Wydział Matematyki i Nauk Informacyjnych
Politechniki Warszawskiej

Metody Sztucznej Inteligencji

System wnioskujący oparty o zasadę rezolucji

Andżelika Domańska, Przemysław Proszewski, Paweł Rzepiński,
Ryszard Szymański

4 grudnia 2018

Spis treści

1 Wstęp

Jednym z obszarów badań w zakresie metod sztucznej inteligencji są systemy eksperckie. Wspomagają one korzystanie z wiedzy i ułatwiają podejmowanie decyzji. Systemy ekspertowe znajdują zastosowanie w wielu dziedzinach, gdzie istnieją wysoko wyspecjalizowane zawody pomagając np. w przypadku oceny przez niewykwalifikowane osoby czy dana sytuacja wymaga interwencji chirurga.

2 Cel

Celem projektu jest stworzenie systemu wnioskującego na temat pewnego modelu rzeczywistości. System ten ma za zadanie odpowiadać na zapytania użytkownika. Na podstawie wiedzy domenowej zapisanej w postaci klauzul program ma przeprowadzić wnioskowanie metodą rezolucji i przedstawić odpowiedź.

3 Pojęcia wstępne

3.1 Klauzule

Klauzulą nazywamy skończony zbiór formuł logicznych. Klauzula jest uznawana za prawdziwą wtedy i tylko wtedy, gdy alternatywa jej elementów jest prawdziwa. W przypadku, gdy alternatywa formuł jest fałszywa, bądź zbiór ten jest pusty, to klauzulę uznajemy za fałszywą.

Klauzule zapisujemy w sposób następujący: $\vee_{i=1}^{n} x_i$, gdzie x_i są formułami logicznymi.

3.2 Koniunkcyjna postać normalna

Formuła jest w koniunkcyjnej postaci normalnej wtedy i tylko wtedy, gdy jest ona koniunkcją dowolnej liczby klauzul, z których każda jest alternatywą zmiennych zdaniowych, lub ich negacji. Każdą formułę niezawierającą kwantyfikatorów można przedstawić w koniunkcyjnej postaci normalnej.

Sprowadzenie formuły do koniunkcyjnej postaci normalnej jest wymagane do przeprowadzenia wywodu rezolucyjnego. Wyniki wnioskowania przeprowadzonego taką metodą nazywane są rezolwentami.

3.2.1 Algorytm sprowadzania do koniunkcyjnej postaci normalnej

1. Usunięcie implikacji zastępując je równoważną postacią:

   $a \implies b \equiv \neg a \vee b$

2. Zmiana zakresu negacji używając transformacji:

   - $\neg(\neg a) \equiv a$
   - $\neg(\exists x)\ a(x) \equiv (\forall x)\ \neg a(x)$
   - $\neg(\forall x)\ b(x) \equiv (\exists x)\ \neg b(x)$
   - $\neg(a \vee b) \equiv \neg a \wedge \neg b$
   - $\neg(a \wedge b) \equiv \neg a \vee \neg b$

3. Ustandaryzowanie zmiennych tak, aby każdy kwantyfikator był przypisany do innej zmiennej. Przykład takiej transformacji znajduje się poniżej:

   $((\forall x)\ a(x)) \vee ((\forall x)\ b(x)) \equiv ((\forall x)\ a(X)) \vee ((\forall y)\ b(y))$

4. Przeniesienie wszystkich kwantyfikatorów na skrajnie lewą stronę wyrażenia bez zmieniania ich kolejności. Zmiana nazw zmiennych w poprzednim kroku zapewniła, że nie powstaną w tym momencie konflikty związane z nazwenictwem zmiennych.

5. Usunięcie wszystkich kwantyfikatory egzystencjalne metodą skolemizacji. Jeśli zmienna w wyrażeniu posiada kwantyfikator egzystencjalny, to znaczy, że istnieje jej przypisanie dla którego wyrażenie jest prawdziwe. Przykład:

   $\forall x \exists y\ (x \vee y) \equiv \forall x\ (x \vee f(y))$, gdzie $f(y)$ jest funkcją zwracającą wartość egzystencjonalnego y

6. Usunięcie wszystkich kwantyfikatorów uniwersalnych

7. Sprowadzenie wyrażenie do koniunkcji klauzul wykorzystując transformacje z punktu 2. Przykład:

   $(x \vee (\neg y \wedge z)) \equiv (\neg x \wedge (y \vee z))$

3.3 Baza wiedzy

Baza wiedzy jest zbiorem wiedzy eksperckiej opisującym pewien model rzeczywistości dostarczonej początkowo do systemu. Do bazy wiedzy zaliczamy zarówno fakty, jak i reguły opisujące przyjęty model. Dostarczane reguły są zapisywane w koniunkcyjnej postaci normalnej. Taka reprezentacja pozwala na łatwe opisywanie związków przyczynowo skutkowych, relacji oraz powiązań.

