CS 477: Formal Methods for Software Development

Sasa Misailovic

Based on previous slides by Gul Agha, Elsa Gunter, Mahesh Viswanathan, and Madhusudan Parathasarathy

University of Illinois at Urbana-Champaign

CS 477

This course gives an overview of mathematical models, languages, and methods for software specification, development, and verification.

- 3 undergraduate hours.
- 4 graduate hours.
- Prerequisites: CS 225; CS 374 or MATH 414.
- Useful: CS 421; CS 475

CS 477: Course Details

Instructor:
Prof. Sasa Misailovic
4110 Siebel Center

Office hours:
TBD
by appointment

Email: *misailo@illinois.edu*

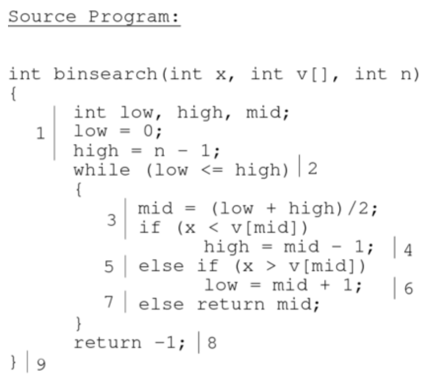
Teaching Assistant:
Vimuth Fernando
3107 Siebel Center

Office hours:
Wed 2-4pm
or by appointment

Email: wvf2@*illinois.edu*

Course Website:
https://courses.grainger.illinois.edu/cs477/fa2021/

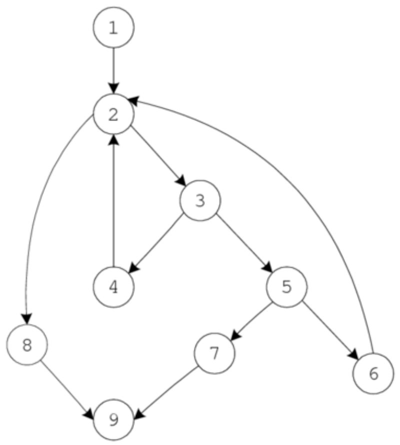
From Real World to Formal Methods

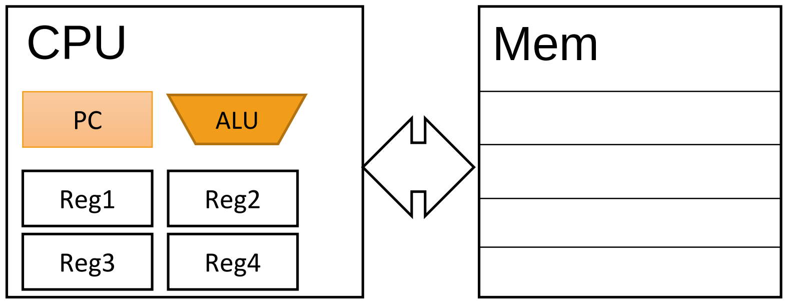


Source Program:

```
int binsearch(int x, int v[], int n)
{
      | int low, high, mid;
    1 | low = 0;
      | high = n - 1;
        while (low <= high) | 2
        {
            | mid = (low + high)/2;
          3 | if (x < v[mid])
                  high = mid - 1;  | 4
          5 | else if (x > v[mid])
                  low = mid + 1;   | 6
          7 | else return mid;
        }
        return -1; | 8
} | 9
```

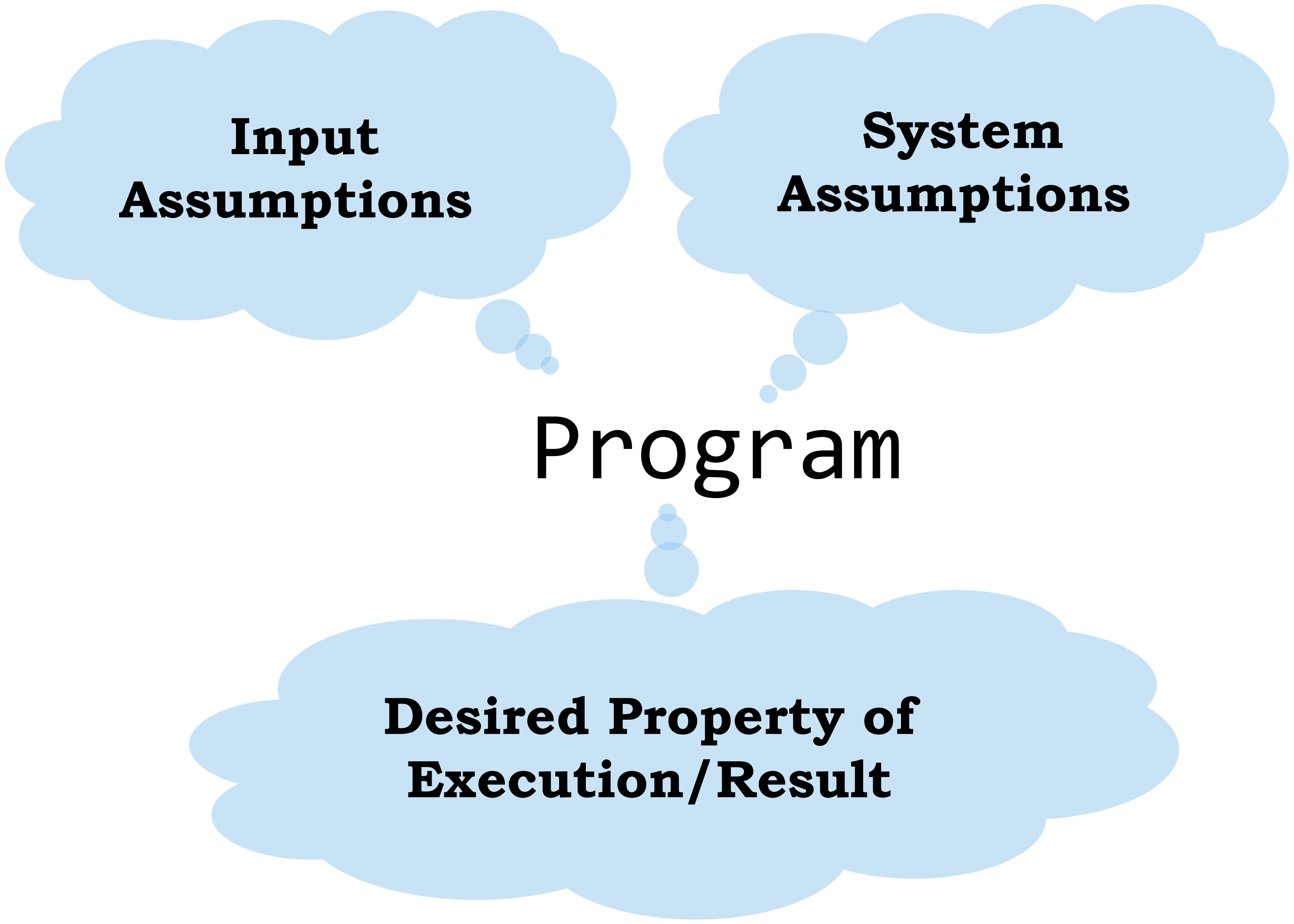
CFG:





Sources: https://www.researchgate.net/figure/Example-of-Control-Flow-Graph_fig5_4065402 and https://freesvg.org/computer-station-vector-graphics

What do we want to do?



