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
FORMAL VERIFICATION OF UML-BASED SPECIFICATIONS

Mathias Soeken, University of Bremen, Talk at 北海道大学
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

Mathias Soeken et.al. Formal Verification of UML-based Specifications
Specification Verification
Specification Verification

Specification (as Textbook) → manually → System model (e.g. in SystemC) → ...
Specification Verification

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

Property Checking

...
Specification Verification

Property Checking

Specified (as Model)

manually

System model (e.g. in SystemC)
Specification Verification

Property Checking

1. Specification (as Model)
   (semi-)automatically
2. System model (e.g. in SystemC)
   ...

Mathias Soeken et.al. Formal Verification of UML-based Specifications
Specification Verification

Property Checking

Specification Checking

Specification (as Model)

(semi-)automatically

System model (e.g. in SystemC)

...
UML Class Diagram

- Classes
- Attributes
- Operations
- Associations
- Constraints

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 Class Diagram

- Classes

Client

- req: String
- process()

Host

- ack: String
- clients: 0..8
- hosts: 1..2
- Command

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

- Classes
- Attributes

**Client**
req: String

**Host**
ack: String

---

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

- Classes
- Attributes
- Operations

Client
req: String

Host
ack: String
process()
UML Class Diagram

- Classes
- Attributes
- Operations
- Associations

Mathias Soeken et al. Formal Verification of UML-based Specifications
UML Class Diagram

- Classes
- Attributes
- Operations
- Associations
- Constraints

```
context Host::process()
pre: clients->size() > 0
post: clients@pre->at(0).req = "exit"
implies ack = "good"
```
UML Object Diagram

- Objects
  - client1: Client
  - host: Host
  - client2: Client

- Attributes
  - req = "date"
  - ack = "good"

- Links
  - Command
    - Command
UML Object Diagram

- Objects

client1: Client

host: Host

client2: Client
UML Object Diagram

- Objects
- Attributes

client1: Client
req = "date"

host: Host
ack = "good"

client2: Client
req = "exit"
### UML Object Diagram

- **Objects**
- **Attributes**
- **Links**

<table>
<thead>
<tr>
<th>client1: Client</th>
<th>Command</th>
<th>host: Host</th>
<th>Command</th>
<th>client2: Client</th>
</tr>
</thead>
<tbody>
<tr>
<td>req = ‘‘date’’</td>
<td>Command</td>
<td>ack = ‘‘good’’</td>
<td>Command</td>
<td>req = ‘‘exit’’</td>
</tr>
</tbody>
</table>
UML Sequence Diagram

- Objects
  - host: Host
  - client1: Client
  - client2: Client

- Operation Calls
  - process()
UML Sequence Diagram

- Objects

host: Host
client1: Client
client2: Client
UML Sequence Diagram

- Objects
- Operation Calls

host: Host

client1: Client

client2: Client

process()

process()
What is Inconsistency?

\[ \text{i5: } A.allInstances().forall(a|a.v=8) \]

\[ \text{i2: } \text{as->one(a|a.w = 0)} \]
\[ \text{i3: } x.isDefined() \]
\[ \text{i4: } \text{cs->exists(c|c.u.isDefined())} \]

The model is UML-inconsistent.

The model is OCL-inconsistent.
What is Inconsistency?

i1: \( v \leq 10 \) implies \( w \)

i2: \( as \rightarrow \text{one}(a | a.w = 0) \)

i3: \( x \text{.isDefined}() \)

i4: \( cs \rightarrow \exists(c | c.u \text{.isDefined}()) \)

i5: \( A \text{.allInstances}().\forall(a | a.v = 8) \)

i6: \( cs \rightarrow \forall(c | c.u < 10) \)

\( x_C \) objects of \( C \)
What is Inconsistency?

