Interaction and Realizability

Manfred Broy

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

Motivation

- A reference model for generalized computation
  - Interaction
  - Concurrency
  - Real time
- Properties of the model
- Computability

The model: functions/relations on streams

- Causality between input and output streams
- Realizability of single output histories for given input histories
- The role of non-realizable output in specific system contexts and for composition
- Relating non-realizable behaviors to state machines
- The concept of interactive computation and computability

Original motivation:

basis for model driven software & systems engineering

Types, Streams, Channels and Histories

- A type is a name for a set of data elements.
- Let TYPE be the set of all types.
- With each type $T \in$ TYPE we associate a set of data elements, the *carrier set* for T.
- We use the following notation:

M^* denotes the set of finite sequences over M including the *empty* sequence $\langle\rangle$,

M^∞ denotes the set of infinite sequences over M (that are represented by the total mappings $IN_+ \to M$ were $IN_+ = IN \setminus \{0\}$).

- By

$$M^\omega = M^* \cup M^\infty$$

we denote the set of streams of elements taken from the set M.

$\langle\rangle$ denotes the empty stream m.

- The set of streams has a rich algebraic and topological structure.

Concatenation

We introduce concatenation ^ as an operator

$$\_\hat{}\_ : M^{\omega} \times M^{\omega} \to M^{\omega}$$

- On finite streams x concatenation is defined as usual on finite sequences.
- We may see finite streams as partial functions $IN_{+} \to M$ and infinite streams as total functions $IN_{+} \to M$.
- For infinite streams r, s: $IN_{+} \to M$ we define:

$$s\hat{}x = s$$

$$s\hat{}r = s$$

$$\langle x_1 \dots x_n\rangle\hat{}\langle s_1 \dots \rangle = \langle x_1 \dots x_n \; s_1 \dots \rangle$$

- Streams are used to represent histories of communications of data messages transmitted within a time frame.
- Given a message set M of type T a *timed stream* is a function

$$s: IN_{+} \to M^{*}$$

- For each time t the sequence s(t) denotes the sequence of messages communicated at time t in the stream s.

Notation

$\langle\rangle$ empty sequence or empty stream,

$\langle m \rangle$ one-element sequence containing m as its only element,

x.t t-th element of the stream x,

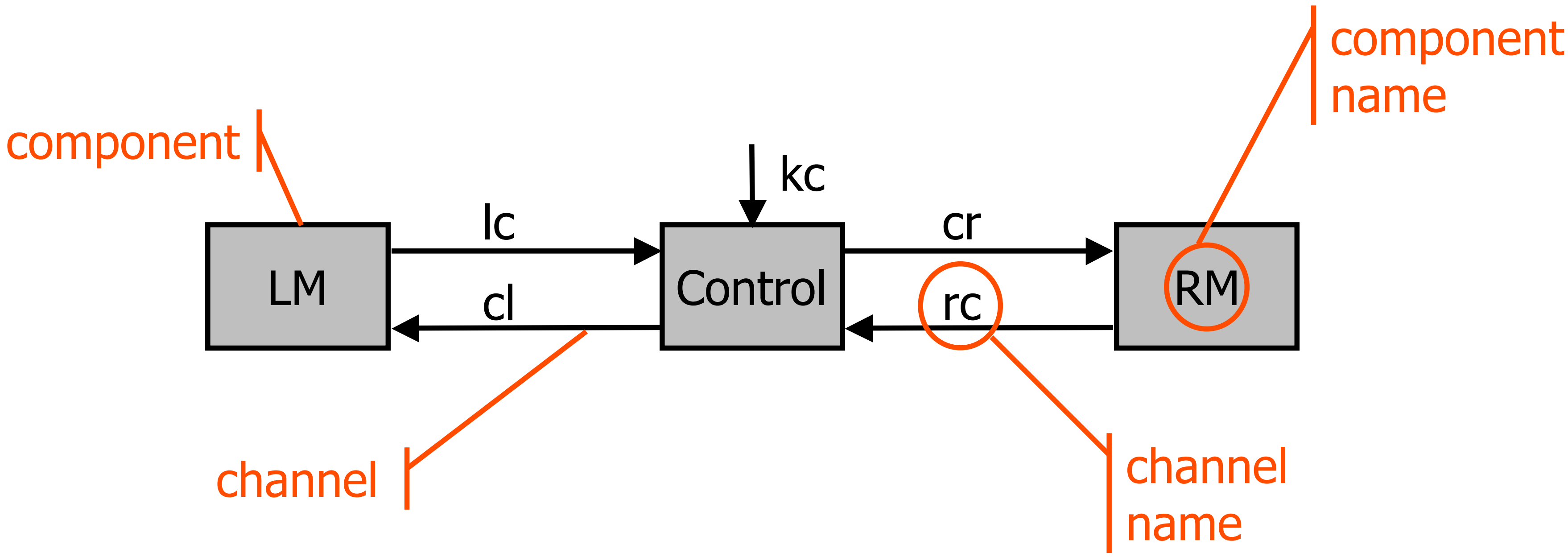
$x \downarrow t$ prefix of length t of the stream x,

#x number of elements in x

$\overline{x}$ finite or infinite stream that is the result of concatenating all sequences in the timed stream x. Note that $\overline{x}$ is finite if x carries only a finite number of nonempty sequences.

