Real-Time Systems

Lecture 21: Wrapup & Questions

2018-02-06

Dr. Bernd Westphal

Albert-Ludwigs-Universität Freiburg, Germany

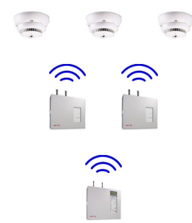
Content

- **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**

The Story So Far…

Project, Situation, Requirements

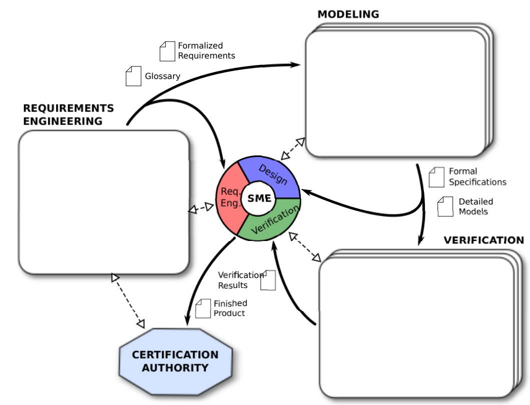
The Project: Wireless Fire Alarm System



(Arenis et al., 2016)

- 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.

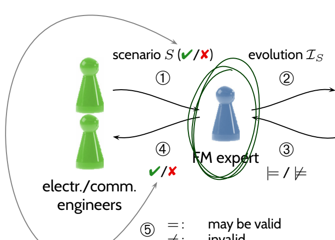
Formal Methods for SME



Requirements Validation Cont'd

Two broad directions:

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

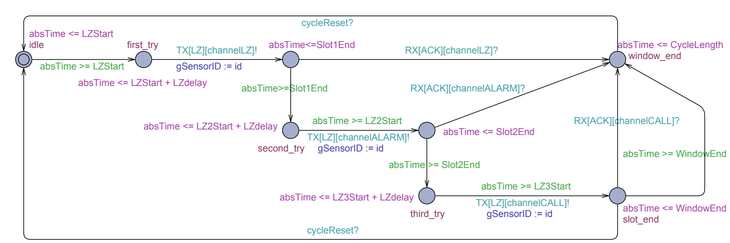


$$\bigwedge_{i \in C} \Box (\lceil \mathit{FAIL} = i \wedge \neg \mathit{DET}_i \rceil \implies \ell \leq 300\mathrm{s}) \quad (\mathrm{Detect}_T)$$

$$\bigwedge_{i \in C} \Box (\lceil \mathit{DET}_i \wedge \neg \mathit{DISP}_i \rceil \implies \ell \leq 100\mathrm{s}) \quad (\mathrm{Display}_T)$$

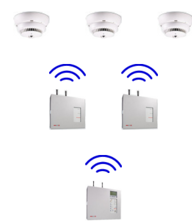
① domain experts **tell** system scenario S (maybe keep back, whether allowed / forbidden),
② FM expert **translates** system scenario to evolution $\mathcal{I}_S$,
③ FM expert **evaluates** formula on $\mathcal{I}_S$,
④ FM expert **translates** outcome to "allowed / forbidden by formula",
⑤ compare expected outcome and real outcome.

Self-Monitoring: Sensor



Project, Situation, Requirements

The Project: Wireless Fire Alarm System



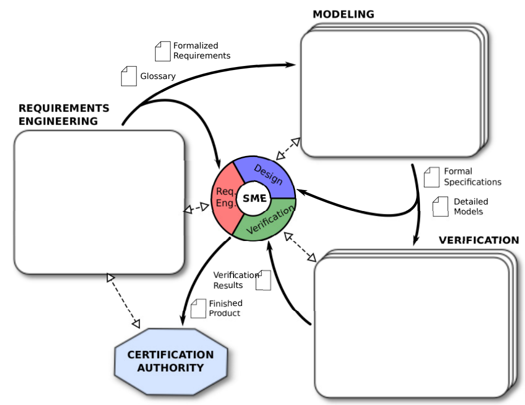
(Arenis et al., 2016)

- 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.

– 20 – 2018-02-01 – Sproject –

4/52

Formal Methods for SME



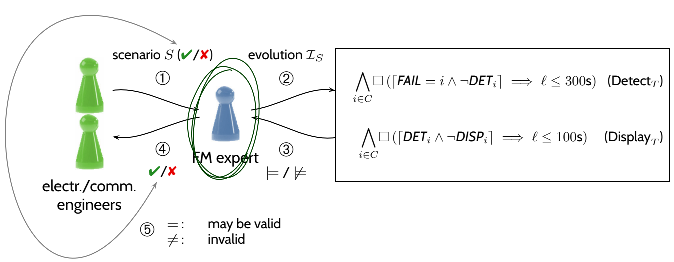
– 20 – 2018-02-01 – Sprocess –

10/52

Requirements Validation Cont'd

Two broad directions:

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



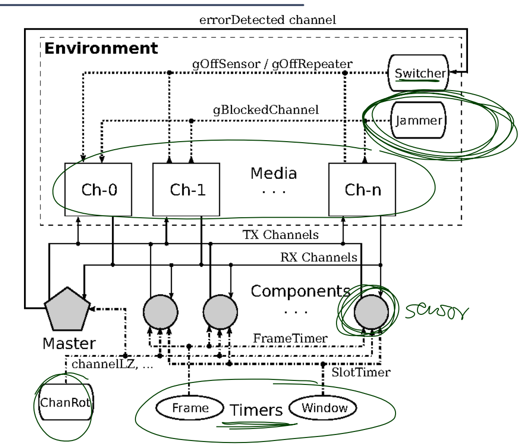
① domain experts **tell** system scenario S (maybe keep back, whether allowed / forbidden),
② FM expert **translates** system scenario to evolution $\mathcal{I}_S$,
③ FM expert **evaluates** formula on $\mathcal{I}_S$,
④ FM expert **translates** outcome to "allowed / forbidden by formula",
⑤ compare expected outcome and real outcome.

