May a computer be wrong?

MICHAŁ SKRZYPCZAK
Institute of Informatics

Latest Discoveries in Informatics

6th March 2024

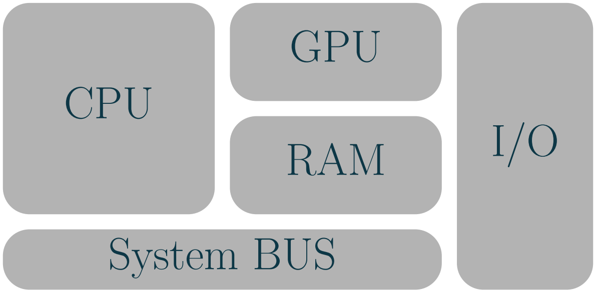
Computers have layers...

Computers have layers...

Hardware:

Computers have layers. . .

Hardware:

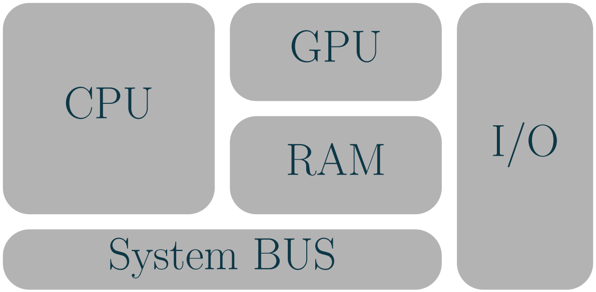


Logic + Physics

Computers have layers...

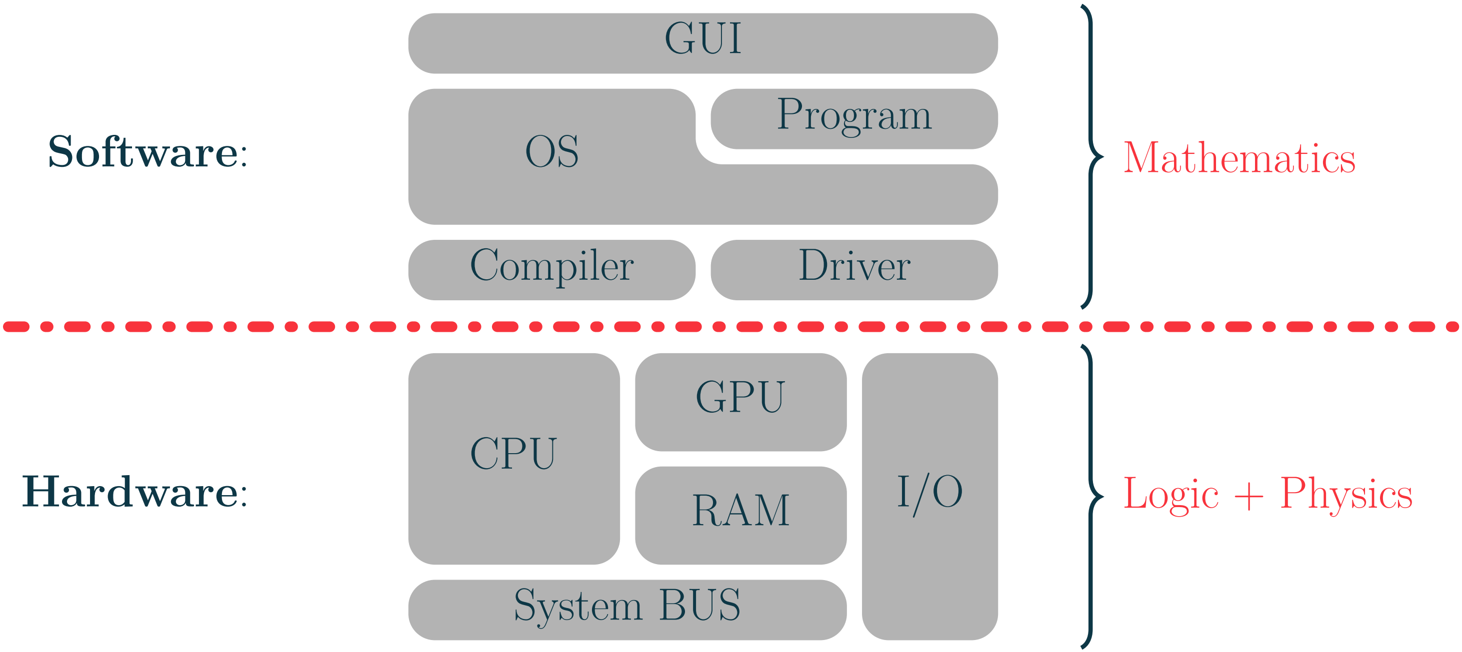
Software:

Hardware:



Logic + Physics

Computers have layers...

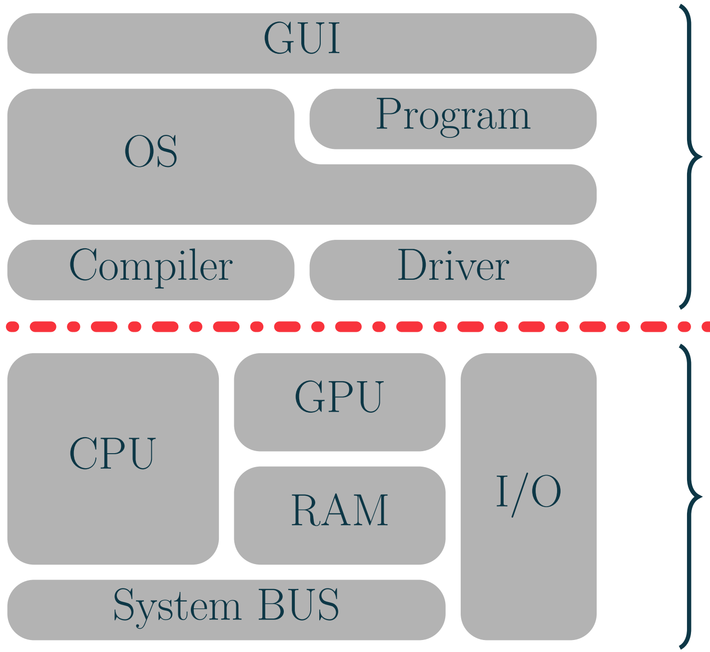


Computers have layers...

Organic:

Software:

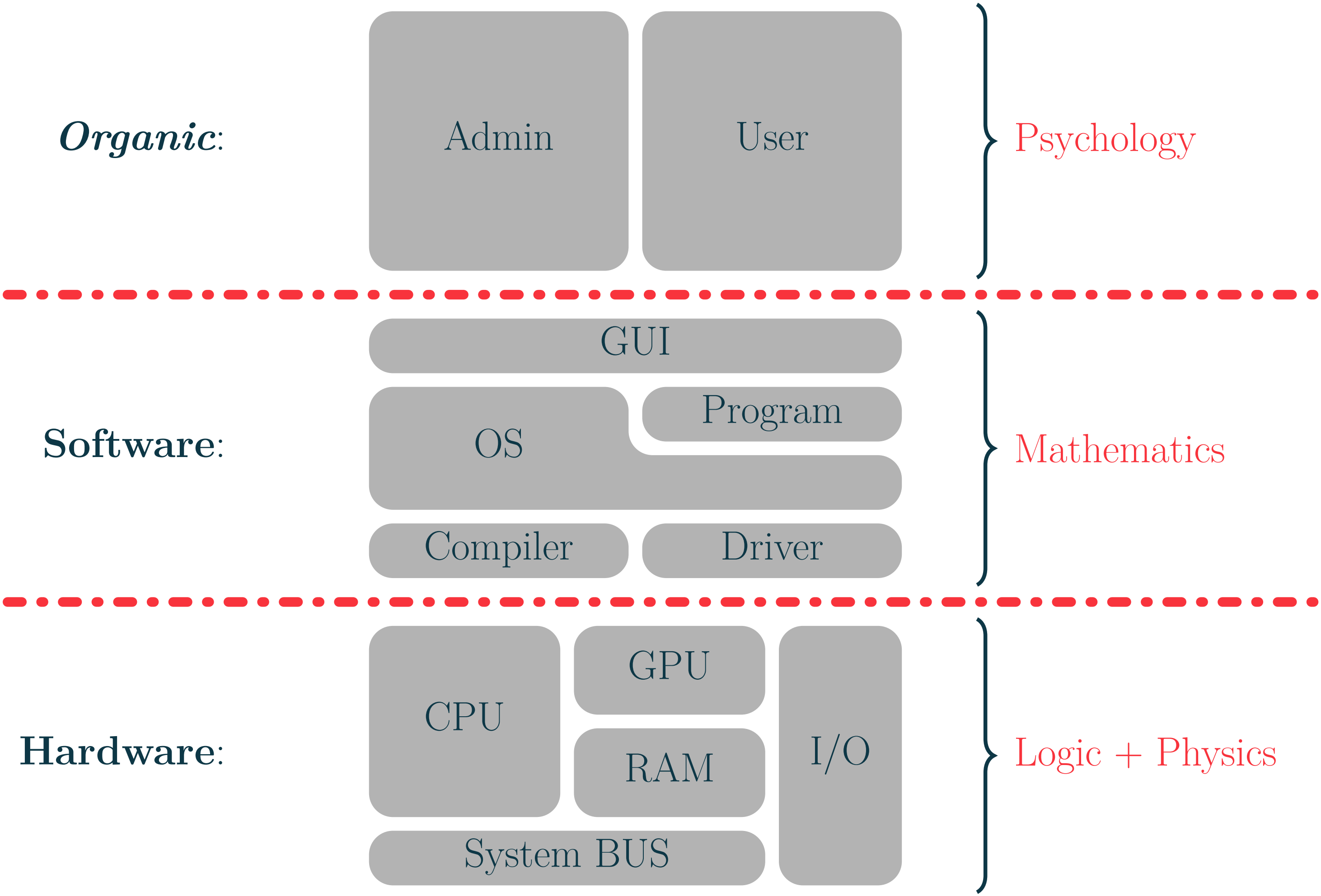
Hardware:



Mathematics

Logic + Physics

Computers have layers. . .

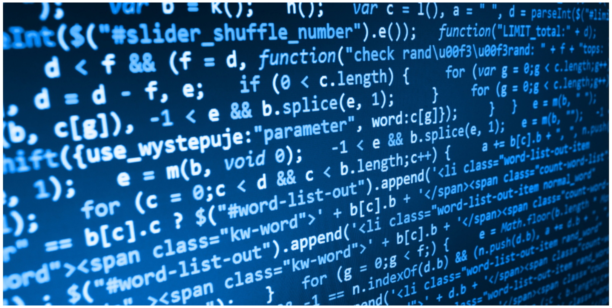


Computers have layers. . .

Organic:

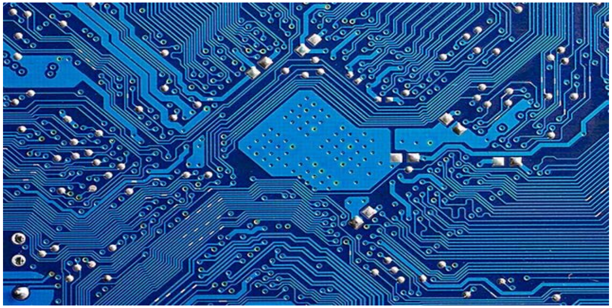
Psychology

Software:



Mathematics

Hardware:



Logic + Physics

Computers have layers...

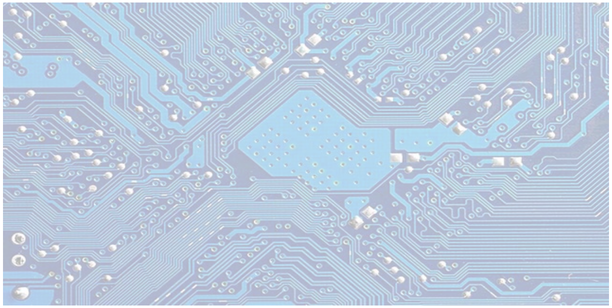
***Organic*:**

Psychology

Software:

Mathematics

Hardware:



Logic + Physics

Organic errors

Organic errors

74% of security breaches in 2023 involved a human vurnerability

(FBI, Verizon, IBM, . . .)

Organic errors

74% of security breaches in 2023 involved a human vurnerability

(FBI, Verizon, IBM, . . .)

80% of aviation accidents involve human errors

(FAA estimation)

Organic errors

74% of security breaches in 2023 involved a human vurnerability

(FBI, Verizon, IBM, . . .)

80% of aviation accidents involve human errors

(FAA estimation)

35% of aviation accidents in 2015–2019 in USA were caused by human error

(another study)

Organic errors

74% of security breaches in 2023 involved a human vurnerability

(FBI, Verizon, IBM, . . .)

80% of aviation accidents involve human errors

(FAA estimation)

35% of aviation accidents in 2015–2019 in USA were caused by human error

(another study)

37% of train accidents in 2001–2005 in USA were caused by human error

(US DoT)

Organic errors

74% of security breaches in 2023 involved a human vurnerability

(FBI, Verizon, IBM, . . .)

80% of aviation accidents involve human errors

(FAA estimation)

35% of aviation accidents in 2015–2019 in USA were caused by human error

(another study)

37% of train accidents in 2001–2005 in USA were caused by human error

(US DoT)

93% of car collisions in USA were caused by human error

(Indiana University study)

Organic errors

74% of security breaches in 2023 involved a human vurnerability

(FBI, Verizon, IBM, . . .)

80% of aviation accidents involve human errors

(FAA estimation)

35% of aviation accidents in 2015–2019 in USA were caused by human error

(another study)

37% of train accidents in 2001–2005 in USA were caused by human error

(US DoT)

93% of car collisions in USA were caused by human error

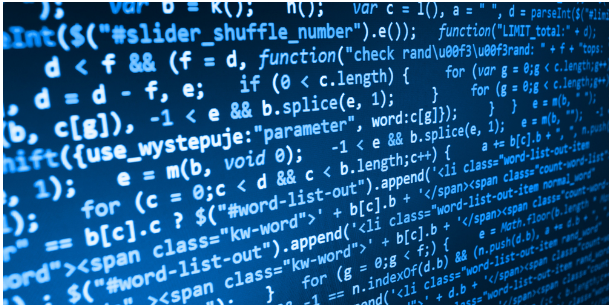
(Indiana University study)

errare humanum est

Organic:

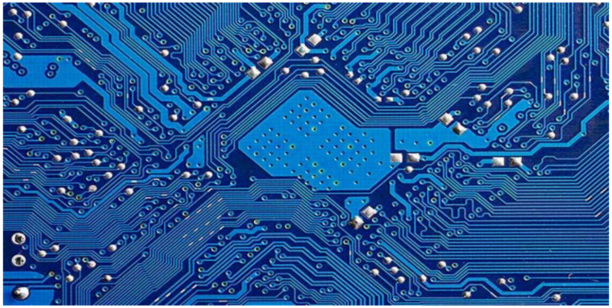
Psychology

Software:



Mathematics

Hardware:



Logic + Physics

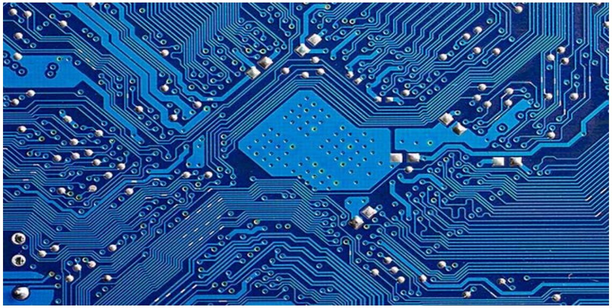
Organic:

Psychology

Software:

Mathematics

Hardware:



Logic + Physics

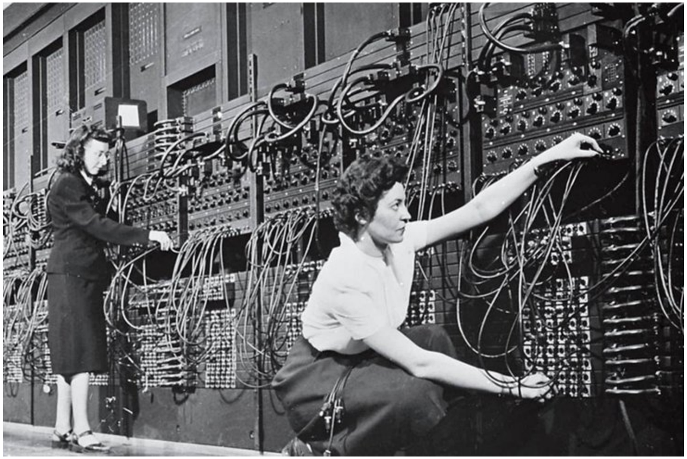
Hardware errors

Hardware errors

THEN:

Hardware errors

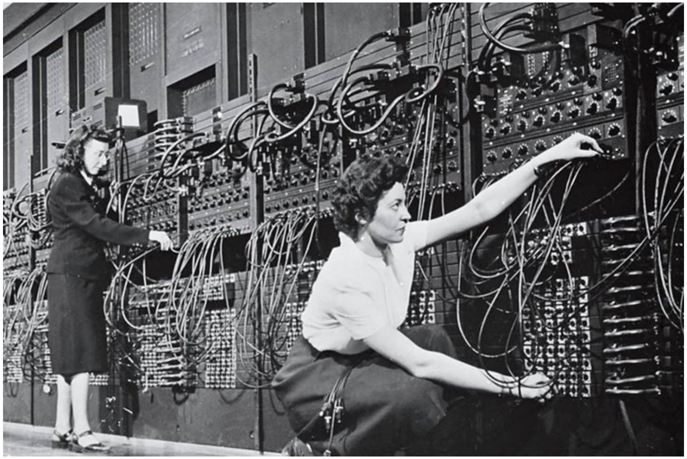
THEN: **ENIAC** 1945 – 1955



Hardware errors

THEN: **ENIAC** 1945 – 1955

- ∼20'000 vacuum tubes
- ∼5'000'000 hand-made solders
- power rating 150kW (∼100 households)

