Contract Theories

Albert Benveniste
B. Caillaud D. Nickovic R. Passerone J-B. Raclet
W. Damm T. Henzinger K. Larsen
A. Sangiovanni-Vincentelli

INRIA Rennes

October 17, 2016

Contracts in embedded systems design: why?

- Not enough research done by “formalists” to mathematically support the early phases of system design
- The formal methods needed differ from formal verification

Contracts in embedded systems design: why?

- Not enough research done by “formalists” to mathematically support the early phases of system design
- The formal methods needed differ from formal verification

- Contract based design has been developed by the MDE community (Model Driven Engineering, UML-related)
  - Syntax is addressed, not so much semantics
- Addressing semantics
  - The background from formal verification is needed
  - Theories have been developed by “formalists” for:

    Refinement, Interfaces, Specifications
  - Other skills are needed too: control, AI and natural languages. . .

Contracts in embedded systems design: why?

- Not enough research done by “formalists” to mathematically support the early phases of system design
- The formal methods needed differ from formal verification

- Contract based design has been developed by the MDE community (Model Driven Engineering, UML-related)
  - Syntax is addressed, not so much semantics
- Addressing semantics
  - The background from formal verification is needed
  - Theories have been developed by “formalists” for:

    Refinement, Interfaces, Specifications
  - Other skills are needed too: control, AI and natural languages. . .

- What are the requirements for contract based design?

Contracts in embedded systems design: why?

- Formalizing OEM/supplier relations: “contracts for contracts”

  Complements Legal Contracts with Technical Contracts

- Structuring requirements or specifications

  Requirements are structured into chapters/viewpoints/aspects (function, safety, performance & timing, QoS. . .)

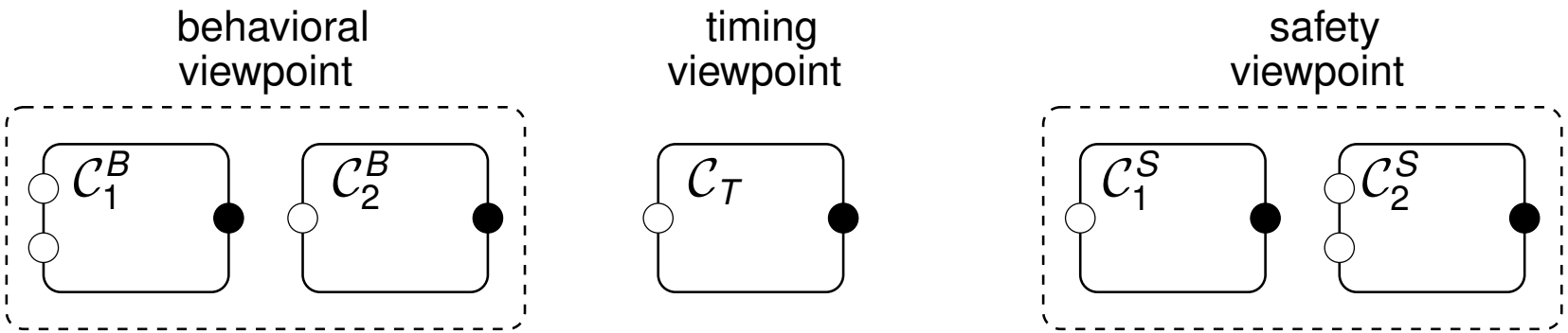
- Concurrent development at the system designer

  Different viewpoints are developed by different teams
  Weaving viewpoints must be sound and correct

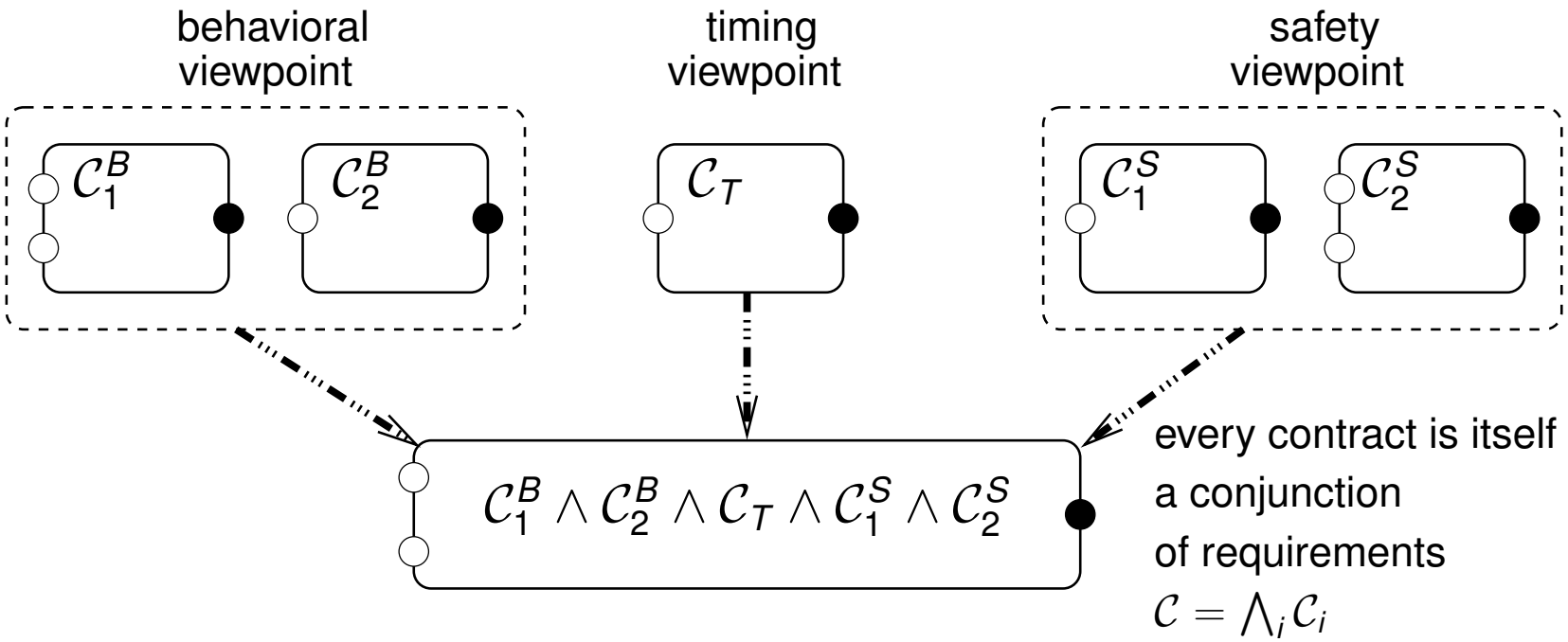
- Independent development by the suppliers

  Suppliers must be able to develop their sub-systems having all the info they need; system integration must be correct

Structuring requirements or specifications
Concurrent development

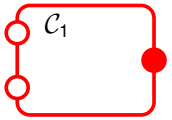


Structuring requirements or specifications
Concurrent development

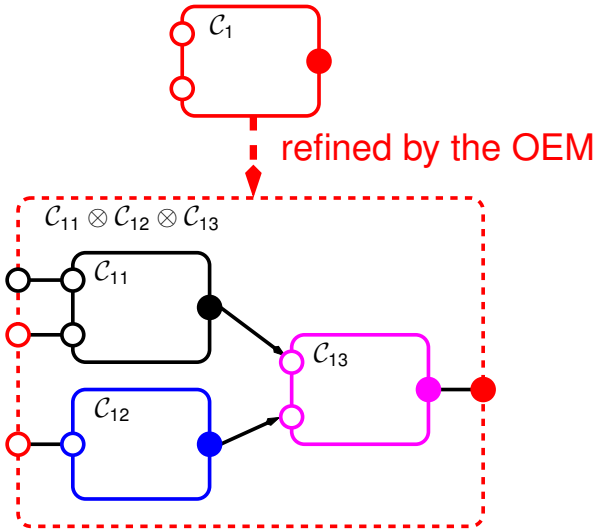


Requirements are combined by using “contract conjunction”
Viewpoints are fused by using “contract conjunction”

