Czy komputer może się mylić?

Michał Skrzypczak
Uniwersytet Warszawski

LIX Szkoła Matematyki Poglądowej

18 lutego 2019, Wola Ducka

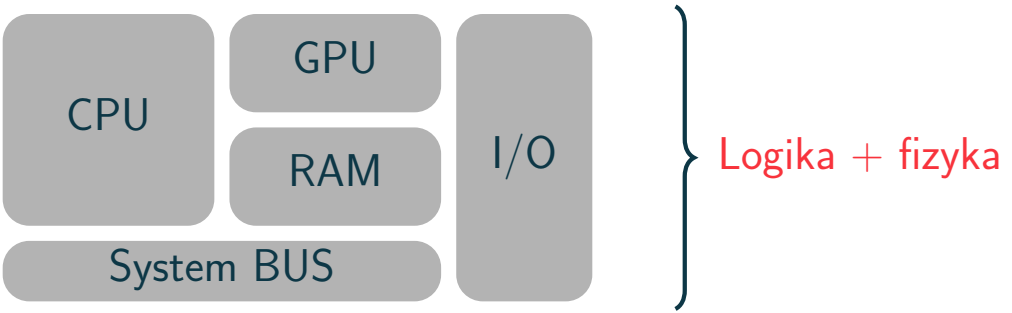
Komputer ma warstwy...

Komputer ma warstwy...

Sprzętowa:

Komputer ma warstwy...

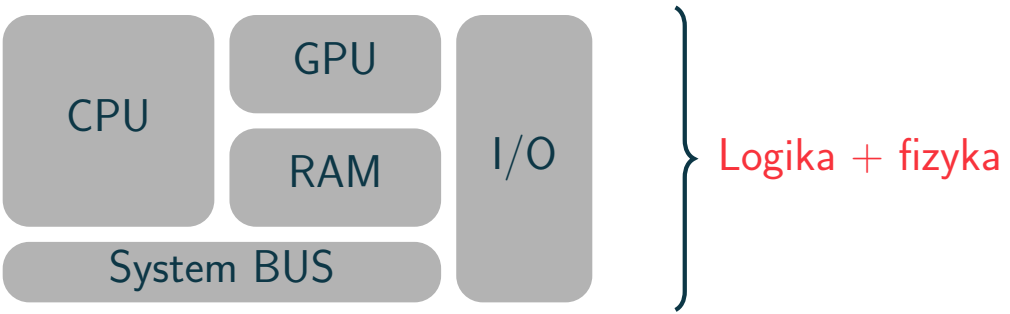
Sprzętowa:



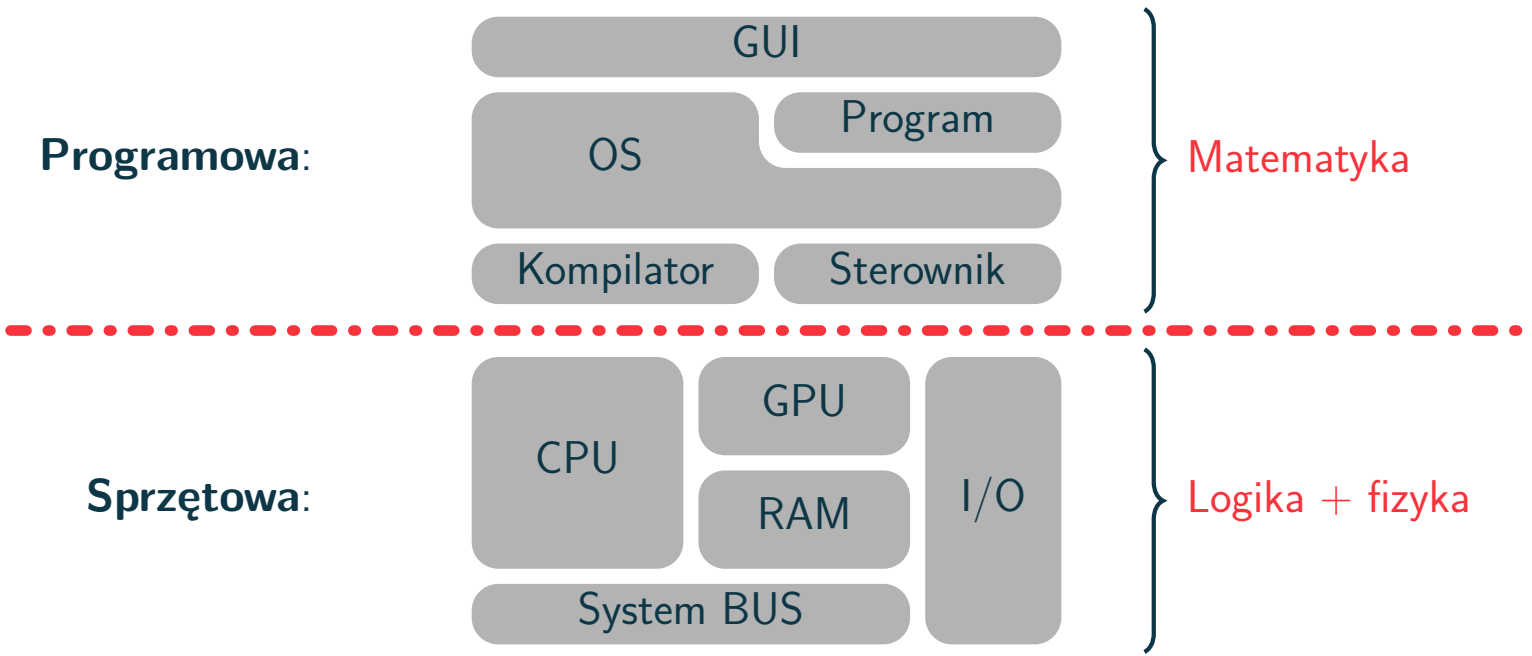
Komputer ma warstwy...

Programowa:

Sprzętowa:



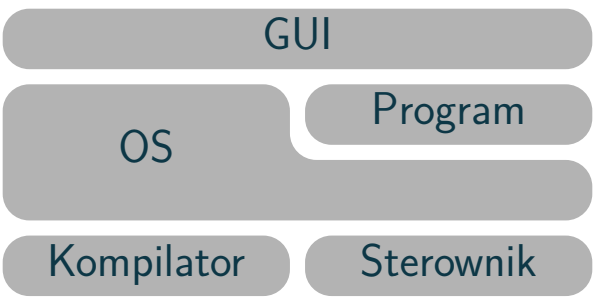
Komputer ma warstwy...



Komputer ma warstwy...

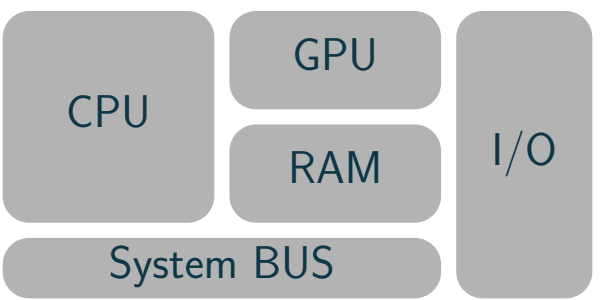
Organiczna:

Programowa:



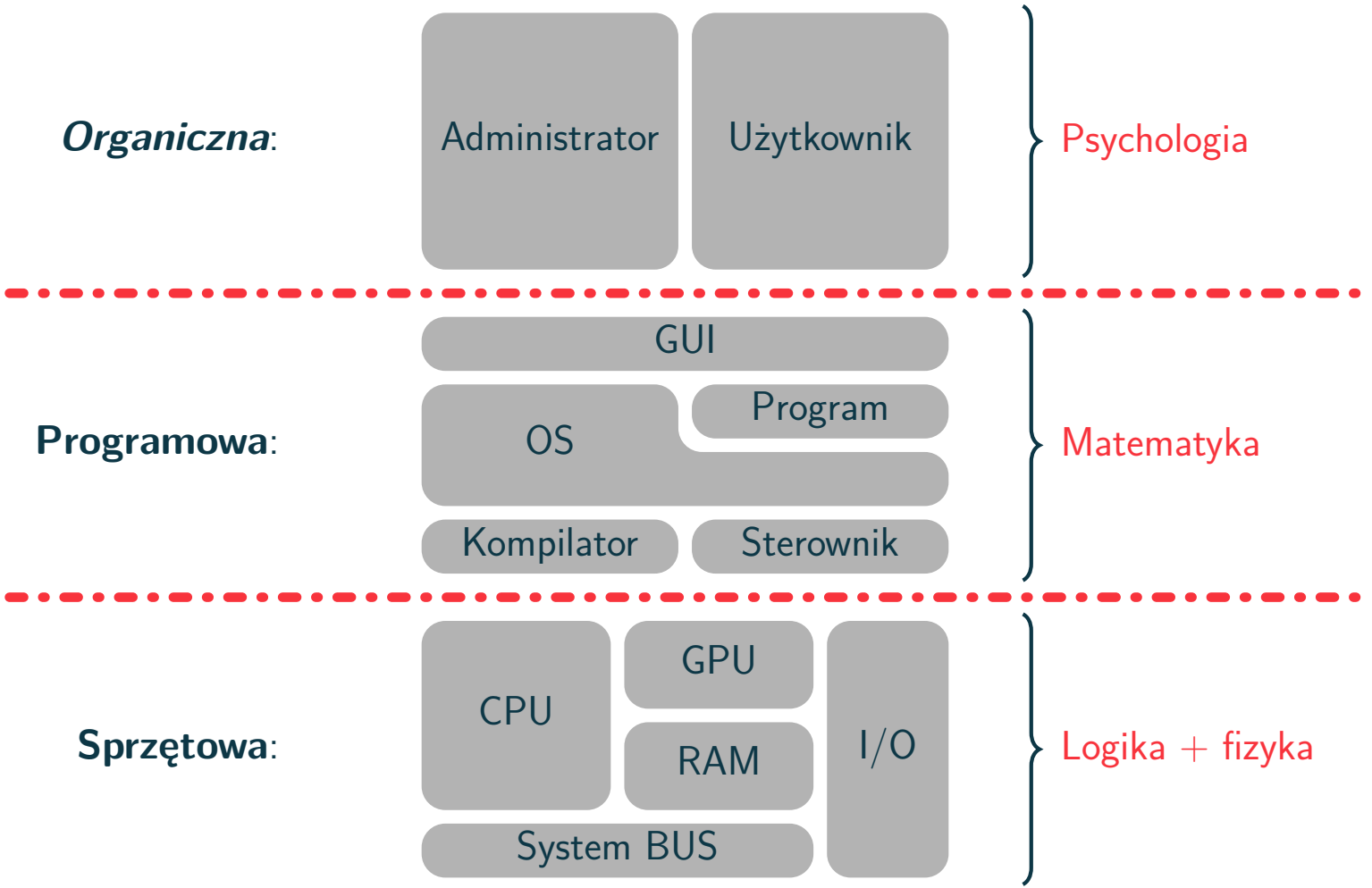
Matematyka

Sprzętowa:



Logika + fizyka

Komputer ma warstwy...

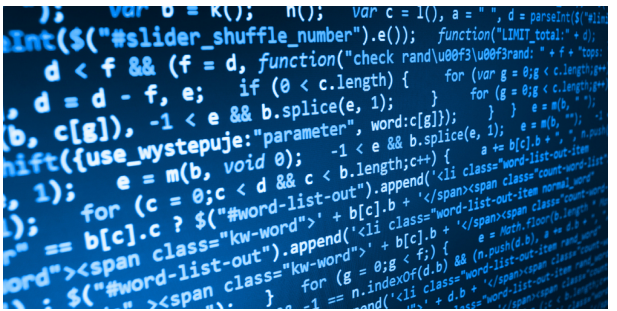


Komputer ma warstwy...

Organiczna:

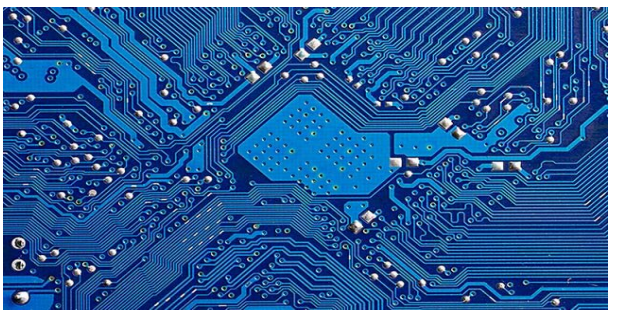
Psychologia

Programowa:



Matematyka

Sprzętowa:



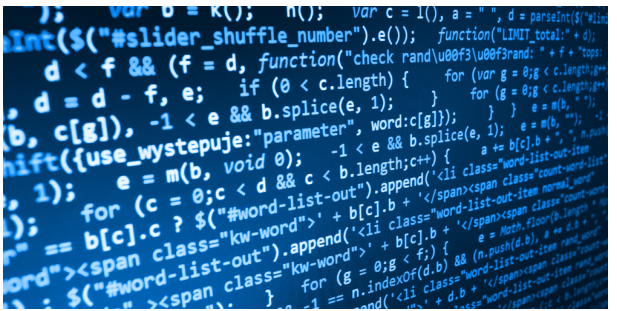
Logika + fizyka

Komputer ma warstwy...