Zbiór stwierdzeń uznawanych za prawdziwe może zmieniać swoją zawartość wraz z pojawianiem się nowych wniosków wynikających ze sprawdzania reguł.

4 Wnioskowanie

Wnioskowanie to proces wyprowadzania nowych zdań ze zdań przyjętych jako prawdziwe (tzn. reprezentujących fakty). Poprawne wnioskowanie powinno zapewniać prawdziwość wyprowadzanych zdań. Podczas tego procesu korzysta się między innymi z reguły *modus ponens* - jeżeli z przesłanki A wynika B oraz A jest prawdziwe przyjmujemy, że B również jest prawdziwe. Można ją zapisać jako:

$$\frac{A \Rightarrow B, A}{B}$$

W przypadku tego projektu system posiada pewne zdania w zapisane jako klauzule(w koniunkcyjnej postaci normalnej) w bazie wiedzy, otrzymuje zapytanie od użytkownika i przeprowadza wnioskowanie metodą rezolucji opisaną w osobnej sekcji.

5 Metoda rezolucji

Rezolucja jest metodą automatycznego dowodzenia twierdzeń, polega ona na generowaniu kolejnych klauzul prowadzących do sprzeczności. Korzystając z tego sposobu można udowodnić, że dane twierdzenie nie jest spełnialne lub, że jego zaprzeczenie jest tautologią. Działanie metoda rezolucji zostanie przedstawiona na przykładach w sekcji 8.

6 Opis systemu

Tworzony system wnioskujący na wejściu przyjmuje zdefiniowaną bazę wiedzy, na którą składają się fakty oraz reguły zapisane w koniunkcyjnej postaci normalnej. Do systemu użytkownik może wprowadzić tezę (również w koniunkcyjnej postaci normalnej), której prawdziwość ma za zadanie określić program. Przeprowadzony zostaje dowód za pomocą metody rezolucji oraz zostaje wyświetlone drzewo wywodu. Ponadto użytkownik dostaje informację zwrotną o tym, czy postawiona przez niego teza jest prawdziwa.

7 Implementacja

Algorytmy i podejścia wykorzystane w projekcie bazują na pracy [1] oraz wykładach [2].

7.1 Rezolucja

Funkcja zaprezentowana w 1 na podstawie otrzymanych argumentów: bazy wiedzy oraz zanegowanej tezy podanych w postaci zbioru klauzul, przeprowadza dowód rezolucyjny i zwraca informację, czy postawiona teza jest prawdziwa, oraz skonstruowane drzewo wywodu jako graf skierowany.

Listing 1: Pseudokod metody rezolucji

```
1  begin procedure resolution(knowledgeBase, negatedThesis)
2      clauses ← knowledgeBase ∪ negatedThesis
3      initialize resolutionTree as empty directed graph
4      add clauses to resolutionTree nodes
5
6      while true
7          new ← { }
8          for each pair C_i, C_j in clauses
9              resolvents ← resolve(C_i, C_j)
10
11             add resolvents to resolutionTree nodes
12             for each resolvent r in resolvents
13                 add edge (C_i, r) to resolutionTree
14                 add edge (C_j, r) to resolutionTree
15
16             if ∅ in resolvents
17                 resolutionTree ← reduce_tree(resolutionTree)
18                 return (true, resolutionTree)
19
20             new ← new ∪ resolvents
21
22         if new ⊂ clauses
23             return (false, resolutionTree)
24
25         clauses ← clauses ∪ new
26 end procedure
```