Basic system model

System class: distributed, reactive systems



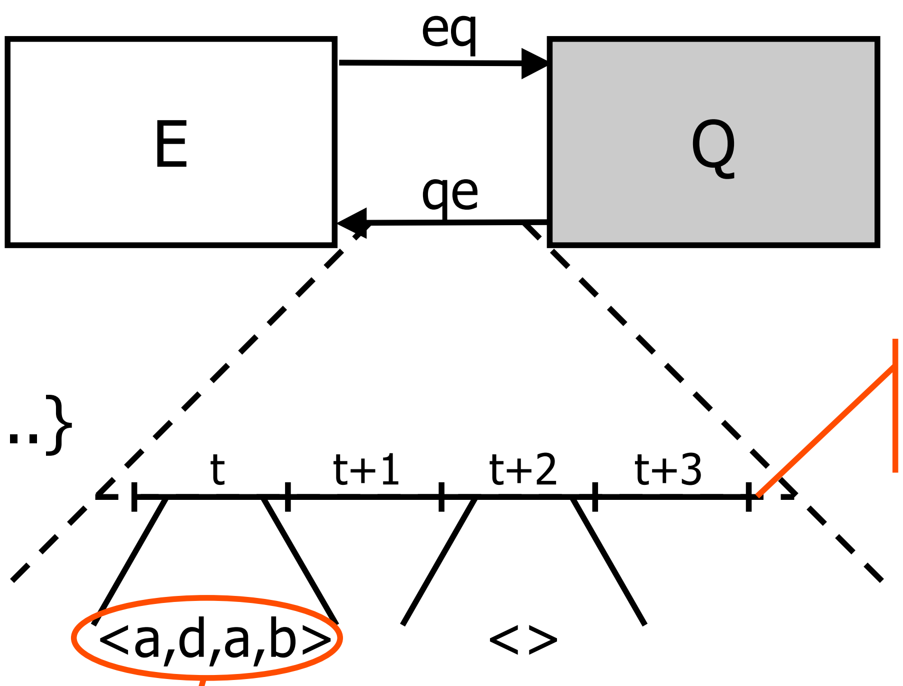
System consists of

- named components (with local state)
- named channels

driven by global, discrete clock

Basic Model

Timed Streams: Semantic Model for Black-Box-Behavior



Message set:

M = {a, b, c, ...}

infinite channel history

Messages
transmitted at time t

The Basic Behaviour Model: Streams and Functions

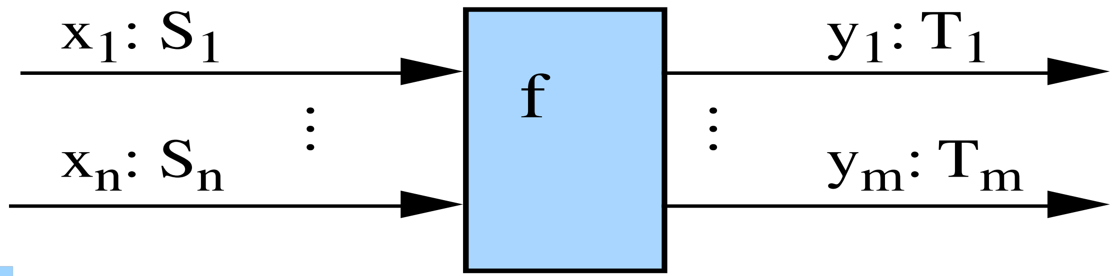
C	set of channels
Type: $C \rightarrow TYPE$	type assignment
$x : C \rightarrow (\mathbb{N} \setminus \{0\} \rightarrow M^*)$	channel history for messages of type M
$\vec{C}$	set of channel histories for channels in C

Channel: Identifier of Type stream

$I = \{ x_1 , x_2 , \dots \}$ set of typed input channels

$O = \{ y_1 , y_2 , \dots \}$ set of typed output channels

Interface behavior



$$f : \vec{I} \to \vec{O}$$

Set of interfaces: DIF[I ▶ O]

Causality

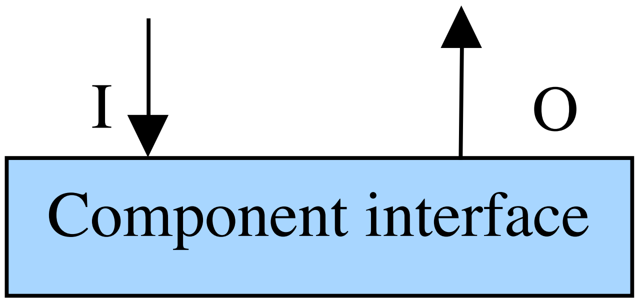
(I ▶ O) *syntactic interface* with set of input channels I and of output channels O

$f: \vec{I} \rightarrow \vec{O}$ *semantic interface* for (I ▶ O) with *timing property addressing causality* (let $x, z \in \vec{I}$, $y \in \vec{O}$, $t \in \mathbb{N}$):

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

$x \downarrow t$ prefix of history x with t finite sequences

Functions with this property are called strongly causal.



