A Modular Model Checker for Reference Nets: MoMoC

Sven Willrodt, Daniel Moldt and Michael Simon
17.06.2020

University of Hamburg
Faculty of Mathematics, Informatics and Natural Sciences
Department of Informatics
http://www.informatik.uni-hamburg.de/TGI/
Introduction
MoMoC is a novel Model Checking tool for Reference nets, featuring a modular architecture.
MoMoC is a novel Model Checking tool for Reference nets, featuring a modular architecture.

MoMoC pursues two goals:
MoMoC is a novel Model Checking tool for Reference nets, featuring a modular architecture.

MoMoC pursues two goals:

- Teaching model checking
- Extensibility, to form a basis for further research on model checking of Reference nets
Reference Nets
The Java Reference Net Formalism

- Coloured Petri net (CPN) formalism
- Primary formalism of the RENEW simulator
- Java code inscriptions
- Tokens: Java objects or net instances
- Interaction: synchronous channels
• nets-within-nets
• follows the object-oriented paradigm

  net template  class
  net instance  instance/object
  net elements  internals of a class
  uplinks of a net  methods/interface of a class
  invoking an uplink  invoking a method
Java Reference Net Example

\[
\begin{align*}
&[] \rightarrow \text{ni} \rightarrow \text{ni: new netB} \\
&\text{a} \rightarrow \text{ni: ch(x)} \\
&\text{b} \rightarrow \text{ch(x)}
\end{align*}
\]
root net instance

ni: new netB

ni: ch(x)
Java Reference Net Example

root net instance

netB

ni: new netB

ni: ch(x)

"b"

: ch(x)
root net instance

ni: new netB

ni: ch(x)

x

("b")
Features
Features

Modular Model Checker (MoMoC)

- Explicit CTL-Model Checking for Reference Nets
- Parsing
- Result visualization (exploration, colorization, layouting)
- Net Instance Quantifier
- Simpler net formalisms (P/T nets, CPNs) can be treated as flat Reference nets
Atomic Propositions

- FIREABLE(T)
- DEADLOCK
Atomic Propositions

- FIREABLE(T)
- DEADLOCK
- Marking predicates
**Problem:** During runtime, net instances are not uniquely identifiable by a name that must be entered before runtime.
**Problem:** During runtime, net instances are not uniquely identifiable by a name that must be entered before runtime.

**Proposed solution:** Net-Instance-Quantifier
**Problem:** During runtime, net instances are not uniquely identifiable by a name that must be entered before runtime.

**Proposed solution:** Net-Instance-Quantifier

\[ !(\text{Net}, p) \equiv \text{Every net instance of the template Net satisfies } p. \]

\[ ?(\text{Net}, p) \equiv \text{There exists a net instance of the template Net that satisfies } p. \]
Net-Instance-Quantifier

**Problem:** During runtime, net instances are not uniquely identifiable by a name that must be entered before runtime.

**Proposed solution:** Net-Instance-Quantifier

\[ !\langle Net, p \rangle \equiv \text{Every net instance of the template } Net \text{ satisfies } p. \]

\[ ?\langle Net, p \rangle \equiv \text{There exists a net instance of the template } Net \text{ that satisfies } p. \]

Scales independently of the size of the reachability graph, however net instances cannot be tracked over multiple states.
Uses ANTLR as a framework for parsing.

**Parsing features of MoMoC:**

- Parsing of different notations
- Normalization
- Reduction
- Encoding
Goal: Comprehensive results that help teaching (CTL) Model Checking
Result Visualization

Goal: Comprehensive results that help teaching (CTL) Model Checking

- States of the RG can be explored
- RG can interactively be colorized with results of subroutines
Architecture
Goal: An extensible architecture that allows quick prototyping.
Goal: An extensible architecture that allows quick prototyping. Query is handled by an interaction of three types of interchangeable modules.

- **Binding Core** - Finds bindings and calculates successive markings, thus defines the semantics
Goal: An extensible architecture that allows quick prototyping. Query is handled by an interaction of three types of interchangeable modules.

- **Binding Core** - Finds bindings and calculates successive markings, thus defines the semantics

- **Storage Manager** - Stores the reachability graph and finds cycles in the graph
Goal: An extensible architecture that allows quick prototyping. Query is handled by an interaction of three types of interchangeable modules.

