Towards design of reliable synchronous reactive systems

S. Chabane, R. Ameur-Boulifa, M. Mezghiche

MSR 2019

November 13, 2019
Angers, France
Outline

1. Context
2. Modelling
3. Composition
4. Validation
5. Conclusion and ongoing works
**CONTEXT**

**EMBEDDED SYSTEMS**

Increasing complexity, Integrate software and hardware with critical functionalities.

**CHALLENGE**

- Need of **reuse**, **reliability**, **high quality** designs.
Reusability

Component-based approach

Each part of a system is designed as independent and well-defined component that are quite generic to be reused.
Reusability

Component-based approach

Each part of a system is designed as independent and well-defined component that are quite generic to be reused.
Each part of a system is designed as independent and well-defined component that are quite generic to be reused.
Reusability

Component-based approach

Each part of a system is designed as independent and well-defined component that are quite generic to be reused.
Reusability

Component-based approach

Each part of a system is designed as independent and well-defined component that are quite generic to be reused.
Formal Methods

“Formal methods are mathematical approaches to system development which support the rigorous specification, design and verification of computer systems.”

“Formal methods exploit the power of mathematical notation and mathematical proofs”.

High Quality of Designs
High quality of designs

Towards design of reliable synchronous systems

**CORRECTNESS BY CONSTRUCTION**

Designs that improve reliability and aim to obtain correct by construction systems.
**Synchronous system**

![Diagram](image)

- **Synchronous component**: All executions are driven by a clock, when each reaction are simultaneous and instantaneous at each tick of the clock.

**Parameters**

- **Latency** is the delay from input into a system to desired output.
- **Memory** is the maximum amount of information that can be stored.
Modelling

- I/O automata suitable for modelling synchronous systems.

**SR-model (Synchronous reactive model)**

Given a component $C$ with the parameters $M$ and $L$, The SR-model of $C$ is an I/O-automaton $A = \langle S, s_0, \Sigma, \rightarrow \rangle$ associated to $M$ and $L$, such that:

$$\forall s \in S. \quad \text{Determinist}(s) \land \text{Receptiv}(s, M, L) \land \text{OutAfterCount}(s, L) \land \text{IdleAfterCount}(s, L)$$

\[\rightarrow\]

Implementation as $\text{Build-prim}(M, L)$.
Behavior of automaton

\[ \langle \rangle \quad \langle 1 \rangle \quad \langle 2; 1 \rangle \quad \langle 2 \rangle \]

The component is empty. waiting for an input.
Behavior of automaton

- Only one data in the component.
- Available memory.
- No output.

S. Chabane, R. Ameur-Boulifa, M. Mez
Towards design of reliable synchronous systems
Behavior of automaton

- Only one data in the component.
- Available memory.
- Output is possible.
Behavior of automaton

- Full memory.
- Output is possible.
Composition

- Cascade composition.
- Parallel composition.
**Composition (Cont.)**

- **Cascade composition ✓**
- **Parallel composition.**

\[
\begin{align*}
C &= C_1 + C_2 + C_3 \\
M &= M_1 + M_2 + M_3 \\
L &= L_1 + L_2 + L_3
\end{align*}
\]

\[ S. Chabane, R. Ameur-Boulifa, M. MezC \] Towards design of reliable synchronous systems
I/O automata theory provides a composition operator $A_1 \parallel A_2$ based on the following rules:

\[ \forall s_1 \in A_1, s_2 \in A_2, \begin{cases} (s_1 \xrightarrow{\alpha/o} s'_1 \land s_2 \xrightarrow{i/\beta} s'_2) \\ \lor \\ (s_1 \xrightarrow{\alpha/\bar{o}} s'_1 \land s_2 \xrightarrow{\bar{i}/\beta} s'_2) \end{cases} \]

\[ (s_1, s_2) \xrightarrow{\alpha/\beta} (s'_1, s'_2) \quad \text{Otherwise undefined} \]
Composition rules

- I/O automata theory provides a composition operator $A_1 \parallel A_2$ based on the following rules:

\[
\forall s_1 \in A_1, s_2 \in A_2, \\
\begin{cases}
(s_1 \xrightarrow{\alpha/o} s_1' \land s_2 \xrightarrow{i/\beta} s_2') \\
\lor \\
(s_1 \xrightarrow{\alpha/\bar{o}} s_1' \land s_2 \xrightarrow{\bar{i}/\beta} s_2')
\end{cases}
\]

- Otherwise undefined

- But
Case 1

M = 2, L = 4

M = 4, L = 6

S. Chabane, R. Ameur-Boulifa, M. Mez
**Problem**

Case 1

Adding bubbles

The first component is full and cannot produce an output.

The second component can receive new inputs.
Problem (Cont.)

Case 2

$M_2 = 2 \quad L_2 = 2$

$M_1 = 2 \quad L_1 = 4$

$M = 4 \quad L = 6$
Case 2

The first component is ready to output.
The second component is full.

ADD DELAYS/INCREASING OF LATENCY
Illustration (Cont.)

But

Result of composition

≠

Expected model
Pipeline is not the baseline behavior of the corresponding component.

