Flexible Modular Formalization of UML Sequence Diagrams

Luciano Baresi, Mohammad Mehdi Pourhashem K., Matteo Rossi

Politecnico di Milano (DEIB)

1.1 UML Formal Verification

› Model defects are a significant concern as model transformations and code generation may propagate errors to other notations where they are harder to detect and trace.

› UML formal verification comes into help by
  – Transforming UML models into mature and well-know formalism (Petri net, Temporal Logic, LTS, ...)
  – Checking the correctness of user-defined properties with respect to the formal model.
1.2 Sample Sequence Diagram (SD)
2. Formalizing SD with CF

1. Introduction

3. Many meanings of SD

4. Semantic Variation Points

5. Modular Semantics

6. Initial Evaluation
2.1 Formalizing SD with Combined Fragment (CF)?

- Specifying meaning of different CFs.
- Mapping UML elements to their corresponding atomic proposition.
- Writing semantics\(^1\) for every CF, that corresponds to their meaning.

1. Formal semantics are written in TRIO logic (on which formal methods researchers at Politecnico di Milano have been working more than 20 years).
3. Many meanings of SD
3.1 Many Meanings of SD

Why are there many meanings?

› OMG’s semantics is not specified in detail to allow using UML in many domains. When UML is used in a concrete domain, the modeler has to choose from the different possible variants.

› Besides, some authors do not completely comply with OMG’s semantics, which is very permissive and counter-intuitive in some cases.
3.2 Many Meanings of SD

› More than 10 groups of authors proposed their semantics based on their interpretation and preference.

› **Configurability** is missing in their semantics, which led them to fail in realizing OMG’s ambition of keeping UML useful for many domains.

› We identified those variation points that are important for verification purpose analysis.
4. Semantic Variation Points
4.1 Semantic Variation Points

- **Combine**
  How to combine CF with the rest of SD?

- **Loop**
  How to combine Loop with the rest of SD?

- **When**
  When CF’s guard(s) should be evaluated?

- **Choice**
  Which operand should be chosen?
4.2 Semantic Variation Points [combine]

[Combine]: How to combine CF with the rest of SD?

1. Synchronized (SYNC):
   All lifelines must enter and leave CF in a synchronized way.

2. Weak Sequencing (WS):
   Every lifeline is allowed to enter or leave CF as soon as it is done with its preceding events.
4.3 [combine].SYNC vs. [combine].WS

When \( l2.a = 0 \):

Only possible sequence of events for **SYNC** is  

\( !m1, ?m1, !m3, ?m3 \)

 Whereas in **WS**, there are more sequences of events including  

\( !m3, !m1, ?m3, ?m1 \)
4.4 Semantic Variation Points [Loop]

[Loop]: How to combine *Loop* with the rest of SD?

1. Synchronized (*SYNC*):
   All iterations are strictly separated.

2. Weak Sequencing (*WS*):
   Events in different iterations can be interleaved.
4.5 [Loop].SYNC vs. [Loop].WS

SYNC: \{<!m1, !m2, ?m1, ?m2, !m1, !m2, ?m1, ?m2>, ...\}

WS:
\{<!m_{lt1}, !m_{lt2}, !m_{lt1}, !m_{lt2}, ?m_{lt1}, ?m_{lt2}, ?m_{lt1}, ?m_{lt2}>}, ...\}
4.6 Semantic Variation Points [When]

[When]: When CF’s guard(s) should be evaluated?

1. Single global time (SGT):
   When all lifelines are ready to enter CFx they evaluate the guard(s) in a synchronized manner. (consistent iff CFx.[combine] = SYNC)

2. Independent local times (ILT):
   The first lifeline that reaches to CFx evaluates the guard(s), and forces other lifelines to follow its footstep. (consistent iff CFx.[combine] = WS)
4.7 Semantic Variation Points [choice]

[Choice]: Which operand should be chosen?

In Alternative CF if there are several true guarded operands, there are two options to choose one operand among them.

1. Nondeterministic (ND):
   One of true guarded operand is chosen nondeterministically.

2. First from top (FFT):
   First true guarded operand from top of Alternative CF is chosen.
4.8 Six Semantic Variants
5. Modular Semantics

1. Introduction
2. Formalizing SD with CF
3. Many meanings of SD
4. Semantic Variation Points
5. Modular Semantics
6. Initial Evaluation
5.1 Modular Semantics

› Ordering semantics:
  – OrderMonoD(ev1, ev2, guard, exception, isConcur)
  – OrderMonoDRev(ev1, ev2, guard, exception, isConcur)
  – Order(ev1, ev2, guard, exception, isConcur)

› Borders(Module, exception)

› Auxiliary Operators
  – SomFIn_i(ev_1, enclosingModule)
  – SomPIn_i(ev_1, enclosingModule)

› Combined Fragments
  – AltCF(Alt, config)
  – OptCF(Opt, config)
  – ParallelCF(Parallel, config)
  – LoopCF(Loop, config)

› Link_Pre_Post(CF, config)

› Combine(Module, config)
5.2 Mono-Directional Order

If the guard holds at the moment ev1 occurs, ev2 will happen. But if we have ev2 now, it does not necessarily mean that we had ev1 earlier.

