ERATO セミナ 2010 - No. 34

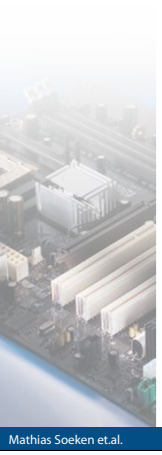
Formal Verification of UML-based Specifications

Dr. Mathias Soeken
University of Bremen

2011/1/31

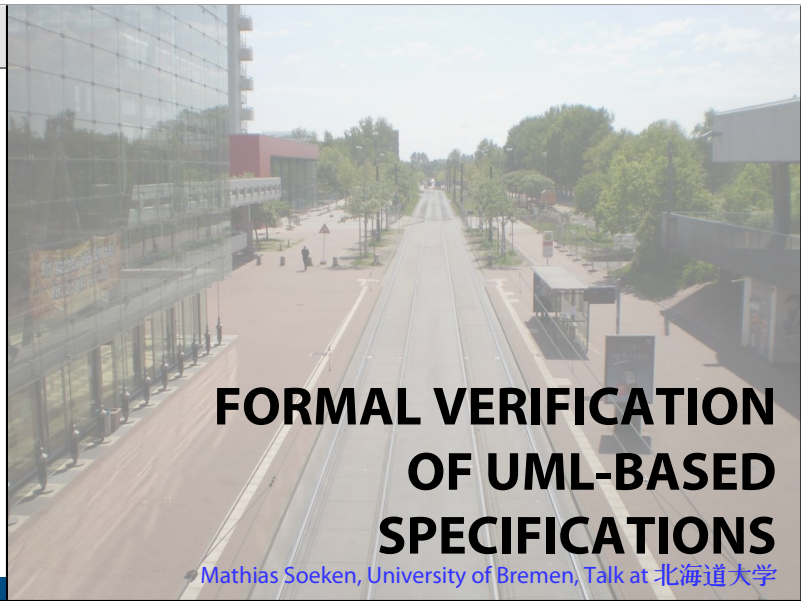
概要

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 verifcation 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)

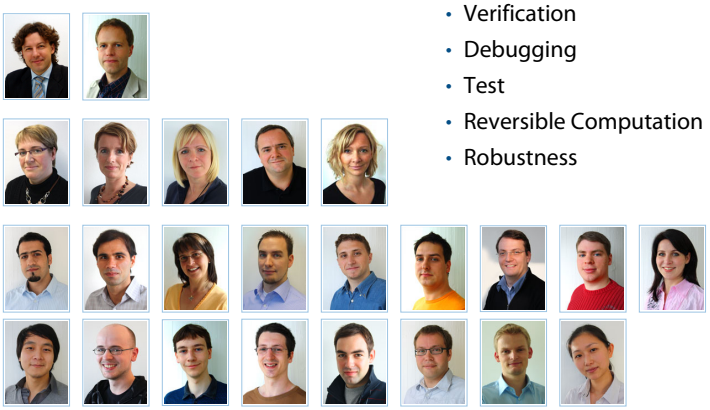
Curriculum Vitae

- 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 WWW

Computer Architecture at University of Bremen WWW



- Verification
- Debugging
- Test
- Reversible Computation
- Robustness

Specification Verification

Specification (as Textbook)

System model (e.g. in SystemC)

Specification Verification

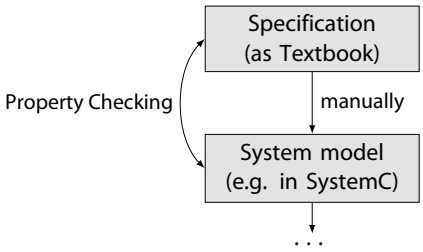
Specification (as Textbook)

manually

System model (e.g. in SystemC)

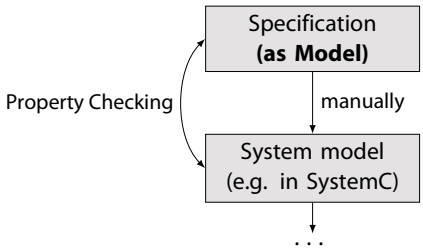
. . .

Specification Verification



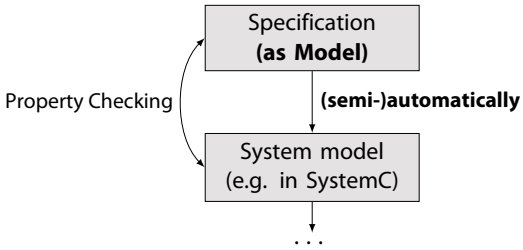
Universität Bremen - Computer Architecture

Specification Verification



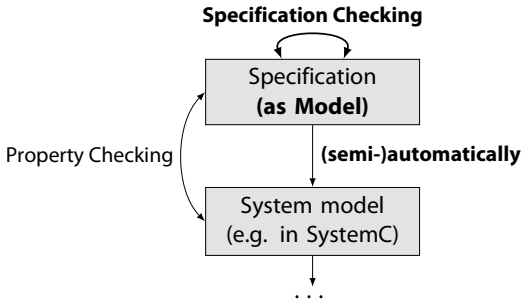
Universität Bremen - Computer Architecture