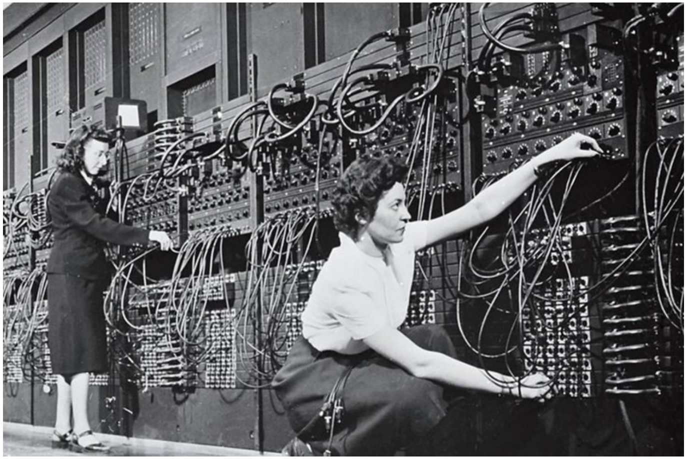


Hardware errors

THEN: **ENIAC** 1945 – 1955

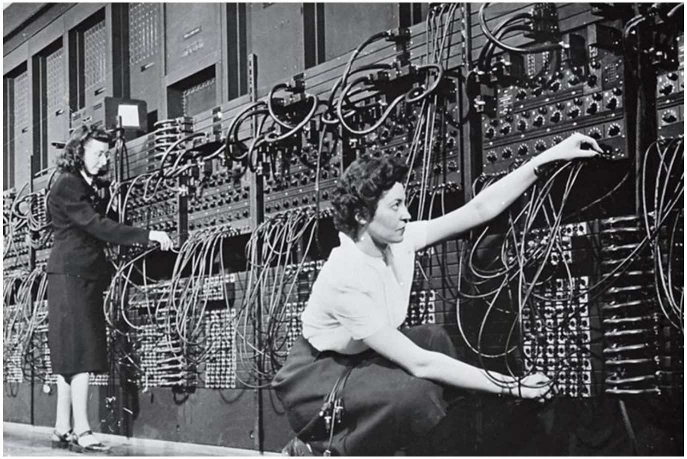
- $\sim$20'000 vacuum tubes
- $\sim$5'000'000 hand-made solders
- power rating 150kW ($\sim$100 households)
- average time between breakdowns:

  10 min. (1945) $\rightarrow$ >12 h (1955)



Hardware errors

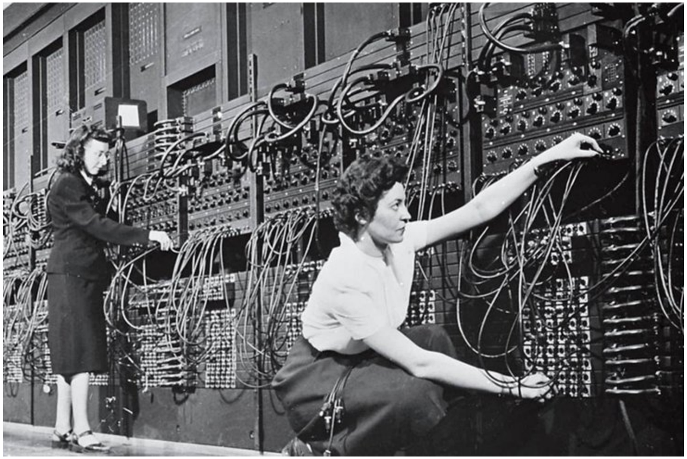
THEN: **ENIAC** 1945 – 1955



- ∼20'000 vacuum tubes
- ∼5'000'000 hand-made solders
- power rating 150kW (∼100 households)
- average time between breakdowns:
  10 min. (1945) → >12 h (1955)
- maximal continuous operating time: 116 hours (1954)

Hardware errors

THEN: **ENIAC** 1945 – 1955



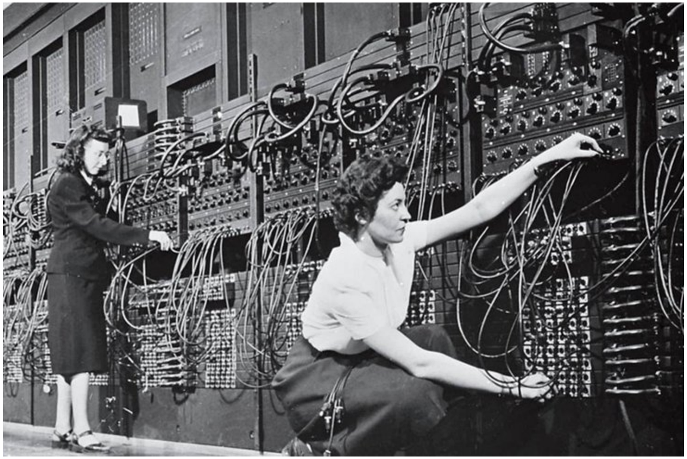
- $\sim$20'000 vacuum tubes
- $\sim$5'000'000 hand-made solders
- power rating 150kW ($\sim$100 households)
- average time between breakdowns:

  10 min. (1945) $\rightarrow$ >12 h (1955)
- maximal continuous operating time: 116 hours (1954)

NOW:

Hardware errors

THEN: **ENIAC** 1945 – 1955



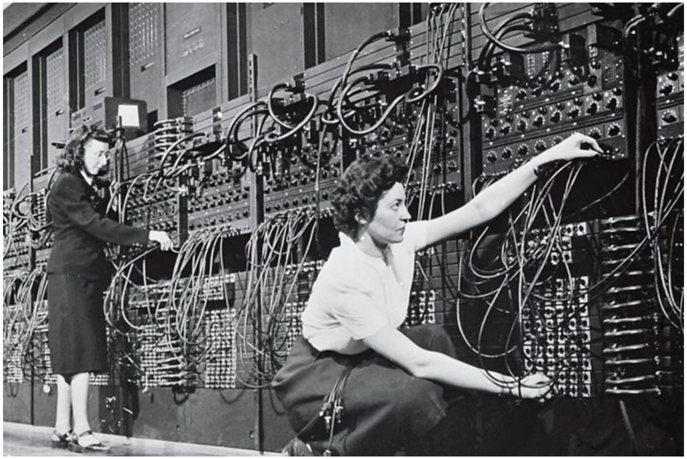
- $\sim$20'000 vacuum tubes
- $\sim$5'000'000 hand-made solders
- power rating 150kW ($\sim$100 households)
- average time between breakdowns:

  10 min. (1945) $\rightarrow$ >12 h (1955)
- maximal continuous operating time: 116 hours (1954)

NOW: **PC** 2018

Hardware errors

THEN: **ENIAC** 1945 – 1955



- ~20'000 vacuum tubes
- ~5'000'000 hand-made solders
- power rating 150kW (~100 households)
- average time between breakdowns:

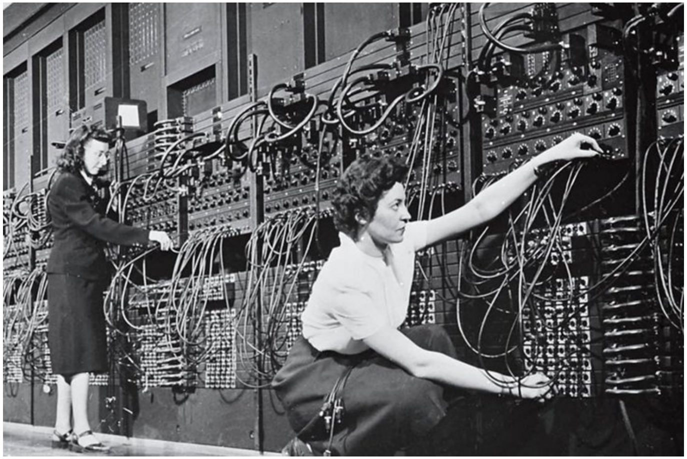
  10 min. (1945) → >12 h (1955)
- maximal continuous operating time: 116 hours (1954)

NOW: **PC** 2018

- $\sim 2 \cdot 10^9$ transistors of CPU
- $\sim 64 \cdot 10^9$ transistors of RAM
- power rating 1–100W

Hardware errors

THEN: **ENIAC** 1945 – 1955



- ~20’000 vacuum tubes
- ~5’000’000 hand-made solders
- power rating 150kW (~100 households)
- average time between breakdowns:
  10 min. (1945) → >12 h (1955)
- maximal continuous operating time: 116 hours (1954)

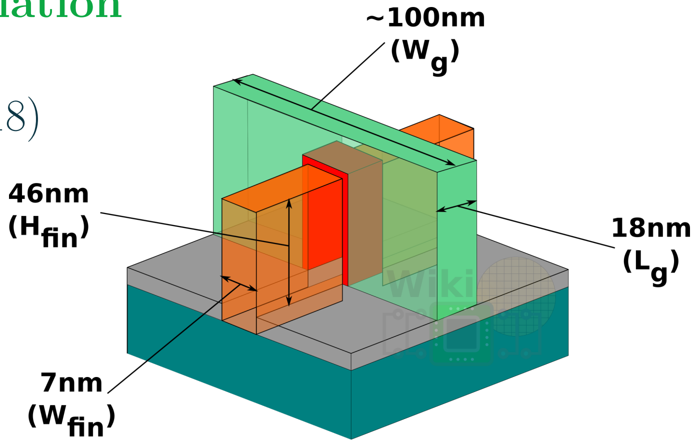
NOW: **PC** 2018

- $\sim 2 \cdot 10^9$ transistors of CPU
- $\sim 64 \cdot 10^9$ transistors of RAM
- power rating 1–100W
- average time between breakdowns:
  1100–3285 years (RAM), 126–220 years (CPU)

Cosmic radiation

Cosmic radiation

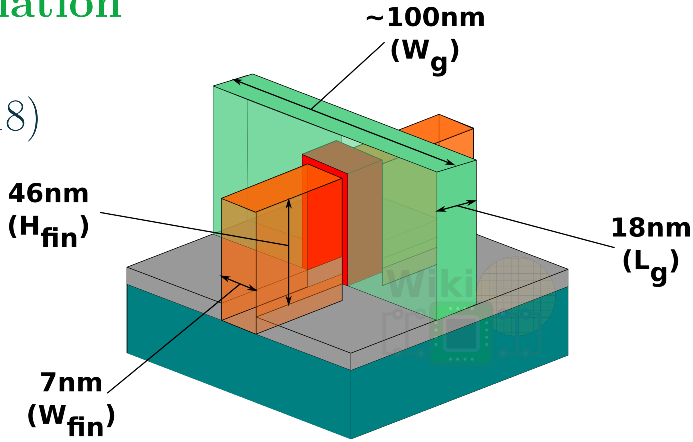
Integrated circuits in “10nm” technology (2018)



Cosmic radiation

Integrated circuits in “10nm” technology (2018)

Paths of width in hundreds of atoms!

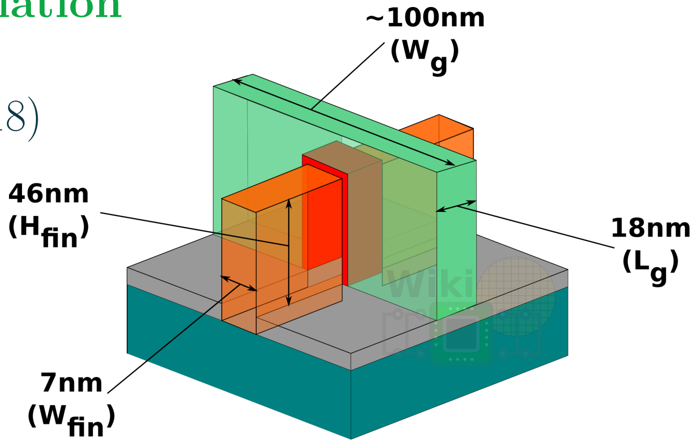


Cosmic radiation

Integrated circuits in “10nm” technology (2018)

Paths of width in hundreds of atoms!

$\rightsquigarrow$ risk of *Single Event Upset*

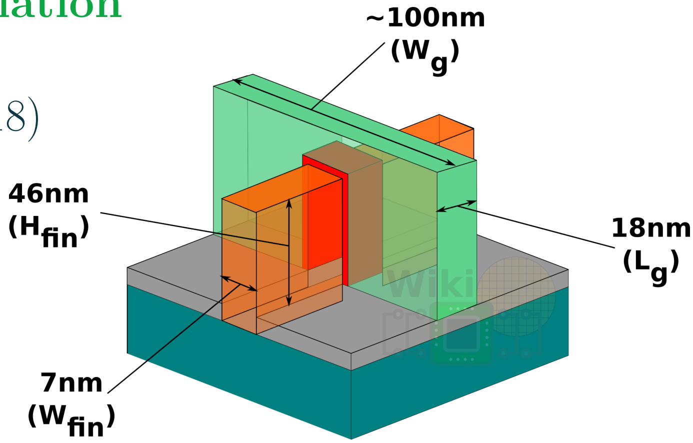


Cosmic radiation

Integrated circuits in “10nm” technology (2018)

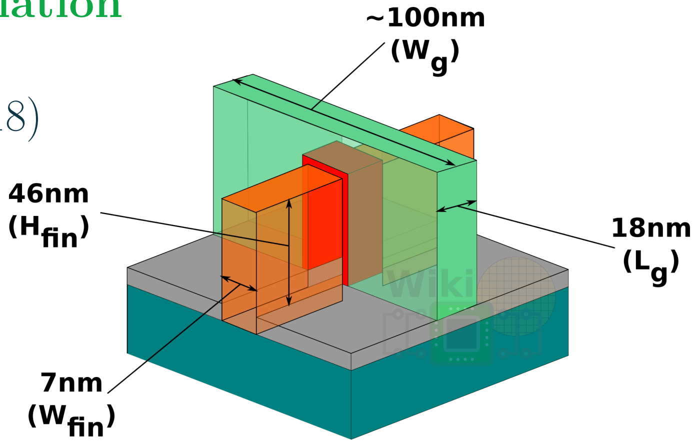
Paths of width in hundreds of atoms!

⇝ risk of *Single Event Upset*



Confirmed cases:

Cosmic radiation



Integrated circuits in “10nm” technology (2018)

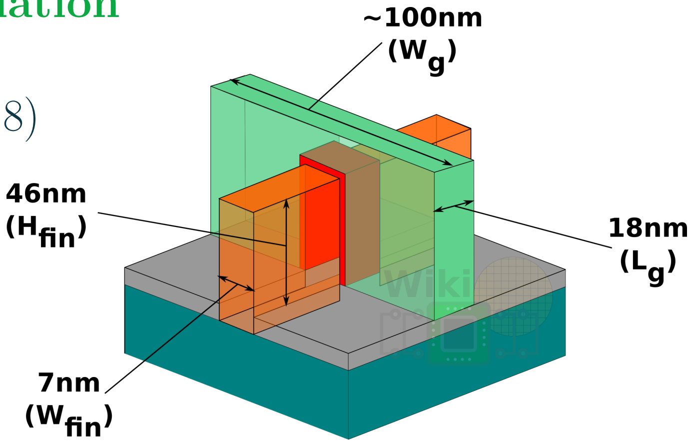
Paths of width in hundreds of atoms!

⇝ risk of *Single Event Upset*

Confirmed cases:

- In 1972 a communication satelite Hughes broke down for 96 seconds.

Cosmic radiation



Integrated circuits in “10nm” technology (2018)

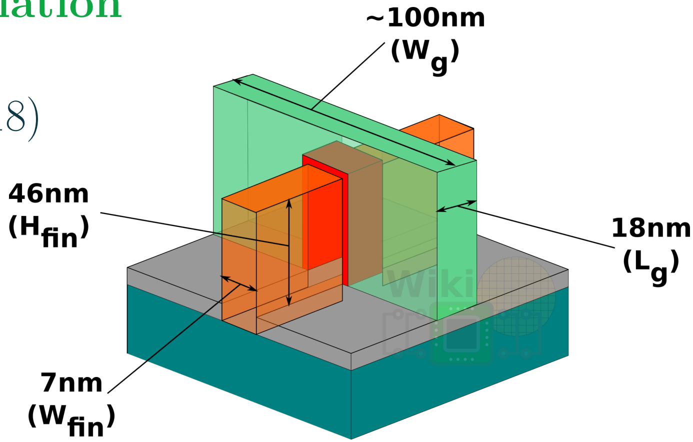
Paths of width in hundreds of atoms!

⇝ risk of *Single Event Upset*

Confirmed cases:

- In 1972 a communication satelite Hughes broke down for 96 seconds.
- In 2003 in Schaerbeek (Belgium) a candidate got 4096 too many votes.

Cosmic radiation



Integrated circuits in “10nm” technology (2018)

Paths of width in hundreds of atoms!

⇝ risk of *Single Event Upset*

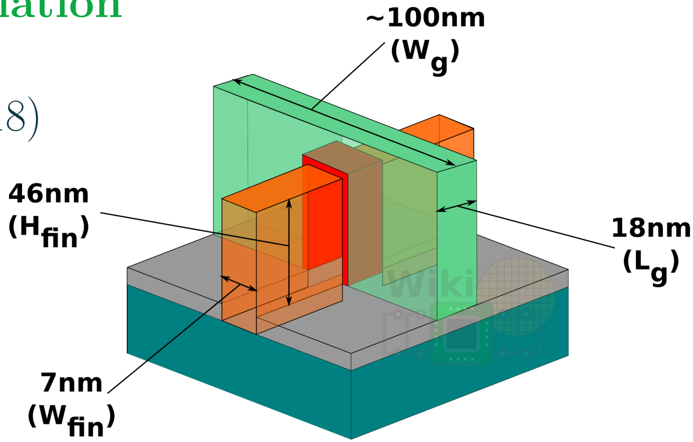
Confirmed cases:

- In 1972 a communication satelite Hughes broke down for 96 seconds.
- In 2003 in Schaerbeek (Belgium) a candidate got 4096 too many votes.

$$4096 = 2^{12}$$

Cosmic radiation

Integrated circuits in "10nm" technology (2018)



Paths of width in hundreds of atoms!

