



Instytut Informatyki
Wydział Informatyki
Politechnika Poznańska
ul. Piotrowo 2, Poznań

PRACA DYPLOMOWA MAGISTERSKA

System wspomagający jednoczesne stosowanie wielu wytycznych klinicznych dla jednego pacjenta

Dariusz Radka 100383

Promotor

dr hab. inż. Szymon Wilk

Poznań, 2015

Spis treści

1	Wstęp	1
1.1	Wprowadzenie	1
1.2	Cel i zakres pracy	2
1.3	Struktura pracy	2
2	Przegląd literatury	3
2.1	Wytyczne postępowania klinicznego	3
2.2	Wykrywanie i usuwanie konfliktów w wytycznych postępowania klinicznego	3
2.2.1	Programowanie logiczne z ograniczeniami	3
2.2.2	Logika pierwszego rzędu	4
2.2.3	Podejście wykorzystujące paradygmat mieszanej inicjatywy	4
3	Programowanie logiczne z ograniczeniami (CLP)	6
4	Wykorzystane biblioteki i narzędzia	8
4.1	ECLiPSe	8
4.2	Choco 3	9
4.3	Graphviz	9
4.4	JPGD - A Java parser for Graphviz documents	11
5	Implementacja	13
5.1	Wybór chorób	15
5.2	Wyświetlanie grafów oraz udzielanie odpowiedzi na pytania	16
5.3	Wyszukiwanie konfliktów	18
5.4	Wyświetlanie wyników	21
6	Działanie systemu	24
6.1	Przypadek 1 - atak astmy i wrzód trawienny	24
6.2	Przypadek 2 - migotanie przedsionków, przewlekła choroba nerek i nadciśnienie . .	28
6.3	Przypadek 3 - wrzód dwunastnicy i przemijający atak niedokrwienny	32
7	Podsumowanie	35
7.1	Osiągnięte cele	35
7.2	Problemy przy realizacji pracy	35
7.3	Kierunki dalszego rozwoju	35
	Bibliografia	37

Rozdział 1

Wstęp

1.1 Wprowadzenie

Medycyna jest ważnym działem nauki, ponieważ dotyczy każdego człowieka. Stara się leczyć ludzi z chorób przy zastosowaniu odpowiednich terapii, na które składają się m.in. prawidłowo dobierane leki. Istotne znaczenie przy leczeniu pacjenta ma wiedza, jaką dysponuje lekarz. Informatyka, która jest dziedziną nauki zajmującą się przetwarzaniem informacji, może pomóc lekarzom w zdobywaniu wiedzy. Ważnym elementem zdobywania wiedzy medycznej jest system wspomagania decyzji klinicznych. Jest to program komputerowy, który pozwala personelowi medycznemu podejmować odpowiednie decyzje dotyczące pacjenta. Wśród systemów tego typu wyróżnia się: systemy do zarządzania informacją i wiedzą, systemy do zwracania uwagi, przypominania i alarmowania oraz systemy do opracowywania zaleceń. Dzięki tego typu systemom korzystającym z baz danych zawierających opisy leków, chorób, procedur medycznych oraz danych dotyczących pacjentów można w łatwiejszy sposób dobrać odpowiednią terapię dla pacjenta, co ma istotny wpływ na przebieg jego leczenia.

Przykładem takiego systemu medycznego jest Eskulap. Jest to system zrealizowany przez pracowników Politechniki Poznańskiej. Eskulap jest skierowany dla różnych placówek medycznych, m. in. szpitali, przychodni oraz aptek. Korzysta z niego wiele placówek na terenie całej Polski. Eskulap składa się z kilkudziesięciu modułów. Do modułów tych należą m. in. eRejestracja, Apteka, Laboratorium, Elektroniczna Dokumentacja Medyczna. Innym przykładem systemu medycznego jest Isabel. Jest to system diagnostyczny oparty na technologii webowej. Objawy i cechy charakterystyczne pacjenta są wprowadzane do systemu w formie tekstowej lub są pobierane z elektronicznego rekordu pacjenta. System zwraca w wyniku listę możliwych diagnoz pacjenta. Dodatkowo, istnieje możliwość skorzystania z książek, artykułów itp. dla każdej postawionej diagnozy.

Ostatnio coraz większą popularność zyskują wytyczne postępowania klinicznego (ang. clinical practice guidelines, CPG), które pozwalają opisać postępowanie dla pacjenta chorującego na określoną chorobę. Niestety, wytyczne te opisują jedynie jedną przypadłość, co może doprowadzić do konfliktów w przypadku pacjenta chorującego na kilka dolegliwości. W przypadku takich pacjentów konieczne jest dobranie takich terapii, które nie pozostają ze sobą w konflikcie. Jeśli takie konflikty występują, należy znaleźć odpowiedni zamiennik lub przepisać dodatkowy lek. W niektórych przypadkach trzeba zrezygnować z leków, aby nie pogorszyć stanu zdrowia pacjenta. Zadaniem systemu, którego dotyczy niniejsza praca magisterska, jest znajdowanie konfliktów wynikających ze stosowania wielu terapii u jednego pacjenta. W przypadku znalezienia konfliktu,

należy dokonać zmian w wytycznych tak, aby zminimalizować skutki uboczne.

1.2 Cel i zakres pracy

Celem pracy magisterskiej jest rozszerzenie podejścia opisanego w pracy "Mitigation of adverse interactions in pairs of clinical practice guidelines using constraint logic programming"[12], a także implementacja rozszerzonego podejścia. Rozszerzenie podejścia wiąże się z umożliwieniem stosowania więcej niż dwóch wytycznych, dopuszczeniem stosowania wielu zmian w wytycznych oraz uwzględnieniem stosowania dawek zarówno przy wykrywaniu konfliktów, jak i wprowadzania zmian w wytycznych. Na implementację systemu składa się opracowanie reprezentacji dla wytycznych, opisu konfliktów i wprowadzanych zmian, uwzględnienie dodatkowych danych pacjenta, które nie występują w wytycznych, a także implementacja rozszerzonego podejścia. Implementacja rozszerzonego podejścia polega na wykonaniu systemu pozwalającego na krokowe wykonywanie algorytmów i znajdowanie konfliktów oraz wprowadzanie ewentualnych zmian w wytycznych. Tworzony system ma zostać zaimplementowany w języku Java oraz ma korzystać z dodatkowych bibliotek dostępnych na licencji open source. System ten ma wykorzystywać programowanie logiczne z ograniczeniami.

1.3 Struktura pracy

W rozdziale 2 zamieszczono przegląd literatury na temat wytycznych postępowania klinicznego oraz ich roli w medycynie. W tym rozdziale zaprezentowano także podejścia do wykrywania i usuwania interakcji w wytycznych postępowania klinicznego. Rozdział 3 opisuje paradygmat programowania logicznego z ograniczeniami. Opisano w tym rozdziale podstawowe właściwości podejścia, a także zamieszczono przykład ilustrujący jego wykorzystanie. W rozdziale 4 zaprezentowano narzędzia i biblioteki użyte podczas wykonywania pracy magisterskiej. Rozdział 5 opisuje główną część pracy, czyli rozszerzenie podejścia do wykrywania i usuwania konfliktów oraz implementację systemu. W tym rozdziale przedstawiono poszczególne części systemu. Rozdział 6 prezentuje przykłady działania systemu. Rozdział 7 stanowi podsumowanie pracy.

Rozdział 2

Przegląd literatury

2.1 Wytyczne postępowania klinicznego

Wytyczne postępowania klinicznego[6, 7] to dokument, którego zadaniem jest pomoc lekarzom i personelowi medycznemu w podejmowaniu decyzji związanych z określonymi obszarami opieki medycznymi, w szczególności z leczeniem pacjentów z różnych dolegliwości. Kolejnym celem wytycznych jest poprawa jakości opieki medycznej poprzez jej standaryzację i zwiększenie wydajności personelu medycznego. Dzięki temu można obniżyć koszty usług czy zlecanych badań. Wytyczne są tworzone przez ekspertów medycznych. Rozwój wytycznych wysokiej jakości wymaga specjalistycznego zespołu ludzi i wystarczającego budżetu. Wytyczne postępowania klinicznego mogą mieć różną reprezentację. Do przykładowych reprezentacji należą dokumenty tekstowe oraz reprezentacje formalne takie, jak tablice decyzyjne czy formaty grafowe. Implementacja wytycznych w formie systemów komputerowych daje możliwość personalizacji wytycznych, czyli uwzględnia osobistą charakterystykę pacjenta. Komputerowo interpretowane wytyczne, które posiadają dostęp do elektronicznego rekordu pacjenta, są w stanie dostarczyć porady dotyczące konkretnego pacjenta. Niestety, wytyczne dotyczą z reguły jedynie konkretnej dolegliwości. W związku z tym, wytyczne w przypadku pacjentów chorujących na kilka chorób, którymi często są osoby starsze, mogą doprowadzić do niewłaściwego wyboru terapii. Ważnym aspektem w tej kwestii jest znajdowanie konfliktów występujących między wytycznymi i ich odpowiedniego rozwiązania.

2.2 Wykrywanie i usuwanie konfliktów w wytycznych postępowania klinicznego

2.2.1 Programowanie logiczne z ograniczeniami

Wytyczne postępowania klinicznego są w tej technice prezentowane w postaci grafu akcji.[12] Graf akcji jest grafem skierowanym, na który składają się trzy typy węzłów. Pierwszy typ to węzeł kontekstu. Jest to węzeł początkowy opisujący określoną chorobę. Kolejnym typem jest węzeł akcji, który opisuje akcję medyczną, którą należy wykonać. Ostatnim typem jest węzeł decyzji, który zawiera pytanie, na które należy odpowiedzieć. Graf akcji jest w tym podejściu transformowany następnie do modelu logicznego. Model ten składa się z wyrażeń logicznych opisujących poszczególne ścieżki w grafie. Akcje, których nie ma na ścieżce są zapisywane w wyrażeniu w postaci negacji.

W tym podejściu stosowane są także tzw. operatory interakcji i rewizji. Operator interakcji reprezentuje niepożądany konflikt (zazwyczaj lek-lek lub lek-choroba), który jest w postaci zdania zbudowanego z elementów wyrażeń logicznych modelu logicznego. Operator rewizji opisuje natomiast zmiany, jakie należy wprowadzić do modeli logicznych, aby konflikt usunąć. Modele logiczne reprezentujące wytyczne są zamieniane na programy CLP, które są automatycznie wykonywane. Uzyskane rozwiązanie wskazuje na ścieżki, jakie należy przejść podczas leczenia pacjenta.

2.2.2 Logika pierwszego rzędu

Wytyczne postępowania klinicznego są w tym podejściu opisane, podobnie jak w poprzednim podejściu, za pomocą grafu akcji z węzłami kontekstu, decyzji i akcji.[11] Słownictwo podejścia logiki pierwszego rzędu składa się ze stałych (pisanych dużymi literami), zmiennych (pisanych małymi literami) oraz predykatów. Do predykatów należą m.in. $\text{action}(x)$ - x jest węzłem akcji, $\text{decision}(x)$ - x jest węzłem decyzji, $\text{disease}(d)$ - d jest chorobą.

