CISC/CMPE422, CISC835: Formal Methods in Software Engineering

Juergen Dingel

Fall 2019

Lecture 1: Admin, Motivation & Overview

Admin

- Marking scheme for CISC/CMPE422
  - Final exam: 50% of mark
  - 1 midterm: 20% of mark
  - Assignments (4, individual, weighted equally): 30% of mark
- Marking scheme for CISC835
  - Final: 40%, midterm: 15%, assignments: 25%, project: 20%
- Exams (closed book, 1 8.5”x11” datasheet):
  - Midterm: Week 10 (Thurs, Nov 14), in class
  - Final: tba
  - Accommodations? Contact exams office or me
- Course web page: **www.cs.queensu.ca/~cisc422**
  - Syllabus, assignments, etc
- Course material:
  - Courseware available in bookstore
- TAs:
  - Anika Anwar, Karim Jahed, Lama Moukahal, Liam Walsh

Admin (Cont’d)

- JD will be away
  - Week 2/3 (Sept 16 - Sept 19)

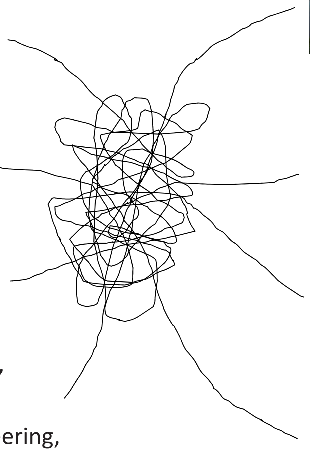
About Me

Small town Germany: Born, raised, etc

Berlin: UG

Pittsburgh: PhD

Kingston: since 2000, Formal methods, Model-Driven Engineering, SW Eng



A

B

C

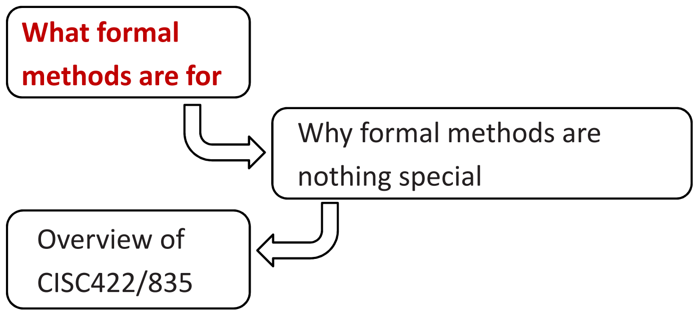
D

A Definition

Formal methods

- Notations, techniques and tools to
  - capture relevant aspects of software unambiguously and precisely and
  - allow analysis
- Another title: “Formal Modeling and Analysis”

Overview of this Lecture



What formal methods are for
Why formal methods are nothing special
Overview of CISC422/835

What Formal Methods Are For

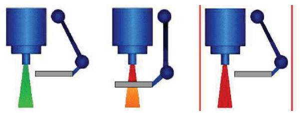
Statement 1:
“Sometimes, it is very important that certain $\boxed{\textit{software failures}}$ *don’t occur and that there is acceptable supporting evidence for this”*

Statement 1: Examples

- ‘Safety-’ or ‘mission-critical’ software
  - Military, nuclear, medical, automotive, avionics, aerospace
- Infrastructure
  - Energy, telecom, avionics
- Economy
  - Financial

Example 1: Therac-25 (1985-87)

- Radiotherapy machine with SW controller
- SW failed to maintain essential invariants:
  - To generate X-rays:
    - either use low-power electron beam, or
    - use high-power beam w/ intervening 'beam spreader plate'
- Several deaths ($\geq$ 6) due to burning

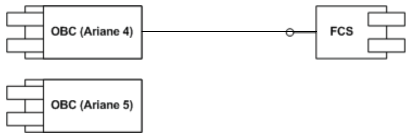


Example 2: ESA Ariane 5 (June 1996)

- On June 4, 1996, unmanned Ariane 5 launched by ESA explodes 40 seconds after lift-off
- One decade of development costing $7billion lost

- What went wrong?
  - Bad reuse of code from Ariane 4
  - Bad fault-tolerance mechanism
  - Bad coding practices

