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

Rozdział 1

Wstęp

1.1 Wprowadzenie

Szybki rozwój medycyny (np. pojawianie się nowych leków, testów i procedur), rosnąca lawinowo ilość gromadzonych danych klinicznych, a także pojawianie się coraz większej liczby złożonych przypadków (co jest spowodowane m.in. starzeniem się społeczeństw) powoduje, że coraz bardziej istotne staje się wspomaganie lekarzy podczas podejmowania decyzji diagnostycznych i terapeutycznych. W tym celu tworzy się systemy wspomagania decyzji klinicznych (SWDK), przez które rozumie się wszelkie systemy komputerowe pomagające personelowi medycznemu podejmować decyzje[11]. Wśród SWD wyróżnia się: systemy do zarządzania informacją i wiedzą, systemy do zwracania uwagi, przypominania i alarmowania oraz systemy do opracowywania zaleceń.

DO SWDK z pierwszej grupy zalicza się wszelkie systemy służące do zbierania, przechowywania i udostępniania danych pacjentów. Przykładem takiego systemu jest Eskulap tworzony 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, m.in. eRejestracja, Apteka, Laboratorium, Elektroniczna Dokumentacja Medyczna. Do SWDK z drugiej grupy należą systemy wbudowywane w aparaturę pomiarową (np. monitorującą funkcje życiowe) lub laboratoryjną. Dzięki czemu na bieżąco można informować personel kliniczny o niewłaściwych (np. wykraczających poza normy) wartościach obserwowanych lub testowanych parametrów. Wreszcie do trzeciej grupy SWDK należą systemy, które dla konkretnego pacjenta przygotowują sugestie diagnostyczne i terapeutyczne, korzystając przy tym z szeroko rozumianych modeli decyzyjnych, które stosowane są do dostępnych danych pacjenta.

Znanym przykładem diagnostycznego SWDK jest system Isabel [4], który wykorzystuje model decyzyjny w formie odpowiednio poindeksowanych publikacji medycznych. Objawy i cechy charakterystyczne pacjenta są wprowadzane do systemu w formie tekstowej (mogą być także pobierane z elektronicznej karty pacjenta). System dopasowuje je do dostępnych publikacji i na podstawie tego dopasowania tworzy listę możliwych diagnoz dla pacjenta. Dodatkowo, istnieje możliwość przeglądania fragmentów publikacji, które związane są z poszczególnymi diagnozami.

W praktyce większą popularność zyskują wytyczne postępowania klinicznego (ang. *clinical practice guidelines*, CPG), które pozwalają opisać postępowanie dla pacjenta chorującego na określoną chorobę. Takie wytyczne są coraz częściej formalizowane oraz osadzane w SWDK w celu planowania i nadzorowania wykonywania terapii. Niestety, większość wytycznych jest opracowywana przy założeniu, że pacjent cierpi tylko na jedną przypadłość, co jest bardzo dużym ograniczeniem

niem praktycznym. Z uwagi na proces starzenia się społeczeństw wzrasta liczba pacjentów, którzy cierpią jednocześnie na wiele schorzeń. W takich sytuacjach bezkrytyczne jednoczesne stosowanie wielu wytycznych może przynieść efekt odwrotny do zamierzonego, tzn. może pogorszyć jakość oferowanej opieki [8]. Dlatego też niezwykle istotne jest szybkie wykrywanie możliwych konfliktów (niekorzystnych interakcji) pomiędzy wytycznymi i wprowadzenie takich zmian w wytycznych, aby uniknąć lub osłabić te konflikty. Wprowadzane zmiany mogą polegać na wprowadzeniu zamienników dla konfliktowych leków, przepisaniu dodatkowych leków, zmianie dawek leków lub też rezygnacji z części leków.

O znaczeniu problemu wykrywania i usuwania konfliktów między wytycznymi świadczy to, iż jest on jednym z „wielkich wyzwań" dla wspomagania decyzji klinicznych [15]. Niniejsza praca jest próbą zmierzenia się z tym wyzwaniem poprzez opracowanie SWDK wspomagającego jednoczesne stosowanie wielu wytycznych dla jednego pacjenta.

1.2 Cel i zakres pracy

Cele główne pracy magisterskiej są następujące:

1. rozszerzenie podejścia do wykrywania i usuwania konfliktów wykorzystującego programowanie logiczne z ograniczeniami i opisanego w [14],
2. implementacja rozszerzonego podejścia w formie samodzielnego SWDK.

Pierwszy z celów głównych wiąże się z następującymi celami szczegółowymi:

1. umożliwieniem stosowania więcej niż dwóch wytycznych,
2. dopuszczeniem stosowania wielu zmian w wytycznych (wielu operacji modyfikujących wytyczne),
3. uwzględnieniem stosowania dawek zarówno przy wykrywaniu konfliktów, jak i wprowadzania zmian w wytycznych.

Drugi z celów głównych jest zdekomponowany na następujące cele szczegółowe:

1. opracowanie reprezentacji dla wytycznych, opisu konfliktów i wprowadzanych zmian,
2. uwzględnienie dodatkowych danych pacjenta, które nie występują w wytycznych, a które należy uwzględnić podczas wykrywania konfliktów,
3. stworzenie SWD pozwalającego na krokowe wykonywanie wytycznych, wyszukiwanie konfliktów oraz wprowadzanie zmian w wytycznych. System ten ma zostać zaimplementowany w języku Java oraz ma korzystać z dodatkowych bibliotek dostępnych na licencji *open source*.

1.3 Struktura pracy

W rozdziale 2 krótko omówiono wytyczne postępowania klinicznego oraz ich rolę 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 działanie systemu na wybranych scenariuszach klinicznych. Rozdział 7 stanowi wreszcie podsumowanie pracy.

Rozdział 2

Przegląd literatury

2.1 Wytyczne postępowania klinicznego

Wytyczne postępowania klinicznego to przygotowany w sposób systematyczny zestaw zaleceń odnośnie postępowania (diagnozowania, planowania terapii, realizacji procesu terapeutycznego) w specyficznych warunkach, np. dla określonego schorzenia czy też urazu [7] (dla uproszczenia, w dalszej części tekstu będzie mowa o specyficznych chorobach).

Pierwsze wytyczne kliniczne zaczęły pojawiać się ponad 30 lat temu. Ich celem było ograniczenie zmienności w postępowaniu, jego ujednolicenie oraz optymalizacja pod względem wykorzystywanych kosztów i innych zasobów. Początkowo wytyczne były przygotowywane dla personelu pomocniczego (np. dla pielęgniarek), dopiero potem do grona potencjalnych „użytkowników" włączono lekarzy (było to związane z obawami środowiska lekarskiego przed tzw. *cookbook medicine*, czyli medycyną według przepisu) [7]. Praktyczna akceptacja wytycznych jest wciąż ograniczona – o przyczynach takiego stanu rzeczy będzie mowa w dalszej części rozdziału.
Do reprezentowania wytycznych wykorzystywane są dwa modele:

1. model tekstowy, w którym wytyczne są prezentowane w postaci dokumentu tekstowego. Aby ułatwić jego przeglądanie lub przeszukiwanie, tekst jest wzbogacany o dodatkowe znaczniki (standard GEM) nadające semantykę wybranym jego fragmentom,
2. model sieci zadań (ang. *task network model*), w którym wytyczne są reprezentowane jako graf skierowany. Wierzchołki w grafie odpowiadają podstawowym krokom w wytycznych – zebraniu danych, podjęciu decyzji, czy też wykonaniu akcji klinicznej. Krawędzie (łuki) wskazują natomiast na zależności kolejnościowe między poszczególnymi krokami. Ten właśnie model stanowi podstawę wielu formalnych reprezentacji wytycznych (np. GLIF3, PROforma, GLARE, SAGE).

W ciągu ostatnich lat rośnie popularność reprezentacji wytycznych wykorzystujących sieci zadań ze względu na możliwość ich osadzenia w systemach komputerowych – mówi się nawet o nowej klasie wytycznych, czyli o wytycznych interpretowanych komputerowo (ang. *computer interpretable guidelines*, CIG). Dzięki temu można budować SWDK, które pozwalają na połączenie wytycznych z danymi dostępnymi w wersji elektronicznej automatyzując tym samym proces opracowywania sugestii diagnostycznych i terapeutycznych dla konkretnego pacjenta. Takie „skomputeryzowane" wytyczne są przedmiotem intensywnych badań, które można podzielić na następujące kategorie:

1. modelowanie i reprezentacja wytycznych,

2. pozyskiwanie i definiowanie wytycznych,
3. integracja z systemami szpitalnymi (zwłaszcza z elektroniczną kartą pacjenta),
4. walidacja i weryfikacja wytycznych,
5. wykonywanie wytycznych,
6. obsługa sytuacji wyjątkowych,
7. utrzymanie i „pielęgnacja" wytycznych,
8. współdzielenie wytycznych.

Niniejsza praca magisterska dotyczy obsługi sytuacji wyjątkowych związanych z konfliktami, które mogą pojawić się podczas jednoczesnego wykonania wielu wytycznych dla jednego pacjenta. Odpowiada ona w ten sposób na wyzwanie związane z jednym z głównych ograniczeń wytycznych, tzn. skupieniem się na jednym specyficznym problemie. Ograniczenie to jest jedną z istotnych przyczyn słabego praktycznego przyjęcia i wykorzystania wytycznych i jest też jednym z głównych wyzwań dla SWDK [15].

2.2 Wykrywanie i usuwanie konfliktów w wytycznych

2.2.1 Programowanie logiczne z ograniczeniami

W pracy [14] przedstawiono podejście wykorzystujące programowanie logiczne z ograniczeniami (ang. *constraint logic programming*, CLP – szczegółowy opis w rozdziale 3) do wykrywania i usuwania konfliktów dla pary wytycznych. Podejście to zakłada, że wytyczne prezentowane są w postaci grafu akcji (ang. *actionable graph*). Graf akcji jest uproszczoną siecią zadań – jest grafem skierowanym, w którym występują trzy typy węzłów:

- *węzeł kontekstu* wskazujący na chorobę, której dotyczą wytyczne,
- *węzeł akcji* opisujący akcję kliniczną, jaką należy wykonać,
- *węzeł decyzyjny* opisujący decyzję i możliwe opcje.

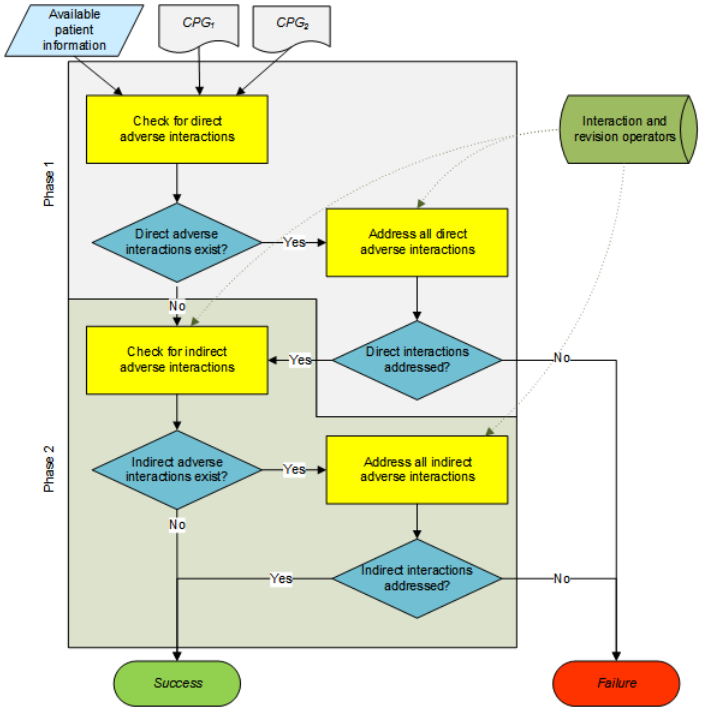
W grafie akcji zrezygnowano z jawnego węzła odpowiadającego pozyskaniu danych i założono, że dane są pozyskiwane w węzłach decyzyjnych (np. poprzez zadanie odpowiedniego pytania lekarzowi). Przykład grafu akcji (rozszerzonego o możliwość zawierania ścieżek równoległych) przedstawiono na rys. 5.1.

Grafy akcji są automatycznie tłumaczone na modele logiczne i automatycznie przetwarzane. Podejście to wykorzystuje dodatkową wiedzę dziedzinową (nie pojawiającą się jawnie w wytycznych) reprezentowaną w formie operatorów interakcji (ang. *interaction operators*) oraz modyfikacji (ang. *revision operators*). Operator interakcji reprezentuje możliwy konflikt (zazwyczaj lek-lek lub lek-choroba), natomiast operator modyfikacji opisuje natomiast zmiany, jakie należy wprowadzić do modeli logicznych, aby konflikt usunąć. Oba operatory przedstawione są w formie wyrażeń

logicznych.

Schemat działania podejścia przedstawiono na rys. 2.1. Obejmuje on dwie fazy:

1. wyszukiwanie bezpośrednich konfliktów między wytycznymi (tzn. konfliktów, gdzie jedne wytyczne zalecają akcję A, a drugie zabraniają tej akcji – konflikty takie objawiają się jako niespójne wartości zmiennych współdzielonych przez modele logiczne) i ich usunięcie za pomocą operatorów modyfikacji,
2. wyszukanie pośrednich konfliktów (opisanych za pomocą operatorów interakcji) i ich usunięcie za pomocą operatorów modyfikacji.



Rysunek 2.1: Schemat działania podejścia wykorzystującego CLP [14]

Podczas obu faz działania na podstawie modeli logicznych tworzone są programy CLP, które są następnie wykonywane/rozwiązywane. Uzyskane rozwiązanie wskazuje na terapię rozumianą jako ścieżki w grafach akcji, jakie należy przejść podczas leczenia pacjenta. Rozwiązanie uzyskane w ostatniej fazie jest ostateczną terapią prezentowaną lekarzowi.

Opisane podejście posiada kilka wad – jest ograniczone tylko do dwóch wytycznych, nie pozwala na uwzględnianie dawek leków oraz pozwala tylko na proste zmiany (jedna operacja) wprowadzane przez operatory modyfikacji. Rozszerzenia zaproponowane w ramach tej pracy magisterskiej usuwają te wady.

2.2.2 Logika pierwszego rzędu

W pracy [13] przedstawiono rozwinięcie opisanego powyżej podejścia, w którym CLP zastąpiono przez logikę pierwszego rzędu (ang. *first-order logic*, FOL). Dzięki temu uzyskano znacznie bogatszą semantykę uwzględniającą zależności kolejnościowe między krokami, dawki leków, czy też złożone operacje modyfikacji wytycznych. Zmodyfikowano również algorytm wykrywania i usuwania konfliktów, aby uwzględniał wiele wytycznych. Podobnie jak poprzednio, wytyczne są podane w formie grafów akcji, a wiedza dziedzinowa w formie operatorów interakcji i rewizji.

