

# Aplikacja wizualizująca działanie transducerów Mealy'ego i Moore'a

(Mealy's and Moore's transducers visualizer)

Bartłomiej Najdecki

Praca licencjacka

**Promotor:** dr Jakub Michaliszyn

Uniwersytet Wrocławski  
Wydział Matematyki i Informatyki  
Instytut Informatyki

30 czerwca 2017

Bartłomiej Najdecki

.....

.....

(adres zameldowania)

.....

.....

(adres korespondencyjny)

PESEL: .....

e-mail: .....

Wydział Matematyki i Informatyki

stacjonarne studia I stopnia

kierunek: informatyka

nr albumu: 273975

### **Oświadczenie o autorskim wykonaniu pracy dyplomowej**

Niniejszym oświadczam, że złożoną do oceny pracę zatytułowaną *Aplikacja wizualizująca działanie transducerów Mealy’ego i Moore’a* wykonałem/am samodzielnie pod kierunkiem promotora, dr Jakuba Michaliszyna. Oświadczam, że powyższe dane są zgodne ze stanem faktycznym i znane mi są przepisy ustawy z dn. 4 lutego 1994 r. o prawie autorskim i prawach pokrewnych (tekst jednolity: Dz. U. z 2006 r. nr 90, poz. 637, z późniejszymi zmianami) oraz że treść pracy dyplomowej przedstawionej do obrony, zawarta na przekazanym nośniku elektronicznym, jest identyczna z jej wersją drukowaną.

Wrocław, 30 czerwca 2017

(czytelny podpis)

## Streszczenie

Celem pracy jest zaprezentowanie aplikacji wizualizującej model matematyczny zwany transducerem. Transducery są niezwykle przydatne przy przetwarzaniu języka i mowy [4]. Dzięki zastosowaniu transducerów potrafimy często uprościć rozwiązania wielu problemów zaczynając od prostych problemów jak sprawdzanie palindromiczności słowa [Przykład 2.3.3.], kończąc na zmianie struktury systemów rozpoznających mowę [3].

---

In this thesis we present the Transducer Visualizer application which is able to visualize mathematical model called transducer. Main purpose of using transducer is language and speech processing [4]. Transducers can simplify and speed up solutions of many problems like palindrom checking [Example 2.3.3.] or even speech recognition [3].



# Spis treści

<b>1. Wprowadzenie</b>	<b>7</b>
1.1. Automat skończony . . . . .	7
1.2. Automat skończony ze stosem . . . . .	8
1.3. Transducer Mealy’ego . . . . .	8
1.4. Transducer Moore’a . . . . .	10
1.5. Transducery ze stosem . . . . .	11
<b>2. Aplikacja</b>	<b>13</b>
2.1. Technologie . . . . .	13
2.1.1. Biblioteka graficzna . . . . .	13
2.1.2. Format wejścia . . . . .	13
2.2. Funkcjonalności . . . . .	16
2.2.1. Pojedynczy transducer . . . . .	16
2.2.2. Kolejka transducerów . . . . .	16
2.2.3. Widok . . . . .	17
2.3. Przykłady . . . . .	18
2.3.1. Konwersja z systemu unarnego na system binarny . . . . .	18
2.3.2. Korekta literówek . . . . .	18
2.3.3. Proste wykrywanie parzystych palindromów . . . . .	20
<b>3. Podsumowanie</b>	<b>23</b>
3.1. Zastosowania . . . . .	23
3.2. Dalszy rozwój . . . . .	23



# Rozdział 1.

## Wprowadzenie

### 1.1. Automat skończony

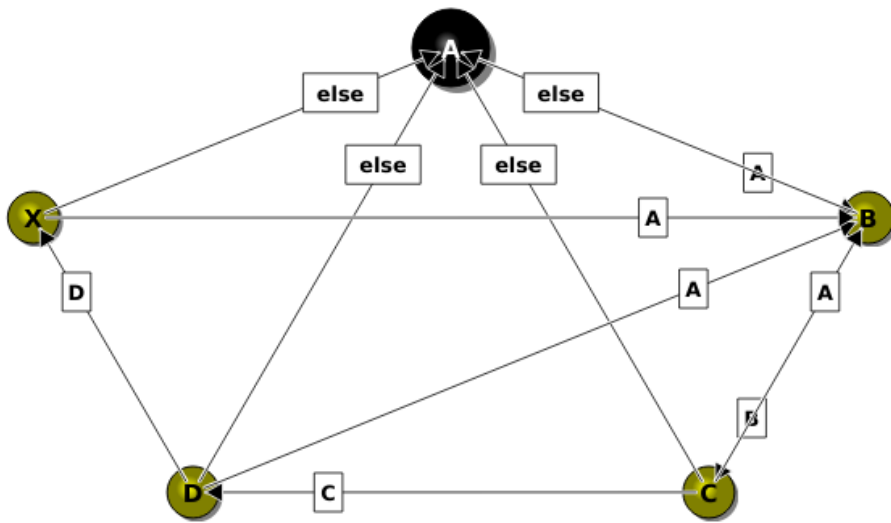
Automat skończony (Deterministic Finite-state Automaton) jest modelem matematycznym systemu o dyskretnych wejściach i wyjściach [2]. Automaty najczęściej wykorzystuje się do sprawdzenia przynależności danego słowa do definiowanego przez nie języka. DFA jest 5 elementową krotką  $\langle \Sigma, Q, q_0, F, \delta \rangle$ , gdzie  $\Sigma$  to alfabet nad którym język rozważamy,  $Q$  to zbiór stanów automatu,  $q_0$  - wyróżniony stan początkowy,  $F$  - podzbiór  $Q$ , oznaczający stany akceptujące oraz  $\delta$  - funkcja przejścia pomiędzy stanami, definiowana jako funkcja typu:

$$Q \times \Sigma \rightarrow Q$$

Automat odczytuje słowo litera po literze zmieniając stany przy użyciu funkcji przejścia, a następnie, w zależności od tego czy jest w stanie akceptującym czy nie, akceptuje bądź odrzuca słowo. Prosty przykład DFA jest ukazany na rysunku 1.1 - jest to automat akceptujący słowa, których sufiks to ABCD. Automat rozpoczyna w stanie A, a następnie zmienia stan na B, gdy odczyta A, natomiast w przeciwnym przypadku nie zmienia stanu. Stanem akceptującym jest X (nie zaznaczony). Sporą zaletą definiowanych w ten sposób automatów jest liniowa złożoność przetwarzania słów (zakładając, że automat już wcześniej został stworzony), jednak są to modele mocno ograniczone - języki przez nie rozpoznawane są bardzo proste. Języki rozpoznawalne przez automaty skończone nazywamy językami regularnymi. Dla ułatwienia definiujemy rozszerzenie funkcji  $\delta$  w ten sposób:

$$\hat{\delta}(q, a) = \delta(q, a)$$

$$\hat{\delta}(q, wa) = \delta(\hat{\delta}(q, w), a)$$



Rysunek 1.1: Przykład DFA - na czarno zaznaczony wierzchołek początkowy. Stanem akceptującym jest X

## 1.2. Automat skończony ze stosem

Aby zwiększyć zbiór możliwych do rozpoznania języków dodamy do automatu stos - oprócz stanu, automat będzie miał stos będący skończonym słowem nad pewnym alfabetem  $\Sigma'$  z wyróżnionym znakiem  $k_{\Sigma'}$  oznaczającym dno stosu. Zmiany wymaga również funkcja przejścia:

$$Q \times \Sigma \times \Sigma' \rightarrow Q \times \Sigma'^*$$

Funkcja  $\delta$  zależy teraz również od tego co jest na szczycie stosu. Istnieją dwie konwencje akceptowania przez taki automat:

- Przez pusty stos
- Przez osiągnięcie stanu akceptowalnego

W przypadku transducerów, które są tematem tej pracy, to rozróżnienie nie jest potrzebne (transducer nie akceptuje). Dzięki takiemu rozszerzeniu klasa języków rozpoznawalnych przez automaty rozszerza się do języków bezkontektowych.

## 1.3. Transducer Mealy'ego

Automaty ze stosem są potężnym modelem matematycznym, jednak potrafią one tylko rozstrzygać problem przynależności słowa do języka. Automaty potrafią z łatwością znajdować wzorce w tekście - dlaczego miałyby nie umieć ich zastępować



albo zliczać?

Transducerem Mealy'ego [6] nazywamy krotkę

$$\langle \Sigma, \Sigma_1, Q, q_0, \delta, \sigma \rangle$$

w której  $\langle \Sigma, Q, q_0, \emptyset, \delta \rangle$  jest DFA z pustym zbiorem akceptującym, natomiast  $\sigma$  jest funkcją  $Q \times \Sigma \rightarrow \Sigma_1^*$ . Dodatkowo dla transducera T definiujemy

$$f_T(wa) = f_T(w)\sigma(\delta(w, q_0), a)$$

$f_T$  jest funkcją wyjścia transducera - to za jej pomocą generujemy wynik działania transducera.

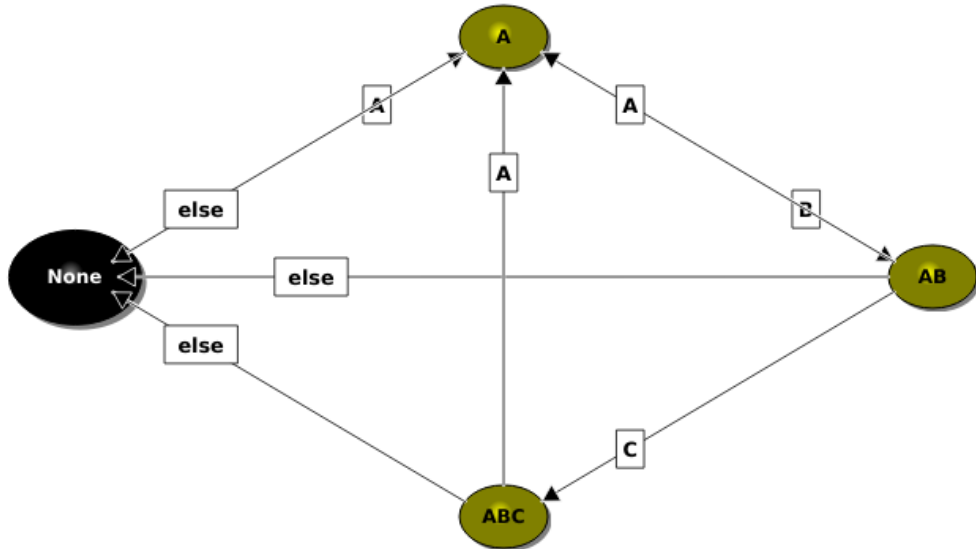
**Przykład 1.3..1.** Rozważmy problem wyszukiwania wzorca ABCD w tekście i zastępowania go wzorcem EF. Zdefiniujmy zbiór stanów w następujący sposób:

$$Q = \{None, A, AB, ABC\}$$

Następnie definiujemy deltę w ten sposób:

	None	A	AB	ABC
a	A	A	A	A
b	None	AB	None	None
c	None	None	ABC	None
inna	None	None	None	None

Tablica 1.1: Funkcja przejścia



Rysunek 1.2: DFA należące do transducera

Pozostało już tylko odpowiednio zdefiniować funkcję  $\sigma$ . Chcemy, by  $\sigma$  nie wypisywała

nic, gdy natrafi na kolejną literę wzorca lub wypisała EF gdy wzorec się skończy, natomiast gdy wczytana litera nie jest kolejną literą wzorca funkcja powinna przepisać literę. Dla wygody oznaczmy przez \$ literę wczytaną przez  $\sigma$ :

Zdefiniowany w ten sposób transducer pamięta w stanie ile liter wyszukiwanego

	None	A	AB	ABC
a	$\epsilon$	a	ab	abc
b	b	$\epsilon$	abb	abcb
c	c	ac	$\epsilon$	abcc
d	d	ad	abd	ef
inna	\$	a\$	ab\$	abc\$

Tablica 1.2: Funkcja przejścia

wzorca wczytał, by móc w odpowiednim momencie wypisać 'ef' lub wczytany prefiks wzorca.