What do we want to do?

What assumptions do we need to make about inputs?
What assumptions do we need to make about hardware?

```
read(x)        x=abs(x)
y=sqrt(x)
return y
```

```
double sqrt_newtonmethod(double x) {
      if (x==0) return x;
      double div = x, val = x, last;
      do {
          last = val;
          val = (val + div / val) * 0.5;
      } while (abs(val - last) > 1e-9);
      return val;
  }
```

Is the program going to experience an error?
Is the program going to terminate?
Is the program going to produce a correct/accurate result?

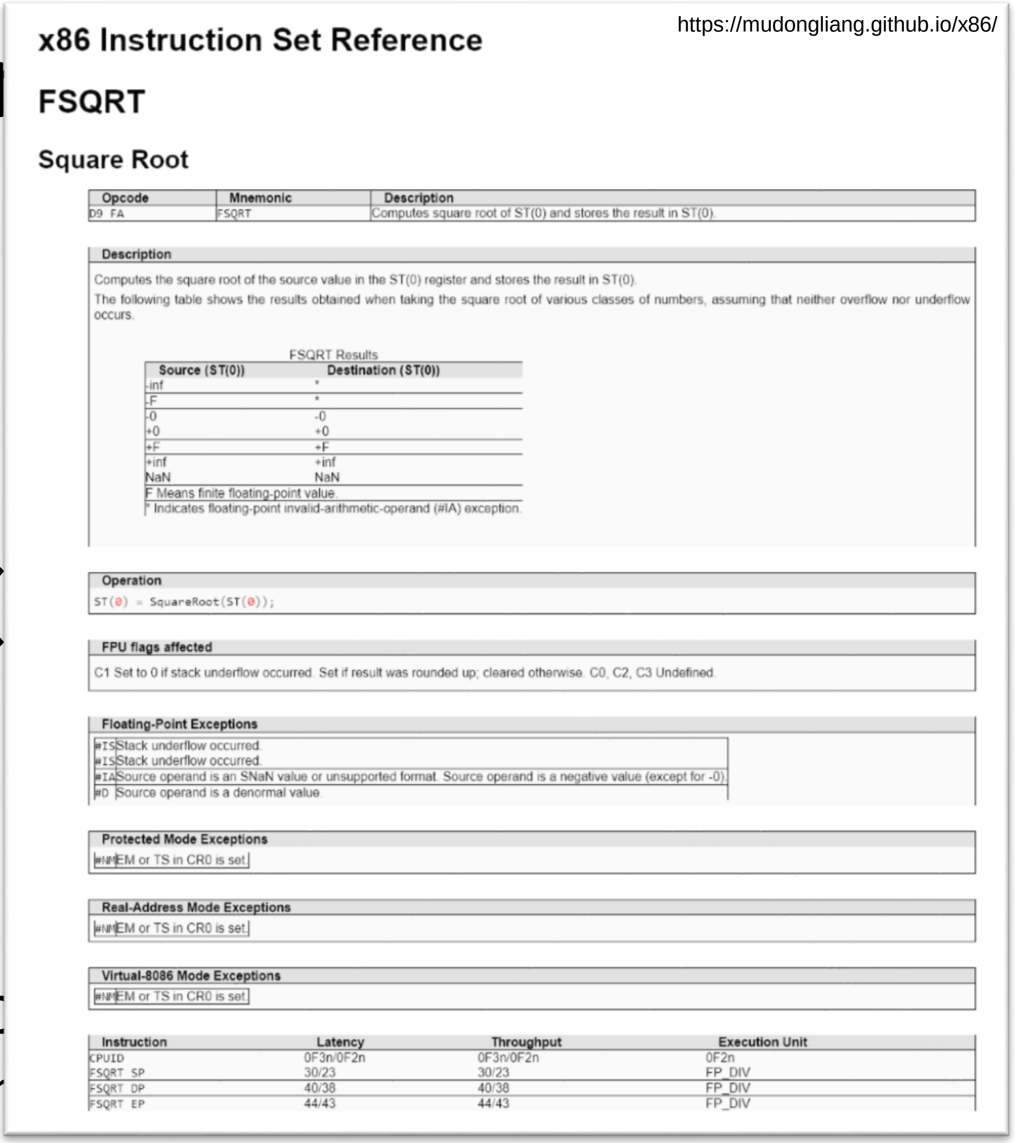
What do we wa

What assumptions do we n
What assumptions do we n

```
read(x)
y=sqrt(x)
return y
```

x=abs(

Is the program going to exp
Is the program going to ter
Is the program going to produce a correct/accurate result?



https://mudongliang.github.io/x86/

x86 Instruction Set Reference

FSQRT

Square Root

Opcode	Mnemonic	Description
D9 FA	FSQRT	Computes square root of ST(0) and stores the result in ST(0).

Description

Computes the square root of the source value in the ST(0) register and stores the result in ST(0).

The following table shows the results obtained when taking the square root of various classes of numbers, assuming that neither overflow nor underflow occurs.

FSQRT Results

Source (ST(0))	Destination (ST(0))
-inf	*
-F	*
-0	-0
+0	+0
+F	+F
+inf	+inf
NaN	NaN
F Means finite floating-point value.	
* Indicates floating-point invalid-arithmetic-operand (#IA) exception.	

Operation

```
ST(0) = SquareRoot(ST(0));
```

FPU flags affected

C1 Set to 0 if stack underflow occurred. Set if result was rounded up; cleared otherwise. C0, C2, C3 Undefined.

Floating-Point Exceptions

#IS	Stack underflow occurred.
#IS	Stack underflow occurred.
#IA	Source operand is an SNaN value or unsupported format. Source operand is a negative value (except for -0).
#D	Source operand is a denormal value.

Protected Mode Exceptions

#NM	EM or TS in CR0 is set.

Real-Address Mode Exceptions

#NM	EM or TS in CR0 is set.

Virtual-8086 Mode Exceptions

#NM	EM or TS in CR0 is set.

