Towards a Theory of Architectural Contracts: **Schemes and Patterns of Assumption/Promise Based System Specification**

Manfred Broy

Technische Universität München
Institut für Informatik
D-85748 Garching, Germany

Topics

- Discrete Systems
- Discrete System modelling theory
  - Discrete System
    - Interface
    - Logical specification
  - Architectures
    - Composition
    - Compositional reasoning
- Contracts
  - Assumption/Promise
  - Logical interpretation
  - Safety and Liveness
- Architectures
  - Design by assumption/promise
- Generalizations

Part I
System Modelling

Motivation & Foundations

Technische Universität München
Institut für Informatik
D-85748 Garching, Germany

What is a (discrete) system?

A system

- has a scope (a boundary)
- a behaviour
  - ◇ black box view: interface
    - syntactic interface: defines the discrete events at the system boundary by input and output via ports, channels, messages (events, signals)
    - dynamic interface, interface behaviour: the processes of interaction in terms of discrete events at the system boundary
  - ◇ glass/white box view: an internal structure (state and/or distribution into sub-systems)
    - architecture in terms of sets of sub-systems and their relationships (communication connections)
    - state space

    and a behaviour
    - state transition relation with input and output
    - interactions between components
- properties
  - ◇ quality profile (performance, ...)
  - ◇ ...

The role of modelling in software & systems engineering (S&SE)

Software & systems engineering means

- capturing requirements
  - domain specific
  - functional, logical, technical, methodological
- specification of the system’s overall functionality
- design of a solution in terms of
  - architecture
  - specifying components
- implementing components
- verifying components and
- integrating them into systems and verifying the integration
- verification of system
- further evolution

These are complex error prone tasks!

The role of modelling in software & systems engineering

Modelling helps for:

- expressing and documenting the requirements
- specifying the system
- describing the architecture
  - specifying the components
  - their composition and interaction
- modelling the components
- verifying of the components and
- integrating them into the system and verifying the system

On models and modelling

What is a model?

- An abstraction!

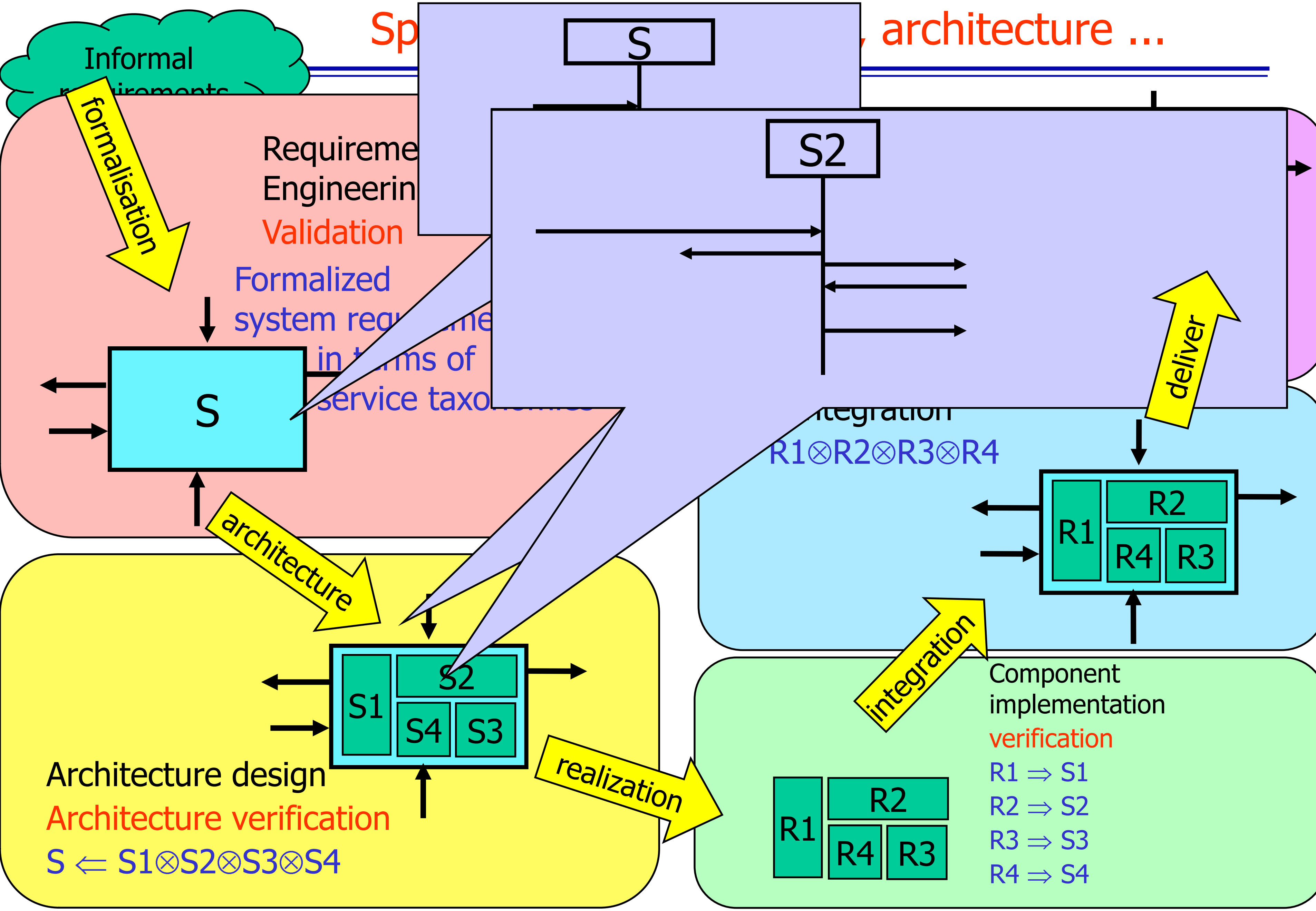
Which representations for models?