Organiczna:

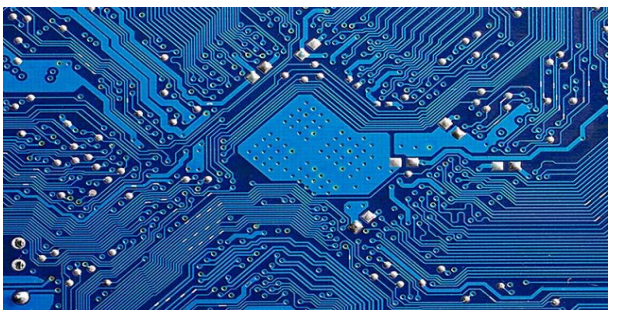
Psychologia

Programowa:



Matematyka

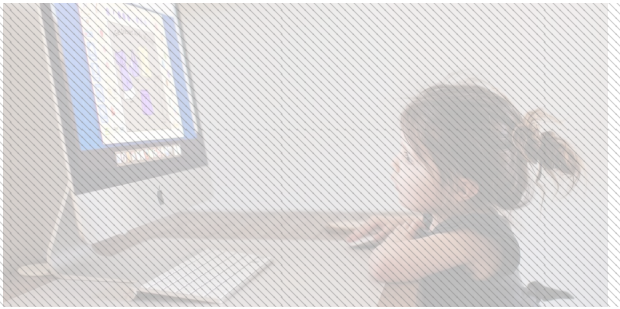
Sprzętowa:



Logika + fizyka

Komputer ma warstwy...

Organiczna:

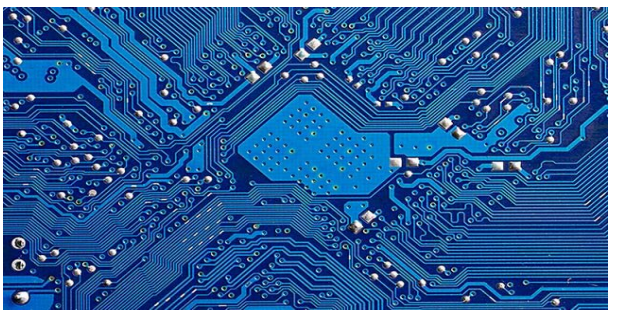


Psychologia

Programowa:

Matematyka

Sprzętowa:



Logika + fizyka

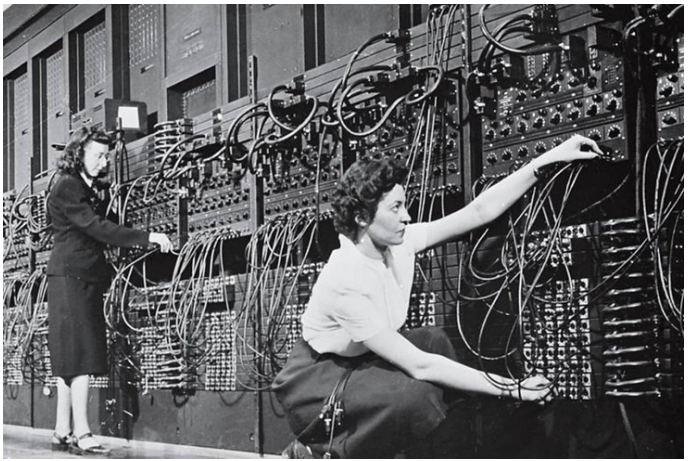
Błędy sprzętowe

Błędy sprzętowe

Kiedyś:

Błędy sprzętowe

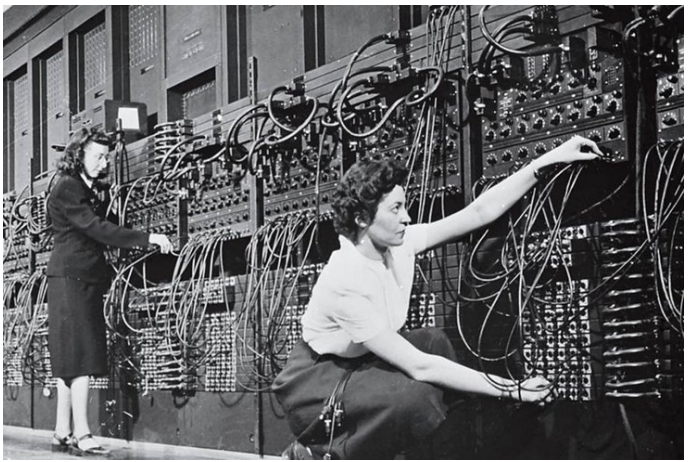
Kiedyś: **ENIAC** 1945 – 1955



Błędy sprzętowe

Kiedyś: **ENIAC** 1945 – 1955

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

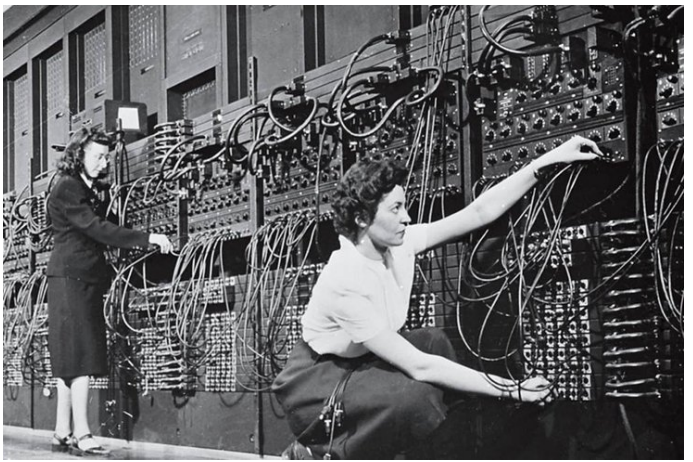


Błędy sprzętowe

Kiedyś: **ENIAC** 1945 – 1955

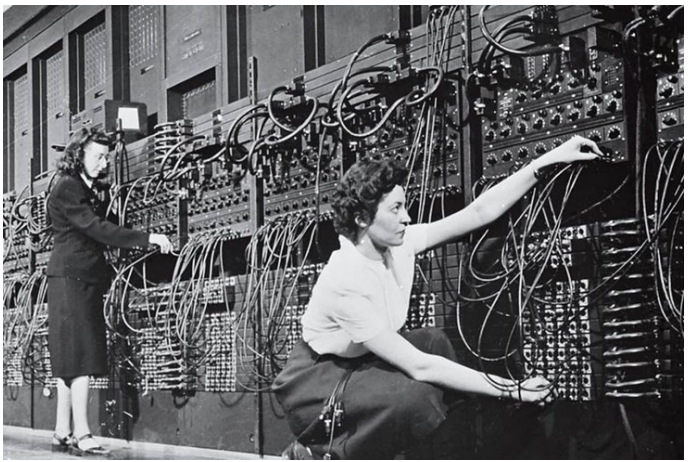
- ∼20'000 lamp próżniowych
- ∼5'000'000 ręcznych lutów
- 150kW zasilania
- Średni czas pomiędzy awariami:

  10 min. ('45r.) → >12 godz. ('55r.)



Błędy sprzętowe

Kiedyś: **ENIAC** 1945 – 1955

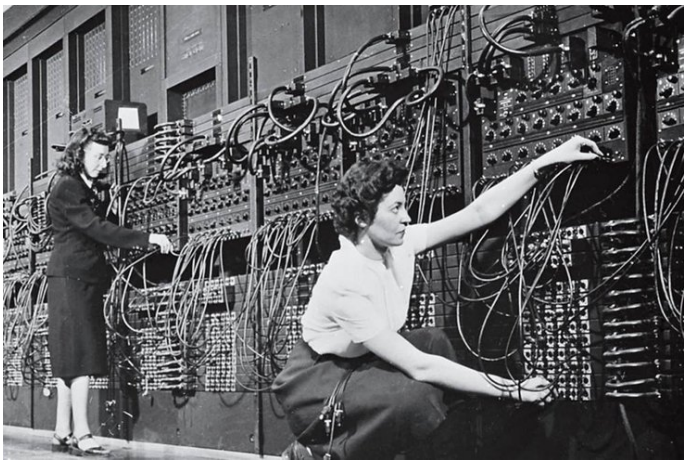


- ∼20'000 lamp próżniowych
- ∼5'000'000 ręcznych lutów
- 150kW zasilania
- Średni czas pomiędzy awariami:

  10 min. ('45r.) → >12 godz. ('55r.)
- Maksymalny czas pracy bez awarii: 116 godzin ('54r.)

Błędy sprzętowe

Kiedyś: **ENIAC** 1945 – 1955



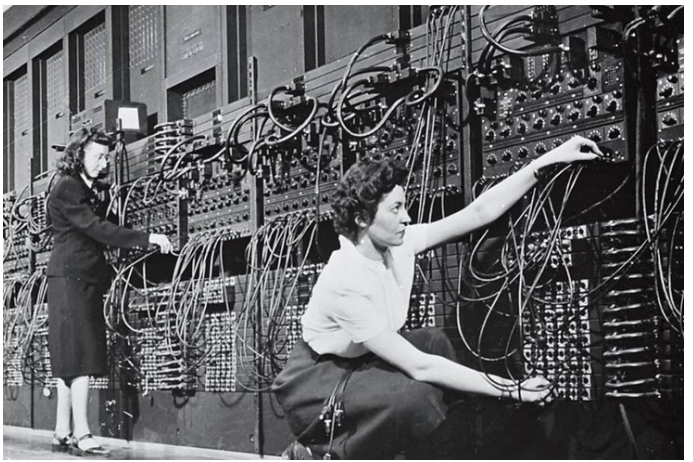
- ∼20'000 lamp próżniowych
- ∼5'000'000 ręcznych lutów
- 150kW zasilania
- Średni czas pomiędzy awariami:

  10 min. ('45r.) → >12 godz. ('55r.)
- Maksymalny czas pracy bez awarii: 116 godzin ('54r.)

Dziś:

Błędy sprzętowe

Kiedyś: **ENIAC** 1945 – 1955



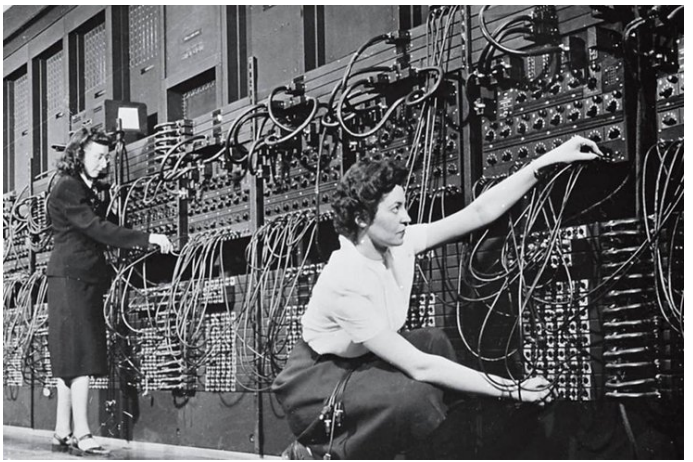
- ∼20'000 lamp próżniowych
- ∼5'000'000 ręcznych lutów
- 150kW zasilania
- Średni czas pomiędzy awariami:

  10 min. ('45r.) → >12 godz. ('55r.)
- Maksymalny 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:

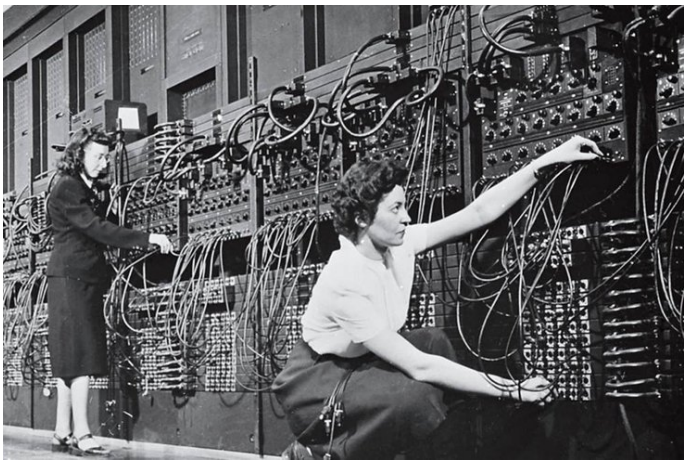
  10 min. ('45r.) → >12 godz. ('55r.)
- Maksymalny czas pracy bez awarii: 116 godzin ('54r.)

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.)
- Maksymalny czas pracy bez awarii: 116 godzin ('54r.)

