Cel ćwiczenia

Celem ćwiczenia była implementacja skorupowego systemu ekspertowego. System powinien umożliwiać wnioskowanie w oparciu o różnego rodzaju bazy wiedzy, aczkolwiek w pewnym standardowym określonym formacie. Prowadzący ćwiczenie pozostawił dowolność odnośnie technologii, w której stworzone ma być narzędzie.

Technologie

Z racji tego, że my jako autorzy w trakcie studiów zdobyliśmy komercyjne doświadczenie w języku C# to w tym właśnie języku postanowiliśmy stworzyć system skorupowy. Postanowiliśmy zaimplementować aplikację desktopową przy użyciu technologii WPF (Windows Presentation Foundation) w asyście wspomnianego języka C#.

W czasie rozwoju oprogramowania korzystaliśmy z szeregu dostępnych funkcjonalności języka C# oraz samej technologii WPF. Częste zastosowanie znajdują w naszej implementacji wyrażenia lambda oparte o LINQ (Language INtegrated Query), umożliwiające szybkie dokonywanie operacji na kolekcjach. Zaś szata graficzna programu została opracowana w oparciu o bibliotekę typu open source o nazwie MahApps.

Nie omieszkaliśmy również nie skorzystać z dobrodziejstw wzorców projektowych. Jako swego rodzaju standard w komercyjnych aplikacjach desktopowych wykorzystywany jest wzorzec architektoniczny MVVM (Model-View-ViewModel), zatem również i my postanowiliśmy zaimplementować go w naszej aplikacji. Ów wzorzec, jak sama nazwa wskazuje, pozwala w sensowny sposób odseparować warstwę danych od warstwy wizualnej aplikacji. Celem uproszczenia i przyspieszenia implementacji wzorca skorzystaliśmy z biblioteki open source o nazwie MVVMLight.

Całość implementacji bazuje na programowaniu orientowanym obiektowo.

Skorupowy system ekspertowy

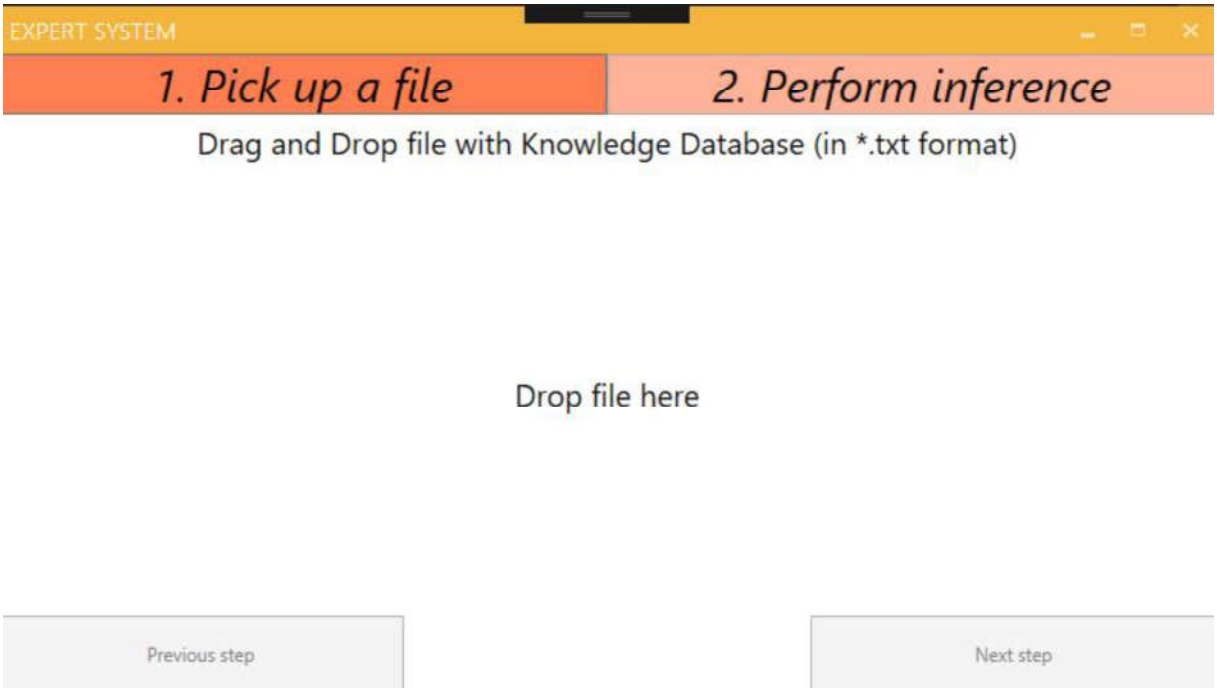
W ramach systemu skorupowego wyszczególnić można dwa podstawowe elementy: system wnioskujący (zaprogramowany algorytm) oraz bazę wiedzy(dostarczaną przez użytkownika). Aczkolwiek, pojęcie systemu skorupowego można też nieco poszerzyć, co dobrze ukazuje rysunek 1.

