

Introduction

Verifying learning-enabled cyberphysical systems

August 26th 2025, ECEB 3013

Sayan Mitra

CSL 266

mitras@illinois.edu

@Mitrasayn

What is verification?

Definition. *Verification* *is the action of demonstrating or proving statements to be true by means of evidence. OED*

This class:

statement = formal specifications (or requirements) about learning-enabled cyber-physical systems

evidence = mathematical proof

Learning-enabled cyber-physical systems

A cyber-physical system is a computer system monitoring or controlling a physical process.

A learning-enabled system uses machine learning models for perception or control

Examples: drone delivery, autonomous car, smart electric grid, insulin pump



The number of possible behaviors of such systems is usually *uncountably infinite*

Formal Specifications or Requirements

Requirements are statements about behaviors of a system

Drone visits waypoints while avoiding collisions

The vehicle stays within the lanes and maintains distance to leading car

Insulin pump maintains blood glucose level to within the prescribed range

Testing: evaluates requirements on a finite number of behaviors

Testing can show presence of bugs but not their absence ---
E. Dijkstra

Verification: aims to prove requirements over all behaviors



Failures in real world: Uber Fatality (Tempe, 2018)

An Uber autonomous test vehicle struck and killed **Elaine Herzberg** pushing a bicycle across a dark road.

The perception system had detected her six seconds before impact — but kept switching labels: first a vehicle, then a bicycle, then an “unknown object.” Because of this uncertainty, the system never triggered emergency braking.

Uber had disabled the car’s automatic emergency braking to avoid “jerky” rides, leaving only the human backup driver, who was looking at her phone.

Without verified fail-safes and robust uncertainty handling, autonomy can *see* a hazard but still fail to act.



Failures in real world: Toyota Unintended Acceleration (2009–11)

Drivers reported sudden, uncontrollable acceleration in Toyota vehicles.

Lawsuits and investigations revealed evidence of software flaws in the electronic throttle control: **stack overflows, race conditions, and inadequate fail-safes.**

Hundreds of crashes and fatalities were attributed to this failure mode, and Toyota paid billions in settlements.

NASA's technical team found that **memory corruption could disable safety tasks**, leaving the throttle stuck open.

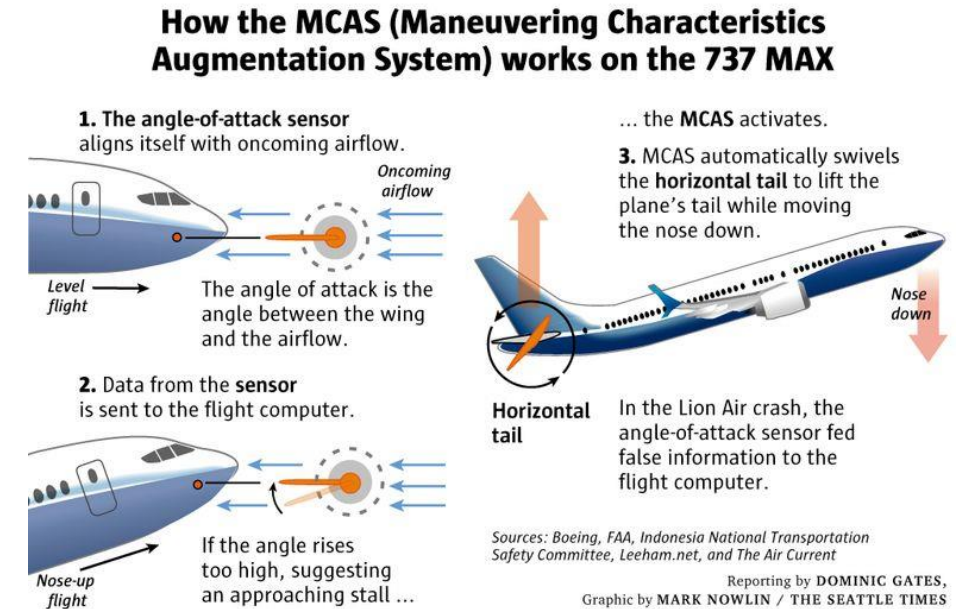
Failures in real world: Boeing 737 MAX – MCAS (2018–19)

Boeing added the MCAS system to the 737 MAX to automatically push the nose down if it sensed a high angle of attack, preventing stalls.

