

# Verifying Cyber-Physical Systems

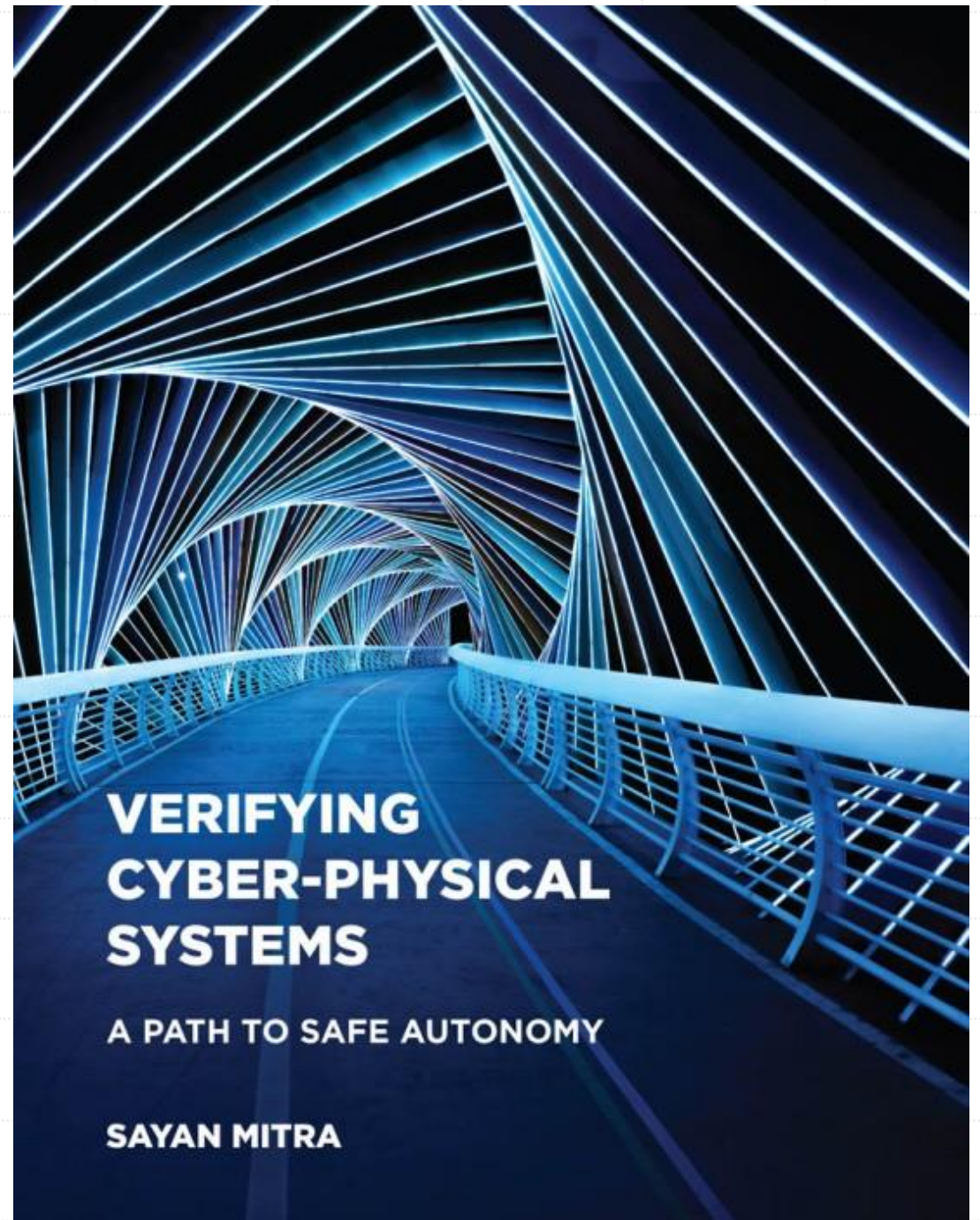
Chapter 3 – 4

Abenezer Taye

March , 2022

## Chapter 4:

# Modeling Cyber-Physical Systems



# Introduction

- **Cyber Physical System (CPS):** physical systems that are controlled by computer
- **Physical system modeling:** ODE
- **Computer operation:** Automata
- **CPS:** ODE + Automata (Hybrid Automata)