W tym podejściu także stosowane są operatory interakcji i rewizji. Operator interakcji opisuje konflikt. Operator rewizji składa się z dwóch części. Pierwsza część jest podobna do operatora interakcji, również zbudowana jest ze zdania opisującego konflikt, który może wystąpić. Druga część składa się z par formuł, które opisują pojedyncze zmiany. Pary formuł mogą być w trzech postaciach:

- (x, \emptyset) – oznacza, że formuła x jest usuwana
- (\emptyset, x) – oznacza, że formuła x jest dodawana
- (x, y) – oznacza, że formuła x jest zamieniana na formułę y

2.2.3 Podejście wykorzystujące paradygmat mieszanej inicjatywy

Podejście to polega na tym, że użytkownik współpracuje z systemem komputerowym, aby rozwiązać problem.[9] Oznacza to, że nie jest to podejście automatyczne. Ten sposób wykrywania i usuwania konfliktów oferuje techniki unikania i naprawiania konfliktów. Do technik unikania konfliktów tych należą: wybieranie bezpiecznej alternatywy oraz czasowe unikanie konfliktu. Wybieranie bezpiecznej alternatywy polega na wyborze alternatywnej ścieżki w wytycznych, która unika interakcji. Tymczasowe unikanie polega natomiast na stosowaniu leków w takich momentach czasu, w których nie występuje interakcja pomiędzy nimi. Do technik naprawiania konfliktów należą: modyfikacja dawek leków, monitorowanie efektów oraz osłabianie interakcji poprzez rozszerzenie zaleceń o dodatkowe akcje.

Wytyczne w tej metodzie wykrywania i usuwania konfliktów są reprezentowane w postaci hierarchicznego grafu składającego się z węzłów będących akcjami i krawędzi modelujących relacje między akcjami. Podejście rozróżnia akcje atomowe i akcje złożone (plany). Ontologie wykorzy-

stywane w tym podejściu są mocno zintegrowane z obecnymi ontologiami medycznymi, takimi jak SNOMED CT dla pojęć medycznych i ATC dla klasyfikacji leków.

Rozdział 3

Programowanie logiczne z ograniczeniami (CLP)

Programowanie logiczne z ograniczeniami[2] pozwala na użycie programowania logicznego do rozwiązywania problemów z ograniczeniami. Przykładowym ograniczeniem może być wyrażenie postaci $X+Y>5$. Program CLP składa się z następujących elementów:

- Skończonego zbioru zmiennych z wartościami ze skończonych dziedzin
- Zbioru ograniczeń między zmiennymi
- Rozwiązań problemu polegających na przypisaniu wartości do zmiennych, które spełniają ograniczenia

Przykładowym zastosowaniem programowania logicznego z ograniczeniami jest zagadka SEND + MORE = MONEY.[8] Zagadka ta polega na przypisaniu cyfr z zakresu od 0 do 9 do zmiennych odpowiadających literom zawartym w równaniu tak, aby równanie było spełnione. Każda litera ma swoją unikalną wartość cyfrową. Ponadto litery S i M mają wartości różne od 0.

```
SEND
+ MORE
-----
MONEY
```

Rozwiązaniem tego problemu jest następujący program napisany w ECLiPSe (jest to kompilator programów CLP, a nie popularne środowisko programistyczne Eclipse, bardziej szczegółowy opis tego programu znajduje się w punkcie 4.1):


```

:-lib(ic).
sendmore1(Digits):-
Digits = [S,E,N,D,M,O,R,Y],
Digits :: [0..9],
alldifferent(Digits),
S #\= 0,
M #\= 0,
1000*S + 100*E + 10*N + D
+ 1000*M + 100*O + 10*R + E
#= 10000*M + 1000*O + 100*N + 10*E + Y,
labeling(Digits).

```

Po skompilowaniu tego programu wystarczy wywołać funkcję `sendmore1(Digits)`, aby otrzymać rozwiązanie zagadki. Rozwiązaniem jest następujące przypisanie cyfr do zmiennych: $S=9$, $E=5$, $N=6$, $D=7$, $M=1$, $O=0$, $R=8$, $Y=2$.

Można zauważyć na podstawie przykładu, że komendy w ECLiPSe składają się ze zdań zakończonych kropką, poszczególne fragmenty zdań są oddzielone od siebie przecinkami. Znak równości między zmiennymi lub wartościami liczbowymi to „#=”, znak nierówności to „#\=”.

Można stosować także operatory `and` i `or` i przypisywać ich wartość do zmiennych za pomocą znaku równości.

CLP może być wykorzystane do rozwiązywania innych zadań.[10] Do przykładowych zadań należy popularne zadanie z farmerem, wilkiem, gęsią i kapustą. Polega ono na tym, że farmer posiada łódkę, na której może się zmieścić on i jeszcze jeden element. Zadaniem farmera jest przewieźć wszystkie elementy z jednej strony rzeki na drugą. Problem stanowi fakt, że farmer nie może zostawić wilka i gęsi samych, bo wilk zje gęś, a także nie może zostawić gęsi z kapustą, ponieważ gęś zje kapustę. Innym przykładem jest zadanie o nazwie osiem królowych. Polega ono na tym, że na szachownicy o wymiarach 8×8 należy umieścić 8 królowych w taki sposób, aby żadna z nich nie mogła zbić innej królowej.

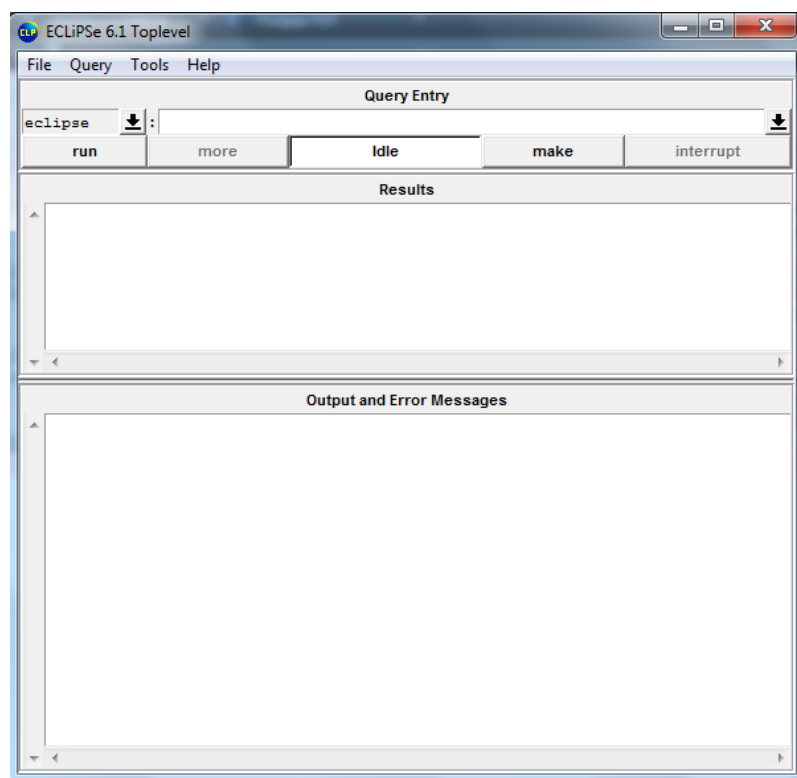
Programowanie logiczne z ograniczeniami może być również użyte do rozwiązywania trudniejszych problemów. Do problemów tych należą m.in. harmonogramowanie pracy lakierni samochodowej czy projektowanie inteligentnych systemów okablowania dla dużych budynków.

Rozdział 4

Wykorzystane biblioteki i narzędzia

4.1 ECLiPSe

ECLiPSe[5] jest systemem typu open-source do wykonywania aplikacji wykorzystujących paradygmat programowania logicznego z ograniczeniami. System ten użyty został do testowania przykładowych procedur medycznych. Nie współpracuje natomiast z wykonanym w ramach pracy magisterskiej systemem, jest to odrębny program. Okno systemu ECLiPSe składa się z trzech części. Pierwsza część służy do wprowadzania komend. Zamiast wprowadzania komend można wczytać gotowy program z pliku za pomocą polecenia Compile znajdującym się w menu File. Druga część programu wyświetla wyniki, trzecia natomiast pokazuje ewentualne błędy oraz inne komunikaty. Poniżej przedstawiono wygląd okna programu ECLiPSe.



4.2 Choco 3

Choco[1] jest darmowym oprogramowaniem typu open-source, które pozwala na rozwiązywanie problemów CLP. Jest to biblioteka oparta o język Java w wersji 8. Główną klasą biblioteki jest klasa `Solver`. Do obiektu typu `Solver` można dołączyć zmienną (klasa `IntVar`) podając obiekt `Solver-a` w ostatnim argumencie metody `VariableFactory.bounded`. Pozostałe argumenty tej metody to nazwa zmiennej oraz dolne i górne ograniczenie zmiennej. W pracy magisterskiej wykorzystywane są w większości zmienne, dla których dolne ograniczenie jest równe 0, a górne ograniczenie jest równe 1, czyli są to zmienne przyjmujące wartości prawda/fałsz. Za pomocą funkcji `Solver.post` można dodawać nowe ograniczenia. Ograniczenia tworzy się m.in. za pomocą klasy `IntConstraintFactory`. Jedną z podstawowych metod tworzących ograniczenia jest funkcja `arithm`. Przykładowo, można za jej pomocą określić, że suma dwóch zmiennych X i Y ma być mniejsza od 5. Po określeniu ograniczeń można uruchomić `Solver` i wygenerować rozwiązanie za pomocą metody `findSolution`. Kolejne rozwiązania można uzyskać za pomocą metody `nextSolution`. Odczytanie wartości zmiennej określonego rozwiązania polega na wywołaniu metody `IntVar.getValue`. Poniżej przedstawiono prosty program Choco3 szukający takich zmiennych X i Y (są to zmienne przyjmujące wartości 0 lub 1), których suma jest równa 1. Rozwiązaniem poniższego programu są dwa przypadki: $X=1, Y=0$ oraz $X=0, Y=1$.

```
Solver solver = new Solver("my first problem");
IntVar x = VariableFactory.bounded("X", 0, 1, solver);
IntVar y = VariableFactory.bounded("Y", 0, 1, solver);
solver.post(IntConstraintFactory.arithm(x, "+", y, "=", 1));
solver.findSolution();
do
{
    System.out.println("X="+x.getValue()+" , Y="+y.getValue());
}while(solver.nextSolution());
```

4.3 Graphviz

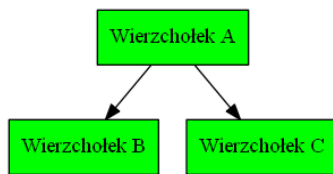
Graphviz[3] jest oprogramowaniem służącym do wizualizacji grafów. Pozwala na konwersję pliku tekstowego w formacie dot do obrazu przedstawiającego graf. Program automatycznie porządkuje węzły na obrazie, nie jest konieczne podawanie pozycji węzłów, czyli ich współrzędnych. Ponadto, program automatycznie rysuje krawędzie tak, aby ograniczyć liczbę ich przecięć. Program z pakietu Graphviz o nazwie `gvedit.exe` jest programem okienkowym, który pozwala na wybranie w oknie dialogowym pliku o rozszerzeniu dot. Po wybraniu tego pliku albo wypisywana jest lista błędów,

które należy poprawić, albo wyświetlany jest obraz przedstawiający graf. Podobną funkcjonalność ma program dot.exe, z tą różnicą, że jest to program konsolowy. Program dot.exe posiada 3 argumenty. Pierwszym argumentem jest ścieżka do pliku w formacie dot, drugim jest format generowanego obrazu (przykładowo dla uzyskania formatu png obrazka podajemy drugą wartość argumentu równą -Tpng). Między drugim a trzecim argumentem należy podać przełącznik „-o”. Trzecim argumentem jest ścieżka wynikowego obrazu.

Jeśli chodzi o plik w formacie dot, jest to plik, który posiada swoją własną składnię. Na początku pliku umieszczone jest słowo „digraph”, po którym umieszcza się nazwę grafu. Wszystkie pozostałe właściwości grafu są umieszczone w bloku otoczonym nawiasami klamrowymi. W bloku tym można podać globalne atrybuty dla węzłów oraz krawędzi. Atrybuty dla węzłów mogą być podane po słowie node w bloku otoczonym nawiasami kwadratowymi, atrybuty są oddzielone od siebie przecinkami. Do przykładowych globalnych atrybutów węzłów należą m. in. kształt (box – prostokąt, circle – koło, diamond – romb), kolor wypełnienia, kolor konturu, grubość linii konturu, rodzaj czcionki, wielkość czcionki. Jeśli chodzi o globalne atrybuty krawędzi, to można je podać w podobny sposób jak globalne atrybuty węzłów, z tą różnicą, że zamiast słowa „node” należy podać słowo „edge”. Do atrybutów globalnych krawędzi należą przede wszystkim wielkość i rodzaj czcionki (krawędzie mogą posiadać etykiety).