⇝ risk of *Single Event Upset*

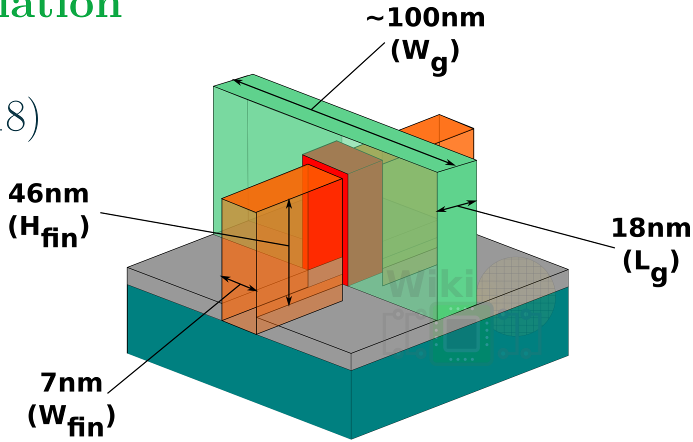
Confirmed cases:

- In 1972 a communication satelite Hughes broke down for 96 seconds.
- In 2003 in Schaerbeek (Belgium) a candidate got 4096 too many votes.

$$4096 = 2^{12}$$

Mitigation techniques:

Cosmic radiation



Integrated circuits in "10nm" technology (2018)

Paths of width in hundreds of atoms!

$\rightsquigarrow$ risk of *Single Event Upset*

Confirmed cases:

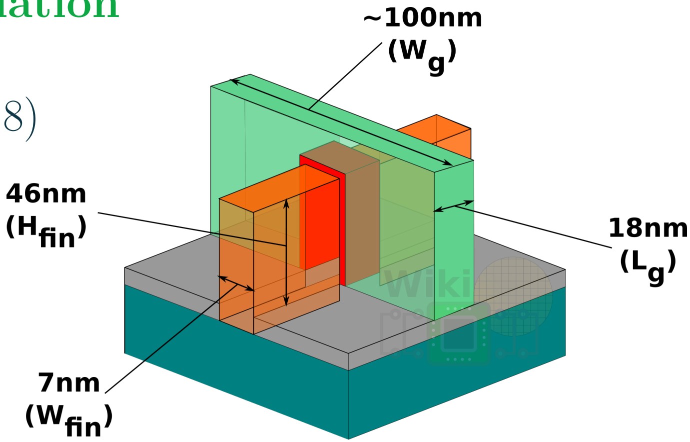
- In 1972 a communication satelite Hughes broke down for 96 seconds.
- In 2003 in Schaerbeek (Belgium) a candidate got 4096 too many votes.

$$4096 = 2^{12}$$

Mitigation techniques:

- Computers at ISS are based on i386 CPUs (1μm$= 100 \times 10$nm technology).

Cosmic radiation



Integrated circuits in "10nm" technology (2018)

Paths of width in hundreds of atoms!

⇝ risk of *Single Event Upset*

Confirmed cases:

- In 1972 a communication satelite Hughes broke down for 96 seconds.
- In 2003 in Schaerbeek (Belgium) a candidate got 4096 too many votes.

$$4096 = 2^{12}$$

Mitigation techniques:

- Computers at ISS are based on i386 CPUs (1μm$= 100 \times 10$nm technology).
- Trippled computer systems in *fly-by-wire* aircrafts.

Hardware errors

Hardware errors

1993 error in FDIV instruction of Intel Pentium CPUs

Hardware errors

1993 error in FDIV instruction of Intel Pentium CPUs

A pre-computed array of 1066 numbers from $\{-2, -1, 0, 1, 2\}$

had 5 wrong entries.

Hardware errors

1993 error in FDIV instruction of Intel Pentium CPUs

A pre-computed array of 1066 numbers from $\{-2, -1, 0, 1, 2\}$ had 5 wrong entries.

In unfavourable circumstances fourth significant decimal digit was wrong:

Hardware errors

1993 error in FDIV instruction of Intel Pentium CPUs

A pre-computed array of 1066 numbers from $\{-2,-1,0,1,2\}$

had 5 wrong entries.

In unfavourable circumstances fourth significant decimal digit was wrong:

$$\frac{4'195'835}{3'145'727} = 1.333820449136241002$$

Hardware errors

1993 error in FDIV instruction of Intel Pentium CPUs

A pre-computed array of 1066 numbers from $\{-2, -1, 0, 1, 2\}$

had 5 wrong entries.

In unfavourable circumstances fourth significant decimal digit was wrong:

$$\frac{4'195'835}{3'145'727} = 1.333820449136241002$$

$$\frac{4'195'835}{3'145'727} = 1.333739068902037589$$

Hardware errors

1993 error in FDIV instruction of Intel Pentium CPUs

A pre-computed array of 1066 numbers from $\{-2,-1,0,1,2\}$

had 5 wrong entries.

In unfavourable circumstances fourth significant decimal digit was wrong:

$$\frac{4'195'835}{3'145'727} = 1.3338204491362410 02$$

$$\frac{4'195'835}{3'145'727} = 1.333{\color{red}739068902037589}$$

Only 1 in 9 billion divisions with random parameters produced wrong results.

Hardware errors

1993 error in FDIV instruction of Intel Pentium CPUs

A pre-computed array of 1066 numbers from $\{-2, -1, 0, 1, 2\}$ had 5 wrong entries.

In unfavourable circumstances fourth significant decimal digit was wrong:

$$\frac{4'195'835}{3'145'727} = 1.33382044913624100 2$$

$$\frac{4'195'835}{3'145'727} = 1.33373906890203758 9$$

Only 1 in 9 billion divisions with random parameters produced wrong results.

- June 13, 1994: error discovered by Thomas R. Nicely

Hardware errors

1993 error in FDIV instruction of Intel Pentium CPUs

A pre-computed array of 1066 numbers from $\{-2,-1,0,1,2\}$ had 5 wrong entries.

In unfavourable circumstances fourth significant decimal digit was wrong:

$$\frac{4'195'835}{3'145'727} = 1.3338204491362410 02$$

$$\frac{4'195'835}{3'145'727} = 1.333739068902037589$$

Only 1 in 9 billion divisions with random parameters produced wrong results.

- June 13, 1994: error discovered by Thomas R. Nicely
- October 20, 1994: error reported

Hardware errors

1993 error in FDIV instruction of Intel Pentium CPUs

A pre-computed array of 1066 numbers from $\{-2, -1, 0, 1, 2\}$

had 5 wrong entries.

In unfavourable circumstances fourth significant decimal digit was wrong:

$$\frac{4'195'835}{3'145'727} = 1.3338204491362410025$$

$$\frac{4'195'835}{3'145'727} = 1.333739068902037589$$

Only 1 in 9 billion divisions with random parameters produced wrong results.

- June 13, 1994: error discovered by Thomas R. Nicely
- October 20, 1994: error reported
- December 20, 1994: Intel offers replacement of sold chips

Hardware errors

1993 error in FDIV instruction of Intel Pentium CPUs

A pre-computed array of 1066 numbers from $\{-2,-1,0,1,2\}$ had 5 wrong entries.

In unfavourable circumstances fourth significant decimal digit was wrong:

$$\frac{4'195'835}{3'145'727} = 1.3338204491362410025$$

$$\frac{4'195'835}{3'145'727} = 1.3337390689020375894$$

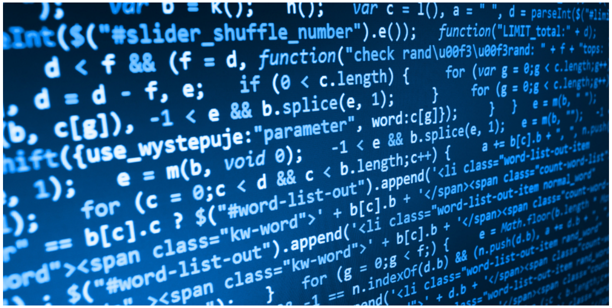
Only 1 in 9 billion divisions with random parameters produced wrong results.

- June 13, 1994: error discovered by Thomas R. Nicely
- October 20, 1994: error reported
- December 20, 1994: Intel offers replacement of sold chips
- Total cost: **475 million $**

Organic:

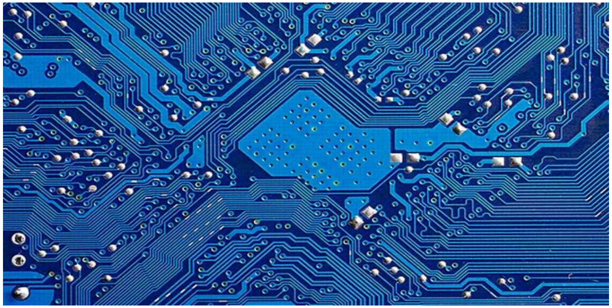
Psychology

Software:



Mathematics

Hardware:

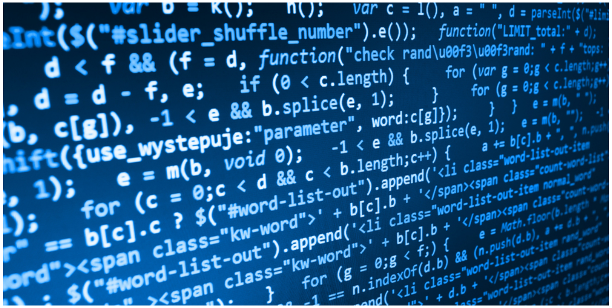


Logic + Physics

Organic:

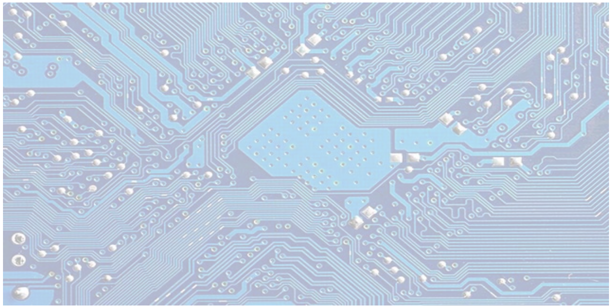
Psychology

Software:



Mathematics

Hardware:



Logic + Physics

Correct programs?

Correct programs?

1986 computer-controlled radiotherapy method Therac-25

Correct programs?

1986 computer-controlled radiotherapy method Therac-25

Correct programs?

1986 computer-controlled radiotherapy method Therac-25

Race condition in concurrent code

Correct programs?

1986 computer-controlled radiotherapy method Therac-25

Race condition in concurrent code

Previously used hardware interlocks were exchanged to software ones

Correct programs?

1986 computer-controlled radiotherapy method Therac-25

Race condition in concurrent code

Previously used hardware interlocks were exchanged to software ones

Approximately 100 times bigger dose than expected

Correct programs?

1986 computer-controlled radiotherapy method Therac-25

Race condition in concurrent code

Previously used hardware interlocks were exchanged to software ones

Approximately 100 times bigger dose than expected

⇝ 6 seriously overdosed patients, at least 3 fatalities

Correct programs??

Correct programs??

1996 Ariane 5 (ESA) rocket (software partially based on Ariane 4)

Correct programs??

1996 Ariane 5 (ESA) rocket (software partially based on Ariane 4)

Original code of Ariane 4 **was** formally verified

Correct programs??

1996 Ariane 5 (ESA) rocket (software partially based on Ariane 4)

Original code of Ariane 4 **was** formally verified

But Ariane 5 had ∼3x more powerfull engines

Correct programs??

1996 Ariane 5 (ESA) rocket (software partially based on Ariane 4)

Original code of Ariane 4 **was** formally verified

But Ariane 5 had ∼3x more powerfull engines

Integer overflow occured

$$2'147'483'647 + 1 = -2'147'483'648$$

Correct programs??

1996 Ariane 5 (ESA) rocket (software partially based on Ariane 4)

Original code of Ariane 4 **was** formally verified

But Ariane 5 had ∼3x more powerfull engines

Integer overflow occured

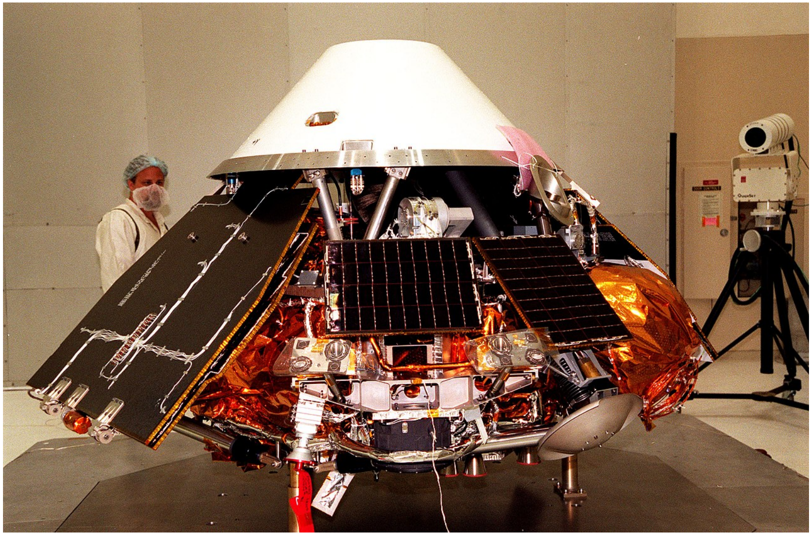
$$2'147'483'647 + 1 = -2'147'483'648$$

$\rightsquigarrow$ explosion in 30th second of flight, estimated loss of 442 milion €

Correct programs???

Correct programs???

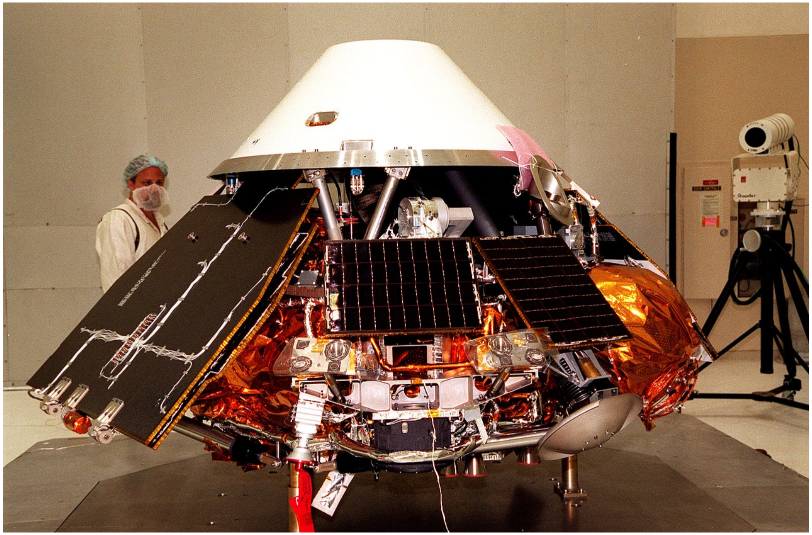
1999 Mars Polar Lander



Correct programs???

1999 Mars Polar Lander

Incorrect handling of sensor data from landing legs

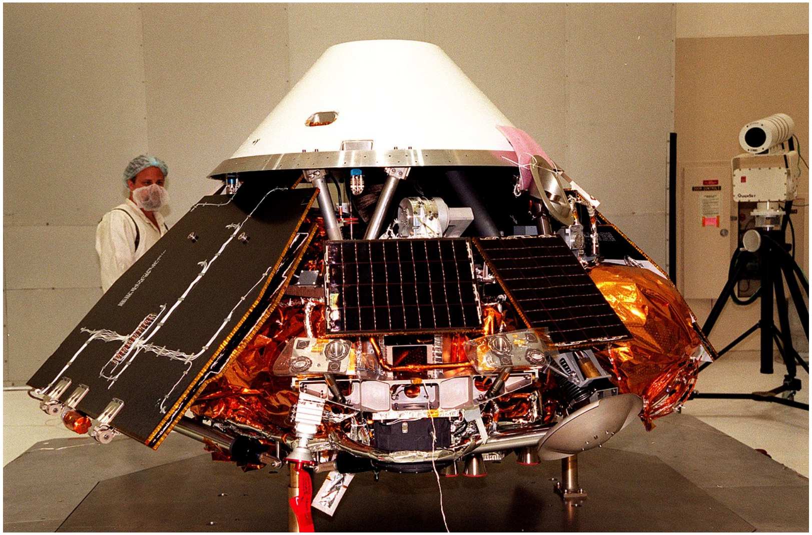


Correct programs???