– 20 – 2018-02-01 – Svalid –

24/52

Self-Monitoring: Sensor

Self-Monitoring: Model Architecture



– 20 – 2018-02-01 – Smodel –

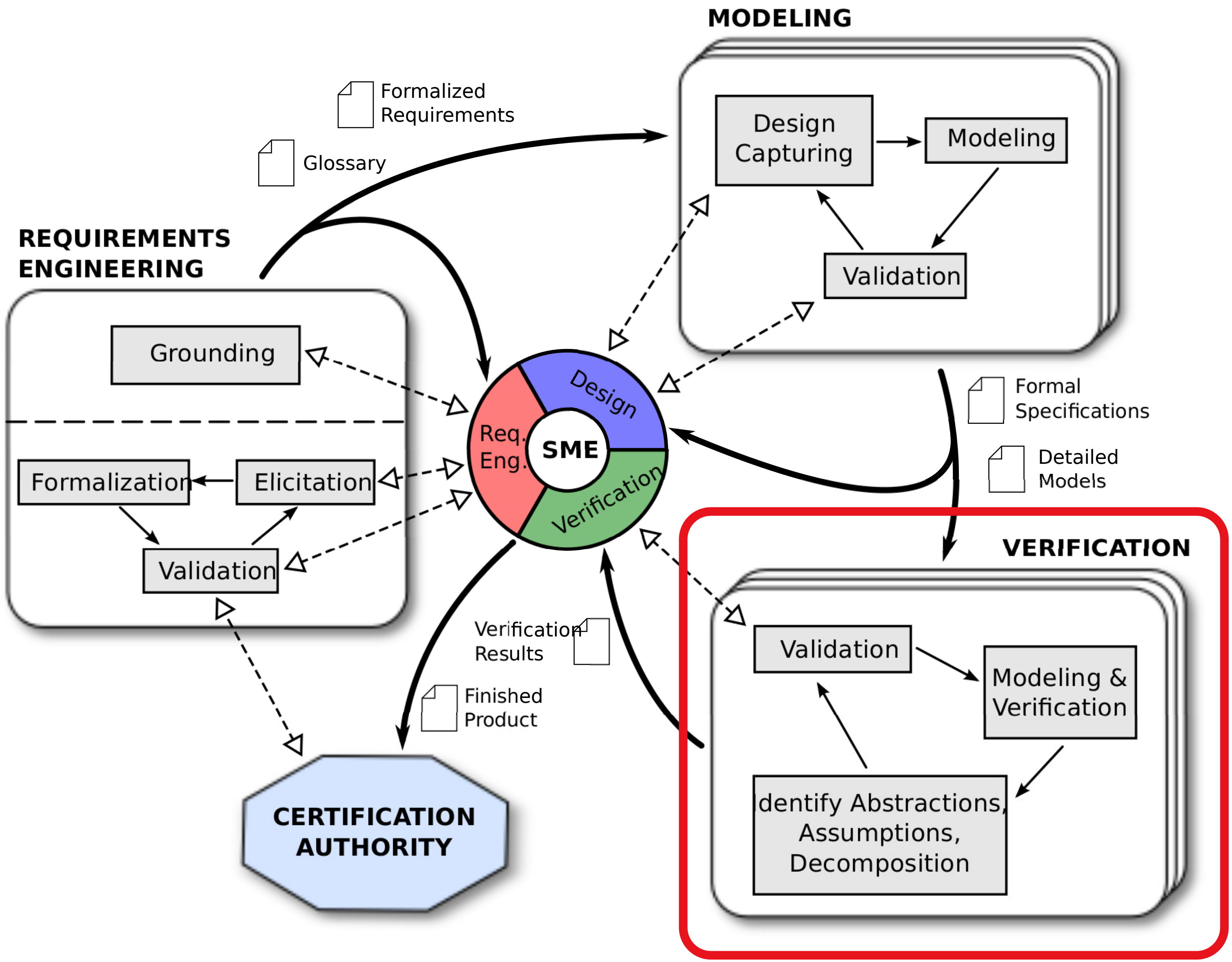
33/52

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**

Verification

Formal Verification



From DC Formulae to Queries: Self-Monitoring

- **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"

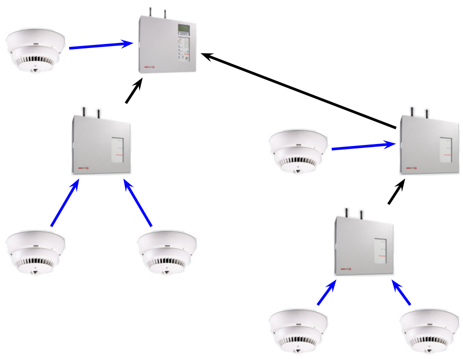
    (check **with** sensor switcher and **with** channel blocker)

