# Prover - automatyzacja procesu wnioskowania logicznego z wykorzystaniem drzew prawdy

Stanisław Maciąg, Piotr Mitana

2014

# 1 Założenia projektowe

W ramach projektu zaprojektowana oraz zaimplementowana zostanie aplikacja, realizująca proces wnioskowania na podstawie dostarczonych przez użytkownika formuł logicznych stanowiących przesłanki oraz konkluzje. Poprawność wnioskowania sprawdzana będzie z wykorzystaniem algorytmu operującego na drzewach prawdy (ang. *Truth Trees*).

## 1.1 Środowisko

1. System operacyjny Linux
2. Języki programowania C++ oraz D

## 1.2 Implementowane funkcjonalności

1. Graficzny interfejs użytkownika (*GUI*), umożliwiający wygodną obsługę aplikacji
2. Dane wejściowe (formuły logiczne) wprowadzane ręcznie w postaci ciągu znaków, lub z pliku tekstowego (składnia wyrażeń opisana w 2.1)
3. Zapis danych wejściowych, otrzymanego drzewa prawdy oraz wyniku do pliku
4. Wizualizacja drzewa prawdy

# 2 Analiza leksykalna

## 2.1 Składnia formuły wejściowej

| **Ciąg znaków** | **Znaczenie** |
|---|---|
| Ciąg znaków alfanumerycznych | Zmienna logiczna rozumiana jako formuła atomowa |
| ![formuła] | Jednoargumentowy operator negacji |
| [formuła] & [formuła] | Dwuargumentowy operator koniunkcji logicznej |
| [formuła] \| [formuła] | Dwuargumentowy operator alternatywy logicznej |
| [formuła] ^ [formuła] | Dwuargumentowy operator alternatywy wykluczającej |
| [formuła] > [formuła] | Dwuargumentowy operator implikacji logicznej |
| [formuła] = [formuła] | Dwuargumentowy operator ekwiwalencji logicznej |
| ([formuła]) | Nawiasy okrągłe - grupowanie wyrażeń |

## 2.2 Hierarchia operatorów

| **Operator** | **Priorytet** |
|---|---|
| Negacja | Najwyższy |
| Koniunkcja, alternatywa, alternatywa wykluczająca | Średni |
| Implikacja, Ekwiwalencja | Najniższy |

## 2.3 Reguły wydzielania tokenów

1. Białe znaki rozdzielające tokeny są ignorowane, jednak ich wystąpienie powoduje natychmiastowe odcięcie tokena od reszty formuły (w wypadku poprawnej składniowo formuły nie wpłynie to na sposób jej podziału)

2. Nawiasy oraz operator negacji są samodzielnymi tokenami, więc są natychmiastowo odcinane i zwracane

3. Ciąg znaków niealfanumerycznych (z wyłączeniem białych znaków, nawiasów i wykrzyknika) zostanie zwrócony jako jeden token (operator)

4. Ciąg znaków alfanumerycznych zostanie zwrócony jako jeden token (zmienna)

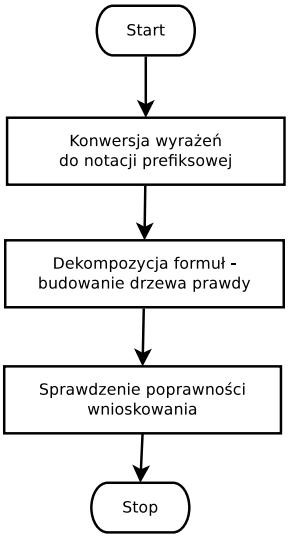
## 2.4 Reguły walidacji

Definiowane są w następujący sposób – dla każdego elementu składniowego określane jest, czy może stać na początku oraz jakie elementy dozwolone są za nim.

| **Element języka** | **Może zaczynać formułę** | Dozwolone następne tokeny |
|---|---|---|
| Zmienna logiczna | Tak | &, \|, ^, >, =, ), koniec formuły |
| ! | Tak | Zmienna logiczna, !, ( |
| &, \|, ^, >, = | Nie | Zmienna logiczna, !, ( |
| ( | Tak | Zmienna logiczna, !, ( |
| ) | Nie | &, \|, ^, >, =, ), koniec formuły |

Dodatkowo, ilości nawiasów otwierających i zamykających muszą być równe.