Consequences of Causality I

Causal functions have unique fixpoints:

Given a function:

$$f : \vec{I} \rightarrow \vec{I}$$

then there exists a unique fixpoint $y \in \vec{I}$ with

$$f(y) = y$$

Consequences of Causality II

Time abstractions are prefix monotonic

Given a function:

$$f : \vec{I} \to \vec{O}$$

a function

$$\bar{f} : (I \to M^{(\omega)}) \to (O \to M^{(\omega)})$$

is called time abstraction of f if for all $x \in \vec{I}$ we have

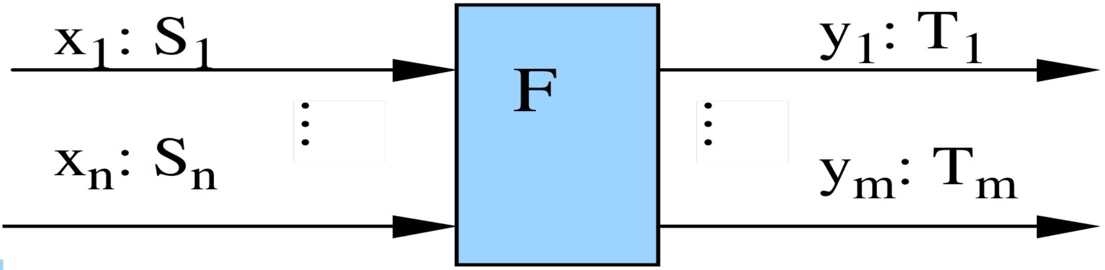
$$\bar{f}(\bar{x}) = \overline{f(x)}$$

If f is causal then $\bar{f}$ (if it exists) is prefix monotonic

General (Nondeterministic) Interface model

Channel: Identifier of Type stream

$I = \{ x_1 , x_2 , ... \}$ set of typed input channels
$O = \{ y_1 , y_2 , ... \}$ set of typed output channels



Interface behavior

$$F : \vec{I} \rightarrow \wp(\vec{O})$$

Set of interfaces: IF[I ▶ O]

Causality

$(I \blacktriangleright O)$ — *syntactic interface* with set of input channels I and of output channels O

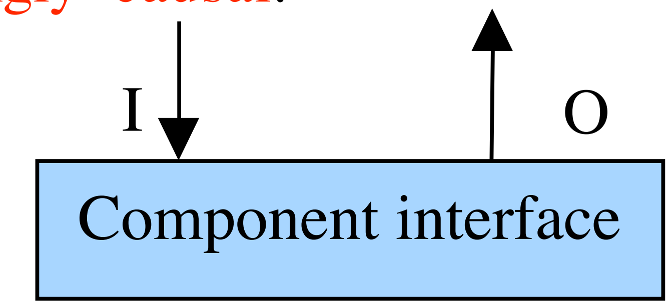
$F : \vec{I} \to \wp(\vec{O})$ — *semantic interface* for $(I \blacktriangleright O)$ with *timing property addressing causality* (let $x, z \in \vec{I}$, $y \in \vec{O}$, $t \in \mathbb{N}$):

$$\boxed{x{\downarrow}t = z{\downarrow}t \Rightarrow \{y{\downarrow}t{+}1 : y \in F(x)\} = \{y{\downarrow}t{+}1 : y \in F(z)\}}$$

$x \downarrow t$ — prefix of history x with t finite sequences

Functions with this propertiy are called strongly causal.

A component is a total behavior



Consequences of Causality

Either all result sets are empty or none

Given a function:

$$F : \vec{I} \rightarrow \wp(\vec{O})$$

then if

$$F(z) = \emptyset$$

for some $z \in \vec{I}$, then

$$F(x) = \emptyset$$

for all $x \in \vec{I}$.

Then F is called paradoxical.

Definition: Realizability

An I/O-behavior F is called *realizable*,
if there exists a strongly causal total function

$$f: \vec{I} \rightarrow \vec{O}$$

such that we have:

$$\forall\, x \in \vec{I}: f(x) \in F(x)$$

f is called a *realization* of F.

By $[\![F]\!]$ we denote the set of all realizations of F.

Definition: Realizability

An output history $y \in F(x)$ is called *realizable* for an interactive I/O-behavior F with input x, if there exists a realization $f \in [\![F]\!]$ with

$$y = f(x).$$

Observation

- There are causal non-paradox behaviours that are not realizable

$$F(x) = \{y: x \neq y\}$$

Proof: Assume F is realizable!

Then there exist f with $f(x) \in F(x)$ for all x.

F has a fixpoint y with $y = f(y)$.

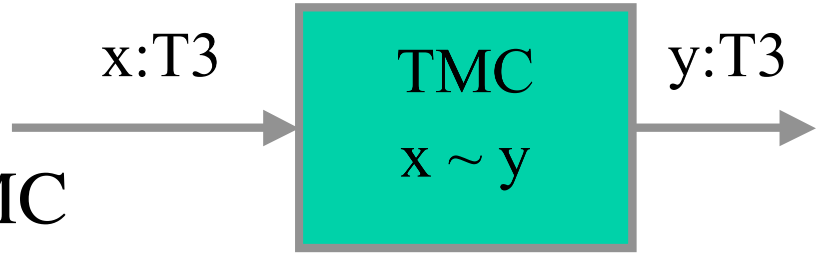
We get

$y = f(y) \in F(x)$

But by the definition of F:

$y \notin F(x)$

Example: Component interface specification



Spec name

A transmission component TMC

Input channel

TMC

in x: T3

out y: T3

$x \sim y$

Output channel

$$x \sim y \equiv (\forall\, m \in T3: \{m\}©\bar{x} = \{m\}©\bar{y})$$

Specifying assertion

State Machines with Input and Output

A state machine (Δ, Λ) with input and output according to

- the set I of input channels and
- the set O of output channels

is given by

- a state space Σ, which represents a set of states,
- a set $\Lambda \subseteq \Sigma$ of initial states as well as
- a state transition function

$$\Delta: (\Sigma \times (I \rightarrow M^*)) \rightarrow \wp(\Sigma \times (O \rightarrow M^*))$$

For each

- state $\sigma \in \Sigma$ and
- each valuation α: $I \rightarrow$ M* of the input channels in I by sequences of messages we obtain by every pair

$$(\sigma', \beta) \in \Delta(\sigma, \alpha)$$

  a successor state σ' and a valuation β: O $\rightarrow$ M* of the output channels consisting of the sequences produced by the state transition.

Such state machines are also called *Mealy machines*.

Classes of state machines

- A state machine (Δ, Λ) is called
- *deterministic*, if, for all states $\sigma \in \Sigma$ and input α, $\Delta(\sigma, \alpha)$ and Λ are sets with at most one element.
- *total*, if for all states $\sigma \in \Sigma$ and all inputs α the sets $\Delta(\sigma, \alpha)$ and Λ are not empty; otherwise the machine (Δ, Λ) is called *partial*,
- a (generalized) *Moore machine*, if its output depends only on the state and not on the actual input of the machine. Then the following equation holds for all input sequences α, α' and output sequences β, and all states σ:

$$\exists\, \sigma' \in \Sigma: (\sigma', \beta) \in \Delta(\sigma, \alpha) \Leftrightarrow \exists\, \sigma' \in \Sigma: (\sigma', \beta) \in \Delta(\sigma, \alpha')$$

Moore Machines

- A way to characterize a Moore machine is to require functions

$$\text{out}: \Sigma \to \wp(O \to M^*)$$

$$\text{next}: \Sigma \times (I \to M^*) \times (O \to M^*) \to \wp(\Sigma)$$

  such that

$$\Delta(\sigma, \alpha) = \{(\sigma', \beta): \beta \in \text{out}(\sigma) \wedge \sigma' \in \text{next}(\sigma, \alpha, \beta)\}$$

- Subtle point: the choice of the output β does not depend on the input α, but the choice of the successor state σ' may depend both on the input α and on the choice of the output β. We therefore require that for each $\beta \in \text{out}(\sigma)$ there exists a successor state:

$$\forall\, \beta \in \text{out}(\sigma): \exists\, \sigma' \in \Sigma: \sigma' \in \text{next}(\sigma, \alpha, \beta)$$

- By SM[I → O] we denote the set of all total Moore machines with input channels I and output channels O. By DSM[I → O] we denote the set of deterministic total Moore machines.

Computations of State Machines

$$\sigma_0 \xrightarrow{x_1/y_1} \sigma_1 \xrightarrow{x_2\ /y_2} \sigma_2 \longrightarrow \dots$$

- a stream x of input : x_1 , x_2, …
- a stream y of output : y_1 , y_2, …
- a stream s of states : σ_0 , σ_1, …
- The computation is generated given the input stream x and the initial state σ_0 by choosing step by step

$$(\sigma_{i+1}, y_{i+1}) \in \Delta(\sigma_i, x_{i+1})$$

Computations

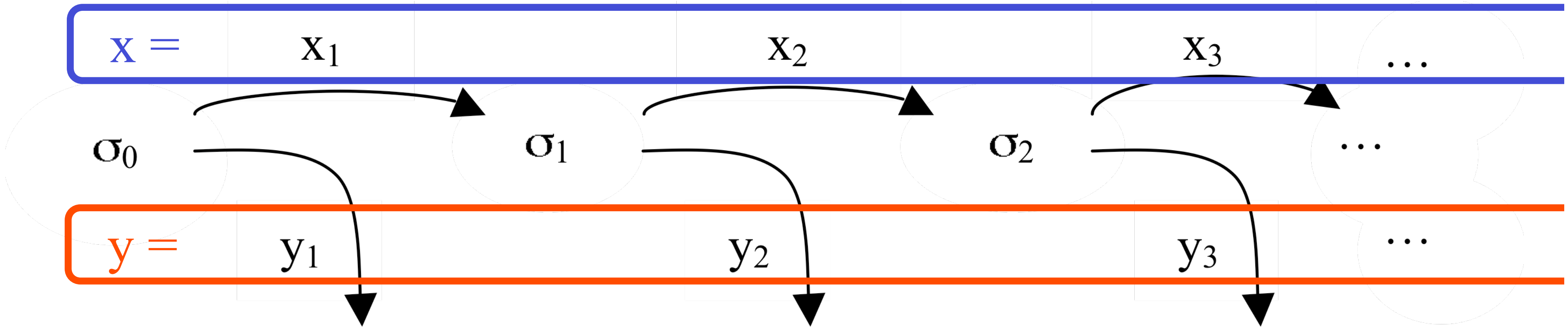
- A computation for a state machine (Δ, Λ) and an input history x is given by a sequence of states

$$\{\sigma_t : t \in \mathrm{IN}\}$$

  and an output history y such that for all times $t \in \mathrm{IN}$:

$$(\sigma_{t+1}, y.t+1) \in \Delta(\sigma_t, x.t+1) \text{ and } \sigma_0 \in \Lambda$$

- The history y is called an *output* of the computation of the state machine (Δ, Λ) for input x and initial state σ_0.
- The machine computes the output history y for the input history x and initial state σ_0.