It is used when ev2 has several triggers including ev1.

\[
\text{OrderMonoD}(ev_1, ev_2, \text{guard, exception, isConcurrent}) \overset{\text{def}}{=} \\
\begin{cases} 
(1) & \text{if isConcurrent} = \text{true} \\
(1) \land (2) & \text{otherwise}
\end{cases}
\]

\[
ev_1 \land \text{guard} \Rightarrow \left( \text{Until}_{ei}(\neg ev_1 \land \neg ev_2, \text{exception}) \lor \text{Until}_{ei}(\neg ev_1 \land \neg \text{exception}, ev_2) \right). \quad (1)
\]

\[
ev_1 \land \text{guard} \Rightarrow \neg ev_2. \quad (2)
\]
5.3 Mono-Directional Order Reverse

ev2 is preceded by true guarded ev1. But having true guarded ev1 does not necessarily mean that ev2 will occur. It is used when ev2 is one of possible consequences of true guarded ev1.

\[
\text{OrderMonoDRev}(ev_1, ev_2, \text{guard}, \text{exception}, \text{isConcurrent}) \triangleq \\
\left\{ \\
(1) \quad \text{if } \text{isConcurrent} = \text{true} \\
(1) \land (2) \quad \text{otherwise} \\
\right.
\]

\[
ev_2 \Rightarrow \text{Since}_ai((-ev_2 \land \neg\text{exception}), (ev_1 \land \text{guard})) . \quad (1)
\]

\[
ev_1 \land \text{guard} \Rightarrow \neg ev_2. \quad (2)
\]
5.4 Bidirectional Order

If true guarded ev1 occurs, ev2 will happen and if ev2 occurs, it means that we had true guarded ev1.

\[
\text{Order}(ev_1, ev_2, \text{guard}, \text{exception}, \text{isConcur}) \overset{\text{def}}{=} \text{OrderMonoD}(ev_1, ev_2, \text{guard}, \text{exception}, \text{isConcur}) \land \text{OrderMonoDRev}(ev_1, ev_2, \text{guard}, \text{exception}, \text{isConcur})
\]
5.5 Mapping UML Elements to Atomic Propositions

3. Alt_l1_Start
4. Alt_l2_Start
5. Alt_Start
6. Alt_Op_l1_Start
7. Alt_Op_l2_Start
8. Alt_Op_Start
11. Alt_Op_l1_End
12. Alt_Op_l2_End
13. Alt_Op_End
14. Alt_l1_End
15. Alt_l2_End
16. Alt_End
5.6 AltCF (WS, ND) vs. AltCF (WS, FFT)

\[ \text{AltCF(WS, FFT):} \]

\[
\text{Borders}(Alt, S_{Stop}) 
\]

\[
(\bigvee_{i=1}^{n} Alt_{L_{Start}^i} \Rightarrow Alt) \land (Alt_{Start} \Rightarrow \bigvee_{i=1}^{n} Alt_{L_{Start}^i})
\]

\[
\bigwedge_{i=1}^{m} \bigwedge_{j=1}^{n} \text{Order}(Alt_{L_{Start}^j}, Alt_{OP_i^i}_{L_{Start}^j}, Alt_{OP_i^i}, S_{Stop}, true)
\]

\[
\bigwedge_{i=1}^{m} \bigwedge_{j=1}^{n} \text{OrderMonoD}(Alt_{OP_i^i}_{L_{End}^j}, Alt_{L_{End}^j}, true, S_{Stop}, true)
\]

\[
Alt_{OP_{Else}}^i \Leftrightarrow (\neg (\bigvee_{i=1}^{m} Alt_{Guard^i}) \land Alt_{Start})
\]

\[
\bigwedge_{i=1}^{m} Alt_{OP_{Start}^i} \Leftrightarrow Alt_{Start} \land Alt_{Guard^i} \land \neg \bigvee_{j=1}^{i-1} Alt_{Guard^j}
\]

\[
\bigvee_{i \in [1, m] \cup Else} Alt_{OP_{End}^i} \Leftrightarrow Alt_{End}
\]

\[
\bigwedge_{i=1}^{m} (Alt_{OP_{Start}^i} \Rightarrow Alt_{Start} \land Alt_{Guard^i})
\]
5.7 Alternative CF (SYNC, ND)