1999 Mars Polar Lander

Incorrect handling of sensor data from landing legs

Spurious touchdown detection at 40 meters above surface



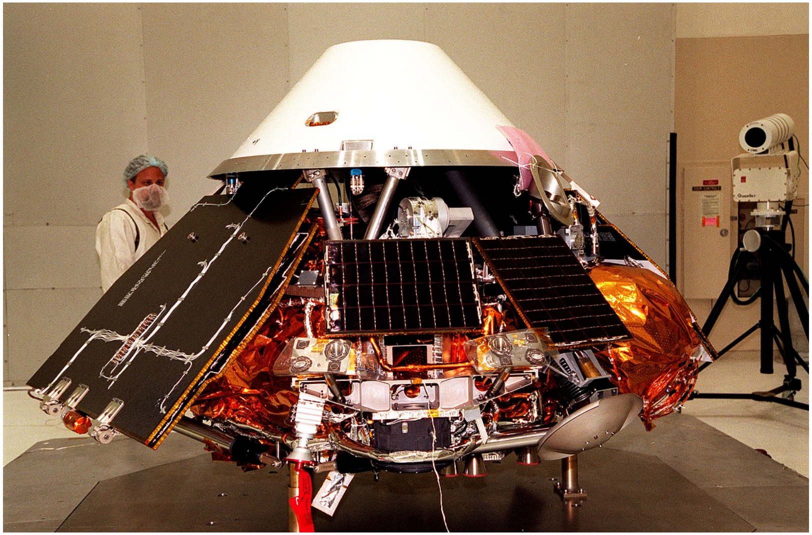
Correct programs???

1999 Mars Polar Lander

Incorrect handling of sensor data from landing legs

Spurious touchdown detection at 40 meters above surface

Premature engines shutdown



Correct programs???

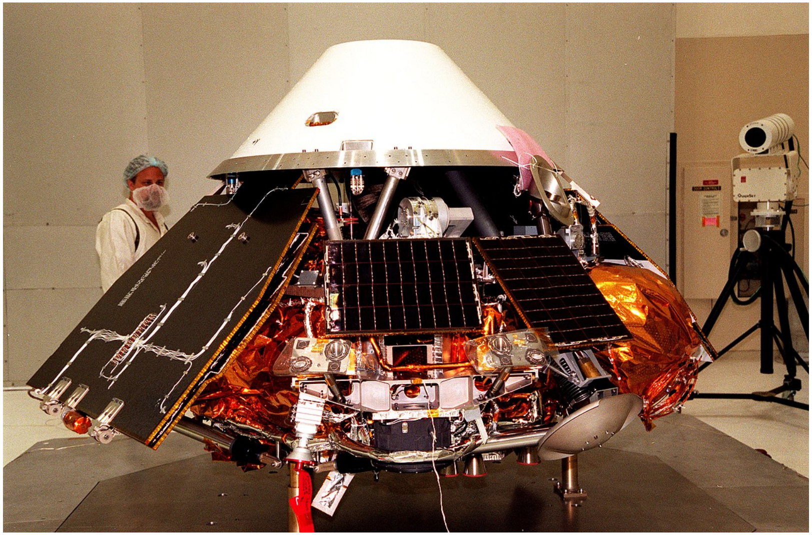
1999 Mars Polar Lander

Incorrect handling of sensor data from landing legs

Spurious touchdown detection at 40 meters above surface

Premature engines shutdown

⇝ impact at 22 m/s instead of 2.4 m/s, estimated loss of 100 milion $



Correct programs!!!

Correct programs!!!

Which science is always* right?

* Except some very rare cases…

Correct programs!!!

Which science is always* right?

MATHEMATICS!

* Except some very rare cases...

Correct programs!!!

Correct programs!!!

$\mathcal{P}$

program

Correct programs!!!

$$\underset{\text{program}}{\mathcal{P}} \quad : \quad \underset{\text{input}}{d}$$

Correct programs!!!

$$\underset{\text{program}}{\mathcal{P}} \; : \; \underset{\text{input}}{d} \; \xmapsto[\text{computation}]{\langle \rho_0, \rho_1, \ldots, \rho_n \rangle = \rho}$$

Correct programs!!!

$$\mathcal{P} \; : \; d \xmapsto{\langle \rho_0, \rho_1, \ldots, \rho_n \rangle = \rho} d'$$

program input computation result

Correct programs!!!

$$\mathcal{P} \;:\; d \xmapsto{\;\langle \rho_0, \rho_1, \ldots, \rho_n \rangle = \rho\;} d'$$

program input computation result

Fact 1: $\mathcal{P}$, d, ρ, and d' are sequences of **bits** $\rightsquigarrow$ **numbers**!

Correct programs!!!

$$\underset{\text{program}}{\mathcal{P}} \;:\; \underset{\text{input}}{d} \xmapsto[\text{computation}]{\langle \rho_0, \rho_1, \ldots, \rho_n \rangle = \rho} \underset{\text{result}}{d'}$$

Fact 1: $\mathcal{P}$, d, ρ, and d' are sequences of **bits** $\rightsquigarrow$ **numbers**!

Fact 2: There exists a **mathematical formula** expressing the above property.

Correct programs!!!

$$\underset{\text{program}}{\mathcal{P}} \;:\; \underset{\text{input}}{d} \;\xmapsto[\text{computation}]{\langle \rho_0, \rho_1, \ldots, \rho_n \rangle = \rho}\; \underset{\text{result}}{d'}$$

Fact 1: $\mathcal{P}$, d, ρ, and d' are sequences of **bits** $\rightsquigarrow$ **numbers**!

Fact 2: There exists a **mathematical formula** expressing the above property.

Therefore: the following condition is a **mathematical property**:

Correct programs!!!

$$\underset{\text{program}}{\mathcal{P}} \; : \; \underset{\text{input}}{d} \; \xmapsto[\text{computation}]{\langle \rho_0, \rho_1, \ldots, \rho_n \rangle = \rho} \; \underset{\text{result}}{d'}$$

Fact 1: $\mathcal{P}$, d, ρ, and d' are sequences of **bits** $\rightsquigarrow$ **numbers**!

Fact 2: There exists a **mathematical formula** expressing the above property.

Therefore: the following condition is a **mathematical property**:

$$\left(\begin{array}{ll} \textbf{IF} & \text{an input } d \text{ satisfies the assumptions } \varphi \\ \textbf{THEN} & \text{the result } d' \text{ satisfies the requirements } \psi. \end{array} \right)$$

Correct programs!!!

$$\mathcal{P} \ : \ d \xmapsto{\langle \rho_0, \rho_1, \ldots, \rho_n \rangle = \rho} d'$$

program input computation result

Fact 1: $\mathcal{P}$, d, ρ, and d' are sequences of **bits** $\rightsquigarrow$ **numbers**!

Fact 2: There exists a **mathematical formula** expressing the above property.

Therefore: the following condition is a **mathematical property**:

(

IF an input d satisfies the assumptions φ

THEN the result d' satisfies the requirements ψ.

shortly: $[\varphi]\,\mathcal{P}\,[\psi]$

)

Correct programs!!!

$$\mathcal{P} \quad : \quad d \quad \xmapsto{\langle \rho_0, \rho_1, \ldots, \rho_n \rangle = \rho} \quad d'$$

program input computation result

Fact 1: $\mathcal{P}$, d, ρ, and d' are sequences of **bits** $\rightsquigarrow$ **numbers**!

Fact 2: There exists a **mathematical formula** expressing the above property.

Therefore: the following condition is a **mathematical property**:

$$\left(\begin{array}{ll} \textbf{IF} & \text{an input } d \text{ satisfies the assumptions } \varphi \\ \textbf{THEN} & \text{the result } d' \text{ satisfies the requirements } \psi. \\ & \text{shortly: } [\varphi]\,\mathcal{P}\,[\psi] \end{array} \right)$$

For instance: $\left[d \geqslant 0\right] \mathcal{P}_{\mathrm{sqrt}} \left[\sqrt{d} - 1 < d' \leqslant \sqrt{d}\right]$

Correct programs!!!

$$\mathcal{P} \;:\; d \xmapsto{\langle \rho_0, \rho_1, \ldots, \rho_n \rangle = \rho} d'$$

program input computation result

Fact 1: $\mathcal{P}$, d, ρ, and d' are sequences of **bits** $\rightsquigarrow$ **numbers**!

Fact 2: There exists a **mathematical formula** expressing the above property.

Therefore: the following condition is a **mathematical property**:

$$\left(\begin{array}{ll} \textbf{IF} & \text{an input } d \text{ satisfies the assumptions } \varphi \\ \textbf{THEN} & \text{the result } d' \text{ satisfies the requirements } \psi. \\ & \text{shortly: } [\varphi]\,\mathcal{P}\,[\psi] \end{array}\right)$$

For instance: $[d \geqslant 0]\,\mathcal{P}_{\mathrm{sqrt}}\,[\sqrt{d} - 1 < d' \leqslant \sqrt{d}]$

So: such properties can be **proven!!!**

Correct programs!!!

$$\mathcal{P} \;:\; d \xmapsto{\langle \rho_0, \rho_1, \ldots, \rho_n \rangle = \rho} d'$$

program input computation result

Fact 1: $\mathcal{P}$, d, ρ, and d' are sequences of **bits** $\rightsquigarrow$ **numbers**!

Fact 2: There exists a **mathematical formula** expressing the above property.

Therefore: the following condition is a **mathematical property**:

$$\left(\begin{array}{ll} \textbf{IF} & \text{an input } d \text{ satisfies the assumptions } \varphi \\ \textbf{THEN} & \text{the result } d' \text{ satisfies the requirements } \psi. \\ & \text{shortly: } [\varphi]\,\mathcal{P}\,[\psi] \end{array}\right)$$

For instance: $[d \geqslant 0]\,\mathcal{P}_{\mathrm{sqrt}}\,[\sqrt{d} - 1 < d' \leqslant \sqrt{d}]$

So: such properties can be **proven!!!** [**Hoare** logic (1969)]

Correct programs!!!

$$\mathcal{P} \;:\; d \xmapsto{\;\langle \rho_0, \rho_1, \ldots, \rho_n \rangle = \rho\;} d'$$

program input computation result

Fact 1: $\mathcal{P}$, d, ρ, and d' are sequences of **bits** $\rightsquigarrow$ **numbers**!

Fact 2: There exists a **mathematical formula** expressing the above property.

Therefore: the following condition is a **mathematical property**:

$$\left(\begin{array}{ll} \textbf{IF} & \text{an input } d \text{ satisfies the assumptions } \varphi \\ \textbf{THEN} & \text{the result } d' \text{ satisfies the requirements } \psi. \\ & \text{shortly: } [\varphi]\,\mathcal{P}\,[\psi] \end{array} \right)$$

For instance: $[d \geqslant 0]\,\mathcal{P}_{\mathrm{sqrt}}\,[\sqrt{d} - 1 < d' \leqslant \sqrt{d}]$

So: such properties can be **proven!!!** [**Hoare** logic (1969)]

$\rightsquigarrow$ **formal verification** of programs

Formal verification workflow

Formal verification workflow

1. Write a **specification**: assumptions φ and requirements ψ

Formal verification workflow

1. Write a **specification**: assumptions φ and requirements ψ

2. Write an **implementation**: a program $\mathcal{P}$

Formal verification workflow

1. Write a **specification**: assumptions φ and requirements ψ

2. Write an **implementation**: a program $\mathcal{P}$

3. Design **invariants**: an annotatation $\widehat{\mathcal{P}}$ of $\mathcal{P}$

Formal verification workflow

1. Write a **specification**: assumptions φ and requirements ψ

2. Write an **implementation**: a program $\mathcal{P}$

3. Design **invariants**: an annotatation $\widehat{\mathcal{P}}$ of $\mathcal{P}$

$\Big[$ typically $\widehat{\mathcal{P}}$ is three times longer than $\mathcal{P}$ $\Big]$

Formal verification workflow

1. Write a **specification**: assumptions φ and requirements ψ

2. Write an **implementation**: a program $\mathcal{P}$

3. Design **invariants**: an annotatation $\widehat{\mathcal{P}}$ of $\mathcal{P}$

[typically $\widehat{\mathcal{P}}$ is three times longer than $\mathcal{P}$]

4. Automatically **verify** that

$\widehat{\mathcal{P}}$ **proves** that $[\varphi]\,\mathcal{P}\,[\psi]$ holds.

Formal verification workflow

1. Write a **specification**: assumptions φ and requirements ψ

2. Write an **implementation**: a program $\mathcal{P}$

3. Design **invariants**: an annotatation $\widehat{\mathcal{P}}$ of $\mathcal{P}$

[typically $\widehat{\mathcal{P}}$ is three times longer than $\mathcal{P}$]

4. Automatically **verify** that

$\widehat{\mathcal{P}}$ **proves** that $[\varphi]\,\mathcal{P}\,[\psi]$ holds.

...the program used for **4.** is short and simple and everyone trusts it...

Applications of formal verification

Applications of formal verification

- 1998 Ligne 14 of Paris Underground (autonomical)

  Key elements of the traffic-control system

Applications of formal verification

- 1998 Ligne 14 of Paris Underground (autonomical)

  Key elements of the traffic-control system

  – 110'000 lines of code (including proofs)

  – 0 bugs found until 2009

Applications of formal verification

- 1998 Ligne 14 of Paris Underground (autonomical)

  Key elements of the traffic-control system

  – 110'000 lines of code (including proofs)

  – 0 bugs found until 2009

- 1999 formal verification of a *smart cards* interpreter

Applications of formal verification

- 1998 Ligne 14 of Paris Underground (autonomical)

  Key elements of the traffic-control system

  – 110'000 lines of code (including proofs)

  – 0 bugs found until 2009

- 1999 formal verification of a *smart cards* interpreter

- 2005 verification of VAL system, connecting terminals at CDG

Applications of formal verification

- 1998 Ligne 14 of Paris Underground (autonomical)

  Key elements of the traffic-control system

  – 110'000 lines of code (including proofs)

  – 0 bugs found until 2009

- 1999 formal verification of a *smart cards* interpreter

- 2005 verification of VAL system, connecting terminals at CDG

  $\rightsquigarrow$ $\geqslant$ 20 other locations around the globe

Applications of formal verification

- 1998 Ligne 14 of Paris Underground (autonomical)

  Key elements of the traffic-control system

  – 110’000 lines of code (including proofs)

  – 0 bugs found until 2009

- 1999 formal verification of a *smart cards* interpreter

- 2005 verification of VAL system, connecting terminals at CDG

  $\rightsquigarrow$ $\geqslant 20$ other locations around the globe

B-method

Applications of formal verification

- 1998 Ligne 14 of Paris Underground (autonomical)

  Key elements of the traffic-control system

  – 110'000 lines of code (including proofs)

  – 0 bugs found until 2009

- 1999 formal verification of a *smart cards* interpreter

- 2005 verification of VAL system, connecting terminals at CDG

  $\rightsquigarrow$ $\geqslant$ 20 other locations around the globe

B-method

- 1997–99 full verification of Intel Pentium 4 CPU

Applications of formal verification

- 1998 Ligne 14 of Paris Underground (autonomical)

  Key elements of the traffic-control system

  – 110'000 lines of code (including proofs)

  – 0 bugs found until 2009

- 1999 formal verification of a *smart cards* interpreter

- 2005 verification of VAL system, connecting terminals at CDG

  $\rightsquigarrow$ $\geqslant$ 20 other locations around the globe

B-method

- 1997–99 full verification of Intel Pentium 4 CPU

- 2005 multiple subsystems of Airbus A380

Applications of formal verification

- 1998 Ligne 14 of Paris Underground (autonomical)

  Key elements of the traffic-control system

  – 110'000 lines of code (including proofs)

  – 0 bugs found until 2009

- 1999 formal verification of a *smart cards* interpreter

- 2005 verification of VAL system, connecting terminals at CDG

  $\rightsquigarrow$ $\geqslant$ 20 other locations around the globe

B-method

- 1997–99 full verification of Intel Pentium 4 CPU

- 2005 multiple subsystems of Airbus A380

- 2018 complete verification of Amazon's implementation of TLS protocol

Applications of formal verification

- 1998 Ligne 14 of Paris Underground (autonomical)

  Key elements of the traffic-control system

  – 110’000 lines of code (including proofs)

  – 0 bugs found until 2009

- 1999 formal verification of a *smart cards* interpreter

- 2005 verification of VAL system, connecting terminals at CDG

  $\rightsquigarrow$ $\geqslant$ 20 other locations around the globe

B-method

- 1997–99 full verification of Intel Pentium 4 CPU

- 2005 multiple subsystems of Airbus A380

- 2018 complete verification of Amazon’s implementation of TLS protocol

- . . .

No such thing as a free lunch. . .

No such thing as a free lunch...

Conjecture (Goldbach [1742]) [AKA Hilbert's 8th problem]

Every even number greater than 2 is a sum of two primes.

No such thing as a free lunch. . .

Conjecture (Goldbach [1742]) [AKA Hilbert's 8th problem]

Every even number greater than 2 is a sum of two primes.

[worth 1 milion $]

No such thing as a free lunch...

Conjecture (Goldbach [1742]) [AKA Hilbert's 8th problem]

Every even number greater than 2 is a sum of two primes.

[worth 1 milion \$]

$\mathcal{P}_{\text{Gold}}$:

No such thing as a free lunch...

Conjecture (Goldbach [1742]) [AKA Hilbert's 8th problem]

Every even number greater than 2 is a sum of two primes.

[worth 1 milion $]

$\mathcal{P}_{\mathrm{Gold}}$:

```
n := 2;
while true do {
    n := n + 2;
    if (n is not a sum of two primes) then
        return 1;
}
```

No such thing as a free lunch...

Conjecture (Goldbach [1742]) [AKA Hilbert's 8th problem]

Every even number greater than 2 is a sum of two primes.

[worth 1 milion \$]

$\mathcal{P}_{\text{Gold}}$:
```
n := 2;
while true do {
    n := n + 2;
    if (n is not a sum of two primes) then
        return 1;
}
```

Fact: $\neg\big[\textbf{Goldbach Conjecture}\big] \iff [\,]\,\mathcal{P}_{\text{Gold}}\,[d' = 1]$

No such thing as a free lunch...

Conjecture (Goldbach [1742]) [AKA Hilbert's 8th problem]

Every even number greater than 2 is a sum of two primes.