Minusem tego podejścia są bardziej złożone modele reprezentujące wytyczne (konieczne jest definiowanie reguł kontrolujących proces wnioskowania) oraz konieczność stosowania bardziej złożonych narzędzi (m.in. systemów do dowodzenia twierdzeń). Z uwagi na te komplikacje, w pracy magisterskiej skupiono się na wcześniejszym podejściu z CLP.

2.2.3 Podejście wykorzystujące paradygmat mieszanej inicjatywy

Paradygmat mieszanej inicjatywy (ang. *mixed initiative paradigm*) polega na tym, że podczas rozwiązywania problemu użytkownik współpracuje z systemem komputerowym, a zatem nie ma tutaj działania w pełni automatycznego. Paradygmat ten został wykorzystany w pracy [10] do wykrywania i usuwania konfliktów między wytycznymi. Podobnie jak we wcześniej opisywanych podejściach, również tutaj wytyczne reprezentowane są w postaci grafu (formalizm GLARE) składającego się z węzłów będących akcjami (dopuszcza sie zarówno akcje atomowe, jak i złożone, czyli plany) i krawędzi modelujących relacje między akcjami. Natomiast wiedza dziedzinowa o możliwych interakcjach reprezentowana jest w formie ontologii (wraz z towarzyszącą bazą wiedzy), która wykorzystuje istniejące terminologie i klasyfikacje medyczne, np. SNOMED CT dla pojęć medycznych i ACT dla leków.

Opisywane podejście stosuje dwie grupy metod do unikania i rozwiązywania zidentyfikowanych konfliktów:

1. unikanie konfliktów
    - wybieranie bezpiecznej alternatywy, tzn. takiej ścieżki w grafie, w której konflikt nie występuje,
    - czasowe unikanie konfliktu, np. poprzez odpowiednie planowanie działań,
2. naprawienie konfliktów
    - modyfikacja dawek leków,
    - monitorowanie efektów (tutaj dopuszcza się mniej poważne konflikty i na bieżąco monitoruje ich następstwa),
    - osłabianie interakcji poprzez rozszerzanie zaleceń o dodatkowe akcje.

Do przetwarzania wytycznych wykorzystywane jest wsteczne przeszukiwanie grafów reprezen-

tujących wytyczne (np. w celu szukania bezpiecznych ścieżek), planowanie bazujące na celach (np. w celu monitorowania efektów konfliktowych akcji) oraz wnioskowanie temporalne (w celu unikania konfliktów lub planowania razem wzmacniających się akcji). Podejście to uwzględnia również pozytywne interakcje i pozwala na takie planowanie akcji, aby dochodziło do pożądanego wzmacniania ich efektów oraz wykrywa dublujące się akcje (np. podane tego samego leku pojawiające się w kilku zaleceniach).

Jak już wspomniano, podejście to nie jest automatyczne – lekarz wskazuje na odpowiedni sposób postępowania, a system stara się go zrealizować. Poza tym podejście to wymaga rozbudowanej i szczegółowej wiedzy dziedzinowej podanej w formie ontologii.

Rozdział 3

Programowanie logiczne z ograniczeniami

Programowanie logiczne z ograniczeniami [2] wykorzystuje programowanie logiczne (ang. *logic programming*, LP) do rozwiązywania problemów spełniania ograniczeń (ang. *constraint satisfaction problem*, CSP).

Program CLP składa się z następujących elementów:

- zbioru zmiennych o wartościach ze skończonych dziedzin,
- zbioru ograniczeń narzuconych na zmienne (np. X + Y > 5).

Rozwiązaniem programu CLP jest takie przyporządkowanie wartości do zmiennych, aby spełnione zostały wszystkie zdefiniowane ograniczenia.

Poniżej opisano zastosowanie CLP do rozwiązania zagadki logicznej SEND + MORE = MONEY [9] pokazanej na rys. 3.1. Rozwiązaniem tej zagadki jest takie przypisanie cyfr z zakresu od 0 do 9 do zmiennych odpowiadających literom zawartym w równaniu, aby było ono spełnione. Każda zmienna (litera) powinna mieć unikalną wartość. Ponadto wartości zmiennych S i M muszą być różne od 0.

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

Rysunek 3.1: Zagadka SEND + MORE = MONEY

Do rozwiązania tej zagadki służy następujący program pokazany na rys. 3.2 i przygotowany w środowisku w ECLiPSe (chodzi tutaj o specjalizowane narzędzie dla CLP, a nie popularne środowisko programistyczne dla języka Java – bardziej szczegółowy opis tego środowiska znajduje się w rozdziale 4.1):

```
:-lib(ic).
sendmore(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).
```

Rysunek 3.2: Program w środowisku ECLiPSe rozwiązujący zagadkę

Można zauważyć, że poszczególne klauzule programu 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.

Po skompilowaniu tego programu wystarczy wywołać funkcję `sendmore(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.

CLP może być wykorzystywane nie tylko do rozwiązywania popularnych zagadek logicznych (np. osiem królowych), ale jest także stosowane w poważnych, rzeczywistych problemach, m.in. m.in. harmonogramowaniu pracy lakierni samochodowej czy projektowaniu inteligentnych systemów okablowania dla dużych budynków [12].

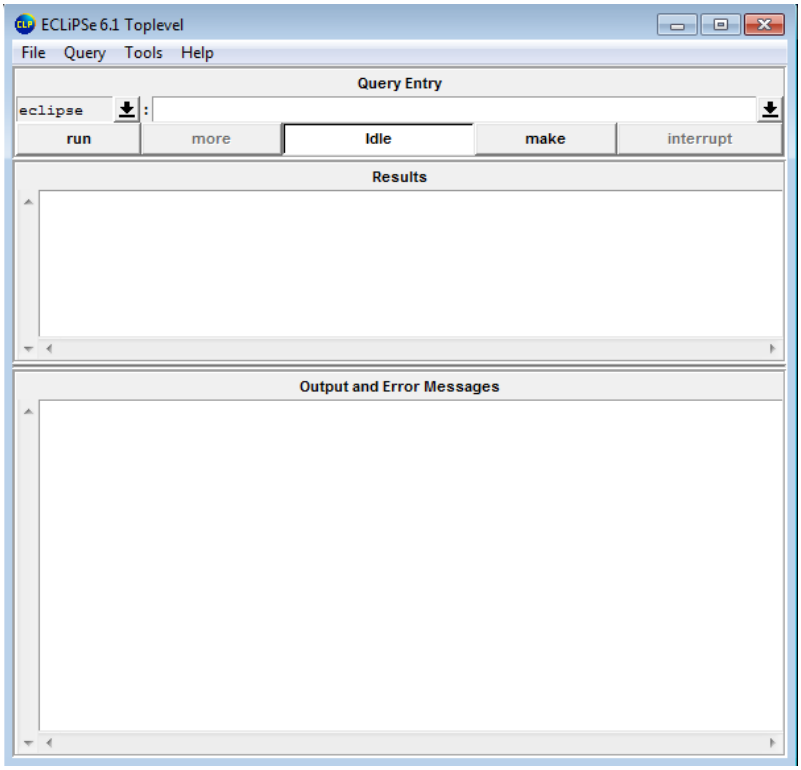
Rozdział 4

Wykorzystane biblioteki i narzędzia

W rozdziale tym opisano krótko biblioteki oraz narzędzia wykorzystane podczas realizacji pracy magisterskiej do weryfikacji wytycznych klinicznych oraz do implementacji systemu wspomagania decyzji klinicznych.

4.1 ECLiPSe

System ECLiPSe [6] jest środowiskiem do tworzenia i wykonywania programów CLP. Jest ono udostępniane na licencji *open source* i może działać na wielu systemach operacyjnych. Główne okno systemu ECLiPSe zostało przedstawione na rys. 4.1. Składa się ono z trzech części. Pierwsza część służy do edycji programów (mogą zostać one odczytane z zewnętrznych plików źródłowych i skompilowane – przykładowy program przedstawiono na rys. 3.1), druga część okna wyświetla wyniki, trzecia natomiast pokazuje ewentualne błędy oraz inne komunikaty.



Rysunek 4.1: Główne okno systemu ECLiPSe

W niniejszej pracy system ECliPSe został użyty do testowania przykładowych wytycznych

medycznych. Nie współpracuje on natomiast ze zrealizowanym systemem wspomagania decyzji klinicznych – do tego celu wykorzystano system Choco 3 opisany w kolejnym punkcie.

4.2 Choco 3

Choco 3 [1] jest darmową biblioteką w języku Java (wersja 8), która pozwala na rozwiązywanie problemów CLP. Podobnie jak system ECLiPSe, jest ona udostępniana jako *open source*.

Główną klasą biblioteki jest klasa `Solver`. Do obiektu z tej klasy 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 metody `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`.

Na rys. 4.2 przedstawiono prosty program dla Choco 3 szukający takich zmiennych X i Y (są to zmienne przyjmujące wartości 0 lub 1), których suma jest równa 1. Problem ten posiada dwa rozwiąnia: 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());
```

Rysunek 4.2: Przykładowy program CLP dla biblioteki Choco 3

4.3 Graphviz

Graphviz [3] jest pakietem narzędzi służących do wizualizacji grafów. Pozwala na konwersję pliku tekstowego w formacie DOT do obrazu przedstawiającego graf. Program automatycznie rozmieszcza wierzchołki grafu – nie jest konieczne podawanie ich współrzędnych. Ponadto program automatycznie rysuje krawędzie tak, aby ograniczyć liczbę ich przecięć.

W skład pakietu wchodzi program `gvedit`, który jest programem okienkowym pozwalającym na weryfikację i wizualizację plików w formacie DOT. Po otwarciu takiego pliku wypisywana jest lista błędów, które należy poprawić (jeśli plik jest niepoprawny), albo wyświetlany jest obraz przedstawiający graf. Podobną funkcjonalność ma program `dot`, z tą różnicą, że jest to program konsolowy (bez graficznego interfejsu użytkownika). Program `dot` przyjmuje 3 parametry. Pierwszym argumentem jest ścieżka do pliku w formacie DOT, drugim jest format generowanego obrazu (przykładowo dla uzyskania formatu PNG obrazka podajemy drugi parametr równy `-Tpng`). Trzeci parametr poprzedzony jest przełącznikiem `-o` i jest to ścieżka do wynikowego obrazu.

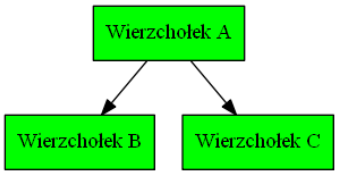
Składnia pliku w formacie DOT jest następująca. Na początku pliku znajduje się słowo `digraph`, po którym umieszcza się nazwę grafu. Wszystkie pozostałe właściwości grafu są umieszczone w bloku otoczonym nawiasami klamrowymi. W bloku tym można podać globalne atrybuty dla wierzchołków oraz krawędzi. Atrybuty dla wierzchołków 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`. Można też określić atrybuty globalne dla całego grafu – do nich należą przede wszystkim wielkość i rodzaj czcionki (krawędzie mogą posiadać etykiety).

Następnie podaje się wierzchołki i krawędzie z ich unikalnymi atrybutami. Atrybut pojedynczego węzła lub krawędzi, jeśli już wystąpił w globalnych atrybutach węzłów lub krawędzi, zostaje nadpisany przez wersję lokalną. Opis pojedynczego węzła polega na podaniu jego unikalnego identyfikatora, a następnie jego atrybutów w bloku otoczonym nawiasami kwadratowymi. Krawędzie natomiast tworzy się podając na początku identyfikator węzła źródłowego krawędzi, następnie wstawia się 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 (`--`).

Na rys. 4.3 zaprezentowano bardzo prosty przykład pliku w formacie DOT, a na rys. 4.4 graf wygenerowany na podstawie tego pliku.

```
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;
}
```

Rysunek 4.3: Prosty plik w formacie DOT



Rysunek 4.4: Graf wygenerowany na podstawie pliku z rys. 4.3

4.4 JPGD

Biblioteka JPGD (a Java parser for Graphviz documents) [5] służy do tworzenia obiektowej reprezentacji pliku w formacie DOT – obiektu klasy `Graph` posiadającego 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 DOT. Listę odczytanych grafów można uzyskać za pomocą metody `Parser.getGraphs`.

Wierzchołki grafu można pozyskać za pomocą metody `Graph.getNodes`, natomiast krawędzie grafu za pomocą `Graph.getEdges`. Wierzchołki i oraz krawędzie posiadają atrybuty. Do atrybutów wierzchołka należy etykieta, kształt, kolor wypełnienia, kolor konturu i grubość linii konturu. Krawędzie posiadają przede wszystkim jeden istotny atrybut -– etykietę. Wartości wszystkich tych atrybutów można odczytać za pomocą metody `getAttribute`, której argumentem jest nazwa atrybutu. Natomiast ustawienie wartości atrybutu odbywa się za pomocą metody `setAttribute`, której pierwszym argumentem jest nazwa atrybutu, a drugim jego wartość.

Każdy wierzchołek grafu zapisanego w formacie DOT posiada także swój unikalny identyfikator. Identyfikatory przechowywane są jako obiekty klasy `Id`. Można je uzyskać wywołując metodę `Node.getId`. Ponowne wywołanie metody `getId`, w tym przypadku dla obiektu klasy `Id` daje dostęp do rzeczywistego identyfikatora węzła typu `String`.

Dla krawędzi istnieje możliwość odczytania wierzchołka źródłowego (początkowego) oraz docelowego (końcowego). Jest to możliwe dzięki metodom `Edge.getSource` (dla uzyskania węzła

źródłowego) oraz `Edge.getTarget` (dla uzyskania węzła docelowego). Obie metody zwracają obiekty klasy `PortNode`, z którego następnie możemy uzyskać obiekt klasy `Node` za pomocą metody `getNode`. Ważną metodą jest też `Graph.toString`. Pozwala ona na uzyskanie zaktualizowanego grafu w formacie DOT, uwzględniającego zmiany wprowadzone za pomocą metody `setAttribute` dla obiektów klasy `Node` lub `Edge`.

Na rys. 4.5 przedstawiono przykładowy kod źródłowy ilustrujący 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;
}
```

Rysunek 4.5: Przykład wykorzystania biblioteki JPGD

Rozdział 5

Implementacja systemu

Rozdział ten opisuje implementację zrealizowanego w ramach pracy systemu. Jego działanie obejmuje następujące kroki:

1. Na początku użytkownik wybiera choroby, na które choruje pacjent.
2. Następnie program wyświetla graficzną reprezentację wytycznych dla wskazanych chorób oraz wyświetla listy pól wyboru, które pozwalają na udzielanie odpowiedzi na pytania zawarte w wytycznych.
3. Po udzieleniu odpowiedzi na wybrane pytania (informacja nie musi być kompletna) program rozwiązuje problem CLP, w wyniku którego uzyskujemy listę konfliktów, które wystąpiły między wytycznymi oraz grafy wynikowe z wprowadzonymi zmianami.
4. Po uzyskaniu rozwiązania problemu można wybrać inne odpowiedzi na pytania i wygenerować nowe rozwiązania problemu. Można także wybrać inne choroby i powiązane z nimi wytyczne.

