# Czy komputery są nieomylne?

Michał Skrzypczak
Uniwersytet Warszawski

**XXIII Festiwal Nauki**

24 września 2019, Warszawa


UNIWERSYTET
WARSZAWSKI

# Czy komputery są nieomylne?

Michał Skrzypczak
Uniwersytet Warszawski

**XXIII Festiwal Nauki**

24 września 2019, Warszawa


UNIWERSYTET
WARSZAWSKI

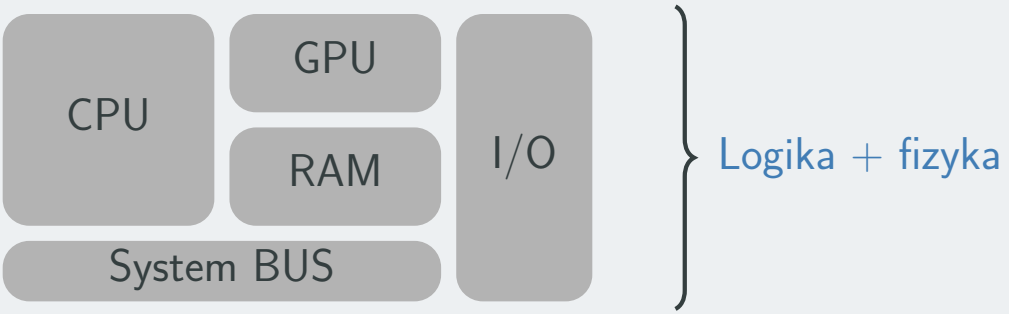
# Komputer ma warstwy...

# Komputer ma warstwy...

**Sprzętowa**:

# Komputer ma warstwy...

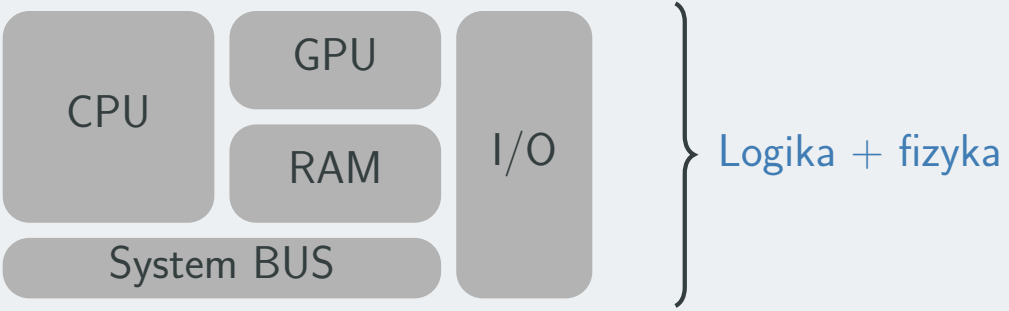
**Sprzętowa**:



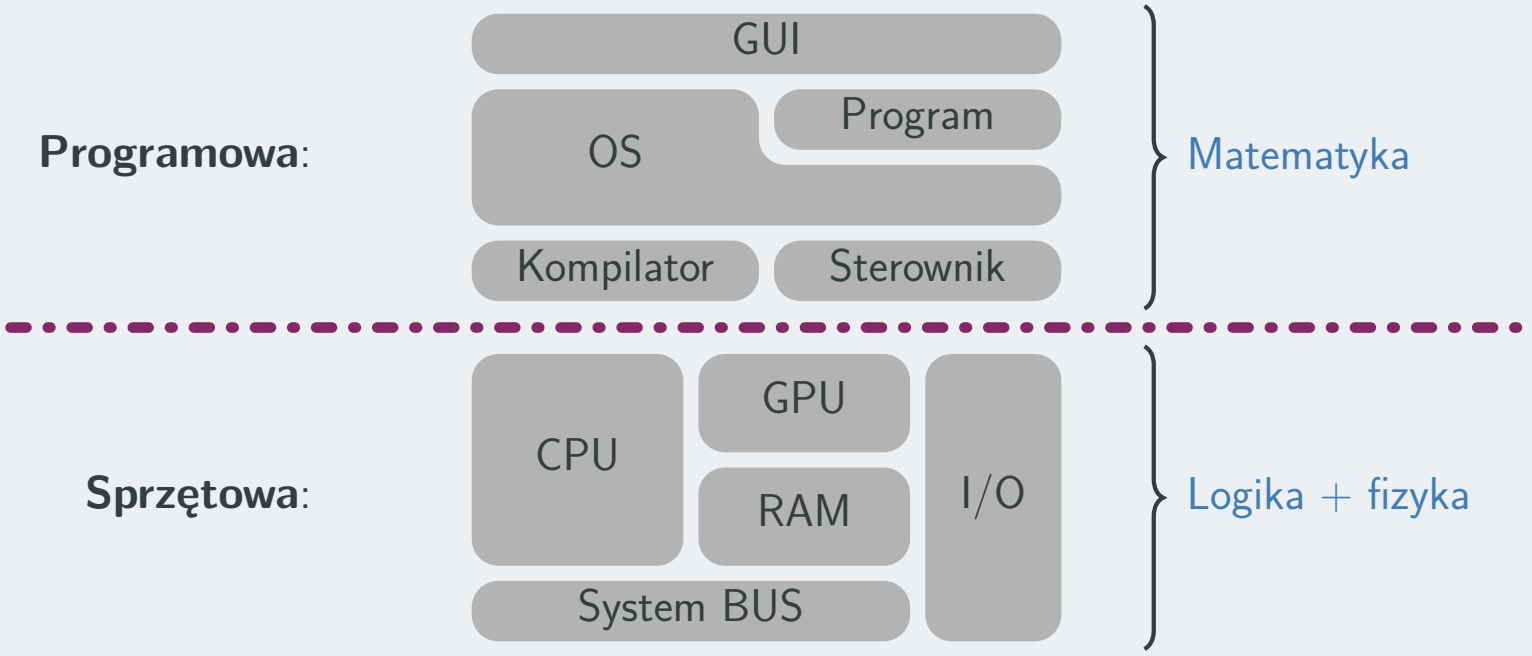
# Komputer ma warstwy...

**Programowa**:

**Sprzętowa**:



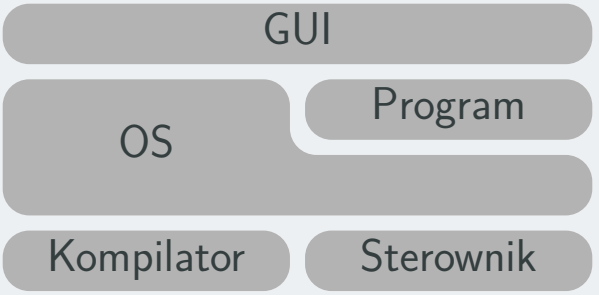
# Komputer ma warstwy...



# Komputer ma warstwy...

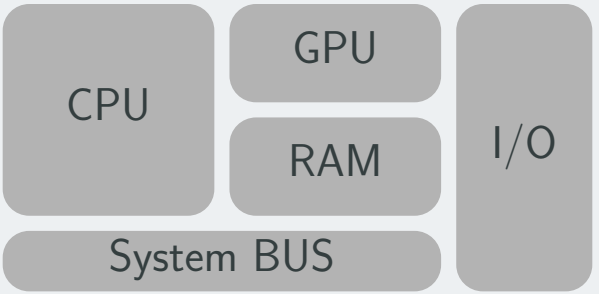
***Organiczna***:

**Programowa**:



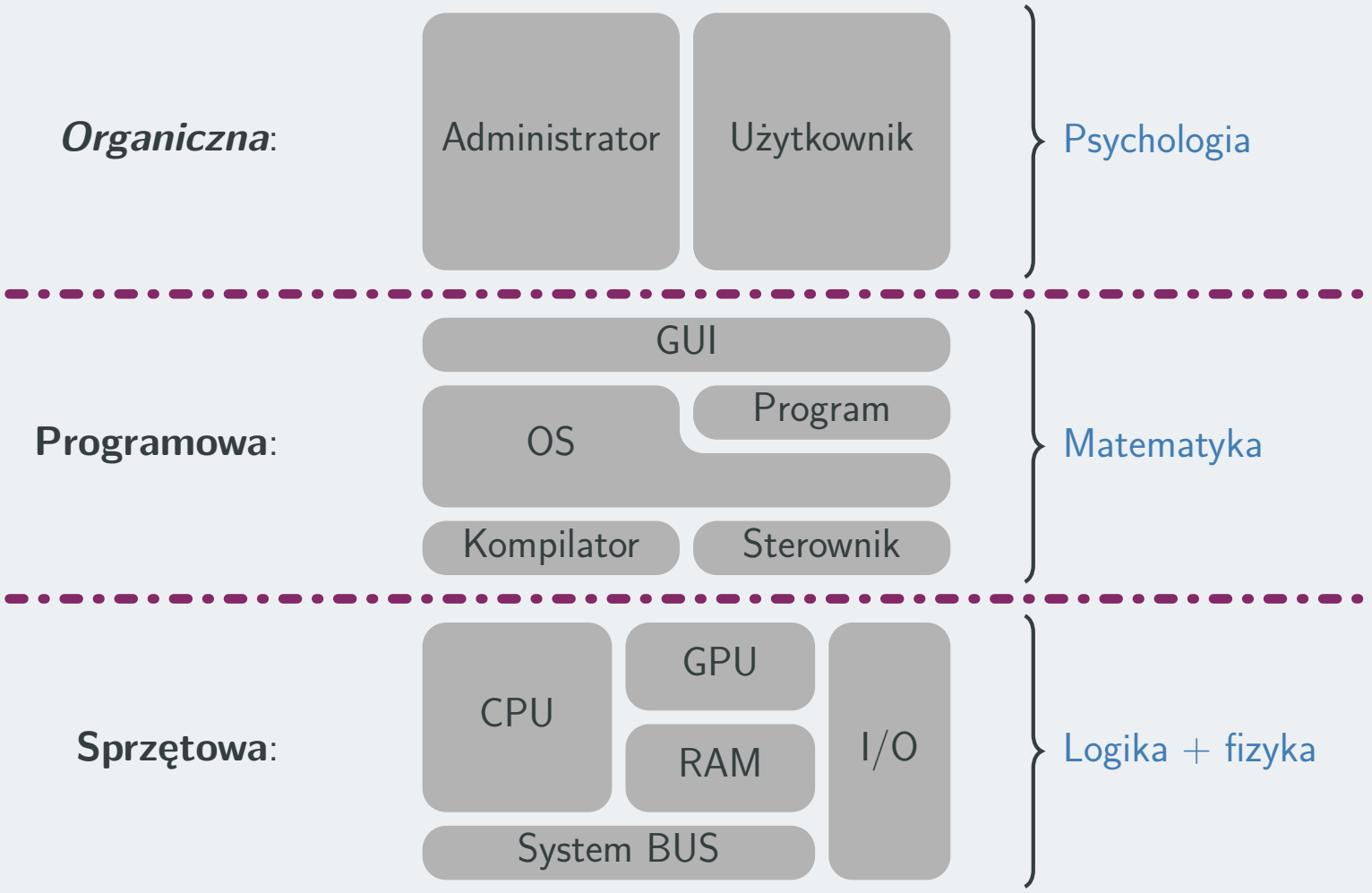

Matematyka

**Sprzętowa**:




Logika + fizyka

# Komputer ma warstwy...

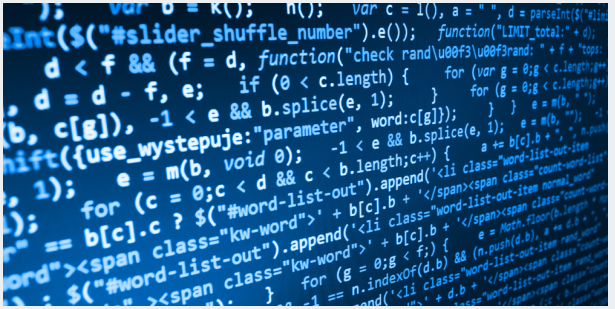


# Komputer ma warstwy...

***Organiczna***:

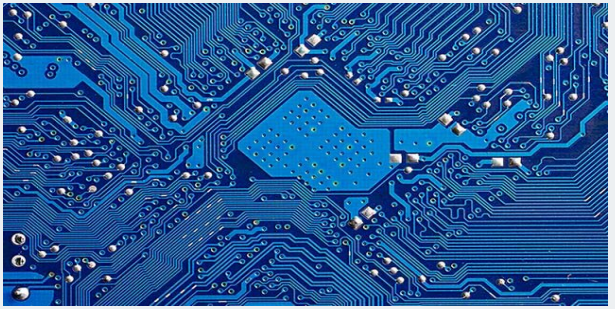
Psychologia

**Programowa**:



Matematyka

**Sprzętowa**:



Logika + fizyka

# Komputer ma warstwy...

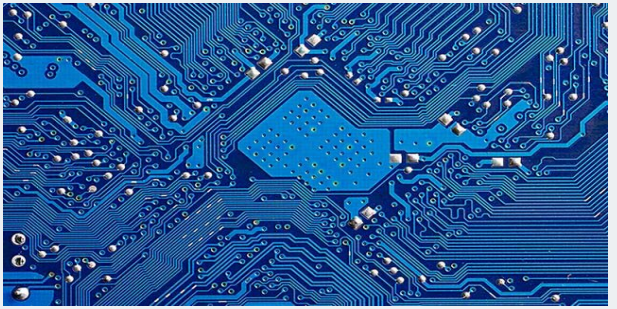
***Organiczna***:

Psychologia

**Programowa**:

Matematyka

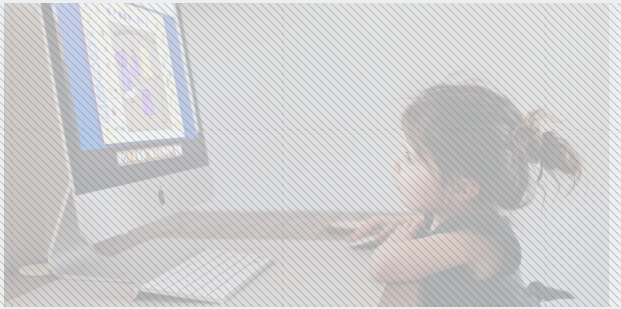
**Sprzętowa**:



Logika + fizyka

# Komputer ma warstwy...

*Organiczna*:

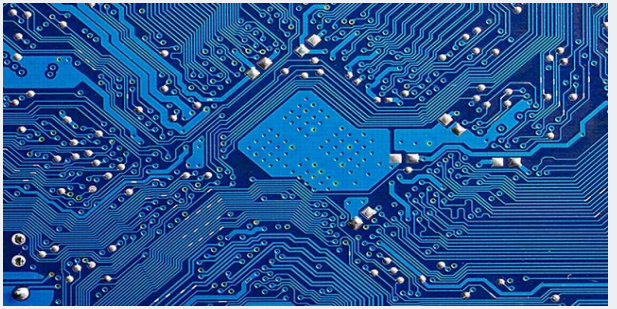


Psychologia

Programowa:

