

# Applied Formal Methods

Peter Höfner

# Disclaimer

- this is not a normal scientific talk
  - introduces my research interests (partly)
  - hopefully inspires discussion (and cooperation)
  - advertises future talks (if there is interest)



# Formal Methods (FM)

In computer science ... formal methods are a particular kind of mathematically rigorous techniques for the specification, development and verification of software and hardware systems. (wikipedia)

# Formal Methods (FM)

In computer science ... formal methods are a particular kind of mathematically rigorous techniques for the specification, development and verification of software and hardware systems. (wikipedia)

- What is “Applied Formal Methods”

# Formal Methods (FM)

In computer science ... formal methods are a particular kind of mathematically rigorous techniques for the specification, development and verification of software and hardware systems. (wikipedia)

- What is “Applied Formal Methods”
- bridge the gap between FM and ‘real’ applications  
(over the years a lot of FM techniques were developed but not deployed)

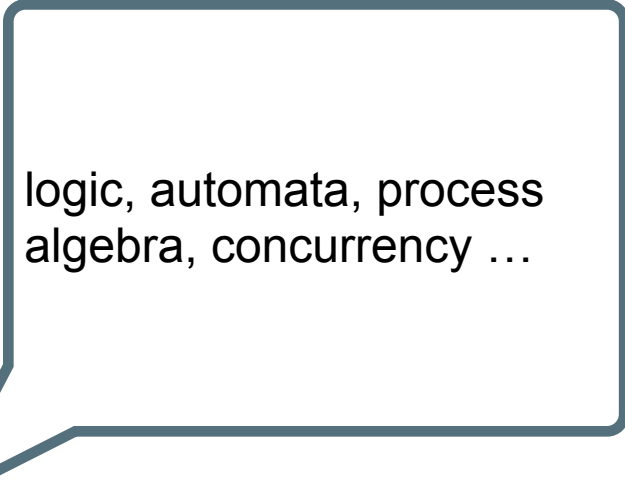


# Applied Formal Methods

- missing link between ‘theory’/FM and applications

# Applied Formal Methods

- missing link between ‘theory’/FM and applications



logic, automata, process algebra, concurrency ...

# Applied Formal Methods

- missing link between ‘theory’/FM and applications

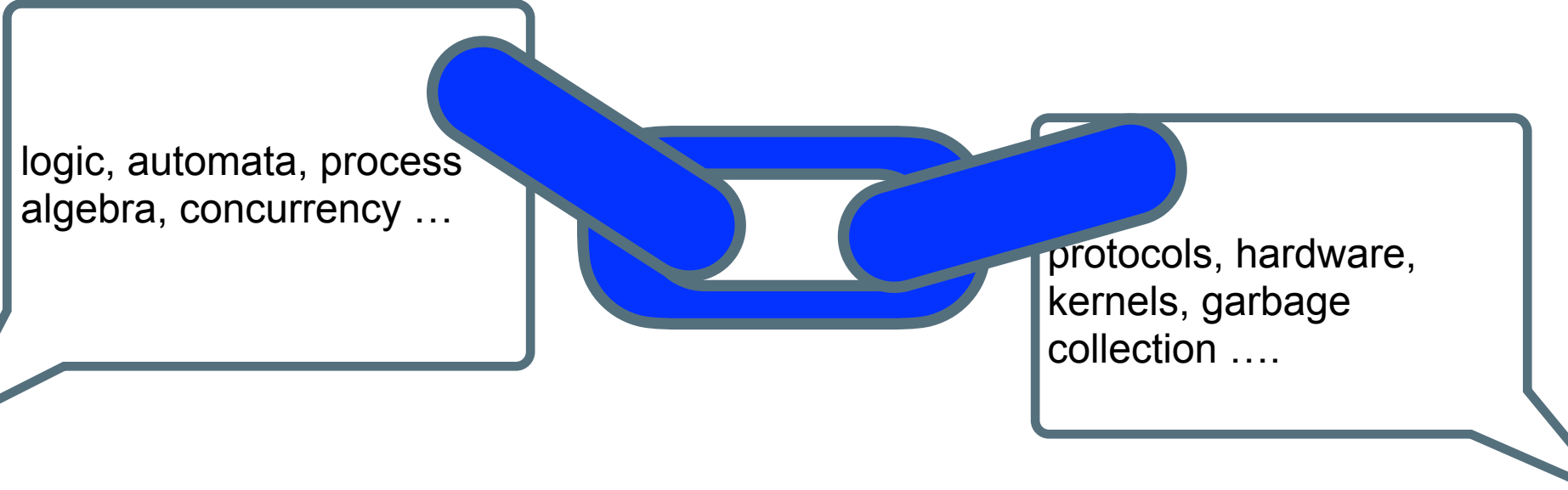
logic, automata, process algebra, concurrency ...