Instruction	Latency	Throughput	Execution Unit
CPUID	0F3n/0F2n	0F3n/0F2n	0F2n
FSQRT SP	30/23	30/23	FP_DIV
FSQRT DP	40/38	40/38	FP_DIV
FSQRT EP	44/43	44/43	FP_DIV

What Kinds of Math?

- Sets, Graphs, Trees
- Automata and formal models of computing
- Logic and Proof theory, Temporal logics
- Induction, especially structural induction and well-founded induction, inductive relations

---

- Polyhedral Analysis / Convex optimization
- Probability and Statistics

  . . .

- Differential Equations, PDEs

What Kinds of Tools?

- Abstract interpreters, e.g. ASTREE
- Code contracts, e.g., Dafny
- SAT/SMT solvers, e.g., Z3
- Model Checkers, e.g., SPIN, Java PathFinder, . . .
- Interactive Theorem Provers, e.g., Isabelle, Coq, HOL4, PVS, . . .
- Runtime Monitoring, e.g., JavaMOP

Aims of this Course

Theoretical: The fundamental mathematics behind proving a program correct by reducing it to logic

- Static analysis using abstraction; abstract interpretations.
- Floyd-Hoare logic; contracts; pre/post conditions; inductive invariants verification conditions, strongest postcondition, weakest precondition
- Contract-based programming for both sequential and concurrent programs; developing software using contracts.
- Model checking; representing programs as transition systems; systematic exploration of execution paths
- Understand the advantages and limitations of formal methods, and some workarounds

Aims of this Course

Practical: Use existing tools or build your own, e.g.,

- Build static analysis algorithms for some analysis problems using abstraction, and learn to use some abstract-interpretation tools
- Proving small programs correct using a modern program verification tool (Floyd-style)
- Use SMT solvers to solve logical constraints; understand how program verification can be done using these solvers.
- Learn contract based programming using Dafny; use to generate unit tests and proofs
- Using model checkers to find errors in programs

Topics covered in the course

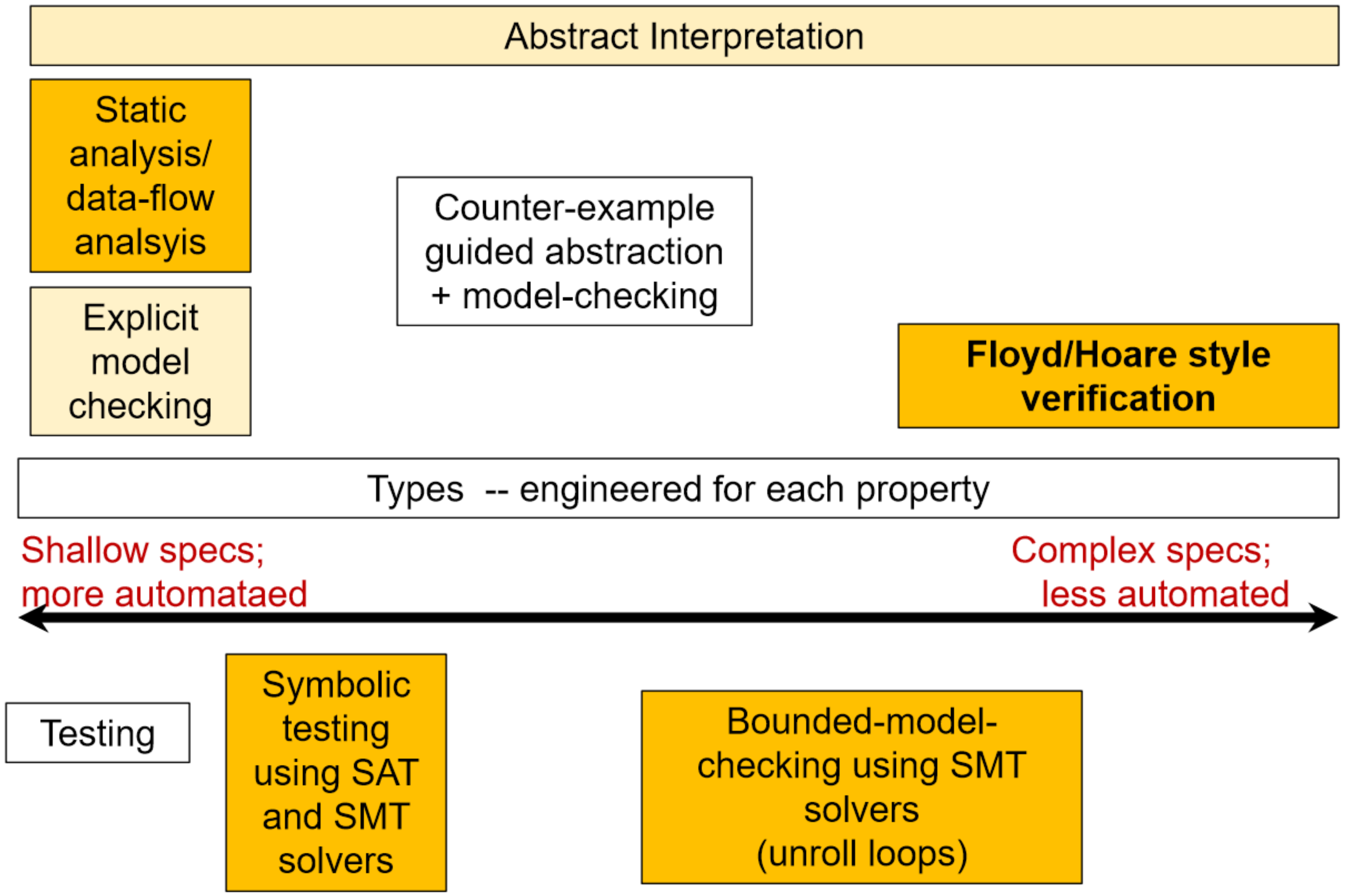
Core topics:

- Introduction to logic and proofs
- Relation between program semantics and analysis
- Abstract interpretation
- Contract languages and constraint checking
- Model checking

Cross-cutting topics:

- Nondeterministic and concurrent systems
- Probabilistic and statistical methods
- Program synthesis (if time permits)

Landscape of Formal Approaches


Abstract Interpretation
Static analysis/ data-flow analsyis
Explicit model checking
Counter-example guided abstraction + model-checking
Floyd/Hoare style verification
Types -- engineered for each property
Shallow specs; more automataed
Complex specs; less automated
Testing
Symbolic testing using SAT and SMT solvers
Bounded-model-checking using SMT solvers (unroll loops)

Some Formal Methods Not Covered

- Type systems (see CS 421 and Spring 2022 CS-598)
- Process algebras (see CS 524)
- Testing/Symbolic Execution (CS 527, CS 498)
- Theorem proving (Spring 22 CS-598, sometimes CS477)
- Computational Learning for Verification (Fall 2021 CS-598)
- Formal methods for Cyber-physical systems (ECE/CS 584)
- Term-rewriting systems (CS 476, CS 522)
- Runtime verification (CS 522, CS 598)
- Compositional methods such as assume-rely guarantees
- Methods to derive programs from models (e.g., synthesis)
- Integration of multiple formal methods (various graduate-level formal methods courses)

The Software Challenge

- Software failures were estimated to cost the US economy about $60 billion* annually [NIST 2002]
  - Improvements in software testing infrastructure may save one-third of this cost.
- One-third to one half of the total cost of ownership is preparing for, or recovering from, *failures*.
- A study of safety-critical systems (prior to 2001) reported 1,100 deaths attributable to computer error.

