UMCS

UNIWERSYTET MARII CURIE-SKŁODOWSKIEJ
W LUBLINIE
Wydział Matematyki, Fizyki i Informatyki

Kierunek: **Informatyka**

**Patryk Popek**
nr albumu: 303843

# Rozwiązywanie równań i układów równań nad skończonymi strukturami algebraicznymi z wykorzystaniem SAT-solvera.

Solving equations and system of equations over finte algebraic structures using a SAT-solver

Praca licencjacka
napisana w Katedrze Systemów Inteligentnych
pod kierunkiem doktora Jacka Krzaczkowskiego

**Lublin rok 2023**

# Spis treści

# Wstęp

SAT jest problemem należącym do klasy NP. Jest to ważny problem, ponieważ wszystkie problemy należące do tej klasy dają się do niego sprowadzić. Dlatego powstają specjalne programy, noszące nazwę SAT-solverów, dedykowane do rozwiązywania SAT-a. Programów tych powstaje na tyle dużo, że co roku organizowany jest konkurs [1] wyłaniający najlepszy z nich.

Celem niniejszej pracy było napisanie aplikacji, która rozwiązywałaby układy równań nad algebrą podaną w formacie *UACalc* za pomocą narzędzia Sat-Solver, poprzez zredukowanie tego problemu do problemu *SAT*.

W rozdziale pierwszym opisane zostały pojęcia teoretyczne, które są używane w dalszej części pracy. Omawiane są tam pojęcia związane z algebrą oraz z użytymi podczas redukcji formułami logicznymi. Rozdział drugi zawiera opis redukcji układu równań do formuły w formie CNF. Przedstawione zostały tam poszczególne kroki, za pomocą których redukcja została wykonana. Rozdział trzeci dotyczy aplikacji. Omawiane są tam zagadnienia techniczne związane z pracą, takie jak formaty plików użytych w trakcie implementacji, logika programu oraz testy wykonane w celu sprawdzenia działania programu.

# 1 Wprowadzenie

Poniżej przedstawione są pojęcia związanie z pracą. Są one zgodne z książką pod tytułem "A Course in Universal Algebra" autorstwa Stanley'a Burrisa oraz H. P. Sankappanavar [2]

## 1.1 Algebra

Zbiór $\beta$, nazywany **językiem**, jest zbiorem **symboli funkcyjnych**, takich że każdemu elementowi $f \in \beta$ przypisywana jest liczba naturalna $n$. Liczba ta nazywana jest **arnością** elementu *f*. Wtedy $f$ jest n-arnym symbolem funkcyjnym. Jeżeli $\beta$ jest językiem algebry, to **algebrą** nad językiem $\beta$ nazywamy uporządkowaną parę $\{A, F\}$, gdzie $A$ jest niepustym zbiorem, noszącym nazwę **uniwersum** oraz $F$ jest zbiorem **operacji** o skończonej arności, indeksowanych przez język $\beta$, takich że każdemu symbolowi funkcyjnemu $f \in \beta$ odpowiada n-arna operacja $f^A \in F$ przekształcająca $\{A^n \to A\}$. **Tabela operacji** to tabela przechowująca wyniki danej operacji dla wszystkich możliwych wartości argumentów. Każdy element $a \in A$ oraz zmienna jest **wielomianem** nad algebrą. Wyrażenie postaci $f(t_1, ..., t_n)$ gdzie f jest symbolem funkcyjnym oraz $t_i$ są poprawnymi wielomianami, też jest wielomianem. Wyrażenie $w_1 = w_2$ gdzie $w_1, w_2$ to wielomiany zdefiniowane nad algebrą to **równanie**. Równanie postaci $t_1(x_1, ..., x_n) = t_2(x_1, ..., x_n)$ nazywamy **spełnialnym**, jeżeli istnieje wartościowanie $(a_1, ..., a_n) \in A^n$ takie że, $t_1(a_1, ..., a_n) = t_2(a_1, ..., a_n)$. Metoda zapisu wielomianu, w której najpierw zapisywany jest symbol funkcyjny, a po nim argumenty nazywany jest **Notacją Polską**. Przykładowo zapis $/ + xy - ab$ jest równoważny zapisowi $(x+y)/(a+b)$. Notacja Polska umożliwia beznawiasowy zapis wyrażenia.

## 1.2 Sat i spełnialność

**Zmienna zdaniowa** jest symbolem przechowującym wartość logiczną w logice rachunków zdań. Przyjmuje wartości 0 (fałsz) lub 1 (prawda). **Operatorem logicznym** nazywamy operację zwracającą wartość logiczną, której argumentami są wartości logiczne. Wyrażenie składające się ze zmiennych zdaniowych połączonych operatorami to **formuła logiczna**. Operatorami logicznymi są między innymi: **negacja**, **koniunkcja**, **alternatywa** i **implikacja**. **Literał** jest to zmienna zdaniowa lub jej zaprzeczenie. **Klauzula** jest alternatywą literałów dowolnej długości. Jeżeli w klauzuli występuje literał oraz jego

zaprzeczenie, to klauzula jest spełniona niezależnie od wartości reszty literałów. Klauzula posiadająca co najwyżej jeden literał niezanegowany nazywana jest **klauzulą Horna**. Powstała ona z przekształcenia implikacji postaci $((x_1 \wedge ... \wedge x_n) \rightarrow w)$ **Wartościowanie** to operacja przypisująca każdej zmiennej zdaniowej w formule wartość logiczną $\{0, 1\}$. Jeżeli istnieje wartościowanie, dla którego dana formuła jest spełniona, to jest to formuła **spełnialna**.