Rysunek 1 Schemat systemu skorupowego

Napisanej przez nas aplikacja implementuje funkcjonalności związane z interfejsem użytkownika (GUI napisane w oparciu o technologię WPF), systemem wnioskującym (algorytm

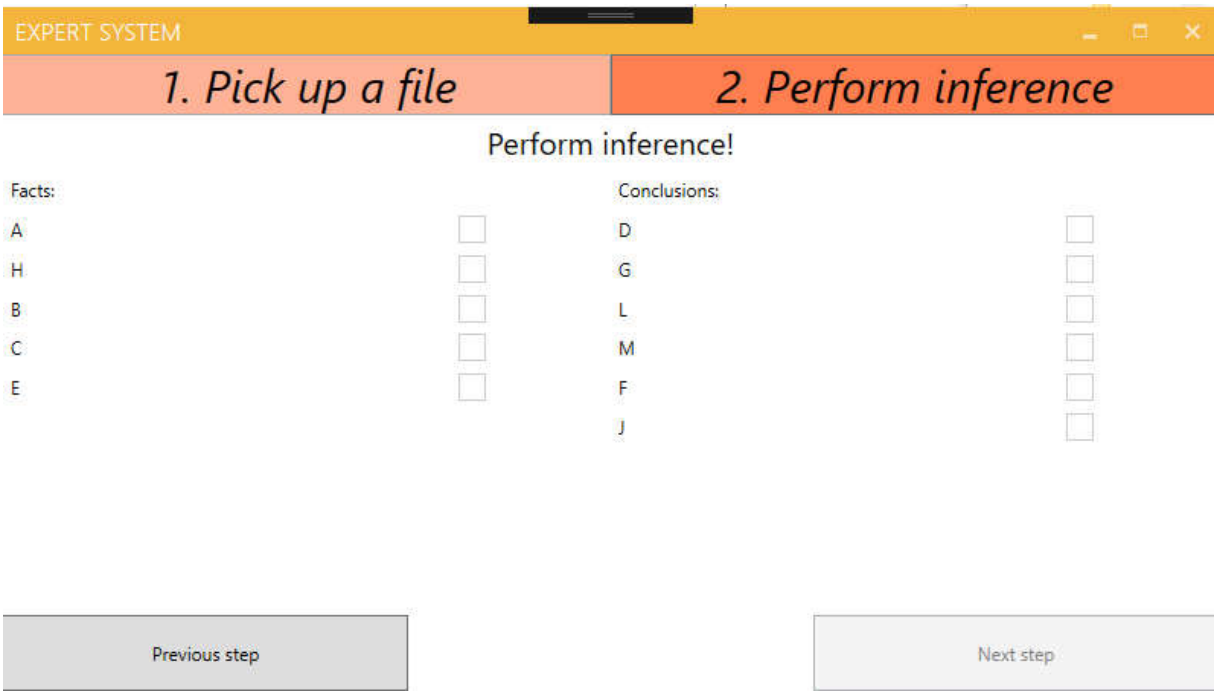
opisany w dalszej części sprawozdania) oraz dynamiczną bazą danych (struktury danych „we wnętrzu” aplikacji przechowujące właściwości faktów i konkluzji).