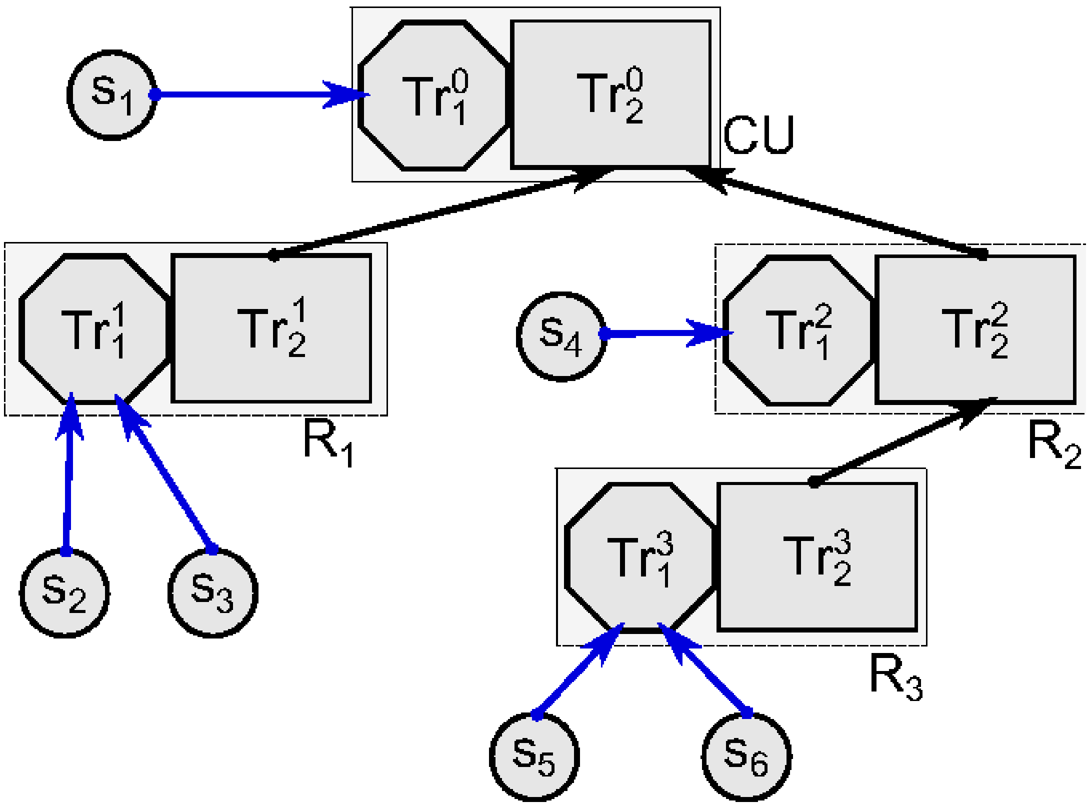
  - `A[] !center.ERROR`

    requirement: "no spurious errors"

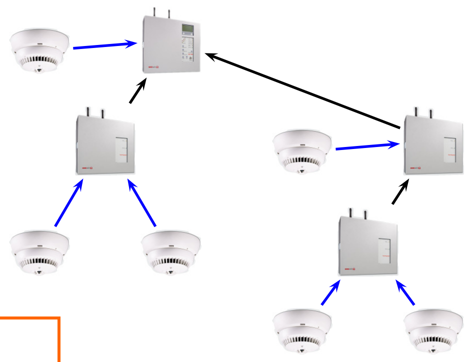
    (check **without** sensor switcher, **with** channel blocker)