- Informal: language, informal diagrams, ...
- Semiformal: formalized graphical or textual presentation languages
- Mathematical: in terms of mathematical theories
- Formal models: formalized syntax, semantics and logics

What do we use models for?

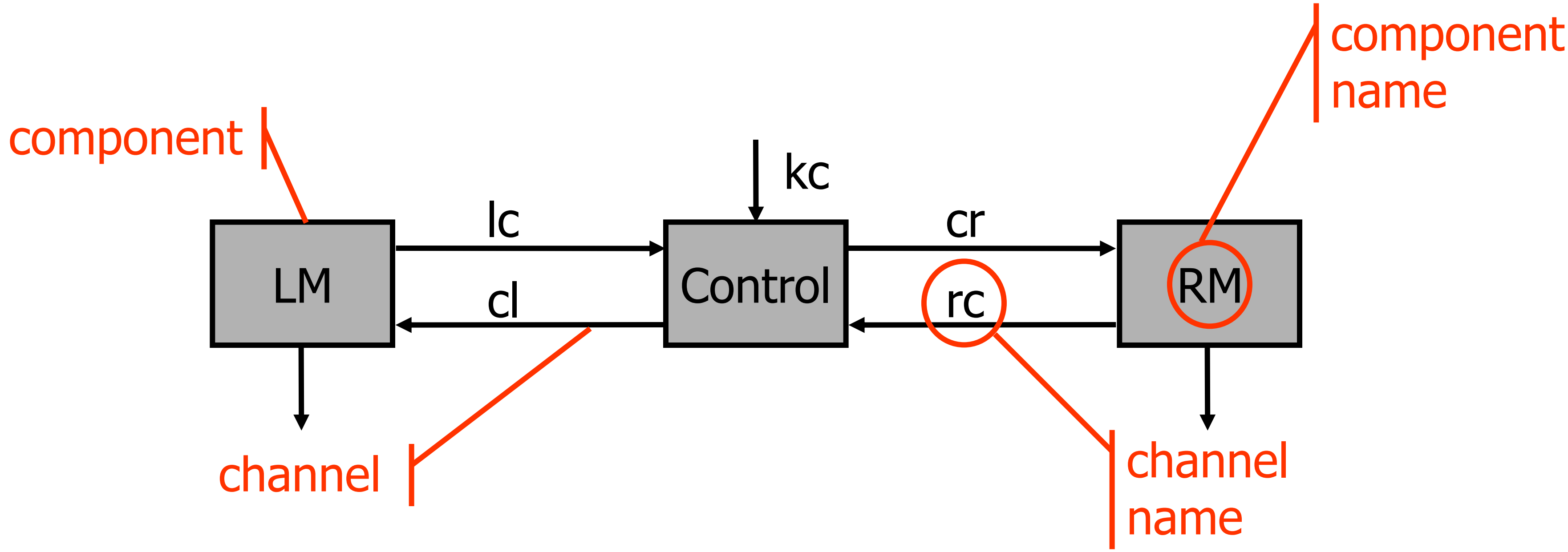
- for understanding - Gedankenmodell
- for communication
- for specification, design and documentation
- for analysis, validation, simulation, verification, certification
- for generation of implementations and tests
- for reuse