Matematyka

**Sprzętowa**:



Logika + fizyka

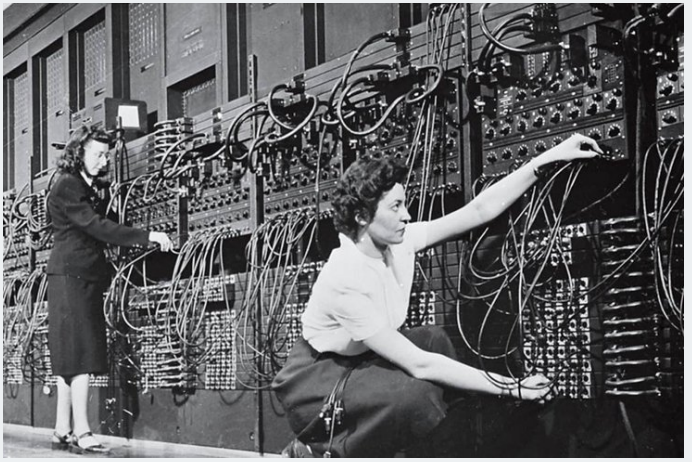
# Błędy sprzętowe

# Błędy sprzętowe

**Kiedyś**:

# Błędy sprzętowe

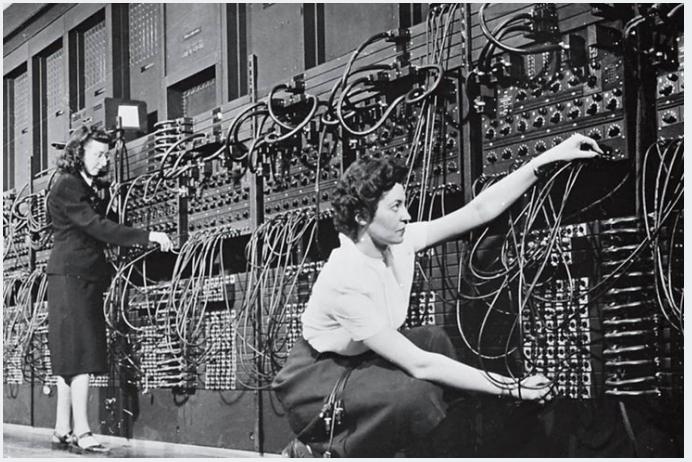
**Kiedyś**: **ENIAC** 1945 – 1955



# Błędy sprzętowe

**Kiedyś**: **ENIAC** 1945 – 1955

- ∼20’000 lamp próżniowych
- ∼5’000’000 ręcznych lutów
- 150kW zasilania

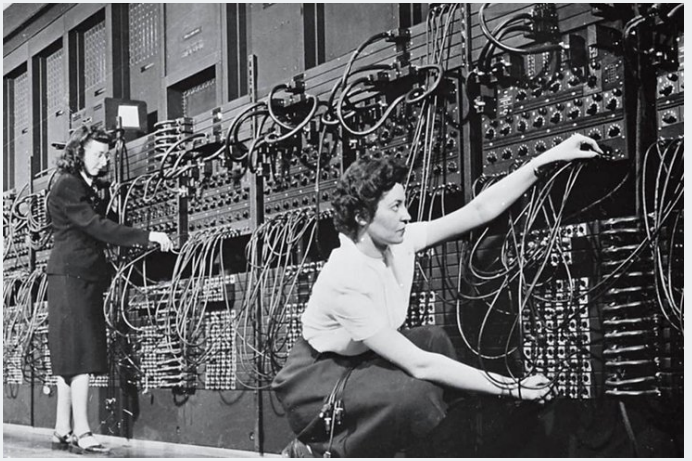


# Błędy sprzętowe

**Kiedyś**: **ENIAC** 1945 – 1955

- $\sim$20'000 lamp próżniowych
- $\sim$5'000'000 ręcznych lutów
- 150kW zasilania
- Średni czas pomiędzy awariami:

  10 min. ('45r.) $\rightarrow$ $>$12 godz. ('55r.)



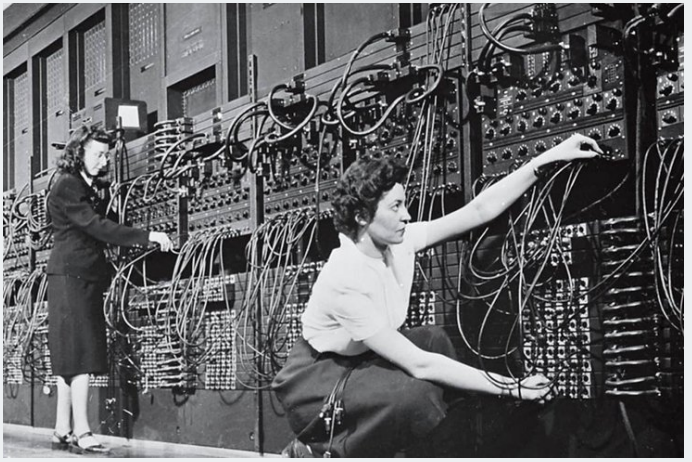
# Błędy sprzętowe

**Kiedyś**: **ENIAC** 1945 – 1955

- ∼20’000 lamp próżniowych
- ∼5’000’000 ręcznych lutów
- 150kW zasilania
- Średni czas pomiędzy awariami:

  10 min. (’45r.) → >12 godz. (’55r.)
- Maks. czas pracy bez awarii:

  116 godzin (’54r.)



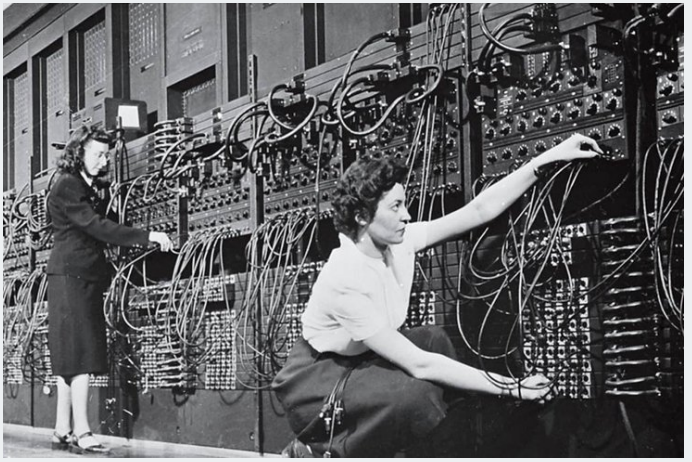
# Błędy sprzętowe

**Kiedyś**: **ENIAC** 1945 – 1955

- ∼20’000 lamp próżniowych
- ∼5’000’000 ręcznych lutów
- 150kW zasilania
- Średni czas pomiędzy awariami:

  10 min. (’45r.) → >12 godz. (’55r.)
- Maks. czas pracy bez awarii:

  116 godzin (’54r.)



**Dziś**:

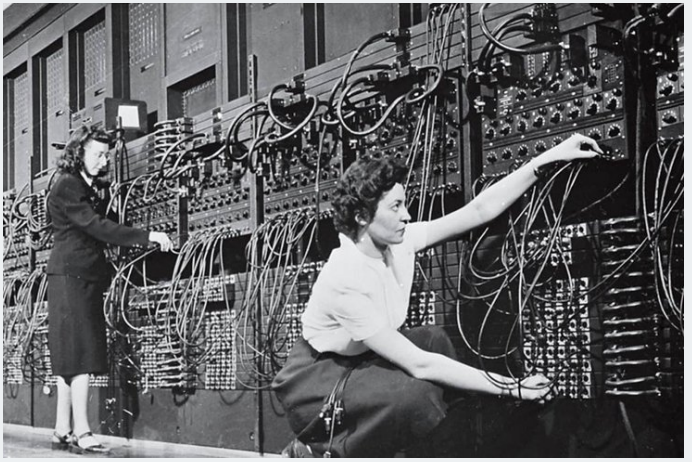
# Błędy sprzętowe

**Kiedyś**: **ENIAC** 1945 – 1955

- ∼20’000 lamp próżniowych
- ∼5’000’000 ręcznych lutów
- 150kW zasilania
- Średni czas pomiędzy awariami:

  10 min. (’45r.) → >12 godz. (’55r.)
- Maks. czas pracy bez awarii:

  116 godzin (’54r.)



**Dziś**: **PC** 2018

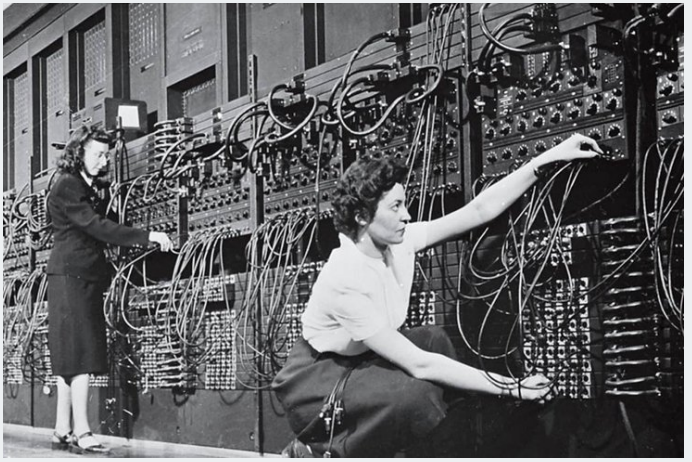
# Błędy sprzętowe

**Kiedyś**: **ENIAC** 1945 – 1955

- ∼20'000 lamp próżniowych
- ∼5'000'000 ręcznych lutów
- 150kW zasilania
- Średni czas pomiędzy awariami:

  10 min. ('45r.) → >12 godz. ('55r.)
- Maks. czas pracy bez awarii:

  116 godzin ('54r.)

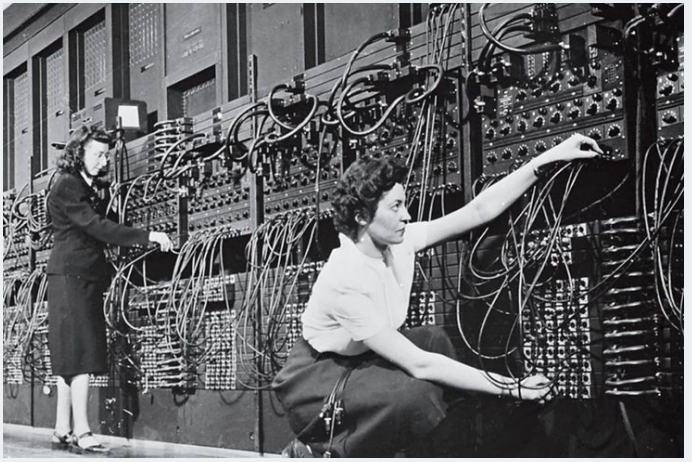


**Dziś**: **PC** 2018

- $\sim 2 \cdot 10^9$ tranzystorów procesora CPU
- $\sim 64 \cdot 10^9$ tranzystorów pamięci RAM
- 1–100W zasilania

# Błędy sprzętowe

**Kiedyś**: **ENIAC** 1945 – 1955



- ∼20'000 lamp próżniowych
- ∼5'000'000 ręcznych lutów
- 150kW zasilania
- Średni czas pomiędzy awariami:

  10 min. ('45r.) → >12 godz. ('55r.)
- Maks. czas pracy bez awarii:

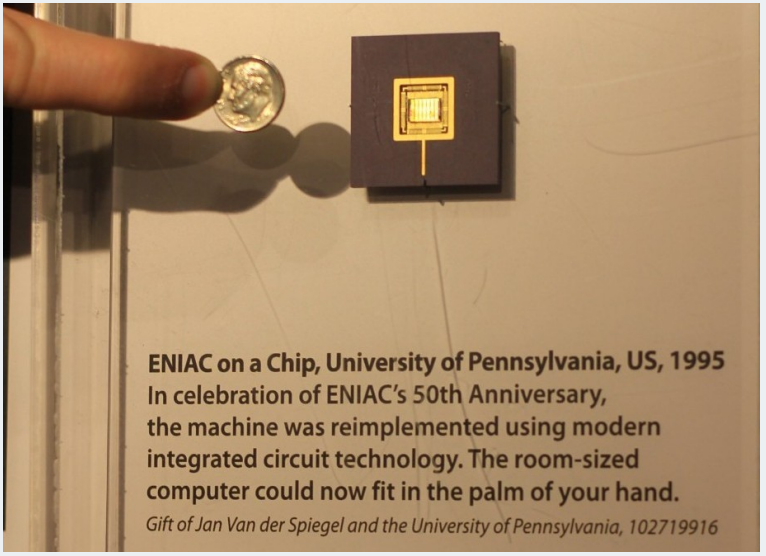
  116 godzin ('54r.)

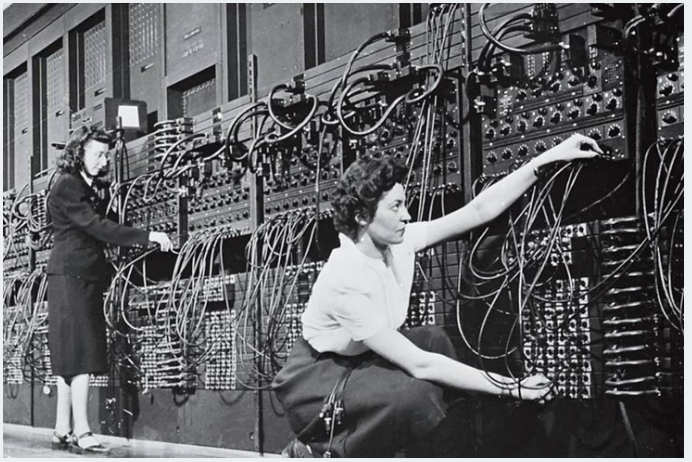
**Dziś**: **PC** 2018

- $\sim 2 \cdot 10^9$ tranzystorów procesora CPU
- $\sim 64 \cdot 10^9$ tranzystorów pamięci RAM
- 1–100W zasilania
- Średni czas pomiędzy awariami: 1100–3285 lat (RAM), 126–220 lat (CPU)

# Błędy sprzętowe

**Kiedyś**: **ENIAC** 1945 – 1955



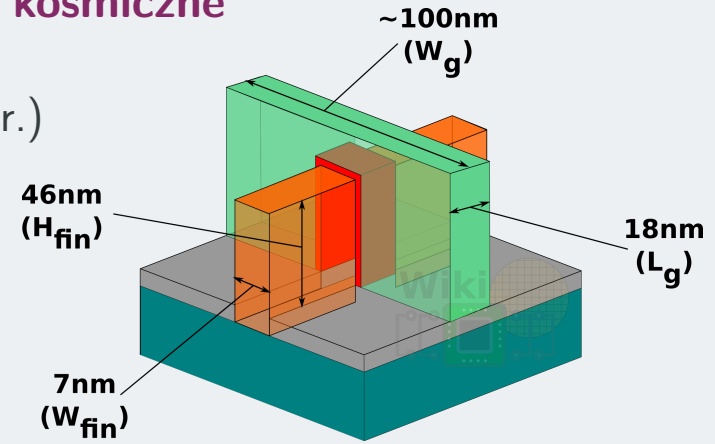
**Dziś**: **PC** 2018

- $\sim 2 \cdot 10^9$ tranzystorów procesora CPU
- $\sim 64 \cdot 10^9$ tranzystorów pamięci RAM
- 1–100W zasilania
- Średni czas pomiędzy awariami: 1100–3285 lat (RAM), 126–220 lat (CPU)

# Promieniowanie kosmiczne

# Promieniowanie kosmiczne

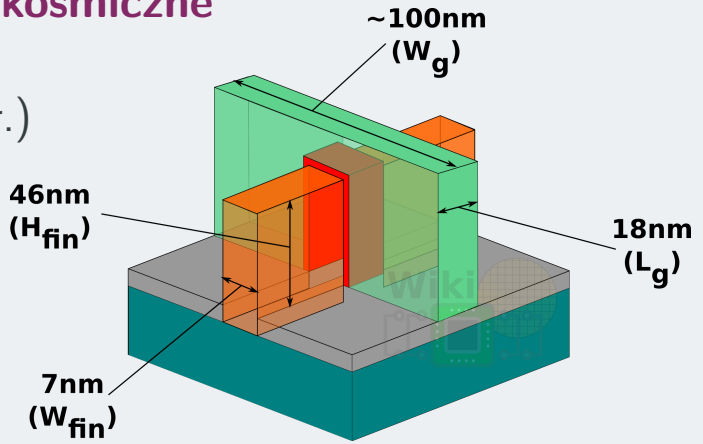
Układy scalone w technologii “10nm” (2018 r.)