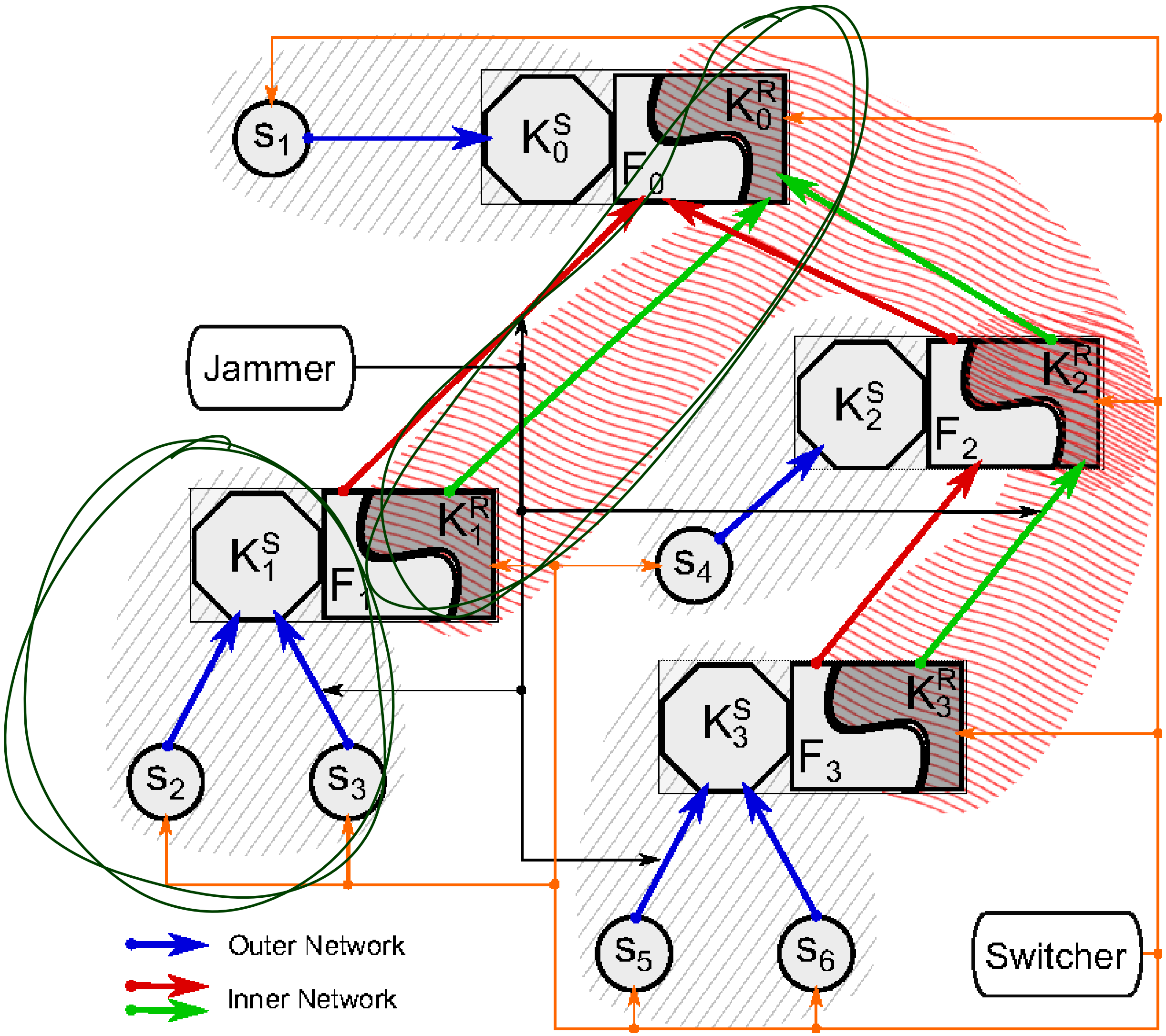
Model Decomposition





Model Decomposition





Verification Results: Self-Monitoring

Query	Sensors as slaves, $N = 126$. seconds	MB	States explored
Detection possible `E<> switcher.DETECTION`	10,205.13	557.00	26,445,788
No message collision `A[] not deadlock`	12,895.17	2,343.00	68,022,052
Detect_T `A[] (switcher.DETECTION imply switcher.timer <= 300*Second)`	36,070.78	3,419.00	190,582,600
NoSpur_T `A[] !center.ERROR`	97.44	44.29	640,943

Query	Repeaters as slaves, $N = 10$. seconds	MB	States explored
Detection possible `E<> switcher.DETECTION`	38.21	55.67	1,250,596
No message collision `A[] not deadlock`	368.58	250.91	9,600,062
Detect_T `A[] (switcher.DETECTION imply switcher.timer <= 300*Second)`	231.84	230.59	6,009,120
NoSpur_T `A[] !center.ERROR`	3.94	10.14	144,613

(Opteron 6174 2.2Ghz, 64GB, UPPAAL 4.1.3 (64-bit), options `-s -t0 -u`)

Models and Corresponding Sizes

Model	Templates	Instances	Total Locations	Clocks
Self-Monitoring:				
Sensors as slaves	9	137	1040	6
Repeaters as slaves	9	21	82	6
Alarm:				
One alarm	6	16	101	16
Two alarms in 2 seconds	5	16	108	12
Ten simultaneous alarms	6	25	200	15

From DC Formulae to Queries: Alarm

- **Queries**:
  - `A[] !Center.ALARMED imply time < 10*Second`

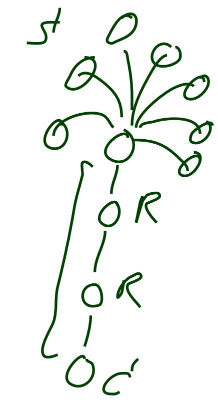
    requirement: "exactly **one alarm** displayed within 10 s"
  - `A[] (!Sensor0.DONE || !Sensor1.DONE) imply time <= 10*Second`