The flaw was that MCAS relied on a *single* angle-of-attack sensor. When that sensor failed, the system kept forcing the nose down, while the pilots, unaware of MCAS's authority, fought back.

Two crashes in Indonesia and Ethiopia killed 346 people and grounded the global fleet for almost two years.

Investigators pointed not only to the sensor design but also to Boeing's **assumptions** about pilot response times and training.



Successes of Verification

Hardware verification now standard in EDA tools from Synopsys, Cadence, etc.

[SLAM](#) tool from MSR routinely used for verification of Device Drivers at Microsoft:

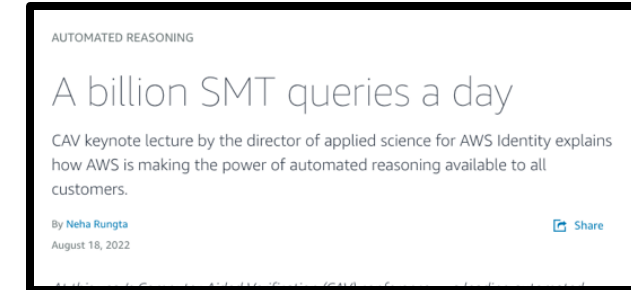
[AMAZON](#) AWS developers write proofs using CBMC and other Automated reasoning tools

[Google](#) runs static analysis tools on their entire codebase

Formal modeling and analysis is becoming part of certification process for avionics (e.g., ASTREE); DO-333 supplement of DO-178C identifies aspects of airworthiness certification that pertains to software using *formal methods*

Coverity, Galois, SRI, and others

LE-CPS, Automotive, and manufacturing ...

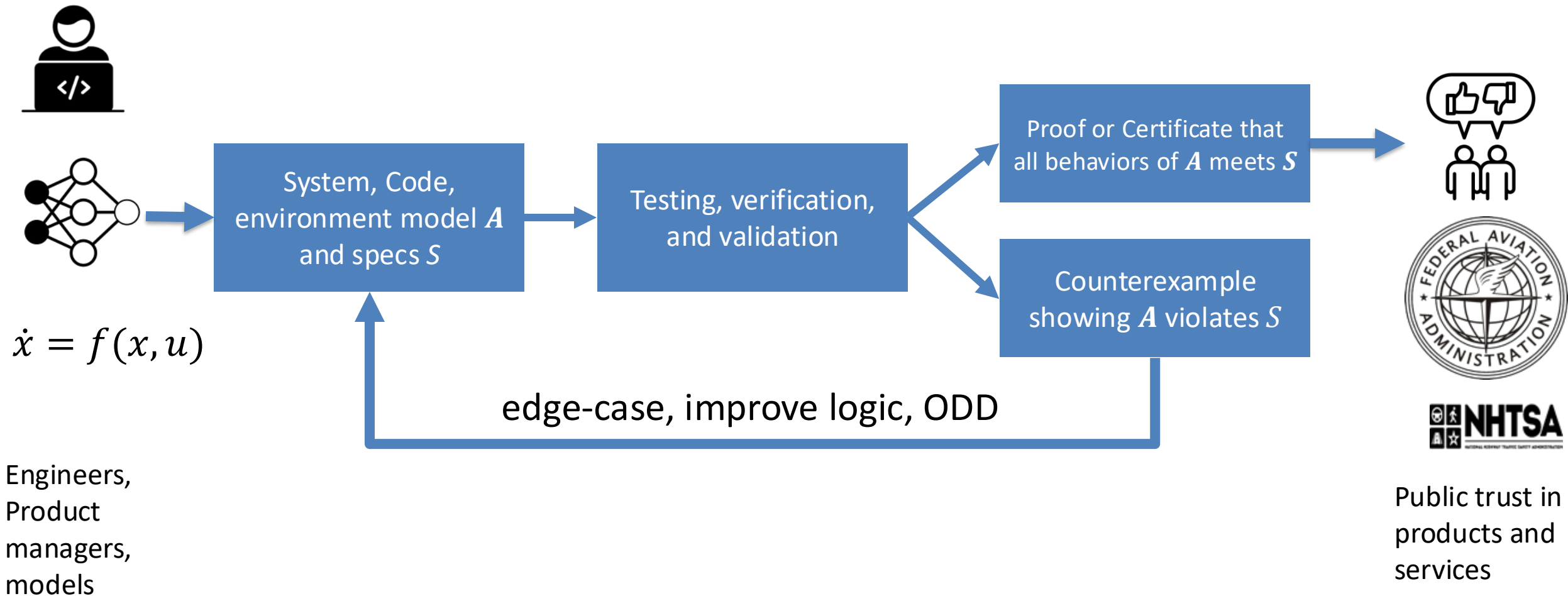


Sadowski, et al. Lessons from Building Static Analysis Tools at Google. CACM, 18.