- Need to fix the problem of break of pipeline.
- Proposal of new compositional rules that describe reliably the behavior of a pipeline.

**Our Proposition**

Extension of I/O automata composition to improve the reliability of SR-models.
Safe composition denoted $A_1^{+m_1} \prod_{(l_1,l_2,M_1,M_2)} A_2^{+m_2}$ of two SR–models $A_1 = \langle S_1, s_{01}, \Sigma, \rightarrow \rangle$ associated to $M_1$ and $L_1$, and $A_2 = \langle S_2, s_{02}, \Sigma, \rightarrow \rangle$ associated to $M_2$ and $L_2$. is an SR–model $A = \langle S_1 \times S_2, (s_{01}, s_{02}), \Sigma, \rightarrow \rangle$ such as:

$m_1 = \begin{cases} 
\min(L_1 - M_1, M_2) & \text{if } M_1 < L_1 \\
0 & \text{otherwise}
\end{cases}$

$m_2 = \begin{cases} 
L_2 - M_2 & \text{if } M_2 < L_2 \\
0 & \text{otherwise}
\end{cases}$
### Safe Composition (Cont.)

#### Implementation as Build-comp($M_1, L_1, M_2, L_2$).

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>$s_1 \xrightarrow{i/\alpha^<em>} s_1' \land s_2 \xrightarrow{\beta^</em>/o} s_2'$</td>
<td>$s_1 \xrightarrow{i/\alpha^<em>} s_1' \land s_2 \xrightarrow{\beta^</em>/o} s_2'$</td>
<td></td>
</tr>
<tr>
<td>$s_1 \xrightarrow{i/o} s_1', s_2 \xrightarrow{i/o} s_2'$</td>
<td>$s_1 \xrightarrow{i/\alpha^<em>} s_1' \land s_2 \xrightarrow{\beta^</em>/o} s_2'$</td>
<td></td>
</tr>
<tr>
<td>$s_1 \xrightarrow{i/\bar{o}} s_1', s_2 \xrightarrow{i/o} s_2'$</td>
<td>$s_1 \xrightarrow{i/\alpha^<em>} s_1' \land s_2 \xrightarrow{\beta^</em>/o} s_2'$</td>
<td></td>
</tr>
<tr>
<td>$(s_1, s_2) \xrightarrow{i/o} (s_1', s_2')$</td>
<td>$(s_1, s_2) \xrightarrow{i/o} (s_1', s_2')$</td>
<td></td>
</tr>
</tbody>
</table>

- if $idle(s_2)$ then
  - $(s_1, s_2) \xrightarrow{i/\bar{o}} (s_1, s_2)$
- else
  - $s_1 \xrightarrow{i/o} s_1', s_2 \xrightarrow{i/\bar{o}} s_2'$ if $delayed(s_2)$
  - $(s_1, s_2) \xrightarrow{i/\bar{o}} (s_1', s_2')$
How it works?
How it works?
How it works?
How it works?
Validation

Verification by Model checking - CADP tool

- Bisimilarity between the proposed composition and the expected model.

\[ \forall M_1, L_1, M_2, L_2 \in \mathbb{N}. \]

\[ \text{Build-comp}(M_1, L_1, M_2, L_2) \sim \text{Build-prim}(M_1 + M_2, L_1 + L_2) \]

- Verification of properties with MCL (Model checking language) formulas.

\[ [(\neg "i/\bar{o}"") * .("i/\bar{o}"){M}] < (\neg "i/\bar{o}" ) > true \]
Implementation of Algorithms

module SR = {
    proc build_prim(M:int,L:int): type_ioa ={
        ....
        /* Construction of an SR-model */
        ....
        return (s0,states,transitions); }
    proc build_comp(L1:int, M1 :int, L2:int, M2 :int):ioa={
        ....
        /* Safe composition of SR-models */
        ....
        return (s0,states,transitions); }
}

Verification by Theorem Proving - EasyCrypt

S. Chabane, R. Ameur-Boulifa, M. Mez.
lemma well_formed(A:ioa, m l:int):Determinist(A)
   \ Receptiv(A,m,l) \ OutAfterCount(A,l)/\
   IdleAfterCount(A,l).

lemma well_formed_comp:forall (m1 l1 m2 l2:int)
hoare [SR.Safe_composition:
 ( 0<m1 \ m1=M1 \ 0<l1 \ l1=L1 ) \/
 ( 0<m2 \ m2=M2 \ 0<l2 \ l2=L2 ) =>
 (well_formed SR.ioA1 m1 l1) \/
 (well_formed SR.ioA2 m2 l2) ]
 => well_formed res (m1+m2) (l1+l2)].
Conclusion

- Definition of Synchronous reactive model.
- Definition of composition of SR-models.
- Implementation of the proposed algorithms.
- Validation of the solution:
  - Validation by model-checking with CADP tool.
  - Validation by theorem proving with Easycrypt.
Ongoing works

- Complete the composition operator to:
  - Deal with parallel composition.
  - Deal with more data relationships between input data and output data of components.

- Extend our approach to other kinds of systems with different characteristics/parameters.