“THEY DIDN’T KNOW THEY WERE DOING MATHEMATICS”

INTRODUCING FORMAL METHODS USING REWRITING LOGIC

Peter Ölveczky

University of Oslo

FMfun’19, December 3, 2019

OUTLINE

- How to introduce formal methods to undergraduates?
- Fun!
- Introducing formal methods using rewriting logic/Maude
- Experiences/feedback

OUTLINE

- How to introduce formal methods to undergraduates?
- Fun! 
- Introducing formal methods using rewriting logic/Maude
- Experiences/feedback

OUTLINE

- How to introduce formal methods to undergraduates?
- Fun!
- Introducing formal methods using rewriting logic/Maude
- Experiences/feedback

OUTLINE

- How to introduce formal methods to undergraduates?
- Fun!
- Introducing formal methods using rewriting logic/Maude
- Experiences/feedback

Challenges

Challenge

> In industry, formal methods have a reputation for requiring a huge amount of training and effort to verify a tiny piece of relatively straightforward code, so the return on investment is

How Amazon web Services Uses Formal Methods (Comm. ACM'15)

CHALLENGES (CONT.)

Challenge

Perception that formal methods only for safety-critical systems

Solution

1. Society increasingly dependent on safety-critical systems
   - self-driving cars, airplanes (737-MAX), power distribution, ...
2. Modern (cloud-based) computing "winner takes all"
   - ensure good quality also essential for non-critical systems
     - Gmail, facebook, etc data loss + availability
     - electronic contracts
     - ...

CHALLENGES (CONT.)

Challenge

Perception that formal methods only for safety-critical systems

Solution

1. Society increasingly dependent on safety-critical systems
   - self-driving cars, airplanes (737-MAX), power distribution, ...
2. Modern (cloud-based) computing "winner takes all"
   - ensure good quality also essential for non-critical systems
     - Gmail, facebook, etc data loss + availability
     - electronic contracts
     - ...

CHALLENGES (CONT.)

Challenge

Perception that formal methods only for safety-critical systems

Solution

1. Society increasingly dependent on safety-critical systems
   - self-driving cars, airplanes (737-MAX), power distribution, ...
2. Modern (cloud-based) computing "winner takes all"
   - ensure good quality also essential for non-critical systems
     - Gmail, facebook, etc data loss + availability
     - electronic contracts
     - ...

Global 500 2017

The most valuable brands of 2017

Brand	Rank 2017	Rank 2016	BV 2017	BV 2016	Change	Brand Rating
Google	1	2	$109,470m	$88,173m	+24%	AAA+
Apple	2	1	$107,141m	$145,918m	-27%	AAA
amazon.com	3	3	$106,369m	$69,642m	+53%	AAA-
AT&T	4	6	$87,016m	$59,904m	+45%	AAA
Microsoft	5	4	$76,265m	$ 67,258m	+13%	AAA
Samsung	6	7	$ 66,219m	$ 58,619m	+13%	AAA-
verizon	7	5	$ 65,875m	$ 63,116m	+4%	AAA-
Walmart	8	8	$ 62,496m	$ 53,657m	+16%	AA+
Facebook	9	17	$ 61,998m	$ 34,002m	+82%	AAA
ICBC	10	13	$ 47,832m	$ 36,334m	+32%	AAA

CHALLENGES (CONT.)

Challenge

- Worse and worse mathematical background
- Skeptic to mathematics

Solution

- Accessible/intuitive formal methods
- Cannot require [much/any] mathematical background

  $\longrightarrow$ no nontrivial mathematical prerequisites

CHALLENGES (CONT.)

Challenge

- Worse and worse mathematical background
- Skeptic to mathematics

Solution

- Accessible/intuitive formal methods
- Cannot require [much/any] mathematical background

  $\longrightarrow$ no nontrivial mathematical prerequisites

CHALLENGES (CONT.)

Challenge

- Worse and worse mathematical background
- Skeptic to mathematics

Solution

- Accessible/intuitive formal methods
- Cannot require [much/any] mathematical background

  $\longrightarrow$ no nontrivial mathematical prerequisites

CHALLENGES (CONT.)

Challenge

- Not part of core curriculum
- Students prefer “practical”/”job-related” courses

4. semester	IN2000 – Software Engineering med prosjektarbeid		IN2140 – Introduksjon til operativsystemer og datakommunikasjon /IN2080 – Beregninger og kompleksitet/IN2100 – Logikk for systemanalyse
3. semester	IN2010 – Algoritmer og datastrukturer	IN2120 – Informasjonssikkerhet -	IN2090 – Databaser og datamodellering
2. semester	IN1010 – Objektorientert programmering	IN1030 – Systemer, krav og konsekvenser	IN1150 – Logiske metoder

Solution

CHALLENGES (CONT.)

Challenge

- Not part of core curriculum
- Students prefer "practical"/"job-related" courses

Solution

- Show relevance/usefulness early
  $\longrightarrow$ nontrivial problems/examples/applications
- Make it fun over years
- ???

CHALLENGES (CONT.)

Challenge

- Not part of core curriculum
- Students prefer "practical"/"job-related" courses

Solution

- Show relevance/usefulness early
  $\longrightarrow$ nontrivial problems/examples/applications
- Make it fun over years
- ???

CHALLENGES (CONT.)

Challenge

FM teaching not integrated with other courses

Solution

Model/analyze systems encountered in other courses

- security (protocols?)
- networking/communication
- databases/distributed transactions
- operating systems
- ...

CHALLENGES (CONT.)

Challenge

FM teaching not integrated with other courses

Solution

Model/analyze systems encountered in other courses

- security (protocols?)
- networking/communication
- databases/distributed transactions
- operating systems
- ...

Challenge

Relevant problems not addressed

Abstract. Courses on formal methods are often based on examples and case studies, supposed to show students how to apply formal methods in practice. However, examples often fall into one of two categories: First, many are constructed and thus do not relate to practice. Second, examples are based on projects of industry partners and are, thus, way too involved for students to understand them.

Solution

- Address problems which look relevant
  - social media
  - online shopping
  - cloud applications (Gmail, eBay, facebook, ...)
  - authentication

$\longrightarrow$ need expressive formalism

Challenge

Relevant problems not addressed

Solution

- Address problems which look relevant
  - social media
  - online shopping
  - cloud applications (Gmail, eBay, facebook, ...)
  - authentication

⟶ need expressive formalism

Challenge

Relevant problems not addressed

Solution

- Address problems which look relevant
  - social media
  - online shopping
  - cloud applications (Gmail, eBay, facebook, ...)
  - authentication

$\longrightarrow$ need expressive formalism

Fun! 

MAKING FORMAL METHODS FUN

- Why did you [continue] study CS?
  - programming!
- What programming did you enjoy?
  - Java/Pascal imperative programming
  - assembly
  - C
  - functional programming (LISP, ...)
  - logic programming
  - . . .

MAKING FORMAL METHODS FUN

- Why did you [continue] study CS?
  - programming!
- What programming did you enjoy?
  - Java/Pascal imperative programming
  - assembly
  - C
  - functional programming (LISP, ...)
  - logic programming
  - ...

MAKING FORMAL METHODS FUN

- Why did you [continue] study CS?
    - programming!
- What programming did you enjoy?
    - Java/Pascal imperative programming
    - assembly
    - C
    - functional programming (LISP, ...)
    - logic programming
    - ...

MAKING FORMAL METHODS FUN (CONT.)

- Interesting/relevant applications
    - card tricks, Pac-Man, chess?
    - relevant in industry
    - security?
- Nice/mature tool(s)
- Fun programming
    - no hacking/horrible encodings

MAKING FORMAL METHODS FUN (CONT.)

- Interesting/relevant applications
  - card tricks, Pac-Man, chess?
  - relevant in industry
  - security?
- Nice/mature tool(s)
- Fun programming
  - no hacking/horrible encodings

MAKING FORMAL METHODS FUN (CONT.)

- Interesting/relevant applications
  - card tricks, Pac-Man, chess?
  - relevant in industry
  - security?
- Nice/mature tool(s)
- Fun programming
  - no hacking/horrible encodings

MAKING FORMAL METHODS FUN (CONT.)

- Interesting/relevant applications
  - card tricks, Pac-Man, chess?
  - relevant in industry
  - security?
- Nice/mature tool(s)
- Fun programming
  - no hacking/horrible encodings

MAKING FORMAL METHODS FUN (CONT.)

- Interesting/relevant applications
  - card tricks, Pac-Man, chess?
  - relevant in industry
  - security?
- Nice/mature tool(s)
- Fun programming
  - no hacking/horrible encodings

What Should a Formal Methods Course Look Like?

