<table>
<thead>
<tr>
<th>Title</th>
<th>Formal Verification of UML-based Specifications</th>
</tr>
</thead>
<tbody>
<tr>
<td>Author(s)</td>
<td>Soeken, Mathias</td>
</tr>
<tr>
<td>Citation</td>
<td>2010年度科学技術振興機構ERATO湊離散構造処理系プロジェクト講究録. p.205-221.</td>
</tr>
<tr>
<td>Issue Date</td>
<td>2011-06</td>
</tr>
<tr>
<td>Doc URL</td>
<td><a href="http://hdl.handle.net/2115/48438">http://hdl.handle.net/2115/48438</a></td>
</tr>
<tr>
<td>Type</td>
<td>conference presentation</td>
</tr>
<tr>
<td>Note</td>
<td>ERATOセミナNo.34. 2011年1月31日</td>
</tr>
<tr>
<td>File Information</td>
<td>34_all.pdf</td>
</tr>
</tbody>
</table>

Hokkaido University Collection of Scholarly and Academic Papers: HUSCAP
In the recent years, researchers started to enrich the classical textbook specification of embedded systems by more formal descriptions such as the Unified Modeling Language (UML). This enables to lift the starting point for verification techniques to the specification level and leads to an improved design flow, where crucial flaws can be detected, even if no executable implementation is available. However, research on efficient solving techniques for the respective verification tasks is just at the beginning. In the talk, the improved design flow based on specifications enriched with formal descriptions is introduced. We illustrate how early design flaws become evident within this flow and briefly review algorithms aimed at detecting them. Furthermore, first experimental results are presented. Overall, the aim of the talk is to provide an introduction into this emerging area and to present first results.
Formal Verification of UML-based Specifications

Mathias Soeken, Robert Wille, and Rolf Drechsler
Institute of Computer Science, University of Bremen
Bremen, Germany
msoeken@informatik.uni-bremen.de
January 31, 2011

About
Dipl.-Inf. Mathias Soeken (msoeken@informatik.uni-bremen.de)

Curriculum Vitae
• 2004 - 2008: Received Diploma Degree in CS at University of Bremen
• 2008 - 2009: Internship at Mentor Graphics, Hamburg
• since 2009: PhD Student at the Computer Architecture Group of Prof. Rolf Drechsler at University of Bremen

Research Interests
• Formal Verification
  • PhD Thesis about Specification Checking
• Reversible Computation and Quantum Computation
  • Supervisor of a Graduate Students Project
  • Development of RevKit, a Toolkit for Reversible Computation

Computer Architecture at University of Bremen
• Verification
• Debugging
• Test
• Reversible Computation
• Robustness

Specification Verification

Specification (as Textbook)

System model (e.g. in SystemC)

...
Specification Verification

- Specification (as Textbook)
  - Property Checking
    - manually
  - System model (e.g. in SystemC)

- Specification (as Model)
  - Property Checking
    - (semi-)automatically
  - System model (e.g. in SystemC)

Property Checking

- Specification Checking
Host
ack: String
process()

Client
req: String

0..8 clients
1..2 hosts

Command
context Host::process()
pre: clients->size() > 0
post: clients@pre->at(0).req = "exit"
implies ack = "good"
inv: ack.isDefined()
UML Object Diagram
- Objects
- Attributes
- Links

client1: Client
req = "date"
host: Host
ack = "good"
client2: Client
req = "exit"

Mathias Soeken et. al.
Formal Verification of UML-based Specifications
7/29

UML Sequence Diagram
- Objects
- Operation Calls

host: Host
client1: Client
client2: Client
process()

VERIFICATION OF STATIC ASPECTS

What is Inconsistency?