$x_C$ objects of $C$

$2x_C$ objects of $A$

\begin{itemize}
  \item i2: $as \rightarrow one(a | a.w = 0)$
  \item i3: $x.isDefined()$
  \item i4: $cs \rightarrow exists(c | c.u.isDefined())$
  \item i5: $A.allInstances().forAll(a | a.v = 8)$
  \item i6: $cs \rightarrow forall(c | c.u < 10)$
\end{itemize}
What is Inconsistency?

\[ i1: v \leq 10 \implies w \]

\[ i2: \text{as} \to \text{one}(a|a.w = 0) \]

\[ i3: x.isDefined() \]

\[ i4: \text{cs} \to \exists(c|c.u.isDefined()) \]

\[ i5: \text{A.allInstances()}.\text{forAll}(a|a.v=8) \]

\[ i6: \text{cs} \to \forall(c|c.u < 10) \]

\( x_C \) objects of \( C \)

\( 2x_C \) objects of \( A \)

\( 5x_C \) objects of \( B \)
What is Inconsistency?

\[ i1: \text{v} \leq 10 \implies \text{w} \]

\[ i2: \text{as} \rightarrow \text{one}(a | a\text{.w} = 0) \]

\[ i3: x\text{.isDefined()} \]

\[ i4: \text{cs} \rightarrow \exists(c | c\text{.u}.\text{isDefined()}) \]

\[ i5: A\text{.allInstances()}.\text{forall}(a | a\text{.v} = 8) \]

\[ x_C \text{ objects of C} \]

\[ 2x_C \text{ objects of A} \]

\[ 5x_C \text{ objects of B} \]

\[ 3x_C \text{ objects of C} \]
What is Inconsistency?

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

\[ 2 \] as
\[ 5 \] bs

\[ i_2 : \text{as} \rightarrow \text{one}(a | a.w = 0) \]
\[ i_3 : x.\text{isDefined()} \]
\[ i_4 : \text{cs} \rightarrow \text{exists}(c | c.u.\text{isDefined()}) \]
\[ i_5 : A.\text{allInstances()}.\forall a | a.v = 8 \]

\( x_C \) objects of \( C \)
\( 2x_C \) objects of \( A \)
\( 5x_C \) objects of \( B \)
\( 3x_C \) objects of \( C \)

Model is UML-inconsistent
What is Inconsistency?

i1: v ≤ 10 implies w

i2: as->one(a|a.w = 0)

i3: x.isDefined()

i4: cs->exists(c|c.u.isDefined())

i5: A.allInstances().forall(a|a.v=8)

i6: cs->forall(c|c.u < 10)
What is Inconsistency?

i1: \( v \leq 10 \) implies \( w \)

i2: \( \text{as} \to \text{one}(a|a.w = 0) \)

i3: \( x.\text{isDefined()} \)

i4: \( \text{cs} \to \exists(c|c.u.\text{isDefined()}) \)

i5: \( A.\text{allInstances()}.\forall(a|a.v=8) \)

i6: \( \text{cs} \to \forall(c|c.u < 10) \)
What is Inconsistency?

i1: \( v \leq 10 \) implies \( w \)

i2: \( \text{as} \rightarrow \text{one}(a|a.w = 0) \)

i3: \( x \text{.isDefined()} \)

i4: \( \text{cs} \rightarrow \text{exists}(c|c.u \text{.isDefined()} \)

i5: \( A.\text{allInstances()} \rightarrow \forall(a|a.v = 8) \)

i6: \( \text{cs} \rightarrow \forall(c|c.u < 10) \)

---

**Model is UML-inconsistent**

**Model is OCL-inconsistent**

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

- i2: \( \text{as}\rightarrow\text{one}(a|a.w = 0) \)
- i3: \( x.\text{isDefined()} \)
- i4: \( \text{cs}\rightarrow\exists(c|c.u.\text{isDefined()}) \)
- i5: \( A.\text{allInstances().forall}(a|a.v = 8) \)
- i6: \( \text{cs}\rightarrow\forall(c|c.u < 10) \)