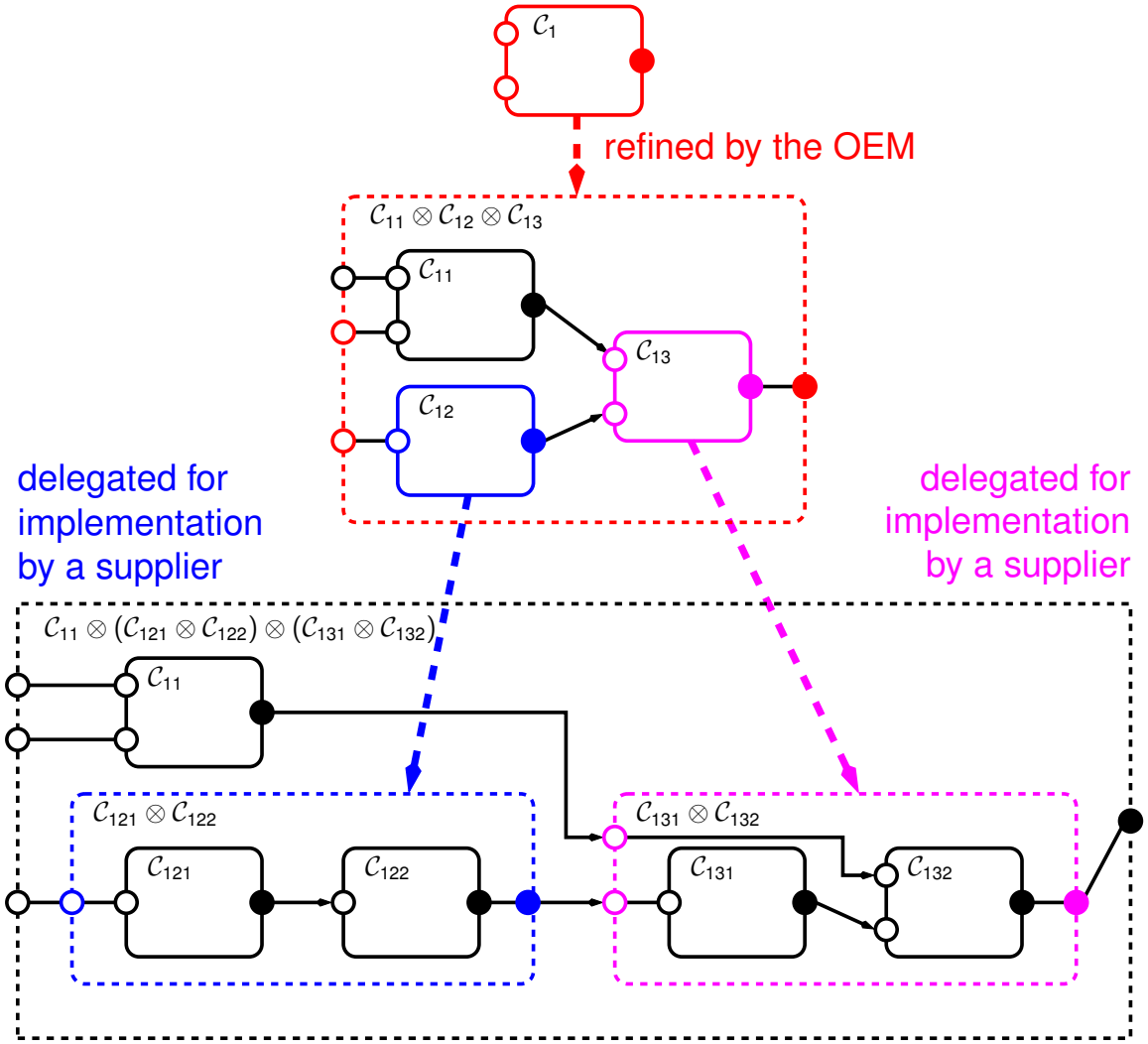
Structuring requirements or specifications
Independent development



Structuring requirements or specifications
Independent development



Structuring requirements or specifications
Independent development



"refined", "implementation", $\otimes$: new concepts

A meta-theory of contracts

Motivations for a meta-theory of contracts

A wide and diverse bibliography on scattered topics:

- Systems from Components:

  OO-programming in the 80's [B. Meyer. . .]

- Refinement
  - by simulation: in the 80's [Milner], OK for closed systems
  - by alternating simulation for open systems:
    early 90's [Abadi, Lamport, Wolper]
    late 90's [de Alfaro, Kupferman, Henzinger]

- Composition and compatibility:

  [Abadi-Lamport 1993] [de Alfaro-Henzinger 2000] [Lüttgen-Vogler2014]
  [Jonsson 2011]

- Conjunction and consistency:
  [Passerone, Raclet, Caillaud, Benveniste... 2008]

- Product lines (not discussed here): [Larsen, Nyman, Wasowski 2008]

Motivations for a meta-theory of contracts

Fact:

- Different theories have been proposed to address similar issues:
  - specifications
  - interfaces
  - contracts

  Hence the need for a

meta-theory

Goal:

- Capture the essence of the above frameworks
- Highlight their very nature
- Develop new generic tools and techniques
- Instantiate to known frameworks, hoping for new results

The meta-theory: Components and Contracts

- **Component**: actual piece of SW/HW/devices, open system
- **Environment**: context of use (a component), often unkown at design time
- Components cannot constrain their environment

The meta-theory: Components and Contracts

- **Component**: actual piece of SW/HW/devices, open system
- **Environment**: context of use (a component), often unkown at design time
- Components cannot constrain their environment

- **Contracts** are intentionally abstract
- Pinpoint responsibilities of component vs. environment

$$\text{semantics}(\mathcal{C}) = \left(\underbrace{\mathcal{E}_{\mathcal{C}}}_{\substack{\text{set of} \\ \text{environments}}} , \underbrace{\mathcal{M}_{\mathcal{C}}}_{\substack{\text{set of} \\ \text{components}}} \right)$$

The meta-theory

- We assume some primitive concepts:

Component	M	
Composition	$\times$	is partially defined, commutative and associative
Composability	$M_1 \times M_2$	being well-defined is a typing relation
Environment	E	is an environment for M iff $E \times M$ is well-defined

- On top of these primitive concepts we define
  - generic concepts and operators
  - satisfying generic properties

- How concepts, operators, and properties, are made effective
  - depends on the specific framework

The meta-theory

- Generic Relations and Operators:

Contract	sem $(\mathcal{C}) = (\mathcal{E}_c, \mathcal{M}_c)$ where $\mathcal{C} \in \mathbf{C}$: underlying class of contracts
Consistency	$\mathcal{M}_c \neq \emptyset$
Compatibility	$\mathcal{E}_c \neq \emptyset$

The meta-theory

- Generic Relations and Operators:

Contract	$\mathrm{sem}\,(\mathcal{C}) = (\mathcal{E}_{\mathcal{C}}, \mathcal{M}_{\mathcal{C}})$ where $\mathcal{C} \in \mathbf{C}$: underlying class of contracts
Consistency	$\mathcal{M}_{\mathcal{C}} \neq \emptyset$ say that $(\mathcal{C}_1, \mathcal{C}_2)$ is consistent iff $\mathcal{C}_1 \wedge \mathcal{C}_2$ is consistent
Compatibility	$\mathcal{E}_{\mathcal{C}} \neq \emptyset$
Implementation	$M \models^{\mathsf{M}} \mathcal{C}$ iff $M \in \mathcal{M}_{\mathcal{C}}$; $E \models^{\mathsf{E}} \mathcal{C}$ iff $E \in \mathcal{E}_{\mathcal{C}}$
Refinement	$\mathcal{C}' \preceq \mathcal{C}$ iff $\mathcal{E}_{\mathcal{C}'} \supseteq \mathcal{E}_{\mathcal{C}}$ and $\mathcal{M}_{\mathcal{C}'} \subseteq \mathcal{M}_{\mathcal{C}}$
Conjunction	$\mathcal{C}_1 \wedge \mathcal{C}_2 =$ GLB for $\preceq$ within $\mathbf{C}$ $\mathcal{C}_1 \vee \mathcal{C}_2 =$ LUB for $\preceq$ within $\mathbf{C}$

The meta-theory

- Generic Relations and Operators:

Contract	$\text{sem}(\mathcal{C}) = (\mathcal{E}_c, \mathcal{M}_c)$ where $\mathcal{C} \in \mathbf{C}$: underlying class of contracts
Consistency	$\mathcal{M}_c \neq \emptyset$ say that $(\mathcal{C}_1, \mathcal{C}_2)$ is consistent iff $\mathcal{C}_1 \wedge \mathcal{C}_2$ is consistent
Compatibility	$\mathcal{E}_c \neq \emptyset$ say that $(\mathcal{C}_1, \mathcal{C}_2)$ is compatible iff $\mathcal{C}_1 \otimes \mathcal{C}_2$ is compatible
Implementation	$M \models^{\mathsf{M}} \mathcal{C}$ iff $M \in \mathcal{M}_c$; $E \models^{\mathsf{E}} \mathcal{C}$ iff $E \in \mathcal{E}_c$
Refinement	$\mathcal{C}' \preceq \mathcal{C}$ iff $\mathcal{E}_{c'} \supseteq \mathcal{E}_c$ and $\mathcal{M}_{c'} \subseteq \mathcal{M}_c$
Conjunction	$\mathcal{C}_1 \wedge \mathcal{C}_2 =$ GLB for $\preceq$ within $\mathbf{C}$ $\mathcal{C}_1 \vee \mathcal{C}_2 =$ LUB for $\preceq$ within $\mathbf{C}$
Composition	$\mathcal{C}_1 \otimes \mathcal{C}_2 = \min \left\{ \mathcal{C} \;\middle\vert\; \begin{bmatrix} \forall M_1 & \models^{\mathsf{M}} & \mathcal{C}_1 \\ \forall M_2 & \models^{\mathsf{M}} & \mathcal{C}_2 \\ \forall E & \models^{\mathsf{E}} & \mathcal{C} \end{bmatrix} \Rightarrow \begin{bmatrix} M_1 \times M_2 & \models^{\mathsf{M}} & \mathcal{C} \\ E \times M_2 & \models^{\mathsf{E}} & \mathcal{C}_1 \\ E \times M_1 & \models^{\mathsf{E}} & \mathcal{C}_2 \end{bmatrix} \right\}$
Quotient	$\mathcal{C}_1 / \mathcal{C}_2 = \max\{ \mathcal{C} \in \mathbf{C} \mid \mathcal{C} \otimes \mathcal{C}_2 \preceq \mathcal{C}_1 \}$

