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

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, ...)
  - ◇ ...

Systems: the model

Sets of typed channels

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

$O = \{y_1 : T'_1, y_2 : T'_2, \dots \}$

syntactic interface $(I \blacktriangleright O)$

data stream of type T

$STREAM[T] = \{IN\backslash\{0\} \rightarrow T^*\}$

valuation of channel set C

$\vec{C} = \{C \rightarrow STREAM[T]\}$

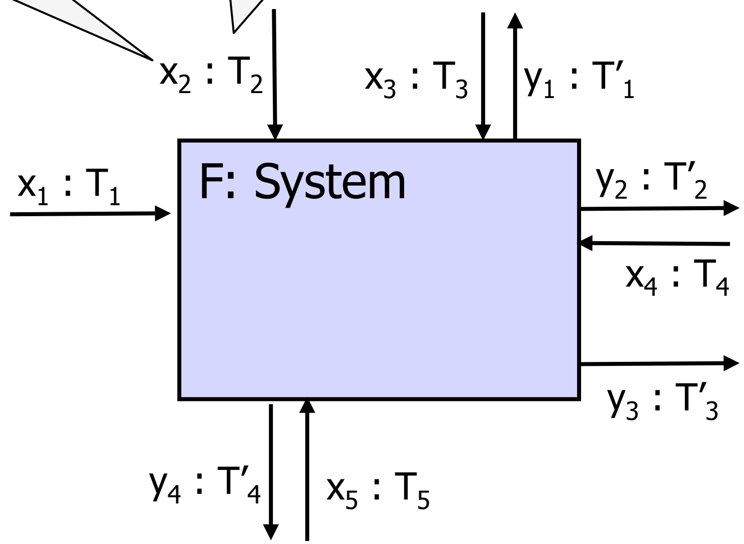
interface behavior for syn. interface $(I \blacktriangleright O)$

$[I \blacktriangleright O] = \{\vec{I} \rightarrow \wp(\vec{O})\}$

interface specification

$p: I \cup O \rightarrow IB$

represented as interface assertion S - logical formulae with channel names as attributes of type stream



Causality

Definition. Causal Behavior

For a mapping

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

the set

$$\text{dom}(F) = \{x: F(x) \neq \emptyset\}$$

is called the *domain* of F.

F is called *total*, if $\text{dom}(F) = \vec{I}$, otherwise F is called *partial*.

F is called *causal*, if for all $t \in \mathbb{N}$ and all input histories $x, z \in \vec{I}$:

$$x, z \in \text{dom}(F) \wedge x{\downarrow}t = z{\downarrow}t \Rightarrow \{y{\downarrow}t: y \in F(x)\} = \{y{\downarrow}t: y \in F(z)\}$$

F is called *strongly causal*, if (for all $t \in \mathbb{N}$ and all input histories $x, z \in \vec{I}$):

$$x, z \in \text{dom}(F) \wedge x{\downarrow}t = z{\downarrow}t \Rightarrow \{y{\downarrow}t+1: y \in F(x)\} = \{y{\downarrow}t+1: y \in F(z)\}$$

Realizability

For an interface behaviour F: $\vec{I} \to \wp(\vec{O})$
a strongly causal total function f: $\vec{I} \to \vec{O}$ such that

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

is called realisation.

F is called *realizable* if there exists a realisation for F.

Realisation f: $\vec{I} \to \vec{O}$ provides a deterministic strategy to calculate for every input history x a particular output history f(x) with

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

f essentially defines a deterministic Mealy machine.

Example: Nonrealizable strongly causal behaviour

Consider

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

Note that F is strongly causal but not realizable.

Full Realizability

Let [F] denote the set of realisations for F.

Definition: Full Realizability

An interface behaviour F is called fully realizable if it is realizable and for all input histories x:

$$F(x) = \{f(x): f \in [F]\}$$

holds.

Realizability and state machines

Theorem:

A strongly causal behaviour F is realizable iff there exists a deterministic Moore machine with f as its interface abstraction that is a realisation of F.

A strongly causal behaviour F is fully realizable iff there exists a Moore machine with F as its interface abstraction.

Designing Architectures:
Composing and Decomposing Systems

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

Composition and Decomposition of Systems

$F_k \in IF[I_k \blacktriangleright O_k]$ for k = 1, 2

where $O_1 \cap O_2 = \emptyset$, $I_k \cap O_k = \emptyset$

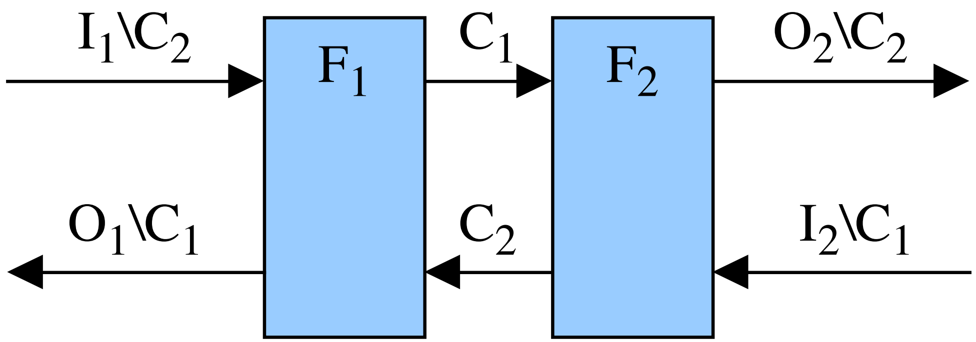
shared channels :

$C_1 = O_1 \cap I_2$

$C_2 = O_2 \cap I_1$

$I = I_1 \backslash C_2 \cup I_2 \backslash C_1$

$O = O_1 \backslash C_1 \cup O_2 \backslash C_2$



$F_1 \times F_2 \in IF[I \blacktriangleright C]$, where $C = (O_1 \cup O_2 \cup C_1 \cup C_2)$