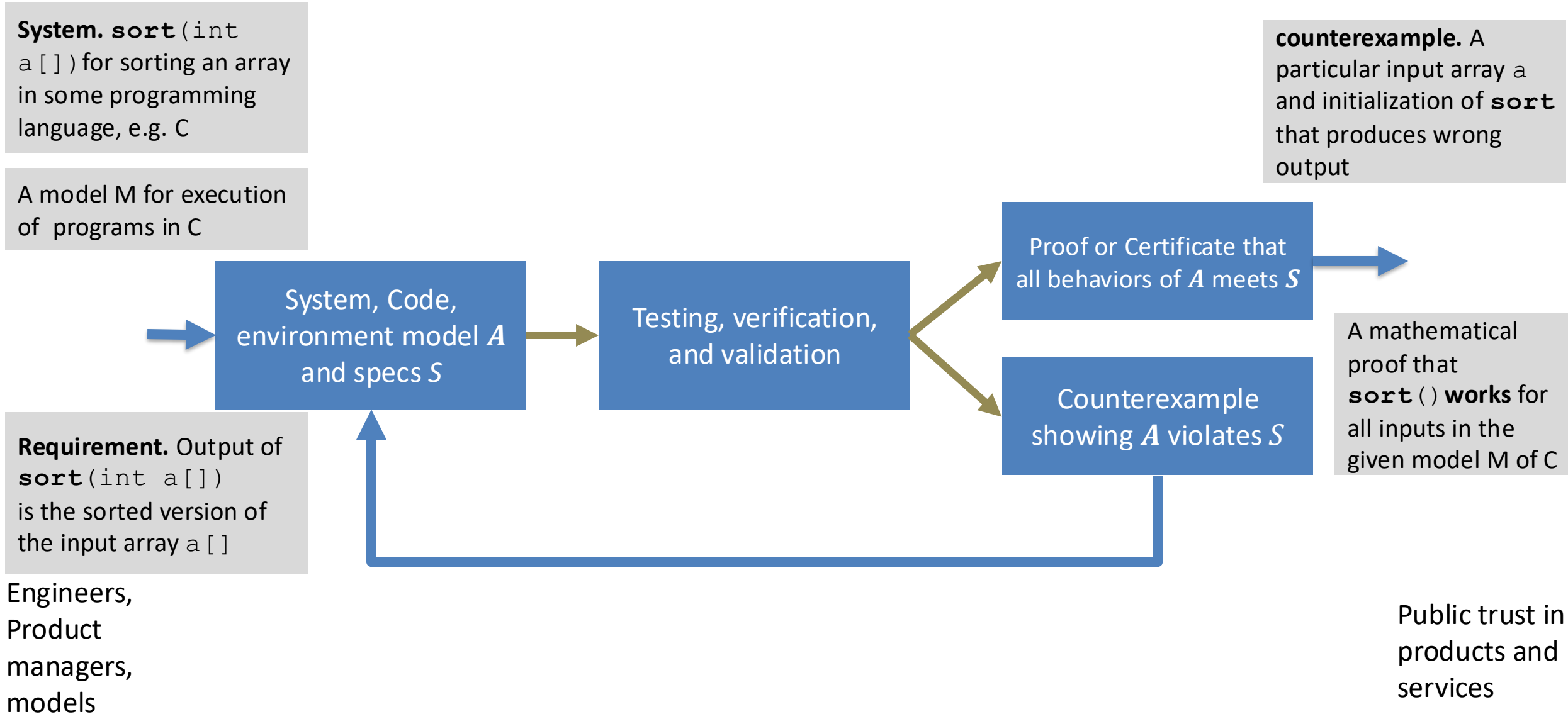
Chong, et al. Code-level model checking in the software development workflow. ICSE, 20.

O'Hearn. Continuous reasoning: Scaling the impact of formal methods. LICS '18.