Specification Verification

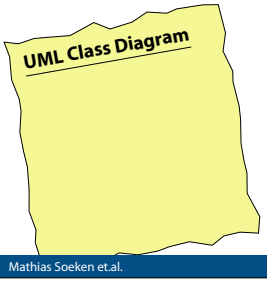


Universität Bremen - Computer Architecture

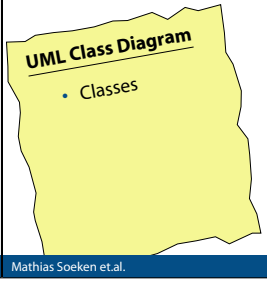
Specification Verification

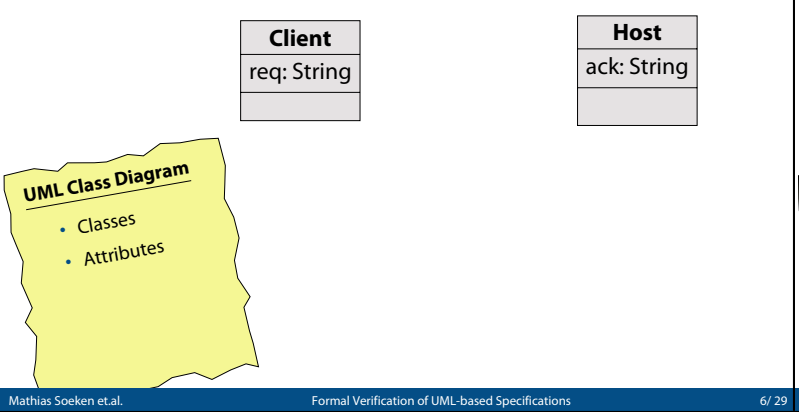


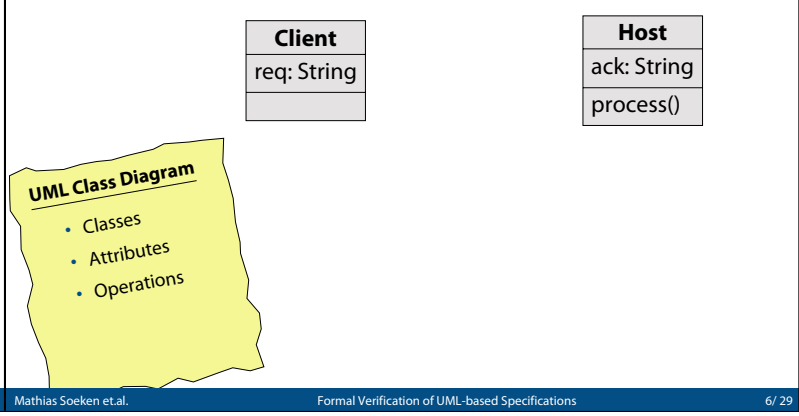
Universität Bremen - Computer Architecture

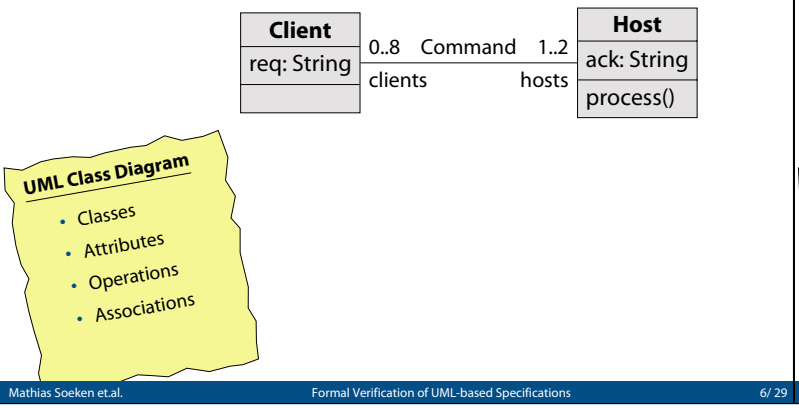


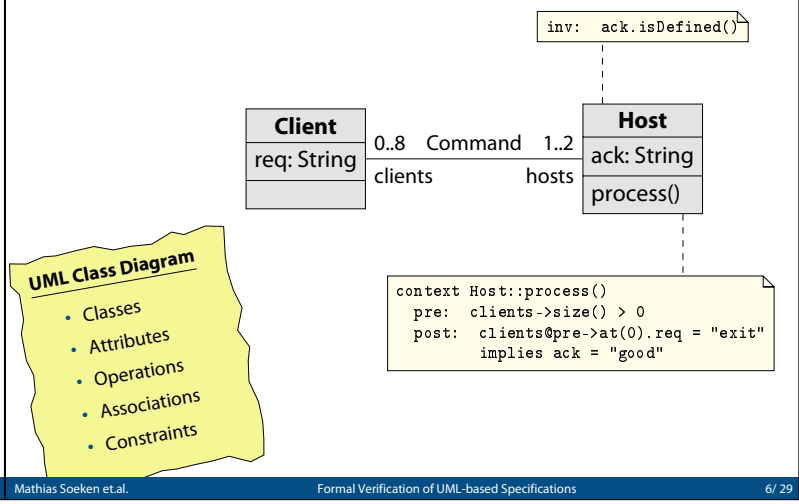
Universität Bremen - Computer Architecture

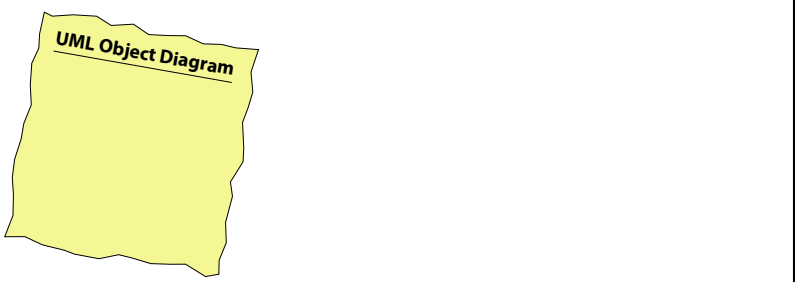


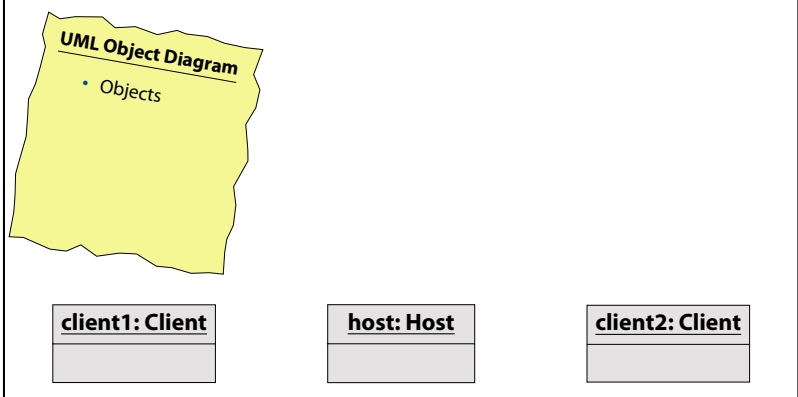

Client
req: String
Host
ack: String
UML Class Diagram
Classes
Attributes


Client
req: String
Host
ack: String
process()
UML Class Diagram
Classes
Attributes
Operations


