- Lecture 20 Continued:
  - Formal Methods in the Development Process
    - Verification
      - Model Decomposition, Resource Consumption
  - Conclusion

- Lecture 21: Code Generation

- Looking Back (and Forward: Exam)

- Advertisements
The Story So Far...
The Project: Wireless Fire Alarm System

- Develop new communication protocol for wireless fire alarm systems (WFAS).
- Main functionality:
  - self-monitoring, and (display non-operational sensors at central unit)
  - alarm notification. (display fire indications (smoke, heat, etc.) at central unit)
- Timing constraints are regulated by European Norm EN 54, Part 25.
- Goal: satisfy EN 54-25 — and have a good, robust, efficient overall product.

Requirements Validation Cont’d

Two broad directions:
- **Option 1**: teach DC (usually not economic).
- **Option 2**: serve as translator / mediator.

1. domain experts tell system scenario \( S \) (maybe keep back, whether allowed / forbidden).
2. FM expert translates system scenario to evolution \( I_S \).
3. FM expert evaluates formula on \( I_S \).
4. FM expert translates outcome to “allowed / forbidden by formula”.
5. compare expected outcome and real outcome.
The Project: Wireless Fire Alarm System

- Develop new communication protocol for wireless fire alarm systems (WFAS).
- Main functionality:
  - self-monitoring, and (display non-operational sensors at central unit)
  - alarm notification. (display fire indications (smoke, heat, etc.) at central unit)
- Timing constraints are regulated by European Norm EN 54, Part 25.
- Goal: satisfy EN 54-25 — and have a good, robust, efficient overall product.

Requirements Validation Cont’d

Two broad directions:
- Option 1: teach DC (usually not economic).
- Option 2: serve as translator / mediator.

Self-Monitoring: Sensor
Self-Monitoring: Model Architecture
Lecture 20 Continued:

- Formal Methods in the Development Process
  - Verification
    - Model Decomposition, Resource Consumption
  - Conclusion

Lecture 21: Code Generation

Looking Back (and Forward: Exam)

Advertisements
Verification
• Queries:
  
  • E<> switcher.DETECTION
    
    **sanity-check**: “it is possible to detect one missing sensor”
    (check *with* sensor switcher and *with* channel blocker)
  
  • A[] not deadlock
    
    **sanity-check**: no deadlock
  
  • A[] (switcher.DETECTION imply switcher.timer <= 300*Second)
    
    **requirement**: “detection takes at most 300 s”
    (check *with* sensor switcher and *with* channel blocker)
  
  • A[] !center.ERROR
    
    **requirement**: “no spurious errors”
    (check *without* sensor switcher, *with* channel blocker)
Model Decomposition
Model Decomposition
## Verification Results: Self-Monitoring

<table>
<thead>
<tr>
<th>Query</th>
<th>Sensors as slaves, $N = 126$.</th>
<th>Repeaters as slaves, $N = 10$.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>seconds</td>
<td>MB</td>
</tr>
<tr>
<td>Detection possible E&lt;&gt; switcher.DETECTION</td>
<td>10,205.13</td>
<td>557.00</td>
</tr>
<tr>
<td>No message collision A[] not deadlock</td>
<td>12,895.17</td>
<td>2,343.00</td>
</tr>
<tr>
<td>Detect$_T$ A[] (switcher.DETECTION imply switcher.timer &lt;= 300*Second)</td>
<td>36,070.78</td>
<td>3,419.00</td>
</tr>
<tr>
<td>NoSpur$_T$ A[] !center.ERROR</td>
<td>97.44</td>
<td>44.29</td>
</tr>
</tbody>
</table>

(Opteron 6174 2.2Ghz, 64GB, UPPAAL 4.1.3 (64-bit), options -s -t0 -u)
## Models and Corresponding Sizes

<table>
<thead>
<tr>
<th>Model</th>
<th>Templates</th>
<th>Instances</th>
<th>Total Locations</th>
<th>Clocks</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Self-Monitoring:</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Sensors as slaves</td>
<td>9</td>
<td>137</td>
<td>1040</td>
<td>6</td>
</tr>
<tr>
<td>Repeaters as slaves</td>
<td>9</td>
<td>21</td>
<td>82</td>
<td>6</td>
</tr>
<tr>
<td><strong>Alarm:</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>One alarm</td>
<td>6</td>
<td>16</td>
<td>101</td>
<td>16</td>
</tr>
<tr>
<td>Two alarms in 2 seconds</td>
<td>5</td>
<td>16</td>
<td>108</td>
<td>12</td>
</tr>
<tr>
<td>Ten simultaneous alarms</td>
<td>6</td>
<td>25</td>
<td>200</td>
<td>15</td>
</tr>
</tbody>
</table>
Queries:

- $A[] \neg \text{Center.ALARMED}$ imply $\text{time} < 10\ast\text{Second}$

  requirement: “exactly one alarm displayed within 10 s”

- $A[] (\neg \text{Sensor0.DONE} \mid\mid \neg \text{Sensor1.DONE})$ imply $\text{time} \leq 10\ast\text{Second}$

  requirement: “exactly two (simultaneous) alarms displayed within 10 s”

- $A[] (\neg \text{Sensor0.DONE} \mid\mid \neg \text{Sensor1.DONE} \mid\mid \ldots \mid\mid \neg \text{Sensor9.DONE})$ imply $\text{time} \leq 100\ast\text{Second}$

  requirement: “exactly ten (simultaneous) alarms displayed within 100 s”
## Verification Results: Alarm

### $T = T_1$ (palm tree, full collision)

<table>
<thead>
<tr>
<th>Query $T$</th>
<th>ids</th>
<th>$T$ (seconds)</th>
<th>MB (MB)</th>
<th>States expl.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alarm1$_T$</td>
<td></td>
<td>$3.6 \pm 1$</td>
<td>$43.1 \pm 1$</td>
<td>$59k \pm 15k$</td>
</tr>
<tr>
<td>A[] !Center.ALARMED imply time &lt; 10*Second</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Alarm2$_T$</td>
<td>sequential</td>
<td>$4.7$</td>
<td>$67.1$</td>
<td>$110,207$</td>
</tr>
<tr>
<td>A[] (!Sensor0.DONE</td>
<td></td>
<td>!Sensor1.DONE) imply time &lt;= 10*Second</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Alarm10$_T$</td>
<td>sequential</td>
<td>$44.6 \pm 11$</td>
<td>$311.4 \pm 102$</td>
<td>$641k \pm 159k$</td>
</tr>
<tr>
<td>optimized</td>
<td></td>
<td>$41.8 \pm 10$</td>
<td>$306.6 \pm 80$</td>
<td>$600k \pm 140k$</td>
</tr>
<tr>
<td>A[] (!Sensor0.DONE</td>
<td></td>
<td>!Sensor1.DONE</td>
<td></td>
<td>...</td>
</tr>
</tbody>
</table>

### $T = T_2$ (palm tree, limited collision)

<table>
<thead>
<tr>
<th>Query $T$</th>
<th>ids</th>
<th>$T$ (seconds)</th>
<th>MB (MB)</th>
<th>States expl.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alarm1$_T$</td>
<td></td>
<td>$1.4 \pm 1$</td>
<td>$38.3 \pm 1$</td>
<td>$36k \pm 14k$</td>
</tr>
<tr>
<td>A[] !Center.ALARMED imply time &lt; 10*Second</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Alarm2$_T$</td>
<td>sequential</td>
<td>$0.5$</td>
<td>$24.1$</td>
<td>$19,528$</td>
</tr>
<tr>
<td>A[] (!Sensor0.DONE</td>
<td></td>
<td>!Sensor1.DONE) imply time &lt;= 10*Second</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Alarm10$_T$</td>
<td>sequential</td>
<td>$17.3 \pm 6$</td>
<td>$179.1 \pm 61$</td>
<td>$419k \pm 124k$</td>
</tr>
<tr>
<td>optimized</td>
<td></td>
<td>$17.1 \pm 6$</td>
<td>$182.2 \pm 64$</td>
<td>$412k \pm 124k$</td>
</tr>
<tr>
<td>A[] (!Sensor0.DONE</td>
<td></td>
<td>!Sensor1.DONE</td>
<td></td>
<td>...</td>
</tr>
</tbody>
</table>
Testing the Real System

<table>
<thead>
<tr>
<th></th>
<th>Model sequential</th>
<th>Model optimized</th>
<th>Model test scenario</th>
<th>Measured Avg.</th>
</tr>
</thead>
<tbody>
<tr>
<td>First Alarm</td>
<td>3.26s</td>
<td>2.14s</td>
<td>3.31s</td>
<td>2.79s ± 0.53s</td>
</tr>
<tr>
<td>All 10 Alarms</td>
<td>29.03s</td>
<td>27.08s</td>
<td>29.81s</td>
<td>29.65s ± 3.26s</td>
</tr>
</tbody>
</table>
Lecture 20 Continued:
- Formal Methods in the Development Process
  - Verification
    - Model Decomposition, Resource Consumption
  - Conclusion

Lecture 21: Code Generation

Looking Back (and Forward: Exam)

Advertisements
Conclusion
Conclusion

- Verifying “a whole system design” (i.e., every bit and detail of: car, plane, even WFAS) can be very expensive,
  gaining confidence into “the core design ideas” (or crucial aspects of the design) can be much more feasible.