# Promieniowanie kosmiczne

Układy scalone w technologii “10nm” (2018 r.)

Ścieżki szerokości kilkudziesięciu cząsteczek!

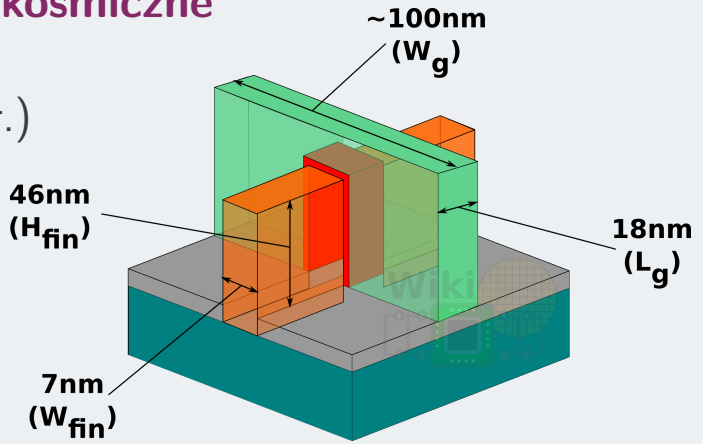


# Promieniowanie kosmiczne

Układy scalone w technologii “10nm” (2018 r.)

Ścieżki szerokości kilkudziesięciu cząsteczek!

⇝ ryzyko tzw. *Single Event Upset*



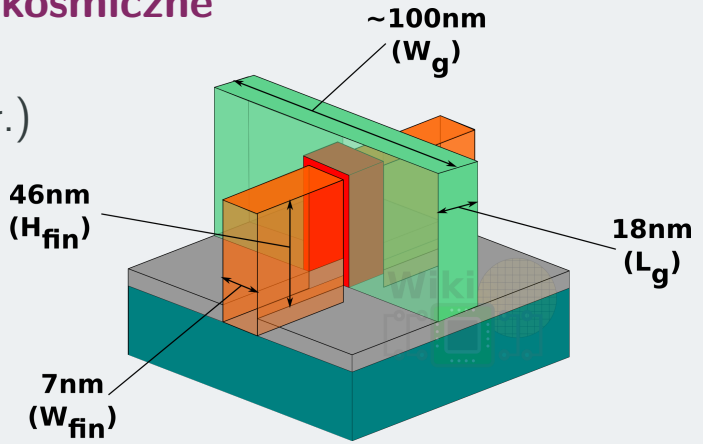
# Promieniowanie kosmiczne

Układy scalone w technologii “10nm” (2018 r.)

Ścieżki szerokości kilkudziesięciu cząsteczek!

⇝ ryzyko tzw. *Single Event Upset*

**Potwierdzone przykłady**:

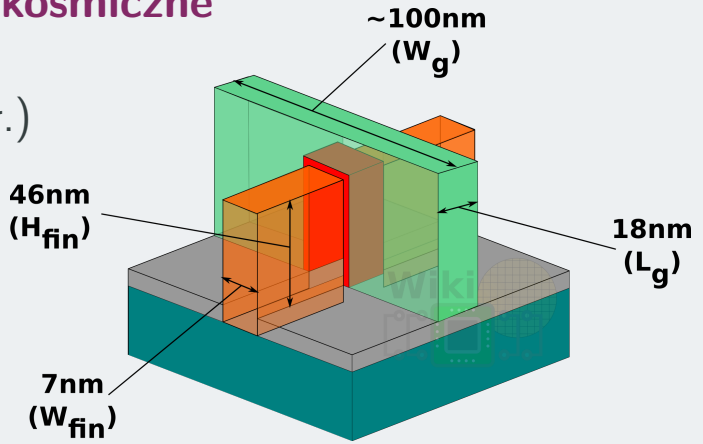


# Promieniowanie kosmiczne

Układy scalone w technologii “10nm” (2018 r.)

Ścieżki szerokości kilkudziesięciu cząsteczek!

⇝ ryzyko tzw. *Single Event Upset*




**Potwierdzone przykłady**:

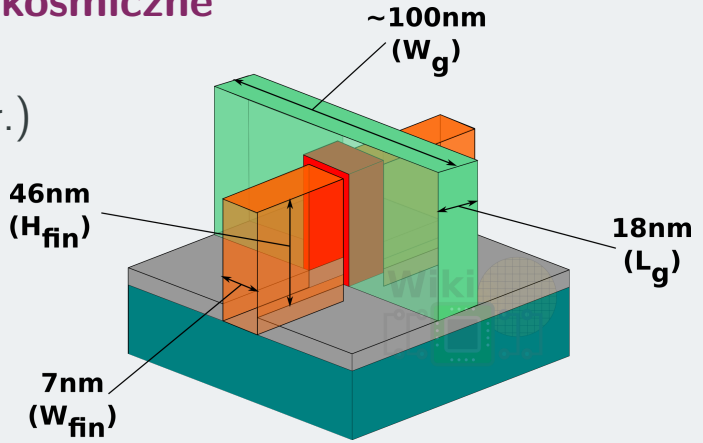
- W 1972 r. satelita komunikacyjny Hughes zawiesił komunikację na 96 sekund.

# Promieniowanie kosmiczne

Układy scalone w technologii “10nm” (2018 r.)

Ścieżki szerokości kilkudziesięciu cząsteczek!

⇝ ryzyko tzw. *Single Event Upset*




**Potwierdzone przykłady**:

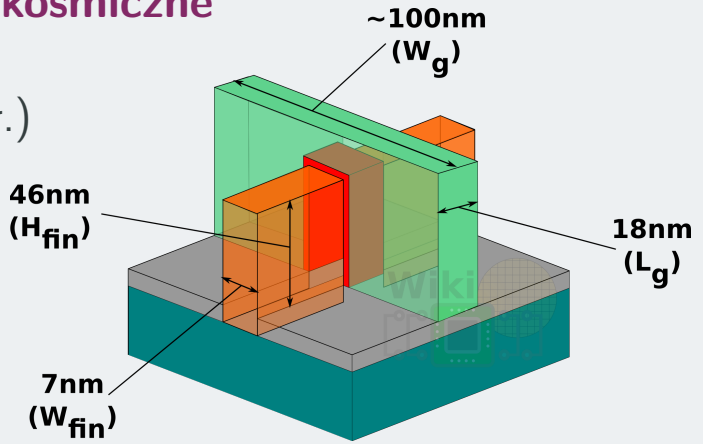
- W 1972 r. satelita komunikacyjny Hughes zawiesił komunikację na 96 sekund.
  ⇝ pierwszy potwierdzony przypadek SEU

# Promieniowanie kosmiczne

Układy scalone w technologii “10nm” (2018 r.)

Ścieżki szerokości kilkudziesięciu cząsteczek!

⇝ ryzyko tzw. *Single Event Upset*




**Potwierdzone przykłady**:

- W 1972 r. satelita komunikacyjny Hughes zawiesił komunikację na 96 sekund.

  ⇝ pierwszy potwierdzony przypadek SEU

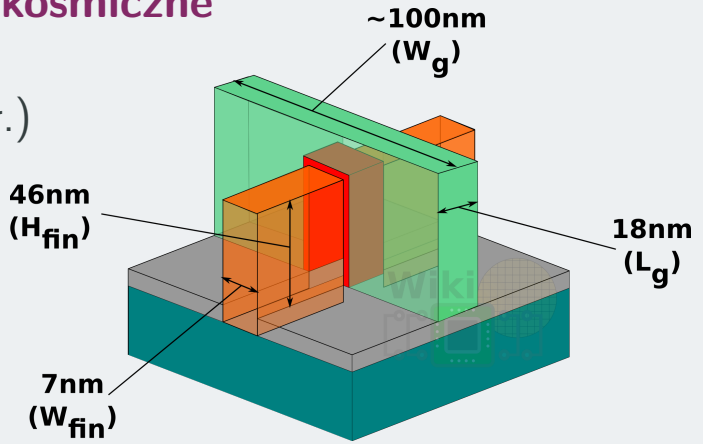
- W wyborach w 2003 r. w Schaerbeek (Belgia) zliczono o 4096 głosów za dużo.

# Promieniowanie kosmiczne

Układy scalone w technologii “10nm” (2018 r.)

Ścieżki szerokości kilkudziesięciu cząsteczek!

⇝ ryzyko tzw. *Single Event Upset*




**Potwierdzone przykłady**:

- W 1972 r. satelita komunikacyjny Hughes zawiesił komunikację na 96 sekund.

  ⇝ pierwszy potwierdzony przypadek SEU

- W wyborach w 2003 r. w Schaerbeek (Belgia) zliczono o 4096 głosów za dużo.

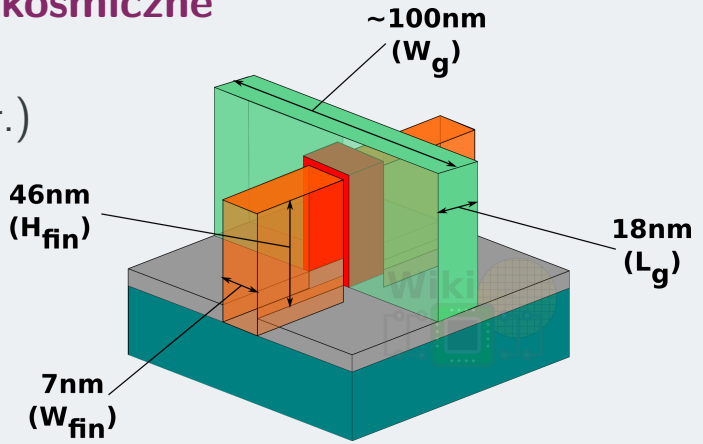
  ⇝ $4096 = ???$

# Promieniowanie kosmiczne

Układy scalone w technologii “10nm” (2018 r.)

Ścieżki szerokości kilkudziesięciu cząsteczek!

⇝ ryzyko tzw. *Single Event Upset*




**Potwierdzone przykłady**:

- W 1972 r. satelita komunikacyjny Hughes zawiesił komunikację na 96 sekund.

  ⇝ pierwszy potwierdzony przypadek SEU

- W wyborach w 2003 r. w Schaerbeek (Belgia) zliczono o 4096 głosów za dużo.

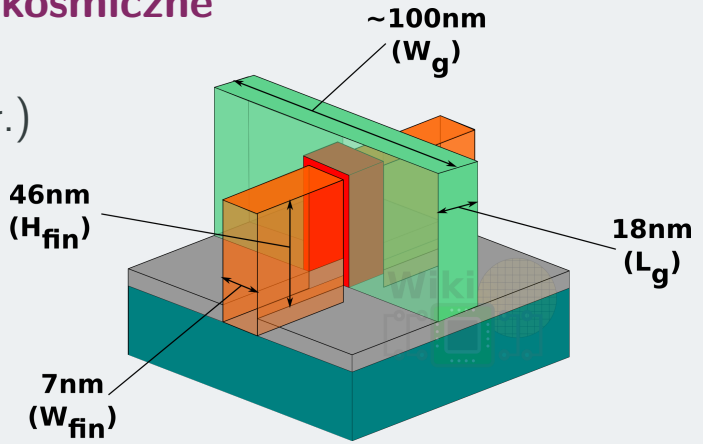
  ⇝ $4096 = 2^{12}$

# Promieniowanie kosmiczne

Układy scalone w technologii “10nm” (2018 r.)

Ścieżki szerokości kilkudziesięciu cząsteczek!

⇝ ryzyko tzw. *Single Event Upset*




**Potwierdzone przykłady**:

- W 1972 r. satelita komunikacyjny Hughes zawiesił komunikację na 96 sekund.

  ⇝ pierwszy potwierdzony przypadek SEU

- W wyborach w 2003 r. w Schaerbeek (Belgia) zliczono o 4096 głosów za dużo.

  ⇝ $4096 = 2^{12}$

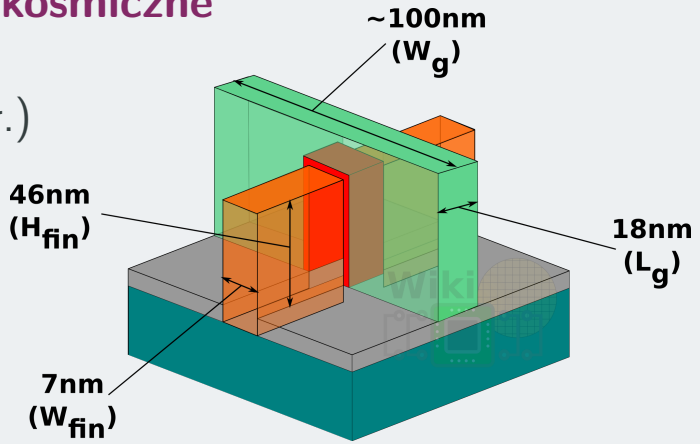
**Metody zabezpieczeń**:

# Promieniowanie kosmiczne

Układy scalone w technologii “10nm” (2018 r.)

Ścieżki szerokości kilkudziesięciu cząsteczek!