Teaching Formal Methods in the Context of Software Engineering

Shaoying Liu
Department of Computer Science
Faculty of Computer and Information Sciences
Hosei University
3-7-2 Kajino-cho Koganei-shi
Tokyo, 184-8584, Japan
sliu@hosei.ac.jp

Kazuhiro Takahashi
Research Center
The Nippon Signal Co., Ltd.
1836-1 Oaza Ezura, Kuki,
Saitama 346-8524, Japan
ktaka@signal.co.jp

Toshinori Hayashi
Research Center
The Nippon Signal Co., Ltd.
1836-1 Oaza Ezura, Kuki,
Saitama 346-8524, Japan
hayashit@signal.co.jp

Toshihiro Nakayama
Research Center
The Nippon Signal Co., Ltd.
1836-1 Oaza Ezura, Kuki,
Saitama 346-8524, Japan
nkym-ts@signal.co.jp

Teaching Formal Methods in the Context of Software Engineering

Shaoying Liu
Department of Computer Science
Faculty of Computer and Information Sciences
Hosei University
3-7-2 Kajino-cho Koganei-shi
Tokyo, 184-8584, Japan
sliu@hosei.ac.jp

Kazuhiro Takahashi
Research Center
The Nippon Signal Co., Ltd.
1836-1 Oaza Ezura, Kuki,
Saitama 346-8524, Japan
ktaka@signal.co.jp

Toshinori Hayashi
Research Center
The Nippon Signal Co., Ltd.
1836-1 Oaza Ezura, Kuki,
Saitama 346-8524, Japan
hayashit@signal.co.jp

Toshihiro Nakayama
Research Center
The Nippon Signal Co., Ltd.
1836-1 Oaza Ezura, Kuki,
Saitama 346-8524, Japan
nkym-ts@signal.co.jp

- VDM, refinement, Hoare logic ...
- "However, none of these techniques is easy to use by ordinary practitioners to deal with real software projects."
- "most effective for students [...] is to write formal specs by hand, as they learn English as a foreign language."
- "our experience suggests that each course should not be too ambitious; instead, it should be focused"
- "there is little hope to apply the refinement calculus in practice"

Teaching Formal Methods in the Context of Software Engineering

Shaoying Liu
Department of Computer Science
Faculty of Computer and Information Sciences
Hosei University
3-7-2 Kajino-cho Koganei-shi
Tokyo, 184-8584, Japan
sliu@hosei.ac.jp

Kazuhiro Takahashi
Research Center
The Nippon Signal Co., Ltd.
1836-1 Oaza Ezura, Kuki,
Saitama 346-8524, Japan
ktaka@signal.co.jp

Toshinori Hayashi
Research Center
The Nippon Signal Co., Ltd.
1836-1 Oaza Ezura, Kuki,
Saitama 346-8524, Japan
hayashit@signal.co.jp

Toshihiro Nakayama
Research Center
The Nippon Signal Co., Ltd.
1836-1 Oaza Ezura, Kuki,
Saitama 346-8524, Japan
nkym-ts@signal.co.jp

- VDM, refinement, Hoare logic ...
- "However, none of these techniques is easy to use by ordinary practitioners to deal with real software projects."
- "most effective for students [...] is to write formal specs by hand, as they learn English as a foreign language."
- "our experience suggests that each course should not be too ambitious; instead, it should be focused"
- "there is little hope to apply the refinement calculus in practice"

Teaching Formal Methods in the Context of Software Engineering

Shaoying Liu
Department of Computer Science
Faculty of Computer and Information Sciences
Hosei University
3-7-2 Kajino-cho Koganei-shi
Tokyo, 184-8584, Japan
sliu@hosei.ac.jp

Kazuhiro Takahashi
Research Center
The Nippon Signal Co., Ltd.
1836-1 Oaza Ezura, Kuki,
Saitama 346-8524, Japan
ktaka@signal.co.jp

Toshinori Hayashi
Research Center
The Nippon Signal Co., Ltd.
1836-1 Oaza Ezura, Kuki,
Saitama 346-8524, Japan
hayashit@signal.co.jp

Toshihiro Nakayama
Research Center
The Nippon Signal Co., Ltd.
1836-1 Oaza Ezura, Kuki,
Saitama 346-8524, Japan
nkym-ts@signal.co.jp

- VDM, refinement, Hoare logic ...
- "However, none of these techniques is easy to use by ordinary practitioners to deal with real software projects."
- "most effective for students [...] is to write formal specs by hand, as they learn English as a foreign language."
- "our experience suggests that each course should not be too ambitious; instead, it should be focused"
- "there is little hope to apply the refinement calculus in practice"

Teaching Formal Methods in the Context of Software Engineering

Shaoying Liu
Department of Computer Science
Faculty of Computer and Information Sciences
Hosei University
3-7-2 Kajino-cho Koganei-shi
Tokyo, 184-8584, Japan
sliu@hosei.ac.jp

Kazuhiro Takahashi
Research Center
The Nippon Signal Co., Ltd.
1836-1 Oaza Ezura, Kuki,
Saitama 346-8524, Japan
ktaka@signal.co.jp

Toshinori Hayashi
Research Center
The Nippon Signal Co., Ltd.
1836-1 Oaza Ezura, Kuki,
Saitama 346-8524, Japan
hayashit@signal.co.jp

Toshihiro Nakayama
Research Center
The Nippon Signal Co., Ltd.
1836-1 Oaza Ezura, Kuki,
Saitama 346-8524, Japan
nkym-ts@signal.co.jp

- VDM, refinement, Hoare logic ...
- "However, none of these techniques is easy to use by ordinary practitioners to deal with real software projects."
- "most effective for students [...] is to write formal specs by hand, as they learn English as a foreign language."
- "our experience suggests that each course should not be too ambitious; instead, it should be focused"
- "there is little hope to apply the refinement calculus in practice"

Teaching Formal Methods in the Context of Software Engineering

Shaoying Liu
Department of Computer Science
Faculty of Computer and Information Sciences
Hosei University
3-7-2 Kajino-cho Koganei-shi
Tokyo, 184-8584, Japan
sliu@hosei.ac.jp

Kazuhiro Takahashi
Research Center
The Nippon Signal Co., Ltd.
1836-1 Oaza Ezura, Kuki,
Saitama 346-8524, Japan
ktaka@signal.co.jp

Toshinori Hayashi
Research Center
The Nippon Signal Co., Ltd.
1836-1 Oaza Ezura, Kuki,
Saitama 346-8524, Japan
hayashit@signal.co.jp

Toshihiro Nakayama
Research Center
The Nippon Signal Co., Ltd.
1836-1 Oaza Ezura, Kuki,
Saitama 346-8524, Japan
nkym-ts@signal.co.jp

- VDM, refinement, Hoare logic ...
- "However, none of these techniques is easy to use by ordinary practitioners to deal with real software projects."
- "most effective for students [...] is to write formal specs by hand, as they learn English as a foreign language."
- "our experience suggests that each course should not be too ambitious; instead, it should be focused"
- "there is little hope to apply the refinement calculus in practice"

Teaching Formal Methods for Software Engineering – Ten Principles

Antonio Cerone[1], Markus Roggenbach[2], Bernd-Holger Schlingloff[3], Gerardo Schneider[4], and Siraj Ahmed Shaikh[5]

1. Formal Methods too large to gain encyclopaedic knowledge
   $\longrightarrow$ choose representatives
   - "loads to gain by intensively studying selected few methods"
2. Formal Methods more than pure/poor Mathematics $\longrightarrow$ focus on Engineering
3. Formal Methods need tools
   - "Tools for simulation and visualization [...] essential"
4. Modelling versus programming: work out the differences
5. Tools teach the method: use them

Teaching Formal Methods for Software Engineering – Ten Principles

Antonio Cerone[1], Markus Roggenbach[2], Bernd-Holger Schlingloff[3], Gerardo Schneider[4], and Siraj Ahmed Shaikh[5]

1. Formal Methods too large to gain encyclopaedic knowledge $\longrightarrow$ choose representatives
   - "loads to gain by intensively studying selected few methods"
2. Formal Methods more than pure/poor Mathematics $\longrightarrow$ focus on Engineering
3. Formal Methods need tools
   - "Tools for simulation and visualization [...] essential"
4. Modelling versus programming: work out the differences
5. Tools teach the method: use them

Teaching Formal Methods for Software Engineering – Ten Principles

Antonio Cerone[1], Markus Roggenbach[2], Bernd-Holger Schlingloff[3], Gerardo Schneider[4], and Siraj Ahmed Shaikh[5]

1. Formal Methods too large to gain encyclopaedic knowledge $\longrightarrow$ choose representatives
   - "loads to gain by intensively studying selected few methods"
2. Formal Methods more than pure/poor Mathematics $\longrightarrow$ focus on Engineering
3. Formal Methods need tools
   - "Tools for simulation and visualization [...] essential"
4. Modelling versus programming: work out the differences
5. Tools teach the method: use them

Teaching Formal Methods for Software Engineering – Ten Principles

Antonio Cerone[1], Markus Roggenbach[2], Bernd-Holger Schlingloff[3], Gerardo Schneider[4], and Siraj Ahmed Shaikh[5]

6. Formal Methods need lab classes ⟶ stable platform
7. Formal Methods best taught by examples
8. Each Formal Method consists of syntax, semantics and algorithms
9. Formal Methods have several dimensions ⟶ use a taxonomy
10. Formal Methods are fun ⟶ shout it out loud!
    - “human learning capacity is highest when we enjoy what we are doing”

Teaching Formal Methods for Software Engineering – Ten Principles

Antonio Cerone[1], Markus Roggenbach[2], Bernd-Holger Schlingloff[3], Gerardo Schneider[4], and Siraj Ahmed Shaikh[5]

6. Formal Methods need lab classes ⟶ stable platform
7. Formal Methods best taught by examples
8. Each Formal Method consists of syntax, semantics and algorithms
9. Formal Methods have several dimensions ⟶ use a taxonomy
10. Formal Methods are fun ⟶ shout it out loud!
    - "human learning capacity is highest when we enjoy what we are doing"

Teaching Formal Methods for Software Engineering – Ten Principles

Antonio Cerone[1], Markus Roggenbach[2], Bernd-Holger Schlingloff[3], Gerardo Schneider[4], and Siraj Ahmed Shaikh[5]

6. Formal Methods need lab classes $\longrightarrow$ stable platform
7. Formal Methods best taught by examples
8. Each Formal Method consists of syntax, semantics and algorithms
9. Formal Methods have several dimensions $\longrightarrow$ use a taxonomy
10. Formal Methods are fun $\longrightarrow$ shout it out loud!
    - "human learning capacity is highest when we enjoy what we are doing"

WHAT TO TEACH IN INTRODUCTORY FM COURSE?