protocols, hardware, kernels, garbage collection ....



# Applied Formal Methods

- missing link between ‘theory’/FM and applications



logic, automata, process  
algebra, concurrency ...

protocols, hardware,  
kernels, garbage  
collection ....

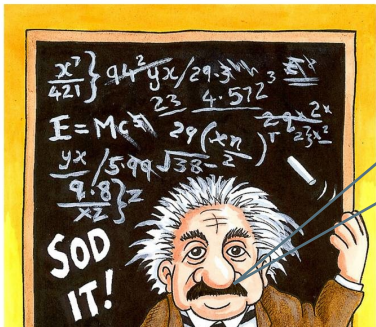
# Price to Pay

- sometimes it is extremely frustrating

# Price to Pay

- sometimes it is extremely frustrating

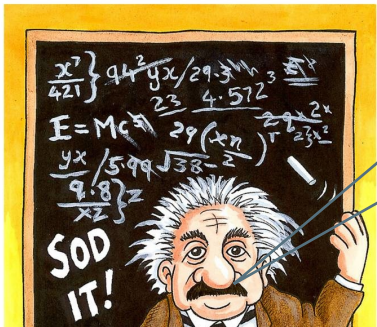
we know that X can  
be used for Y;  
it's **just** a big case  
study; no research!



# Price to Pay

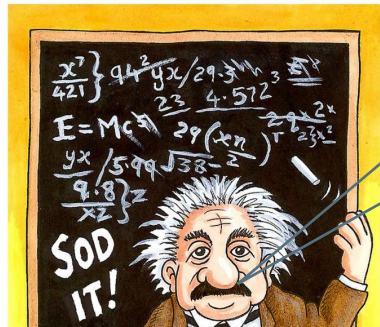
- sometimes it is extremely frustrating

we who cares? the program runs  
be most of the time  
it's (correct = we know;  
stu incorrect = not realistic)

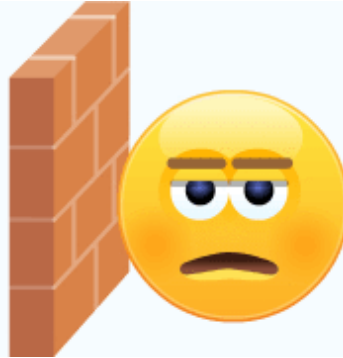


# Price to Pay

- sometimes it is extremely frustrating



we who  
be most  
it's (corr  
stu ince



am runs  
stic)



## ... however

- **rewarding** (at least at a personal level)  
German prof:  
“we do not need another researchers who sits in his office and proves yet another theorem” (that’s what I did in my Ph.D.)
- **challenging** (as it requires knowledge in multiple areas)
- **reveals shortcomings in FM**  
(scalability, missing foundations ...)
- **long-term impact** (hopefully)

