

The following document contains pages 3-49 of the original undergraduate theis written by Michalina Nikonowicz and Kazimierz Wojciechowski, Computer Science students in the faculty of Mathematics and Information Science at Warsaw University of Technology

## **Streszczenie**

Wysoko wydajne obliczanie słów synchronizujących dla automatów skończonych

W niniejszej pracy dyplomowej zostaje przedstawiony wydajny rozproszony system obliczeniowy sprawdzający hipotezę Černýego. Obliczenia potwierdzają prawdziwość hipotezy dla automatów binarnych o liczbie stanów do 10 włącznie.

W celu sprawdzenia hipotezy system generuje nieizomorficzne automaty unarne, wstępnie eliminuje wygenerowane automaty, a następnie generuje automaty binarne i sprawdza prawdziwość hipotezy badając długość słowa synchronizującego dla wygenerowanych automatów które są synchronizowalne. System podsumowuje zebrane wyniki z węzłów obliczeniowych oraz wyświetla stan obliczeń w czasie rzeczywistym, niektóre automaty o długim słowie synchronizującym oraz ewentualne kontrprzykłady do hipotezy. Ponadto, system umożliwia interaktywną wizualizację znalezionych automatów lub ich grafów potęgowych w środowisku trójwymiarowym.

**Słowa kluczowe:** Hipoteza Černýego, obliczenia naukowe, automat synchronizowalny, obliczenia rozproszone, .Net Core, Three.js, Material Design, SignalR



## Abstract

High-performance computation of synchronizing words for finite automata

The following thesis presents a high-performance distributed computing system in order to validate the Černý conjecture. Our computations confirm the validity of the conjecture for binary automata of up to 10 states.

In order to verify the conjecture the system generates all nonisomorphic unary automata, preliminarily eliminates some of the generated automata, generates all possible binary automata out of the remaining unary automata and finally verifies the conjecture by checking the length of a shortest synchronizing word of every generated binary automaton that is synchronizable. The system summarizes all gathered data from computational nodes and displays the state of the computation in real-time, some of the automata with long synchronizing words and possibly automata that violate the conjecture. The system gives the ability to investigate found automata or their power graphs in an interactive 3D environment.

**Keywords:** Černý conjecture, scientific computing, synchronizable automaton, distributed computing, .Net Core, Three.js, Material Design, SignalR



Warszawa, dnia .....

#### Oświadczenie

Oświadczam, że moją część pracy inżynierskiej (zgodnie z podziałem zadań opisany w pkt. 5) pod tytułem „Wysoko wydajne obliczanie słów synchronizujących dla automatów skończonych”, której promotorem jest dr Michał Dębski, wykonałem/wykonałam samodzielnie, co poświadczam własnoręcznym podpisem.

.....



# Spis treści

|  |           |
|--|-----------|
| <b>Wstęp</b> . . . . .   | <b>11</b> |
| <b>1. Wstęp teoretyczny</b> . . . . .                                    | <b>12</b> |
| 1.1. Podstawowe definicje . . . . .                                      | 12        |
| 1.2. Hipoteza Ćernego . . . . .  | 13        |
| 1.3. Wykorzystane własności . . . . .                                    | 14        |
| <b>2. Specyfikacja</b> . . . . .   | <b>16</b> |
| 2.1. Opis biznesowy . . . . .  | 16        |
| 2.2. Wymagania funkcjonalne . . . . .                                    | 17        |
| 2.2.1. Use case . . . . .  | 17        |
| 2.2.2. User stories . . . . .  | 18        |
| 2.3. Wymagania niefunkcjonalne . . . . .                                 | 19        |
| <b>3. Architektura rozwiązania</b> . . . . .                             | <b>20</b> |
| 3.1. Opis użytych technologii wraz z uzasadnieniem . . . . .             | 20        |
| 3.2. Architektura całości systemu . . . . .                              | 21        |
| <b>4. Ograniczenia systemu</b> . . . . .                                 | <b>25</b> |
| 4.1. Ograniczenie liczby stanów . . . . .                                | 25        |
| <b>5. Podział prac</b> . . . . .   | <b>26</b> |
| <b>6. Wysoko wydajne algorytmy użyte w rozwiązaniu</b> . . . . .         | <b>27</b> |
| 6.1. Algorytm znajdowania najkrótszego słowa synchronizującego . . . . . | 27        |
| 6.1.1. Idea algorytmu . . . . .  | 27        |
| 6.1.2. Cel optymalizacji . . . . .                                       | 27        |
| 6.1.3. Użyty wzorzec projektowy . . . . .                                | 27        |
| 6.1.4. Reprezentacja wewnętrzna stanów . . . . .                         | 27        |
| 6.1.5. Opis znaczących usprawnień algorytmu . . . . .                    | 27        |
| 6.1.6. Wyniki wydajnościowe . . . . .                                    | 28        |
| 6.2. Algorytm generowania nieizomorficznych automatów unarnych . . . . . | 29        |
| 6.2.1. Idea algorytmu . . . . .  | 29        |

|   |           |
|---|-----------|
| 6.2.2. Porównanie z algorytmem naiwnym . . . . .  | 29        |
| 6.3. Algorytm przeszukania całej przestrzeni automatów binarnych w poszukiwaniu najdłuższych słów synchronizujących . . . . . | 29        |
| <b>7. Wymagania systemowe . . . . .</b>   | <b>31</b> |
| <b>8. Biblioteki wraz z określeniem licencji . . . . .</b>  | <b>32</b> |
| <b>9. Instrukcja instalacji . . . . .</b>   | <b>33</b> |
| 9.1. Kompilacja Klienta . . . . .   | 33        |
| 9.2. Kompilacja Serwera . . . . .   | 33        |
| 9.3. Kompilacja Prezentacji . . . . .   | 33        |
| <b>10. Instrukcja uruchomienia . . . . .</b>  | <b>34</b> |
| 10.1. Uruchomienie Klienta na platformie Windows x64 . . . . .  | 34        |
| 10.2. Uruchomienie Serwera na platformie Windows x64 . . . . .  | 34        |
| 10.3. Uruchomienie Prezentacji . . . . .  | 34        |
| 10.3.1. Windows . . . . .   | 34        |
| <b>11. Instrukcja użycia . . . . .</b>  | <b>35</b> |
| 11.1. Serwer . . . . .  | 35        |
| 11.2. Klient . . . . .  | 35        |
| 11.3. Prezentacja . . . . .   | 36        |
| 11.4. Wizualizacja . . . . .  | 39        |
| <b>12. Instrukcja utrzymania . . . . .</b>  | <b>42</b> |
| <b>13. Model danych . . . . .</b>   | <b>43</b> |
| <b>14. Raport z przeprowadzonych testów . . . . .</b>   | <b>44</b> |
| <b>15. Przeprowadzone obliczenia . . . . .</b>  | <b>46</b> |

## Wstęp

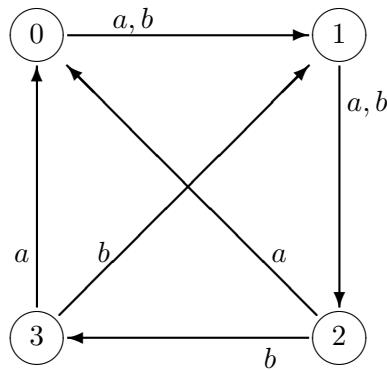
Motywacją dla tematu jest hipoteza Černego z 1964 roku mówiąca, że każdy synchronizowalny automat skończony o  $n$  stanach ma słowo synchronizujące o długości co najwyżej  $(n - 1)^2$ . Hipoteza do dziś pozostaje otwarta i jest potwierdzona tylko dla wąskich klas automatów, w tym dla automatów o nie więcej niż 12 stanach [4]. Jest to najprawdopodobniej najstarszy problem w kombinatorycznej teorii automatów skończonych którego dotąd nie udało się rozwiązać.

Niniejsza praca ma na celu niezależne potwierdzenie prawdziwości hipotezy Černego dla automatów o niewielkiej liczbie stanów. Ponadto, najciekawsze wyniki obliczeń oraz potencjalne automaty przeczące hipotezie przedstawione zostają w estetycznej formie. Projekt składa się z trzech modułów: Klienta, Serwera oraz Prezentacji.

# 1. Wstęp teoretyczny

## 1.1. Podstawowe definicje

Niech  $\mathcal{A} = (Q, \Sigma, \delta)$  będzie *skończonym automatem deterministycznym*, gdzie  $Q$  jest zbiorem stanów,  $\Sigma$  jest skończonym alfabetem wejściowym, natomiast  $\delta : Q \times \Sigma \rightarrow Q$  jest funkcją przejścia. Taki automat możemy przedstawić za pomocą grafu skierowanego (z możliwymi pętlami), którego wierzchołkami są stany, a krawędzie etykietowane alfabetem wejścia odpowiadają funkcji przejścia.



Rysunek 1.1: Automat o czterech stanach i dwuelementowym alfabetie

W poniższej pracy zajmować się będziemy głównie automatami o dwuelementowym alfabetie, które nazywać będziemy *automatami binarnymi*. Automat którego alfabet ma jeden symbol nazwiemy *automatem unarnym*.

Funkcję przejścia  $\delta : Q \times \Sigma \rightarrow Q$  możemy w naturalny sposób rozszerzyć na ciąg symboli. Niech  $\hat{\delta} : Q \times \Sigma^* \rightarrow Q$  będzie funkcją zadaną poprzez warunki:

- $\hat{\delta}(q, \varepsilon) = q$ ,
- $\hat{\delta}(q, (x_1, \dots, x_n)) = \hat{\delta}(\delta(q, x_1), (x_2, \dots, x_n))$ ,

gdzie  $q \in Q$  oraz  $x_i \in \Sigma$  dla  $i = 1, \dots, n$ .

Łatwo zauważyc, że z powyższych warunków wynika, że  $\hat{\delta}(q, (x)) = \delta(q, x)$ , jest to zatem dobrze zdefiniowane rozszerzenie funkcji  $\delta$ .

## 1.2. HIPOTEZA ĆERNEGO

Słowem synchronizującym automatu  $\mathcal{A} = (Q, \Sigma, \delta)$  nazywamy taki skończony ciąg  $\bar{x} \in \Sigma^*$  dla którego spełniony jest warunek  $\exists s \in Q \quad \forall q \in Q \quad \hat{\delta}(q, \bar{x}) = s$ . Najkrótszym słowem synchronizującym automatu  $\mathcal{A}$  nazywamy dowolne słowo synchronizujące  $\bar{x} \in \Sigma^*$  automatu  $\mathcal{A}$  o długości  $k$  takie, że nie istnieje słowo synchronizujące tegoż automatu o długości krótszej od  $k$ .

Automat, dla którego istnieje słowo synchronizujące będziemy nazywać *automatem synchronizowalnym*. W przeciwnym wypadku będziemy taki automat nazywać *automatem niesynchronizowalnym*.

### 1.2. Hipoteza Ćernego

**Hipoteza 1.1.** Dla każdego skończonego synchronizowalnego automatu deterministycznego jeśli liczba jego stanów wynosi  $n$  to najkrótsze słowo synchronizujące tego automatu ma długość co najwyżej równą  $(n - 1)^2$ .

