Why Modalities are good for Interface Theories

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

Jean-Baptiste Raclet – INRIA Rennes, now with INRIA Grenoble
Eric Badouel, Albert Benveniste, Benoît Caillaud – INRIA Rennes
Roberto Passerone – PARADES / University of Trento
Tom Henzinger – EPFL

What do we expect from Interface Theories?

Anatomy of Interface Theories (1)

Refinement & Implementations: $C \leq C' \Leftrightarrow M \models C \Rightarrow M \models C'$

Substituability: $M \models C$ and $M' \models C' \Rightarrow M \times M' \models C \otimes C'$

Side issues:
- Deadlock freeness (additional issue, "compatible interfaces")

[de Alfaro Henzinger 01, 04] [Larsen & al 07]

Anatomy of Interface Theories (1)

Refinement & Implementations: $C \leq C' \;\Leftrightarrow\; M \models C \Rightarrow M \models C'$

Substituability: $M \models C$ and $M' \models C' \;\Rightarrow\; M \times M' \models C \otimes C'$

Some theories ignore implementations and work directly with refinement, which amounts to considering that implementations and interfaces are identical.

Whence $\models$ coincides with $\leq$ in this case.

Side issues:
- Deadlock freeness (additional issue, "compatible interfaces")

[de Alfaro Henzinger 01, 04] [Larsen & al 07]

Anatomy of Interface Theories (2)

Refinement & Implementations: $C \leq C' \Leftrightarrow M \models C \Rightarrow M \models C'$

Substituability: $M \models C$ and $M' \models C' \Rightarrow M \times M' \models C \otimes C'$

Conjunction: $M \models C$ and $M \models C' \Leftrightarrow M \models C \wedge C'$

Required if multiple viewpoint reasoning is wanted

Also, corresponds to the current practice of requirements capture in system design, where the specification of a component consists of a set of elementary requirements

Anatomy of Interface Theories (3)

Refinement & Implementations: $C \leq C' \iff M \models C \Rightarrow M \models C'$

Substituability: $M \models C$ and $M' \models C' \Rightarrow M \times M' \models C \otimes C'$

Conjunction: $M \models C$ and $M \models C' \iff M \models C \wedge C'$

Residuation: $M \models C/C' \iff \forall M' \models C', M \times M' \models C$

Solves the following side issues:

- Assume/Guarantee reasoning
  Represent C as C=G/A
- Solving equations, $\max_X: X \otimes C' \leq C$
  solution given by X=C/C'

Anatomy of Interface Theories (sumup)

Refinement & Implementations: $C \leq C' \iff M \models C \Rightarrow M \models C'$

Substituability: $M \models C$ and $M' \models C' \Rightarrow M \times M' \models C \otimes C'$

Conjunction: $M \models C$ and $M \models C' \iff M \models C \wedge C'$

Residuation: $M \models C/C' \iff \forall M' \models C', M \times M' \models C$

Side issues:

- Deadlock freeness, compatibility
- Assume/Guarantee reasoning
  Represent C as C=G/A
- Solving equations, $\max_X: X \otimes C' \leq C$
  solution given by X=C/C'

[this talk]

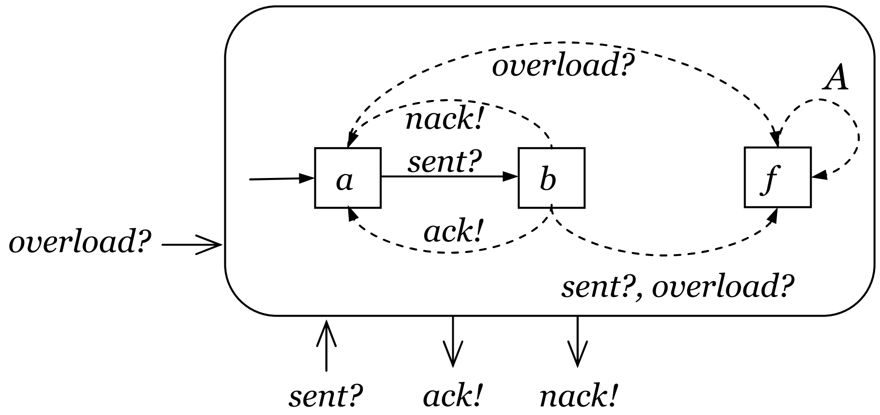
Modal Automata provide such a framework

Modal Automata & variants

- *Modal Automata* are automata with *may* and *must* transitions
- They were proposed by Kim Larsen in 1989 to study refinement specification; further developed by his school since then
- Recent PhD Thesis (2008) by Ulrik Nyman
  - Interface Theories and Product Line Theories
  - establishes a link with Interface Automata
- PhD Thesis (2007) by Jean-Baptiste Raclet
  - Residuation
  - A new variant of Modal Automata, the *Acceptance Automata*, with increased expressiveness (e.g., expressing deadlock freeness requirements and providing algorithms for its synthesis)