Client
req: String
0..8
Command
1..2
clients
hosts
Host
ack: String
process()
UML Class Diagram
Classes
Attributes
Operations
Associations


inv: ack.isDefined()
Client
req: String
0..8
Command
1..2
clients
hosts
Host
ack: String
process()
context Host::process()
pre: clients->size() > 0
post: clients@pre->at(0).req = "exit"
implies ack = "good"
UML Class Diagram
Classes
Attributes
Operations
Associations
Constraints


UML Object Diagram


UML Object Diagram
Objects
client1: Client
host: Host
client2: Client

UML Object Diagram
- Objects
- Attributes

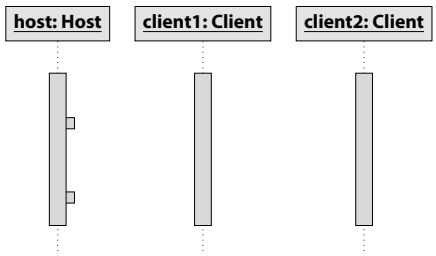
client1: Client	host: Host	client2: Client
req = "date"	ack = "good"	req = "exit"

UML Object Diagram
- Objects
- Attributes
- Links

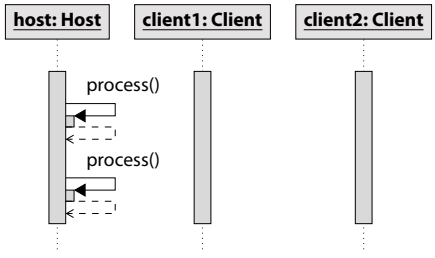
client1: Client	Command	host: Host	Command	client2: Client
req = "date"		ack = "good"		req = "exit"

UML Sequence Diagram

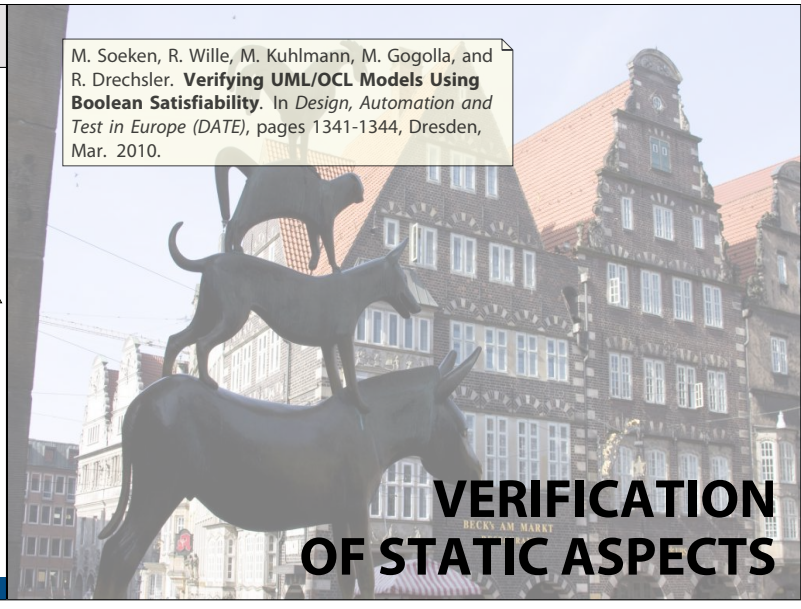
UML Sequence Diagram
- Objects



UML Sequence Diagram
- Objects
- Operation Calls

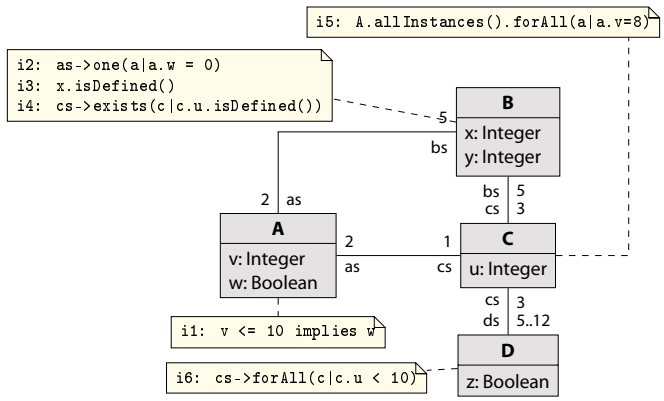


M. Soeken, R. Wille, M. Kuhlmann, M. Gogolla, and R. Drechsler. **Verifying UML/OCL Models Using Boolean Satisfiability**. In *Design, Automation and Test in Europe (DATE)*, pages 1341-1344, Dresden, Mar. 2010.



VERIFICATION OF STATIC ASPECTS

What is Inconsistency?