$z \in IH[I_1 \cup I_2 \cup O_1 \cup O_2]$

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

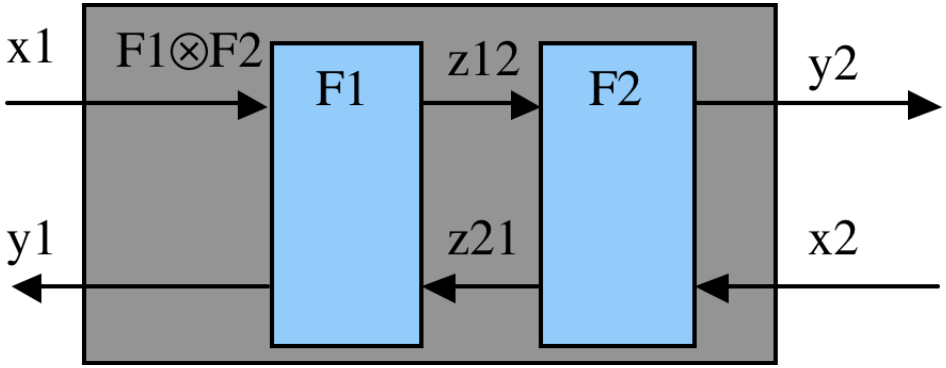
$F_1 \otimes F_2 \in IF[I \blacktriangleright O]$

$(F_1 \otimes F_2).(x) = \{y|O: y \in (F_1 \times F_2).(x)\}$ „closed view“

Interface specification composition rule (closed form: Hiding Internal Channels)

F1⊗F2

Hiding internal channels



F1

in x1, z21: T **out** y1, z12: T
P1

F2

in x2, z12: T **out** y2, z21: T
P2

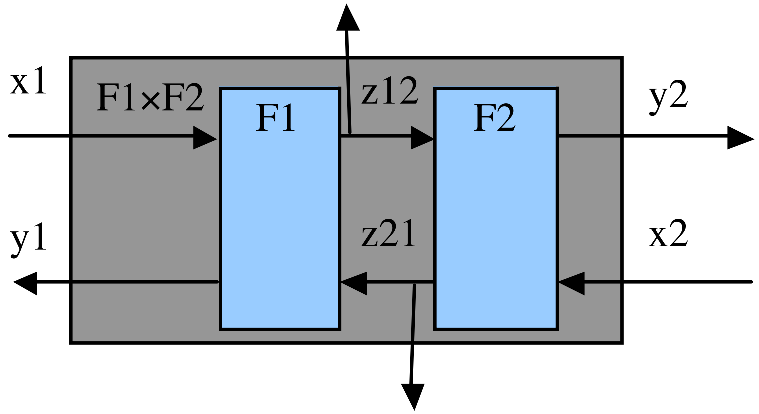
F1⊗F2

in x1, x2: T **out** y1, y2: T
$\exists$ z12, z21: P1 $\wedge$ P2

Interface specification composition rule (open form)

F1×F2

Visibility of internal channels



F1
in x1, z21: T **out** y1, z12: T
P1

F2
in x2, z12: T **out** y2, z21: T
P2

F1×F2
in x1, x2: T **out** y1, y2, z12, z21: T
P1 $\wedge$ P2

Remarks Composition

- Given a set of components K with $F_k \in$ IF we write

$$\otimes \{F_k : k \in K\}$$

  for the interface behaviour of the architecture formed by

$$F_1 \otimes F_2 \otimes F_3 \otimes F_4 \ldots$$

- Operator $\otimes$ is parallel composition including feedback
- Operator $\otimes$ is logically represented by logical conjunction for interface assertions and existential quantification for channel hiding
- Strong causality
  - ◇ reflects the flow of time
  - ◇ guarantees unique fixpoints of feedback loops in the case of deterministic systems

Contracts
Assumption/Promise

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

Contracts

An interface specification can be seen as a contract between

- the user of a system
  - interacting with the system
  - using the system as sub-system (“component”) in a larger system
- the implementer of a system

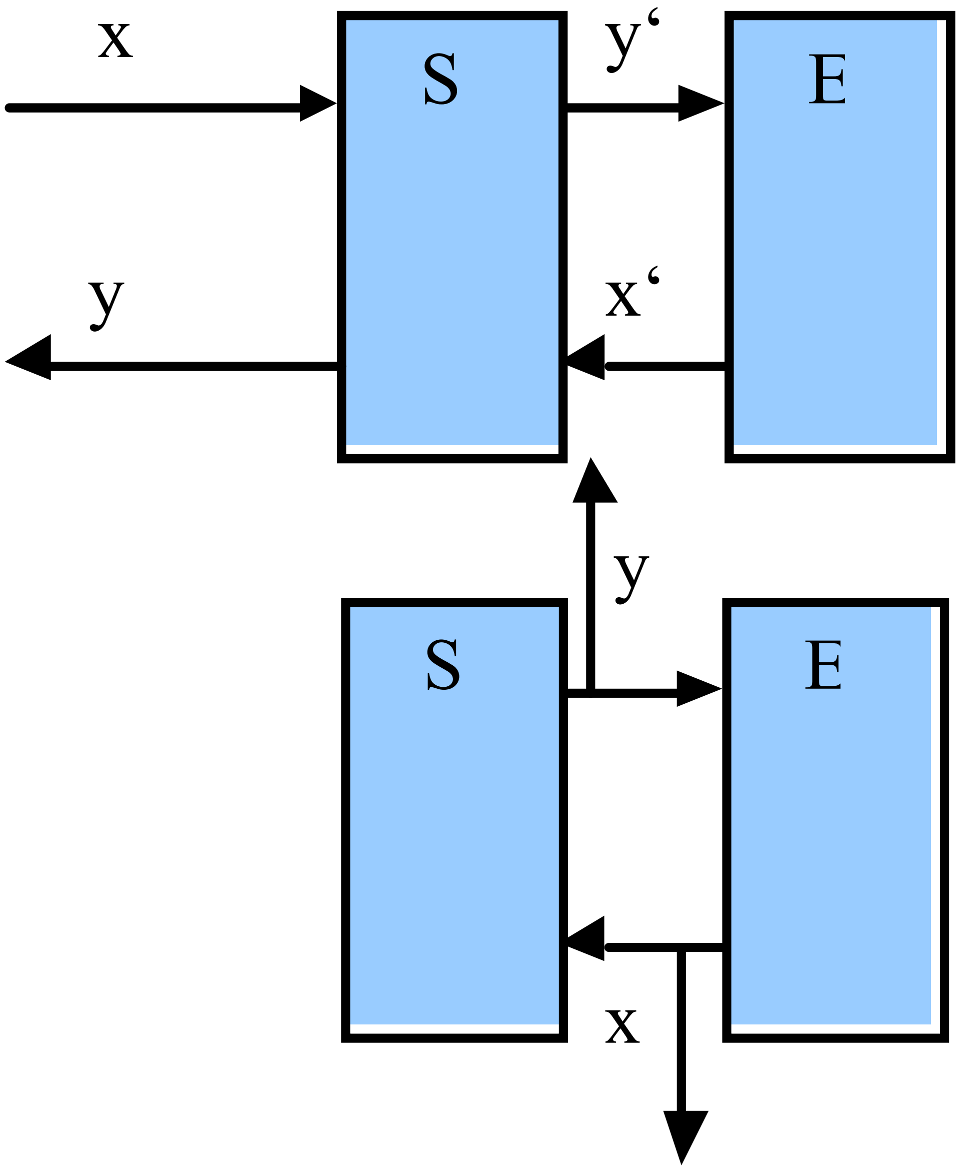
Key idea:

The contract includes all information needed

- for proper use
- for correct implementation

of the SuC (system under consideration)

Composition



E⊗S

Hiding internal channels

E×S

Visibility of internal channels

A/P-Pattern

- Let System be the set of all systems.
- Composing system $S \in System$ with environment $E \in Env(S) \subseteq System$ results in

  $$E \times S \in System$$

- Based on composition operator $\times$ we formulate contracts by assumptions and promises:

  $$Con(S) \equiv \forall\, E \in Env(S): Asu(E) \Rightarrow Pro(E \times S)$$

  where
  - ◇ $Con(S)$ is a system specification called contract,
  - ◇ $Asu(E)$ is an environment specification called assumption and
  - ◇ $Pro(E \times S)$ is a specification about the system $E \times S$ called a promise.
- The predicates specify system properties

  $$Con, Asu, Pro: System \rightarrow IB$$

Semantics of Assumption/Promise: Interface Assertions

Given a syntactic interface (I▸O) an interface assertion

is a Boolean expression p(x, y) where p is a predicate

$$p: \vec{I} \times \vec{O} \rightarrow \mathbb{B}.$$

and $x \in \vec{I}$ and $y \in \vec{O}$ are input and output histories

Semantics of Contracts by Logical Implication

- Interface assertions structured into following pattern:

  assumption: asu(x, y)

  promise: pro(x, y)

  with the meaning: if the environment fulfils the assumption

  asu(x, y)

  then the system fulfils the promise

  pro(x, y)

- We require of environment E the assumption specified by

  $\mathrm{Asu}(E) \equiv [\forall\, x, y{:}\ x \in E(y) \Rightarrow \mathrm{asu}(x, y)]$

  and of the system S and its environment E the promise is specified by

  $\mathrm{Pro}(E, S) \equiv [\forall\, x, y{:}\ y \in (E \times S)(x) \Rightarrow \mathrm{pro}(x, y)]$

  The combination of these predicates then specifies a contract

  $\mathrm{Con}(S) \equiv [\forall\, E{:}\ \mathrm{Asu}(E) \Rightarrow \mathrm{Pro}(E, S)]$

  This defines the meaning of a functional contract.

Deriving Implicative Assertions from Contracts

- Consider

  $$\mathrm{Asu}(E) \equiv [\forall\, x, y: x \in E(y) \Rightarrow \mathrm{asu}(x, y)]$$

  $$\mathrm{Pro}(E, S) \equiv [\forall\, x, y: y \in (E \times S)(x) \Rightarrow \mathrm{pro}(x, y)]$$

  $$\mathrm{Con}(S) \equiv [\forall\, E: \mathrm{Asu}(E) \Rightarrow \mathrm{Pro}(E, S)]$$

  which unfolds into

  $$\mathrm{Con}(S) \equiv$$

  $$[\forall\, E: [\forall\, x, y: x \in E(y) \Rightarrow \mathrm{asu}(x, y)] \Rightarrow$$

  $$[\forall\, x, y: y \in (E \times S)(x) \Rightarrow \mathrm{pro}(x, y)]]$$

- The restriction of causality and realizability for environment E and S allows us to derive further contract properties.

Causality and Realizability

- In case assertion asu(x, y) is causal and fully realizable there exists a most general environment E_{gen} such the following property holds :

  $$\forall\ x, y: x \in E_{gen}(y) \Leftrightarrow asu(x, y)$$

- If a most general environment exists, then

  $$Con(S) \equiv [\forall\ x, y: y \in (E_{gen} \times S)(x) \ \Rightarrow pro(x, y)]$$

  This semantic interpretation of the A/P pattern is equivalent to

  $$Con(S) \equiv [\forall\ x, y: y \in S(x) \wedge x \in E_{gen}(y) \Rightarrow pro(x, y)]$$

  which leads by the specification of E_{gen} to the following contract:

  $$Con(S) \equiv \forall\ x, y: asu(x, y) \Rightarrow (y \in S(x) \Rightarrow pro(x, y))$$

  and to interface assertion con(x, y) for contract Con(S)

  $$con(x, y) \equiv [asu(x, y) \Rightarrow pro(x, y)]$$

Why Assumptions have to Speak about System Output

- Consider a system with input channel x and output channel y which numbers as messages specified by

  $$\mathrm{asu}(x, y) \equiv \forall\, t: \forall\, n \in \mathrm{IN}: n\#(x{\downarrow}t) \le (n\#y{\downarrow}t)+1$$

  $$\mathrm{pro}(x, y) \equiv \forall\, n \in \mathrm{IN}: n\#x = n\#y$$

- We get the specification in terms of an interface assertion

  $$\mathrm{con}(x, y) \equiv [\mathrm{asu}(x, y) \Rightarrow \mathrm{pro}(x, y)]$$

- The promise is only guaranteed if a next copy of a number n is never sent to the system before the copy previously sent has been forwarded.