Verification in Engineering Ecosystem



Verification in the engineering ecosystem



The Reach of Formal Verification

Formal Verification Covers and connects ideas in CS, control, ML

Turing Awards: Lamport (2014), Clarke, Sifakis & Emerson (2008), Pnueli (1997), Lampson (1992), Milner (1991), Hoare (1980), Dijkstra (1972) ...

ACM Doctoral Dissertation Award: [Chuchu Fan](#) (2020) alumni of this class

Vibrant community: [CAV](#), [TACAS](#), ATVA, PLDI (programming languages),

HSCC, EMSOFT, ICCPS (hybrid and cyber-physical systems)

Special tracks in Robotics (ICRA, RSS), automatic control

Ideas permeating into AI and machine learning (Neural network verification)

Faculty and research positions: Alumni of this course are professors at WashU, Vanderbilt, UNC Chapel Hill, MIT, Stony Brook, and researchers at Waymo, Tesla, Amazon, Toyota, Boeing

Course Objectives

- Learn to write verification algorithms and tools
- Understand fundamental limits of creating such tools
- Learn how to create models of of LE-CPS and the trade-offs in different modeling paradigms
- Execute a research project in formal verification

Learning objectives

1. Learn a unified mathematical language for **modeling state machines** combining automata (CS) and differential equations (ODEs)
2. Learn fundamental concepts such as **invariance, reachability, contraction, stability** and how they cut across automata, ODEs, and ML models
3. Learn powerful algorithms and tools for reasoning about invariance, stability, contraction
4. Jumpstart research

discrete and continuous states, valuations, equilibria, programs, solutions, trajectories, executions, observables, measurements, composition, abstraction

Invariance, safety, induction, reachability, liveness, Lyapunov theory, ranking functions, dwell time, CEGAR, SMT, temporal logics,

SMT solvers, CVXOPT, theorem provers, CROWN, Verse


semester-long project, feedback, presentation, hardware, software, and data resources

How the course works

ADMINISTRIVIA

Fall 2025 Edition

- <https://mitras.ece.illinois.edu/ECE584/>
- Lectures TR 12:30 – 1:50
- ECEB: 3013
- [Textbook](#)
- Homework: 4-5 sets. Analysis and some coding
- Midterm Exam --- in class **Oct 21**
- [Project](#): Semester long research project, becomes basis for publication



Fall 2025

ECE/CS 584: Embedded & Cyberphysical System Verification

Home
Administrivia
Project Jumpstart
Schedule
Problem Sets
Resources
Archives 2025

Course Administrivia

Verification is the art and science of establishing that a given system meets a given set of requirements. For example, an assertion such as "an adaptive cruise control system in a car does not lead to any collisions."

Cyberphysical systems (CPS) combine software components with sensors, actuators & communication. From driverless cars to air-traffic control systems, medical devices to manufacturing robots, many new and old engineering systems fit the description of what are now called cyberphysical systems.

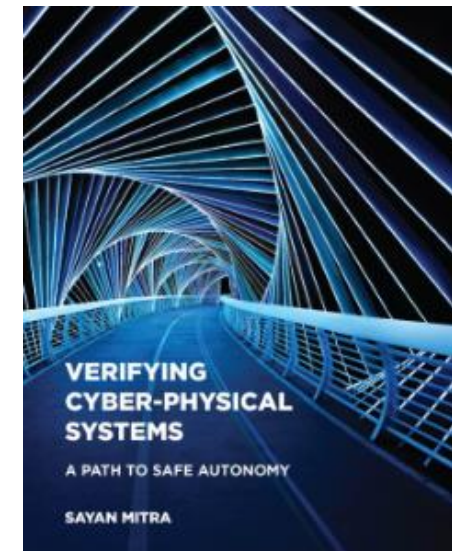
Many CPS systems are mission-critical, so verification is essential for their applications. In particular, the **use of machine learning and AI in these systems** will make the verification of these systems even more challenging.

In this course, you will:

1. Learn to create mathematical models of cyberphysical systems
2. Learn the fundamental principles of verification
3. Gain experience in using powerful verification tools (**model checkers, SMT solvers, theorem provers, and neural network verifiers**)
4. Read and discuss recent ideas from research papers
5. Gain in experience in formulating research problems in the area of cyberphysical systems/formal verification. This course especially highlights the **recent developments in the formal verification of machine-learning-enabled systems.**

People Involved

Sayan Mitra,
Lecturer
mitras at illinois.edu
Phone: 333-7824
Office: CSL 266



Expectations and grading

Preparation: Do the readings, review notes, and come with questions.