Universität Bremen - Computer Architecture

What is Inconsistency?

x_C objects of **C**

Mathias Soeken et.al. Formal Verification of UML-based Specifications 10/ 29

Universität Bremen - Computer Architecture

What is Inconsistency?

x_C objects of **C**
$2x_C$ objects of **A**

Mathias Soeken et.al. Formal Verification of UML-based Specifications 10/ 29

Universität Bremen - Computer Architecture

What is Inconsistency?

x_C objects of **C**
$2x_C$ objects of **A**
$5x_C$ objects of **B**

Mathias Soeken et.al. Formal Verification of UML-based Specifications 10/ 29

Universität Bremen - Computer Architecture

What is Inconsistency?

x_C objects of **C**
$2x_C$ objects of **A**
$5x_C$ objects of **B**
$3x_C$ objects of **C**

Mathias Soeken et.al. Formal Verification of UML-based Specifications 10/ 29

Universität Bremen - Computer Architecture

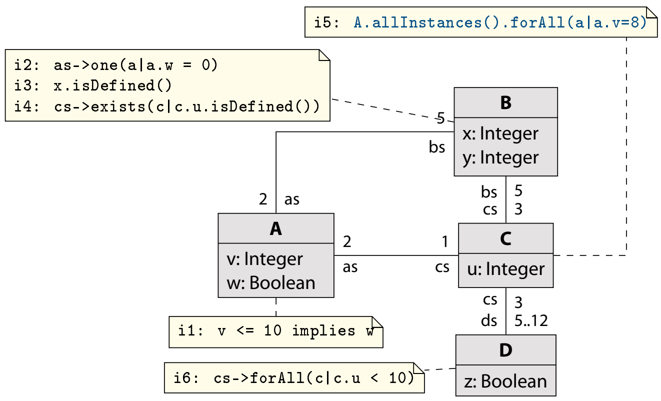
What is Inconsistency?

x_C objects of **C**
$2x_C$ objects of **A**
$5x_C$ objects of **B**
$3x_C$ objects of **C**

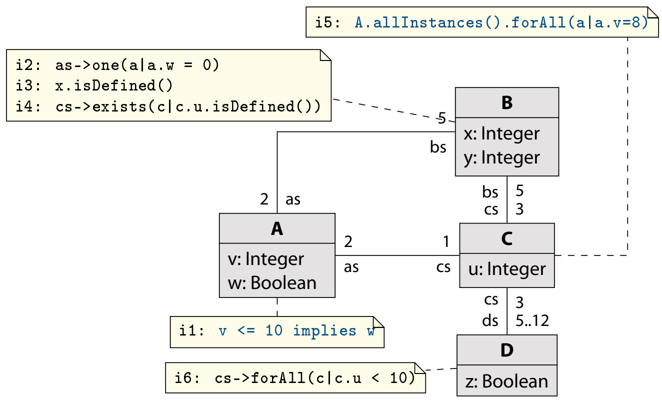
Model is UML-inconsistent

Mathias Soeken et.al. Formal Verification of UML-based Specifications 10/ 29

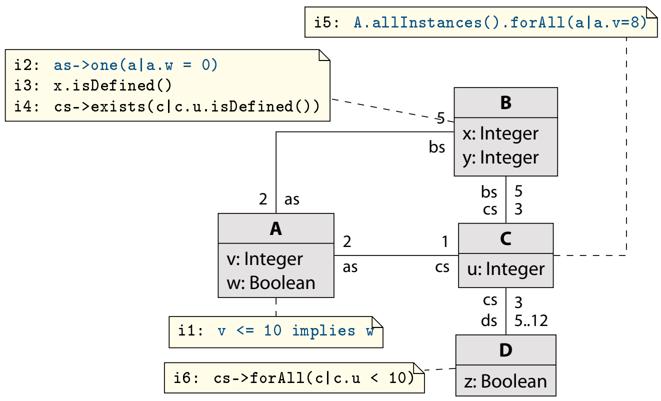
What is Inconsistency?



What is Inconsistency?

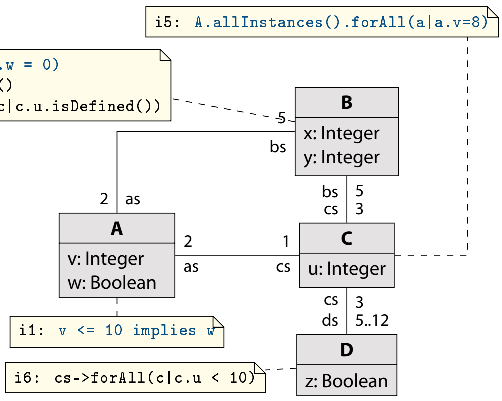


What is Inconsistency?

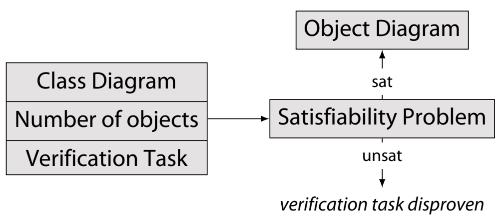


What is Inconsistency?

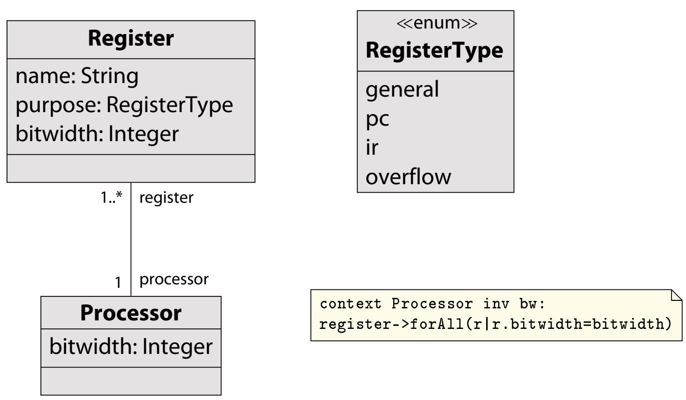
Model is OCL-inconsistent



Solving Flow

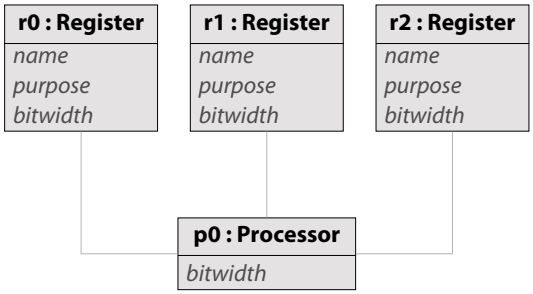


Running Example



Running Example

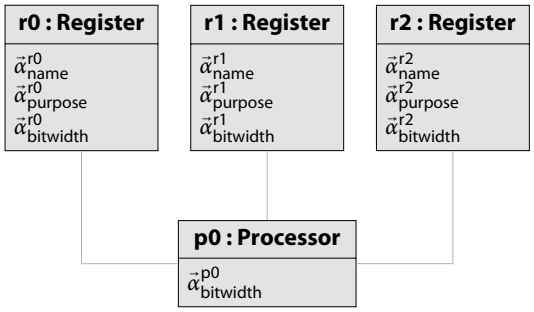
Is there a valid system state consisting of one processor and three registers?