Implicative interface assertions

Tab. 1 Cases of Validity of con(x, y), asu(x, y), and pro(x, y)

<table>
<tr><th>con(x, y)</th><th>asu(x, y)</th><th>pro(x, y)</th><th>Interpretation</th></tr>
<tr><td>true</td><td>true</td><td>true</td><td>for system S history output y is a correct output for legal input history x</td></tr>
<tr><td>false</td><td>true</td><td>false</td><td>for system S history y is an incorrect output for legal input history x</td></tr>
<tr><td>true</td><td>false</td><td>true</td><td rowspan="2">for system S and output history y input history x is illegal, environments that produce y in reaction to x are illegal, y is irregular output</td></tr>
<tr><td>true</td><td>false</td><td>false</td></tr>
</table>

Architectural Contracts & Implicative Assertions

- Given an A/P specification

  assumption: $asu(x, y)$

  promise: $pro(x, y)$

  one interpretation is that the system S is only used in environments E for which assumption $asu(x, y)$ holds. Then we get

  $$asu(x, y) \wedge pro(x, y)$$

  This interpretation is called *architectural contract*.
- The implicative assertion

  $$con(x, y) \equiv [asu(x, y) \Rightarrow pro(x, y)]$$

  specifies the properties implied for system S by the A/P specification.
- If system S is used in environments E with specifying assertion $env(x, y)$ we get by composition for the composite system $E \times S$

  $$env(x, y) \wedge (asu(x, y) \Rightarrow pro(x, y))$$

  which is different to the architectural contract interpretation.

Example: General Implicative Assertions

- Let n be a given natural number.
- Given system contract with input channel x and output channel y, both carrying natural numbers

$$\mathrm{con}(x, y) \equiv [n\#y = 0 \Rightarrow n\#x = 0]$$

- The premise is not a meaningful assumption, since
  - ◇ there does not exist an environment that guarantees assertion $n\#y = 0$
  - ◇ it does not speak about input x but only about output y.
- Assertion $n\#y = 0$ is not causal in history y, since

$$y{\downarrow}t = y'{\downarrow}t \Rightarrow \forall\, x: (n\#y = 0) \equiv (n\#y' = 0)$$

  which does not hold.
- Assertion $n\#y = 0$ is not a healthy assumption, since it is not realizable by any environment.

Example: Implicative Assertions (ctd)

- Assertion

  $$con'(x, y) \equiv [n\#x > 0 \Rightarrow n\#y > 0]$$

  is equivalent to assertion $con(x, y)$ by contraposition
- Assertion $n\#x > 0$ is causal in history y since the formula

  $$y\downarrow t = y'\downarrow t \Rightarrow \forall x: (n\#x\downarrow t > 0) \equiv (n\#x\downarrow t > 0)$$

  holds.
- This assertion may be interpreted as an A/P-format

  assumption: $n\#x > 0$

  promise: $n\#y > 0$

  which is a meaningful (but rather simple) contract.

Non-satisfiable Specifications

- We call assumption Asu(E) about environment E non-satisfiable if there does not exist some environment E such that Asu(E) holds.

  Then contract Con(S) is trivial.
- Let Asu be specified based on asu as defined above.
  - ◇ If asu(x, y) is false, then Asu is non-satisfiable.
  - ◇ Even in cases where asu(x, y) is not identical to false, predicate Asu may be non-satisfiable.

Non-satisfiable Specifications

Theorem:

- If every environment E can be represented by a total Mealy machine, then Asu(E) is satisfiable if and only if asu(x, y) is realizable (for the environment with input y and output x).
- Proof:
  For a specification asu(x, y) there exists a Mealy machine that satisfies asu(x, y) if and only if asu(x, y) is realizable.

Safety and Liveness of Interface Assertions

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

Notation

$s{\downarrow}t$ prefix of length $t \in \mathbb{N}$ of the stream s (which is a sequence of length t),

$z{\downarrow}t$ history of streams prefix of length $t \in \mathbb{N}$ of the history $z \in \vec{C}$

$$(z{\downarrow}t)(c) = (z(c)){\downarrow}t$$

Notation: Given a predicate

$$p\colon \vec{C} \to \mathbb{B}$$

we specify for every time $t \in \mathbb{N}$

$$p(z{\downarrow}t) \equiv \exists\, z' \in \vec{C} : z{\downarrow}t = z'{\downarrow}t \wedge p(z')$$

Safety & Liveness

A predicate R is a pure safety property if the following equivalence holds for all histories x and y:

$$R(x, y) \equiv \forall\ t: R(x{\downarrow}t, y{\downarrow}t)$$

R is a pure liveness property if

$$\forall\ t: R(x{\downarrow}t, y{\downarrow}t)$$

Canonical Decomposition Assertions into Safety and Liveness Parts

- The safety part R^* of an interface assertion $R(x, y)$ is given by the following equation

$$R^*(x, y) \equiv \forall\, t: R(x{\downarrow}t, y{\downarrow}t)$$

- R is called safety realizable if:

$$\forall\, x: \exists\, y: R^*(x, y)$$

- For predicate R we get liveness property R^∞ included in property R by

$$R^\infty(x, y) \equiv (\neg R^*(x, y) \vee R(x, y))$$

- To show that R^∞ is a liveness property we have to prove

$$\forall\, t: R^\infty(x{\downarrow}t, y{\downarrow}t)$$

- Theorem: $R(x, y) \equiv R^*(x, y) \wedge R^\infty(x, y)$

Assumption/Promises as Safety and Liveness Properties

- We consider the interface assertion con(x, y) with

$$\text{con}(x, y) \equiv [\text{asu}(x, y) \Rightarrow \text{pro}(x, y)]$$

- The liveness conditions in assertion asu(x, y) for input history x may depend on safety properties of y.