**SAT** [3] jest to problem decyzyjny należący do klasy NP dotyczący formuł logicznych. Odpowiada on na pytanie, czy dana formuła logiczna jest spełnialna. Często oprócz informacji o spełnialności, potrzebna jest informacja o wartościowaniu spełniającym daną formułę. Popularnym rodzajem formuł, dla których sprawdzany jest powyższy problem to formuły w **koniunkcyjnej postaci normalnej (CNF)**, czyli formuły logiczne będące koniunkcją klauzul dowolnej długości. Aby taka formuła była spełniona to z definicji koniunkcji każda klauzula w formule też musi być spełniona.

## 1.3 Cel pracy

Celem pracy było zaprojektowanie oraz zaimplementowanie aplikacji, na którą składa się graficzny interfejs użytkownika oraz algorytm redukujący problem spełnialności układów równań wielomianów do problemu *SAT*. Cały proces, w założeniach, miał przebiegać nad algebrą skończoną podaną do aplikacji w popularnym formacie *UACalc*, będącym standardem wśród algebraików zajmujących się badaniem struktur algebraicznych. Równania matematyczne miałyby być podawane do programu przez użytkownika z klawiatury. Otrzymana instancja problemu *SAT* miała by być rozwiązywana za pomocą *SAT-solvera*.

# 2 Redukcja

## 2.1 Klauzule redukcji

Wstępnym krokiem redukcji jest uproszczenie podanego do aplikacji układu równań do postaci, w której pojedyncze równanie ma formę: $f(t_1, ..., t_m) = w$. W powyższym zapisie $f$ jest symbolem funkcyjnym, $m$ jego arnością, zaś $\{t_1, ..., t_m, w\}$ są to zmienne równania lub stałe algebry. Jeżeli równanie podane w rozpatrywanym na wejściu układzie równań ma postać: $f_1(t_1, ..., t_n) = f_2(t_1, ..., t_n)$, to przekształcane jest ono do postaci $f_1(t_1, ..., t_n) = w_i$ oraz $f_2(t_1, ..., t_n) = w_i.$, gdzie $w_i$ jest nową zmienną, przechowującą wynik operacji. Jeżeli zaś równanie jest postaci: $f_1(t_1, ..., t_{i-1}\ f_2(v_1, ..., v_n), t_{i+1}, ..., t_m) = w_k$ to postać ta zamieniana jest na: $f_2(v_1, ..., v_n) = w_j$ oraz $f_1(t_1, ..., t_{i-1}, w_j, t_{i+1}, ..., t_m) = w_k$.

Dla każdej, pojedynczej, unikalnej zmiennej, znajdującej się w nowo powstałym układzie, tworzony jest zbiór zmiennych zdaniowych $Z_i = \{x_i^0, ..., x_i^{m-1}\}$, gdzie $m$ oznacza liczność uniwersum wczytanej algebry, zaś $i$ oznacza indeks zmiennej równania, dla której utworzono zbiór. Zmiennej zdaniowej $x_i^j$ przypisywana jest wartość logiczna *prawda* wtedy i tylko wtedy, gdy zmiennej równania $x_i$ została przypisana wartość $j$ z uniwersum algebry. Formuła w postaci CNF, która jest wynikiem redukcji, tworzona jest za pomocą dwóch typów klauzul opisanych poniżej.

1. Pierwszy typ klauzul ma za zadanie zapewnić dokładnie jedną wartość logiczną *prawda* w zbiorze zmiennych zdaniowych $Z_i$ . Efekt ten uzyskiwany jest poprzez połączenie dwóch rodzajów klauzul. Pierwszy rodzaj klauzul wymusza co najmniej jedną wartość *prawda*, drugi zaś wymusza co najwyżej jedną wartość *prawda*. Pierwszy rodzaj klauzul ma postać: $(x_i^0 \vee ... \vee x_i^{m-1})$, gdzie m jest licznością uniwersum algebry. Jest to alternatywa wszystkich zmiennych zdaniowych znajdujących się w zbiorze $Z_i$. Aby klauzula ta była spełniona, to z definicji alternatywy, co najmniej jedna zmienna zdaniowa $x_i^j$ musi być spełniona. Drugi rodzaj klauzul ma postać: $(\neg x_i^j \vee \neg x_i^k)$ gdzie $k, j \in \{0, ..., m-1\}$ i $k \neq j$. Klauzula taka jest tworzona dla każdego możliwego dwuelementowego podzbioru $\{x_i^j, x_i^k\}$ zmiennych zdaniowych znajdujących się w zbiorze $Z_i$. Aby klauzula ta była spełniona, to z definicji koniunkcji, obie zmienne zdaniowe znajdujące się w klauzuli nie mogą być jednocześnie spełnione. Dlatego połączenie tych dwóch typów klauzul zapewnia dokładnie jedno wystąpienie wartości *prawda* w zbiorze zmiennych zdaniowych $Z_i$

2. Następnym krokiem redukcji, dla każdego równania postaci $f(x_1, ..., x_n) = w_j$ znajdującego się w układzie równań, jest wygenerowanie wszystkich możliwych wartościowań zmiennych w zakresie uniwersum algebry. Dla każdego znalezionego wartościowania $(a_1, ..., a_n) \in A^n$, takiego że $f(a_1, ..., a_n) = k$, tworzona jest implikacja postaci: $((x_1 = a_1 \wedge ... \wedge x_n = a_n) \rightarrow w_j = k)$. Implikacja ta przekształcana jest do **klauzuli Horna** postaci: $(\neg x_1^{a_1} \vee ... \vee \neg x_n^{a_n} \vee w_l^k)$ gdzie $w_l$ to zmienna przechowująca wynik rozpatrywanej operacji. Jeżeli argumentem była stała algebry, to jest ona pomijana w zapisie klauzuli.