Poszczególne kroki zostały szczegółowo opisane w punkcie 5.3.

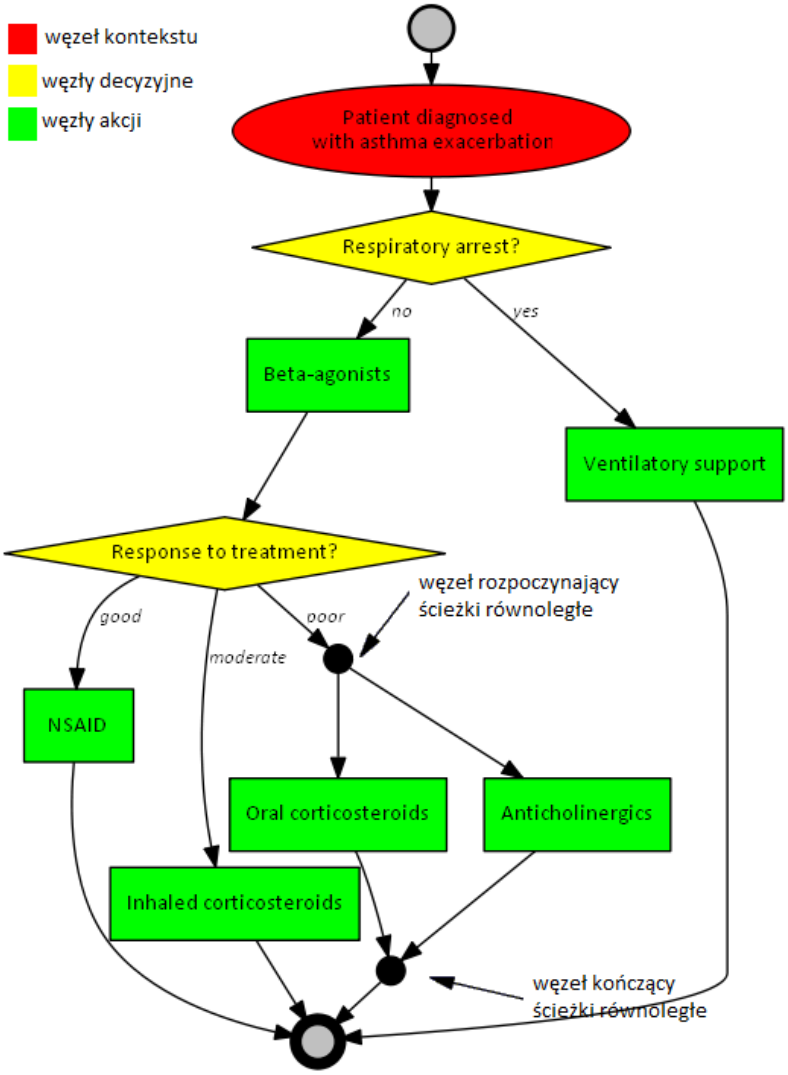
Program przetwarza wytyczne reprezentowane w postaci grafu skierowanego, w którym mogą wystąpić następujące rodzaje węzłów:

- *węzeł początkowy* i *końcowy* oznaczający odpowiednio rozpoczęcie i zakończenie wytycznych,
- *węzeł kontekstowy* opisujący chorobę, dla której sformułowane są wytyczne,
- *węzeł akcji* definiujący akcję (podanie leku, badanie, procedura), jaką należy wykonać,
- *węzeł decyzyjny* wskazujący na dane opisujące pacjenta, które należy pozyskać i sprawdzić w celu wyboru jednego z kilku możliwych sposobów postępowania,
- *węzeł równoległy* rozpoczynający lub kończący ścieżki równoległe.

Wszystkie węzły mają unikalne identyfikatory. Ponadto łuki (w dalszej części tekstu będziemy stosowali określenie krawędzie) wychodzące z węzłów decyzyjnych opisane są za pomocą etykiet odpowiadających poszczególnym decyzjom. Wreszcie węzły akcji i decyzyjne posiadają dodatkowe etykiety z dodatkowym, czytelnym ich opisem. Przykładowy graf przedstawiono na rys. 5.1.

Grafy przechowywane są w plikach w formacie DOT. Format ten nie pozwala na użycie dodatkowej informacji semantycznej o typach węzłów (można określić jedynie ich identyfikatory oraz etykiety). Aby odróżnić węzły równoległe od decyzyjnych, sprawdzane są ich etykiety (a dokładnie ich dostępność lub brak). Poza tym, aby odróżnić węzeł rozpoczynający ścieżki równoległe od węzła kończącego, sprawdzane są liczby krawędzi wchodzących i wychodzących. Węzeł rozpoczynający ścieżki równoległe charakteryzuje się tym, że nie ma etykiety oraz posiada więcej niż jedną krawędź wyjściową. Natomiast węzeł kończący ścieżki równoległe nie posiada etykiety, ma więcej

niż jedną krawędź wejściową oraz liczba jego krawędzi wyjściowych jest większa od zera (liściem jest zawsze węzeł końcowy).



Rysunek 5.1: Przykład ścieżek równoległych

W dalszych opisach wykorzystano pojęcia *terapia* oraz *element terapii*. *Terapia* jest to pojedyncza ścieżka w grafie. *Element terapii* natomiast to dla węzłów decyzji identyfikator węzła i etykieta wybranej krawędzi oddzielone znakiem zapytania, dla pozostałych węzłów *elementem terapii* jest identyfikator węzła.

5.1 Wykorzystywane dane

Dane wykorzystywane oraz generowane przez program znajdują się w następujących katalogach:

- `Algorytmy` – katalog zawiera pliki o rozszerzeniu DOT opisujące grafy reprezentujące dostępne wytyczne kliniczne,
- `Konflikty` – katalog zawiera opisy konfliktów, jakie mogą wystąpić między wytycznymi oraz definicje zmiany, które należy wprowadzić w przypadku wystąpienia tych konfliktów,
- `Grafy` – katalog 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 kończenia pracy programu zawartość tego katalogu jest kasowana

5.2 Główne klasy

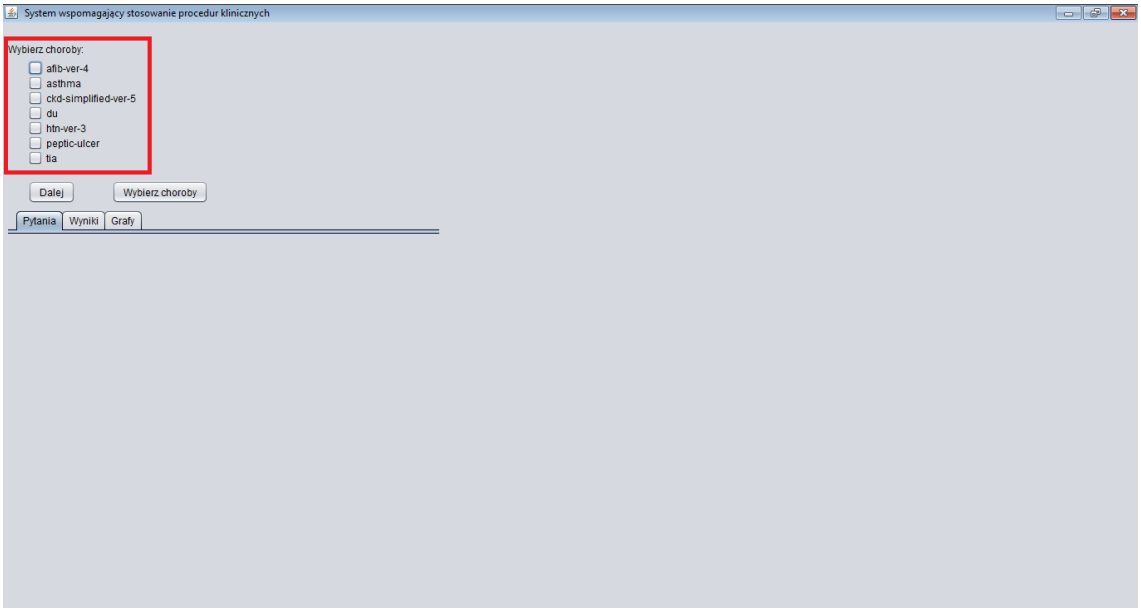
Program zaimplementowano w języku Java. Poniżej przedstawiono listę głównych klas wykorzystywanych w programie (i pojawiających się w dalszych opisach):

- `AddToTherapy` - dodawanie identyfikatorów węzłów do listy opisującej konkretną terapię,
- `ChocoClass` - rozwiązanie problemu CLP za pomocą solvera Choco,
- `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 przejścia między poszczególnymi krokami działania programu,
- `RadioButtonList` - tworzenie i obsługa zdarzeń list pól wyboru służących do udzielania odpowiedzi na pytania,
- `Results` - wyświetlanie wyników,
- `Window` - główne okno programu.

5.3 Główne kroki działania

5.3.1 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 `Window.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 `MainClass.selectedDiseases` i program przechodzi do fazy wyświetlania grafów oraz udzielania odpowiedzi na pytania.

5.3.2 Wyświetlanie grafów oraz udzielanie odpowiedzi na pytania

Ten krok pozwala na utworzenie graficznej reprezentacji wytycznych oraz wybór ścieżki w grafie (terapii) zgodnej z dostępnymi danymi pacjenta. Po wybraniu chorób i kliknięciu przycisku „Dalej" program dla każdej choroby odczytuje za pomocą metody `GraphFunctions.getGraph` grafy z plików w formacie DOT. Następnie program pobiera korzenie każdego z grafów za pomocą metody `GraphFunctions.getStartNode`, a potem wywołuje metodę `GoForward.goForward`, która przemieszcza się po grafie (startując w jego korzeniu) do momentu, gdy napotka pierwszy węzeł decyzji.

Metoda `goForward` dodaje aktualny węzeł do listy elementów terapii. Następnie wykonuje pętlę `while`, której warunek kontynuacji obejmuje trzy przypadki. Pierwszy warunek sprawdza, czy węzeł posiada jedną krawędź wyjściową. Drugi warunek sprawdza, czy węzeł rozpoczyna ścieżki równoległe, a trzeci czy węzeł kończy ścieżki równoległe.

Wewnątrz pętli wykonywane są następujące akcje. Po pierwsze wykonywana jest kolejna, wewnętrzna pętla `while` (jej warunek jest taki sam, jak pierwszy z warunków w pętli zewnętrznej), aby dodać do listy elementów terapii wszystkie węzły, które mają tylko jedną krawędź wyjściową, czyli droga w grafie, po której należy się poruszać jest jednoznacznie określona. Po wykonaniu tej pętli `goForward` zatrzymuje się na węźle, który jest liściem (nie posiada żadnej krawędzi wyjścio-

wej), albo ma więcej niż jedną krawędź wyjściową.

Następnie następuje sprawdzenie, czy aktualny węzeł rozpoczyna ścieżki równoległe (jest to też drugi warunek w pętli zewnętrznej). Jeśli warunek ten jest 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 ścieżki równoległe. Metoda `GoForward.parallelPath` jest w postaci pętli `while`, która działa dopóki program nie przejdzie przez wszystkie ścieżki równoległe związane z aktualnym węzłem i uzyskany węzeł nie jest węzłem decyzyjnym. Wszystkie przebyte po drodze węzły dodawane są do listy elementów terapii.

Po wywołaniu metody `goForward` program wywołuje metodę `Color.color`, która zaznacza przebytą ścieżkę w grafie kolorując oraz pogrubiając kontury przebytych węzłów oraz przebyte krawędzie.

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 zewnętrzny program `dot` 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 w oknie (a dokładnie w przeznaczonym dla niego polu).

Jeśli szerokość lub wysokość obrazu przekracza ustalony próg, obraz zmniejszany jest do 2/3 wielkości tak, aby był on czytelny (w tym przypadku do pola z obrazem dodawane są suwaki). Ponadto, jeżeli szerokość i wysokość obrazu jest mniejsza od wielkości pola, to na etykiecie umieszczany jest obraz bez skalowania (tzn. 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 zwierające znak zapytania odpowiadają krokom decyzyjnym. Pierwszym elementem panelu jest etykieta węzła. Pozostałe elementy stanowią pola wyboru z etykietami, których wartości są równe etykietom krawędzi wychodzących z węzła decyzyjnego. Do tych pól wyboru dodawany jest jeszcze jedno z etykietą „brak wartości”, przydatne w sytuacji, gdy dane nie są znane. 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, do każdego pola wyboru podczepiana jest metoda `RadioButtonList.updateRadioButtonList` obsługująca zdarzenia związane ze zmianą wartości pola.