Ján Černy publikując swoją hipotezę w 1964 roku podał również klasę automatów których najkrótsze słowo synchronizujące ma długość równą ograniczeniu z hipotezy, czyli  $(n - 1)^2$ . Niech

$$\mathcal{C}_n = (\{0, 1, \dots, n - 1\}, \{a, b\}, \delta) \text{ dla } n \geq 2, \text{ gdzie}$$

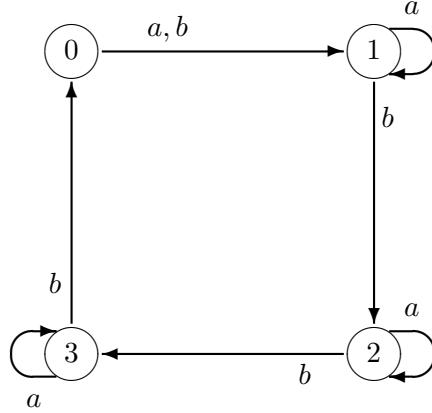
$$\delta(n - 1, b) = 0,$$

$$\delta(i, b) = i + 1 \quad \text{dla } i = 0, \dots, n - 2,$$

$$\delta(0, a) = 1,$$

$$\delta(i, a) = i \quad \text{dla } i = 1, \dots, n - 1.$$

Automaty  $\mathcal{C}_n$  nazywa się *automatami Černego*.



Rysunek 1.2: Automat Černego  $\mathcal{C}_4$

Nietrudno można zauważyć, że najkrótsze słowo synchronizujące automatu  $\mathcal{C}_4$  przedstawionego na rysunku 1.2 to *abbababbba*. Słowo to faktycznie ma długość  $9 = (n - 1)^2$ .

### 1.3. Wykorzystane własności

Poniżej przedstawimy kilka własności, definicji i lematów, które okazują się być pomocne podczas badania synchronizowalności automatów.

**Definicja 1.2.** *Grafem potęgowym* automatu  $\mathcal{A} = (Q, \Sigma, \delta)$  nazywamy graf, którego wierzchołkami są podzbiory zbioru stanów  $Q$ , a krawędź etykietowana symbolem  $x \in \Sigma$  z wierzchołka  $\{q_1, \dots, q_k\}$  prowadzi do wierzchołka  $\{\delta(x, q_1), \dots, \delta(x, q_k)\}$ .

**Lemat 1.3.** Automat jest synchronizowalny wtedy i tylko wtedy, gdy w jego grafie potęgowym istnieje ścieżka łącząca wierzchołek reprezentujący wszystkie stany z jednym z wierzchołków reprezentujących pojedynczy stan.

Dodatkowo jeśli automat jest synchronizowalny to długość jego najkrótszego słowa synchronizującego jest równa długości najkrótszej ścieżki opisanej powyżej.

Zanim przedstawimy kolejny lemat warto przypomnieć, że automat  $\mathcal{A} = (Q, \Sigma, \delta)$  nazywamy silnie spójnym jeśli dla każdych dwóch stanów  $p$  i  $q$  istnieje takie słowo  $\bar{x}$ , że  $\hat{\delta}(p, \bar{x}) = q$ .

**Lemat 1.4.** Jeśli hipoteza Ćernego jest spełniona dla wszystkich automatów o liczbie stanów ze zbioru  $\{1, \dots, n - 1\}$  to jest spełniona również dla automatów o  $n$  stanach, które nie są silnie spójne.

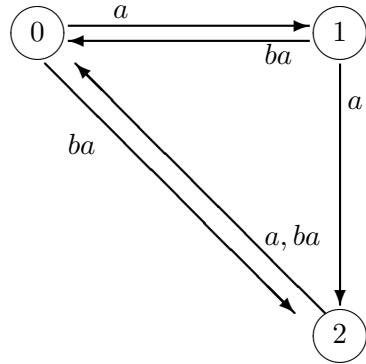
Korzystając z powyższego lematu, łatwo możemy zauważyc, że wystarczy udowodnić hipotezę dla automatów silnie spójnych, aby następnie udowodnić ją dla wszystkich automatów.

W celu ograniczenia liczby automatów, dla których będziemy poszukiwać długości najkrótszego słowa synchronizującego wykorzystamy własności bezpośrednio związane z badaniem hipotezy Ćernego na automatach binarnych. Zanim jednak przedstawimy te własności warto zapoznać się z pewnym sposobem redukcji automatu binarnego.

Niech  $\mathcal{A} = (Q, \{a, b\}, \delta)$  będzie automatem binarnym o  $n$  stanach. Zdefiniujemy zredukowany automat  $\mathcal{A}_C = (\{q \in Q : \exists p \in Q \quad \delta(p, a) = q\}, \{a, ba\}, \delta_C)$ . Funkcję przejścia definiujemy warunkami:  $\delta_C(q, a) = \delta(q, a)$  oraz  $\delta_C(q, ba) = \delta(\delta(q, b), a)$ . Jest to zatem usunięcie z automatu  $\mathcal{A}$  stanów do których nie możemy wejść za pomocą symbolu  $a$ .

W analogiczny sposób możemy zredukować automat ze względu na drugi symbol alfabetu, taką redukcję oznaczamy następująco:  $A_C(\{ab, b\})$ .

### 1.3. WYKORZYSTANE WŁASNOŚCI



Rysunek 1.3: Automat  $\mathcal{A}_C$  automatu z rysunku 1.1

**Lemat 1.5.** Niech  $\mathcal{A}$  będzie binarnym automatem o  $n$  stanach. Jeśli automat  $\mathcal{A}_C$  ma również  $n$  stanów to  $\mathcal{A}$  jest niesynchronizowalny lub  $\mathcal{A}_C(\{ab, b\})$  ma mniej niż  $n$  stanów.

**Lemat 1.6.** Jeśli binarny automat  $\mathcal{A}$  ma  $n$  stanów i jest silnie spójny, a automat  $\mathcal{A}_C$  ma mniej niż  $\frac{n}{2}$  stanów to  $\mathcal{A}_C(\{ab, b\})$  ma więcej niż  $\frac{n}{2}$  stanów.

**Lemat 1.7.** Niech  $\mathcal{A}$  automat binarny, jeśli automat  $\mathcal{A}_C$  jest synchronizowalny i długość jego najkrótszego słowa synchronizującego wynosi  $k$  to  $\mathcal{A}$  jest również synchronizowalny, a długość jego najkrótszego słowa synchronizującego wynosi co najwyżej  $2k + 1$ .

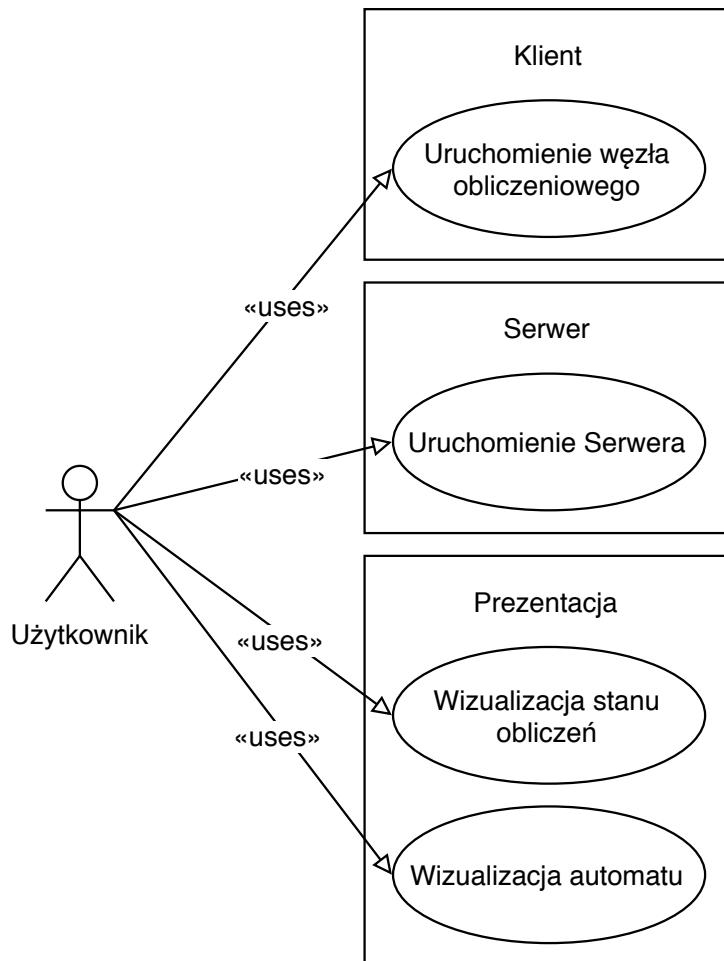
## **2. Specyfikacja**

### **2.1. Opis biznesowy**

Niniejszy system służy do wydajnego sprawdzania synchronizowalności i wyznaczania długości najkrótszych słów synchronizujących w obszernych zbiorach deterministycznych automatów skończonych. Do zadań systemu należy wygenerowanie wszystkich parami nieizomorficznych automatów skończonych z zadanego zbioru (unarnych automatów o zadanej liczbie stanów i nieredukujących się do automatu o krótkim słowie synchronizującym), wysoko wydajne wyznaczenie długości najkrótszego słowa resetującego każdego z nich, następnie gromadzenie i podsumowanie wyników z węzłów obliczeniowych i wizualizacja wyników obliczeń.

## 2.2. Wymagania funkcjonalne

### 2.2.1. Use case



Rysunek 2.1: Diagram przypadków użycia

| ID   | Aktor      | Moduł  | Nazwa                             | Opis  | Odpowiedź systemu   |
|------|------------|--------|-----------------------------------|---|---|
| User | Użytkownik | Klient | Uruchomienie węzła obliczeniowego | Uruchomienie i konfiguracja Klienta czyli węzła obliczeniowego. Podanie danych Serwera w celu nawiązania łączności. | Informacja o nawiązaniu połączenia lub nieprawidłowej konfiguracji. |

Tablica 2.1: Opis poszczególnych przypadków użycia - węzeł obliczeniowy

| ID   | Aktor      | Moduł  | Nazwa                       | Opis  | Odpowiedź systemu   |
|------|------------|--------|-----------------------------|---|---|
| User | Użytkownik | Serwer | Uruchomienie serwera        | Konfiguracja serwera do prawnej komunikacji z węzłami obliczeń.                           | Komunikat o udanej konfiguracji lub o błędzie uruchomienia. Ponadto w trakcie trwania obliczeń zapisywany jest plik z dotychczasową historią obliczeń.  |
|      |            |        | Wizualizacja stanu obliczeń | Wyświetlenie aktualnego stanu obliczeń.   | W panelu obliczeń wyświetlane są w czasie rzeczywistym najciekawsze automaty dotąd napotkane w trakcie obliczeń w tym potencjalnie przeczące hipotezie. |
|      |            |        | Wizualizacja automatu       | Interaktywna prezentacja automatu lub jego grafu potęgowego wybranego przez użytkownika.. | Animowany, interaktywny graf z zaznaczoną ścieżką wzdłuż słowa synchronizującego.   |

Tablica 2.2: Opis poszczególnych przypadków użycia

### 2.2.2. User stories

1. Jako użytkownik uruchamiam skonfigurowany Serwer w celu rozpoczęcia obliczeń dla danej wielkości automatów.
2. Jako użytkownik uruchamiam odpowiednio skonfigurowany Serwer w celu wznowienia przerwanych obliczeń.
3. Jako użytkownik uruchamiam skonfigurowanego Klienta w celu połączenia z serwerem i następnie wspomożenia trwających obliczeń.
4. Jako użytkownik uruchamiam odpowiednio skonfigurowaną prezentację aby odczytać całkowity stan obliczeń w celu oszacowania wydajności systemu oraz całkowitego czasu obliczeń.