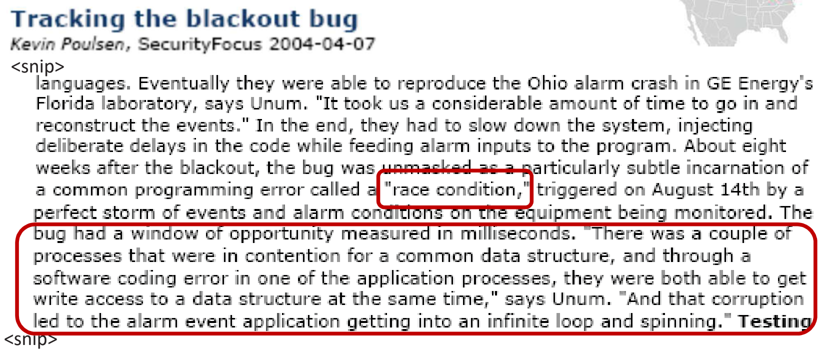
Example 2: ESA Ariane 5 (June 1996) (Cont'd)



- Example of how not to do reuse:
  - Parts of FCS from Ariane 4
  - v_x much greater for Ariane 5
  - Conversion operation in FCS fails
  - OBC interprets error code as flight data
  - …
  - Launcher self-destructs
- Example of how not to achieve fault-tolerance:
  - FCS and backup FCS identical, thus backup also failed
- Example of how not to code:
  - When code caused exception, it wasn't even needed anymore
- References:
  - [Gle96] and **www.ima.umn.edu/~arnold/disasters/ariane.html**

Example 3: The Blackout Bug

- Aug 13, 2003: >50 Million people w/o electricity for hours, days
- Cause: Race condition in alarm system (10^6 Loc of C)
- Worst black out in North American history
- Cost: US$ 6 billion



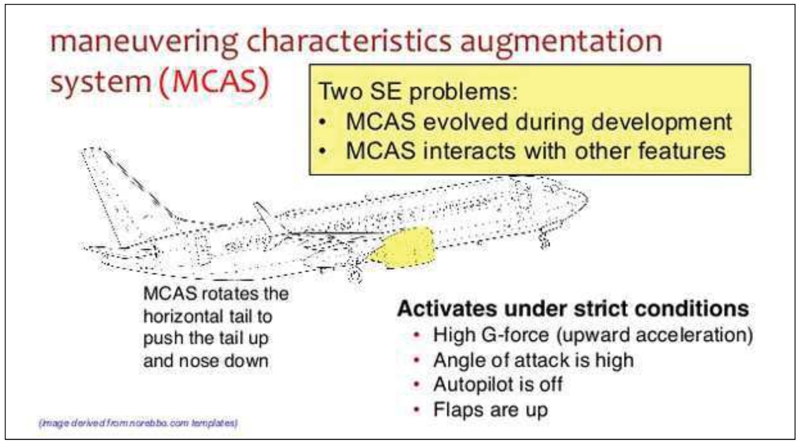
Tracking the blackout bug
Kevin Poulsen, SecurityFocus 2004-04-07
<snip>
languages. Eventually they were able to reproduce the Ohio alarm crash in GE Energy's Florida laboratory, says Unum. "It took us a considerable amount of time to go in and reconstruct the events." In the end, they had to slow down the system, injecting deliberate delays in the code while feeding alarm inputs to the program. About eight weeks after the blackout, the bug was unmasked as a particularly subtle incarnation of a common programming error called a "race condition," triggered on August 14th by a perfect storm of events and alarm conditions on the equipment being monitored. The bug had a window of opportunity measured in milliseconds. "There was a couple of processes that were in contention for a common data structure, and through a software coding error in one of the application processes, they were both able to get write access to a data structure at the same time," says Unum. "And that corruption led to the alarm event application getting into an infinite loop and spinning." **Testing**
<snip>

Example 4: 2010 Toyota Prius

- Three systems
  - Hybrid brake system
    - Normal
    - Regenerative
  - Anti-lock brake system (ABS)
- Unintended interaction
  - Braking force reduced after ABS actuation
  - $\Rightarrow$ Increased stopping distance
  - $\Rightarrow$ 62 crashes, 12 injuries

