# Repairing Finite Automata with Missing Components

Naprawianie automatów skończonych
z brakującymi stanami i krawędziami

**Julia Cygan, Patryk Flama**

Bachelor of Engineering Thesis
**Supervisor**: PhD Jakub Michaliszyn

University of Wrocław
Faculty of Mathematics and Computer Science
Institute of Computer Science

2 stycznia 2026

**Julia Cygan**

. . . . . .
(adres zameldowania)

. . . . . .
(adres korespondencyjny)

PESEL: . . . . . .

e-mail: . . . . . .

Wydział Matematyki i Informatyki
stacjonarne studia I stopnia
kierunek: informatyka
nr albumu: . . . . . .

**Patryk Flama**

. . . . . .
(adres zameldowania)

. . . . . .
(adres korespondencyjny)

PESEL: . . . . . .

e-mail: . . . . . .

Wydział Matematyki i Informatyki
stacjonarne studia I stopnia
kierunek: informatyka
nr albumu: . . . . . .

## Oświadczenie o autorskim wykonaniu pracy dyplomowej

Niniejszym oświadczamy, że złożoną do oceny pracę zatytułowaną *Repairing Finite Automata with Missing Components* wykonaliśmy samodzielnie pod kierunkiem promotora, dr Jakuba Michaliszyna. Oświadczamy, że powyższe dane są zgodne ze stanem faktycznym i znane nam 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, 2 stycznia 2026

. . . . . .
(czytelny podpis)

. . . . . .
(czytelny podpis)

# Abstract

Niniejsza praca dotyczy zagadnienia naprawy uszkodzonego deterministycznego automatu skończonego na podstawie próbek pozytywnych i negatywnych. Rozważany automat może zawierać brakujące krawędzie lub stany, co uniemożliwia jego poprawne działanie. W pracy analizowana jest złożoność obliczeniowa problemu, w tym jego przynależność do klasy NP oraz własności parametryzowane w kontekście algorytmów FPT i hierarchii W. Zaproponowano algorytm rozwiązujący rozważany problem, przeprowadzono jego analizę teoretyczną oraz przedstawiono implementację wraz z omówieniem optymalizacji i heurystyk wpływających na czas działania.

---

This thesis addresses the problem of repairing a damaged deterministic finite automaton using positive and negative samples. The considered automaton may contain missing transitions or states, which prevents it from functioning correctly. The work analyzes the computational complexity of the problem, including its membership in the class NP and its parameterized properties in the context of FPT algorithms and the W-hierarchy. An algorithm for solving the problem is proposed, followed by its theoretical analysis, as well as an implementation accompanied by a discussion of optimizations and heuristics affecting the running time.

# Spis treści

# Rozdział 1

# Wstęp

Teoria automatów jest dziedziną informatyki teoretycznej, zajmująca się głównie badaniem abstrakcyjnych maszyn wykorzystywanych w celu modelowania obliczeń. Automat to model, który przetwarza dane wejściowe poprzez wykonywanie przejść pomiędzy kolejnymi stanami zgodnie ze zdefiniowanym sposobem reagowania na poszczególne symbole. Automat najczęściej definiowany jest jako graf z oznaczonymi krawędziami i określonymi wierzchołkami początku i końca. Jednym z najważniejszych i najczęściej badanych modeli są automaty skończone, które charakteryzują się skończonym zbiorem stanów.

Jedną z najważniejszych klas automatów skończonych są deterministyczne automaty skończone (ang. *Deterministic Finite Automata*, DFA). W modelu tym dla każdego stanu oraz symbolu alfabetu wejściowego zdefiniowane jest dokładnie jedno przejście do kolejnego stanu. Dzięki tej własności działanie automatu jest jednoznaczne i w pełni przewidywalne, co znacząco upraszcza jego analizę oraz implementację.