    requirement: "exactly **two (simultaneous) alarms** displayed within 10 s"
  - ```
    A[] (!Sensor0.DONE || !Sensor1.DONE || ... || !Sensor9.DONE)
        imply time <= 100*Second
    ```

    requirement: "exactly **ten (simultaneous) alarms** displayed within 100 s"
    ```

Verification Results: Alarm

Query	ids	$T = T_1$ (palm tree, full collision) seconds	MB	States expl.
Alarm1$_T$	-	3.6 ± 1	43.1 ± 1	$59k \pm 15k$
`A[] !Center.ALARMED imply time < 10*Second`				
Alarm2$_T$	sequential	4.7	67.1	$110,207$
`A[] (!Sensor0.DONE || !Sensor1.DONE) imply time <= 10*Second`				
Alarm10$_T$	sequential	44.6 ± 11	311.4 ± 102	$641k \pm 159k$
	optimized	41.8 ± 10	306.6 ± 80	$600k \pm 140k$
`A[] (!Sensor0.DONE || !Sensor1.DONE || ... || !Sensor9.DONE) imply time <= 100*Second`				



Query	ids	$T = T_2$ (palm tree, limited collision) seconds	MB	States expl.
Alarm1$_T$	-	1.4 ± 1	38.3 ± 1	$36k \pm 14k$
`A[] !Center.ALARMED imply time < 10*Second`				
Alarm2$_T$	sequential	0.5	24.1	$19,528$
`A[] (!Sensor0.DONE || !Sensor1.DONE) imply time <= 10*Second`				
Alarm10$_T$	sequential	17.3 ± 6	179.1 ± 61	$419k \pm 124k$
	optimized	17.1 ± 6	182.2 ± 64	$412k \pm 124k$
`A[] (!Sensor0.DONE || !Sensor1.DONE || ... || !Sensor9.DONE) imply time <= 100*Second`				

Testing the Real System

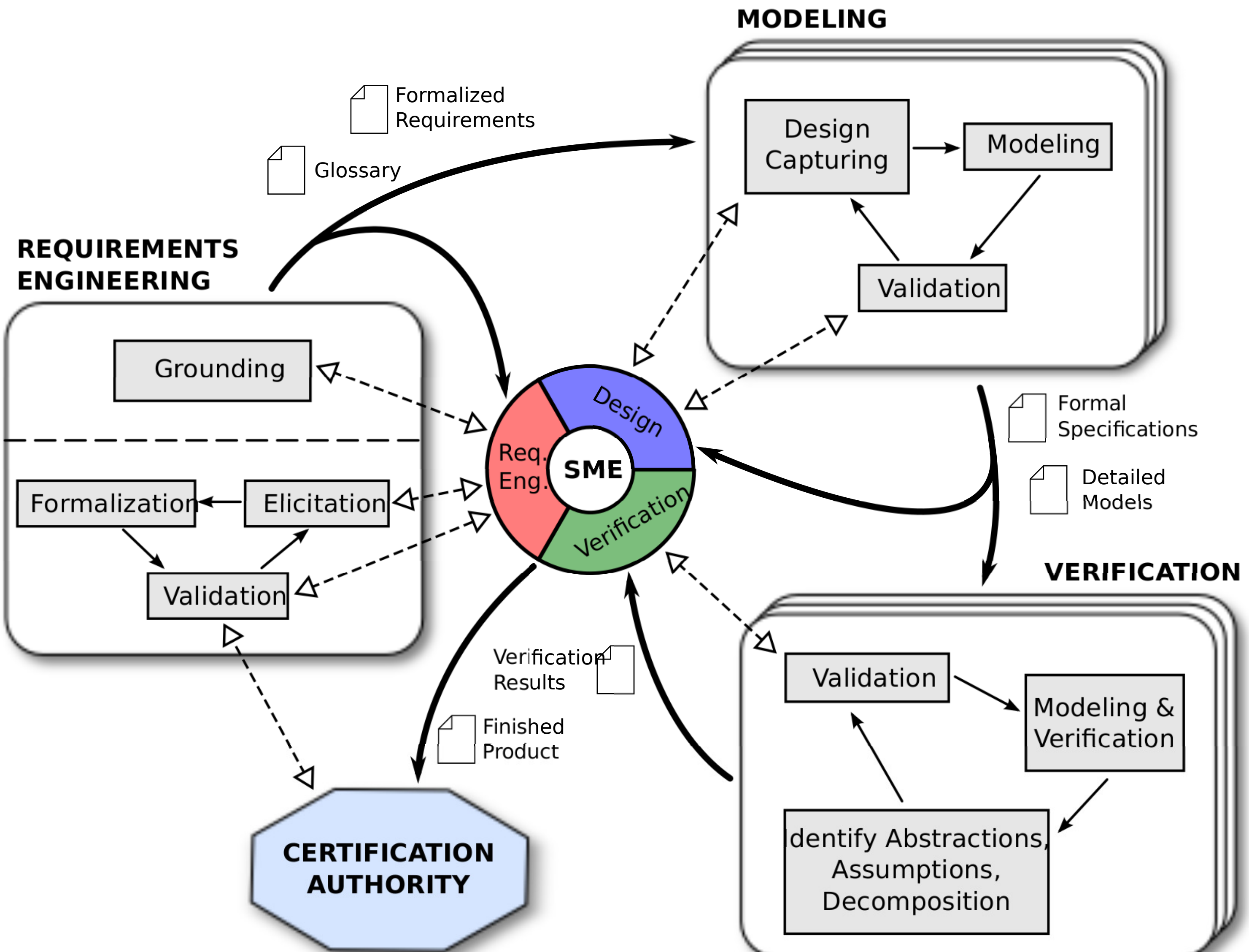
	Model sequential	Model optimized	Model test scenario	Measured Avg.
First Alarm	$3.26s$	$2.14s$	$3.31s$	$2.79s \pm 0.53s$
All 10 Alarms	$29.03s$	$27.08s$	$29.81s$	$29.65s \pm 3.26s$

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**

Conclusion

Looking Back



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.

Conclusion from the Conclusion

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?

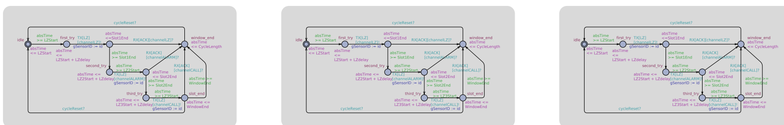
1,5M
6M
2M
M
4M
5M
6M
6-8M
6M
4-6M
6-8M
9M

Content

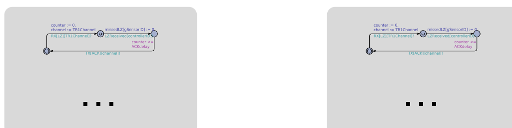
Lecture 21: Dependency on Central Scheduling

Motivation *(F. A., W., et al., FAOC, 2016)*

sensor TA sensor TA sensor TA



repeater TAs repeater TAs

