Analysis, Synthesis and Control of Concurrent Systems

Benoît Caillaud

Rennes, 23 March 2011
Outline

1. **Introduction**
2. **Synthesis of Communicating Systems through PN Synthesis**
   - Petri net synthesis
   - Application
3. **Correct-by-construction asynchronous implementation of modular synchronous specifications**
   - Introduction
   - Model and problem statement
   - Sufficient conditions
4. **Interface Theories**
   - Anatomy of an Interface Theory
   - Modal Interfaces
5. **Current and future research directions**
Context: Embedded Software Design

- Control design (Simulink)
- Embedded electronics architecture (AADL)
- Safety and reliability
- Mapping data to wires and functions to ECU
- Co-modeling & simu of functions and computing infrastructure
- Testing V&V
- System integration
- Migrating from virtual to real
- Validation
- Orchestration
- Versioning
- Linking
- Design flow
- Sub-systems
- (AUTOSAR platform)
Safety critical: Loss of attitude reference in IMC leads to unrecoverable aircraft attitude in less than 60s

Quite complex: \( \approx 3000 \) requirements

Considerable effort: 200 in-house engineers and \( \approx 10 \) subcontractors for software development and testing
Requirements: Structure, Nature

- Structured into viewpoints: functional, performance, resource consumption, reliability, ...
- Oblivious to the implementation architecture:
  - High-level functional requirements, implementation architecture often left to designers
  - Specification paradigm (automata, reactive synchronous, sequence diagrams, ...) ≠ Implementation paradigm (distributed, asynchronous communication, ...)
- What designers need:
  - Theory to bridge the gap between specification and implementation paradigms
  - Correct-by-construction computer-assisted methods requirements → implementations
  - Scalable compositional reasoning methods to cope with system complexity
Contributions detailed in this talk

1. Synthesis of asynchronous communicating systems: Petri net synthesis, linear algebra
2. Synthesis of globally asynchronous, locally synchronous systems (GALS) from synchronous specifications
3. Compositional reasoning and contract-based design with modal interfaces
Outline

1. Introduction

2. Synthesis of Communicating Systems through PN Synthesis
   - Petri net synthesis
   - Application

3. Correct-by-construction asynchronous implementation of modular synchronous specifications
   - Introduction
   - Model and problem statement
   - Sufficient conditions

4. Interface Theories
   - Anatomy of an Interface Theory
   - Modal Interfaces

5. Current and future research directions
Context: Computer Assisted Design of Asynchronous Communicating Systems

Problem
- Given a specification = behavior + architecture,
- Construct a network of communicating automata realizing the specification:
  1. Matches the specified architecture
  2. Branching bisimilar to the specified behavior

Approach
- Pragmatic, semi-automated, yields efficient communication policies, polynomial complexity
- Correct by construction
- Communication is hidden from the designer
- Manual design steps: Checkable refinements of the specification
PN synthesis in a nutshell

Net synthesis problem
Given a finite transition system $A$, decide whether exists a net $N$ such that $N^* \simeq A$. 
PN synthesis in a nutshell

- Net = union of 1-place nets
- Region = 1-place net satisfying:

\[
\begin{align*}
    & m, \cdot s, s^\bullet, \ldots \cdot d, \ d^\bullet \geq 0 \\
    & m \geq \cdot s \\
    & m + s^\bullet - \cdot s \geq \cdot d \\
    & \vdots \\
    & m + s^\bullet - \cdot s + d^\bullet - \cdot d \geq \cdot a \\
    & s^\bullet - \cdot s + r^\bullet - \cdot r = 0
\end{align*}
\]
PN synthesis in a nutshell

- **Net** = union of 1-place nets
- **State separation**: states 0 and 1

\[
\begin{align*}
 m, \cdot s, s^\cdot, \ldots \cdot d, d^\cdot & \geq 0 \\
 m & \geq \cdot s \\
 m + s^\cdot - \cdot s & \geq \cdot d \\
 & \vdots \\
 m + s^\cdot - \cdot s + d^\cdot - \cdot d & \geq \cdot a \\
 s^\cdot - \cdot s + r^\cdot - \cdot r & = 0 \\
 s^\cdot - \cdot s & \neq 0
\end{align*}
\]
PN synthesis in a nutshell

