Residual for Component Specifications

Jean-Baptiste Raclet

Team S4, IRISA, INRIA Rennes

PhD Defense - December 18, 2007

Motivation

- Component-based design improves **readability** and **reliability**
- Crucial question : how to characterize reuse of a component ?
  - Reuse at the **signature** level:
    $\Rightarrow$ Syntactic

  vs.

  - Reuse at the **behavioral** level.
    $\Rightarrow$ Semantic

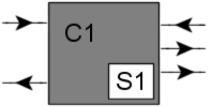
The quotient operation (1/2)

- Component = a pair $(\mathcal{C}, \mathcal{S})$ such that $\mathcal{C} \models \mathcal{S}$

The quotient operation (1/2)

- Component = a pair $(\mathcal{C}, \mathcal{S})$ such that $\mathcal{C} \models \mathcal{S}$

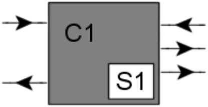
- Behavioral reuse :



The quotient operation (1/2)

- Component = a pair $(\mathcal{C}, \mathcal{S})$ such that $\mathcal{C} \models \mathcal{S}$

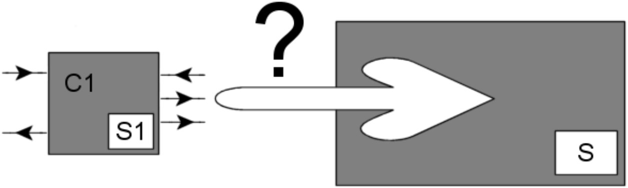
- Behavioral reuse :



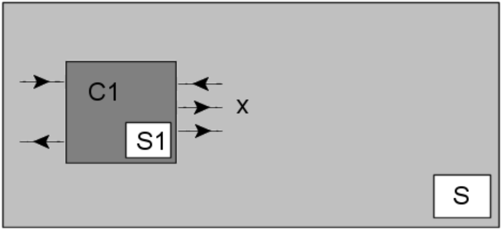
The quotient operation (1/2)

- Component = a pair $(\mathcal{C}, \mathcal{S})$ such that $\mathcal{C} \models \mathcal{S}$

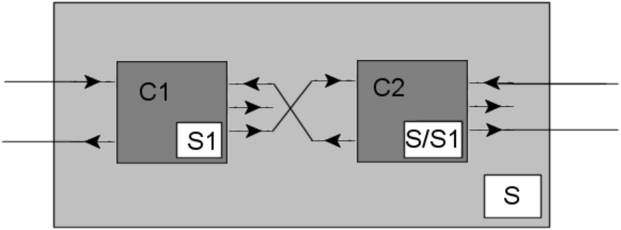
- Behavioral reuse :



The quotient operation (2/2)



The quotient operation (2/2)



Characteristic property :

$$\mathcal{C}_2 \models \mathcal{S}/\mathcal{S}_1 \;\Leftrightarrow\; \forall \mathcal{C}_1 . [\, \mathcal{C}_1 \models \mathcal{S}_1 \Rightarrow \mathcal{C}_1 \times \mathcal{C}_2 \models \mathcal{S} \,]$$

Related works

- Equation solving : $C \times X \sim G$
  - CCS processes (*Larsen, Xinxin, Parrow, ...*)
  - Languages (*Petrenko, Yevtushenko, Sangiovanni-Vincentelli, ...*)
  - Automata, Mealy machines (*von Bochmann, Merlin, ...*)
- Behavioral types (*de Alfaro, Henzinger, ...*)
- Controller synthesis
- Quotient of mu-calculus formulas (*Arnold, Vincent, Walukiewicz*) :

$$\mathrm{P} \models \psi/\varphi \Leftrightarrow \exists \mathrm{C}.[\,\mathrm{C} \models \varphi \;\wedge\; \mathrm{P} \times \mathrm{C} \models \psi\,]$$

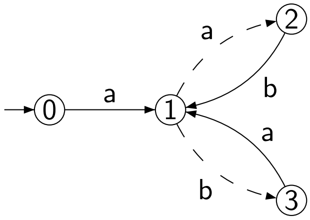
Related works

- Equation solving : $C \times X \sim G$
  - CCS processes (*Larsen, Xinxin, Parrow, ...*)
  - Languages (*Petrenko, Yevtushenko, Sangiovanni-Vincentelli, ...*)
  - Automata, Mealy machines (*von Bochmann, Merlin, ...*)
- Behavioral types (*de Alfaro, Henzinger, ...*)
- Controller synthesis
- Quotient of mu-calculus formulas (*Arnold, Vincent, Walukiewicz*) :

$$\mathrm{P} \models \neg((\neg\psi)/\varphi) \Leftrightarrow \forall \mathrm{C}.[\, \mathrm{C} \models \varphi \Rightarrow \mathrm{P} \times \mathrm{C} \models \psi \,]$$