- A typical example would be
  - ◊ If y(t) is a query, then
  - ◊ there exists a time t' > t such that x(t') is a reply to this query.

Assumption asu and promise pro are both safety property

- Then the A/P-scheme is equivalent to the following assertion:

  $$\text{con}(x, y) \equiv \forall\, t: [\text{asu}(x{\downarrow}t, y{\downarrow}t) \Rightarrow \text{pro}(x{\downarrow}t, y{\downarrow}t)]$$

  This is the consequence of the required causality of con(x, y).

Assumption asu is safety, promise pro is liveness property

- Then the A/P-specification con(x, y) is equivalent to the following assertion:

$$\text{con}(x, y) \equiv [\forall\, t\colon \text{asu}(x{\downarrow}t, y{\downarrow}t)] \Rightarrow \text{pro}(x, y)$$

  This is the consequence of the required causality of con(x, y).

Assumption asu is liveness, promise pro is safety property:

- In this case we can strengthen the specification according to the realizability on con(x, y)

$$con(x, y) \equiv pro(x, y)$$

- Since the violation of assumption asu(x, y) cannot be observed in finite time, but promise pro can only be violated in finite time, a computation strategy has to observe promise pro in any case.

An example

$asu(x, y) \equiv (true\#x = \infty)$

$pro(x, y) \equiv (true\#y = 0)$

Assume a realization f that

fulfils $true\#f(x) > 0$ for some x with $true\#x = n \in IN$.

This leads to a contradiction since by

$true\#f(x) > 0$ there exists some t with

$true\#f(x){\downarrow}t > 0$ and thus for history x' with

$x{\downarrow}t = x'{\downarrow}t$ and $true\#x' > \infty$

we get $true\#f(x') > 0$ which violates the specification

$true\#x = \infty \Rightarrow true\#f(x) = 0$

Assumption asu and promise pro as liveness properties

- In this case the condition

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

  can be fulfilled by fulfilling promise pro(x, y) in any case.
- Otherwise, the liveness condition have to fit together.

An example

$$asu(x, y) \equiv (\text{true}\#x = \infty)$$

$$pro(x, y) \equiv (\text{true}\#y < \infty)$$

Decomposing A/P Specification Into Safety and Liveness

- We decompose assumption asu and promise pro into pure safety properties asu_S, pro_S and pure liveness properties asu_L and pro_L such that

$$con(x, y) \equiv$$

$$[asu_S(x, y) \land asu_L(x, y) \Rightarrow pro_S(x, y) \land pro_L(x, y)]$$

- For a strongly causal and realizable specification con(x, y) we can derive specific assertions

$$asu_S(x, y) \Rightarrow pro_S(x, y)$$

$$asu_S(x, y) \land asu_L(x, y) \Rightarrow pro_L(x, y)$$

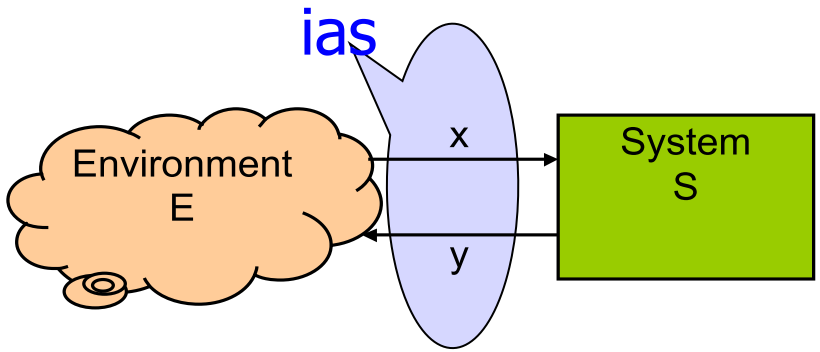
Conclusion

- Analysing the assumption/promise pattern additional consequences are derived by
  - Causality and realizability requirements
  - Safety and liveness considerations

Contracts and Architectures

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

From interaction assertions ias to contracts



- Let ias(x, y) be an assertion that characterizes the interaction between system S and its environment E, called interaction assertion.
- ias provides an observation/specification of the traffic between E and S
- Can we derive of a contract for S from assertion ias(x, y) that captures the obligations of system S w.r.t. ias?

Pathological cases

- It is clear that we cannot expect to get a reasonable contract from ias(x, y) in every case.
- A simple example would be ias(x, y) = false.

Separation

- Given the healthiness condition for the interface assertion

  $\exists$ x, y: ias(x, y)

  we can do a separation of R into an assumption and a promise (for the safety properties in R) as follows.
- We specify the responsibilities of the system S that accepts the input history x and issues and output history y such that assertion

  ias(x, y)

  holds.
- We are looking for assertions asu(x, y) and pro(x, y) such that

  asu(x, y) $\wedge$ pro(x, y) $\Rightarrow$ ias(x, y)

  and

  assumption asu(x, y)
  promise pro(x, y)

  is a healthy assumption promise specification.
- If ias(x, y) is strongly causal in x and fully realizable, then asu(x, y) $\equiv$ true is a valid choice.