[worth 1 milion \$]

$\mathcal{P}_{\text{Gold}}$:
```
n := 2;
while true do {
    n := n + 2;
    if (n is not a sum of two primes) then
        return 1;
}
```

Fact: $\neg\big[\textbf{Goldbach Conjecture}\big] \iff [\,]\,\mathcal{P}_{\text{Gold}}\,[d' = 1]$

$\rightsquigarrow$ it is ***enough*** to show that $\neg[\,]\,\mathcal{P}_{\text{Gold}}\,[d' = 1]$...

No such thing as a free lunch...

Conjecture (Goldbach [1742]) [AKA Hilbert's 8th problem]

Every even number greater than 2 is a sum of two primes.

[worth 1 milion \$]

$\mathcal{P}_{\text{Gold}}$:
```
n := 2;
while true do {
    n := n + 2;
    if (n is not a sum of two primes) then
        return 1;
}
```

Fact: $\neg\big[\textbf{Goldbach Conjecture}\big] \iff [\,]\,\mathcal{P}_{\text{Gold}}\,\big[d' = 1\big]$

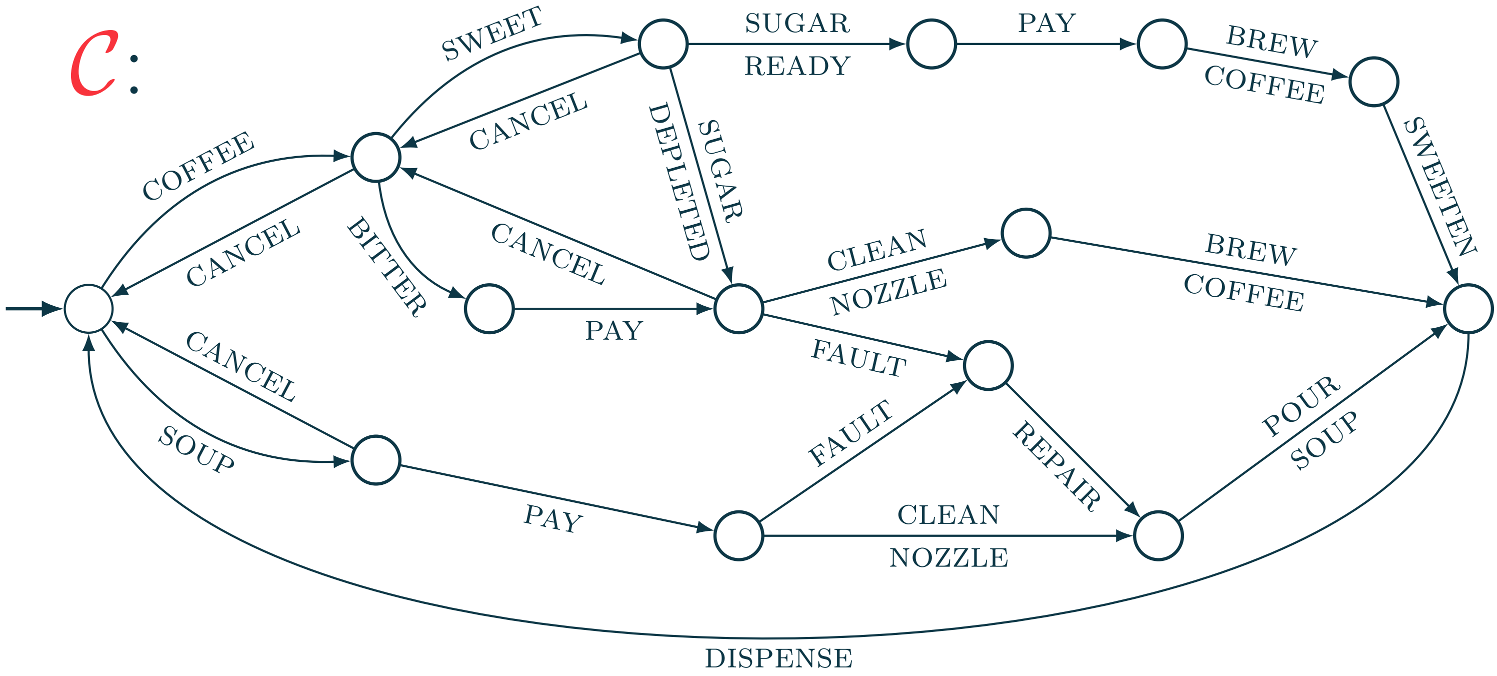
$\rightsquigarrow$ it is ***enough*** to show that $\neg[\,]\,\mathcal{P}_{\text{Gold}}\,\big[d' = 1\big]\ldots$

[even worse, as a program can enumerate proofs]

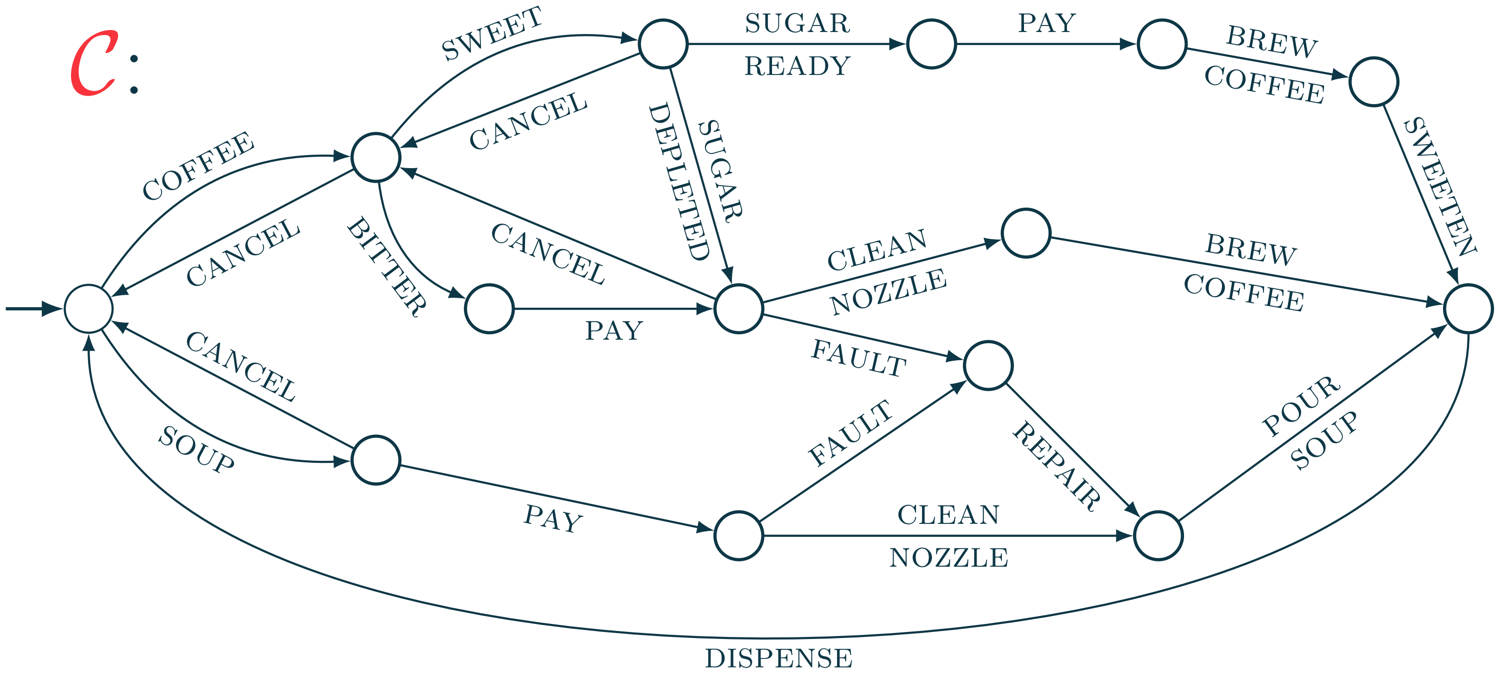
„It is all because of numbers”

„It is all because of numbers” $\rightsquigarrow$ consider *numberless* machines = **automata**

$\mathcal{C}$:

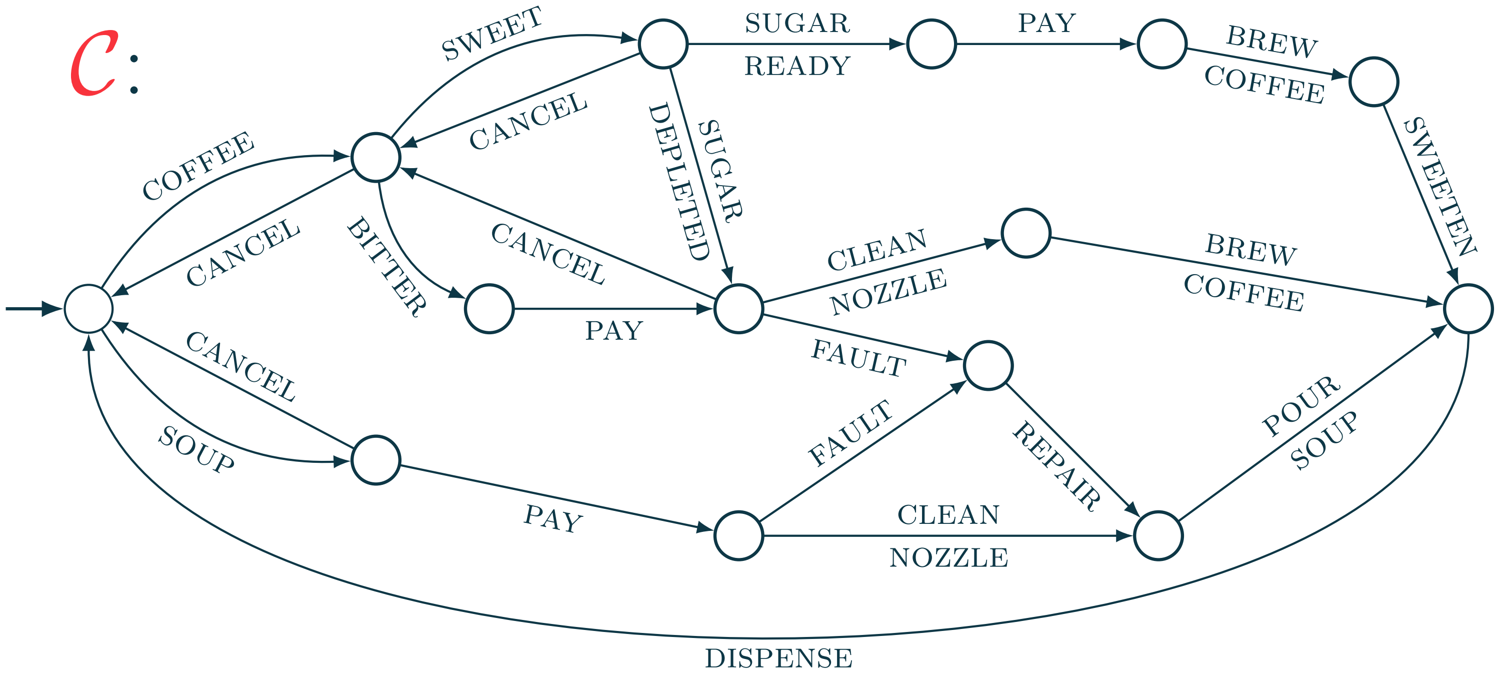


SWEET
SUGAR
READY
PAY
BREW
COFFEE
CANCEL
COFFEE
SUGAR
DEPLETED
SWEETEN
CANCEL
BITTER
CANCEL
CLEAN
NOZZLE
BREW
COFFEE
PAY
FAULT
CANCEL
FAULT
REPAIR
POUR
SOUP
SOUP
PAY
CLEAN
NOZZLE
DISPENSE



Set of actions: $\textcolor{red}{\mathbf{A}} = \{\text{COFFEE}, \text{SOUP}, \text{PAY}, \ldots\}$

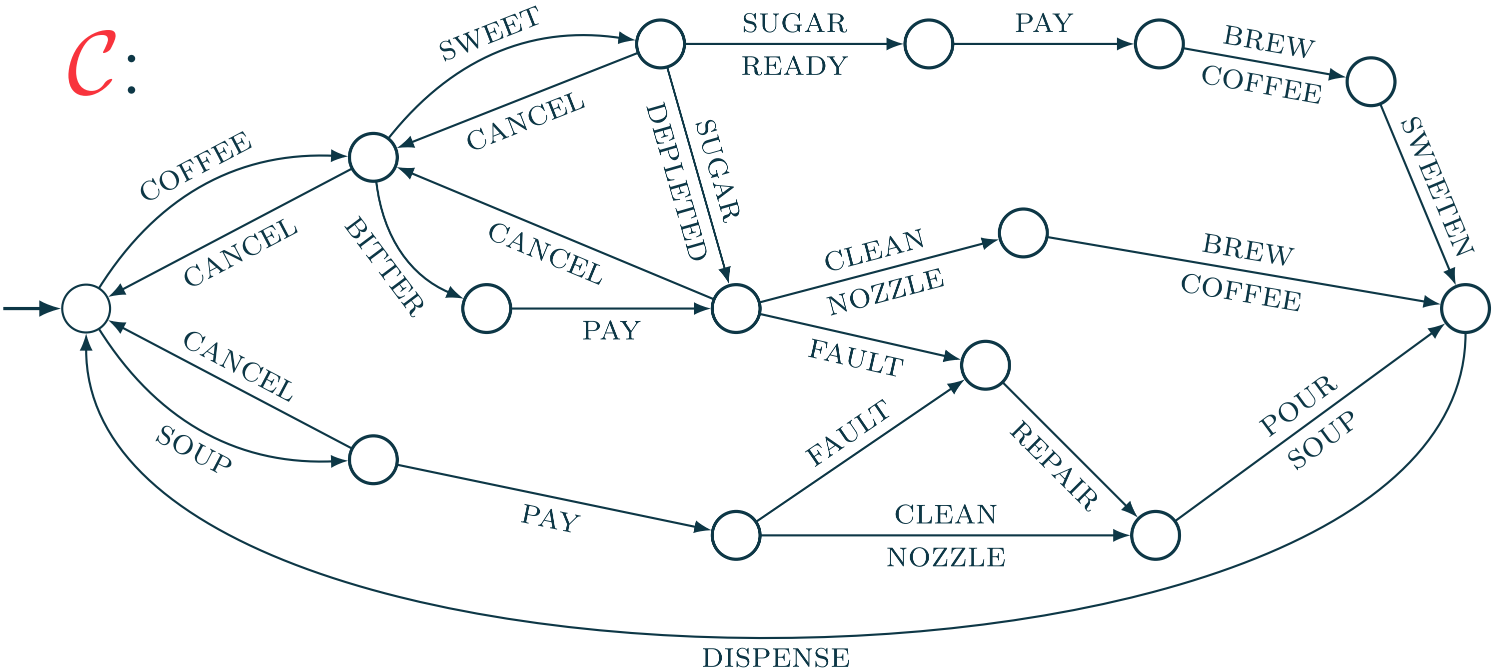
$\mathcal{C}$:



Set of actions: $\mathbf{A} = \{\text{COFFEE, SOUP, PAY, } \ldots\}$

Possible executions: $\mathbf{A}^* \supseteq [\![\mathcal{C}]\!]$

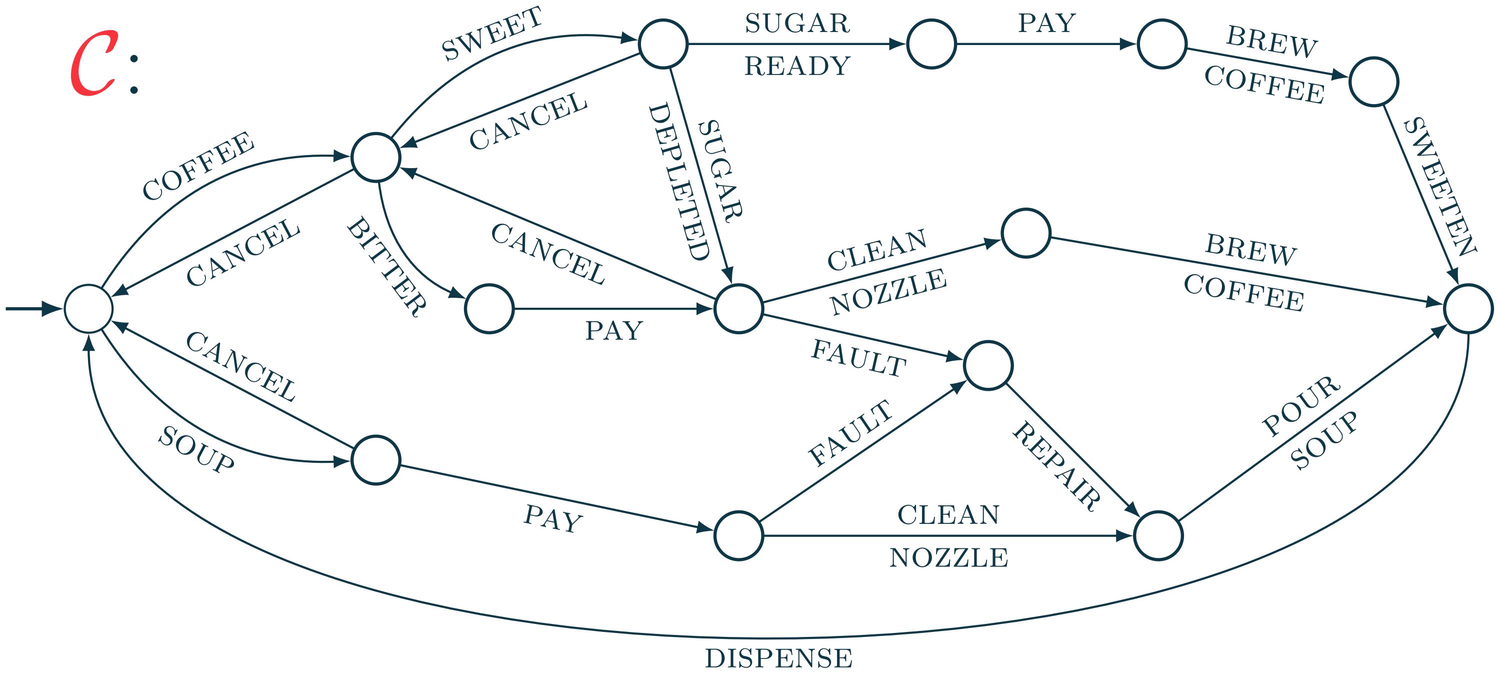
$\mathcal{C}$:



Set of actions: $\mathbf{A} = \{\text{COFFEE}, \text{SOUP}, \text{PAY}, \ldots\}$

Possible executions: $\mathbf{A}^* \supseteq [\![\mathcal{C}]\!] \ni \langle \text{SOUP}, \text{PAY}, \text{FAULT}, \text{REPAIR} \rangle$

$\mathcal{C}$:

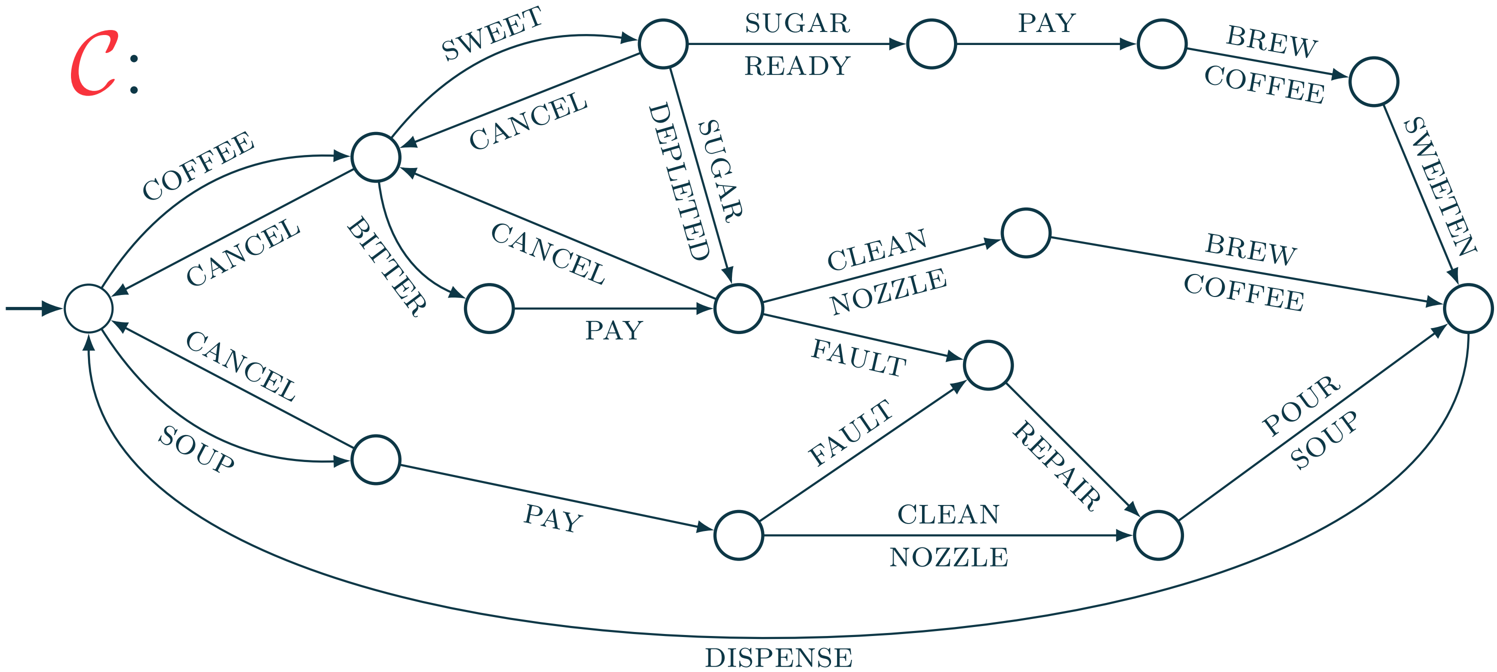


Set of actions: $\mathbf{A} = \{\text{COFFEE}, \text{SOUP}, \text{PAY}, \ldots\}$

Possible executions: $\mathbf{A}^* \supseteq [\![\mathcal{C}]\!] \ni \langle \text{SOUP}, \text{PAY}, \text{FAULT}, \text{REPAIR} \rangle$

Specification: $\mathcal{S}$: “no DISPENSE without prior PAY”

$\mathcal{C}$:

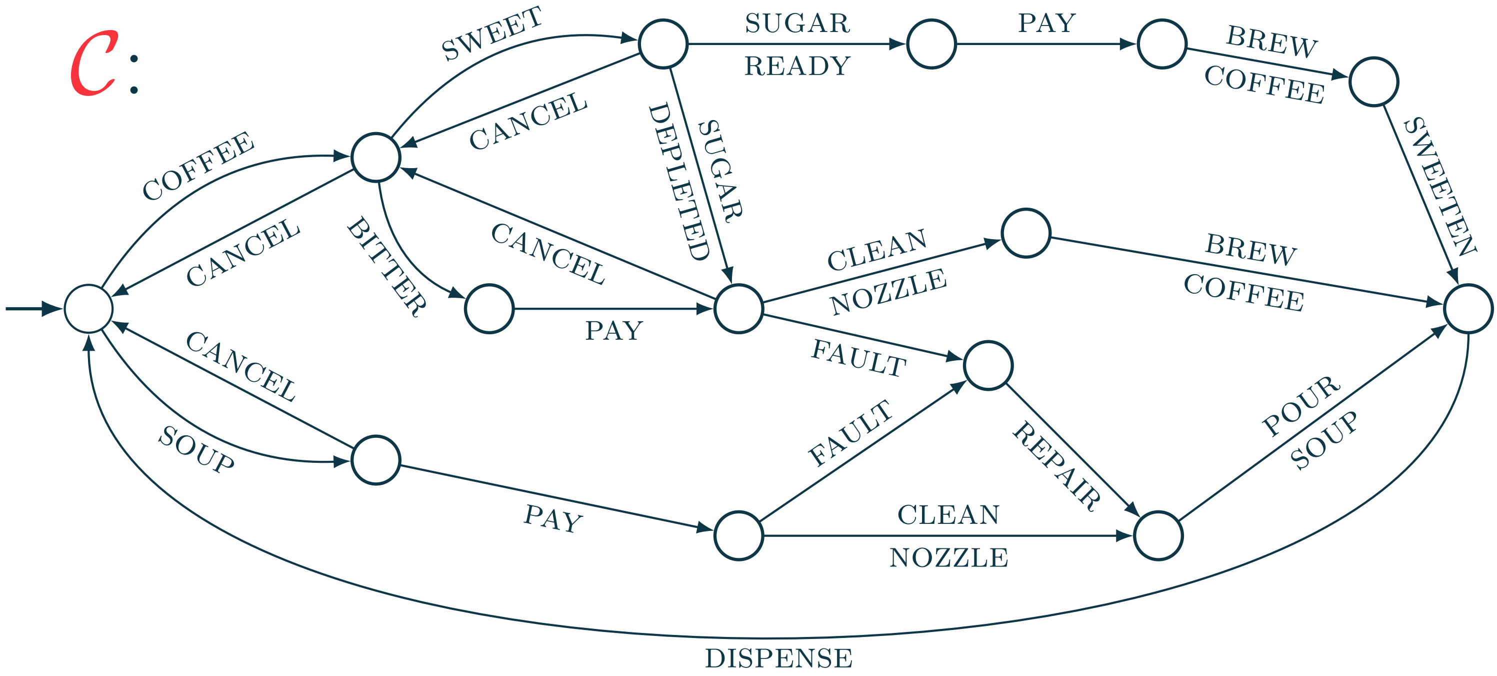


Set of actions: $\mathbf{A} = \{\text{COFFEE}, \text{SOUP}, \text{PAY}, \ldots\}$

Possible executions: $\mathbf{A}^* \supseteq [\![\mathcal{C}]\!] \ni \langle \text{SOUP}, \text{PAY}, \text{FAULT}, \text{REPAIR} \rangle$

Specification: $\mathcal{S}$: "no DISPENSE without prior PAY"

Correct executions: $\mathbf{A}^* \supseteq [\![\mathcal{S}]\!]$



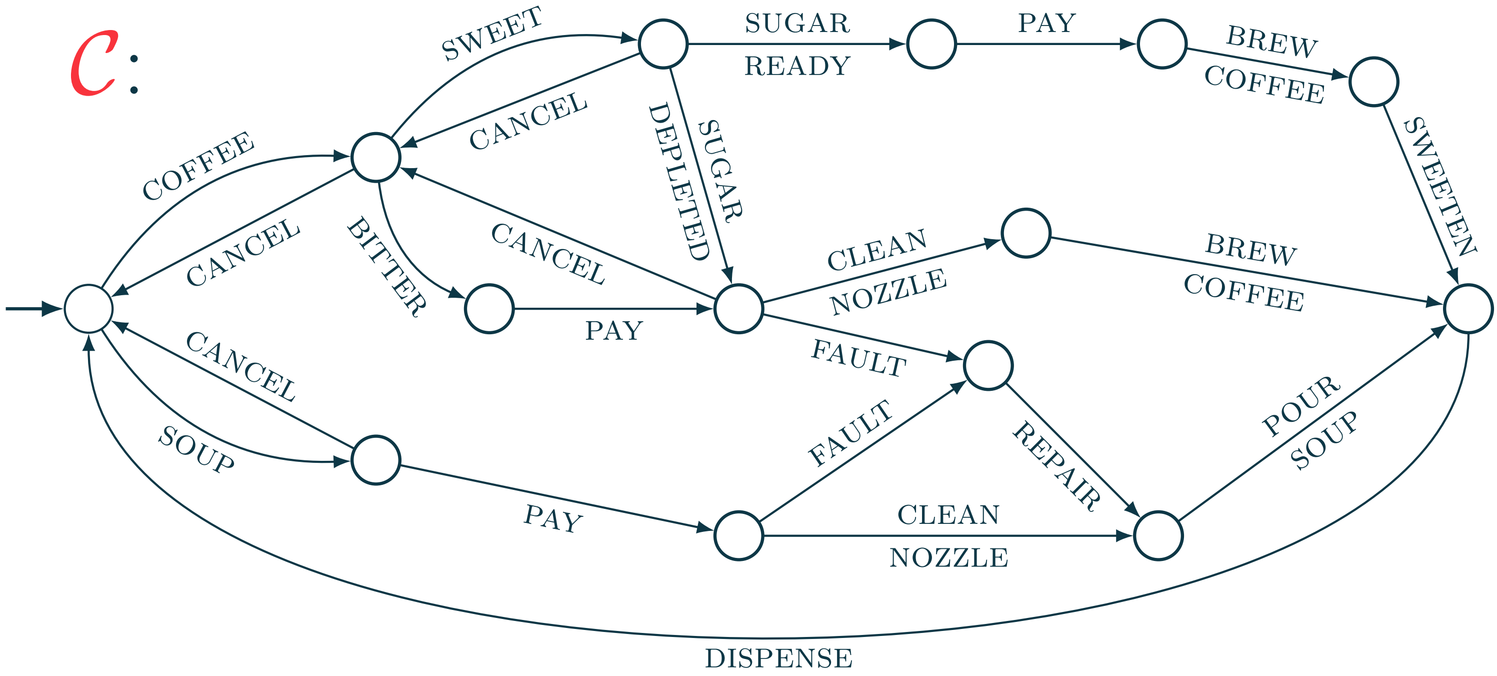
Set of actions: $\mathbf{A} = \{\text{COFFEE}, \text{SOUP}, \text{PAY}, \ldots\}$

Possible executions: $\mathbf{A}^* \supseteq [\![\mathcal{C}]\!] \ni \langle \text{SOUP}, \text{PAY}, \text{FAULT}, \text{REPAIR} \rangle$

Specification: $\mathcal{S}$: "no DISPENSE without prior PAY"

Correct executions: $\mathbf{A}^* \supseteq [\![\mathcal{S}]\!] \ni \langle \text{PAY}, \text{FAULT}, \text{DISPENSE} \rangle$

$\mathcal{C}$:



Set of actions: $\mathbf{A} = \{\text{COFFEE}, \text{SOUP}, \text{PAY}, \ldots\}$

Possible executions: $\mathbf{A}^* \supseteq [\![\mathcal{C}]\!] \ni \langle \text{SOUP}, \text{PAY}, \text{FAULT}, \text{REPAIR} \rangle$

Specification: $\mathcal{S}$: "no DISPENSE without prior PAY"

Correct executions: $\mathbf{A}^* \supseteq [\![\mathcal{S}]\!] \ni \langle \text{PAY}, \text{FAULT}, \text{DISPENSE} \rangle$

Model-checking problem:

$$[\![\mathcal{C}]\!] \overset{???}{\subseteq} [\![\mathcal{S}]\!]$$

Specification: $\mathcal{S}$: “no DISPENSE without prior PAY”

Specification: $\mathcal{S}$: "no DISPENSE without prior PAY"

---

1. Formula of MSO: φ: $\forall t.\ \textsc{dispense}(t) \Rightarrow \exists t' < t.\ \textsc{pay}(t')$

Specification: $\mathcal{S}$: "no DISPENSE without prior PAY"

---

1. Formula of MSO: φ: $\forall t.\ \textsc{dispense}(t) \Rightarrow \exists t' < t.\ \textsc{pay}(t')$

$$[\![\varphi]\!] \stackrel{\text{def}}{=} \{\rho \in \mathbf{A}^* \mid \rho \text{ satisfies } \varphi\}$$

Specification: $\mathcal{S}$: "no DISPENSE without prior PAY"

---

1. Formula of MSO: φ: $\forall t.\ \text{DISPENSE}(t) \Rightarrow \exists t' < t.\ \text{PAY}(t')$

$[\![\varphi]\!] \stackrel{\text{def}}{=} \{\rho \in \mathbf{A}^* \mid \rho \text{ satisfies } \varphi\}$

---

2. Regular expression: R: $[\hat{}\,\text{DISPENSE}]^* + \left([\hat{}\,\text{DISPENSE}]^* \cdot \text{PAY} \cdot \mathbf{A}^*\right)$

Specification: $\mathcal{S}$: "no DISPENSE without prior PAY"

---

1. Formula of MSO: φ: $\forall t.\ \text{DISPENSE}(t) \Rightarrow \exists t' < t.\ \text{PAY}(t')$

$$[\![\varphi]\!] \stackrel{\text{def}}{=} \{\rho \in \mathbf{A}^* \mid \rho \text{ satisfies } \varphi\}$$

---

2. Regular expression: R: $[\hat{}\,\text{DISPENSE}]^* + \big([\hat{}\,\text{DISPENSE}]^* \cdot \text{PAY} \cdot \mathbf{A}^*\big)$

$$[\![R_1 + R_2]\!] \stackrel{\text{def}}{=} [\![R_1]\!] \cup [\![R_2]\!], \ldots$$

Specification: $\mathcal{S}$: "no DISPENSE without prior PAY"

---

1. Formula of MSO: φ: $\forall t.\ \textsc{dispense}(t) \Rightarrow \exists t' < t.\ \textsc{pay}(t')$

$[\![\varphi]\!] \stackrel{\text{def}}{=} \{\rho \in \mathbf{A}^* \mid \rho \text{ satisfies } \varphi\}$

---

2. Regular expression: R: $[\hat{}\,\textsc{dispense}]^* + \big([\hat{}\,\textsc{dispense}]^* \cdot \textsc{pay} \cdot \mathbf{A}^*\big)$

$[\![R_1 + R_2]\!] \stackrel{\text{def}}{=} [\![R_1]\!] \cup [\![R_2]\!], \ldots$

---

3. Verifier:

Specification: $\mathcal{S}$: “no DISPENSE without prior PAY”

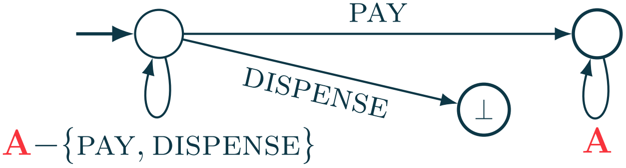
1. Formula of MSO: φ: $\forall t.\ \text{DISPENSE}(t) \Rightarrow \exists t' < t.\ \text{PAY}(t')$

$$[\![\varphi]\!] \stackrel{\text{def}}{=} \{\rho \in \mathbf{A}^* \mid \rho \text{ satisfies } \varphi\}$$

2. Regular expression: R: $[\hat{\ }\text{DISPENSE}]^* + \big([\hat{\ }\text{DISPENSE}]^* \cdot \text{PAY} \cdot \mathbf{A}^*\big)$

$$[\![R_1 + R_2]\!] \stackrel{\text{def}}{=} [\![R_1]\!] \cup [\![R_2]\!], \ldots$$

3. Verifier: $\mathcal{V}$:



Specification: $\mathcal{S}$: "no DISPENSE without prior PAY"

---

1. Formula of MSO: φ: $\forall t.\ \textsc{dispense}(t) \Rightarrow \exists t' < t.\ \textsc{pay}(t')$

$$[\![\varphi]\!] \stackrel{\text{def}}{=} \{\rho \in \mathbf{A}^* \mid \rho \text{ satisfies } \varphi\}$$

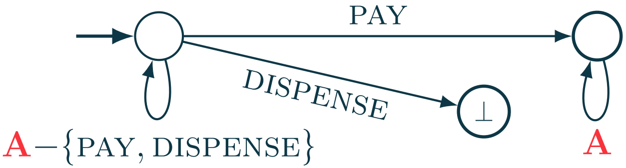
---

2. Regular expression: R: $[\hat{}\,\textsc{dispense}]^* + \big([\hat{}\,\textsc{dispense}]^* \cdot \textsc{pay} \cdot \mathbf{A}^*\big)$

$$[\![R_1 + R_2]\!] \stackrel{\text{def}}{=} [\![R_1]\!] \cup [\![R_2]\!], \ldots$$