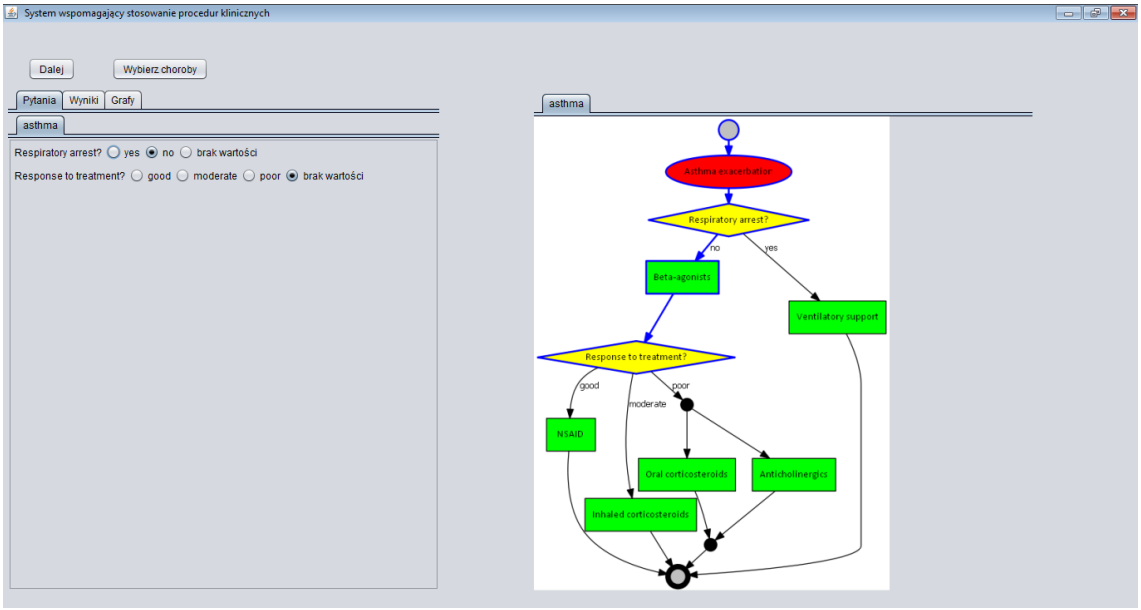
Po wywołaniu metoda `updateRadioButtonList` na początku szuka elementu w liście elementów

terapii, którego dotyczy wybrane pytanie (i związane z nim pole wyboru). Jeśli zaznaczone pole wyboru ma etykietę „brak wartości", usuwane są wszystkie elementy terapii od elementu, którego dotyczy wybrane pytanie, do ostatniego elementu listy pytań. Innymi słowy następuje cofnięcie się w grafie, co oznacza, że pytania i odpowiedzi znajdujące się „poniżej" wybranej lokalizacji są odrzucane.

Jeśli natomiast zaznaczone pole wyboru nie posiada etykiety równej „brak wartości" i nie istnieje element na liście elementów terapii, który jest związany z pytaniem (użytkownik odpowiada na to pytanie po raz pierwszy), to do tej listy dodawany jest element o wartości równej `question?answer`, gdzie `answer` jest etykietą krawędzi, z którą związane jest zaznaczone pole wyboru. Jeśli natomiast istnieje element związany z pytaniem (użytkownik modyfikuje wcześniej udzieloną odpowiedź), 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 (analogicznie jak w przypadku wyboru „braku wartości").

Następnie aktualnym węzłem staje się węzeł, do którego dochodzi krawędź związana z wybraną odpowiedzią (tzn. krawędź o etykiecie `answer` wychodzącej z węzła o identyfikatorze `question`). Węzeł ten jest punktem startowym dla kolejnego wywołania metody `goForward`. Metoda `goForward` nie jest wywoływana, gdy pole wyboru ma etykietę „brak wartości". Potem metoda `updateRadioButtonList` koloruje graf na nowo na podstawie zaktualizowanej 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`.

Zaktualizowane grafy prezentowane są po prawej stronie ekranu na osobnych zakładkach. Każda zakładka dotyczy wytycznych związanych z jedną z wybranych chorób. Z lewej strony ekranu pojawiają się natomiast zakładki z listami pytań i pól wyboru pozwalającymi na uzupełnienie danych pacjenta. W tym przypadku również jedna zakładka dotyczy jednej choroby. Przykładowy ekran z grafami i polami wyboru przedstawiono na rys. 5.3.



Rysunek 5.3: Wyświetlanie grafów

5.3.3 Wyszukiwanie konfliktów

Celem tego kroku jest znalezienie konfliktów pojawiających się między wytycznymi. Wyszukiwanie konfliktów rozpoczyna się od metody ChocoClass.solve. Na początku program szuka w katalogu `Konflikty` plików opisujących konflikty i sposoby ich rozwiązania, które można zastosować do aktualnego zestawu wytycznych. Nazwa każdego pliku opisującego konflikty składa się z listy nazw chorób oddzielonych przecinkami (konflikty pojawiają się między wytycznymi dla tych chorób). Jeśli jakakolwiek choroba z tej listy została wybrana podczas działania programu chorobach, plik zostaje użyty.

Plik z konfliktami zawiera jeden lub więcej wpisów (każdy poświęcony jednemu konfliktowi), a wpis (umieszczony w jednej linii) składa się z dwóch części. Pierwsza część zawiera elementy, których jednoczesne wystąpienie powoduje konflikt. Elementy te są oddzielone spacjami. Druga część 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 te elementy opisujące konflikt, które oznaczają dodatkowe pytania (tzn. odwołują się do danych, które jawnie nie występują w wytycznych) – nazwy tych elementów rozpoczynają się znakiem „&”. Wreszcie zanegowane 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 odpowiedni element konfliktu 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 liczbowa 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 zrealizowanej w metodzie `ChocoClass.solveNextPart`. Metoda ta najpierw wywołuje metodę `ChocoClass.findSolutions`, która dla każdego możliwego konfliktu (odczytanego z pliku) wykonuje szereg operacji. Najpierw sprawdza, czy konflikt znajduje się na liście `foundConflicts`. Lista `foundConflicts` zawiera rzeczywiste konflikty, które zostały dotychczas zidentyfikowane przez program (tutaj należy zaznaczyć, że nie każdy możliwy konflikt musi zachodzić). 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 `ChocoClass.setAdditionalVariables`. Metoda ta sprawdza, czy pytanie jest typu tak/nie lub czy odpowiedzią na pytanie jest wartość liczbowa. W pierwszej sytuacji, jeśli odpowiedź jest równa tak, program tworzy zmienną 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 liczbowa, program tworzy zmienną `IntVar` o wartości równej podanej liczbie.

Po wykonaniu metody `setAdditionalVariables` program wywołuje metodę `setVariables`. Metoda ta dla każdych wytycznych tworzy tablicę terapii. Dla każdej terapii (w ramach poszczególnych wytycznych) tworzona jest zmienna `IntVar` o nazwie *<choroba>_terapia<n>*, gdzie *choroba* jest nazwą choroby, a *n* jest indeksem terapii. Zmienna ta przyjmuje wartości zero, gdy określona terapia nie zostaje użyta lub jeden, gdy zostaje użyta. Zmienna jest zapisywana w tablicy terapii. Następnie metoda 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 (innymi słowy są to te elementy, które występują w pozostałych terapiach dla danych wytycznych).

Metoda `setVariables` tworzy też tablicę `vars`, która będzie zawierała zmienne wchodzące w skład pojedynczej terapii. 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 zapisywana jest zmienna w tablicy `vars`. Jeśli zmienna odnosi się do elementu terapii (akcji) oznaczającego podanie leku, w ramach której określono jego dawkę, tworzona jest zmienna *<X>_dosage*, gdzie *X* jest elementem terapii. Następnie metoda dodaje ograniczenie mówiące, że zmienna *<choroba>_terapia<n>* przyjmuje wartość jeden, tylko wtedy, gdy suma zmiennych należących do tablicy `vars` jest równa wielkości tej tablicy

(innymi słowy, gdy udało się przejść przez całą ścieżkę odpowiadającą terapii i jednocześnie nie wystąpił żaden z zanegowanych elementów konfliktu). W przeciwnym razie *<choroba>_terapia<n>* przyjmuje wartość zero. Po przejściu przez wszystkie terapie dla danych wytycznych metoda 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 odpowiadające konfliktom. Najpierw dodawane są ograniczenia dla tych możliwych konfliktów, których udało się uniknąć (tzn. po uwzględnieniu związanych z nimi ograniczeń udało się uzyskać poprawne rozwiązanie – konflikty takie znajdują się na liście `avoidedConflicts`). Następnie w metodzie `ChocoClass.setConflictConstraint` dodawane jest ograniczenie dla konfliktu sprawdzanego w aktualnej iteracji pętli. Najpierw tworzy ona listę `constraintsList` przechowującą to ograniczenie. Następnie metoda iteracyjnie przetwarza elementy wchodzące w skład aktualnego konfliktu:

- Jeśli element konfliktu jest zanegowany (jego opis zaczyna się od „not"), do listy `constraintsList` dodawane jest ograniczenie *not(<X> = 1)*, gdzie *X* jest elementem terapii wymienionym w elemencie konfliktu.
- Jeśli element konfliktu zawiera jeden z operatorów „<", „<=", „=", „>", lub „>=" wówczas wywoływana jest metoda `ChocoClass.conflictWithDosage`, która sprawdza, jaki charakter ma element terapii z elementu konfliktu i dodaje odpowiednie ograniczenia do `constraintsList`. Jeśli element terapii jest związany z dodatkowym pytaniem (jego nazwa rozpoczyna się od „&"), wówczas dodawane jest ograniczenie *<X> <operator> <wartość>*, gdzie *X* jest elementem terapii, a *wartość* jest liczbą występującą w elemencie konfliktu. W przeciwnym razie (element terapii jest związany z akcją, w tym z podaniem leku), do `constraintsList` dodawane są dwa ograniczenia. Pierwsze jest w postaci *<X> = 1* (odpowiada ono wykonaniu wskazanej akcji), natomiast drugie ma formę *<X>_dosage <operator> <wartość>* i dotyczy dawki związanej z daną akcją.
- W pozostałych przypadkach do `constraintsList` dodawane jest ograniczenie *<X> = 1*, gdzie *X* to odpowiedni element terapii.

Po zakończeniu przetwarzania poszczególnych elementów konfliktu do obiektu klasy `Solver` dodawane jest ograniczenie w formie *not(and(`constraintsList`))*, aby zabezpieczyć się przed wystąpieniem aktualnego konfliktu.

Następnie wywoływana jest metoda `findSolution`, która szuka rozwiązania problemu. Jeśli rozwiązanie istnieje, aktualny konflikt dodawany jest to listy do `avoidedConflicts`. Jeśli natomiast rozwiązanie nie istnieje, aktualny konflikt jest dodawany do `foundConflicts`, a do listy `executedInteractions` dopisywane są zmiany powiązane z danym konfliktem. Ponadto, gdy nie ma rozwiązania program wywołuje metodę `ExecuteInteractions.executeInteractions`, która dokonuje zmian w wytycznych (a dokładnie w terapiach), a także program wywołuje rekurencyjnie

metodę `findSolutions`, aby sprawdzić, czy wprowadzone zmiany nie spowodowały wystąpienia konfliktów, które zostały wcześniej sprawdzone, oraz aby sprawdzić kolejne konflikty.

Ostatecznie, program znajduje rozwiązania problemu z ograniczeniami dla tych konfliktów, które znajdują się na liście `avoidedConflicts` (jak już wspomniano, są to konflikty, których udało się uniknąć – dla konfliktów, które wystąpiły, wprowadzono odpowiednie zmiany do terapii). Po wygenerowaniu pierwszego rozwiązania program tworzy listę o nazwie `solutions`. Następnie w pętli, która działa dopóki istnieje kolejne rozwiązanie, program zapisuje do zmiennej `solution` nazwy zmiennych, które w rozwiązaniu posiadają wartość równą jeden. Następnie, jeśli zmienna `solution` nie znajduje się jeszcze w liście `solutions`, dodawana jest do tej listy.

Na końcu program do listy `therapies` dodaje rozwiązania. Polega to na tym, że dla każdego rozwiązania z listy `solutions` znajdujemy odpowiadające mu terapie w liście `therapiesDiseases`. Lista `therapiesDiseases` zawiera terapie wszystkich wybranych chorób zgodne z udzielonymi odpowiedziami na pytania znajdujące się w wytycznych. Znalezienie odpowiedniej terapii polega na odczycie nazwy choroby i identyfikatora terapii ze zmiennych <choroba>_terapia<n> znajdujących się w liście `solutions`.

5.3.4 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 `Results.setResults`. Na początku metoda wywołuje inną metodę o nazwie `Results.setGraphs`. Zajmuje się ona modyfikowaniem grafów, wprowadzając niezbędne zmiany usuwające znalezione konflikty. Dla każdej modyfikacji sprawdzany jest jej typ, który może być jednym z następujących:

- *replace <X> with <Y>*, gdzie węzeł X zamienia się na węzeł Y,
- *add <X> before/after <Y>*, gdzie węzeł X jest dodawany przed lub po elemencie Y,
- *remove <X>*, węzeł X jest usuwany,
- *increase_dosage/decrease_dosage <X> <DV>*, gdzie dawka leku z węzła X jest zwiększana lub zmniejszana o wartość DV,
- *change_dosage <X> <V>*, gdzie dawka leku z węzła X jest ustalana na V.

Zmiana grafu uzależniona jest od typu modyfikacji i przeprowadzana jest w następujący sposób:

- Dla modyfikacji *replace*, najpierw wyszukiwany jest węzeł X. Po znalezieniu takiego węzła z pliku `Konflikty/nazwy.txt` odczytywana jest etykieta węzła Y. Wreszcie identyfikator i etykieta znalezionego węzła są zmieniane na nowe wartości.
- Dla modyfikacji *add*, poszukiwany jest węzeł Y, przed lub za którym ma zostać umieszczony nowy węzeł X. Następnie program tworzy węzeł X, nadaje mu etykietę pobraną z pliku `nazwy.txt` i dodaje węzeł do grafu. Jeśli węzeł X jest wstawiany po elemencie postaci

pytanie?odpowiedź, wówczas staje się on węzłem docelowym dla krawędzi z etykietą *odpowiedź*, oraz wstawiana jest dodatkowa krawędź od węzła X do poprzedniego węzła docelowego. W przeciwnym razie wstawiana jest krawędź z Y do X (dla *add after*) lub z X do Y (dla *add before*).

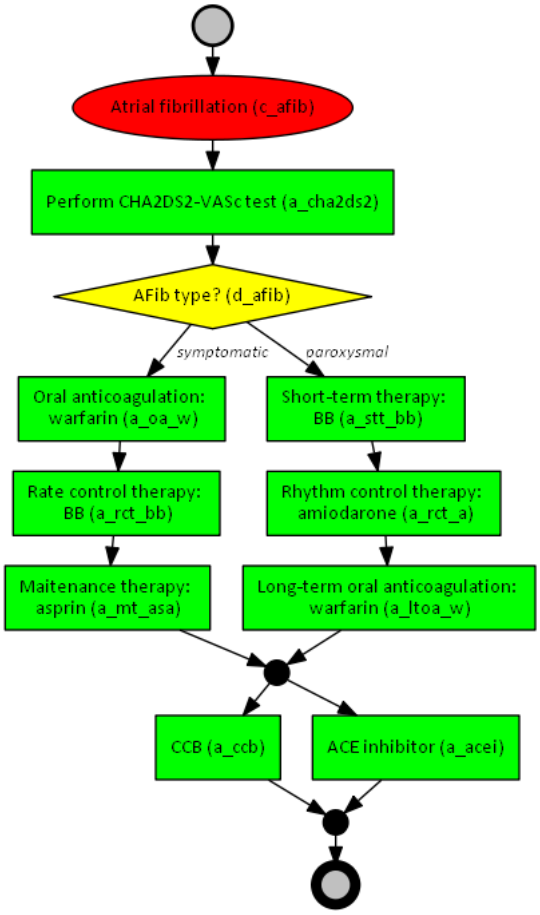
- Dla modyfikacji *remove* atrybuty usuwanego węzła X modyfikowane są w taki sposób, że węzeł staje się niewidoczny na wynikowym grafie
- Dla modyfikacji *increase_dosage*, *decrease_dosage* i *change_dosage* odpowiednio zmienia się końcową część etykiety zmienianego węzła X, gdzie w nawiasach kwadratowych umieszczona jest zmieniona dawka leku związanego z węzłem.