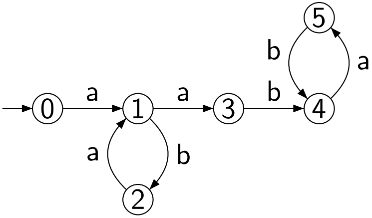
Modal automata (MA) as specifications

- $\mathcal{S} = (Q, q^0, \Delta, may, must)$ with $may, must : Q \rightarrow \mathcal{P}(\Sigma)$
- $must(q) \subseteq may(q)$
- example :

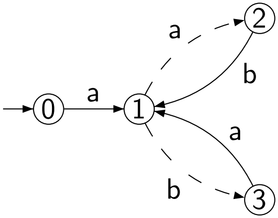


The satisfaction relation $\models$

- $\mathcal{C} = (R, r^0, \delta)$ (with $out(r) := \{a \in \Sigma.\ \delta(r, a)$ defined $\})$
- $must(q) \subseteq out(r) \subseteq may(q)$
- example :



$\models$

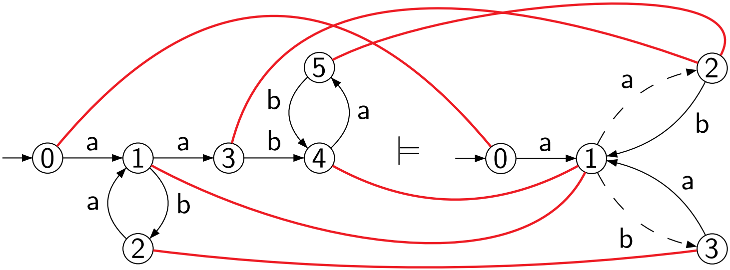


The satisfaction relation $\models$

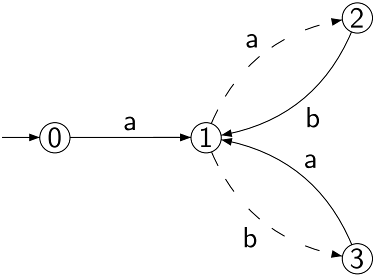
- $\mathcal{C} = (R, r^0, \delta)$ (with $out(r) := \{a \in \Sigma.\ \delta(r, a) \text{ defined }\})$

- $must(q) \subseteq out(r) \subseteq may(q)$

- example :

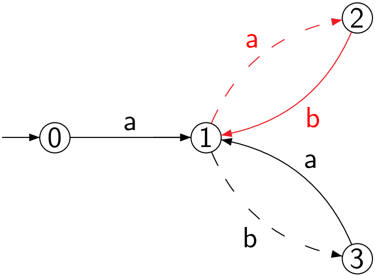


How to build a model of a MA



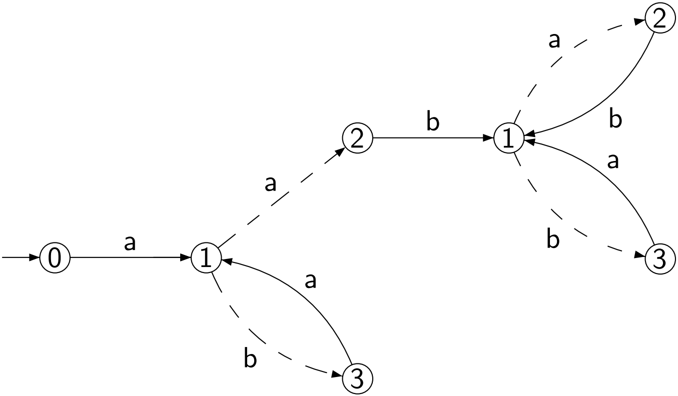
Unfold, ...

How to build a model of a MA



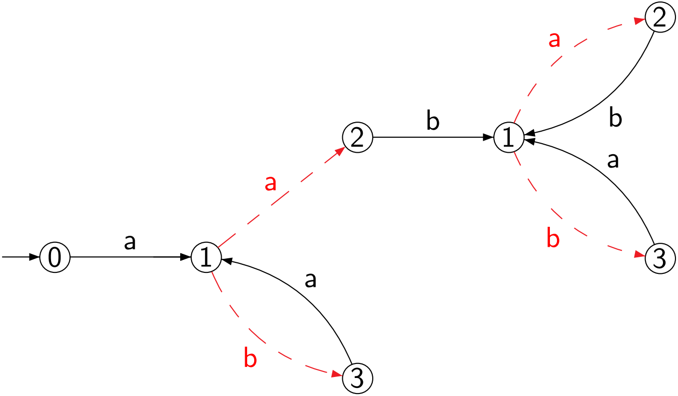
Unfold, ...

How to build a model of a MA



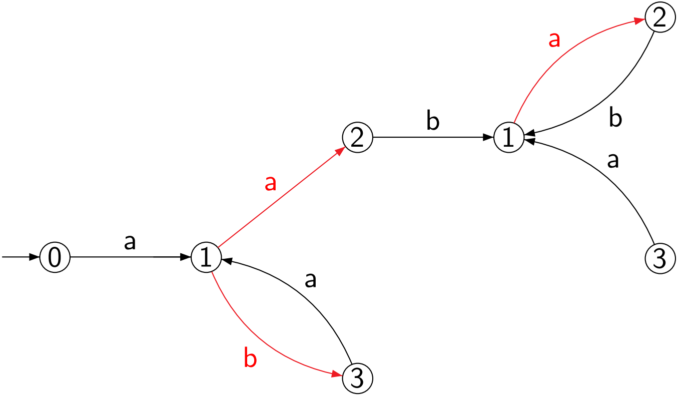
Unfold, ...