⇝ ryzyko tzw. *Single Event Upset*




**Potwierdzone przykłady**:

- W 1972 r. satelita komunikacyjny Hughes zawiesił komunikację na 96 sekund.
  ⇝ pierwszy potwierdzony przypadek SEU
- W wyborach w 2003 r. w Schaerbeek (Belgia) zliczono o 4096 głosów za dużo.
  ⇝ $4096 = 2^{12}$

**Metody zabezpieczeń**:

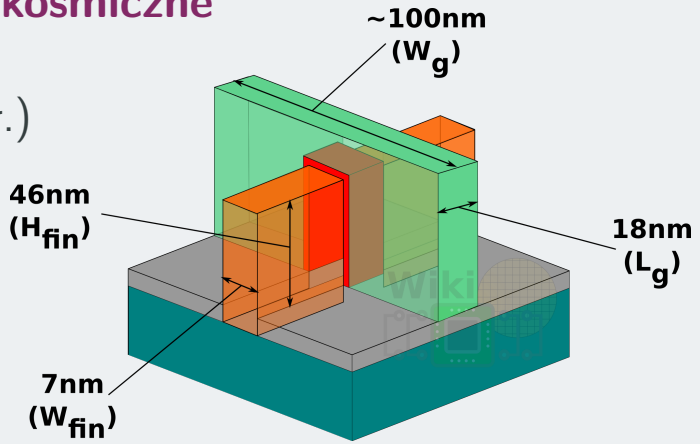
- Komputery dowodzenia ISS bazują na i386 (proces $1\mu$m$=$ 100$x$10nm).

# Promieniowanie kosmiczne

Układy scalone w technologii “10nm” (2018 r.)

Ścieżki szerokości kilkudziesięciu cząsteczek!

⇝ ryzyko tzw. *Single Event Upset*




**Potwierdzone przykłady**:

- W 1972 r. satelita komunikacyjny Hughes zawiesił komunikację na 96 sekund.
  ⇝ pierwszy potwierdzony przypadek SEU
- W wyborach w 2003 r. w Schaerbeek (Belgia) zliczono o 4096 głosów za dużo.
  ⇝ $4096 = 2^{12}$

**Metody zabezpieczeń**:

- Komputery dowodzenia ISS bazują na i386 (proces $1\mu$m$=$ 100$x$10nm).
- Potrojone systemy komputerowe w samolotach *fly-by-wire*.

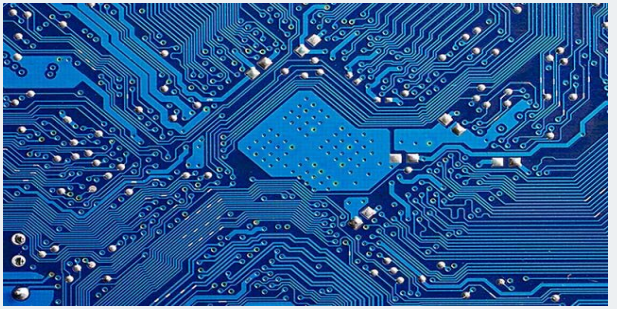
***Organiczna***:

Psychologia

**Programowa**:

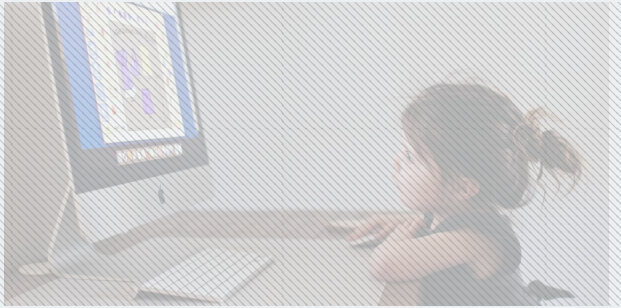
Matematyka

**Sprzętowa**:



Logika + fizyka

*Organiczna*:

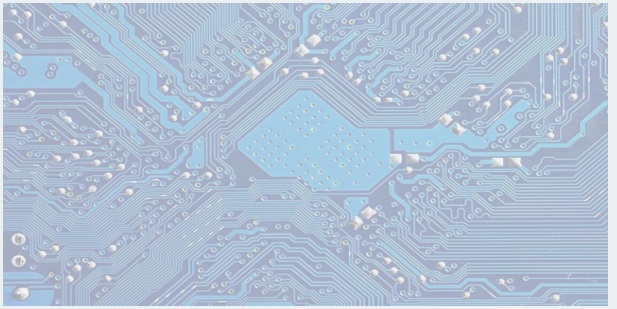


Psychologia

**Programowa**:

Matematyka

Sprzętowa:



Logika + fizyka

# Poprawne programy?

## Poprawne programy?

- 1986 r., sterowana komputerowo radioterapia Therac-25

## Poprawne programy?

- 1986 r., sterowana komputerowo radioterapia Therac-25

  błąd kontroli współbieżności (tzw. *race condition*)

## Poprawne programy?

- 1986 r., sterowana komputerowo radioterapia Therac-25

  błąd kontroli współbieżności (tzw. *race condition*)

  ⇝ 6cioro pacjentów poważnie napromieniowanych, dwie ofiar śmiertelnych

## Poprawne programy?

- 1986 r., sterowana komputerowo radioterapia Therac-25

  błąd kontroli współbieżności (tzw. *race condition*)

  ⇝ 6cioro pacjentów poważnie napromieniowanych, dwie ofiar śmiertelnych

- 1996 r., rakieta Ariane 5 (ESA), oprogramowanie częściowo z Ariane 4

## Poprawne programy?

- 1986 r., sterowana komputerowo radioterapia Therac-25

  błąd kontroli współbieżności (tzw. *race condition*)

  ⇝ 6cioro pacjentów poważnie napromieniowanych, dwie ofiar śmiertelnych

- 1996 r., rakieta Ariane 5 (ESA), oprogramowanie częściowo z Ariane 4

  błąd przekroczenia zakresu (silniki Ariane 5 dają ∼3x więcej ciągu od Ariane 4)

## Poprawne programy?

- 1986 r., sterowana komputerowo radioterapia Therac-25

  błąd kontroli współbieżności (tzw. *race condition*)

  ⇝ 6cioro pacjentów poważnie napromieniowanych, dwie ofiar śmiertelnych

- 1996 r., rakieta Ariane 5 (ESA), oprogramowanie częściowo z Ariane 4

  błąd przekroczenia zakresu (silniki Ariane 5 dają ∼3x więcej ciągu od Ariane 4)

  ⇝ wybuch w 30 sekundzie lotu, straty szacowane na 442 M€


## Poprawne programy?

- 1986 r., sterowana komputerowo radioterapia Therac-25

  błąd kontroli współbieżności (tzw. *race condition*)

  ⇝ 6cioro pacjentów poważnie napromieniowanych, dwie ofiar śmiertelnych

- 1996 r., rakieta Ariane 5 (ESA), oprogramowanie częściowo z Ariane 4

  błąd przekroczenia zakresu (silniki Ariane 5 dają ∼3x więcej ciągu od Ariane 4)

  ⇝ wybuch w 30 sekundzie lotu, straty szacowane na 442 M€

- 1993 r., procesor Intel Pentium, błąd instrukcji FDIV (475 M$)

## Poprawne programy?

- 1986 r., sterowana komputerowo radioterapia Therac-25

  błąd kontroli współbieżności (tzw. *race condition*)

  ⇝ 6cioro pacjentów poważnie napromieniowanych, dwie ofiar śmiertelnych

- 1996 r., rakieta Ariane 5 (ESA), oprogramowanie częściowo z Ariane 4

  błąd przekroczenia zakresu (silniki Ariane 5 dają ∼3x więcej ciągu od Ariane 4)

  ⇝ wybuch w 30 sekundzie lotu, straty szacowane na 442 M€

- 1993 r., procesor Intel Pentium, błąd instrukcji FDIV (475 M$)

- 1999 r., Mars Polar Lander (100 M$)

## Poprawne programy?

- 1986 r., sterowana komputerowo radioterapia Therac-25

  błąd kontroli współbieżności (tzw. *race condition*)

  ⇝ 6cioro pacjentów poważnie napromieniowanych, dwie ofiar śmiertelnych

- 1996 r., rakieta Ariane 5 (ESA), oprogramowanie częściowo z Ariane 4

  błąd przekroczenia zakresu (silniki Ariane 5 dają ∼3x więcej ciągu od Ariane 4)

  ⇝ wybuch w 30 sekundzie lotu, straty szacowane na 442 M€

- 1993 r., procesor Intel Pentium, błąd instrukcji FDIV (475 M$)

- 1999 r., Mars Polar Lander (100 M$)

- 2019 r., liczne błędy w projekcie Boeinga 737 MAX

## Poprawne programy?

- 1986 r., sterowana komputerowo radioterapia Therac-25

  błąd kontroli współbieżności (tzw. *race condition*)

  ⇝ 6cioro pacjentów poważnie napromieniowanych, dwie ofiar śmiertelnych

- 1996 r., rakieta Ariane 5 (ESA), oprogramowanie częściowo z Ariane 4

  błąd przekroczenia zakresu (silniki Ariane 5 dają $\sim$3x więcej ciągu od Ariane 4)

  ⇝ wybuch w 30 sekundzie lotu, straty szacowane na 442 M€

- 1993 r., procesor Intel Pentium, błąd instrukcji FDIV (475 M$)

- 1999 r., Mars Polar Lander (100 M$)

- 2019 r., liczne błędy w projekcie Boeinga 737 MAX

  m.in. zmiana zakresu wychylenia sterów wysokości przez MCAS z $0.6^\circ$ do $2.5^\circ$

## Poprawne programy?

- 1986 r., sterowana komputerowo radioterapia Therac-25

  błąd kontroli współbieżności (tzw. *race condition*)

  ⇝ 6cioro pacjentów poważnie napromieniowanych, dwie ofiar śmiertelnych

- 1996 r., rakieta Ariane 5 (ESA), oprogramowanie częściowo z Ariane 4

  błąd przekroczenia zakresu (silniki Ariane 5 dają $\sim$3x więcej ciągu od Ariane 4)

  ⇝ wybuch w 30 sekundzie lotu, straty szacowane na 442 M€

- 1993 r., procesor Intel Pentium, błąd instrukcji FDIV (475 M$)

- 1999 r., Mars Polar Lander (100 M$)

- 2019 r., liczne błędy w projekcie Boeinga 737 MAX

  m.in. zmiana zakresu wychylenia sterów wysokości przez MCAS z 0.6° do 2.5°

  ⇝ dwie katastrofy samolotów pasażerskich, 346 ofiar śmiertelnych

## Poprawne programy?

- 1986 r., sterowana komputerowo radioterapia Therac-25
  błąd kontroli współbieżności (tzw. *race condition*)
  ⇝ 6cioro pacjentów poważnie napromieniowanych, dwie ofiar śmiertelnych

- 1996 r., rakieta Ariane 5 (ESA), oprogramowanie częściowo z Ariane 4
  błąd przekroczenia zakresu (silniki Ariane 5 dają $\sim$3x więcej ciągu od Ariane 4)
  ⇝ wybuch w 30 sekundzie lotu, straty szacowane na 442 M€

- 1993 r., procesor Intel Pentium, błąd instrukcji FDIV (475 M$)

- 1999 r., Mars Polar Lander (100 M$)

- 2019 r., liczne błędy w projekcie Boeinga 737 MAX
  m.in. zmiana zakresu wychylenia sterów wysokości przez MCAS z $0.6^\circ$ do $2.5^\circ$
  ⇝ dwie katastrofy samolotów pasażerskich, 346 ofiar śmiertelnych

- ...

# Poprawne programy!

# Poprawne programy!

$\mathcal{P}$

program

# Poprawne programy!

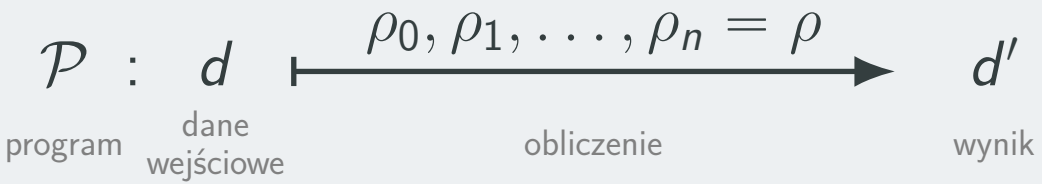
$$\underset{\text{program}}{\mathcal{P}} : \underset{\substack{\text{dane} \\ \text{wejściowe}}}{d}$$

## Poprawne programy!

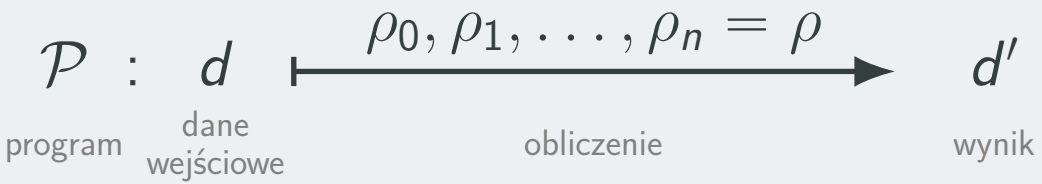
$$\mathcal{P} \; : \; d \;\longmapsto\; \rho_0, \rho_1, \ldots, \rho_n = \rho \longrightarrow$$

program — dane wejściowe — obliczenie

## Poprawne programy!

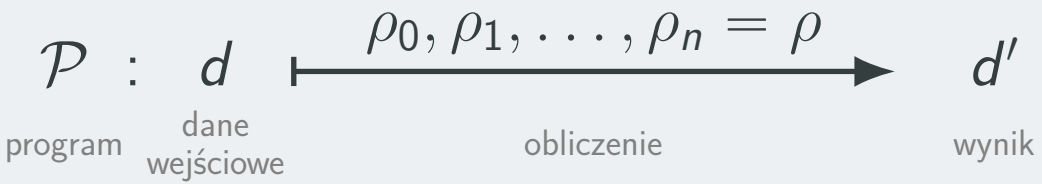


## Poprawne programy!




**Fakt 1**: $\mathcal{P}$, $d$, $\rho$ i $d'$ to ostatecznie **ciągi bitów** $\rightsquigarrow$ liczby naturalne!

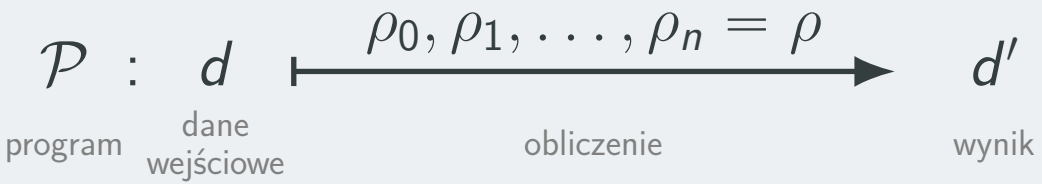
## Poprawne programy!