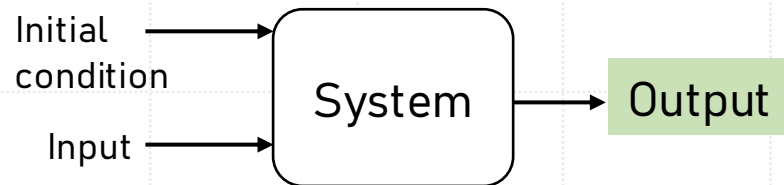


# Hybrid Automata (HA)

- Why study HA?
  - When the two modeling paradigms collide → strange behavior happens
    - ✓ It is possible for an HA to perform infinitely many transitions in finite time
    - ✓ It is possible for an HA to become unstable even though its ODEs are stable

# Testing Vs Verification

- **Testing:** analyzing the resulting behavior data to check whether the requirement is met

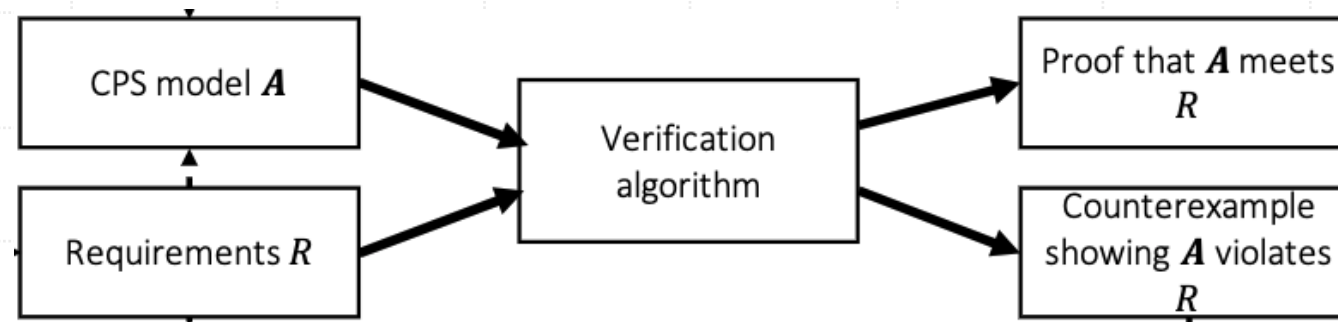


- For many systems, using testing for establishing system requirements is impossible
  - The amount of data required to guarantee a probability of  $10^{-9}$  fatality/hr is  $10^9$  hrs of data  $\approx 30 \times 10^9$  miles (Shalev-Shwartz et al. (2017))
- **Verification:** mathematically proves that a model of the system meets the requirements