W następnym kroku można podać węzły i krawędzie z ich atrybutami. Atrybut pojedynczego węzła lub krawędzi, jeśli już wystąpił w globalnych atrybutach węzłów lub krawędzi, zostaje nadpisany. Opis pojedynczego węzła polega na podaniu jego unikalnego identyfikatora, a następnie jego atrybutów w bloku otoczonym nawiasami kwadratowymi (atrybuty są podawane po przecinku). Krawędzie natomiast tworzy się, podając na początku identyfikator węzła źródłowego krawędzi, następnie należy umieścić tzw. strzałkę („->”), a na końcu identyfikator węzła docelowego. Po podaniu tych elementów można podać atrybuty krawędzi, przede wszystkim etykietę. Co ciekawe, krawędź może być także nieskierowana, wtedy zamiast strzałki („->”) należy umieścić podwójną kreskę („-”). Poniżej zaprezentowano bardzo prosty przykład pliku w formacie dot i jego graf:

```
digraph graf{
    node [shape=box, style=filled, fillcolor=green];
    A [label="Wierzchołek A"];
    B [label="Wierzchołek B"];
    C [label="Wierzchołek C"];
    A->B;
    A->C;
}
```



4.4 JPGD - A Java parser for Graphviz documents

Biblioteka[4] ta służy do konwersji pliku o rozszerzeniu dot na obiekt klasy `Graph` posiadający listę obiektów klasy `Node` oraz `Edge`. Do konwersji wykorzystywany jest obiekt klasy `Parser`. Klasa `Parser` posiada funkcję `parse`, której konstruktor jako parametr przyjmuje obiekt klasy `FileReader` odwołujący się do określonego pliku o rozszerzeniu dot. W następnym kroku można odczytać obiekt klasy `Graph` z listy tych obiektów uzyskanej za pomocą funkcji `getGraphs` (jest to funkcja klasy `Parser`).

Węzły grafu można odczytać za pomocą funkcji `getNodes` wywołanej dla obiektu klasy `Graph`. Krawędzie grafu można natomiast uzyskać za pomocą funkcji `getEdges`, która również jest funkcją klasy `Graph`. Węzły oraz krawędzie posiadają atrybuty. Do atrybutów węzłów należy zaliczyć etykietę, kształt, kolor wypełnienia, kolor konturu i grubość linii konturu. Krawędzie posiadają przede wszystkim jeden istotny atrybut – etykietę. Odczytać wartości atrybutów można za pomocą funkcji `getAttribute`, której argumentem jest nazwa atrybutu. Ustawić wartości atrybutu można natomiast za pomocą metody `setAttribute`, której pierwszym argumentem jest nazwa atrybutu, a drugim jego wartość.

Każdy węzeł grafu będący w formacie dot posiada także swój unikalny identyfikator. Identyfikatory przechowywane są w obiektach klasy `Id`. Obiekt takiej klasy dla określonego węzła można uzyskać wywołując funkcję `getId` na rzecz obiektu klasy `Node`. Ponowne wywołanie funkcji `getId`, w tym przypadku dla obiektu klasy `Id` uzyskuje rzeczywisty identyfikator węzła typu `String`.

Jeśli chodzi o krawędzie, to posiadają one możliwość odczytania węzła źródłowego oraz docelowego danej krawędzi. Jest to możliwe dzięki wywołaniu funkcji `getSource` (dla uzyskania węzła źródłowego) oraz `getTarget` (dla uzyskania węzła docelowego). Dzięki tym funkcjom uzyskujemy obiekt klasy `PortNode`, z którego następnie możemy uzyskać obiekt klasy `Node` za pomocą funkcji `getNode`. Ważną funkcją jest też funkcja `toString` wywoływana na rzecz obiektu klasy `Graph`. Pozwala ona na uzyskanie grafu w formacie dot zawierającym zmiany wprowadzone za pomocą metody `setAttribute` dla obiektów klasy `Node` lub `Edge`.

Poniższy kod źródłowy prezentuje przykładowe wykorzystanie biblioteki JPGD do znalezienia krawędzi wyjściowych węzła n.

```
public static ArrayList<Edge> getOutEdges(Graph graph, Node n)
{
    ArrayList<Edge> list = new ArrayList<Edge>();
    for(Edge e:graph.getEdges())
    {
        if(e.getSource().getNode()==n)
        {
            list.add(e);
        }
    }
    return list;
}
```

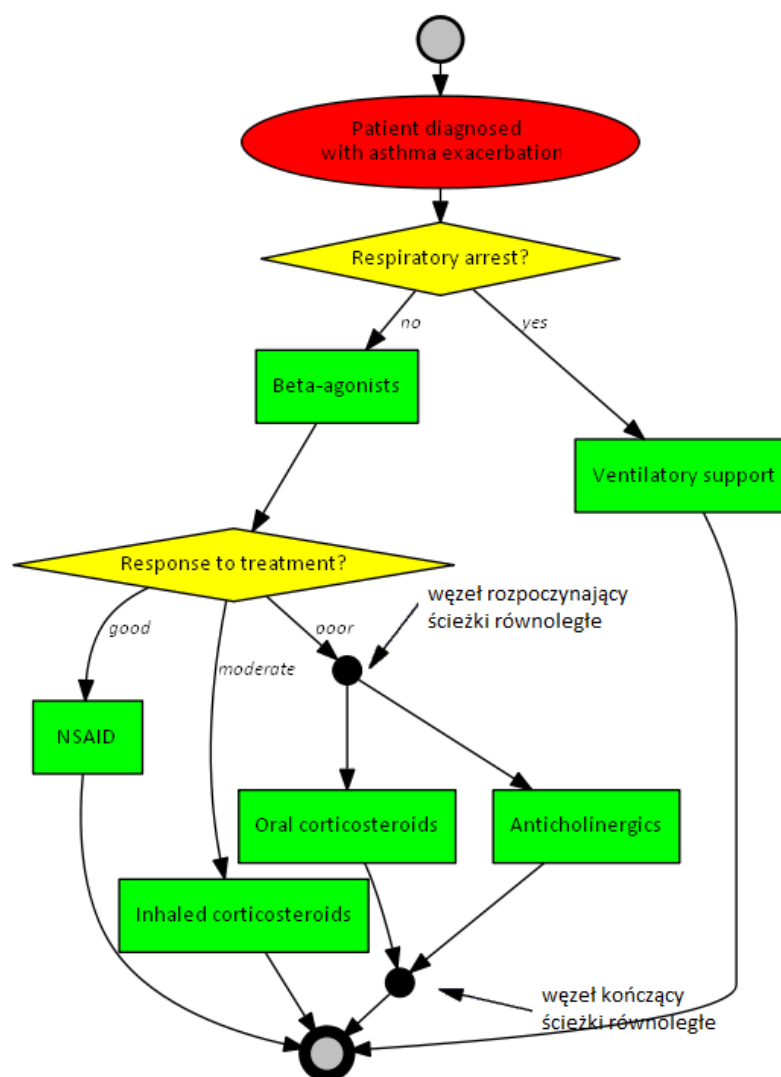
Rozdział 5

Implementacja

Rozdział ten opisuje zasadę działania programu. Program składa się z kilku kroków. Na początku użytkownik wybiera choroby, na które choruje pacjent. Następnie program wyświetla graficzną reprezentację wytycznych w formie grafu oraz wyświetla listy pól wyboru, które pozwalają na udzielanie odpowiedzi na pytania zawarte w wytycznych. Po udzieleniu odpowiedzi na pytania i kliknięciu przycisku „Dalej” program rozwiązuje problem CLP, w wyniku którego uzyskujemy listę konfliktów, które wystąpiły między wytycznymi oraz grafy wynikowe z wprowadzonymi zmianami pozwalającymi uniknąć konfliktu. Po uzyskaniu rozwiązania problemu można wybrać inne odpowiedzi na pytania i po kliknięciu przycisku „Dalej” wygenerować nowe rozwiązania problemu. Można także wybrać inne choroby za pomocą przycisku „Wybierz choroby”.

W tym rozdziale wykorzystywane są pojęcia „terapia” oraz „element terapii”. Terapia jest to pojedyncza ścieżka w grafie. Element terapii natomiast to identyfikator węzła w przypadku węzłów akcji, natomiast dla węzłów decyzji elementem terapii są identyfikator węzła i etykieta wybranej krawędzi oddzielone znakiem zapytania. Węzłami akcji są węzły, które zawierają informację o leku, jaki należy zażyć lub zabiegu, jaki należy przeprowadzić. Węzły decyzji są to z kolei węzły, które posiadają pytanie.

W grafach może zajść sytuacja, w której należy poruszać się po ścieżkach równoległych. Sytuacja taka występuje na rysunku 5.1. Węzeł rozpoczynający ścieżki równoległe charakteryzuje się tym, że posiada więcej niż jedną krawędź wyjściową oraz nie ma etykiety. Natomiast węzeł kończący ścieżki równoległe posiada więcej niż jedną krawędź wejściową, nie ma etykiety oraz liczba jego krawędzi wyjściowych jest większa od zera.



Rysunek 5.1: Przykład ścieżek równoległych

Program posiada 3 istotne katalogi, które przechowują różnego typu danych. Należą do nich:

- Algorytmy – zawiera pliki o rozszerzeniu dot opisujące grafy procedur medycznych chorób
- Konflikty – zawiera opisy konfliktów, jakie występują między chorobami oraz zmiany, które należy wprowadzić w przypadku wystąpienia konfliktów
- Grafy – zawiera zmodyfikowane grafy chorób przedstawiające aktualnie przebytą ścieżkę oraz grafy wynikowe prezentujące rozwiązania. Grafy są w dwóch formatach – tekstowym w formacie dot oraz graficznym w formacie png. Podczas zamykania programu zawartość tego katalogu jest kasowana

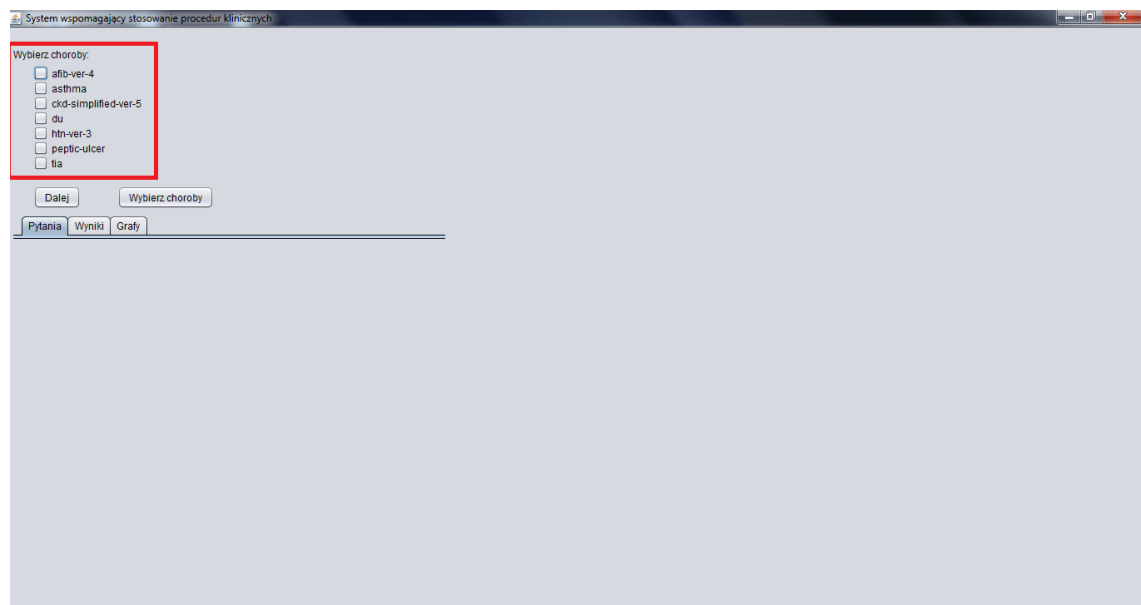
Program posiada następujące klasy:

- AddToTherapy - dodawanie identyfikatorów węzłów do listy opisującej konkretną terapię
- ChocoClass - rozwiązanie problemu CLP

- Color - kolorowanie wierzchołków i krawędzi grafów
- CreateTherapies - generowanie terapii
- ExecuteInteractions - wprowadzanie zmian w terapiach w przypadku wykrycia konfliktów
- GoForward - przechodzenie do kolejnego węzła decyzyjnego
- GraphFunctions - przydatne metody związane z grafami, np. znalezienie węzłów docelowych określonego węzła
- ImageGraph - wyświetlanie grafów
- MainClass - obsługa zdarzenia kliknięcia przycisku Dalej
- RadioButtonList - tworzenie i obsługa zdarzeń list pól wyboru służących do udzielania odpowiedzi na pytania
- Results - wyświetlanie wyników
- Window - okno programu

5.1 Wybór chorób

Celem tego kroku jest wybór tych wytycznych, które będą brane pod uwagę przy ustalaniu terapii. W katalogu Algorytmy program szuka plików posiadających rozszerzenie dot. Dla każdego takiego pliku tworzone jest pole wyboru. Pole wyboru posiada etykietę równą nazwie choroby. Utworzone pole wyboru jest następnie dodawane do globalnej listy pól wyboru o nazwie `checkBoxGroup` oraz do panelu znajdującego się w lewym górnym rogu okna programu (rys. 5.2).



Rysunek 5.2: Panel wyboru chorób

Po wybraniu chorób, tzn. po kliknięciu w odpowiednie pola wyboru i kliknięciu przycisku

„Dalej”, nazwy wybranych chorób są dodawane do listy o nazwie `selectedDiseases` i program przechodzi do fazy wyświetlania grafów oraz udzielania odpowiedzi na pytania.

5.2 Wyświetlanie grafów oraz udzielanie odpowiedzi na pytania

Ten krok pozwala na utworzenie graficznej reprezentacji wytycznych oraz wybór ścieżki grafu zgodnej z danymi pacjenta. Po wybraniu chorób i kliknięciu przycisku „Dalej” program dla każdej choroby odczytuje za pomocą metody `GraphFunctions.getGraph` grafy z pliku o rozszerzeniu `dot`. Następnie, program pobiera korzenie każdego z grafów za pomocą metody `GraphFunctions.getStartNode`. W następnym etapie program wywołuje metodę `GoForward.goForward`, która przemieszcza się po grafie do momentu, gdy napotka pierwszy węzeł decyzyjny. Parametrem metody `goForward` jest `m.in.` korzeń grafu.

Najpierw `goForward` wywołuje metodę `addToTherapy`. Metoda `addToTherapy` jest wywoływana, jeśli węzeł `next` zawiera nie więcej niż jedną krawędź wyjściową lub węzeł `next` jest węzłem zaczynającym ścieżki równoległe i nie jest to węzeł kończący ścieżki równoległe w przypadku, gdy program nie przeszedł jeszcze wszystkich ścieżek równoległych w określonym miejscu. Metoda `addToTherapy`, jeśli etykieta węzła posiada wartość dawki podaną w nawiasach kwadratowych, dodaje do listy elementów terapii identyfikator węzła, po nim znak równości, a następnie wartość dawki. Gdy węzeł nie posiada podanej dawki, do listy elementów terapii dodawany jest tylko jego identyfikator.

Następnie program wykonuje pętlę `while`, której warunek kontynuacji obejmuje 3 przypadki. Pętla ta stanowi część metody `goForward`. Pierwszy warunek sprawdza, czy węzeł posiada jedną krawędź wyjściową i nie jest to węzeł kończący ścieżki równoległe chyba, że program zakończył przechodzenie po ścieżkach równoległych związanych z tym węzłem. Drugi warunek sprawdza, czy węzeł rozpoczyna ścieżki równoległe, a trzeci czy węzeł kończy ścieżki równoległe.

Pierwszy warunek jest sprawdzany w kolejnej pętli `while`, aby dodać do listy elementów terapii wszystkie węzły, które mają tylko jedną krawędź wyjściową, czyli droga, po której należy się poruszać jest jednoznacznie określona. Po wykonaniu tej pętli `while` uzyskujemy węzeł, który jest liściem (nie posiada żadnej krawędzi wyjściowej), albo ma więcej niż jedną krawędź wyjściową.

W kolejnym kroku sprawdzany jest dla aktualnego węzła drugi warunek w instrukcji `if`, czyli czy węzeł rozpoczyna ścieżki równoległe. Jeśli jest on spełniony, wywoływana jest metoda `parallelPath`. Metoda ta jest wywoływana również w dla trzeciego przypadku, czyli gdy uzyskany węzeł kończy ścieżki równoległe, a program nie przeszedł jeszcze przez wszystkie te ścieżki. Metoda `parallelPath` jest w postaci pętli `while`, która działa dopóki program nie przejdzie przez wszystkie ścieżki równoległe związane z węzłem rozpoczynającym ścieżki równoległe i uzyskany węzeł nie jest

węzłem zawierającym pytanie. Pętla `while` zapisuje do listy elementów terapii wszystkie przebyte po drodze węzły.

Po wywołaniu metody `goForward` program wywołuje metodę `Color.color`, która zaznacza przebytą ścieżkę w grafie. Metoda `goForward` do listy `list` o nazwie `dataIdList` dodawała identyfikatory węzłów, na które program natrafił. Dzięki `dataIdList` metoda `color` może pokolorować kontury przebytych węzłów oraz przebyte krawędzie, a także je pogrubić.

Po wywołaniu metody `color` wywołana zostaje metoda `ImageGraph.newImageGraph`, której zadaniem jest wygenerowanie i wyświetlenie nowego obrazu grafu. Na początku metoda zapisuje do pliku wynik metody `toString` wywołanej dla grafu. Następnie wywoływana jest metoda `ImageGraph.getImageGraphPath`, która uruchamia program `dot.exe` i tworzy z zapisanego wcześniej pliku tekstowego graf w postaci obrazu w formacie PNG. W kolejnym kroku metoda `ImageGraph.newImageGraph` tworzy obiekt klasy `BufferedImage` z wygenerowanym w poprzednim kroku obrazem. Później metoda dokonuje skalowania obrazu tak, aby mógł on się zmieścić na etykiecie.

Jeśli szerokość lub wysokość obrazu przekracza próg 900 pikseli, obraz zmniejszany jest do 2/3 wielkości tak, aby był on czytelny (w tym przypadku do etykiety dodawane są suwaki). Ponadto, jeżeli szerokość i wysokość obrazu jest mniejsza od wielkości etykiety, to na etykiecie umieszczany jest obraz bez skalowania, w skali 1:1.

Ostatnim krokiem jest wywołanie metody `RadioButtonList.createRadioButtonList`. Metoda ta dla każdego elementu terapii, który posiada znak zapytania tworzy panel. Elementy terapii zawierające znak zapytania odpowiadają krokom decyzyjnym. Pierwszym elementem panelu jest etykieta węzła. Pozostałe elementy stanowią pola wyboru z etykietami, których wartości są równe etykietom krawędzi węzła decyzyjnego. Do tych pól wyboru dodawany jest jeszcze jeden z etykietą „brak wartości”, przydatny w sytuacji, gdy nie znamy jeszcze danych. Część elementu terapii po znaku zapytania pozwala na przechowanie informacji o wyborze aktywnego pola wyboru. Na końcu tworzony jest jeszcze jeden panel, tym razem dla pytania, na które jeszcze nie została udzielona odpowiedź, dla niego zaznaczone jest pole wyboru z etykietą „brak wartości”. Przy pierwszym wyświetleniu grafu tworzony jest tylko ten panel. Ponadto, dla każdego pola wyboru przypisywane jest zdarzenie `RadioButtonList.updateRadioButtonList`.

Metoda `updateRadioButtonList` na początku szuka elementu w liście elementów terapii, którego dotyczy pytanie. Jeśli zaznaczone pole wyboru ma etykietę „brak wartości”, usuwane są wszystkie elementy terapii od elementu, którego dotyczy pytanie, do ostatniego elementu listy. W sytuacji tej cofamy się z udzielaniem odpowiedzi na pytania do jednego z poprzednich pytań.

Jeśli natomiast zaznaczone pole wyboru nie posiada etykiety równej „brak wartości” i nie istnieje element na liście elementów terapii, który jest związany z pytaniem, to do tej listy dodawany jest element o wartości równej `question?answer`, gdzie `answer` jest parametrem metody

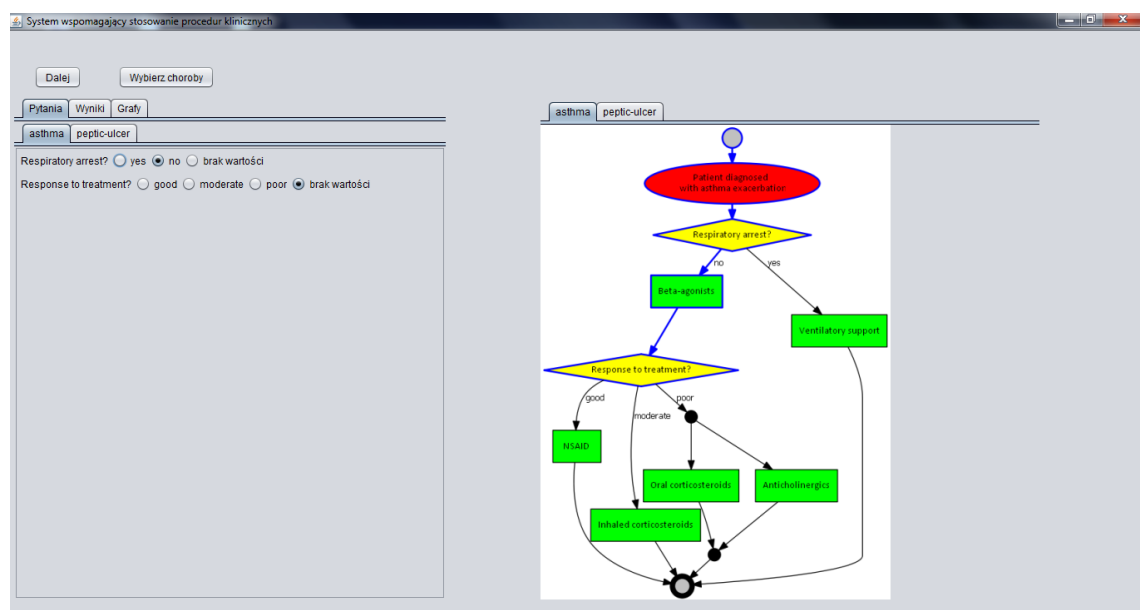
`updateRadioButtonList` o wartości równej etykiecie krawędzi, z którą jest związane zaznaczone pole wyboru. W sytuacji tej użytkownik udziela po raz pierwszy odpowiedzi na pytanie.

Jeśli istnieje element związany z pytaniem, to w liście elementów terapii podmieniany jest element, który jest związany z pytaniem na wartość `question?answer`, a następnie usuwane są wszystkie elementy listy terapii, które się znajdują za podmienionym elementem.

