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:
    - Syntactic
  - vs.

  - Reuse at the **behavioral** level.
    - Semantic
The quotient operation (1/2)

- Component = a pair \((C, S)\) such that \(C \models S\)
The quotient operation (1/2)

- Component = a pair \((C, S)\) such that \(C \models S\)

Behavioral reuse:

- Behavioral reuse:
The quotient operation (1/2)

- Component = a pair \((C, S)\) such that \(C \models S\)

- Behavioral reuse:

```
\[
\begin{array}{c}
C1 \\
S1
\end{array}
\]
```

```
\[
\begin{array}{c}
C1 \\
S1
\end{array}
\]
```

```
\[
\begin{array}{c}
S
\end{array}
\]
```
The quotient operation (1/2)

- Component = a pair \((C, S)\) such that \(C \models S\)

Behavioral reuse:

- A diagram showing the interaction of \(C\) and \(S\), with an arrow indicating the flow from \(C\) to \(S\).
The quotient operation (2/2)
The quotient operation (2/2)

Characteristic property:

\[ C_2 \models S/S_1 \iff \forall C_1. [C_1 \models S_1 \Rightarrow C_1 \times C_2 \models S] \]
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):
  \[
P \models \psi / \varphi \iff \exists C. [ C \models \varphi \land P \times C \models \psi ]
\]
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):
  \[ P \models \neg((\neg \psi)/\varphi) \iff \forall C. [ C \models \varphi \Rightarrow P \times C \models \psi ] \]
Modal automata (MA) as specifications

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

- \( \text{must}(q) \subseteq \text{may}(q) \)

- example:

```
\[ \begin{array}{c}
\text{0} \rightarrow a \\
\text{1} \rightarrow b \\
\text{2} \rightarrow \text{a}
\end{array} \]
```
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:

\[
\begin{array}{c c c c c c c}
0 & \rightarrow & 1 & \rightarrow & 3 & \rightarrow & 4 & \rightarrow & 5
\end{array}
\]

\[
\begin{array}{c c c c c c c}
0 & \rightarrow & 1 & \rightarrow & 2 & \rightarrow & 0
\end{array}
\]
The satisfaction relation $\models$

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

- $\text{must}(q) \subseteq \text{out}(r) \subseteq \text{may}(q)$

- Example:

```
|   |   |   |   |   |
0  | a | a | b | a | 1 |
  | a |   | b | b |   |
  |   | a |   |   |   |
0  | a |   |   | a |   |
  | a | b |   |   |   |
  |   | a | b |   | 3 |
```

Jean-Baptiste Raclet (Team S4)
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 \( \text{must}(q) \not\subseteq \text{may}(q) \) be possible ie. an event can both be required and forbidden
- Reduction (\( \downarrow \)) : \( pMA \rightarrow MA \) such that : \( C \models S \iff C \models (S)\downarrow \)

\[
\begin{array}{c}
\text{①} \quad \text{②}
\end{array}
\]

A state with inconsistency can’t belong to a simulation relation stating \( \models \) (ie. be a state s.t. \( \text{must}(q) \subseteq \text{out}(r) \subseteq \text{may}(q) \)).
Pseudo modal automata (pMA)

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

\[\begin{array}{c}
0 \quad \rightarrow \quad 1 \quad \rightarrow \quad 2
\end{array}\]

A state with inconsistency can’t belong to a simulation relation stating $\models$ (i.e. be a state s.t. $\text{must}(q) \subseteq \text{out}(r) \subseteq \text{may}(q)$).
Pseudo modal automata (pMA)

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

A state with inconsistency can’t belong to a simulation relation stating $\models$ (ie. be a state s.t. $\text{must}(q) \subseteq \text{out}(r) \subseteq \text{may}(q)$).
Pseudo modal automata (pMA)

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

\[ \begin{array}{c}
\circ \quad \downarrow \quad 1 \quad \downarrow \quad 2 \\
\end{array} \]

A state with inconsistency can’t belong to a simulation relation stating $\models$ (i.e. be a state s.t. $\text{must}(q) \subseteq \text{out}(r) \subseteq \text{may}(q)$).
Pseudo modal automata (pMA)

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

\[0\]

A state with inconsistency can’t belong to a simulation relation stating \( \models \) (i.e. be a state s.t. \( \text{must}(q) \subseteq \text{out}(r) \subseteq \text{may}(q) \)).
Two particular pseudo modal automata

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

- $S_\bot = (\{\bot\}, \bot, \Delta, \text{may}, \text{must})$ with $\text{must}(\bot) = \Sigma$ and $\text{may}(\bot) = \emptyset$ and $\Delta(\bot, a)$ undefined for all $a \in \Sigma$
Synchronous product of implementations

- $C_1 \times C_2$ recognizes the language $\mathcal{L}(C_1) \cap \mathcal{L}(C_2)$.