Refinement and Equivalence of State Machines

- Two state machines are called *(observably) equivalent* if for each input history their sets of output histories coincide.
- A state machine is called *equivalent to a behavior* F, if for each input history x the state machine computes exactly the output histories in the set F(x).
- A state machine (Δ_2, Λ_2) with transition function

$$\Delta_2: (\Sigma_2 \times (I \rightarrow M^*)) \rightarrow \wp(\Sigma_2 \times (O \rightarrow M^*))$$

  is called a *transition refinement* or a *simulation* of a state machine (Δ_1, Λ_1) with the transition function

$$\Delta_1: (\Sigma_1 \times (I \rightarrow M^*)) \rightarrow \wp(\Sigma_1 \times (O \rightarrow M^*))$$

  if there is a mapping $\rho: \Sigma_2 \rightarrow \Sigma_1$ such that for all states $\sigma \in \Sigma_2$, and all input $\alpha \in I \rightarrow M^*$ we have:

$$\{(\rho(\sigma'), \beta): (\sigma', \beta) \in \Delta_2(\sigma, \alpha)\} \subseteq \Delta_1(\rho(\sigma), \alpha), \qquad \{\rho(\sigma): \sigma \in \Lambda_2\} \subseteq \Lambda_1$$

- A special case is given if ρ is the identity; then the equation simplifies to:

$$\Delta_2(\sigma, \alpha) \subseteq \Delta_1(\sigma, \alpha) \wedge \Lambda_2 \subseteq \Lambda_1$$

Nondeterministic versus deterministic computations

Theorem:

Every computation of a total non-deterministic Moore machine is also a computation of a total deterministic Moore machine.

Refinements

- To capture refinements we need the more general state space with

$$\Sigma_2 = (\Sigma_1 \times \text{IN})$$

  where for the Moore machine (Δ_2, Λ) we only require:

$$\Delta_2((\sigma, t), \alpha) = \{((\sigma', t+1), \beta)\} \quad \text{where} \quad (\sigma', \beta) \in \Delta_1(\sigma, \alpha)$$

- Each such state machine $(\Delta_{2,} \{(\sigma_0 , 0)\})$ with $\sigma_0 \in \Lambda_1$ is called a *deterministic enhanced refinement* of state machine (Δ_1, Λ_1).
- Consider two state machines:

$$\Delta_1, \Delta_2 : \Sigma \times (I \rightarrow M^*) \rightarrow \wp(\Sigma \times (O \rightarrow M^*))$$

  where Δ_1 produces arbitrary output and arbitrary successor states. Δ_1 is trivially a Moore machine.
- Every machine Δ_2 in DSM[I → O] is a refinement of Δ_1. In fact, every Mealy machine Δ_2 is a refinement of Δ_1, too.
- To make sure that we obtain Moore machines in the construction above we have to strengthen the formula slightly as follows:

$$\forall \alpha : \Delta_2((\sigma, t), \alpha) = \{((\sigma', t+1), \beta)\} \quad \text{where} \quad (\sigma', \beta) \in \Delta_1(\sigma, \alpha)$$

- Since β does not depend on α in the original machine, this formula can be fulfilled for each output β.

Combination of State Machines

- We can also combine sets of state machines into one state machine. Let a set of state machines be given (where K is an arbitrary set of names for state machines)

$$\{(\Delta_k, \Lambda_k):\ k \in K\} \text{ with } \Delta_k: (\Sigma_k \times (I \to M^*)) \to \wp(\Sigma_k \times (O \to M^*))$$

We define the composed state machine

$$(\Delta, \Lambda) = \coprod_{k \in K} (\Delta_k, \Lambda_k)$$

as follows (let w.l.o.g. all state spaces Σ_k for the machines (Δ_k, Λ_k), with $k \in K$ be pairwise disjoint):

$$\Lambda = \bigcup_{k \in K} \Lambda_k$$

$$\Delta(\sigma, \alpha) = \Delta_k(\sigma, \alpha) \text{ for } \sigma \in \Sigma_k$$

Sets of deterministic machines

Theorem:

Every total Moore machine is equivalent to (a state machine composed of) a set of deterministic Moore machines.

State machines define behaviors

Theorem:

Every total deterministic Moore machine (Δ, Λ) with the transition function

$$\Delta: (\Sigma \times (I \rightarrow M^*)) \rightarrow \wp(\Sigma \times (O \rightarrow M^*))$$

defines a deterministic behavior

$$F_\sigma^\Delta : \vec{I} \rightarrow \wp(\vec{O})$$

for every state $\sigma \in \Sigma$ where for each input x the output of the state machine (Δ, Λ) is the history y where F_σ^Δ(x) = {y}.

The function F_σ^Δ is strongly causal.

