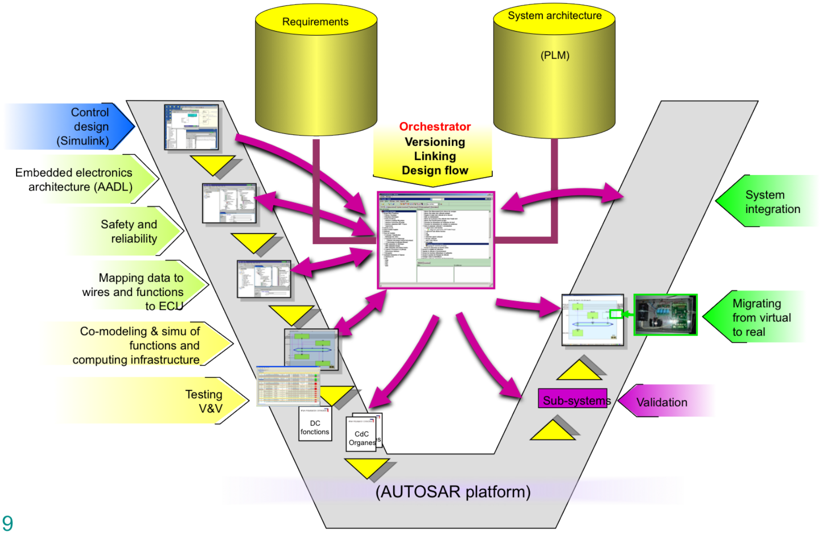
Analysis, Synthesis and Control of Concurrent Systems

Benoît Caillaud

Rennes, 23 March 2011

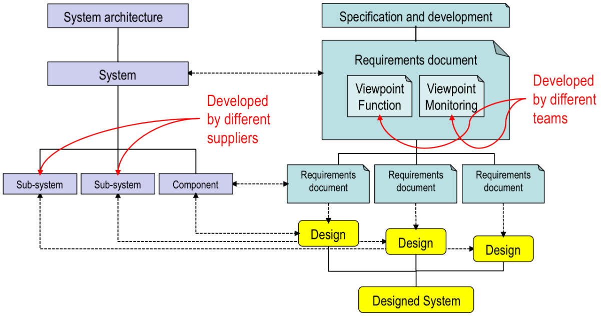
Outline

Context: Embedded Software Design


Requirements
System architecture
(PLM)
Orchestrator
Versioning
Linking
Design flow
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
DC fonctions
CdC Organes
System integration
Migrating from virtual to real
Sub-systems
Validation
(AUTOSAR platform)

Context: OEM/Suppliers Tree
Example: Integrated navigation/air-data/attitude reference system

- 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, ...) $\neq$ 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 $\rightarrow$ 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

Context: Computer Assisted Design of Asynchronous Communicating Systems

Problem

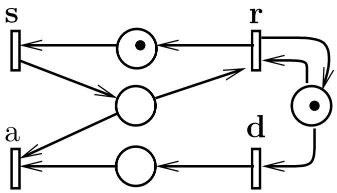
- Given a specification = behavior + architecture,
- Construct a network of communicating automata realizing the specification:
  1. Matches the specified architecture
  2. Branching bisimilar to the specified behavior

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

$$\begin{tikzcd}
{} \arrow[r, squiggly] & 0 \arrow[r, "\mathbf{d}"] \arrow[d, bend right, "\mathbf{s}"'] & 2 \arrow[d, "\mathbf{s}"] \\
& 1 \arrow[u, bend right, "\mathbf{r}"'] \arrow[r, "\mathbf{d}"] & 3 \arrow[r, "\mathbf{a}"] & 4
\end{tikzcd}$$

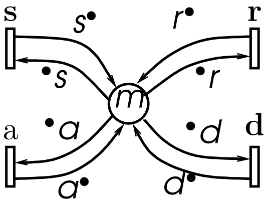


Net synthesis problem

Given a finite transition system A, decide whether exists a net N such that $N^* \simeq A$.

PN synthesis in a nutshell

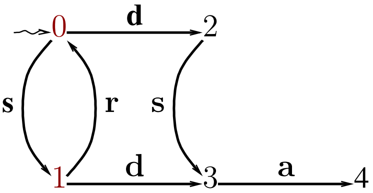
$$\begin{tikzcd}
{} \arrow[r, squiggly] & 0 \arrow[r, "\mathbf{d}"] \arrow[d, bend right, "\mathbf{s}"'] & 2 \arrow[d, bend right, "\mathbf{s}"'] & \\
& 1 \arrow[u, bend right, "\mathbf{r}"'] \arrow[r, "\mathbf{d}"] & 3 \arrow[r, "\mathbf{a}"] & 4
\end{tikzcd}$$

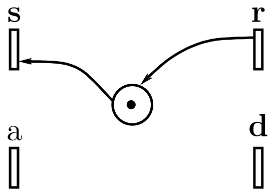


- Net = union of 1-place nets
- Region = 1-place net satisfying:

$$\begin{cases} m, {}^\bullet s, s^\bullet, \ldots {}^\bullet d, d^\bullet \geq 0 \\ m \geq {}^\bullet s \\ m + s^\bullet - {}^\bullet s \geq {}^\bullet d \\ \vdots \\ m + s^\bullet - {}^\bullet s + d^\bullet - {}^\bullet d \geq {}^\bullet a \\ s^\bullet - {}^\bullet s + r^\bullet - {}^\bullet r = 0 \end{cases}$$

PN synthesis in a nutshell

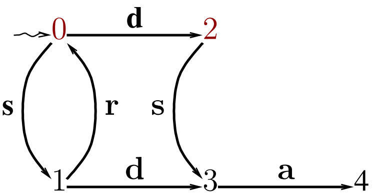


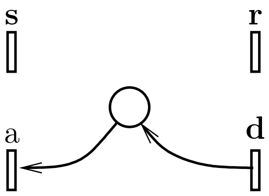


- Net = union of 1-place nets
- State separation: states 0 and 1

$$\begin{cases} m, {}^\bullet s, s^\bullet, \ldots {}^\bullet d, d^\bullet \geq 0 \\ m \geq {}^\bullet s \\ m + s^\bullet - {}^\bullet s \geq {}^\bullet d \\ \vdots \\ m + s^\bullet - {}^\bullet s + d^\bullet - {}^\bullet d \geq {}^\bullet a \\ s^\bullet - {}^\bullet s + r^\bullet - {}^\bullet r = 0 \\ s^\bullet - {}^\bullet s \neq 0 \end{cases}$$