## 2.2 Przykładowa redukcja

Poniżej została przedstawiona przykładowa redukcja równania w grupie dodawania modulo 4. Uniwersum algebry w tym przypadku to $\{0, 1, 2, 3\}$ czyli jej liczność to 4. Arność działania wynosi 2. Przykładowe równanie będzie miało postać: $d(2, d(0, x_1)) = d(x_2, 3)$

| x/y | 0 | 1 | 2 | 3 |
|---|---|---|---|---|
| 0 | 0 | 1 | 2 | 3 |
| 1 | 1 | 2 | 3 | 0 |
| 2 | 2 | 3 | 0 | 1 |
| 3 | 3 | 0 | 1 | 2 |

Tabela 1: Tabela reprezentująca dodawanie modulo 4

Krokiem wstępnym do wykonania redukcji jest utworzenie układu równań na podstawie napotkanych symboli funkcyjnych w podanym zapisie równania. Układ ten będzie miał postać: { [ d(2,$w_1$) = $w_0$ ], [ $d(0, x_1) = w_1$ ], [ $d(x_2, 3) = w_0$ ]}. Powstały układ składa się z 3 równań. Posiada on cztery zmienne: dwie będące zmiennymi równania oraz dwie przechowujące wyniki operacji.

Kolejnym krokiem redukcji jest stworzenie zbiorów zmiennych zdaniowych.Przykładowo, dla zmiennej $x_i$, gdzie $i \in \{1, 2\}$ tworzony jest zbiór $Z_i = \{x\,{}^0_i, x\,{}^1_i, x\,{}^2_i, x\,{}^3_0\}$. Następnie budowana jest alternatywa wszystkich zmiennych zdaniowych ze zbioru: $(x\,{}^0_i \vee x\,{}^1_i \vee x\,{}^2_i \vee x\,{}^3_i)$. Na końcu tworzone są zanegowane pary: $(\neg x\,{}^0_i \vee \neg x\,{}^1_i)$, $(\neg x\,{}^0_i \vee \neg x\,{}^2_i)$, $(\neg x\,{}^0_i \vee \neg x\,{}^3_i)$, $(\neg x\,{}^1_i \vee \neg x\,{}^2_i)$, $(\neg x\,{}^1_i \vee \neg x\,{}^3_i)$, $(\neg x\,{}^2_i \vee \neg x\,{}^3_i)$.

Ostatnim krokiem redukcji jest stworzenie klauzul Horna. Redukcja ta zostanie pokazana dla równania [$d(x_2, 3) = w_0$ ]. Funkcja ta posiada tylko jeden argument, będący zmienną. W pierwszej kolejności tworzone są implikacje na podstawie znalezionych wartościowań zmiennych równania. Implikacje te będą wyglądały następująco:

$[(x_2 = 0 \wedge 3 = 3) \rightarrow w_0 = 3]$, $[(x_2 = 1 \wedge 3 = 3) \rightarrow w_0 = 0]$, $[(x_2 = 2 \wedge 3 = 3) \rightarrow w_0 = 1]$, $[(x_2 = 3 \wedge 3 = 3) \rightarrow w_0 = 2]$. Z powyższych implikacji usuwane są stałe i formuły te przekształcane są do postaci wynikowych klauzul, wyglądających w tym przypadku następującą: $(\neg x\,{}^{0}_{2} \vee w\,{}^{3}_{0})$ , $(\neg x\,{}^{1}_{2} \vee w\,{}^{0}_{0})$, $(\neg x\,{}^{2}_{2} \vee w\,{}^{1}_{0})$, $(\neg x\,{}^{3}_{2} \vee w\,{}^{2}_{0})$.

# 3 Aplikacja

## 3.1 Formaty użytych plików

*UACalc*[4] jest ogólnodostępnym oprogramowaniem do wykonywania obliczeń w dziedzinie algebry uniwersalnej. Służy on do badania i dowodzenia własności struktur algebraicznych. Przyjmuje on algebrę w plikach o rozszerzeniu *.ua*.

```
<?xml version='1.0' encoding='utf-8'?>
<algebra>
  <basicAlgebra>
    <algName>P4</algName>
    <cardinality>4</cardinality>
    <operations>
      <op>
        <opSymbol>
          <opName>d</opName>
          <arity>2</arity>
        </opSymbol>
        <opTable>
          <intArray>
            <row r="[0]">0,1,2,3</row>
            <row r="[1]">1,0,3,2</row>
            <row r="[2]">2,3,0,1</row>
            <row r="[3]">3,2,1,0</row>
          </intArray>
        </opTable>
      </op>
      <op>
        <opSymbol>
          <opName>m</opName>
          <arity>2</arity>
        </opSymbol>
        <opTable>
          <intArray>
            <row r="[0]">0,0,0,0</row>
            <row r="[1]">0,1,2,3</row>
            <row r="[2]">0,2,0,2</row>
            <row r="[3]">0,3,2,1</row>
          </intArray>
        </opTable>
      </op>
    </operations>
  </basicAlgebra>
</algebra>
```