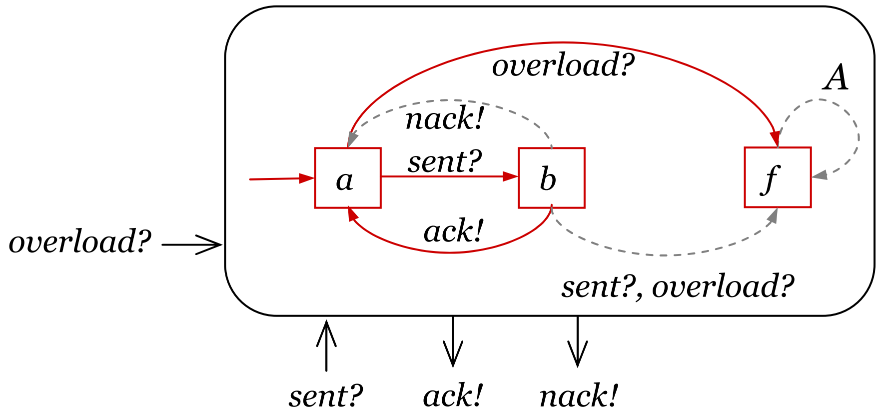
Modal Automata

- 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* $\subseteq$ *may*



Modal Automata

- 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*

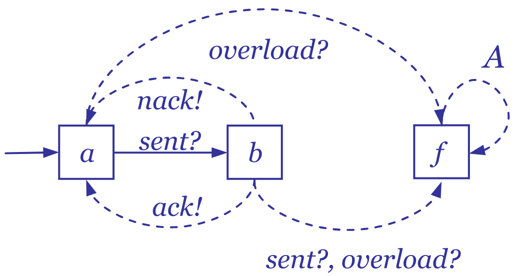


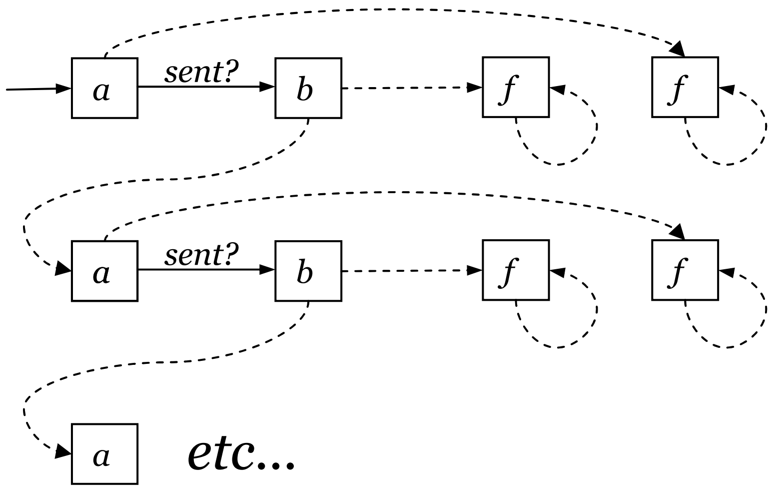
An implementation;

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

Modal Automata

- 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*

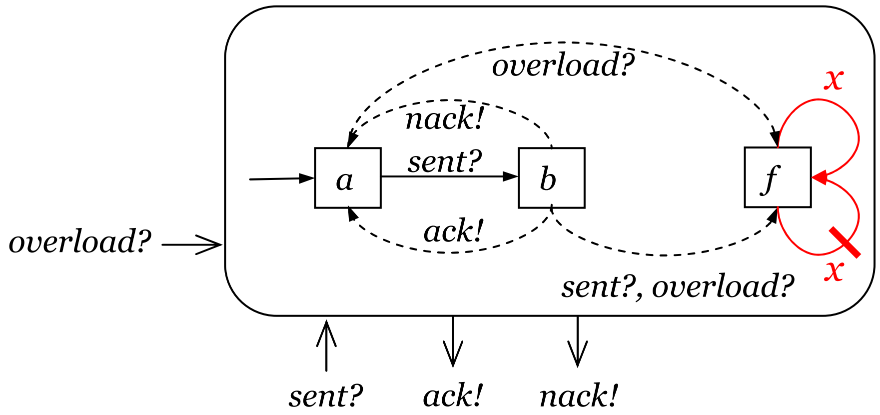




All implementations are obtained by 1/ unfolding as shown, 2/ cutting some of the dashed branches, and 3/ keeping the connected component of the initial state

Modal Automata

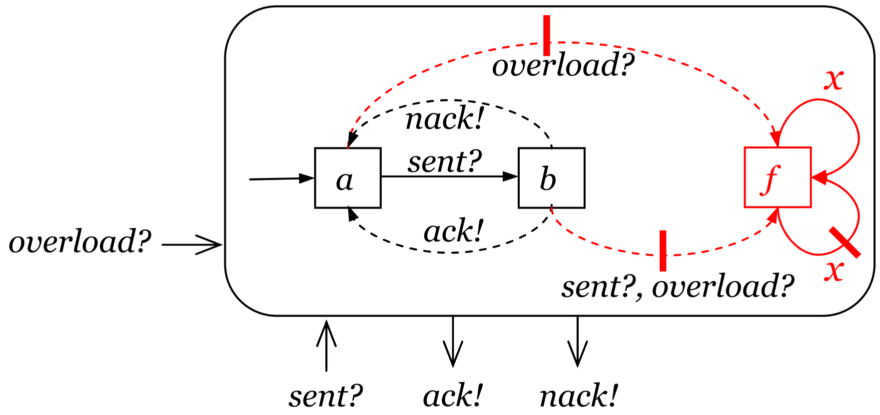
- Automaton C = (S,A,$\rightarrow$,s_0) = (states, actions, trans, init)
- Transitions are given a label ***may*** or/and ***must***
  - ***implementations:*** keep all *must* and some *may*
  - ***refinement:*** have more *must* and less *may*
  - ***consistency***: *must* $\subseteq$ *may*; may get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



f inconsistent;

Modal Automata

- 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* $\subseteq$ *may*; may get violated $\Rightarrow$ backward pruning



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

Modal Automata

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

Modal Automata: residuation C_1/C_2

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 no *must* transitions), then the residuation C_1/C_2 has both modalities. This shows that modalities are needed to define residuation.

Modal Automata offer:

Refinement & Implementations:	$C \leq C' \Leftrightarrow M \models C \Rightarrow M \models C'$
Substituability:	$M \models C$ and $M' \models C' \Rightarrow M \times M' \models C \otimes C'$
Conjunction:	$M \models C$ and $M \models C' \Leftrightarrow M \models C \wedge C'$
Residuation:	$M \models C/C' \Leftrightarrow \forall M' \models C', M \times M' \models C$
(later) Side issues:	❑ Deadlock freeness
	❑ Assume/Guarantee reasoning Represent C as C=G/A
	❑ Solving equations, $\max_X$: $X \otimes C' \leq C$ $X = C/C'$

Warning

- The above results fail when dealing with non-deterministic modal automata
  - for example, modal refinement is sound but not complete if non-determinism is allowed

- One way to enforce determinism is to work with *modal specifications*, consisting in equipping languages, not machines, with modalities; makes the maths simpler

- The above results need to be slightly modified when the alphabets of actions differ for the different modal automata, see below

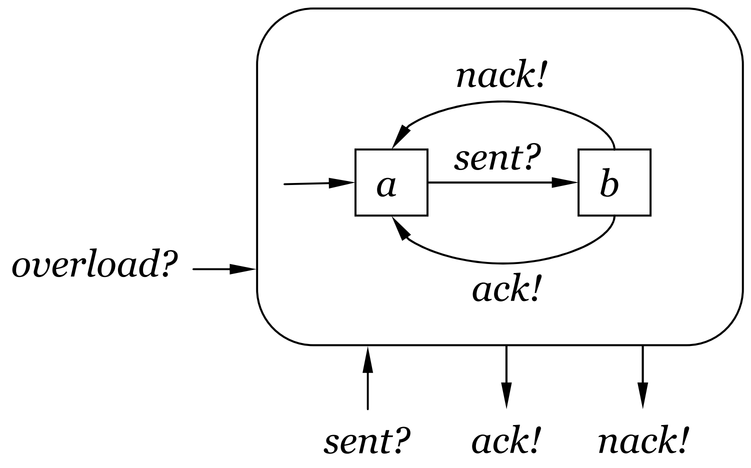
From Interface Automata to Modal Automata

[Larsen-Nyman-Wasowski 2007]