Na zakończenie metoda `setGraphs` dla każdego grafu wywołuje metodę `color` zaznaczającą przebyte węzły i krawędzie, a następnie metodę `newImageGraph`, która powoduje wygenerowanie grafu w postaci obrazkowej.

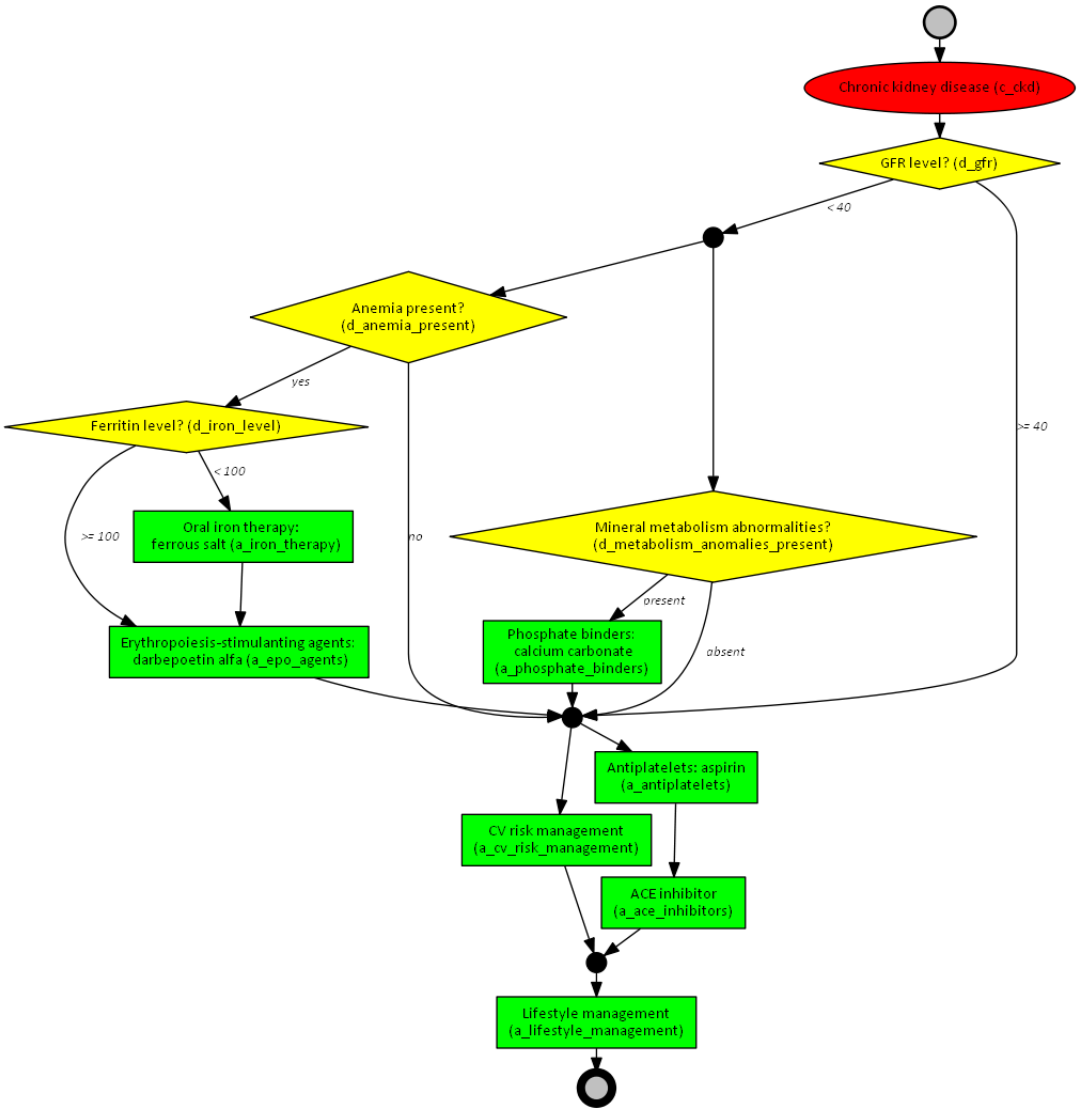
Po wywołaniu metody `setGraphs` program tworzy także tekstową reprezentację uzyskanych rozwiązań. Dla każdej z otrzymanych terapii obejmuje ona identyfikatory oraz etykiety elementów terapii (czyli odwiedzonych węzłów w grafach). Reprezentacja ta zawiera również opis napotkanych konfliktów oraz listę wprowadzonych zmian. Przy ustalaniu etykiet węzłów grafach wykorzystywane są informacje z pliku `nazwy.txt`.

5.4 Przykład działania programu

W tym punkcie przedstawiono działanie programu na wybranym przykładzie klinicznym obejmującym wytyczne dla dwóch chorób: migotania przedsionków (ang. *atrial fibrillation*, rys. 5.4) oraz przewlekłej choroby nerek (ang. *chronic kidney disease*, rys. 5.5). W przypadku węzłów odpowiadających akcjom i decyzjom podano ich etykiety oraz identyfikatory (w nawiasach okrągłych).



Rysunek 5.4: Wytyczne dla migotania przedsionków



Rysunek 5.5: Wytyczne dla przewlekłej choroby nerek

W przypadku wytycznych dla migotania przedsionków (rys. 5.4) program zatrzymuje się na pierwszym węźle decyzyjnym *AFib type?*, zapisując wcześniej do listy elementów terapii węzeł startowy, węzeł kontekstowy określający chorobę oraz węzeł *Perform CHA2DS2-VASc test*. Po wskazaniu przez użytkownika odpowiedzi *paroxysmal* program dodaje do listy *d_afib?paroxysmal*, a następnie trzy węzły: *Short-term therapy: BB*, *Rhythm control therapy: amiodarone* i *Long-term oral anticoagulation: warfarin*. Następnie program dodaje węzeł rozpoczynający ścieżki równoległe, następnie dwa węzły znajdujące się na ścieżkach równoległych (najpierw węzeł *CCB*, następnie węzeł *ACE inhibitor*), a później węzeł kończący ścieżki równoległe. Ostatecznie program dodaje węzeł końcowy grafu.

W wytycznych dla choroby nerek (rys. 5.5) program zatrzymuje się na węźle decyzyjnym *GFR level?*, dodając po drodze do listy elementów terapii węzeł startowy oraz węzeł choroby. Po

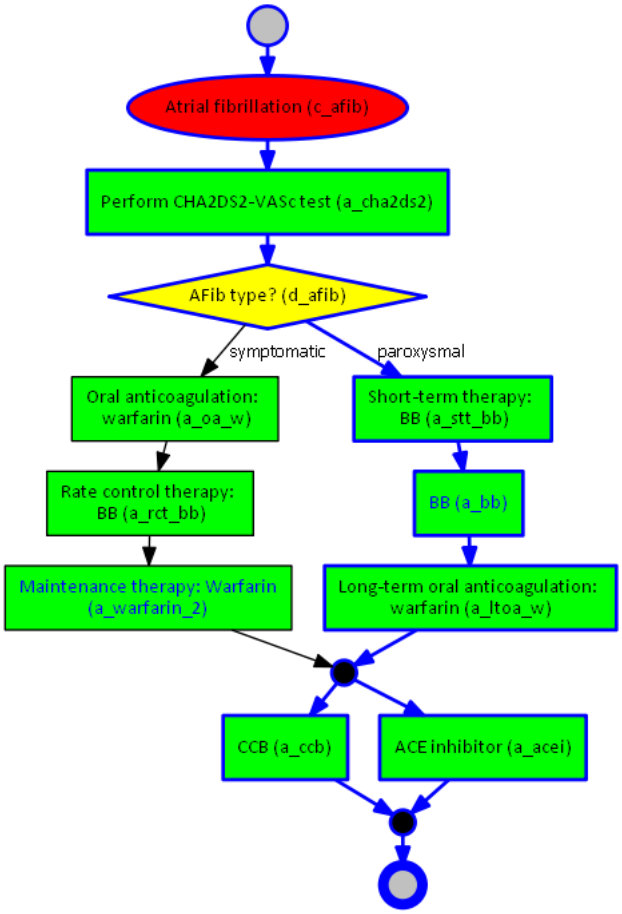
udzieleniu odpowiedzi *<40* program dodaje do listy *d_gfr?<40*, a następnie trafia na węzeł rozpoczynający ścieżki równoległe, który również jest dodawany do listy. W kolejnym kroku program zatrzymuje się na decyzji znajdującej się na lewej ścieżce równoległej: *Anemia present?*. Po uzyskaniu odpowiedzi *no* program dodaje do listy element *d_anemia_present?no*, a następnie wraca do prawej ścieżki równoległej i zatrzymuje się na decyzji *Mineral metabolism abnormalities?*. Po uzyskaniu odpowiedzi *absent* program dodaje do listy *d_metabolism_anomalies_present?absent*. Następnie program umieszcza na liście węzeł kończący ścieżki równoległe, który jednocześnie rozpoczyna kolejne ścieżki równoległe. W kolejnym kroku program dodaje element znajdujący się na lewej ścieżce równoległej, czyli *CV risk management*, a następnie elementy znajdujące się na prawej ścieżce równoległej, czyli *Antiplatelets: aspirin* oraz *ACE inhibitor*. Ostatecznie dodawany jest węzeł kończący ścieżki równoległe, węzeł *Lifestyle management* oraz węzeł końcowy grafu.

Po określeniu terapii program przechodzi do fazy wyszukiwania konfliktów. Najpierw program odczytuje plik z konfliktami dotyczącymi migotania przedsionków, przewlekłej choroby nerek oraz nadciśnienia. Plik ten ma następującą zawartość:

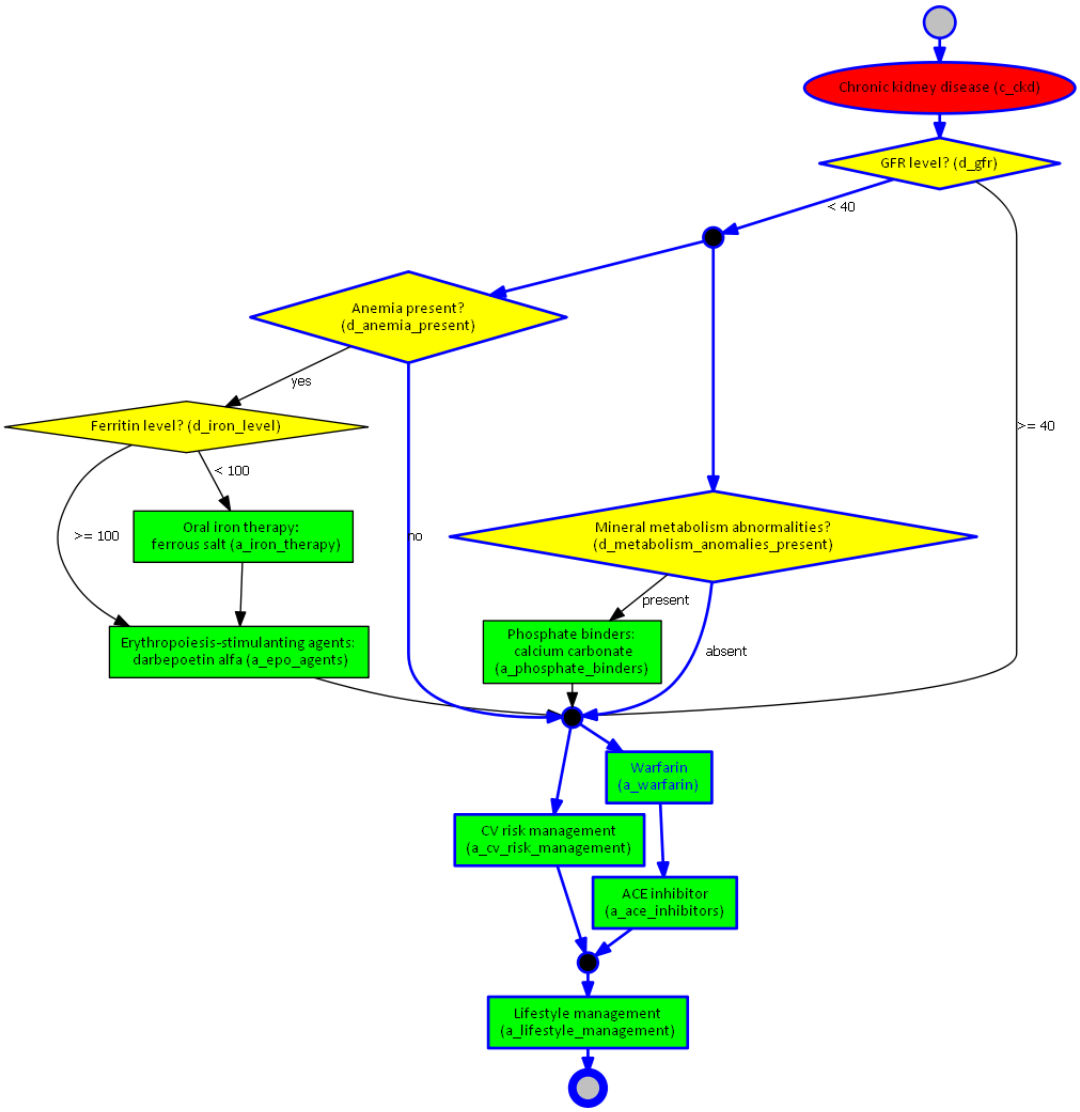
```
c_htn c_ckd:remove a_step1_acei,remove a_step1_ccb
c_afib c_ckd c_htn:remove a_step3_diuretric
c_afib c_ckd:replace a_antiplatelets with a_warfarin,replace a_rct_a with a_bb
c_afib c_ckd &CHA2DS2-VASc>2:replace a_mt_asa with a_warfarin_2
c_afib c_ckd &CHA2DS2-VASc<=1:replace a_oa_w with a_aspirin_1,
replace a_ltoa_w with a_aspirin_2
```

Następnie program prosi o uzupełnienie danych pacjenta poprzez podanie wartości zmiennej *CHA2-DS2-VASc*. Użytkownik uzupełnia dane – niech wartość tej zmiennej będzie równa 5, a program zaczyna wyszukiwać konflikty. Ostatecznie program znajduje dwa konflikty: (*c_afib c_ckd* – współwystąpienie obu chorób) oraz (*c_afib c_ckd &CHA2DS2-VASc>2* – podwyższona wartość CHA2DS2-VASc w połączeniu z migotaniem przedsionków). Pierwsze dwa konflikty nie wystąpiły, ponieważ pacjent nie choruje na nadciśnienie (*c_htn*), więc odpowiednie wytyczne nie są rozważane. Ostatni konflikt nie wystąpił natomiast dlatego, że zmienna CHA2DS2-VASc przyjmuje wartość większą od 1.