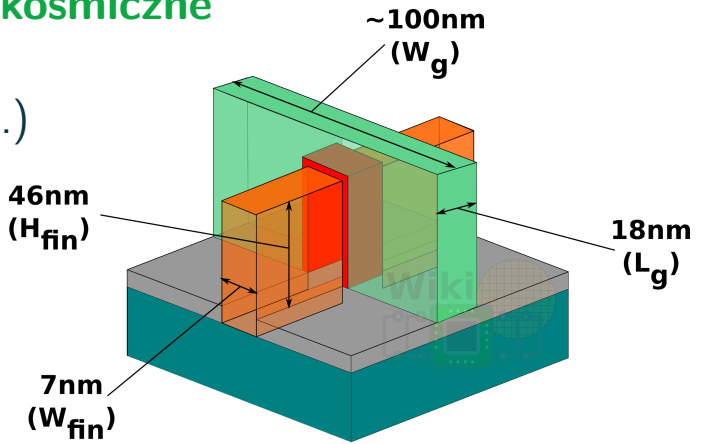
Dziś: **PC** 2018

- $\sim 2 \cdot 10^9$ 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)

Promieniowanie kosmiczne

Promieniowanie kosmiczne

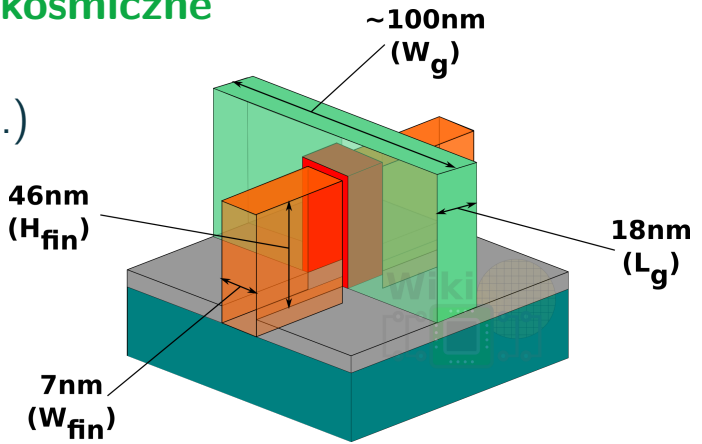
Układy scalone w technologii “10nm” (2018 r.)



Promieniowanie kosmiczne

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

Ścieżki szerokości kilkudziesięciu cząsteczek!

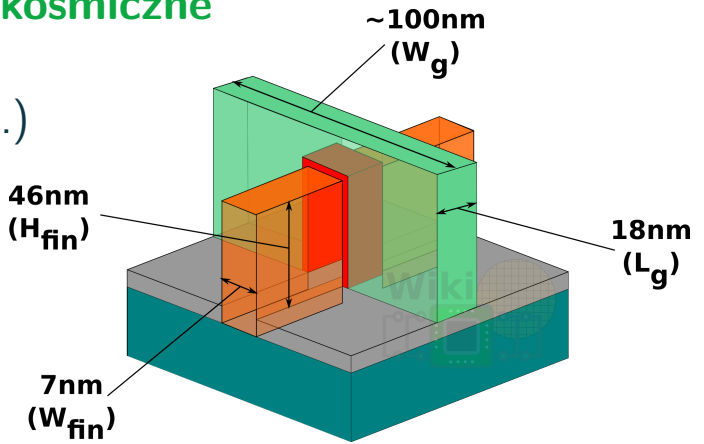


Promieniowanie kosmiczne

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

Ścieżki szerokości kilkudziesięciu cząsteczek!

⇝ ryzyko tzw. *Single Event Upset*

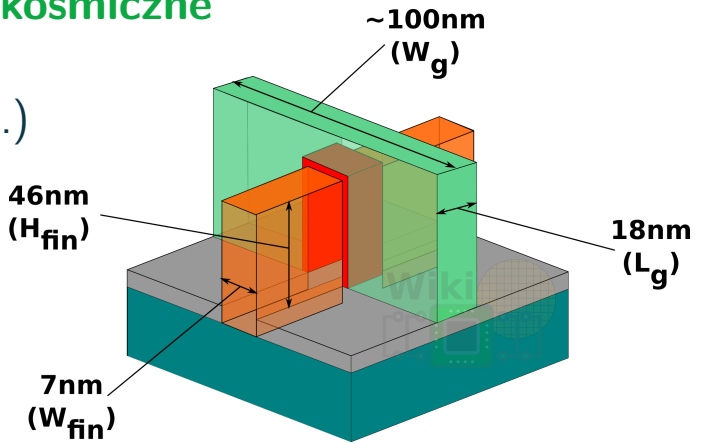


Promieniowanie kosmiczne

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

Ścieżki szerokości kilkudziesięciu cząsteczek!

⇝ ryzyko tzw. *Single Event Upset*



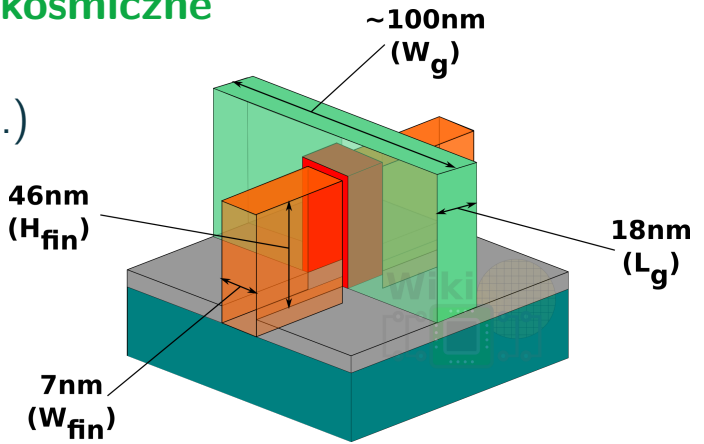
Potwierdzone przykłady:

Promieniowanie kosmiczne

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

Ścieżki szerokości kilkudziesięciu cząsteczek!

⇝ ryzyko tzw. *Single Event Upset*



Potwierdzone przykłady:

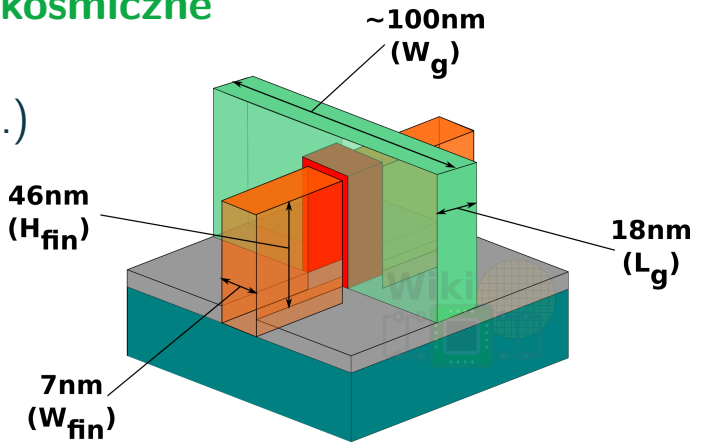
- W 1972 r. satelita komunikacyjny Hughes zawiesił komunikację na 96 sekund.

Promieniowanie kosmiczne

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

Ścieżki szerokości kilkudziesięciu cząsteczek!

⇝ ryzyko tzw. *Single Event Upset*



Potwierdzone przykłady:

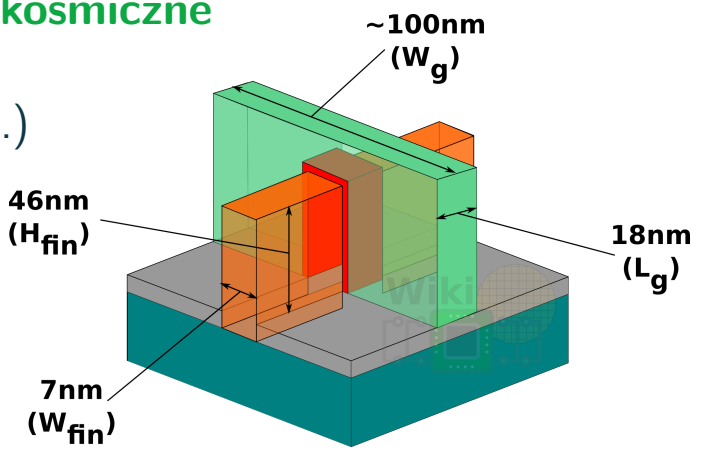
- W 1972 r. satelita komunikacyjny Hughes zawiesił komunikację na 96 sekund.
- W wyborach w 2003 r. w Schaerbeek (Belgia) zliczono o 4096 głosów za dużo.

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

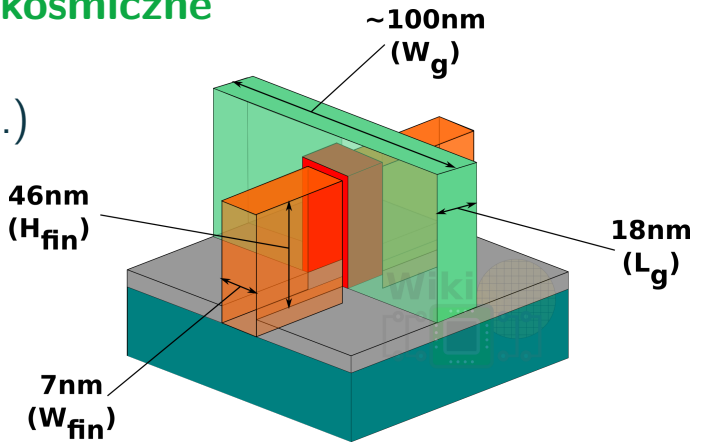
Metody zabezpieczeń:

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

Metody zabezpieczeń:

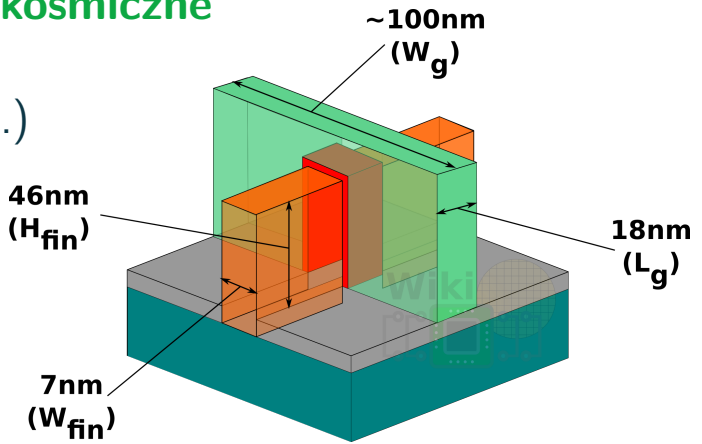
- Komputery dowodzenia ISS bazują na i386 (proces 1μm$=$ 100x10nm).

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

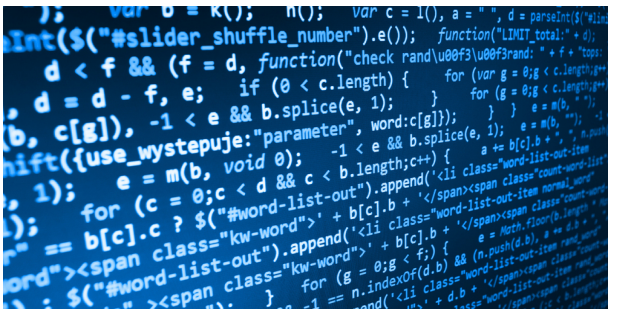
Metody zabezpieczeń:

- Komputery dowodzenia ISS bazują na i386 (proces 1μm$=$ 100x10nm).
- Potrojone systemy komputerowe w samolotach *fly-by-wire*.

Organiczna:

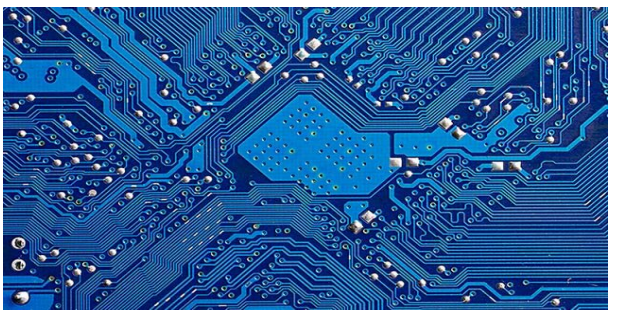
Psychologia

Programowa:



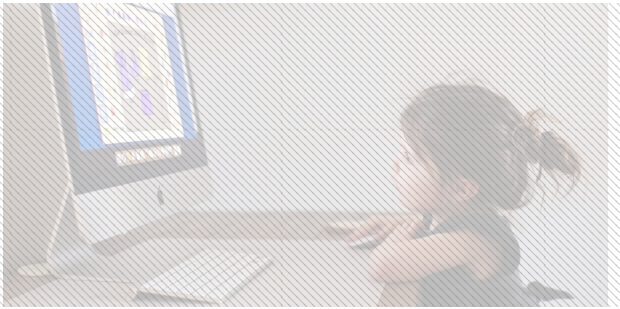
Matematyka

Sprzętowa:



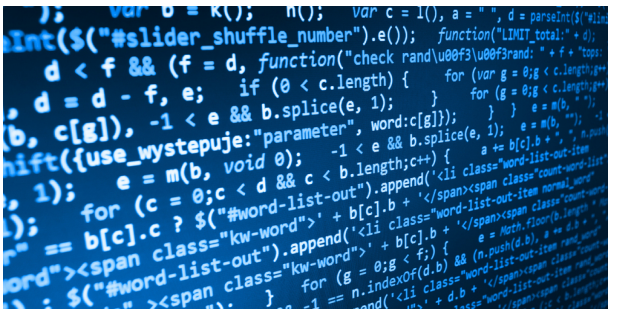
Logika + fizyka

Organiczna:



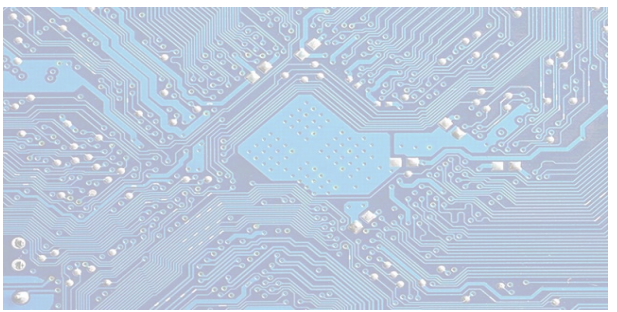
Psychologia

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 ofiary śmiertelne

Poprawne programy?