W kontekście uczenia automatów wyróżnia się dwa główne podejścia: uczenie aktywne i uczenie pasywne. Uczenie pasywne polega na konstruowaniu modelu automatu wyłącznie na podstawie gotowego zbioru przykładów wejściowych, bez możliwości zadawania dodatkowych pytań czy testowania hipotez w trakcie procesu uczenia. W praktyce oznacza to, że algorytm otrzymuje skończony zbiór słów oznaczonych jako akceptowane lub odrzucane i na tej podstawie próbuje odtworzyć strukturę automatu, który najlepiej odzwierciedla obserwowane zachowanie systemu. Podejście pasywne jest szczególnie użyteczne w sytuacjach, w których brak jest dostępu do „czarnej skrzynki" systemu lub gdy interaktywne testowanie wszystkich możliwych sekwencji wejściowych jest niemożliwe lub kosztowne. Pomimo swojej prostoty, uczenie pasywne wiąże się z szeregiem trudności teoretycznych i praktycznych, w tym ograniczeniem informacji wynikającym z niekompletności danych, możliwością istnienia wielu automatów zgodnych z tym samym zbiorem próbek oraz problemem minimalizacji otrzymanego modelu.

Jednym z fundamentalnych wyników w teorii pasywnego uczenia języków formalnych było wykazanie, że klasa języków regularnych nie jest identyfikowalna w granicy wyłącznie na podstawie pozytywnych przykładów **Language identification in the limit**. Istotnie ogranicza to możliwości pasywnego uczenia automatów skończonych bez dodatkowych założeń. Wynik ten miał istotny wpływ na dalszy rozwój wnioskowania gramatyk, wskazując na konieczność wykorzystywania zarówno przykładów pozytywnych, jak i negatywnych, bądź wprowadzania dodatkowych ograniczeń na strukturę danych uczących lub klasę rozważanych automatów.

W ostatnich latach problem pasywnego uczenia deterministycznych automatów skończonych pozostaje przedmiotem intensywnych badań. W jednej z nowszych prac autorzy

koncentrują się na formalnej analizie problemu DFA-consistency, czyli określenia, czy istnieje deterministyczny automat skończony, który akceptuje wszystkie pozytywne przykłady i odrzuca wszystkie negatywne przykłady dostarczone w zbiorze uczącym. Badając złożoność obliczeniową tego problemu oraz warianty wynikające z różnych ograniczeń na alfabet i strukturę danych uczących. Wykazano, że problem DFA-consistency jest NP-zupełny, nawet w przypadku alfabetów binarnych, co oznacza, że w ogólności nie istnieje znany algorytm wielomianowy rozwiązujący go dla wszystkich instancji **Learning from Positive and Ne**

Inne podejście prezentują prace, w których rekonstrukcja deterministycznych automatów skończonych realizowana jest poprzez redukcję problemu uczenia do problemów spełnialności logicznej (SAT). Przykładem takiego rozwiązania jest narzędzie DFAMiner **DFAMiner: Mining minimal separating DFAs from labelled samples**, które konstruuje automat pośredni w postaci trójwartościowego automatu skończonego (3DFA), zawierającego stany akceptujące, odrzucające oraz stany typu nieistotne, umożliwiające dokładne rozpoznanie dostarczonych przykładów uczących. Następnie automat ten jest minimalizowany poprzez redukcję do problemu SAT, co pozwala na uzyskanie minimalnego automatu separującego, czyli deterministycznego automatu skończonego o najmniejszej możliwej liczbie stanów, który akceptuje wszystkie przykłady pozytywne i jednocześnie odrzuca wszystkie przykłady negatywne. Tego rodzaju automat nie musi w pełni określać języka docelowego, lecz jedynie rozdzielać (separować) dostarczone zbiory próbek. Przeprowadzone badania empiryczne wskazują, że tego typu podejścia mogą znacząco przewyższać klasyczne metody uczenia pasywnego pod względem efektywności obliczeniowej. Jednocześnie skuteczność metod opartych na redukcji do SAT pozostaje silnie uzależniona od kompletności oraz spójności danych uczących, a w przypadku próbek niepełnych lub sprzecznych liczba potencjalnych modeli rośnie wykładniczo, co istotnie komplikuje proces uczenia.