Na początku inicjalizowany jest zbiór klauzul *clauses* jako suma zbiorów *knowledgeBase* oraz *negatedThesis*. Jest to baza do przeprowadzenia dowodu za pomocą metody rezolucji. Następnie do budowanego drzewa wywodu dodawane są liście, czyli wszystkie elementy początkowego zbioru *clauses*. Jedno przejście pętli `while` odpowiada tworzeniu kolejnego poziomu w drzewie wywodu. Wówczas wyprowadzane są wszystkie możliwe rezolwenty dla każdej pary klauzul z aktualnego zbioru *clauses*. Każda nowa

rezolwenta r jest dodawana jako wierzchołek do drzewa wywodu oraz dodawane są krawędzie od klauzul, z których została wyprowadzona, do tej rezolwenty. To pozwala nam na późniejsze odtworzenie kolejnych kroków wnioskowania. Zaproponowane rozwiązanie zapewnia nam, że zostanie znalezione minimalne drzewo wywodu pod względem ilości poziomów. Jest tak, ponieważ gdy podczas wyprowadzania rezolwent na pewnym poziomie, uzyskamy $\emptyset$, dalsze obliczenia są przerywane, zwracane zostaje `true` oraz drzewo wywodu. Ponadto możliwe jest szybkie wykrycie tego, że postawiona teza jest fałszywa, ponieważ w każdym przejściu pętli sprawdzane jest to, czy udało się w danej iteracji wyprowadzić jakąkolwiek nową rezolwentę. Jeśli nie, dalsze obliczenia ponownie nie utworzą nowej klauzuli, zatem zostają przerwane, zwracany jest `false` oraz aktualne drzewo wywodu. Otrzymane drzewo wywodu będzie zawierało wszystkie otrzymane rezolwenty oraz klauzule wejściowe. Zatem będą istniały liście nie będące klauzulami pustymi $\emptyset$, z których nie przeprowadzono dalszego wywodu. W celu usunięcia z drzewa zbędnych informacji przed zakończeniem funkcji drzewo jest upraszczane. Dokładny opis tej operacji został umieszczony w sekcji 7.3.

7.2 Wyznaczanie rezolwent

Zaprezentowana poniżej funkcja `resolve` wyprowadza wszystkie możliwe rezolwenty danej pary klauzul: $clause_i$, $clause_j$.

```
1  begin procedure resolve(clause_i, clause_j)
2      resolvents ← { }
3      for literal_i in literals of clause_i
4          for literal_j in literals of clause_j
5              if literal_i.name == literal_j.name and literal_j.isNegated xor
                   literal_j.isNegated
6                  transformation ← find_transformation(literal_i, literal_j)
7              if transformation is None //there is no suitable transformation
8                  continue
9              else
10                 clause_i' ← substitute(clause_i, transformation)
11                 clause_j' ← substitute(clause_j, transformation)
12                 resolvents ← resolvents ∪ { combine(clause_i', clause_j') }
13     return resolvents
14 end procedure
```

Na początku inicjalizowany jest pusty zbiór $resolvents$. W klauzulach wejściowych wyszukiwane są przeciwne sobie literały tzn. takie, które mają tę samą nazwę i jeden z nich jest zanegowany. Następnie wyszukiwane jest odpowiednie podstawienie, które pozwoli na unifikację wejściowej pary klauzul. Poprawnymi podstawieniami są podstawienia zmiennych na stałe lub zmiennych na zmienne.

W przypadku gdy nie istnieje poprawna transformacja dla np. dla

$$(POSIADA(Janek, x), POSIADA(Franek, x))$$

, rozważana jest kolejna para literałów. W przeciwnym przypadku znalezione podstawienia zostają zastosowane do wejściowych klauzul i tak z przedstawionych klauzul wyznaczana jest rezolwenta za pomocą funkcji `combine`. Funkcja ta usuwa redukujące się literały i łączy pozostałe w jedną klauzulę, która jest następnie dodawana do zbioru $resolvents$.