Modelling concepts provide methods for modelling!


Sp
, architecture ...
Informal
formalisation
Requireme
Engineerin
Validation
Formalized
system requ
in t rms of
service taxo
S
S
S2
deliver
tegration
R1⊗R2⊗R3⊗R4
R1
R2
R4
R3
architecture
S1
S2
S4
S3
Architecture design
Architecture verification
S ⇐ S1⊗S2⊗S3⊗S4
realization
integration
Component
implementation
verification
R1
R2
R4
R3
R1 ⇒ S1
R2 ⇒ S2
R3 ⇒ S3
R4 ⇒ S4

Towards a uniform model: Basic system model

System class: distributed, reactive systems

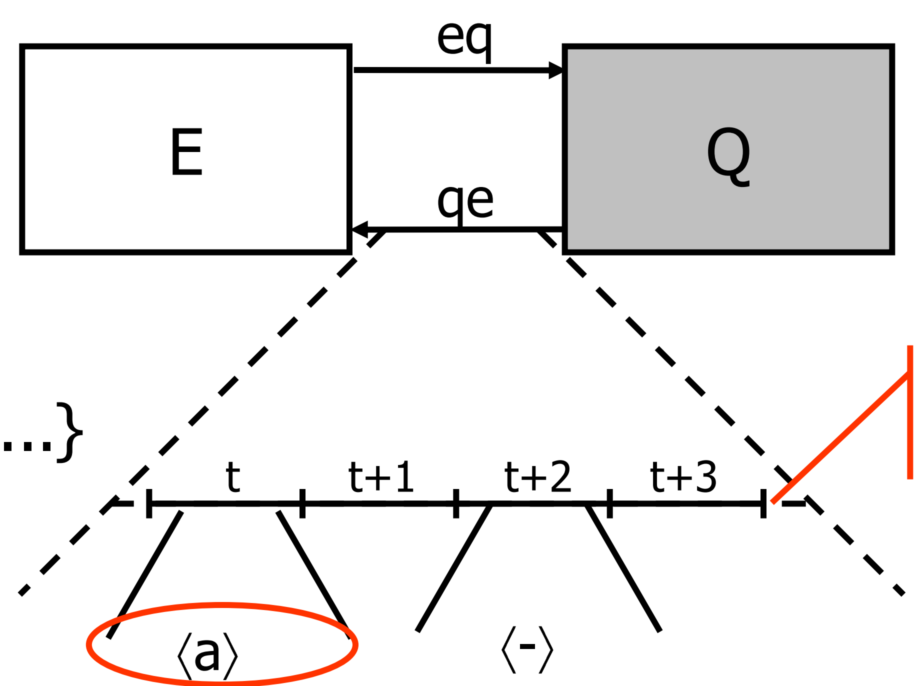


System consists of

- named components (with local state)
- named channels

driven by a global, discrete clock

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: Timed Streams and Channels