- **One approach:**
  - fix a **budget** (time, effort, ...),
  - identify and **formalise core requirements** (balance priority and budget),
  - **validate** using positive / negative **examples**,  
  - **model as far as possible**, on an appropriate level of abstraction (balance level of detail and budget),
  - **validate** using simulation of **example runs**,  
  - **verify as far as possible** (if infeasible: limit considered scenarios, at least simulate).

- **Other way round**: **fix the goal** of the formal analysis.
In my opinion,

- Everybody in this room (or on the “broadcast receiver” at home)
- has been exposed to all the knowledge and experience
- that it takes to do the WFAS project.

What’s your opinion?
Content

- Lecture 20 Continued:
  - Formal Methods in the Development Process
    - Verification
      - Model Decomposition, Resource Consumption
  - Conclusion

- Lecture 21: Code Generation

- Looking Back (and Forward: Exam)

- Advertisements
Lecture 21: Dependency on Central Scheduling
**Motivation** *(F.A., W., et al., FAOC, 2016)*

Question: Can't we generate the code **automatically**?
Code Generation from TA in the Literature

The Rendezvous Transition Rule may Block Senders

**Example:** (sender blocked in some configurations)

\[ l_0 \xrightarrow{a!} l_1 \quad m_0 \xrightarrow{a?} m_1 \xrightarrow{m_2} \]

**Example:** (sender never blocked)

\[ l_0 \xrightarrow{x > 1} a! \quad l_1 \quad m_0 \xrightarrow{a?} m_1 \xrightarrow{y \leq 1} m_2 \]

**Another Example:** (one of the senders blocked)

\[ l_0 \xrightarrow{a!} l_1 \quad m_0 \xrightarrow{a?} m_1 \quad n_0 \xrightarrow{a!} n_1 \]
**Operational semantics:**

labelled transition relations \( \stackrel{\Delta}{\rightarrow}\subseteq \text{Conf}(\mathcal{N}) \times \text{Conf}(\mathcal{N}), \)  
\( \text{Conf}(\mathcal{N}) = \{\langle \vec{l}, \nu \rangle \mid \nu \models I(\vec{l})\}. \)

- **(delay transition)** \( \langle \vec{l}, \nu \rangle \xrightarrow{t} \langle \vec{l}, \nu + t \rangle, \) \( t \in \mathbb{R}_0^{+}, \) if and only if \( \forall t' \in [0, t] \bullet \nu + t' \models I(\vec{l}). \)

- **(local action transition)** \( \langle \vec{l}, \nu \rangle \xrightarrow{\varphi_i, \vec{r}_i, \vec{l}'_i} \langle \vec{l}', \nu' \rangle, \) if and only if there is an edge \( e = (\ell, \tau, \varphi, \vec{r}, \ell') \) in \( \mathcal{A}_i \) such that \( \langle \vec{l}, \nu \rangle \models \text{loc} e \) and \( \langle \vec{l}', \nu' \rangle = \langle \vec{l}, \nu \rangle[e] \)

- **(rendezvous transition)** \( \langle \vec{l}, \nu \rangle \xrightarrow{a!} \langle \vec{l}', \nu' \rangle, \) if and only if there is an edge \( e_0 = (\ell_i, a!, \varphi_i, \vec{r}_i, \ell'_i) \) in \( \mathcal{A}_i \) such that \( \langle \vec{l}, \nu \rangle \models \text{loc} e_0, \) and there is an edge \( e_1 = (\ell_j, a?, \varphi_j, \vec{r}_j, \ell'_j) \) in \( \mathcal{A}_j, i \neq j, \) such that \( \langle \vec{l'}, \nu' \rangle \models \text{loc} e_1, \) and \( \langle \vec{l'}, \nu' \rangle = \langle \vec{l}, \nu \rangle[e_0; e_1]. \)
Lemma. A closed component network \( N_{loc} = \{A_1, \ldots, A_n\} \) does not depend on a global scheduler if and only if