Rysunek 1: Przykładowy plik reprezentujący algebrę

*Rysunek 1* przedstawia przykładowy opis algebry. Opis ten przedstawia algebrę o uniwersum {0,1,2,3} wraz z dwoma operacjami: dodawania modulo 4 oraz mnożeniem modulo 4. Opis algebry składa się z kilku sekcji. Sekcja *cardinality* informuje o ilości elementów w uniwersum algebry. Sekcja *operations* zawiera opisy wszystkich funkcji zadeklarowanych

w strukturze. Opis pojedynczej operacji znajduje się w podsekcji *op*. Sekcja ta składa się z trzech kolejnych pól i są to odpowiednio: **opName**, **arity** oraz **opTable**. Sekcja *opName* przechowuje informacje o nazwie operacji, *Arity* przechowuje informację o liczbie argumentów przyjmowanych przez operację, zaś *opTable* zawiera tablicę operacji.

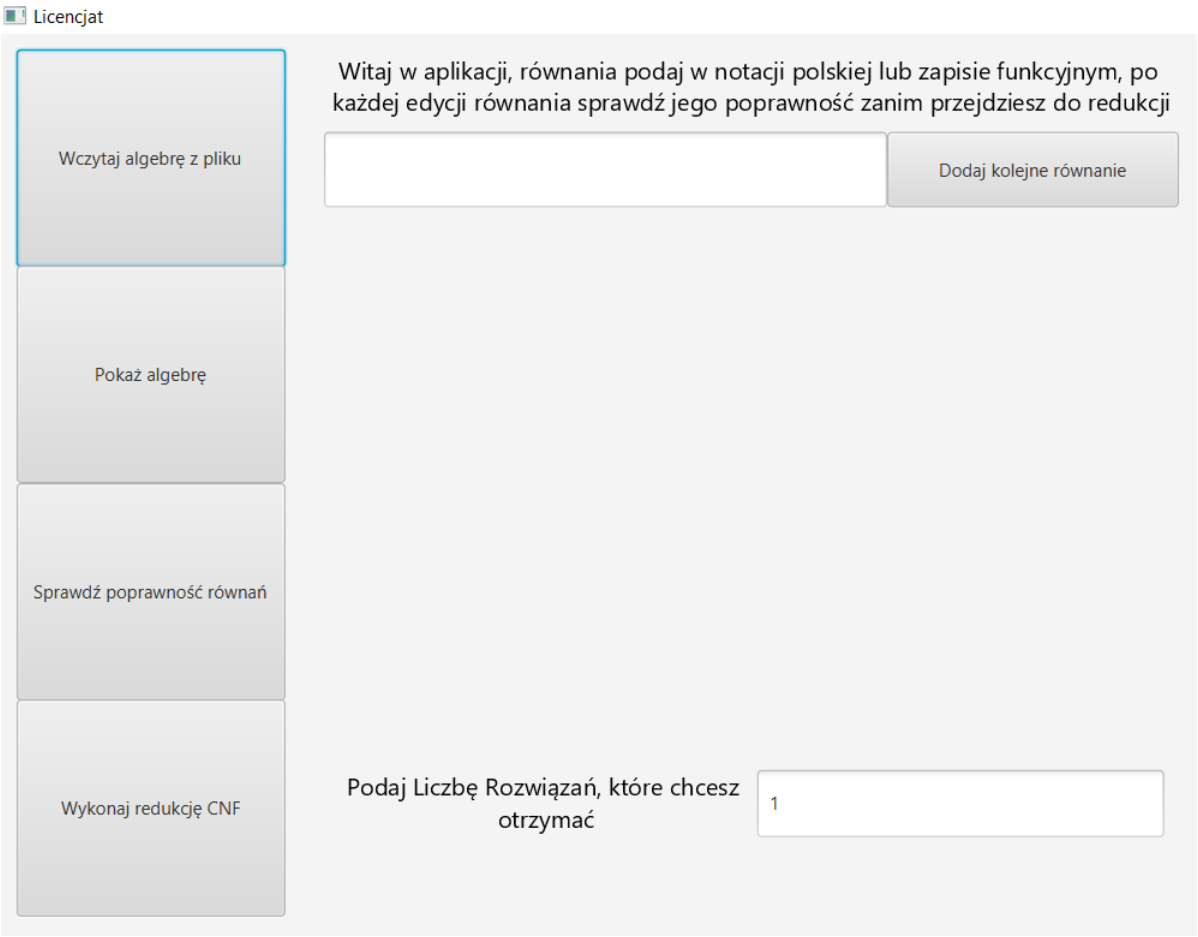
```
c 1=W0_0 2=W0_1 3=W0_2
c 4=y_0 5=y_1 6=y_2
c 7=x_0 8=x_1 9=x_2
p cnf 9 18
1 2 3 0
-1 -2 0
-1 -3 0
-2 -3 0
4 5 6 0
-4 -5 0
-4 -6 0
-5 -6 0
-4 1 0
-5 2 0
-6 3 0
7 8 9 0
-7 -8 0
-7 -9 0
-8 -9 0
-7 3 0
-8 1 0
-9 2 0
```