2: \( m \cdot \text{size}(a, m.v = 0) \)
3: \( m \cdot \text{isDefined}() \)
4: \( c \cdot \text{exists}(c|c.u \text{ isDefined}()) \)

\( A \)
\( v: \text{Integer} \)
\( w: \text{Boolean} \)

\( B \)
\( x: \text{Integer} \)
\( y: \text{Integer} \)

\( C \)
\( u: \text{Integer} \)

\( D \)
\( z: \text{Boolean} \)

2 \( \rightarrow \) as

Model is UML-inconsistent

Mathias Soeken et al. Formal Verification of UML-based Specifications 10/29
What is Inconsistency?

\[ i2: \neg \exists a \in \text{A} . a.v = 0 \]
\[ i4: \exists c \in \text{C} . c.u \text{ is defined}() \]

\[ |A| \in \{0, 1\} \]
\[ |B| \in \{1, \ldots, 2\} \]
\[ |C| \in \{1\} \]
\[ |D| \in \{1\} \]

\[ v: \text{Integer} \]
\[ w: \text{Boolean} \]
\[ u: \text{Integer} \]
\[ x: \text{Integer} \]
\[ y: \text{Integer} \]
\[ z: \text{Boolean} \]

\[ i1: v <= 10 \implies w \]
\[ i2: \neg \exists a \in \text{A} . a.v = 0 \]
\[ i3: x \text{ is defined}() \]
\[ i4: \exists c \in \text{C} . c.u \text{ is defined}() \]

\[ A \supseteq B \]
\[ B \subseteq C \]
\[ C \subseteq D \]

Running Example

Register

| name: String |
| purpose: RegisterType |
| bitwidth: Integer |

RegisterType

| general pc ir overflow |

Processor

| bitwidth: Integer |

context Processor inv bs:
\[ \text{register->exists(r\in r.\text{bitwidth}=\text{bitwidth})} \]
Running Example

Is there a valid system state consisting of one processor and three registers?

<table>
<thead>
<tr>
<th>r0 : Register</th>
<th>name</th>
<th>purpose</th>
<th>bitwidth</th>
</tr>
</thead>
<tbody>
<tr>
<td>r1 : Register</td>
<td>name</td>
<td>purpose</td>
<td>bitwidth</td>
</tr>
<tr>
<td>r2 : Register</td>
<td>name</td>
<td>purpose</td>
<td>bitwidth</td>
</tr>
<tr>
<td>p0 : Processor</td>
<td></td>
<td></td>
<td>bitwidth</td>
</tr>
</tbody>
</table>

Encoding Attributes

How many bits are needed for the bit-vectors?

\[ \log_2(n + 1) \] bits are needed. The additional value is for representing undefined attributes. Determination of \( n \) by type of attribute:

- **Boolean** \( n = 2 \)
- **Enumeration** \( n = \text{Number of fields} \)
- **String** \( n = \text{Number of all possible strings} \)
- **Integer** \( n = 2^{\ell - 1} \) for \( \ell \)-bit integers
**Encoding Attributes**

How many bits are needed for the bit-vectors?

\[ \log_2(n + 1) \] bits are needed. The additional value is for representing undefined attributes. Determination of \( n \) by type of attribute:

- **Boolean** \( n = 2 \)
- **Enumeration** \( n = \) Number of fields
- **String** \( n = \) Number of all possible strings
- **Integer** \( n = 2^l - 1 \) for \( l \)-bit integers

**Running Example**

\( r_0 : \text{Register} \)

- \( \alpha r_0 \text{name} \in B^2 \)
- \( \alpha r_0 \text{purpose} \in B^3 \)
- \( \alpha r_0 \text{bitwidth} \in B^8 \)

\( r_1 : \text{Register} \)

- \( \alpha r_1 \text{name} \in B^2 \)
- \( \alpha r_1 \text{purpose} \in B^3 \)
- \( \alpha r_1 \text{bitwidth} \in B^8 \)

\( r_2 : \text{Register} \)

- \( \alpha r_2 \text{name} \in B^2 \)
- \( \alpha r_2 \text{purpose} \in B^3 \)
- \( \alpha r_2 \text{bitwidth} \in B^8 \)

\( p_0 : \text{Processor} \)

- \( \alpha p_0 \text{bitwidth} \in B^8 \)

**Encoding Links**

- \( \lambda \alpha p_0 \text{register} \) is a bit-mask and each bit enables or disables a possible link to a register
- \( \lambda \alpha p_0 \text{register} = \lambda_0 \alpha 1 \alpha 2 \)

- There is a link between \( p_0 \) and \( r_i \) iff \( \lambda_i = 1 \)
Encoding Links

- $\lambda_{p0}$ register is a bit-mask and each bit enables or disables a possible link to a register
- $\lambda_{p0}$ register $= \lambda_0 \lambda_1 \lambda_2$
- There is a link between $p0$ and $r_i$ iff $\lambda_i = 1$