State machines define realizable behaviours

We define an operator along the lines of the proof of the theorem above

$$\Psi: \mathrm{DSM}[I \blacktriangleright O] \rightarrow (\vec{I} \rightarrow \wp(\vec{O}))$$

that maps every total deterministic Moore machine onto its interface abstraction

$$\Psi((\Delta, \{\sigma 0\})) = F^{\Delta}_{\sigma 0}$$

Corollary:

Every total Moore machine can be represented by a fully realizable behavior.

Behaviours can be represented by state machines

Theorem:
Every deterministic behavior

$$F: \quad \vec{I} \to \wp(\vec{O})$$

defines a total dete rministic Moore machine (Δ, Λ) with a transition function

$$\Delta: (\Sigma \times (I \to M^*)) \to \wp(\Sigma \times (O \to M^*)).$$

Corollary:
Every fully realizable interactive behavior can be represented by a total Moore machine.

Interactive Computations

A computation is carried out for each time interval $t \in \mathrm{IN}$ in two steps:

(1) The input x.t is provided to the system,

(2) The output y.t+1 is selected. It must and can depend only on the initial state, the input till time interval t and the output, which is produced so far, till time interval t.

- To model interactive computations we assume for each initial state of the considered system a function:

$$g: \{z: I \cup O \rightarrow (M^*)^t: t \in \mathrm{IN}\} \times (I \rightarrow M^*) \rightarrow \wp(O \rightarrow M^*)$$

such that for given input history x we define the output history y and the state of the computation z inductively, where

$$z : \mathrm{IN} \rightarrow (I \cup O \rightarrow (M^*)^t)$$

$$y.t{+}1 \in g(z.t) \text{ where } (z.t)\big|I = x{\downarrow}t \text{ and } (z.t)\big|O = y{\downarrow}t$$

- The function g is called an *interactive computation strategy*.
- By Out(g)(x) we denote the set of output histories y that can be constructed by the computation strategy g in this way.

Correct strategies

- The computation strategy g is called *correct* for the interactive behavior F if $Out(g)(x) \subseteq F(x)$;
- it is called *deterministic* if $Out(g)(x)$ is always a one element set.

Interactive computations

For interactive computations

the following observations about strategies g hold:

- As long as g(z) is never empty, Out(g) is never empty.
- Each strategy can be refined into a set of deterministic strategies G where
  - ◇ for each $g' \in G$ we require that $g'(z) \in g(z)$ holds and $g'(x)$ contains exactly one element,
  - ◇ a deterministic strategy is equivalent to a deterministic behavior,
  - ◇ we get $g(x) = \{g'(x): g' \in G\}$.

Game theoretic view

- Assume we say that there is a winning strategy for the partial computation

$$z: I \cup O \rightarrow (M^*)^t$$

for some time $t \in \mathrm{IN}$ (which represents a partially played game) if there is a strategy g with $g(x) \in F(x)$ for all input histories x with $z|I = x{\downarrow}t$ that finds some $y \in F(x)$ such that $z|O = y{\downarrow}t$, where $g(x) = \{y\}$.

- If for a partial computation z every $y \in F(x)$ with $z|O = y$ is not realizable then there does not exist a winning strategy.

Realizability

We study the situation where a behavior F is not fully realizable:

- This means that there is some input x and some output $y \in F(x)$ such that
  - there does not exist a strongly causal total function f such that $\forall\, x \in : f(x) \in F(x)$ and $y = f(x)$.
- We show that then there is a time $t \in IN$ such that the partial computation z with $z|I = x{\downarrow}t$ and $z|O = y{\downarrow}t$ is a dead end.

Infinite Tree of Partial Computations

- We call winning states (states for which a winning strategy exists) "white" nodes and loosing states (states for which a winning strategy does not exist) "black" nodes.
- Each node is characterized by a pair of evaluations for the channels

$$(a, b) \in (I \to (M^*)^t, O \to (M^*)^t).$$

- An interactive computation step is the transition of a state (a, b) to a new state (a′, b′) where there exists