Model is OCL-inconsistent
Solving Flow

Class Diagram
- Number of objects
- Verification Task

Object Diagram

Satisfiability Problem

verification task disproven
Running Example

Register

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

1..* register

Processor

| bitwidth: Integer |

1 processor

<enum>

RegisterType

general
pc
ir
overflow

context Processor inv bw:
register->forAll(r|r.bitwidth=bitwidth)
Running Example

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

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

p0 : Processor

bitwidth
Running Example

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

- **r0 : Register**
  - name\textalpha r^0
  - purpose\textalpha r^0
  - \textalpha bitwidth

- **r1 : Register**
  - name\textalpha r^1
  - purpose\textalpha r^1
  - \textalpha bitwidth

- **r2 : Register**
  - name\textalpha r^2
  - purpose\textalpha r^2
  - \textalpha bitwidth

- **p0 : Processor**
  - \textalpha p^0
  - \textalpha bitwidth
Encoding Attributes

How many bits are needed for the bit-vectors?

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

- **Boolean** \( n = 2 \)
Encoding Attributes

How many bits are needed for the bit-vectors?

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

- **Boolean** \( n = 2 \)
- **Enumeration** \( n = \text{Number of fields} \)
Encoding Attributes

How many bits are needed for the bit-vectors?

\[ \lceil \log_2(n + 1) \rceil \] 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
Encoding Attributes

How many bits are needed for the bit-vectors?

\[ \lceil \log(n + 1) \rceil \] 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** \( n = 2^l - 1 \) for \( l \)-bit integers
Encoding Attributes

How many bits are needed for the bit-vectors?

$\lceil \log_2(n+1) \rceil$ 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** \( n = 2^l - 1 \) for \( l \)-bit integers
Running Example

\[ r_0 : \text{Register} \]
\[ \alpha_{\text{r}_0}^{\text{name}} \in B^2 \]
\[ \alpha_{\text{r}_0}^{\text{purpose}} \in B^3 \]
\[ \alpha_{\text{r}_0}^{\text{bitwidth}} \in B^8 \]

\[ r_1 : \text{Register} \]
\[ \alpha_{\text{r}_1}^{\text{name}} \in B^2 \]
\[ \alpha_{\text{r}_1}^{\text{purpose}} \in B^3 \]
\[ \alpha_{\text{r}_1}^{\text{bitwidth}} \in B^8 \]

\[ r_2 : \text{Register} \]
\[ \alpha_{\text{r}_2}^{\text{name}} \in B^2 \]
\[ \alpha_{\text{r}_2}^{\text{purpose}} \in B^3 \]
\[ \alpha_{\text{r}_2}^{\text{bitwidth}} \in B^8 \]

\[ p_0 : \text{Processor} \]
\[ \alpha_{\text{p}_0}^{\text{bitwidth}} \in B^8 \]
Running Example

- **r0 : Register**
  - \(\alpha^r_0\) name = 0
  - \(\alpha^r_0\) purpose = 1
  - \(\alpha^r_0\) bitwidth = 32

- **r1 : Register**
  - \(\alpha^r_1\) name = 1
  - \(\alpha^r_1\) purpose = 0
  - \(\alpha^r_1\) bitwidth = 32

- **r2 : Register**
  - \(\alpha^r_2\) name = 3
  - \(\alpha^r_2\) purpose = 4
  - \(\alpha^r_2\) bitwidth = 32

- **p0 : Processor**
  - \(\alpha^p_0\) bitwidth = 32
Running Example

r0 : Register
- $\alpha_{\text{name}}^{r0} \in B^2$
- $\alpha_{\text{purpose}}^{r0} \in B^3$
- $\alpha_{\text{bitwidth}}^{r0} \in B^8$
- $\lambda^{r0}_{\text{processor}}$

r1 : Register
- $\alpha_{\text{name}}^{r1} \in B^2$
- $\alpha_{\text{purpose}}^{r1} \in B^3$
- $\alpha_{\text{bitwidth}}^{r1} \in B^8$
- $\lambda^{r1}_{\text{processor}}$

r2 : Register
- $\alpha_{\text{name}}^{r2} \in B^2$
- $\alpha_{\text{purpose}}^{r2} \in B^3$
- $\alpha_{\text{bitwidth}}^{r2} \in B^8$
- $\lambda^{r2}_{\text{processor}}$

p0 : Processor
- $\alpha_{\text{bitwidth}}^{p0} \in B^8$
- $\lambda^{p0}_{\text{register}}$
Encoding Links