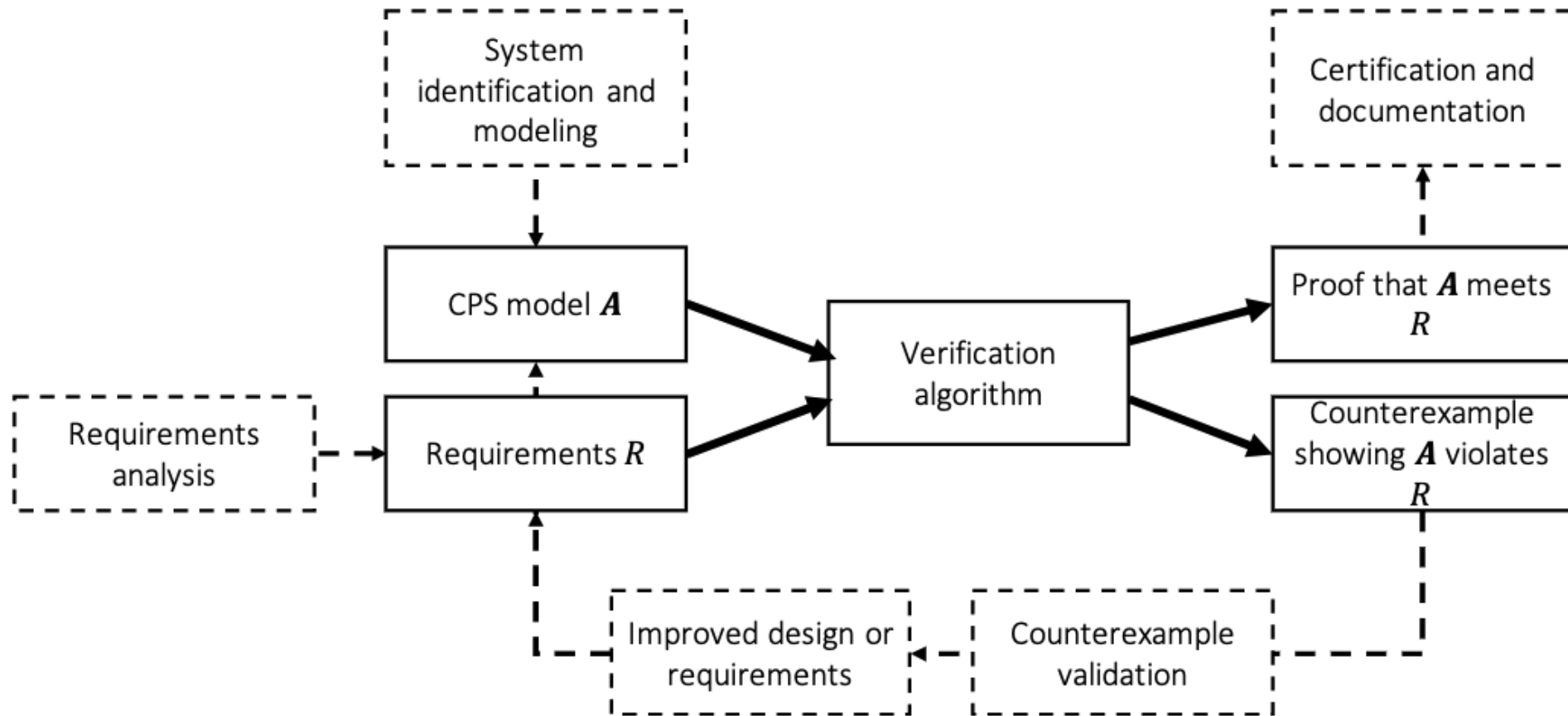
# Verification

- The verification process looks like:



- CPS model (A): is usually a hybrid model (ODE + automata)
- There could be two possible meanings to counterexample  $\alpha$ :
  - **Spurious counterexample:** because of the gap between model and the actual system
    - Solution  $\rightarrow$  Model validation
  - **Real counterexample:**
    - A design bug  $\rightarrow$  debug and modify the system
    - The requirements are wrong/inconsistent  $\rightarrow$  modify the requirements

# System Design Ecosystem



# Challenges of Verification

## 1. **Theoretical undecidability :**

- There is no algorithm that checks whether a model of CPS meets a requirement.
- To address this → relaxing the verification question (e.g. bounded time horizon or tolerate some false positives)