40 ans la révolution de l'information

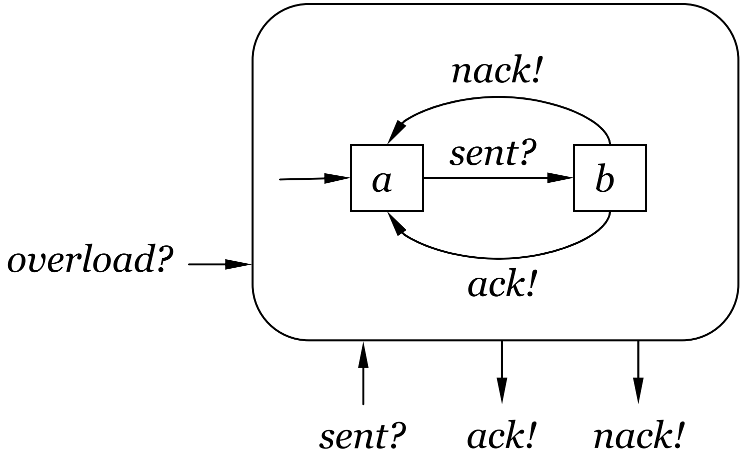
INRIA

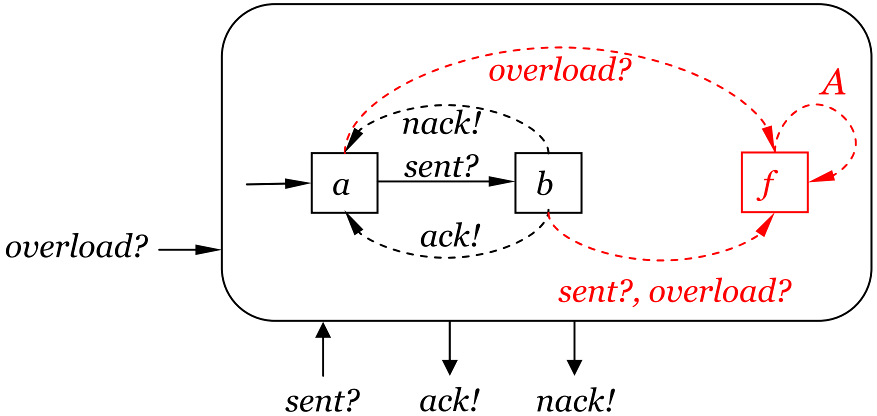
An interface automaton



An interface automaton and its translation as an MA

- Inputs:
  - the IA specifies Assumptions on the environment by forbidding some transitions
  - the MA is input enabled by accepting that the environment may not comply with Assumptions (red *f* state and transitions)
  - compliant: *must*
  - non-compliant: *may*
- Outputs:
  - best effort semantics: *may*
  - in state *f* contract is breached and thus any output is allowed





Results regarding this translation

[Larsen-Nyman-Wasowski 2007]

Alternating Simulation

$\downarrow$

Modal Refinement

- The above result expresses that this translation is faithful
- Parallel composition, however, raises a difficulty: the treatment of *error states* and *compatibility* is incorrect (with the unfortunate consequence that compatibility is not stable under modal refinement [Doyen-Jobstmann])

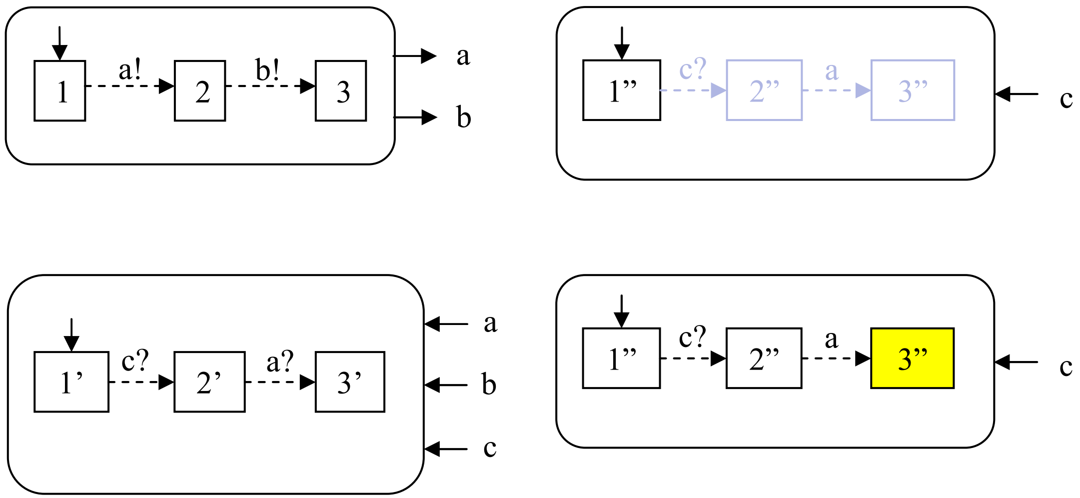
Addressing compatibility and deadlock freeness

Thanks are due to Barbara Jobstmann and Laurent Doyen, who pointed the counter-example to us