University study:

⟶ teach concepts

- not formalism/tool/logic

- avoid many tools

WHAT TO TEACH IN INTRODUCTORY FM COURSE?

University study:

$\longrightarrow$ teach concepts

- not formalism/tool/logic

- avoid many tools

WHAT TO TEACH IN INTRODUCTORY FM COURSE? (CONT.)

What are key FM concepts?

- mathematical modeling
    - of systems/designs
    - of requirements
- reasoning about models
    - automatic model checking
    - verification
    - performance
- mathematical analysis of program/code

WHAT TO TEACH IN INTRODUCTORY FM COURSE? (CONT.)

What are key FM concepts?

- mathematical modeling
  - of systems/designs
  - of requirements
- reasoning about models
  - automatic model checking
  - verification
  - performance
- mathematical analysis of program/code

WHAT TO TEACH IN INTRODUCTORY FM COURSE? (CONT.)

What are key FM concepts?

- mathematical modeling
  - of systems/designs
  - of requirements
- reasoning about models
  - automatic model checking
  - verification
  - performance
- mathematical analysis of program/code

WHAT TO TEACH IN INTRODUCTORY FM COURSE? (CONT.)

What are key FM concepts?

- mathematical modeling
  - of systems/designs
  - of requirements
- reasoning about models
  - automatic model checking
  - verification
  - performance
- mathematical analysis of program/code

WHAT TO TEACH IN INTRODUCTORY FM COURSE? (CONT.)

Logical reasoning in general

- logics
- deduction
- satisfiability, . . .
- theoretical results/folklore
    - undecidability results
    - notions need for “FM literacy”
    - . . .

Cannot cover too much!

WHAT TO TEACH IN INTRODUCTORY FM COURSE? (CONT.)

Logical reasoning in general

- logics
- deduction
- satisfiability, . . .
- theoretical results/folklore
    - undecidability results
    - notions need for "FM literacy"
    - . . .

Cannot cover too much!

SUMMARIZING REQUIREMENTS

1. Fun(ctional) modeling/programming!
2. Relevant applications/examples
   - related to other courses
   - relevant problems
   - security
3. Few (mature) tools/formalisms
   - industry-relevant
4. Motivate with industrial success!
5. Introduce key FM concepts
   - formal modeling (designs; requirements)
   - analysis
     - model checking and verification
     - (correctness and performance)
   - verification of programs
   - logic; models; deduction; folklore results

SUMMARIZING REQUIREMENTS

1. Fun(ctional) modeling/programming!
2. Relevant applications/examples
   - related to other courses
   - relevant problems
   - security
3. Few (mature) tools/formalisms
   - industry-relevant
4. Motivate with industrial success!
5. Introduce key FM concepts
   - formal modeling (designs; requirements)
   - analysis
     - model checking and verification
     - (correctness and performance)
   - verification of programs
   - logic; models; deduction; folklore results

SUMMARIZING REQUIREMENTS

1. Fun(ctional) modeling/programming!
2. Relevant applications/examples
   - related to other courses
   - relevant problems
   - security
3. Few (mature) tools/formalisms
   - industry-relevant
4. Motivate with industrial success!
5. Introduce key FM concepts
   - formal modeling (designs; requirements)
   - analysis
     - model checking and verification
     - (correctness and performance)
   - verification of programs
   - logic; models; deduction; folklore results

SUMMARIZING REQUIREMENTS

1. Fun(ctional) modeling/programming!
2. Relevant applications/examples
   - related to other courses
   - relevant problems
   - security
3. Few (mature) tools/formalisms
   - industry-relevant
4. Motivate with industrial success!
5. Introduce key FM concepts
   - formal modeling (designs; requirements)
   - analysis
     - model checking and verification
     - (correctness and performance)
   - verification of programs
   - logic; models; deduction; folklore results

SUMMARIZING REQUIREMENTS

1. Fun(ctional) modeling/programming!
2. Relevant applications/examples
   - related to other courses
   - relevant problems
   - security
3. Few (mature) tools/formalisms
   - industry-relevant
4. Motivate with industrial success!
5. Introduce key FM concepts
   - formal modeling (designs; requirements)
   - analysis
     - model checking and verification
     - (correctness and performance)
   - verification of programs
   - logic; models; deduction; folklore results

SUMMARIZING REQUIREMENTS

1. Fun(ctional) modeling/programming!
2. Relevant applications/examples
   - related to other courses
   - relevant problems
   - security
3. Few (mature) tools/formalisms
   - industry-relevant
4. Motivate with industrial success!
5. Introduce key FM concepts
   - formal modeling (designs; requirements)
   - analysis
     - model checking and verification
     - (correctness and performance)
   - verification of programs
   - logic; models; deduction; folklore results

SUMMARIZING REQUIREMENTS

1. Fun(ctional) modeling/programming!
2. Relevant applications/examples
   - related to other courses
   - relevant problems
   - security
3. Few (mature) tools/formalisms
   - industry-relevant
4. Motivate with industrial success!
5. Introduce key FM concepts
   - formal modeling (designs; requirements)
   - analysis
     - model checking and verification
     - (correctness and performance)
   - verification of programs
   - logic; models; deduction; folklore results

IT FOLLOWS THAT ...

- expressive formalism
  - yet simple and intuitive
  - not much/any math prerequisite
- executable formalism
- industrial relevance

IT FOLLOWS THAT ...

- expressive formalism
  - yet simple and intuitive
  - not much/any math prerequisite
- executable formalism
- industrial relevance

IT FOLLOWS THAT ...

- expressive formalism
  - yet simple and intuitive
  - not much/any math prerequisite
- executable formalism
- industrial relevance

Introducing Formal Methods using Rewriting Logic

COURSE OVERVIEW

- One semester course (90 min lecture pr week; 15 weeks)
- Second-year course
  - previously years 3-5
- No prerequisites!
  - students may know basic logic
- Norwegian students don't study (much)

COURSE OVERVIEW

- One semester course (90 min lecture pr week; 15 weeks)
- Second-year course
  - previously years 3-5
- No prerequisites!
  - students may know basic logic
- Norwegian students don't study (much)

COURSE OVERVIEW

- One semester course (90 min lecture pr week; 15 weeks)
- Second-year course
  - previously years 3-5
- No prerequisites!
  - students may know basic logic
- Norwegian students don't study (much)

REWRITING LOGIC

Rewriting logic [Meseguer 1990-]

- expressive and intuitive logic
- executable
- fun(ctional) programming style
- mature tool: Maude

APPLICATIONS OF MAUDE

- Security
  - found unknown address bar and status bar spoof attacks in web browsers
  - Maude-NPA: Cathy Meadows NLA Protocol Analyzer
- Semantics for programming languages
  - C, Java, JVM, Scheme, EVM, ...
  - K framework (G. Rosu)
  - errors in electronic contracts on blockchain
- Semantics for modeling languages and frameworks (MOF, ...)
- Logical framework
  - automatically translate HOL libraries to NuPrl
- Network protocols and cloud computing
  - Apache Cassandra, Google's Megastore, ZooKeeper, ...
- Biological systems
  - cell biology (Pathway logic)
  - brains (Alzheimer, Parkinson, ...)

APPLICATIONS OF MAUDE

- Security
  - found unknown address bar and status bar spoof attacks in web browsers
  - Maude-NPA: Cathy Meadows NLA Protocol Analyzer
- Semantics for programming languages
  - C, Java, JVM, Scheme, EVM, ...
  - K framework (G. Rosu)
  - errors in electronic contracts on blockchain
- Semantics for modeling languages and frameworks (MOF, ...)
- Logical framework
  - automatically translate HOL libraries to NuPrl
- Network protocols and cloud computing
  - Apache Cassandra, Google's Megastore, ZooKeeper, ...
- Biological systems
  - cell biology (Pathway logic)
  - brains (Alzheimer, Parkinson, ...)