US NHTSA,
https://www.nhtsa.gov/vehicle/2010/TOYOTA/PRIUS/4%252520DR#investigations

Example 5: Boeing 737 Max



maneuvering characteristics augmentation system (MCAS)

Two SE problems:
- MCAS evolved during development
- MCAS interacts with other features

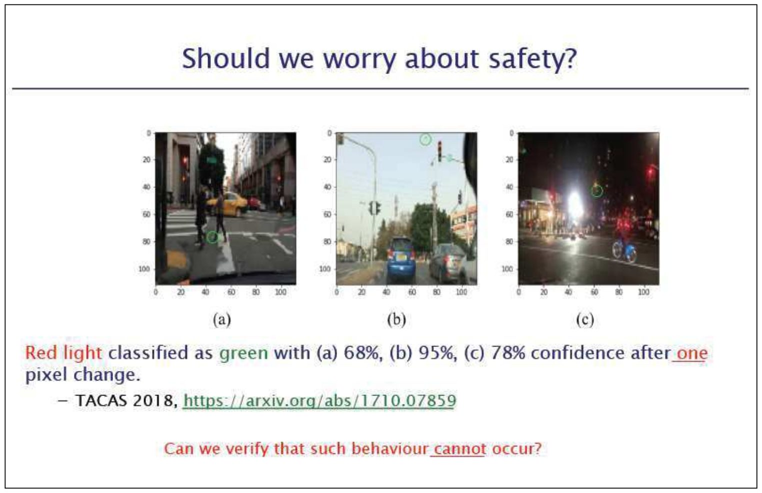
MCAS rotates the horizontal tail to push the tail up and nose down

Activates under strict conditions
- High G-force (upward acceleration)
- Angle of attack is high
- Autopilot is off
- Flaps are up

(image derived from norebbo.com templates)

[Slide from Jo Atlee, Living with Feature Interactions, FSE'19]

Example 6: Deep Neural Nets for Autonomous Driving



Should we worry about safety?

(a) (b) (c)

Red light classified as green with (a) 68%, (b) 95%, (c) 78% confidence after one pixel change.

– TACAS 2018, https://arxiv.org/abs/1710.07859

Can we verify that such behaviour cannot occur?

[Marta Kwiatkowska, Safety and Robustness for Deep Learning with Provable Guarantees, FSE'19]

What Formal Methods Are For

Statement 1:
"Sometimes, it is very important that certain software failures don't occur and that there is acceptable supporting evidence for this"

Certification of Software in Medical Devices

The FDA's analysis of 3140 medical device recalls conducted between 1992 and 1998 reveals that 242 of them (7.7%) are attributable to software failures

[…] any medical device software product developed after June 1, 1997 […] is subject to applicable design control provisions. (See of 21 CFR §820.30.) […]
Other design controls, such as planning, input, verification, and reviews, are required for medical device software. (See 21 CFR §820.30.)

The corresponding documented results from these activities can provide additional support for a conclusion that medical device software is validated.

[FDA] U.S. Department of Health and Human Services, Food and Drug Administration, Center for Devices and Radiological Health, Center for Biologics Evaluation and Research. General Principles of Software Validation; Final Guidance for Industry and FDA Staff. Jan 2002] http://www.fda.gov/RegulatoryInformation/Guidances/ucm085281.htm

Certification of Avionics Software

DO-178C

"is an acceptable means, but not the only means, for showing compliance with the applicable airworthiness regulations for the software aspects of airborne systems and equipment certification"

- Software levels
  - From E (failure has no effect) to A (failure has catastrophic effect)
- Certification objectives
  - the higher the level, the more objectives
- Examples of activities necessary to satisfy objectives
  - Review of requirements, design, and code; testing; configuration management

[Radio Technical Commission for Aeronautics (RTCA). DO-178C: Software Considerations in Airborne Systems and Equipment Certification. Jan 2012] https://en.wikipedia.org/wiki/DO-178C