Interfejs użytkownika



Rysunek 2 Interfejs użytkownika – etap pierwszy, wybór bazy wiedzy

Staraliśmy się zaprojektować w miarę schludny i przejrzysty interfejs graficzny, umożliwiający użytkownikowi łatwą i intuicyjną interakcję z aplikacją. Jak widać na rysunku 2 widok startowy aplikacji wyposażony jest w kilka bloków tekstowych – instrukcji dla użytkownika oraz przyciski nawigacyjne (previous oraz next step), które zależnie od podjętych przez użytkownika akcji są dostępne bądź nie. W ramach aplikacji wyróżniamy dwa etapy jej działania: wczytanie bazy wiedzy oraz wnioskowanie. Każdy z etapów opatrzony jest stosownym polem tekstowym w górnej części okna programu. Zależnie od aktywnego etapu odpowiednie pole jest wyszarzane, bądź nie. Przycisk umożliwiający przejście użytkownikowi do następnego etapu uaktywnia się dopiero w momencie, gdy poprawnie wczytana zostanie baza wiedzy. Z kolei przycisk umożliwiający powrót do poprzedniego etapu staje się aktywny po przejściu etapu pierwszego. Warto nadmienić, że w przypadku przeciągnięcia pliku z bazą wiedz w niewłaściwym formacie (np. *.pdf) do użytkownika zostanie wystosowany komunikat o błędzie.



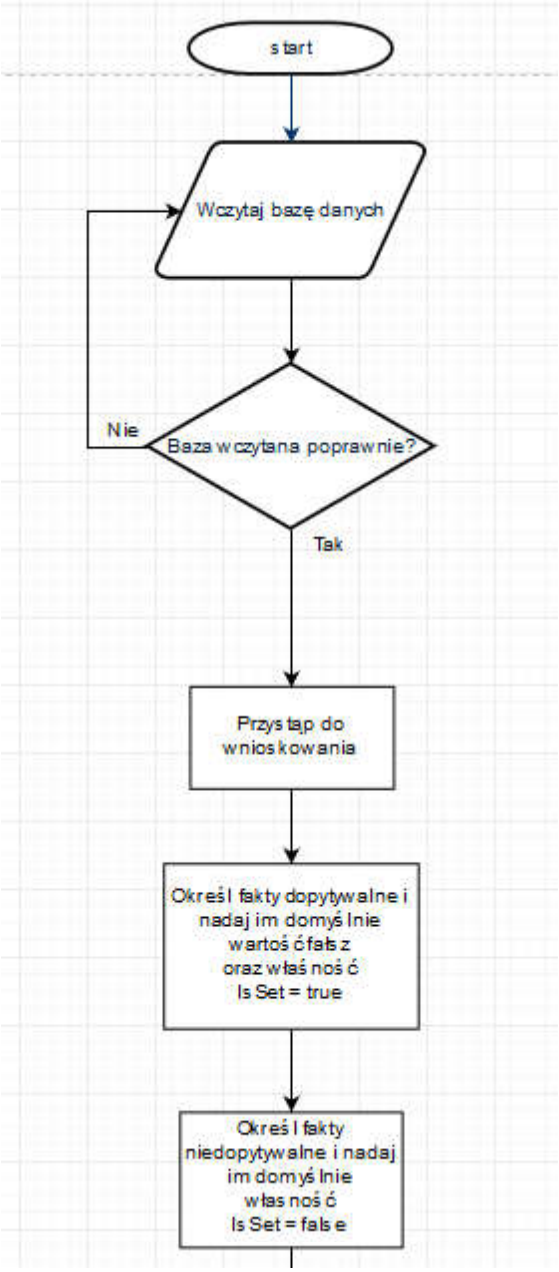
Rysunek 3 Interfejs użytkownika - etap drugi, wnioskowanie

Drugi etap działania aplikacji czyli wnioskowanie widoczny jest na rysunku 3. Użytkownik dowolnie może zmieniać wartość logiczną faktów dopytywalnych. Z kolei wartość logiczna faktów niedopytywanych ustalana jest na drodze wnioskowania w przód, którego algorytm zostanie opisany w następnym punkcie sprawozdania. Po każdorazowym zaznaczeniu lub odznaczeniu elementów typu „tickbox" skojarzonych z faktami dopytywalnymi automatycznie przeprowadzane jest wnioskowanie, czego efektem jest widoczna natychmiastowa zmiana wartości logicznej faktów niedopytywalnych.