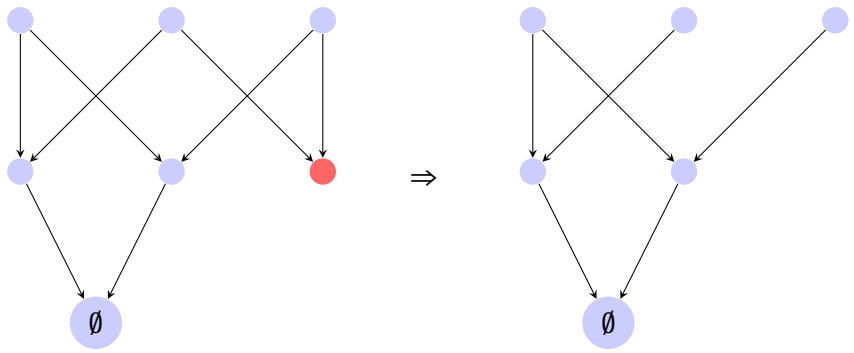
7.3 Upraszczanie drzewa wywodu

Funkcja zaprezentowana w 2 redukuje wejściowe drzewo wywodu $resolutionTree$.

```
1  begin procedure reduce_tree(resolutionTree)
2      change ← true
3      while change
4          nodesToDelete ← { }
5          for each node v in resolutionTree
6              if v.outDegree == 0 and v ≠ ∅
7                  nodesToDelete ← nodesToDelete ∪ {v}
8          change ← nodesToDelete.length > 0
9          remove nodesToDelete from resolutionTree
10     return resolutionTree
11 end procedure
```

Listing 2: Pseudokod funkcji $reduce\_tree$

W kolejnych iteracjach, dopóki zachodzą jakieś zmiany w drzewie, wyszukiwane są zbędne wierzchołki, a następnie usuwane. Kandydatem do usunięcia jest wierzchołek, który ma stopień wyjściowy równy 0 i nie jest klauzulą pustą. Przykład uproszczenia drzewa wywodu został przedstawiony na rysunku 1



Rysunek 1: Przykładowe uproszczenie drzewa wywodu

8 Przykłady

8.1 Janek i zwierzęta

> Wszystkie psy szczekają w nocy. Każda osoba posiadająca kota nie ma w domu myszy. Osoby kiepsko sypiające nie posiadają zwierzaka szczekającego w nocy. Janek ma psa lub kota. Czy jeśli Janek kiepsko sypia, to nie ma myszy w domu?

Korzystając z metody rezolucji oraz z informacji zawartych w powyższym tekście określona zostanie prawdziwość stwierdzenia *Jeśli Janek kiepsko sypia, to nie ma w domu myszy*.

Przykład jest dostępny w repozytorium w pliku *example.txt*.

8.1.1 Wyodrębnienie reguł oraz tezy

Najpierw należy z historyjki wyodrębnić reguły oraz tezę (5):

1. Wszystkie psy szczekają w nocy.
2. Każda osoba posiadająca kota nie ma w domu myszy.
3. Osoby kiepsko sypiające nie posiadają zwierzaka szczekającego w nocy.
4. Janek ma psa lub kota.
5. Jeśli Janek kiepsko sypia, to nie ma myszy. (TEZA)

8.1.2 Zapisanie reguł w logice pierwszego rzędu

Następnie znalezione reguły należy zapisać w logice pierwszego rzędu:

1. $\forall_x \ (PIES(x) \implies WYJE(x))$
2. $\forall_x \forall_y \ (POSIADA(x,y) \land KOT(y)) \implies \neg\exists_z \ (POSIADA(x,z) \land MYSZ(z))$
3. $\forall_x \ (KIEPSKO\_SYPIA(x) \implies \neg\exists_y \ (POSIADA(x,y) \land WYJE(y)))$
4. $\exists_x \ (POSIADA(Janek,x) \land (KOT(x) \lor PIES(x)))$
5. $KIEPSKO\_SYPIA(Janek) \implies \neg\exists_z \ (POSIADA(Janek,z) \land MYSZ(z))$

8.1.3 Sprowadzenie do koniunkcyjnej postaci normalnej

Kolejnym etapem jest sprowadzenie powyższych zasad do koniunkcyjnej postaci normalnej:

1. $\neg PIES(x) \vee WYJE(x)$
2. $\neg POSIADA(x,y) \vee \neg KOT(y) \vee \neg POSIADA(x,z) \vee \neg MYSZ(z)$
3. $\neg KIEPSKO\_SYPIA(x) \vee \neg POSIADA(x,y) \vee \neg WYJE(y)$
4. $POSIADA(Janek,x) \wedge (KOT(x) \vee PIES(x))$
5. $\neg KIEPSKO\_SYPIA(Janek) \vee \neg POSIADA(Janek,z) \vee \neg MYSZ(z)$ (TEZA)

8.1.4 Zanegowanie tezy

Następnie negujemy tezę:

5. $KIEPSKO\_SYPIA(Janek) \wedge POSIADA(Janek,z) \wedge MYSZ(z)$ (TEZA)