**Fakt 1**: $\mathcal{P}$, $d$, $\rho$ i $d'$ to ostatecznie **ciągi bitów** $\rightsquigarrow$ liczby naturalne!

**Fakt 2**: Istnieje **formuła matematyczna** opisująca powyższą zależność.

## Poprawne programy!

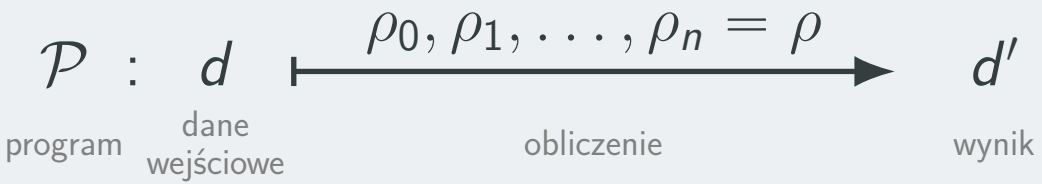



**Fakt 1**: $\mathcal{P}$, $d$, $\rho$ i $d'$ to ostatecznie **ciągi bitów** $\rightsquigarrow$ liczby naturalne!

**Fakt 2**: Istnieje **formuła matematyczna** opisująca powyższą zależność.

**Czyli**: Następujące zdanie jest własnością matematyczną:

## Poprawne programy!



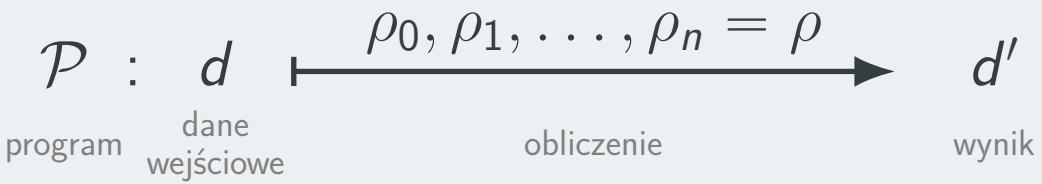

**Fakt 1**: $\mathcal{P}$, $d$, $\rho$ i $d'$ to ostatecznie **ciągi bitów** $\rightsquigarrow$ liczby naturalne!

**Fakt 2**: Istnieje **formuła matematyczna** opisująca powyższą zależność.

**Czyli**: Następujące zdanie jest własnością matematyczną:

$\Big($ Ilekroć dane $d$ spełniają założenia $\varphi$ to wynik $d'$ spełnia wymagania $\psi$. $\Big)$

## Poprawne programy!




**Fakt 1**: $\mathcal{P}$, $d$, $\rho$ i $d'$ to ostatecznie **ciągi bitów** $\rightsquigarrow$ liczby naturalne!

**Fakt 2**: Istnieje **formuła matematyczna** opisująca powyższą zależność.

**Czyli**: Następujące zdanie jest własnością matematyczną:

$$\left(\begin{array}{c} \text{Ilekroć dane } d \text{ spełniają założenia } \varphi \text{ to wynik } d' \text{ spełnia wymagania } \psi. \\ \text{w skrócie: } [\varphi]\mathcal{P}[\psi] \end{array}\right)$$

## Poprawne programy!

$$\underset{\text{program}}{\mathcal{P}} : \underset{\substack{\text{dane}\\ \text{wejściowe}}}{d} \xmapsto[\text{obliczenie}]{\rho_0, \rho_1, \ldots, \rho_n = \rho} \underset{\text{wynik}}{d'}$$

**Fakt 1**: $\mathcal{P}$, $d$, $\rho$ i $d'$ to ostatecznie **ciągi bitów** $\rightsquigarrow$ liczby naturalne!

**Fakt 2**: Istnieje **formuła matematyczna** opisująca powyższą zależność.

**Czyli**: Następujące zdanie jest własnością matematyczną:

$$\left(\begin{array}{c}\text{Ilekroć dane } d \text{ spełniają założenia } \varphi \text{ to wynik } d' \text{ spełnia wymagania } \psi.\\ \text{w skrócie: } [\varphi]\mathcal{P}[\psi]\end{array}\right)$$

**Na przykład**: $[d \geqslant 0]\mathcal{P}_{\text{pierwiastek}}[d' \cdot d' \leqslant d < (d'+1)^2]$

## Poprawne programy!

$$\underbrace{\mathcal{P}}_{\text{program}} : \underbrace{d}_{\substack{\text{dane}\\ \text{wejściowe}}} \xrightarrow[\text{obliczenie}]{\rho_0, \rho_1, \ldots, \rho_n = \rho} \underbrace{d'}_{\text{wynik}}$$

**Fakt 1**: $\mathcal{P}$, $d$, $\rho$ i $d'$ to ostatecznie **ciągi bitów** $\rightsquigarrow$ liczby naturalne!

**Fakt 2**: Istnieje **formuła matematyczna** opisująca powyższą zależność.

**Czyli**: Następujące zdanie jest własnością matematyczną:

$$\left(\begin{array}{c}\text{Ilekroć dane } d \text{ spełniają założenia } \varphi \text{ to wynik } d' \text{ spełnia wymagania } \psi.\\ \text{w skrócie: } [\varphi]\mathcal{P}[\psi]\end{array}\right)$$

**Na przykład**: $[d \geqslant 0]\mathcal{P}_{\text{pierwiastek}}[d' \cdot d' \leqslant d < (d'+1)^2]$

**Więc**: Tego typu własności można **dowodzić!**

## Poprawne programy!

$$\underbrace{\mathcal{P}}_{\text{program}} : \underbrace{d}_{\substack{\text{dane}\\ \text{wejściowe}}} \xrightarrow[\text{obliczenie}]{\rho_0, \rho_1, \ldots, \rho_n = \rho} \underbrace{d'}_{\text{wynik}}$$

**Fakt 1**: $\mathcal{P}$, $d$, $\rho$ i $d'$ to ostatecznie **ciągi bitów** $\rightsquigarrow$ liczby naturalne!

**Fakt 2**: Istnieje **formuła matematyczna** opisująca powyższą zależność.

**Czyli**: Następujące zdanie jest własnością matematyczną:

$$\left(\begin{array}{c}\text{Ilekroć dane } d \text{ spełniają założenia } \varphi \text{ to wynik } d' \text{ spełnia wymagania } \psi.\\ \text{w skrócie: } [\varphi]\mathcal{P}[\psi]\end{array}\right)$$

**Na przykład**: $[d \geqslant 0]\mathcal{P}_{\text{pierwiastek}}[d' \cdot d' \leqslant d < (d'+1)^2]$

**Więc**: Tego typu własności można **dowodzić!** (Hoare [1969])

## Poprawne programy!

$$\underbrace{\mathcal{P}}_{\text{program}} : \underbrace{d}_{\substack{\text{dane}\\ \text{wejściowe}}} \xrightarrow[\text{obliczenie}]{\rho_0, \rho_1, \ldots, \rho_n = \rho} \underbrace{d'}_{\text{wynik}}$$

**Fakt 1**: $\mathcal{P}$, $d$, $\rho$ i $d'$ to ostatecznie **ciągi bitów** $\rightsquigarrow$ liczby naturalne!

**Fakt 2**: Istnieje **formuła matematyczna** opisująca powyższą zależność.

**Czyli**: Następujące zdanie jest własnością matematyczną:

$$\left(\begin{array}{c}\text{Ilekroć dane } d \text{ spełniają założenia } \varphi \text{ to wynik } d' \text{ spełnia wymagania } \psi.\\ \text{w skrócie: } [\varphi]\mathcal{P}[\psi]\end{array}\right)$$

**Na przykład**: $[d \geqslant 0]\mathcal{P}_{\text{pierwiastek}}[d' \cdot d' \leqslant d < (d'+1)^2]$

**Więc**: Tego typu własności można **dowodzić!** (Hoare [1969])

$\rightsquigarrow$ **formalna weryfikacja** programów

# Zastosowania formalnej weryfikacji

## Zastosowania formalnej weryfikacji

- 1998 r., Linia 14 paryskiego metra (autonomiczna)

  Kluczowe składniki oprogramowania sterującego ruchem

## Zastosowania formalnej weryfikacji

- 1998 r., Linia 14 paryskiego metra (autonomiczna)

  Kluczowe składniki oprogramowania sterującego ruchem

  – 110'000 linii kodu wraz z dowodami

  – 0 błędów wykrytych do 2009 roku

## Zastosowania formalnej weryfikacji

- 1998 r., Linia 14 paryskiego metra (autonomiczna)

  Kluczowe składniki oprogramowania sterującego ruchem

  – 110'000 linii kodu wraz z dowodami

  – 0 błędów wykrytych do 2009 roku

- 1999 r., weryfikacja interpretera w tzw. *smart cards*

## Zastosowania formalnej weryfikacji

- 1998 r., Linia 14 paryskiego metra (autonomiczna)

  Kluczowe składniki oprogramowania sterującego ruchem

  – 110'000 linii kodu wraz z dowodami

  – 0 błędów wykrytych do 2009 roku

- 1999 r., weryfikacja interpretera w tzw. *smart cards*

- 2005 r., połączenie terminali na CDG (system VAL)

## Zastosowania formalnej weryfikacji

- 1998 r., Linia 14 paryskiego metra (autonomiczna)

  Kluczowe składniki oprogramowania sterującego ruchem

  – 110'000 linii kodu wraz z dowodami

  – 0 błędów wykrytych do 2009 roku

- 1999 r., weryfikacja interpretera w tzw. *smart cards*

- 2005 r., połączenie terminali na CDG (system VAL)

  $\rightsquigarrow \geqslant 20$ innych lokacji na całym świecie

## Zastosowania formalnej weryfikacji

- 1998 r., Linia 14 paryskiego metra (autonomiczna)

  Kluczowe składniki oprogramowania sterującego ruchem

  – 110'000 linii kodu wraz z dowodami

  – 0 błędów wykrytych do 2009 roku

- 1999 r., weryfikacja interpretera w tzw. *smart cards*

- 2005 r., połączenie terminali na CDG (system VAL)

  $\rightsquigarrow \geqslant$ 20 innych lokacji na całym świecie

} **B-method**

## Zastosowania formalnej weryfikacji

- 1998 r., Linia 14 paryskiego metra (autonomiczna)

  Kluczowe składniki oprogramowania sterującego ruchem

  – 110'000 linii kodu wraz z dowodami

  – 0 błędów wykrytych do 2009 roku

- 1999 r., weryfikacja interpretera w tzw. *smart cards*

- 2005 r., połączenie terminali na CDG (system VAL)

  $\rightsquigarrow \geqslant$ 20 innych lokacji na całym świecie

**B-method**

- 1997–99 r., pełna weryfikacja procesora Intel Pentium 4

## Zastosowania formalnej weryfikacji

- 1998 r., Linia 14 paryskiego metra (autonomiczna)

  Kluczowe składniki oprogramowania sterującego ruchem

  – 110'000 linii kodu wraz z dowodami

  – 0 błędów wykrytych do 2009 roku

- 1999 r., weryfikacja interpretera w tzw. *smart cards*

- 2005 r., połączenie terminali na CDG (system VAL)

  $\rightsquigarrow \geqslant$ 20 innych lokacji na całym świecie

**B-method**

- 1997–99 r., pełna weryfikacja procesora Intel Pentium 4

- 2005 r., liczne podsystemy Airbusa A380

## Zastosowania formalnej weryfikacji

- 1998 r., Linia 14 paryskiego metra (autonomiczna)

  Kluczowe składniki oprogramowania sterującego ruchem

  – 110'000 linii kodu wraz z dowodami

  – 0 błędów wykrytych do 2009 roku

- 1999 r., weryfikacja interpretera w tzw. *smart cards*

- 2005 r., połączenie terminali na CDG (system VAL)

  $\rightsquigarrow \geqslant$ 20 innych lokacji na całym świecie

} **B-method**

- 1997–99 r., pełna weryfikacja procesora Intel Pentium 4

- 2005 r., liczne podsystemy Airbusa A380

- 2018 r., pełna weryfikacja implementacji TLS u Amazona

## Zastosowania formalnej weryfikacji

- 1998 r., Linia 14 paryskiego metra (autonomiczna)

  Kluczowe składniki oprogramowania sterującego ruchem

  – 110'000 linii kodu wraz z dowodami

  – 0 błędów wykrytych do 2009 roku

- 1999 r., weryfikacja interpretera w tzw. *smart cards*

- 2005 r., połączenie terminali na CDG (system VAL)

  $\rightsquigarrow \geqslant$ 20 innych lokacji na całym świecie

} **B-method**

- 1997–99 r., pełna weryfikacja procesora Intel Pentium 4

- 2005 r., liczne podsystemy Airbusa A380

- 2018 r., pełna weryfikacja implementacji TLS u Amazona

- ...

# Nie ma nic za darmo...

## Nie ma nic za darmo...

**Hipoteza** (Goldbach [1742]) [AKA Ósmy problem Hilberta]

Każda liczba parzysta większa od 2 jest sumą dwóch liczb pierwszych.

## Nie ma nic za darmo...

**Hipoteza** (Goldbach [1742]) [AKA Ósmy problem Hilberta]

Każda liczba parzysta większa od 2 jest sumą dwóch liczb pierwszych.

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

## Nie ma nic za darmo...

**Hipoteza** (Goldbach [1742]) [AKA Ósmy problem Hilberta]

Każda liczba parzysta większa od 2 jest sumą dwóch liczb pierwszych.

$\mathcal{P}_{\text{Gold}}$ :
```
n := 2;
while true do {
    n := n + 2;
    if (n nie jest sumą dwóch liczb pierwszych) then
        return 1;
}
```

## Nie ma nic za darmo...

**Hipoteza** (Goldbach [1742]) [AKA Ósmy problem Hilberta]

Każda liczba parzysta większa od 2 jest sumą dwóch liczb pierwszych.

$\mathcal{P}_{\text{Gold}}$ :
```
n := 2;
while true do {
    n := n + 2;
    if (n nie jest sumą dwóch liczb pierwszych) then
        return 1;
}
```

**Fakt**: $\neg$[**Hipoteza Goldbacha**] wtedy i tylko wtedy gdy $[\ ]\mathcal{P}_{\text{Gold}}[d' = 1]$

## Nie ma nic za darmo...

**Hipoteza** (Goldbach [1742]) [AKA Ósmy problem Hilberta]

Każda liczba parzysta większa od 2 jest sumą dwóch liczb pierwszych.

$\mathcal{P}_{\text{Gold}}$ :
```
n := 2;
while true do {
    n := n + 2;
    if (n nie jest sumą dwóch liczb pierwszych) then
        return 1;
}
```

**Fakt**: $\neg$[**Hipoteza Goldbacha**] wtedy i tylko wtedy gdy $[\,]\mathcal{P}_{\text{Gold}}[d' = 1]$

⇝ ***wystarczy*** pokazać, że $\neg[\,]\mathcal{P}_{\text{Gold}}[d' = 1]$...