Następnie pod węzeł o nazwie `next` podstawiany jest węzeł, który jest pierwszym węzłem krawędzi o etykiecie `answer` wychodzącej z węzła o identyfikatorze `question`. W kolejnym kroku wywoływana jest metoda `goForward`, która jako parametr przyjmuje m.in. węzeł `next`.

Metoda `updateRadioButtonList` w kolejnym kroku odznacza wszystkie węzły i krawędzie i następnie koloruje je na nowo na podstawie listy elementów terapii. Tworzony jest także nowy obraz grafu za pomocą metody `newImageGraph`. Na końcu tworzona jest nowa lista pytań i odpowiedzi za pomocą metody `createRadioButtonList`.

Ostatecznie grafy prezentowane są po prawej stronie ekranu na zakładkach. Każda zakładka dotyczy wytycznych związanych z jedną z wybranych chorób. Z lewej strony ekranu pojawiają się natomiast zakładki z listami pól wyboru. W tym przypadku również jedna zakładka dotyczy jednej choroby. Listy pól wyboru pozwalają na udzielanie odpowiedzi na pytania zawarte w wytycznych klinicznych. Przykład wyświetlanych grafów przedstawiono na rys. 5.3.



Rysunek 5.3: Wyświetlanie grafów

5.3 Wyszukiwanie konfliktów

Celem tego kroku jest znalezienie konfliktów występujących między wytycznymi. Na początku program szuka w katalogu konflikty plików o rozszerzeniu `txt` oprócz pliku `nazwy.txt`. Następnie

program sprawdza, czy można użyć pliku z konfliktami. Nazwa każdego pliku z konfliktami składa się z listy chorób oddzielonych przecinkami, których dotyczą konflikty. Jeśli jakaś choroba z tej listy znajduje się w wybranych podczas działania programu chorobach, plik zostaje użyty. Każdy plik z konfliktami składa się z linii złożonych z dwóch części. Pierwsza część zawiera elementy, których jednoczesne wystąpienie powoduje wystąpienie konfliktu. Elementy te są oddzielone spacją. Druga część linii zawiera zmiany, jakie należy wprowadzić w przypadku zaistnienia konfliktu. Zmiany te są oddzielone od siebie przecinkami. Jeśli plik zostaje użyty, do listy `conflictsList` dodawane są konflikty, a do listy `interactionsList` zmiany. Ponadto, do listy `additionalQuestions` dodawane są elementy opisujące konflikt, które rozpoczynają się znakiem „&”. Dla elementów tych będzie trzeba udzielić odpowiedzi, ponieważ są to dane, które nie wystąpiły jawnie w wytycznych. Elementy konfliktów rozpoczynające się od „not” dodawane są do listy `notConflictElems`. Następnie program tworzy okienko dialogowe, które pozwala udzielić odpowiedzi na dodatkowe pytania. Pytania mogą być dwóch typów. Pierwszy typ występuje, gdy element nie posiada znaku równości, mniejszości ani większości. Wtedy udzielana odpowiedź ma postać tak/nie. Drugi typ to „zmienna operator liczba”. Operator może być postaci „=”, „>”, „<”, „>=” lub „<=”. Dla tego typu elementu podawana jest wartość liczbową w okienku dialogowym, a program sprawdza czy podana liczba spełnia warunek występujący w elemencie.

Po udzieleniu odpowiedzi na wszystkie dodatkowe pytania program przechodzi do kolejnej części wyszukiwania konfliktów o nazwie `solveNextPart`. Metoda `solveNextPart` najpierw wywołuje metodę `findSolutions`. Metoda ta dla każdego konfliktu wykonuje szereg operacji. Najpierw metoda sprawdza, czy konflikt znajduje się na liście `foundConflicts`. Lista `foundConflicts` zawiera konflikty, które zostały znalezione przez program. Jeśli konflikt nie znajduje się na tej liście tworzony jest obiekt klasy `Solver`. Następnie dodawane są zmienne na podstawie wcześniej udzielonych odpowiedzi na dodatkowe pytania. Dokonuje tego metoda `setAdditionalVariables`. Metoda ta sprawdza, czy pytanie jest typu tak/nie czy odpowiedzią na pytanie jest wartość liczbową. W pierwszej sytuacji, jeśli odpowiedź jest równa tak, program tworzy zmienną biblioteki Choco typu `IntVar` o wartości równej jeden. Jeżeli odpowiedź jest równa nie, program tworzy zmienną `IntVar` o wartości równej zero. Jeśli odpowiedzią na pytanie jest wartość liczbową, program tworzy zmienną `IntVar` o wartości równej podanej liczbie.

Po wykonaniu metody `setAdditionalVariables` program wywołuje metodę `setVariables`. Metoda ta dla każdej choroby tworzy tablicę terapii. W kolejnym kroku dla każdej terapii choroby tworzona jest zmienna `IntVar` o nazwie „choroba_terapiaX”, gdzie choroba jest nazwą choroby, a X jest numerem terapii. Zmienna ta przyjmuje wartości zero, gdy określona terapia nie zostaje użyta lub jeden, gdy zostaje użyta. Zmienna jest zapisywana w tablicy terapii. Następnie program tworzy listę `notConflictElemsTherapy`, do której dodawane są te elementy konfliktu z listy `notConflictElems`, które nie znajdują się na liście elementów konkretnej terapii, ale znajdują się w

grafie związanym z terapią. Elementy listy `notConflictElemsTherapy` są unikalne, nie powtarzają się. Następnie program tworzy tablicę `vars`, która będzie zawierała zmienne wchodzące w skład pojedynczej terapii. W kolejnym kroku dla każdego elementu listy `notConflictElemsTherapy` w tablicy `vars` zapisywane są zmienne o nazwie „not_X”, gdzie X jest elementem terapii z `notConflictElemsTherapy`. Zmienna ta przyjmuje wartość 0, gdy zmienna związana z elementem terapii jest równa 1 i odwrotnie. Ponadto, dla każdego elementu terapii program zapisuje zmienną w tablicy `vars`. Jeśli zmienna posiada dawkę, tworzona jest zmienna „X_dosage”, gdzie X jest elementem terapii. Po dodaniu wszystkich elementów określonej terapii program dodaje do `vars` zmienną dodatkową, która pozwala na skorzystanie tylko z jednej terapii dla każdej z chorób. Następnie program dodaje ograniczenie polegające na tym, że zmienna „choroba_terapiaX” przyjmuje wartość jeden, gdy suma zmiennych należących do tablicy `vars` jest równa wielkości tej tablicy. Wartość zero przyjmuje zmienna terapii w przeciwnym razie. Po przejściu przez wszystkie terapie określonej choroby program dodaje ograniczenie polegające na tym, że suma zmiennych terapii choroby ma być równa jeden, czyli dla każdej choroby ma zostać użyta tylko jedna terapia.

Po wykonaniu metody `setVariables` program dodaje ograniczenia konfliktów. Najpierw dodawane są ograniczenia tych konfliktów, które występowały w poprzednich iteracjach pętli `for` i po których dodaniu zostało wygenerowane poprawne rozwiązanie (są to konflikty, które znajdują się na liście `avoidedConflicts`). Następnie dodawane jest ograniczenie konfliktu, które odpowiada aktualnej iteracji pętli. Metoda dodawania ograniczeń konfliktu ma nazwę `setConflictConstraint`. Metoda ta najpierw tworzy listę o nazwie `constraintsList`. Następnie metoda wywołuje pętlę `for` dla każdego elementu wchodzącego w skład konfliktu. Pętla ta najpierw sprawdza czy element konfliktu zawiera znak równości, mniejszości lub większości. Jeśli tak nie jest, ale element konfliktu rozpoczyna się od operatora „not”, program dodaje ograniczenie „not(zmienna=1)” do listy `constraintsList`. Jeśli element konfliktu nie zawiera znaku równości, mniejszości ani większości i nie rozpoczyna się od „not”, program dodaje ograniczenie postaci „zmienna=1” do listy `constraintsList`. Jeżeli natomiast element konfliktu zawiera znak równości, mniejszości lub większości, metoda wywołuje metodę o nazwie `conflictWithDosage`. Metoda ta dodaje ograniczenie postaci „zmienna operator wartość” do listy `constraintsList` w przypadku, gdy nazwa zmiennej rozpoczyna się od „&”, czyli jest to zmienna dodatkowa, na którą udzieliło się odpowiedzi w okienku dialogowym. Jeśli zmienna nie rozpoczyna się od „&”, dodawane są dwa ograniczenia do `constraintsList`. Pierwsze ograniczenie jest postaci „zmienna=1”, drugie natomiast jest postaci „zmienna_dosage operator wartość”. Oba te ograniczenia są dodawane do listy `constraintsList`. Po wykonaniu pętli dla każdego elementu konfliktu metoda tworzy z listy `constraintsList` tablicę, a następnie dodaje do solvera ograniczenie postaci `not(and(ograniczenia))`.

W następnym kroku program wywołuje metodę `findSolution` obiektu klasy `Solver`, które dokonuje znalezienia rozwiązania problemu. Jeśli rozwiązanie istnieje, do `avoidedConflicts` do-

dawany jest numer konfliktu na liście `conflictsList`. Jeśli natomiast nie ma rozwiązania, do `foundConflicts` dodawany jest konflikt oraz do `interactionsList` dodawane są zmiany odpowiadające konfliktowi. Ponadto, gdy nie ma rozwiązania program wywołuje metodę `executeInteractions`, która dokonuje zmian w terapiach, a także program wywołuje rekurencyjnie metodę `findSolutions`, aby sprawdzić, czy wprowadzone zmiany nie spowodowały wystąpienia konfliktów, dla których już dokonało się przeglądu, oraz aby sprawdzić kolejne konflikty. Po wywołaniu metody `findSolutions` program ustawia wartość `stop` na `true`, co powoduje, że program nie sprawdza wystąpienia kolejnych konfliktów, ponieważ zrobiła już to wywołana rekurencyjnie metoda `findSolutions`.

Ostatecznie, program dokonuje rozwiązania problemu z tymi ograniczeniami w postaci konfliktów, które znajdują się na liście `avoidedConflicts`. Po wygenerowaniu pierwszego rozwiązania program tworzy listę o nazwie `solutions`. Następnie w pętli `do/while`, która działa dopóki istnieje kolejne rozwiązanie, program zapisuje do zmiennej `solution` po przecinku nazwy zmiennych terapii, które posiadają wartość równą jeden. Następnie, jeśli zmienna `solution` nie znajduje się jeszcze w liście `solutions`, zmienna dodawana jest do tej listy. Na końcu program do listy `therapies` dodaje rozwiązania. Polega to na tym, że dla każdego elementu listy `solutions` o nazwie `elem` program tworzy listę o nazwie `therapiesSolution`. Następnie tworzy tablicę o nazwie `array` zawierającą elementy, które były w zmiennej `elem` rozdzielone przecinkami. W kolejnym kroku dla każdej choroby znajdującej się w liście `diseases` szuka elementu w tablicy `array`, którego nazwa rozpoczyna się od nazwy choroby. Następnie do listy `therapiesSolution` dodaje listę z `therapiesDiseases` o numerze równym numerowi choroby i podnumerze równym `X` znajdującym się w nazwie zmiennej terapii „choroba_terapiaX”. Na końcu program wywołuje metodę `setResults`, która pozwala na zaprezentowanie wyników.

5.4 Wyświetlanie wyników

Ostatni krok polega na wyświetleniu grafów wynikowych prezentujących rozwiązania, a także utworzeniu listy znalezionych konfliktów wraz z wprowadzanymi zmianami. Program prezentuje wyniki za pomocą metody `setResults`. Na początku metoda wywołuje inną metodę o nazwie `setGraphs`. Zajmuje się ona wyświetleniem wyników w postaci grafów. Najpierw metoda usuwa wszystkie zakładki z rozwiązaniami z zakładki „Grafy”. W kolejnym kroku wywołuje dwie pętle, z których druga się zawiera w pierwszej. Pierwsza pętla porusza się po rozwiązaniach, druga po terapiach pojedynczego rozwiązania. Dla każdej terapii, która jest związana z określoną chorobą, tworzony jest odpowiedni graf. Następnie metoda wywołuje pętlę po grupach zmian poszczególnych konfliktów oraz po pojedynczych zmianach.