Net = union of 1-place nets
State separation: states 0 and 2

\[
\begin{align*}
m, s, s', \ldots d, d' & \geq 0 \\
n & \geq s \\
m + s' - s & \geq d \\
\vdots \\
m + s' - s + d' - d & \geq a \\
s' - s + r' - r & = 0 \\
d' - d & \neq 0
\end{align*}
\]
PN synthesis in a nutshell

- Net = union of 1-place nets
- State separation: ...
- Event-state separation: event a in state 2

\[
\begin{align*}
\begin{cases}
  m, s, s, \ldots \cdot d, d \geq 0 \\
m \geq s \\
m + s \cdot s + d \cdot d \geq a \\
\vdots \\
m + s \cdot s + r \cdot r = 0 \\
m + d \cdot d < a
\end{cases}
\end{align*}
\]

Solved in polynomial time (linear algebra)
PN synthesis in a nutshell

- Net = union of 1-place nets
- State separation: ...
- Event-state separation: event r in state 3

\[
\begin{align*}
&m, s^\bullet, \ldots \cdot d, d^\bullet \geq 0 \\
&m \geq s^\bullet \\
&m + s^\bullet - s \geq d^\bullet \\
&\vdots \\
&m + s^\bullet - s + d^\bullet - d \geq a^\bullet \\
&s^\bullet - s + r^\bullet - r = 0 \\
&m + s^\bullet - s + d^\bullet - d < r^\bullet
\end{align*}
\]
PN synthesis in a nutshell

- Net = union of 1-place nets
- State separation: ...
- Event-state separation: ...
- Solved in polynomial time (linear algebra)
- SYNET tool: PN synthesis wrt. graph isomorphism, language equality, distributable nets, ...
Distributable nets

- Distributable net = Petri net + architecture
Distributable nets

- Distributable net = Petri net + architecture
- Syntactic class of nets with no distributed conflict
- Admits trivial asynchronous distributed implementations
Distributable nets

- Distributable net = Petri net + architecture
- Syntactic class of nets with no distributed conflict
- Admits trivial asynchronous distributed implementations
- Distributable net synthesis: Add constraints of the form
  \[ t = 0 \] for every non-local transition}
Design Process Examplified

Specification = Behavior + Architecture
Design Process Examplified

- Specification = Behavior + Architecture
- Separation failure state 2, event $a$
Design Process Examplified

- Specification = Behavior + Architecture
- Separation failure state 2, event a
- Refinement of event a into $\tau.a$
Design Process Examplified

- Specification = Behavior + Architecture
- Separation failure state 2, event a
- Refinement of event a into $\tau.a$
- Synthesized distributable net
Design Process Examplified

- Specification = Behavior + Architecture
- Separation failure state 2, event $a$
- Refinement of event $a$ into $\tau.a$
- Synthesized distributable net
- Communication inserted in the net
Specification = Behavior + Architecture

Separation failure state 2, event \( a \)

Refinement of event \( a \) into \( \tau.a \)

Synthesized distributable net

Communication inserted in the net

Communicating automata
Conclusion

A Correct by Construction Semi-Automated Derivation of Asynchronous Communicating Systems:

1. System specification: Automaton + architecture
2. Apply distributable PN synthesis and refine specification until separation is achieved
3. Insert communication into distributable net
4. Compute communicating automata

- Polynomial time algorithm (linear algebra)
- SYNET tool: PN synthesis wrt. graph isomorphism, language equality, distributable nets, ...
Analysis, Synthesis and Control of Concurrent Systems
Correct-by-construction asynchronous implementation of modular synchronous specifications

Outline

1. Introduction
2. Synthesis of Communicating Systems through PN Synthesis
   - Petri net synthesis
   - Application
3. Correct-by-construction asynchronous implementation of modular synchronous specifications
   - Introduction
   - Model and problem statement
   - Sufficient conditions
4. Interface Theories
   - Anatomy of an Interface Theory
   - Modal Interfaces
5. Current and future research directions
**Synchrony, asynchrony, GALS**

- **Synchronous specification**
  - Global clock $\Rightarrow$ specification, verification
  - Popular, efficient tools for system design (digital circuits, safety-critical systems)

- **Distributed implementation**
  - Distributed software, complex digital circuits (SoC), heterogenous systems
  - Loosely-connected components (asynchronous FIFOs...)