40 ans
la révolution de l'information

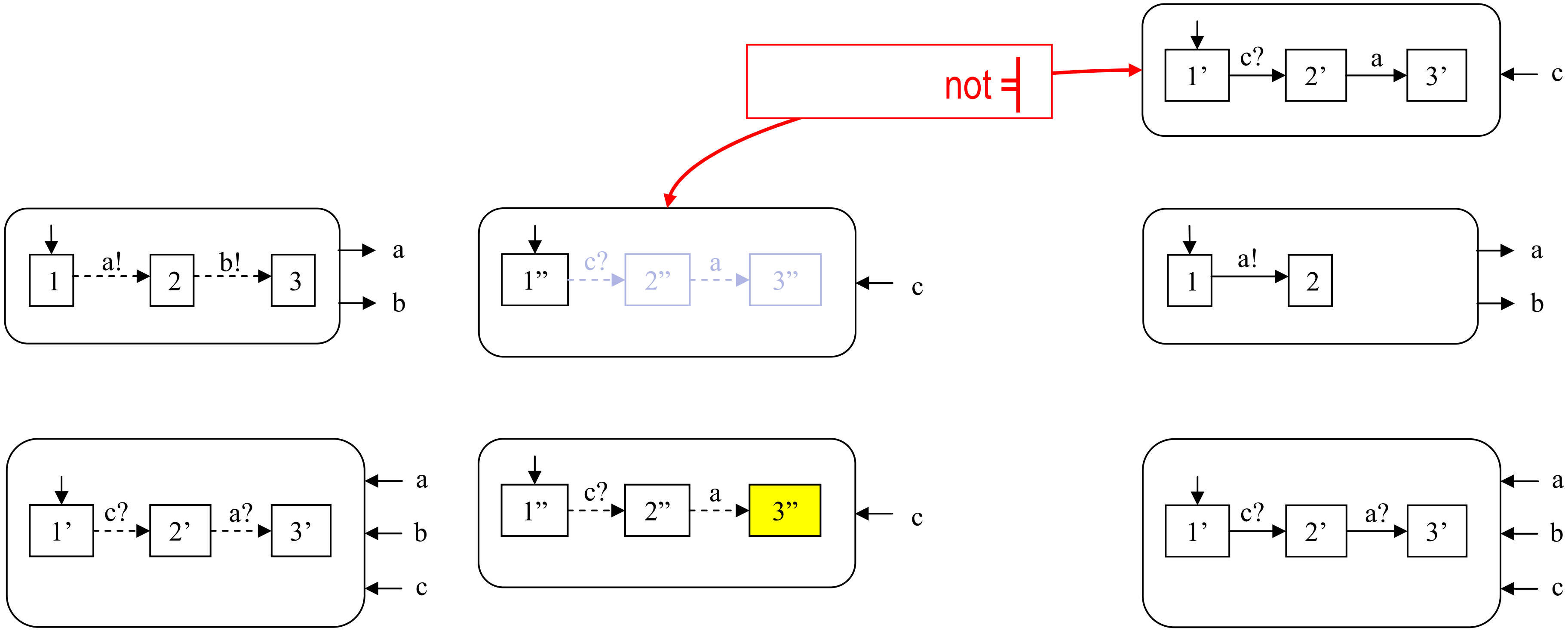
INRIA

A counter-example



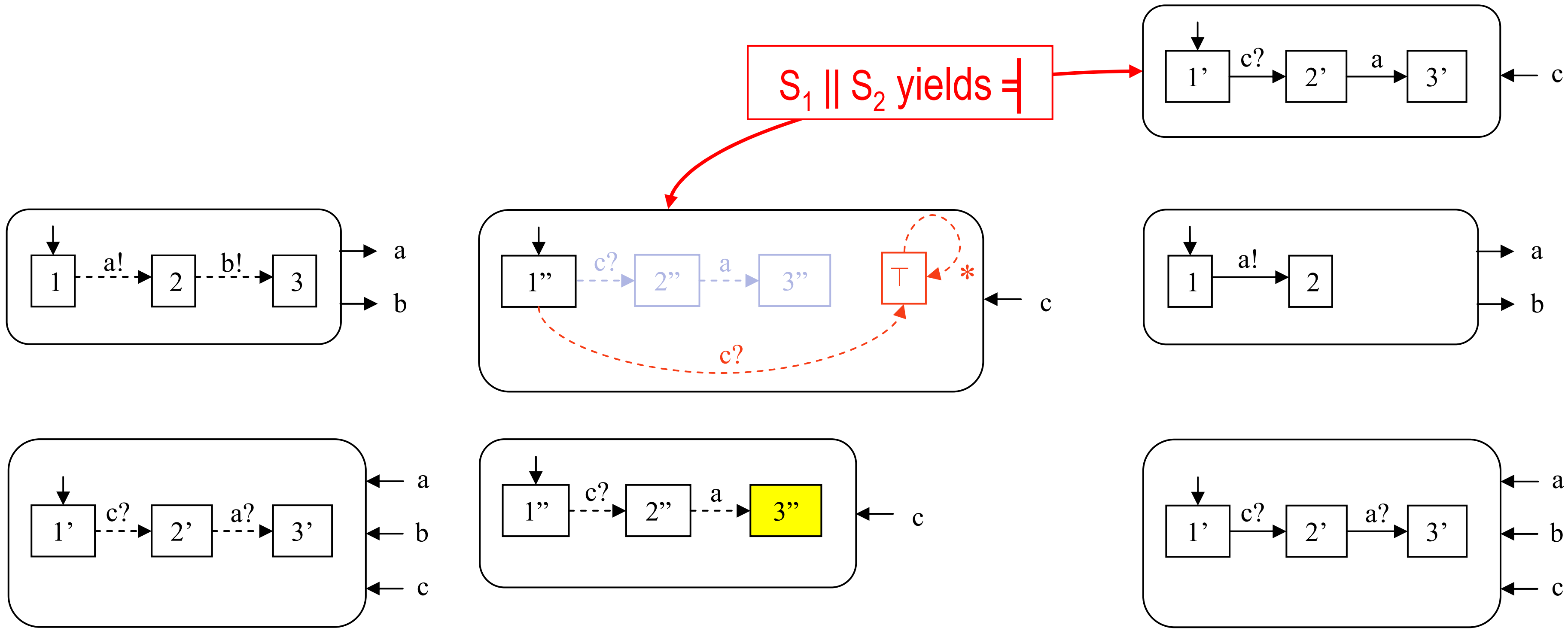
1. S_1, S_2 shown on the left
2. Bottom-mid: $S_1 \otimes S_2$ before pruning; error state in yellow; backward pruning should erase *a* and then *c?* Result shown on top-mid.

A counter-example



1. S_1, S_2 shown on the left
2. Bottom-mid: $S_1 \otimes S_2$ before pruning; error state in yellow; backward pruning should erase *a* and then *c?* Result shown on top-mid.
3. Right: implementations of S_1 and S_2 and their product (top)

A counter-measure



1. S_1, S_2 shown on the left
2. Bottom-mid: $S_1 \otimes S_2$ before pruning; error state in yellow; backward pruning should erase *a* and then *c?* Rather than removing *c?* we re-route it as indicated next, by adding a top state: $S_1 \,||\, S_2$
3. Right: implementations of S_1 and S_2 and their product (top)

Some essential subtleties when equalizing alphabets

(…where non-modal frameworks fail…)

40 ans
la révolution de l'information

INRIA

Alphabet equalization (see next example)

Needed in the following cases:

- Implementations can have a larger alphabet
- Different components can have different alphabets
- In taking conjunctions, contracts may have different alphabets

Problems with alphabet equalization

- How to extend a specification to a larger alphabet of actions?
- *Such an extension should be **neutral**, meaning that it should not constrain what other interfaces may want to require regarding these extra actions*;