Universität Bremen - Computer Architecture

Running Example

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



Mathias Soeken et.al. Formal Verification of UML-based Specifications 13/ 29

Universität Bremen - Computer Architecture

Encoding Attributes

How many bits are needed for the bit-vectors?

$\lceil \mathrm{ld}(n+1) \rceil$ bits are needed. The additional value is for representing undefined attributes. Determination of n by type of attribute:

Boolean $n = 2$

Universität Bremen - Computer Architecture

Encoding Attributes

How many bits are needed for the bit-vectors?

$\lceil \mathrm{ld}(n+1) \rceil$ 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

Mathias Soeken et.al. Formal Verification of UML-based Specifications 14/ 29

Universität Bremen - Computer Architecture

Encoding Attributes

How many bits are needed for the bit-vectors?

$\lceil \mathrm{ld}(n+1) \rceil$ 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

Mathias Soeken et.al. Formal Verification of UML-based Specifications 14/ 29

Universität Bremen - Computer Architecture

Encoding Attributes

How many bits are needed for the bit-vectors?

$\lceil \mathrm{ld}(n+1) \rceil$ 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

Integer $n = 2^l - 1$ for l-bit integers

Mathias Soeken et.al. Formal Verification of UML-based Specifications 14/ 29

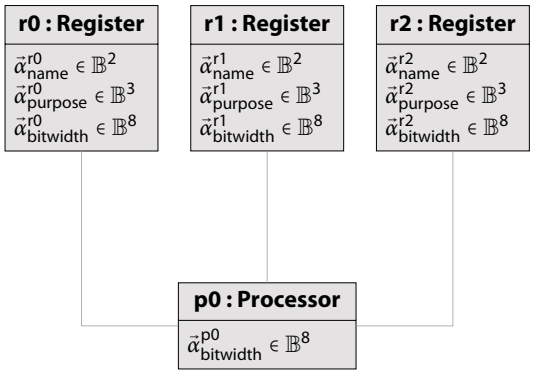
Encoding Attributes

How many bits are needed for the bit-vectors?

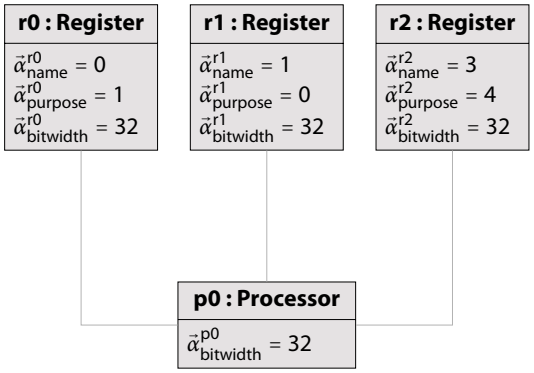
$\lceil \mathrm{ld}(n+1) \rceil$ bits are needed. The additional value is for representing undefined attributes. Determination of n by type of attribute:

- Boolean: $n = 2$
- Enumeration: n = Number of fields
- String: n = Number of all possible strings
- Integer: $n = 2^l - 1$ for l-bit integers

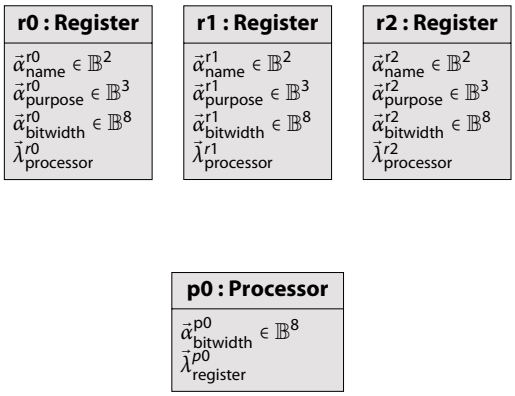
Running Example



Running Example



Running Example



Encoding Links

- $\vec{\lambda}^{p0}_{register}$ is a bit-mask and each bit enables or disables a possible link to a register
- $\vec{\lambda}^{p0}_{register} = \lambda_0\lambda_1\lambda_2$

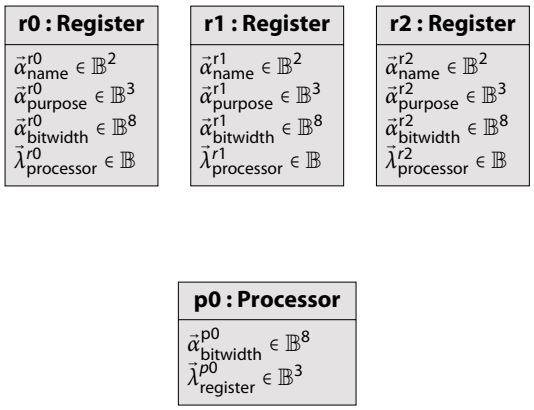
Encoding Links

- $\vec{\lambda}^{p0}_{register}$ is a bit-mask and each bit enables or disables a possible link to a register
- $\vec{\lambda}^{p0}_{register} = \lambda_0\lambda_1\lambda_2$
- There is a link between p0 and r*i* iff $\lambda_i = 1$

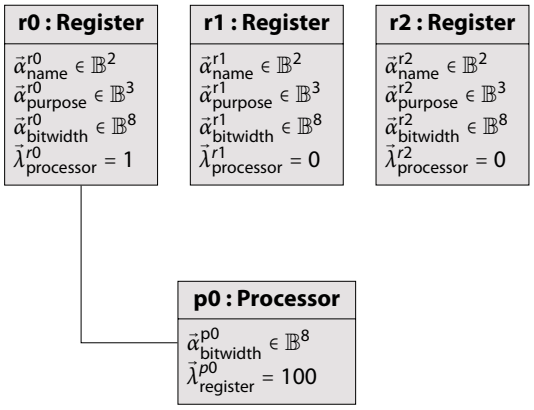
Encoding Links

- $\vec{\lambda}^{p0}_{register}$ is a bit-mask and each bit enables or disables a possible link to a register
- $\vec{\lambda}^{p0}_{register} = \lambda_0\lambda_1\lambda_2$
- There is a link between p0 and *ri* iff $\lambda_i = 1$