## 1.4. Transducer Moore'a

Różnica pomiędzy transducerem Moore'a a Mealy'ego polega na innej definicji funkcji  $\sigma$ [7]. W transducerze Mealy'ego funkcja  $\sigma$  miała typ:

$$Q \times \Sigma \rightarrow \Sigma_1^*$$

Natomiast w transducerze Moore'a  $\sigma$  jest funkcją:

$$Q \rightarrow \Sigma_1^*$$

Poniższe twierdzenie jest znane z literatury (zobacz [8]).

**Twierdzenie 1.** *Powyższe modele są równoważne.*

*Dowód.* Aby z transducera Moore'a zrobić transducer Mealy'ego wystarczy pomijając drugi parametr - niech  $\sigma_{Mo}$  oznacza funkcję  $\sigma$  dla transducera Moore'a. Wtedy definiujemy  $\sigma_{Me}$  równoważnego transducera Mealy'ego jako:

$$\sigma_{Me}(q, a) = \sigma_{Mo}(q)$$

Konwersja automatu Mealy'ego na automat Moore'a jest nieco bardziej skomplikowana. Niech  $\langle \Sigma, \Sigma_1, Q, q_0, \delta, \sigma \rangle$  oznacza pewien automat Mealy'ego. Chcemy znaleźć równoważny automat Moore'a  $\langle \Sigma', \Sigma'_1, Q', q'_0, \delta', \sigma' \rangle$ . Z oczywistych powodów, wiemy że:

$$\Sigma' = \Sigma$$

$$\Sigma'_1 = \Sigma_1$$

Następnie zdefiniujmy zbiór stanów  $Q'$ :

$$Q' = Q \times \Sigma$$

Teraz możemy pamiętać ostatnio wczytaną literę potrzebną funkcji  $\sigma$  w transducerze Mealy'ego.

$$\delta'((q, a), b) = (\delta(q, a), b)$$

$$\sigma'((q, a)) = \sigma(q, a)$$

Poprawność wynika z konstrukcji. □

## 1.5. Transducery ze stosem

Wiedząc już co to są transducery bardzo łatwo możemy rozszerzyć ich definicję do transducerów ze stosem. Transducerem ze stosem nazwiemy krotkę

$$\langle \Sigma, \Sigma_1, \Sigma', Q, q_0, \delta, \sigma \rangle$$

w której  $\langle \Sigma, \Sigma', Q, \emptyset, q_0, \delta \rangle$  jest automatem skończonym ze stosem, zaś  $\sigma$  jest funkcją:

- $Q \times \Sigma \times \Sigma' \rightarrow \Sigma_1^*$  dla transducerów ze stosem Mealy'ego
- $Q \times \Sigma' \rightarrow \Sigma_1^*$  dla transducerów Moore'a

Co więcej, również te transducery są równoważne (dowód taki sam jak w twierdzeniu [1]).



## Rozdział 2.

# Aplikacja

Link do repozytorium aplikacji [1]. Znajduje się tam również dokładna instrukcja instalacji.

### 2.1. Technologie

#### 2.1.1. Biblioteka graficzna

Aplikacja w całości została napisana w C++ wraz z biblioteką Qt. Qt jest zestawem przenośnych bibliotek dedykowanych C++ i Javie udostępnianych na licencji GPL, LGPL i komercyjnej. Jest to bardzo dojrzała biblioteka - pierwsza komercyjna wersja powstała w latach 90 [5]. Obecnie jest to najpopularniejsza biblioteka dla aplikacji okienkowych w C++ dla systemu Linux - na Qt bazuje m.in. KDE. Dzięki temu aplikacja jest stabilna i przenośna. Do testowania aplikacji użyłem komponentu biblioteki Qt - QtTest.

#### 2.1.2. Format wejścia

Transducery są wczytywane z plików o rozszerzeniu 'json' w formacie przypominającym uproszczonego JSON'a. Format JSON został wybrany ze względu na czytelność oraz łatwość opisu i rozbudowywania. Opis każdego transducer'a musi zawierać wewnątrz klamr `{}`. Tablica 2.1 opisuje pola, które powinien posiadać opis transducer.

Typ funkcji  $\sigma$  oraz  $\delta$  zależy od typu transducer - tabela 2.2. W tej aplikacji opis  $\delta$

Nazwa pola	Znaczenie
<b>name</b>	Wyświetlana w programie nazwa transducera
<b>states</b>	Dyskretny zbiór stanów transducera prezentowany w postaci grafu w aplikacji
<b>initial_state</b>	Stan początkowy
<b>type</b>	Ma wartość Mealy jeśli opisujemy transducer Mealy’ego. Ma wartość Moore jeśli opisujemy transducer Moore’a.
<b>delta</b>	Funkcja przejścia DFA, więcej poniżej
<b>sigma</b>	Funkcja wyjścia transducera, więcej poniżej
hasStack	Jeśli ma wartość 'true' to transducer ma stos. Jeśli pominiemy pole lub ustawimy na inną wartość, to transducer nie będzie miał stosu

Tablica 2.1: Lista właściwości transducera - pogrubione właściwości są obowiązkowe

oraz  $\sigma$  wygląda w podobny sposób i przypomina konstrukcję switch-case - opisujemy przypadki pozbywając się kolejnych argumentów aż do usunięcia wszystkich akcji. Po przejściu przez wszystkie argumenty wykonujemy zapisaną akcję. Przykładowy opis znajduje się na obrazku 2.1.