$$\text{input } \alpha: I \to M^*$$

$$\text{output } \beta : O \to M^*$$

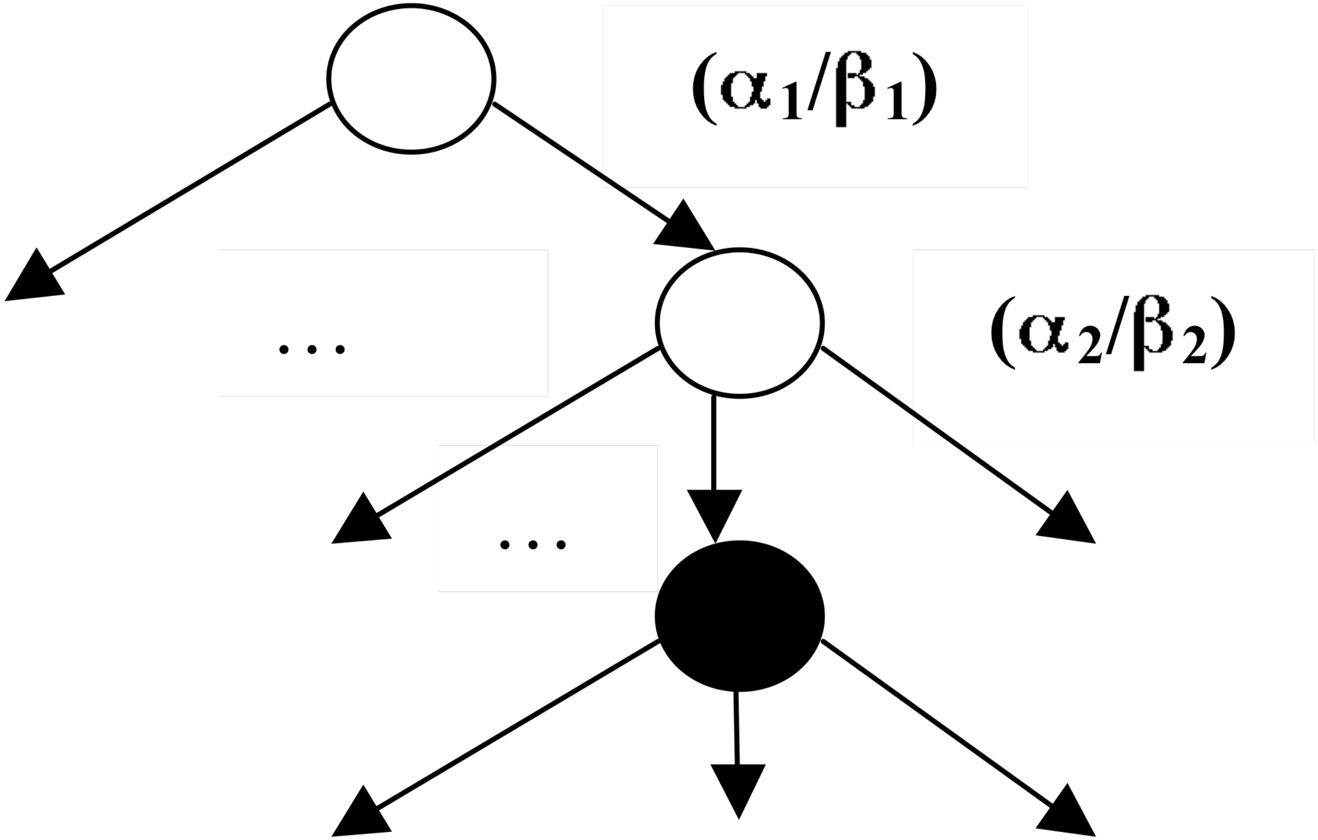
such that

$$a' = a\hat{\ }\langle\alpha\rangle$$

$$b' = b\hat{\ }\langle\beta\rangle$$

Infinite Tree of Partial Computations

Black Nodes Denoting Loosing States



Correctness of computation trees

- A step is called *correct*, if (a', b') is again a partial computation, i.e. if there exist histories x and $y \in F(x)$ with

$$x{\downarrow}(t+1) = a' \wedge y{\downarrow}(t+1) = b'$$

- For each behavior F we obtain a tree of partial computations.

Colouring nodes

- A node in the tree is white if and only if for every input α: I $\rightarrow$ M* there exists some output β: O $\rightarrow$ M* such that there is an arc that is labeled by α/β and leads to a white node.
- A node is black if and only
  - ◇ if there exists some input α: I $\rightarrow$ M* such that
  - ◇ for each feasible output β: O $\rightarrow$ M* the arcs labeled by α/β lead to black nodes.
- A behavior is realizable, if the root of its computation tree is white.
- It is fully realizable if its computation tree contains only white nodes.

Computation paths

- For each input history x and each output history $y \in F(x)$ we obtain a path in the computation tree. We get:
  (1) The history $y \in F(x)$ is realizable for the input x if and only if its corresponding computation path is colored by white nodes only.
  (2) The history $y \in F(x)$ is not realizable if and only if the is at least one node on its path in the computation tree that is black.
  (3) For a not realizable history $y \in F(x)$ there is a least partial computation (a, b) with $a = x{\downarrow}t$ and $b = y{\downarrow}t$ such that its node is black and all nodes $(a{\downarrow}t', b{\downarrow}t')$ with $t' < t$ are white.
  This means that all output histories $y' \in F(x')$ with
  $$y'{\downarrow}t = y{\downarrow}t \qquad \wedge \qquad x'{\downarrow}t = x{\downarrow}t$$
  are not realizable since there is a black node on their computation paths.
  (4) Due to strong causality, if $y \in F(x)$ and y is not realizable there exists a time t such that all input histories x' with $x{\downarrow}t = x'{\downarrow}t$ contain not realizable output histories in $F(x)$.

Interactive Computability

We call the state machine *computable* if the state transition function is Turing computable. We call a deterministic behavior *computable* if its state machine representation is computable.

Theorem:

If a behavior is computable it is realizable.