How to build a model of a MA



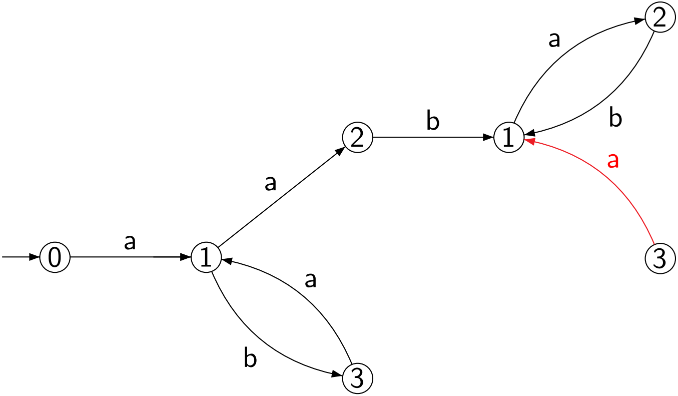
Unfold, cut (or not), ...

How to build a model of a MA



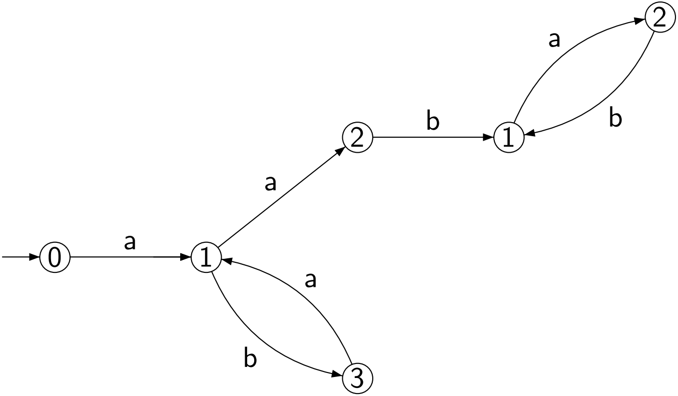
Unfold, cut (or not), ...

How to build a model of a MA



Unfold, cut (or not) and clean

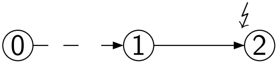
How to build a model of a MA



Unfold, cut (or not) and clean

Pseudo modal automata (pMA)

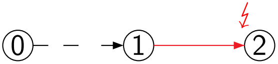
- To represent inconsistency, we let $must(q) \not\subseteq may(q)$ be possible ie. an event can both be required and forbidden
- Reduction $()^{\downarrow}$: $pMA \rightarrow MA$ such that : $\mathcal{C} \models \mathcal{S} \;\Leftrightarrow\; \mathcal{C} \models (\mathcal{S})^{\downarrow}$



A state with inconsistency can't belong to a simulation relation stating $\models$ (ie. be a state s.t. $must(q) \subseteq out(r) \subseteq may(q)$).

Pseudo modal automata (pMA)

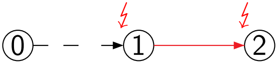
- To represent inconsistency, we let $must(q) \not\subseteq may(q)$ be possible ie. an event can both be required and forbidden
- Reduction $()^{\downarrow}$: $pMA \rightarrow MA$ such that : $\mathcal{C} \models \mathcal{S} \;\Leftrightarrow\; \mathcal{C} \models (\mathcal{S})^{\downarrow}$



A state with inconsistency can't belong to a simulation relation stating $\models$ (ie. be a state s.t. $must(q) \subseteq out(r) \subseteq may(q)$).

Pseudo modal automata (pMA)

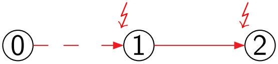
- To represent inconsistency, we let $must(q) \not\subseteq may(q)$ be possible ie. an event can both be required and forbidden
- Reduction $()^{\downarrow}$: $pMA \rightarrow MA$ such that : $\mathcal{C} \models \mathcal{S} \;\Leftrightarrow\; \mathcal{C} \models (\mathcal{S})^{\downarrow}$



A state with inconsistency can't belong to a simulation relation stating $\models$ (ie. be a state s.t. $must(q) \subseteq out(r) \subseteq may(q)$).

Pseudo modal automata (pMA)

- To represent inconsistency, we let $must(q) \not\subseteq may(q)$ be possible ie. an event can both be required and forbidden
- Reduction $()^{\downarrow} : pMA \rightarrow MA$ such that : $\mathcal{C} \models \mathcal{S} \;\Leftrightarrow\; \mathcal{C} \models (\mathcal{S})^{\downarrow}$



A state with inconsistency can't belong to a simulation relation stating $\models$ (ie. be a state s.t. $must(q) \subseteq out(r) \subseteq may(q)$).