Rysunek 2: Przykładowy plik reprezentujący redukcje CNF

*DIMACS-CNF* jest standardowym formatem plików przyjmowanych przez programy rozwiązujące SAT-a. Zawiera on numery indeksów zmiennych zdaniowych użytych w formule. *Rysunek 2* przedstawia przykładowy plik *DIMACS-CNF*. Został on wygenerowany poprzez redukcję równania $\mathbf{d(2,x) = d(y,0)}$ w grupie dodawania modulo 3. Linie zaczynające się od znaku 'c' są komentarzami i nie są rozpatrywane przez SAT-solver. Linie te służą jako legenda, pokazująca indeksowanie zmiennych zdaniowych. Pierwsza linia, która nie jest komentarzem, informuje o strukturze całego pliku. 'p cnf' oznacza, że plik przechowuje formułę w postaci CNF. Kolejne dwie liczby oznaczają odpowiednio: ilość zmiennych zdaniowych oraz liczbę klauzul rozpatrywanej formuły. Dalsza część pliku jest

zarezerwowana dla klauzul. ’0’ jest specjalnym symbolem oznaczającym koniec pojedynczej klauzuli. Pojedyncza klauzula składa się z indeksów, gdzie dana wartość indeksu reprezentuje konkretną zmienną zdaniową. Jeżeli wartość indeksu jest ujemna, to znaczy że zmienna zdaniowa reprezentowana przez ten indeks jest zanegowana.

## 3.2 Interfejs




Rysunek 3: Interfejs aplikacji

Do napisania aplikacji użyty został język Java w wersji 11 [6]. Został on wybrany, ponieważ posiada on wiele bibliotek znacząco ułatwiających pracę nad zadanym problemem. Jedną z nich jest biblioteka graficzna *javaFX* [7], która w połączeniu z programem *Scene Builder* ułatwia pracę nad budową interfejsu użytkownika. Kolejną biblioteką przemawiającą za wyborem Javy jest *DocumentBuilderFactory*. Biblioteka ta pozwala na podział pliku XML po znacznikach znajdujących się w środku dokumentu, co znacznie przyspiesza odczyt potrzebnych danych z plików, w tym przypadku z plików z rozszerzeniem *.ua*.

Sat-Solver używany w programie znajduje się w bibliotece *sat4j* [8]. Jest to najpopularniejsza biblioteka w języku Java, która udostępnia SAT-solvery, stąd też jej zastosowanie w tej pracy.

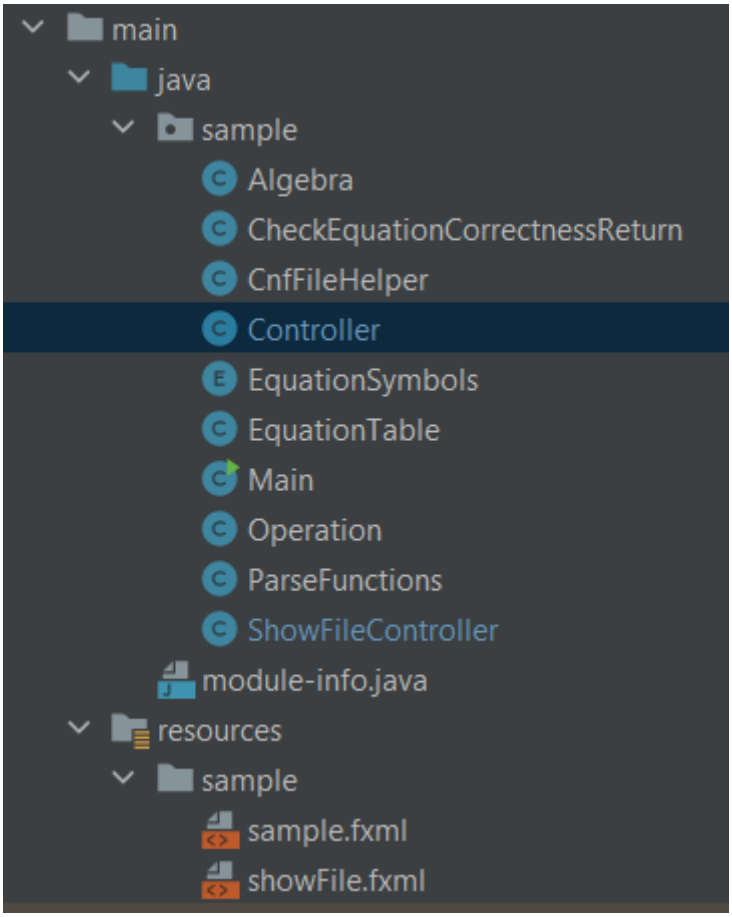
Stworzony interfejs jest prostym oknem do dwustronnej komunikacji z programem. Podzielony jest on na dwie sekcje: po lewej stronie znajdują się przyciski odpowiadające za logikę programu. Po prawej stronie zaś znajduje się miejsce do wprowadzenia danych, które aplikacja ma przetworzyć. Interfejs zezwala na wprowadzenie więcej niż jednego równania, poprzez dodawanie nowych okien do wpisywania równań przez użytkownika. Użytkownik może też podać ilość rozwiązań, które chce otrzymać. Sat-Solver domyślnie zwraca tylko jedno wartościowanie. Aby uzyskać ich więcej, należy uruchomić SAT-solver wielokrotnie, za każdym razem dopisując na koniec pliku klauzulę. Klauzula ta ma za zadanie wykluczyć poprzednio znalezione wartościowanie spełniające formułę.