Engagement: Participate actively in discussions and project reviews.

Integrity: Collaboration is encouraged, but all submitted work must be your own.

Professionalism: Be respectful, give constructive feedback, and contribute fairly in group work.

Research mindset: Treat assignments and the project as opportunities to explore publishable ideas, not just homework.

Individual Homework (4-5 sets): 40%

- Problems will involve some coding and pencil paper proofs

Midterm 15 %

Project in teams of 2: 40%

- Proposal: 5%
- Midterm presentation: 5%
- Final presentation (EOS): 15%
- Report (EOS): 15%
- Code and documentation: 5%

Participation: 5%

- Class participation
- Feedback on projects
- New problem suggestions and solutions

COURSE PROJECT

- You will work on a semester-long research project building a verification system. Often the projects become publishable.
- Good project will require ~6-8 hours/week — higher towards the end of the semester.
- Start early, and get frequent feedback. This week: Read and start here
- Projects can either be individual or be done in a team of 2 people. If you are sure that you want to work in a larger team (3 or more people), you can make a case and we can consider.
- Projects will be graded based on
 - soundness of claims,
 - significance (novelty, impact, harness),
 - presentation quality, and
 - effort
- Advice
 - Choose a high-risk project that if success will make you proud
 - Clearly define the problem ASAP
 - Get feedback frequently

Course Staff

Prof. Sayan Mitra

CSL 266

Office Hour:

TA: Chenxi Ji

CSL 247

Office Hour:

TA's role: Grading HWs, exams, explain lecture/HW materials

Provide constructive feedback on project ideas

Preliminaries

DISCRETE MODELS

Outline

- State machines or Automata
- Semantics: executions, reachable states
- Examples
- Invariance
 - Reachable state
 - Inductive invariance

Models: State machines or Automata

A state machine or an automaton is a discrete-time(step-by-step) mathematical model describing the evolution of a system in term of its states

Example 1:

x: Int = 5

While True // step

if $x \% 2 == 0$ **then** $x = x / 2$ **else** $x = 3x + 1$

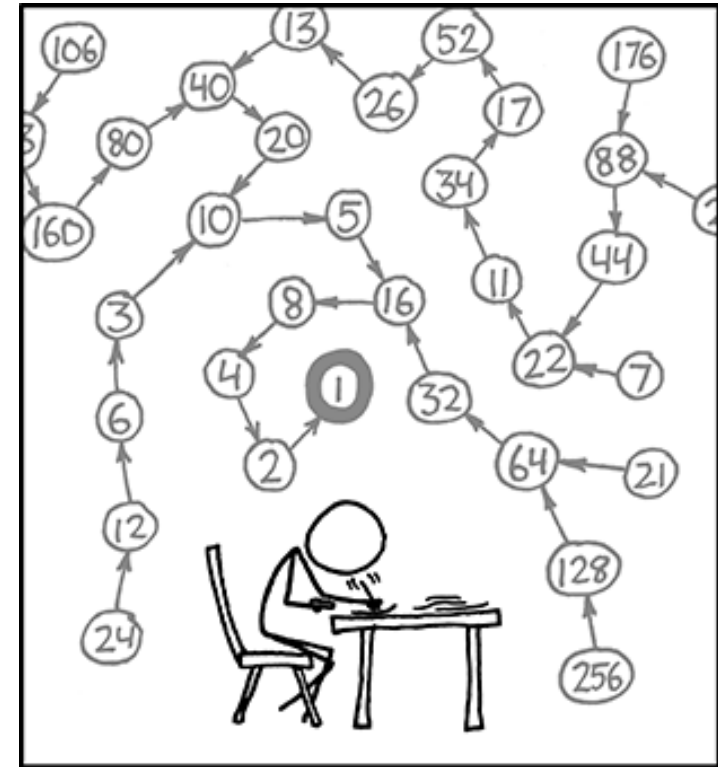
Example run: $x = 5, x=16, x=8, x=4, 2, 1, 4, 2, 1, 4, 2, 1$

Another run: $x = 6, x=3, 10, 5, 16, 8, 4, 2, 1, 4, 2, 1, \dots$

Another run: $x = 7, x=22, x=11, x=34, x=17, x=52, \dots ?$

Do all runs end in the 4,2,1 loop?