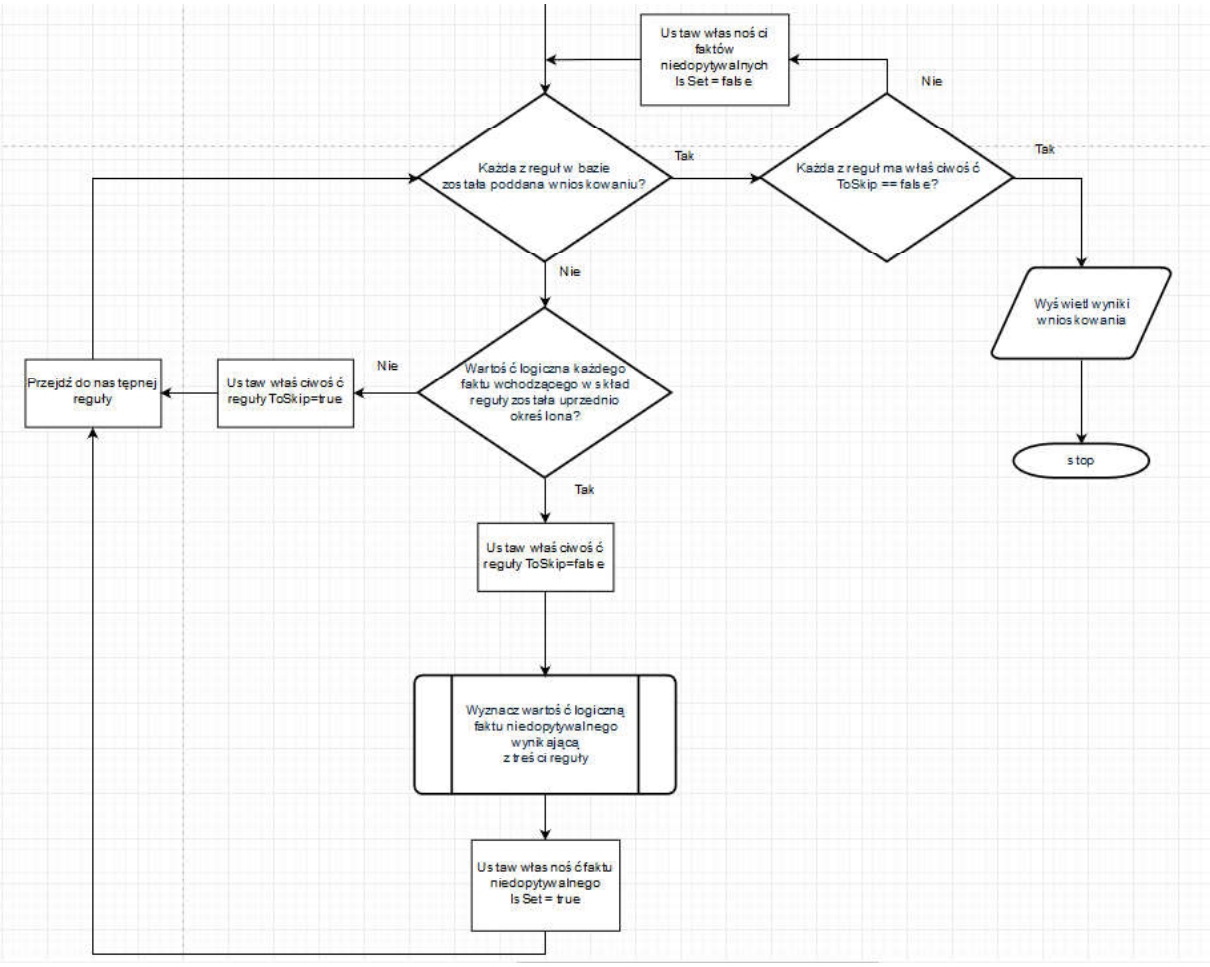
System wnioskujący

Implementacja naszego systemu wnioskującego zakłada obiektową strukturę reguł, oraz jej składowych: faktów oraz konkluzji (wniosków). Przy czym każdy fakt i konkluzja dziedziczą po klasie Predykat. Taki sposób porządkowania bazy wiedzy umożliwia umiejscawianie predykatów w strukturze bazy wiedzy przetworzonej przez program zarówno jako fakt jak i jako konkluzja. Sposób implementacji programu umożliwia również mechanizm zmiany wartości logicznej faktu w każdym miejscu jego wystąpienia w bazie wiedzy po każdorazowej zmianie jego wartości logicznej.