- \( \lambda_{\text{register}}^{p_0} \) is a bit-mask and each bit enables or disables a possible link to a register.
- \( \lambda_{\text{register}}^{p_0} = \lambda_0 \lambda_1 \lambda_2 \)
Encoding Links

- $\lambda^p_0$ is a bit-mask and each bit enables or disables a possible link to a register
- $\lambda^p_0$ = $\lambda_0 \lambda_1 \lambda_2$
- There is a link between p0 and ri iff $\lambda_i = 1$
Encoding Links

- \( \lambda_{\text{register}}^{p_0} \) is a bit-mask and each bit enables or disables a possible link to a register.
- \( \lambda_{\text{register}}^{p_0} = \lambda_0 \lambda_1 \lambda_2 \)
- There is a link between \( p_0 \) and \( r_i \) iff \( \lambda_i = 1 \)
### Running Example

<table>
<thead>
<tr>
<th>r0 : Register</th>
<th>r1 : Register</th>
<th>r2 : Register</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\alpha_{\text{name}} \in \mathbb{B}^2$</td>
<td>$\alpha_{\text{name}} \in \mathbb{B}^2$</td>
<td>$\alpha_{\text{name}} \in \mathbb{B}^2$</td>
</tr>
<tr>
<td>$\alpha_{\text{purpose}} \in \mathbb{B}^3$</td>
<td>$\alpha_{\text{purpose}} \in \mathbb{B}^3$</td>
<td>$\alpha_{\text{purpose}} \in \mathbb{B}^3$</td>
</tr>
<tr>
<td>$\alpha_{\text{bitwidth}} \in \mathbb{B}^8$</td>
<td>$\alpha_{\text{bitwidth}} \in \mathbb{B}^8$</td>
<td>$\alpha_{\text{bitwidth}} \in \mathbb{B}^8$</td>
</tr>
<tr>
<td>$\lambda r^0_{\text{processor}} \in \mathbb{B}$</td>
<td>$\lambda r^1_{\text{processor}} \in \mathbb{B}$</td>
<td>$\lambda r^2_{\text{processor}} \in \mathbb{B}$</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>p0 : Processor</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\alpha_{\text{bitwidth}} \in \mathbb{B}^8$</td>
</tr>
<tr>
<td>$\lambda p^0_{\text{register}} \in \mathbb{B}^3$</td>
</tr>
</tbody>
</table>
Running Example

<table>
<thead>
<tr>
<th>r0 : Register</th>
<th>r1 : Register</th>
<th>r2 : Register</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\alpha_{\text{name}}^{r0} \in B^2$</td>
<td>$\alpha_{\text{name}}^{r1} \in B^2$</td>
<td>$\alpha_{\text{name}}^{r2} \in B^2$</td>
</tr>
<tr>
<td>$\alpha_{\text{purpose}}^{r0} \in B^3$</td>
<td>$\alpha_{\text{purpose}}^{r1} \in B^3$</td>
<td>$\alpha_{\text{purpose}}^{r2} \in B^3$</td>
</tr>
<tr>
<td>$\alpha_{\text{bitwidth}}^{r0} \in B^8$</td>
<td>$\alpha_{\text{bitwidth}}^{r1} \in B^8$</td>
<td>$\alpha_{\text{bitwidth}}^{r2} \in B^8$</td>
</tr>
<tr>
<td>$\lambda^{r0}_{\text{processor}} = 1$</td>
<td>$\lambda^{r1}_{\text{processor}} = 0$</td>
<td>$\lambda^{r2}_{\text{processor}} = 0$</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>p0 : Processor</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\alpha_{\text{bitwidth}}^{p0} \in B^8$</td>
</tr>
<tr>
<td>$\lambda^{p0}_{\text{register}} = 100$</td>
</tr>
</tbody>
</table>
Encoding Invariants