Pomimo znacznego postępu w dziedzinie pasywnego uczenia DFA, większość istniejących metod zakłada, że automat uczony jest konstruowany od podstaw na podstawie zbioru przykładów. W praktycznych zastosowaniach często spotyka się jednak sytuacje, w których dostępny jest częściowo zdefiniowany automat, zawierający brakujące stany, niepełne przejścia lub fragmentaryczną wiedzę o strukturze systemu. Tego rodzaju przypadki pojawiają się m.in. w inżynierii odwrotnej, analizie dziedziczonych systemów, rekonstrukcji protokołów komunikacyjnych oraz w procesach naprawy modeli formalnych.

Celem niniejszej pracy jest analiza problemu naprawy brakujących deterministycznych automatów skończonych na podstawie skończonego zbioru przykładów pozytywnych i negatywnych. Przez brakujący deterministyczny automat skończony rozumiany jest automat, w którym zbiór stanów oraz część przejść są określone poprawnie, natomiast pozostałe przejścia nie zostały zdefiniowane. Naprawa automatu polega na uzupełnieniu brakujących przejść w taki sposób, aby otrzymany automat był deterministyczny oraz zgodny z dostarczonym zbiorem przykładów. W rozważanym problemie zakłada się, że automat wejściowy jest dany z góry, a zbiór przykładów pozytywnych i negatywnych jest niesprzeczny, tzn. istnieje co najmniej jeden deterministyczny automat skończony, który jest zgodny zarówno z istniejącą strukturą automatu, jak i z dostarczonymi danymi uczącymi.

W pracy skoncentrowano się na teoretycznej analizie złożoności obliczeniowej problemu naprawy brakujących deterministycznych automatów skończonych. W szczególności wykazane zostanie, że rozważany problem jest NP-zupełny, a ponadto omówiona zostanie jego trudność w sensie klas parametryzowanych W[1] oraz W[2], jak również jego przynależność do klasy W[p]. Oprócz wyników teoretycznych zaprezentowany zostanie algorytm, rozwiązujący ten problem w czasie lepszym niż podejście brute force, wykorzy-

stujący skoki przez zdefiniowane krawędzie. Skuteczność zaproponowanego rozwiązania zostanie porównana z heurystyką lokalnej naprawy automatów. Dodatkowo przeanalizowany zostanie wpływ struktury automatu oraz zastosowanych strategii algorytmicznych na czas działania proponowanych metod.

# Rozdział 2

# Teoria

## 2.1 Definicja problemu

### 2.1.1 Definicja problemu 1

### 2.1.2 Trywialność przypadku brakujących stanów

### 2.1.3 Definicja problemu 2

## 2.2 NP-zupełność

### 2.2.1 Przynależność do NP

### 2.2.2 NP-trudność

## 2.3 FPT

### 2.3.1 W[1]-trudność

### 2.3.2 W[2]-trudność

### 2.3.3 Przynależność do W[P]

# Rozdział 3

# Algorytmy

## 3.1 Algorytm ze skokami (Jump Tables)

Algorytm ze skokami stanowi optymalizację algorytmu pełnego przeszukiwania przestrzeni możliwych uzupełnień brakujących przejść deterministycznego automatu skończonego. Jego głównym celem jest zmniejszenie przestrzeni przeszukiwań poprzez eliminację powtarzalnych fragmentów symulacji próbek wejściowych.

Podstawową obserwacją wykorzystywaną przez algorytm jest fakt, że w trakcie przeszukiwania przestrzeni rozwiązań zmieniają się jedynie brakujące przejścia automatu, a brakujący automat dany na wejściu pozostaje niezmienny. W klasycznym podejściu brute force każda próba weryfikacji wymaga pełnej symulacji wszystkich próbek symbol po symbolu, co prowadzi do znacznej redundancji obliczeń.