Najlepszym sposobem opisu zaimplementowanego przez nas algorytmu wnioskowania w przód będzie schemat blokowy przedstawiony poniżej:



Rysunek 4 Pierwsza część schematu blokowego



Rysunek 5 Druga część schematu blokowego

Przy czym wartość logiczna każdej z konkluzji w ramach każdej reguły określana jest (o ile jest to możliwe) jako iloczyn logiczny faktów nań składających się.

Wyniki wnioskowania w oparciu o prostą, testową bazę wiedzy

Baza wiedzy którą nasz program jest w stanie przetworzyć musi zawierać reguły zapisane w następujący sposób:

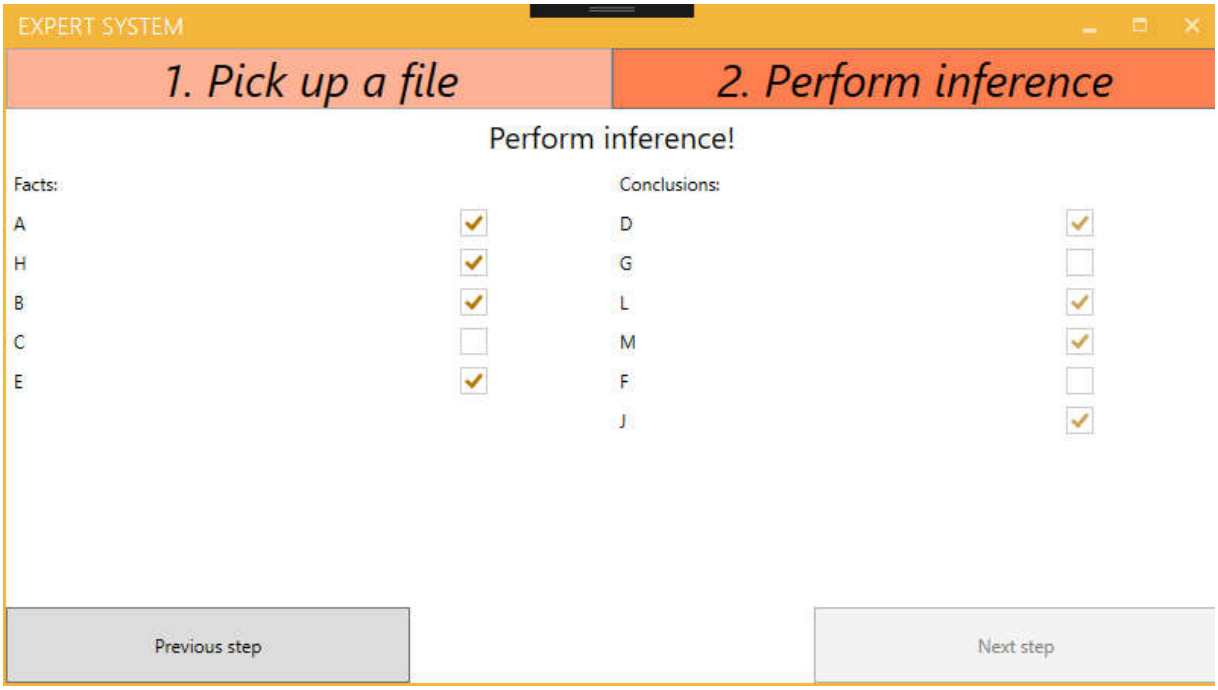
fakt, fakt, … fakt -> wniosek

A zatem nasza baza wiedzy musi być zapisana w formie klauzul Horna. Następujące po sobie fakty należy rozdzielać przecinkiem, a wniosek poprzedzać znakami ‘’->’’. Każda kolejna reguła powinna zostać zapisana w kolejnej linii pliku *.txt.