---

3. Verifier: $\mathcal{V}$:



$$[\![\mathcal{V}]\!] \stackrel{\text{def}}{=} \{\rho \in \mathbf{A}^* \mid \mathcal{V} \text{ accepts } \rho\}$$

Specification: $\mathcal{S}$: "no DISPENSE without prior PAY"

---

1. Formula of MSO: φ: $\forall t.\ \textsc{dispense}(t) \Rightarrow \exists t' < t.\ \textsc{pay}(t')$

$$[\![\varphi]\!] \stackrel{\text{def}}{=} \{\rho \in \mathbf{A}^* \mid \rho \text{ satisfies } \varphi\}$$

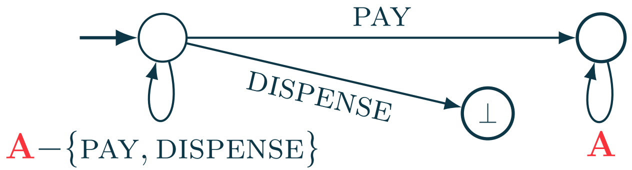
---

2. Regular expression: R: $[\hat{\ }\textsc{dispense}]^* + \left([\hat{\ }\textsc{dispense}]^* \cdot \textsc{pay} \cdot \mathbf{A}^*\right)$

$$[\![R_1 + R_2]\!] \stackrel{\text{def}}{=} [\![R_1]\!] \cup [\![R_2]\!], \ldots$$

---

3. Verifier: $\mathcal{V}$:



$$[\![\mathcal{V}]\!] \stackrel{\text{def}}{=} \{\rho \in \mathbf{A}^* \mid \mathcal{V} \text{ accepts } \rho\}$$

Theorem (Büchi, Elgot, Trachtenbrot [~1960])

Specification: $\mathcal{S}$: "no DISPENSE without prior PAY"

---

1. Formula of MSO: φ: $\forall t.\ \textsc{dispense}(t) \Rightarrow \exists t' < t.\ \textsc{pay}(t')$

$$[\![\varphi]\!] \stackrel{\text{def}}{=} \{\rho \in \mathbf{A}^* \mid \rho \text{ satisfies } \varphi\}$$

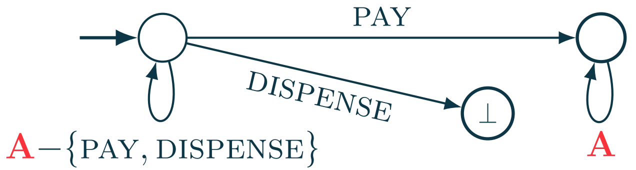
---

2. Regular expression: R: $[\hat{}\,\textsc{dispense}]^* + \big([\hat{}\,\textsc{dispense}]^* \cdot \textsc{pay} \cdot \mathbf{A}^*\big)$

$$[\![R_1 + R_2]\!] \stackrel{\text{def}}{=} [\![R_1]\!] \cup [\![R_2]\!], \ldots$$

---

3. Verifier: $\mathcal{V}$:



$$[\![\mathcal{V}]\!] \stackrel{\text{def}}{=} \{\rho \in \mathbf{A}^* \mid \mathcal{V} \text{ accepts } \rho\}$$

Theorem (Büchi, Elgot, Trachtenbrot [∼1960])

It is possible to effectively translate between **1.**, **2.**, and **3.**.

Specification: $\mathcal{S}$: "no DISPENSE without prior PAY"

---

1. Formula of MSO: φ: $\forall t.\ \text{DISPENSE}(t) \Rightarrow \exists t' < t.\ \text{PAY}(t')$

$$[\![\varphi]\!] \stackrel{\text{def}}{=} \{\rho \in \mathbf{A}^* \mid \rho \text{ satisfies } \varphi\}$$

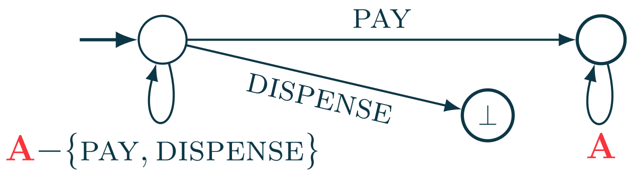
---

2. Regular expression: R: $[\hat{}\,\text{DISPENSE}]^* + \big([\hat{}\,\text{DISPENSE}]^* \cdot \text{PAY} \cdot \mathbf{A}^*\big)$

$$[\![R_1 + R_2]\!] \stackrel{\text{def}}{=} [\![R_1]\!] \cup [\![R_2]\!], \ldots$$

---

3. Verifier: $\mathcal{V}$:



$$[\![\mathcal{V}]\!] \stackrel{\text{def}}{=} \{\rho \in \mathbf{A}^* \mid \mathcal{V} \text{ accepts } \rho\}$$

Theorem (Büchi, Elgot, Trachtenbrot [∼1960])

It is possible to effectively translate between **1.**, **2.**, and **3.**.

Application: To check if $[\![\mathcal{C}]\!] \subseteq [\![\mathcal{S}]\!]$ it is enough to

Specification: $\mathcal{S}$: "no DISPENSE without prior PAY"

---

1. Formula of MSO: φ: $\forall t.\ \textsc{dispense}(t) \Rightarrow \exists t' < t.\ \textsc{pay}(t')$

$$[\![\varphi]\!] \stackrel{\text{def}}{=} \{\rho \in \mathbf{A}^* \mid \rho \text{ satisfies } \varphi\}$$

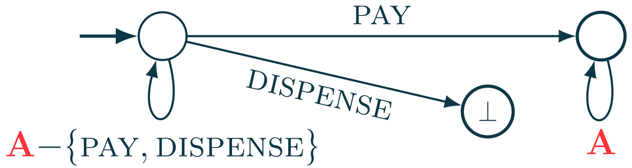
---

2. Regular expression: R: $[\hat{}\,\textsc{dispense}]^* + \left([\hat{}\,\textsc{dispense}]^* \cdot \textsc{pay} \cdot \mathbf{A}^*\right)$

$$[\![R_1 + R_2]\!] \stackrel{\text{def}}{=} [\![R_1]\!] \cup [\![R_2]\!], \ldots$$

---

3. Verifier: $\mathcal{V}$:



$$[\![\mathcal{V}]\!] \stackrel{\text{def}}{=} \{\rho \in \mathbf{A}^* \mid \mathcal{V} \text{ accepts } \rho\}$$

Theorem (Büchi, Elgot, Trachtenbrot [~1960])

It is possible to effectively translate between **1.**, **2.**, and **3.**.

Application: To check if $[\![\mathcal{C}]\!] \subseteq [\![\mathcal{S}]\!]$ it is enough to

1. Translate $\mathcal{S}$ into $\mathcal{V}$

Specification: $\mathcal{S}$: “no DISPENSE without prior PAY”

---

1. Formula of MSO: φ: $\forall t.\ \textsc{dispense}(t) \Rightarrow \exists t' < t.\ \textsc{pay}(t')$

$$[\![\varphi]\!] \stackrel{\text{def}}{=} \{\rho \in \mathbf{A}^* \mid \rho \text{ satisfies } \varphi\}$$

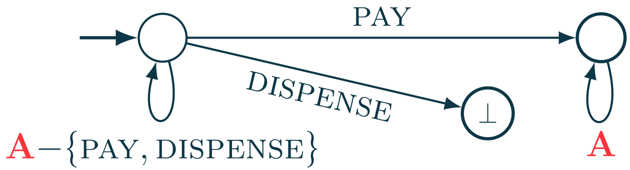
---

2. Regular expression: R: $[\hat{}\,\textsc{dispense}]^* + \big([\hat{}\,\textsc{dispense}]^* \cdot \textsc{pay} \cdot \mathbf{A}^*\big)$

$$[\![R_1 + R_2]\!] \stackrel{\text{def}}{=} [\![R_1]\!] \cup [\![R_2]\!], \ldots$$

---

3. Verifier: $\mathcal{V}$:



$$[\![\mathcal{V}]\!] \stackrel{\text{def}}{=} \{\rho \in \mathbf{A}^* \mid \mathcal{V} \text{ accepts } \rho\}$$

Theorem (Büchi, Elgot, Trachtenbrot [~1960])

It is possible to effectively translate between **1.**, **2.**, and **3.**.

Application: To check if $[\![\mathcal{C}]\!] \subseteq [\![\mathcal{S}]\!]$ it is enough to

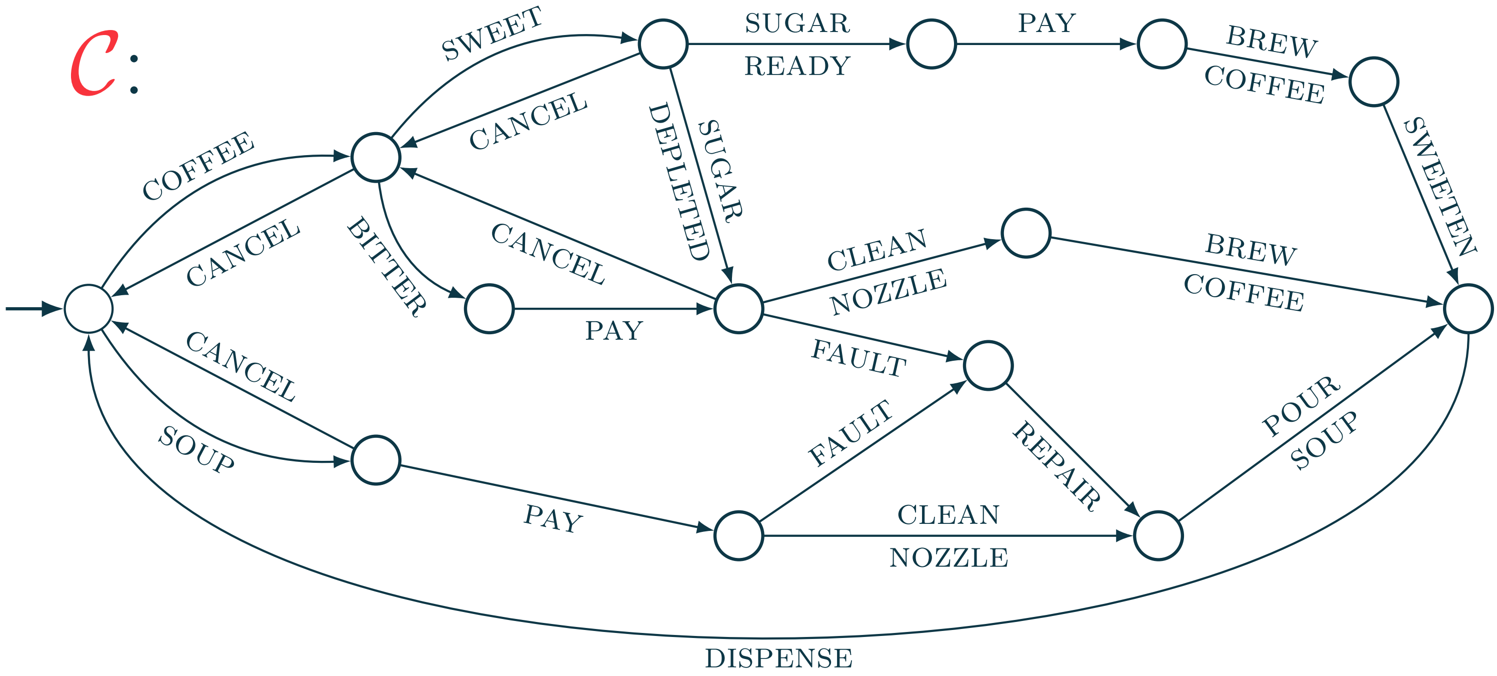
1. Translate $\mathcal{S}$ into $\mathcal{V}$

2. Check if

$$\mathcal{C} \times \mathcal{V} \longrightarrow^* (\_, \bot)$$

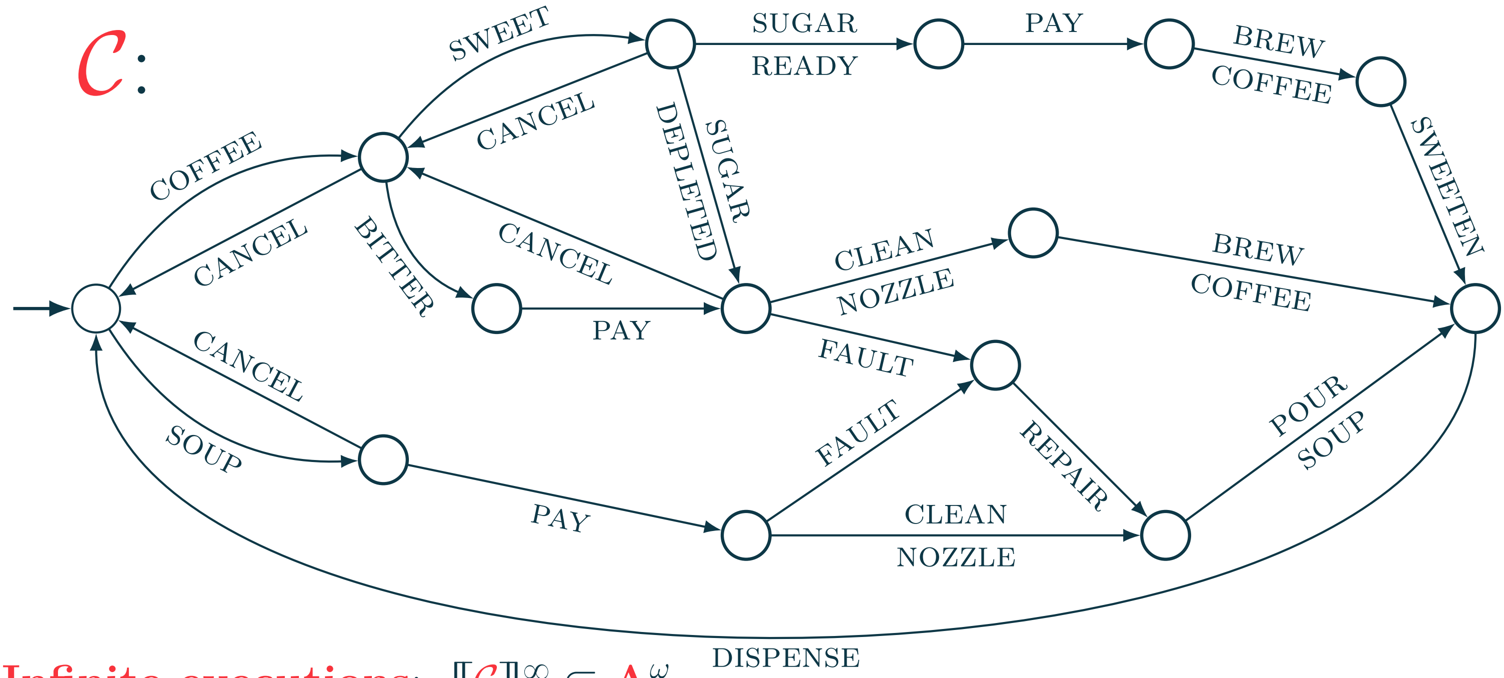
If that's not enough. . .