PN synthesis in a nutshell

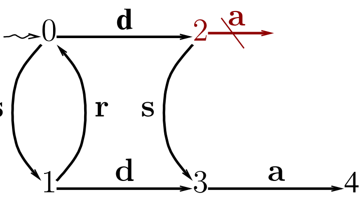


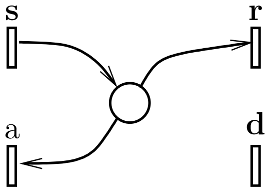


- Net = union of 1-place nets
- State separation: states 0 and 2

$$\begin{cases} m, {}^\bullet s, s^\bullet, \ldots {}^\bullet d, d^\bullet \geq 0 \\ m \geq {}^\bullet s \\ m + s^\bullet - {}^\bullet s \geq {}^\bullet d \\ \vdots \\ m + s^\bullet - {}^\bullet s + d^\bullet - {}^\bullet d \geq {}^\bullet a \\ s^\bullet - {}^\bullet s + r^\bullet - {}^\bullet r = 0 \\ d^\bullet - {}^\bullet d \neq 0 \end{cases}$$

PN synthesis in a nutshell

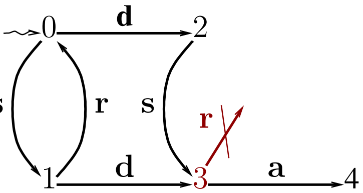


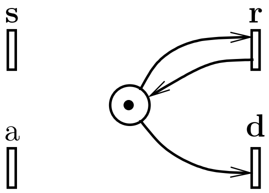


- Net = union of 1-place nets
- State separation: ...
- Event-state separation: event a in state 2

$$\begin{cases} m, {}^\bullet s, s^\bullet, \ldots {}^\bullet d, d^\bullet \geq 0 \\ m \geq {}^\bullet s \\ m + s^\bullet - {}^\bullet s \geq {}^\bullet d \\ \vdots \\ m + s^\bullet - {}^\bullet s + d^\bullet - {}^\bullet d \geq {}^\bullet a \\ s^\bullet - {}^\bullet s + r^\bullet - {}^\bullet r = 0 \\ m + d^\bullet - {}^\bullet d < {}^\bullet a \end{cases}$$

PN synthesis in a nutshell



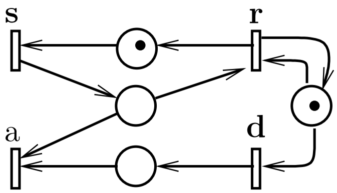


- Net = union of 1-place nets
- State separation: ...
- Event-state separation: event r in state 3

$$\begin{cases} m, {}^\bullet s, s^\bullet, \ldots {}^\bullet d, d^\bullet \geq 0 \\ m \geq {}^\bullet s \\ m + s^\bullet - {}^\bullet s \geq {}^\bullet d \\ \vdots \\ m + s^\bullet - {}^\bullet s + d^\bullet - {}^\bullet d \geq {}^\bullet a \\ s^\bullet - {}^\bullet s + r^\bullet - {}^\bullet r = 0 \\ m + s^\bullet - {}^\bullet s + d^\bullet - {}^\bullet d < {}^\bullet r \end{cases}$$

PN synthesis in a nutshell

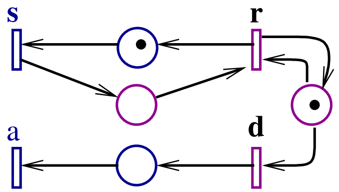
$$\begin{tikzcd}
\arrow[r, squiggly] & 0 \arrow[r, "\mathbf{d}"] \arrow[d, bend right, "\mathbf{s}"'] & 2 \arrow[d, bend right, "\mathbf{s}"'] & \\
& 1 \arrow[u, bend right, "\mathbf{r}"'] \arrow[r, "\mathbf{d}"] & 3 \arrow[r, "\mathbf{a}"] & 4
\end{tikzcd}$$



- 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, ...
- `http://www.irisa.fr/s4/tools/synet/`

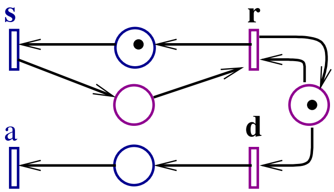
Distributable nets

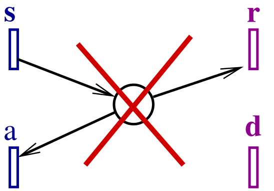
- Distributable net = Petri net + architecture



Distributable nets

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

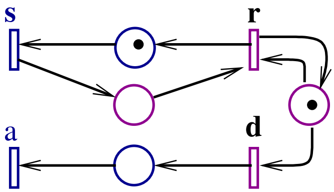


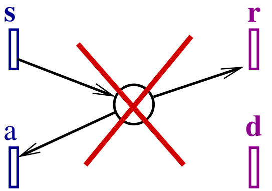


Distributable nets

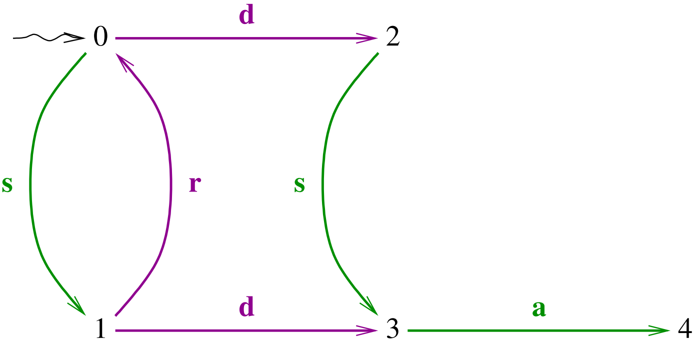
- Distributable net = Petri net + architecture
- Syntactic class of nets with no distributed conflict
- Admits trivial asynchronous distributed implementations
- Distributable net synthesis: Add constraints of the form

  $^\bullet t = 0$ for every non-local transition t





Design Process Examplified



$$\begin{tikzcd}
\arrow[r, squiggly] & 0 \arrow[r, "d"] \arrow[d, bend right, "s"'] & 2 \arrow[d, bend right, "s"'] & \\
& 1 \arrow[u, bend right, "r"'] \arrow[r, "d"] & 3 \arrow[r, "a"] & 4
\end{tikzcd}$$

