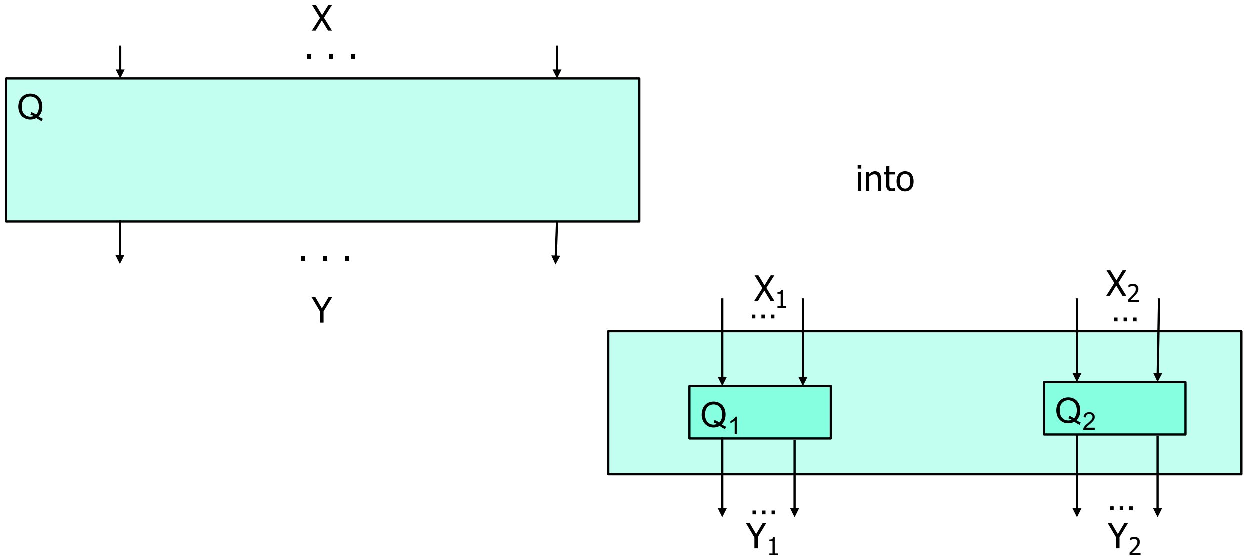
where $X' \subseteq X,\ Y' \subseteq Y$

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

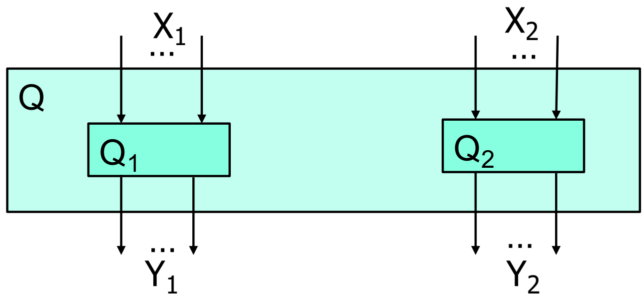
Can we decompose a system



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

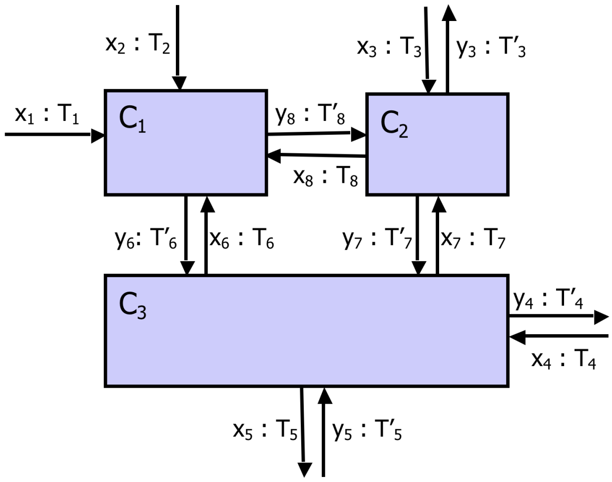
$$Q(x, y) = (Q_1(x|X_1, y|Y_1) \wedge Q_2(x|X_2, y|Y_2))$$



