Model Checking Reo Connectors with mCRL2

Natallia Kokash
Introduction

- COMPAS Project
  - Formalization of Business Process Models
  - Formalization of compliance requirements
- Reo
- mCRL2
- Model checking Reo process models with mCRL2
  - Translation from Reo to mCRL2
  - Unified Control and Data Flow Analysis
- Tool support and examples
- Conclusions and Future Work
COMPAS project

- **COMPAS**
  - Compliance-driven Models, Languages, and Architectures for Services

- **Goal**
  - Ensure dynamic and on-going compliance of software services to business regulations and user requirements

- **Methodology, terminology and research results**
  - **Compliance** is conformity in fulfilling compliance requirements
  - **Compliance requirement** is a constraint or assertion that results from the interpretation of the compliance sources
  - **Compliance source** is a document that is the origin of compliance requirements (e.g., SOX, HIPAA, license)
Formal BP analysis in COMPAS

- **Business process** is a composition of activities into a structured order that implements the procedure to be followed in order to achieve a business goal.
- **Behavioral model** is a description of how an actor (e.g., stakeholder, service) acts or interacts with other actors.
- **Compliance rule** is an operative definition of a compliance requirement.
- **Goal:** Check whether compliance rules hold for formal behavioral models of business processes.
Reo coordination language

- Components (services) communicate through channels composed to complex connectors
- Semantics:
  - Constraint automata
  - Intentional automata (support for context-dependency)

<table>
<thead>
<tr>
<th>Sync, SyncDrain</th>
<th>LossySync</th>
<th>AsyncDrain</th>
<th>Fifo1</th>
</tr>
</thead>
<tbody>
<tr>
<td>A ←--→ B A ←--→ B</td>
<td>A ←--→ B A ←--→ B</td>
<td>A ←--→ B A ←--→ B</td>
<td></td>
</tr>
<tr>
<td>{A, B} circle</td>
<td>{B} circle {A} circle</td>
<td>{A} circle {A, B} circle</td>
<td></td>
</tr>
</tbody>
</table>

- Replicator
  - {A, B, C} circle
- Merger
  - {B, C} circle {A, C} circle
- Router
  - {A, C} circle {A, B} circle
Formalization of business processes

UMLADs2Reo converter

BPMN2Reo converter
Check that admissible states reject or sendFormOut are eventually reached

\[ G(request \rightarrow F (reject \cup sendFormOut)) \]
Business process analysis

- Control flow analysis
  - **Deadlock** is a situation when a service or a process stays idle waiting for resources permanently blocked by another party or messages that will never arrive.
  - **Livelock (starvation of progress)** is a situation when a process or a service is not blocked but doesn't make any progress nevertheless.
  - **Problems with synchronization** conflicts can be introduced by joining fork concurrent paths with a merge structure which in some workflow specification languages (e.g., BPMN) results into unintentional multiple activation of states that follow the merge node.

- Data flow analysis
  - Deadlocks, livelocks and problems with synchronization may appear due to improper conditions on data (e.g., while data-based choice gateway is used in BPMN).
  - Compliance requirements are data dependent.
mCRL2

- Behavioral specification language
- Associated toolset
  - Developed at TU Eindhoven (+ LaQuSo, CWI and Twente University)
- Based on the algebra of communicating processes (ACP)
- Extended with data and time
  - Built-in data types: *Bool, Nat, Pos, Int, Real*
  - Algebraic data types
    - constructors, recognition and projection functions
  - Built-in support for lists, sets and bags
  - User-defined functions (λ calculus)
- Number of industrial case studies
- [http://www.mcrl2.org/](http://www.mcrl2.org/)
mCRL2 specification language

- **Actions** are atomic events (e.g. a firing of a port or a request arrival in a Reo connector)
- **Processes** are the active entities defined as expressions over actions and other processes
  - Multiaction: $a/b$ (synchronized actions)
  - Alternative composition: $a + b$ (nondeterministic choice)
  - Sequence composition: $a.b$ ($b$ started after $a$)
  - Conditional: $\text{exp} \rightarrow a \diamond b$ (if-then-else)
  - At operator: $a^t$ (action $a$ happens at time $t$)
  - Parallel composition: $a//b$ (interleavings $a.b + b.a + a|b$)
- Actions and processes can be parametrized with **data**
  - Summation: $\sum_{d \in D} a(d)$ ($a(d_1) + a(d_2) + a(d_3)\ldots$)
mCRL2 specification language

