# 1 Sposób rozwiązania zadania

## 1.1 Wnioskowanie w przód

W projekcie wykorzystany został standardowy algorytm realizujący wnioskowanie w przód. Funkcja realizująca algorytm zakłada, iż wiedza przedstawiona jest w postaci listy udowodnionych literałów oraz listy formuł. Formuły tworzone są na podstawie klauzul podanych w pliku i mają postać implikacji składającej się z listy przesłanek oraz literału będącego konkluzją. Algorytm systematycznie przegląda listę formuł szukając takich, które na podstawie aktualnego zbioru udowodnionych literałów pozwolą wywnioskować nowy literał, który jest następnie dodawany do tego zbioru. Warunkiem stopu algorytmu jest znalezienie formuły, której konkluzją jest literał, który należało dowieść lub sytuacja, gdy na podstawie dostępnej wiedzy nie da się wywnioskować kolejnego literału. Wynik działania algorytmu jest podstawą do budowy grafu wizualizującego proces wnioskowania w przód, którego węzły początkowe oznaczają obserwowane fakty, pozostałe węzły reprezentują wywnioskowane zdania, zaś krawędź prowadząca do węzła oznacza klauzulę, która posłużyła do wywnioskowania prawdziwości zdania reprezentowanego przez ten węzeł.

## 1.2 Wnioskowanie w tył

W projekcie został użyty standardowy algorytm wnioskowania w tył. Funkcja realizująca algorytm zakłada, iż cała dostępna wiedza wyrażona jest w postaci formuł, mających taką samą postać jak w przypadku wnioskowania w przód. Algorytm przegląda listę formuł w poszukiwaniu takich, które mogłyby posłużyć do udowodnienia literału znajdującego się w zbiorze hipotez niepotwierdzonych. Po znalezieniu odpowiedniej formuły, jej konkluzja uznana zostaje za hipotezę wstępnie potwierdzoną natomiast jej nieudowodnione przesłanki uznane zostają za hipotezy niepotwierdzone. Warunkiem stopu algorytmu jest osiągnięcie stanu, w którym zbiór hipotez niepotwierdzonych jest pusty bądź sytuacja, w której nie można znaleźć formuł służących do udowodnienia literałów ze zbioru hipotez niepotwierdzonych. Wynik działania algorytmu jest podstawą do budowy acyklicznego grafu, budowanego od strony konkluzji, która ma zostać udowodniona lub zweryfikowana.

# 2 Struktura programu

Projekt został zrealizowany jako dwuwątkowa aplikacja okienkowa. Składa się z trzech głównych modułów:

1. Controller
2. Calculation Engine
3. Vision

## 2.1 Controller

Zgodnie z nazwą, zarządza przepływem danych pomiędzy głównymi modułami oraz reaguje na zdarzenia zarejestrowane w module Vision (np. naciśnięcie przycisku). Controller jest uruchamiany we własnym wątku. Komendy do Calculation Engine wydawane są synchronicznie. Wydawanie komend do Vision odbywa się asynchronicznie, poprzez wywołanie publicznych metod Vision w osobno tworzonych, jednorazowych wątkach.

### 2.2 Calculation Engine

Moduł zajmuje się:

- wczytywaniem oraz interpretacją zawartości plików z danymi
- przekształcaniem wczytanych z pliku klauzul w formuły
- wykonaniem algorytmu wnioskowania dla uzyskanych formuł (w dwóch wariantach - w przód oraz wstecz)

Calculation Engine zwraca wyliczone wyniki do Controller.

### 2.3 Vision

Vision zajmuje się przedstawieniem graficznym wykonanego zadania. Najpierw wyświetla ekran menu głównego udostępniający opcje rozpoczęcia wnioskowania w obu wariantach. Vision wizualizuje proces wnioskowania w postaci grafu w trybie pracy krokowej. Graf jest rysowany z wykorzystaniem biblioteki mxGraph.

## 3 Składnia plików wejściowych

Dane wejściowe podawane są do programu w pliku *data.txt*, który w obecnej wersji musi znajdować się w folderze, z któregu uruchamiany jest program. W pliku tym zadane są klauzule wejsciowe oraz literał, który ma zostać udowoniony. Kolejne wiersze danych mogą (ale nie muszą) kończyć się średnikami.
Literał do udowonienia oraz literały w klauzulach oznaczane są wartościami liczb całkowitych (bez zera, np. 1,2,3,...). Negacja literału oznacza jest wartościami ujemnymi. Np. zaprzeczenia literału L2 przyjmuje postać '-2'.
Klauzule reprezentowane są jako alternatywy literałów. Znak alternatywy oznaczny jest jako 'v'. Przykładowa klauzula: "2 v -3 v -5".
Format pliku jest następujacy:

```
Prove: <literał do udowonienia>
<klauzula1>
<klauzula2>
 ...
<klauzulaN>
```

Przykład pliku wejściowego:

```
Prove: -2
-1 v 2 v 3
-2 v 4
-3 v -4
3 v 4
-3 v 5
-4 v -5
5
```

## 4 Biblioteka mxGraph

W celu uruchomienia programu konieczne jest załączenie do proejktu biblioteki mxGraph. Plik *.jar* mxGraph znajduje się w katalogu */lib* projektu.
Dodanie mxGraph do proejktu w Eclipse: Kliknij prawym przyciskiem myszy na plik *jgraphx.jar* z katalogu lib. Wybierz opcje 'Build Path', a następnie 'Add to Build Path'.
Załączona biblioteka powinna pojawić się w katalogu *ReferencedLibraries*.