“Mathematics is not yet ready for such problems.” --- Paul Erdős



THE COLLATZ CONJECTURE STATES THAT IF YOU PICK A NUMBER, AND IF IT'S EVEN DIVIDE IT BY TWO AND IF IT'S ODD MULTIPLY IT BY THREE AND ADD ONE, AND YOU REPEAT THIS PROCEDURE LONG ENOUGH, EVENTUALLY YOUR FRIENDS WILL STOP CALLING TO SEE IF YOU WANT TO HANG OUT.

Models: State machines or Automata

A state machine or an automaton is a discrete-time(step-by-step) mathematical model describing the evolution of a system in term of its states

Example 1:

x: Int = 5

While True // step

If x % 2 == 0 **then** x = x / 2 **else** x = 3x + 1

Example run: x = 5, x=16, x=8, x=4, 2, 1, 4, 2, 1, 4, 2, 1

Another run: x = 6, x=3, 10, 5, 16, 8, 4, 2, 1, 4, 2, 1, ...

Another run: x = 7, x=22, x=11, x=34, x=17, x=52, ... ?

Do all runs end in the 4,2,1 loop?

“Mathematics is not yet ready for such problems.” ---

Paul Erdős

An **automaton** \mathcal{A} is a 4-tuple $\langle V, \Theta, A, \mathcal{D} \rangle$ where

- V is a set of names of variables; each variable $v \in V$ is associated with a type, $type(v)$
 - A **valuation** for V maps each variable name to its type
 - Set of all valuations: $val(V)$ defines the **state space** of the automaton \mathcal{A}
 - if $|val(V)|$ is finite then \mathcal{A} is a **finite state automaton**
- $\Theta \subseteq val(V)$ is the set of **initial** or **start states**
- A is a set of names of **actions**
- $\mathcal{D} \subseteq val(V) \times A \times val(V)$ is the set of **transitions**
 - a transition is a triple (u, a, u') which says “from state u , upon performing action a , the automaton **can** go to state u' ”
 - We write it as $u \rightarrow_a u'$

Collatz Automata

A state machine or an automaton is a discrete-time(step-by-step) mathematical model describing the evolution of a system in term of its states

```
Collatz:
x: Int = 5
While True           // step
    If x % 2 == 0 then x = x / 2 else x = 3x + 1
```

Example run: x = 5, x=16, x=8, x=4, 2, 1, 4, 2, 1, 4, 2, 1

Another run: x = 6, x=3, 10, 5, 16, 8, 4, 2, 1, 4, 2, 1, ...

Another run: x = 7, x=22, x=11, x=34, x=17, x=52, ... ?

An **automaton** \mathcal{A} is a 4-tuple $\langle V, \Theta, A, \mathcal{D} \rangle$ where

- $V = \{x\}$
- $type(x) = \text{Int}$
- A **valuation** for V : $\langle x \mapsto 5 \rangle$ we write this as ν
- $val(\{x\})$ **state space** $\equiv \mathbb{Z}^+$
- $\Theta = \langle x \mapsto 5 \rangle \subseteq val(\{x\})$ **start states**
- $A = \{step\}$
- $\mathcal{D} = \{ \langle x, a, x' \rangle \mid \left(x \text{ even and } x' = \frac{x}{2} \right) \text{ or } (x \text{ odd and } x' = 3x + 1), a = step \}$ **transition relation**
 - This is a special case **transition function** $\mathcal{D}: \text{Int} \rightarrow \text{Int}$

Automata, executions, and determinism

An automaton is a tuple $\mathcal{A} = \langle V, \Theta, A, \mathcal{D} \rangle$ where

- V variables; a valuation \mathbf{v} is a state and the set of all valuations $\text{val}(V)$ is the state space
- $\Theta \subseteq \text{val}(V)$ is the set of initial or start states
- A is a set of names of actions
- $\mathcal{D} \subseteq \text{val}(V) \times A \times \text{val}(V)$ is the set of transitions
 - a transition is a triple $(\mathbf{v}, a, \mathbf{v}')$ often written as $\mathbf{v} \rightarrow_a \mathbf{v}'$

An action a is said to be **enabled** at state \mathbf{v} if there exists \mathbf{v}' such that $\mathbf{v} \rightarrow_a \mathbf{v}'$

An **execution** or a run of \mathcal{A} is a finite or infinite alternating sequence of states and actions $\mathbf{v}_0, a_1, \mathbf{v}_1, a_2, \dots$ such that (1) $\mathbf{v}_0 \in \Theta$ and (2) $(\mathbf{v}_i, a_{i+1}, \mathbf{v}_{i+1}) \in \mathcal{D}$