Separation

- If

  $\forall x: \exists y: ias(x, y)$

  does not hold, then we need to construct an assumption $asu(x, y)$ and a promise $pro(x, y)$ such that

  (1) $asu(x, y)$ is causal in y and realizable

  (2) $asu(x, y) \Rightarrow pro(x, y)$ is strongly causal in x and realizable.

  and

  $asu(x, y) \wedge pro(x, y) \Rightarrow R(x, y)$

Separating ias

- we derive an A/P-specification for system S with the weakest assumption by the following steps:

1. Separate ias into a safety and a liveness part
2. Do the canonical separation of the safety part of ias into an assumption and a promise
3. Separate the liveness part of ias in an assumption and a promise
4. Construct the A/P-specification of S from the liveness and safety parts of the assumption and the promise.

Separation: Safety part

$asu(x, y) \equiv$
$[ias(x, y{\downarrow}0) \wedge (\forall\ t: ias(x{\downarrow}t, y{\downarrow}t) \Rightarrow ias(x{\downarrow}t+1, y{\downarrow}t))]$

$pro(x, y) \equiv$
$(\forall\ t: ias(x{\downarrow}t+1, y{\downarrow}t) \Rightarrow ias(x{\downarrow}t+1, y{\downarrow}t+1))$

Theorem:

Under the condition that assertion ias(x, y) is a pure safety property:

$$(asu(x, y) \wedge pro(x, y)) \Leftrightarrow ias(x, y)$$

Separation: Liveness part

- There are liveness conditions that can be seen as assumptions as well as promises.

- An example is the assertion

  $$\{1\}\#x + \{0\}\#y = \infty$$

  which can either be fulfilled
  - by assuming an infinite number of copies of 1 in input history x or
  - by promising an infinite number of copies of 0 in y (or both).

Separation: Liveness part

Given an interaction assertion

$$ias(x, y)$$

that is a liveness condition we define an assumption asu_{ias} as follows

$$asu_{ias}(x) \equiv \exists\, y: ias(x, y)$$

and a promise pro_{ias} by the equation

$$pro_{ias}(x, y) \equiv ias(x, y)$$

by this definition those parts of the liveness property ias that can either be fulfilled by the environment or by the system under consideration.

- In the later case they are made part of the promise.
- This way we get a weakest assumption and the strongest promise.

Example. From interaction to interface assertions

- Given the specification

  $$ias(x, y) \equiv (x \approx y \wedge \forall t: (\#y{\downarrow}t)+b \geq \#x{\downarrow}t \wedge \#x{\downarrow}t \geq \#y{\downarrow}t)$$

  where x and y are streams of data and b is a given number and $x \approx y$ specifies that x and y carry the same stream of messages (eliminating empty slots "-")
- We choose the assumption

  $$asu(x, y) \equiv \forall t: (\#y{\downarrow}t)+b \geq \#x{\downarrow}t$$
- $asu(x, y)$ is causal in y and realizable.
- We choose the promise

  $$pro(x, y) \equiv (x \approx y \wedge \forall t: \#x{\downarrow}t \geq \#y{\downarrow}t)$$
- Assertion $pro(x, y)$ is strongly causal and fully realizable.
- We get

  $$pro(x, y) \wedge asu(x, y) \Rightarrow ias(x, y)$$

Example. Non-realizable Specification

- Consider a system with input channel x and output channel y both carrying boolean messages:

  $$R(x, y) = \quad [\ (true\#x < \infty \Rightarrow true\#y = \infty)$$
  $$\wedge\ (true\#x = \infty \Rightarrow true\#y < \infty)\ ]$$