C	set of channels
$\text{Type}: C \to \text{TYPE}$	type assignment
$x : C \to (\mathbb{N}\backslash\{0\} \to M \cup \{-\})$	channel history for messages of type M
$\vec{C}$ or $\mathbb{H}[C]$	set of channel histories for channels in C

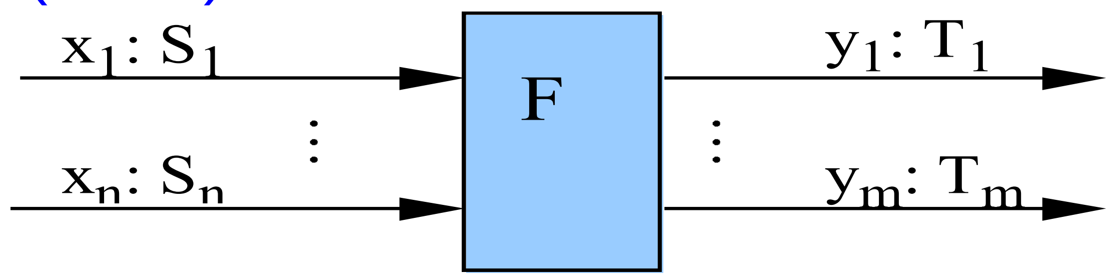
System interface model

Channel: Identifier of Type stream

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

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

Syntactic interface: $(I \blacktriangleright O)$



Interface behavior

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

Set of interface behaviours with input channels I and output channels O:

$$IF[I \blacktriangleright O]$$

Set of all interface behaviours: IF

System interface behaviour - causality

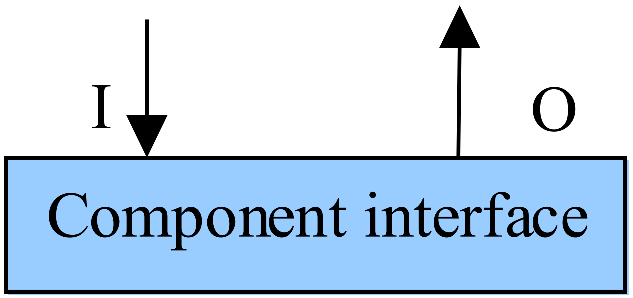
(I ▶ O) *syntactic interface* with set of input channels I and of output channels O

$F : \vec{I} \rightarrow \wp(\dot{O})$ *semantic interface* for (I ▶ O) with *timing property addressing strong causality*

let $x, z \in \dot{I}$, $y \in \dot{O}$, $t \in \mathbb{N}$:

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

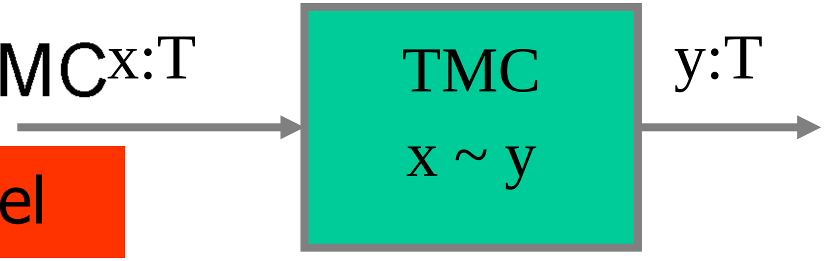
A system shows a total behavior



Example: Component interface specification

Spec name

A transmission component TMC



Input channel

TMC

in x: T
out y: T
x ~ y

Output channel

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

Specifying assertion

$\{m\}\#x$ denotes the number of occurrences of m in stream x

Verification: Proving properties about specified components

From the interface assertions we can prove

- Safety properties

$$\{m\}\#y > 0 \wedge y \in TMC(x) \Rightarrow \{m\}\#x > 0$$

- Liveness properties

$$\{m\}\#x > 0 \wedge y \in TMC(x) \Rightarrow \{m\}\#y > 0$$

System model: conclusion

A system

- has a scope (a boundary)
- a behaviour
  - ◇ black box view: interface
    - syntactic interface: defines the discrete events at the system boundary by input and output via ports, channels, messages (events, signals)
    - dynamic interface, interface behaviour: the processes of interaction in terms of discrete events at the system boundary
  - ◇ glass/white box view: an internal structure (state and/or distribution into sub-systems)
    - architecture in terms of sets of sub-systems and their relationships (communication connections)
    - state space

    and a behaviour
    - state transition relation with input and output
    - interactions between components
- properties
  - ◇ quality profile (performance, ...)
  - ◇ ...