Pseudo modal automata (pMA)

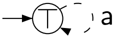
- To represent inconsistency, we let $must(q) \not\subseteq may(q)$ be possible ie. an event can both be required and forbidden
- Reduction $()^{\downarrow}$: $pMA \rightarrow MA$ such that : $\mathcal{C} \models \mathcal{S} \ \Leftrightarrow \ \mathcal{C} \models (\mathcal{S})^{\downarrow}$

0

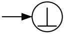
A state with inconsistency can't belong to a simulation relation stating $\models$ (ie. be a state s.t. $must(q) \subseteq out(r) \subseteq may(q)$).

Two particular pseudo modal automata

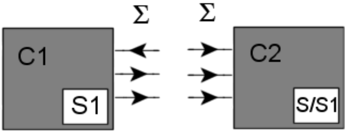
- $\mathcal{S}_\top = (\{\top\}, \top, \Delta, may, must)$ with $must(\top) = \emptyset$ and $may(\top) = \Sigma$ and $\Delta(\top, a) = \top$ for all $a \in \Sigma$



- $\mathcal{S}_\perp = (\{\perp\}, \perp, \Delta, may, must)$ with $must(\perp) = \Sigma$ and $may(\perp) = \emptyset$ and $\Delta(\perp, a)$ undefined for all $a \in \Sigma$



Synchronous product of implementations



- $\mathcal{C}_1 \times \mathcal{C}_2$ recognizes the language $\mathcal{L}(\mathcal{C}_1) \cap \mathcal{L}(\mathcal{C}_2)$.
- Given $\mathcal{C}_1 = (R_1, r_1^0, \delta_1)$ and $\mathcal{C}_2 = (R_2, r_2^0, \delta_2)$:

  $\mathcal{C}_1 \times \mathcal{C}_2 = (R_1 \times R_2, (r_1^0, r_2^0), \delta)$ with:

$$\delta((r_1, r_2), a) = (r_1', r_2') \text{ iff } \left\{ \begin{array}{lcl} \delta_1(r_1, a) & = & r_1' \\ \delta_2(r_2, a) & = & r_2' \end{array} \right.$$

Quotient of modal automata

- Given $\mathcal{S} = (Q, q^0, \Delta, may, must)$ and $\mathcal{S}_1 = (Q_1, q_1^0, \Delta_1, may_1, must_1)$
- $\mathcal{S}/\mathcal{S}_1 = ((Q \times Q_1) \cup \{\bot, \top\}, (q^0, q_1^0), \Delta_/, may_/, must_/)$

$\mathcal{S}$ **(global spec.)**	$\mathcal{S}_1$ **(reused part)**	$\mathcal{S}/\mathcal{S}_1$ **(residual spec.)**
$q \xrightarrow{a} q'$	$q_1 \xrightarrow{a} q_1'$	$(q, q_1) \xrightarrow{a} (q', q_1')$
$q \xrightarrow{a} q'$	$(q_1 \overset{a}{\dashrightarrow} q_1'$ or $q_1 \overset{a}{\not\rightarrow})$	$(q, q_1) \xrightarrow{a} \bot$
$q \overset{a}{\not\rightarrow}$	$(q_1 \xrightarrow{a} q_1'$ or $q_1 \overset{a}{\dashrightarrow} q_1')$	$(q, q_1) \overset{a}{\not\rightarrow}$
$q \overset{a}{\not\rightarrow}$	$q_1 \overset{a}{\not\rightarrow}$	$(q, q_1) \overset{a}{\dashrightarrow} \top$
$q \overset{a}{\dashrightarrow} q'$	$q_1 \overset{a}{\not\rightarrow}$	$(q, q_1) \overset{a}{\dashrightarrow} \top$
$q \overset{a}{\dashrightarrow} q'$	$(q_1 \xrightarrow{a} q_1'$ or $q_1 \overset{a}{\dashrightarrow} q_1')$	$(q, q_1) \overset{a}{\dashrightarrow} (q', q_1')$

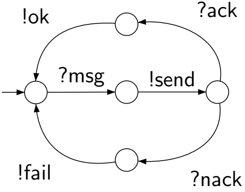
Proposition :

$$\mathcal{C}_2 \models \mathcal{S}/\mathcal{S}_1 \iff \forall \mathcal{C}_1.[\mathcal{C}_1 \models \mathcal{S}_1 \Rightarrow \mathcal{C}_1 \times \mathcal{C}_2 \models \mathcal{S}\,]$$

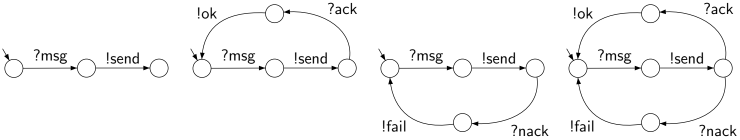
- Complexity : $\mathcal{O}(|\Sigma| \times |\mathcal{S}| \times |\mathcal{S}_1|)$

Expressivity of modal automata

- Logical fragment equivalent : conjunctive ν-calculus
- Limit of expressivity : local liveness
  Example:

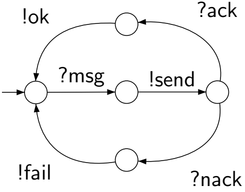


Every message sent must be acknowledged either positively or negatively.