\mathcal{A} is **deterministic** if $|\Theta| = 1$ and there is at most one next state \mathbf{v}' from any state \mathbf{v}

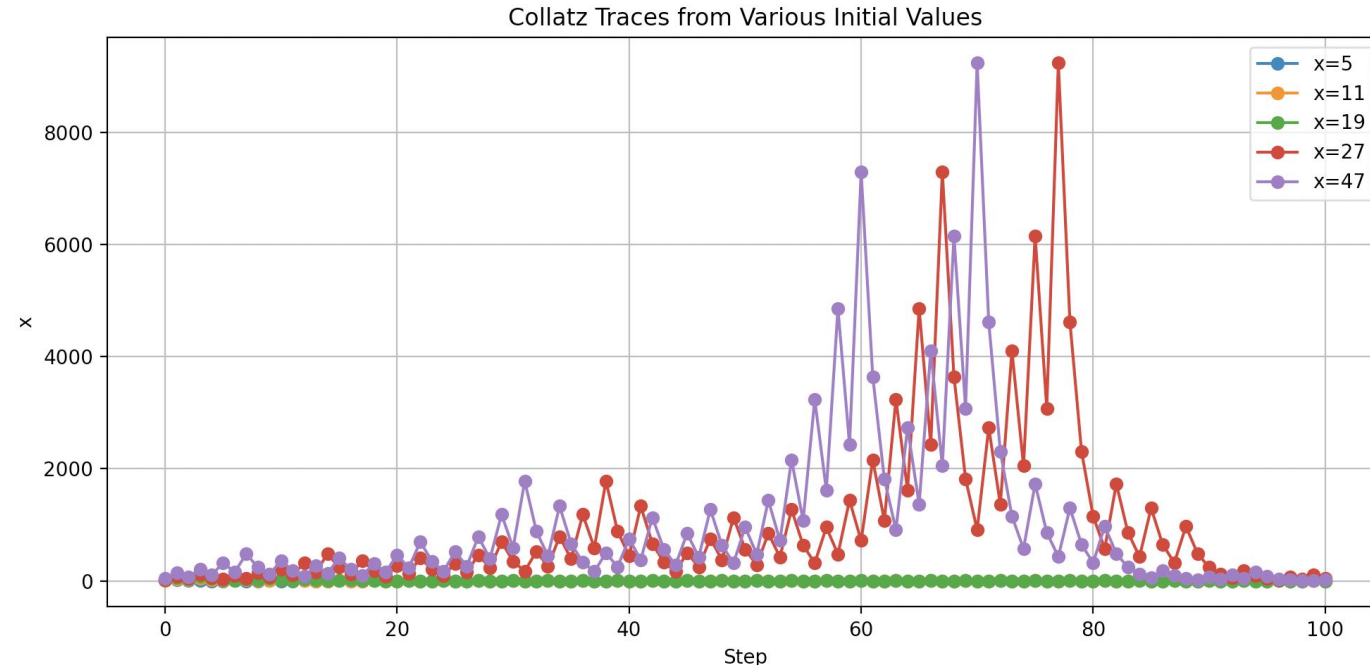
Specifications and verification

A **specification** is an assertion about executions of an automaton

Examples:

All executions of Collatz (with $\Theta \equiv \mathbb{Z}^+$) eventually enter reaches $\langle x \mapsto 1 \rangle$

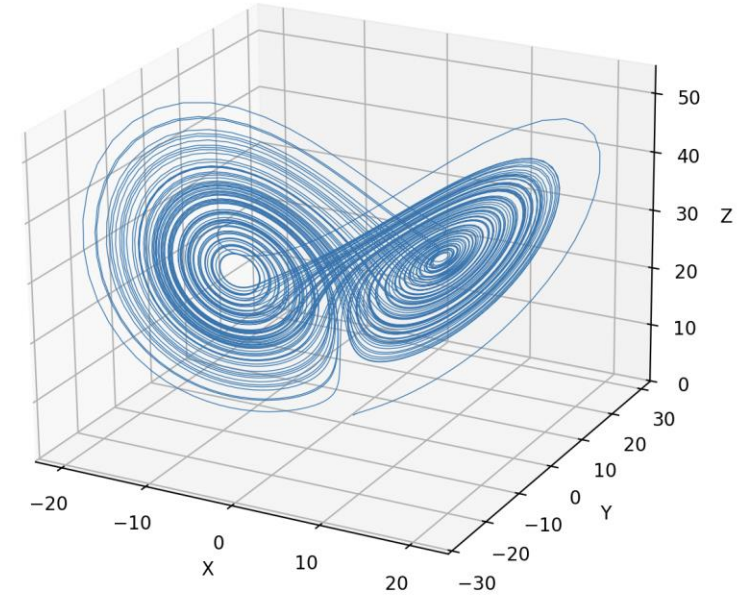
All executions of Collatz with $x_0 \in [1, 200]$ stays inside $x_i \leq 10,000$