- 1986 r., sterowana komputerowo radioterapia Therac-25

  błąd kontroli współbieżności (tzw. *race condition*)

  ⇝ 6cioro pacjentów poważnie napromieniowanych, dwie ofiary śmiertelne

- 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 ofiary śmiertelne

- 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 ofiary śmiertelne

- 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 ofiary śmiertelne

- 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$)

Poprawne programy?

- 1986 r., sterowana komputerowo radioterapia Therac-25

  błąd kontroli współbieżności (tzw. *race condition*)

  ⇝ 6cioro pacjentów poważnie napromieniowanych, dwie ofiary śmiertelne

- 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 ofiary śmiertelne

- 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!

Poprawne programy!

$\mathcal{P}$

program

Poprawne programy!

$$\mathcal{P} \quad : \quad d$$

program dane

Poprawne programy!

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

program dane obliczenie

Poprawne programy!

$$\mathcal{P} \;:\; d \;\longmapsto^{\rho_0, \rho_1, \ldots, \rho_n = \rho} \mathcal{P}(d) = d'$$

program dane obliczenie wynik

Poprawne programy!

$$\underbrace{\mathcal{P}}_{\text{program}} : \underbrace{d}_{\text{dane}} \xmapsto{\rho_0, \rho_1, \ldots, \rho_n = \rho} \underbrace{\mathcal{P}(d) = d'}_{\text{wynik}}$$

obliczenie

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

Poprawne programy!

$$\underbrace{\mathcal{P}}_{\text{program}} \; : \; \underbrace{d}_{\text{dane}} \; \xmapsto{\overbrace{\rho_0, \rho_1, \ldots, \rho_n = \rho}^{\text{obliczenie}}} \; \underbrace{\mathcal{P}(d) = d'}_{\text{wynik}}$$

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

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

Poprawne programy!

$$\underbrace{\mathcal{P}}_{\text{program}} : \underbrace{d}_{\text{dane}} \xmapsto{\rho_0, \rho_1, \ldots, \rho_n = \rho} \underbrace{\mathcal{P}(d) = d'}_{\text{wynik}}$$

obliczenie

Fakt 1: $\mathcal{P}$, d, ρ 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!

$$\underbrace{\mathcal{P}}_{\text{program}} : \underbrace{d}_{\text{dane}} \xmapsto{\rho_0, \rho_1, \ldots, \rho_n = \rho} \underbrace{\mathcal{P}(d) = d'}_{\text{wynik}}$$

obliczenie

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

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

Czyli: Następujące zdanie jest własnością matematyczną:

$$\left(\text{Ilekroć dane } d \text{ spełniają założenia } \varphi \text{ to wynik } d' \text{ spełnia wymagania } \psi. \right)$$

Poprawne programy!

$$\underbrace{\mathcal{P}}_{\text{program}} : \underbrace{d}_{\text{dane}} \xmapsto{\rho_0, \rho_1, \ldots, \rho_n = \rho} \underbrace{\mathcal{P}(d) = d'}_{\text{wynik}}$$

obliczenie

Fakt 1: $\mathcal{P}$, d, ρ 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!

$$\underbrace{\mathcal{P}}_{\text{program}} : \underbrace{d}_{\text{dane}} \xmapsto{\rho_0, \rho_1, \ldots, \rho_n = \rho} \underbrace{\mathcal{P}(d) = d'}_{\text{wynik}}$$

obliczenie

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

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

Czyli: Następujące zdanie jest własnością matematyczną:

(Ilekroć dane d spełniają założenia φ to wynik d' spełnia wymagania ψ.
w skrócie: $[\varphi]\mathcal{P}[\psi]$)

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

Poprawne programy!

$$\underbrace{\mathcal{P}}_{\text{program}} : \underbrace{d}_{\text{dane}} \xmapsto{\overbrace{\rho_0, \rho_1, \ldots, \rho_n = \rho}^{\text{obliczenie}}} \underbrace{\mathcal{P}(d) = d'}_{\text{wynik}}$$

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

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

Czyli: Następujące zdanie jest własnością matematyczną:

(Ilekroć dane d spełniają założenia φ to wynik d' spełnia wymagania ψ.
w skrócie: $[\varphi]\mathcal{P}[\psi]$)

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

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

Poprawne programy!

$$\underbrace{\mathcal{P}}_{\text{program}} : \underbrace{d}_{\text{dane}} \xmapsto{\rho_0, \rho_1, \ldots, \rho_n = \rho} \underbrace{\mathcal{P}(d) = d'}_{\text{wynik}}$$

obliczenie

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

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

Czyli: Następujące zdanie jest własnością matematyczną:

(Ilekroć dane d spełniają założenia φ to wynik d' spełnia wymagania ψ.
w skrócie: $[\varphi]\mathcal{P}[\psi]$)

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

Więc: Tego typu własności można **dowodzić!** [logika **Hoare'a** (1969 r.)]

Poprawne programy!

$$\underbrace{\mathcal{P}}_{\text{program}} : \underbrace{d}_{\text{dane}} \xmapsto{\overbrace{\rho_0, \rho_1, \ldots, \rho_n = \rho}^{\text{obliczenie}}} \underbrace{\mathcal{P}(d) = d'}_{\text{wynik}}$$

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

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

Czyli: Następujące zdanie jest własnością matematyczną:

(Ilekroć dane d spełniają założenia φ to wynik d' spełnia wymagania ψ.
w skrócie: $[\varphi]\mathcal{P}[\psi]$)

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

Więc: Tego typu własności można **dowodzić!** [logika **Hoare'a** (1969 r.)]

$\rightsquigarrow$ **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**] $\iff$ $[\,]\mathcal{P}_{\text{Gold}}[d' = 1]$

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**] $\iff$ $[\,]\mathcal{P}_{\text{Gold}}[d' = 1]$