- Specification = Behavior + Architecture

Design Process Examplified

$$\begin{tikzcd}
{} \arrow[r, squiggly] & 0 \arrow[r, "d"] \arrow[d, bend right, "s"'] & 2 \arrow[r, "a"] \arrow[d, "s"] & {} \\
& 1 \arrow[r, "d"] \arrow[u, bend right, "r"'] & 3 \arrow[r, "a"] & 4
\end{tikzcd}$$

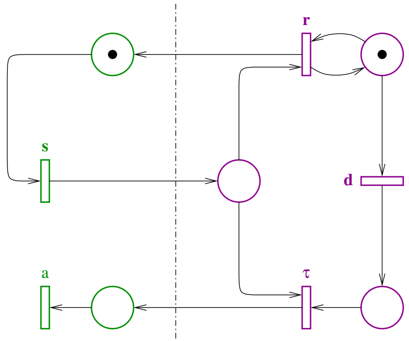
- Specification = Behavior + Architecture
- Separation failure state 2, event *a*

Design Process Examplified

$$\begin{tikzcd}
{} \arrow[r, squiggly] & 0 \arrow[r, "d"] \arrow[d, bend right, "s"'] & 2 \arrow[d, "s"'] & & \\
& 1 \arrow[u, bend right, "r"'] \arrow[r, "d"] & 3 \arrow[r, "\tau"] & 4 \arrow[r, "a"] & 5
\end{tikzcd}$$

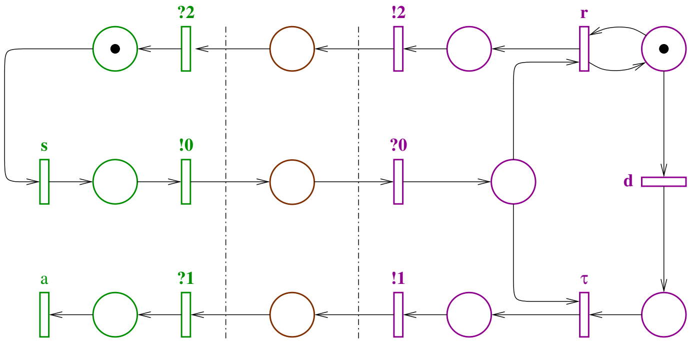
- Specification = Behavior + Architecture
- Separation failure state 2, event *a*
- Refinement of event *a* into $\tau.a$

Design Process Examplified



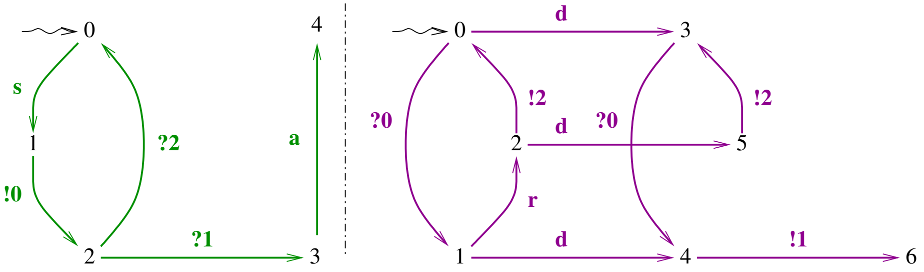
- Specification = Behavior + Architecture
- Separation failure state 2, event *a*
- Refinement of event *a* into $\tau.a$
- Synthesized distributable net

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

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
- 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

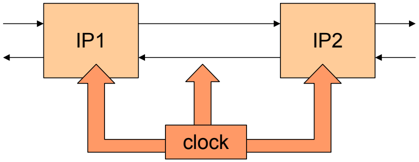
- Polynomial time algorithm (linear algebra)
- SYNET tool: PN synthesis wrt. graph isomorphism, language equality, distributable nets, ...
- `http://www.irisa.fr/s4/tools/synet/`

Outline

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

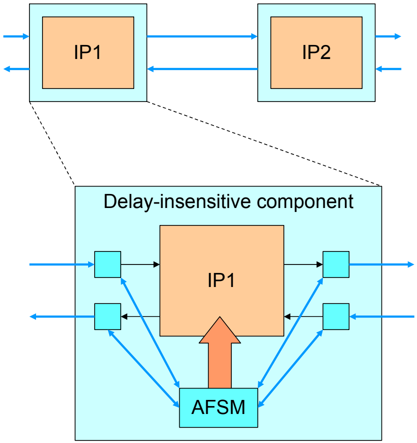
What we want



1. Take a modular synchronous specification

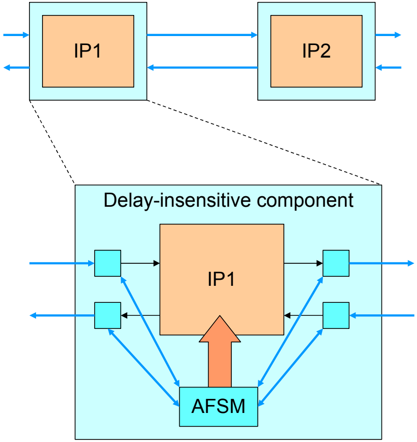
What we want

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



What we want

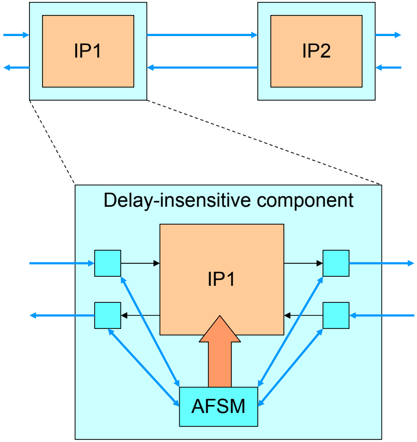
1. Take a modular synchronous specification
2. Replace synchronous comm. with asynchronous FIFOs and wrappers
3. Preserve:
    - Functionality
    - Correctness (no “extra” traces, no deadlocks)



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

$$\Sigma = (S, s_0, V, \rightarrow), \quad \rightarrow \subseteq S \times L(V) \times S, \quad L(V) = \prod_{v \in V} (D_v \cup \perp)$$

- Composition by synchronized product:

$$\begin{tikzcd}
 & & & 2 & & \\
0 \arrow[r, "A=1\ B=\perp\ D=\perp"'] & 1 & \times\ 0 \arrow[ur, "A=1\ B=\perp\ C=3"] \arrow[dr, "A=1\ B=7\ C=3"'] & & = & 0{,}0 \arrow[r, "A=1\ B=\perp\ C=3\ D=\perp"'] & 1{,}2 \\
 & & & 1 & &
\end{tikzcd}$$