# 3 Algorytm działania




Rysunek 1: Przebieg głównego algorytmu

## 3.1 Konwersja do notacji prefiksowej

Przed procesem dekompozycji formuła zostaje przetworzona do postaci prefiksowej (będącej de facto prostą reprezentacją drzewa), w której tokeny oddzielone są pojedynczą spacją. Proces ten należy poprzedzić walidacją formuły, ponieważ funkcja budująca drzewo już zakłada jej poprawność.

### 3.1.1 Walidacja

Proces walidacji polega na sprawdzeniu sąsiedztwa każdego z tokenów zgodnie z regułami podanymi wcześniej. Faktycznie są one jednak sprawdzane pośrednio – funkcja walidująca sprawdza poprzedzający operator, a nie następny (słowo a może stać przed słowem b wtedy i tylko wtedy, gdy słowo b może stać za słowem a). Dodatkowe reguły:

- Na początku procesu zmienna przechowująca poprzedni token jest pusta – sprawdzenie, czy dany token może zaczynać formułę sprowadzone jest do sprawdzenia, czy może być poprzedzone przez pusty token
- Koniec formuły wykryty jest, gdy zostanie wczytany pusty token. Następuje więc sprawdzenie, czy pusty token może być poprzedzane przez poprzedni (będący ostatnim w formule)

- Na końcu procesu walidacji sprawdzany jest balans nawiasów

### 3.1.2 Algorytm konwersji

Proces ten dokonywany jest przy użyciu dwóch stosów – operatorów oraz operandów. Stosy są reprezentowane przez ciągi znaków, których elementy oddzielone są tabulacjami. Poniżej wypisane są kroki algorytmu.

1. Pobieraj kolejne słowa formuły, dla każdego wykonaj:
   - 1.1 Jeżeli „(", odłóż na stos operatorów
   - 1.2 Jeżeli „)", to:
     - 1.2.1 Zdejmuj ze stosu operatorów tak długo, aż napotkasz „("
     - 1.2.2 Dla każdego operatora zdejmij właściwą ilość argumentów ze stosu operandów Odłóż na stos operandów wartość zawierającą operator oraz jego argumenty oddzielone spacjami
   - 1.3 Jeżeli inny operator (przyjmijmy $a$), to:
     - 1.3.1 Zdejmuj ze stosu operatorów (przyjmijmy $b$), dopóki operator $b$ ma taki sam lub większy priorytet od $a$. Dla każdego wykonaj:
       - 1.3.1.1 Zdejmij właściwą dla operatora ilość argumentów ze stosu operandów
       - 1.3.1.2 Odłóż na stos operandów wartość zawierającą operator oraz jego argumenty oddzielone spacjami
     - 1.3.2 Odłóż $b$ na stos operatorów ($b$ będzie pierwszym o priorytecie mniejszym od $a$)
     - 1.3.3 Odłóż $a$ na stos operatorów
   - 1.4 W innym wypadku (czyli nazwa zmiennej) odłóż na stos operandów
2. Pobierz i zwróć wynik ze stosu operandów

## 3.2 Dekompozycja formuły

### 3.2.1 Drzewa prawdy

Drzewa prawdy stanowią jedno z narzędzi służących do rozwiązywania problemów logicznych. Ich główną zaletą jest duża efektywność, przekładająca się na mniejszą złożoność czasową. Metody wykorzystujące drzewa prawdy nie wymagają rozwiązywania trudnych problemów decyzyjnych, dzięki czemu

są dobrze algorytmizowalne. Poniżej przedstawiono listę problemów, które można rozwiązać korzystając z drzew prawdy:

- Dowodzenie sprzeczności lub niesprzeczności zbioru twierdzeń
- Sprawdzanie poprawności dowodu (wnioskowania)
- Wykrywanie tautologii i kontrtautologii
- Sprawdzenie semantycznej równoważności

Bez względu na charakter analizowanego problemu, rozwiązanie zawsze uzyskuje się w trzech następujących krokach:

1. Zdefiniowanie korzenia drzewa, zgodnie z typem rozpatrywanego problemu
2. Budowanie drzewa poprzez zastosowanie reguł dekompozycji do złożonych wyrażeń logicznych (przedstawione w 3.2.2
3. Interpretacja wyników