- **Renaming:** $\rho_R(a)$ where $R$ is a set of renamings of the form $b \rightarrow c$, meaning that every occurrence of $b$ in $a$ is replaced by $c$
- **Hiding:** $\tau_H(a)$ renames all actions of $H$ in $a$ to $t$
- **Restriction (allow):** $\nabla_R(a)$ where $R$ specifies which actions are allowed to occur in $a$
- **Blocking:** $\partial_B(a)$ where $B$ is a set of actions that is not allowed to occur in $a$
- **Communication:** $\Gamma_C(p)$, where $C$ is a set of allowed communications of the form $a_0|\ldots|a_n \rightarrow c$, $n \geq 1$ which means that every group of actions $a_0|\ldots|a_n$ within a multiaction is replaced by an action $c$
Example of mCRL2 specification

Dining philosophers

eqn \( K = 2 \);
map \( K: \text{Pos} \);

act \( \text{get}, \text{put}, \text{up}, \text{down}, \text{lock}, \text{free}: \text{Pos}\#\text{Pos} \);
\( \text{eat}: \text{Pos} \);

proc
\( \text{Phil}(n: \text{Pos}) = \text{get}(n, n) \cdot \text{get}(n, \text{if} (n == K, 1, n+1)) \cdot \text{eat}(n) \cdot \text{put}(n, n) \cdot \text{put}(n, \text{if} (n == K, 1, n+1)) \cdot \text{Phil}(n) \);

\( \text{Fork}(n: \text{Pos}) = \text{sum} m: \text{Pos} \cdot \text{up}(m, n) \cdot \text{down}(m, n) \cdot \text{Fork}(n) \);

init allow( \{ \text{lock}, \text{free}, \text{eat} \},
\quad \text{comm}( \{ \text{get|up->lock, put|down->free} \},
\quad \text{Phil}(1) \ | \ ... \ | \ \text{Phil}(K) \ | \ \text{Fork}(1) \ | \ ... \ | \ \text{Fork}(K) ));
Reo to mCRL2 (Constraint automata semantics)

- Data flow observed at a channel end = action
- Synchronous channel, synchronous drain
  - $Sync = A/B.Sync$
- Non-deterministic synchronous lossy channel
  - $LossySync = (A/B + A).LossySync$
- Asynchronous drain
  - $AsyncDrain = (A + B).AsyncDrain$
- FIFO1
  - $FIFO1 = A.B.FIFO1$
  - $FullFIFO1 = B.FIFO1$
  - Alternative encoding: $FIFO1(f: Bool) = (\neg f \rightarrow A \diamond B).FIFO1(\neg f)$
- Replication node
  - $Replicator = X|Y|Z.Replicator$
- Merge node
  - $Merger = (X|Z + Y|Z).Merger$
Reo to mCRL2: Data support

\textbf{act} \(A, B: Data\)

- Synchronous channel
  \(Sync = \sum_{d \in Data} A(d) | B(d) . Sync;\)

- Synchronous drain
  \(SyncDrain = \sum_{d_1, d_2 \in Data} A(d_1) | B(d_2) . SyncDrain;\)

- Synchronous lossy channel
  \(LossySync = \sum_{d \in Data} (A(d) | B(d) + A(d)) . LossySync;\)

- Asynchronous drain
  \(AsyncDrain = \sum_{d \in Data} (A(d) + B(d)) . AsyncDrain;\)

- Filter
  \(Filter = \sum_{d \in Data} (exp(d) \rightarrow A(d) | B(d) \diamond A(d)) . Filter, \text{ where } exp(d) \text{ is a boolean expression}\)

- Transformer
  \(Transformer = \sum_{d \in Data} A(d) | B(exp(d)) . Transformer;\)