- Renaming operator:

$$\Sigma_1[D/C] : \begin{tikzcd} 0 \arrow[r, "A=1\ B=\perp\ C=\perp"'] & 1 \end{tikzcd}$$

- Labels

$A=1\ B=\perp\ C=3 \ \equiv\ A=1\ C=3$

$A=1\ C=3\ \leq\ A=1\ B=7\ C=3$

$A=1\ C=3 - A=1 = C=3$

- Finite traces:

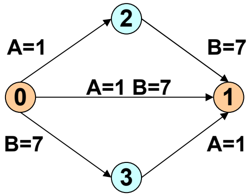
$A=1\ C=3\ ;\ B=2\ ;\ ;$

$A=1\ C=3\ ;\ B=2\ ;\ ;\ <\ A=1\ C=3\ ;\ B=2\ ;\ ;\ A=2;$

The Model: Basic Definitions

- Generalized concurrent transition systems(GCTS)
  - Void transitions: $\begin{tikzcd} s \arrow[r, "\perp"] & s \end{tikzcd}$
  - Prefix closure: $\begin{tikzcd} s \arrow[r, "r"] & s' \end{tikzcd}$ $q \leq r$ $\Rightarrow$ $\begin{tikzcd} s \arrow[r, "q"] & s'' \arrow[r, "r-q"] & s' \end{tikzcd}$

- Example:



The Model: I/O Transition Systems

- Point-to-point communication:
  - Broad/Multicast can be simulated…
  - Communication channels:
    $c = (!c, ?c) \qquad D_{!c} = D_{?c} = D_c$
  - Dissociate emission from reception!
- Clocks: $\tau\ \tau_1 \ldots$ of domain $D_{clk} = \{T\}$
- I/O transition system:
  - GCTS where all variables are channels or clocks
  - Example:

$$\begin{tikzcd}
 & & 2 \arrow[r, "\tau_1\tau_2"] & 4 \\
0 \arrow[loop left, "\tau_1"] \arrow[r, bend right, "?R=4"'] & 1 \arrow[l, "!A=1"'] \arrow[ur, "!A=2"] \arrow[dr, "?R=3"'] & & \\
 & & 3 \arrow[u, "?B"'] \arrow[loop right, "\tau_1"] &
\end{tikzcd}$$

The Model: Synchronous Systems

- Synchronous system: $\Sigma = (S, s_0, V, \tau, \rightarrow)$ I/O transition system, one clock, and satisfying:
  1. Clock transitions: $$\begin{tikzcd} s \arrow[r, "r"] & s' \end{tikzcd} \quad r(\tau) = \top \;\Big|\; \Rightarrow \; \mathbf{r} \text{ equals } \perp \text{ over } \mathbf{V}$$
  2. Stuttering invariance: $$\begin{tikzcd} s_0 \arrow[r, "\tau"] & s_0 \end{tikzcd} \qquad \begin{tikzcd} s \arrow[r, "\tau"] & s' \end{tikzcd} \Rightarrow \begin{tikzcd} s' \arrow[loop right, "\tau"] \end{tikzcd}$$
  3. Synchrony hypothesis: $$\begin{tikzcd} \bar{s}_0 \arrow[r, "r_1"] & s_1 \arrow[r, "r_2"] & \dots \arrow[r, "r_n"] & s_n \end{tikzcd} \quad r_i \neq \tau \;\Big|\; \Rightarrow \; \mathbf{supp(r_i) \cap supp(r_j) = \emptyset} \text{ for all } \mathbf{i} \neq \mathbf{j}$$
- Example:

$$\begin{tikzcd} & & 2 \arrow[loop right, "\tau_1"] \\ 0 \arrow[loop left, "\tau_1"] \arrow[r, "!A"] & 1 \arrow[ur, "?B"] \arrow[dr, "?R"'] & \\ & & 3 \arrow[loop right, "\tau_1"] \end{tikzcd}$$

The model: Composition

- Synchronous 1-place FIFO:

SFIFO(c, τ):
$$\begin{tikzcd} c_0 \arrow[r, "!c=x"] & c_x \arrow[r, "?c=x"] & c_1 \arrow[ll, bend left, "\tau"] \end{tikzcd}$$
for all $x \in D_c$

- Synchronous composition (on clock τ) :

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

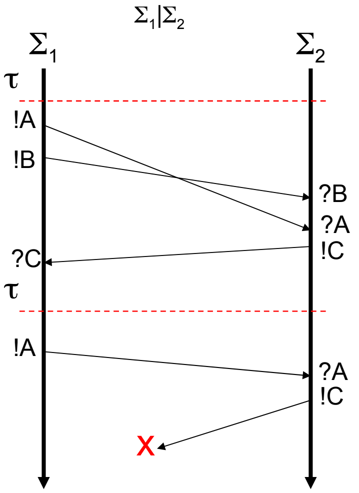
- Asynchronous FIFO:

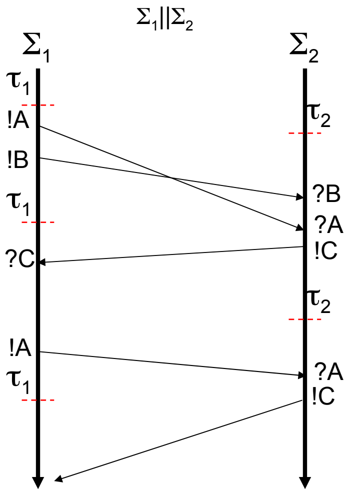
AFIFO(c):
$$\begin{tikzcd} x_1 \ldots x_n \arrow[r, "!c=x_{n+1}"] & x_1 \ldots x_{n+1} \arrow[r, "?c=x_1"] & x_2 \ldots x_n \end{tikzcd}$$
for all $x_1, \ldots, x_n, x_{n+1} \in D_c$

- Asynchronous composition:

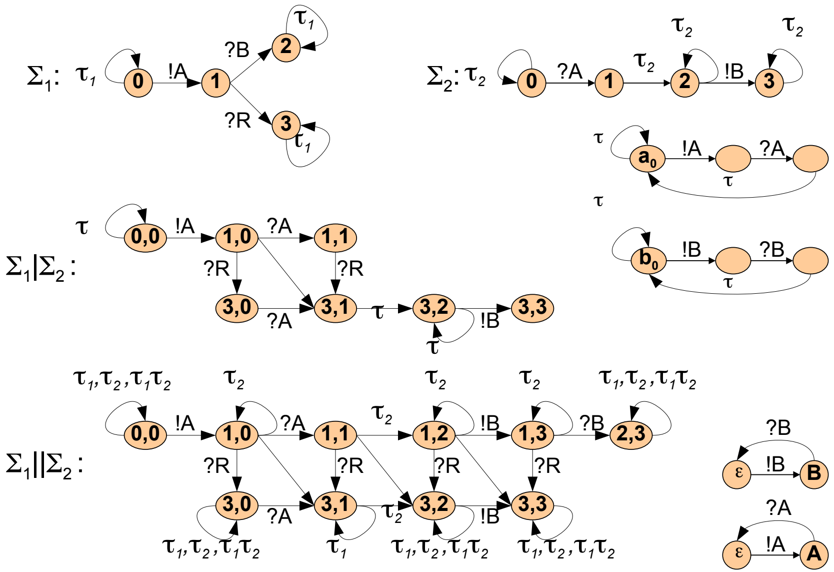
$$\Sigma_1||\Sigma_2 = \Sigma_1 \times \Sigma_2 \times \mathrm{AFIFO}(c_1) \times \ldots \times \mathrm{AFIFO}(c_n)$$

The model: Composition

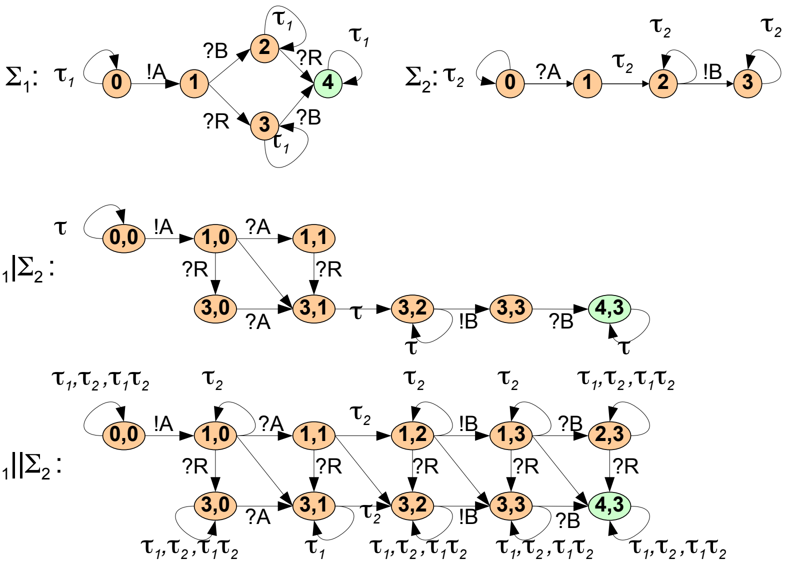




Example


Σ1: τ1
0
!A
1
?B
2
τ1
?R
3
τ1
Σ2: τ2
0
?A
1
τ2
2
τ2
!B
3
τ2
τ
a0
!A
?A
τ
τ
b0
!B
?B
τ
Σ1|Σ2 :
τ
0,0
!A
1,0
?A
1,1
?R
?R
3,0
?A
3,1
τ
3,2
!B
3,3
τ
Σ1||Σ2 :
τ1,τ2,τ1τ2
τ2
τ2
τ2
τ1,τ2,τ1τ2
0,0
!A
1,0
?A
1,1
τ2
1,2
!B
1,3
?B
2,3
?R
?R
?R
?R
3,0
?A
3,1
τ2
3,2
!B
3,3
τ1,τ2,τ1τ2
τ1
τ1,τ2,τ1τ2
τ1,τ2,τ1τ2
?B
ε
!B
B
?A
ε
!A
A

Example


Σ_1:
τ_1
0
!A
1
?B
2
?R
3
4
?R
?B
τ_1
τ_1
τ_1
Σ_2: τ_2
0
?A
1
τ_2
2
!B
3
τ_2
τ_2
$\Sigma_1|\Sigma_2$:
τ
0,0
!A
1,0
?A
1,1
?R
?R
3,0
?A
3,1
τ
3,2
!B
3,3
?B
4,3
τ
τ
$\Sigma_1||\Sigma_2$:
$\tau_1,\tau_2,\tau_1\tau_2$
τ_2
τ_2
τ_2
τ_2
$\tau_1,\tau_2,\tau_1\tau_2$
0,0
!A
1,0
?A
1,1
1,2
!B
1,3
?B
2,3
?R
?R
?R
?R
?R
3,0
?A
3,1
τ_2
3,2
!B
3,3
?B
4,3
$\tau_1,\tau_2,\tau_1\tau_2$
τ_1
$\tau_1,\tau_2,\tau_1\tau_2$
$\tau_1,\tau_2,\tau_1\tau_2$
$\tau_1,\tau_2,\tau_1\tau_2$

Correctness

- Some notations:

  $!A=1\ ;\tau_1;\ ?A=1\ ;\tau_2;\ !C=3\ ;\ \sim\ !A=1\ ?A=1\ ;\tau_1\tau_2;\ !C=3\ ;\tau_2;$

  $!A=1\ ;\tau_1;\tau_2;\ !C=3\ ;\ \preccurlyeq\ !A=1\ ?A=1\ ;\tau_1\tau_2;\ !C=3\ ;\tau_2;$

- Formal correctness criterion

  $A = \Sigma_1||\ldots||\Sigma_n$ is correct w.r.t. $S = \Sigma_1|\ldots|\Sigma_n$ if

  for all $s \in RSS(Sync)$ and all $\varphi \in Traces_A(s)$

  there exist $\alpha \in Traces_A(s)$ and $\beta \in Traces_S(s)$

  such that $\varphi \preccurlyeq \alpha$ and $\alpha \sim \beta$

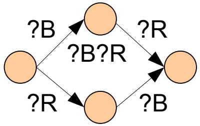
- Intuition: every trace of $\Sigma_1||\ldots||\Sigma_n$ can be completed to one that is equivalent to a synchronous trace

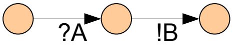
Weak endochrony

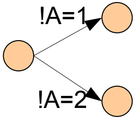
- Compositional delay-insensitivity criterion (signal absence information is not needed)
- Axioms (part 1):