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

\[\bigwedge_{i=0}^{\text{oid(Register)}-1} \left[ \lambda^0_{\text{register}}[i] \Rightarrow (\alpha^{ri}_{\text{bitwidth}} = \alpha^0_{\text{bitwidth}}) \right] \]
Encoding Invariants

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

\[\bigwedge_{i=0}^{\text{oid(\text{Register})}-1} \left[ \lambda r^0_{\text{register}}[i] \Rightarrow \left( \alpha^r_i \text{bitwidth} = \alpha^p_0 \text{bitwidth} \right) \right]\]

- Can easily be converted to a CNF using Boolean transformations
- Other OCL functions can be translated in a similar way
Encoding Invariants

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

\[
\bigwedge_{i=0}^{\text{oid(\text{Register})}-1} \left[ \lambda^p_{\text{register}[i]} \Rightarrow \left( \lambda^r_{\text{bitwidth}} = \lambda^p_{\text{bitwidth}} \right) \right]
\]

- Can easily be converted to a CNF using Boolean transformations
- Other OCL functions can be translated in a similar way
Verification Tasks

• Let $\mathcal{I} = \{i_1, \ldots, i_n\}$ be a set of invariants

• Let $\sigma$ denote a system state, i.e. a concrete assignment of attributes and links
Verification Tasks

• Let $\mathcal{I} = \{i_1, \ldots, i_n\}$ be a set of invariants
• Let $\sigma$ denote a system state, i.e. a concrete assignment of attributes and links
• Let $\sigma(i) \in \mathbb{B}$ with $i \in \mathcal{I}$ be the evaluation of $i$ in $\sigma$
Verification Tasks

- Let $\mathcal{I} = \{i_1, \ldots, i_n\}$ be a set of invariants
- Let $\sigma$ denote a system state, i.e. a concrete assignment of attributes and links
- Let $\sigma(i) \in \mathbb{B}$ with $i \in \mathcal{I}$ be the evaluation of $i$ in $\sigma$

Consistency

$$\exists \sigma : \bigwedge_{i \in \mathcal{I}} \sigma(i)$$
Verification Tasks

- Let $\mathcal{I} = \{i_1, \ldots, i_n\}$ be a set of invariants
- Let $\sigma$ denote a system state, i.e. a concrete assignment of attributes and links
- Let $\sigma(i) \in \mathbb{B}$ with $i \in \mathcal{I}$ be the evaluation of $i$ in $\sigma$

Consistency

$$\exists \sigma : \bigwedge_{i \in \mathcal{I}} \sigma(i)$$

Independence If the model is consistent, an invariant $j \in \mathcal{I}$ is independent, if

$$\exists \sigma : \left( \bigwedge_{i \in \mathcal{I} \setminus \{j\}} \sigma(i) \right) \land \neg \sigma(j)$$
Verification Tasks

• Let $\mathcal{I} = \{i_1, \ldots, i_n\}$ be a set of invariants
• Let $\sigma$ denote a system state, i.e. a concrete assignment of attributes and links
• Let $\sigma(i) \in \mathbb{B}$ with $i \in \mathcal{I}$ be the evaluation of $i$ in $\sigma$

Consistency

$$\exists \sigma : \bigwedge_{i \in \mathcal{I}} \sigma(i)$$

Independence If the model is consistent, an invariant $j \in \mathcal{I}$ is independent, if

$$\exists \sigma : \left( \bigwedge_{i \in \mathcal{I} \setminus \{j\}} \sigma(i) \right) \land \neg \sigma(j)$$
Verification Tasks