- Given $C_1 = (R_1, r_1^0, \delta_1)$ and $C_2 = (R_2, r_2^0, \delta_2)$:

  
  $C_1 \times C_2 = (R_1 \times R_2, (r_1^0, r_2^0), \delta)$ with:

  $$\delta((r_1, r_2), a) = (r_1', r_2') \text{ iff } \begin{cases} 
  \delta_1(r_1, a) = r_1' \\
  \delta_2(r_2, a) = r_2'
  \end{cases}$$
Quotient of modal automata

- Given $S = (Q, q^0, \Delta, \textit{may}, \textit{must})$ and $S_1 = (Q_1, q_1^0, \Delta_1, \textit{may}_1, \textit{must}_1)$
- $S/S_1 = ((Q \times Q_1) \cup \{\bot, \top\}, (q^0, q_1^0), \Delta/, \textit{may}/, \textit{must}/)$

<table>
<thead>
<tr>
<th>$S$ (global spec.)</th>
<th>$S_1$ (reused part)</th>
<th>$S/S_1$ (residual spec.)</th>
</tr>
</thead>
<tbody>
<tr>
<td>$q \xrightarrow{a} q'$</td>
<td>$q_1 \xrightarrow{a} q_1'$</td>
<td>$(q, q_1) \xrightarrow{a} (q', q_1')$</td>
</tr>
<tr>
<td>$q \xrightarrow{a} q'$</td>
<td>$(q_1 \xrightarrow{a} q_1'$ or $q_1 \xrightarrow{a}$)</td>
<td>$(q, q_1) \xrightarrow{a} \bot$</td>
</tr>
<tr>
<td>$q \xrightarrow{a}$</td>
<td>$(q_1 \xrightarrow{a} q_1'$ or $q_1 \xrightarrow{a}$)</td>
<td>$(q, q_1) \xrightarrow{a}$</td>
</tr>
<tr>
<td>$q \xrightarrow{a}$</td>
<td>$q_1 \xrightarrow{a}$</td>
<td>$(q, q_1) \xrightarrow{a} \top$</td>
</tr>
<tr>
<td>$q \xrightarrow{a} q'$</td>
<td>$q_1 \xrightarrow{a}$</td>
<td>$(q, q_1) \xrightarrow{a} \top$</td>
</tr>
<tr>
<td>$q \xrightarrow{a} q'$</td>
<td>$(q_1 \xrightarrow{a} q_1'$ or $q_1 \xrightarrow{a}$)</td>
<td>$(q, q_1) \xrightarrow{a} (q', q_1')$</td>
</tr>
</tbody>
</table>

Proposition:

$C_2 \models S/S_1 \iff \forall C_1. [ C_1 \models S_1 \Rightarrow C_1 \times C_2 \models S ]$

- Complexity: $O(|\Sigma| \times |S| \times |S_1|)$
Expressivity of modal automata

- Logical fragment equivalent: conjunctive $\nu$-calculus
- Limit of expressivity: local liveness

Example:

![Automaton Diagram]

Every message sent must be acknowledged either positively or negatively.
Expressivity of modal automata

- Logical fragment equivalent: conjunctive $\nu$-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 $\nu$-calculus
- Limit of expressivity: local liveness

Example:

```
?msg !send ?ack

?msg !send !ok ?ack

?msg !send !fail ?nack
```

*Every message sent must be acknowledged either positively or negatively.*
Expressivity of modal automata

- Logical fragment equivalent: conjunctive $\nu$-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 \( \nu \)-calculus
- Limit of expressivity: local liveness

Example:

\[
\begin{array}{c}
\text{?msg} & \text{!send} & \text{?ack} \\
\text{!ok} & \text{?msg} & \text{!send} \\
\text{!fail} & \text{?nack} \\
\end{array}
\]

Every message sent must be acknowledged either positively or negatively.
Expressivity of modal automata

- Logical fragment equivalent: conjunctive \( \nu \)-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 $\nu$-calculus
- Limit of expressivity: local liveness

Example:

```
?msg
!send
?ack
!ok
!fail
```

Every message sent must be acknowledged either positively or negatively.
Expressivity of modal automata

- Logical fragment equivalent: conjunctive $\nu$-calculus
- Limit of expressivity: local liveness

Example:

```
?msg !send ?ack

!ok

?msg !send

!fail

?nack
```

Every message sent must be acknowledged either positively or negatively.
Expressivity of modal automata

- Logical fragment equivalent: conjunctive $\nu$-calculus
- Limit of expressivity: local liveness

Example:

Every message sent must be acknowledged either positively or negatively.
Expressivity of modal automata

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

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

- When \( \text{may}(q) = \{\text{ack}, \text{nack}\} \) and \( \text{must}(q) = \emptyset \), we have:

\[
\text{Acc}(q) = \{\emptyset, \{\text{ack}\}, \{\text{nack}\}, \{\text{ack}, \text{nack}\}\}
\]

We would like to specify:

\[
\text{Acc}(q) = \{\{\text{ack}\}, \{\text{nack}\}, \{\text{ack}, \text{nack}\}\}
\]
Expressivity of modal automata

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

- When \( \text{may}(q) = \{\text{ack}, \text{nack}\} \) and \( \text{must}(q) = \emptyset \), we have:
  \[
  \text{Acc}(q) = \{\emptyset, \{\text{ack}\}, \{\text{nack}\}, \{\text{ack}, \text{nack}\}\}
  \]

  We would like to specify:
  \[
  \text{Acc}(q) = \{\{\text{ack}\}, \{\text{nack}\}, \{\text{ack}, \text{nack}\}\}
  \]

- We remove the closure under intersection
  (Hennessy - Acceptance Trees)
  \[\Rightarrow\] acceptance automata
Acceptance automata (AA) as specifications

- \( S = (Q, q^0, \Delta, Acc) \) with \( Acc : Q \rightarrow \wp(\wp(\Sigma)) \)

- Example:

\[
\begin{align*}
\{msg\} & \xrightarrow{?msg} \{send\} \\
\{send\} & \xrightarrow{!send} \{ack\}, \{nack\}, \{ack, nack\} \\
\{ok\} & \xrightarrow{?ack} \\
\{fail\} & \xrightarrow{!fail} \{nack\}
\end{align*}
\]
The satisfaction relation $\models$

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

- $\text{out}(r) \in \text{Acc}(q)$

- example:

```plaintext
?msg !send

\{msg\} \{send\} \{ack\}, \{nack\}, \{ack, nack\}
```

![Diagram](image-url)
The satisfaction relation \( \models \)

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

- \( \text{out}(r) \in \text{Acc}(q) \)

example:

\[
\begin{align*}
\text{!ok} & \quad \text{?ack} \\
\text{?msg} & \quad \text{!send} \\
\text{!ok} & \quad \{ \text{ok} \} \\
\text{?msg} & \quad \{ \text{msg} \} \\
\text{!send} & \quad \{ \text{send} \} \\
\text{!fail} & \quad \{ \text{fail} \}
\end{align*}
\]
Pseudo acceptance automata (pAA)

- To represent inconsistency, we let $\text{Acc}(q) = \emptyset$ be possible.

- A state with incoherently specified events can’t belong to a simulation relation stating $\models (\text{ie. be a state s.t. } \text{out}(r) \in \text{Acc}(q))$.

- Reduction ($\downarrow$) : $pAA \rightarrow AA$ such that : $C \models S \iff C \models (S)\downarrow$. 

Two particular pAA

- $S_\top = (\{\top\}, \top, \Delta, Acc)$ with $Acc(\top) = \emptyset(\Sigma)$ and $\Delta(\top, a) = \top$ for all $a \in \Sigma$

  ![Top Diagram]

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

  ![Bottom Diagram]
Quotient of acceptance automata

- Given: \( S = (Q, q^0, \Delta, Acc) \) and \( S_1 = (Q_1, q^0_1, \Delta_1, Acc_1) \)

- \( S/S_1 = ((Q \times Q_1) \cup \{\top\}, (q^0, q^0_1), \Delta/\), Acc/\) 
  - \( Acc/((q, q_1)) = \)
    \[ \{ Y \in \wp(\Sigma) \text{ st. } \forall X \in Acc_1(q_1), X \cap Y \in Acc(q) \} \]
  - 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:

\[ C_2 \models S/S_1 \iff \forall C_1. [C_1 \models S_1 \Rightarrow C_1 \times C_2 \models S] \]

- Complexity: \( O(2^{(|\Sigma|+1)} \times |S| \times |S_1|) \)
Mixed product as composition operation

- Mixed product: $C_1 \sqcap C_2$

**Proposition:**

Given $S(\Sigma)$, $S_1(\Sigma_1)$ and $\Sigma_2$ such that $\Sigma_1 \cup \Sigma_2 = \Sigma$:

$C_2 \models S \sqcap S_1 \iff \forall C_1. [C_1 \models S_1 \Rightarrow C_1 \sqcap C_2 \models S]$

with $S \sqcap S_1 = \Pi_{\Sigma_2}(S/\tau_\Sigma(S_1))$. 