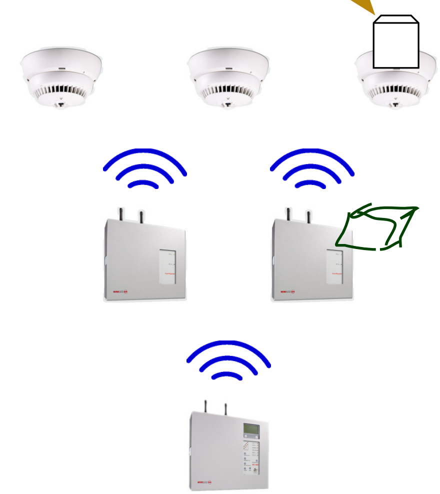


central unit TAs

implement manually

very regular implementation of location/edge behaviour

compile / deploy

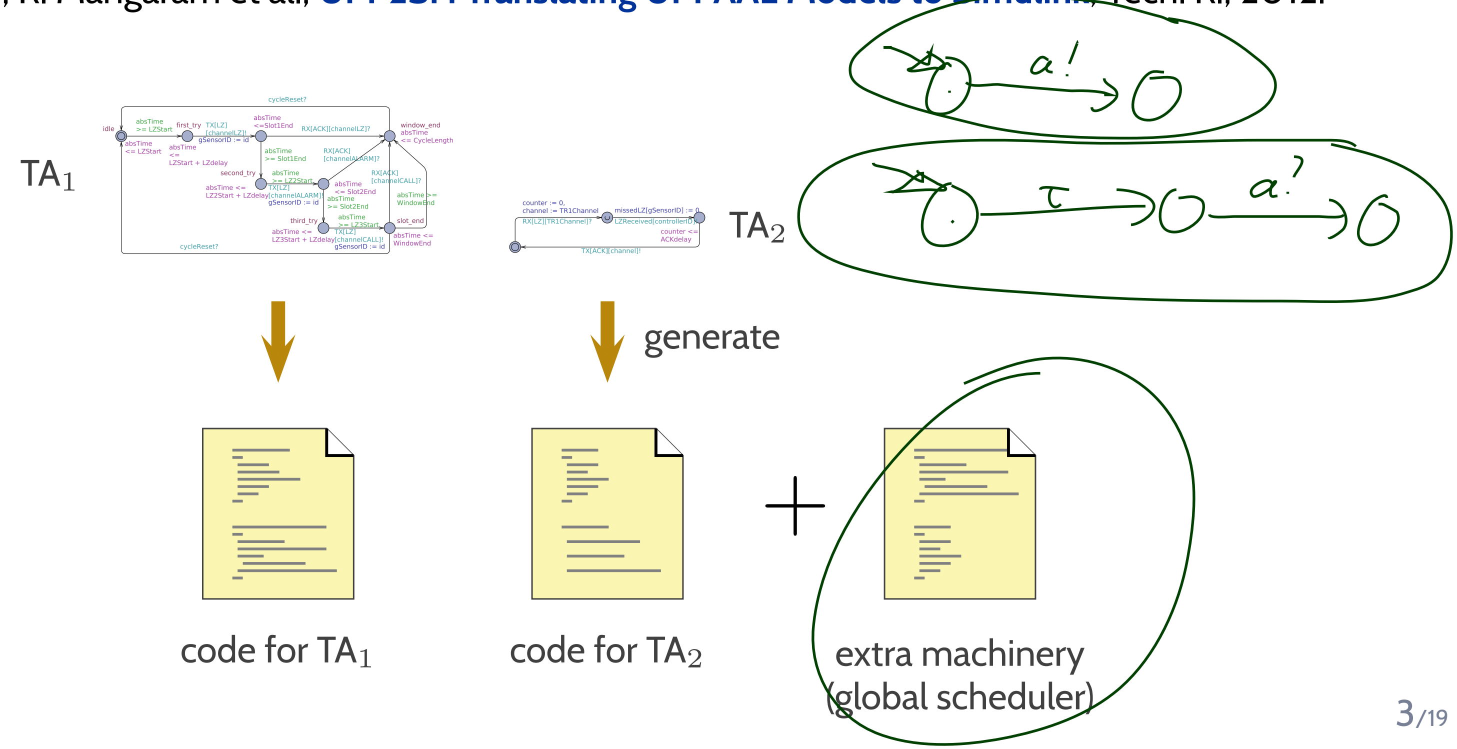


wireless fire alarm system

Question:
Can't we generate the code **automatically**?

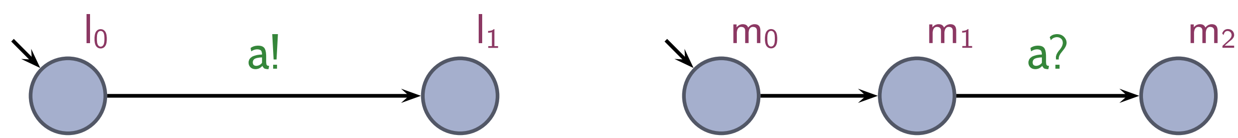
Code Generation from TA in the Literature

- M. Hendriks, **Translating UPPAAL to not quite C**, CSI-R0108, 2001.
- T. Amnell, E. Fersman, P. Pettersson, W. Yi, and H. Sun, **Code synthesis for TA**, Nordic JC, 2002.
- J. Kristensen, A. Mejlholm, and S. Pedersen, **Automatic translation from UPPAAL to C**, Tech. R., 2005.
- K. Altisen and S. Tripakis, **Implementation of TA: An issue of semantics or modeling?**, FORMATS, 2005.
- T. Abdellatif, J. Combaz, and J. Sifakis, **Model-based implementation of RT applications**, EMSOFT, 2010.
- N. Hakimipour, P. Strooper, A. Wellings, **TART: TA to Real-Time Java Tool**, SEFM, 2010.
- M. Pajic, I. Lee, R. Mangaram et al., **UPP2SF: Translating UPPAAL Models to Simulink**, Tech. R., 2012.

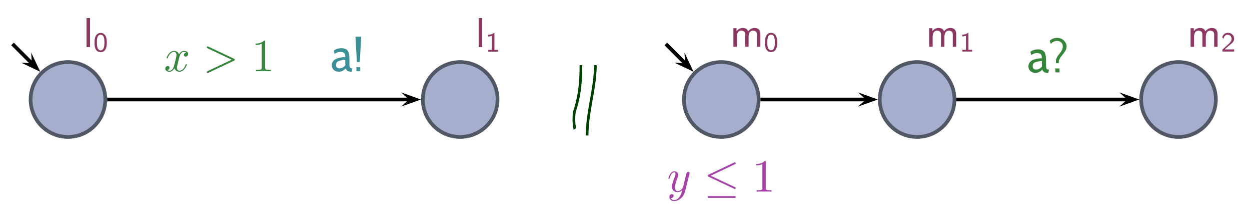


The Rendezvous Transition Rule may Block Senders

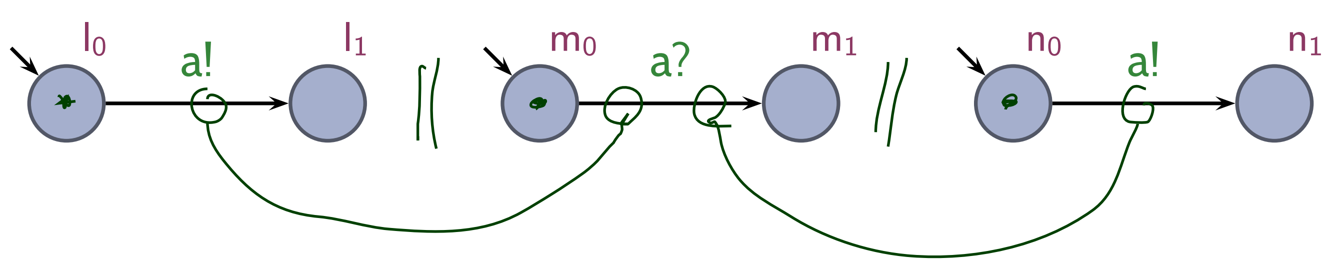
Example: (sender blocked in some configurations)



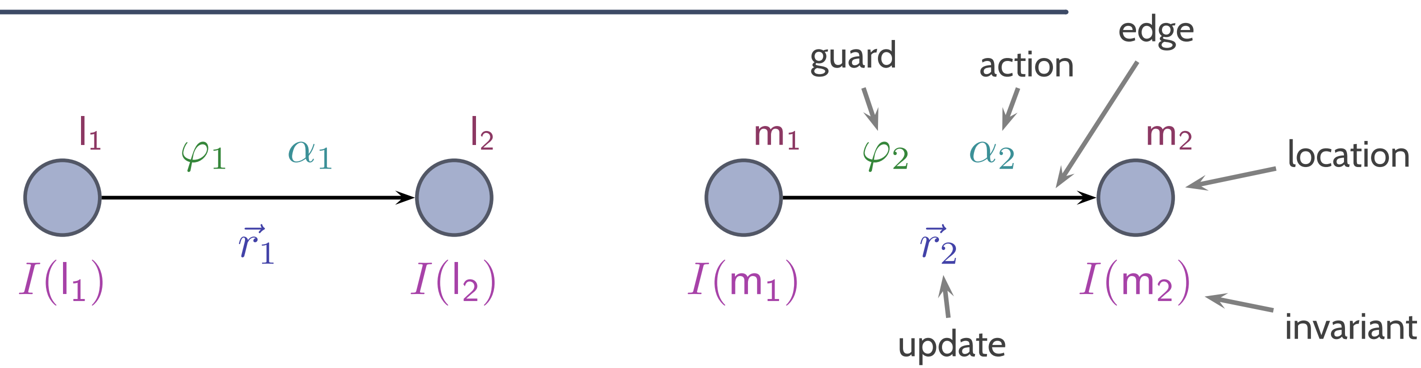
Example: (sender never blocked)



Another Example: (one of the senders blocked)



Recall: Operational Semantics of Networks of TA



location vector valuation

Operational semantics:
labelled transition relations $\xrightarrow{\lambda} \subseteq \mathit{Conf}(\mathcal{N}) \times \mathit{Conf}(\mathcal{N})$, $\quad \mathit{Conf}(\mathcal{N}) = \{\langle \vec{\ell}, \nu \rangle \mid \nu \models I(\vec{\ell})\}$.

- ***(delay transition)*** $\langle \vec{\ell}, \nu \rangle \xrightarrow{t} \langle \vec{\ell}, \nu + t \rangle$, $t \in \mathbb{R}_0^+$, if and only if $\forall t' \in [0, t] \bullet \nu + t' \models I(\vec{\ell})$.
- ***(local action transition)*** $\langle \vec{\ell}, \nu \rangle \xrightarrow{\tau} \langle \vec{\ell}', \nu' \rangle$, if and only if