- **GALS architectures = good implementation model**
  - Synchronous components, asynchronous communication
  - Problem: preserve the semantic coherency between a synchronous specification and its GALS implementation
Introduction

What we want

1. Take a modular synchronous specification
What we want

1. Take a modular synchronous specification
2. Replace synchronous comm. with asynchronous FIFOs and wrappers

---

Delay-insensitive component

AFSM

IP1

IP2
What we want

1. Take a modular synchronous specification
2. Replace synchronous comm. with asynchronous FIFOs and wrappers
3. Preserve:
   - Functionality
   - Correctness (no “extra” traces, no deadlocks)

Warning: Correctness of desynchronization is undecidable (emptiness of intersection of rational relations)
Introduction

What we want

1. Take a modular synchronous specification
2. Replace synchronous comm. with asynchronous FIFOs and wrappers
3. Preserve:
   - Functionality
   - Correctness (no “extra” traces, no deadlocks)

**Warning:** Correctness of desynchronization is undecidable (emptyness of intersection of rational relations)
1. Define a model and criteria ensuring that:
   - Creating delay-insensitive wrappers that preserve the semantics is possible without adding new signals
   - Connecting through FIFOs the resulting components produces a semantics-preserving, deadlock-free GALS implementation

2. Possible approaches to enforce criteria:
   - Encode (part of) the “absent” events (Carloni et al.)
   - Add new signals
   - Decide that none is necessary due to environment constraints
The Model: Basic definitions

- **The basics:** (incomplete) automata
  \[ \Sigma = (S, s_0, V, \rightarrow), \rightarrow \subseteq S \times L(V) \times S, \ L(V) = \prod_{v \in V} (D_v \cup \bot) \]

- Composition by synchronized product:

- Renaming operator:

- Labels

- Finite traces:
The Model: Basic Definitions

- Generalized concurrent transition systems (GCTS)
  - Void transitions: \( s \rightarrow s \)
  - Prefix closure: \( s \xrightarrow{r} s' \quad \text{if} \quad q \leq r \)

Example:

```
A=1  B=7  A=1  B=7
0  \rightarrow  2  \rightarrow  1
\quad \rightarrow \quad \quad \quad \quad \quad \rightarrow \quad \quad \quad \quad \rightarrow 3
B=7  A=1  B=7  A=1
```
The Model: I/O Transition Systems

- **Point-to-point communication:**
  - Broad/Multicast can be simulated…
  - Communication channels:
    \[ c = (!c, ?c) \quad D_b = D_\tau = D_c \]
  - Dissociate emission from reception!

- **Clocks:** \( \tau, \tau_1 \ldots \) of domain \( D_\tau = \{T\} \)

- **I/O transition system:**
  - GCTS where all variables are channels or clocks
  - Example:

\[ \begin{aligned}
&0 \xrightarrow{\tau_1} 1 \\
&1 \xrightarrow{\tau_1} 0, 4
\end{aligned} \]

\[ \begin{aligned}
&1 \xrightarrow{!A=2} 2 \\
&3 \xrightarrow{?B} 2
\end{aligned} \]

\[ \begin{aligned}
&3 \xrightarrow{?R=3} 1 \\
&4 \xrightarrow{\tau_1 \tau_2} 2
\end{aligned} \]
The Model: Synchronous Systems

• **Synchronous system**: \( \Sigma = (S, s_0, V, \tau, \rightarrow) \) I/O transition system, one clock, and satisfying:

1. **Clock transitions**: \( r: S \rightarrow S' \) \( r(\tau) = T \)

2. **Stuttering invariance**: 

3. **Synchrony hypothesis**: 

\[ \text{supp}(r_i) \cap \text{supp}(r_j) = \emptyset \text{ for all } i \neq j \]

• **Example**:
The model: Composition

- **Synchronous 1-place FIFO:**

  \[ \text{SFIFO}(c, \tau): \quad !c=x \rightarrow c_x \rightarrow ?c=x \quad \text{for all } x \in D_c \]

- **Synchronous composition (on clock } \tau) :**

  \[ \Sigma_1|\Sigma_2 = \Sigma_1[\tau_1/\tau] \times \Sigma_2[\tau_2/\tau] \times \text{SFIFO}(c_1, \tau) \times \ldots \times \text{SFIFO}(c_n, \tau) \]

- **Asynchronous FIFO:**

  \[ \text{AFIFO}(c): \quad x_1 \ldots x_n \rightarrow !c=x_{n+1} \rightarrow x_1 \ldots x_{n+1} \rightarrow ?c=x_1 \rightarrow x_2 \ldots x_n \quad \text{for all } x_1, \ldots, x_n, x_{n+1} \in D_c \]