* Estimated to over 200 billion in 2015

What are Formal Methods?

- Mathematical formalisms used to specify and reason about software.
  - Consist of a large collection of techniques varying in complexity and usability.
- Not all methods are equally used.
  - Light-weight methods such as design by contract are common
- More rigorous methods used design and maintenance of safety critical systems.

Motivation for Formal Methods

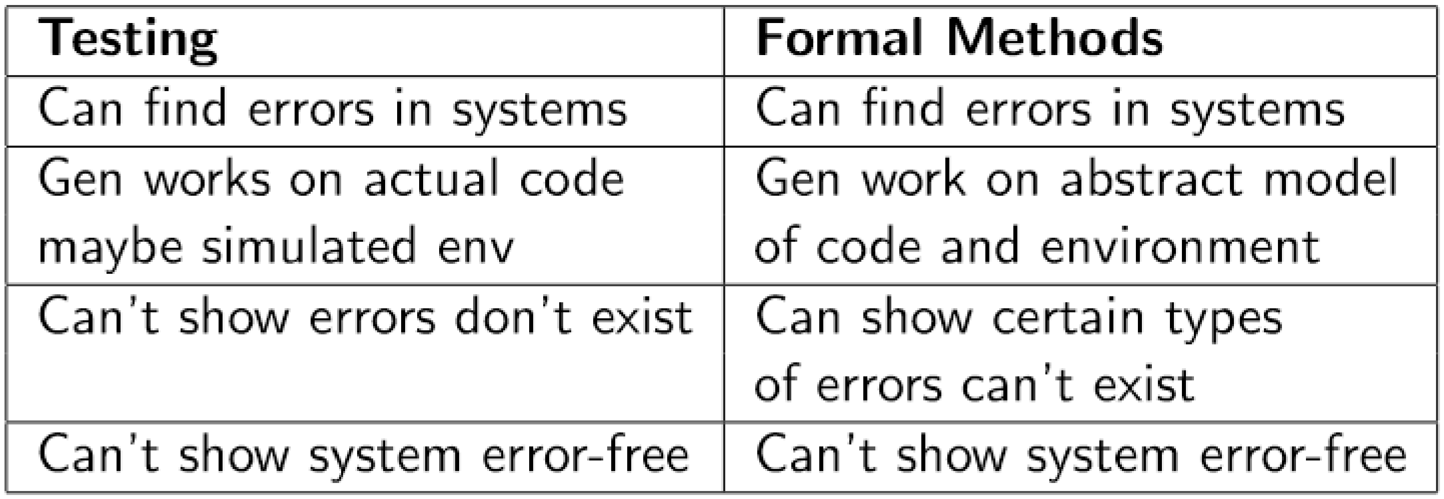
Introduce **rigor** to improve the software development process for

- Improved code structure
- Increased maintainability
- Reduced errors

Usually formal methods are applied to only some part of a large software system (perhaps 10%).

Why Formal Methods?

- To **catch bugs**
- To **eliminate whole classes of errors**
- Testing vs formal methods:



Testing	**Formal Methods**
Can find errors in systems	Can find errors in systems
Gen works on actual code maybe simulated env	Gen work on abstract model of code and environment
Can't show errors don't exist	Can show certain types of errors can't exist
Can't show system error-free	Can't show system error-free

Limitations of Formal Methods

- From CS theory we know that most interesting properties about the program (e.g. its termination) are <u>undecidable</u> (see Rice's theorem)

- But we choose not to settle for the defeat: practical formal methods are about <u>simplifying</u> assumptions, making <u>tractable</u> specifications, and automating <u>special cases</u> that cover many *real-world* programs

- ***While not all programs can be analyzed, many programs that developers care about can!***

Limitations of Formal Methods

- No method proves a program “*correct” !*
  - cf. *Karl Popper: “Science is about refutation, not proofs.”*
  - *How a program works and our model of how it works are not the same.*
- Formal methods can only establish some part of the system is correct with respect to a specification under some assumptions.
- Specifications are often incomplete and may be erroneous.
- From the statistical world: “*All models are wrong, but some are useful*” (George Box)

Limitations of Different Formal Method Techniques

- *Static Methods*: may result in falsely identifying errors.
- *Test generation*: generally incomplete.
- *Model checking*: limitations on how large a state space can be handled.
- *Theorem proving*: generally requires human expert intervention.

Formal Methods "Levels"