ISO Standard for Automotive Software

Goals of ISO 26262

- Covers functional safety aspects of the entire development process
- Provides an automotive-specific risk-based approach for determining risk classes (Automotive Safety Integrity Levels, ASILs)
- Uses ASILs for specifying the item's necessary safety requirements for achieving an acceptable residual risk
- Provides requirements for validation and confirmation measures to ensure a sufficient and acceptable level of safety is being achieved

[International Standards Organization (ISO). "Road vehicles – Functional safety (ISO 26262)". 2011] https://en.wikipedia.org/wiki/ISO_26262

What Formal Methods Are For

Statement 1:
"Sometimes, it is very important that certain software failures don't occur and that there is acceptable supporting evidence for this"

Statement 2:
"Sometimes, relevant aspects of the software (e.g., requirements, development context, operating conditions) are so complex that Statement 1 is impossible to achieve with 'standard' methods"

The Limits of Testing

"We test exhaustively, we test with third parties, and we had in excess of three million online operational hours in which nothing had ever exercised that bug. […] I'm not sure that more testing would have revealed that."

Manager at GE,
maker of Energy Management System responsible for Blackout Bug in 2003
in 'Tracking the blackout bug'

"Typically, testing alone cannot fully verify that software is complete and correct. In addition to testing, other verification techniques and a structured and documented development process should be combined to ensure a comprehensive validation approach"

In [FDA Guidelines]

"Testing shows the presence, not the absence of bugs."
Edsger W. Dijkstra

E.W.Dijkstra.
Turing Award 1972

Software Complexity: In Lines of Code

- **Windows OSs**
  - NT 3.1 (1993): 0.5 million LoC
  - 95:
    - 11
    tes
  - 2000: 29 million LoC
  - XP (2001): 35 million LoC
  - Vista (2007): 50 million LoC
  - Windows 7: 40 million LoC
- **Windows**
  - Office (2001): 25 million LoC
  - Office (2013): 44 million LoC
  - Visual Studio (2012): 50 million LoC
- **Mac OS X "Tiger":** 85 million LoC

- **Average iPhone app:** 40,000 LoC
- **Pacemaker:** 100,000 LoC
- million LoC
- n LoC
- **Boeing 787:** 14 million LoC
- **F-35 fighter jet:** 24 million LoC
- **Large Hadron Collider:** 50 million LoC
- **Facebook:** 60 million LoC
- **Car**
  - 1981: 50,000 LoC
  - 2005: 10 million LoC
  - 2014: 100 million LoC

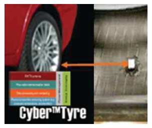
Software is one of the most complex man-made artifacts!

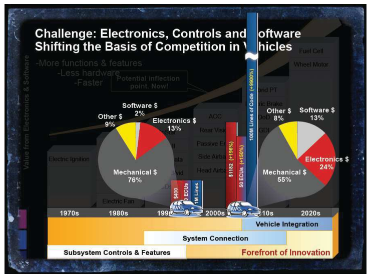
1M LoC = 18,000 pages of printed text
= stack 6 feet high

[Charette. "Why Software Fails". IEEE Spectrum, Sept 2005]
[McCandless, www.informationisbeautiful.net/visualizations/million-lines-of-code]

It is Not Going to Get Easier

- **More complexity**
  - Less mechanical, more electronic & computerized
  - More features & capabilities
  - More integration
  - More virtualization, distribution & concurrency





[from A. Sangiovanni-Vincentelli]

What Formal Methods Are For

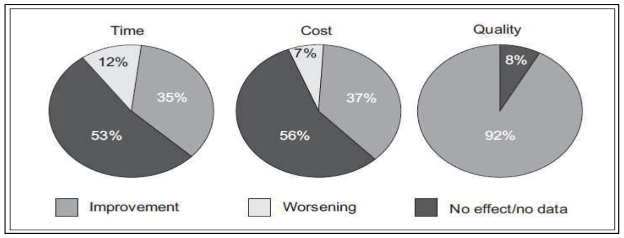
Statement 1:
"Sometimes, it is very important that certain software failures don't occur and that there is acceptable supporting evidence for this"

Statement 2:
"Sometimes, relevant aspects of the software (e.g., requirements, development context, operating conditions) are so complex that Claim 1 is impossible to achieve with 'standard' methods"