## Nie ma nic za darmo...

**Hipoteza** (Goldbach [1742]) [AKA Ósmy problem Hilberta]

Każda liczba parzysta większa od 2 jest sumą dwóch liczb pierwszych.

$\mathcal{P}_{\text{Gold}}$ :
```
n := 2;
while true do {
   n := n + 2;
   if (n nie jest sumą dwóch liczb pierwszych) then
      return 1;
}
```

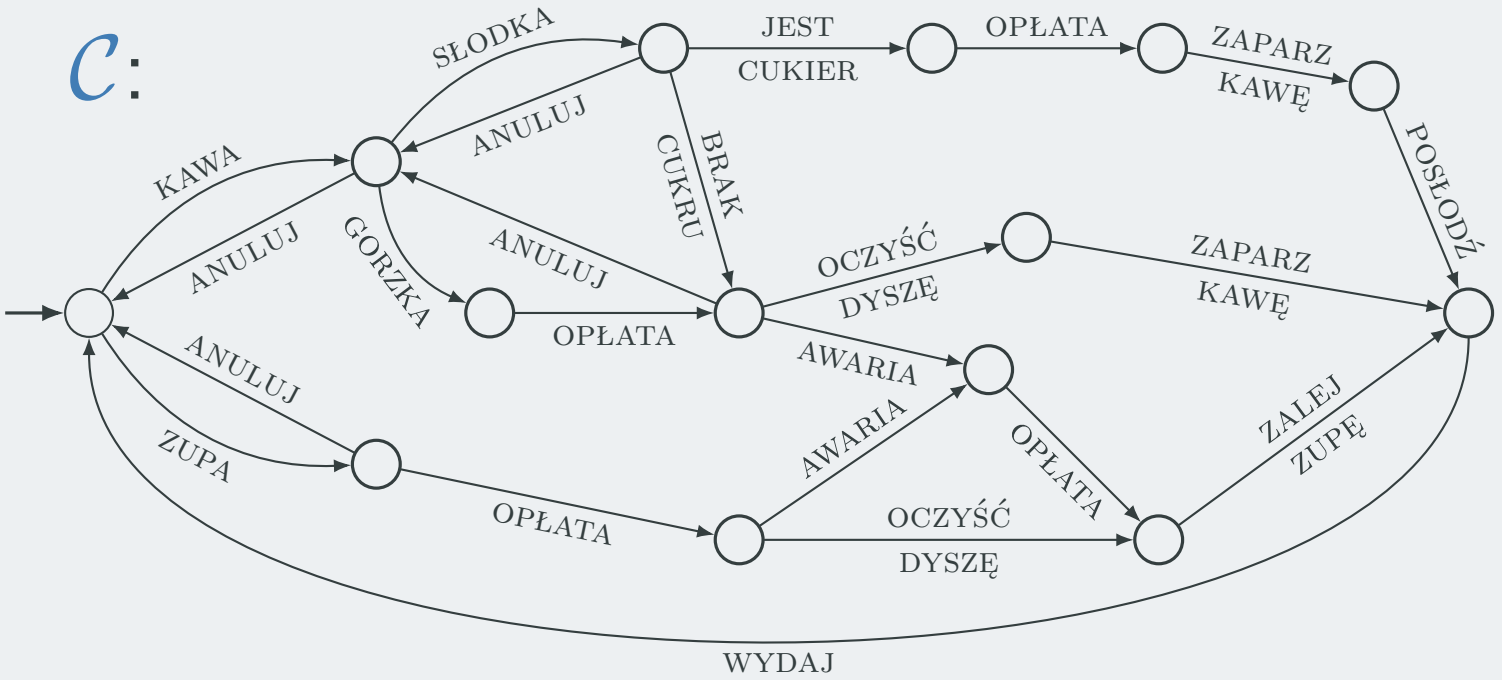
**Fakt**: $\neg$[**Hipoteza Goldbacha**] wtedy i tylko wtedy gdy $[\,]\mathcal{P}_{\text{Gold}}[d' = 1]$

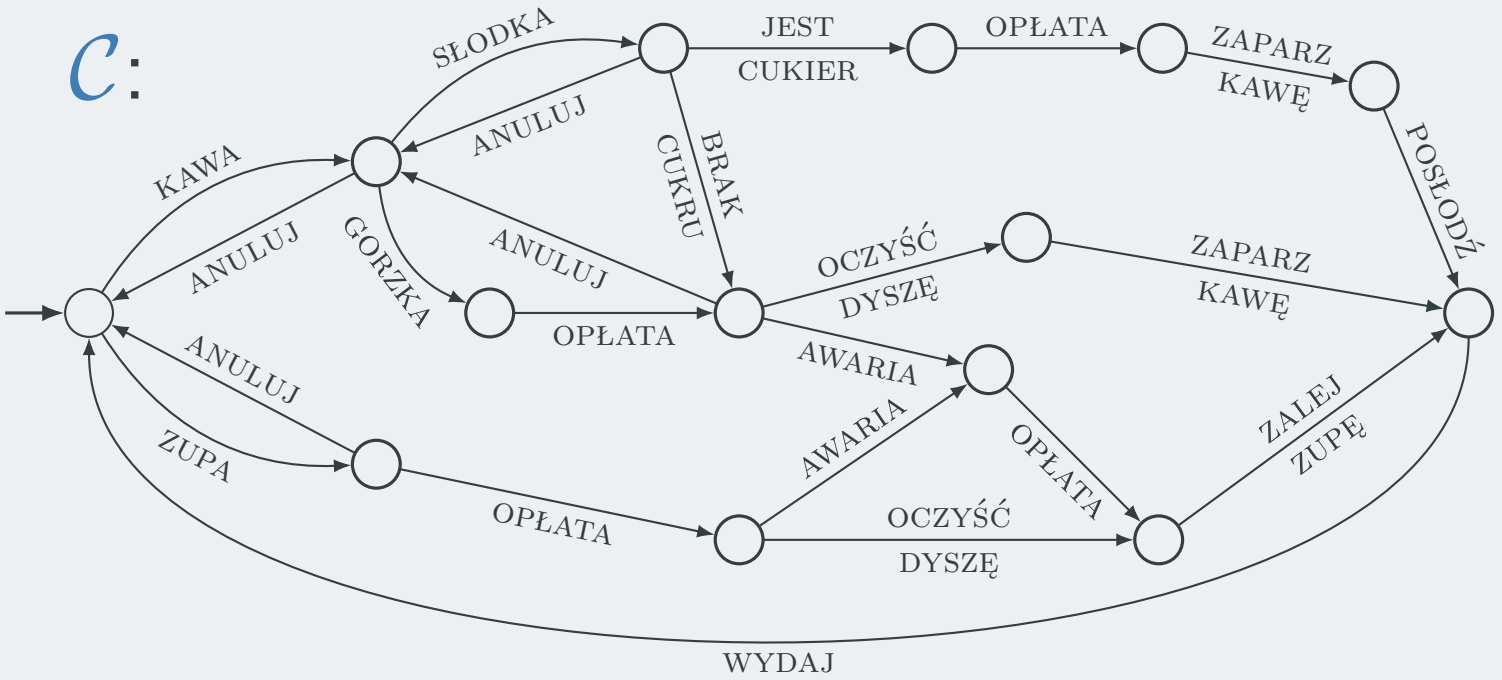
⇝ ***wystarczy*** pokazać, że $\neg[\,]\mathcal{P}_{\text{Gold}}[d' = 1]$...

[ a nawet gorzej, bo program może szukać dowodów ]

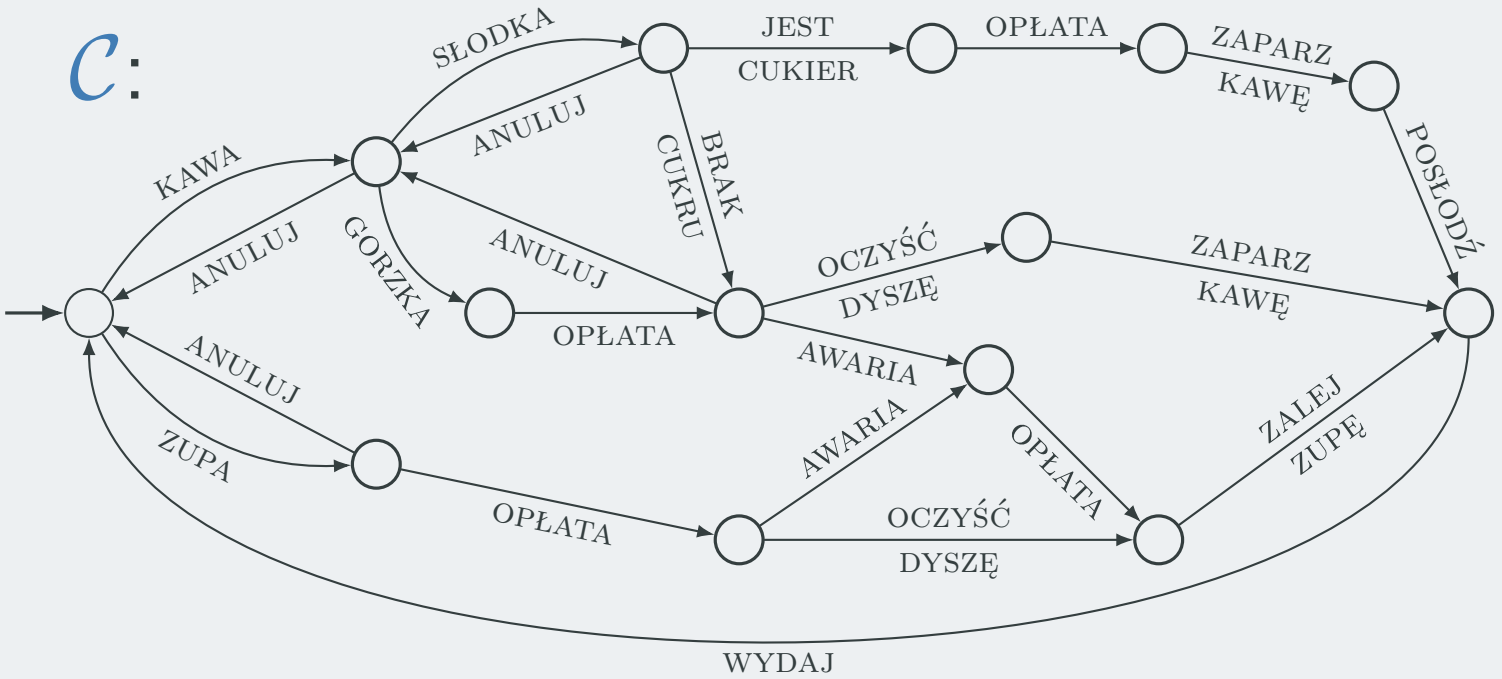
„Wszystkiemu winne są liczby.”

„Wszystkiemu winne są liczby.” $\rightsquigarrow$ rozważmy maszyny *bez liczb* = **automaty**


$\mathcal{C}$:
SŁODKA
JEST
CUKIER
OPŁATA
ZAPARZ
KAWĘ
POSŁODŹ
KAWA
ANULUJ
ANULUJ
BRAK
CUKRU
GORZKA
ANULUJ
OPŁATA
OCZYŚĆ
DYSZĘ
ZAPARZ
KAWĘ
AWARIA
ANULUJ
ZUPA
OPŁATA
AWARIA
OPŁATA
OCZYŚĆ
DYSZĘ
ZALEJ
ZUPĘ
WYDAJ