APPLICATIONS OF MAUDE

- Security
  - found unknown address bar and status bar spoof attacks in web browsers
  - Maude-NPA: Cathy Meadows NLA Protocol Analyzer
- Semantics for programming languages
  - C, Java, JVM, Scheme, EVM, ...
  - K framework (G. Rosu)
  - errors in electronic contracts on blockchain
- Semantics for modeling languages and frameworks (MOF, ...)
- Logical framework
  - automatically translate HOL libraries to NuPrl
- Network protocols and cloud computing
  - Apache Cassandra, Google's Megastore, ZooKeeper, ...
- Biological systems
  - cell biology (Pathway logic)
  - brains (Alzheimer, Parkinson, ...)

APPLICATIONS OF MAUDE

- Security
  - found unknown address bar and status bar spoof attacks in web browsers
  - Maude-NPA: Cathy Meadows NLA Protocol Analyzer
- Semantics for programming languages
  - C, Java, JVM, Scheme, EVM, ...
  - K framework (G. Rosu)
  - errors in electronic contracts on blockchain
- Semantics for modeling languages and frameworks (MOF, ...)
- Logical framework
  - automatically translate HOL libraries to NuPrl
- Network protocols and cloud computing
  - Apache Cassandra, Google's Megastore, ZooKeeper, ...
- Biological systems
  - cell biology (Pathway logic)
  - brains (Alzheimer, Parkinson, ...)

APPLICATIONS OF MAUDE

- Security
  - found unknown address bar and status bar spoof attacks in web browsers
  - Maude-NPA: Cathy Meadows NLA Protocol Analyzer
- Semantics for programming languages
  - C, Java, JVM, Scheme, EVM, ...
  - K framework (G. Rosu)
  - errors in electronic contracts on blockchain
- Semantics for modeling languages and frameworks (MOF, ...)
- Logical framework
  - automatically translate HOL libraries to NuPrl
- Network protocols and cloud computing
  - Apache Cassandra, Google's Megastore, ZooKeeper, ...
- Biological systems
  - cell biology (Pathway logic)
  - brains (Alzheimer, Parkinson, ...)

REWRITING LOGIC (CONT.)

Data types/functions: algebraic equational specifications

```
fmod NAT-ADD is
  sort Nat .
  op 0 : -> Nat [ctor] .
  op s : Nat -> Nat [ctor] .
  op _+_ : Nat Nat -> Nat .

  vars M N : Nat .

  eq 0 + M = M .
  eq s(M) + N = s(M + N) .
endfm
```

LISTS

```
sorts List NeList .
subsorts Nat < NeList < List .
op nil : -> List [ctor] .
op __ : List List -> List [assoc id: nil ctor] .
op __ : NeList NeList -> NeList [assoc id: nil ctor] .

op length : List -> Nat .
ops first last : NeList -> Nat .
op rest : NeList -> List .

vars M N : Nat .   var L : List .

eq length(nil) = 0 .
eq length(N L) = 1 + length(L) .

eq last(L N) = N .        eq rest(N L) = L .
```

LISTS

```
sorts List NeList .
subsorts Nat < NeList < List .
op nil : -> List [ctor] .
op __ : List List -> List [assoc id: nil ctor] .
op __ : NeList NeList -> NeList [assoc id: nil ctor] .

op length : List -> Nat .
ops first last : NeList -> Nat .
op rest : NeList -> List .

vars M N : Nat .   var L : List .

eq length(nil) = 0 .
eq length(N L) = 1 + length(L) .

eq last(L N) = N .      eq rest(N L) = L .
```

QUICKSORT IN MAUDE

```
op quicksort : List -> List .

vars L L’ : List .
vars M N : Int .

eq quicksort(nil) = nil .
eq quicksort(L N L’) =
              quicksort(smallerElements(L L’, N))
              equalElements(L N L’, N)
              quicksort(greaterElements(L L’, N)) .
```

QUICKSORT: AUXILIARY FUNCTIONS

```
ops smallerElements greaterElements
    equalElements : List Int -> List .

eq smallerElements(nil, N) = nil .
eq smallerElements(N L, M) =
      if N < M then (N smallerElements(L, M))
      else smallerElements(L, M) fi .

eq equalElements(nil, N) = nil .
eq equalElements(N L, M) =
      if N == M then (N equalElements(L, M))
      else equalElements(L, M) fi .

eq greaterElements(nil, N) = nil .
eq greaterElements(N L, M) =
      if N > M then (N greaterElements(L, M))
      else greaterElements(L, M) fi .
```

```
fmod MERGE-SORT is protecting LIST-INT .
  op mergeSort : List -> List .
  op merge : List List -> List [comm] .

  vars L L' : List .
  vars NEL NEL' : NeList .
  vars I J : Int .

  eq mergeSort(nil) = nil .
  eq mergeSort(I) = I .

  ceq mergeSort(NEL NEL') =
          merge(mergeSort(NEL), mergeSort(NEL'))
      if length(NEL) == length(NEL')
         or  length(NEL) == s length(NEL') .

  eq merge(nil, L) = L .
  ceq merge(I L, J L') = I merge(L, J L') if I <= J .
endfm
```

```
fmod MERGE-SORT is protecting LIST-INT .
  op mergeSort : List -> List .
  op merge : List List -> List [comm] .

  vars L L' : List .
  vars NEL NEL' : NeList .
  vars I J : Int .

  eq mergeSort(nil) = nil .
  eq mergeSort(I) = I .

  ceq mergeSort(NEL NEL') =
        merge(mergeSort(NEL), mergeSort(NEL'))
    if length(NEL) == length(NEL')
       or  length(NEL) == s length(NEL') .

  eq merge(nil, L) = L .
  ceq merge(I L, J L') = I merge(L, J L') if I <= J .
endfm
```

```
sort MSet .   subsort NzNat < MSet .
op none : -> MSet [ctor] .
op _ _ : MSet MSet -> MSet [ctor assoc comm id: none] .

op subsetSum : MSet NzNat -> Bool .

vars NZN NZN1 NZN2 : NzNat .     var S : MSet .

eq subsetSum(none, NZN) = false .
eq subsetSum(NZN S, NZN) = true .

ceq subsetSum(NZN1 S, NZN2)
  = subsetSum(S, NZN2 - NZN1)   --- pick element NZN1
    or subsetSum(S, NZN2)       --- or don't
  if NZN2 > NZN1 .

ceq subsetSum(NZN1 S, NZN2) = subsetSum(S, NZN2)
      if NZN1 > NZN2 .    --- cannot pick element NZN1
```

EQUATIONAL SPECIFICATIONS

- Fun programming
- Term rewrite theory (termination; confluence; ...)
  - expressiveness of term + confluent specs
  - undecidability of termination, confluence, ...
- Equational logic
  - completeness—incompleteness
- Models (algebra)
- Inductive theorems

DYNAMIC BEHAVIORS

- Dynamic behaviors modeled by rewrite rules
  - not terminating/confluent
- Maude analysis:
  - simulation
  - explicit-state reachability analysis
  - LTL model checking

DYNAMIC BEHAVIORS

- Dynamic behaviors modeled by rewrite rules
  - not terminating/confluent
- Maude analysis:
  - simulation
  - explicit-state reachability analysis
  - LTL model checking

EXAMPLE: SOCCER

```
mod GAME is protecting NAT .
  protecting STRING .
  sort Game .

  op _-_ _:_ : String String Nat Nat -> Game [ctor] .

  vars HOME AWAY : String .
  vars M N : Nat .

  rl [home-goal] :
     HOME - AWAY M : N  =>  HOME - AWAY M + 1 : N .

  rl [away-goal] :
     HOME - AWAY M : N  =>  HOME - AWAY M : N + 1 .
endm
```

SIMULATING A SOCCER GAME

```
Maude> rew [3] "Malmö FF" - "Nottingham Forest" 0 : 0 .

result Game: "Malmö FF" - "Nottingham Forest" 2 : 1

Maude> rew [5] "Italy" - "Brazil" 0 : 0 .
result Game: "Italy" - "Brazil" 3 : 2
```

SIMULATING A SOCCER GAME

```
Maude> rew [3] "Malmö FF" - "Nottingham Forest" 0 : 0 .

result Game: "Malmö FF" - "Nottingham Forest" 2 : 1

Maude> rew [5] "Italy" - "Brazil" 0 : 0 .
result Game: "Italy" - "Brazil" 3 : 2
```

SIMULATING A SOCCER GAME

```
Maude> rew [3] "Malmö FF" - "Nottingham Forest" 0 : 0 .

result Game: "Malmö FF" - "Nottingham Forest" 2 : 1

Maude> rew [5] "Italy" - "Brazil" 0 : 0 .
result Game: "Italy" - "Brazil" 3 : 2
```

REACHABILITY ANALYSIS

Can away team win a game?

```
Maude> search [2]
         "Man U" - "Malmö FF" 0 : 0  =>*
         "Man U" - "Malmö FF" M:Nat : N:Nat
             such that M:Nat + 5 < N:Nat .

Solution 1 (state 27)
M --> 0
N --> 6

Solution 2 (state 35)
M --> 0
N --> 7
```