## 2. **Practical Scalability :**

- Lack of scalability is a huge challenge for the progress in verification

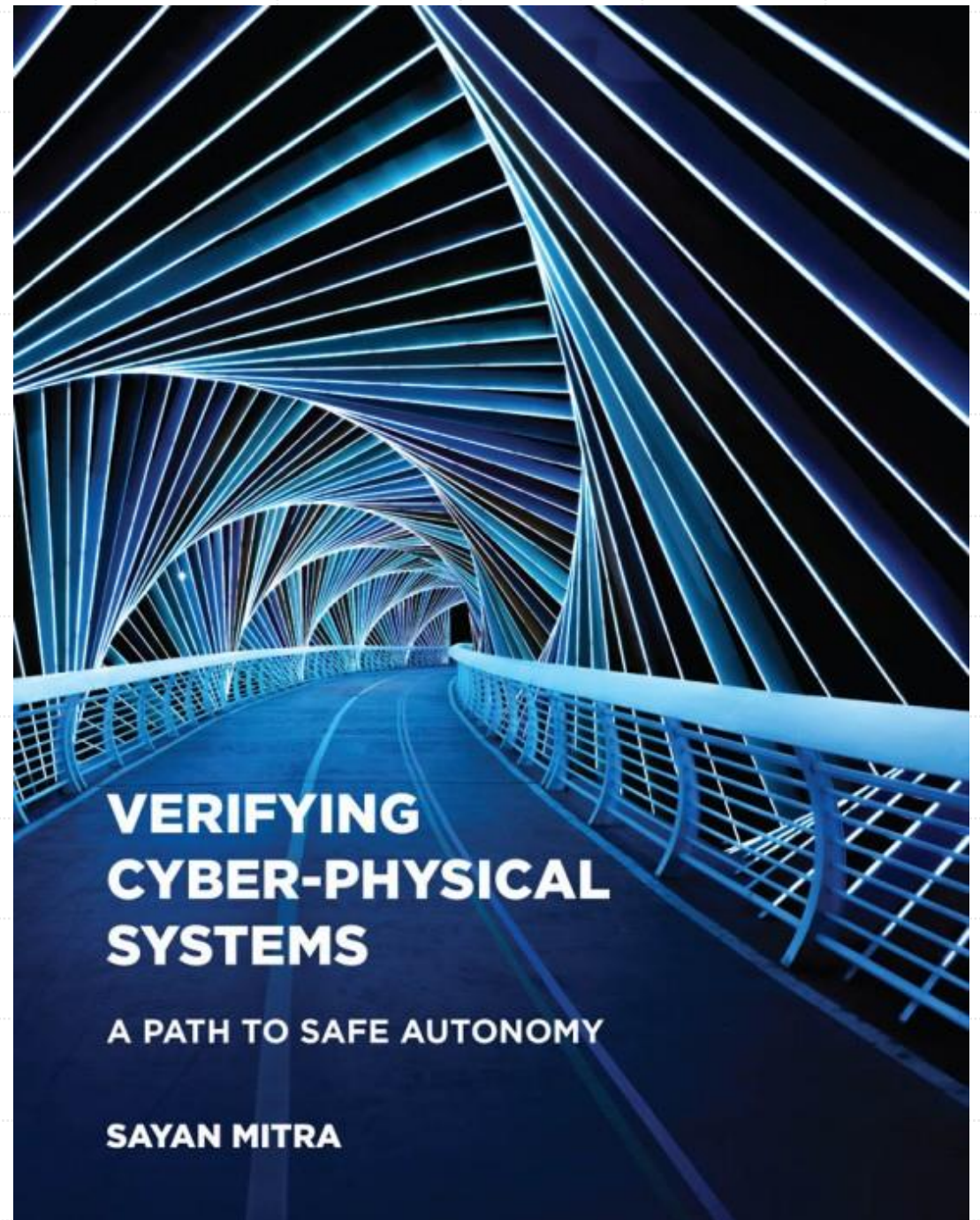
## 3. **Missing Models :**

- Complete models of CPS systems are difficult to come by
  - Oftentimes models lack succinct descriptions, are distributed, and are protected intellectual property

## 4. **Learning-enabled systems and missing requirements :**

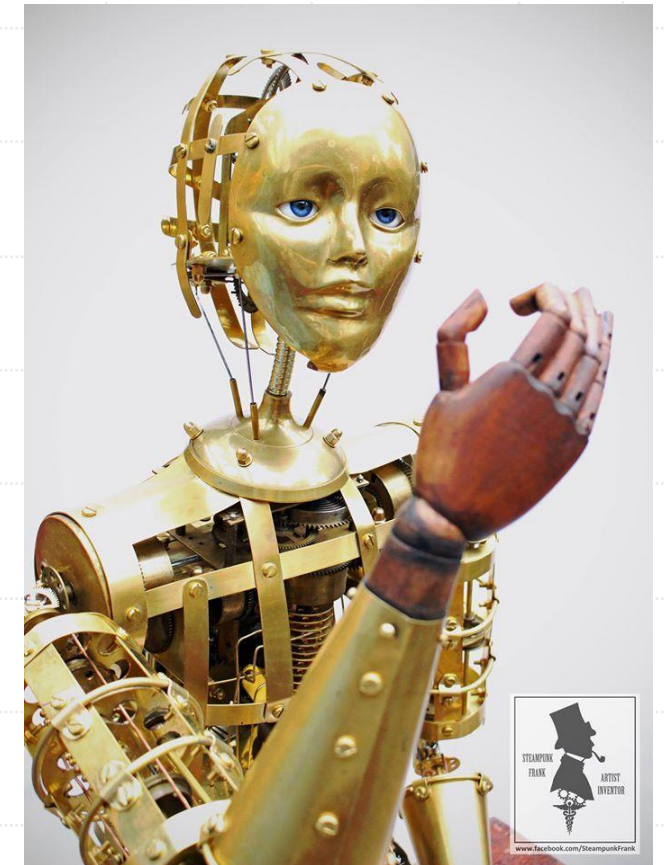
- Recent CPS systems use DNN → lack models and requirements