### 3.2.2 Reguły dekompozycji

| Postać dekompozycji | Nazwa |
|---|---|
| $p \wedge q$<br>$\vert$<br>$p$<br>$q$ | Koniunkcja |
| $p \vee q$<br>$\diagup\diagdown$<br>$p \quad q$ | Alternatywa |
| $p \Rightarrow q$<br>$\diagup\diagdown$<br>$\neg p \quad q$ | Implikacja |
| $p \Leftrightarrow q$<br>$\diagup\diagdown$<br>$p \quad \neg p$<br>$q \quad \neg q$ | Ekwiwalencja |
| $\neg(p \wedge q)$<br>$\diagup\diagdown$<br>$\neg p \quad \neg q$ | Negacja koniunkcji |
| $\neg(p \vee q)$<br>$\vert$<br>$\neg p$<br>$\neg q$ | Negacja alternatywy |
| $\neg(p \Rightarrow q)$<br>$\vert$<br>$p$<br>$\neg q$ | Negacja implikacji |
| $\neg(p \Leftrightarrow q)$<br>$\diagup\diagdown$<br>$p \quad \neg p$<br>$\neg q \quad q$ | Negacja ekwiwalencji |
| $\neg\neg p$<br>$\vert$<br>$p$ | Negacja negacji |

### 3.2.3 Algorytm sprawdzania poprawności dowodu

1. Tworzenie korzenia drzewa poprzez zapis przesłanek oraz *zaprzeczenia* konkluzji w postaci ciągu wyrażeń (kolumny) o numerowanych (rozróżnialnych) wierszach

2. Dekompozycja następnego w kolejności wyrażenia logicznego. Kolejność dekomponowania wyrażeń znajdujących się w węzłach drzewa jest dowolna, jednak ze względu efektywność algorytmu najlepiej przyjąć następujący porządek:

    1 Dekomponowanie wyrażeń niepowodujących rozgałęziania - koniunkcja, zaprzeczenie alternatywy oraz zaprzeczenie implikacji

    2 Dekomponowanie wyrażeń powodujących rozgałęzianie oraz tworzących dwa pod-wyrażenia w węzłach potomnych - ekwiwalencja oraz zaprzeczenie ekwiwalencji

    3 Dekomponowanie wyrażeń powodujących rozgałęzianie oraz tworzących jedno wyrażenie w węzłach potomnych - zaprzeczenie koniunkcji, alternatywa, implikacja

3. Dekompozycję powtarza się, dopóki w liściach drzewa nie pozostaną same pojedyncze zmienne logiczne bądź ich zaprzeczenia

4. Zamykanie gałęzi - gałąź drzewa określa się jako zamkniętą, jeśli na ścieżce od korzenia drzewa do liścia występuje pojedyncza zmienna logiczna oraz jej zaprzeczenie

5. Interpretacja wyników - jeżeli drzewo posiada przynajmniej jedną otwartą gałąź to wnioskowanie jest niepoprawne

### 3.2.4 Programowa realizacja dekompozycji wyrażeń

W zastosowanym przez nas algorytmie postać prefiksowa zostanie przekształcona do postaci właściwego drzewa, którego węzły będą przechowywały tablice napisów zawierające kolejne zdania logiczne w postaci prefiksowej. Poniżej zaprezentowano fragment algorytmu wykonujący jeden krok dekompozycji pojedynczej formuły:

1. Jeżeli słowo jest zmienną, wróć

2. Jeżeli jest operatorem negacji, pobierz następne słowo (będzie operatorem) i połącz z poprzednim

3. Każdy z argumentów operatora pobierz w następujący sposób:

3.1 Ustal ilość wymaganych symboli na 1

3.2 Pobieraj kolejny symbol, dopóki ilość wymaganych symboli jest większa od 0. Dla każdego wykonuj:

3.2.1 Jeżeli symbol jest zmienną, zmniejsz ilość wymaganych symboli o 1

3.2.2 Jeżeli symbol jest operatorem, zwiększ ilość wymaganych symboli o ilość argumentów tego operatora - 1

3.3 Argumentem są wszystkie pobrane symbole w kolejności ich występowania

4. Zgodnie z regułami dekompozycji utwórz jeden lub dwa węzły potomne i przypisz im odpowiednie wartości