REACHABILITY ANALYSIS

Can away team win a game?

```
Maude> search [2]
         "Man U" - "Malmö FF" 0 : 0  =>*
         "Man U" - "Malmö FF" M:Nat : N:Nat
             such that M:Nat + 5 < N:Nat .

Solution 1 (state 27)
M --> 0
N --> 6

Solution 2 (state 35)
M --> 0
N --> 7
```

OBJECT-ORIENTED SPECS

- Classes, objects
- Distributed state: multiset of objects and messages
- Full Maude

OBJECT-ORIENTED SPECS: EXAMPLE

Example

```
class Person | age : Nat, status : Status .
```

Object is represented as a term

Example

```
< "Edward" : Person | age : 35, status : single >
```

OBJECT-ORIENTED SPECS: EXAMPLE

Example

```
class Person | age : Nat, status : Status .
```

Object is represented as a term

Example

```
< "Edward" : Person | age : 35, status : single >
```

OBJECT-ORIENTED SPECS: EXAMPLE

Example

```
crl [engage] : < X : Person | age : N, status : single >
                 < X' : Person | age : N', status : single >
               =>
                 < X : Person | status : engaged(X') >
                 < X' : Person | status : engaged(X) >
               if N > 15 /\ N' > 15 .


rl [death] :  < X : Person | >  => none .


rl [birthday] :  < X : Person | age : N >  =>
                   < X : Person | age : s N > .
```

OBJECT-ORIENTED SPECS: EXAMPLE

Example

```
crl [engage] : < X : Person | age : N, status : single >
                 < X' : Person | age : N', status : single >
               =>
                 < X : Person | status : engaged(X') >
                 < X' : Person | status : engaged(X) >
               if N > 15 /\ N' > 15 .

rl [death] :  < X : Person | >  => none .

rl [birthday] :  < X : Person | age : N >  =>
                   < X : Person | age : s N > .
```

OBJECT-ORIENTED SPECS: EXAMPLE

Example

```
crl [engage] : < X : Person | age : N, status : single >
                 < X' : Person | age : N', status : single >
               =>
                 < X : Person | status : engaged(X') >
                 < X' : Person | status : engaged(X) >
               if N > 15 /\ N' > 15 .

rl [death] :  < X : Person | >  => none .

rl [birthday] :  < X : Person | age : N >  =>
                 < X : Person | age : s N > .
```

APPLICATIONS/EXAMPLES

Distributed systems algorithms:

- (Dining philosophers)
- TCP, alternating bit protocol, sliding window, ...
- Two-phase commit for distributed transactions
  - failures (crash; Byzantine)
- Distributed mutual exclusion
  - central server algorithm
  - token ring
  - Maekawa's voting algorithm
- Distributed leader election
  - token ring
  - spanning-tree (wireless)
- Distributed consensus (sketch)

Security: NSPK

- crash course in cryptography

APPLICATIONS/EXAMPLES

Distributed systems algorithms:

- (Dining philosophers)
- TCP, alternating bit protocol, sliding window, ...
- Two-phase commit for distributed transactions
  - failures (crash; Byzantine)
- Distributed mutual exclusion
  - central server algorithm
  - token ring
  - Maekawa's voting algorithm
- Distributed leader election
  - token ring
  - spanning-tree (wireless)
- Distributed consensus (sketch)

Security: NSPK

- crash course in cryptography

APPLICATIONS (CONT.)

- Relevant for other courses
  - database, networking, security, OS
- "Modern" scenarios
  - distributed transaction

    reserve(X, OSL-CDG, KLM, Dec 6 to 15);
    reserve(X, Ritz, Imperial Suite, Dec 6 to 15);
    reserve(X, Chez M, dinner, Dec 9);
    pay(X, *6000*, MasterCard, *1234567891234567*, *11*/*20*, ...);

  - cloud computing: eBay item sold in Vanuatu and Bergen
    - cancel both purchases?
    - sell to "leader"/reach consensus on buyer?
- Industrial
  - 2PC, leader election, Paxos, ... key in Google, Facebook, etc.
- Security always sexy
  - no authentication $\longrightarrow$ no email, facebook, eBay, ...

APPLICATIONS (CONT.)

- Relevant for other courses
  - database, networking, security, OS
- "Modern" scenarios
  - distributed transaction

$$\begin{array}{l}
\mathit{reserve}(X, \text{OSL-CDG}, \text{KLM}, \text{Dec 6 to 15});\\
\mathit{reserve}(X, \text{Ritz}, \text{Imperial Suite}, \text{Dec 6 to 15});\\
\mathit{reserve}(X, \text{Chez M}, \text{dinner}, \text{Dec 9});\\
\mathit{pay}(X, \mathit{6000}, \text{MasterCard}, \mathit{1234567891234567}, \mathit{11/20}, \ldots);
\end{array}$$

  - cloud computing: eBay item sold in Vanuatu and Bergen
    - cancel both purchases?
    - sell to "leader"/reach consensus on buyer?
- Industrial
  - 2PC, leader election, Paxos, ... key in Google, Facebook, etc.
- Security always sexy
  - no authentication $\longrightarrow$ no email, facebook, eBay, ...

APPLICATIONS (CONT.)

- Relevant for other courses
  - database, networking, security, OS
- "Modern" scenarios
  - distributed transaction

    $$\begin{aligned}&\mathit{reserve}(X, \text{OSL-CDG}, \text{KLM}, \text{Dec 6 to 15});\\&\mathit{reserve}(X, \text{Ritz}, \text{Imperial Suite}, \text{Dec 6 to 15});\\&\mathit{reserve}(X, \text{Chez M}, \text{dinner}, \text{Dec 9});\\&\mathit{pay}(X, \mathit{6000}, \text{MasterCard}, \mathit{1234567891234567}, \mathit{11/20}, \ldots);\end{aligned}$$

  - cloud computing: eBay item sold in Vanuatu and Bergen
    - cancel both purchases?
    - sell to "leader"/reach consensus on buyer?
- Industrial
  - 2PC, leader election, Paxos, ... key in Google, Facebook, etc.
- Security always sexy
  - no authentication $\longrightarrow$ no email, facebook, eBay, ...

APPLICATIONS (CONT.)

- Relevant for other courses
  - database, networking, security, OS
- "Modern" scenarios
  - distributed transaction

    reserve(X, OSL-CDG, KLM, Dec 6 to 15);
    reserve(X, Ritz, Imperial Suite, Dec 6 to 15);
    reserve(X, Chez M, dinner, Dec 9);
    pay(X, *6000*, MasterCard, *1234567891234567*, *11/20*, ...);

  - cloud computing: eBay item sold in Vanuatu and Bergen
    - cancel both purchases?
    - sell to "leader"/reach consensus on buyer?
- Industrial
  - 2PC, leader election, Paxos, ... key in Google, Facebook, etc.
- Security always sexy
  - no authentication $\longrightarrow$ no email, facebook, eBay, ...

APPLICATIONS (CONT.)

- Relevant for other courses
  - database, networking, security, OS
- "Modern" scenarios
  - distributed transaction

    $\textit{reserve}(X, \text{OSL-CDG}, \text{KLM}, \text{Dec 6 to 15});$
    $\textit{reserve}(X, \text{Ritz}, \text{Imperial Suite}, \text{Dec 6 to 15});$
    $\textit{reserve}(X, \text{Chez M}, \text{dinner}, \text{Dec 9});$
    $\textit{pay}(X, 6000, \text{MasterCard}, 1234567891234567, 11/20, \ldots);$

  - cloud computing: eBay item sold in Vanuatu and Bergen
    - cancel both purchases?
    - sell to "leader"/reach consensus on buyer?
- Industrial
  - 2PC, leader election, Paxos, ... key in Google, Facebook, etc.
- Security always sexy
  - no authentication $\longrightarrow$ no email, facebook, eBay, ...