# Part I: Protocol Analysis

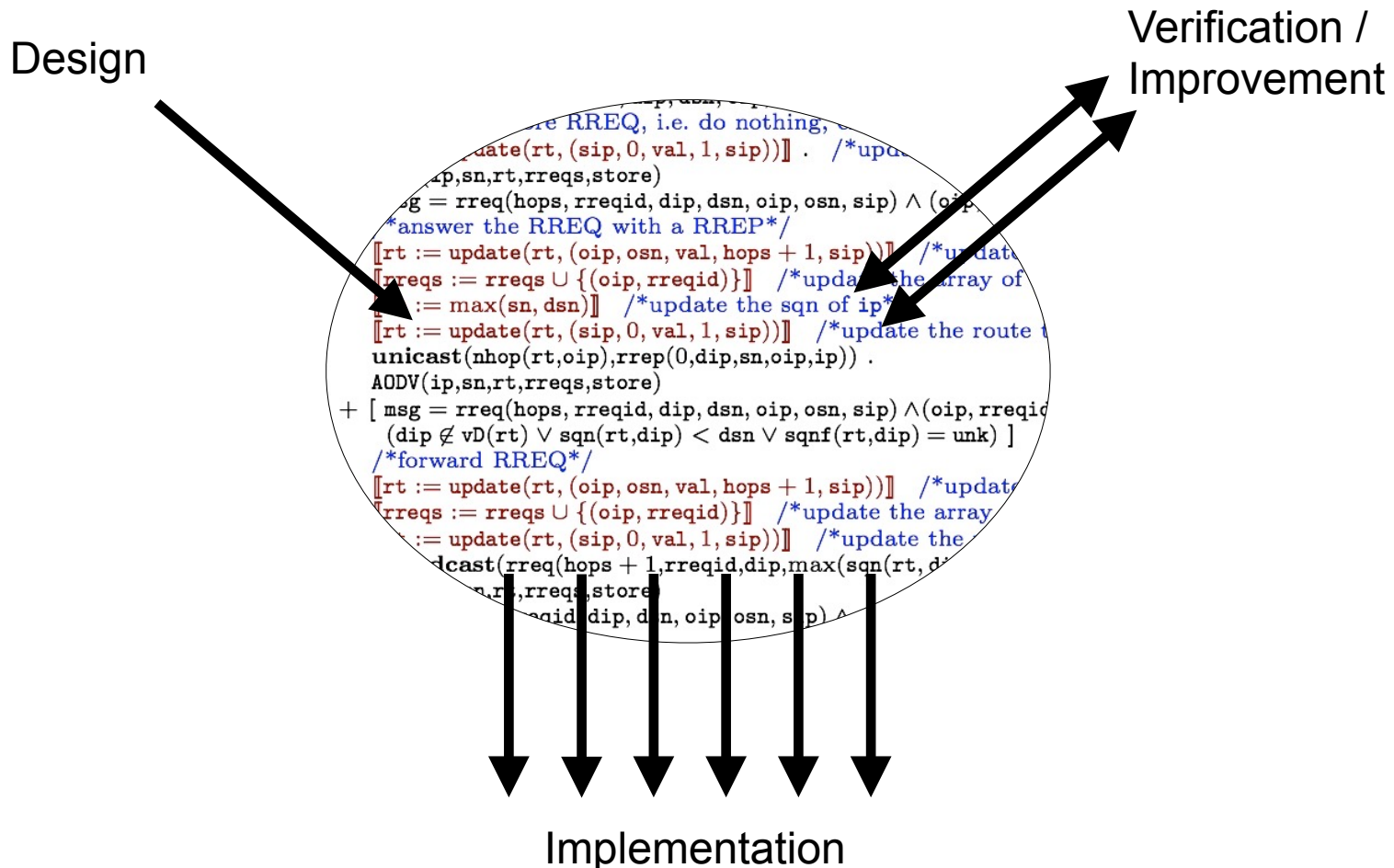
- modelling
- analysis
- verification

# Key Outcomes

- new process algebra (as it was required)
- model checking: quick check for counterexamples
- theorem proving: verification and proof automation
- case studies
  - AODV: complete and detailed model (including time)  
found short comings (AODV is not loop free)
  - OLSR & OSPF model completed (partly funded by DST)
  - communication protocols including CAN bus  
(funded by DARPA)
  - revealed a problem with verifying liveness (see below)