The meta-theory

Generic Properties showing that meta-contracts meet the requirements for contract based design:

Refinement	substituability $\nearrow$ of sets of environments substituability $\searrow$ of sets of implementations
Composition	$\left.\begin{array}{r}(\mathcal{C}_1, \mathcal{C}_2) \text{ compatible} \\ \mathcal{C}'_i \preceq \mathcal{C}_i\end{array}\right\} \Rightarrow \left\{\begin{array}{l}(\mathcal{C}'_1, \mathcal{C}'_2) \text{ compatible} \\ \mathcal{C}'_1 \otimes \mathcal{C}'_2 \preceq \mathcal{C}_1 \otimes \mathcal{C}_2\end{array}\right.$ independent implementability $\mathcal{C}_1 \otimes \mathcal{C}_2 \otimes \mathcal{C}_3 \quad \preceq \quad \mathcal{C}_1 \otimes (\mathcal{C}_2 \otimes \mathcal{C}_3)$ sub-associativity; additional assumptions ensure associativity $[(\mathcal{C}_{11} \wedge \mathcal{C}_{21}) \otimes (\mathcal{C}_{12} \wedge \mathcal{C}_{22})] \quad \preceq \quad [(\mathcal{C}_{11} \otimes \mathcal{C}_{12}) \wedge (\mathcal{C}_{21} \otimes \mathcal{C}_{22})]$ sub-distributivity: sets the freedom in design processes, fusing viewpoints before/after composing sub-systems
Quotient	$\mathcal{C} \preceq \mathcal{C}_1/\mathcal{C}_2 \quad \Leftrightarrow \quad \mathcal{C} \otimes \mathcal{C}_2 \preceq \mathcal{C}_1$

Abstracting and testing

- Restrictions must hold for relations and operators on contracts to be analyzable
- Such restrictions may not hold for system models in practice
- Typical obstacles are infinite data types and functions operating on them

Abstracting and testing

- Restrictions must hold for relations and operators on contracts to be analyzable
- Such restrictions may not hold for system models in practice
- Typical obstacles are infinite data types and functions operating on them

- Two complementary ways of overcoming this consist in
  - performing abstractions
  - performing testing

- The meta-theory offers generic

  contract abstraction on top of
  abstract interpretation for components

Bibliographical note

Few attempts to develop a meta-theory. Recent papers:

- Sebastian S. Bauer, Alexandre David, Rolf Hennicker, Kim G. Larsen, Axel Legay, Ulrik Nyman, and Andrzej Wasowski. Moving from specifiations to contracts in component-based design. FASE 2012
  - Starts from an abstract notion of specification with axioms—refinement, conjunction, composition, quotient
  - Then it defines contracts as pairs (A, G) of specs
  - It establishes a link from abstract specs to modal automata
- Taolue Chen, Chris Chilton, Bengt Jonsson, Marta Z. Kwiatkowska: A Compositional Specification Theory for Component Behaviours. ESOP 2012: 148-168
  - trace based abstract specification
  - ports split into uncontrolled/controlled (or input/output)
  - assumptions involve inputs and guarantees involve outputs
  - conjunction, composition, quotient

About sub-associativity of contract composition

- Sub-associativity is sufficient to support independent development

- Associativity is guaranteed if the following property holds, for the considered framework of contracts:
  - there exists a refinement relation $\sqsubseteq$ for components, and
  - every contract $\mathcal{C}$ possesses
    - a maximal implementation $M_{\mathcal{C}}$, and
    - a minimal implementation $m_{\mathcal{C}}$,

    such that
    - every $M : m_{\mathcal{C}} \sqsubseteq M \sqsubseteq M_{\mathcal{C}}$ is an implementation of $\mathcal{C}$, and
    - min and max implementations are compositional:

$$\begin{aligned} M_{\mathcal{C}_1 \otimes \mathcal{C}_2} &= M_{\mathcal{C}_1} \times M_{\mathcal{C}_2} \\ m_{\mathcal{C}_1 \otimes \mathcal{C}_2} &= m_{\mathcal{C}_1} \times m_{\mathcal{C}_2} \end{aligned}$$

- Contract composition is a difficult operator that has been the subject of lots of errors in many publications. . .

Abstracting contracts $\mathcal{C} = (\mathcal{E}_c, \mathcal{M}_c)$

Approach:

1. Assume a Galois connection on components
2. Yields a canonical abstraction on sets of components
3. Yields a canonical abstraction for contracts

Properties:

- Consistency and Compatibility can be proved on abstractions (positive semi-decision)
- Contract abstraction is monotonic with respect to refinement
- Contract abstraction distributes over conjunction
- Contract abstraction “sub-distributes” over composition

There are obstructions to getting an abstraction with stronger properties

Abstracting contracts $\mathcal{C} = (\mathcal{E}_c, \mathcal{M}_c)$

1. Following [Cousot&Cousot], recall the notion of Galois connection:

$$\begin{array}{lcl} \alpha : (\mathcal{X}_{\mathbf{C}}, \sqsubseteq_{\mathbf{C}}) \mapsto (\mathcal{X}_{\mathbf{A}}, \sqsubseteq_{\mathbf{A}}) & : & \text{the abstraction} \\ \gamma : (\mathcal{X}_{\mathbf{A}}, \sqsubseteq_{\mathbf{A}}) \mapsto (\mathcal{X}_{\mathbf{C}}, \sqsubseteq_{\mathbf{C}}) & : & \text{the concretization} \end{array}$$

   two monotonic maps such that

$$X_{\mathbf{C}} \sqsubseteq_{\mathbf{C}} \gamma(X_{\mathbf{A}}) \iff \alpha(X_{\mathbf{C}}) \sqsubseteq_{\mathbf{A}} X_{\mathbf{A}}$$

2. From Galois connection on X's to abstractions on sets-of-X:
   - Let $\mathcal{X}^{<} \subseteq 2^{\mathcal{X}}$ collect all $\sqsubseteq$-downward closed subsets of $\mathcal{X}$
   - Equip $\mathcal{X}_{\mathbf{C}}^{<}$ and $\mathcal{X}_{\mathbf{A}}^{<}$ with their inclusion orders $\subseteq_{\mathbf{C}}$ and $\subseteq_{\mathbf{A}}$
   - Set

$$\widehat{\alpha}(\chi_{\mathbf{C}}) = \gamma^{-1}(\chi_{\mathbf{C}}) = \{X_{\mathbf{A}} \mid \gamma(X_{\mathbf{A}}) \text{ well defined and } \in \chi_{\mathbf{C}}\}$$

3. the canonical way of defining abstractions for contracts is:

$$\alpha(\mathcal{C}_{\mathbf{C}}) = \left(\widehat{\alpha}(\mathcal{E}_{c_{\mathbf{C}}}), \widehat{\alpha}(\mathcal{M}_{c_{\mathbf{C}}})\right)$$

Assume/Guarantee contracts: summary

- Component:
  - Kahn Process Network (KPN) or
  - Synchronous Transition System (STS), e.g., discrete time Simulink
- Contract: pair (Assumption, Guarantee) = (KPN,KPN) or (STS,STS)
  $\mathcal{C} = (A, G)$ defines a contract $(\mathcal{E}_{\mathcal{C}}, \mathcal{M}_{\mathcal{C}})$ following the meta-theory:

$$\begin{aligned} \mathcal{E}_{\mathcal{C}} &= \{E \mid E \subseteq A\} \\ \mathcal{M}_{\mathcal{C}} &= \{M \mid A \times M \subseteq G\} \end{aligned}$$

Assume/Guarantee contracts: summary

- Component:
  - Kahn Process Network (KPN) or
  - Synchronous Transition System (STS), e.g., discrete time Simulink
- Contract: pair (Assumption, Guarantee) = (KPN,KPN) or (STS,STS)
  $\mathcal{C} = (A, G)$ defines a contract $(\mathcal{E}_{\mathcal{C}}, \mathcal{M}_{\mathcal{C}})$ following the meta-theory:

$$\begin{aligned} \mathcal{E}_{\mathcal{C}} &= \{E \mid E \subseteq A\} \\ \mathcal{M}_{\mathcal{C}} &= \{M \mid A \times M \subseteq G\} \end{aligned}$$

- Contracts $\mathcal{C}$ and $\mathcal{C}'$ such that

$$A = A' \quad \text{and} \quad G \cup \neg A \;=\; G' \cup \neg A'$$

  are equivalent as they yield identical sets of environments and components.
  $\mathcal{C}$ can always be saturated by performing the transformation

$$(A, G) \;\mapsto\; (A, G \cup \neg A)$$

Assume/Guarantee contracts: summary

- Assuming contracts in saturated form,
  the following definitions specialize the meta-theory:

$$
\begin{aligned}
\mathcal{C}' \preceq \mathcal{C} &\equiv A' \supseteq A \text{ and } G' \subseteq G \\
\mathcal{C}_1 \wedge \mathcal{C}_2 &\equiv (A_1 \cup A_2, G_1 \cap G_2) \\
\mathcal{C}_1 \otimes \mathcal{C}_2 &\equiv ((A_1 \cap A_2) \cup \neg(G_1 \cap G_2), G_1 \cap G_2)
\end{aligned}
$$