$\rightsquigarrow$ ***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;
}
```

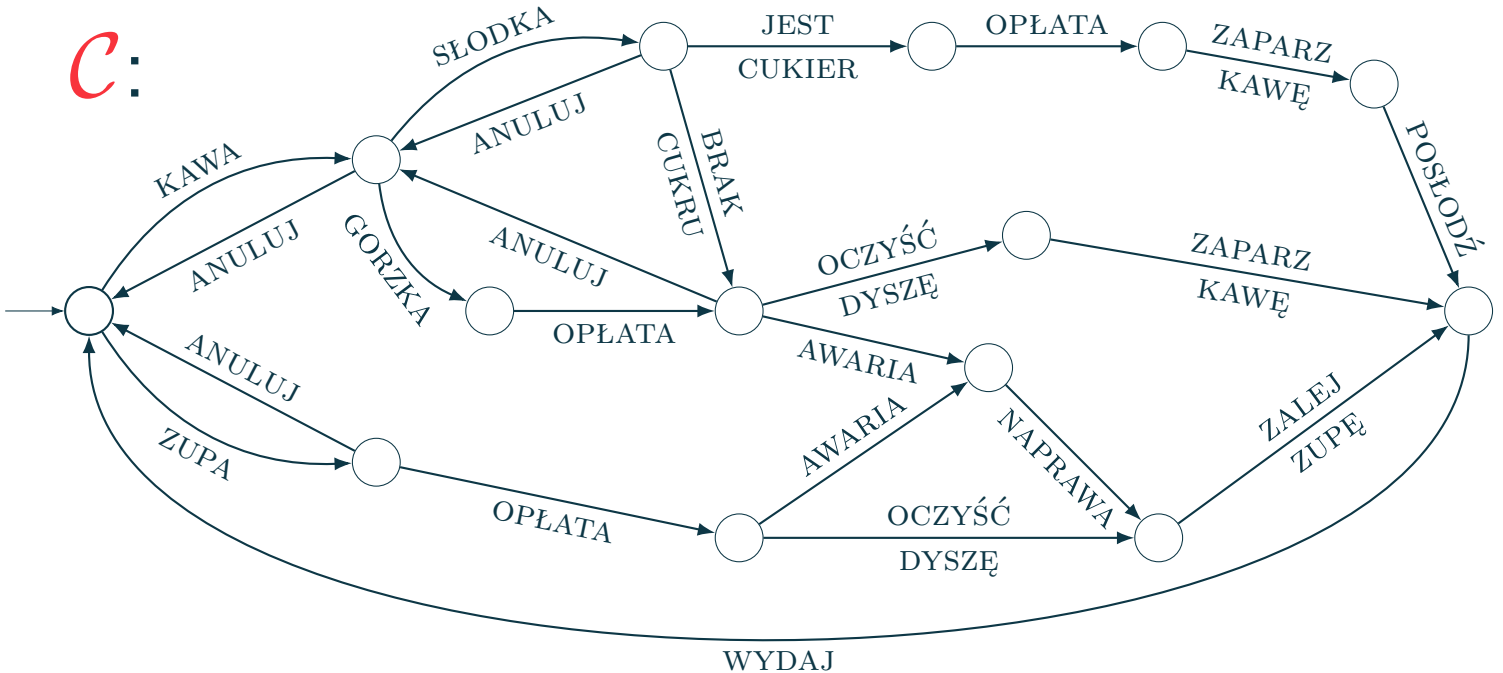
Fakt: $\neg$[**Hipoteza Goldbacha**] $\iff$ $[\,]\mathcal{P}_{\text{Gold}}[d' = 1]$

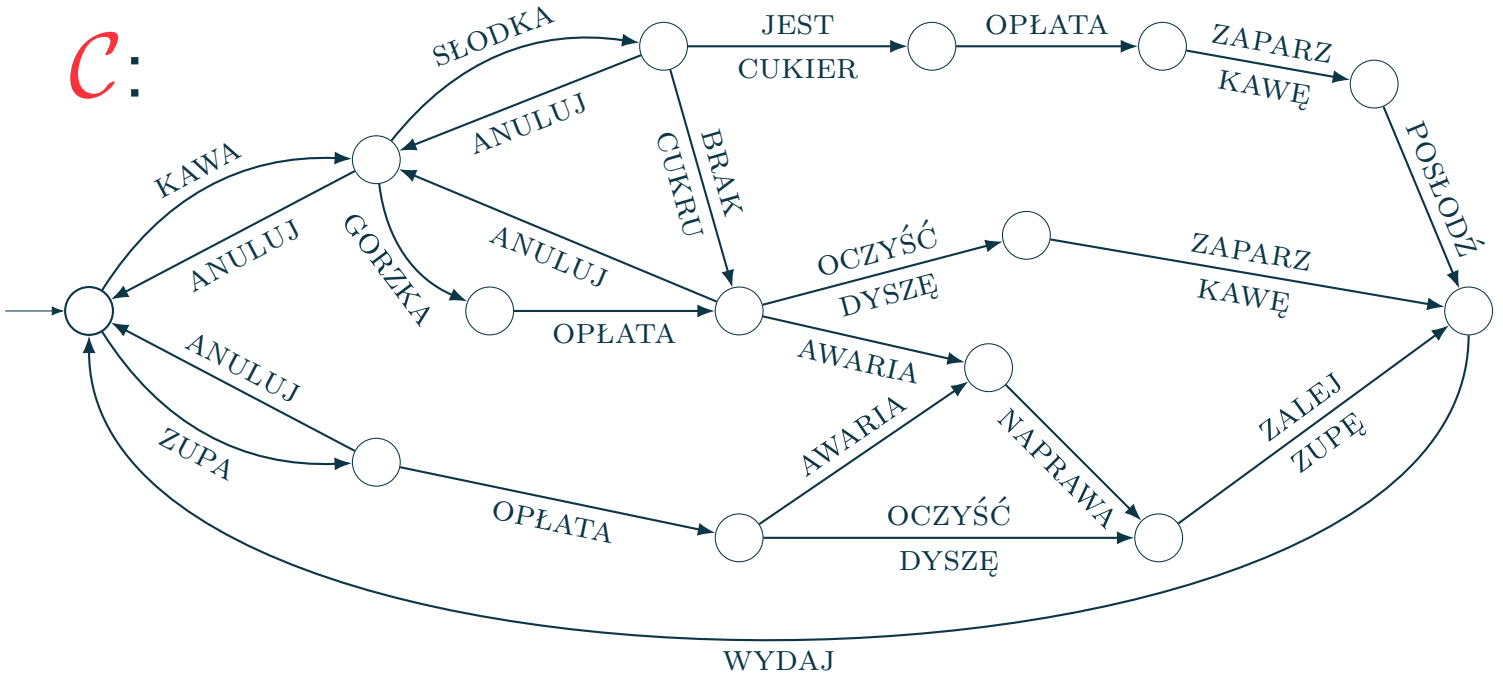
$\rightsquigarrow$ ***wystarczy*** pokazać, że $\neg[\,]\mathcal{P}_{\text{Gold}}[d' = 1]$...

[a nawet gorzej, bo program może szukać dowodów]

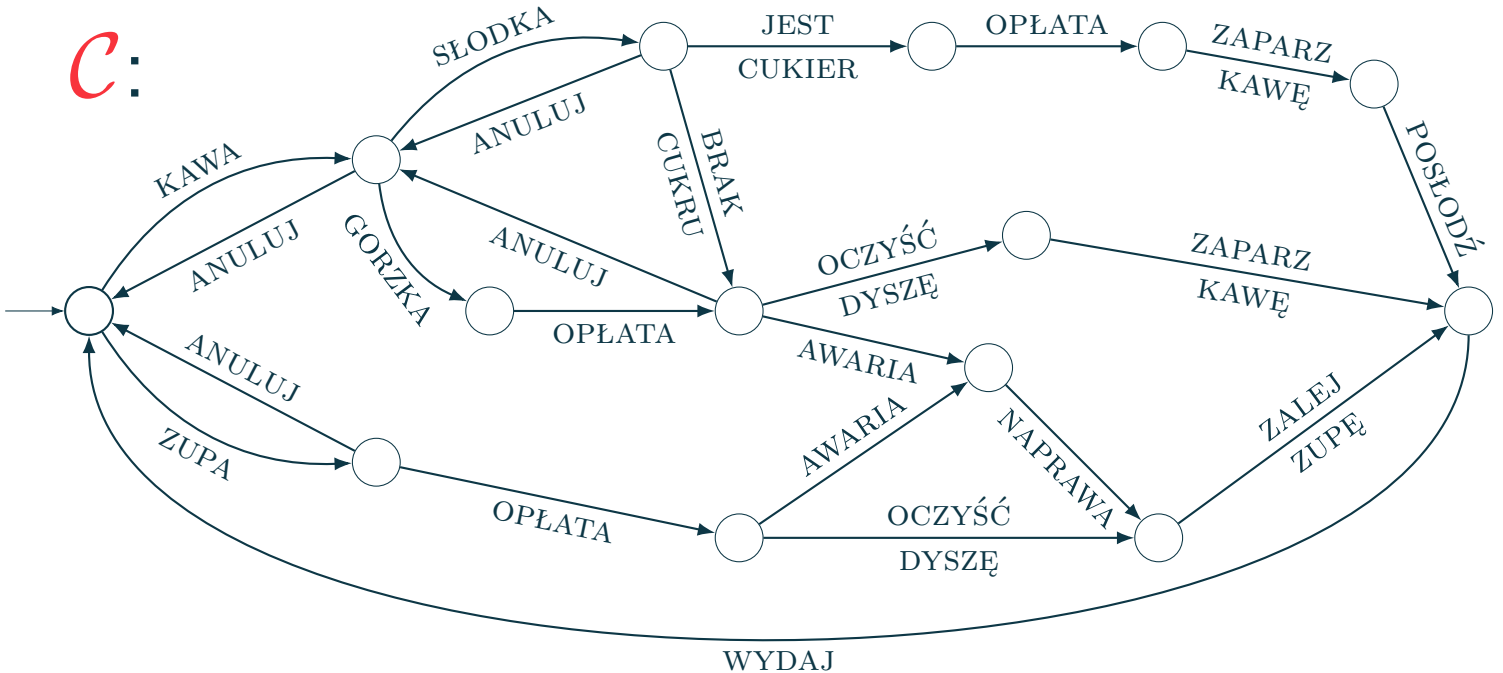
„Wszystkiemu winne są liczby.”

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


C:
SŁODKA
JEST
CUKIER
OPŁATA
ZAPARZ
KAWĘ
POSŁODŹ
KAWA
ANULUJ
ANULUJ
BRAK
CUKRU
ANULUJ
GORZKA
OPŁATA
OCZYŚĆ
DYSZĘ
ZAPARZ
KAWĘ
AWARIA
ANULUJ
ZUPA
OPŁATA
AWARIA
NAPRAWA
OCZYŚĆ
DYSZĘ
ZALEJ
ZUPĘ
WYDAJ

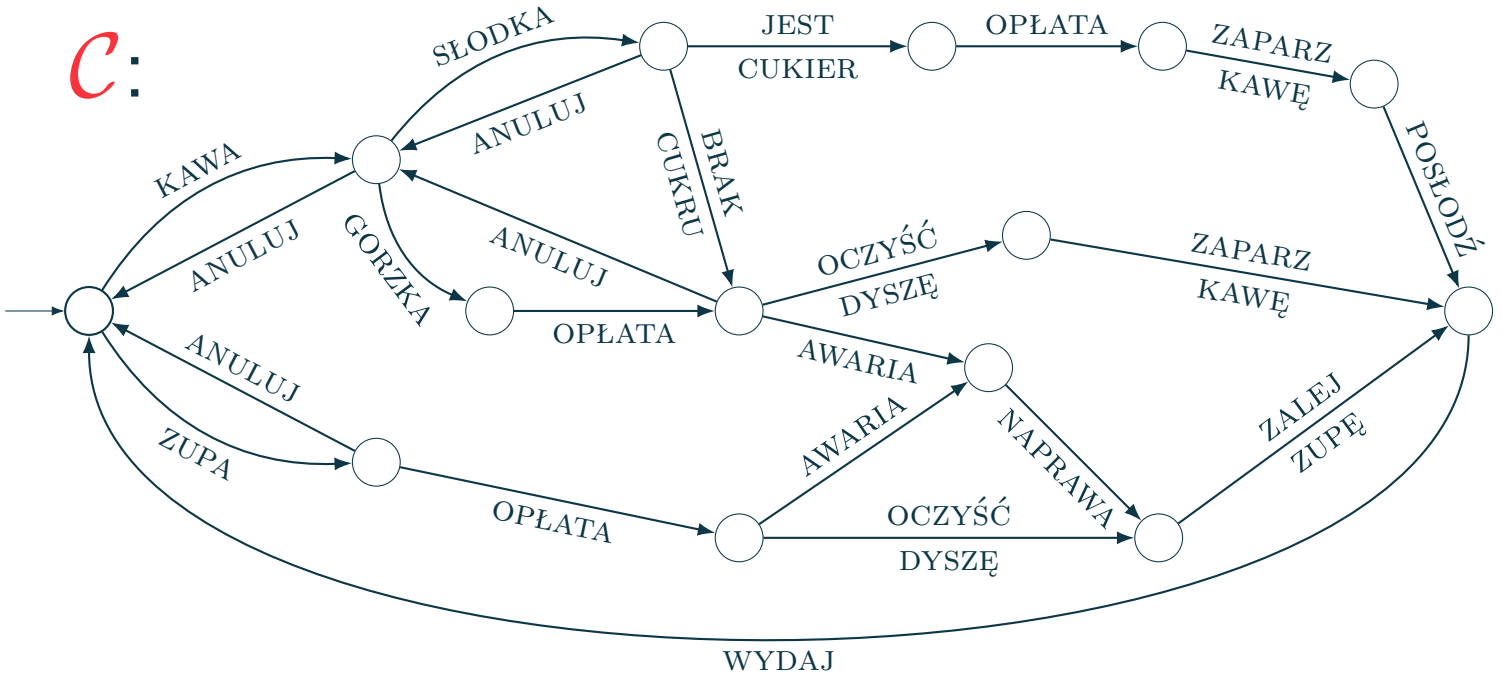


Zbiór akcji: $\mathbf{A} = \{\text{KAWA}, \text{ZUPA}, \text{OPŁATA}, \ldots\}$



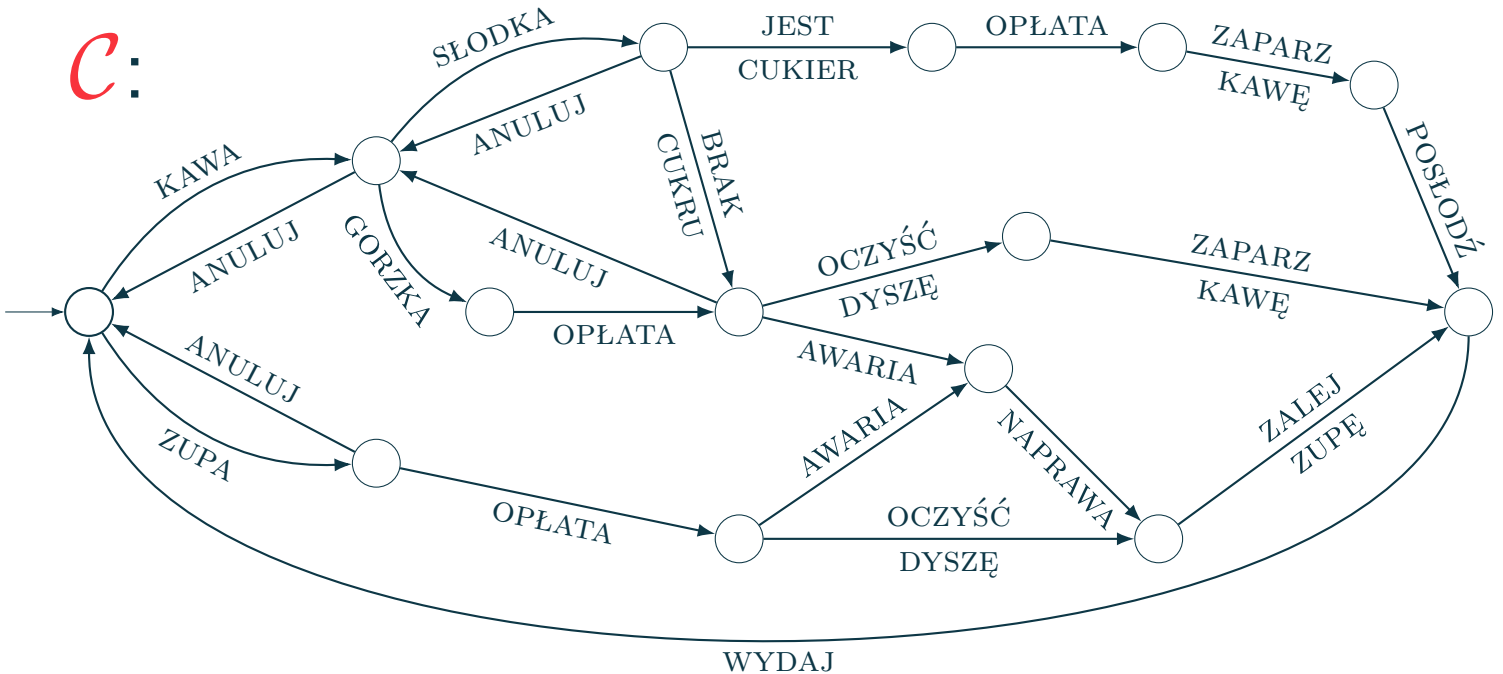
Zbiór akcji: $\mathbf{A} = \{\text{KAWA}, \text{ZUPA}, \text{OPŁATA}, \ldots\}$

Możliwe przebiegi: $\mathbf{A}^* \supseteq [\![\mathcal{C}]\!]$



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

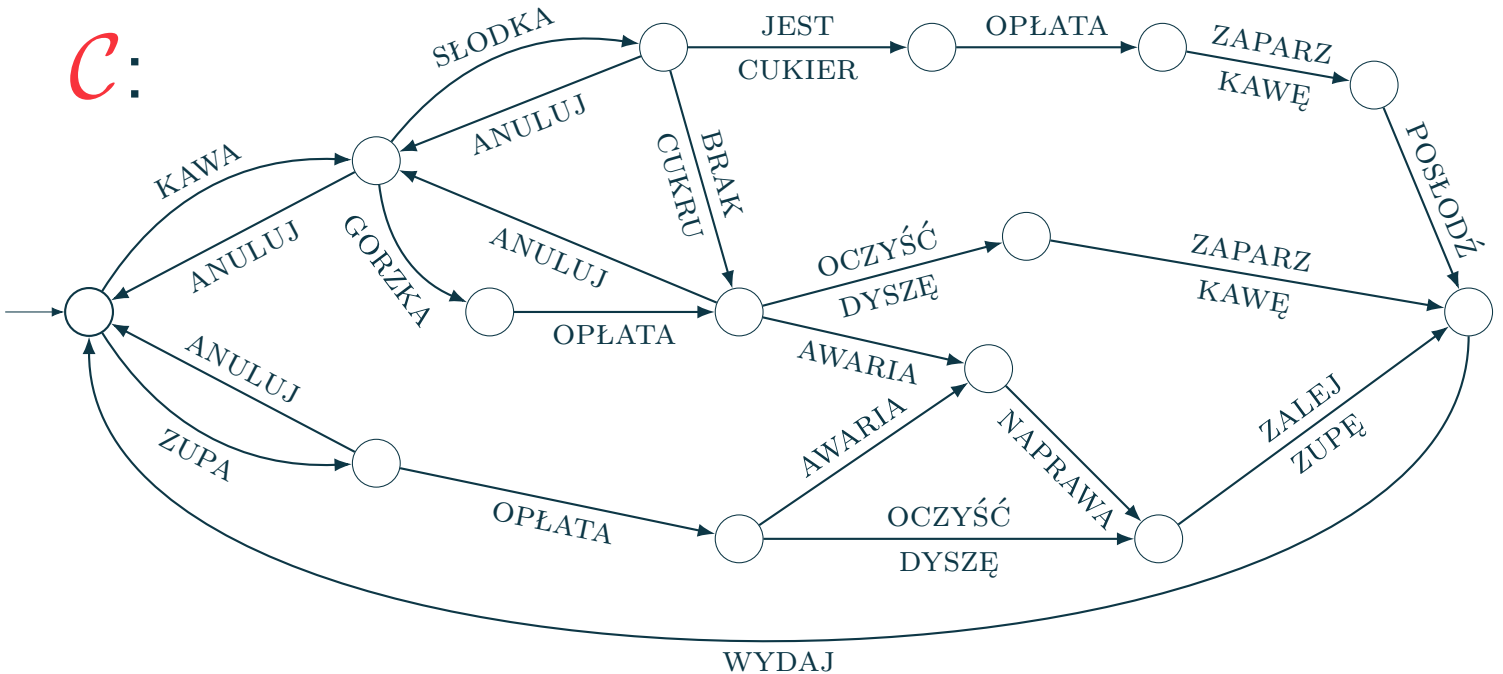
Możliwe przebiegi: $\mathbf{A}^* \supseteq [\![\mathcal{C}]\!] \ni \langle \text{ZUPA}, \text{OPŁATA}, \text{AWARIA}, \text{NAPRAWA} \rangle$



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

Możliwe przebiegi: $\mathbf{A}^* \supseteq [\![\mathcal{C}]\!] \ni \langle \text{ZUPA}, \text{OPŁATA}, \text{AWARIA}, \text{NAPRAWA} \rangle$

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

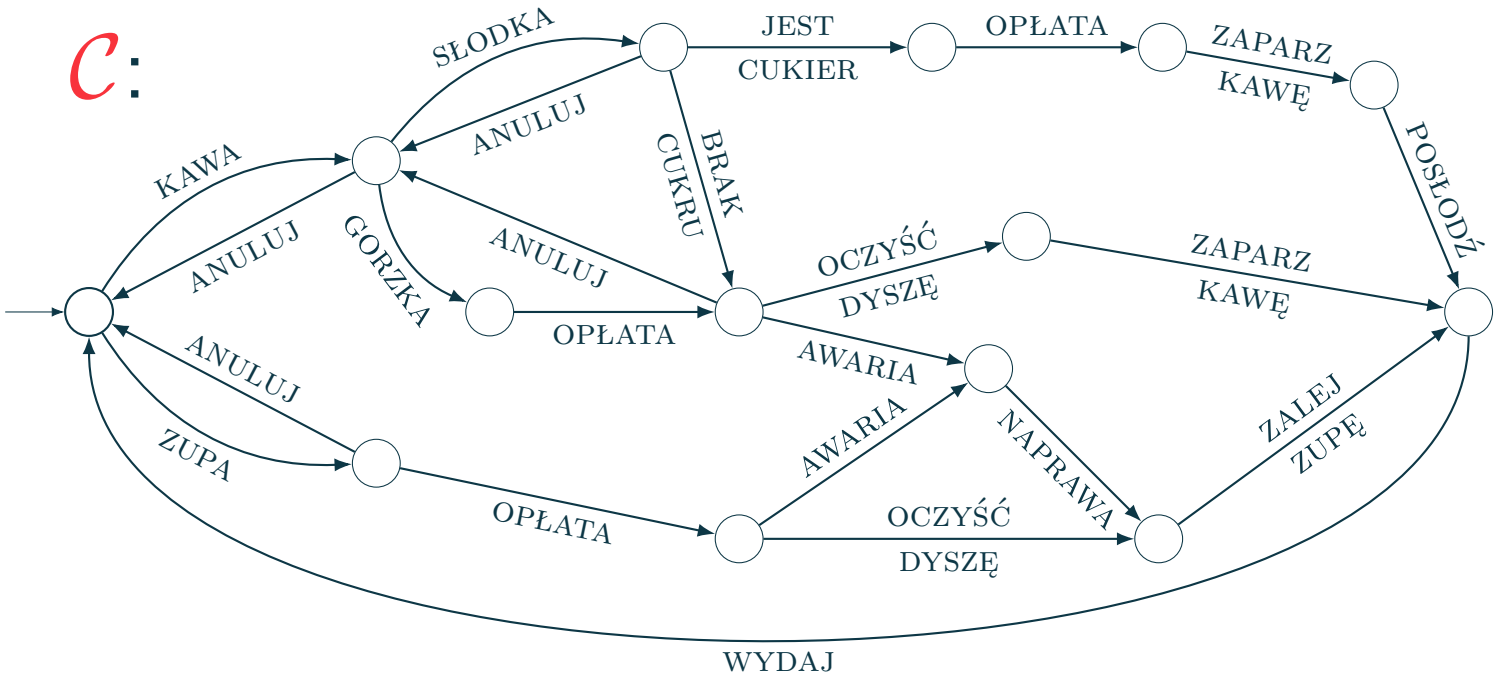


Zbiór akcji: $\mathbf{A} = \{\text{KAWA}, \text{ZUPA}, \text{OPŁATA}, \ldots\}$

Możliwe przebiegi: $\mathbf{A}^* \supseteq [\![\mathcal{C}]\!] \ni \langle \text{ZUPA}, \text{OPŁATA}, \text{AWARIA}, \text{NAPRAWA} \rangle$

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

Poprawne przebiegi: $\mathbf{A}^* \supseteq [\![\mathcal{S}]\!]$

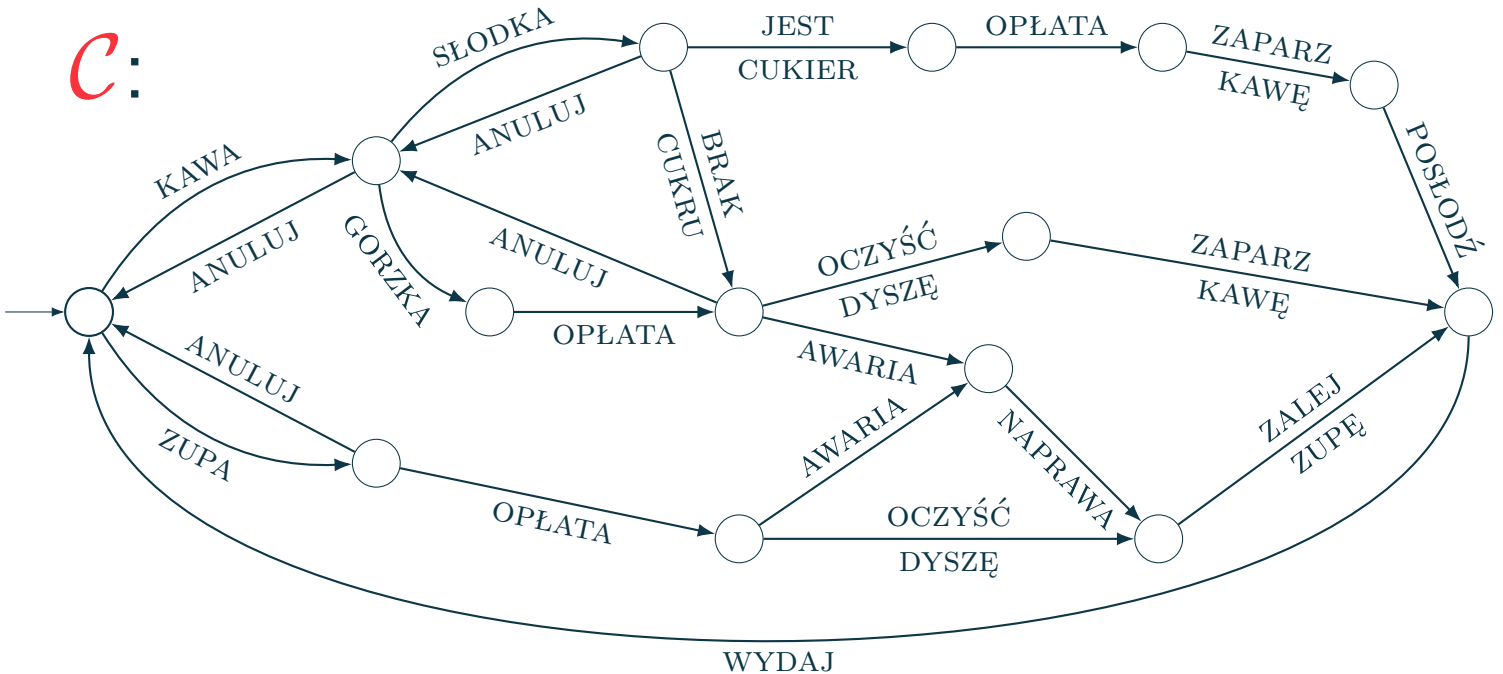


Zbiór akcji: $\mathbf{A} = \{\text{KAWA}, \text{ZUPA}, \text{OPŁATA}, \ldots\}$

Możliwe przebiegi: $\mathbf{A}^* \supseteq [\![\mathcal{C}]\!] \ni \langle \text{ZUPA}, \text{OPŁATA}, \text{AWARIA}, \text{NAPRAWA} \rangle$

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

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



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

Możliwe przebiegi: $\mathbf{A}^* \supseteq [\![\mathcal{C}]\!] \ni \langle \text{ZUPA}, \text{OPŁATA}, \text{AWARIA}, \text{NAPRAWA} \rangle$

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

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

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: φ: $\forall t.\ \textsc{wydaj}(t) \Rightarrow \exists t' < t.\ \textsc{opłata}(t')$