Statement 3:
"In these cases, formal methods can help by allowing the construction of unambiguous artifacts modeling relevant aspects of the system such that it can be analyzed w.r.t. desirable properties"

Examples of Uses of Formal Methods

- DO-178C for avionics software allows formal methods to complement testing
- Survey of 62 int'l FM projects
  - **Domains:** Real-time, distributed & parallel, transaction processing, high-data volume, control, services



[Radio Technical Commission for Aeronautics (RTCA). DO-333: Formal Methods Supplement to DO-178C and DO-278A.
[Woodcock et al. Formal Methods: Practice and Experience. ACM Computing Surveys 41(4). 2009]

A Definition

Formal methods

- Notations, techniques and tools to
  - capture aspects of software unambiguously and precisely and
  - allow analysis
  - make software engineering more rigorous

What is Software Engineering?

engineering:

"The application of scientific and mathematical principles to practical ends such as the design, manufacture, and operation of efficient and economical structures, machines, processes, and systems"

American Heritage Dictionary

software engineering:

"The application of a systematic, disciplined, quantifiable approach to the development, operation, and maintenance of software, that is, the application of engineering to software"

IEEE Standard 610.12

Yeah, right!

What is Software Engineering (Cont'd)

Software Engineering currently isn't like engineering at all!

Engineering

1. build (mathematical) models
2. analyze models rigorously
3. refine models
4. build artifact
5. little testing

Characteristics

- Very rigorous
- "front-loaded"
- **Main QA technique:**
  Modeling & analysis

Software Engineering

1. some (informal) modeling
2. build artifact
3. some (informal) reuse
4. lots of testing

Characteristics

- Mostly informal
- "back-loaded"
- **Main QA technique:**
  Testing (often >50% of total development effort)

Overview of this Lecture

What formal methods are for

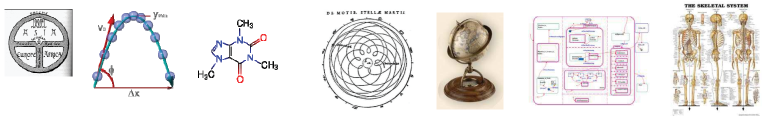
→ **Why formal methods are nothing special**

→ Overview of CISC422/835

‘Formal Methods’ in Other Disciplines

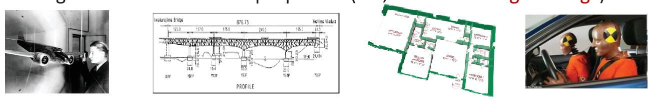
Natural sciences

- Understanding, predicting existing phenomena (c.f., “Backwards Engineering”)



Engineering

- Building artifacts with certain properties (c.f., “Forwards Engineering”)



Entertainment

- Doing what normally would be impossible

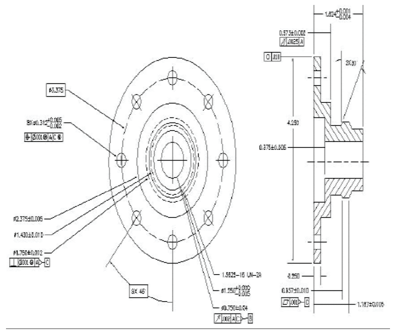
Modeling is central, except in SW Eng

Formal Modeling in Manufacturing

Mechanical design from 1800 to about 1980:

1. Draftsmen create 3-view drawings
2. Machinists create parts from drawings

$\Rightarrow$ laborious, error-prone, inefficient



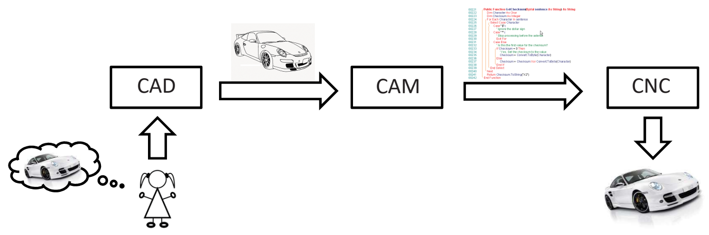
Formal Modeling in Manufacturing (Cont’d)