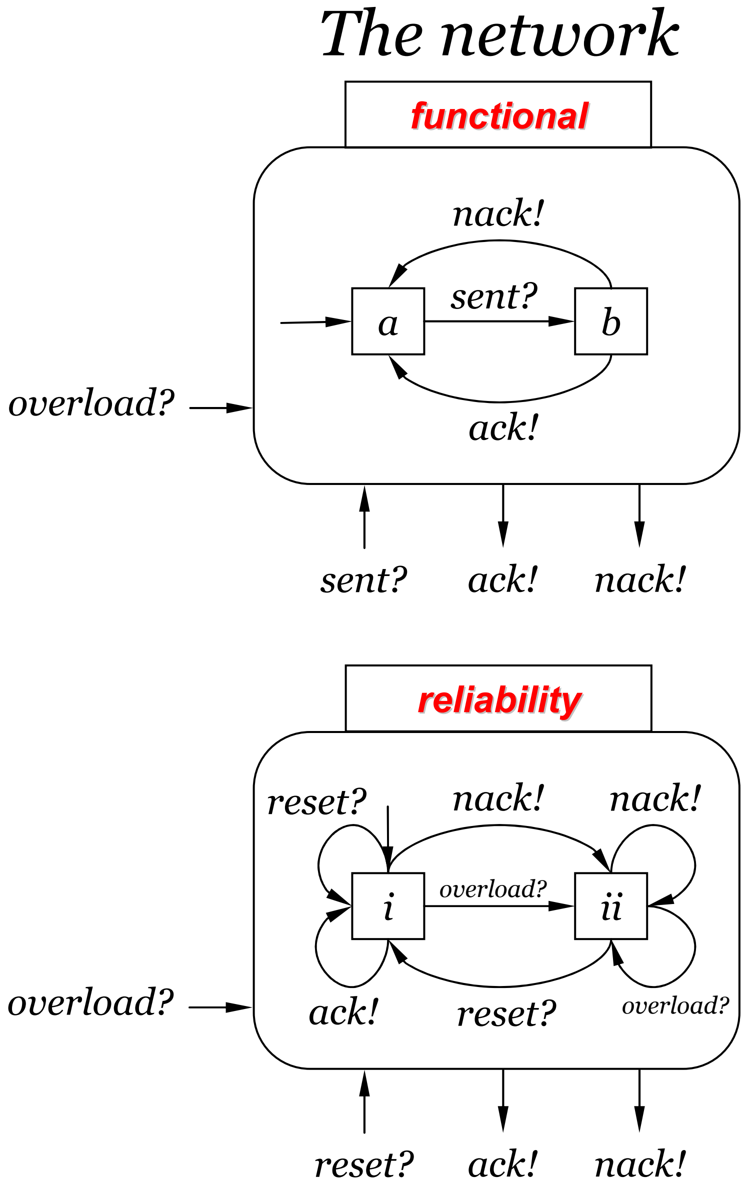
Should "neutral" differ

- for input or output actions? (we know it should)
- for product? for conjunction? (we will see it has to)

The *inverse projection* classically used in equalizing alphabets cannot adjust properly to all these different situations

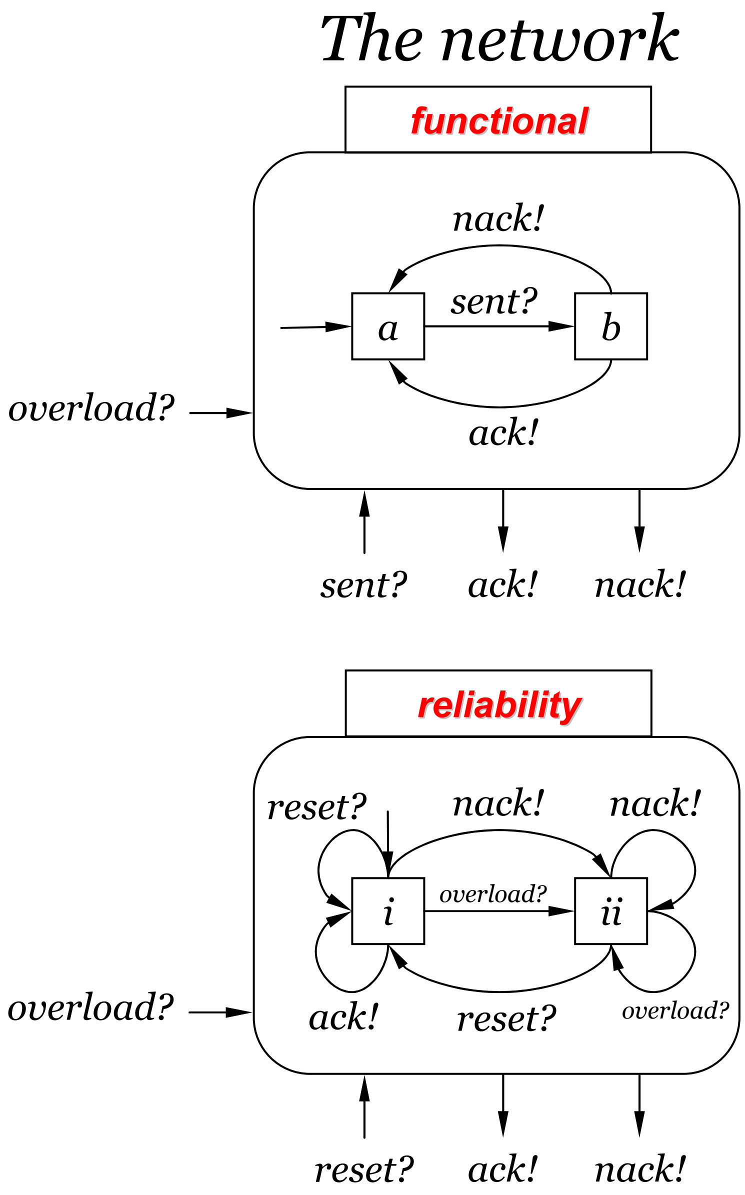
The *network* ex. as an IA

- The network has a user who submits messages; it is subject to overload; it has a supervisor that can reset it
- The network specification possesses two viewpoints:
  - ***functional***: when receiving a request *sent* from the server, it can answer either *ack* (in case transmission succeeds) or *nack* (otherwise); observe that this functional model assumes no overload for the network
  - ***reliability***: under overload the network will not stop failing to transmit until a *reset* action is performed by the supervisor; overload of the network can happen any time; it causes moving to state [ii]



The *network* ex. as an IA

- These two IA possess different alphabets
  - cf. *send?* and *reset?*: “functional” does not care about *reset?* And vice-versa for “reliability”
- How can we avoid constraining what other interfaces may want to require regarding these extra actions, i.e., how to be ***neutral*** ?



- These two IA possess different alphabets
  - cf. *send?* and *reset?*: “functional” does not care about *reset?* And vice-versa for “reliability”
- How can we avoid constraining what other interfaces may want to require regarding these extra actions, i.e., how to be ***neutral*** ?