# Chapter 2: MODELING COMPUTATION

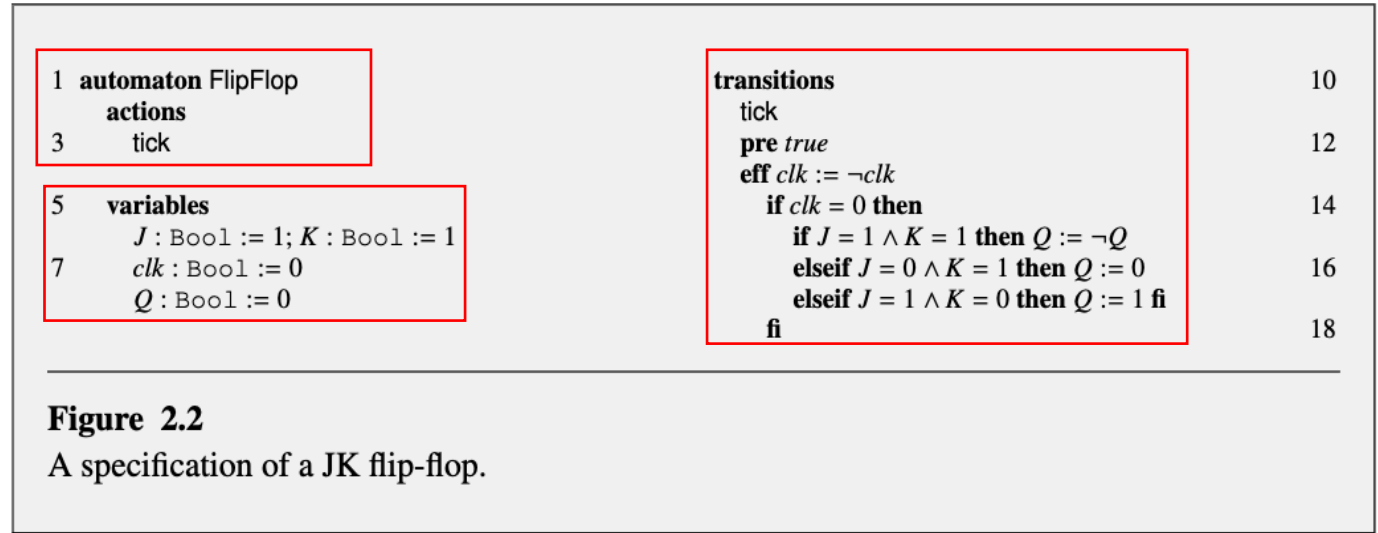
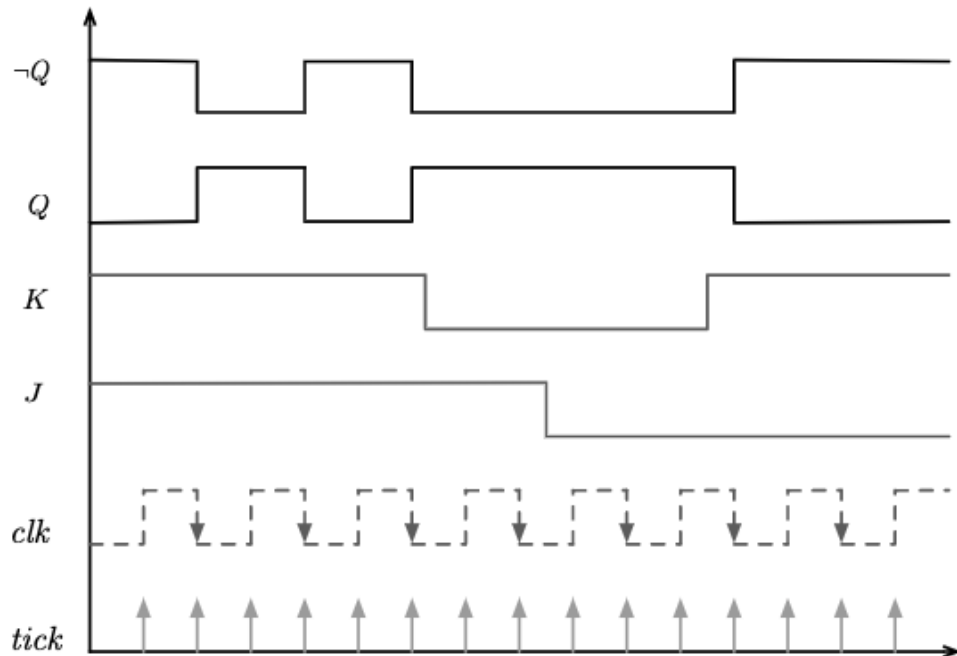