Distribution and Architecture Composition

Composition

We compose systems syntactically and semantically by their interfaces



Syntactic composability

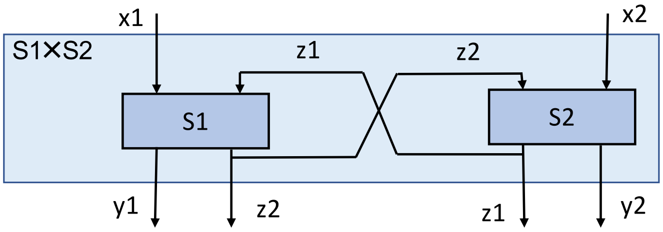
Specifications $S_k = (X_k \blacktriangleright Y_k):Q_k$ where $k = 1, 2$, are composable if

$$X_1 \cap X_2 = \emptyset$$
$$Y_1 \cap Y_2 = \emptyset$$

To make life simple we usually assume in addition:

$$X_1 \cap Y_1 = \emptyset$$
$$X_2 \cap Y_2 = \emptyset$$

Composition Diagrams



S1

in x1, z1: M
out y1, z2: M

Q1

S2

in x2, z2: M
out y2, z1: M

Q2

S1 × S2

in x1, x2: M
out y1, z2, y2, z1: M

$Q1 \wedge Q2$

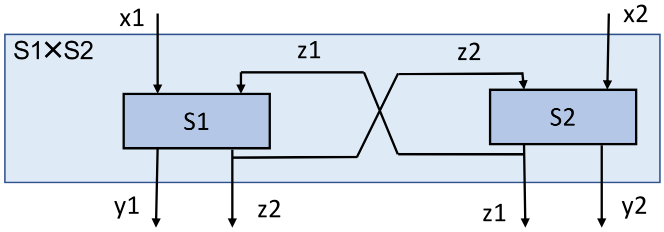
Example: Interface Specification: Strong Causality and Composition

MIX = (x, z: Tstr M ▶ y: Tstr M): $\forall$ m $\in$ M: m#x+m#z = m#y

FOW = (y: Tstr M ▶ z: Tstr M): $\forall$ m $\in$ M: m#z = m#y

(MIX(x, z, y) $\wedge$ FOW(y, z)) $\Rightarrow$ $\forall$ m $\in$ M: m#x+m#y = m#y

Composition Diagrams



S1

in x1, z1: M
out y1, z2: M

Q1

S2

in x2, z2: M
out y2, z1: M

Q2

S1 × S2

in x1, x2: M
out y1, z2, y2, z1: M

$Q1^{®} \wedge Q2^{®}$

Example: Interface Specification: Strong Causality and Composition

$$MIX^{®}(x, y) = \forall m \in M: m\#x+m\#z = m\#y \wedge \forall t \in \mathbb{N}: m\#(x\downarrow t)+m\#(z\downarrow t) \geq m\#(y\downarrow t+1)$$

$$FOW^{®}(y, z) = \forall m \in M: m\#z = m\#y \wedge \forall t \in \mathbb{N}: m\#(y\downarrow t) \geq m\#(z\downarrow t+1)$$

$$(MIX(x, z, y) \wedge FOW(y, z)) \Rightarrow \forall m \in M: m\#x+m\#y = m\#y$$

$$(MIX^{®}(x, z, y) \wedge FOW^{®}(y, z)) \Rightarrow \forall m \in M: m\#x+m\#y = m\#y \wedge \forall t \in \mathbb{N}: m\#(x\downarrow t)+m\#(y\downarrow t) \geq m\#(y\downarrow t+1)$$

$$\Rightarrow (m\#x = 0 \Rightarrow m\#y = 0)$$

Composition and Full Realizability

If two composable specifications S1 = (X1 ▸ Y1): Q1 and S2 = (X2 ▸ Y2): Q2

- are fully realizable
- then their composition S1 × S2 with assertion $Q1 \wedge Q2$ is fully realizable