Example 2: Butterfly effect and the Lorenz Attractor

$$\begin{aligned}\frac{dx}{dt} &= \sigma(y - x) \\ \frac{dy}{dt} &= x(\rho - z) - y \\ \frac{dz}{dt} &= xy - \beta z\end{aligned}$$

$$\begin{aligned}x' &= x + \Delta t \sigma(y - x) \\ y' &= y + \Delta t x(\rho - z) - y \\ z' &= z + \Delta t xy - \beta z\end{aligned}$$



In 1961, while working on models of atmospheric convection to improve weather forecasting, Edward Lorenz made a surprising discovery. He reran a computer simulation of ODEs but started it from **intermediate values rounded to three decimal places** instead of the full six.

To his astonishment, the new simulation initially matched the previous run but soon diverged completely, producing a different weather pattern. This unexpected sensitivity to initial conditions led Lorenz to realize that **deterministic systems** could behave unpredictably, laying the foundation for chaos theory.

Lorentz Automaton

Lorentz:

$$x' = x + \Delta t \sigma(y - x)$$

$$y' = y + \Delta t x(\rho - z) - y$$

$$z' = z + \Delta t xy - \beta z$$

$V = \{x, y, z\}$, $\text{type}(x) = \text{type}(y) = \text{type}(z) = \mathbb{R}$

A **valuation** for V : $\langle x \mapsto 5, y \mapsto 35.6, z \mapsto -3.5 \rangle$

state space $Q = \text{Val}(V) \equiv \mathbb{R}^3$

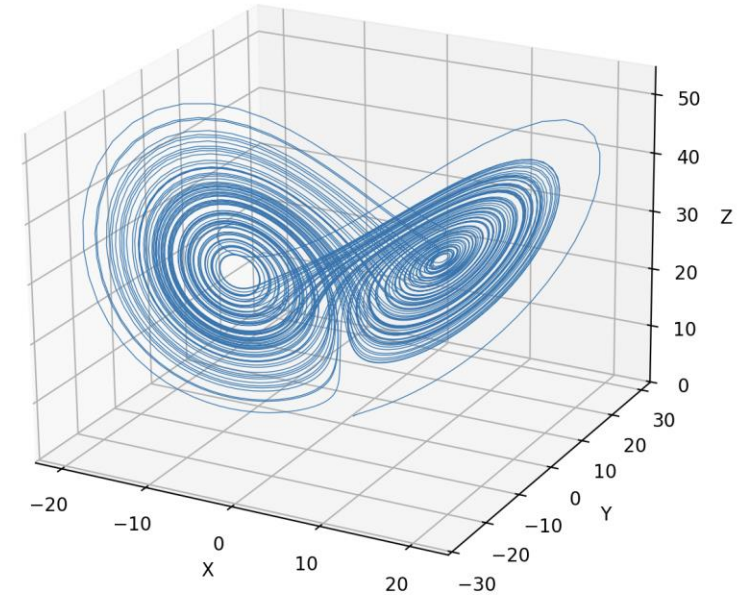
$A = \{\text{step}\}$

$\mathcal{D}_{\Delta t} = \{ \langle v, a, v' \rangle \mid v'.x = v.x + \Delta t \sigma(v.y - v.x), \dots \}$

Do all executions converge? Do they stay bounded?

Do executions starting nearby, stay close?

Even though this is a deterministic automaton, because of infinite state space and sensitivity to initial conditions, these questions are nontrivial



Assignments

- Read. Modeling computation: Chapter 2 of CPSBook, first part of Chapter 7, and section on SAT/SMT
- Form team (optional*) and go through Project Jumpstart exercise
- Next: Invariance and reachability