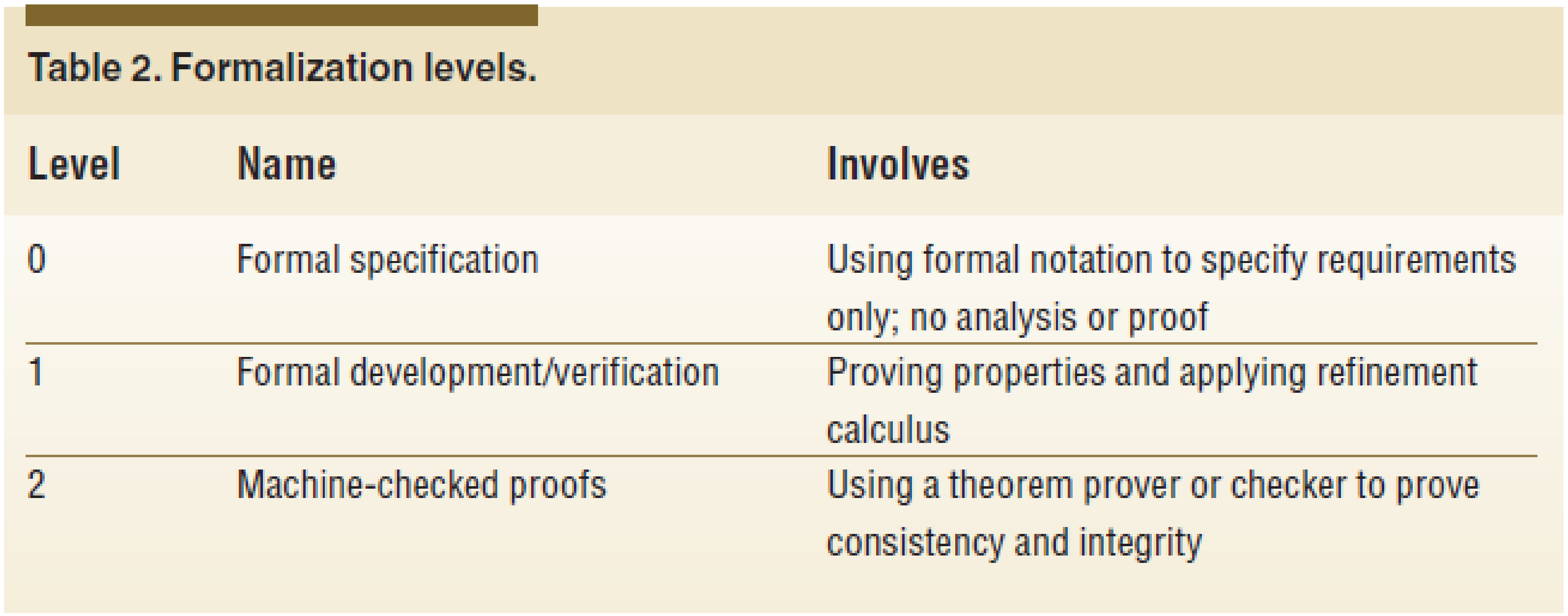


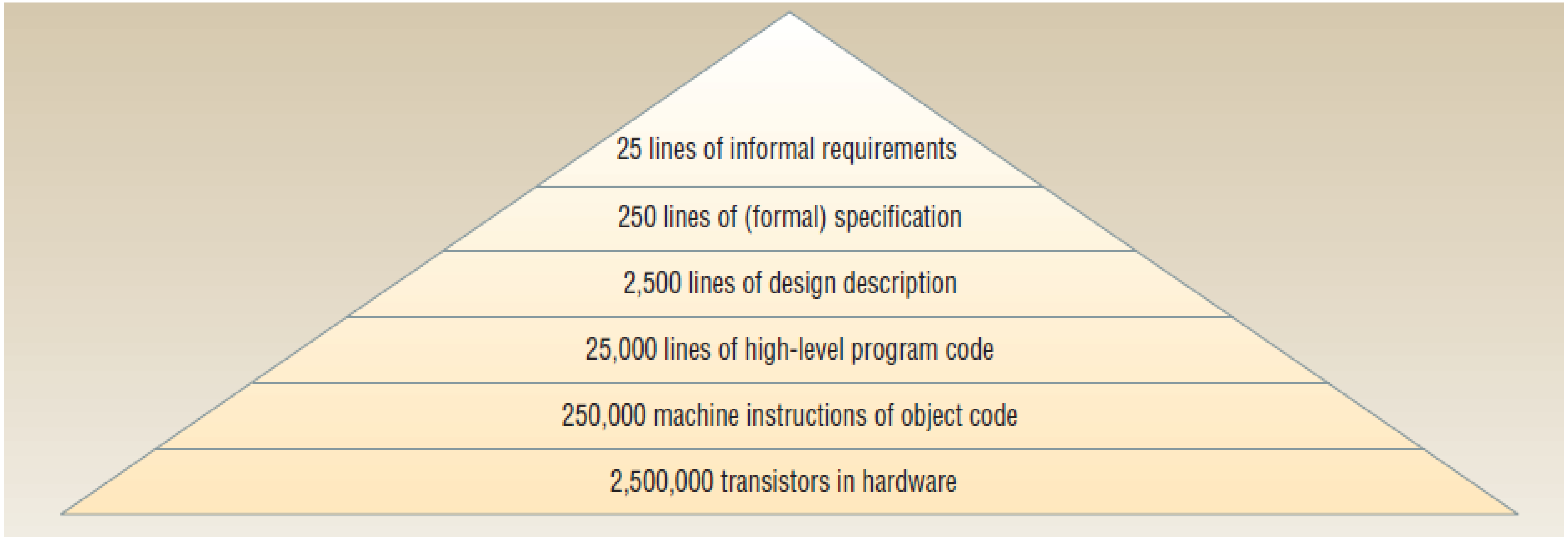
Table 2. Formalization levels.

Level	Name	Involves
0	Formal specification	Using formal notation to specify requirements only; no analysis or proof
1	Formal development/verification	Proving properties and applying refinement calculus
2	Machine-checked proofs	Using a theorem prover or checker to prove consistency and integrity

Source: Bownen and Hinchey , "Ten Commandments of Formal Methods .. Ten Years Later, " IEEE Computer 2006

Getting the Design Right

The size explodes as we proceed down the development implementation hierarchy (hypothetical numbers):