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

---

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

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

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

---

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

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

---

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

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

---

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

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

---

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: φ: $\forall t.\ \text{WYDAJ}(t) \Rightarrow \exists t' < t.\ \text{OPŁATA}(t')$

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

---

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: φ: $\forall t.\ \mathrm{WYDAJ}(t) \Rightarrow \exists t' < t.\ \mathrm{OPŁATA}(t')$

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

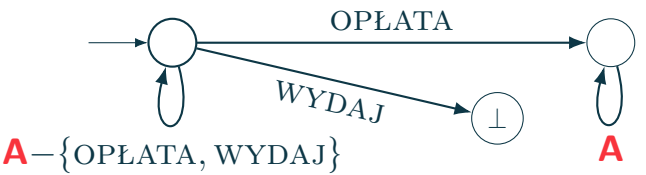
---

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

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

---

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



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

---

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

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

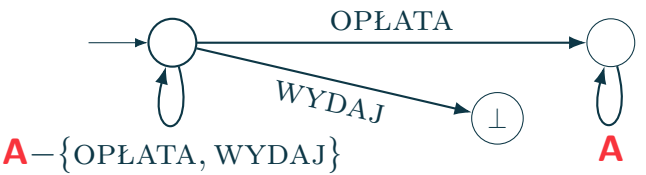
---

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: $\mathcal{V}$:



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

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

---

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

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

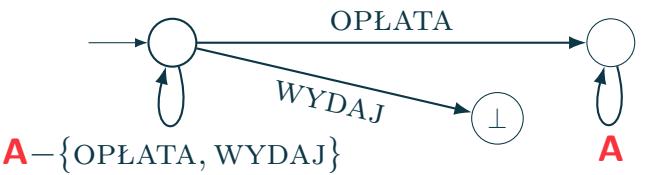
---

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: $\mathcal{V}$:



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

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

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

---

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

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

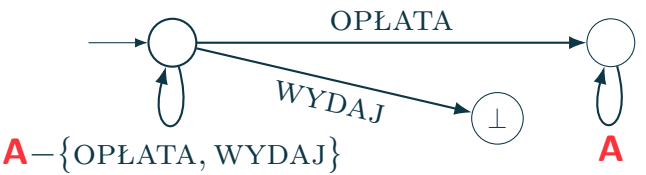
---

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: $\mathcal{V}$:



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

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

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

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

---

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

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

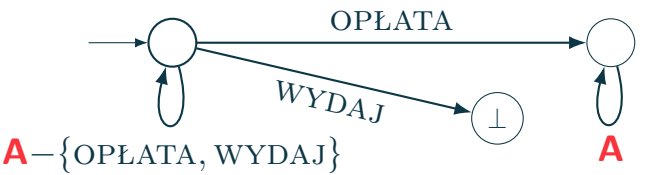
---

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: $\mathcal{V}$:



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

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: φ: $\forall t.\ \text{WYDAJ}(t) \Rightarrow \exists t' < t.\ \text{OPŁATA}(t')$