Translate these two IA into MA using the above translation and then make the following observations:

- These two IA possess different alphabets
  - cf. *send?* and *reset?*: "functional" does not care about *reset?* And vice-versa for "reliability"
- How can we avoid constraining what other interfaces may want to require regarding these extra actions, i.e., how to be ***neutral*** ?
  - in the context of $\otimes$ we must equip unknown actions with *must*, since the latter is neutral for $\otimes$: *must*(a) and *may*(a) yields *may*(a)
  - in the context of $\wedge$ we must equip unknown actions with *may*, since the latter is neutral for $\wedge$: *must*(a) and *may*(a) yields *must*(a)

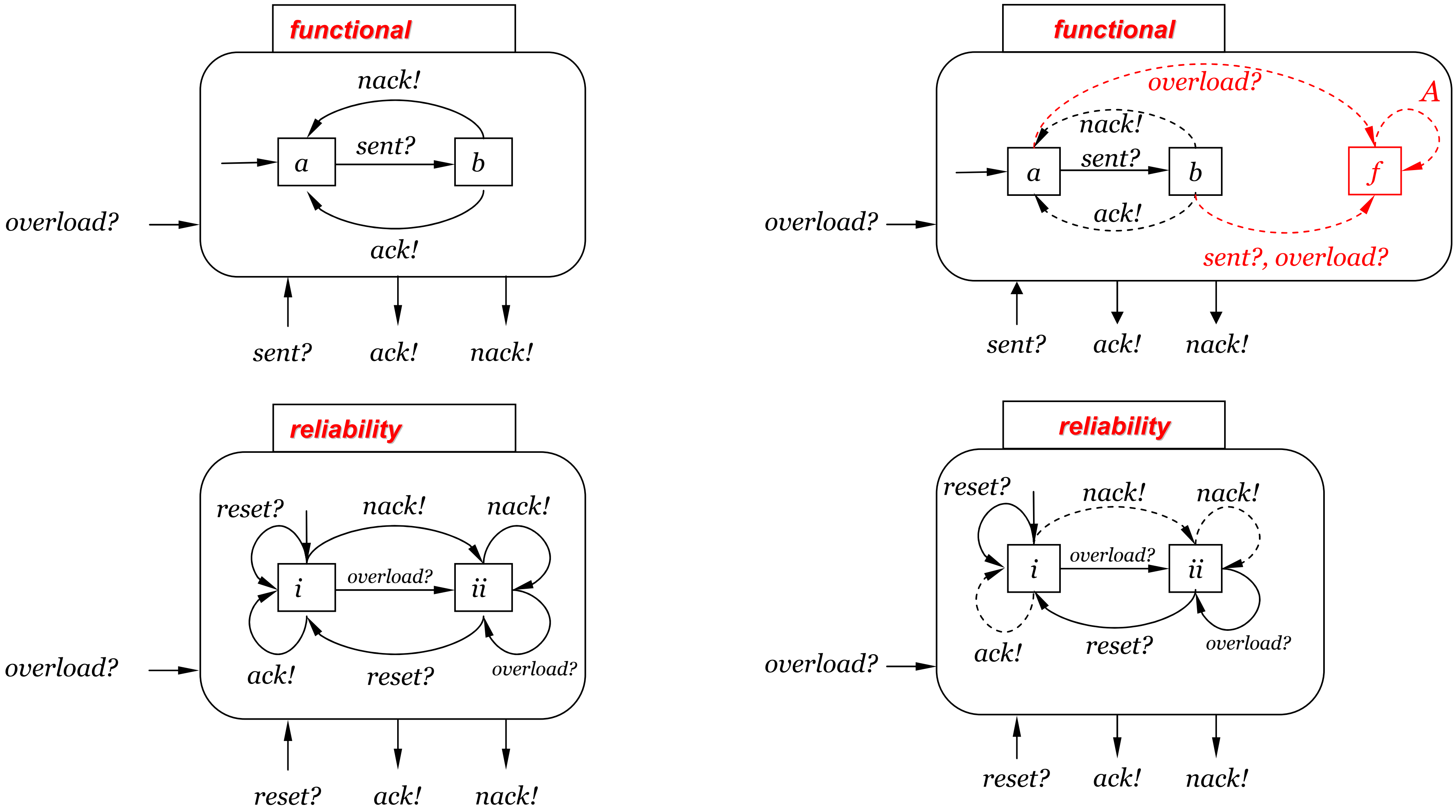
Translate these two IA into MA using the above translation and then make the following observations:

Observe that what is said here makes explicit use of modalities

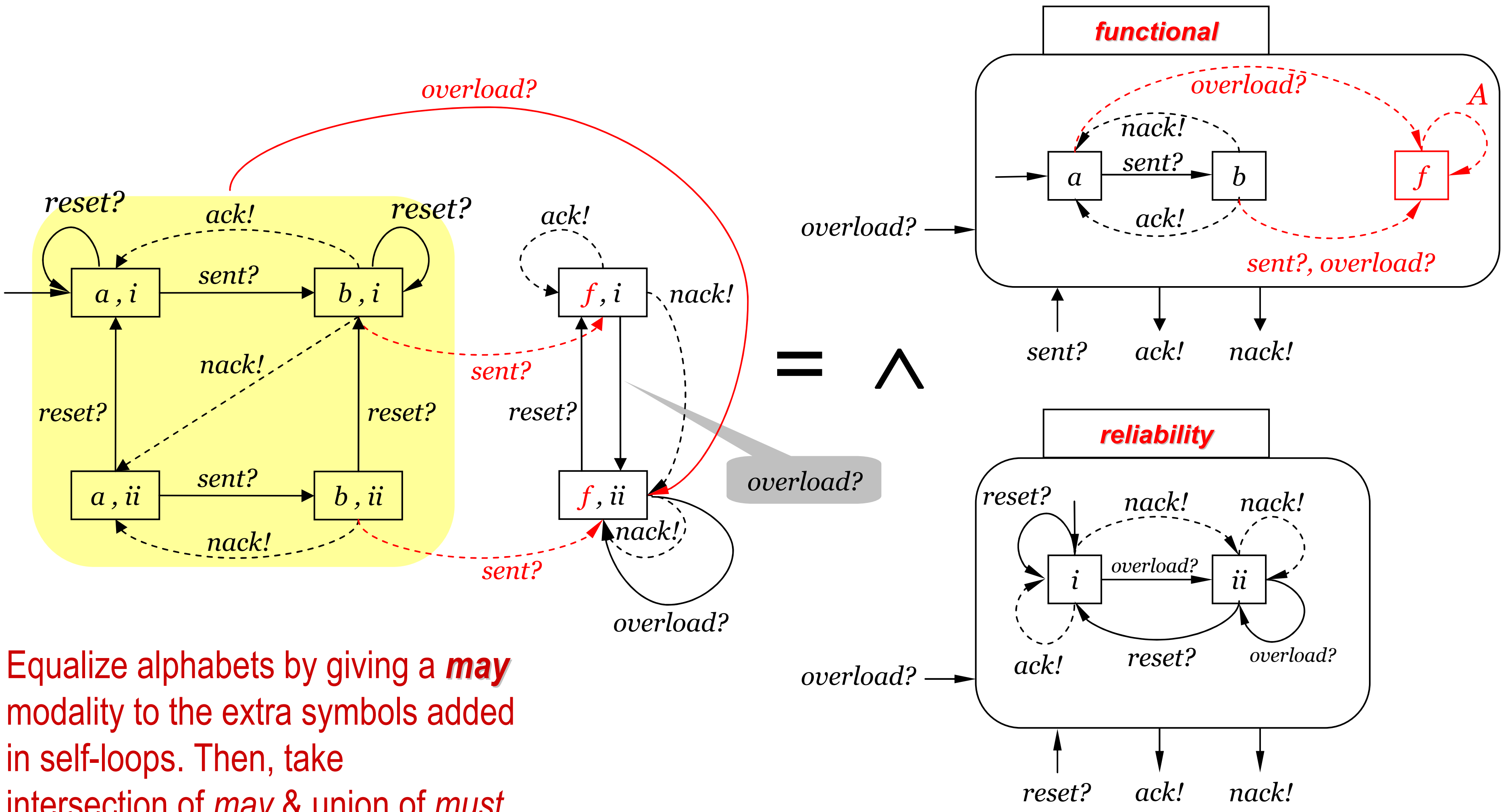
We do not know how to do without modalities