W celu sprawdzenia poprawności implementacji na przykładowej prostej bazie wiedzy przeprowadziliśmy wnioskowanie:

```
A -> D
F, H -> G
B -> L
D, J -> M
C, D -> F
A, E -> J
```

Przykładowy wynik wnioskowanie prezentowany jest na rysunku 6:

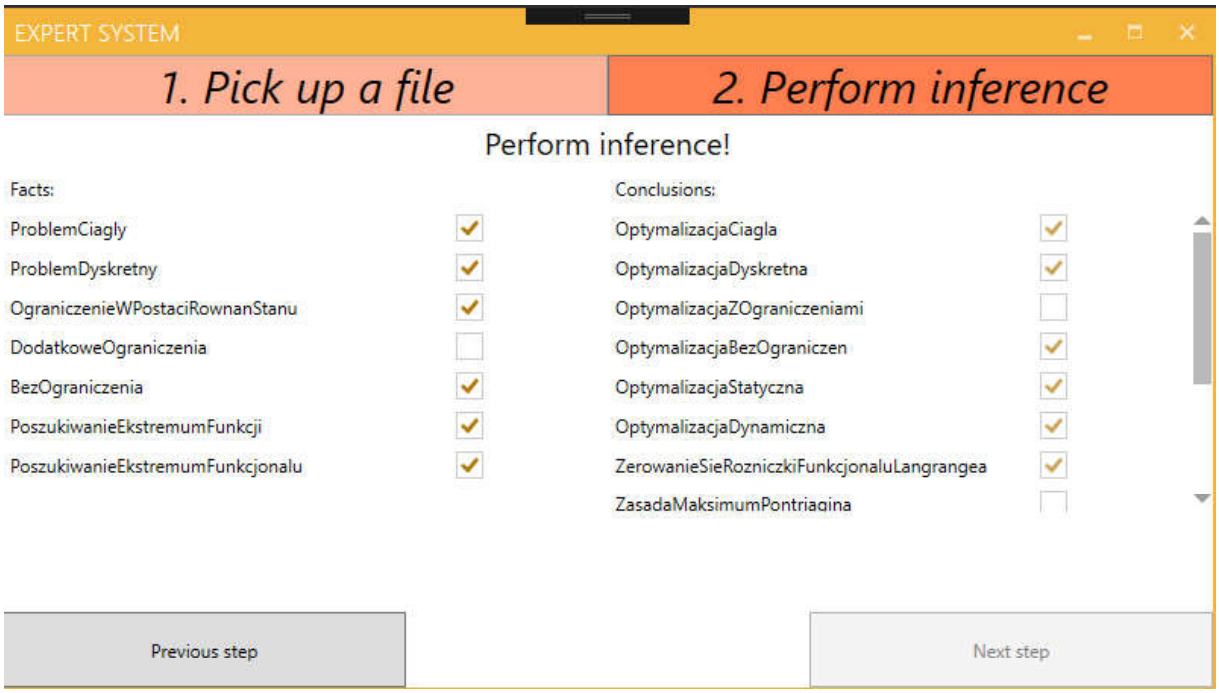


Rysunek 6 Wynik wnioskowania z wykorzystaniem bazy wiedzy o niewielkiej złożoności

Konfrontując wynik wnioskowania wypracowany przez system wnioskujący z wynikiem uzyskanym na drodze analizy bazy wiedzy można stwierdzić, że system działa poprawnie

Wyniki wnioskowania w oparciu o bardziej złożoną bazę wiedzy

Po sprawdzeniu poprawności działania systemu wnioskującego na prostej bazie testowej, można przystąpić do przetestowania działania aplikacji z wykorzystaniem bazy wiedzy o nieco bardziej złożonej strukturze.