Systems as State Machines

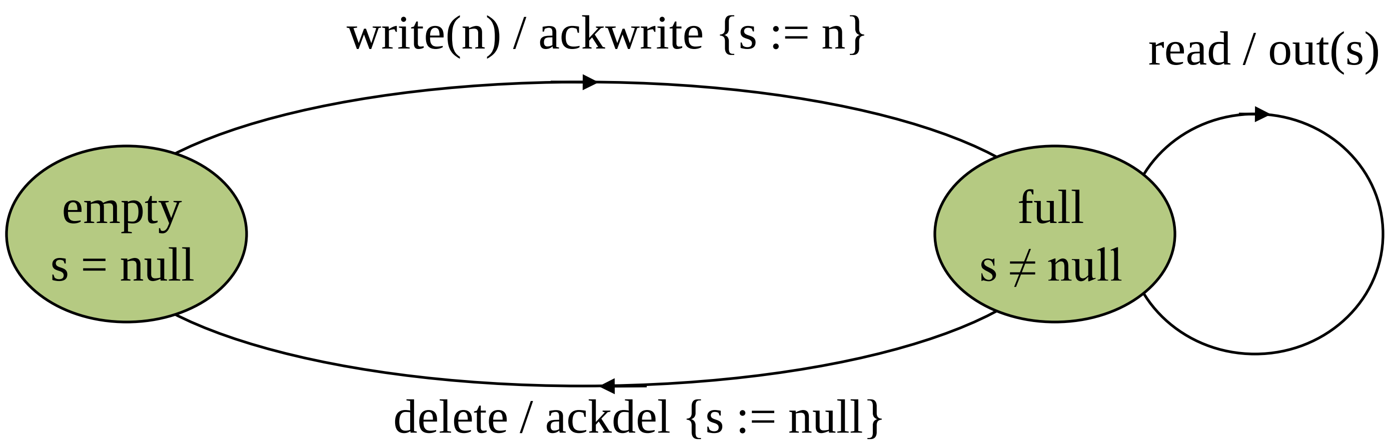
Technische Universität München
Institut für Informatik
D-85748 Garching, Germany

System and States

- Systems have states
- A state is an element of a state space
- We characterize state spaces by
  - ◇ a set of state attributes together with their types
- The behaviour of a system with states can be described by its state transitions

Example: Memory Cell as State Machine with Input/Output

Graphically (interpreted): state attribute s : Int | {null}



Representation of the State Machine as a Table

State s	Input	State s	Output
null	write(n)	n	ackwrite
n	read	n	out(n)
n	delete	null	ackdel

Representation as a Mathematical State Machine

State space: $\copyright = Z \ast \{\text{null}\}$

Input set: $E = \{\text{read, delete}\} \ast \{\text{write}(z): z \,\square\, Z\}$

Output set: $A = \{\text{ackwrite, ackdel}\} \ast \{\text{out}(z): z \,\square\, Z\}$

Equations for the state transition function:

$$\otimes : \copyright \times \oplus \,\square\, \copyright \times \Im$$

$\otimes(\text{null}, \text{write}(z)) = (z, \text{ackwrite})$

$\otimes(z, \text{read}) = (z, \text{out}(z))$

$\otimes(z, \text{delete}) = (\text{null}, \text{ackdel})$

State Machines in general

A state machine $(\otimes, \not\subset)$ consists of

- a set $\copyright$ of states - the state space
- a set $\not\subset \square \copyright$ of initial states
- a state transition function or relation $\otimes$
  - ◇ in case of a state machine with input/output:

    events (inputs E) trigger the transitions and events (outputs A) are produced by them respectively:

$$\otimes : \copyright \times \oplus \square \copyright \times \Im$$

    in the case of nondeterministic machines:

$$\otimes : \copyright \times \oplus \square \wp(\copyright \times \Im)$$

- Given a syntactic interface with sets I and O of input and output channels:

$$E = I \square M \cup \{-\}$$

$$A = O \square M \cup \{-\}$$

Computations of a State Machine with Input/Output

A state machine (⊗, ⊄) defines for each initial state

$$⌠_0 \;□\; ⊄$$

and each sequence of inputs

$$e_1, e_2, e_3, \ldots \;□\; E$$

a sequence of states

$$⌠_1, ⌠_2, ⌠_3, \ldots \;□\; ©$$

and a sequence of outputs

$$a_1, a_2, a_3, \ldots \;□\; A$$

through

$$(⌠_{i+1}, a_{i+1}) \;□\; ⊗(⌠_i, e_{i+1})$$

Computations of a State Machine with Input/Output

In this manner we obtain computations of the form

$$\sigma_0 \xrightarrow{a_1/b_1} \sigma_1 \xrightarrow{a_2/b_2} \sigma_2 \xrightarrow{a_3/b_3} \sigma_3 \quad \ldots$$

For each initial state ⌠0 □ Σ we define a function

$$F_{\sigma 0} : \dot{I} \to \wp(\dot{O})$$

with

$$F_{⌠0}(x) = \{y: \exists\ ⌠_i:\ ⌠0 = ⌠_0 \wedge \forall\ i\ □\ IN: (⌠_{i+1}, x_{i+1}) = ⊗(⌠_i, y_{i+1})\}$$

$F_{⌠0}$: interface behavior of transition function Δ for initial state ⌠0.

We define

$$Abs((⊗, ⊄)) = F_{\Lambda}$$

where:

$$F_{\Lambda}(x) = \{y\ □\ F_{⌠}(x) : y\ □\ F_{⌠}(x) \wedge ⌠\ □\ ⊄\}$$

F_{Λ} is called the interface behavior of the state machine (⊗, ⊄) .

Moore Machines

- A Mealy machine (⊗, ⊄) with

$$⊗ : © \times ⊕ □ \wp(© \times \Im)$$

  is called Moore machine if for all states ⌠ □ © and inputs e □ E the set

  out(⌠, e) = {a □ A: (⌠, a) = ⊗(⌠, e) }

  does not depend on the input e but only on state ⌠.

- Formally: then for all e, e′ □ E we have

  out(⌠, e) = out(⌠, e′)

Theorem:

If (⊗, ⊄) is a Mealy machine then $F_{⊄}$ is causal.

If (⊗, ⊄) is a Moore machine then $F_{⊄}$ is strongly causal.

Interface Abstraction

- For a given state machine with input and output we define the interface through
  - its syntactical interface (signature)
  - its interface behavior

- We call the transition of the state machine to its interface the interface abstraction.

Verification/derivation of interface assertions for state machines

- similar to program verification (find an invariant)
- needs sophisticated techniques

Conclusion Systems as State Machines

- Each state machines defines an interface behaviour
- Each interface behaviour represents a state machine
- State machines can be described
  - mathematically by their state transition function
  - graphically by state machine diagrams
  - structured by state transition tables
  - by programs
- State machines define a kind of operational semantics
- Systems given by state machines can be simulated
- From state machines we can generate code
  - state machines can represent implementations
- From state machines we can generate test cases

System model: conclusion

A system

- has a scope (a boundary)
- a behaviour
  - black box view: interface
    - syntactic interface: defines the discrete events at the system boundary by input and output via ports, channels, messages (events, signals)
    - dynamic interface, interface behaviour: the processes of interaction in terms of discrete events at the system boundary
  - glass/white box view: an internal structure (state and/or distribution into sub-systems)
    - architecture in terms of sets of sub-systems and their relationships (communication connections)
    - state space

    and a behaviour
    - state transition relation with input and output
    - interactions between components
- properties
  - quality profile (performance, ...)
  - ...