Dla każdej zmiany sprawdzany jest jej typ. Zmiany mogą być kilku typów. Pierwszy typ

to „replace X with Y”, który polega na tym, że węzeł X zamienia się na węzeł Y. Kolejny typ to „add X before/after Y”, który charakteryzuje się tym, że węzeł X jest dodawany przed lub po elemencie Y w zależności od tego, czy zostało użyte before czy after. Kolejnym typem jest „remove X”, które polega na usunięciu węzła X. Istnieją jeszcze typy zmian postaci „increase_dosage X Y”, „decrease_dosage X Y”, które powodują zwiększenie lub zmniejszenie dawki węzła X o Y. Ostatnim typem jest „change_dosage X Y”, które polega na zmianie dawki węzła X na wartość Y.

Jeśli zmiana jest typu „replace”, najpierw program szuka węzła o identyfikatorze równym elementowi, który należy zamienić. Po znalezieniu takiego węzła z pliku nazwy.txt odczytywana jest etykieta elementu, który ma się znaleźć na miejscu zamienianego elementu. Etykieta jest następnie zapisywana jako etykieta znalezionego węzła. Ponadto, program zamienia identyfikator znalezionego węzła na nowy. Jeśli zmiana jest typu „add”, program szuka węzła, przed lub za którym ma zostać umieszczony nowy węzeł. Następnie program tworzy nowy węzeł, nadaje mu etykietę pobraną z pliku nazwy.txt i dodaje węzeł do grafu. Następnie, jeżeli element, względem którego ma zostać wstawiony nowy węzeł jest postaci „pytanie?odpowiedź”, dla krawędzi, która ma etykietę „odpowiedź”, program ustawia węzeł docelowy krawędzi na nowo utworzony węzeł. Następnie, program tworzy nową krawędź, której węzłem źródłowym jest nowo utworzony węzeł, a węzłem docelowym jest węzeł docelowy krawędzi o etykiecie „odpowiedź”. Powoduje to umieszczenie nowego węzła na początku krawędzi z etykietą „odpowiedź”. Jeżeli natomiast element, względem którego ma być wstawiony nowy węzeł nie jest typu „pytanie?odpowiedź”, mogą wystąpić dwa przypadki. Pierwszy przypadek występuje, gdy zmiana jest typu „add X after Y”, drugi, gdy zmiana jest typu „add X before Y”. Dla tych sytuacji nowy węzeł jest umieszczany odpowiednio za lub przed węzłem Y.

W przypadku, gdy typ zmiany to „remove”, program przypisuje znalezionemu węzłowi atrybut style na wartość „invis”, atrybut fixedsize na „true” oraz atrybuty height i width na „0”. Powoduje to, że węzeł usunięty staje się niewidoczny na tworzonym grafie. Dla typów zmian „increase_dosage”, „decrease_dosage” i „change_dosage” odpowiednio zmienia się końcową część etykiety z nawiasami kwadratowymi, która przedstawia dawkę. Na końcu metoda `setGraphs` dla każdego grafu wywołuje metodę `color` zaznaczającą przebyte węzły i krawędzie, a następnie metodę `newImageGraph`, która powoduje wygenerowanie grafu w postaci obrazkowej.

Po wywołaniu metody `setGraphs` program tworzy rozwiązania tekstowe. Polega to na utworzeniu dla każdego rozwiązania i dla każdej choroby pojedynczego rozwiązania dwóch pól tekstowych. Pierwsze z nich przedstawia etykiety przebytych węzłów, a drugie ich identyfikatory. Etykietę węzła otrzymuje się przez znalezienie węzła o identyfikatorze równym elementowi terapii, a następnie odczytanie jego etykiety. Dla węzła pytającego program wypisuje dodatkowo etykietę wybranej krawędzi. Jeśli jest taka potrzeba, program szuka etykiety węzła w pliku nazwy.txt. Drugie pole tekstowe, z identyfikatorami, program tworzy wypisując elementy terapii z listy. Elementy są od-

dzielone od siebie znakiem nowej linii. Po utworzeniu pól tekstowych program umieszcza je na panelu, który jest z kolei umieszczany na zakładce. Na końcu program wypisuje w odpowiednim polu tekstowym konflikty i związane z nimi zmiany. Podczas wypisywania konfliktów i zmian program szuka odpowiednich etykiet węzłów w grafie lub w pliku nazwy.txt.

Rozdział 6

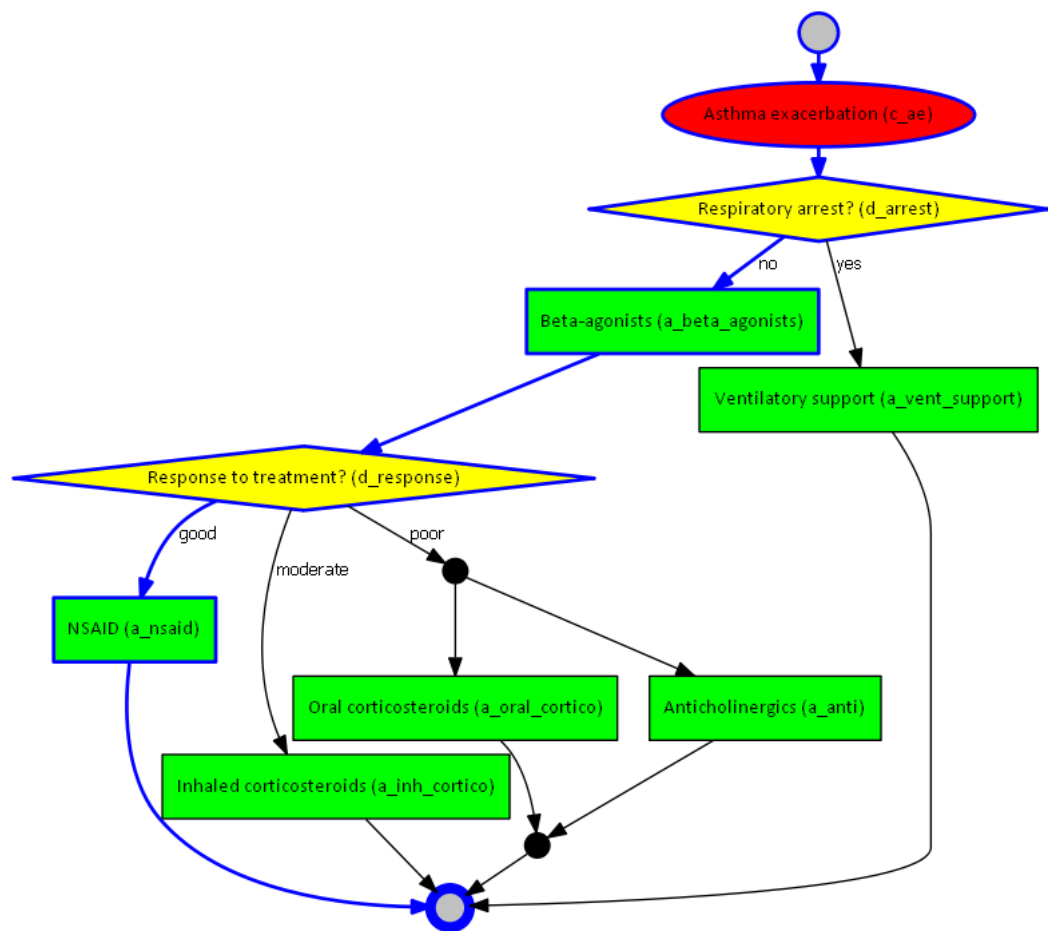
Działanie systemu

Działanie systemu zostanie zaprezentowane w oparciu o kilka przykładów. Dla każdego przykładu przedstawione zostały grafy z zaznaczonymi ścieżkami, które zostały wybrane podczas etapu udzielania odpowiedzi. Odpowiedzi na pytania są także przedstawione w formie tekstowej. Następnie, dla każdego przykładu przedstawiono listę konfliktów, które mogą wystąpić oraz zmiany, jakie należy wprowadzić w przypadku ich wystąpienia. Pogrubioną czcionką zaznaczono znalezione konflikty.

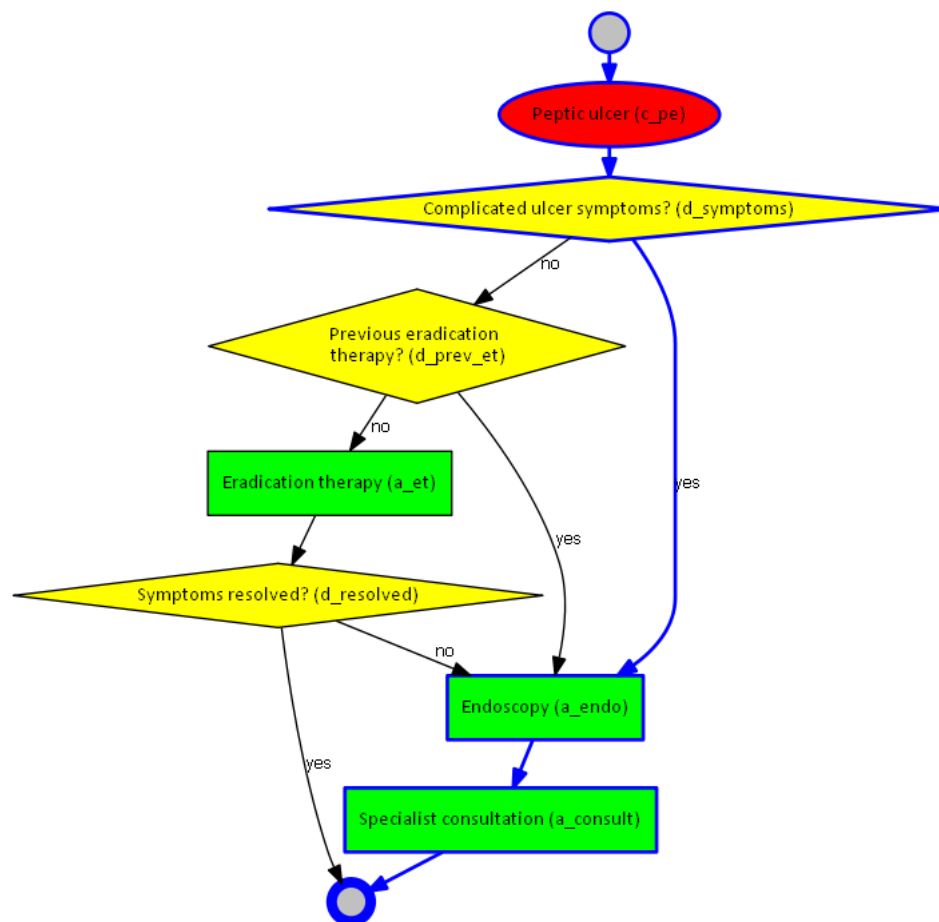
6.1 Przypadek 1 - atak astmy i wrzód trawienny

Wytyczne dla ataku astmy (ang. *asthma exacerbation*) przedstawiono na rys. 6.1.

Wytyczne dla wrzodu trawiennego (ang. *peptic ulcer*) przedstawiono na rys. 6.2.