- **Asynchronous composition:**

  \[ \Sigma_1||\Sigma_2 = \Sigma_1 \times \Sigma_2 \times \text{AFIFO}(c_1) \times \ldots \times \text{AFIFO}(c_n) \]
The model: Composition

\[ \Sigma_1 | \Sigma_2 \]

\[ \Sigma_1 \ll | \Sigma_2 \]

\[ \tau \]

\[ !A \]

\[ !B \]

\[ !C \]

\[ ?C \]

\[ \tau \]

\[ !A \]

\[ !B \]

\[ !C \]

\[ ?A \]

\[ ?B \]

\[ ?C \]

\[ \tau \]

\[ !A \]

\[ !B \]

\[ !C \]

\[ ?A \]

\[ ?B \]

\[ ?C \]

\[ \tau \]

\[ !A \]

\[ !B \]

\[ !C \]

\[ ?A \]

\[ ?B \]

\[ ?C \]
Correct-by-construction asynchronous implementation of modular synchronous specifications

Model and problem statement

Example
Model and problem statement

Example

\[
\Sigma_1: \tau_1
\]

\[
\begin{align*}
\Sigma_1: \tau_1 & \quad 0 \quad \text{!A} \quad 1 \quad \text{?B} \quad 2 \quad \text{?R} \quad 3 \quad \tau_1 \\
& \quad \text{?R} \quad \text{?B} \\
\end{align*}
\]

\[
\Sigma_2: \tau_2
\]

\[
\begin{align*}
\Sigma_2: \tau_2 & \quad 0 \quad \text{?A} \quad 1 \quad \text{tau} \quad 2 \quad \text{!B} \quad 3 \\
& \quad \text{?R} \\
\end{align*}
\]

\[
\Sigma_1 | \Sigma_2:
\]

\[
\begin{align*}
\Sigma_1 | \Sigma_2: & \quad 0,0 \quad \text{!A} \quad 1,0 \quad \text{?A} \quad 1,1 \\
& \quad \text{?R} \quad \text{?R} \\
& \quad 3,0 \quad 3,1 \quad \tau \quad 3,2 \quad \text{!B} \quad 3,3 \quad 4,3 \\
\end{align*}
\]

\[
\Sigma_1 || \Sigma_2:
\]

\[
\begin{align*}
\Sigma_1 || \Sigma_2: & \quad 0,0 \quad \text{!A} \quad 1,0 \quad \text{?A} \quad 1,1 \quad 1,2 \quad 1,3 \\
& \quad \text{?R} \quad \text{?R} \quad \text{?R} \quad \text{?R} \quad \text{?R} \quad \text{?R} \\
& \quad 3,0 \quad 3,1 \quad 3,2 \quad 3,3 \quad 4,3 \\
\end{align*}
\]
Correctness

- Some notations:

\[ A = \Sigma_1 \parallel \ldots \parallel \Sigma_n \text{ is correct w.r.t. } S = \Sigma_1 | \ldots | \Sigma_n \text{ if} \]

\[ \forall s \in \text{RSS(Sync)} \text{ and all } \varphi \in \text{Traces}_A(s) \]

\[ \exists \alpha \in \text{Traces}_A(s) \text{ and } \beta \in \text{Traces}_S(s) \]

\[ \varphi \preceq \alpha \text{ and } \alpha \sim \beta \]

- Formal correctness criterion

\[ A = \Sigma_1 \parallel \ldots \parallel \Sigma_n \text{ is correct w.r.t. } S = \Sigma_1 | \ldots | \Sigma_n \text{ if} \]

\[ \forall s \in \text{RSS(Sync)} \text{ and all } \varphi \in \text{Traces}_A(s) \]

\[ \exists \alpha \in \text{Traces}_A(s) \text{ and } \beta \in \text{Traces}_S(s) \]

\[ \varphi \preceq \alpha \text{ and } \alpha \sim \beta \]

- Intuition: every trace of \( \Sigma_1 \parallel \ldots \parallel \Sigma_n \) can be completed to one that is equivalent to a synchronous trace
Weak endochrony

• Compositional delay-insensitivity criterion (signal absence information is not needed)

• Axioms (part 1):
  A1: Determinism
  A2: In every state, non-clock transitions sharing no common variable are independent
Axioms (continued):

A1: **Determinism**

A2: In every state, non-clock transitions sharing no common variable are independent