W kolejnym kroku program tworzy zmodyfikowane grafy wynikowe, które zawierają zmiany wprowadzone w celu uniknięcia konfliktów. Graf dla migotania przedsionków został przedstawiony na rys. 5.6, natomiast graf dla przewlekłej choroby nerek jest na rys. 5.7. W grafie dla migotania przedsionków węzeł *Maintenance therapy: aspirin* został zmieniony na *Maintenance therapy: Warfarin* oraz węzeł *Rhytm control therapy: amiodarone* na węzeł *BB*. Natomiast w grafie dla przewlekłej choroby nerek węzeł *Antiplatelets: aspirin* został zmieniony na *Warfarin*.



Rysunek 5.6: Wytyczne dla migotania przedsionków



Rysunek 5.7: Wytyczne dla przewlekłej choroby nerek

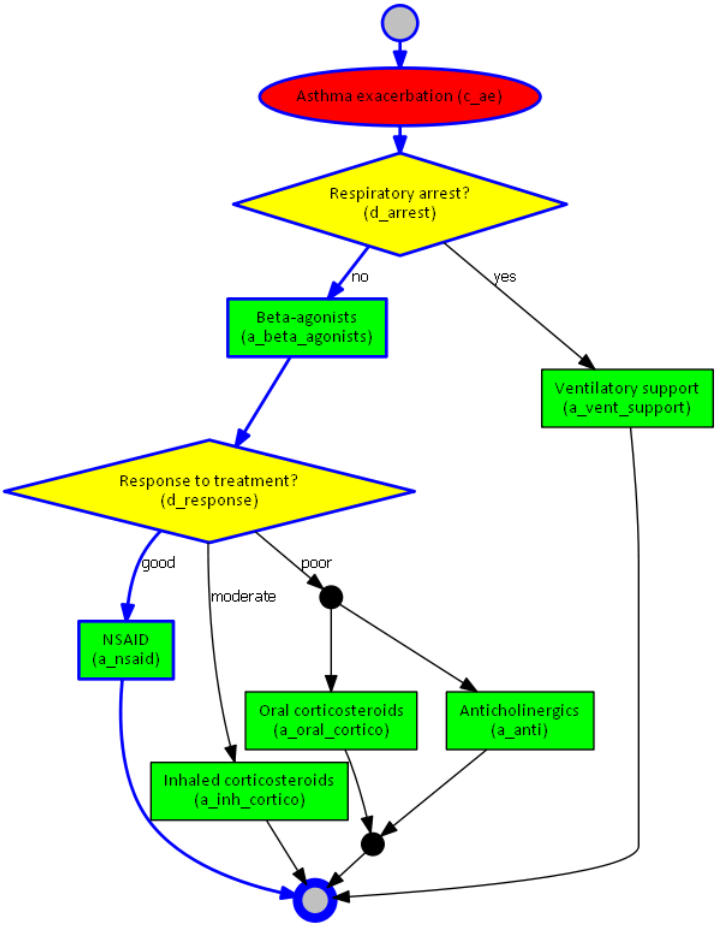
Rozdział 6

Scenariusze kliniczne

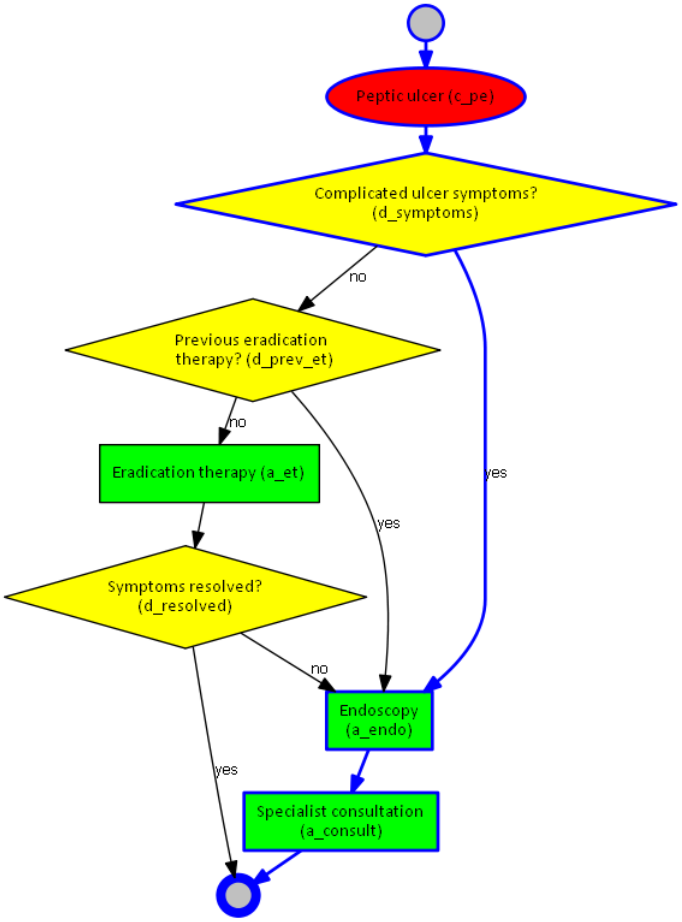
W tym rozdziale zaprezentowano działanie systemu z wykorzystaniem wybranych przykładowych scenariuszy klinicznych. Wytyczne wykorzystane w tych scenariuszach zostały wcześniej skonsultowane z lekarzami, przy czym część została uproszczona na potrzeby wcześniejszych publikacji, np. [13, 14]. Dla każdego scenariusza przedstawione zostały grafy reprezentujące zastosowane wytyczne. Na grafach tych zaznaczono ścieżki, które zostały wybrane podczas etapu zbierania danych pacjenta i udzielania odpowiedzi na pytania związane z węzłami decyzyjnymi. Odpowiedzi na pytania są także przedstawione w formie tekstowej. Następnie, dla każdego scenariusza przedstawiono listę możliwych konfliktów oraz zmiany, jakie należy wprowadzić w przypadku ich wystąpienia (do opisu zmian wykorzystano składnię wprowadzoną w rozdziale 5.3.4). W celu zachowania większej zwięzłości prezentacji w opisach konfliktów i wymaganych zmian zastosowano identyfikatory węzłów – są one przedstawione na grafach (w nawiasach po etykietach poszczególnych węzłów). Ponadto pogrubioną czcionką zaznaczono znalezione konflikty (nie wszystkie możliwe konflikty musiały wystąpić). Na końcu każdego scenariusza podane zostały grafy wynikowe. Dodane węzły w grafach wynikowych zostały oznaczone niebieską czcionką.

6.1 Scenariusz 1 - atak astmy i wrzód trawienny

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

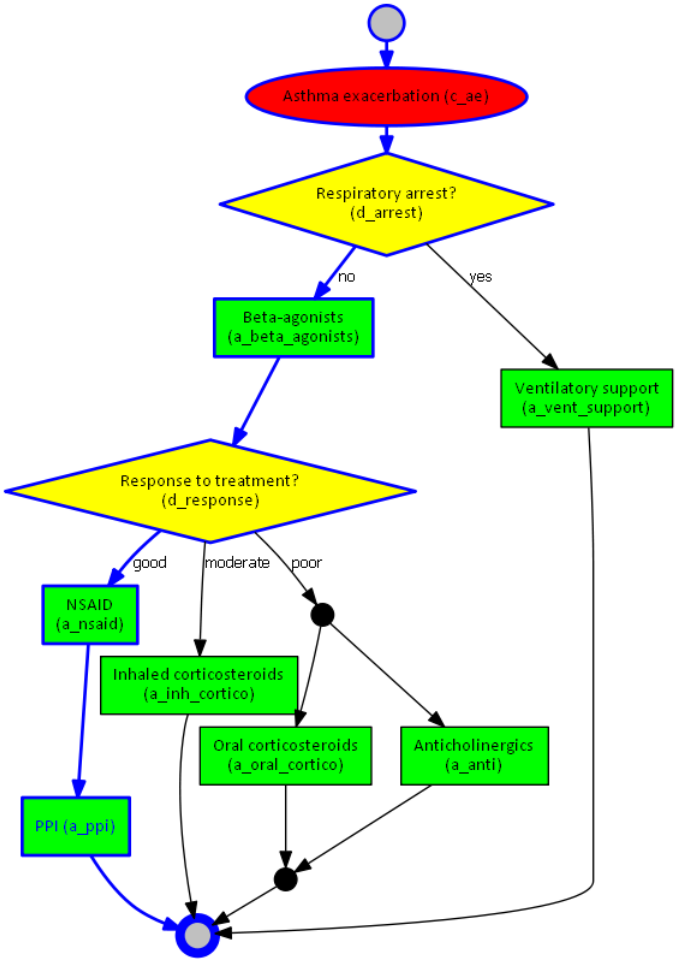
Dane opisujące stan pacjenta (odpowiedzi na pytania z wytycznych) są następujące:

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

Dla rozważanych wytycznych możliwe są następujące konflikty:

1. c_pe a_oral_cortico: replace a_oral_cortico with a_inh_cortico2
2. **c_pe a_nsaid: add a_ppi after a_nsaid**
3. a_et a_inh_cortico: replace a_inh_cortico with a_oral_cortico2

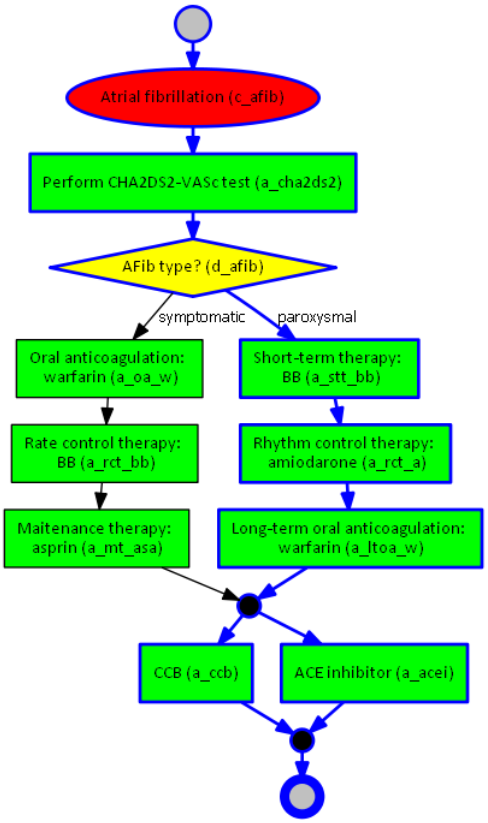
Graf wynikowy dla ataku astmy przedstawiono na rys. 6.3, natomiast graf wynikowy dla wrzodu trawiennego jest identyczny jak ten z rys. 6.2.



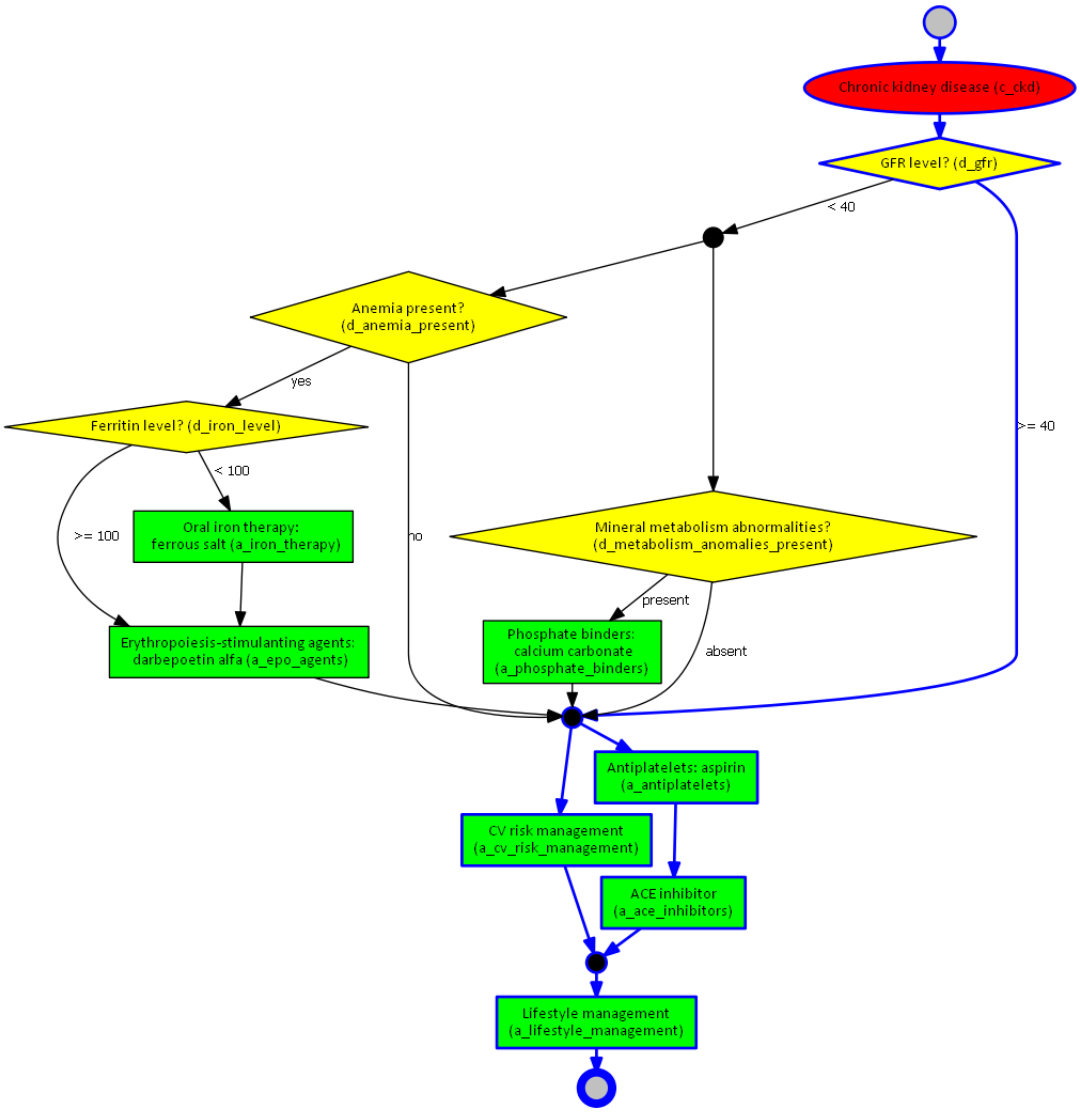
Rysunek 6.3: Graf wynikowy dla ataku astmy