- All the involved assertions are liveness properties. We get

  $$\forall\ x: \exists\ y: R(x, y)$$

- However, there does not exist a causal function f with

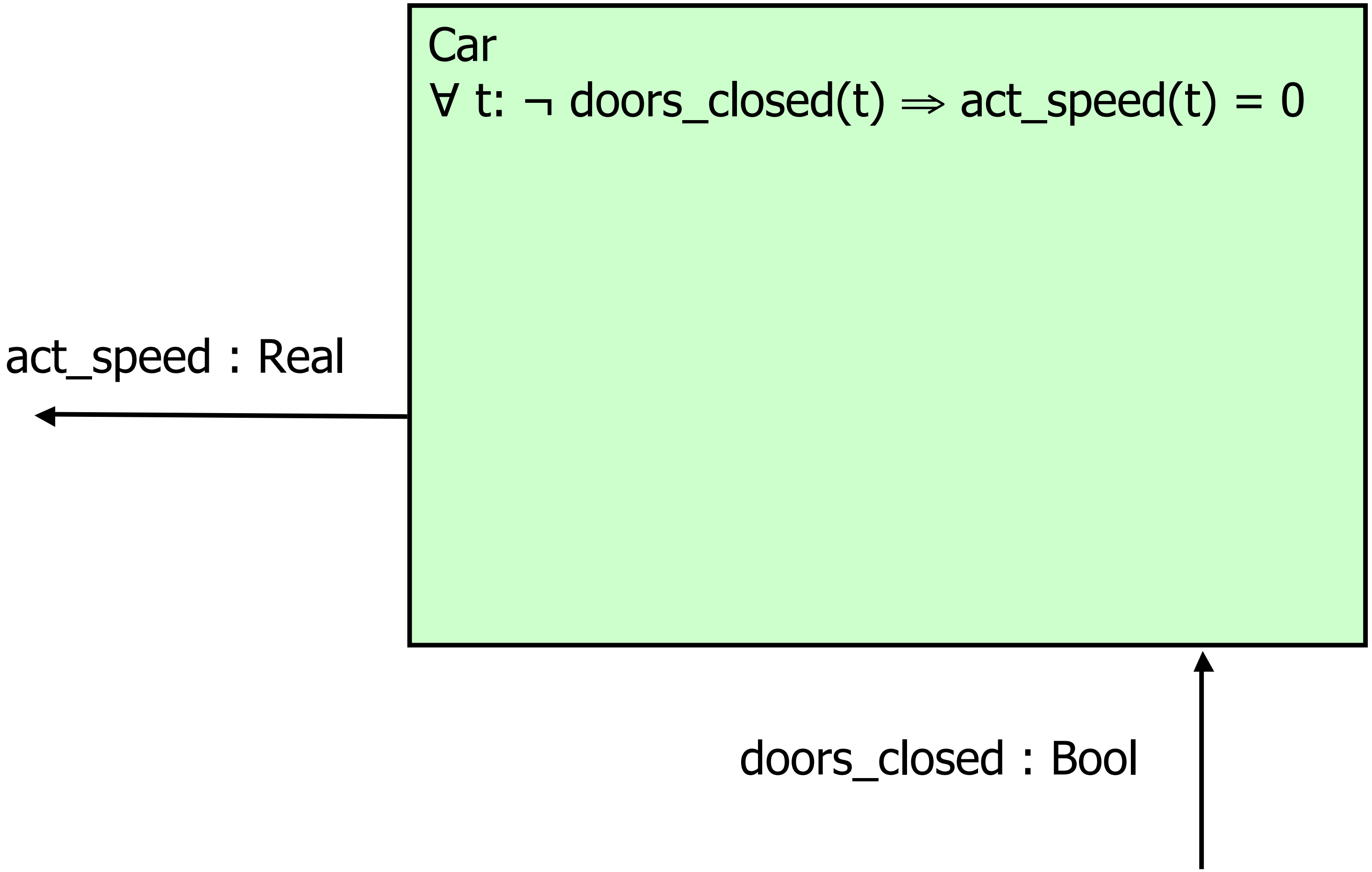
  $$\forall\ x: R(x, f(x))$$

- Otherwise, there would exist a strongly causal function f with a fixpoint $y = f(y)$ such that $R(x, y)$ holds which delivers a contradiction.
- The example suggests that non-realizable specifications include liveness properties that cannot be realized.

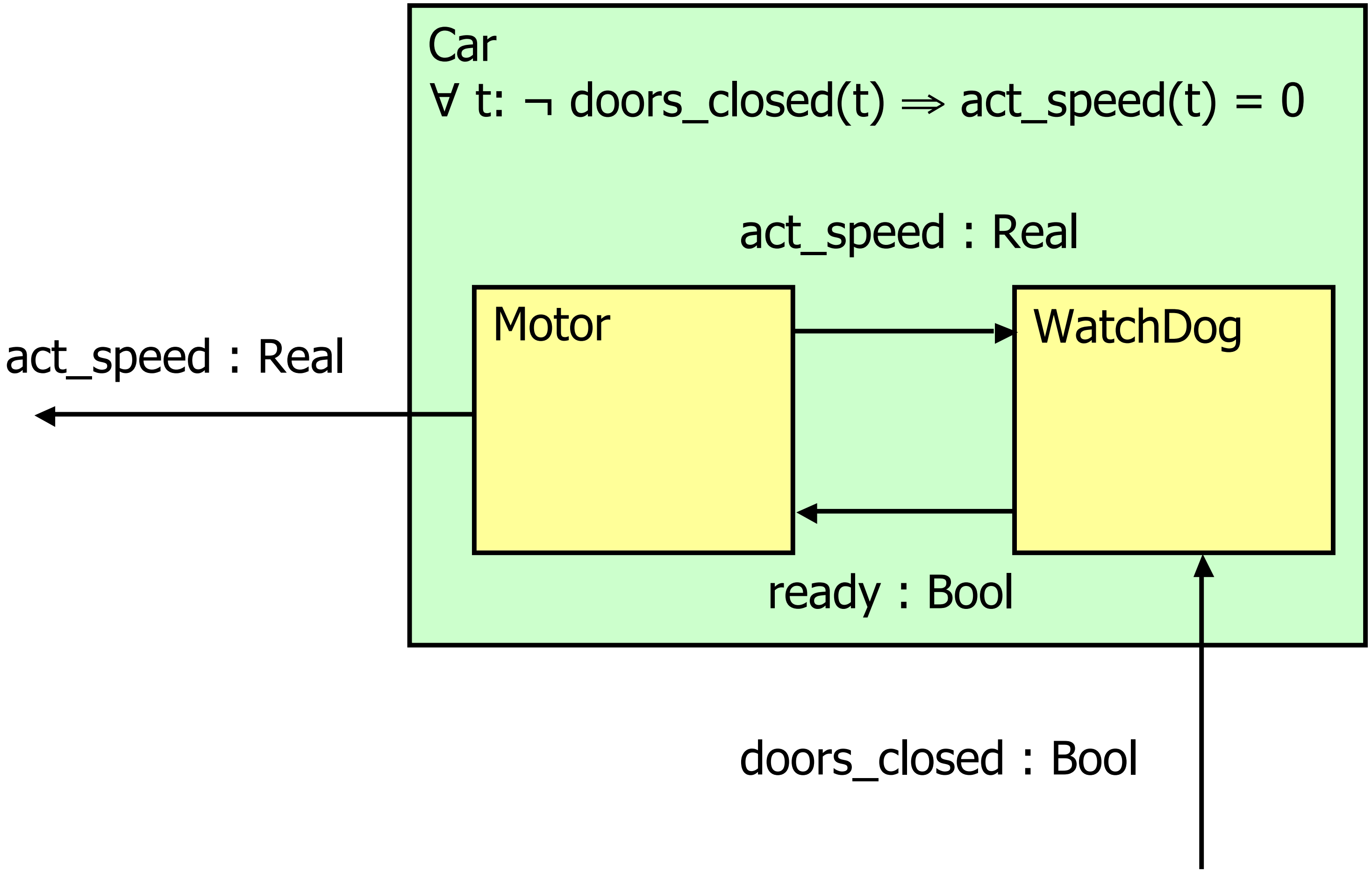
Assumptions in Architectural Modelling

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

Example: Simple Watch Guard in a Car



Glass Box Specification of a Car´s Architecture



Example: how A/P-specifications can be formulated

- The specification

  $\forall$ t: $\neg$ doors_closed(t) $\Rightarrow$ act_speed(t) = 0

  can only be guaranteed if the two inner components work together. This requires

  $\forall$ t: $\neg$ ready(t) $\Rightarrow$ act_speed(t) = 0

- Then the system specification holds if

  $\forall$ t: $\neg$ doors_closed(t) $\Rightarrow$ $\neg$ ready(t)