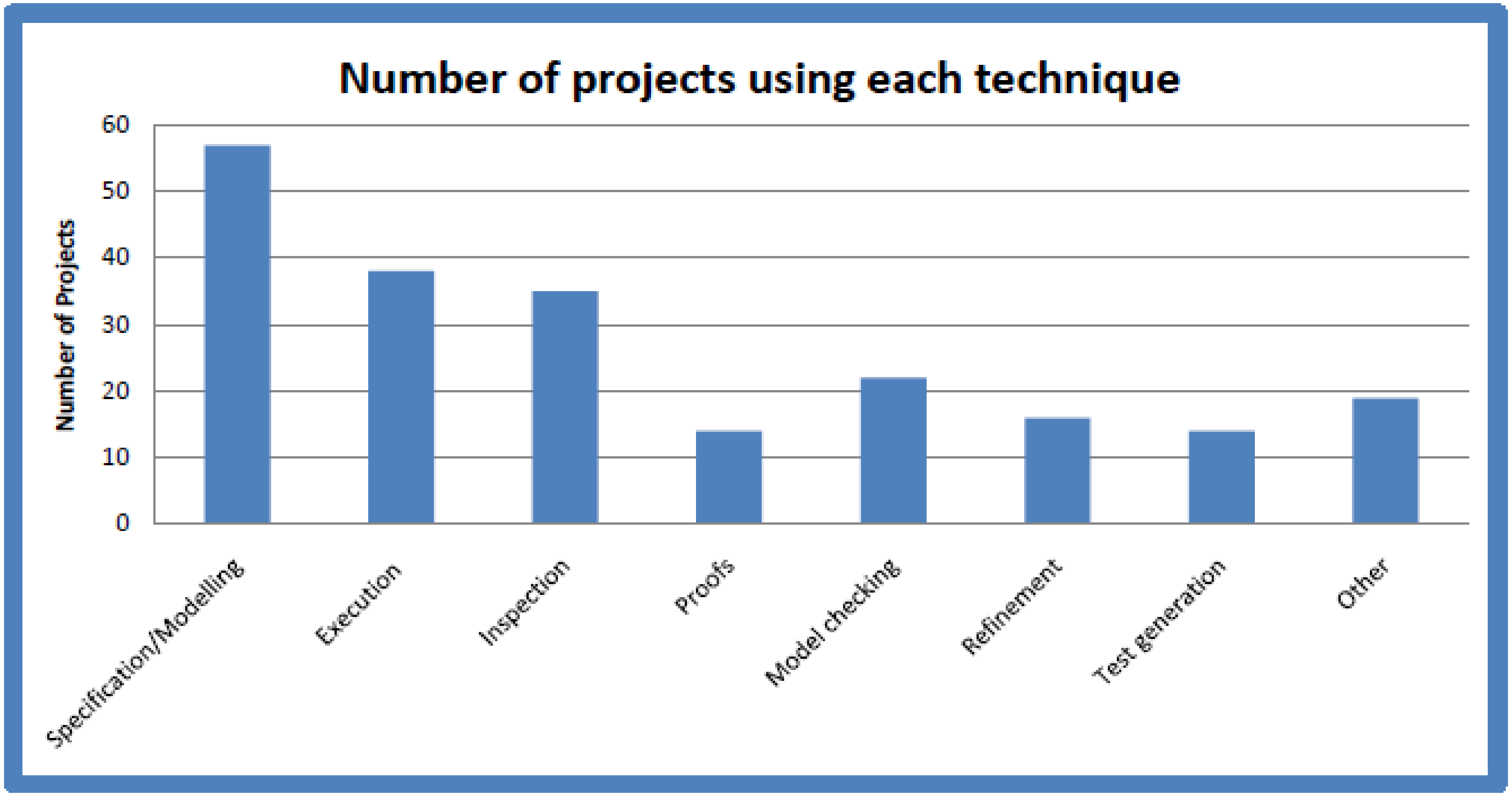

25 lines of informal requirements
250 lines of (formal) specification
2,500 lines of design description
25,000 lines of high-level program code
250,000 machine instructions of object code
2,500,000 transistors in hardware

Source: Bownen and Hinchey , "Ten Commandments of Formal Methods .. Ten Years Later, " IEEE Computer 2006

Notes on Kinds of Formal Methods Used

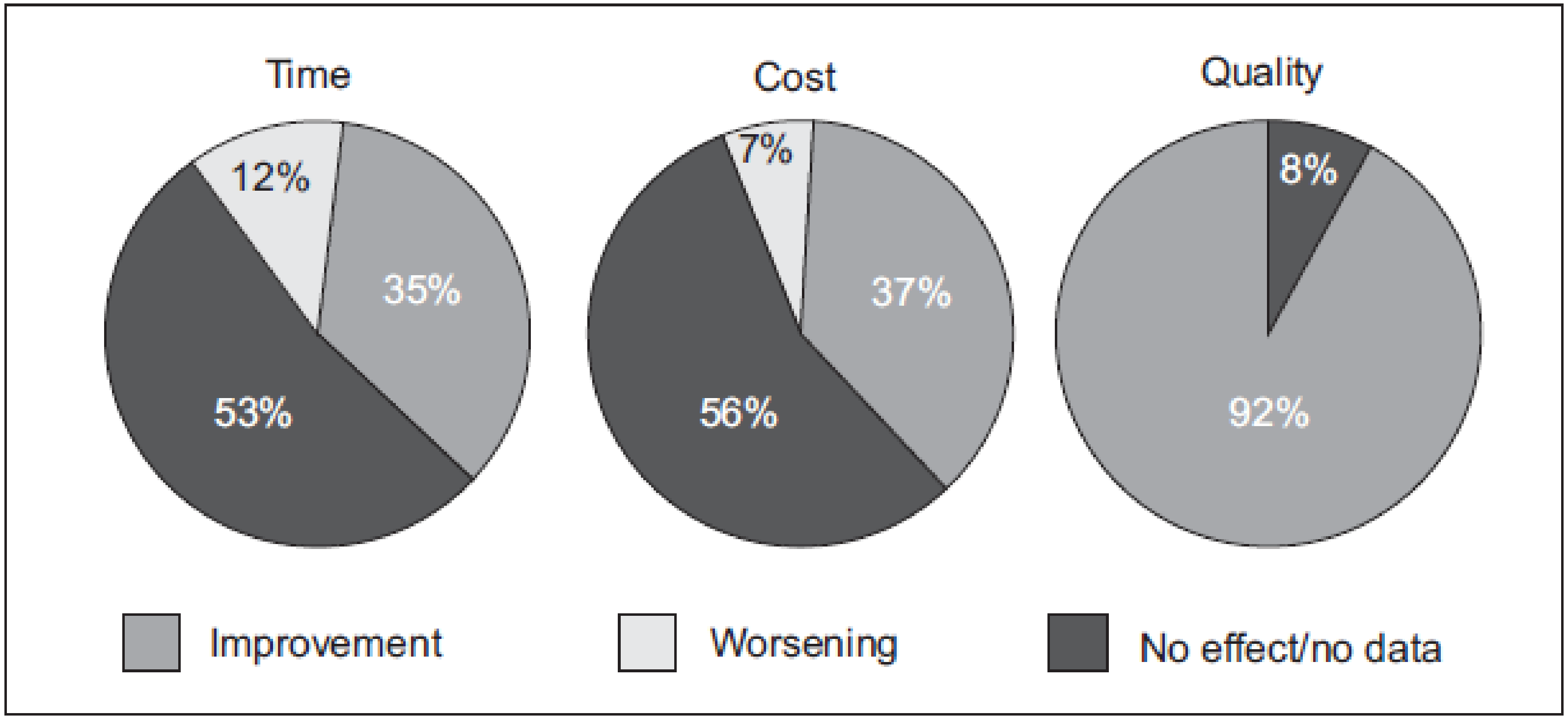
- Many projects use more than one technique.
- Different techniques applied to different parts of the code.
- Formal methods often applied only to some parts of a large code (e.g. judged safety-critical).