### 2.3. WYMAGANIA NIEFUNKCJONALNE

5. Jako użytkownik uruchamiam w trakcie obliczeń prezentację pokazującą listę najciekawszych dotąd znalezionych automatów podczas obliczeń w tym potencjalnie automatów przewiązanych hipotezie.
6. Jako użytkownik uruchamiam wizualizację jednego z wyróżnionych podczas obliczeń automatów w celu analizy oraz przestrzennego wyobrażenia słowa synchronizującego.

### 2.3. Wymagania niefunkcjonalne

| Obszar wymagań                 | Nr wymagania | Opis   |
|--------------------------------|--------------|--|
| Użyteczność<br>(Usability)     | 1            | Dowolny węzeł, choćby naj wolniejszy, nie spowalnia obliczeń lecz je wspomaga swoją mocą obliczeniową, choćby najmniejszą.   |
|                                | 2            | Intuicyjna i estetyczna prezentacja stanu obliczeń dostosowana do monitora biurowego o rozdzielczości full HD.   |
|                                | 3            | Przejrzystość instrukcji uruchamiania modułów.   |
|                                | 4            | Węzeł obliczeniowy gotowy do wniesienia wkładu w obliczenia bez zbędnej instalacji.  |
| Niezawodność<br>(Reliability)  | 5            | Awaria jednego z węzłów obliczeniowych nie spowoduje nieodwracalnych braków w obliczeniach ani nie wpłynie na poprawność obliczeń.   |
| Wydajność<br>(Performance)     | 6            | System sprawnie weryfikuje hipotezę Černýego dla liczby stanów mniejszej bądź równej 7   |
| Utrzymanie<br>(Supportability) | 7            | Serwer zapisuje na bieżąco stan obliczeń w celu ich odtworzenia na wypadek (planowanej bądź nie) przerwy w działaniu. Obliczenia w ten sposób mogą być bezpiecznie wznowione poprzez ponowne uruchomienie Serwera z dodatkowym parametrem. |

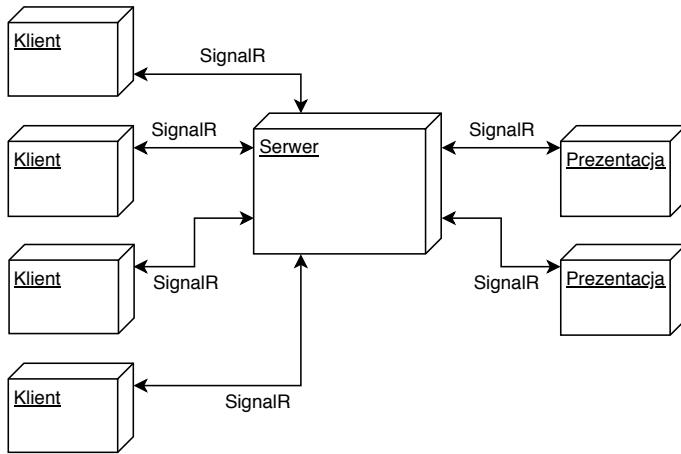
Tablica 2.3: Niefunkcjonalne wymagania systemu

### 3. Architektura rozwiązania

#### 3.1. Opis użytych technologii wraz z uzasadnieniem

| Moduł       | Technologia                  | Uzasadnienie  |
|-------------|------------------------------|---|
| Klient      | .NET Core 2.1                | Technologia firmy Microsoft stworzona na bazie sprawdzonego środowiska .NET Framework. Mamy doświadczenie w programowaniu na tej wszechstronną platformę.   |
|             | ASP.NET Core<br>SignalR      | Technologia tworzona przez Microsoft służąca do komunikacji międzyprocesowej RPC. Technologia korzysta przedwszystkim z nowoczesnego standardu gniazd WebSockets. Ta technologia przeznaczona jest do częstej wymiany danych w obie strony. W razie braku systemowego wsparcia wykorzystywane są kolejno inne protokoły o podobnych właściwościach. |
| Serwer      | ASP.NET Core                 | Nowoczesny i odświeżony framework służący do m.in. tworzenia wszechstronnych aplikacji webowych.  |
|             | ASP.NET Core<br>SignalR      | Uzasadnienie powyżej.   |
| Prezentacja | ASP.NET Core<br>SignalR      | Uzasadnienie powyżej.   |
|             | WinForms                     | Technologia w której mamy doświadczenie, w której się komfortowo tworzy interfejs użytkownika.  |
|             | Biblioteki Java-Script i CSS | Intuicyjne i łatwe w obsłudze biblioteki umożliwiające animowaną (jQuery), trójwymiarową wizualizację (Three.js) oraz estetyczny interfejs użytkownika (Materialize)  |

### 3.2. Architektura całości systemu



Rysunek 3.1: Poglądowy schemat architektury z oznaczeniem technologii wymiany danych pomiędzy modułami. Wymiana danych następuje między komponentami odpowiedzialnymi za komunikację w ramach każdego modułu.

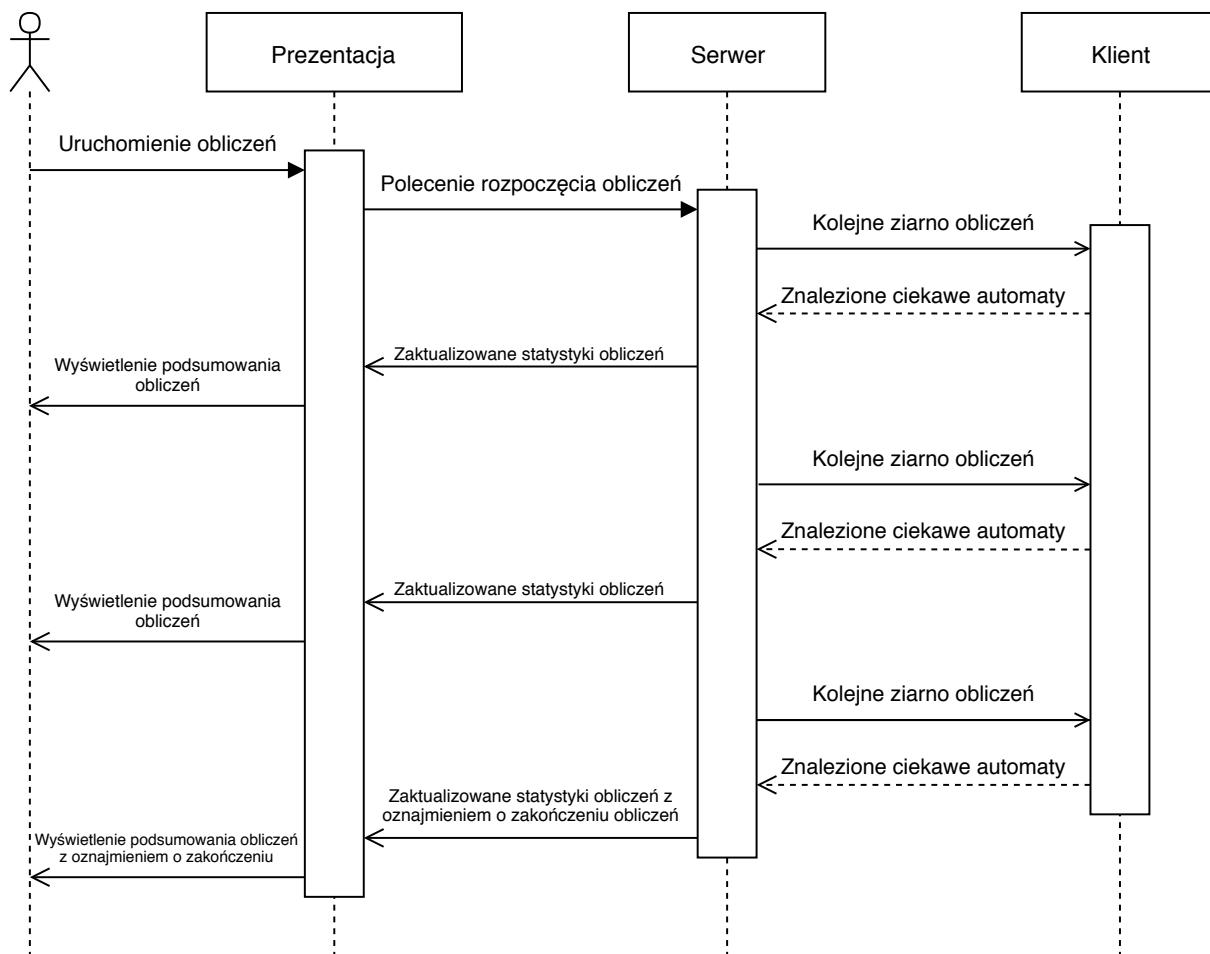
Aplikacja Klienta jest programem sprawdzającym długość słowa synchronizującego podzbioru automatów binarnych o danej liczbie stanów (stałej w ramach jednej sesji obliczeń). Klient otrzymuje poprzez protokół WebSocket (jeśli nie jest wspierany, biblioteka SignalR będzie próbowała połączyć się poprzez alternatywne protokoły) ziarno obliczeń poprzez zdalne wywoływanie procedury u Klienta - RPC. Korzystając z deterministycznego algorytmu generowania nieizomorficznych automatów unarnych Klient, na podstawie ziarna obliczeń, w skomplikowany sposób wygeneruje specjalny podzbior automatów binarnych, potencjalnie zawierający automaty o dłuższym słowie synchronizującym. Długość słowa synchronizującego automatów w tym podzbiorze jest sprawdzana wprost. Następnie Klient zwraca wygenerowane automaty (wywołuje zdalnie procedurę u Serwera) o najdłuższym słowie synchronizującym. Nie są to wszystkie najdłuższe automaty jakie determinuje ziarno - stąd nazwa *najciekawsze* zamiast *najdłuższe*, jednakże określenie też dotyczy automatów o długości dłuższej niż podana w hipotezie - inaczej podsumowanie nie byłoby wartościowe. Zdecydowaliśmy, że ziarna, automaty oraz inne parametry wywołań zdalnych procedur w ramach całego systemu kodowane są w formacie binarnym.

Dzięki błyskawicznemu generowaniu automatów unarnych Klient może je generować dynamicznie dla  $n \leq 12$  zamiast wczytywać z plików o niemałych rozmiarach. Powyżej  $n = 13$  warto jednak rozważyć przechowywanie dużego zbioru wygenerowanych zaraz po ich generowaniu, ale nienajnowszy system wspiera obliczenia dla liczby stanów nie większej niż 12.