- Let $\mathcal{I} = \{i_1, \ldots, i_n\}$ be a set of invariants
- Let $\sigma$ denote a system state, i.e. a concrete assignment of attributes and links
- Let $\sigma(i) \in \mathbb{B}$ with $i \in \mathcal{I}$ be the evaluation of $i$ in $\sigma$

Consistency

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

Independence If the model is consistent, an invariant $j \in \mathcal{I}$ is independent, if

\[ \exists \sigma : \left( \bigwedge_{i \in \mathcal{I} \setminus \{j\}} \sigma(i) \right) \land \neg \sigma(j) \]
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
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

![Graph showing experimental results for demo and arbiter models with time on the Y-axis and different model approaches on the X-axis.](image)
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
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

![Graph showing experimental results]

- **t**
- **demo**
- **arbiter**
- **simple-cpu**
- **train**

- Enumerative
- UML2Alloy
- SAT based
Running Example

**TrafficLight**

- **pedLight**: Boolean
- **carLight**: Boolean
- **request**: Boolean

**Button**

- **counter**: Integer
- **requesting()**
  - button: 2
  - light: 1

**context requesting()**

- **pre**: tl.pedLight = false
- **post**: tl.request = true
- **post**: counter = counter@pre + 1

**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: 0

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

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

context TrafficLight::switchCarLight()
predLight = false
carLight = true
request = false
post: carLight != carLight@pre

context TrafficLight
inv: not(pedLight = true and carLight = true)

Mathias Soeken et.al.  Formal Verification of UML-based Specifications
Running Example

Time: 0

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

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

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

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

Time: 1

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

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

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

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

Time: 1

counter = 1

counter = 0

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

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

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

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

Time: 2

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

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

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

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

**Time: 2**

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

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

**context TrafficLight::switchCarLight()**
- **post:** carLight != carLight@pre

**context TrafficLight**
- **inv:** not(pedLight = true and carLight = true)
Running Example

Time: 3

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

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

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

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

Time: 3

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

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

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

context TrafficLight
inv: not(pedLight = true and carLight = true)
**Encoding**

- Introduce **time** model, i.e. we consider \( k \) steps (operation calls)
- This leads to \( k + 1 \) system states \( \sigma_0, \ldots, \sigma_k \)
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 \( \tilde{\alpha} \) and \( \tilde{\lambda} \) variables for each time step
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 $\tilde{a}$ and $\tilde{\lambda}$ variables for each time step
• Let $OP$ be the set of all possible operation calls
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 $\tilde{\alpha}$ and $\tilde{\lambda}$ variables for each time step
- Let $OP$ be the set of all possible operation calls
  - Let $op_i$ with $i \in \{0, \ldots, k - 1\}$ 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 \( \tilde{\alpha} \) and \( \tilde{\lambda} \) variables for each time step
- Let \( \mathcal{OP} \) be the set of all possible operation calls
- Let \( \textit{op}_i \) with \( i \in \{0, \ldots, k - 1\} \) the operation call executed in system state \( i \)
- Besides the invariants, let \( \textit{\pre}_{op} \) and \( \textit{\post}_{op} \) with \( op \in \mathcal{OP} \) the pre- and post-conditions of the operation associated to the operation call \( op \)
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 \( \vec{\alpha} \) and \( \vec{\lambda} \) variables for each time step
- Let \( OP \) be the set of all possible operation calls
- Let \( op_i \) with \( i \in \{0, \ldots, k - 1\} \) the operation call executed in system state \( i \)
- Besides the invariants, let \( \preff_{op} \) and \( \postff_{op} \) 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:

\[
f = \bigwedge_{t=0}^{k} \sigma_t(I) \land \bigwedge_{t=0}^{k-1} (\sigma_t(\preff_{op_t}) \land \sigma_{t+1}(\postff_{op_t})) \land \tau
\]
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 \( \tilde{\alpha} \) and \( \tilde{\lambda} \) variables for each time step
- Let \( OP \) be the set of all possible operation calls
- Let \( op_i \) with \( i \in \{0, \ldots, k-1\} \) the operation call executed in system state \( i \)
- Besides the invariants, let \( \preceq_{op} \) and \( \succeq_{op} \) 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:

\[
f = \bigwedge_{t=0}^{k} \sigma_t(I) \land \bigwedge_{t=0}^{k-1} (\sigma_t(\preceq_{op_t}) \land \sigma_{t+1}(\succeq_{op_t})) \land \tau
\]
## Encoding