# Vision: Practical Protocol Engineering



# What's Next

- “standard” stuff
  - formally analyse OLSR & OSPF
  - improve tool support (Isabelle, Uppaal, mCRL2, ...)
- vulnerabilities
  - build attack models (DST + Alwen + Ph.D.)
  - analyse protocols (backwards reasoning?)
- comparison of Protocols
  - not sure how to do this; requires formal definitions
  - cooperate with Data61 and UQ
- from process algebra to real code
  - maybe PanCake (not CakeML)

# Possible Talks

- **Using Process Algebra to Design better Protocols**  
(extended version of the talk I gave during my interview)
- **AWN: A Process Algebra for Wireless Networks**
- How to formalise AWN in Isabelle/HOL
- **A Mechanized Proof of Loop Freedom of the (Untimed) AODV Routing Protocol.**
- From Process Algebra to Model Checking in a Correct Way  
(mCRL2 - AWN to come)
- Routing in Networks: details and difference of AODV, OSPF and OLSR
- **Statistical Model Checking of Wireless Mesh Routing Protocols**

## Part II: Multicore SeL4

- from problem analysis to product
- based (most likely) on Rely-Guarantee reasoning

# Key Outcomes (not by me)

- seL4:  
world-first formal machine-checked general-purpose OS kernel
- but it is single core
- eChronos:  
interruptible eChronos embedded operating system

# Vision: Verified “multicore SeL4”

# What's Next

- understand the fundamental problems (Data61, Michael)
  - is Rely-Guarantee Reasoning good enough
  - where is concurrency needed  
(kernel/kernel, kernel/user, user/user)
- language
  - COMPLEX vs Gammie/Hosking (Tony)
- build formally verified (Isabelle) concurrent data structures (DST+ Data61)
  - let's start with simple locks
- **it's only the beginning**

# Possible Talks

- basics on Rely-Guarantee Reasoning
- the foundations of COMPLEX  
(maybe invite Corey from Data61)



## Part III: Verifying Liveness Properties

- theoretical foundation
- when progress is too weak and fairness too strong

# Key Outcomes

- standard techniques, as used since the 80s, do not always work for verifying liveness of distributed systems.

“When Progress is too Weak and Fairness too Strong”

- a fair scheduler cannot be proven to be fair.
- proposed a replacement of Fairness, called *Justness*
- we believe it's the right level of abstraction

# Vision: Theoretical Sound Foundations for Verifying Liveness Properties in Distributed Systems

# What's Next

- theoretical sound definition
- replacement/refinement of standard concepts such as bisimulation (CRP with Data61 + Ph.D.)
- proof of concept
  - liveness of GC (Tony)
  - liveness of (multicore) seL4

# Possible Talks

- **Justness:**  
**when progress is too weak and fairness it too strong**
- Bisimulation does not work: what's a possible replacement (early ideas)

## Part IV: Program Algebras

- algebras for program logics
- algebras for program semantics
- algebras for simplifying verification tasks
- slightly orthogonal of the other topics  
(takes longer to have impact; but makes live neater)

# Key Outcomes

- Kleene algebra subsumes Hoare logic (Kozen)  
Forward/Backward reasoning is “chaining inequalities”
- quantale subsumes Separation Logic  
algebraic version of “frame calculation”
- algebra of rely-guarantee (Hayes et al.)
- mathematics of program construction (e.g. graph algorithms)

Vision:

Use algebraic reasoning to make  
verification easier/redundant



# What's Next

- apply current knowledge to real problems  
(graph algorithms, forward backward reasoning ...)
- can algebras for Hoare logic (separation logic) be combined with RG algebra and refinement

# Possible Talks

- **From High-School Math to Program Verification in 30 Minutes**
- Kleene Algebra and Hoare logic
- **Forwards and Backwards in Separation Algebra**
- **False Failure: Creating Failure Models for Separation Algebra**