A3: Non-contradictory reactions can be united

A4: Choice does not change with time
Correct-by-construction asynchronous implementation of modular synchronous specifications

Sufficient conditions

Example

\[ \Sigma_1: \tau_1 \]

Diagram:

- States: 0, 1, 2, 3
- Time: \( \tau_1 \)

Node connections:
- 0 to 1: !A
- 1 to 2: ?B
- 1 to 3: ?R
- 2 to 0: \( \tau_1 \)
- 3 to 1: \( \tau_1 \)
Example

\[ \Sigma_1^\prime: \tau_1 \]

0 \rightarrow 1 (\!A)
1 \rightarrow 2 (\?D=0)
1 \rightarrow 3 (\?D=1)
3 \rightarrow 3 (\?R)
3 \rightarrow 0 (\tau_1)
2 \rightarrow 2 (\?B)
2 \rightarrow 0 (\tau_1)}
Example

\[ \Sigma_1': \tau_1 \]

\[ \Sigma_3: \tau_1 \]
Non blocking

- Semantics preservation

\[ \Sigma_1, \ldots, \Sigma_n \text{ weak endochronous and non blocking} \]

imply

\[ \Sigma_1 \parallel \cdots \parallel \Sigma_n \text{ is a correct desynchronization of } \Sigma_1 \mid \cdots \mid \Sigma_n \]
Non blocking

- Semantics preservation

\[ \Sigma_1, \ldots, \Sigma_n \text{ weak endochronous and non blocking} \]

imply

\[ \Sigma_1 \parallel \ldots \parallel \Sigma_n \text{ is a correct desynchronization of } \Sigma_1 | \ldots | \Sigma_n \]
Conclusion

- Decidable sufficient conditions for correct GALS implementation of synchronous specifications
- Improves previous work by taking causality into account
- Related and open problems:
  - Make synchronous automata weak endochronous. Optimality issues.
  - Heuristics for synchronous programming languages and specifications. Scaling issues (large specifications). (Potop et al.) (Ouy et al.)
Outline

1. Introduction
2. Synthesis of Communicating Systems through PN Synthesis
   - Petri net synthesis
   - Application
3. Correct-by-construction asynchronous implementation of modular synchronous specifications
   - Introduction
   - Model and problem statement
   - Sufficient conditions
4. Interface Theories
   - Anatomy of an Interface Theory
   - Modal Interfaces
5. Current and future research directions
An interface allows to represent the behavior of a family of components at design level.

- Allows independent reasoning
- Supports component-based design of large systems
- Reduces complexity of the design
- Existing Theories: Interface Automata...
An interface allows to represent the behavior of a family of components at design level

- Allows independent reasoning
- Supports component-based design of large systems
- Reduces complexity of the design
- Existing Theories: Interface Automata...

**Contributions:** Modal Interfaces (..., ACSD’09, Emsoft’09, Fund. Infor. 2011), Constraint Markov Chains (QEST’10)
An interface allows to represent the behavior of a family of components at design level.

- Allows independent reasoning
- Supports component-based design of large systems
- Reduces complexity of the design

Existing Theories: Interface Automata...

Contributions: Modal Interfaces (..., ACSD’09, Emsoft’09, Fund. Infor. 2011), Constraint Markov Chains (QEST’10)

But ... What is an interface theory?
Systems and Specifications: Implementation/Refinement Relations
Systems and Specifications: Implementation/Refinement Relations
Systems and Specifications: Implementation/Refinement Relations
Systems and Specifications: Implementation/Refinement Relations

\[ S \leq T \text{ iff } \forall M, \ M \models S \Rightarrow M \models T \]
Composition Operators

Conjunction

\[ M \models S \land T \text{ iff } M \models S \text{ and } M \models T \]
Composition Operators

Product

\[ S \otimes T = \min \{ X \mid M \models S \text{ and } N \models T \text{ implies } M \times N \models X \} \]
Quotient $S/T = \max\{X \mid X \otimes T \leq S\}$ is the adjoint of product
Summary

- Implementations $M$, parallel composition $\times$
- Specifications $S$, satisfaction relation $M \models S$
- Product
  $S \otimes T = \min\{X \mid M \models S \text{ and } N \models T \implies M \times N \models X\}$
  - Combine specifications related to distinct components
  - Build architectures
- Conjunction $M \models S \land T$ iff $M \models S$ and $M \models T$
  - Combine aspects/viewpoints related to the same component
- Quotient $S/T = \max\{X \mid X \otimes T \leq S\}$
  - Component reuse
  - Incremental design
  - Assume/Guarantee Reasoning $C = (A, G) = (G \otimes A)/A$
Modal Specifications