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

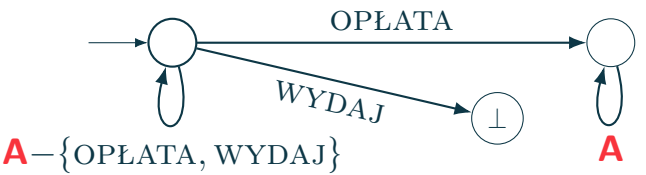
---

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: φ: $\forall t.\ \text{WYDAJ}(t) \Rightarrow \exists t' < t.\ \text{OPŁATA}(t')$

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

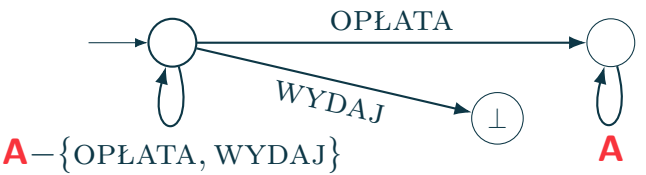
---

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])

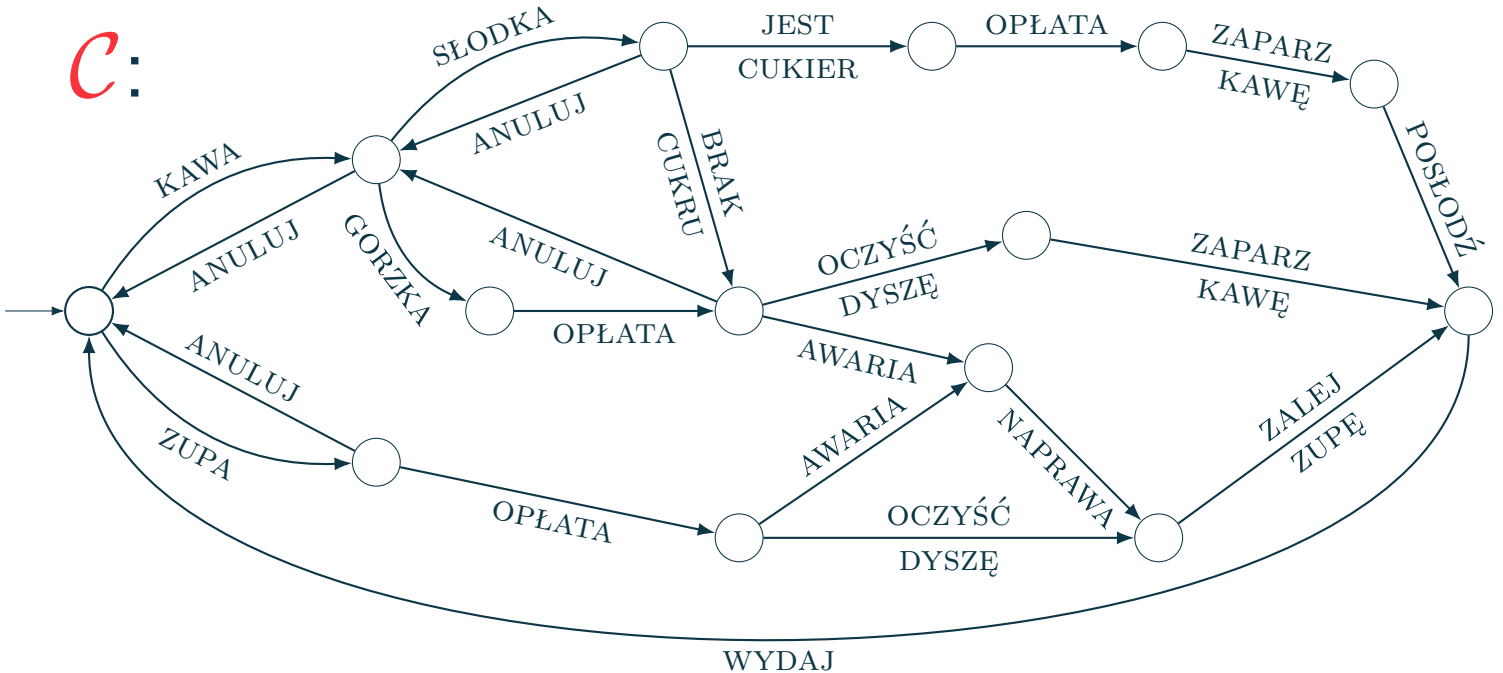
Istnieją obliczalne translacje pomiędzy **1.**, **2.** i **3.**.

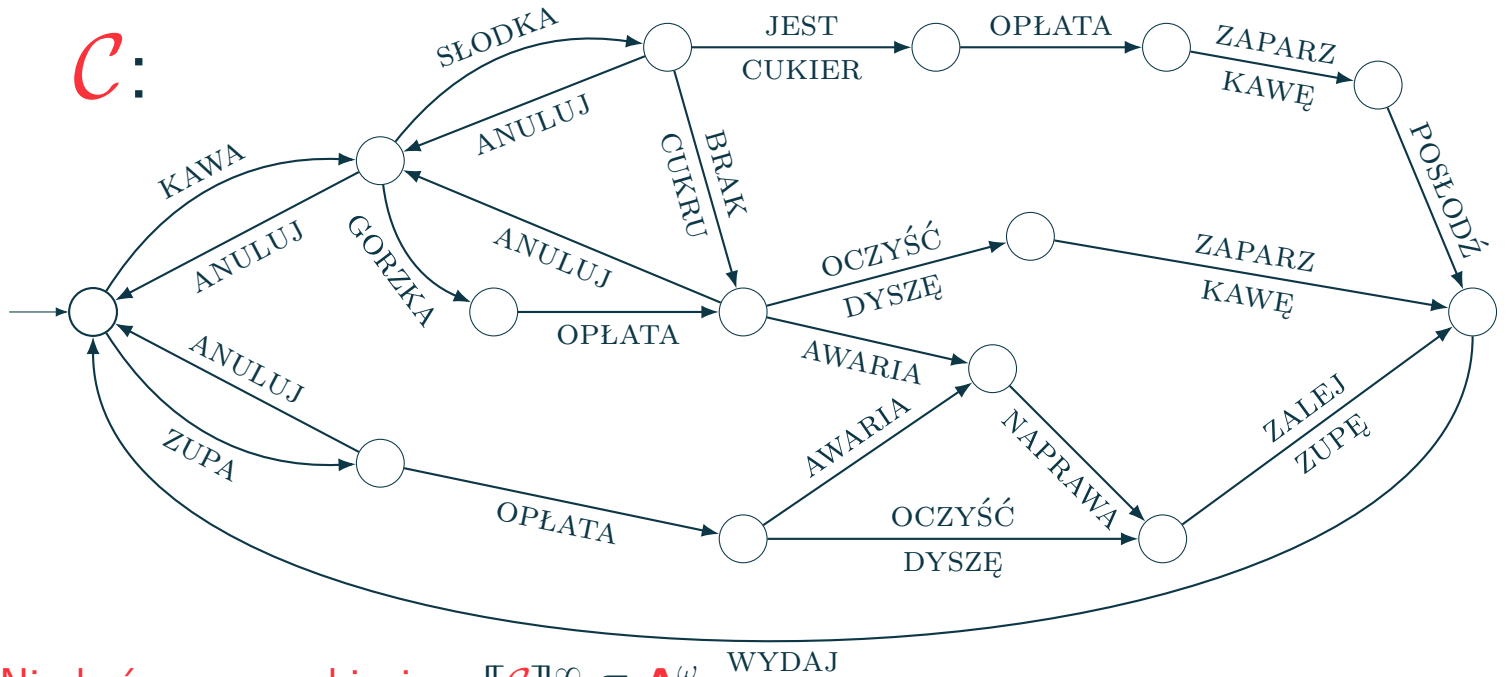
Wniosek: By sprawdzić czy $[\![\mathcal{C}]\!] \subseteq [\![\mathcal{S}]\!]$ wystarczy

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

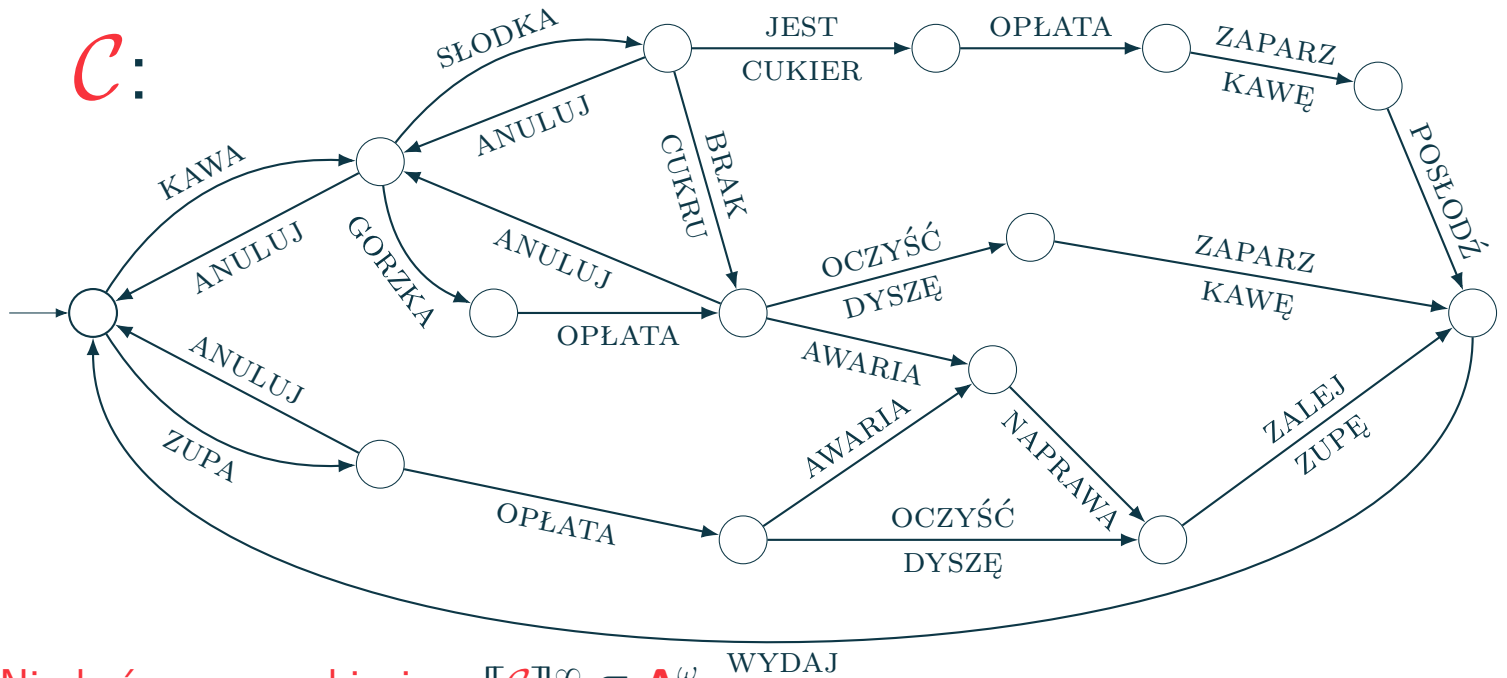
2. Sprawdzić czy: $\mathcal{C} \times \mathcal{V} \longrightarrow^* (\_, \bot)$

A gdyby tego było mało...


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

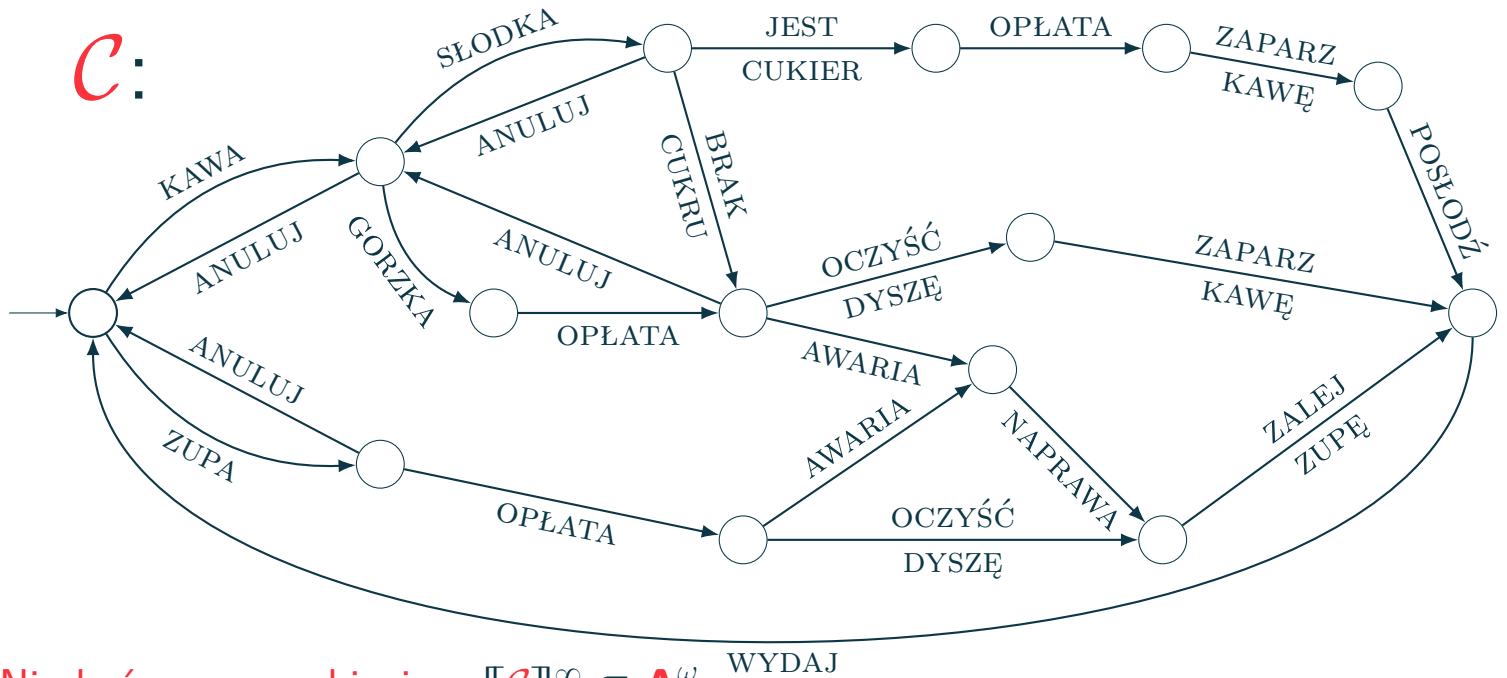


Nieskończone przebiegi: $[\![\mathcal{C}]\!]^\infty \subseteq \mathbf{A}^\omega$



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

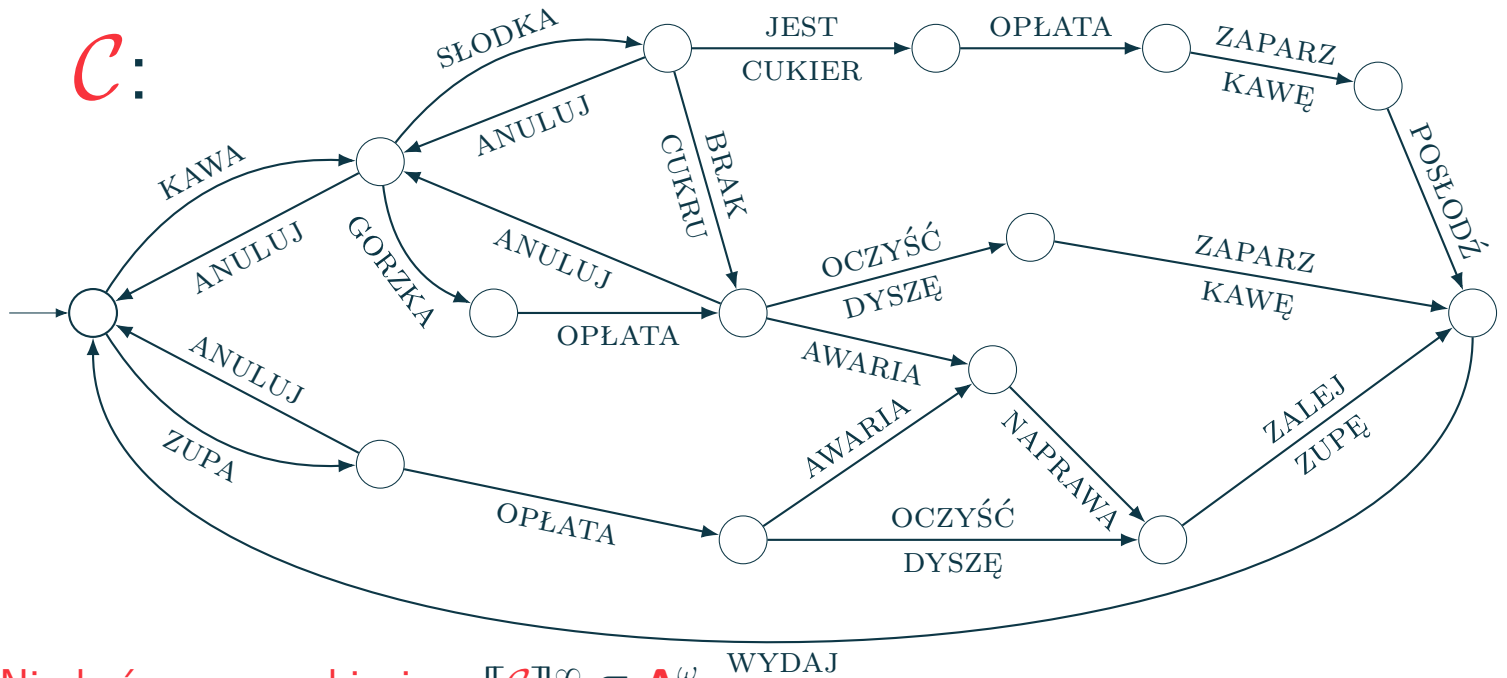
Specyfikacja: „każde obliczenie bez ANULUJ, nieskończenie wiele razy WYDAJE"



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

Specyfikacja: „każde obliczenie bez ANULUJ, nieskończenie wiele razy WYDAJE"

1. Formuła MSO: $\big(\forall t.\ \neg \textsc{anuluj}(t)\big) \Rightarrow \forall t.\ \exists t' > t.\ \textsc{wydaj}(t')$

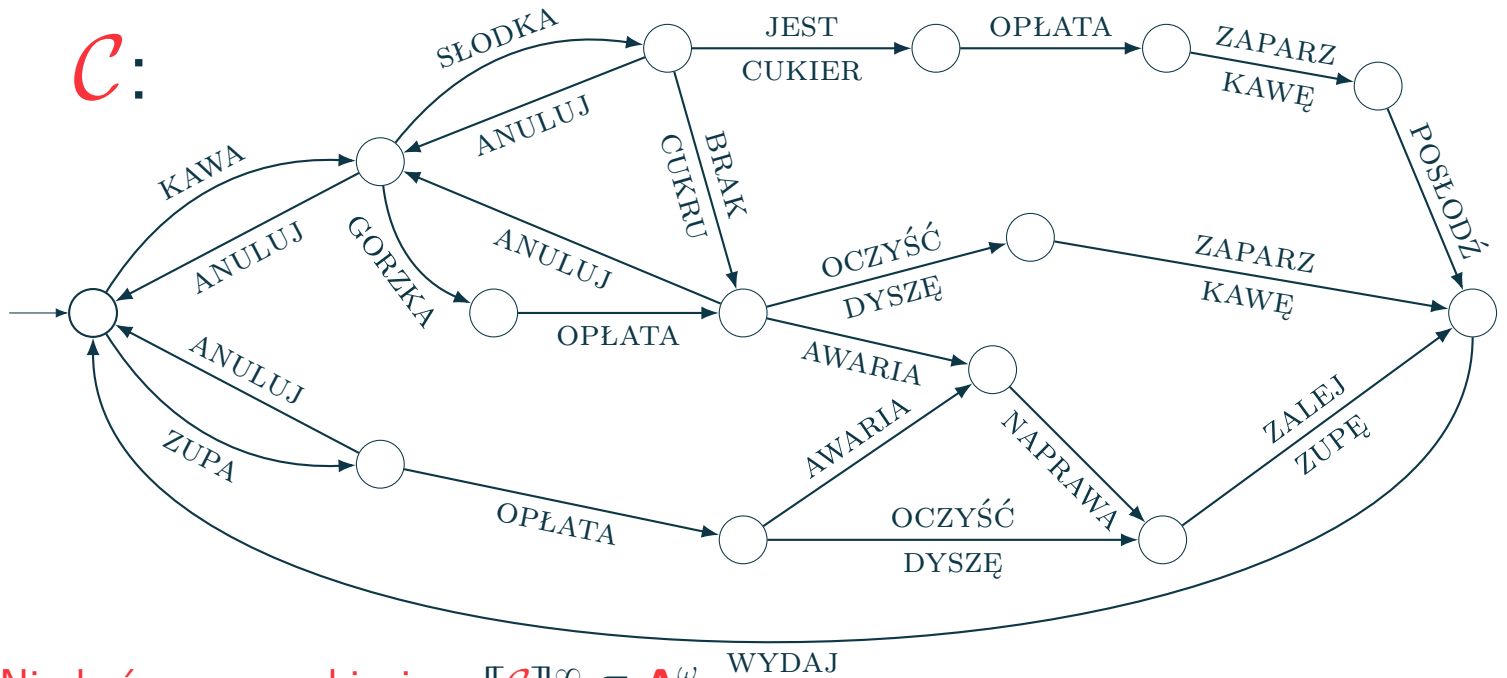


Nieskończone przebiegi: $[\![\mathcal{C}]\!]^\infty \subseteq \mathbf{A}^\omega$

Specyfikacja: „każde obliczenie bez ANULUJ, nieskończenie wiele razy WYDAJE"

1. Formuła MSO: $\big(\forall t.\ \neg \textsc{anuluj}(t)\big) \Rightarrow \forall t.\ \exists t' > t.\ \textsc{wydaj}(t')$

2. Wyrażenie regularne: $\mathbf{A}^* \cdot \textsc{anuluj} \cdot \mathbf{A}^\infty + \big(\mathbf{A}^* \cdot \textsc{wydaj}\big)^\infty$



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

Specyfikacja: „każde obliczenie bez ANULUJ, nieskończenie wiele razy WYDAJE"

1. Formuła MSO: $\big(\forall t.\ \neg \textsc{anuluj}(t)\big) \Rightarrow \forall t.\ \exists t' > t.\ \textsc{wydaj}(t')$

2. Wyrażenie regularne: $\mathbf{A}^* \cdot \textsc{anuluj} \cdot \mathbf{A}^\infty + \big(\mathbf{A}^* \cdot \textsc{wydaj}\big)^\infty$

3. Weryfikator: • • •

Twierdzenie (Büchi [1962])

Twierdzenie (Büchi [1962])

Istnieją obliczalne translacje pomiędzy następującymi formalizmami dla przebiegów **nieskończonych**(!):

1. formułami **MSO**,
2. wyrażeniami regularnymi,
3. automatami skończonymi.

Twierdzenie (Büchi [1962])

Istnieją obliczalne translacje pomiędzy następującymi formalizmami dla przebiegów **nieskończonych**(!):