Computability of Interactive Behaviors

- For simplicity we consider only functions over untimed streams. The generalization to tuples of streams is quite straightforward. We consider functions on streams

$$f: IN^{\omega} \rightarrow IN^{\omega}$$

- We call the stream function f *computable* iff there exists a computable function:

$$f^*: IN^* \times IN \rightarrow IN$$

such that for all sequences $x \in IN^*$, $t \in IN$:

$f^*(x, t) = f(x).t$ iff $\#f(x) \geq t$ and $f^*(x, t)$ undefined otherwise

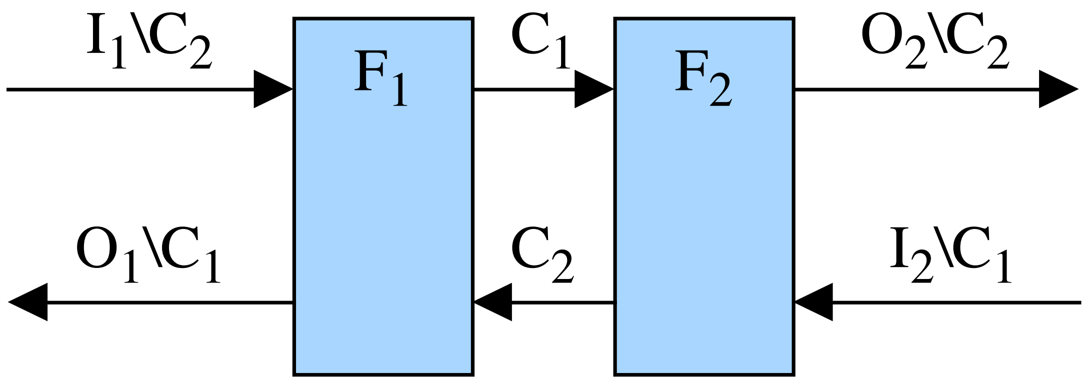
and all $x \in IN^{\infty}$, $t \in IN$ there exists $x' \in IN^*$ such that

$$f(x).t = f^*(x, t)$$

- Note that the second condition is actually with what is called continuity in fixpoint theory.

Composition of Components and Services

$F_1 \in IF[I_1 \blacktriangleright O_1]$
$F_2 \in IF[I_2 \blacktriangleright O_2]$



$C_1 = O_1 \cap I_2$
$C_2 = O_2 \cap I_1$
$I = I_1\backslash C_2 \cup I_2\backslash C_1$
$O = O_1\backslash C_1 \cup O_2\backslash C_2$

$F_1 \otimes F_2 \in IF[I \blacktriangleright O]$,

$(F_1 \otimes F_2).x = \{z|O: x = z|I \wedge z|O_1 \in F_1(z|I_1) \wedge z|O_2 \in F_2(z|I_2)\}$

Composition of specifications

Input channels

Output channels

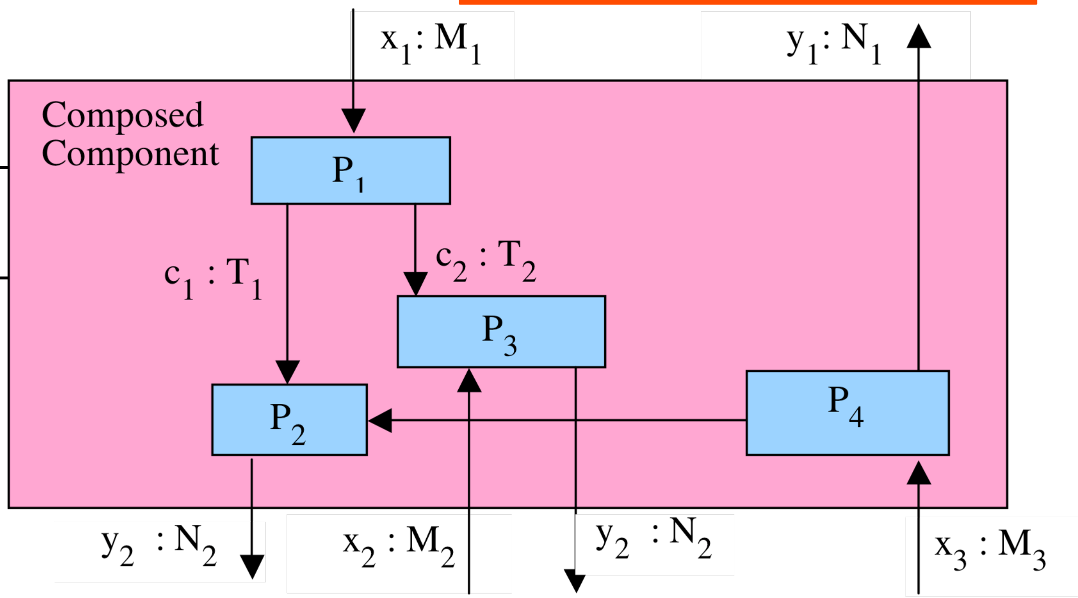
Internal channels

Composed components spec

in $x_1: M_1, x_2: M_2, \ldots$

out $y_1: N_1, y_2: N_2, \ldots$

$\exists c_1, c_2, \ldots : P_1 \wedge \ldots \wedge P_n$



System composition = logical and

Channel Hiding = existential quantification

Concluding Remarks

- Realizability is a notion that only arises in the context of specifications of interactive computations. It is a fundamental issue when asking whether a behavior corresponds to a computation.
- We choose to work out the theory of realizability in terms of Moore machines because they are a more intuitive model of interactive computation. The realizability is not a problem of Moore machines only but applies as well to Mealy machines.
- The bottom line of our investigation is that state machines with input and output, in particular, generalized Moore machines, are an appropriate concept to model interactive computations.
- Moore machines, in particular, take care of a delay between input and output.
- Realizable functions are the abstractions of state machines, such as partial functions are the abstractions of Turing machines. They extend the idea of computability as developed for non-interactive computations to interactive computations.