- Automaton $C = (S,A,\rightarrow,s_0) = (\text{states, actions, trans, init})$

- Transitions are given a label **may** or/and **must**
  - in drawings, **may** transitions are dashed
  - in drawings, **must** transitions that are also **may** are solid
  - consistency: **must** $\subseteq$ **may**
  - $\rightarrow$ deterministic

```
overload?

sent?

ack!

nack!
```

```
A
```

```
overload?

sent?, overload?
```

```
```

```
```
Automaton \( C = (S,A,\rightarrow,s_0) = (\text{states, actions, trans, init}) \)

- Transitions are given a label **may** or/and **must**
  - **implementation**: simul. rel. st. keep all **must** and some **may**
  - **refinement**: simul. rel. st. have more **must** and less **may**

![Diagram](image)
Automaton $C = (S, A, \rightarrow, s_0)$ = (states, actions, trans, init)

- Transitions are given a label **may** or/and **must**
  - **implementation**: simul. rel. st. keep all **must** and some **may**
  - **refinement**: have more **must** and less **may**

All implementations are obtained by 1/ unfolding as shown, 2/ cutting some of the dashed branches, and 3/ keeping the connected component of the initial state.
Reduction

- Automaton \( C = (S,A,\rightarrow,s_0) = (\text{states, actions, trans, init}) \)

- Transitions are given a label \( \textit{may} \) or/and \( \textit{must} \)
  - \textit{implementations}: keep all \( \textit{must} \) and some \( \textit{may} \)
  - \textit{refinement}: have more \( \textit{must} \) and less \( \textit{may} \)
  - \textit{consistency}: \( \textit{must} \subseteq \textit{may} \); may get violated \( \Rightarrow \) backward pruning
Reduction

- Automaton \( C = (S, A, \rightarrow, s_0) = (\text{states, actions, trans, init}) \)

- Transitions are given a label \textbf{may} or/and \textbf{must}
  - \textit{implementations}: keep all \textit{must} and some \textit{may}
  - \textit{refinement}: have more \textit{must} and less \textit{may}
  - \textit{consistency}: \textit{must} \textit{⊂} \textit{may}; \textit{may} may get violated \(\Rightarrow\) backward pruning
Analysis, Synthesis and Control of Concurrent Systems
Interface Theories
Modal Interfaces

Product

- Automaton \( C = (S,A,\rightarrow,s_0) = (\text{states, actions, trans, init}) \)
- Transitions are given a label \textit{may} or/and \textit{must}
- Product \( \otimes \) : \textit{may} = \textit{may}_1 \cap \textit{may}_2 , \textit{must} = \textit{must}_1 \cap \textit{must}_2

<table>
<thead>
<tr>
<th>( C_1 )</th>
<th>( C_2 )</th>
<th>( \text{mustnot} )</th>
<th>( \text{may} )</th>
<th>( \text{must} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \text{mustnot} )</td>
<td>( \text{mustnot} )</td>
<td>( \text{mustnot} )</td>
<td>( \text{mustnot} )</td>
<td>( \text{mustnot} )</td>
</tr>
<tr>
<td>( \text{may} )</td>
<td>( \text{mustnot} )</td>
<td>( \text{may} )</td>
<td>( \text{may} )</td>
<td>( \text{may} )</td>
</tr>
<tr>
<td>( \text{must} )</td>
<td>( \text{mustnot} )</td>
<td>( \text{may} )</td>
<td>( \text{must} )</td>
<td>( \text{must} )</td>
</tr>
</tbody>
</table>

The product of underlying automata; \textit{may/must} as follows.
Conjunction

- Automaton \( C = (S, A, \rightarrow, s_0) \) = (states, actions, trans, init)
- Transitions are given a label \textit{may} or/and \textit{must}
- Conjunction \( \wedge : \textit{may} = \textit{may}_1 \cap \textit{may}_2, \textit{must} = \textit{must}_1 \cup \textit{must}_2 \) (inconsistency?)