1. formułami **MSO**,
2. wyrażeniami regularnymi,
3. automatami skończonymi.

Logiczne konsekwencje

Twierdzenie (Büchi [1962])

Istnieją obliczalne translacje pomiędzy następującymi formalizmami dla przebiegów **nieskończonych**(!):

1. formułami **MSO**,
2. wyrażeniami regularnymi,
3. automatami skończonymi.

Logiczne konsekwencje

⇝ **Rozstrzygalna** jest teoria **MSO** struktury $\langle \mathbb{N}, \leqslant \rangle$.

Twierdzenie (Büchi [1962])

Istnieją obliczalne translacje pomiędzy następującymi formalizmami dla przebiegów **nieskończonych**(!):

1. formułami **MSO**,
2. wyrażeniami regularnymi,
3. automatami skończonymi.

Logiczne konsekwencje

⇝ **Rozstrzygalna** jest teoria **MSO** struktury $\langle \mathbb{N}, \leqslant \rangle$.

Twierdzenie (Rabin [1969])

Twierdzenie (Büchi [1962])

Istnieją obliczalne translacje pomiędzy następującymi formalizmami dla przebiegów **nieskończonych**(!):

1. formułami **MSO**,
2. wyrażeniami regularnymi,
3. automatami skończonymi.

Logiczne konsekwencje

⇝ **Rozstrzygalna** jest teoria **MSO** struktury $\langle \mathbb{N}, \leqslant \rangle$.

Twierdzenie (Rabin [1969])

Analogicznie dla przebiegów **rozgałęziających**!

Twierdzenie (Büchi [1962])

Istnieją obliczalne translacje pomiędzy następującymi formalizmami dla przebiegów **nieskończonych**(!):

1. formułami **MSO**,
2. wyrażeniami regularnymi,
3. automatami skończonymi.

Logiczne konsekwencje

⇝ **Rozstrzygalna** jest teoria **MSO** struktury $\langle \mathbb{N}, \leqslant \rangle$.

Twierdzenie (Rabin [1969])

Analogicznie dla przebiegów **rozgałęziających**!

⇝ **Rozstrzygalna** jest teoria **MSO** pełnego drzewa binarnego.

Twierdzenie (Büchi [1962])

Istnieją obliczalne translacje pomiędzy następującymi formalizmami dla przebiegów **nieskończonych**(!):

1. formułami **MSO**,
2. wyrażeniami regularnymi,
3. automatami skończonymi.

Logiczne konsekwencje

⇝ **Rozstrzygalna** jest teoria **MSO** struktury $\langle \mathbb{N}, \leqslant \rangle$.

Twierdzenie (Rabin [1969])

Analogicznie dla przebiegów **rozgałęziających**!

⇝ **Rozstrzygalna** jest teoria **MSO** pełnego drzewa binarnego.

"The mother of all decidability results"

Summary

Summary

1. Komputery są omylne.

Summary

1. Komputery są omylne.

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

Summary

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).

Summary

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.

Summary

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).

Summary

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.

Summary

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.

Summary

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.

Summary

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.