- in each reachable configuration,
  - if there is a sending edge locally enabled, then
    - there is at least one locally enabled receiver in a different automaton,
    - and no other sending edge in a different automaton,

i.e.

\[
\forall c \in \text{Conf}(N_{loc})|_{\text{reach}} \forall 1 \leq i \leq n \forall a \in A \forall e \in E(A_i)|_a ! \bullet c \vdash_{loc} e \\
\implies (c \vdash e \land \forall 1 \leq j \leq n \forall b \in A \forall e' \in E(A_j)|_b ! \bullet c \vdash_{loc} e' \implies j = i).
\]
Content

- Lecture 20 Continued:
  - Formal Methods in the Development Process
    - Verification
      - Model Decomposition, Resource Consumption
  - Conclusion

- Lecture 21: Code Generation

- Looking Back (and Forward: Exam)

- Advertisements
Wrapup
Content

Introduction

- Observables and Evolutions
- Duration Calculus (DC)
- Semantical Correctness Proofs
- DC Decidability
- DC Implementables
- PLC-Automata

- Timed Automata (TA), Uppaal
- Networks of Timed Automata
- Region/Zone-Abstraction
- TA model-checking
- Extended Timed Automata
- Undecidability Results

\[ \text{obs} : \text{Time} \rightarrow \mathcal{D}(\text{obs}) \]  
\[ \langle \text{obs}_0, \nu_0 \rangle, t_0 \xrightarrow{\lambda_0} \langle \text{obs}_1, \nu_1 \rangle, t_1 \ldots \]

- Automatic Verification...
  ...whether a TA satisfies a DC formula, observer-based

- Recent Results:
  - Timed Sequence Diagrams
  - Quasi-equal Clocks
  - Automatic Code Generation
  ...
21 – 2018-02-06 – Swrapup –

Looking Back

- Lect. 1: **real-time system** (vs. hybrid), state variables
- Lect. 2: **evolutions**, timing diagrams, classes of timed properties
- Lect. 3: **DC symbols**, state assertions, terms (syntax / semantics)
- Lect. 4: **DC formulae**, abbreviations, satisfiable / realisable / valid (from 0)
- Lect. 5: semantics-based **correctness proof**; real-world obstacles
- Lect. 6: **DC calculus**; **decidability** of RDC / discrete time
  - Lect. 7: **undecidability** of RDC / continuous time
- Lect. 8: **DC Implementables**, standard forms, control automata
- Lect. 9: **PLC**: characteristics, programming model
- Lect. 10: PLC automata, DC semantics
- Lect. 11: **timed automata** (syntax / semantics); tr. seq. / comp. path / run
- Lect. 12: **parallel composition** of TA (syntactical / semantical); **Uppaal**
- Lect. 13: **TA location reachability**, time-abstract system, regions
- Lect. 14: **zones**, zone-based reachability, Difference-Bounds-Matrices
- Lect. 15: **Extended Timed Automata** (variables, urgent/committed)
- Lect. 16: **query language**, evolutions vs. transition sequences
- Lect. 17: **testability**, observer construction, untestable DC formulae
- Lect. 18: undecidability results for **Timed Büchi Automata**
- Lect. 19: **quasi-equal clocks**, bisimulation
- Lect. 20: formal methods for **RTS in practice**

prelim. exam axes:
1 sheet of A4
Advertisements (Again)
ADVERTISEMENTS

ADVERTISEMENTS

ADVERTISEMENTS

ADVERTISEMENTS

ADVERTISEMENTS

ADVERTISEMENTS
Advertisements

- **BSc. / MSc. projects** (6 – 16 ECTS)
- **BSc, / MSc. thesis**
  - modelling real-time systems
  - extend timed automata **tools**
  - work on timed automata **theory**
  - ⟨ your (real-time) topic here ⟩

- **Student assistant jobs**
  - programming
  - modelling

- **Tutor jobs**
  - e.g., **Software Engineering** in Summer 2018

→ **contact me**
References
References

I have improved my capabilities in scientific problem solving. (Ich habe meine Fähigkeiten im wissenschaftlichen Problemlösen verbessert.)

- (1) **task** (in own words),
- (2) **solution** (in full sentences),
- (3) **correctness argument**.

That’s already “half of the story”. ; - )