Komunikacja między Serwerem a Prezentacją (systemem odpowiedzialnym za kontrolę obliczeń)

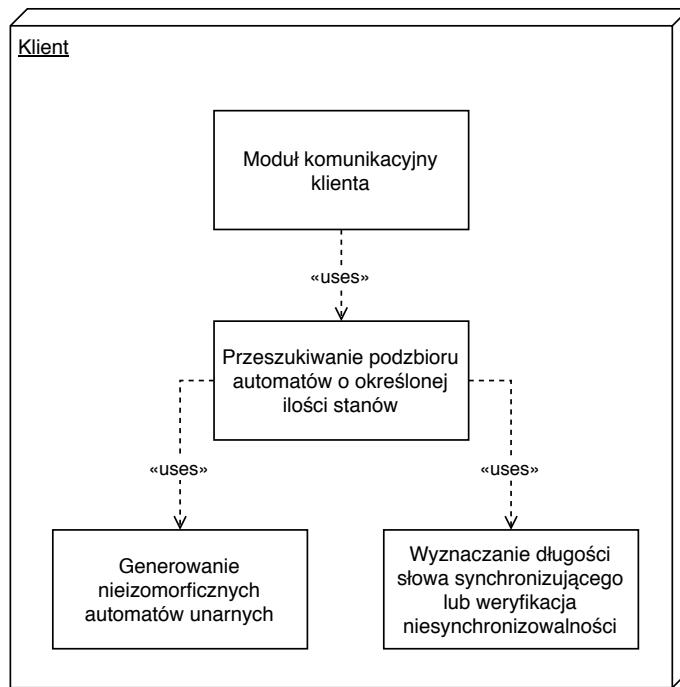
### 3. ARCHITEKTURA ROZWIĄZANIA

czeń oraz wizualizację najciekawszych automatów) odbywa się poprzez ten sam ciąg protokołów co komunikacja między Serwerem a klientami. Statystyki są aktualizowane w czasie rzeczywistym.

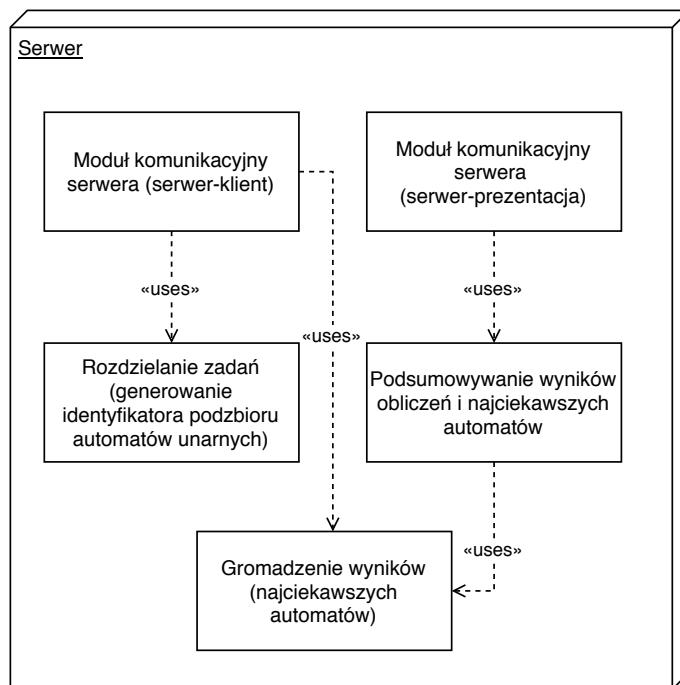


Rysunek 3.2: Diagram Sekwencji. Widoczna jest duża liczba asynchronicznych wywołań.

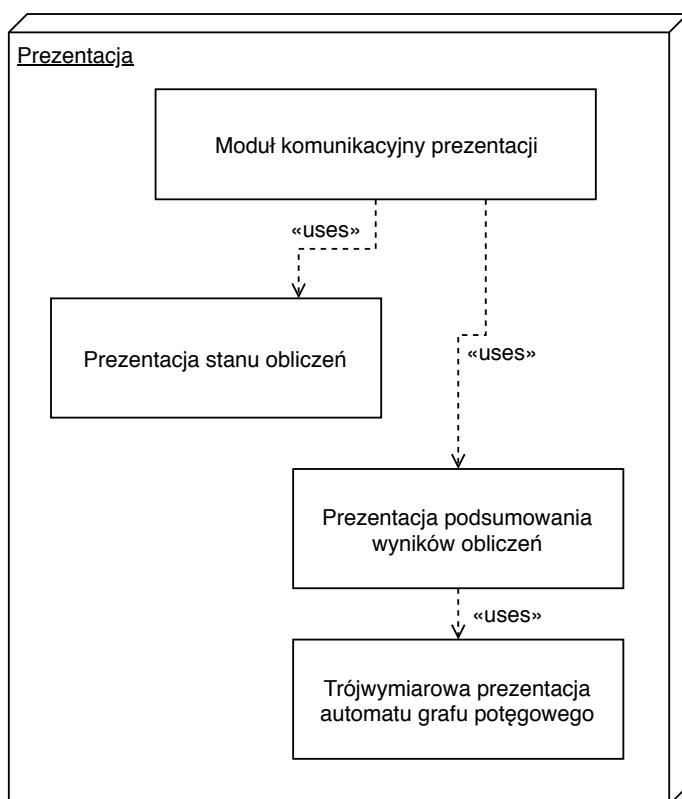
### 3.2. ARCHITEKTURA CAŁOŚCI SYSTEMU



Rysunek 3.3: Schemat architektury Klienta



Rysunek 3.4: Schemat architektury Serwera



Rysunek 3.5: Schemat architektury Prezentacji

## **4. Ograniczenia systemu**

### **4.1. Ograniczenie liczby stanów**

System gwarantuje sprawdzenie hipotezy Ćernego dla liczby stanów od 3 do 12.

## **5. Podział prac**

Michalina Nikonorowicz napisała algorytm filtrujący automaty unarne pod kątem długości słowa synchronizującego oraz generujący z nieredukowalnego automatu unarnego wszystkie automaty binarne z pozostałymi funkcjami przejść. Michalina także napisała statystyczne wnioski w module prezentacji.

Kazimierz Wojciechowski napisał algorytm znajdowania długości słowa synchronizującego automatu binarnego, algorytm generowania nieizomorficznych automatów unarnych o zadanej liczbie stanów, komunikację między modułami, zarządzanie wynikami zadania w module Klienta, moduł Serwera oraz trójwymiarową wizualizację w module Prezentacji.

## 6. Wysoko wydajne algorytmy użyte w rozwiązaniu

### 6.1. Algorytm znajdowania najkrótszego słowa synchronizującego

#### 6.1.1. Idea algorytmu

Algorytm polega na przeszukiwaniu wszerz dynamicznie generowanego grafu potęgowego zgodnie z lematem 1.3. Algorytm rozpoczyna przeszukiwanie od wierzchołka oznaczającego pełen zbiór stanów  $Q$ . Algorytm kończy działanie gdy zostanie odnaleziony wierzchołek oznaczający singleton lub gdy zostaną odnalezione wszystkie osiągalne wierzchołki.

#### 6.1.2. Cel optymalizacji

Celem optymalizacji jest maksymalizacja wydajności obliczania i generowania wyników dla jak największej liczby losowych automatów binarnych niewielkiego rozmiaru rzędu 12.

#### 6.1.3. Użyty wzorzec projektowy

Został użyty wzorzec projektowy iteratora z użyciem innowacyjnych udogodnień syntaktycznych `yield return` jakie oferuje język C#.

#### 6.1.4. Reprezentacja wewnętrzna stanów

Każdy stan jest reprezentowany w wewnętrznej implementacji algorytmu przez liczbę, gdzie w reprezentacji binarnej każdy bit oznacza przebywanie w danym stanie w oryginalnym automacie. Czyli 10100 oznacza stan reprezentujący jednocześnie przebywanie w stanach 3 oraz 5. Wierzchołek początkowy jest reprezentowany liczbą  $2^n - 1$ .

#### 6.1.5. Opis znaczących usprawnień algorytmu

1. rezygnacja z istotnie spowalniających udogodnień jakie oferuje LINQ
2. powszechnie użycie stałych zamiast zmiennych

3. własna implementacja kolejki BFS ze sprawdzaniem bieżącej długości ścieżki w czasie stałym i jej aktualizowaniem w czasie stałym
4. tablica służąca do błyskawicznego podglądu czy dany stan już został odkryty po kolejnej iteracji (tablica ma  $2^n$  elementów) praktycznie nigdy nie jest czyszczona, gdyż zamiast informacji binarnej statusu odkrycia przechowywana jest zmienna pętli czyli unikalny identyfikator aktualnego automatu. Obliczenie statusu odkrycia sprowadza się do porównania elementu tablicy z aktualną zmienną pętli
5. wstępne przetworzenie tablicy przejść czyli zastąpienie liczby  $i$  przez  $2^i$  w celu błyskawicznego zastosowania funkcji przejścia operacją bitową
6. sprawdzenie czy stan reprezentuje przebywanie w jednym wierzchołku oznacza, że w reprezentacji binarnej liczby zapalony jest dokładnie jeden bit, jest to sprawdzane w czasie stałym za pomocą jednej operacji arytmetycznej i jednej operacji bitowej
7. przejście do kolejnego stanu pierwotnie polegało na iterowaniu po kolejnych bitach liczby i aplikowaniu operacji bitowych w celu osiągnięcia liczby następnych stanów
8. przejście do kolejnego stanu zostało następnie znacznie usprawnione aby zawczasu przygotować tablicę przejść nie dla pojedynczego bitu lecz dla całej grupy bitów (np. czterech) i na dodatek dla obu przejść  $a$  i  $b$  w ramach jednej operacji, czyli przykładowo zamiast wykonywać 12 operacji dla każdego z dwóch przejść (w sumie 24) to wykonywane są tylko 3 operacje dla obu przejść na raz. Jest to najbardziej kosztowny obliczeniowo krok algorytmu

#### 6.1.6. Wyniki wydajnościowe

Algorytm BFS, który jest zoptymalizowany zgodnie z powyższymi punktami osiąga wydajność o kilka-kilkanaście procent mniejszą od wydajności algorytmu napisanego w C++. Wysoko zoptymalizowana implementacja algorytmu na karcie graficznej a nawet jednoczesne obliczenia CPU i GPU są mniej wydajne niż uruchomienie algorytmu na CPU na wielu wątkach. Ostatnia optymalizacja sama w sobie zwiększa wydajność o 170%. Każda z wymienionych optymalizacji miała istotny wkład we wzrost wydajności czego owocem jest skrajnie zoptymalizowany algorytm pozбавiony zjawiska *wąskiego przekroju*.