- **Example: Concorde (1976 – 2003)**
  - > 100,000 drawings
  - in 2 languages, using both metric and imperial systems
  - $\Rightarrow$ worked, but 7x over budget

Formal Modeling in Manufacturing (Cont'd)

Mechanical design from about 1972: CAD/CAM

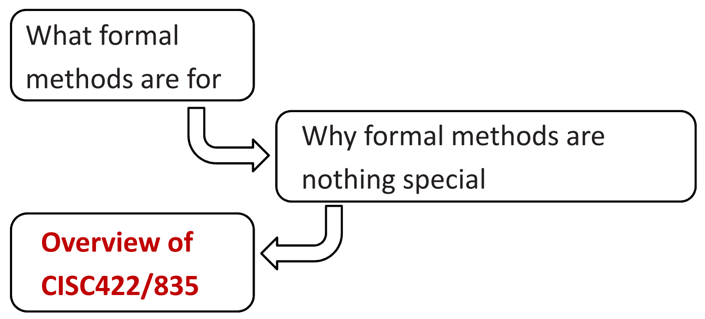
1. Create drawings w/ computer (CAD)
2. From drawing, computer automatically generates program to drive milling and CNC machines (CAM)



⇒ much better analysis capabilities and productivity

⇒ **CAD/CAM has revolutionized manufacturing**

Overview of this Lecture


What formal methods are for
Why formal methods are nothing special
Overview of CISC422/835

CISC422/835: Overview

- Will consider three different artifacts
  - requirements
  - designs (object models)
  - finite state machines
- For each artifact we will look at
  - a formal notation allowing the artifact to be modeled formally
  - an technique that analyzes the model automatically
  - a tool that implements this analysis
- Things you are going to learn
  - Details about notations, analysis techniques, and tools
  - Formalization

CISC422/835: Overview (Cont'd)

More precisely, the course will cover the following 3 main topics:

- Formal modeling and analysis of requirements (~3 weeks)
  - Logic review
    - propositional logic
    - predicate logic & theorem proving (briefly)
  - Z
- Formal modeling and analysis of class models (~3 weeks)
  - Alloy & constraint checking
- Formal modeling and analysis of programs (~4 weeks)
  - Finite state machines & model checking

Bugs Often Creep in Early in Development…

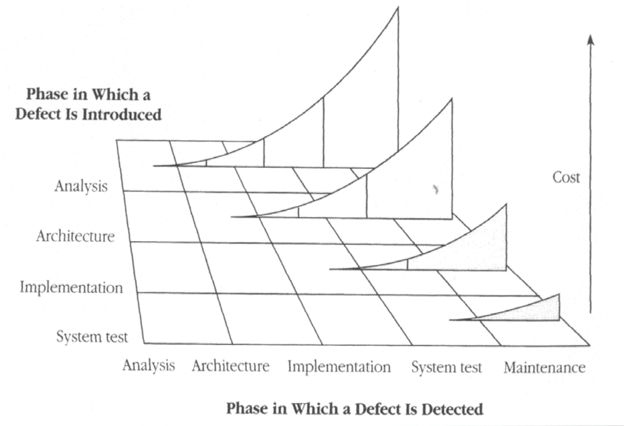
1. *“70% of errors in embedded safety-critical software are introduced in the requirements (35%) and architecture design phases (35%)”*
2. *“80% of all errors are not discovered until system integration or later*

[Feiler, Goodenough, Gurfinkel, Weinstock, Wrage. Four Pillars for Improving the Quality of Safety-Critical Software Reliant Systems. White Paper. SEI. 2013]

What’s the relationship between time bug is discovered and costs?

… at High Costs

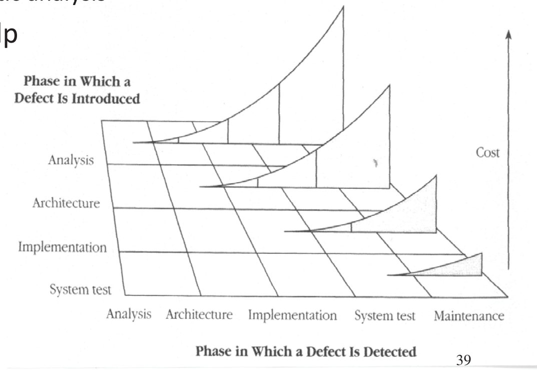
What’s the relationship between time bug is discovered and costs?