If assertions W1 and W2 are weaker than fully realizable: $Q1 \Rightarrow W1$, $Q2 \Rightarrow W2$

Then $W1 \wedge W2$ is generally a weaker assertion (correct but not necessary complete)

$$(Q1 \wedge Q2) \Rightarrow (W1 \wedge W2)$$

Composing Moore machines

We compose Moore machines $(\Delta_k, \Lambda_k)::(X_k \blacktriangleright Y_k)$ for $k = 1, 2$, where $X_1 \cap X_2 = \varnothing$, $Y_1 \cap Y_2 = \varnothing$ by parallel composition to a Moore machine

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

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

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

where for

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

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

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

Composing Moore machines

For composable Moore machines $(\Delta_k, \Lambda_k)::(X_k \blacktriangleright Y_k)$ for $k = 1, 2,$ we get

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

For every fully realizable interface predicate $Q::(X \blacktriangleright Y)$ there exists a Moore machine such that

$$(\Delta, \Lambda)::(X \blacktriangleright Y) \text{ with } Q = [[(\Delta, \Lambda)]]$$

For a Moore machine $(\Delta, \Lambda)::(X \blacktriangleright Y)$ the interface predicate $[[(\Delta, \Lambda)]] ::(X \blacktriangleright Y)$ is

- fully realizable and
- the set of fully realizable interface predicates forms a denotational semantics for systems implemented by Moore machines.

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

Layered Architectures

Layers in Layered Architectures

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

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

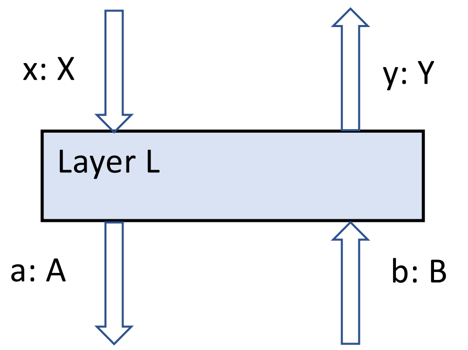
Let the interface behavior

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

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)

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

that fit syntactically together, if

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

and semantically if the provided service

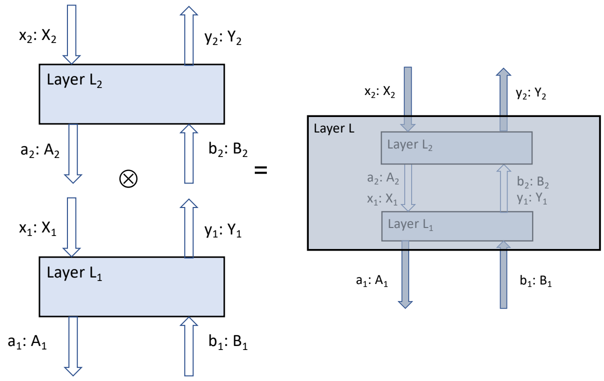
$S_1 = (x_1: \vec{X}_1 ► y_1: \vec{Y}_1): Q_1(x_1, y_1)$

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

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

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

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

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

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

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

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

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

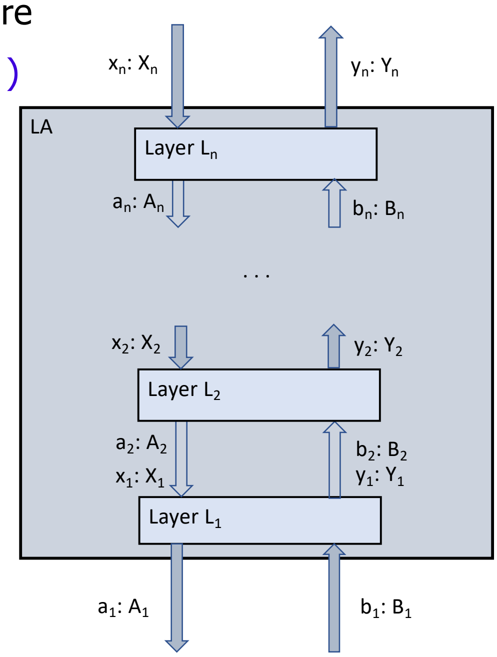
$S_k = (x_k: \vec{X}_k \blacktriangleright y_k: \vec{Y}_k): Q_k(x_k, y_k)$