$\mathcal{C}$:



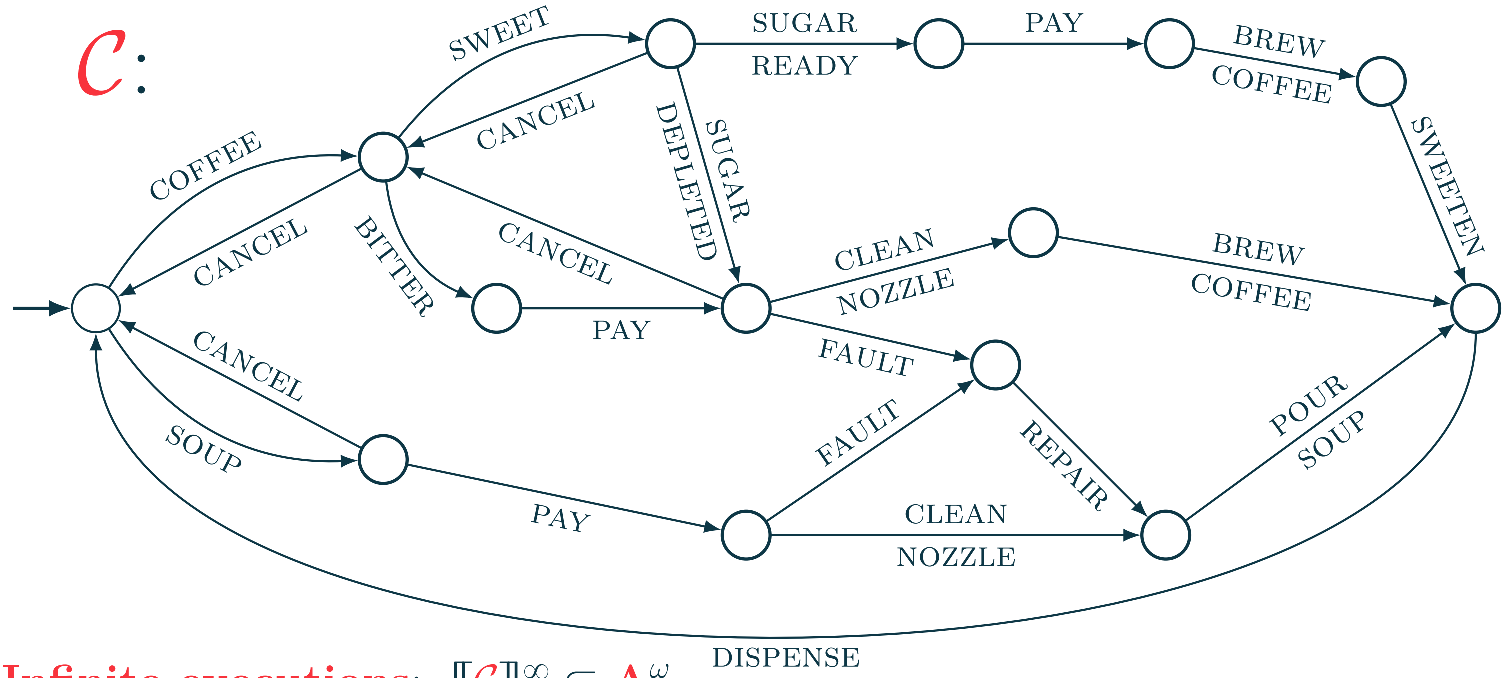
SWEET
SUGAR
READY
PAY
BREW
COFFEE
CANCEL
DEPLETED
SUGAR
SWEETEN
COFFEE
CANCEL
BITTER
CANCEL
CLEAN
NOZZLE
BREW
COFFEE
PAY
FAULT
CANCEL
FAULT
REPAIR
POUR
SOUP
SOUP
PAY
CLEAN
NOZZLE
DISPENSE

$\mathcal{C}$:



Infinite executions: $[\![\mathcal{C}]\!]^{\infty} \subseteq \mathbf{A}^{\omega}$

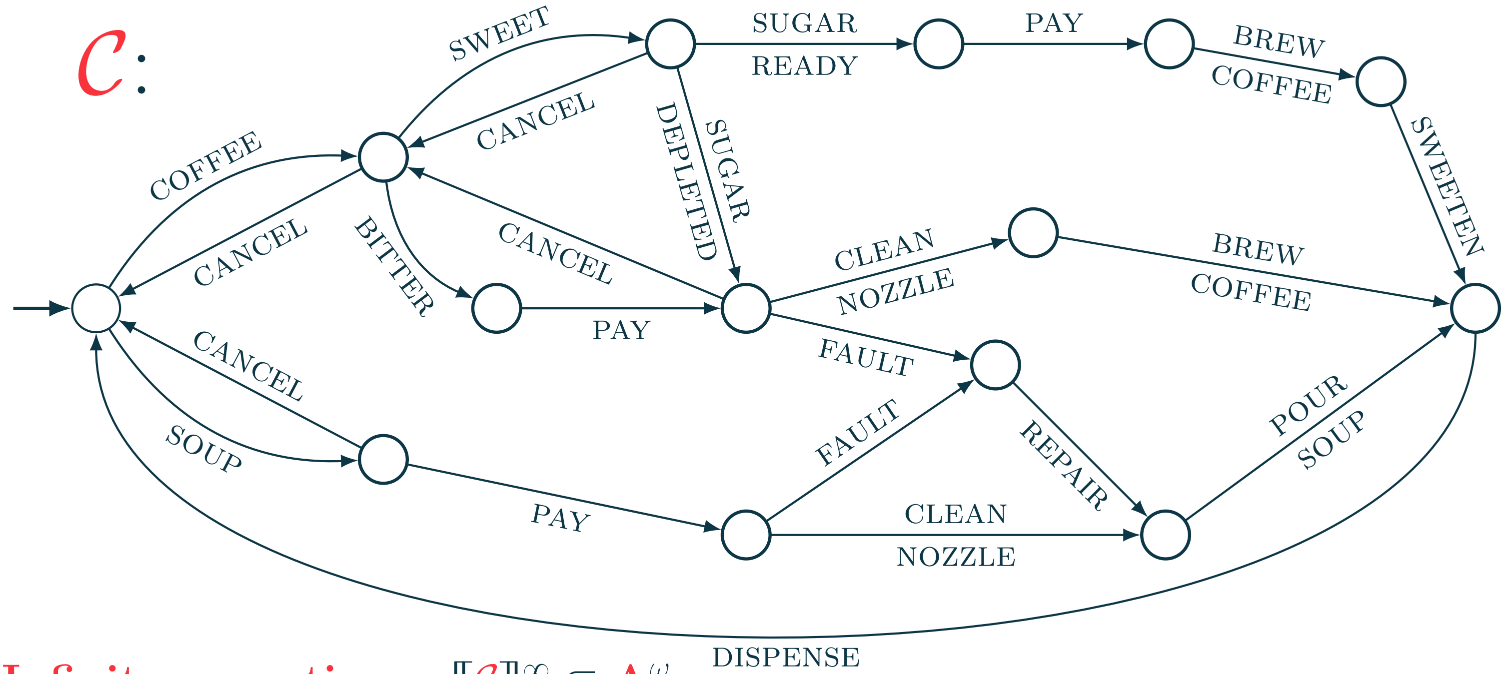
$\mathcal{C}$:



Infinite executions: $[\![\mathcal{C}]\!]^\infty \subseteq \mathbf{A}^\omega$

Specification: „execution without CANCEL, infinitely many times DISPENSE”

$\mathcal{C}$:

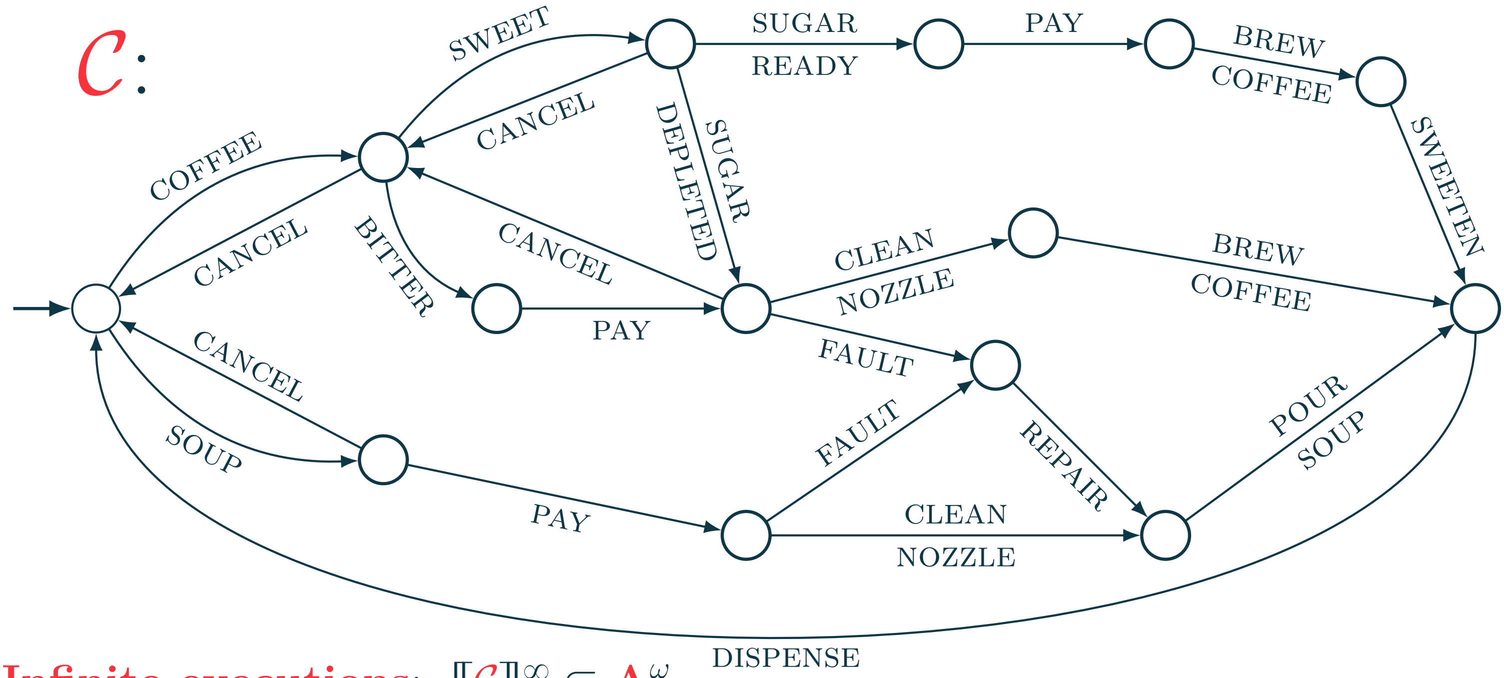


Infinite executions: $[\![\mathcal{C}]\!]^{\infty} \subseteq \mathbf{A}^{\omega}$

Specification: „execution without CANCEL, infinitely many times DISPENSE"

1. Formula of MSO: $\big(\forall t.\ \neg\textsc{cancel}(t)\big) \Rightarrow \forall t.\ \exists t' > t.\ \textsc{dispense}(t')$

$\mathcal{C}$:



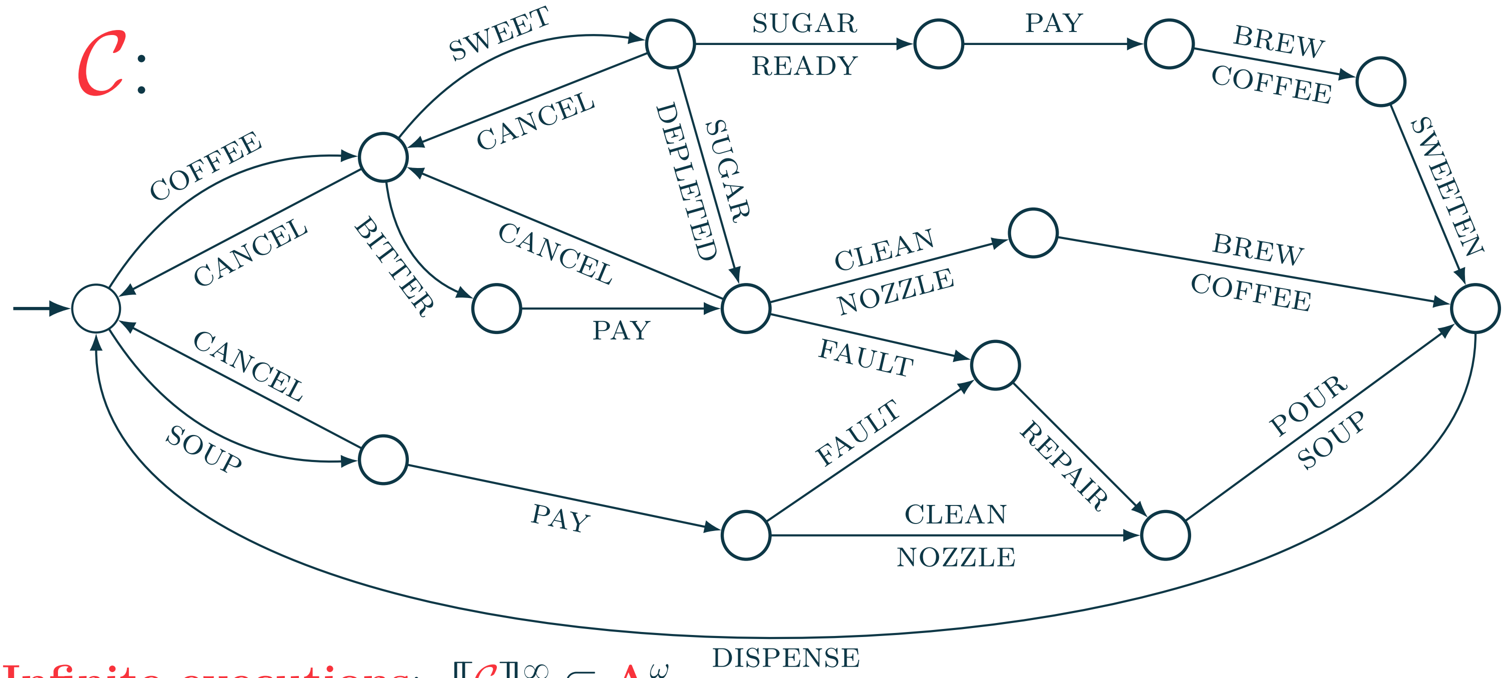
Infinite executions: $[\![\mathcal{C}]\!]^\infty \subseteq \mathbf{A}^\omega$

Specification: „execution without CANCEL, infinitely many times DISPENSE"

1. Formula of MSO: $\big(\forall t.\ \neg\text{CANCEL}(t)\big) \Rightarrow \forall t.\ \exists t' > t.\ \text{DISPENSE}(t')$

2. Regular expression: $\mathbf{A}^* \cdot \text{CANCEL} \cdot \mathbf{A}^\infty + \big(\mathbf{A}^* \cdot \text{DISPENSE}\big)^\infty$

$\mathcal{C}$:



Infinite executions: $[\![\mathcal{C}]\!]^\infty \subseteq \mathbf{A}^\omega$

Specification: „execution without CANCEL, infinitely many times DISPENSE"

1. Formula of MSO: $\big(\forall t.\ \neg\mathrm{CANCEL}(t)\big) \Rightarrow \forall t.\ \exists t' > t.\ \mathrm{DISPENSE}(t')$

2. Regular expression: $\mathbf{A}^* \cdot \mathrm{CANCEL} \cdot \mathbf{A}^\infty + \big(\mathbf{A}^* \cdot \mathrm{DISPENSE}\big)^\infty$

3. Verifier: $\cdots$

Theorem (Büchi [1962])

Theorem (Büchi [1962])

It is possible to effectively translate between the following formalisms for **infinite** executions(!):

1. formulae of **MSO**,
2. regular expressions,
3. ω-automata.

Theorem (Büchi [1962])

It is possible to effectively translate between the following formalisms for **infinite** executions(!):

1. formulae of **MSO**,
2. regular expressions,
3. ω-automata.

Mathematical consequences

Theorem (Büchi [1962])

It is possible to *effectively* translate between the following formalisms for **infinite** executions(!):

1. formulae of **MSO**,
2. regular expressions,
3. ω-automata.

Mathematical consequences

$\rightsquigarrow$ The **MSO** theory of **natural numbers** is **decidable**.

Theorem (Büchi [1962])

It is possible to effectively translate between the following formalisms for **infinite** executions(!):

1. formulae of **MSO**,
2. regular expressions,
3. ω-automata.

Mathematical consequences

⇝ The **MSO** theory of **natural numbers** is **decidable**.

Theorem (Rabin [1969])

Theorem (Büchi [1962])

It is possible to effectively translate between the following formalisms for **infinite** executions(!):

1. formulae of **MSO**,
2. regular expressions,
3. ω-automata.

Mathematical consequences

$\rightsquigarrow$ The **MSO** theory of **natural numbers** is **decidable**.

Theorem (Rabin [1969])

Analogue for **branching** executions!

Theorem (Büchi [1962])

It is possible to effectively translate between the following formalisms for **infinite** executions(!):

1. formulae of **MSO**,
2. regular expressions,
3. ω-automata.

Mathematical consequences

⇝ The **MSO** theory of **natural numbers** is **decidable**.

Theorem (Rabin [1969])

Analogue for **branching** executions!

⇝ The **MSO** theory of **the full binary tree** is **decidable**.

Theorem (Büchi [1962])

It is possible to effectively translate between the following formalisms for **infinite** executions(!):

1. formulae of **MSO**,

2. regular expressions,

3. ω-automata.

Mathematical consequences

$\rightsquigarrow$ The **MSO** theory of **natural numbers** is **decidable**.

Theorem (Rabin [1969])

Analogue for **branching** executions!

$\rightsquigarrow$ The **MSO** theory of **the full binary tree** is **decidable**.

"The mother of all decidability results"

Summary

Summary

1. Computers may be wrong.

Summary

1. Computers may be wrong.

- Hardware errors are **rare** (and one can make them even rarer).

Summary

1. Computers may be wrong.

- Hardware errors are **rare** (and one can make them even rarer).
- Software errors are **common** (and it is hard to avoid them).

Summary

1. Computers may be wrong.

- Hardware errors are **rare** (and one can make them even rarer).
- Software errors are **common** (and it is hard to avoid them).

2. Methods of formal verification guarantee mathematical safety.

Summary

1. Computers may be wrong.

- Hardware errors are **rare** (and one can make them even rarer).
- Software errors are **common** (and it is hard to avoid them).

2. Methods of formal verification guarantee mathematical safety.

- Their application requires effort (and skills $\Rightarrow$ costs).

Summary

1. Computers may be wrong.

- Hardware errors are **rare** (and one can make them even rarer).
- Software errors are **common** (and it is hard to avoid them).

2. Methods of formal verification guarantee mathematical safety.

- Their application requires effort (and skills $\Rightarrow$ costs).
- In general it cannot be automatized.

Summary

1. Computers may be wrong.

- Hardware errors are **rare** (and one can make them even rarer).
- Software errors are **common** (and it is hard to avoid them).

2. Methods of formal verification guarantee mathematical safety.

- Their application requires effort (and skills $\Rightarrow$ costs).
- In general it cannot be automatized.

3. Verification of automata (= finite state machines) is easier.

Summary

1. Computers may be wrong.

- Hardware errors are **rare** (and one can make them even rarer).
- Software errors are **common** (and it is hard to avoid them).

2. Methods of formal verification guarantee mathematical safety.

- Their application requires effort (and skills $\Rightarrow$ costs).
- In general it cannot be automatized.

3. Verification of automata (= finite state machines) is easier.

- Actual applications to simple drivers and devices.

Summary

1. Computers may be wrong.

- Hardware errors are **rare** (and one can make them even rarer).
- Software errors are **common** (and it is hard to avoid them).

2. Methods of formal verification guarantee mathematical safety.

- Their application requires effort (and skills $\Rightarrow$ costs).
- In general it cannot be automatized.

3. Verification of automata (= finite state machines) is easier.

- Actual applications to simple drivers and devices.
- Interesting mathematical consequences.