Source: S. McConnell. Code Complete. 1993

Formal Modeling and Analysis of Requirements

- Bugs in requirements can be very costly
- Informal English sometimes inappropriate:
  - Verbose
  - Ambiguous
  - Not amenable to automatic analysis
- Formal notations can help
- We’ll look at
  - propositional logic
  - predicate logic
  - Z



Formal Modeling and Analysis of Class Models

- *“A picture says more than a 1000 words”*
- UML de-facto standard, but
  - is not completely language-independent
  - does not have precise semantics
    - not amenable for automatic analysis
- We’ll look at an class-modeling language (Alloy), that is
  - Reminiscent of UML
  - Language-independent
  - Easy/easier to use
  - Has precise semantics
  - Comes with usable, powerful, automatic analysis tool
  - *“Brings specifications to life”*

Formal Modeling and Analysis of Programs

- Even small pieces of code can be very intricate
- Example: Tie-breaker protocol for mutual exclusion

```
P1 =
while true do
  f1  := true;
  last  := 1;
  await  (!f2 or last!=1);
  criticalSection1;
  f1  := false
end
```

```
P2 =
while true do
  f2  := true;
  last  := 2;
  await  (!f1 or last!=2);
  criticalSection2;
  f2  := false
end
```

What if

```
f1:=true; last:=1
```

is replaced by

```
last:=1; f1:=true
```

in **`P1`** and similarly for **`P2`**?

BTW, embedded code very often is concurrent

Formal Modeling and Analysis of Programs (Cont'd)

- Resulting version of Tie-breaker protocol is incorrect

```
P1 =
while true do
  last  := 1;
  f1  := true;
  await  (!f2 or last!=1);
  criticalSection1;
  f1  := false
od
```

```
P2 =
while true do
  last  := 2;
  f2  := true;
  await  (!f1 or last!=2);
  criticalSection2;
  f2  := false
od
```

$$\begin{tikzcd}
\begin{array}{l} f1=false \\ f2=false \\ last=* \end{array} \arrow[r, "P2"] & \begin{array}{l} f1=false \\ f2=false \\ last=2 \end{array} \arrow[r, "P1"] & \begin{array}{l} f1=true \\ f2=false \\ last=1 \\ \text{/* P1 in CS */} \end{array} \arrow[r, "P2"] & \begin{array}{l} f1=true \\ f2=true \\ last=1 \\ \text{/* P2 in CS */} \end{array}
\end{tikzcd}$$

Formal Modeling and Analysis of Programs (Cont'd)

- Model checking
  - Perfect for these kinds of problems
  - Analysis technique for finite state machines and protocols based on exhaustive state space exploration and temporal logic
- Temporal logic
  - Logic that allows specification of how computation unfolds
  - 2 kinds of properties
    - Something bad will never happen (safety property)
      - *"x will never be negative"*
      - *"the system will never deadlock"*
    - Something good will eventually happen (liveness property)
      - *"every request will eventually be granted"*

Summary

- Software is becoming more pervasive & complex
- Formal modeling and analysis can help
- CISC422/835 offers a comparative study of different formal modeling notations and analysis techniques for different artifacts:
  - Requirements
    - Propositional and Predicate logic & theorem proving
  - Class models & constraint solving
  - Finite state machines & model checking

Admin

- Marking scheme for CISC/CMPE422
  - Final exam: 50% of mark
  - 1 midterm: 20% of mark
  - Assignments (4, individual, weighted equally): 30% of mark
- Marking scheme for CISC835
  - Final: 40%, midterm: 15%, assignments: 25%, project: 20%
- Exams (closed book, 1 8.5”x11” datasheet):
  - Midterm: Week 10 (Thurs, Nov 14), in class
  - Final: tba
  - Accommodations? Contact exams office or me
- Course web page: **www.cs.queensu.ca/~cisc422**
  - Syllabus, assignments, etc
- Course material:
  - Courseware available in bookstore
- TAs:
  - Anika Anwar, Karim Jahed, Lama Moukahal, Liam Walsh