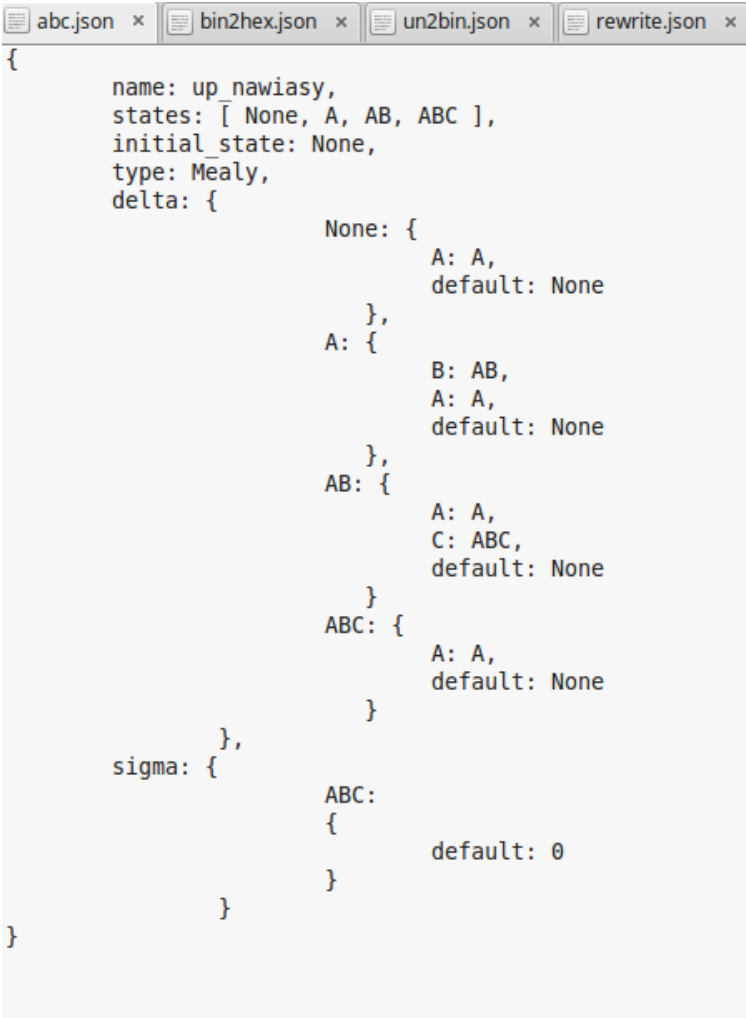
Type transducera	typ funkcji $\delta$	typ funkcji $\sigma$
Transducer Mealy’ego bez stosu	$Q \times \Sigma \rightarrow Q$	$Q \times \Sigma \rightarrow \Sigma_1^*$
Transducer Moore’a bez stosu	$Q \times \Sigma \rightarrow Q$	$Q \rightarrow \Sigma_1^*$
Transducer Mealy’ego ze stosu	$Q \times \Sigma \times \Sigma' \rightarrow Q$	$Q \times \Sigma \times \Sigma' \rightarrow \Sigma_1^*$
Transducer Moore’ego ze stosu	$Q \times \Sigma \times \Sigma' \rightarrow Q$	$Q \times \Sigma' \rightarrow \Sigma_1^*$

Tablica 2.2: Typy funkcji w zależności od typu transducera

Dodatkowo definiujemy symbole specjalne 'default' lub równoważnie ' \_ ' jako oznaczające dowolną inną wartość. Oprócz tego do dyspozycji mamy zarezerwowane słowo 'epsilon' oznaczające epsilon przejście. Jeśli epsilon występuje w miejscu wyboru wartości wczytanego symbolu, to nie powinien wystąpić żaden inny symbol (program zignoruje dodatkowy wybór)! W przypadku automatu ze stosem w miejscu selekcji górnego elementu stosu mamy do dyspozycji słowo 'empty', które oznacza pusty stos.

Przejdźmy teraz do opisu akcji, której wybór omówiliśmy powyżej. Akcje dla automatów bez stosu są opisane jako stan dla funkcji  $\delta$  lub jako słowo dla funkcji  $\sigma$ . Dodatkowo funkcja  $\sigma$  podstawia za każde wystąpienie \$ wczytany znak (gdy akcja została wybrana jako default).

Dla automatów ze stosem akcja jest opisana jako dwuelementowa lista - pierwszy



```

{
  name: up_nawiasy,
  states: [ None, A, AB, ABC ],
  initial_state: None,
  type: Mealy,
  delta: {
    None: {
      A: A,
      default: None
    },
    A: {
      B: AB,
      A: A,
      default: None
    },
    AB: {
      A: A,
      C: ABC,
      default: None
    },
    ABC: {
      A: A,
      default: None
    }
  },
  sigma: {
    ABC: {
      default: 0
    }
  }
}

```

Rysunek 2.1: Opis transducera zliczającego w systemie unarnym wystąpienia wzorca abc

element jest taki jak opisany powyżej, natomiast drugi jest poleceniem stosowym. Opis funkcji  $\sigma$  jest prawie taki sam - musimy dodać decyzję na podstawie stosu, dodatkowo każde wystąpienie % zostanie zastąpione elementem ze szczytu stosu. Tablica 2.3 opisuje polecenia stosowe dla funkcji  $\delta$ .

Polecenie	Znaczenie
$\vee X$	Usuń $X$ elementów ze stosu - $X$ można pominąć (wtedy $X=1$ ).
$\wedge X$	Dodaj litery słowa $X$ na stos. Po dodaniu na szczycie stosu będzie ostatnia litera $X$ .
-	Nic nie rób

Tablica 2.3: Polecenia stosowe

## 2.2. Funkcjonalności

### 2.2.1. Pojedynczy transducer

Transducer jest prezentowany jako kolejno graf stanów z zaznaczonym aktualnym stanem, nazwą transducera, stosem wyświetlającym maksymalnie 12 elementów znajdujących się na szczycie, listą wczytanych znaków oraz aktualnym wyjściem transducera. Każdy transducer może zostać wczytany z pliku - więcej w następnej sekcji.

### 2.2.2. Kolejka transducerów

W celu uproszczenia rozwiązań zaimplementowałem kolejkę transducerów - wyjście każdego kolejnego transducera jest traktowane jako wejście kolejnego.