Running Example



Running Example



Encoding Invariants

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

$$\bigwedge_{i=0}^{|\text{oid(Register)}|-1} \left[\vec{\lambda}^{p0}_{register}[i] \Rightarrow \left(\vec{\alpha}^{ri}_{bitwidth} = \vec{\alpha}^{p0}_{bitwidth}\right)\right]$$

Encoding Invariants

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

$$\bigwedge_{i=0}^{|\text{oid(Register)}|-1} \left[\vec{\lambda}^{p0}_{register}[i] \Rightarrow \left(\vec{\alpha}^{ri}_{bitwidth} = \vec{\alpha}^{p0}_{bitwidth}\right)\right]$$

- 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)
```

$$\bigwedge_{i=0}^{|\text{oid(Register)}|-1} \left[\vec{\lambda}^{p0}_{register}[i] \Rightarrow \left(\vec{\alpha}^{ri}_{bitwidth} = \vec{\alpha}^{p0}_{bitwidth}\right)\right]$$

- 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 σ denote a system state, i.e. a concrete assignment of attributes and links

Universität Bremen - Computer Architecture

Verification Tasks

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

Universität Bremen - Computer Architecture

Verification Tasks

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

Consistency

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

Universität Bremen - Computer Architecture

Verification Tasks

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

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

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

Universität Bremen - Computer Architecture

Verification Tasks

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

Consistency

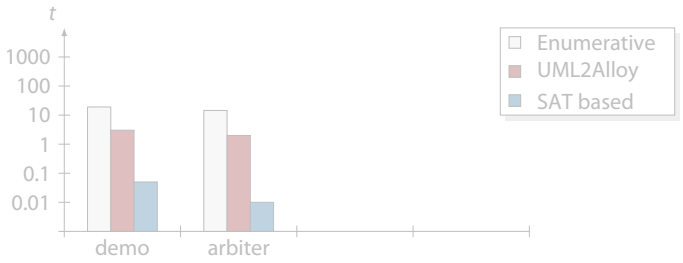
$$\exists \sigma : \bigwedge_{i \in \mathcal{I}} \sigma(i)$$

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

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

Experimental Results

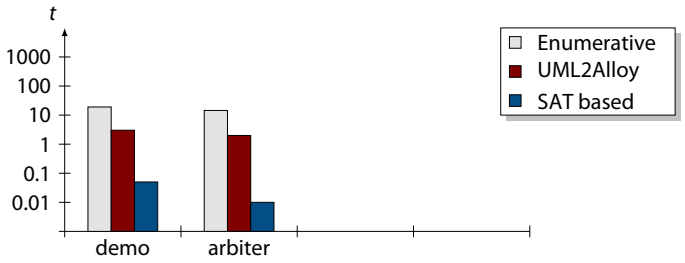
- Consistency check for consistent models
- Enumerative approach is USE
- UML2Alloy converts UML models to an Alloy model which is solved by a SAT solver



Universität Bremen - Computer Architecture

Experimental Results

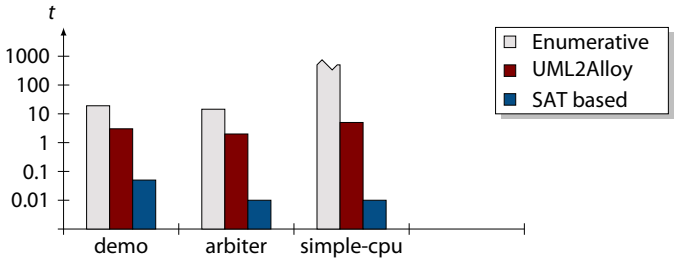
- Consistency check for consistent models
- Enumerative approach is USE
- UML2Alloy converts UML models to an Alloy model which is solved by a SAT solver



Universität Bremen - Computer Architecture

Experimental Results

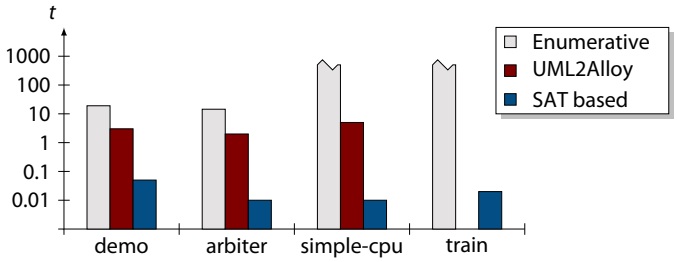
- Consistency check for consistent models
- Enumerative approach is USE
- UML2Alloy converts UML models to an Alloy model which is solved by a SAT solver

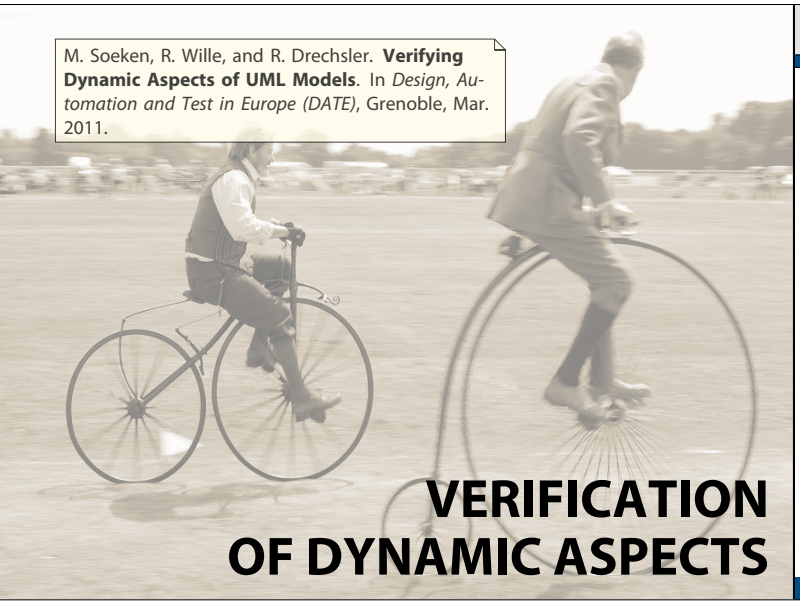


Universität Bremen - Computer Architecture

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



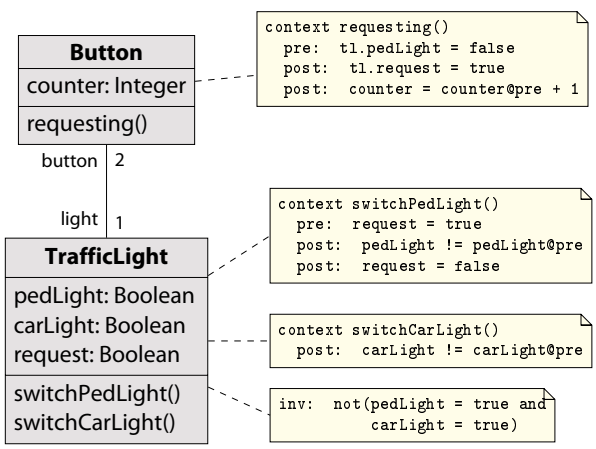


M. Soeken, R. Wille, and R. Drechsler. **Verifying Dynamic Aspects of UML Models**. In *Design, Automation and Test in Europe (DATE)*, Grenoble, Mar. 2011.

VERIFICATION OF DYNAMIC ASPECTS

Universität Bremen - Computer Architecture

Running Example



Running Example

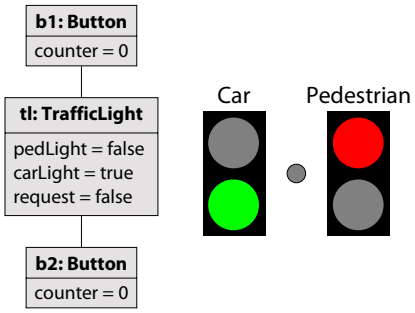
Time: 0