## 6.2. Algorytm generowania nieizomorficznych automatów unarnych

### 6.2.1. Idea algorytmu

Algorytm polega na:

1. wygenerowaniu możliwie małego nadzbioru nieizomorficznych automatów unarnych mając do dyspozycji wszystkie nieizomorficzne automaty o mniejszej liczbie stanów
2. dla każdego wygenerowanego automatu obliczana jest w krótkim czasie (wielomianowym) unikalna liczba (hash) odzwierciedlająca jego strukturę wewnętrzną niezależnie od permutacji wierzchołków
3. odrzuceniu nadmiarowych automatów które są opisane tą samą liczbą (są izomorficzne), aby na każdą unikalną liczbę przypadał dokładnie jeden automat

Dla izomorficznych automatów wyliczone unikalne liczby są identyczne. Ponadto, unikalna liczba jest różna dla nieizomorficznych automatów (własność potwierdzona dla  $n \leq 14$ ). Własność wynika z faktu, że liczba różnych unikalnych liczb pokrywa się z liczbą nieizomorficznych automatów (OEIS A001372). Gdyby dwa nieizomorficzne automaty miały tę samą unikalną liczbę to liczność przeciwdziedziny funkcji haszującej byłaby mniejsza niż liczba nieizomorficznych automatów. Wspomniana równoliczność implikuje że znalezione automaty o tej samej liczbie są izomorficzne. Stąd algorytm sprawdzania izomorficzności automatów jest zaimplementowany tylko i wyłącznie pod kątem testów jednostkowych i jest nieobecny w czasie obliczeń Klienta.

### 6.2.2. Porównanie z algorytmem naiwnym

Dla ułatwienia algorytm naiwny polegałby wyłącznie na potwierdzeniu nieizomorficzności między już gotowymi automatami. Gdyby dla każdej pary ze zbioru nieizomorficznych automatów o 12 stanach potwierdzenie nieizomorficzności było równe szybkie jak trzykrotne wykonanie pętli z pustym ciałem, to taki algorytm byłby wolniejszy od prezentowanego w niniejszej pracy.

## 6.3. Algorytm przeszukania całej przestrzeni automatów binarnych w poszukiwaniu najdłuższych słów synchronizujących

Przeszukiwanie przestrzeni automatów binarnych o  $n$  stanach w celu poszukiwania kontrprzykładu dla hipotezy Ćernego składa się z kilku kroków, zawierających się w nich zarówno algorytmy

opisane powyżej jak i algorytmy generowania wszystkich możliwych funkcji przejścia dla symbolu  $b$ , które nie zostały tu dokładnie opisane ponieważ są to proste algorytmy z powrotami generujące wszystkie możliwe ciągi długości  $n$  nad zbiorem  $\{0, \dots, n-1\}$ .

1. Wygeneruj wszystkie nieizomorficzne automaty unarne o  $n$  stanach.
2. Dla każdego automatu unarnego:
  - (a) Oblicz liczbę stanów w automacie zredukowanym  $\mathcal{A}_C$  o tablicy przejścia  $a$  jak w podanym automacie unarnym.
  - (b) Jeśli liczba ta wynosi  $n$  lub jest mniejsza niż  $\frac{n}{2}$  przejdź do kolejnego automatu unarnego.
  - (c) Wygeneruj wszystkie możliwe automaty  $\mathcal{A}_C$  o funkcji przejścia dla symbolu  $a$  jak w podanym automacie unarnym (generujemy tylko funkcję przejścia dla  $ba$  tylko dla stanów należących do  $\mathcal{A}_C$ ).
  - (d) Dla każdego automatu  $\mathcal{A}_C$ :
    - i. Oblicz długość najkrótszego słowa synchronizującego automatu  $\mathcal{A}_C$  (o ile jest synchronizowalny), niech długość ta wynosi  $k$ .
    - ii. Jeśli jest synchronizowalny i zachodzi  $2k+1 \leq (n-1)^2$  przejdź do kolejnego automatu  $\mathcal{A}_C$ .
    - iii. Wygeneruj wszystkie możliwe funkcje przejścia dla  $b$  tak aby otrzymać automaty  $\mathcal{A}$ , które zredukowałyby się do  $\mathcal{A}_C$ .
    - iv. Policz długości najkrótszych słów synchronizujących automatów  $\mathcal{A}$  i zwróć tylko te automaty, dla których długość ta przekracza  $(n-1)^2$ .

Hipoteza Ćernego jest prawdziwa dla automatów binarnych o  $n$  stanach wtedy i tylko wtedy, gdy powyższy algorytm nie zwróci żadnych automatów.

W praktyce w naszym algorytmie pozwalamy na zamianę ograniczenia z podpunktu 2(d)iv na jakąś inną liczbę (najlepiej bliską  $(n-1)^2$ ), żeby algorytm zwrócił kilka automatów o długich słowach synchronizujących, a potem sprawdzamy czy wszystkie zwrócone automaty mają długość najkrótszego słowa synchronizującego nie większą niż  $(n-1)^2$ .

## 7. Wymagania systemowe

Moduły Klienta i Serwera:

- Do kompilacji jest potrzebne środowisko .Net Core 2.1 SDK lub nowsze
- Do uruchomienia jest potrzebne środowisko .Net Core 2.1 Runtime (wersja przenośna) lub nowsze albo środowisko .Net Framework 4.6 lub nowsze (wersja skompilowana dla systemu Windows)

Moduł prezentacji (oprócz wizualizacji):

- Do kompilacji i uruchomienia jest potrzebne środowisko .Net Framework 4.6.1 lub nowsze

Wizualizacja interaktywna:

- Przeglądarka wspierająca technologię WebGL

Podsumowując:

- Skompilować i uruchomić moduły Klienta i Serwera można na systemach Windows, Linux, macOS z zainstalowanym środowiskiem .NET Core 2.1
- Można uruchomić moduły Klienta i Serwera bez środowiska .Net Core 2.1 ale wyłącznie na systemie Windows
- Skompilować i uruchomić moduł Prezentacji (nie włączając wizualizacji) można wyłącznie na systemie Windows
- Uruchomić wizualizację można na każdej znaczącej współczesnej przeglądarce, gdyż technologia WebGL jest od lat powszechnie wspierana (Internet Explorer nie jest współczesną przeglądarką, zalecana jest Mozilla Firefox lub Google Chrome)

Szczegółowe informacje o wymaganiach poszczególnych środowisk:

- <https://github.com/dotnet/core/blob/master/release-notes/2.1/2.1-supported-os.md>
- <https://docs.microsoft.com/en-us/dotnet/framework/get-started/system-requirements>
- <https://docs.microsoft.com/en-gb/visualstudio/productinfo/vs2017-system-requirements-vs>

## 8. Biblioteki wraz z określeniem licencji

| Moduł                                 | Biblioteka   | Licencja                     |
|---------------------------------------|--|------------------------------|
| Klient, Serwer, Prezentacja           | SignalR Core   | Apache License, Version 2.0  |
| Klient, Serwer, Prezentacja           | MessagePack  | Apache License, Version 2.0  |
| Klient                                | xUnit  | Apache License, Version 2.0  |
| Prezentacja                           | MaterialSkin   | The MIT License              |
| Prezentacja (automatyczna kompilacja) | Visual Studio Locator<br>( <code>vshere.exe</code> ) | The MIT License              |
| Prezentacja (wizualizacja)            | Three.js   | The MIT License              |
| Prezentacja (wizualizacja)            | Materialize  | The MIT License              |
| Prezentacja (wizualizacja)            | jQuery   | The MIT License              |
| Prezentacja (wizualizacja)            | jQuery UI  | The MIT License              |
| Prezentacja (wizualizacja)            | MathJax  | Apache License, Version 2.0  |
| Prezentacja (wizualizacja)            | Google Material Icons                                | Apache License, Version 2.0  |
| Prezentacja (wizualizacja)            | GUST Latin Modern                                    | LaTeX Project Public License |

## **9. Instrukcja instalacji**

### **9.1. Kompilacja Klienta**

Należy upewnić się, że środowisko .Net Core 2.1 SDK lub nowsze jest zainstalowane. Należy uruchomić program `build-client-win-x64.cmd`. Skompilowany moduł znajdziemy w folderze `./Copied-ready-to-run-win-x64/Client/`

### **9.2. Kompilacja Serwera**

Należy upewnić się, że środowisko .Net Core 2.1 SDK lub nowsze jest zainstalowane. Należy uruchomić program `build-server-win-x64.cmd`. Skompilowany moduł znajdziemy w folderze `./Copied-ready-to-run-win-x64/Server/`

### **9.3. Kompilacja Prezentacji**

Należy upewnić się, że środowisko .Net Framework 4.6.1 lub nowsze oraz Visual Studio obsługujące to środowisko są zainstalowane. Należy uruchomić program `build-presentation-win-x64.cmd`. Skompilowany moduł znajdziemy w folderze `./Copied-ready-to-run-win-x64/Presentation/`

## **10. Instrukcja uruchomienia**

### **10.1. Uruchomienie Klienta na platformie Windows x64**

Należy uruchomić program `run-client.cmd http://192.168.145.184:62752` jeśli serwer jest zlokalizowany pod adresem `http://192.168.145.184:62752`. Zalecane jest uruchomienie wielu instancji.

### **10.2. Uruchomienie Serwera na platformie Windows x64**

Należy uruchomić program `run-server.cmd 10 101 http://192.168.145.184:62752 ./file.xml` jeśli serwer jest zlokalizowany pod adresem `http://192.168.145.184:62752` oraz rozmiar rozważanego automatu to 10 stanów oraz jeśli pragniemy aby zebranych było co najwyżej 101 najciekawszych automatów. Jeżeli pragniemy wznowić obliczenia to należy jako dodatkowy parametr wskazać ścieżkę do pliku wygenerowanego podczas poprzednich obliczeń `./file.xml`.

### **10.3. Uruchomienie Prezentacji**

#### **10.3.1. Windows**

Należy uruchomić program `run-presentation.cmd http://192.168.145.184:62752` jeśli serwer jest zlokalizowany pod adresem `http://192.168.145.184:62752`. Jest to parametr opcjonalny, gdyż w oknie programu także istnieje możliwość podania adresu.