Assume/Guarantee contracts: summary

- Assuming contracts in saturated form,
  the following definitions specialize the meta-theory:

$$
\begin{aligned}
\mathcal{C}' \preceq \mathcal{C} &\equiv A' \supseteq A \text{ and } G' \subseteq G \\
\mathcal{C}_1 \wedge \mathcal{C}_2 &\equiv (A_1 \cup A_2, G_1 \cap G_2) \\
\mathcal{C}_1 \otimes \mathcal{C}_2 &\equiv ((A_1 \cap A_2) \cup \neg(G_1 \cap G_2), G_1 \cap G_2)
\end{aligned}
$$

- No quotient exists
- Dealing with variable alphabets of variables is unsatisfactory, due to an unfortunate handling of assumptions in contract conjunction

Assume/Guarantee contracts: summary

- Assuming contracts in saturated form,
  the following definitions specialize the meta-theory:

$$
\begin{aligned}
\mathcal{C}' \preceq \mathcal{C} &\equiv A' \supseteq A \text{ and } G' \subseteq G \\
\mathcal{C}_1 \wedge \mathcal{C}_2 &\equiv (A_1 \cup A_2, G_1 \cap G_2) \\
\mathcal{C}_1 \otimes \mathcal{C}_2 &\equiv ((A_1 \cap A_2) \cup \neg(G_1 \cap G_2), G_1 \cap G_2)
\end{aligned}
$$

- No quotient exists
- Dealing with variable alphabets of variables is unsatisfactory, due to an unfortunate handling of assumptions in contract conjunction
- Complementing sets of trajectories is problematic. Fortunately, the saturation operation

$$(A, G) \mapsto G \cup \neg A$$

  can be made effective by considering an appropriate game borrowed from the *Moore Interfaces* [Chakrabarti et al. 2002]

A/G contracts: the Component Model

- To simplify we present the theory for a fixed alphabet Σ and without distinguishing input vs. output
- A/G contract theory builts on top of component models that are assertions (sets of behaviors). We can consider both
  - asynchronous Kahn Process Networks (KPN)
  - Synchronous Transition Systems (synchronous languages)
  - for both cases, parallel composition is by intersection

$$M = (\Sigma, P) \quad = \quad P \quad \text{for short since } \Sigma \text{ is fixed}$$

$$P \subseteq \begin{cases} \Sigma \mapsto \ \mathrm{Dom}^* \cup \mathrm{Dom}^\omega & \text{KPN} \\ \text{or} \\ (\Sigma \mapsto \mathrm{Dom})^* \cup (\Sigma \mapsto \mathrm{Dom})^\omega & \text{Synchronous} \end{cases}$$

$$M_1 \times M_2 = P_1 \cap P_2$$

A/G contracts: the Contracts

$\mathcal{C} = (A, G)$; A (the assumptions) and G (the guarantees) are assertions over Σ

Behaviors must be of the same kind for both components and contracts
(either both KPN or both synchronous)

A/G contracts: the Contracts

$\mathcal{C} = (A, G)$; A (the assumptions) and G (the guarantees) are assertions over Σ

Behaviors must be of the same kind for both components and contracts (either both KPN or both synchronous)

$\mathcal{C} = (A, G)$ defines a contract $(\mathcal{E}_\mathcal{C}, \mathcal{M}_\mathcal{C})$, where:

$$\begin{aligned} \mathcal{E}_\mathcal{C} &= \{E \mid E \subseteq A\} \\ \mathcal{M}_\mathcal{C} &= \{M \mid A \times M \subseteq G\} \end{aligned}$$

A/G contracts: the Contracts

$\mathcal{C} = (A, G)$; A (the assumptions) and G (the guarantees) are assertions over Σ

Behaviors must be of the same kind for both components and contracts (either both KPN or both synchronous)

$\mathcal{C} = (A, G)$ defines a contract $(\mathcal{E}_{\mathcal{C}}, \mathcal{M}_{\mathcal{C}})$, where:

$$\begin{aligned} \mathcal{E}_{\mathcal{C}} &= \{E \mid E \subseteq A\} \\ \mathcal{M}_{\mathcal{C}} &= \{M \mid A \times M \subseteq G\} \end{aligned}$$

Contracts $\mathcal{C}$ and $\mathcal{C}'$ such that

$$A = A' \quad \text{and} \quad G \cup \neg A \;=\; G' \cup \neg A'$$

are equivalent as they yield identical sets of environments and components.
$\mathcal{C}$ can always be saturated by performing the transformation

$$(A, G) \;\mapsto\; (A, G \cup \neg A)$$

This is assumed next.

A/G contracts: the Contracts

Refinement (for $\mathcal{C}_1, \mathcal{C}_2$ saturated):

$$\mathcal{C}' \preceq \mathcal{C} \text{ holds iff } \begin{cases} A' \supseteq A \\ G' \subseteq G \end{cases}$$

Composition (for $\mathcal{C}_1, \mathcal{C}_2$ saturated):

$$\begin{aligned} G &= G_1 \cap G_2 \\ A &= \max \left\{ A \;\middle|\; \begin{array}{l} A \cap G_2 \subseteq A_1 \\ A \cap G_1 \subseteq A_2 \end{array} \right\} = (A_1 \cap A_2) \cup \neg(G_1 \cap G_2) \end{aligned}$$

No quotient exists

Problem: need for complementing assertions. In fact it is enough to be able to implement the saturation operation

$$(A, G) \mapsto G \cup \neg A$$

A/G contracts with variable alphabet

Variable alphabet is dealt with using a two steps procedure

1. Equalize alphabets in both assumptions and guarantees (by existential inverse projection)
2. Reuse the theory developed for a fixed alphabet

Whereas alphabet equalization is known and works well for components (and environments), we do have a problem when extending it to contracts:

Problem: alphabet equalization for A/G-contracts is well defined but is practically inadequate when dealing with the conjunction, as it yields, for the assumptions and when alphabets are disjoint:

$$A_1 \cup A_2 = \mathit{true}$$

meaning that every environment is considered legal

A/G contracts: saturation via Moore Interfaces

Moore Interface [Chakrabarti et al. 2002]: tuple $(V^{\mathrm{in}}, V^{\mathrm{out}}; I_A, I_G; T_A, T_G)$

- Variables $V = V^{\mathrm{in}} \uplus V^{\mathrm{out}}$; V' "next" variables; finite domains
- Predicates $\underbrace{I_A : V^{\mathrm{in}}, T_A : V \cup (V^{\mathrm{in}})'}_{\text{transition system behaviors } A}$ and $\underbrace{I_G : V^{\mathrm{out}}, T_G : V \cup (V^{\mathrm{out}})'}_{\text{transition system behaviors } G}$

A Moore Interface defines an A/G-contract $\mathcal{C} = (A, G)$.

The components are the receptive ("non-blocking") input-output Moore Machines

A/G contracts: saturation via Moore Interfaces

Moore Interface [Chakrabarti et al. 2002]: tuple $(V^{\mathrm{in}}, V^{\mathrm{out}}; I_A, I_G; T_A, T_G)$

- Variables $V = V^{\mathrm{in}} \uplus V^{\mathrm{out}}$; V' "next" variables; finite domains
- Predicates $\underbrace{I_A : V^{\mathrm{in}}, T_A : V \cup (V^{\mathrm{in}})'}_{\text{transition system behaviors } A}$ and $\underbrace{I_G : V^{\mathrm{out}}, T_G : V \cup (V^{\mathrm{out}})'}_{\text{transition system behaviors } G}$

A Moore Interface defines an A/G-contract $\mathcal{C} = (A, G)$.

The components are the receptive ("non-blocking") input-output Moore Machines

Moore two-player game of $\mathcal{C}$, with players *in* and *out* [Chakrabarti et al. 2002]:

- At each round of the game, player *in* chooses new values for the V^{in} according to I_A at the first round, and then according to T_A;
- Simultaneously and independently, player *out* chooses unconstrained new values for the output variables V^{out};
- Player *out* wins if either the resulting trace is infinite and belongs to G, or if player *in* is blocked at some round while player *out* is not blocked yet.

If *out* wins, the resulting set of traces is $G \cup \neg A$: winning strategy $\rightarrow$ saturation

A/G contracts: saturation via Moore Interfaces

- The authors of [Chakrabarti et al. 2002] give an iterative algorithm whose fixpoint is a winning strategy for the Moore game
- This algorithm is effective

- We have associated a Moore game to contract saturation
- A Moore game can be directly associated to contract refinement
- A Moore game can be directly associated to contract composition

- Moore Interfaces appear as a specialization of A/G-contracts

A/G contracts: observers for testing

- σ: trace; P, Q: assertions; $\mathcal{C} = (A, G)$

$$\begin{aligned}
\beta(\sigma, P, Q) &=_{\text{def}} \begin{cases} \mathsf{T} & \text{if} \quad \sigma \in P \cup \neg Q \\ \mathsf{F} & \text{if} \quad \sigma \notin P \cup \neg Q \end{cases} \\
\mathit{EnvObs}(\mathcal{C}, E) &=_{\text{def}} \sigma_E \rightarrow \beta(\sigma_E, \Omega, A) \\
\mathit{ImpObs}(\mathcal{C}, M) &=_{\text{def}} \sigma_M \rightarrow \beta(\sigma_M, A, G)
\end{aligned}$$

- Semi-decision: observers can disprove the correctness of environments or implementations.
- Observers do not require contracts to be saturated. By construction, a contract and its saturated form possess identical observers.
- The algebra of contracts (with $\wedge, \otimes$) is reflected by an algebra for observers, so that it is enough to test sub-systems against sub-contracts. The modularity is restricted, however, as global traces must be generated and then projected to the different sub-systems.