Rysunek 6.1: Wytyczne dla ataku astmy



Rysunek 6.2: Wytyczne dla wrzodu trawiennego

Odpowiedzi na pytania:

1. Atak astmy:
 - Respiratory arrest?: no
 - Response to treatment?: good
2. Wrzód trawienny:
 - Complicated ulcer symptoms?: yes

Konflikty:

c_pe a_oral_cortico:replace a_oral_cortico with a_inh_cortico2

c_pe a_nsaid:add ppi after a_nsaid

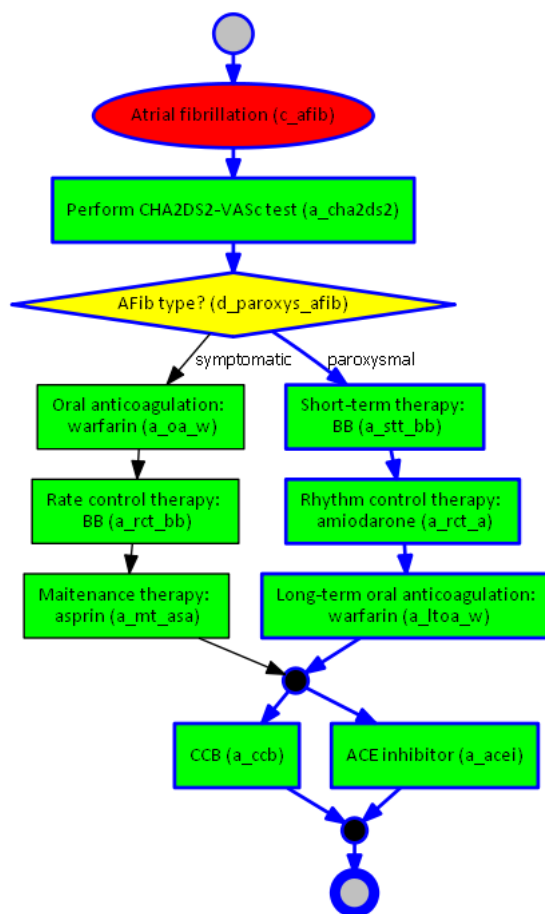
a_et a_inh_cortico:replace a_inh_cortico with a_oral_cortico2

6.2 Przypadek 2 - migotanie przedsionków, przewlekła choroba nerek i nadciśnienie

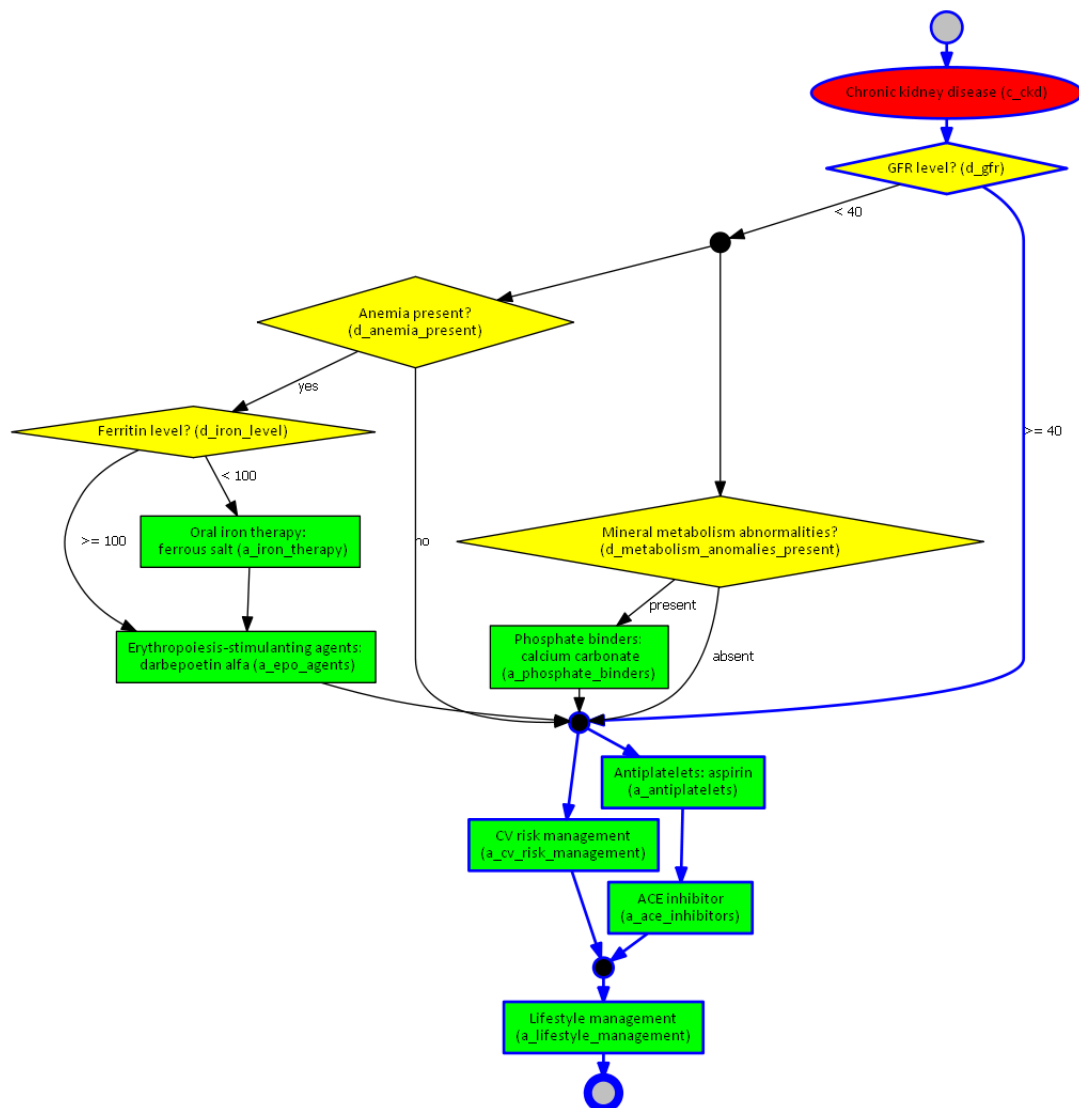
Wytyczne dla migotania przedsionków (ang. *atrial fibrillation*) przedstawiono na rys. 6.3.

Wytyczne dla przewlekłej choroby nerek (ang. *chronic kidney disease*) przedstawiono na rys. 6.5.

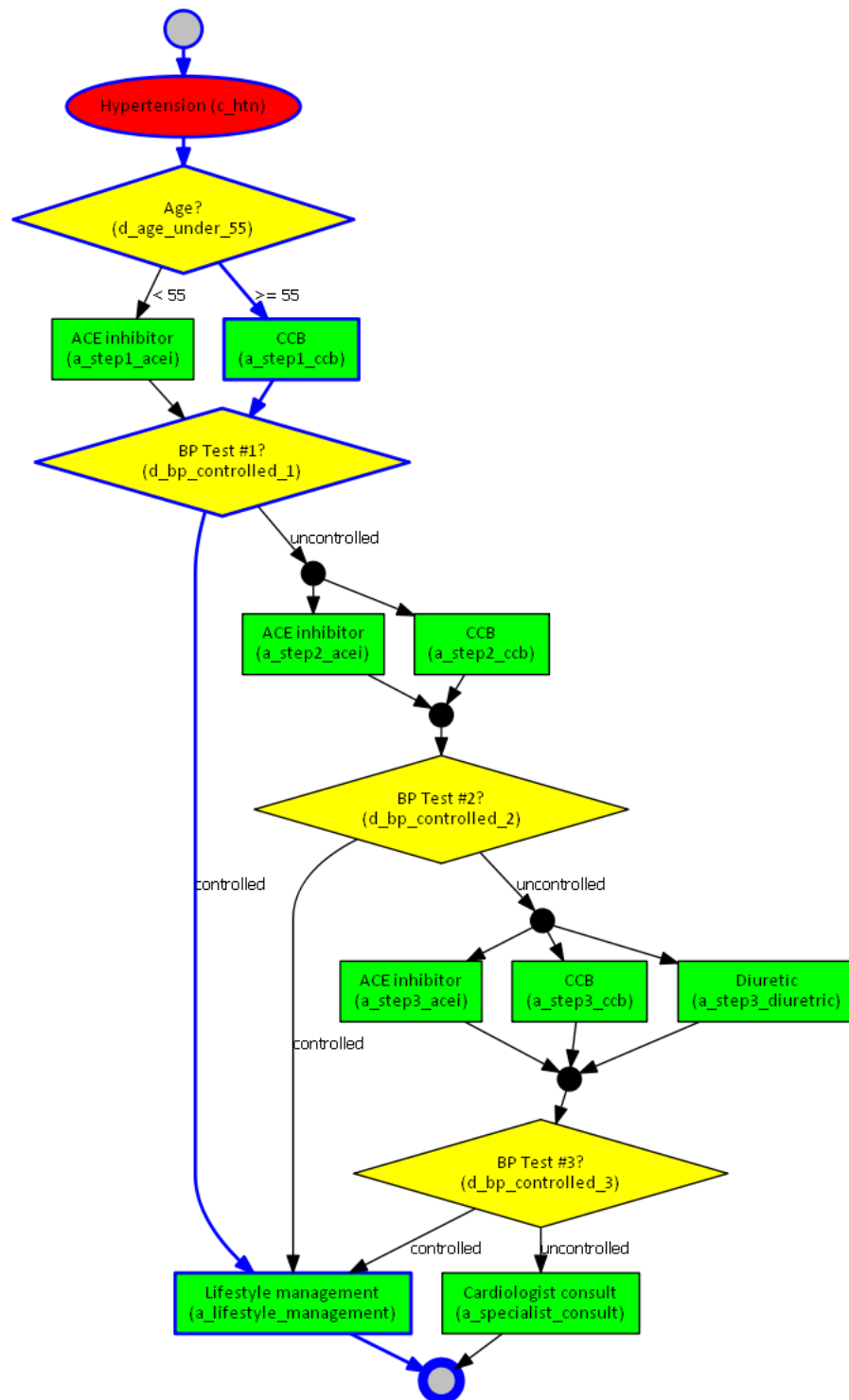
Wytyczne dla nadciśnienia (ang. *hypertension*) przedstawiono na rys. 6.5.



Rysunek 6.3: Wytyczne dla migotania przedsionków



Rysunek 6.4: Wytyczne dla przewlekłej choroby nerek



Rysunek 6.5: Wytyczne dla nadciśnienia

Odpowiedzi na pytania:

1. Migotanie przedsionków:
 - AFib type?: paroxysmal
2. Przewlekła choroba nerek:
 - GFR level?: ≥ 40
3. Nadciśnienie:
 - Age?: ≥ 55
 - BP Test #1?: controlled
4. CHA2DS2-VASc = 5 (Parametr ten nie pojawia się jawnie w wytycznych, użytkownik jest proszony o podanie tej wartości)