- Replication node
  \(Replicator = \sum_{d \in Data} X(d) | Y(d) | Z(d) . Replicator;\)

- Merge node
  \(Merger = \sum_{d \in Data} (X(d) | Z(d) + Y(d) | Z(d)) . Merger;\)
Reo to mCRL2: Data and time support

- **FIFO1**
  - \(\text{DataFIFO1} = \text{struct empty?isEmpty | full(e:Data)?isFull;}\)
  - \(\text{FIFO1}(f: \text{DataFIFO1}) = \sum_{d \in \text{Data}} \text{isEmpty}(f) \rightarrow A(d).\text{Fifo1}(\text{full}(d)) \cdot B(e(f)).\text{FIFO1}(\text{empty})\)

- **\(T\)-timer with off- and reset- options**
  - Reacts differently to different data inputs:
    - \(\text{DataTimer} = \text{struct reset?isReset | off?isOff | timeout | other(e:Data)?isOther}\)
  - Has two states
    - \(\text{State} = \text{struct OFF?isOFF | ON?isON}\)
  - State \(s\) (timer ON or OFF), current time \(x\), timer delay \(t\)
    - \(\text{Timer}(s: \text{State}, x: \text{Real}, t: \text{Real}) = \)
      - \(\text{isOFF}(s) \rightarrow \sum_{d \in \text{DataTimer}} \text{isOther}(d) \rightarrow A(d).\text{Timer(ON, 0, t)} + \text{isON}(s) \rightarrow ((x < t) \rightarrow \sum_{d \in \text{DataTimer}} \text{isReset}(d) \rightarrow A(d).\text{Timer(ON, 0, t)} + \text{isOff}(d) \rightarrow A(d).\text{Timer(OFF, x, t)} + \text{tick}^c_x.\text{Timer(ON, x + 1, t)}) \cdot B(\text{timeout}).\text{Timer(OFF, x, t)}\)
Reo to mCRL2: Global data types

- A connector should deal with any data items consumed by its source nodes.
- Given a set of elementary data types \( DT_1, \ldots, DT_n \) (e.g., inferred from web service interface specifications), the global data type is described as follows:
  - \( Data = \text{struct } D_1(e_1: DT_1) | \ldots | D_n(e_1: DT_n) \)
- **Join node**
  - \( \text{Join} = \sum_{d_1, d_2 \in Data} (X(d_1)|Y(d_2)|Z(\text{tuple}(d_1, d_2))).\text{Join}; \)
- For \( m \)-join node \( \text{tuple}(e_1: Data, e_2: Data, \ldots, e_m: Data) \) is added to the Data description, e.g.,
  - \( Data = \text{struct } D_1(e_1: DT_1) | \ldots | D_n(e_1: DT_n) | \text{tuple}(e_1: Data, e_2: Data) \)
- **Note:** expressions for filter and transformer channels become dependent on the structure of the Reo connector.
Reo to mCRL2: Composition

Synchronize and hide actions corresponding to the connected channels

- **Problem**
  - Reduce the size of the state space while building the LTS for the mCRL2 specification of a Reo connector

- **Idea**
  - Iterated connector construction

1. \( P_0 = \partial \text{ends of connected channels} (\Gamma \text{handshaking at Node}_1 (\text{Node}_1 \parallel \text{Sync}_1 \parallel \text{LossySync}_1 \parallel \text{LossySync}_2 \parallel \text{SyncDrain}_1)) \)
2. \( P_1 = \partial \text{ends of connected channels} (\Gamma \text{handshaking at Node}_2 (\text{Node}_2 \parallel \text{Sync}_2 \parallel \text{Sync}_3 \parallel P_0)) \)
3. \( P_2 = \partial \text{ends of connected channels} (\Gamma \text{handshaking at Node}_3 (\text{Node}_3 \parallel \text{Sync}_4 \parallel P_1)) \)
4. \( P_3 = \partial \text{ends of connected channels} (\Gamma \text{handshaking at Node}_4 (\text{Node}_4 \parallel \text{Sync}_5 \parallel P_2)) \)
Reo to mCRL2 (Intentional automata semantics)