<table>
<thead>
<tr>
<th>\textbf{C}_1</th>
<th>\textbf{C}_2</th>
<th>\textit{mustnot}</th>
<th>\textit{may}</th>
<th>\textit{must}</th>
</tr>
</thead>
<tbody>
<tr>
<td>\textit{mustnot}</td>
<td>\textit{mustnot}</td>
<td>\textit{mustnot}</td>
<td>\textit{inconsistent}</td>
<td></td>
</tr>
<tr>
<td>\textit{may}</td>
<td>\textit{mustnot}</td>
<td>\textit{may}</td>
<td>\textit{must}</td>
<td></td>
</tr>
<tr>
<td>\textit{must}</td>
<td>\textit{inconsistent}</td>
<td>\textit{must}</td>
<td>\textit{must}</td>
<td></td>
</tr>
</tbody>
</table>
Quotient

- Residuation $/ : \text{adjoint of } \otimes$, $C_1 / C_2$ solves $\max_X: X \otimes C_2 \leq C_1$

<table>
<thead>
<tr>
<th>Product of underlying automata</th>
<th>$\text{may/must}$ as follows</th>
</tr>
</thead>
<tbody>
<tr>
<td>$C_1$</td>
<td>$\text{mustnot}$</td>
</tr>
<tr>
<td>$\text{mustnot}$</td>
<td>$\text{may}$</td>
</tr>
<tr>
<td>$\text{may}$</td>
<td>$\text{may}$</td>
</tr>
<tr>
<td>$\text{must}$</td>
<td>$\text{inconsistent}$</td>
</tr>
</tbody>
</table>

Observe that, even if $C_1$ and $C_2$ are both standard automata (with $\text{may} = \text{must}$), then the residuation $C_1 / C_2$ has both modalities. This shows that modalities are needed to define residuation.
Compatibility

Two interfaces are **compatible** if there exists an environment where they can avoid illegal states (i.e., states where one may want to produce an output that may not accepted as input by the other).

Composition $S_1 || S_2$ is obtained as follows:

1. compute $S_1 \otimes S_2$ and identify illegal states $I$
   
   $$S_1 : \{ a!, b! \} \quad S_2 : \{ a?, b?, c? \} \quad S_1 \otimes S_2 : \{ a!, b!, c? \}$$

2. compute exception states $X = \text{pre}_1(I)$

3. replace transitions to $X$ by transitions to the universal state $\top$

$$[S_1 || S_2]_0 : \{ a!, b!, c? \} \quad [S_1 || S_2]_1 : \{ a!, b!, c? \}$$
Conclusion

**Interface Theory:**
- Implementations $M$, parallel composition $\times$
- Specifications $S$, satisfaction relation $M \models S$
- Product
  $$S \otimes T = \min \{ X \mid M \models S \text{ and } N \models T \text{ implies } M \times N \models X \}$$
  - Combine specifications related to distinct components
  - Build architectures
- Conjunction $M \models S \land T$ iff $M \models S$ and $M \models T$
  - Combine aspects/viewpoints related to the same component
- Quotient $S/T = \max \{ X \mid X \otimes T \leq S \}$
  - Component reuse
  - Incremental design
  - Assume/Guarantee Reasoning $C = (A, G) = (G \otimes A)/A$

**Contributions:** Modal Interfaces (ACSD'09, Emsoft'09, Fund. Infor. 2011), Constraint Markov Chains (QEST'10)

**Open issues:** Quotient of stochastic interfaces
Outline

1. Introduction
2. Synthesis of Communicating Systems through PN Synthesis
   - Petri net synthesis
   - Application
3. Correct-by-construction asynchronous implementation of modular synchronous specifications
   - Introduction
   - Model and problem statement
   - Sufficient conditions
4. Interface Theories
   - Anatomy of an Interface Theory
   - Modal Interfaces
5. Current and future research directions
Extending modal interfaces

- Extending modal interfaces to data (synchronous semantics)
- Handling numerical constraints (not just finite datatypes)
- InterSMV: MTBDD based implementation
- Bridging the gap between natural language requirements and modal interfaces.
- Adapting interfaces to other types of systems: services, systems of systems, ...
Analysis, Synthesis and Control of Concurrent Systems
Current and future research directions

InterSMV

- Top level interactive env.
- Inter SMV parser/type checker
- MR to MMS encoding layer
- MMS layer: composition, reduction and analysis
- BddApron Library: finite types (BDD/MTBDD) + numerical values
- MTBDD based dedicated library

Architecture + Requirements

Composition & reduction steps

Refinement verification

InterSMV specification

Marked Modal Specifications

Analysis result