## **11. Instrukcja użycia**

### **11.1. Serwer**

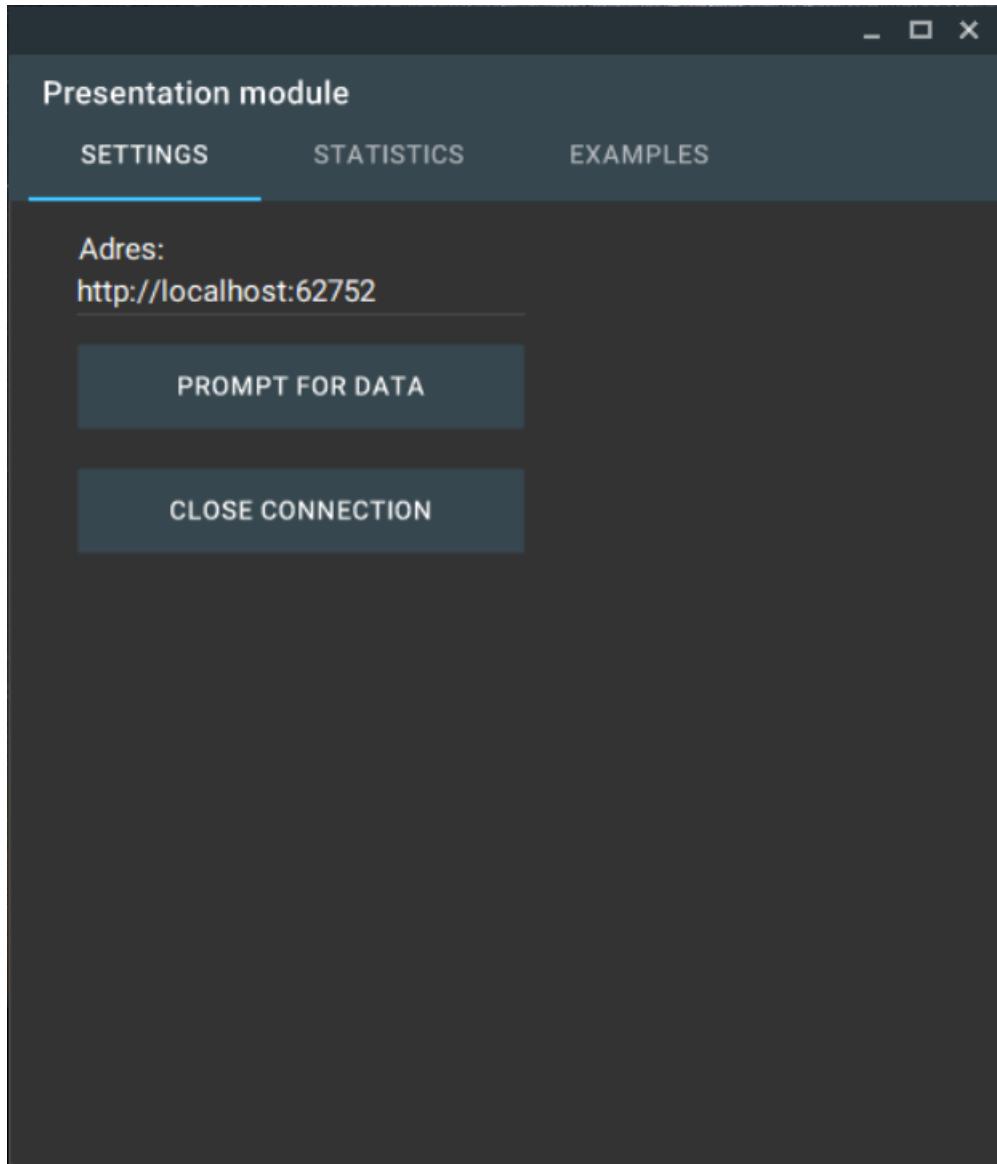
Po uruchomieniu serwera program nasłuchiwa połączeń i w pełni automatycznie wykonuje swoje zadania, także zapisuje stan obliczeń do pliku. Aby zakończyć obliczenia należy zamknąć okno lub wcisnąć kombinację klawiszy **CTRL+C**.

### **11.2. Klient**

Po uruchomieniu Klienta program automatycznie łączy się ze wskazanym serwerem i wykonuje obliczenia. W razie wystąpienia niestandardowego błędu np. komunikacji czy przerwania połączenia klient automatycznie próbuje rozpocząć pracę od nowa nawiązując połączenie zgodnie z pierwotnie podanymi parametrami.

### 11.3. Prezentacja

Po uruchomieniu programu **Presentation** ukazuje się nam następujące okno:



Rysunek 11.1: Pierwsza zakładka modułu Prezentacji

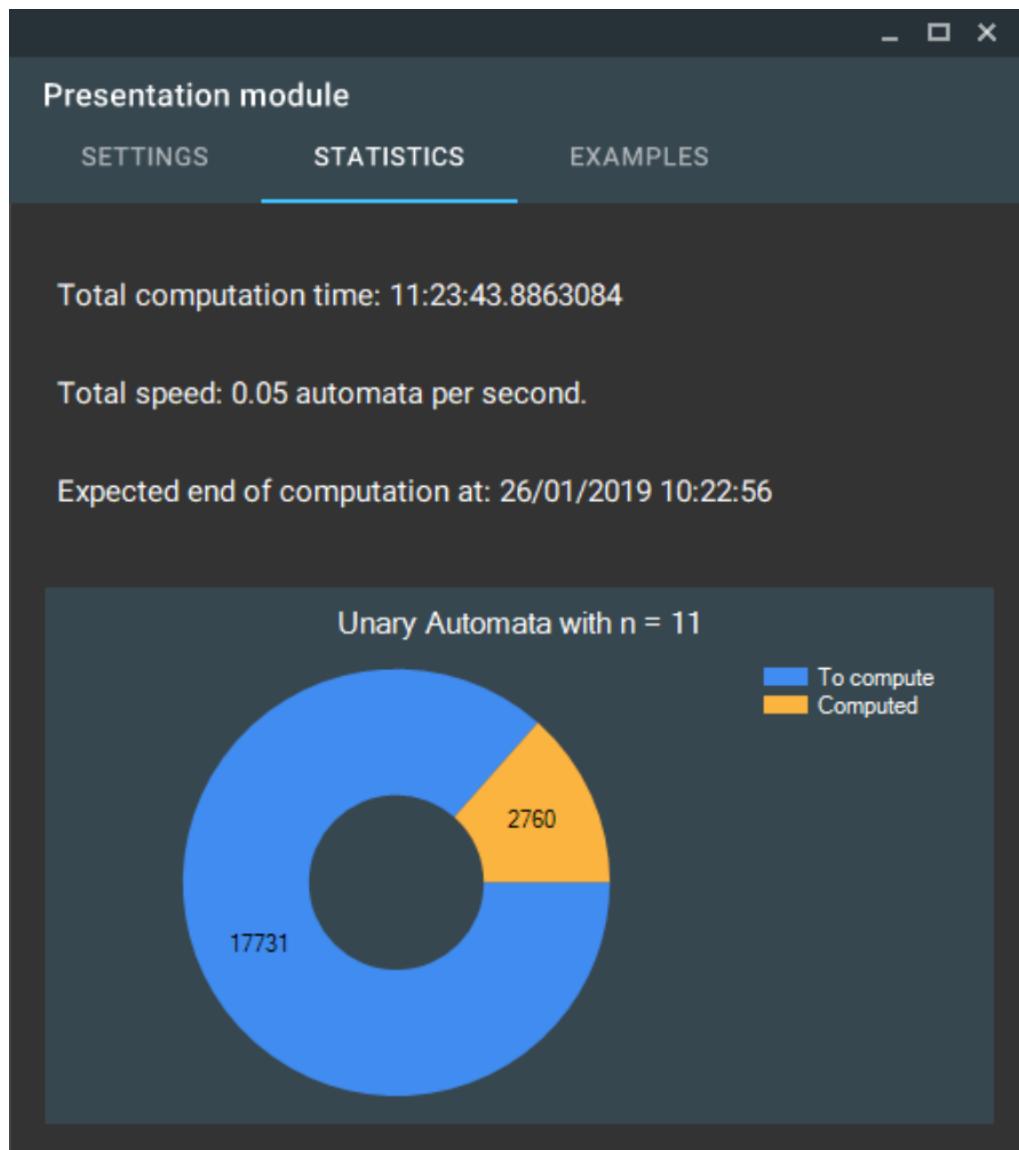
W celu nawiązania połączenia z serwerem i zgłoszenia mu chęci otrzymywania informacji na temat aktualnego stanu obliczeń należy wpisać adres serwera i następnie wcisnąć przycisk **PROMPT FOR DATA**. W razie powodzenia adres powinien podświetlić się na jaskrawy kolor i wkrótce program powinien zacząć wypisywać statystyki obliczeń oraz najciekawsze przykłady znalezionych automatów odpowiednio w dwóch pozostałych zakładkach.

W zakładce **STATISTICS** możemy dowiedzieć się o aktualnym stanie obliczeń.

Widzimy tam dotychczasowy czas obliczeń, średnią prędkość obliczeń oraz estymację czasu

### 11.3. PREZENTACJA

zakończenia obliczeń. Czas obliczeń jest liczyony na podstawie przedziałów czasu obliczeń każdego z zadań. Od pierwszej chwili rozpoczęcia rozwiązywania do ostatniej chwili zakończenia zadania każdy przedział czasu w którym nie są wykonywane obliczenia nie jest wliczany do całkowitego czasu obliczeń. Jeżeli więc kontynuujemy obliczenia po tygodniu, to tygodniowy czas przerwy nie będzie wliczany do czasu obliczeń, ponieważ żaden wynik w tym czasie nie był liczyony mimo uzyskanych wyników obliczeń tak odległych w czasie.

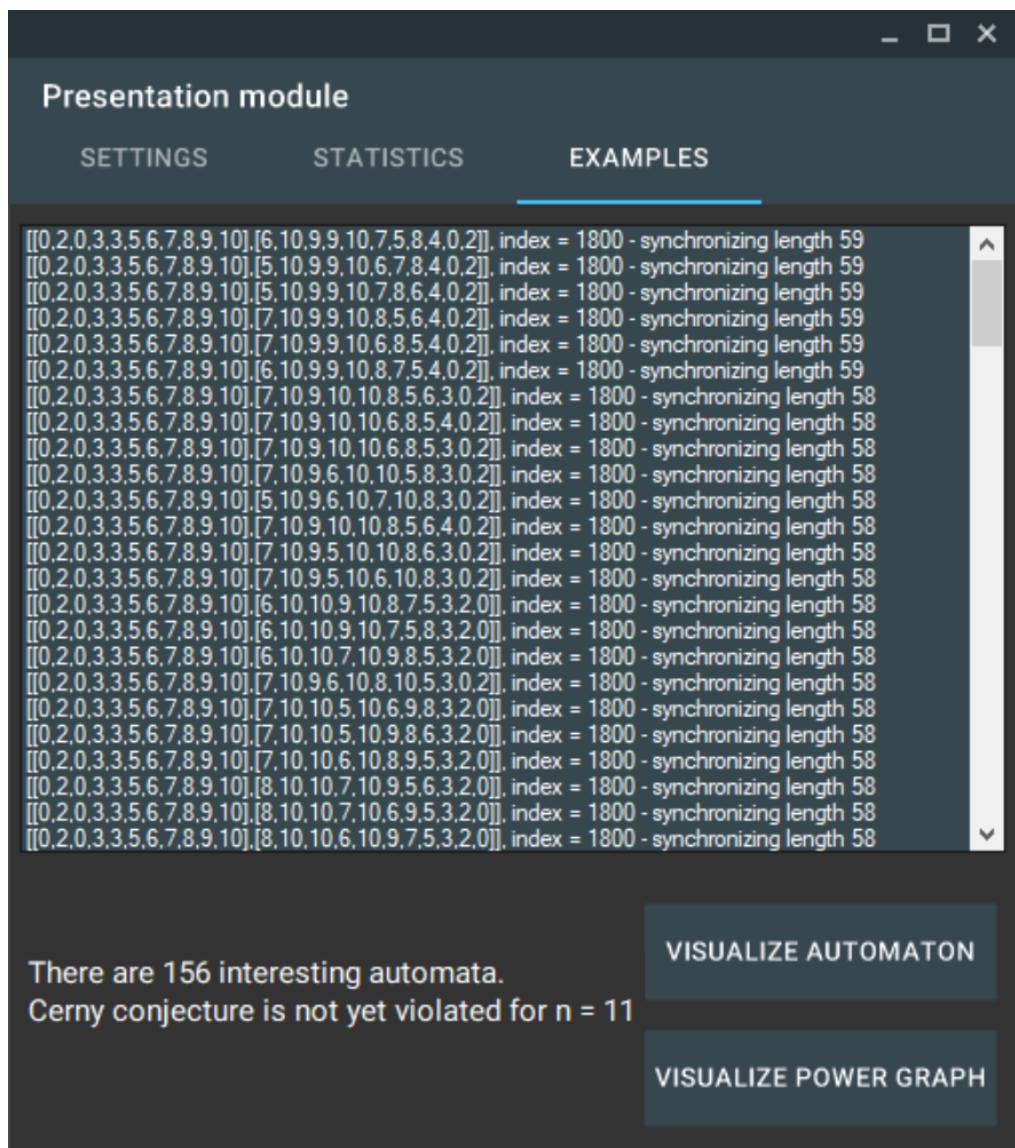


Rysunek 11.2: Moduł Prezentacji - zakładka STATISTICS

Poniżej statystyk dotyczących czasu możemy zobaczyć diagram ukazujący liczbę automatów unarnych już przeanalizowanych w stosunku do tych, które jeszcze zostały do obliczenia.