Jean-Baptiste Raclet (Team S4)
Residual for Component Specifications
PhD Defense 20 / 26
Internalization of events in composition

- Mixed product + projection: \((C_1 \cap C_2) \downarrow \Sigma\)

\[
\begin{array}{c}
\Sigma_1 \quad \Sigma_2 \\
C_1 & S_1 \\
\downarrow & \\
C_2 & S/S_1
\end{array}
\]

\[+ \quad \Sigma_1 \cup \Sigma_2 \rightsquigarrow \Sigma\]

**Proposition:**

Given \(S(\Sigma), S_1(\Sigma_1)\) and \(\Sigma_2\) such that \(\Sigma \subseteq \Sigma_1 \cup \Sigma_2\):

\[C_2 \models S/\downarrow \downarrow S_1 \iff \forall C_1.[C_1 \models S_1 \Rightarrow (C_1 \cap C_2) \downarrow \Sigma \models S]\]

with \(S/\downarrow \downarrow S_1 = \Pi_{\Sigma_2}(S_{\uparrow (\Sigma_1 \cup \Sigma_2)}/\tau_{\Sigma_1 \cup \Sigma_2}(S_1))\).
Interactions between $C_1$ and $C_2$

**Proposition:**

Given $S(\Sigma)$, $S_1(\Sigma_1)$ and $\Sigma_2$ such that $\Sigma \subseteq \Sigma_1 \cup \Sigma_2$:

$$C_2 \models S/\sqcap S_1 \iff \forall C_1. [C_1 \models S_1 \Rightarrow (C_1 \sqcap C_2) \downarrow \Sigma \models S]$$

with $S/\sqcap S_1 = \Pi_{\Sigma_2} (S_{\uparrow(\Sigma_1 \cup \Sigma_2)}/\tau_{\Sigma_1 \cup \Sigma_2}(S_1))$.

- $\Sigma_1 \cap \Sigma_2$ : control
Interactions between $C_1$ and $C_2$

Proposition:

Given $S(\Sigma)$, $S_1(\Sigma_1)$ and $\Sigma_2$ such that $\Sigma \subseteq \Sigma_1 \cup \Sigma_2$:

$$C_2 \models S/\downarrow S_1 \iff \forall C_1.[C_1 \models S_1 \Rightarrow (C_1 \cap C_2)\downarrow \Sigma \models S]$$

with $S/\downarrow S_1 = \Pi \Sigma_2 (S_{\uparrow(\Sigma_1 \cup \Sigma_2)}/\tau \Sigma_1 \cup \Sigma_2 (S_1))$.

- $\Sigma_1 \cap \Sigma_2$: control
- $\Sigma \setminus \Sigma_1$: add behaviors
Interactions between $C_1$ and $C_2$

**Proposition:**

Given $S(\Sigma)$, $S_1(\Sigma_1)$ and $\Sigma_2$ such that $\Sigma \subseteq \Sigma_1 \cup \Sigma_2$:

$$C_2 \models S/\triangledown S_1 \iff \forall C_1. [C_1 \models S_1 \Rightarrow (C_1 \triangledown C_2)\downarrow \Sigma \models S]$$

with $S/\triangledown S_1 = \Pi_{\Sigma_2}(S_{\uparrow(\Sigma_1 \cup \Sigma_2)}/\tau_{\Sigma_1 \cup \Sigma_2}(S_1))$.

- $\Sigma_1 \cap \Sigma_2$ : control
- $\Sigma \setminus \Sigma_1$ : add behaviors
- $\Sigma_1 \setminus \Sigma$ : hide events
Applications (1/2)

- Behavioral reuse
Applications (1/2)

- Behavioral reuse
  - If $S/S_1 = S_\perp$ then 😞 else 😊
  - Optimize reuse with greatest model
Applications (1/2)

- Behavioral reuse
  - If $S/S_1 = S_\perp$ then 😞 else 😊
  - Optimize reuse with greatest model
- Protocol converter synthesis
Applications (1/2)

- Behavioral reuse
  - If $S/S_1 = S_\perp$ then 😞 else 😊
  - Optimize reuse with greatest model

- Protocol converter synthesis
  - Given $(C_1, S_1), (C_2, S_2)$ and $S$ a safety/liveness property
  - Compute $S/(S_1 \otimes S_2)$
  - $Adapt \models S/(S_1 \otimes S_2) \Rightarrow C_1 \times C_2 \times Adapt \models S$
Applications (2/2)

- Contract based design
Contract based design

- Contract = a pair of specifications $(A, G)$
- Satisfaction of a contract:

$$C \models G/A \iff \forall E. \left[ E \models A \Rightarrow C \times E \models G \right]$$
Future works

- Generalization to a family of specification formalism
- Generalization to an abstract product
- Implementation
Thanks for your attention

Questions ?