  there is an edge $e = (\ell, \tau, \varphi, \vec{r}, \ell')$ in $\mathcal{A}_i$ such that $\langle \vec{\ell}, \nu \rangle \vdash_{loc} e$ and $\langle \vec{\ell}', \nu' \rangle = \langle \vec{\ell}, \nu \rangle[e]$
- ***(rendezvous transition)*** $\langle \vec{\ell}, \nu \rangle \xrightarrow{a} \langle \vec{\ell}', \nu' \rangle$, 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 $\langle \vec{\ell}, \nu \rangle \vdash_{loc} e_0$, 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 $\langle \vec{\ell}, \nu \rangle \vdash_{loc} e_1$,
  and $\langle \vec{\ell}', \nu' \rangle = \langle \vec{\ell}, \nu \rangle [e_0; e_1]$.

Characterising "Dependency on Global Scheduler"

Lemma. A closed component network $\mathcal{N}_{loc} = \{\mathcal{A}_1, \dots, \mathcal{A}_n\}$ **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.

$$\forall c \in \mathit{Conf}(\mathcal{N}_{loc})|_{reach}\ \forall 1 \leq i \leq n\ \forall a \in A\ \forall e \in E(\mathcal{A}_i)|_{a!} \bullet c \vdash_{loc} e$$
$$\implies (c \vdash e \wedge \forall 1 \leq j \leq n\ \forall b \in A\ \forall e' \in E(\mathcal{A}_j)|_{b!} \bullet c \vdash_{loc} e' \implies j = i).$$

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

Tell Them What You've Told Them...

Content

Introduction

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

$obs : \mathsf{Time} \to \mathscr{D}(obs)$

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

$\langle obs_0, \nu_0 \rangle, t_0 \xrightarrow{\lambda_0} \langle obs_1, \nu_1 \rangle, t_1 \ldots$

- **Automatic Verification**...

  ...whether a TA satisfies a DC formula, observer-based
- **Recent Results**:
  - ~~Timed Sequence Diagrams~~, or **Quasi-equal Clocks**, or **Automatic Code Generation**, or ...

Looking Back

– (D) (C) (B) (A)

perm. exam aids:
1 sheet of A4

- **Lect. 1**: **real-time system** (vs. hybrid), state variables (D)
- **Lect. 2**: **evolutions**, timing diagrams, (D) classes of timed properties (C)