- Two types of actions:
  - A channel end is ready to write/read ($iA$)
  - Data flow observed at a channel end ($oA$)

- Synchronous channel, synchronous drain
  - $\text{Sync}(a: \text{Bool}, b: \text{Bool}) =$
    - $\neg a \rightarrow iA. \text{Sync}(\neg a, b) +$
    - $\neg b \rightarrow iB. \text{Sync}(a, \neg b) +$
    - $(a \land b) \rightarrow oA|oB. \text{Sync}(\neg a, \neg b)$;

- Context independent lossy channel
  - $\text{LossySync}(a: \text{Bool}, b: \text{Bool}) =$
    - $\neg a \rightarrow iA. \text{LossySync}(\neg a, b) +$
    - $\neg b \rightarrow iB. \text{LossySync}(a, \neg b) +$
    - $a \rightarrow (b \rightarrow oA|oB. \text{LossySync}(\neg a, \neg b) +$
    - $oA. \text{LossySync}(\neg a, b)$;

- Context dependent lossy channel
  - $\text{LossySync}(a: \text{Bool}, b: \text{Bool}) =$
    - $\neg a \rightarrow iA. \text{LossySync}(\neg a, b) +$
    - $\neg b \rightarrow iB. \text{LossySync}(a, \neg b) +$
    - $a \rightarrow (b \rightarrow oA|oB. \text{LossySync}(\neg a, \neg b) +$
    - $\neg b \rightarrow oA. \text{LossySync}(\neg a, b)$;
Tool support
Data-flow analysis with mCRL2

- µ calculus (property specification format for mCRL2)
  - No deadlock:
    - \([true^*]<true>true\)
  - No livelock:
    - \([true^*]mu X.[\tau]X\)
  - Loan request is always denied or approved:
    - Data agnostic:
      - nu X.[true]X && mu Y.(!((denied || approved))Y && <true>true)
    - Data aware:
      - nu X.[true]X && forall d:Data. mu Y.(!((denied(d) || approved(d)))Y && <true>true)
Loan request scenario: different instances

Salary = 1000

Salary = 2000

Salary = 3000
Comparison of model checking tools for Reo

- Vereofy (University of Dresden)
  - **Advantages:**
    - Developed for Reo and Constraint Automata
    - Counterexamples
  - **Disadvantages:**
    - No support for abstract data types
    - Global domain for all components
    - Primitive data constraint specification language (for filter channels)

- mCRL2 toolset
  - **Advantages:**
    - Powerful support for data
    - Very rich property specification format (μ calculus)
  - **Disadvantages:**
    - Hard to extract counterexamples
    - For infinite domains model checker often does not terminate (problems with algorithms for formulae rewriting)
    - Inability to define some useful data domains
    - Intentional model: state explosion problem

- CADP toolset (INRIA)
  - **Advantages:**
    - Fully compatible with mCRL2
    - Many useful tools (e.g., for performance evaluation)
  - **Disadvantages:**
    - License
Conclusions

- Full featured model checking of Reo connectors
  - Control + data flow analysis
  - Abstract data types
  - Complete tool support
    - Automated generation of mCRL2 code from graphical models
    - First implemented plug-in that deals with
      - Filter and transformer channels
      - Intentional semantics
      - Time channels

- Relation between Reo and ACP

- Eclipse Coordination Tools are an interesting toolset for business process and service composition analysis
  - Good alternative to Petri nets
  - Better fits service-oriented computing paradigm

- N. Kokash, C. Krause, E. de Vink: “Data-aware Design and Verification of Service Compositions with Reo and mCRL2”, ACM SAC 2010, Technical track on Service-Oriented Architectures and Programming (SOAP)
Future Work

- Work on tools usability
  - Analysis across several connectors
  - Obtaining data types from WSDL specifications
  - Other verification tools (e.g., CADP, simulators)
- Timed Reo
  - What properties can be checked with mCRL2
- Application to COMPAS case studies
  - Express compliance requirements and common BP properties in μ-calculus
  - **Problem**: property specification language is very expressive, but hard to use