  A1: Determinism

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







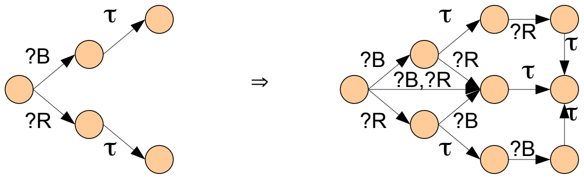
Weak endochrony

- Axioms (continued):

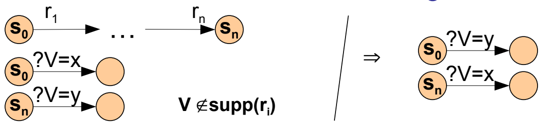
  A1: Determinism

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

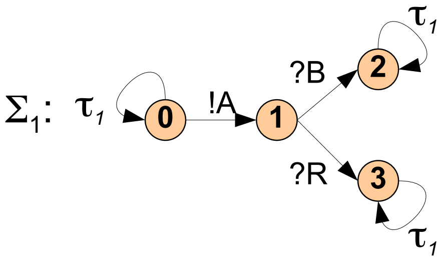
  A3: Non-contradictory reactions can be united



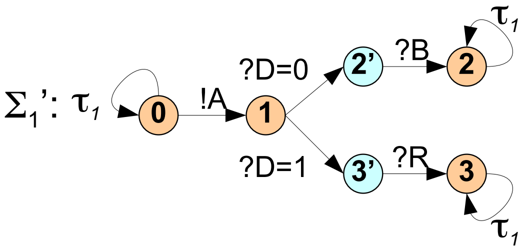
  A4: Choice does not change with time



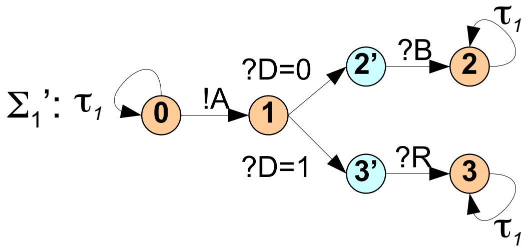
Example

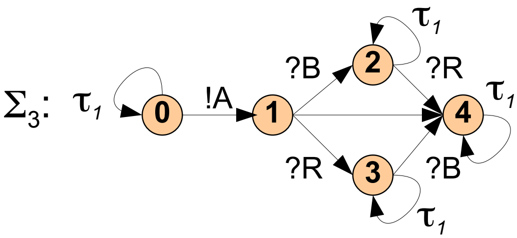


Example



Example





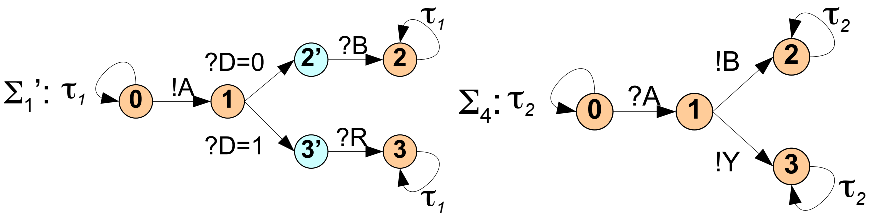
Non blocking

- Semantics preservation

$\Sigma_1,\ldots,\Sigma_n$ weak endochronous and non blocking

imply

$\Sigma_1 \| \ldots \| \Sigma_n$ is a correct desynchronization of $\Sigma_1 | \ldots | \Sigma_n$



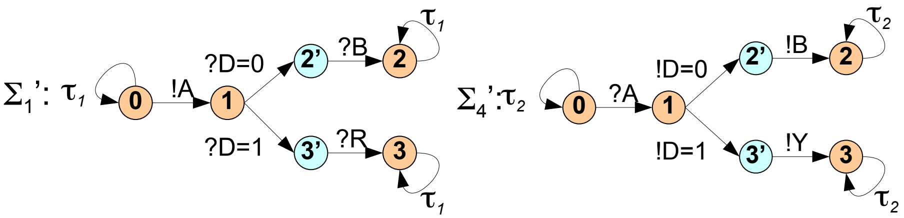
Non blocking

- Semantics preservation

$\Sigma_1, \ldots, \Sigma_n$ weak endochronous and non blocking

imply

$\Sigma_1 \| \ldots \| \Sigma_n$ is a correct desynchronization of $\Sigma_1 \mid \ldots \mid \Sigma_n$



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

Interface Theories:

- 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...

Interface Theories:

- 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)

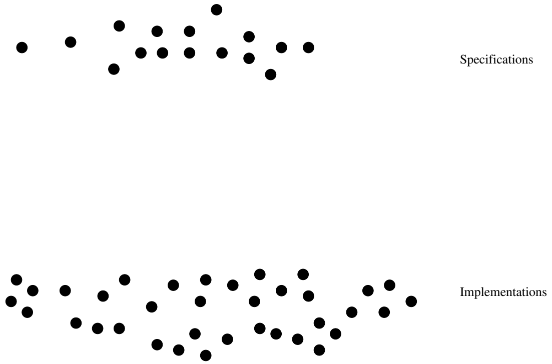
Interface Theories:

- 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...

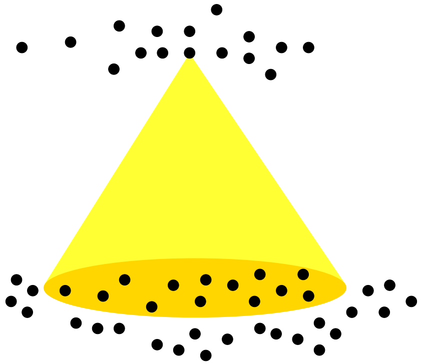
Contributions: Modal Interfaces (...,ACSD'09, Emsoft'09, Fund. Infor. 2011), Constraint Markov Chains (QEST'10)

But ... What is an interface theory?