Aby dodać wierzchołek do listy wystarczy wybrać opcję 'Add to pipe' z menu 'Pipe' lub wcisnąć 'T'. W ten sposób dodajemy transducer z pliku. Nowo dodany transducer zostanie dodany do zakładki pod górnym menu programu. Aby zobaczyć jego składowe wystarczy kliknąć na zakładkę z jego nazwą.

Aby usunąć transducer musimy przełączyć widok za pomocą zakładek na transducer, który chcemy usunąć. Następnie z menu 'Pipe' wybieramy opcję 'Remove transducer'.

Aby ustawić wejście dla kolejki wystarczy wybrać 'Edit input' z menu 'Pipe' lub nacisnąć klawisz 'I'. Symulację można uruchamiać na dwa sposoby:

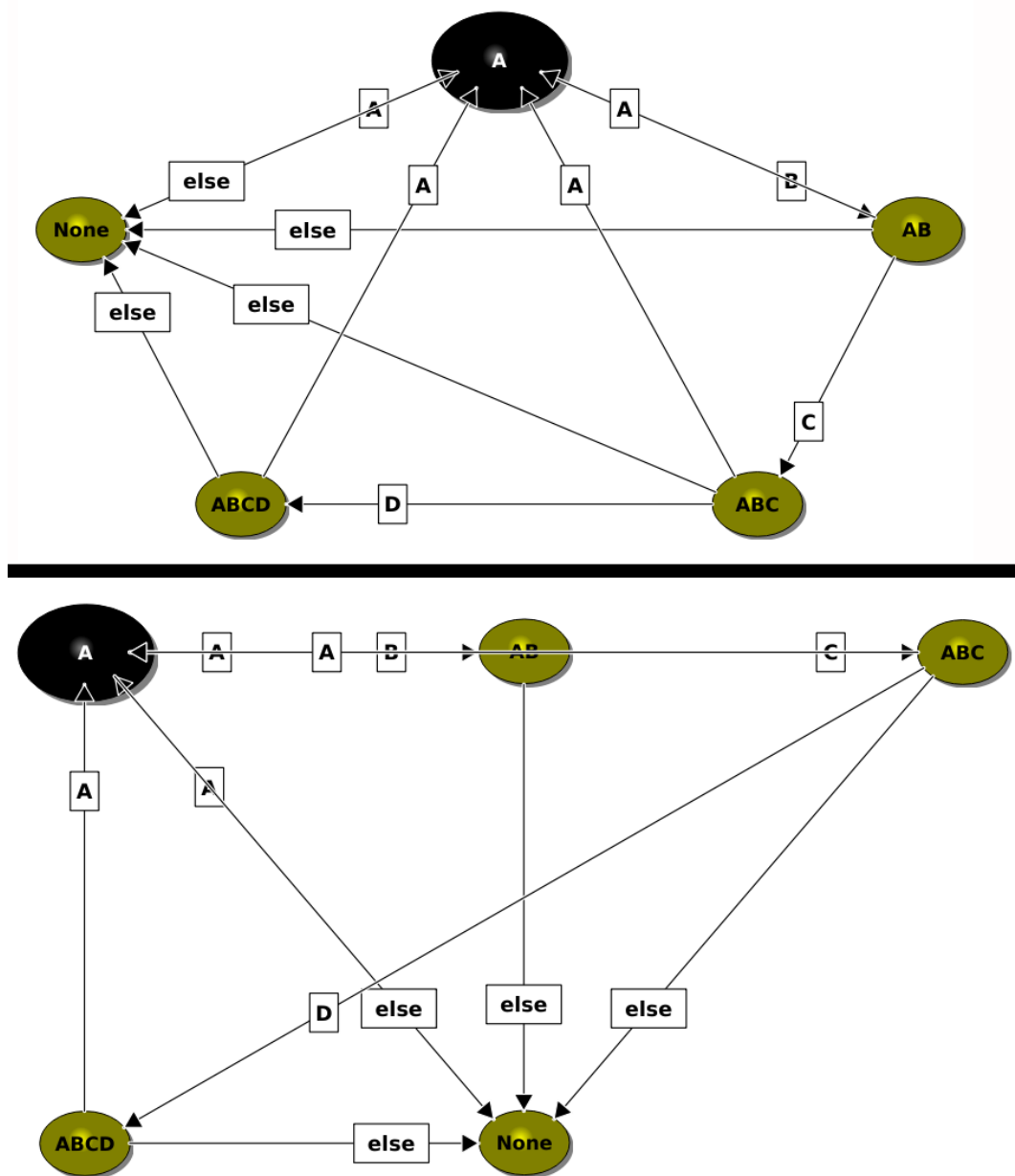
- Tryb krokowy
- Tryb normalny

W przypadku gdy wszystkie wejścia się wyczerpią aplikacja zakończy symulację. W trybie krokowym, aplikacja wykonuje jeden krok. Aby wykonać kolejny krok należy wybrać opcję 'Next step' z menu transducer. Aplikacja dodatkowo przełącza pokazywany transducer na ten w którym został wykonany krok. Stan w którym aktualnie znajduje się transducer jest zaznaczony jako duży czarny wierzchołek, natomiast stan poprzedni jest zaznaczony jako mały, czarny wierzchołek. Aby uruchomić tryb krokowy wystarczy nacisnąć f10 lub wybrać opcję 'Step forward' z menu 'Pipe'. W



trybie normalnym aplikacja natychmiast wykona wszystkie możliwe kroki, a następnie przełączy widok na ostatni transducer, który wykonał akcję. Aby uruchomić tryb normalny wystarczy nacisnąć f5 lub wybrać opcję 'Run' z menu 'Pipe'.

### 2.2.3. Widok



Rysunek 2.2: Porównanie rozmieszczenia na siatce i w wierzchołkach wielokąta foremnego

Aplikacja wyświetla aktualny stan FSM należący do transducera. Widok ten możemy przybliżać/oddalać za pomocą klawisza +/- lub wybierając odpowiednie opcje w menu View. Dodatkowo widok można przewijać za pomocą klawiszy W, S,

A, D - bardzo przydatna opcja w przypadku analizy dużych transducerów. Dodatkowo zaimplementowałem 3 proste algorytmy organizujące FSM:

- Organizuj w linii
- Organizuj na siatce
- Organizuj na wielokacie

Aby użyć algorytmu wystarczy wybrać odpowiednią opcję z zakładki 'Transducer'. Dla małych FSM z małymi stopniami wierzchołków najlepsze jest rozłożenie stanów jako wierzchołki wielokąta foremego. W przypadku większych FSM świetnie sprawdza się rozkładanie ich na planie siatki. Dodatkowo każdy z wierzchołków może zostać przesunięty na dowolną pozycję przy pomocy myszki.