W celu ograniczenia tego zjawiska algorytm wprowadza etap wstępnego przetwarzania w postaci tablic skoków. Dla każdej próbki oraz każdego stanu automatu obliczana jest informacja opisująca najdalszy fragment słowa, który może zostać przetworzony bez przechodzenia przez nieokreślone przejścia. Pozwala to na szybkie pomijanie fragmentów wejścia, które nie zależą od aktualnie testowanego uzupełnienia automatu.

#### Budowa tablic skoków

Tablice skoków budowane są niezależnie dla zbioru przykładów pozytywnych oraz negatywnych. Dla każdej próbki $w$ o długości maksymalnie $L$ konstruowana jest tablica $DP$, w której wpis $DP[i][q]$ opisuje efekt przetworzenia najdłuższego możliwego fragmentu próbki $w$, zaczynając od pozycji $i$ w stanie $q$, bez użycia brakujących przejść.

**Algorithm 1:** Budowa tablic skoków

**Input:** Automat $A = (Q, \Sigma, \delta, q_0, F)$, zbiór próbek $S$
**Output:** Tablica skoków $JT$
**foreach** *próbka* $w \in S$ **do**
  $L \leftarrow |w|$;
  Utwórz tablicę $DP[0 \dots L][0 \dots |Q| - 1]$;
  **foreach** *stan* $q \in Q$ **do**
    $DP[L][q] \leftarrow (q, L)$;
  **for** $i \leftarrow L - 1$ **to** $0$ **do**
    **foreach** *stan* $q \in Q$ **do**
      **if** $\delta(q, w[i])$ *jest nieokreślone* **then**
        $DP[i][q] \leftarrow (q, i)$;
      **else**
        $q' \leftarrow \delta(q, w[i])$;
        $DP[i][q] \leftarrow DP[i + 1][q']$;
  Dodaj $DP$ do $JT$;
**return** $JT$

### Walidacja automatu z użyciem tablic skoków

Po skonstruowaniu tablic skoków algorytm wykorzystuje je podczas walidacji każdego kandydata. Zamiast symulować próbki krok po kroku, algorytm wykonuje skoki pomiędzy pozycjami, aż napotka fragment zależny od brakującego przejścia.

**Algorithm 2:** Walidacja automatu z wykorzystaniem tablic skoków

**Input:** Automat $A$, próbki $S$, tablica skoków $JT$, oczekiwany wynik $b$
**Output:** true jeśli automat jest zgodny z próbkami, false w przeciwnym razie
**foreach** *próbka* $w_i \in S$ **do**
  $q \leftarrow q_0$, $pos \leftarrow 0$;
  **while** $pos < |w_i|$ **do**
    $(q', pos') \leftarrow JT[i][pos][q]$;
    **if** $(q', pos') = (q, pos)$ **then**
      **if** $\delta(q, w_i[pos])$ *jest nieokreślone* **then**
        Przerwij symulację tej próbki;
      $q \leftarrow \delta(q, w_i[pos])$;
      $pos \leftarrow pos + 1$;
    **else**
      $q \leftarrow q'$, $pos \leftarrow pos'$;
  **if** $(q \in F) \neq b$ **then**
    **return** *false*;
**return** *true*

### Integracja z algorytmem pełnego przeszukiwania

Algorytm ze skokami nie modyfikuje samej strategii przeszukiwania przestrzeni możliwych uzupełnień brakujących przejść. Zastępuje on jedynie klasyczną procedurę wali-

dacji automatu zoptymalizowaną wersją wykorzystującą tablice skoków. Dzięki temu zachowana zostaje pełna poprawność algorytmu brute force, przy jednoczesnym istotnym zmniejszeniu czasu weryfikacji pojedynczego kandydata w praktyce.

# Dodatek A

# Aneks

append an x