```
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()
post:  carLight != carLight@pre

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



Universität Bremen - Computer Architecture

Running Example

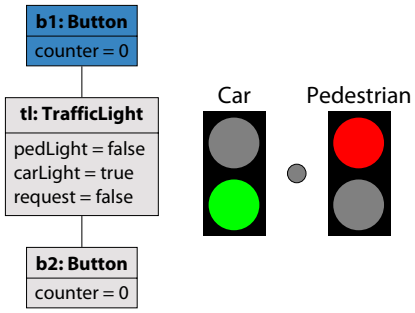
Time: 0

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()
post:  carLight != carLight@pre

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



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

Universität Bremen - Computer Architecture

Running Example

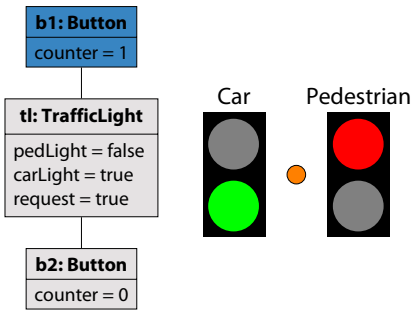
Time: 1

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)
```



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

Universität Bremen - Computer Architecture

Running Example

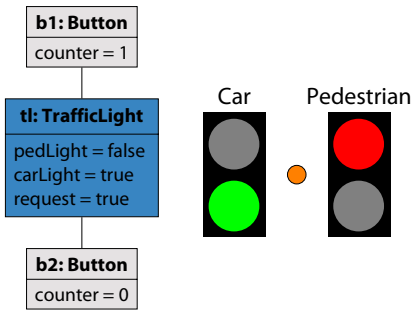
Time: 1

```
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)
```



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

Universität Bremen - Computer Architecture

Running Example

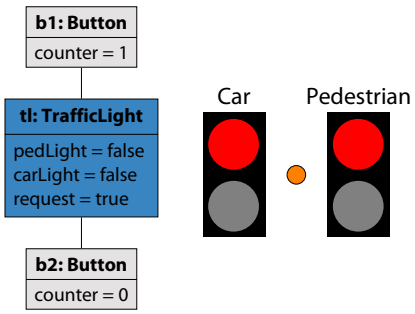
Time: 2

```
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)
```



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

Universität Bremen - Computer Architecture

Running Example

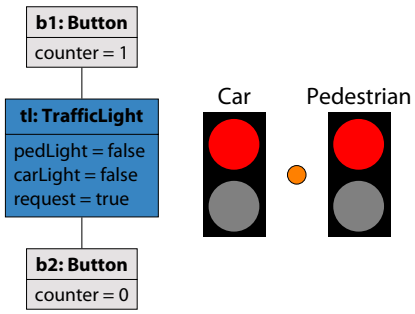
Time: 2

```
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)
```



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

Running Example

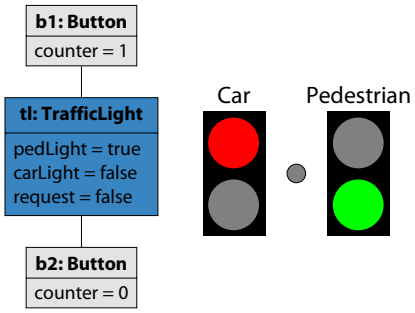
Time: 3

```
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)
```



Universität Bremen - Computer Architecture

Running Example

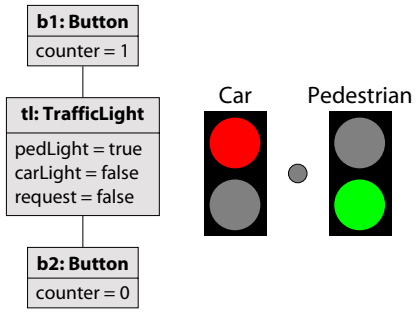
Time: 3