Bibliographical note

- A/G reasoning arises in OO-programming in the late 80's [B. Meyer 1992]
  Contracts quite often deal with complex typing handled with constraints expressed on parameters (OCL)

- Formal behavioral contracts come in the early 90's in the area of compositional verification; main issue here is that of *circular reasoning* [Clarke, de Long, Mc Millan 1989]; see also [Abadi & Lamport 1993]

- A/G behavioral contracts were revisited in [Passerone et al. 2007] by SPEEDS project

- The serie of papers by Chakrabarti, Henzinger, et al. 2000-2010, on *synchronous interfaces*, are refinements of the Moore Interfaces and belong to the family of A/G-contracts.

Interface Automata: summary

- Component: deterministic and receptive input/output automaton
  - $M = (\Sigma^{\text{in}}, \Sigma^{\text{out}}, Q \ni q_0, \rightarrow)$ with usual parallel composition $M_1 \times M_2$
- Contract: deterministic (possibly non receptive) input/output automaton
  - $\mathcal{C} = (\Sigma^{\text{in}}, \Sigma^{\text{out}}, Q, q_0, \rightarrow)$
  - $\mathcal{C}$ defines a contract $(\mathcal{E}_{\mathcal{C}}, \mathcal{M}_{\mathcal{C}})$ following the meta-theory:
    - $\mathcal{E}_{\mathcal{C}}$ collects all E not proposing as output an action that is refused by $\mathcal{C}$ in the composition $E \times \mathcal{C}$
    - $\mathcal{M}_{\mathcal{C}}$ collects all M such that, $\forall E \in \mathcal{E}_{\mathcal{C}}$, $E \times \mathcal{C}$ simulates $E \times M$

Interface Automata: summary

- Refinement, as defined by alternating simulation, specializes the meta-theory;

  Conjunction is difficult, even for a fixed alphabet of actions

- Parallel composition $\otimes$, together with its notion of compatibility, exist and specialize the meta-theory:

  1. start from $\mathcal{C}_1 \times \mathcal{C}_2$, seen as i/o-automata
  2. illegal pair (q_1, q_2) may exist, where (informally) "$\mathcal{M}_{\mathcal{C}_1} \not\subseteq \mathcal{E}_{\mathcal{C}_2}$"
  3. pruning away illegal pairs until fixpoint yields $\mathcal{C}_1 \otimes \mathcal{C}_2$

Interface Automata

We only present the theory for a fixed alphabet Σ
Alphabet equalization is as for A/G contracts (with the same problems)

Interface theories built on top of (a relaxed version of) Nancy Lynch i/o-automata

Deterministic Input/Output Automaton: $M = (\Sigma^{\mathrm{in}}, \Sigma^{\mathrm{out}}, Q, q_0, \rightarrow)$, where:

$\Sigma = \Sigma^{\mathrm{in}} \cup \Sigma^{\mathrm{out}}$	:	alphabet of actions
$q_0 \in Q$	:	initial state
$q \xrightarrow{\alpha} q'$	:	deterministic transition relation $\left.\begin{array}{l} q \xrightarrow{\alpha} q_1 \\ q \xrightarrow{\alpha} q_2 \end{array}\right\} \Rightarrow q_1 = q_2$

Interface Automata: the Component Model

- Component: an i/o-automaton that is receptive:

$$\forall q \in Q, \forall \alpha \in \Sigma^{\mathrm{in}}, \exists q' : \; q \xrightarrow{\alpha} q'$$

Interface Automata: the Component Model

- Component: an i/o-automaton that is receptive:

$$\forall q \in Q, \forall \alpha \in \Sigma^{\mathrm{in}}, \exists q' : \; q \xrightarrow{\alpha} q'$$

- Parallel composition: well defined only if $\Sigma_1^{\mathrm{out}} \cap \Sigma_2^{\mathrm{out}} = \emptyset$; the two components synchronize on their actions

$$M_1 \times M_2 : \begin{cases} \Sigma^{\mathrm{out}} = \Sigma_1^{\mathrm{out}} \cup \Sigma_2^{\mathrm{out}} \\ Q = Q_1 \times Q_2 \\ q_0 = (q_{1,0}, q_{2,0}) \\ (q_1, q_2) \xrightarrow{\alpha} (q_1', q_2') \quad \text{iff} \quad q_i \xrightarrow{\alpha}_i q_i', i = 1, 2 \end{cases}$$

Interface Automata: the Component Model

- Component: an i/o-automaton that is receptive:

$$\forall q \in Q, \forall \alpha \in \Sigma^{\text{in}}, \exists q' : \; q \xrightarrow{\alpha} q'$$

- Parallel composition: well defined only if $\Sigma_1^{\text{out}} \cap \Sigma_2^{\text{out}} = \emptyset$; the two components synchronize on their actions

$$M_1 \times M_2 : \begin{cases} \Sigma^{\text{out}} = \Sigma_1^{\text{out}} \cup \Sigma_2^{\text{out}} \\ Q = Q_1 \times Q_2 \\ q_0 = (q_{1,0}, q_{2,0}) \\ (q_1, q_2) \xrightarrow{\alpha} (q_1', q_2') \quad \text{iff} \quad q_i \xrightarrow{\alpha}_i q_i', i = 1, 2 \end{cases}$$

- Simulation: For $q_i \in Q_i$, say that $q_2 \leq q_1$ if

$$\forall \alpha, q_2' \text{ such that } q_2 \xrightarrow{\alpha}_2 q_2' \quad \Longrightarrow \quad \begin{cases} q_1 \xrightarrow{\alpha}_1 q_1' \\ \text{and } q_2' \leq q_1' \end{cases}$$

Say that $M_2 \leq M_2$ if $q_{2,0} \leq q_{1,0}$

Interface Automata: Contracts

Interface Automaton $\mathcal{C} = (\Sigma^{\text{in}}, \Sigma^{\text{out}}, Q, q_0, \rightarrow)$

- $\Sigma^{\text{in}}, \Sigma^{\text{out}}, Q, \rightarrow$ as in i/o-automata
- We do not request $q_0 \in Q$; thus, $q_0 \notin Q$ is also a possibility

When $q_0 \in Q$, $\mathcal{C}$ defines a contract by fixing a pair $(\mathcal{E}_{\mathcal{C}}, \mathcal{M}_{\mathcal{C}})$, where:

- $\mathcal{E}_{\mathcal{C}}$ collects all E such that:
  1. $\Sigma_E^{\text{in}} = \Sigma^{\text{out}}$ and $\Sigma_E^{\text{out}} = \Sigma^{\text{in}}$. Thus, E and $\mathcal{C}$, seen as i/o-automata, are composable
  2. E is $\mathcal{C}$-compliant:

$$\left.\begin{array}{r} \forall \alpha \in \Sigma_E^{\text{out}}, q_E \xrightarrow{\alpha}_E \\ \forall (q_E, q) \text{ reachable in } E \times \mathcal{C} \end{array}\right\} \Rightarrow (q_E, q) \xrightarrow{\alpha}_{E \times \mathcal{C}} \text{ holds}$$

- $\mathcal{M}_{\mathcal{C}}$ collects all M such that $\forall E \in \mathcal{E}_{\mathcal{C}}$, $\mathcal{C}$ simulates $E \times M$ seen as i/o-automata

Lemma: $q_0 \in Q$ iff $\mathcal{C}$ is both consistent ($\mathcal{M}_{\mathcal{C}} \neq \emptyset$) and compatible ($\mathcal{E}_{\mathcal{C}} \neq \emptyset$)

Interface Automata: Contracts

Refinement is by alternating simulation: for $\mathcal{C}_1, \mathcal{C}_2$ two contracts such that $\Sigma_1^{\mathrm{out}} = \Sigma_2^{\mathrm{out}}$, say that $q_2 \preceq q_1$ if

$$\forall \alpha \in \Sigma_2^{\mathrm{out}}, \forall q_2' \text{ s.t. } q_2 \xrightarrow{\alpha}_2 q_2' \quad \Longrightarrow \quad \begin{cases} q_1 \xrightarrow{\alpha}_1 q_1' \\ q_2' \preceq q_1' \end{cases}$$

$$\forall \alpha \in \Sigma_1^{\mathrm{in}}, \forall q_1' \text{ s.t. } q_1 \xrightarrow{\alpha}_1 q_1' \quad \Longrightarrow \quad \begin{cases} q_2 \xrightarrow{\alpha}_2 q_2' \\ q_2' \preceq q_1' \end{cases}$$

Say that $\mathcal{C}_2 \preceq \mathcal{C}_1$ if $q_{2,0} \preceq q_{1,0}$.

Conjunction exists but is difficult.