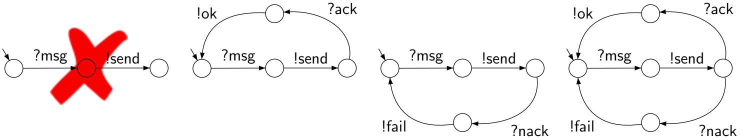


Expressivity of modal automata

- Logical fragment equivalent : conjunctive ν-calculus
- Limit of expressivity : local liveness
  Example:

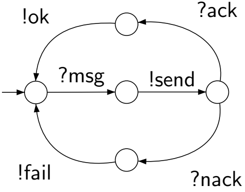


Every message sent must be acknowledged either positively or negatively.

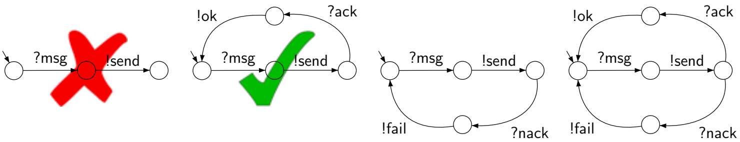


Expressivity of modal automata

- Logical fragment equivalent : conjunctive ν-calculus
- Limit of expressivity : local liveness
  Example:

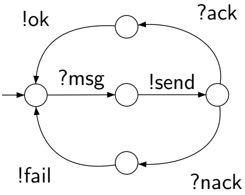


Every message sent must be acknowledged either positively or negatively.

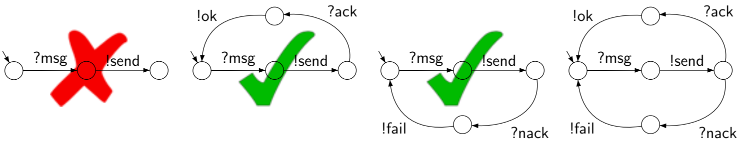


Expressivity of modal automata

- Logical fragment equivalent : conjunctive ν-calculus
- Limit of expressivity : local liveness
  Example:

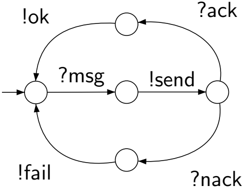


Every message sent must be acknowledged either positively or negatively.

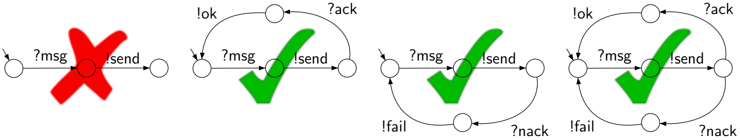


Expressivity of modal automata

- Logical fragment equivalent : conjunctive ν-calculus
- Limit of expressivity : local liveness
  Example:

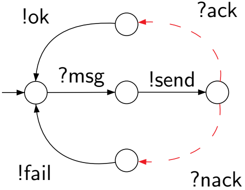


Every message sent must be acknowledged either positively or negatively.

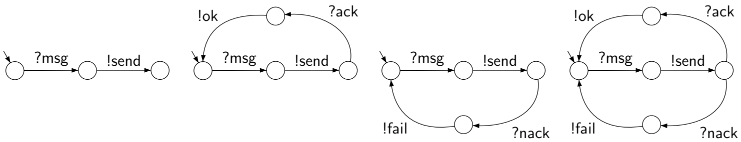


Expressivity of modal automata

- Logical fragment equivalent : conjunctive ν-calculus
- Limit of expressivity : local liveness
  Example:

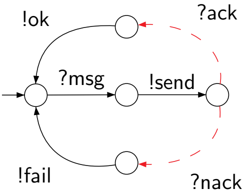


Every message sent must be acknowledged either positively or negatively.

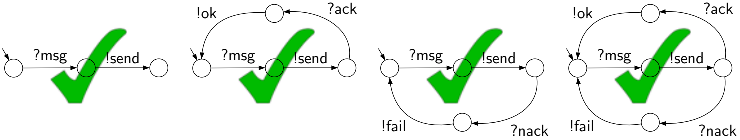


Expressivity of modal automata

- Logical fragment equivalent : conjunctive ν-calculus
- Limit of expressivity : local liveness
  Example:

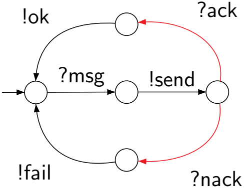


Every message sent must be acknowledged either positively or negatively.

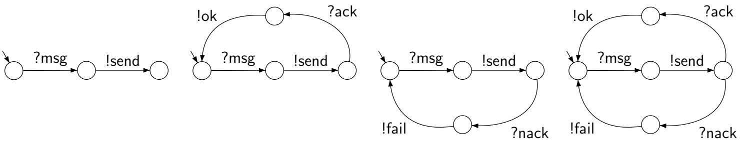


Expressivity of modal automata

- Logical fragment equivalent : conjunctive ν-calculus
- Limit of expressivity : local liveness
  Example:

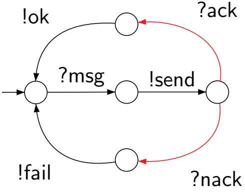


Every message sent must be acknowledged either positively or negatively.