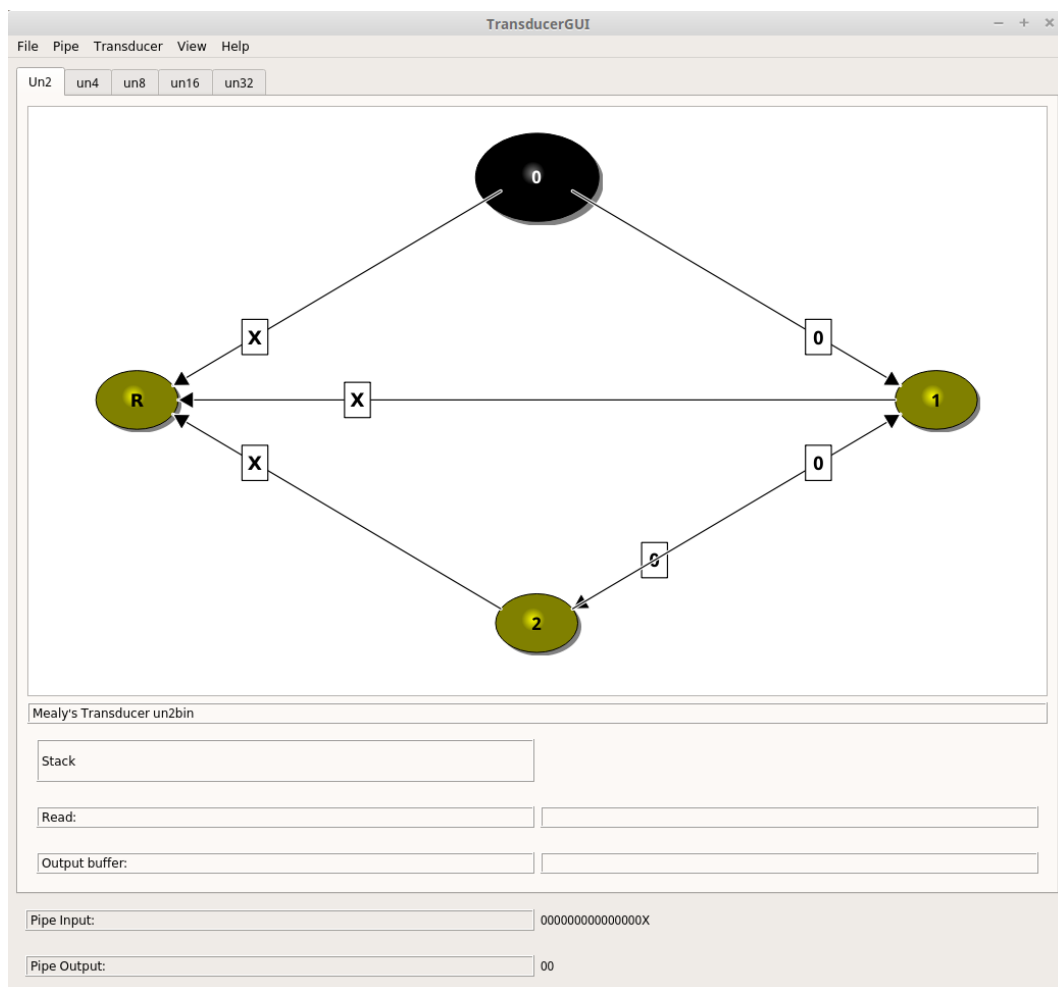
## 2.3. Przykłady

### 2.3.1. Konwersja z systemu unarnego na system binarny

Konwersja liczb ograniczonych przez pewną stałą  $N$  z systemu unarnego na system binarny, może odbyć się w prosty sposób za pomocą pojedynczego transducer'a posiadającego liniową liczbę stanów, jednak nie jest rozwiązanie efektywne pamięciowo. Prezentowane tu rozwiązanie, używa 6 transducerów posiadających po 4 stany dla konwersji liczb nie większych niż  $2^6$  z systemu unarnego do systemu dwójkowego - w ogólności potrzebujemy  $\log(N)$  transducerów by przekonwertować liczby z systemu unarnego do systemu binarnego. Oto opis pojedynczego transducera wraz z wizualizacją stanów: Dokładny opis transducer'a jest zapisany w katalogu Samples jako 'un2bin.json'. Aby przekonwertować daną liczbę musimy wpisać ją w systemie unarnym, a następnie napisać X jako znak końca liczby. Każdy transducer odczytuje postać unarną liczby, by następnie dopisać kolejny bit liczby w systemie binarnym i przepisać wcześniej obliczone już bity. Dodatkowo, każdy transducer skraca zapis unarny liczby dwukrotnie - dlatego potrzebujemy jedynie  $\log(N)$  transducerów. Za zmniejszoną złożoność pamięciową płacimy złożonością czasową - transducer wykonuje  $\log(N)$  przetworzeń wejścia. Widać, że w podobny sposób można stworzyć transducer konwertujący z dowolnego systemu liczbowego na dowolny inny w znacznie mniejszej niż liniowa pamięci.