Ostatnia zakładka służy do obejrzenia dotychczas znalezionych ciekawych automatów.

Automaty te są wypisane w postaci dwóch tablic przejścia z dopisaną długością słowa synchronizującego. Automaty te są posortowane według długości słowa synchronizującego, dlatego

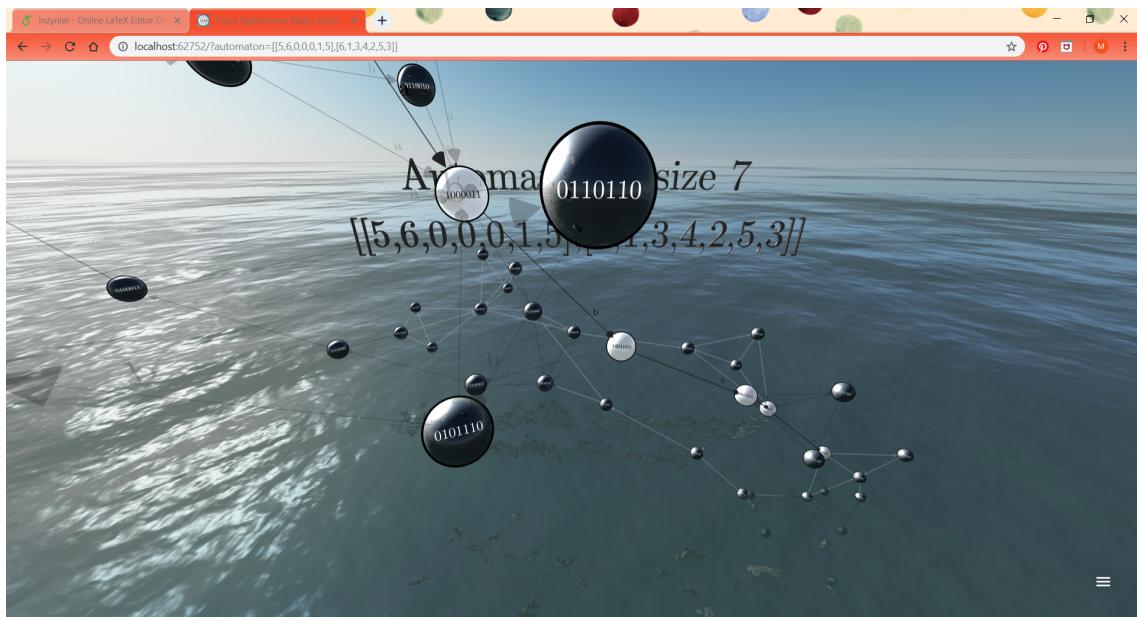


Rysunek 11.3: Zakładka EXAMPLES

w celu weryfikacji hipotezy Černýego wystarczy zbadać długość słowa synchronizującego pierwszego elementu z listy - czy jest większy od  $(n - 1)^2$ .

Pod listą mamy przycisk służący do uruchomienia wizualizacji. Jeśli naciśniemy go bez zaznaczenia konkretnego automatu możemy uruchomić demonstracyjną wizualizację grafu potęgowego automatu Černýego o 4 stanach. Możemy również zaznaczyć konkretny automat z listy i wtedy uruchomić wizualizację jednym z dwóch przycisków. Będzie to skutkowało uruchomieniem wizualizacji grafu potęgowego bądź samego automatu, który został zaznaczony. Wizualizacja uruchamia się w domyślnej przeglądarce (jeśli domyślną jest Internet Explorer to zalecana jest zmiana domyślnej przeglądarki przed uruchomieniem wizualizacji).

## 11.4. Wizualizacja



Rysunek 11.4: Uruchomiona wizualizacja w przeglądarce użytkownika

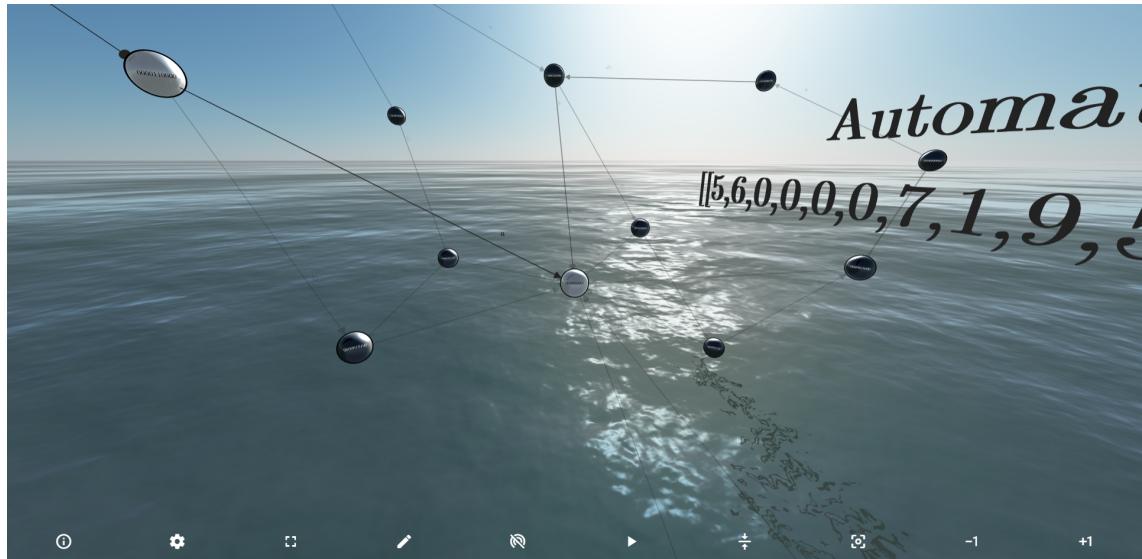
Do uruchomienia wizualizacji z poziomu modułu Prezentacji konieczne jest połączenie z Serwerem.

W ramach wizualizacji możemy obracać kamerą wokół punktu skupienia którym jest środek masy grafu. Klikając myszką na wierzchołek koncentrujemy animację na danym wierzchołku. Odległość kamery od punktu skupienia sterujemy przewijaniem myszki. Podświetlone wierzchołki znajdują się się na ścieżce słowa synchronizującego, natomiast czarne, niepodświetlone to wszystkie pozostałe odkryte wierzchołki.

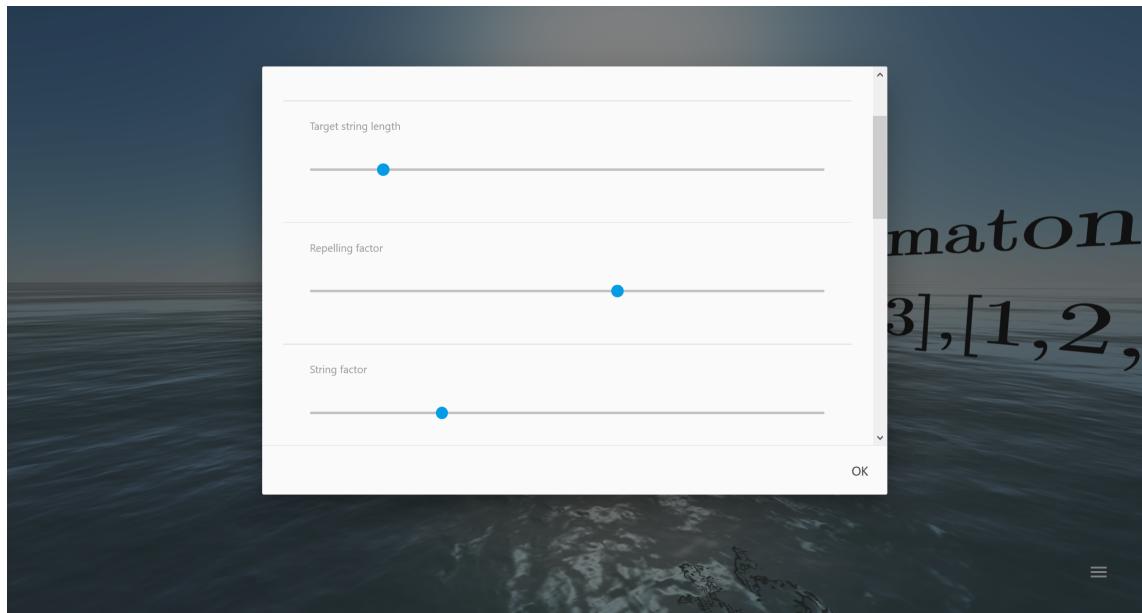
W prawym dolnym rogu znajduje się przycisk którym możemy rozwinąć menu.

Po rozwinięciu menu możemy skorzystać z różnych możliwości animacji. Włączając opcję uruchamiamy animację wzdłuż ścieżki synchronizującej automat. Inną ciekawą opcją jest wymuszenie wyświetlania automatu w płaszczyźnie zamiast w 3 wymiarach. Aby uruchomić tę opcję wystarczy włączyć przycisk . Z rozwiniętego menu możemy wejść w ustawienia. Po wciśnięciu uruchamia się okienko ustawień.

W ustawieniach znajduje się wiele opcji działania wizualizacji m.in. zmiana siły oddziaływania lub docelowe odległości między wierzchołkami, a także zmiana pory dnia.