Konflikty:

c_htn c_ckd:remove a_step1_acei,remove a_step1_ccb

c_afib c_ckd c_htn:remove a_step3_diuretic

c_afib c_ckd:replace a_antiplatelets with warfarin,replace a_rct_a with BB

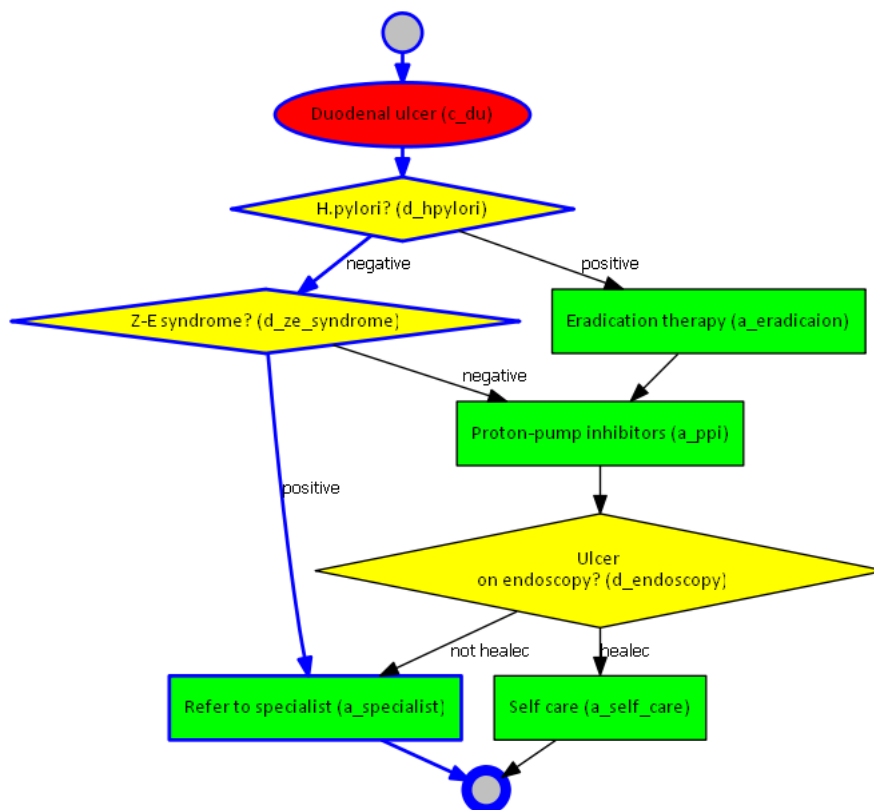
c_afib c_ckd &CHA2DS2-VASc>2:replace a_mt_asa with warfarin2

c_afib c_ckd &CHA2DS2-VASc<=1:replace a_oa_w with aspirin1,replace a_ltoa_w with aspirin2

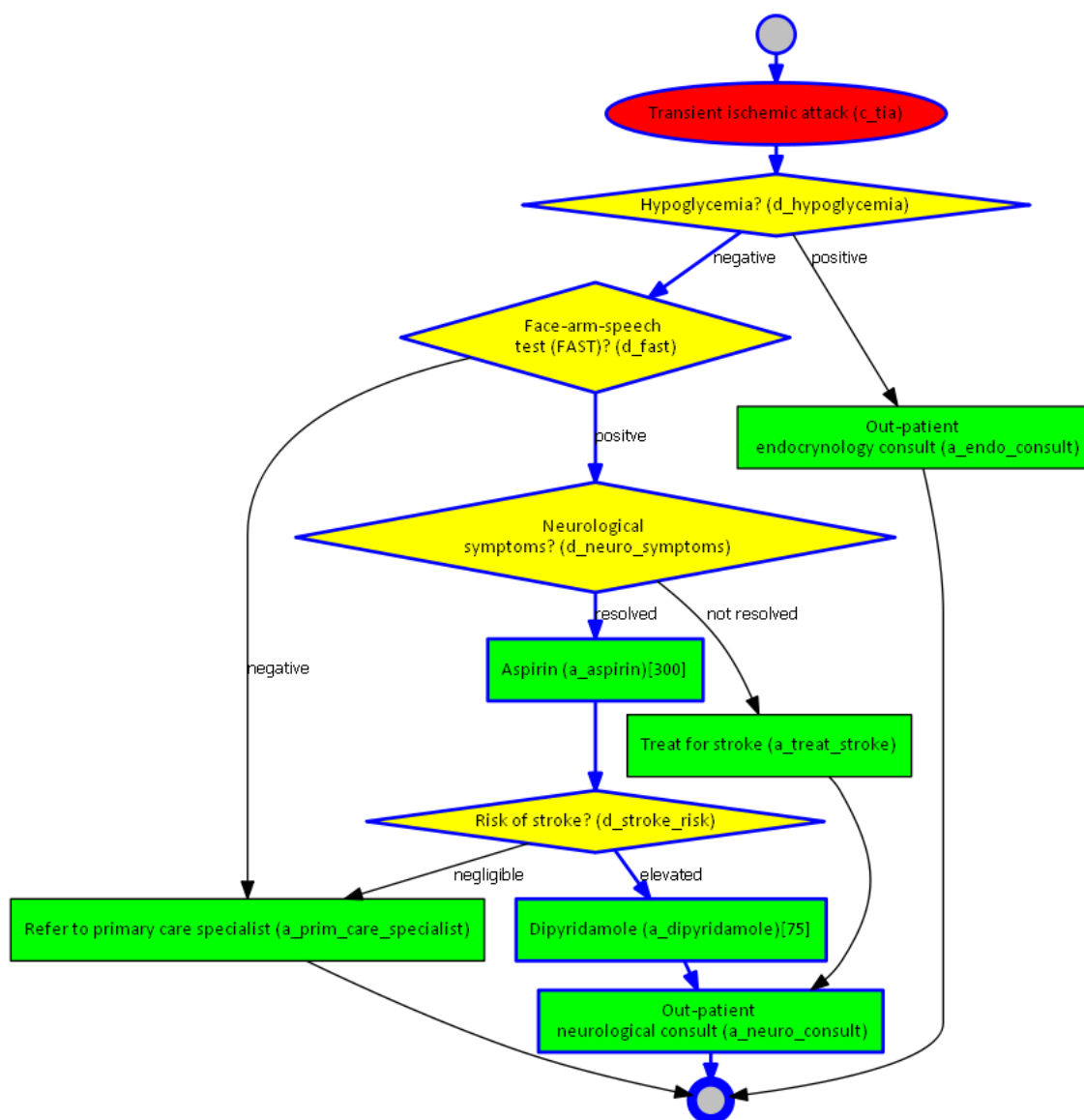
6.3 Przypadek 3 - wrzód dwunastnicy i przemijający atak niedokrwienny

Wytyczne dla wrzodu dwunastnicy (ang. *duodenal ulcer*) przedstawiono na rys. 6.6.

Wytyczne dla przemijającego ataku niedokrwiennego (ang. *transient ischemic attack*) przedstawiono na rys. 6.7.



Rysunek 6.6: Wytyczne dla wrzodu dwunastnicy



Rysunek 6.7: Wytyczne dla przemijającego ataku niedokrwiennego

Odpowiedzi na pytania:

1. Wrzód dwunastnicy:
 - H.pylori?: negative
 - Z-E syndrome?: positive
2. Przemijający atak niedokrwienny:
 - Hypoglycemia?: negative
 - Face-arm-speech test (FAST)?: positive
 - Neurological symptoms?: resolved
 - Risk of stroke?: elevated

Konflikty:

`c_du a_aspirin not(a_ppi) not(a_dipyridamole):replace a_aspirin with cl`

`c_du a_aspirin not(a_ppi) a_dipyridamole:add a_ppi2 after d_ze_syndrome?positive,
decrease_dosage a_aspirin 50`

Rozdział 7

Podsumowanie

7.1 Osiągnięte cele

Podsumowując, wszystkie zamierzenia pracy zostały zrealizowane. Wynikiem pracy jest działający program wyszukujący konflikty występujące między stosowanymi terapiami chorób i proponujący rozwiązania ewentualnych konfliktów.

7.2 Problemy przy realizacji pracy

Do problemów przy realizacji pracy należy zaliczyć kwestię związaną z wyborem biblioteki służącej do przetwarzania grafów. Ostatecznie wybrana została biblioteka JPGD, ponieważ jest to dość prosta biblioteka. W bardzo łatwy sposób uzyskuje się dostęp do obiektu klasy Graph i podrzędnych obiektów klas Node oraz Edge. Niestety, skorzystanie z tej biblioteki wiązało się z naprawą pewnych błędów związanych z ponownym uzyskiwaniem grafu w wersji tekstowej. Konieczna była modyfikacja funkcji toString dla klas Graph, Node oraz Edge, ponieważ generowane początkowo przez bibliotekę grafy w wersji tekstowej nie pozwalały na wygenerowanie grafu w wersji obrazkowej przez program dot.exe. Przyczyna błędu tkwiła w tym, że biblioteka nie radziła sobie z pustymi wartościami atrybutów. Ponadto, trzeba było zrezygnować z korzystania z podgrafów, ponieważ były one niewłaściwie przez bibliotekę interpretowane.

Kolejnym problemem przy realizacji pracy była konieczność zapoznania się z bibliotekami Choco, JPGD oraz oprogramowaniem Graphviz. W przypadku bibliotek Choco i JPGD konieczne było zrozumienie ich dokumentacji. Jeśli chodzi o Graphviz, to główny problem stanowiło zapoznanie się z działaniem programów do tworzenia grafów w formie konsolowej (dot.exe) oraz okienkowej (gvedit.exe).

7.3 Kierunki dalszego rozwoju

Kierunki dalszego rozwoju dzielą się na kierunki związane z podejściem oraz związane z implementacją. Do tych pierwszych można zaliczyć uwzględnianie czasu w wytycznych i konfliktach, które polegałoby na tym, że konflikt występuje wtedy, gdy dwie akcje są wykonywane w tym samym czasie. Ponadto, do kierunków związanych z podejściem mogłoby należeć uwzględnianie kosztów w konfliktach. Wtedy metoda wybierałaby rozwiązanie konfliktu o najmniejszym koszcie.

Jeśli chodzi o implementację, to dalszy rozwój projektu mógłby dążyć do bardziej interaktywnej odpowiedzi na pytania. Program mógłby pozwalać na klikanie na krawędzie grafu zamiast wybierać odpowiedzi za pomocą pól wyboru. Ponadto, program mógłby wspierać także inne formaty grafów, nie tylko format Graphviza o rozszerzeniu dot. Dobrymi pomysłami byłyby także integracja programu z zewnętrznymi systemami w celu pobrania danych pacjenta oraz przygotowanie wersji na aplikacje mobilne.

Bibliografia

- [1] Choco 3 Solver - User Guide. http://choco-solver.org/user_guide/1_overview.html. [Przeglądano 2015-09-29].
- [2] Constraint logic programming - Wikipedia. https://en.wikipedia.org/wiki/Constraint_logic_programming. [Przeglądano 2015-09-29].
- [3] Graphviz - Graph Visualization Software. <http://www.graphviz.org>. [Przeglądano 2015-09-29].
- [4] JPGD - Java-based Parser for Graphviz Documents. <http://www.alexander-merz.com/graphviz>. [Przeglądano 2015-09-29].
- [5] The ECLiPSe Constraint Programming System. <http://eclipseclp.org>. [Przeglądano 2015-09-29].
- [6] A. Latoszek-Berendsen, H. Tange, H. J. van den Herik, A. Hasman. From Clinical Practice Guidelines to Computer-interpretable Guidelines. *Methods Inf Med.*, 49(6):550–570, 2010.
- [7] C. M. Boyd, J. Darer, C. Boult, L. P. Fried, L. Boult, A. W. Wu. Clinical Practice Guidelines and Quality of Care for Older Patients With Multiple Comorbid Diseases. *JAMA*, 294(6):716–724, 2005.
- [8] K. Janczura, T. Gabiga. Wprowadzenie do Programowanie Logicznego z Ograniczeniami z wykorzystaniem ECLiPSe.
- [9] L. Piovesan, P. Terenziani. A Mixed-Initiative approach to the conciliation of Clinical Guidelines for comorbid patients.
- [10] A. Niederliński. *Programowanie w logice z ograniczeniami. Łagodne wprowadzenie dla platformy ECLiPSe*. Wydawnictwo Pracowni Komputerowej Jacka Skalmierskiego, 2014.
- [11] S. Wilk, M. Michalowski, X. Tan and W. Michalowski. Using First-Order Logic to Represent Clinical Practice Guidelines and to Mitigate Adverse Interactions.
- [12] S. Wilk, W. Michalowski, M. Michalowski, K. Farion, M. M. Hing, S. Mohapatra. Mitigation of Adverse Interactions in Pairs of Clinical Practice Guidelines Using Constraint Logic Programming. *Journal of Biomedical Informatics*, 46(2):341–353, 2013.