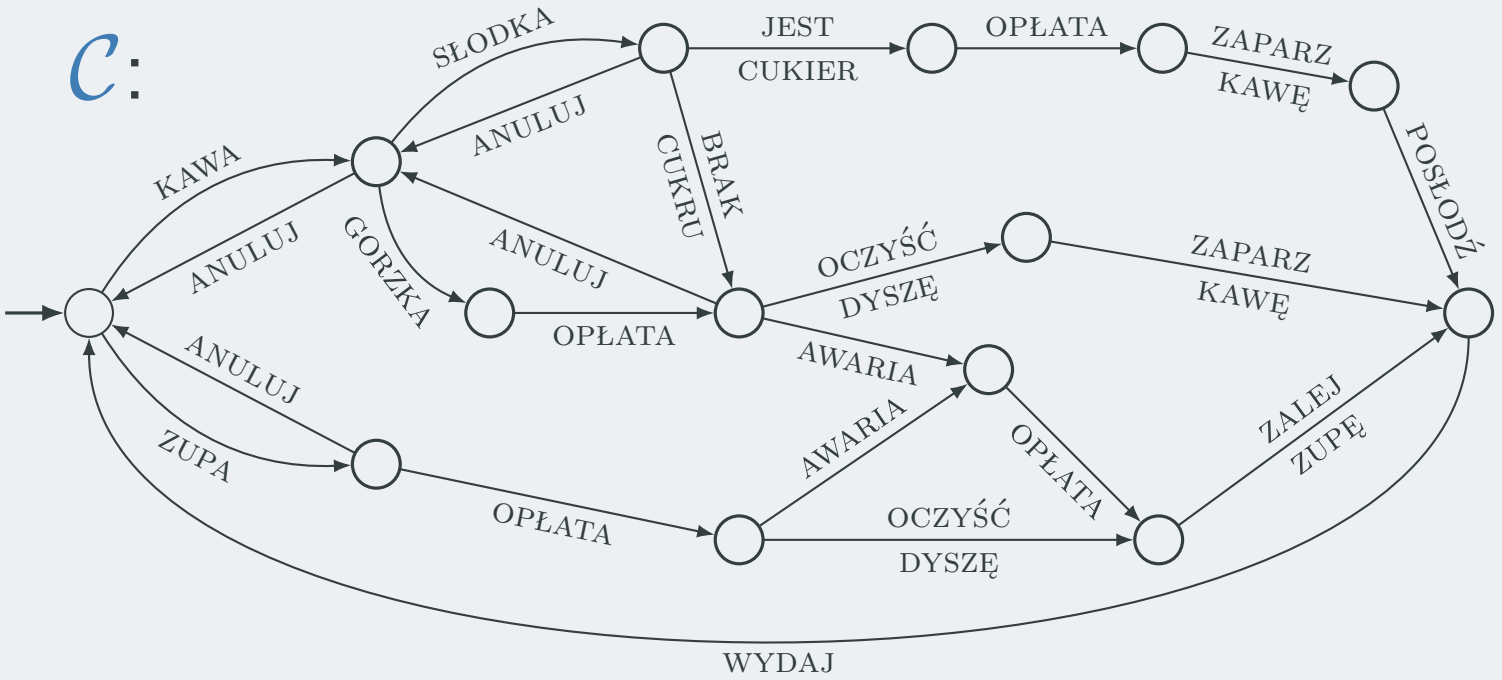



**Zbiór akcji**: $\mathbf{A} = \{\text{KAWA}, \text{ZUPA}, \text{OPŁATA}, \ldots\}$



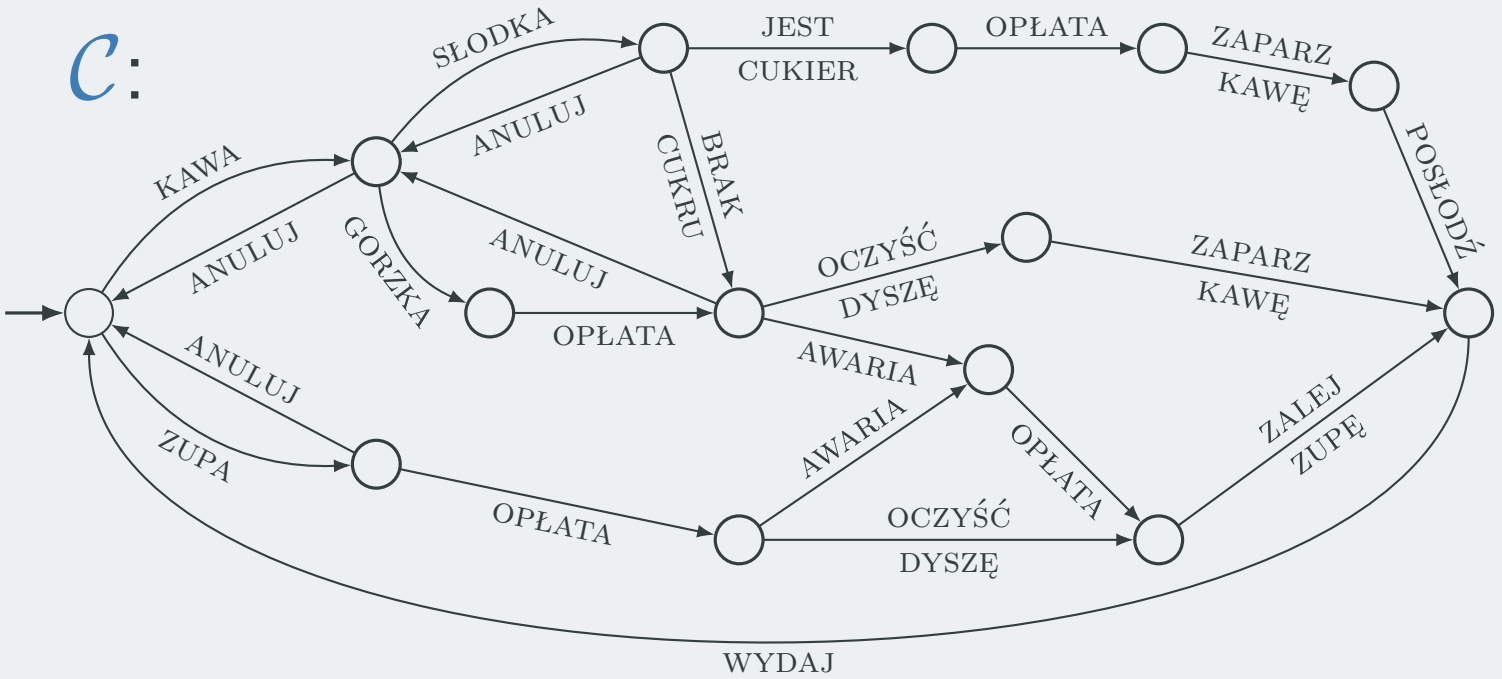

**Zbiór akcji**: $\mathbf{A} = \{\text{KAWA}, \text{ZUPA}, \text{OPŁATA}, \ldots\}$

**Możliwe przebiegi**: $[\![\mathcal{C}]\!]$




**Zbiór akcji**: $\mathbf{A} = \{\text{KAWA}, \text{ZUPA}, \text{OPŁATA}, \ldots\}$

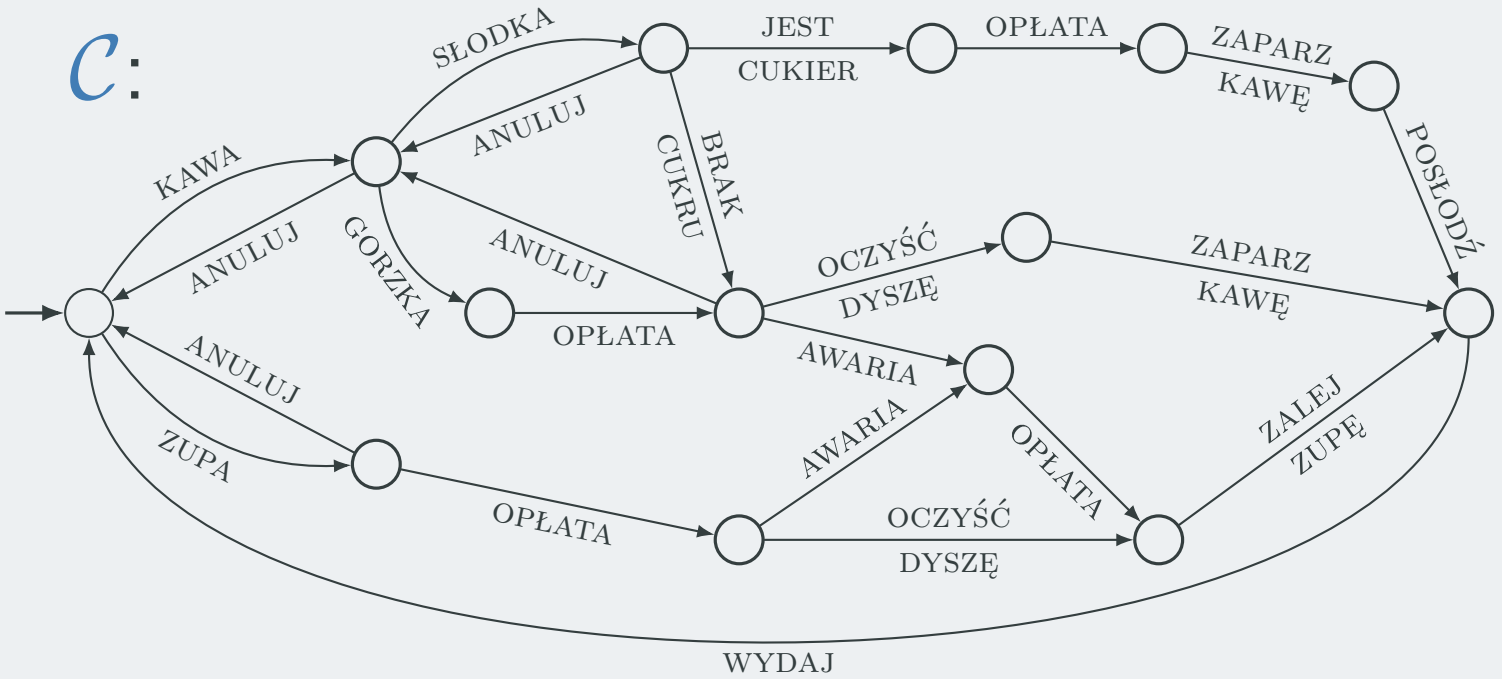
**Możliwe przebiegi**: $[\![\mathcal{C}]\!] \ni \langle \text{KAWA}, \text{ANULUJ}, \text{ZUPA}, \text{OPŁATA} \rangle$




**Zbiór akcji**: $\mathbf{A} = \{\text{KAWA}, \text{ZUPA}, \text{OPŁATA}, \ldots\}$

**Możliwe przebiegi**: $[\![\mathcal{C}]\!] \ni \langle \text{KAWA}, \text{ANULUJ}, \text{ZUPA}, \text{OPŁATA} \rangle$

**Specyfikacja**: $\mathcal{S}$: "nie WYDAM napoju bez OPŁATY"

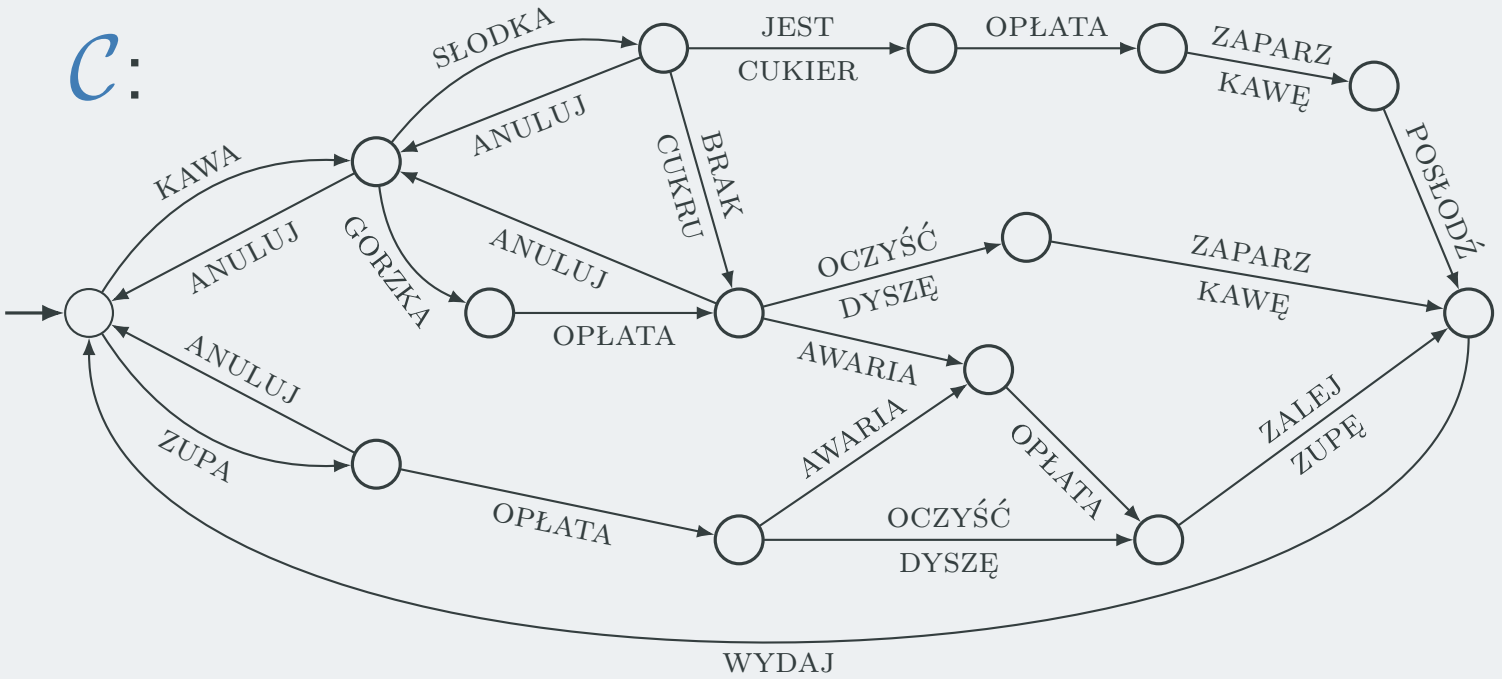



**Zbiór akcji**: $\mathbf{A} = \{\text{KAWA, ZUPA, OPŁATA, } \ldots\}$

**Możliwe przebiegi**: $[\![\mathcal{C}]\!] \ni \langle \text{KAWA, ANULUJ, ZUPA, OPŁATA} \rangle$

**Specyfikacja**: $\mathcal{S}$: "nie WYDAM napoju bez OPŁATY"

**Poprawne przebiegi**: $[\![\mathcal{S}]\!]$

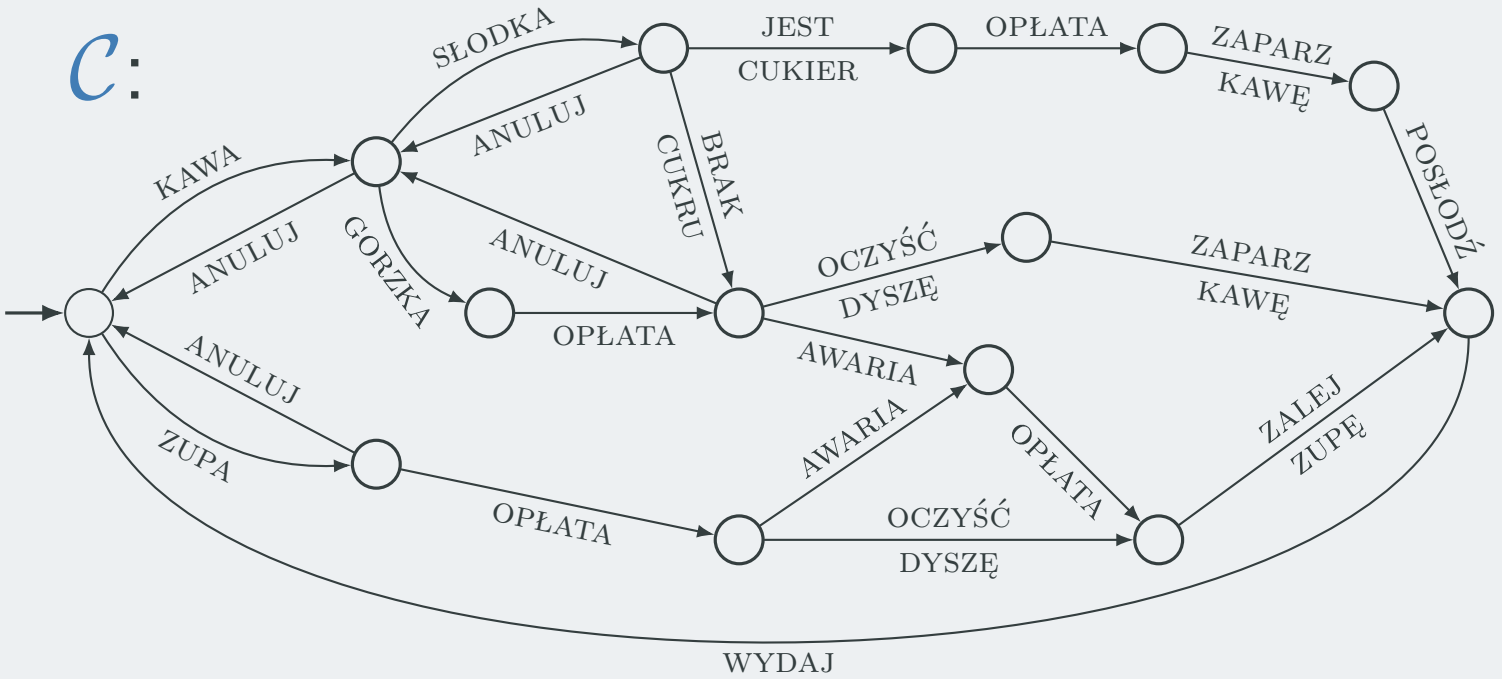



**Zbiór akcji**: $\mathbf{A} = \{\text{KAWA, ZUPA, OPŁATA, } \ldots\}$

**Możliwe przebiegi**: $[\![\mathcal{C}]\!] \ni \langle \text{KAWA, ANULUJ, ZUPA, OPŁATA} \rangle$

**Specyfikacja**: $\mathcal{S}$: "nie WYDAM napoju bez OPŁATY"

**Poprawne przebiegi**: $[\![\mathcal{S}]\!] \ni \langle \text{OPŁATA, AWARIA, WYDAJ} \rangle$




**Zbiór akcji**: $\mathbf{A} = \{\text{KAWA}, \text{ZUPA}, \text{OPŁATA}, \ldots\}$

**Możliwe przebiegi**: $[\![\mathcal{C}]\!] \ni \langle \text{KAWA}, \text{ANULUJ}, \text{ZUPA}, \text{OPŁATA} \rangle$

**Specyfikacja**: $\mathcal{S}$: "nie WYDAM napoju bez OPŁATY"

**Poprawne przebiegi**: $[\![\mathcal{S}]\!] \ni \langle \text{OPŁATA}, \text{AWARIA}, \text{WYDAJ} \rangle$

**Problem model-checkingu**:

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

**Specyfikacja**: $\mathcal{S}$: "nie WYDAM napoju bez OPŁATY"

**Specyfikacja**: $\mathcal{S}$: "nie WYDAM napoju bez OPŁATY"

---

**1. Formuła MSO**: $\varphi$: $\forall t.\ \textsc{wydaj}(t) \Rightarrow \exists t' < t.\ \textsc{opłata}(t')$

**Specyfikacja**: $\mathcal{S}$: "nie WYDAM napoju bez OPŁATY"

---

**1. Formuła MSO**: $\varphi$: $\forall t.\ \textsc{wydaj}(t) \Rightarrow \exists t' < t.\ \textsc{opłata}(t')$

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

**Specyfikacja**: $\mathcal{S}$: "nie WYDAM napoju bez OPŁATY"

---

**1. Formuła MSO**: $\varphi$: $\forall t.\ \textsc{wydaj}(t) \Rightarrow \exists t' < t.\ \textsc{opłata}(t')$

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

---

**2. Wyrażenie regularne**: $R$: $[\hat{}\textsc{wydaj}]^* + \left([\hat{}\textsc{wydaj}]^* \cdot \textsc{opłata} \cdot \mathbf{A}^*\right)$