Types of Formal Methods Used



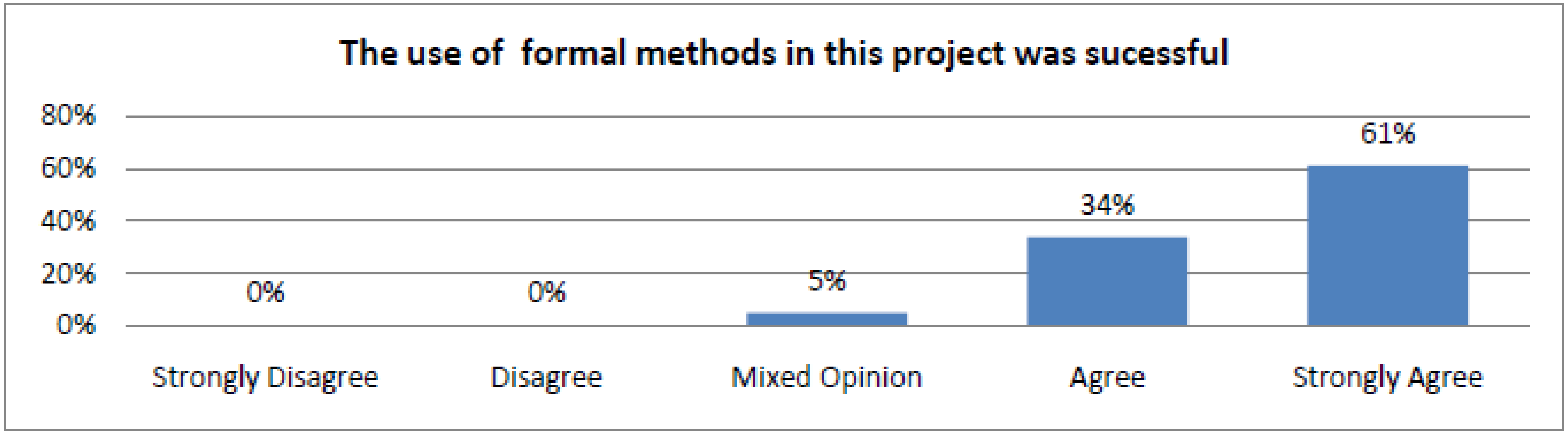
Source: Woodcock, et al. "Formal Methods: Practice and Experience", ACM Computing Surveys, 2009

Perceived Impact



Source: Woodcock, et al. "Formal Methods: Practice and Experience", ACM Computing Surveys, 2009

Use of Formal Methods Judged Successful?



Source: Woodcock, et al. “Formal Methods: Practice and Experience”, ACM Computing Surveys, 2009

Caveat: These projects reported using Formal Methods

You may be less likely to report use if you thought it was a failure.

Some Case Studies

- *Woodcock et al. 2009* highlights a number of industrial projects using formal methods. Software projects include:
  - Railway signaling and train control
  - Smart card based electronic cash system
  - Microcode for the AAMP5 microprocessor
  - Airbus controllers: Flight Control Secondary Computer and the Electric Load Management Unit.
  - Maeslant Kering Storm Surge Barrier

Railway signaling and train control

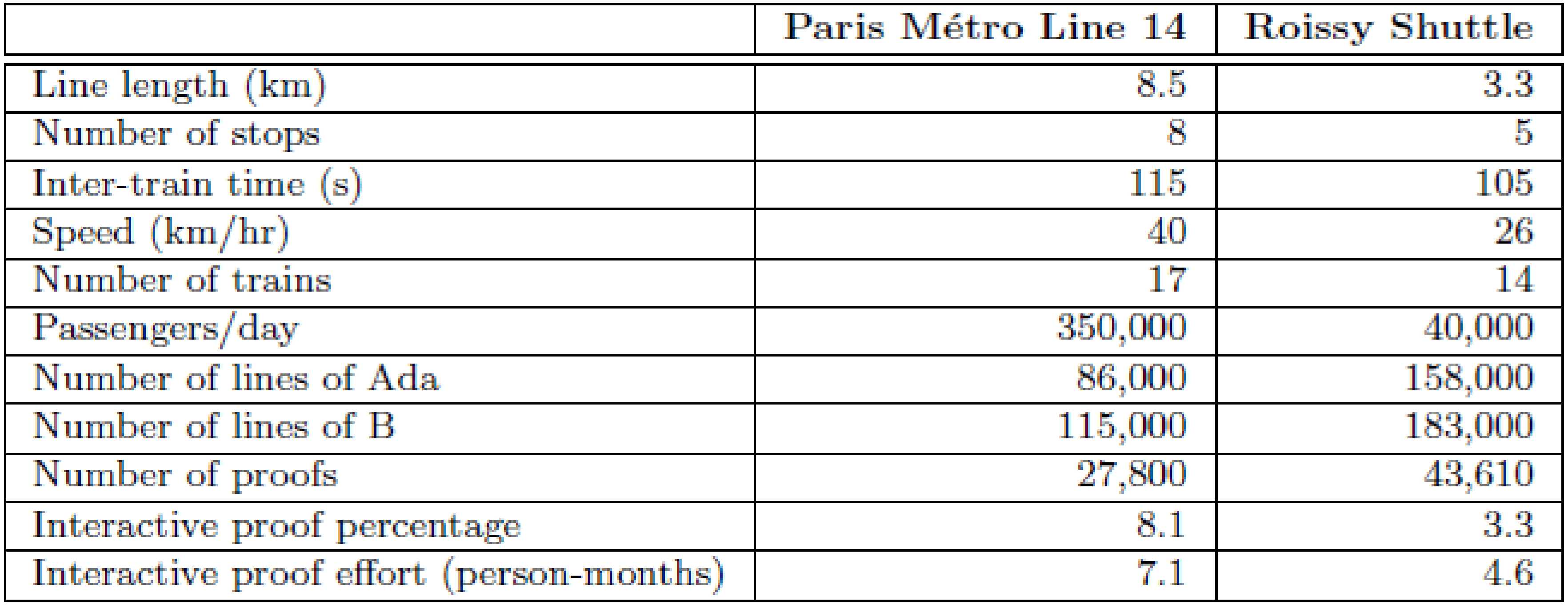
- RATP 1998 (Parisian regional rail system): computerized signaling system
  - Increase network traffic by 25%
  - Preserve safety levels
- SACEM system with embedded hardware and software controls the speed of trains on RER Line A

http://www.flickr.com/photos/bindonlane/5802050248/

RER Line A

- 21,000 lines of Modula-2 code:
  - 63% regarded as safety critical
  - Subjected to formal specification and verification
- Specification written in B and proofs done interactively using automatically generated verification conditions.
- Validation took 100 person years.
- Method used again on other lines..
- Unit tests replaced by global tests, all successful.