- **Binding Core** - Finds bindings and calculates successive markings, thus defines the semantics
- **Storage Manager** - Stores the reachability graph and finds cycles in the graph
- **Procedure** - Contains logic and steps to process a query
Demo
Buffer

:receive("transmit",m) → transmit

:send("fetch",m) → receive

Sender

"Hello"
"Good Day"
"How are you?"

new message

message

paused

transmit

:send("direct",m);

transmit

:send("transmit",m);

Waiting

:receive("fetch",m)

Received

process message

Receiver

:receive("direct",m)
$$EG(! (Buffer, m(stored) = 0) \land AF?(Receiver, m(received) = 1))$$
<table>
<thead>
<tr>
<th>Specification</th>
<th>( EG(\neg Buffer, m[\text{stored}] = 0) \land \neg AF(\neg Receiver, m[\text{received}] = 1) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>Result</td>
<td><strong>TRUE</strong></td>
</tr>
<tr>
<td>Root Net</td>
<td>System[10786]</td>
</tr>
<tr>
<td>Normalized Spec.:</td>
<td>( EG(\neg Buffer, m[\text{stored}] = 0) \land \neg AF(\neg Receiver, m[\text{received}] = 1) )</td>
</tr>
</tbody>
</table>

Show Reachability Graph
<table>
<thead>
<tr>
<th>Specification</th>
<th>EG((Buffer, m[stored]=0) ∧ AF?(Receiver, m[received]=1))</th>
</tr>
</thead>
<tbody>
<tr>
<td>Result</td>
<td>TRUE</td>
</tr>
<tr>
<td>Root Net</td>
<td>System[10786]</td>
</tr>
<tr>
<td>Normalized Spec.:</td>
<td>EG[AND[!(Buffer, m[stored]=0), NOT[EG[NOT[?[(Receiver, m[received]=1)]]]]]]</td>
</tr>
<tr>
<td>Specification</td>
<td>EG(!Buffer, m[stored]=0) ∧ AF(?Receiver, m[received]=1)</td>
</tr>
<tr>
<td>---------------</td>
<td>-----------------------------------------------------</td>
</tr>
<tr>
<td>Result</td>
<td>TRUE</td>
</tr>
<tr>
<td>Root Net</td>
<td>System[10786]</td>
</tr>
<tr>
<td>Normalized Spec.:</td>
<td>EG(AND(!Buffer, m[stored]=0), NOT[EG(NOT(?Receiver, m[received]=1))])</td>
</tr>
</tbody>
</table>

Show Reachability Graph
Specification: $\text{EG}(\text{Buffer, m[stored]=0}) \land \text{AF}(\text{Receiver, m[received]=1})$

Result: TRUE

Root Net: System[10786]

Normalized Spec.: $\text{EG}(\text{AND}((\text{Buffer, m[stored]=0}), \text{NOT}(\text{EG}(\text{NOT}(\text{Receiver, m[received]=1}))))))$
<table>
<thead>
<tr>
<th>Specification</th>
<th>EG((Buffer, m[stored]=0) ∧ AF(Receiver, m[received]=1))</th>
</tr>
</thead>
<tbody>
<tr>
<td>Result</td>
<td>TRUE</td>
</tr>
<tr>
<td>Root Net</td>
<td>System[10786]</td>
</tr>
<tr>
<td>Normalized Spec.:</td>
<td>EG(AND([Buffer, m[stored]=0], NOT(Receiver, m[received]=1)])</td>
</tr>
</tbody>
</table>
Evaluation
Evaluation

• Teaching-size problems (<10k states) are unproblematic with average computing power
• Colorization is helpful
Conclusion

- CTL Model-Checking of Reference nets
- Modular architecture
- Net Instance Quantifier
- Result visualization
Outlook
Outlook

Teaching-oriented goals:

- LTL-Model Checking
- Coverability graph
- Visualization of large RGs
  → Interactive trace visualization
Outlook

Teaching-oriented goals:

- LTL-Model Checking
- Coverability graph
- Visualization of large RGs
  → Interactive trace visualization

Efficiency-oriented goals:

- Code-specific improvements
- Transfer of known techniques to Reference nets
- Techniques that exploit the structural information contained in Reference nets