NSPK CRYPTOGRAPHIC PROTOCOL

Message 1.	$A \rightarrow B:$	$A.B.\{N_a.A\}_{PK_B}$
Message 2.	$B \rightarrow A:$	$B.A.\{N_a.N_b\}_{PK_A}$
Message 3.	$A \rightarrow B:$	$A.B.\{N_b\}_{PK_B}$

- Classic protocol from 1978
- Discussed in Handbook of Applied Cryptography from 1996 without mentioning error
- "Proved correct" using BAN logic by Burrows, Abadi and Needham in 1989
- Error found in 1995 by Lowe using model checking

NSPK IN MAUDE: NONCES AND KEYS

```
sort Nonce .
op nonce : Oid Nat -> Nonce [ctor] .

sort Key .
op pubKey : Oid -> Key [ctor] .
```

NSPK IN MAUDE: MESSAGES (I)

Three kinds of messages:

Message 1.	$A \rightarrow B:$	$A.B.\{N_a.A\}_{PK_B}$
Message 2.	$B \rightarrow A:$	$B.A.\{N_a.N_b\}_{PK_A}$
Message 3.	$A \rightarrow B:$	$A.B.\{N_b\}_{PK_B}$

- Part to be encrypted:

```
sorts PlainTextMsgContent EncrMsgContent .
op _;_ : Nonce Oid -> PlainTextMsgContent [ctor] .
op _;_ : Nonce Nonce -> PlainTextMsgContent [ctor] .
subsort Nonce < PlainTextMsgContent .
```

Example

"Plaintext" $N_a.A$ modeled by term `nonce(A,i) ; A` for some i

NSPK IN MAUDE: MESSAGES (I)

Three kinds of messages:

Message 1.	$A \rightarrow B:$	$A.B.\{N_a.A\}_{PK_B}$
Message 2.	$B \rightarrow A:$	$B.A.\{N_a.N_b\}_{PK_A}$
Message 3.	$A \rightarrow B:$	$A.B.\{N_b\}_{PK_B}$

- Part to be encrypted:

```
sorts PlainTextMsgContent EncrMsgContent .
op _;_ : Nonce Oid -> PlainTextMsgContent [ctor] .
op _;_ : Nonce Nonce -> PlainTextMsgContent [ctor] .
subsort Nonce < PlainTextMsgContent .
```

Example

"Plaintext" $N_a.A$ modeled by term `nonce(A,i) ; A` for some i

NSPK IN MAUDE: MESSAGES (II)

- Encrypted message content:

```
op encrypt_with_ : PlainTextMsgContent Key
                 -> EncrMsgContent [ctor] .
```

- Sender and receiver oid's using wrapper `msg_from_to_`

```
subsort EncrMsgContent < MsgContent .
```

Example

Message $A.B.\{N_a.A\}_{PK_B}$ modeled by

```
msg (encrypt nonce(A,i) ; A with pubKey(B))
from A to B
```

for some i

NSPK IN MAUDE: MESSAGES (II)

- Encrypted message content:

```
op encrypt_with_ : PlainTextMsgContent Key
                   -> EncrMsgContent [ctor] .
```

- Sender and receiver oid's using wrapper `msg_from_to_`

```
subsort EncrMsgContent < MsgContent .
```

Example

Message $A.B.\{N_a.A\}_{PK_B}$ modeled by

```
msg (encrypt nonce(A,i) ; A with pubKey(B))
from A to B
```

for some i

NSPK IN MAUDE: CLASSES

- Multiple concurrent runs with multiple agents
- Classes Initiator and Responder
    - some agents can be both initiator and responder:

```
class InitiatorAndResponder .
subclass
   InitiatorAndResponder < Initiator Responder .
```

NSPK IN MAUDE: CLASSES

- Multiple concurrent runs with multiple agents
- Classes `Initiator` and `Responder`
  - some agents can be both initiator and responder:

```
class InitiatorAndResponder .
subclass
   InitiatorAndResponder < Initiator Responder .
```

NSPK IN MAUDE: INITIATOR (I)

Message 1. $A \rightarrow B: \quad A.B.\{N_a.A\}_{PK_B}$
Message 2. $B \rightarrow A: \quad B.A.\{N_a.N_b\}_{PK_A}$
Message 3. $A \rightarrow B: \quad A.B.\{N_b\}_{PK_B}$

Initiator: how far has it run the protocol with every other agent:

1. not initiated desired contact
2. initiated contact and waiting for response
   - must remember the nonce it sent (N_a)
3. received response with correct nonce

NSPK IN MAUDE: INITIATOR (I)

Message 1.	$A \rightarrow B:$	$A.B.\{N_a.A\}_{PK_B}$
Message 2.	$B \rightarrow A:$	$B.A.\{N_a.N_b\}_{PK_A}$
Message 3.	$A \rightarrow B:$	$A.B.\{N_b\}_{PK_B}$

Initiator: how far has it run the protocol with every other agent:

1. not initiated desired contact
2. initiated contact and waiting for response
   - must remember the nonce it sent (N_a)
3. received response with correct nonce

NSPK IN MAUDE: INITIATOR (I)

Message 1. $A \rightarrow B: \quad A.B.\{N_a.A\}_{PK_B}$
Message 2. $B \rightarrow A: \quad B.A.\{N_a.N_b\}_{PK_A}$
Message 3. $A \rightarrow B: \quad A.B.\{N_b\}_{PK_B}$

Initiator: how far has it run the protocol with every other agent:

1. not initiated desired contact
2. initiated contact and waiting for response
   - must remember the nonce it sent (N_a)
3. received response with correct nonce

NSPK IN MAUDE: INITIATOR (II)

```
sorts Sessions InitSessions .
subsort Sessions < InitSessions .

op notInitiated : Oid -> InitSessions [ctor] .
op initiated : Oid Nonce -> InitSessions [ctor] .
op trustedConnection : Oid -> Sessions [ctor] .

op emptySession : -> Sessions [ctor] .
op __ : InitSessions InitSessions -> InitSessions
          [ctor assoc comm id: emptySession] .
op __ : Sessions Sessions -> Sessions
          [ctor assoc comm id: emptySession] .
```

NSPK IN MAUDE: INITIATOR (III)

Need counter for generating nonces:

```
class Initiator | initSessions : InitSessions,
                  nonceCtr : Nat .
```

NSPK IN MAUDE: SEND MESSAGE 1

Send Message 1 with freshly generated nonce:

Message 1. $A \rightarrow B: \quad A.B.\{N_a.A\}_{PK_B}$

```
vars A B : Oid .                 vars M N : Nat .
vars NONCE NONCE' : Nonce .  var IS : InitSessions .

rl [start-send-1] :
   < A : Initiator | initSessions : notInitiated(B) IS,
                      nonceCtr : N >
  =>
   < A : Initiator | initSessions :
                         initiated(B, nonce(A, N)) IS,
                      nonceCtr : N + 1 >
   msg (encrypt (nonce(A, N) ; A) with pubKey(B))
   from A to B .
```

NSPK IN MAUDE: SEND MESSAGE 3

Initiator replies with Message 3 when it receives Message 2 with the expected nonce:

Message 2.	$B \rightarrow A:$	$B.A.\{N_a.N_b\}_{PK_A}$
Message 3.	$A \rightarrow B:$	$A.B.\{N_b\}_{PK_B}$

```
rl [read-2-send-3] :
   (msg (encrypt (NONCE ; NONCE') with pubKey(A)) from B to A)
   < A : Initiator | initSessions : initiated(B, NONCE) IS >
  =>
   < A : Initiator | initSessions : trustedConnection(B) IS >
   msg (encrypt NONCE' with pubKey(B)) from A to B .
```

NSPK IN MAUDE

- 2 rules for responder
- 10 rules for Dolev-Yao intruder

NSPK IN MAUDE: ANALYSIS

Possible to break classic protocol?

- Agents: "Scrooge", "Bank", and intruder "Beagle Boys"
- "Scrooge" wants no contact with "Bank"
- If state where "Bank" has an established connection with "Scrooge" is reached, the protocol is unsafe!

NSPK IN MAUDE: INITIAL STATE WITH INTRUDER

```
op intruderInit : -> Configuration .
eq intruderInit =
     < "Scrooge" : Initiator |
                   initSessions : notInitiated("Beagle Boys"),
                   nonceCtr : 1 >
     < "Bank" : Responder |
                   respSessions : emptySession,
                   nonceCtr : 1 >
     < "Beagle Boys" : Intruder |
                   initSessions : emptySession,
                   respSessions : emptySession,
                   nonceCtr : 1,
                   agentsSeen : "Bank" ; "Beagle Boys",
                   noncesSeen : emptyNonceSet,
                   encrMsgsSeen : emptyEncrMsg > .
```

NSPK IN MAUDE: SEARCH

Is there a behavior from initial state to state where "Bank" thinks it talks to "Scrooge"?

```
(search [1]
   intruderInit
  =>*
   C:Configuration
   < "Bank" : Responder | respSessions :
                            trustedConnection("Scrooge")
                            RS:RespSessions > .)
```

NSPK IN MAUDE: SEARCH RESULT

After 100 minutes I got an answer

```
Solution 1
C:Configuration -->
 < "Scrooge" : Initiator |
       initSessions : trustedConnection("BeagleBoys"),
       nonceCtr : 2 >
 < "BeagleBoys" : Intruder |
       agentsSeen :("Bank" ; "Scrooge" ; "BeagleBoys"),
       encrMsgsSeen : encrypt nonce("Scrooge",1) ; nonce("Bank",
                      with pubKey("Scrooge"),
       initSessions : emptySession,    nonceCtr : 1,
       noncesSeen : nonce("Bank",1) nonce("Scrooge",1),
       respSessions : emptySession > ;
RS:RespSessions --> emptySession ;
...
```

NSPK IN MAUDE (CONT.)