Paris Metro Railway Signaling



	Paris Métro Line 14	Roissy Shuttle
Line length (km)	8.5	3.3
Number of stops	8	5
Inter-train time (s)	115	105
Speed (km/hr)	40	26
Number of trains	17	14
Passengers/day	350,000	40,000
Number of lines of Ada	86,000	158,000
Number of lines of B	115,000	183,000
Number of proofs	27,800	43,610
Interactive proof percentage	8.1	3.3
Interactive proof effort (person-months)	7.1	4.6

Airbus

- Used SCADE tool suite to manage development of controllers, including Flight Control Secondary Computer and the Electric Load Management Unit.
- Reported significant decrease in coding errors.
- Shorter requirements changes, improved traceability.
- 70% of code generated automatically.

http://www.flickr.com/photos/peterpearson/2683260198/

Boeing 777

- Problems with databus and flight management software delay assembly and integration of fly-by-wire system by more than one year
- Certified to be safe in April 1995
- Total development cost 3 billion; software integration and validation costs were about one-third.

Boeing 777 (WSJ Analysis)

- A Boeing 777 plane operated by Malaysian Airlines, flying from Perth to Kuala Lumpur in August 2005, experiences problems: The plane suddenly zoomed up 3000 feet. The pilot's efforts at gaining manual control succeeded after a physical struggle, and the passengers were safely flown back to Australia.
- Cause: Defective software provided incorrect data about the plane's speed and accelaration.
- "Plane makers are accustomed to testing metals and plastics under almost every conceivable kind of extreme stress, but it's impossible to run a big computer program through every scenario to detect bugs that invariably crop up."
  ". . . problems in aviation software stem not from bugs in code of a single program but rather from the interaction between two different parts of a plane's computer system."

```
A problem has been detect and Windows has been shut down to prevent damage
to your computer.

THREAD_STUCK_IN_DEVICE_DRIVER

If this is the first time you've seen this Stop error screen,
restart your computer. If this screen appears again, follow these steps:

Check to make sure any new hardware or software is properly installed.
If this is a new installation, ask your hardware or software manufacturer
for any Windows updates you might need.

If problems continue, disable or remove any newly installed hardware
or software. Disable BIOS memory options such as caching or shadowing.
If you need to use Safe Mode to remove or disable components, restar
your computer, press F8 to select Advanced Startup options, and then
select Safe Mode.

Technical information:

*** STOP: 0x000000EA (0x00000000, 0x00000000)
```

And for everyone else…

- "Things like even software verification, this has been the Holy Grail of computer science for many decades but now in some very key areas, for example, driver verification we're building tools that can do actual proof about the software and how it works in order to guarantee the reliability." **Bill Gates, April 18, 2002. WinHec Keynote**
- https://www.microsoft.com/en-us/research/project/slam/

More Reports from the Trenches

- **A Few Billion Lines of Code Later: Using Static Analysis to Find Bugs in the Real World**
  Dawson Engler (Stanford/Coverity) et al. CACM 2010:
  https://cacm.acm.org/magazines/2010/2/69354-a-few-billion-lines-of-code-later/fulltext

- **Continuous Reasoning: Scaling the Impact of Formal Methods**
  Peter O’Hearn (UCLondon/Facebook), 2018:
  https://research.fb.com/publications/continuous-reasoning-scaling-the-impact-of-formal-methods/

- **Amazon Relies on Formal Methods for the Security of AWS**
  Keynote by Byron Cook (ex UCLondon/Amazon), FLoC 2018:
  https://blog.adacore.com/amazon-relies-on-formal-methods-for-the-security-of-aws

- **Why Aren't There More Programming Languages Startups?**
  Jean Yang (ex CMU/Akita), blog 2021:
  https://www.akitasoftware.com/blog-posts/why-arent-there-more-programming-languages-startups

Course Administration

CS 477

This course gives an overview of mathematical models, languages, and methods for software specification, development, and verification.

- 3 undergraduate hours.
- 4 graduate hours.
- Prerequisites: CS 225; CS 374 or MATH 414.
- Useful: CS 421; CS 475

Tentative Topics

- Background and Predicate Logic
- Operational Program Semantics
- Static Analysis and Abstract Interpretation
- First Order Logic, Hoare Logic and Code Contracts
- Model Checking
- Advanced Topics

CS 477: Course Details

Instructor:
Prof. Sasa Misailovic
4110 Siebel Center

Office hours:
TBD
by appointment

Email: *misailo@illinois.edu*

Teaching Assistant:
Vimuth Fernando
3107 Siebel Center

Office hours:
Wed 2-4pm
or by appointment

Email: wvf2@*illinois.edu*

Course Website:
https://courses.grainger.illinois.edu/cs477/fa2021/

Grading Scheme

Should be done individually!

For undergraduates registered for 3 credits:

- Four or five homework assignments include machine problems: 75% of the grade
- A final homework: 25% of the grade

For graduate students registered for 4 credits:

- Homework assignments 56.25%
- Final homework 18.75%
- Group term project 25%

Assignments

- Each of the 5 assignments will consist of a theoretical part (proofs) and a practical part (code).
- Submit individual reports. Can discuss with other students but must list all the people.
- Depending on the project, between 1- and 2-week time will be given to solve the homework.
    - ***Dates TBD (first one next week)***
- Late submission will result in a reduced score: 33% penalty per day (unless a valid health reason – contact
- Homework must be *submitted electronically*
- Scanned (or photographed) answers are OK as long as the handwriting is legible.
    - *We'll let you know the submission infrastructure*

Semester Project (for Grads; 4 credit)

- Projects may be done <u>individually</u> or in <u>groups of 2</u>.
- An abstract (1 to 2 pages) describing the project is due before the Spring Break.
- The project must be submitted on the last day of class to be graded.
- Involves either
  - Reading up a set of papers, and writing a report, or
  - Programming a particular technique or developing software using contracts, and submitting a write-up

Sample Project

Develop a memory management system that hands out chunks of memory to processes ensuring no overlap.

Clear simple specification.

Implementation using linked lists.

Specification using separation logic.
Prove correct using VCC/VCDryad/DAFNY.

Tentative Grading Scale

Grade	Score
A	[93,100]
A-	[90,93)
B+	[87,90)
B	[83,87)
B-	[80,83)
C+	[77,80)
C	[73,77)
C-	[70,73)
D	[55,70)
F	[0,55)

Grades may be curved if the spread of points is too high
The curve *may* be separate for undergrads and grads

Academic Integrity

- Students in the class are expected to abide by the University's Student Code. Infractions will be governed by standard policies of the Department of Computer Science.
- Please refer to the University Policy on Academic Integrity and the CS Department's Honor Code.

Course References

No required textbook

- But many existing resources available (+ online)
- May do “inverse lectures” from time to time

Reference Books:

- Introduction to Static Analysis, Rival and Yi, MIT Press 2021
- Principles of Static Analysis: F. Nielson, H. Nielson, Hankin, Springer 2005
- The Calculus of Computation: Bradley and Manna, Springer 2007
- Principles of Model Checking: Baier and Katoen, MIT Press 2008
- Tutorial on Static Inference of Numeric Invariants by Abstract Interpretation, Mine Now Publishers 2017

Many more references on the website!