We do not know if modalities are the *only* solution

The *network* ex. as an MA

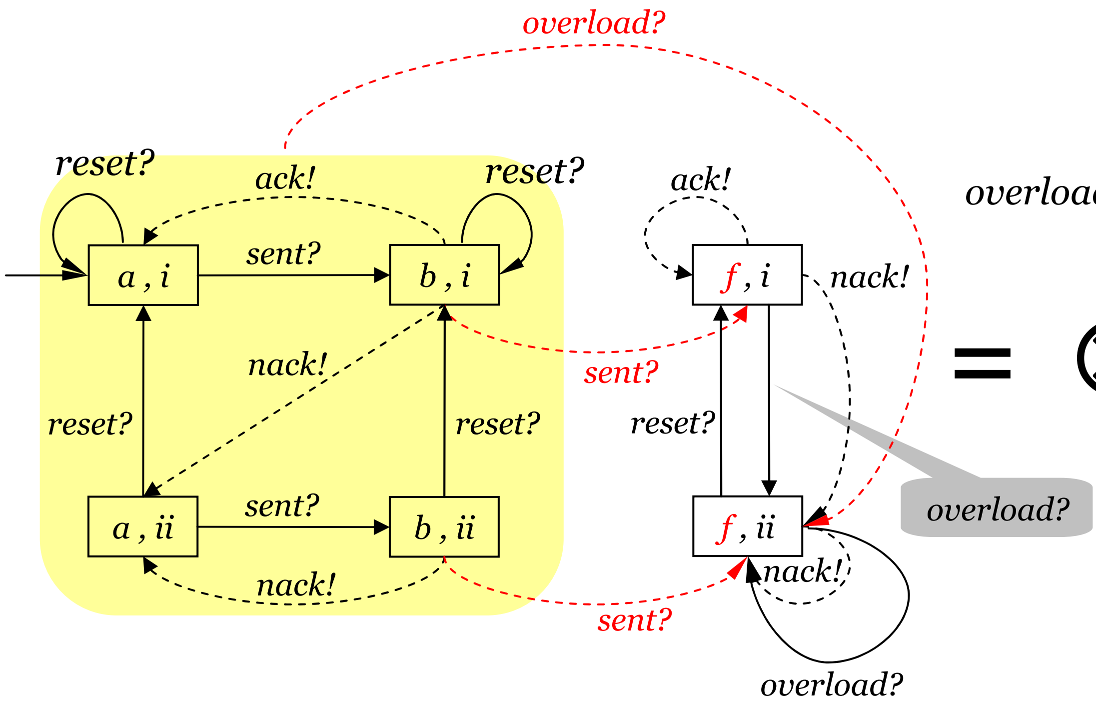


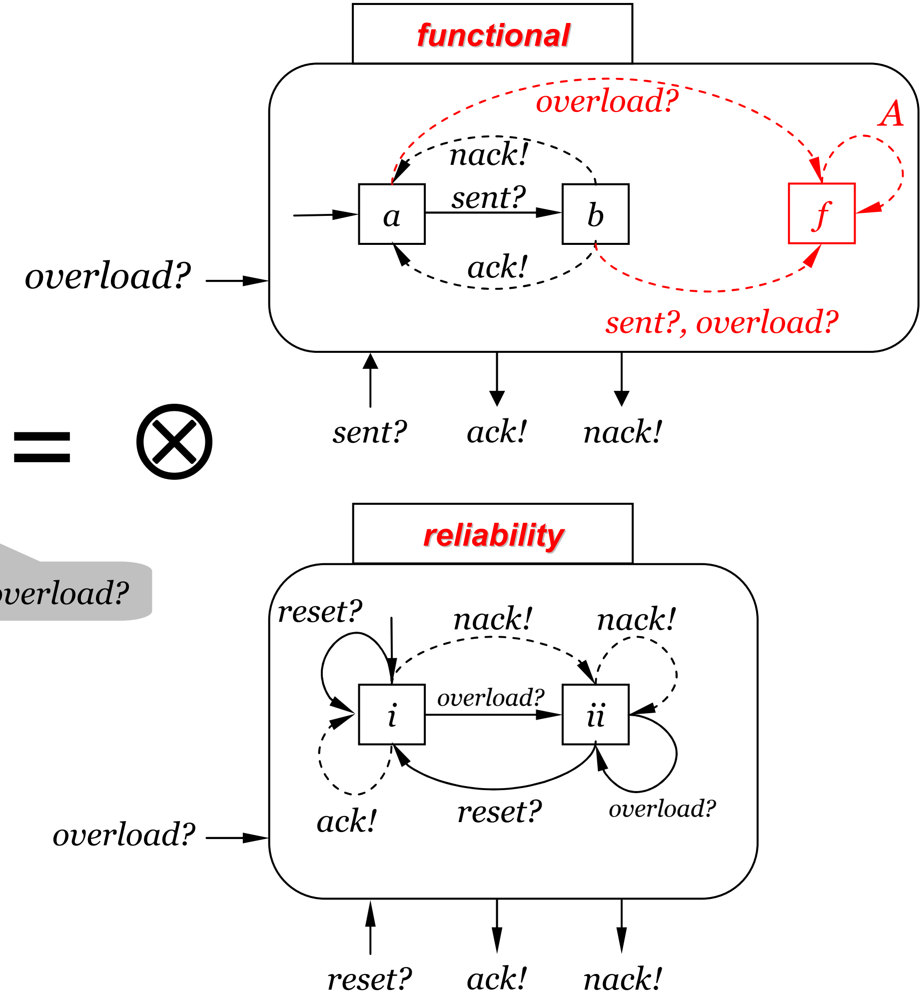
Conjunction of contracts



Equalize alphabets by giving a ***may*** modality to the extra symbols added in self-loops. Then, take intersection of *may* & union of *must*