If the choice of [Combine] is SYNC instead of WS, the positive traces are a subset of those allowed by WS, since there are additional constraints on the start and end of the combined fragment. More precisely, in this case the following formula is added:

\[
(Alt_{Start} \iff \bigwedge_{i=1}^{n} Alt_{L_{Start}^{i}}) \land (Alt_{End} \iff \bigwedge_{i=1}^{n} Alt_{L_{End}^{i}}) \quad (15)
\]
6. Initial Evaluation

1. Introduction
2. Formalizing SD with CF
3. Many meanings of SD
4. Semantic Variation Points
5. Modular Semantics
6. Initial Evaluation
6.1 Our Verification Tool (Corretto)

https://github.com/mmpourhashem/CorrettoUML
6.2 SDSearch

We extended our verification tool, Corretto, to support SD with CF, and performed some experiments regarding History Checking and Completion (HCC), property checking, and time constraints on this SD.

HCC:
T1 = {⟨!?pingS1, !pingS2⟩, ⟨!replyS1⟩, ⟨?pingS2⟩}

Property:

\[ P_1 \overset{\text{def}}{=} \text{Alw} \left( \text{replyS1}_{\text{Start}} \land \text{Futr}(\text{replyS2}_{\text{Start}}, 1) \Rightarrow \neg \text{SomFl}_{1}(\text{searchNextS2}_{\text{Start}}, \text{SDSearch}) \right) \]

Time Constraints:

\[ (@\text{updateResultS1}_{\text{End}} - @\text{updateResultS1}_{\text{Start}}) \geq 1 \]
\[ (@\text{updateResultS2}_{\text{End}} - @\text{updateResultS2}_{\text{Start}}) \geq 1 \]
\[ (@\text{SDSearch}_{\text{End}} - @\text{SDSearch}_{\text{Start}}) \leq 8 \]
### 6.3 Initial Evaluation

<table>
<thead>
<tr>
<th></th>
<th>S-S-ND</th>
<th>S-S-FFT</th>
<th>W-S-ND</th>
<th>W-S-FFT</th>
<th>W-W-ND</th>
<th>W-W-FFT</th>
</tr>
</thead>
<tbody>
<tr>
<td>T1</td>
<td>Result</td>
<td>UNSAT</td>
<td>UNSAT</td>
<td>SAT</td>
<td>SAT</td>
<td>SAT</td>
</tr>
<tr>
<td></td>
<td>Time -s</td>
<td>6</td>
<td>6</td>
<td>31</td>
<td>27</td>
<td>40</td>
</tr>
<tr>
<td>T2</td>
<td>Result</td>
<td>UNSAT</td>
<td>UNSAT</td>
<td>UNSAT</td>
<td>UNSAT</td>
<td>SAT</td>
</tr>
<tr>
<td></td>
<td>Time -s</td>
<td>6</td>
<td>6</td>
<td>15</td>
<td>14</td>
<td>40</td>
</tr>
<tr>
<td>P1</td>
<td>Result</td>
<td>PASSED</td>
<td>PASSED</td>
<td>FAILED</td>
<td>PASSED</td>
<td>FAILED</td>
</tr>
<tr>
<td></td>
<td>Time -s</td>
<td>14</td>
<td>11</td>
<td>41</td>
<td>36</td>
<td>64</td>
</tr>
<tr>
<td>P2</td>
<td>Result</td>
<td>PASSED</td>
<td>PASSED</td>
<td>PASSED</td>
<td>PASSED</td>
<td>PASSED</td>
</tr>
<tr>
<td></td>
<td>Time -s</td>
<td>13</td>
<td>11</td>
<td>57</td>
<td>37</td>
<td>53</td>
</tr>
<tr>
<td>TC</td>
<td>Result</td>
<td>UNSAT</td>
<td>UNSAT</td>
<td>UNSAT</td>
<td>UNSAT</td>
<td>SAT</td>
</tr>
<tr>
<td></td>
<td>Time -s</td>
<td>12</td>
<td>9</td>
<td>30</td>
<td>24</td>
<td>42</td>
</tr>
</tbody>
</table>
7. Conclusion and Future Work

› We presented a modular semantics for UML SD that accommodates existing interpretations.

› First evaluation of the six possible configurations we have identified through Corretto is presented.

› The plan for the future is to cover the whole set of combined fragments.

› Coming soon ...

**NEW** encoding for Bounded Satisfiability Checking of LTL formula (and also MTL and CLTLB with bounded variables), which is both faster and more memory efficient, i.e. it is able to check bigger models (and **100 times faster** for some models).
Questions?

pourhashem.kallehbasti@elet.polimi.it