- **Lect. 3**: DC **symbols**, state assertions, terms (syntax / semantics) (D)
- **Lect. 4**: DC **formulae**, abbreviations, satisfiable / realisable / valid (from 0) (D) C–A
- **Lect. 5**: semantics-based **correctness proof**; real-world obstacles B–A
- **Lect. 6**: DC calculus; **decidability** of RDC / discrete time
- **Lect. 7**: **undecidability** of RDC / continous time

  B-A
- **Lect. 8**: **DC Implementables**, standard forms, control automata (C)-A
- **Lect. 9**: **PLC**: characteristics, programming model (C)
- **Lect. 10**: ~~**PLC automata**, DC semantics~~
- **Lect. 11**: **timed automata** (syntax / semantics); tr. seq. / comp. path / run (D)/A
- **Lect. 12**: **parallel composition** of TA (syntactical / semantical); **Uppaal** (D) C–B
- **Lect. 13**: TA **location reachability**, time-abstract system, regions (C)
- **Lect. 14**: **zones**, zone-based reachability, Difference-Bounds-Matrices (B)-(A)
- **Lect. 15**: **Extended Timed Automata** (variables, urgent/committed) D-C
- **Lect. 16**: **query language**, evolutions vs. transition sequences D-C/A
- **Lect. 17**: **testability**, observer construction, untestable DC formulae C-B
- **Lect. 18**: undecidability results for **Timed Büchi Automata** ~~B~~ A
- **Lect. 19**: ~~**quasi-equal clocks**~~, bisimulation B-A
- **Lect. 20**: ~~formal methods for **RTS in practice**~~

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**
  - $\langle$ your (real-time) topic here $\rangle$
- **Student assistent jobs**
  - **programming**
  - **modelling**
- **Tutor jobs**
  - e.g., **Software Engineering** in Summer 2018

$\rightarrow$ **contact me**

* ADVERTISEMENTS *

* ADVERTISEMENTS *

ADVERTISEMENTS

ADVERTISEMENTS

ADVERTISEMENTS

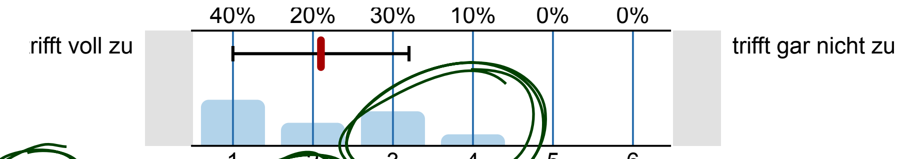
ADVERTISEMENTS

References

References

Olderog, E.-R. and Dierks, H. (2008). *Real-Time Systems - Formal Specification and Automatic Verification*. Cambridge University Press.

One Last Thing



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". ;-)