Interface Automata: Contracts

Parallel Composition for $\mathcal{C}_1, \mathcal{C}_2$ two contracts such that $\Sigma_1^{\mathrm{out}} \cap \Sigma_2^{\mathrm{out}} = \emptyset$:

1. Build $\mathcal{C}_1 \times \mathcal{C}_2$ as an i/o-automaton and try it as the composition

2. By the meta-theory we must have

$$E \models^{\mathsf{E}} \mathcal{C} \quad \Longrightarrow \quad \left[E \times M_2 \models^{\mathsf{E}} \mathcal{C}_1 \text{ and } E \times M_1 \models^{\mathsf{E}} \mathcal{C}_2 \right]$$

   which requires $\forall \alpha \in \Sigma_i^{\mathrm{out}}$: $q_i \xrightarrow{\alpha}_i \quad \Longrightarrow \quad (q_1, q_2) \xrightarrow{\alpha}_{\mathcal{C}_2 \times \mathcal{C}_1}$

   (q_1, q_2) not satisfying this is called illegal and must be pruned away

3. Perform this recursively until fixpoint $\mathcal{C}_1 \otimes \mathcal{C}_2$; the resulting Q can be empty:

$$\begin{array}{lcl} Q \text{ empty} & \Longleftrightarrow & (\mathcal{C}_1, \mathcal{C}_2) \text{ incompatible} \\ Q \text{ nonempty} & \Longleftrightarrow & (\mathcal{C}_1, \mathcal{C}_2) \text{ compatible} \end{array}$$

Bibliographical note

[de Alfaro Henzinger], several papers in the early 2000's

- Composition and compatibility
- Refinement by alternating simulation
- State based models and Variable based models
- Conjunction (called shared refinement by the authors) is more delicate...

Modal Interfaces: Summary

- Component: deterministic and receptive input/output automaton
  - $M = (\Sigma^{\text{in}}, \Sigma^{\text{out}}, Q \ni q_0, \rightarrow)$ with usual parallel composition $M_1 \times M_2$

  Alike for Interface Automata

- Modal Interface: $\mathcal{C} = (\Sigma^{\text{in}}, \Sigma^{\text{out}}, Q, q_0, \mathbb{T}, \underbrace{\rightarrow}_{\text{must}}, \underbrace{\dashrightarrow}_{\text{may}})$ such that $\rightarrow \subseteq \dashrightarrow$
  - $\mathcal{C}$ is *inconsistent* if $q_0 \notin Q$ and *incompatible* if $q_0 = \mathbb{T}$.
  - Otherwise, $\mathcal{C}$ yields $(\mathcal{C}^{must}, \mathcal{C}^{may})$, two non-receptive i/o-automata and defines a contract $(\mathcal{E}_{\mathcal{C}}, \mathcal{M}_{\mathcal{C}})$ following the meta-theory:
    - $\mathcal{E}_{\mathcal{C}}$ collects all E not proposing as output an action that is refused by $\mathcal{C}^{must}$ and not making $(q, \mathbb{T})$ reachable in $E \times \mathcal{C}^{must}$
    - $\mathcal{M}_{\mathcal{C}}$ collects all M such that, $\forall E \in \mathcal{E}_{\mathcal{C}}$, $E \times \mathcal{C}^{may}$ simulates $E \times M$ and $E \times M$ simulates $E \times \mathcal{C}^{must}$
  - Implementations of $\mathcal{C}$ (i.e., elements of $\mathcal{M}_{\mathcal{C}}$) are in bijection with *models* of $\mathcal{C}$, seen as a modal interface.

Modal Interfaces: Summary

- Modal Refinement $\mathcal{C}_2 \preceq \mathcal{C}_1$, defined by

$$\dashrightarrow_2 \subseteq \dashrightarrow_1 \quad \text{and} \quad \longrightarrow_1 \subseteq \longrightarrow_2$$

  specializes the meta-theory;

  Conjunction follows by pruning illegal pairs of states for consistency

- Parallel composition $\otimes$, together with its notion of compatibility, exist. Quotient exists; all specialize the meta-theory

- Variable alphabets are handled by using different alphabet equalizations
  - for $\wedge$: adding *may* self-loops, and
  - for $\otimes$: adding *may*+*must* self-loops

Bibliographical note

- Modal specifications and Modal automata
  - Hennesy in the 80's
  - Kim Larsen 1987
  - introducing variability in design at low cost

- Revitalized in the late 2000's for use as interface models
  - Kim Larsen, Nyman, Wasowski: modal automata (role of determinism in the complexity of the theory), use for product lines
  - Raclet, Caillaud, Benveniste: modal interfaces, full fledged theory dealing with variable alphabets; correction to a mistake in compatibility
  - Variations, more results in the recent years by B. Jonsson et al. and by G. Lüttgen and W. Vogler.

- *A complex theory*: Plenty of bugs on the difficult notions of parallel composition and quotient, successively corrected by different authors, and so on. Hopefully, the recent papers by G. Lüttgen and W. Vogler close the saga.

Details

See paper

Requirement Engineering: the issues

- Requirements are used to specify
  - the overall system
  - the technical relations in a supplier chain
- Typically expressed as Doors sheets
- The naive interpretation is that requirements are combined via conjunction (“all requirements must be met”)
- Is this the right way?

Requirement Engineering: the issues

- Requirements are used to specify
  - the overall system
  - the technical relations in a supplier chain
- Typically expressed as Doors sheets
- Specify the properties guaranteed by the system

$$\begin{array}{c|c|c|c|c} G_1 & \dots & G_k & \dots & G_K \\ \dots & \dots & \{A_{k,1}, \dots, A_{k,\ell_k}\} & \dots & \dots \end{array}$$

  under assumptions that are often left hidden or implicit (!!)

- Hence a requirement is formalized as the entailment

$$\{A_{k,1}, \dots, A_{k,\ell_k}\} \implies G_k$$

Requirement Engineering: the issues

- Requirements are used to specify
  - the overall system
  - the technical relations in a supplier chain
- Typically expressed as Doors sheets
- Specify the properties guaranteed by the system

$$\begin{array}{c|c|c|c|c} G_1 & \dots & G_k & \dots & G_K \\ \dots & \dots & \{A_{k,1}, \dots, A_{k,\ell_k}\} & \dots & \dots \end{array}$$

  under assumptions that are often left hidden or implicit (!!)

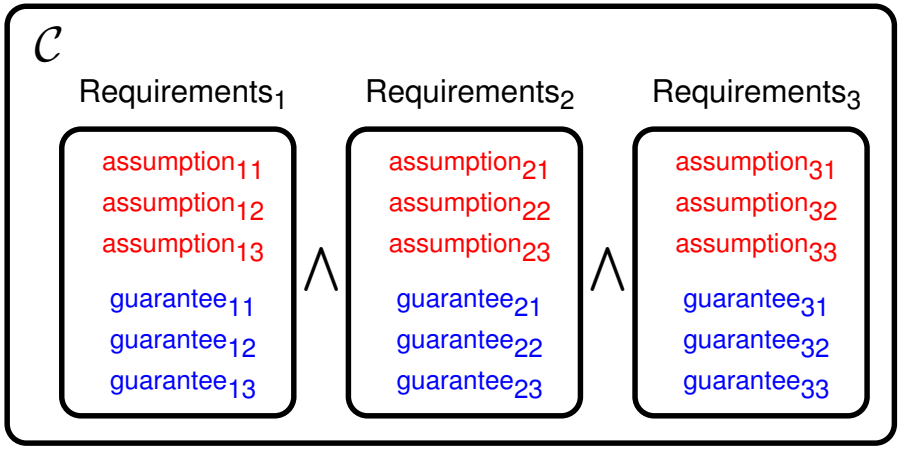
- Hence a requirement is formalized as the entailment

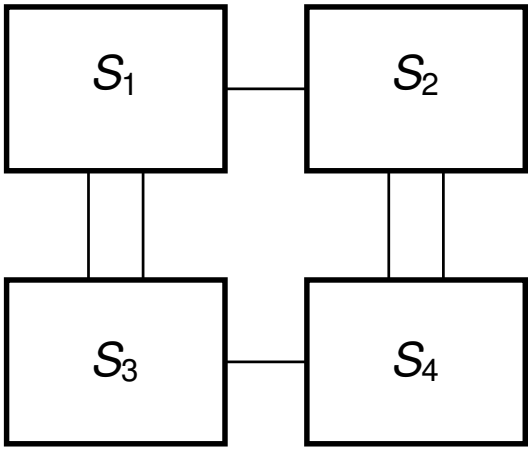
$$\{A_{k,1}, \dots, A_{k,\ell_k}\} \quad \Longrightarrow \quad G_k$$

- This leads to considering a requirements document as a contract:

$$\bigwedge_{k=1}^{K} \mathcal{C}_k, \text{ where } \mathcal{C}_k = (A_{k,1} \cap \dots \cap A_{k,\ell_k}\ ,\ G_k)$$