- $\sigma_0$
  - $b1$: Button
    - counter = 0
  - $tl$: TrafficLight
    - pedLight = false
    - carLight = true
    - request = false
  - $b2$: Button
    - counter = 0

- $\sigma_1$
  - $b1$: Button
    - counter = ?
  - $tl$: TrafficLight
    - pedLight = ?
    - carLight = ?
    - request = ?
  - $b2$: Button
    - counter = ?

- $\sigma_k$
  - $b1$: Button
    - counter = ?
  - $tl$: TrafficLight
    - pedLight = ?
    - carLight = ?
    - request = ?
  - $b2$: Button
    - counter = ?

The operations $op_0 = ?$ and $op_1 = ?$ to $op_{k-1} = ?$ are indicated between the states.
Solving Flow

- Class Diagram
- Object Diagram
- Verification Task
- Depth $k$

Sequence Diagram

Satisfiability Problem

+1

sat

unsat
## Experimental Results

<table>
<thead>
<tr>
<th>Name</th>
<th>Task</th>
<th>#Obj</th>
<th>Depth</th>
<th>Status</th>
<th>Run-time</th>
</tr>
</thead>
<tbody>
<tr>
<td>switch</td>
<td>Reachability</td>
<td>25</td>
<td>23</td>
<td>sat</td>
<td>49.70</td>
</tr>
<tr>
<td>switch</td>
<td>Reachability</td>
<td>25</td>
<td>22</td>
<td>unsat</td>
<td>50.00</td>
</tr>
<tr>
<td>switch</td>
<td>Reachability</td>
<td>25</td>
<td>50</td>
<td>sat</td>
<td>621.70</td>
</tr>
<tr>
<td>switch</td>
<td>Reachability</td>
<td>9</td>
<td>103</td>
<td>sat</td>
<td>147.50</td>
</tr>
<tr>
<td>switch</td>
<td>Reachability</td>
<td>9</td>
<td>102</td>
<td>unsat</td>
<td>90.40</td>
</tr>
<tr>
<td>simple-cpu2</td>
<td>State Gen.</td>
<td>13</td>
<td>100</td>
<td>sat</td>
<td>1.30</td>
</tr>
<tr>
<td>simple-cpu2</td>
<td>State Gen.</td>
<td>13</td>
<td>100</td>
<td>unsat</td>
<td>0.40</td>
</tr>
<tr>
<td>traffic-control</td>
<td>Reachability</td>
<td>6</td>
<td>5</td>
<td>sat</td>
<td>0.00</td>
</tr>
<tr>
<td>traffic-control</td>
<td>Reachability</td>
<td>24</td>
<td>10</td>
<td>sat</td>
<td>1.60</td>
</tr>
<tr>
<td>traffic-control</td>
<td>State Gen.</td>
<td>9</td>
<td>30</td>
<td>unsat</td>
<td>0.10</td>
</tr>
<tr>
<td>traffic-control</td>
<td>State Gen.</td>
<td>9</td>
<td>100</td>
<td>unsat</td>
<td>0.40</td>
</tr>
</tbody>
</table>
WHAT TO DO NEXT?

Debugging

Verification Tasks

Diagram Types

Encoding
WHAT TO DO NEXT?
WHAT TO DO NEXT?

Debugging

Verification Tasks

Diagram Types
WHAT TO DO NEXT?

Debugging

Diagram Types

Verification Tasks

Encoding
WHAT TO DO NEXT?

Debugging

Encoding

Diagram Types

Verification Tasks
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