# Introduction

- **Aim:** how can we develop a mathematical model for computation
- We use differential equations to model physical systems  
→ how about computation?
- **Automaton:** 19<sup>th</sup> century word used to describe mechanical dolls
- In mathematics, we use automaton to describe the rules of state change of any computational process
  - Aka, **discrete-time transition system** or a **state machine**



# Example: JK Flip-Flop



time			
clk	J	k	Q
0	0	0	previous
0	0	1	0
0	1	0	1
0	1	1	toggle

- ### Parts of specification
- 1. Actions:** lists various types of rules for state change
  - 2. Variables:** lists the state variables
  - 3. Transitions:** lists the state change rules for each of the actions

# Specifying automata

## 1. State variables:

- A state represents the information necessary for predicting the future of an automaton

## 2. Valuation:

- Maps each state variable name to a value  $\mathbf{v} = (J \mapsto 1, K \mapsto 1, clk \mapsto 0, Q \mapsto 0)$ .

## 3. Predicates:

- A Boolean-valued formula involving the variables in  $V$  (we use it as a requirement)
  - For example: the state  $\mathbf{v} = (J \mapsto 1, K \mapsto 1, clk \mapsto 0, Q \mapsto 0)$ . satisfies the formula  $\phi_1 := (J = 0 \vee K = 1)$
  - Because  $(\mathbf{v} \models J = 0 \vee \mathbf{v} \models K = 1) \equiv (0 = 0 \vee 1 = 1) \equiv true$ .

## 4. Transitions:

- Specify the rules for state change in a computation

# Automata

- An automaton  $\mathcal{A}$  is a tuple  $(V, \Theta, A, \mathcal{D})$ , where
  1.  $V$  is a set of state variables and  $val(V)$  is valuations of  $V$
  2.  $\Theta \subseteq val(V)$  is a set of start states
  3.  $A$  is a set of actions
  4.  $\mathcal{D} \subseteq val(V) \times A \times val(V)$  is called the set of transitions

$$\begin{aligned}
 & \mathbf{v}' \Vdash J = \mathbf{v} \Vdash J, \mathbf{v}' \Vdash K = \mathbf{v} \Vdash K, \mathbf{v}' \Vdash clk = \neg \mathbf{v} \Vdash clk \\
 & \neg \mathbf{v}' \Vdash clk \wedge \mathbf{v} \Vdash J \wedge \neg \mathbf{v} \Vdash K \Rightarrow \mathbf{v}' \Vdash Q, \\
 & \neg \mathbf{v}' \Vdash clk \wedge \neg \mathbf{v} \Vdash J \wedge \mathbf{v} \Vdash K \Rightarrow \neg \mathbf{v}' \Vdash Q, \\
 & \neg \mathbf{v}' \Vdash clk \wedge \mathbf{v} \Vdash J \wedge \mathbf{v} \Vdash K \Rightarrow (\mathbf{v}' \Vdash Q = \neg \mathbf{v}' \Vdash Q).
 \end{aligned}$$

<b>transitions</b>	10
tick	
<b>pre true</b>	12
<b>eff</b> $clk := \neg clk$	
<b>if</b> $clk = 0$ <b>then</b>	14
<b>if</b> $J = 1 \wedge K = 1$ <b>then</b> $Q := \neg Q$	
<b>elseif</b> $J = 0 \wedge K = 1$ <b>then</b> $Q := 0$	16
<b>elseif</b> $J = 1 \wedge K = 0$ <b>then</b> $Q := 1$ <b>fi</b>	
<b>fi</b>	18

# Semantics

- **Execution** ( $Exces_{\mathcal{A}}(\Theta)$ ): an alternating, possibly infinite, sequence of states and actions  $v_0, a_1, v_2, \dots$ 
  1. Each  $v_i$  is a state  $\text{val}(V)$
  2. Each  $a_i$  is an action in  $A$
  3. Each consecutive triple  $(v_i, a_{i+1}, v_{i+1})$  is a valid transition in  $\mathcal{D}$
- **Reachable state** ( $Reach_{\mathcal{A}}(\Theta)$ ): a state is reachable if there exists a finite execution that ends at  $v$
- **Invariant**: a set of states  $S$  that contains all of  $\mathcal{A}$ 's reachable states



# Thank You!!!