```
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)
```

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

Universität Bremen - Computer Architecture

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

Universität Bremen - Computer Architecture

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

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

Universität Bremen - Computer Architecture

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

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

Universität Bremen - Computer Architecture

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

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 $\vec{\alpha}$ 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 $\triangleleft_{op}$ and $\triangleright_{op}$ with $op \in OP$ 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 $\triangleleft_{op}$ and $\triangleright_{op}$ with $op \in OP$ the pre- and post-conditions of the operation associated to the operation call op
- Then, each verification task τ can be described as:

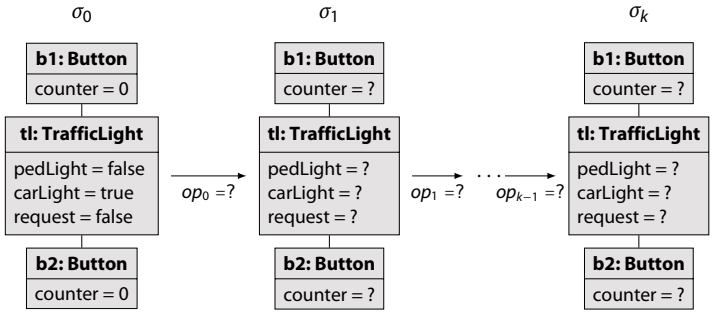
$$f = \bigwedge_{t=0}^{k} \sigma_t(\mathcal{I}) \wedge \bigwedge_{t=0}^{k-1} \left(\sigma_t(\triangleleft_{op_t}) \wedge \sigma_{t+1}(\triangleright_{op_t})\right) \wedge \tau$$

Encoding

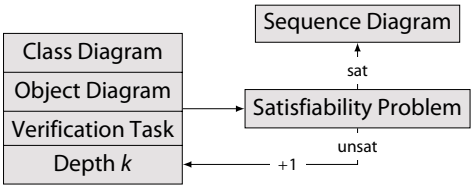
- Introduce time model, i.e. we consider k steps (operation calls)
- This leads to $k+1$ system states $\sigma_0, \ldots, \sigma_k$
- Create $\vec{\alpha}$ and $\vec{\lambda}$ variables for each time step
- Let OP be the set of all possible operation calls
- Let op_i with $i \in \{0, \ldots, k-1\}$ the operation call executed in system state i
- Besides the invariants, let $\triangleleft_{op}$ and $\triangleright_{op}$ with $op \in OP$ the pre- and post-conditions of the operation associated to the operation call op
- Then, each verification task τ can be described as:

$$f = \bigwedge_{t=0}^{k} \sigma_t(\mathcal{I}) \wedge \bigwedge_{t=0}^{k-1} \left(\sigma_t(\triangleleft_{op_t}) \wedge \sigma_{t+1}(\triangleright_{op_t})\right) \wedge \tau$$

Encoding

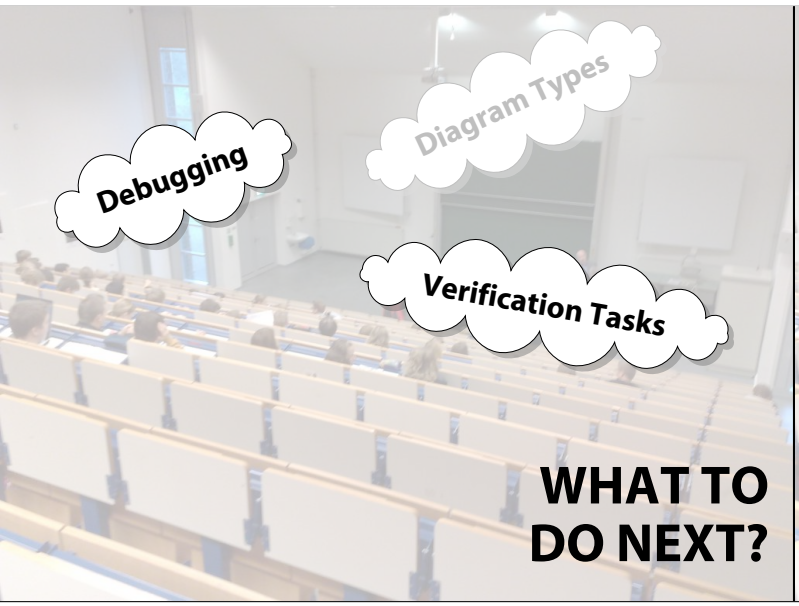


Solving Flow



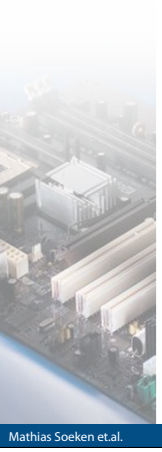
Experimental Results

Name	Task	#Obj	Depth	Status	Run-time
switch	Reachability	25	23	sat	49.70
switch	Reachability	25	22	unsat	50.00
switch	Reachability	25	50	sat	621.70
switch	Reachability	9	103	sat	147.50
switch	Reachability	9	102	unsat	90.40
simple-cpu2	State Gen.	13	100	sat	1.30
simple-cpu2	State Gen.	13	100	unsat	0.40
traffic-control	Reachability	6	5	sat	0.00
traffic-control	Reachability	24	10	sat	1.60
traffic-control	State Gen.	9	30	unsat	0.10
traffic-control	State Gen.	9	100	unsat	0.40



References

- M. Soeken, R. Wille, M. Kuhlmann, M. Gogolla, and R. Drechsler. **Verifying UML/OCL Models Using Boolean Satisfiability**. In *Design, Automation and Test in Europe (DATE)*, pages 1341-1344, Dresden, Mar. 2010. PDF
- M. Soeken, R. Wille, and R. Drechsler. **Verifying Dynamic Aspects of UML Models**. In *Design, Automation and Test in Europe (DATE)*, Grenoble, Mar. 2011. PDF



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