**Specyfikacja**: $\mathcal{S}$: "nie WYDAM napoju bez OPŁATY"

---

**1. Formuła MSO**: $\varphi$: $\forall t.\ \text{WYDAJ}(t) \Rightarrow \exists t' < t.\ \text{OPŁATA}(t')$

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

---

**2. Wyrażenie regularne**: $R$: $[\hat{}\text{WYDAJ}]^* + \left([\hat{}\text{WYDAJ}]^* \cdot \text{OPŁATA} \cdot \mathbf{A}^*\right)$

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

**Specyfikacja**: $\mathcal{S}$: "nie WYDAM napoju bez OPŁATY"

---

**1. Formuła MSO**: $\varphi$: $\forall t.\ \text{WYDAJ}(t) \Rightarrow \exists t' < t.\ \text{OPŁATA}(t')$

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

---

**2. Wyrażenie regularne**: $R$: $[\hat{}\text{WYDAJ}]^* + \left([\hat{}\text{WYDAJ}]^* \cdot \text{OPŁATA} \cdot \mathbf{A}^*\right)$

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

---

**3. Weryfikator**:

**Specyfikacja**: $\mathcal{S}$: "nie WYDAM napoju bez OPŁATY"

---

**1. Formuła MSO**: $\varphi$: $\forall t.\ \mathrm{WYDAJ}(t) \Rightarrow \exists t' < t.\ \mathrm{OPŁATA}(t')$

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

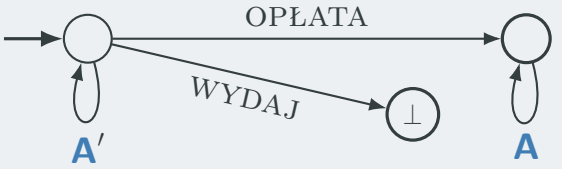
---

**2. Wyrażenie regularne**: $R$: $[\hat{}\mathrm{WYDAJ}]^* + \left([\hat{}\mathrm{WYDAJ}]^* \cdot \mathrm{OPŁATA} \cdot \mathbf{A}^*\right)$

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

---

**3. Weryfikator**: $\mathcal{V}$:



**Specyfikacja**: $\mathcal{S}$: "nie WYDAM napoju bez OPŁATY"

---

**1. Formuła MSO**: $\varphi$: $\forall t.\ \text{WYDAJ}(t) \Rightarrow \exists t' < t.\ \text{OPŁATA}(t')$

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

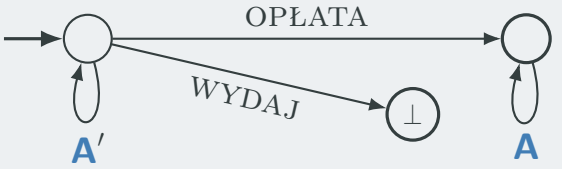
---

**2. Wyrażenie regularne**: $R$: $[\hat{}\text{WYDAJ}]^* + \left([\hat{}\text{WYDAJ}]^* \cdot \text{OPŁATA} \cdot \mathbf{A}^*\right)$

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

---

**3. Weryfikator**: $\mathcal{V}$:




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

---

**Specyfikacja**: $\mathcal{S}$: "nie WYDAM napoju bez OPŁATY"

---

**1. Formuła MSO**: $\varphi$: $\forall t.\ \text{WYDAJ}(t) \Rightarrow \exists t' < t.\ \text{OPŁATA}(t')$

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

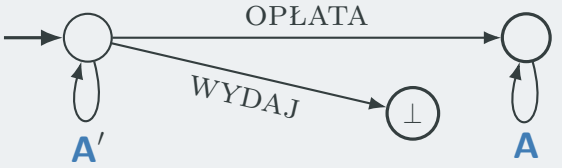
---

**2. Wyrażenie regularne**: $R$: $[\hat{}\text{WYDAJ}]^* + \left([\hat{}\text{WYDAJ}]^* \cdot \text{OPŁATA} \cdot \mathbf{A}^*\right)$

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

---

**3. Weryfikator**: $\mathcal{V}$:




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

---

**Twierdzenie** (Büchi, Elgot, Trachtenbrot [∼1960])

**Specyfikacja**: $\mathcal{S}$: "nie WYDAM napoju bez OPŁATY"

---

**1. Formuła MSO**: $\varphi$: $\forall t.\ \text{WYDAJ}(t) \Rightarrow \exists t' < t.\ \text{OPŁATA}(t')$

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

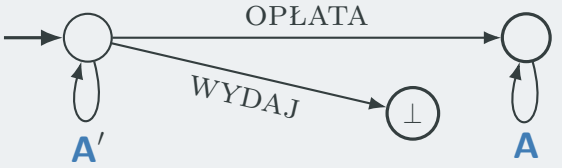
---

**2. Wyrażenie regularne**: $R$: $[\hat{}\text{WYDAJ}]^* + \left([\hat{}\text{WYDAJ}]^* \cdot \text{OPŁATA} \cdot \mathbf{A}^*\right)$

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

---

**3. Weryfikator**: $\mathcal{V}$:




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

---

**Twierdzenie** (Büchi, Elgot, Trachtenbrot [∼1960])

Istnieją obliczalne translacje pomiędzy **1.**, **2.** i **3.**.

**Specyfikacja**: $\mathcal{S}$: "nie WYDAM napoju bez OPŁATY"

---

**1. Formuła MSO**: $\varphi$: $\forall t.\ \text{WYDAJ}(t) \Rightarrow \exists t' < t.\ \text{OPŁATA}(t')$

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

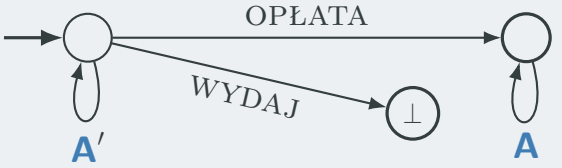
---

**2. Wyrażenie regularne**: $R$: $[\hat{}\text{WYDAJ}]^* + \left([\hat{}\text{WYDAJ}]^* \cdot \text{OPŁATA} \cdot \mathbf{A}^*\right)$

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

---

**3. Weryfikator**: $\mathcal{V}$:




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

---

**Twierdzenie** (Büchi, Elgot, Trachtenbrot [∼1960])

Istnieją obliczalne translacje pomiędzy **1.**, **2.** i **3.**.

**Wniosek**: By sprawdzić czy $[\![\mathcal{C}]\!] \subseteq [\![\mathcal{S}]\!]$ wystarczy

**Specyfikacja**: $\mathcal{S}$: "nie WYDAM napoju bez OPŁATY"

---

**1. Formuła MSO**: $\varphi$: $\forall t.\ \text{WYDAJ}(t) \Rightarrow \exists t' < t.\ \text{OPŁATA}(t')$

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

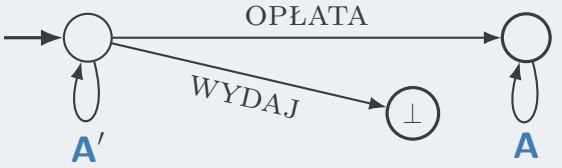
---

**2. Wyrażenie regularne**: $R$: $[\hat{}\text{WYDAJ}]^* + \left([\hat{}\text{WYDAJ}]^* \cdot \text{OPŁATA} \cdot \mathbf{A}^*\right)$

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

---

**3. Weryfikator**: $\mathcal{V}$:




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

---

**Twierdzenie** (Büchi, Elgot, Trachtenbrot [∼1960])

Istnieją obliczalne translacje pomiędzy **1.**, **2.** i **3.**.

**Wniosek**: By sprawdzić czy $[\![\mathcal{C}]\!] \subseteq [\![\mathcal{S}]\!]$ wystarczy

**1.** Przekształcić $\mathcal{S}$ w $\mathcal{V}$

**Specyfikacja**: $\mathcal{S}$: "nie WYDAM napoju bez OPŁATY"

---

**1. Formuła MSO**: $\varphi$: $\forall t.\ \text{WYDAJ}(t) \Rightarrow \exists t' < t.\ \text{OPŁATA}(t')$

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

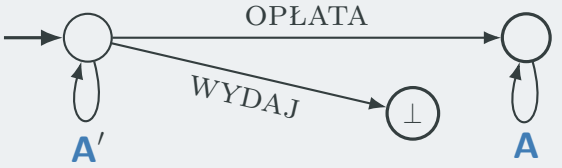
---

**2. Wyrażenie regularne**: $R$: $[\hat{}\text{WYDAJ}]^* + \left([\hat{}\text{WYDAJ}]^* \cdot \text{OPŁATA} \cdot \mathbf{A}^*\right)$

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

---

**3. Weryfikator**: $\mathcal{V}$:




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

---

**Twierdzenie** (Büchi, Elgot, Trachtenbrot [∼1960])

Istnieją obliczalne translacje pomiędzy **1.**, **2.** i **3.**.

**Wniosek**: By sprawdzić czy $[\![\mathcal{C}]\!] \subseteq [\![\mathcal{S}]\!]$ wystarczy

**1.** Przekształcić $\mathcal{S}$ w $\mathcal{V}$

**2.** Sprawdzić czy: $\mathcal{C} \times \mathcal{V} \longrightarrow^* (\_, \bot)$

# Podsumowanie

## Podsumowanie

**1.** Komputery są omylne.

## Podsumowanie

**1.** Komputery są omylne.

- Błędy sprzętowe są rzadkie (a da się sprawić by były jeszcze rzadsze).

## Podsumowanie

**1.** Komputery są omylne.

- Błędy sprzętowe są rzadkie (a da się sprawić by były jeszcze rzadsze).
- Błędy programistyczne są częste (i pomimo wysiłków trudno ich unikać).

## Podsumowanie

1. Komputery są omylne.
   - Błędy sprzętowe są rzadkie (a da się sprawić by były jeszcze rzadsze).
   - Błędy programistyczne są częste (i pomimo wysiłków trudno ich unikać).

2. Metody formalnej weryfikacji gwarantują matematyczne bezpieczeństwo.

## Podsumowanie

1. Komputery są omylne.
   - Błędy sprzętowe są rzadkie (a da się sprawić by były jeszcze rzadsze).
   - Błędy programistyczne są częste (i pomimo wysiłków trudno ich unikać).

2. Metody formalnej weryfikacji gwarantują matematyczne bezpieczeństwo.
   - Ich stosowanie wymaga wysiłku (i umiejętności $\Rightarrow$ koszty).

## Podsumowanie

1. Komputery są omylne.
   - Błędy sprzętowe są rzadkie (a da się sprawić by były jeszcze rzadsze).
   - Błędy programistyczne są częste (i pomimo wysiłków trudno ich unikać).

2. Metody formalnej weryfikacji gwarantują matematyczne bezpieczeństwo.
   - Ich stosowanie wymaga wysiłku (i umiejętności $\Rightarrow$ koszty).
   - W ogólności **nie** dają się zautomatyzować.

## Podsumowanie

1. Komputery są omylne.
   - Błędy sprzętowe są rzadkie (a da się sprawić by były jeszcze rzadsze).
   - Błędy programistyczne są częste (i pomimo wysiłków trudno ich unikać).

2. Metody formalnej weryfikacji gwarantują matematyczne bezpieczeństwo.
   - Ich stosowanie wymaga wysiłku (i umiejętności $\Rightarrow$ koszty).
   - W ogólności **nie** dają się zautomatyzować.

3. Weryfikacja automatów (= maszyn skończenie stanowych) jest prostsza.

## Podsumowanie

1. Komputery są omylne.
   - Błędy sprzętowe są rzadkie (a da się sprawić by były jeszcze rzadsze).
   - Błędy programistyczne są częste (i pomimo wysiłków trudno ich unikać).

2. Metody formalnej weryfikacji gwarantują matematyczne bezpieczeństwo.
   - Ich stosowanie wymaga wysiłku (i umiejętności $\Rightarrow$ koszty).
   - W ogólności **nie** dają się zautomatyzować.

3. Weryfikacja automatów (= maszyn skończenie stanowych) jest prostsza.
   - Faktyczne zastosowania do prostych sterowników i urządzeń.

## Podsumowanie

1. Komputery są omylne.
   - Błędy sprzętowe są rzadkie (a da się sprawić by były jeszcze rzadsze).
   - Błędy programistyczne są częste (i pomimo wysiłków trudno ich unikać).

2. Metody formalnej weryfikacji gwarantują matematyczne bezpieczeństwo.
   - Ich stosowanie wymaga wysiłku (i umiejętności $\Rightarrow$ koszty).
   - W ogólności **nie** dają się zautomatyzować.

3. Weryfikacja automatów (= maszyn skończenie stanowych) jest prostsza.
   - Faktyczne zastosowania do prostych sterowników i urządzeń.
   - Ciekawe konsekwencje matematyczne.