Systems and Specifications: Implementation/Refinement Relations



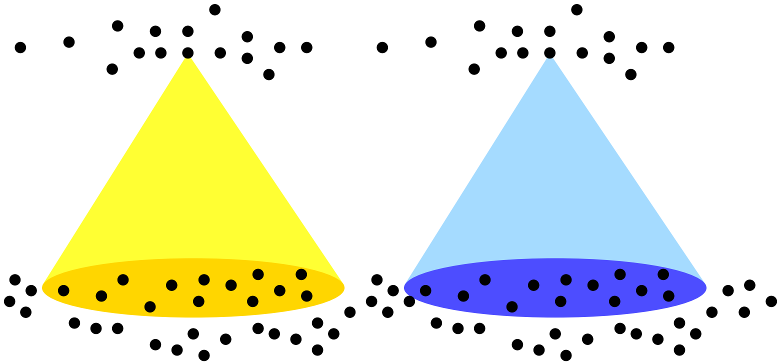
Systems and Specifications: Implementation/Refinement Relations



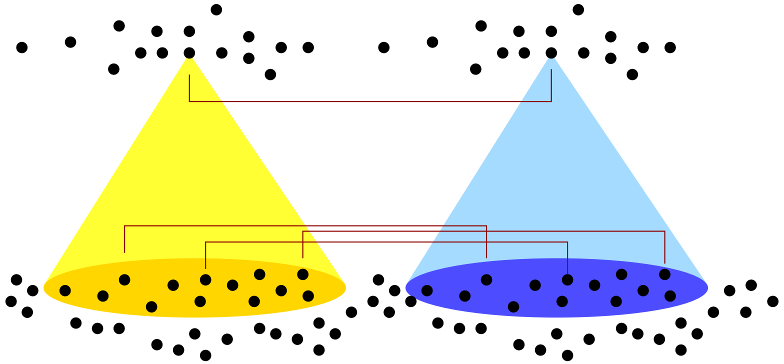
Specifications

Implementations

Systems and Specifications: Implementation/Refinement Relations

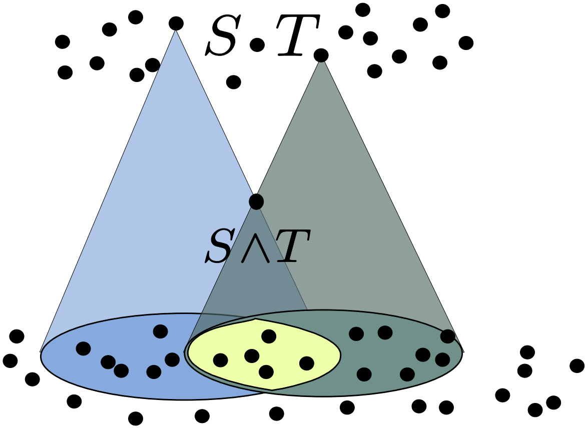


Systems and Specifications: Implementation/Refinement Relations



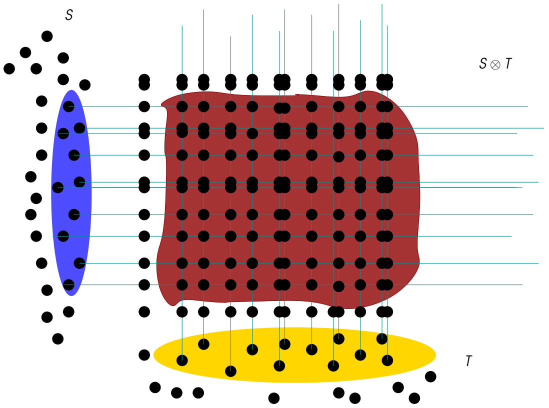
$$S \leq T \text{ iff } \forall M,\ M \models S \Rightarrow M \models T$$

Composition Operators
Conjunction



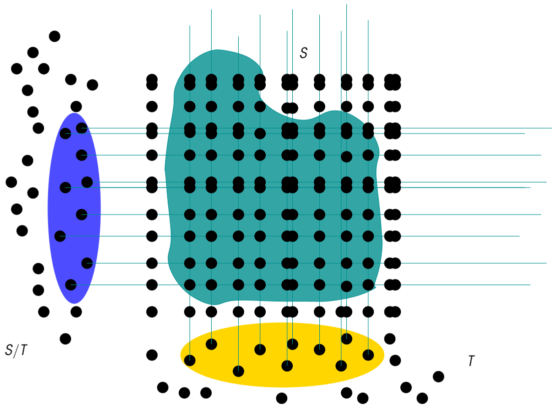
$M \models S \wedge T$ iff $M \models S$ and $M \models T$

Composition Operators
Product



$$S \otimes T = \min\{X \mid M \models S \text{ and } N \models T \text{ implies } M \times N \models X\}$$

Composition Operators
Quotient



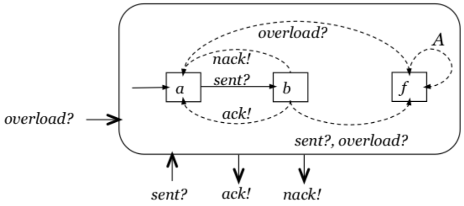
Quotient $S/T = \max\{X \mid X \otimes T \leq S\}$ is the adjoint of product

Summary

- Implementations M, parallel composition $\times$
- Specifications S, satisfaction relation $M \models S$
- Product
  $S \otimes T = \min\{X \mid M \models S \text{ and } N \models T \text{ implies } M \times N \models X\}$
  - Combine specifications related to distinct components
  - Build architectures
- Conjunction $M \models S \wedge T$ 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 $C = (A, G) = (G \otimes A)/A$

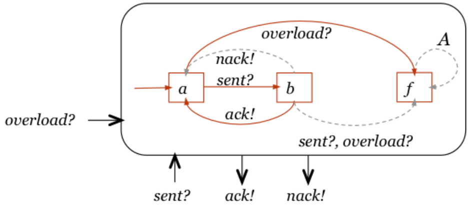
Modal Specifications

- Automaton $C = (S, A, \rightarrow, s_0) =$ (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* $\subset$ *may*
  - $\rightarrow$ deterministic



Implementation and Refinement

- Automaton C = (S,A,→,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:*** simul. rel. st. have more *must* and less *may*

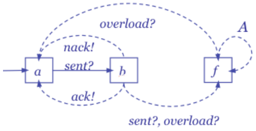


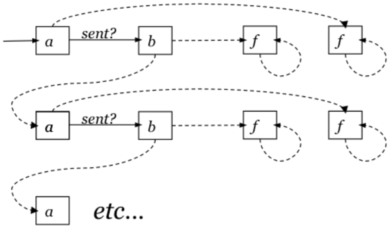
An implementation;

this is a simple one, getting all of them is explained next

Implementation and Refinement

- 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*

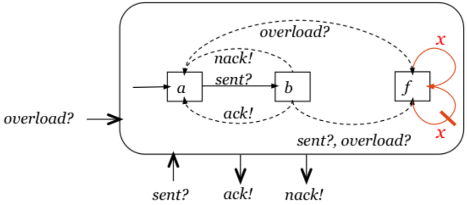




All implementations are obtained by 1/ unfolding as shown, 2/ cutting some of the dashed branches, and 3/ keeping the connected component of the initial state