## 3.3 Klasy użyte w programie

Kod aplikacji podzielony został na kilka klas. Zapewnia to przede wszystkim większą przejrzystość plików źródłowych napisanego programu. Klasami powiązanymi z algebrą w aplikacji są klasy *Operations* oraz *Algebra*. Obiekt *Operations* przechowuje informację o: nazwie operacji, tablicy operacji oraz liczbie przyjmowanych przez operację argumentów. Obiekt *Algebra* przechowuje informacje o całej algebrze wczytanej do aplikacji. Posiada ona informację o liczbie elementów uniwersum oraz listę operacji. Aplikacja posiada unikalny obiekt klasy *Algebra*, dlatego w klasie zadeklarowana jest dodatkowo zmienna logiczna zarządzająca przyciskiem *pokaż algebrę*, który wyświetla użytkownikowi wybraną przez niego algebrę. Dzięki temu użytkownik nie może wyświetlić algebry, dopóki jej nie wczyta.

Klasy *EquationSymbols* oraz *EquationTable* używane są w trakcie parsowania równania. Klasa *EquationSymbols* jest pojedynczym typem wyliczeniowym *enum*. Symbole zdefiniowane w tej klasie służą do klasyfikacji argumentów równania w trakcie działania algorytmu parsującego. Pojedynczy argument może zostać sklasyfikowany jako: symbol funkcyjny, zmienna, stała, lub znak pusty i na podstawie tej klasyfikacji algorytm parsujący podejmuje kolejne działania. Obiekt klasy *EquationTable* reprezentuje pojedyncze równanie znajdujące się w układzie równań po jego uproszczeniu. Obiekt ten przechowuje informację o liczbie i liście argumentów, symbolu funkcyjnym operacji znajdującej się w

równaniu oraz o zmiennej wynikowej $w_i$.




Rysunek 4: Spis wszystkich klas programu

Klasy **CnfFileHelper** oraz **ParseFunctions** używane są w trakcie działania redukcji równania do postaci formuły CNF. Obiekt klasy *CnfFileHelper* jest obiektem globalnym stworzonej aplikacji. Obiekt ten posiada 3 listy. Pierwsza lista przechowuje zmienne równania użyte w redukcji (w celu wykrycia duplikatów zmiennych równania). Lista druga przechowuje indeksy stworzonych zmiennych zdaniowych. Lista trzecia zawiera klauzule zapisane za pomocą indeksów znajdujących się w liście drugiej. Ostatnia z wyżej wymienionych list służy do wygenerowania pliku *DIMACS-CNF* będącego wejściem Sat-Solvera. Obiekt jest czyszczony po zakończeniu redukcji. Klasa *ParseFunctions* przechowuje funkcje związane z parsowaniem pliku oraz redukcją układu równań. Wszystkie te funkcje są funkcjami statycznymi, więc nie wymagane jest tworzenie obiektu tej klasy wewnątrz programu. Każdy opisany w poprzednim rozdziale typ klauzuli generowany jest poprzez inną metodę.

Plik **sample.fxml** oraz klasa **Controller** związane są bezpośrednio z interfejsem do-

stępnym dla użytkownika. W pliku *sample.fxml* opisana jest logika interfejsu wraz z odnośnikami do klasy *Controller*. Klasa ta oprogramowuje interfejs. Posiada ona funkcje powiązanie z każdym przyciskiem zadeklarowanym w pliku **sample.fxml**.

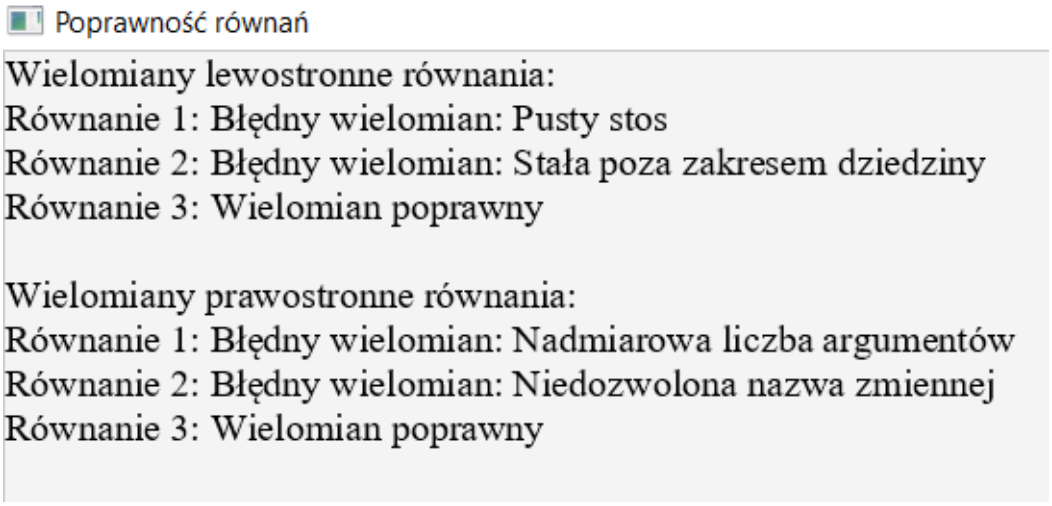
Plik **showFile.fxml** opisuje dodatkowe okno posiadające tylko jeden obiekt *text* mające na celu wyświetlenie wyjścia programu (na przykład wyniku redukcji).