### 2.3.2. Korekta literówek

Transducery najczęściej są używane do przetwarzania języków. Prostym przykładem transducerów może być aplikacja poprawiająca popularne błędy językowe. W tym przykładzie transducer poprawia dwa popularne błędy językowe:



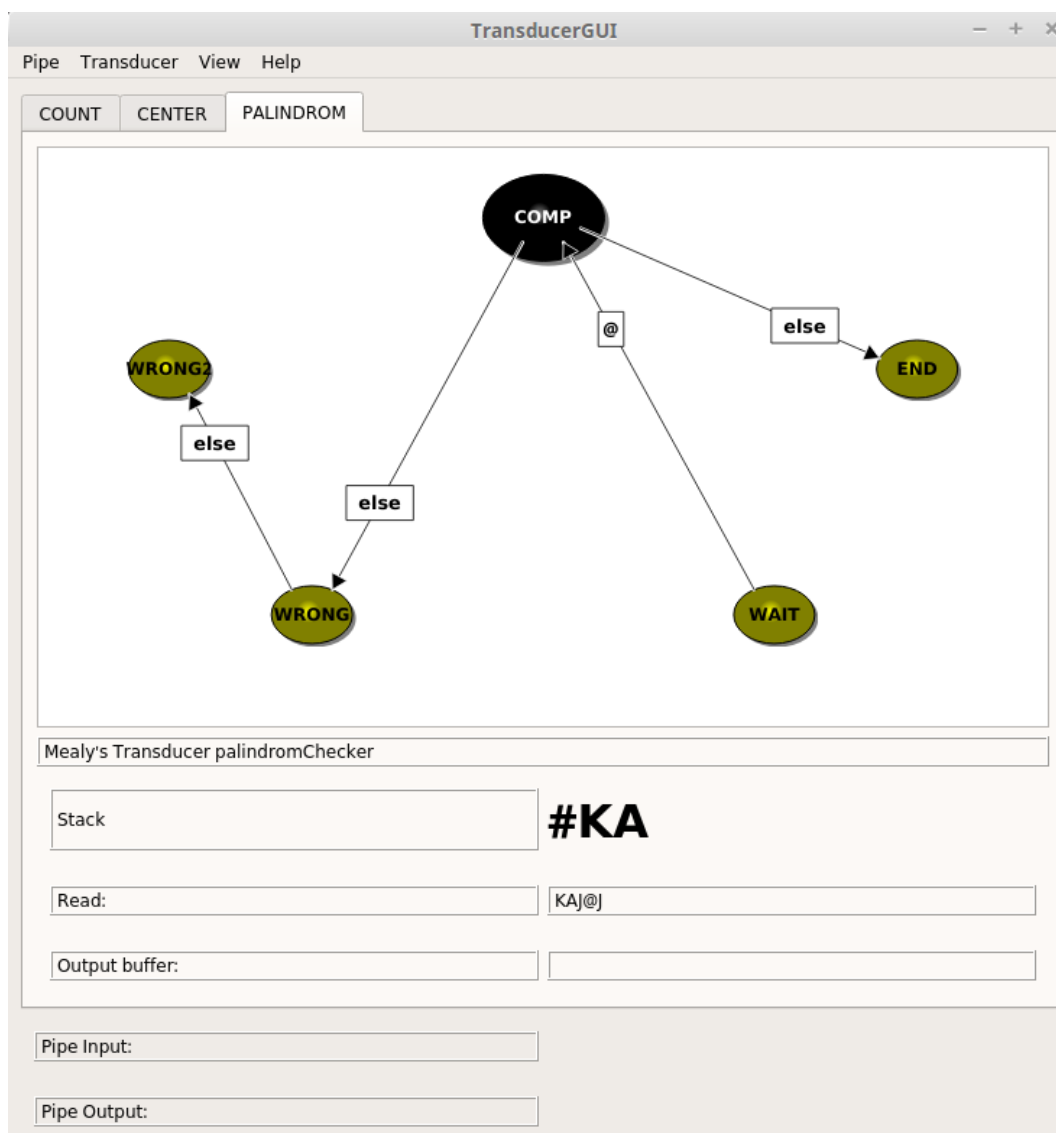
Rysunek 2.3: DFA należące do transducera konwertujące z systemu unarnego na binarny