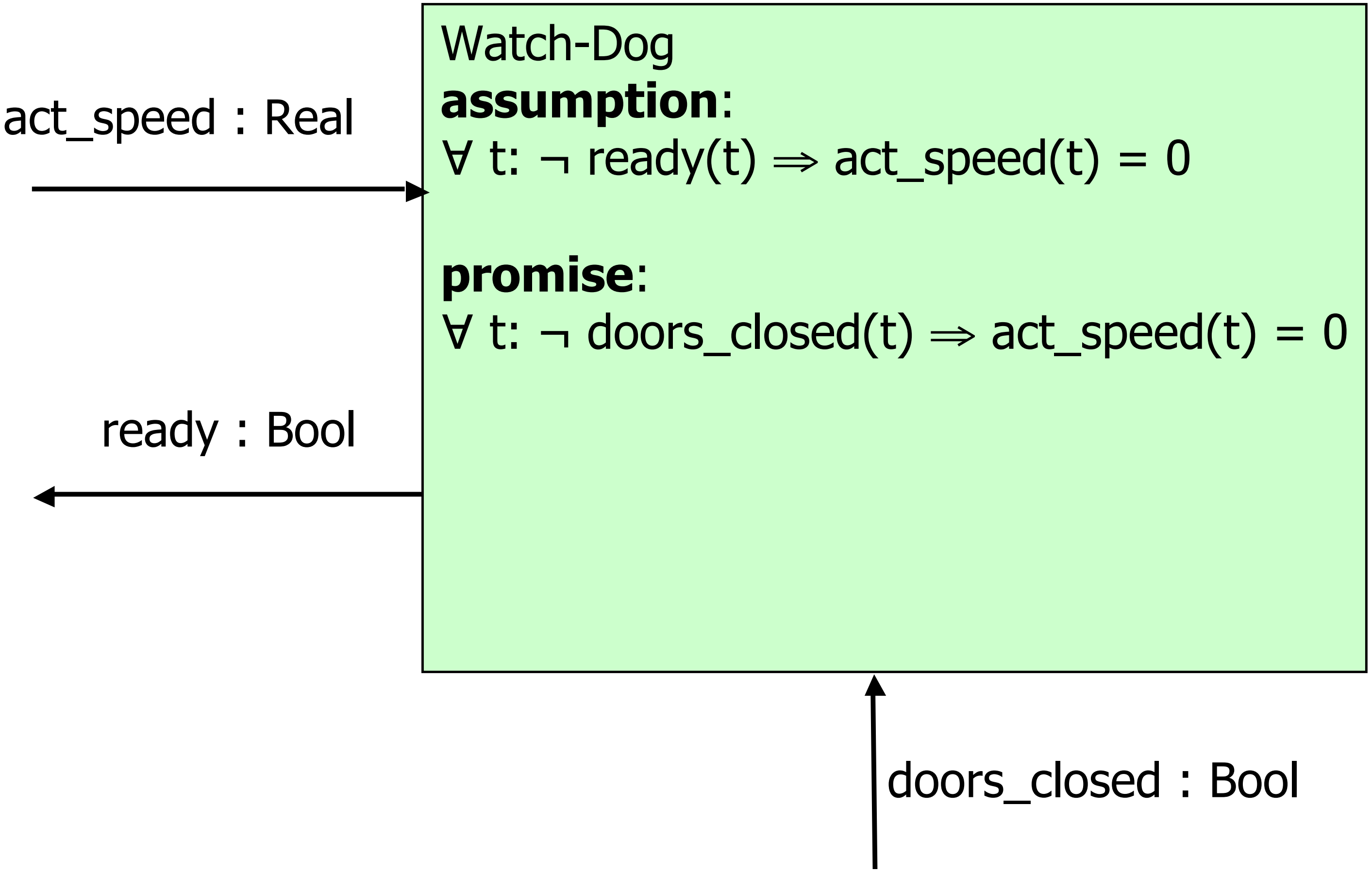
- This is logically equivalent to the A/P-specification for the WatchDog

  assumption: $\forall$ t: $\neg$ ready(t) $\Rightarrow$ act_speed(t) = 0

  promise: $\forall$ t: $\neg$ doors_closed(t) $\Rightarrow$ act_speed(t) = 0

- In other words,
  - the overall system specification can be guaranteed by the watchdog
  - only if the assumption about the behaviour of the component motor holds.

Simple Watch Guard in a Car (Continued)



Assumption/Promise to define Architectural Design Patterns

- A/P-specification

  assumption: $\forall$ t: $\neg$ ready(t) $\Rightarrow$ act_speed(t) = 0

  promise: $\forall$ t: $\neg$ doors_closed(t) $\Rightarrow$ act_speed(t) = 0

  is logically guaranteed by the simple specification

  $\forall$ t: $\neg$ doors_closed(t) $\Rightarrow$ $\neg$ ready(t)
- This assertion no longer speaks about the specification of the environment, but is a pure interface specification.
- The example shows the simplification of an A/P-specification to a plain interface assertion.

Conclusion

- The meaning of A/P specs
  - Simple conditionals/implication under what condition
- A/P specs normalized by
  - Assumption of realizability
  - Analysis of liveness and safety – transformation of A/P specs
- A/P specs for architecture design
  - From interaction assertions to A/P specs
  - From A/P specs to plain specs