Encoding Invariants

```
context Processor inv bw:
register->forall(r|r.bitwidth=bitwidth)
```

- Can easily be converted to a CNF using Boolean transformations
- Other OCL functions can be translated in a similar way
Verification Tasks
• Let \( I = \{i_1, \ldots , i_n\} \) be a set of invariants
• Let \( \sigma \) denote a system state, i.e. a concrete assignment of attributes and links
• Let \( \sigma(i) \in B \) with \( i \in I \) be the evaluation of \( i \) in \( \sigma \)

Consistency
\[
\exists \sigma : \bigwedge_{i \in I} \sigma(i)
\]

Independence If the model is consistent, an invariant \( j \in I \) is independent, if
\[
\exists \sigma : \bigwedge_{i \in I \setminus \{j\}} \sigma(i) \wedge \neg \sigma(j)
\]
Experimental Results

- Consistency check for consistent models
- Enumerative approach is USE
- UML2Alloy converts UML models to an Alloy model which is solved by a SAT solver

\[ t \]

<table>
<thead>
<tr>
<th>demo</th>
<th>arbiter</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.01</td>
<td>0.1</td>
</tr>
<tr>
<td>1</td>
<td>10</td>
</tr>
<tr>
<td>100</td>
<td>1000</td>
</tr>
</tbody>
</table>

- Enumerative
- UML2Alloy
- SAT based


Running Example

TrafficLight
pedLight: Boolean
carLight: Boolean
request: Boolean

Button
counter: Integer
requesting()

context requesting()
pre: tl.pedLight = false
post: tl.request = true
post: counter = counter@pre + 1

TrafficLight
pedLight: Boolean
carLight: Boolean
request: Boolean

context switchPedLight()
pre: request = true
post: pedLight != pedLight@pre
post: request = false

context switchCarLight()
post: carLight != carLight@pre

inv: not(pedLight = true and carLight = true)
Running Example

Time: 3
context Button::requesting()
pre: tl.pedLight = false
post: tl.request = true
post: counter ... 1
b2: Button
counter = 0
Car Pedestrian

Mathias Soeken et.al. Formal Verification of UML-based Specifications 23/ 29

Encoding
• Introduce time model, i.e. we consider \( k \) steps (operation calls)
• This leads to \( k + 1 \) system states \( \sigma_0, \ldots, \sigma_k \)

Mathias Soeken et.al. Formal Verification of UML-based Specifications 24/ 29

Encoding
• Introduce time model, i.e. we consider \( k \) steps (operation calls)
• This leads to \( k + 1 \) system states \( \sigma_0, \ldots, \sigma_k \)
• Create \( \alpha \) and \( \lambda \) variables for each time step
• Let \( \text{OP} \) be the set of all possible operation calls
  • Let \( \text{OP}_i \) with \( i \in \{0, \ldots, k - 1\} \) the operation call executed in system state \( i \)
Encoding

- Introduce time model, i.e. we consider \( k \) steps (operation calls)
- This leads to \( k + 1 \) system states \( \sigma_0, \ldots, \sigma_k \)
- Create \( \bar{a} \) and \( \lambda \) variables for each time step
- Let \( OP \) be the set of all possible operation calls
- Let \( op_i \) with \( i \in \{0, \ldots, k \} \) the operation call executed in system state \( i \)
- Besides the invariants, let \( \prec_{op} \) and \( \succ_{op} \) with \( op \in OP \) the pre- and post-conditions of the operation associated to the operation call \( op \)

Then, each verification task \( \tau \) can be described as:

\[
f = \bigwedge_{t=0}^{k} \sigma_t \land \bigwedge_{t=0}^{k-1} (\sigma_t (\prec_{op}) \land \sigma_{t+1} (\succ_{op}) \land \tau)
\]
WHAT TO DO NEXT?

Debugging
Verification Tasks
Diagram Types
Encoding

References

Formal Verification of UML-based Specifications

Mathias Soeken, Robert Wille, and Rolf Drechsler
Institute of Computer Science, University of Bremen
Bremen, Germany
msoeken@informatik.uni-bremen.de
January 31, 2011