Rysunek 7 Wynik wnioskowania z wykorzystaniem złożonej bazy wiedzy

Baza wiedzy przedstawia się w sposób następujący:

```
ProblemCiagly -> OptymalizacjaCiagla
ProblemDyskretny -> OptymalizacjaDyskretna
OgraniczenieWPostaciRownanStanu, DodatkoweOgraniczenia ->
OptymalizacjaZOgraniczeniami
BezOgraniczenia -> OptymalizacjaBezOgraniczen
PoszukiwanieEkstremumFunkcji -> OptymalizacjaStatyczna
PoszukiwanieEkstremumFunkcjonalu -> OptymalizacjaDynamiczna
OptymalizacjaCiagla, OptymalizacjaDynamiczna, OgraniczenieWPostaciRownanStanu ->
ZerowanieSieRozniczkiFunkcjonaluLangrangea
OptymalizacjaCiagla, OptymalizacjaDynamiczna, OptymalizacjaZOgraniczeniami ->
ZasadaMaksimumPontriagina
OptymalizacjaDyskretna, OgraniczenieWPostaciRownanStanu, OptymalizacjaDynamiczna
-> MetodaGradientuProstegoWPrzestrzeniSterowan
OptymalizacjaDyskretna, OgraniczenieWPostaciRownanStanu, DodatkoweOgraniczenia ->
MetodaPrzesuwnegoFunkcjonaluKary
OptymalizacjaDyskretna, OgraniczenieWPostaciRownanStanu -> ProsteMetodyStrzalow
OptymalizacjaCiagla, OptymalizacjaDynamiczna, OptymalizacjaBezOgraniczen ->
AnalityczneMetodyRozwiazywaniaZadanOptymalizacji
     OptymalizacjaCiagla, OptymalizacjaDynamiczna, DodatkoweOgraniczenia ->
AnalityczneMetodyRozwiazywaniaZadanOptymalizacji
```

Kod źródłowy projektu

Z uwagi na sporą objętość kodu nie będziemy zamieszczać go w całej okazałości. Poniżej zamieszczamy jedynie fragment odpowiedzialny stricte za wnioskowanie. Pełny kod projektu znajduje się na zdalnym repozytorium pod adresem https://github.com/kreeaq/Expert-System. Gdzie z kolei kod można przeglądać/ściągać/modyfikować bez limitu oraz bez zakładania konta w serwisie github.

```
public static void Calculate(List<Rule> rules)
        {
            do
            {
                foreach (Rule rule in rules)
                {
                    bool result = true;
                    foreach (Predicate fact in rule.Facts)
                    {
                        //Check if rule has only SetFacts
                        if (fact.IsSet)
                        {
                            result = result && fact.Value;
                            rule.ToSkip = false;
                        }
                        else
                        {
                            rule.ToSkip = true;
                            break;
                        }
                    }

                    if (rule.ToSkip)
                    {
                        continue;
                    }

                    foreach (Predicate conclusion in rule.Conclusions)
                    {
                        conclusion.Value = result;
                        conclusion.IsSet = true;
                        //Find facts with similar name to conclusion
                        List<Fact> tmp = factsThatAreAlsoConclusions(rules).Where(f => f.Name ==
                        conclusion.Name).ToList();
                        if(tmp.Count > 0)
                        {
                            Fact exactFact = tmp.Where(c => c.Name == conclusion.Name).Single();
```

```
                    //And set its value to the result from inference
                    exactFact.Value = result;
                    exactFact.IsSet = true;
                }
            }
        }
    }
    while (rules.Where(r => r.ToSkip == true).Count() > 0);
}
```

Przygotowaliśmy również wersję instalacyjną aplikacji, którą wraz z bazą wiedzy (KnowledgeDatabase.txt) przesłaliśmy jako plik .7z wraz z niniejszym sprawozdaniem. W celu przetestowania działania wystarczy rozpakować archiwum, uruchomić plik setup.exe w celu instalacji aplikacji. Po zakończeniu instalacji aplikacja uruchomi się.

Wnioski końcowe

Na potrzeby ćwiczenia laboratoryjnego stworzyliśmy w funkcjonalny system skorupowy. Zaimplementowaliśmy zarówno interfejs użytkownika zapewniający swobodne korzystanie z aplikacji, jak i poprawnie działający system wnioskujący korzystający z dostarczonej przez użytkownika bazy wiedzy wchodzący również w interakcję z dynamiczną bazą danych, która w przypadku naszej aplikacji jest reprezentowana przez struktury danych programu. Najważniejszą zaletą napisanego przez nas rozwiązania jest fakt, że jedynym wymaganiem prawidłowego działania systemu wnioskującego jest prawidłowy format bazy wiedzy.