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

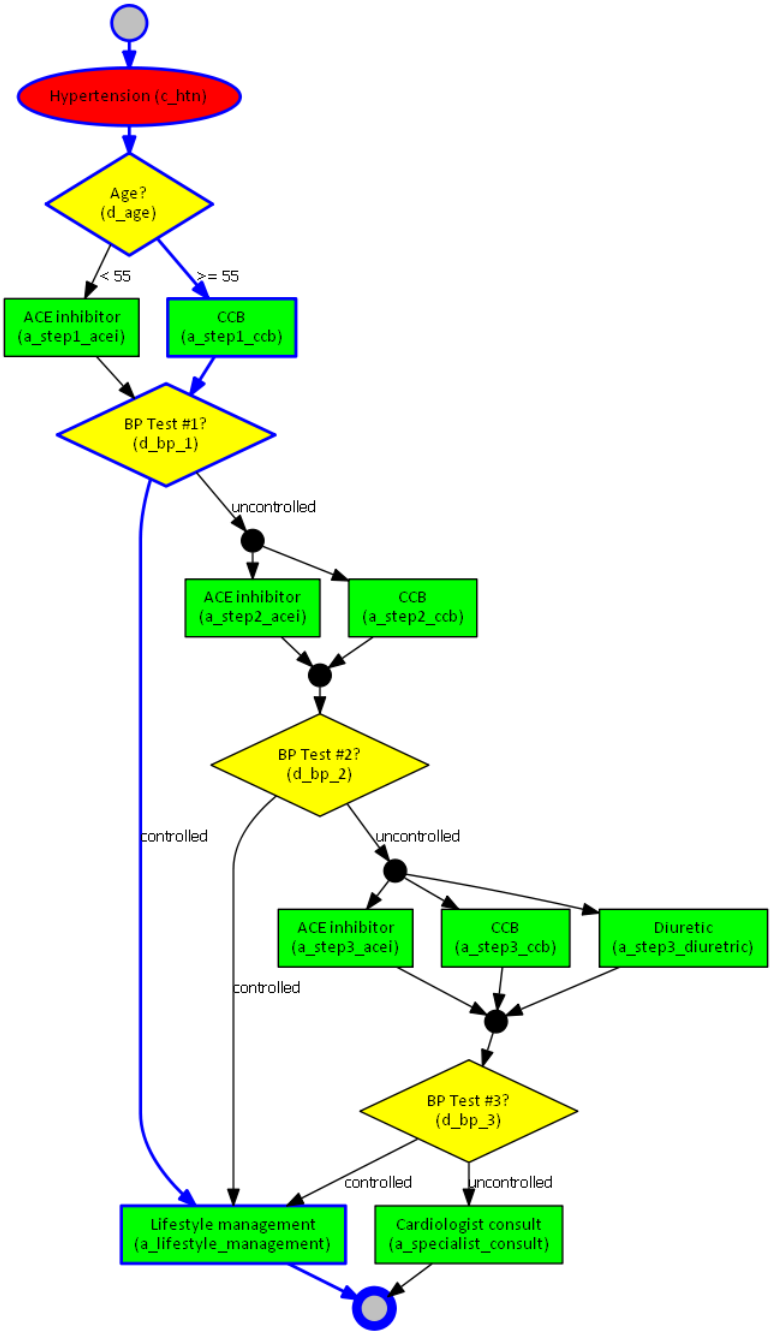
Wytyczne dla migotania przedsionków (ang. *atrial fibrillation*) przedstawiono na rys. 6.4, dla dla przewlekłej choroby nerek (ang. *chronic kidney disease*) na rys. 6.5, a dla dla nadciśnienia (ang. *hypertension*) na rys. 6.6.



Rysunek 6.4: Wytyczne dla migotania przedsionków



Rysunek 6.5: Wytyczne dla przewlekłej choroby nerek



Rysunek 6.6: Wytyczne dla nadciśnienia

Dane opisujące stan pacjenta (odpowiedzi na pytania z wytycznych) są następujące:

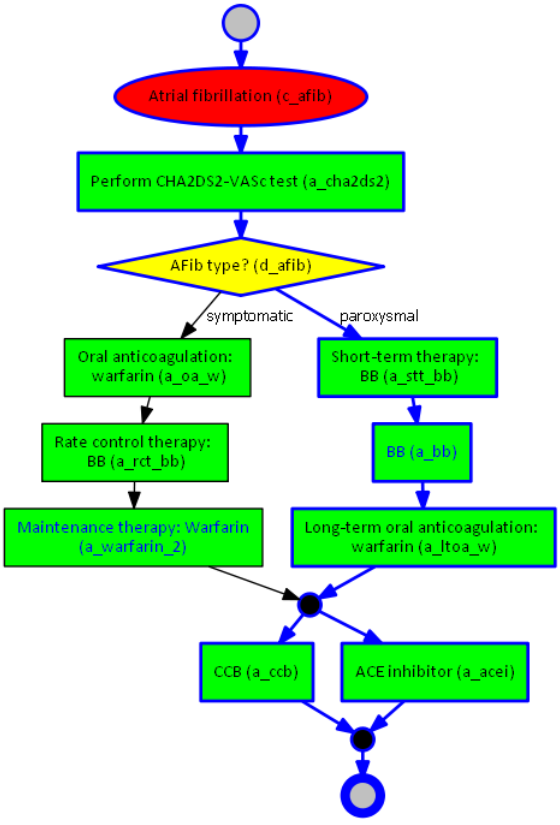
1. Migotanie przedsionków:
   - AFib type?: paroxysmal (d_afib?paroxysmal)
2. Przewlekła choroba nerek:
   - GFR level?: >=40 (d_gfr?>=40)

3. Nadciśnienie:
   - Age?: >=55 (d_age?>=55)
   - BP Test #1?: controlled (d_bp_1?controlled)
4. Dodatkowe dane, które nie pojawiły się jawnie w wytycznych i które zostały uzupełnione podczas fazy poszukiwania konfliktów:
   - CHA2DS2-VASc = 5

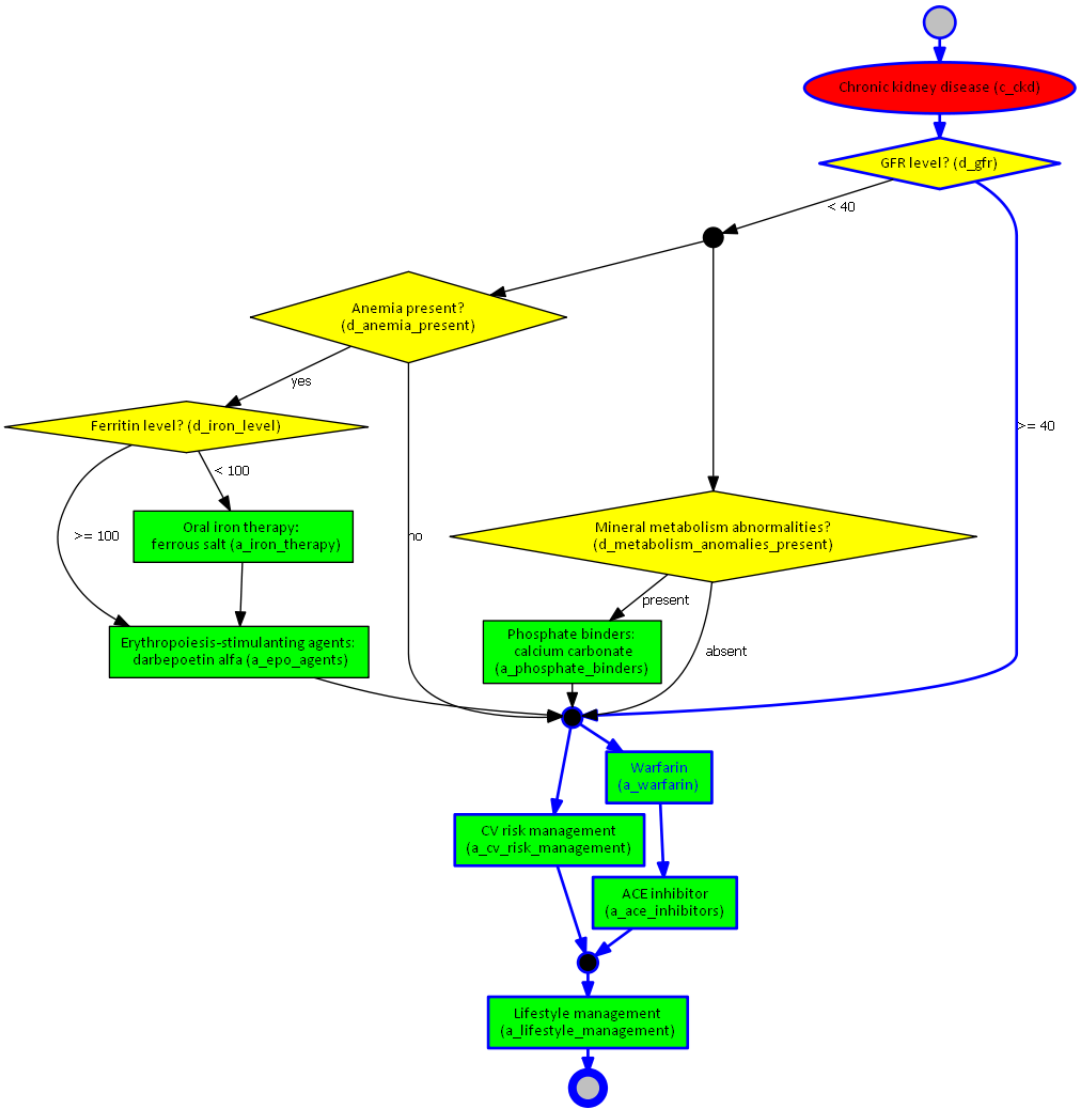
Dla rozważanych wytycznych możliwe są następujące konflikty:

1. **c_htn c_ckd: remove a_step1_acei, remove a_step1_ccb**
2. **c_afib c_ckd c_htn: remove a_step3_diuretric**
3. **c_afib c_ckd: replace a_antiplatelets with a_warfarin, replace a_rct_a with a_bb**
4. **c_afib c_ckd &CHA2DS2-VASc>2: replace a_mt_asa with a_warfarin_2**
5. c_afib c_ckd &CHA2DS2-VASc<=1: replace a_oa_w with a_aspirin_1, replace a_ltoa_w with a_aspirin_2

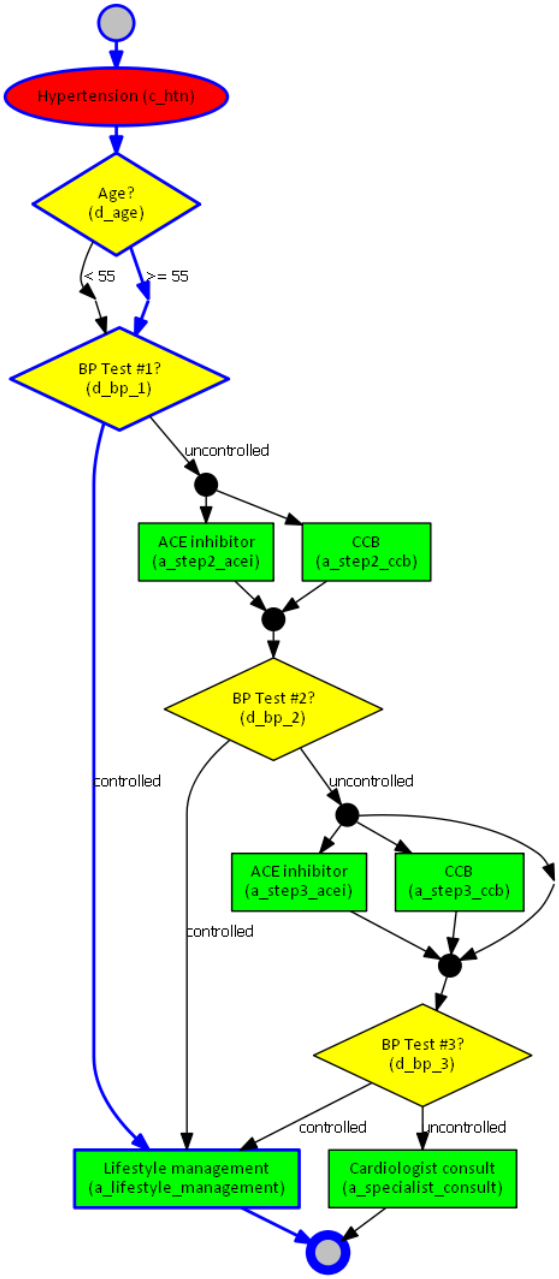
Graf wynikowy dla migotania przedsionków przedstawiono na rys. 6.7, dla przewlekłej choroby nerek na rys. 6.8, natomiast dla nadciśnienia na rys. 6.9.