Often search for compromised keys

- Bank has responded and is waiting for nonce N
- intruder knows nonce N
- analysis takes 15 seconds

EXAMPLE: TIC-TAC-TOE IN MAUDE

"State"

```
| o e x |
| e e e |
| e e e |
```

Only one rule:

```
mod TIC-TAC-TOE-GAME is pr TIC-TAC .
  sort State .

  op board:_turn:_ : Board Player -> State [ctor] .

  vars ROWS ROWS2 : Board .
  vars SQUARES1 SQUARES2 : Squares .
  var NZN : NzNat .
  var PLAYER : Player .

  crl [placePlayer] :
      board: ROWS | SQUARES1 e SQUARES2 | ROWS2 turn: PLAYER
     =>
      board: ROWS | SQUARES1 PLAYER SQUARES2 | ROWS2 turn: opposite(PLAYER)
     if not checkWin(ROWS | SQUARES1 e SQUARES2 | ROWS2, opposite(PLAYER)) .
```

FORMALIZING REQUIREMENTS

- Crash course on temporal logic
- Requirements formalized in LTL
  - including fairness assumptions
- Model checking in Maude

REAL-TIME AND PROBABILISTIC SYSTEMS

- Model-based performance estimation
- Sketched
- Monte-Carlo simulations of blackjack
- Statistical model checking (PVeStA)
  - "dealer-must-hit-all-17s" or "dealer-stands-on-all-17s"?
  - amount left from \$1000 after playing 20 rounds of \$100 blackjack? (\$876)
  - probability of winning > 200? (31%)

OTHER EXAMPLES/EXAM PROBLEMS

- Games: tic-tac-toe, jumping rabbits, Hanoi, coffee beans, ...
- Representing/simulating Turing machines
- "Meta-programming": implementing Ipo
- NP-complete problems: (integer) knapsack; traveling salesman, ...
- ...

Summary and Evaluation

SUMMARY

- Presented “criteria” for teaching FM
- Introduction to formal methods course at U. Oslo
  - second-year (and higher)
  - rewriting logic and Maude
- Course “satisfies” criteria (?)

SUMMARY

- Presented “criteria” for teaching FM
- Introduction to formal methods course at U. Oslo
    - second-year (and higher)
    - rewriting logic and Maude
- Course “satisfies” criteria (?)

SUMMARY (CONT.)

- Fun(ctional) programming/modeling
  - "They didn't know they were doing mathematics"
  - executable, expressive, simple, and intuitive formalism
  - not heavy math
- Example/application-driven
- Applications relevant for other courses and industry
- Single formalism/tool covers lot of ground
  - system modeling
  - requirements modeling (LTL)
  - model checking
  - verification by hand!
    - termination; confluence; inductive theorems; invariants
    - Maude has tool support for this!
  - model-based performance estimation (sketched; SMC)
- Folklore TRS and ADT stuff
- Introduction to logics (deduction rules; models; ...)

SUMMARY (CONT.)

- Fun(ctional) programming/modeling
  - "They didn't know they were doing mathematics"
  - executable, expressive, simple, and intuitive formalism
  - not heavy math
- Example/application-driven
- Applications relevant for other courses and industry
- Single formalism/tool covers lot of ground
  - system modeling
  - requirements modeling (LTL)
  - model checking
  - verification by hand!
    - termination; confluence; inductive theorems; invariants
    - Maude has tool support for this!
  - model-based performance estimation (sketched; SMC)
- Folklore TRS and ADT stuff
- Introduction to logics (deduction rules; models; ...)

SUMMARY (CONT.)

- Fun(ctional) programming/modeling
  - "They didn't know they were doing mathematics"
  - executable, expressive, simple, and intuitive formalism
  - not heavy math
- Example/application-driven
- Applications relevant for other courses and industry
- Single formalism/tool covers lot of ground
  - system modeling
  - requirements modeling (LTL)
  - model checking
  - verification by hand!
    - termination; confluence; inductive theorems; invariants
    - Maude has tool support for this!
  - model-based performance estimation (sketched; SMC)
- Folklore TRS and ADT stuff
- Introduction to logics (deduction rules; models; ...)

SUMMARY (CONT.)

- Fun(ctional) programming/modeling
  - "They didn't know they were doing mathematics"
  - executable, expressive, simple, and intuitive formalism
  - not heavy math
- Example/application-driven
- Applications relevant for other courses and industry
- Single formalism/tool covers lot of ground
  - system modeling
  - requirements modeling (LTL)
  - model checking
  - verification by hand!
    - termination; confluence; inductive theorems; invariants
    - Maude has tool support for this!
  - model-based performance estimation (sketched; SMC)
- Folklore TRS and ADT stuff
- Introduction to logics (deduction rules; models; ...)

SUMMARY (CONT.)

- Fun(ctional) programming/modeling
  - "They didn't know they were doing mathematics"
  - executable, expressive, simple, and intuitive formalism
  - not heavy math
- Example/application-driven
- Applications relevant for other courses and industry
- Single formalism/tool covers lot of ground
  - system modeling
  - requirements modeling (LTL)
  - model checking
  - verification by hand!
    - termination; confluence; inductive theorems; invariants
    - Maude has tool support for this!
  - model-based performance estimation (sketched; SMC)
- Folklore TRS and ADT stuff
- Introduction to logics (deduction rules; models; ...)

NEGATIVES

- Full Maude (for OO models) has defiencies
- Explicit-state analysis
  - takes time
  - scalability
- Lots of stuff missing
  - SMT/symbolic methods; higher-order logics; tool-assisted verification/theorem proving; ...
- Software/code verification missing
  - Maude/K really good for this!

NEGATIVES

- Full Maude (for OO models) has defiencies
- Explicit-state analysis
  - takes time
  - scalability
- Lots of stuff missing
  - SMT/symbolic methods; higher-order logics; tool-assisted verification/theorem proving; ...
- Software/code verification missing
  - Maude/K really good for this!

NEGATIVES

- Full Maude (for OO models) has defiencies
- Explicit-state analysis
  - takes time
  - scalability
- Lots of stuff missing
  - SMT/symbolic methods; higher-order logics; tool-assisted verification/theorem proving; ...
- Software/code verification missing
  - Maude/K really good for this!

NEGATIVES

- Full Maude (for OO models) has defiencies
- Explicit-state analysis
  - takes time
  - scalability
- Lots of stuff missing
  - SMT/symbolic methods; higher-order logics; tool-assisted verification/theorem proving; ...
- Software/code verification missing
  - Maude/K really good for this!

STUDENT FEEDBACK AND RESULTS

- Generally positive
- Industrial relevance very important!
  - not "safety-critical"!!
  - tool and methods used in industry
  - industrial problems/systems
- Not always happy with Full Maude
- They master temporal logic!
- Second-year students better feedback than older students!
  - exam grades (may) influence student feedback!
- Not too early for second-year students!

STUDENT FEEDBACK AND RESULTS

- Generally positive
- Industrial relevance very important!
  - not "safety-critical"!!
  - tool and methods used in industry
  - industrial problems/systems
- Not always happy with Full Maude
- They master temporal logic!
- Second-year students better feedback than older students!
  - exam grades (may) influence student feedback!
- Not too early for second-year students!

STUDENT FEEDBACK AND RESULTS

- Generally positive
- Industrial relevance very important!
  - not "safety-critical"!!
  - tool and methods used in industry
  - industrial problems/systems
- Not always happy with Full Maude
- They master temporal logic!
- Second-year students better feedback than older students!
  - exam grades (may) influence student feedback!
- Not too early for second-year students!

STUDENT FEEDBACK AND RESULTS

- Generally positive
- Industrial relevance very important!
  - not "safety-critical"!!
  - tool and methods used in industry
  - industrial problems/systems
- Not always happy with Full Maude
- They master temporal logic!
- Second-year students better feedback than older students!
  - exam grades (may) influence student feedback!
- Not too early for second-year students!

STUDENT FEEDBACK AND RESULTS

- Generally positive
- Industrial relevance very important!
  - not "safety-critical"!!
  - tool and methods used in industry
  - industrial problems/systems
- Not always happy with Full Maude
- They master temporal logic!
- Second-year students better feedback than older students!
  - exam grades (may) influence student feedback!
- Not too early for second-year students!

STUDENT FEEDBACK AND RESULTS

- Generally positive
- Industrial relevance very important!
  - not "safety-critical"!!
  - tool and methods used in industry
  - industrial problems/systems
- Not always happy with Full Maude
- They master temporal logic!
- Second-year students better feedback than older students!
  - exam grades (may) influence student feedback!
- Not too early for second-year students!

STUDENT FEEDBACK 2019

Hva er ditt generelle inntrykk av kurset? *

Svar	Antall	Prosent
Meget bra	4	50 %
Ganske bra	3	37,5 %
Noe bra	0	0 %
Hverken bra eller dårlig	1	12,5 %
Noe dårlig	0	0 %
Ganske dårlig	0	0 %
Meget dårlig	0	0 %

STUDENT FEEDBACK 2019

- VELDIG trege på tilbakemeldinger på obliger. Språk som er lite eller ikke brukt.
- Meget dyktig foreleser interessant tema
- Lærerikt. Flink foreleser
- Engasjemanget til professoren og det relevante innholdet
- God foreleser
- Gøy pensum, gode forelesninger, god stemning
- Dyktig foreleser og artig pensum
- Bra emne, men tok lang tid å få svar på obligatoriske oppgaver

STUDENT FEEDBACK 2019

Hva synes du om emnets nivå?