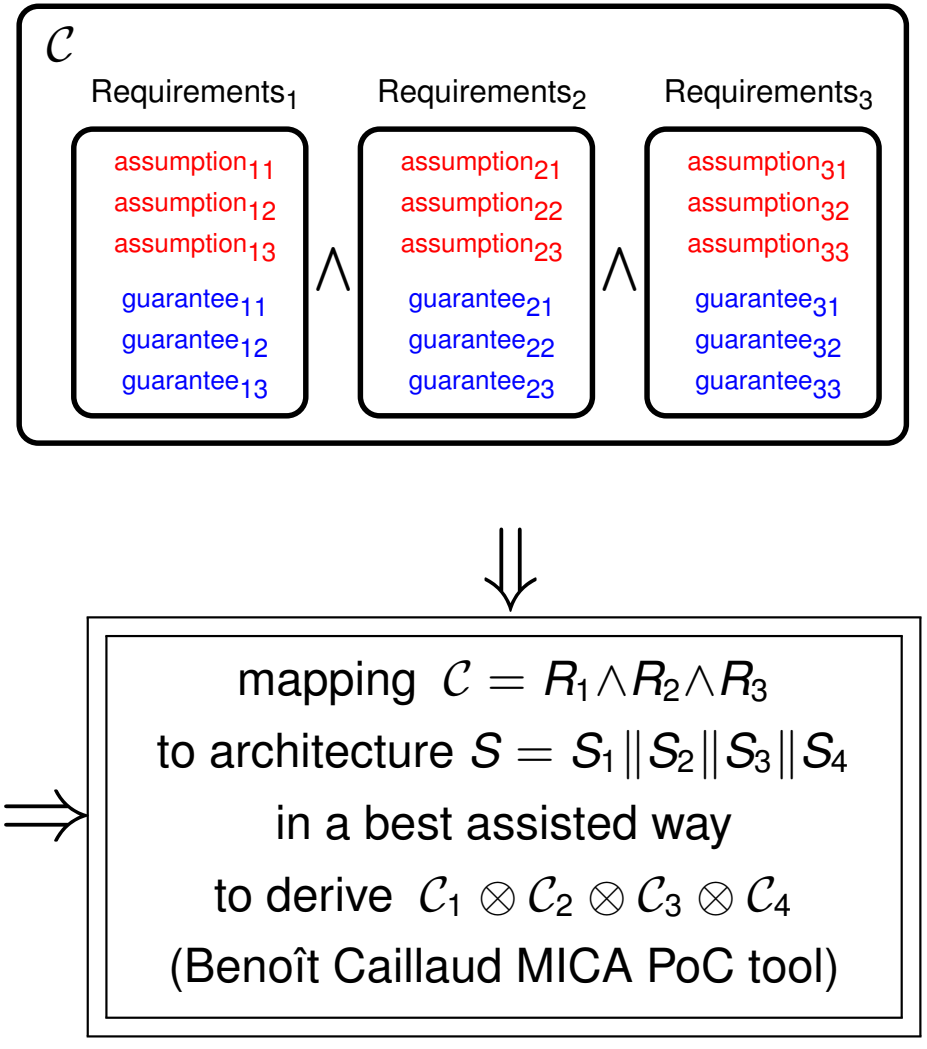
Generating requirements for sub-contractors





system architecture (SysML)

Generating requirements for sub-contractors



system architecture (SysML)

The Parking Garage example

Top-level specification: assumptions & guarantees

viewpoint **gate**(x) where $x \in$ {entry, exit}

- $R_{g.1}(x)$: "vehicles shall not pass when x_gate is closed",
- $R_{g.2}(x)$: after ?vehicle_pass ?vehicle_pass is forbidden
- $R_{g.3}$: after !x_gate_open !x_gate_open is forbidden and after !x_gate_close !x_gate_close is forbidden

viewpoint **payment**

- $R_{p.1}$: "user inserts a coin every time a ticket is inserted and only then"
- $R_{p.2}$: "user may insert a ticket only initially or after an exit ticket has been issued"
- $R_{p.3}$: "exit ticket is issued after ticket is inserted and payment is made and only then"

viewpoint **supervisor**

- $R_{g.1}$(entry)
- $R_{g.1}$(exit)
- $R_{g.2}$(entry)
- $R_{g.2}$(exit)
- $R_{s.1}$: initially and after !entry_gate close !entry_gate open is forbidden
- $R_{s.2}$: after !ticket_issued !entry_gate open must be enabled
- $R_{s.3}$: "at most one ticket is issued per vehicle entering the parking and tickets can be issued only if requested and ticket is issued only if the parking is not full"
- $R_{s.4}$: "when the entry gate is closed, the entry gate may not open unless a ticket has been issued"
- $R_{s.5}$: "the entry gate must open when a ticket is issued"
- $R_{s.6}$: "exit gate must open after an exit ticket is inserted and only then"
- $R_{s.7}$: "exit gate closes only after vehicle has exited parking"

Each requirement is translated to a Modal Interface

The different Modal Interfaces are combined by using conjunction $\Longrightarrow$ viewpoints

The different viewpoints are combined using conjunction as well

The Parking Garage example

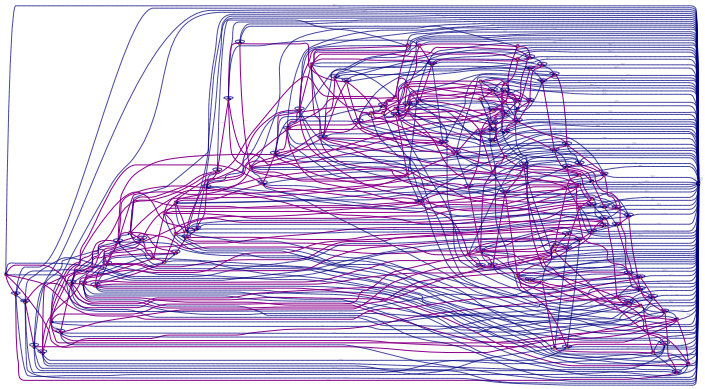
Top-level specification: $\mathcal{C} = \mathcal{C}_{\textbf{entry\_gate}} \wedge \mathcal{C}_{\textbf{exit\_gate}} \wedge \mathcal{C}_{\textbf{payment}} \wedge \mathcal{C}_{\textbf{supervisor}}$

viewpoint **entry_gate** and **exit_gate**
viewpoint **payment**
viewpoint **supervisor**

$\mathrm{R}_{\mathrm{g}.1}$(entry)
$\mathrm{R}_{\mathrm{g}.1}$(exit)
$\mathrm{R}_{\mathrm{g}.2}$(entry)
$\mathrm{R}_{\mathrm{g}.2}$(exit)
$\mathrm{R}_{\mathrm{s}.1}$: initially and after !entry_gate close !entry_gate open is forbidden
$\mathrm{R}_{\mathrm{s}.2}$: after !ticket_issued !entry_gate open must be enabled
$\mathrm{R}_{\mathrm{s}.3}$: “at most one ticket is issued per vehicle entering the parking and tickets can be issued only if requested and ticket is issued only if the parking is not full”
$\mathrm{R}_{\mathrm{s}.4}$: “when the entry gate is closed, the entry gate may not open unless a ticket has been issued”
$\mathrm{R}_{\mathrm{s}.5}$: “the entry gate must open when a ticket is issued”
$\mathrm{R}_{\mathrm{s}.6}$: “exit gate must open after an exit ticket is inserted and only then”
$\mathrm{R}_{\mathrm{s}.7}$: “exit gate closes only after vehicle has exited parking”

The supervisor as a Modal Interface

$\mathcal{C}_{\textbf{supervisor}} =$

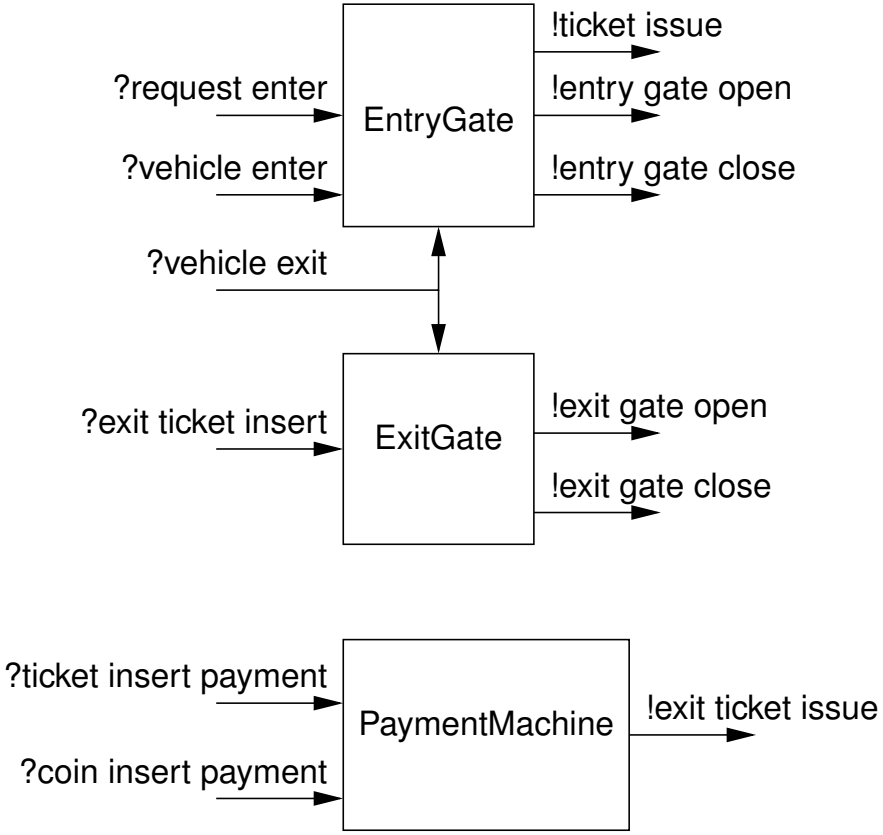


The Parking Garage example

Architecture for sub-contracting $\mathcal{C}$ as a $\otimes$-composition of sub-systems