## 3.4 Logika aplikacji

Stworzona aplikacja posiada strukturę liniową, w której każdy kolejny krok jest jasno określony i generuje wejście kroku następnego. Krokiem pierwszym jest wczytanie algebry. Użytkownik po kliknięciu przycisku *wczytaj algebrę* otrzymuje okno wyboru pliku. Po wyborze pliku tworzony jest obiekt klasy *DocumentBuilderFactory*. Po zapisaniu liczności algebry tworzony jest kolejny obiekt tej klasy grupujący dane po znaczniku *operations*. Każda operacja jest zapisywana do nowego obiektu klasy *Operations*, który jest dopisywany do listy przechowywanej w obiekcie klasy *Algebra*

Krokiem drugim jest podanie równań oraz sprawdzenie ich poprawności. Użytkownik może manipulować ilością wprowadzanych równań za pomocą przycisków *dodaj równanie* oraz *usuń równanie*. Parsowanie odbywa się dla równania zapisanego w notacji polskiej. Usunięcie nawiasów i przecinków z wielomianów podanych w zapisie funkcyjnym automatycznie konwertuje równanie do notacji polskiej. Wprowadzone równania dzielone są po znaku równości i osobno rozpatrywana jest poprawność dla wielomianów lewostronnych i prawostronnych. Poprawność wielomianów sprawdzana jest od prawej do lewej strony. Symbole zakwalifikowane jako stałe oraz zmienne są umieszczane na stosie. Po znalezieniu symbolu, który jest symbolem funkcyjnym, zdejmowana jest ze stosu liczba symboli równa arności znalezionej operacji i na stosie umieszczana jest zmienna reprezentująca wynik operacji.

Algorytm może zwrócić kilka różnych komunikatów. Jeżeli w momencie znalezienia symbolu funkcyjnego, na stosie przechowującym zmienne jest za mało argumentów aby znaleziona funkcja mogła zostać wykonana, to algorytm zwraca informacje o pustym stosie. Jeżeli po zakończeniu algorytmu na stosie znajduje więcej niż jeden element, zmienna wynikowa równania, to znaczy że liczba podanych argumentów jest za duża. W tym przypadku algorytm zwraca komunikat o nadmiarowym argumencie. Jeżeli znaleziona stała nie należy do uniwersum, to algorytm zwraca komunikat o niepoprawnej wartości stałej



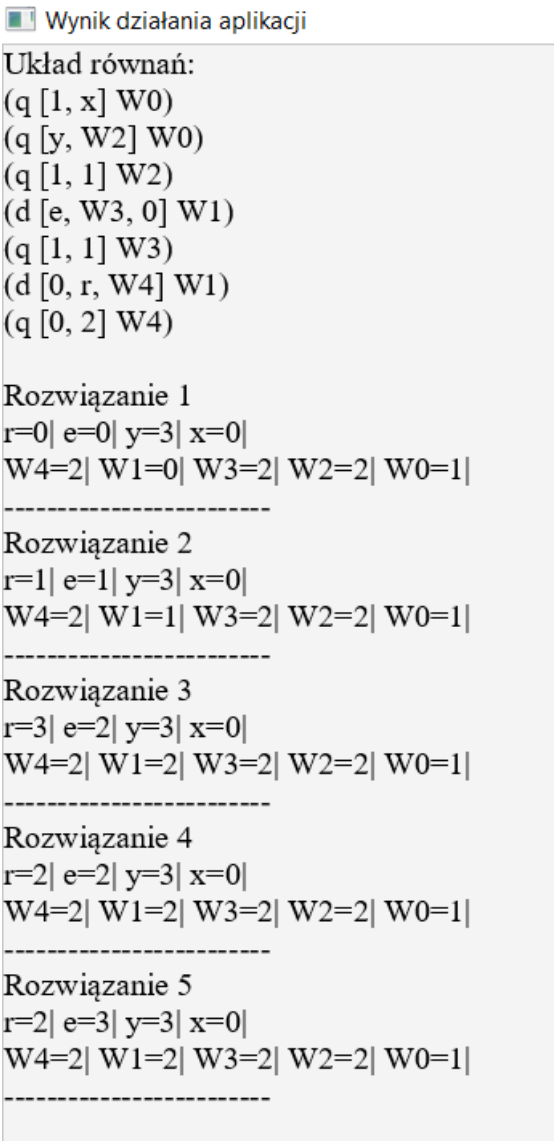

Rysunek 5: Przykładowe wyjście poprawności równań

liczbowej. Symbole będące napisem dowolnej długości i zaczynające się od litery są zmiennymi. Jeżeli więc program znajdzie symbol będący ciągiem znaków, który zaczyna się od cyfry i symbol ten nie jest symbolem funkcyjnym ani stałą, to algorytm zwróci komunikat o niepoprawnej nazwie zmiennej. W każdym innym przypadku podane równanie jest równaniem poprawnym.

Jeżeli parsowanie równań zakończyło się pomyślnie, to tworzony jest układ równań. Algorytm wykorzystuje listę symboli równania, stworzoną w trakcie parsowania. Jeżeli napotkanym symbolem jest symbol funkcyjny to tworzony jest nowy obiekt klasy *EquationTable*, który jest dopisywany na koniec listy reprezentującej układ równań w programie. W przypadku, gdy symbol jest zmienną albo stałą, to dopisywany jest on do listy argumentów pierwszego obiektu *EquationTable*, dla którego lista ta nie została jeszcze zapełniona.

Po utworzeniu układu równań wykonywana jest właściwa część redukcji. Odbywa się ona w pętli dla każdego równania zapisanego w powyższym układzie. W pierwszej kolejności tworzony jest zbiór zmiennych zdaniowych dla zmiennej wynikowej równania, (jeżeli jeszcze nie został stworzony). Następnie sprawdzane jest, czy którykolwiek z argumentów rozpatrywanego równania jest zmienną. Tworzona jest lista wartości, do której stałe są przepisywane, a zmienne równania zastępowane liczbą -1. Jeżeli zmienna zostanie znaleziona i jest to jej pierwsze wystąpienie w redukcji to tworzony jest zbiór zmiennych zdaniowych odpowiadający znalezionej zmiennej równania. Indeksy zmiennych zdaniowych oraz stworzone klauzule są dopisywane do odpowiednich list znajdujących się w obiekcie

klasy *CnfFileHelper*. Następnie wykonywane jest szukanie wartościowań, na podstawie których tworzone są klauzule Horna.