Reduction

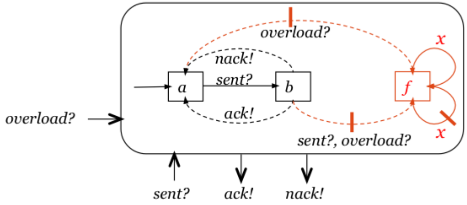
- Automaton $C = (S, A, \rightarrow, s_0)$ = (states, actions, trans, init)
- Transitions are given a label ***may*** or/and ***must***
  - ***implementations:*** keep all *must* and some *may*
  - ***refinement:*** have more *must* and less *may*
  - ***consistency***: *must* $\subset$ *may*; may get violated $\Rightarrow$ backward pruning



f inconsistent;

Reduction

- Automaton $C = (S, A, \rightarrow, s_0)$ = (states, actions, trans, init)
- Transitions are given a label ***may*** or/and ***must***
  - ***implementations:*** keep all *must* and some *may*
  - ***refinement:*** have more *must* and less *may*
  - ***consistency***: *must* ⊂ *may*; may get violated ⇒ backward pruning



f inconsistent; backward kill incoming transitions; if a *must* transition is killed, this causes further inconsistency and thus pruning must be repeated

Product

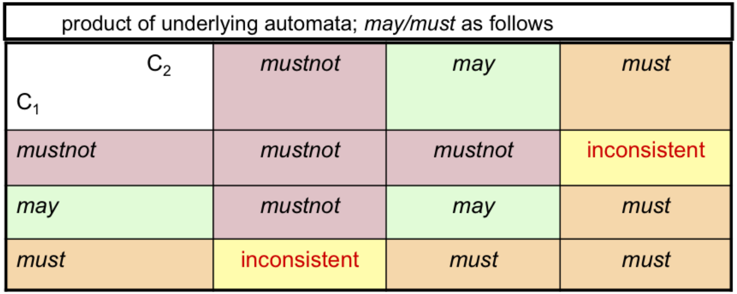
- Automaton $C = (S,A,\rightarrow,s_0)$ = (states, actions, trans, init)
- Transitions are given a label ***may*** or/and ***must***
- Product $\otimes$: *may* = *may*$_1 \cap$ *may*$_2$, *must* = *must*$_1 \cap$ *must*$_2$

product of underlying automata; *may/must* as follows

C_1 \ C_2	*mustnot*	*may*	*must*
mustnot	*mustnot*	*mustnot*	*mustnot*
may	*mustnot*	*may*	*may*
must	*mustnot*	*may*	*must*

Conjunction

- Automaton C = (S,A,$\rightarrow$,s_0) = (states, actions, trans, init)
- Transitions are given a label ***may*** or/and ***must***
- Conjunction $\wedge$: *may* = $may_1 \cap may_2$, *must* = $must_1 \cup must_2$ (inconsistency?)



product of underlying automata; *may/must* as follows

C_2 / C_1	*mustnot*	*may*	*must*
mustnot	*mustnot*	*mustnot*	inconsistent
may	*mustnot*	*may*	*must*
must	inconsistent	*must*	*must*

Quotient

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

product of underlying automata; *may/must* as follows

C_2 C_1	*mustnot*	*may*	*must*
mustnot	*may*	*mustnot*	*mustnot*
may	*may*	*may*	*may*
must	inconsistent	inconsistent	*must*

pruning

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

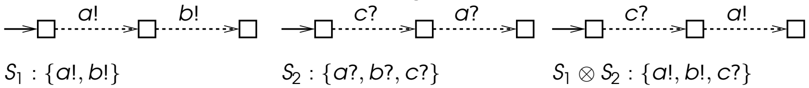
Inputs and Outputs: Compatibility

Compatibility

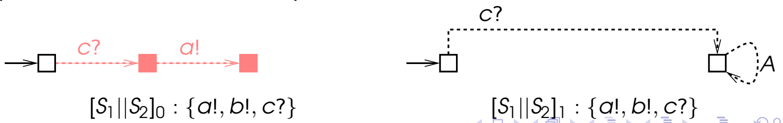
Two interfaces are **compatible** if there exists an environment where they can avoid illegal states (i.e., states where one **may** want to produce an output that **may** not accepted as input by the other).

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

1. compute $S_1 \otimes S_2$ and identify illegal states I



2. compute exception states $X = \mathrm{pre}_!(I)$
3. replace transitions to X by transitions to the universal state $\top$


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

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 $M \models S \wedge T$ 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

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, ...

InterSMV

Top level interactive env.

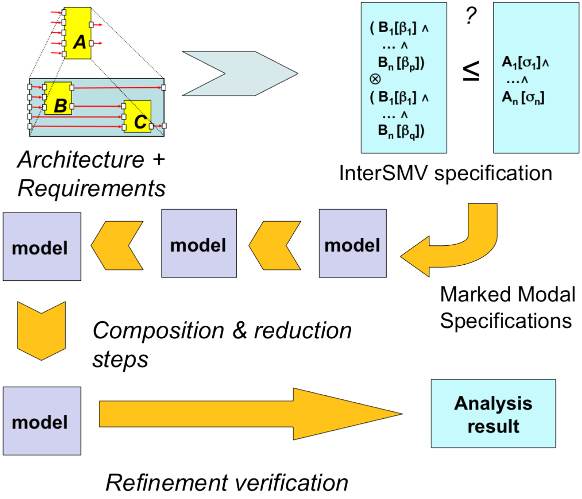
Inter SMV parser/type checker

MR to MMS encoding layer

MMS layer: composition, reduction and analysis

BddApron Library: finite types (BDD/ MTBDD) + numerical values

MTBDD based dedicated library


A
B
C
Architecture + Requirements
$(B_1[\beta_1] \wedge \ldots \wedge B_n[\beta_p]) \otimes (B_1[\beta_1] \wedge \ldots \wedge B_n[\beta_q])$
?
$\leq$
$A_1[\sigma_1] \wedge \ldots \wedge A_n[\sigma_n]$
InterSMV specification
model
model
model
Marked Modal Specifications
Composition & reduction steps
model
Analysis result
Refinement verification