Svar	Antall	Prosent
For vanskelig	0	0 %
Vanskelig	4	50 %
Passe	4	50 %
Lett	0	0 %
For lett	0	0 %

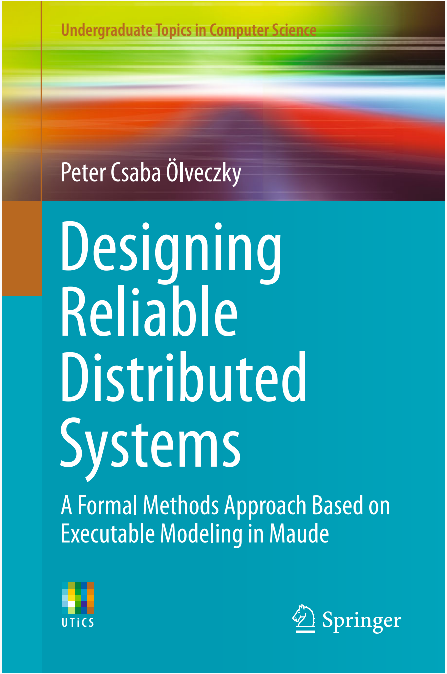
STUDENT FEEDBACK 2019

Hva likte du?

- Veldig interessant og lærerikt kurs Dyptgående kunnskap Et unikt kurs på bachelornivå i informatikk i Norge
- Gir viderekommen forståelse av logikk. Annerledes og kraftig metode for systemanalyse. Kreativt forelest/lærebok av Peter.
- Lære et annerledes programmeringsspråk Lære om algoritmer, og hvordan å modellere de for å sjekke etter sikkerhetstrussler Etter gjennomført emne sitter man inne med relevant kunnskap som flere av verdens fremste selskaper ønsker.
- Gode obliger, Fine oppgaver, interessante tema
- Introduksjon til et alternativt programmeringsparadigme Logikkdelene(konfuens, terminering og temporallogikk) Peter
- Rød tråd gjennom opplegget Dyktig foreleser Programmeringen var artig.
- Interessant, ikke for omfattende pensum, foreleser

STUDENT FEEDBACK 2019

- IN2100 er det beste faget jeg har hatt på UiO. Det er fordi Peter har hatt gode forelesninger og klart å lage en hyggelig atmosfære hvor det er lov å være dum. Norges morsomste foreleser.
- Artig foreleser, som tydelig kunne sitt fagfelt og formidle det, samt tydelig viste viktigheten av hvorfor man tok emnet.


Undergraduate Topics in Computer Science
Peter Csaba Ölveczky
Designing Reliable Distributed Systems
A Formal Methods Approach Based on Executable Modeling in Maude
UTiCS
Springer

Formal Methods at Amazon

66 COMMUNICATIONS OF THE ACM | APRIL 2015 | VOL. 58 | NO. 4

DOI:10.1145/2699417

Engineers use TLA+ to prevent serious but subtle bugs from reaching production.

BY CHRIS NEWCOMBE, TIM RATH, FAN ZHANG, BOGDAN MUNTEANU, MARC BROOKER, AND MICHAEL DEARDEUFF

How Amazon Web Services Uses Formal Methods

AMAZON WEB SERVICES

- Amazon Web Services (AWS):
  - world's largest cloud computing service provider
  - more profitable than Amazon's retail business
- Amazon Simple Storage Service (S3)
  - stores $>$ 3 trillion objects
  - 99.99% availability of objects
  - $>$ 1 million requests per second
- DynamoDB data store

AMAZON WEB SERVICES

- Amazon Web Services (AWS):
  - world's largest cloud computing service provider
  - more profitable than Amazon's retail business
- Amazon Simple Storage Service (S3)
  - stores $>$ 3 trillion objects
  - 99.99% availability of objects
  - $>$ 1 million requests per second
- DynamoDB data store

AMAZON WEB SERVICES AND FORMAL METHODS

- Formal methods used extensively at AWS during design of S3, DynamoDB, . . .
- Used Lamports TLA+
    - model checking

EXPERIENCES AT AMAZON WS

Model checking finds "corner case" bugs that would be hard to find with standard industrial methods:

EXPERIENCES AT AMAZON WS

Model checking finds "corner case" bugs that would be hard to find with standard industrial methods:

- "We have found that standard verification techniques in industry are necessary but not sufficient. We routinely use deep design reviews, static code analysis, stress testing, and fault-injection testing but still find that subtle bugs can hide in complex fault-tolerant systems."
- "the model checker found a bug that could lead to losing data [...]. This was a very subtle bug; the shortest error trace exhibiting the bug included 35 high-level steps. [...] The bug had passed unnoticed through extensive design reviews, code reviews, and testing."

EXPERIENCES AT AMAZON WS

Model checking finds "corner case" bugs that would be hard to find with standard industrial methods:

- "We have found that standard verification techniques in industry are necessary but not sufficient. We routinely use deep design reviews, static code analysis, stress testing, and fault-injection testing but still find that subtle bugs can hide in complex fault-tolerant systems."
- "the model checker found a bug that could lead to losing data [...]. This was a very subtle bug; the shortest error trace exhibiting the bug included 35 high-level steps. [...] The bug had passed unnoticed through extensive design reviews, code reviews, and testing."

EXPERIENCES AT AMAZON WS II

A formal specification is a valuable precise description of an algorithm:

EXPERIENCES AT AMAZON WS II

A formal specification is a valuable precise description of an algorithm:

- "the author is forced to think more clearly, helping eliminating "hand waving," and tools can be applied to check for errors in the design, even while it is being written. In contrast, conventional design documents consist of prose, static diagrams, and perhaps psuedo-code in an ad hoc untestable language."
- "Talk and design documents can be ambiguous or incomplete, and the executable code is much too large to absorb quickly and might not precisely reflect the intended design. In contrast, a formal specification is precise, short, and can be explored and experimented on with tools."

EXPERIENCES AT AMAZON WS II

A formal specification is a valuable precise description of an algorithm:

- "the author is forced to think more clearly, helping eliminating "hand waving," and tools can be applied to check for errors in the design, even while it is being written. In contrast, conventional design documents consist of prose, static diagrams, and perhaps psuedo-code in an ad hoc untestable language."
- "Talk and design documents can be ambiguous or incomplete, and the executable code is much too large to absorb quickly and might not precisely reflect the intended design. In contrast, a formal specification is precise, short, and can be explored and experimented on with tools."

EXPERIENCES AT AMAZON WS III

Formal methods are surprisingly feasible for mainstream software development and give good return on investment:

EXPERIENCES AT AMAZON WS III

Formal methods are surprisingly feasible for mainstream software development and give good return on investment:

- "In industry, formal methods have a reputation for requiring a huge amount of training and effort to verify a tiny piece of relatively straightforward code. Our experience with TLA+ shows this perception to be wrong. [...] Amazon engineers have used TLA+ on 10 large complex real-world systems. In each, TLA+ has added significant value. [...] Engineers have been able to learn TLA+ from scratch and get useful results in two to three weeks."
- "Using TLA+ in place of traditional proof writing would thus likely have improved time to market, in addition to achieving greater confidence in the system's correctness."

EXPERIENCES AT AMAZON WS III

Formal methods are surprisingly feasible for mainstream software development and give good return on investment:

- "In industry, formal methods have a reputation for requiring a huge amount of training and effort to verify a tiny piece of relatively straightforward code. Our experience with TLA+ shows this perception to be wrong. [...] Amazon engineers have used TLA+ on 10 large complex real-world systems. In each, TLA+ has added significant value. [...] Engineers have been able to learn TLA+ from scratch and get useful results in two to three weeks."
- "Using TLA+ in place of traditional proof writing would thus likely have improved time to market, in addition to achieving greater confidence in the system's correctness."

EXPERIENCES AT AMAZON WS III

Quick and easy to experiment with different design choices:

- "We have been able to make innovative performance optimizations [...] we would not have dared to do without having model-checked those changes. A precise, testable description of a system becomes a what-if tool for designs."

EXPERIENCES AT AMAZON WS III

Quick and easy to experiment with different design choices:

- "We have been able to make innovative performance optimizations [...] we would not have dared to do without having model-checked those changes. A precise, testable description of a system becomes a what-if tool for designs."

EXPERIENCES AT AMAZON WS: LIMITATIONS

TLA+ did/could not analyze performance degradation

MAUDE VS TLA+

Maude should be better suited!

- more intuitive and expressive specification language
    - OO
    - hierarchical states
    - dynamic object/message creation/deletion
    - ...
- Support for real-time and probabilistic systems
- Also for performance estimation!

CONCLUSIONS AT AMAZON

» key insights

- Formal methods find bugs in system designs that cannot be found through any other technique we know of.
- Formal methods are surprisingly feasible for mainstream software development and give good return on investment.
- At Amazon, formal methods are routinely applied to the design of complex real-world software, including public cloud services.

CONCLUSIONS AT AMAZON

Conclusion

Formal methods are a big success at AWS, helping us prevent subtle but serious bugs from reaching production, bugs we would not have found through any other technique. They have helped us devise aggressive optimizations to complex algorithms without sacrificing quality. At the time of this writing, seven Amazon teams have used TLA+, all finding value in doing so, and more Amazon teams are starting to use it. Using TLA+ will improve both time-to-market and quality of our systems.