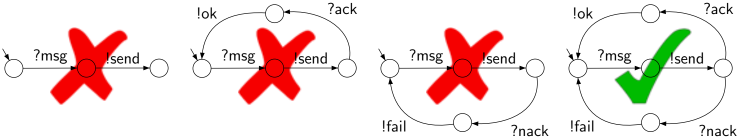


Expressivity of modal automata

- Logical fragment equivalent : conjunctive ν-calculus
- Limit of expressivity : local liveness
  Example:



Every message sent must be acknowledged either positively or negatively.



Expressivity of modal automata

- We let: $Acc(q) = \{X \in \wp(\Sigma) \text{ st. } must(q) \subseteq X \subseteq may(q)\}$.

Expressivity of modal automata

- We let: $Acc(q) = \{X \in \wp(\Sigma) \text{ st. } must(q) \subseteq X \subseteq may(q)\}$.
- When $may(q) = \{ack, nack\}$ and $must(q) = \emptyset$, we have:

$$Acc(q) = \{\emptyset, \{ack\}, \{nack\}, \{ack, nack\}\}$$

We would like to specify:

$$Acc(q) = \{\{ack\}, \{nack\}, \{ack, nack\}\}$$

Expressivity of modal automata

- We let: $Acc(q) = \{X \in \wp(\Sigma) \text{ st. } must(q) \subseteq X \subseteq may(q)\}$.
- When $may(q) = \{ack, nack\}$ and $must(q) = \emptyset$, we have:

$$Acc(q) = \{\emptyset, \{ack\}, \{nack\}, \{ack, nack\}\}$$

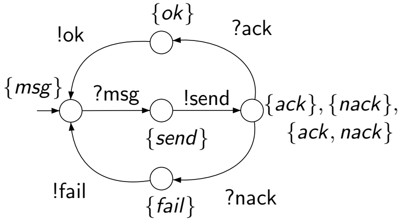
  We would like to specify:

$$Acc(q) = \{\{ack\}, \{nack\}, \{ack, nack\}\}$$

- We remove the closure under intersection
  (Hennessy - *Acceptance Trees*)
  $\Rightarrow$ acceptance automata

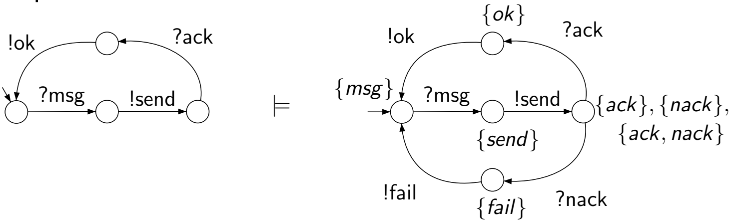
Acceptance automata (AA) as specifications

- $\mathcal{S} = (Q, q^0, \Delta, Acc)$ with $Acc : Q \to \wp(\wp(\Sigma))$
- Example:



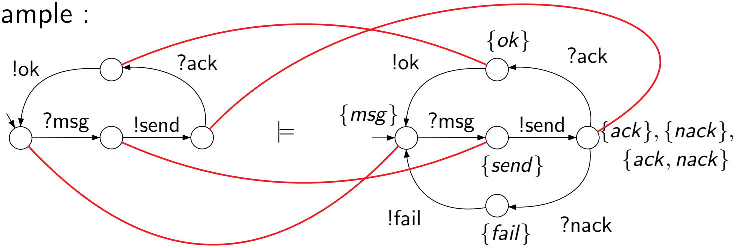
The satisfaction relation $\models$

- $\mathcal{C} = (R, r^0, \delta)$ (with $out(r) := \{a \in \Sigma.\ \delta(r,a) \text{ defined }\}$)
- $out(r) \in Acc(q)$
- example :



The satisfaction relation $\models$

- $\mathcal{C} = (R, r^0, \delta)$ (with $out(r) := \{a \in \Sigma.\ \delta(r, a) \text{ defined }\}$)
- $out(r) \in Acc(q)$
- example :

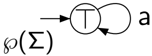


Pseudo acceptance automata (pAA)

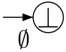
- To represent inconsistency, we let $Acc(q) = \emptyset$ be possible
- A state with incoherently specified events can't belong to a simulation relation stating $\models$ (ie. be a state s.t. $out(r) \in Acc(q)$)
- Reduction $()^{\downarrow}$: $pAA \rightarrow AA$ such that : $\mathcal{C} \models \mathcal{S} \;\Leftrightarrow\; \mathcal{C} \models (\mathcal{S})^{\downarrow}$.