- napewno  $\rightarrow$  na pewno
- nielada  $\rightarrow$  nie lada

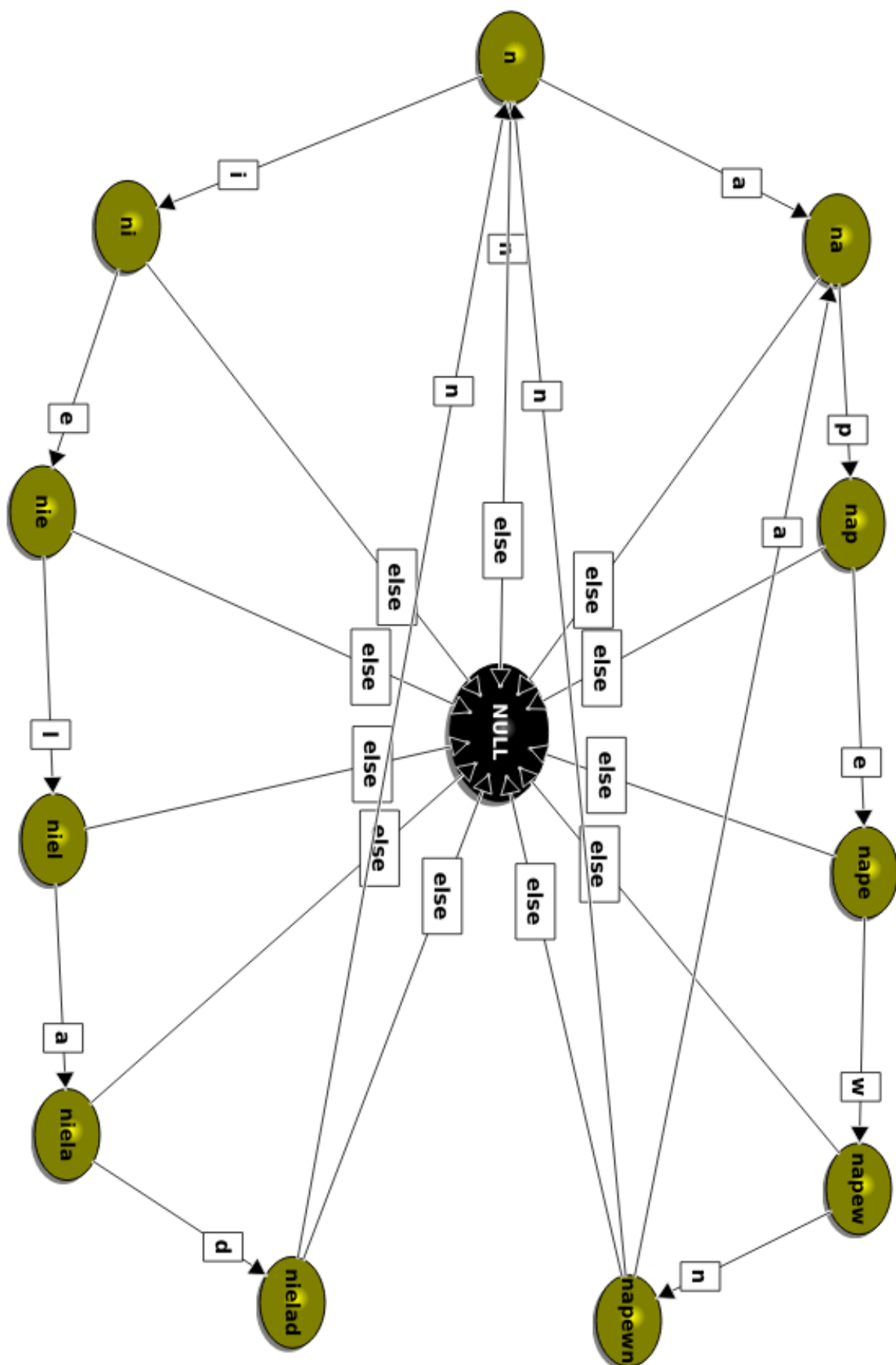
Model FSM ze względu na swój rozmiar znajduje się na następnej stronie. Dokładny opis przykładu został zapisany w katalogu Samples pod nazwą 'corrector.json'. Transducer wyszukuje jednego z wzorców z bazy, a następnie próbuje go zastąpić. Jeśli w trakcie przetwarzania, któryś z wzorców przestaje pasować wypisujemy wczytane znaki. Zaletą tego rozwiązania jest mała złożoność pamięciowa i czasowa - złożoność pamięciowa jest liniowa względem sumy długości wszystkich wzorców w słowniku, natomiast czasowa jest liniowa względem długości wczytywanego tekstu.

### 2.3.3. Proste wykrywanie parzystych palindromów

Transducery mogą być również wykorzystywane do upraszczania rozwiązań. Oto prosty przykład rozwiązywania problemu palindromów. Rozwiązanie korzysta z trzech transducerów. Pierwszy z nich oblicza połowę długości słowa w systemie unarnym, a następnie przepisuje słowo, oddzielając je specjalnym znakiem @. Kolejny transducer przepisuje słowo wstawiając w środku słowa znak @. Ostatni transducer dodaje kolejne litery na stos, a gdy osiągnie znak @ zaczyna porównywać litery ze stosu z literami słowa. W ten sposób potrafimy deterministycznie znaleźć środek, by następnie móc deterministycznie zweryfikować palindromiczność. Słowo musi zostać podane w postaci 'kandydat\_na\_palindrom@@'. Transducery są zapisane odpowiednio jako pliki 'count\_and\_reverse.json', 'find\_center.json' i 'palindromChecker.json' w katalogu 'Samples'.



Rysunek 2.4: Ostatni transducer w kolejce sprawdzający palindromiczność początkowego słowa



### Rysunek 2.5: DFA korektora językowego

## Rozdział 3.

# Podsumowanie

### 3.1. Zastosowania

Transducery mają wiele zastosowań w przetwarzaniu języka. Jednym z głównych zastosowań jest użycie ich w przetwarzaniu mowy [4]. Dzięki skończonemu automatu stanowemu mogą opisywać lokalne w gramatyce zjawiska w przystępnej formie - do pełnej analizy i korekty potrzebne są graficzne narzędzia. Kolejnym argumentem przemawiającym za użyciem transducerów jest efektywność - czas działania nie zależy od wielkości transducera, lecz od wielkości wejścia. Jednym z ciekawszych zastosowań jest użycie transducerów z wagami do reprezentacji ukrytych sieci Markova [3]. Poza pracami naukowymi, transducery zostały użyte w systemie rozpoznającym mowę stworzonym przez North American Business News [3]. Dzięki użyciu transducerów uzyskano rozwiązanie działające w czasie rzeczywistym, zajmujące niewiele więcej pamięci niż rozwiązanie korzystające z trigramów.

### 3.2. Dalszy rozwój

Ponieważ transducery są oparte na automatach skończonych można bardzo łatwo rozszerzać je o wszelkie mechanizmy z FSM, m.in.:

- transducery niedeterministyczne
- transducery operujące na drzewach zamiast słów
- transducery operujące na słowach skończonych
- transducery z wagami

Aby aplikacja stała się wygodniejsza, powinna zostać dodatkowo zaimplementowana funkcjonalność tworzenia projektów - każdorazowe dodawanie transducerów do kolejki oraz ustawianie wierzchołków w czytelnych miejscach nie jest wygodne. Ponie-

waż wszystko jest zapisywane w formacie JSON, implementacja tego rozwiązania nie powinna być skomplikowana.



# Bibliografia

- [1] *Repozytorium aplikacji* <https://github.com/bartek1912/transducerVisualizer>
- [2] *Wprowadzenie do teorii automatów, języków i obliczeń*. John E. Hopcroft, Jeffrey D. Ullman
- [3] *Weighted finite-state transducers in speech recognition*. Mehryar Mohri, Fernando Pereira and Michael Riley
- [4] *Finite-State Transducers in Language and Speech Processing*. Mehryar Mohri
- [5] *Opis projektu QT* <https://www.qt.io/company/>
- [6] *A Method for Synthesizing Sequential Circuits*. Mealy, George H. (1955)
- [7] *Gedanken-experiments on Sequential Machines*. Moore, Edward F (1956)
- [8] *An introduction to formal languages and automata*. Linz, Peter, (2011)