Rysunek 6.7: Graf wynikowy dla migotania przedsionków



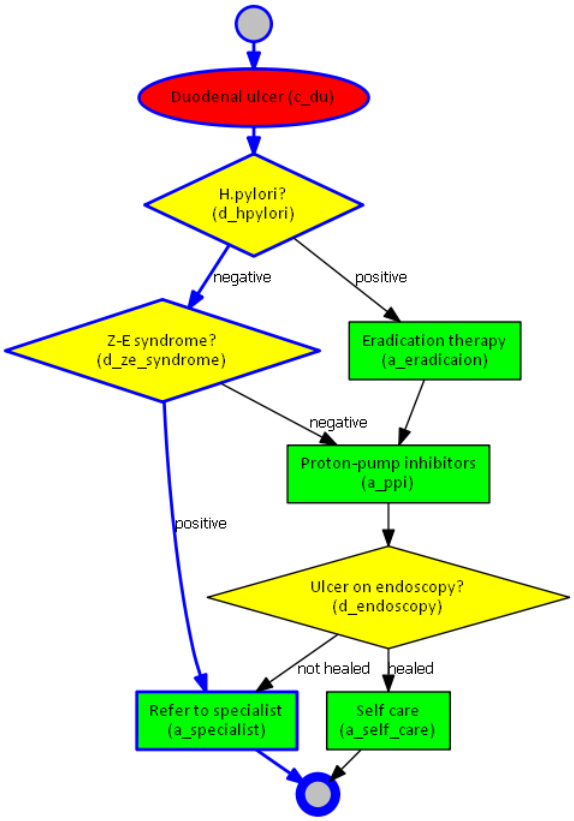
Rysunek 6.8: Graf wynikowy dla przewlekłej choroby nerek



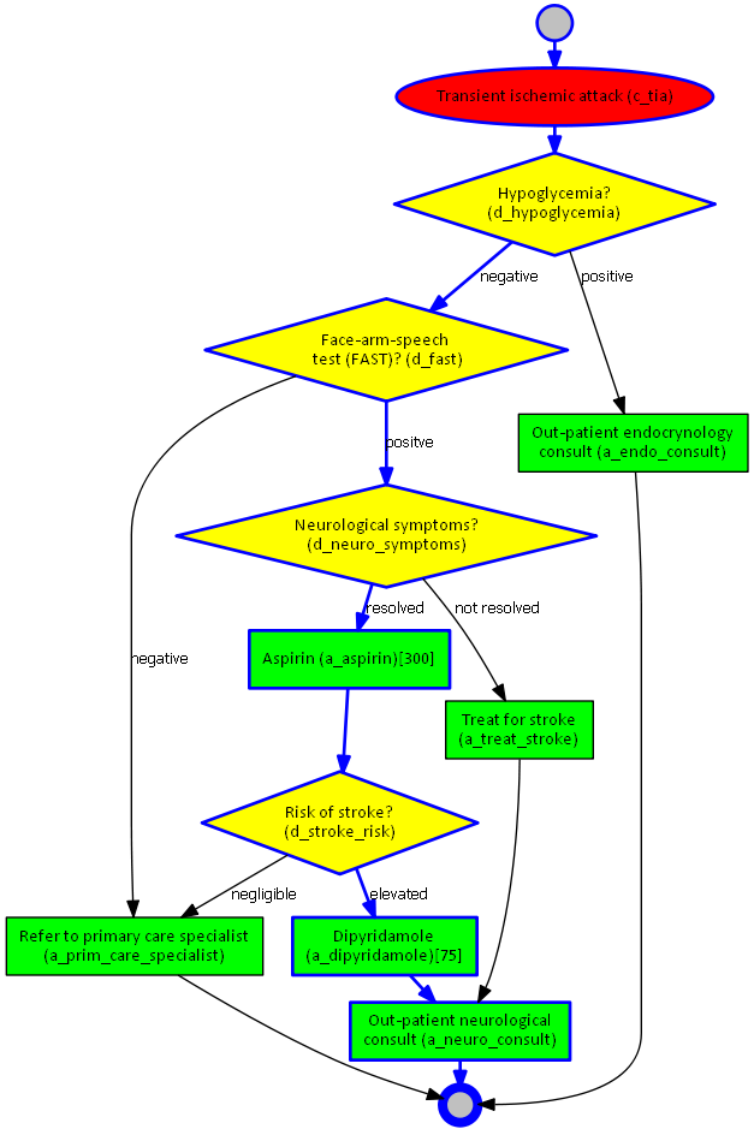
Rysunek 6.9: Graf wynikowy dla nadciśnienia

6.3 Scenariusz 3 - wrzód dwunastnicy i przemijający atak niedokrwienny

Wytyczne dla wrzodu dwunastnicy (ang. *duodenal ulcer*) przedstawiono na rys. 6.10, natomiast dla przemijającego ataku niedokrwiennego (ang. *transient ischemic attack*) na rys. 6.11.



Rysunek 6.10: Wytyczne dla wrzodu dwunastnicy



Rysunek 6.11: Wytyczne dla przemijającego ataku niedokrwiennego

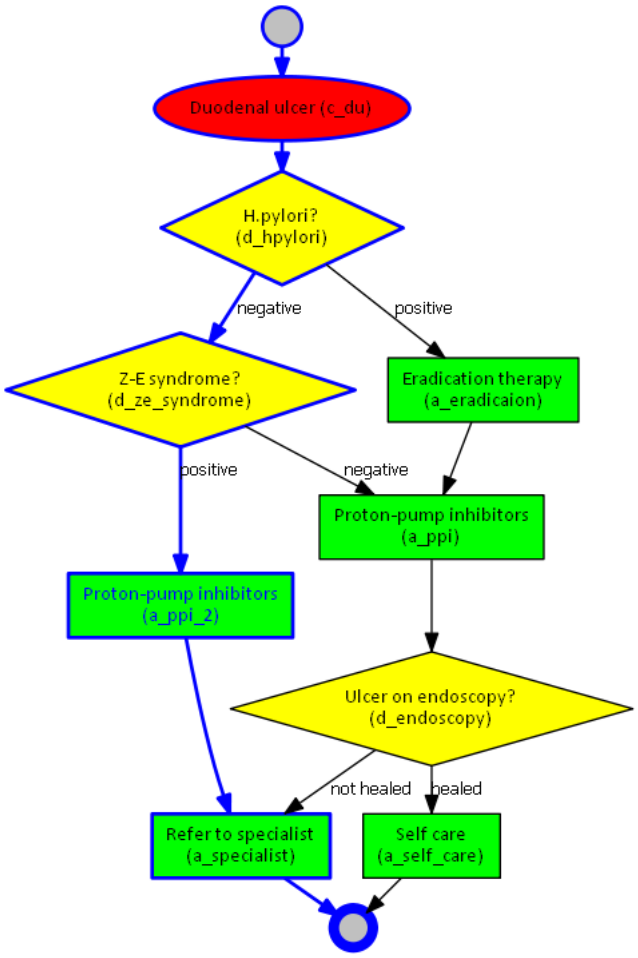
Dane opisujące stan pacjenta (odpowiedzi na pytania z wytycznych) są następujące:

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

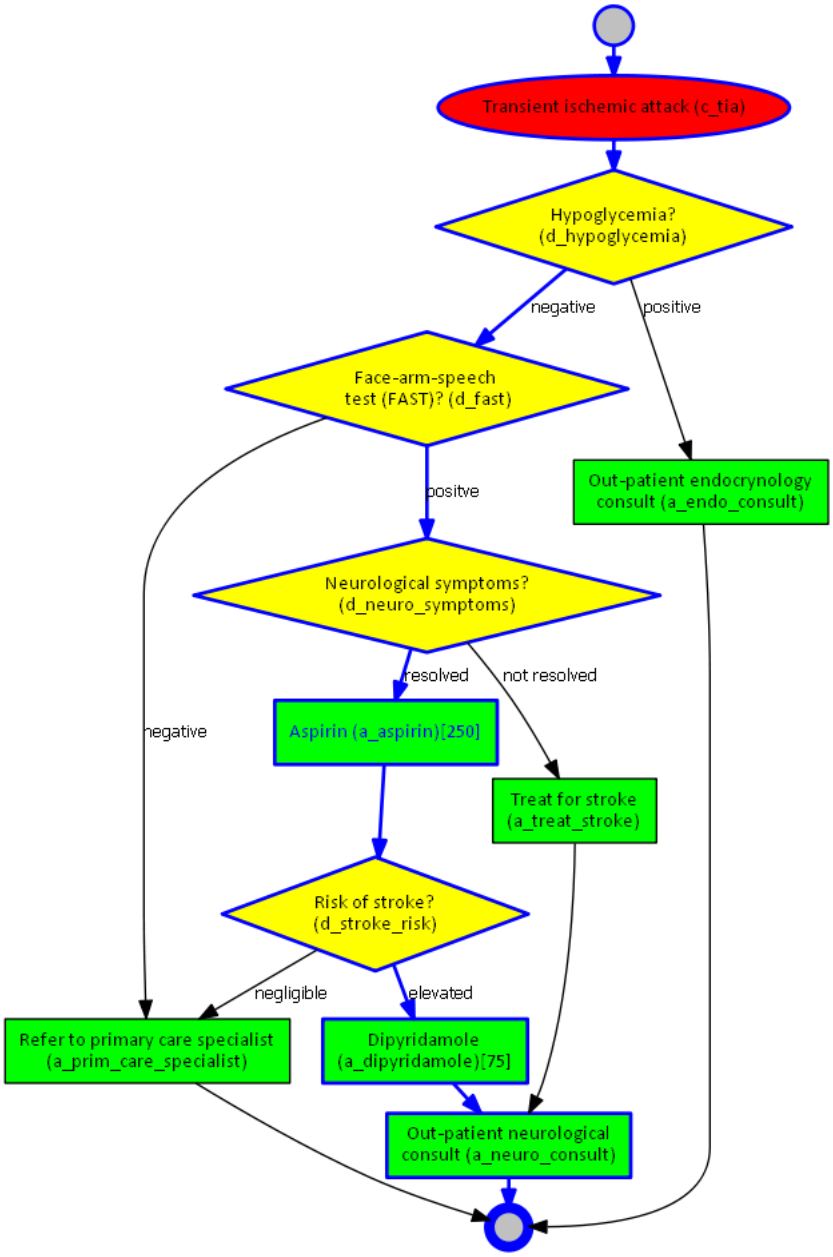
Dla rozważanych wytycznych możliwe są następujące konflikty:

1. c_du a_aspirin not(a_ppi) not(a_dipyridamole): replace a_aspirin with a_cl
2. **c_du a_aspirin not(a_ppi) a_dipyridamole: add a_ppi_2 after d_ze_syndrome?positive, decrease_dosage a_aspirin 50**

Graf wynikowy dla wrzodu dwunastnicy przedstawiono na rys. 6.12, natomiast dla przemijającego ataku niedokrwiennego na rys. 6.13.



Rysunek 6.12: Graf wynikowy dla wrzodu dwunastnicy



Rysunek 6.13: Graf wynikowy dla przemijającego ataku niedokrwiennego

Rozdział 7

Podsumowanie

7.1 Osiągnięte cele

W pracy udało się zrealizować wszystkie postawione cele. W szczególności rozszerzono podejście oparte na programowaniu logicznym z ograniczeniami zaproponowane w [14] pozwalając na stosowanie więcej niż dwóch wytycznych, dopuszczając złożone zmiany w wytycznych obejmujące wiele operacji oraz uwzględniając dawki podawanych leków. Rozszerzone podejście zaimplementowane zostało w formie interaktywnego systemu wspomagania decyzji klinicznych, który wyszukuje konflikty między zastosowanymi wytycznymi, wprowadza niezbędne zmiany do wytycznych i proponuje bezpieczne (tzn. pozbawione konfliktów) terapie.

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 z uwagi na jej prostotę. 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 błędów występujących podczas tworzenia tekstowej reprezentacji grafu. W szczególności konieczna była modyfikacja metody `toString` w klasach `Graph`, `Node` oraz `Edge`, ponieważ generowane początkowo przez bibliotekę tekstowe reprezentacje grafów nie były poprawnie przetwarzane przez program `dot`. Przyczyna błędu tkwiła w tym, że biblioteka JGPD nie radziła sobie z pustymi wartościami atrybutów wspomnianych obiektów. 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`) oraz okienkowej (`gvedit`).

7.3 Kierunki dalszego rozwoju

Kierunki dalszego rozwoju związane są zarówno z podejściem teoretycznym, jak i jego implementacją w formie systemu wspomagania decyzji. W ramach pierwszej grupy można rozważyć uwzględnianie czasu w wytycznych i konfliktach – konflikty występowałyby tylko wtedy, gdy dwie akcje byłyby wykonywane w tym samym czasie. Ponadto można byłoby uwzględnić koszty zmian związanych z usuwaniem konfliktu. Wtedy metoda wybierałaby rozwiązanie konfliktu o najmniejszym koszcie.

Jeśli chodzi o zaimplementowany system, to jego dalszy rozwój mógłby obejmować zastosowanie technik manipulacji bezpośredniej podczas udzielania odpowiedzi na pytania – program mógłby pozwalać na wybieranie krawędzi grafu zamiast pól wyboru. Ponadto program mógłby wspierać także inne reprezentacje grafów, nie tylko DOT. Dobrymi pomysłami byłyby także integracja programu z zewnętrznymi systemami w celu pobrania danych pacjenta oraz przygotowanie wersji na platformy 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] Isabel: the Diagnosis Checklisst. `http://www.isabelhealthcare.com/`. [Przeglądano 2015-09-29].

[5] JPGD - Java-based Parser for Graphviz Documents. `http://www.alexander-merz.com/graphviz`. [Przeglądano 2015-09-29].

[6] The ECLiPSe Constraint Programming System. `http://eclipseclp.org`. [Przeglądano 2015-09-29].

[7] 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.

[8] C.M. Boyd, J. Darer, C. Boult, L.P. Fried, L. Boult, and A.W. Wu. Clinical practice guidelines and quality of care for older patients with multiple comorbid diseases: implications for pay for performance. *JAMA*, 294:716–724, 2005.

[9] K. Janczura, T. Gabiga. Wprowadzenie do Programowanie Logicznego z Ograniczeniami z wykorzystaniem ECLiPSe.

[10] L. Piovesan, P. Terenziani. A Mixed-Initiative approach to the conciliation of Clinical Guidelines for comorbid patients. W: Proceedings of International Joint Workshop KR4HC 2015 - ProHealth 2015 (in conjunction with AIME 2015). 2015, 73-86.

[11] M. Musen, Y. Shahar, and E.H. Shortliffe. Clinical decision support systems. In E.H. Shortliffe and J.Cimino, editors, *Biomedical Informatics. Computer Applications in Health Care and Biomedicine*, pages 698–736. Springer, 2006.

[12] A. Niederliński. *Programowanie w logice z ograniczeniami. Łagodne wprowadzenie dla platformy ECLiPSe.* Wydawnictwo Pracowni Komputerowej Jacka Skalmierskiego, 2014.

[13] S. Wilk, M. Michalowski, X. Tan and W. Michalowski. Using First-Order Logic to Represent Clinical Practice Guidelines and to Mitigate Adverse Interactions. W: S. Miksch, D. Riano,

A. ten Teije (red.): Knowledge Representation for Health Care. 6th International Workshop, KR4HC 2014, held as part of the Vienna Summer of Logic, VSL 2014, Vienna, Austria, July 21, 2014. Revised Selected Papers. Springer, 2014, 45-61.

[14] 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.

[15] D.F. Sittig, A. Wright, J.A. Osheroff, B. Middleton, J.M. Teich, J.S. Ash, E. Campbell, and D.W. Bates. Grand challenges in clinical decision support. *J. Biomed. Inform.*, 41:387–92, 2008.