Two particular pAA

- $\mathcal{S}_\top = (\{\top\}, \top, \Delta, Acc)$ with $Acc(\top) = \wp(\Sigma)$ and $\Delta(\top, a) = \top$ for all $a \in \Sigma$



- $\mathcal{S}_\bot = (\{\bot\}, \bot, \Delta, Acc)$ with $Acc(\bot) = \emptyset$ and $\Delta(\bot, a)$ undefined for all $a \in \Sigma$



Quotient of acceptance automata

- Given : $\mathcal{S} = (Q, q^0, \Delta, Acc)$ and $\mathcal{S}_1 = (Q_1, q_1^0, \Delta_1, Acc_1)$
- $\mathcal{S}/\mathcal{S}_1 = ((Q \times Q_1) \cup \{\top\}, (q^0, q_1^0), \Delta_/, Acc_/)$
  - $Acc_/((q, q_1)) =$

    $$\{Y \in \wp(\Sigma) \ st. \ \forall X \in Acc_1(q_1), X \cap Y \in Acc(q)\}$$

  - For all $a \in Y$ where $Y \in Acc_/((q, q_1))$:

$$\Delta_/((q, q_1), a) = \begin{cases} (q', q_1') & \text{si } \Delta(q, a) = q' \text{ et } \Delta_1(q_1, a) = q_1' \\ \top & \text{sinon} \end{cases}$$

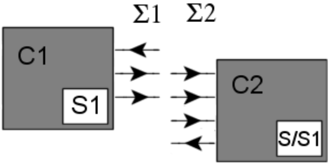
Proposition :

$$\mathcal{C}_2 \models \mathcal{S}/\mathcal{S}_1 \ \Leftrightarrow \ \forall \mathcal{C}_1.[\,\mathcal{C}_1 \models \mathcal{S}_1 \Rightarrow \mathcal{C}_1 \times \mathcal{C}_2 \models \mathcal{S}\,]$$

- Complexity : $\mathcal{O}(2^{(|\Sigma|+1)} \times |\mathcal{S}| \times |\mathcal{S}_1|)$

Mixed product as composition operation

- Mixed product : $\mathcal{C}_1 \sqcap\!\!\sqcap \mathcal{C}_2$



Proposition :

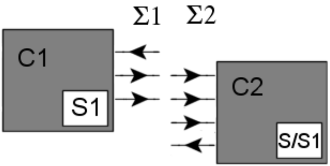
Given $\mathcal{S}(\Sigma)$, $\mathcal{S}_1(\Sigma_1)$ and Σ_2 such that $\Sigma_1 \cup \Sigma_2 = \Sigma$:

$$\mathcal{C}_2 \models \mathcal{S} /_{\sqcap\!\!\sqcap} \mathcal{S}_1 \Leftrightarrow \forall \mathcal{C}_1 . [\mathcal{C}_1 \models \mathcal{S}_1 \Rightarrow \mathcal{C}_1 \sqcap\!\!\sqcap \mathcal{C}_2 \models \mathcal{S}]$$

with $\mathcal{S} /_{\sqcap\!\!\sqcap} \mathcal{S}_1 = \Pi_{\Sigma_2}(\mathcal{S} / \tau_{\Sigma}(\mathcal{S}_1))$.

Internalization of events in composition

- Mixed product + projection : $(\mathcal{C}_1 \sqcap \mathcal{C}_2)_{\downarrow\Sigma}$



$+ \qquad \Sigma_1 \cup \Sigma_2 \rightsquigarrow \Sigma$

Proposition :

Given $\mathcal{S}(\Sigma)$, $\mathcal{S}_1(\Sigma_1)$ *and* Σ_2 *such that* $\Sigma \subseteq \Sigma_1 \cup \Sigma_2$ *:*

$$\mathcal{C}_2 \models \mathcal{S}/_{\sqcap\downarrow}\mathcal{S}_1 \Leftrightarrow \forall \mathcal{C}_1.[\mathcal{C}_1 \models \mathcal{S}_1 \Rightarrow (\mathcal{C}_1 \sqcap \mathcal{C}_2)_{\downarrow\Sigma} \models \mathcal{S}]$$

with $\mathcal{S}/_{\sqcap\downarrow}\mathcal{S}_1 = \amalg_{\Sigma_2}(\mathcal{S}_{\uparrow(\Sigma_1\cup\Sigma_2)}/\tau_{\Sigma_1\cup\Sigma_2}(\mathcal{S}_1))$.

Interactions between $\mathcal{C}_1$ and $\mathcal{C}_2$

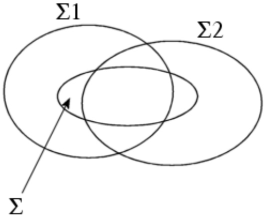
Proposition :

Given $\mathcal{S}(\Sigma)$, $\mathcal{S}_1(\Sigma_1)$ *and* Σ_2 *such that* $\Sigma \subseteq \Sigma_1 \cup \Sigma_2$ *:*

$$\mathcal{C}_2 \models \mathcal{S}/_{\sqcap\downarrow}\mathcal{S}_1 \Leftrightarrow \forall \mathcal{C}_1.[\mathcal{C}_1 \models \mathcal{S}_1 \Rightarrow (\mathcal{C}_1 \sqcap \mathcal{C}_2)_{\downarrow\Sigma} \models \mathcal{S}]$$

with $\mathcal{S}/_{\sqcap\downarrow}\mathcal{S}_1 = \amalg_{\Sigma_2}(\mathcal{S}_{\uparrow(\Sigma_1\cup\Sigma_2)}/\tau_{\Sigma_1\cup\Sigma_2}(\mathcal{S}_1))$.