Parallel Composition of contracts





Equalize alphabets by giving a ***must*** modality to the extra symbols added in self-loops. Then, take <u>intersection of *may* & intersection of *must*</u>

Assume/Guarantee reasoning using residuation

Assume/Guarantee Reasoning

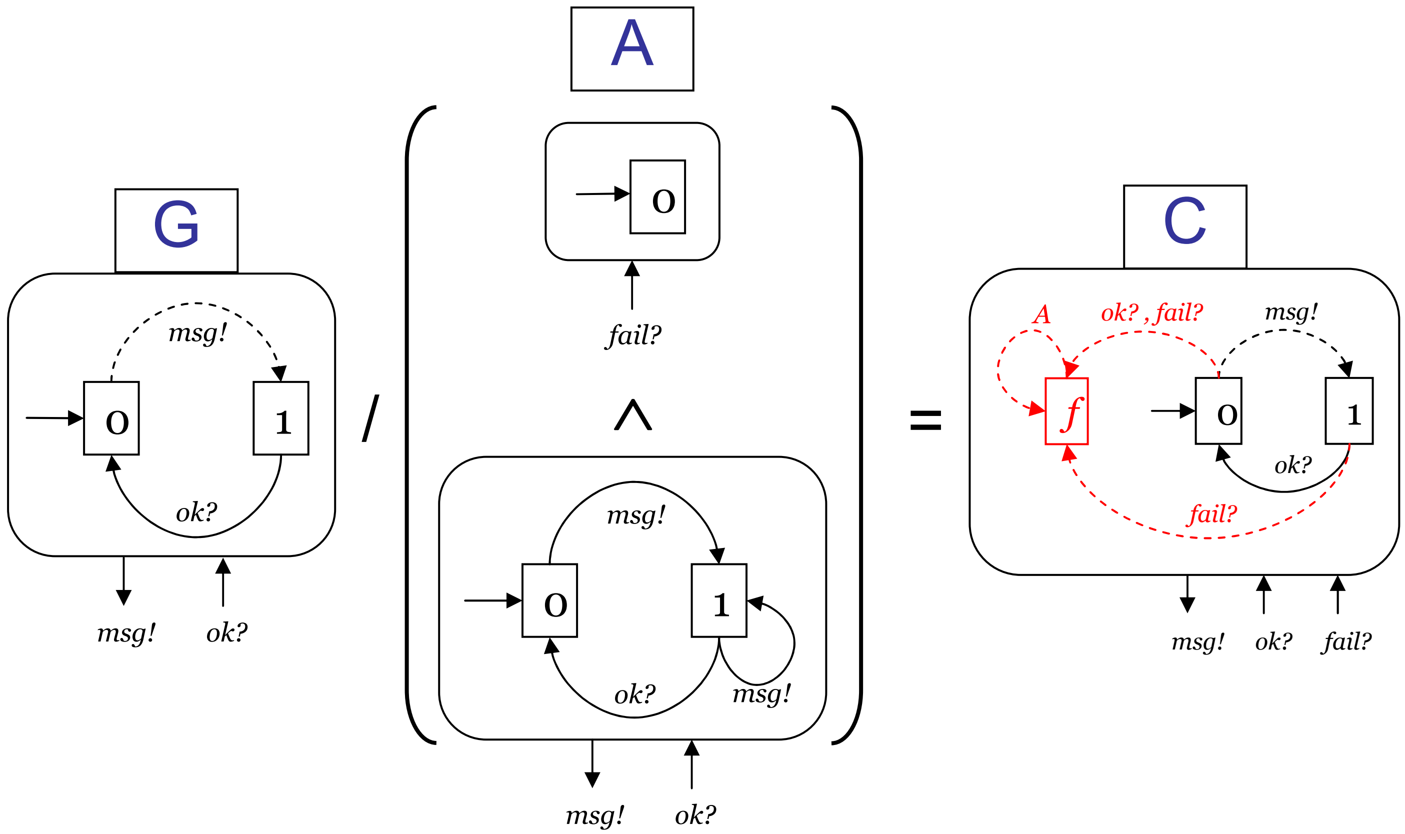
- Contracts = pairs (A,G)
- Express all operations on contracts in terms of this pair. Typically:
  - for $\wedge$: $A_1 \cup A_2$, $G_1 \cap G_2$
  - for $\otimes$: $(A_1 \cap A_2) \cup \neg(G_1 \cap G_2), G_1 \cap G_2$
- Problem: when using non-modal frameworks (e.g., sets of traces, IAs…) the handling of alphabet equalization is inadequate and leads to absurd results – reason for this is the same as before.
- This approach fails!

Assume/Guarantee Reasoning

- Contracts = pairs (A,G)
- Express all operations on contracts in terms of this pair. Typically:
  - for $\wedge$: $A_1 \cup A_2$, $G_1 \cap G_2$
  - for $\otimes$: $(A_1 \cap A_2) \cup \neg(G_1 \cap G_2), G_1 \cap G_2$
- Problem: when using non-modal frameworks (e.g., sets of traces, IAs…) the handling of alphabet equalization is inadequate and leads to absurd results – reason for this is the same as before.
- This approach fails!

- Instead, handle $C = G/A$
  - $M \models C$ ensures that $\forall \mathbf{E} \models \mathbf{A} \Rightarrow \mathbf{M} \otimes \mathbf{E} \models \mathbf{G}$
- Express $\wedge$ and $\otimes$ using C's
- Drawback:
  - in $C = C_1 \wedge C_2$ or $C = C_1 \otimes C_2$, the interpretation of C as a pair (A,G) is lost
  - Re-decomposing C as G/A is possible but not unique and has no interesting solution

C=G/A


G
msg!
0
1
ok?
msg!
ok?
/
A
0
fail?
∧
msg!
0
1
ok?
msg!
msg!
ok?
=
C
A
ok? , fail?
msg!
f
0
1
ok?
fail?
msg!
ok?
fail?

Conclusions

- Modal Automata and their variants offer all the needed services for an interface theory
- Simple and elegant theories
- Modalities seem essential in two issues:
  - dealing with unequal alphabets of actions
  - allowing for residuals
- Of course, modalities can also be used in practice to express progress constraints [Larsen], see also Harel's LSCs
- Side services for free
- Further issues:
  - complexity
  - getting modal extensions of synchronous systems