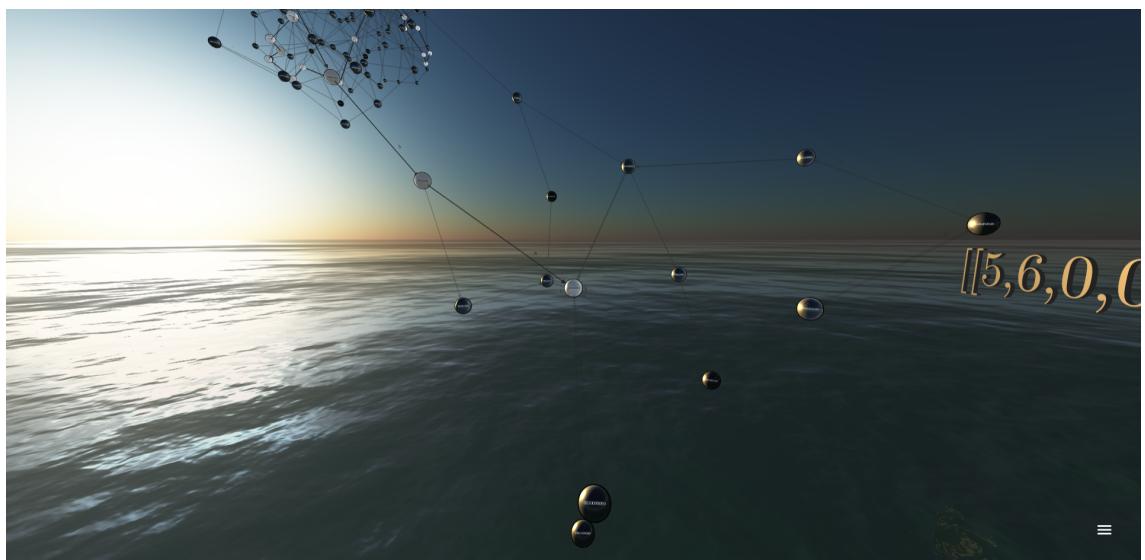


Rysunek 11.5: Rozwinięte menu: widoczne są dodatkowe możliwości animacji grafu



Rysunek 11.6: Panel ustawień

#### 11.4. WIZUALIZACJA



Rysunek 11.7: Ścieżka od stanu początkowego do singletonu jest wyróżniona kolorem białym lub odcieniami koloru szarego w zależności od reprezentacji binarnej stanu

## **12. Instrukcja utrzymania**

W razie niespodziewanego zatrzymania jednego z klientów, możemy go uruchomić ponownie bez obaw o poprawność obliczeń, a nawet oddać zaległe obliczenia pozostałym klientom. Serwer jest w stanie zająć się automatami, których analiza nie została dokonczona i w przyszłości przyznane nie obliczone automaty innemu klientowi.

W razie niespodziewanego zatrzymania serwera możemy odtworzyć obliczenia na podstawie przenośnej bazy jaką na bieżąco zapisuje serwer w formie pliku w katalogu roboczym z którego został wywołany, np. folder uruchomienia skryptu **run-server.cmd**. Procedura odzyskiwania obliczeń została opisana w punkcie 10.2 opisującym proces uruchomienia ze wznowieniem.

## **13. Model danych**

Aktualny stan obliczeń jest przechowywany w pamięci, a za każdym uzupełnieniem bazy danych klasa reprezentująca bazę jest serializowana do przenośnego pliku .xml. Podczas uruchamiania obliczeń przy podanym parametrze ścieżki do pliku sprawdzane jest jego istnienie oraz możliwość jego zdeserializowania w celu wznowienia obliczeń.

## 14. Raport z przeprowadzonych testów

Programy przechodzą odpowiadające im testy jednostkowe:

| Moduł  | Komponent  | Element   | Szczegóły testy  |
|--------|--|---|--|
| Klient | Generowanie nieizomorficznych automatów unarnych             | Komponent sprawdzania izomorficzności   | Sprawdzenie czy komponent poprawnie rozpoznaje automat unarny i jego permutację jako izomorficzne  |
| Klient | Generowanie nieizomorficznych automatów unarnych             | Nieizomorficzność wygenerowanych automatów  | Sprawdzenie czy wygenerowane automaty unarne są parami nieizomorficzne (charakterystyka algorytmu jest taka, że kończy działanie wtedy i tylko wtedy kiedy znajdzie wszystkie automaty nieizomorficzne)  |
| Klient | Szukanie długości słowa synchronizującego automatu binarnego | Poprawność dla automatów Ćernego, przypadku granicznego   | Długość słowa synchronizującego to $(n - 1)^2$   |
| Klient | Szukanie długości słowa synchronizującego automatu binarnego | Identyczność wyników między algorytmem typu brute-force i między implementacjami w różnym stopniu zoptymalizowanymi | Sprawdzane są kolejno wyniki pod względem zgodności (długość słowa synchronizującego lub jego brak). Dla powolnego algorytmu brute-force sprawdzane są losowo automaty o liczbie stanów $n \leq 5$ a między dwoma istotnie różniącymi się szybkimi algorytmami dla $n \leq 12$ |

Tablica 14.1: Testy jednostkowe (1)

|        |  |  |   |
|--------|--|--|---|
| Klient | Szukanie długości słowa synchronizującego automatu binarnego | Zwracanie dokładnie tylu rozwiązań automatorów na wyjściu ile na wejściu | Liczba rozwiązań automatorów (czyli automatorów z informacją o długości słowa synchronizującego lub o braku synchronizowalności) jest taka sama jak liczba automatorów nierożwiązań (automaty nie są gubione, ani ponadmiarowe) |
| Klient | Przeszukiwanie całej przestrzeni automatorów binarnych       | Sprawdzanie czy wierzchołek znajduje się w automacie $A_C$               | Sprawdzenie czy odpowiednie wierzchołki oznaczane są prawidłowo jako znajdujące się w automacie $A_C$   |
| Klient | Przeszukiwanie całej przestrzeni automatorów binarnych       | Tworzenie automatorów $A_C$  | Sprawdzenie czy wygenerowane zostają wszystkie automaty $A_C$ z danego automatu unarnego i czy nie są generowane automaty ponadmiarowe.   |
| Klient | Przeszukiwanie całej przestrzeni automatorów binarnych       | Tworzenie pełnych automatorów  | Sprawdzenie czy wygenerowane zostają wszystkie pełne automaty z danego automatu $A_C$ i czy nie są generowane automaty ponadmiarowe.  |

Tablica 14.2: Testy jednostkowe (2)

## 15. Przeprowadzone obliczenia

Korzystając ze stworzonego przez nas systemu udało nam się przeprowadzić obliczenia potwierdzające hipotezę Černýego dla automatów binarnych mających od 6 do 10 stanów. Dla automatów o 11 stanach udało nam się uruchomić obliczenia, jednak czas obliczeń był zbyt duży aby te obliczenia zakończyć na czas zajęcia sal komputerowych. Poniżej zamieszczamy czasy działania oraz pamięć RAM wykorzystywaną przez jednego klienta dla poszczególnych obliczeń.

| liczba stanów | czas obliczeń     | wykorzystywana pamięć RAM | liczba komputerów | liczba klientów na komputerze |
|---------------|-------------------|---------------------------|-------------------|-------------------------------|
| 6             | 0.11 sek          | 15MB - 18MB               | 15                | 16                            |
| 7             | 1 min 43 sek      | 19MB - 23MB               | 15                | 16                            |
| 8             | 1 min 22 sek      | 20MB - 25MB               | 15                | 16                            |
| 9             | 8 min 13 sek      | 15 MB - 1.5 GB            | 15                | 16                            |
| 10            | 2 h 0 min 46 sek  | 18 MB - 300 MB            | 14                | od 2 do 8                     |
| 11            | prawdop. ok. 100h | 25 MB - 13 GB             | 29                | 1                             |

Tablica 15.1: Przeprowadzone obliczenia

Jak widać dużym problemem poza złożonością czasową okazała się złożoność pamięciowa, czego nie przewidzieliśmy na początku obliczeń. Z tego powodu dla większych automatów nie można było uruchomić na jednym komputerze wielu instancji Klienta. W końcowej fazie projektu udało nam się zmniejszyć złożoność pamięciową (np. dla 9 było to na początku średnio 3GB, a po poprawach jest to średnio 20MB, a dla 10 i 11, wiemy tylko, że wcześniej było to powyżej 16GB, gdyż dla tych klientów brakowało pamięci RAM na komputerach, a teraz można spokojnie uruchomić instancje Klienta dla tych wielkości automatów), jednak wciąż jest to zbyt dużo potrzebnej pamięci, aby swobodnie uruchomić wiele instancji Klienta przy obliczeniach dla 11, czy 12 stanów.

## Bibliografia

- [1] D. Eppstein, Reset Sequences for Monotonic Automata *Reset Sequences for Monotonic Automata SIAM Journal on Computing*, 19 (3), 1990, 500–510, doi:10.1137/0219033.
- [2] A. Kisielewicz, M. Szykuła, Generating Small Automata and the Černý Conjecture *Implementation and Application of Automata*, Publisher, 2013, 340–348.
- [3] M. Volkov, Synchronizing Automata and the Černý Conjecture *Proc. 2nd Int'l. Conf. Language and Automata Theory and Applications (LATA 2008) (PDF)*, LNCS, 5196, Springer-Verlag, 2008, 11–27.
- [4] A. Kisielewicz, J. Kowalski, M. Szykuła, Experiments with Synchronizing Automata *In Implementation and Application of Automata (CIAA 2016)*, LNCS, 9705, 2016, 176–188.

## Spis rysunków

|      |  |    |
|------|--|----|
| 1.1  | Automat o czterech stanach i dwuelementowym alfabetie . . . . .  | 12 |
| 1.2  | Automat Ćernego $\mathcal{C}_4$ . . . . .  | 13 |
| 1.3  | Automat $\mathcal{A}_C$ automatu z rysunku 1.1 . . . . .   | 15 |
| 2.1  | Diagram przypadków użycia . . . . .  | 17 |
| 3.1  | Poglądowy schemat architektury z oznaczeniem technologii wymiany danych pomiędzy modułami. Wymiana danych następuje między komponentami odpowiedzialnymi za komunikację w ramach każdego modułu. . . . . | 21 |
| 3.2  | Diagram Sekwencji. Widoczna jest duża liczba asynchronicznych wywołań. . . . .   | 22 |
| 3.3  | Schemat architektury Klienta . . . . .   | 23 |
| 3.4  | Schemat architektury Serwera . . . . .   | 23 |
| 3.5  | Schemat architektury Prezentacji . . . . .   | 24 |
| 11.1 | Pierwsza zakładka modułu Prezentacji . . . . .   | 36 |
| 11.2 | Moduł Prezentacji - zakładka STATISTICS . . . . .  | 37 |
| 11.3 | Zakładka EXAMPLES . . . . .  | 38 |
| 11.4 | Uruchomiona wizualizacja w przeglądarce użytkownika . . . . .  | 39 |
| 11.5 | Rozwinięte menu: widoczne są dodatkowe możliwości animacji grafu . . . . .   | 40 |
| 11.6 | Panel ustawień . . . . .   | 40 |
| 11.7 | Ścieżka od stanu początkowego do singletonu jest wyróżniona kolorem białym lub odcieniami koloru szarego w zależności od reprezentacji binarnej stanu . . . . .  | 41 |

## **Spis tabel**

|      |  |    |
|------|--|----|
| 2.1  | Opis poszczególnych przypadków użycia - węzeł obliczeniowy . . . . . | 17 |
| 2.2  | Opis poszczególnych przypadków użycia . . . . .                      | 18 |
| 2.3  | Niefunkcjonalne wymagania systemu . . . . .                          | 19 |
| 14.1 | Testy jednostkowe (1) . . . . .                                      | 44 |
| 14.2 | Testy jednostkowe (2) . . . . .                                      | 45 |
| 15.1 | Przeprowadzone obliczenia . . . . .                                  | 46 |