- $\Sigma_1 \cap \Sigma_2$: control



Interactions between $\mathcal{C}_1$ and $\mathcal{C}_2$

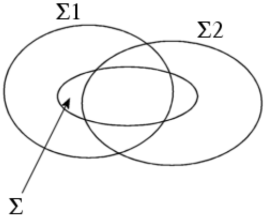
Proposition :

Given $\mathcal{S}(\Sigma)$, $\mathcal{S}_1(\Sigma_1)$ and Σ_2 such that $\Sigma \subseteq \Sigma_1 \cup \Sigma_2$:

$$\mathcal{C}_2 \models \mathcal{S}/_{\sqcap\downarrow}\mathcal{S}_1 \Leftrightarrow \forall \mathcal{C}_1.[\mathcal{C}_1 \models \mathcal{S}_1 \Rightarrow (\mathcal{C}_1 \sqcap \mathcal{C}_2)_{\downarrow\Sigma} \models \mathcal{S}]$$

with $\mathcal{S}/_{\sqcap\downarrow}\mathcal{S}_1 = \amalg_{\Sigma_2}(\mathcal{S}_{\uparrow(\Sigma_1\cup\Sigma_2)}/\tau_{\Sigma_1\cup\Sigma_2}(\mathcal{S}_1))$.

- $\Sigma_1 \cap \Sigma_2$: control
- $\Sigma \setminus \Sigma_1$: add behaviors



Interactions between $\mathcal{C}_1$ and $\mathcal{C}_2$

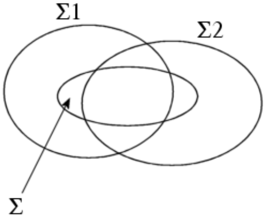
Proposition :

Given $\mathcal{S}(\Sigma)$, $\mathcal{S}_1(\Sigma_1)$ and Σ_2 such that $\Sigma \subseteq \Sigma_1 \cup \Sigma_2$:

$$\mathcal{C}_2 \models \mathcal{S}/_{\sqcap\downarrow}\mathcal{S}_1 \Leftrightarrow \forall \mathcal{C}_1.[\mathcal{C}_1 \models \mathcal{S}_1 \Rightarrow (\mathcal{C}_1 \sqcap \mathcal{C}_2)_{\downarrow\Sigma} \models \mathcal{S}]$$

with $\mathcal{S}/_{\sqcap\downarrow}\mathcal{S}_1 = \amalg_{\Sigma_2}(\mathcal{S}_{\uparrow(\Sigma_1\cup\Sigma_2)}/\tau_{\Sigma_1\cup\Sigma_2}(\mathcal{S}_1))$.

- $\Sigma_1 \cap \Sigma_2$: control
- $\Sigma \setminus \Sigma_1$: add behaviors
- $\Sigma_1 \setminus \Sigma$: hide events



Applications (1/2)

- Behavioral reuse

Applications (1/2)

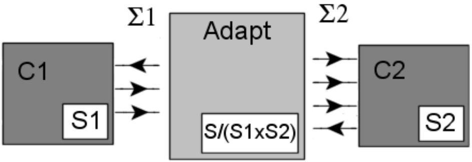
- Behavioral reuse
  - If $\mathcal{S}/\mathcal{S}_1 = \mathcal{S}_\perp$ then else
  - Optimize reuse with greatest model

Applications (1/2)

- Behavioral reuse
  - If $\mathcal{S}/\mathcal{S}_1 = \mathcal{S}_\perp$ then  else 
  - Optimize reuse with greatest model
- Protocol converter synthesis

Applications (1/2)

- Behavioral reuse
  - If $\mathcal{S}/\mathcal{S}_1 = \mathcal{S}_\perp$ then else
  - Optimize reuse with greatest model
- Protocol converter synthesis
  - Given $(\mathcal{C}_1, \mathcal{S}_1)$, $(\mathcal{C}_2, \mathcal{S}_2)$ and $\mathcal{S}$ a safety/liveness property
  - Compute $\mathcal{S}/(\mathcal{S}_1 \otimes \mathcal{S}_2)$
  - $\mathit{Adapt} \models \mathcal{S}/(\mathcal{S}_1 \otimes \mathcal{S}_2) \Rightarrow \mathcal{C}_1 \times \mathcal{C}_2 \times \mathit{Adapt} \models \mathcal{S}$



Applications (2/2)

- Contract based design

Applications (2/2)

- Contract based design
  - Contract = a pair of specifications (A, G)
  - Satisfaction of a contract :
    $\mathcal{C} \models G/A \Leftrightarrow \forall \mathcal{E}.[\, \mathcal{E} \models A \Rightarrow \mathcal{C} \times \mathcal{E} \models G \,]$

Future works

- Generalization to a family of specification formalism
- Generalization to an abstract product
- Implementation

Thanks for your attention

Questions ?