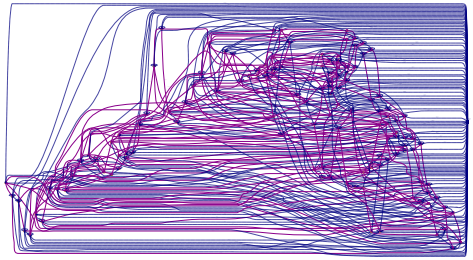
- this is the duty of the designer
- note the change in architecture: supervision performed by the entry gate

$$\mathcal{C} = \begin{bmatrix} & \mathcal{C}_{\textbf{entry\_gate}} \\ \wedge & \mathcal{C}_{\textbf{exit\_gate}} \\ \wedge & \mathcal{C}_{\textbf{payment}} \\ \wedge & \mathcal{C}_{\textbf{supervisor}} \end{bmatrix}$$

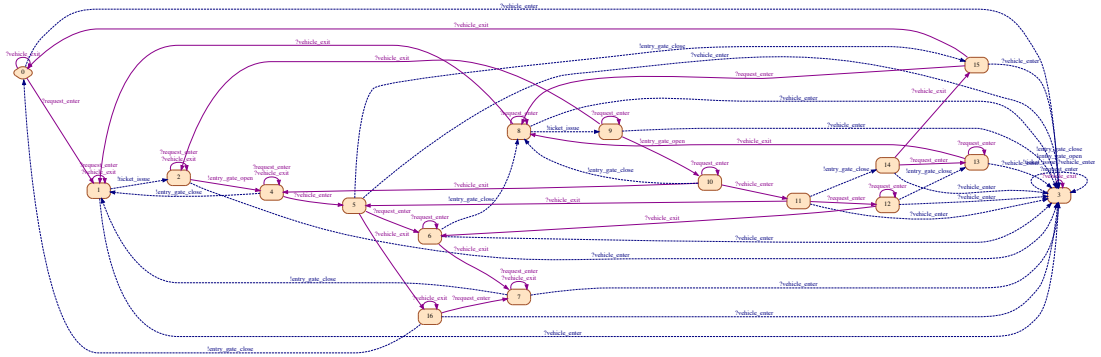


The Parking Garage example

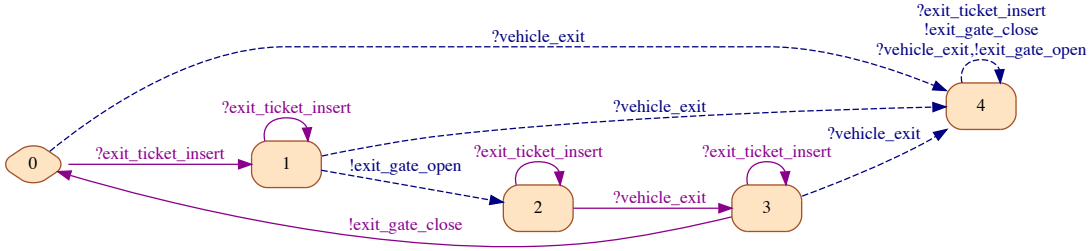
The following $\otimes$-decomposition of $\mathcal{C}$ into three sub-contracts was automatically generated (note the reduction in size)



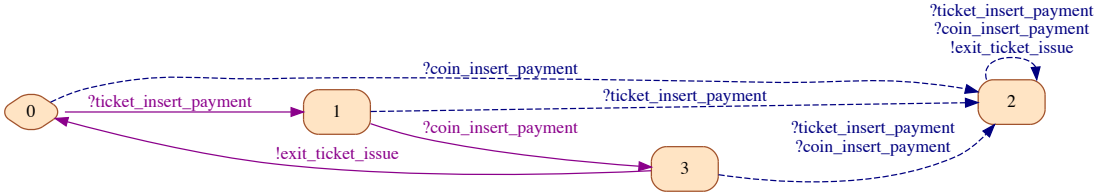
$\mathcal{C}_{\textbf{supervisor}}$



$\mathcal{C}_{\textbf{entry gate}}$



$\mathcal{C}_{\textbf{exit gate}}$



$\mathcal{C}_{\textbf{payment}}$

Translating Assumptions and Guarantees into Modal Interfaces

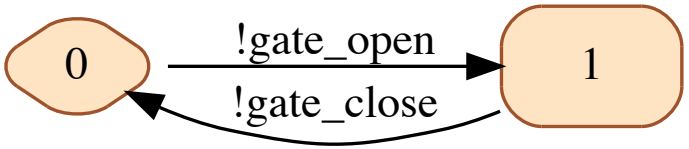
Top-level specification

gate(x) where $x \in$ {entry, exit}

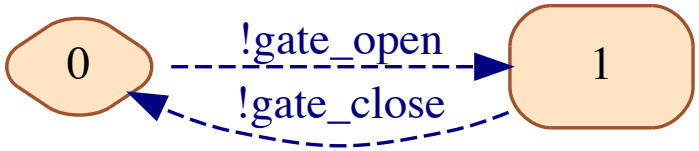
$\mathrm{R}_{\mathrm{g}.1}(x)$: "vehicles shall not pass when x_gate is closed",
$\mathrm{R}_{\mathrm{g}.2}(x)$: after ?vehicle_pass ?vehicle_pass is forbidden
$\mathrm{R}_{\mathrm{g}.3}$: after !x_gate_open !x_gate_open is forbidden and after !x_gate_close !x_gate_close is forbidden

Translating the guarantees:

$\mathrm{R}_{\mathrm{g}.3}$ as an i/o-automaton:



$\mathrm{R}_{\mathrm{g}.3}$ as a Modal Interface:



Note the *may* transitions for outputs

Translating Assumptions and Guarantees into Modal Interfaces

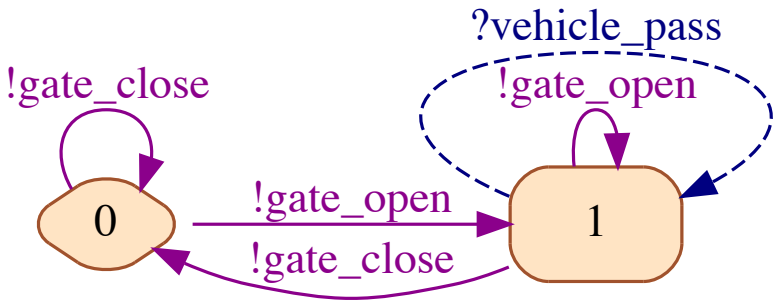
Top-level specification

gate(x) where $x \in$ {entry, exit}

$\mathrm{R}_{\mathrm{g}.1}(x)$: "vehicles shall not pass when x_gate is closed",
$\mathrm{R}_{\mathrm{g}.2}(x)$: after ?vehicle_pass ?vehicle_pass is forbidden
$\mathrm{R}_{\mathrm{g}.3}$: after !x_gate_open !x_gate_open is forbidden and after !x_gate_close !x_gate_close is forbidden

Translating the assumptions:

$\mathrm{R}_{\mathrm{g}.1}(x)$:



Note the *must* transitions for outputs and the *may* transitions for inputs

Finally the contract for the gate viewpoint is:

$$\boxed{\mathcal{C}_{\text{gate}} = ((\mathrm{R}_{\mathrm{g}.1} \wedge \mathrm{R}_{\mathrm{g}.2}) \otimes \mathrm{R}_{\mathrm{g}.3})/(\mathrm{R}_{\mathrm{g}.1} \wedge \mathrm{R}_{\mathrm{g}.2})}$$

Concluding Remarks

- Contracts: large system design by distributed OEM/supplier chains

Concluding Remarks

- Contracts: large system design by distributed OEM/supplier chains
- Contracts support both formal and semi-formal use:
  - formal: possible for specific contract formalisms
  - semi-formal: manual “local reasoning” $\longrightarrow$ system-wide properties

Concluding Remarks

- Contracts: large system design by distributed OEM/supplier chains
- Contracts support both formal and semi-formal use:
  - formal: possible for specific contract formalisms
  - semi-formal: manual “local reasoning” $\longrightarrow$ system-wide properties
- Extending the formal scope of contracts:
  - abstractions
  - observers

Concluding Remarks

- Contracts: large system design by distributed OEM/supplier chains
- Contracts support both formal and semi-formal use:
  - formal: possible for specific contract formalisms
  - semi-formal: manual “local reasoning” $\longrightarrow$ system-wide properties
- Extending the formal scope of contracts:
  - abstractions
  - observers
- The meta-theory clarifies the unique features
  - of contract-based reasoning, versus
  - other techniques of compositional reasoning
- Meta-theory specializes to various formalisms (more than shown)

More...

- Use of Modal Interfaces for the separate compilation of multiple-clocked synchronous programs, showing that contracts yield useful and non trivial theories of interfaces [Benven. & al. 2012]

- Perspective: meta-theory to support heterogeneity

- Perspective: synchronizing safety with {function+physics}:
  - safety analysis: abstract reliability or fault tree models
  - {function+physics+faults}: too complex for being analysable

    $\Longrightarrow$ for use as observers for safety analysis models