Rysunek 6: Wynik aplikacji

Po zakończeniu działania redukcji następuje zapis formuły do pliku *DIMACS-CNF*. Zapisywane są gotowe klauzule przechowywane w liście znajdującej się w *CnfFileHelper*. Na koniec uruchamiany jest Sat-Solver. Jako odpowiedź zwrotną użytkownik otrzymuje układ równań użyty w trakcie redukcji oraz wartości zmiennych, dla których układ równań jest spełniony.

## 3.5 Testy aplikacji

Przeprowadzone zostały testy, mające na celu zweryfikowanie poprawności napisanej aplikacji. Sprawdzono każdy moduł osobno, po czym wszystkie naraz. Działanie programu zostało przetestowane dla losowo generowanych algebr.

| Algebra | Liczność | Arność | Zmienne | Klauzule | Układ | czas (ms) |
|---|---|---|---|---|---|---|
| D2_10 | 2 | 10 | 210 | 10450 | 10 | 91 |
| D2_10 | 2 | 10 | 420 | 29000 | 20 | 176 |
| D5_3 | 5 | 3 | 175 | 1635 | 10 | 28 |
| D5_3 | 5 | 3 | 705 | 7051 | 44 | 77 |
| D10_5 | 10 | 5 | 770 | 1403542 | 14 | 9103 |
| D10_5 | 10 | 5 | 440 | 802024 | 8 | 3573 |
| D10_5 | 10 | 5 | 220 | 401012 | 4 | 1097 |
| D20_3 | 20 | 3 | 280 | 34674 | 4 | 81 |
| D20_3 | 20 | 3 | 2500 | 343875 | 40 | 3505 |
| D105_3 | 105 | 3 | 525 | 49357 | 4 | 163 |
| D105_3 | 105 | 3 | 1260 | 109844 | 8 | 490 |
| D105_3 | 105 | 3 | 1995 | 181251 | 12 | 1119 |
| D105_3 | 105 | 3 | 3045 | 1426666 | 17 | 6384 |
| D8_2 | 8 | 2 | 952 | 7291 | 60 | 56 |

Tabela 2: Wyniki testów

*Tabela 2* przedstawia przykładowe wyniki testów. Użyte algebry posiadały pojedynczą operację. Kolumna *liczność* zawiera rozmiar algebry, zaś *arność* liczbę argumentów operacji. Kolumna *układ* pokazuje liczbę równań w układzie po jego uproszczeniu. Argumentami równań, dla algebr o małej tabeli operacji, były zmienne oraz symbole funkcyjne. Dla algebr *D105_3* oraz *D10_5* dopuszczono możliwość wystąpienia stałych algebry. Kolumna *zmienne* przedstawia ilość stworzonych zmiennych zdaniowych, zaś *klauzule* ilość stworzonych klauzul w trakcie trwania redukcji. Testy wykazały, że prędkość działania aplikacji w dużym stopniu zależy od wielkości tablicy operacji i ilości tworzonych wartościowań na jej podstawie.

# Podsumowanie

W pracy udało się wypełnić wszystkie założenia dotyczące aplikacji. Program przyjmuje algebrę w formacie *UACalc* oraz dowolnie duży układ równań i zwraca wartości zmiennych spełniające układ. Liczba rozwiązań, która ma zostać wygenerowana, podawana jest przez użytkownika. Uzyskano satysfakcjonujące wyniki dotyczące wydajności aplikacji. Istnieje wiele możliwości rozwoju programu poprzez dodanie nowych funkcjonalności. Przykładową funkcjonalnością może być sprawdzanie czy równanie jest identycznością, czyli czy dla każdego możliwego wartościowania z uniwersum algebry równanie jest spełnione. Przydatną funkcjonalnością mogłoby się również okazać generowanie algebr bezpośrednio w programie, zamiast wczytywania ich z pliku. Użytkownik podawałby liczność uniwersum, liczbę operacji oraz wzór operacji, na podstawie której generowana by była zawartość tabeli operacji.

# Literatura


[1] Oficjalna strona konkursu programów SAT-solver
http://www.satcompetition.org/ (Dostęp: 08.06.2023)

[2] Stanley Burris, H. P. Sankappanavar "A Course in Universal Algebra" The Millennium Edition, Springer, New York, 2012

[3] Christos H. Papadimitriou "Złożoność obliczeniowa", Wydawnictwo Helion, 2012

[4] Oficjalna strona UACalc
https://uacalc.org/ (Dostęp: 28.05.2023)

[5] Strona opisująca format DIMACS-CNF
https://people.sc.fsu.edu/~jburkardt/data/cnf/cnf.html (Dostęp: 28.05.2023)

[6] Dokumentacja Java
https://docs.oracle.com/en/java/javase/11/ (Dostęp: 29.05.2023)

[7] Dokumentacja JavaFX
https://openjfx.io/javadoc/17/javafx.graphics/javafx/application/Application.html
(Dostęp: 29.05.2023)

[8] Dokumentaja biblioteki SAT-solvera
https://www.sat4j.org/ (Dostęp: 29.05.2023)