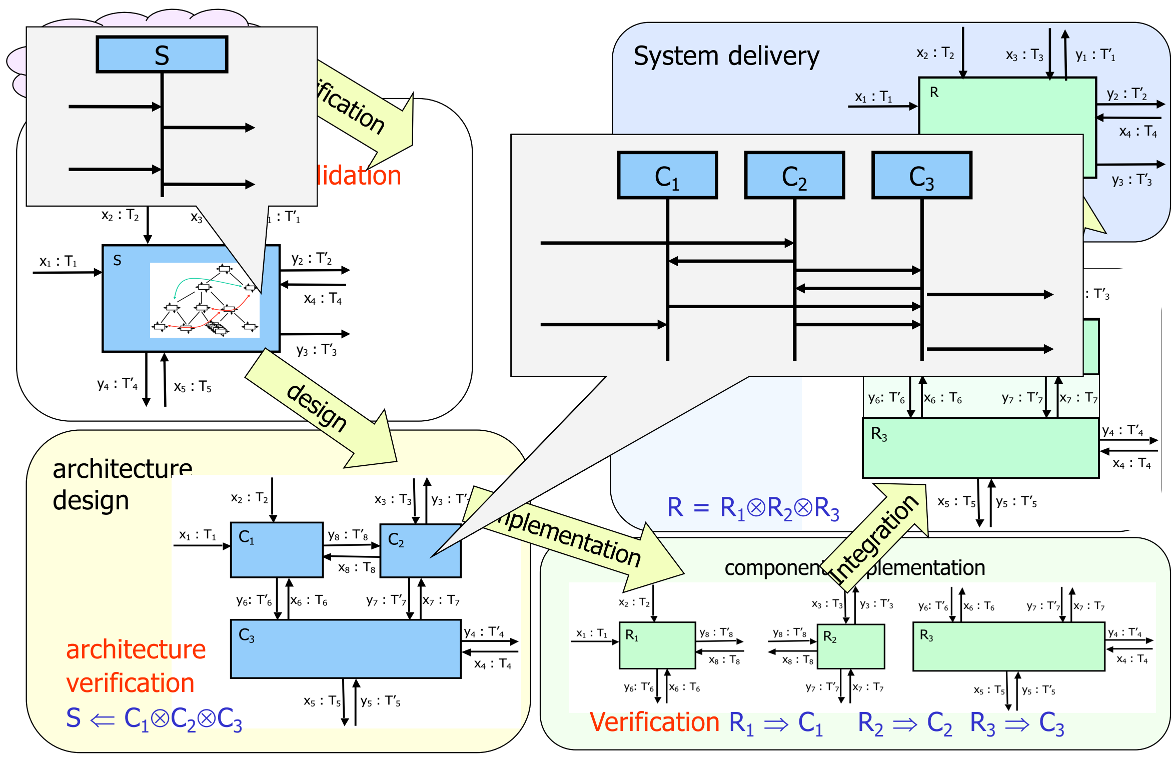
of lower layer L_k is a refinement of the requested service

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

of the upper layer L_2 which means

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





The Two Basic Models

State based models of concurrency

- Influenced by von Neumann architecture: shared state
- Interleaving concurrency
  - ◊ implicit
  - ◊ nondeterminism
  - ◊ deadlock
- State based assertion techniques
  - ◊ ghost variables,
  - ◊ stuttering
  - ◊ prophecy variables
- Composition
  - ◊ fairness
  - ◊ intensional

History based models of concurrency

- Data Flow
- Infinite computations
  - ◊ streams and histories
- Explicit Concurrency
- Safety and liveness
- Composition
  - ◊ compositionality
  - ◊ extensionality principle
- Distribution
- Abstraction: modularity
  - ◊ information hiding/encapsulation
- Components

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
- 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