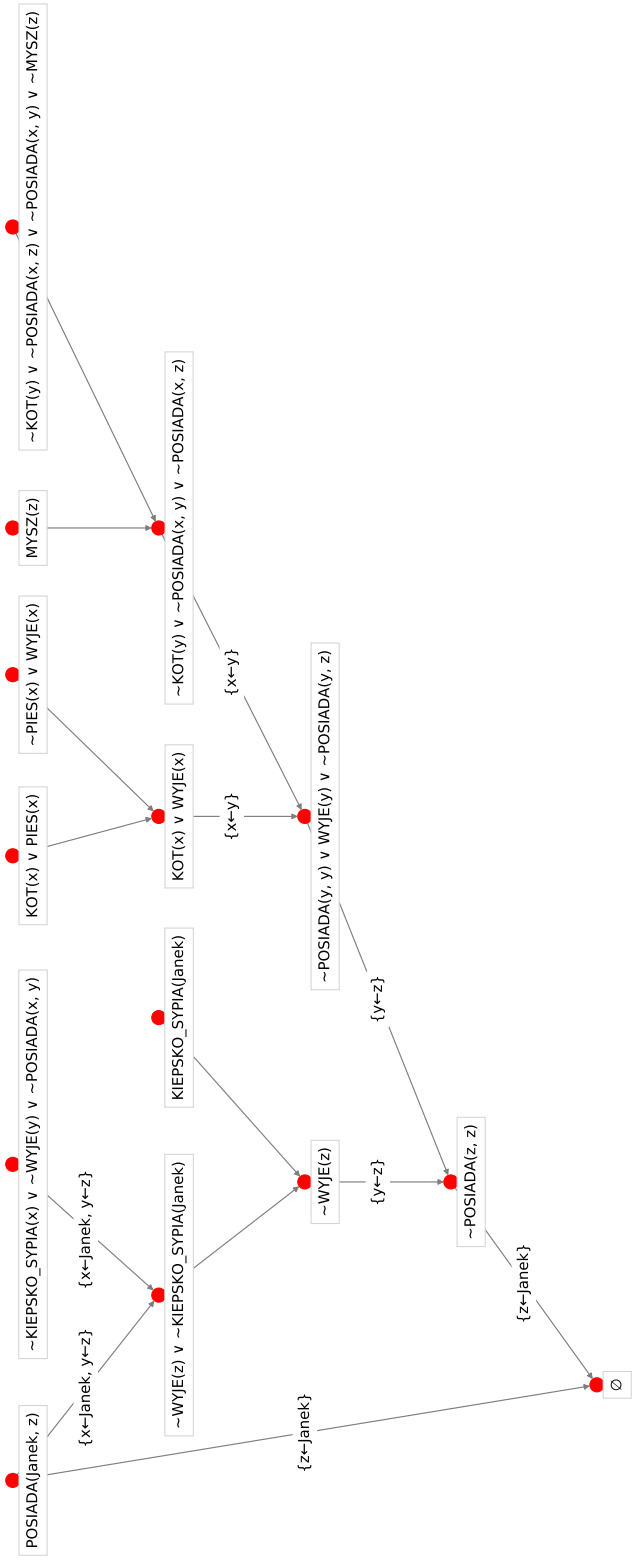
8.1.5 Wydzielenie klauzul

Ze zdań sprowadzonych do koniunkcyjnej postaci normalnej tworzone są klauzule:

1. $\neg PIES(x) \vee WYJE(x)$
2. $\neg POSIADA(x,y) \vee \neg KOT(y) \vee \neg POSIADA(x,z) \vee \neg MYSZ(z)$
3. $\neg KIEPSKO\_SYPIA(x) \vee \neg POSIADA(x,y) \vee \neg WYJE(y)$
4. a) $POSIADA(Janek,a)$
   b) $(KOT(a) \vee PIES(b))$
5. a) $KIEPSKO\_SYPIA(Janek)$
   b) $POSIADA(Janek,b)$
   c) $MYSZ(b)$

8.1.6 Budowanie drzewa wywodu

Mając wyodrębnione klauzule, można przejść do budowania drzewa wywodu. Liście tego drzewa to klazule naszego systemu, zaś pozostałe węzły to rezolwenty. Na podstawie tego drzewa szukamy możliwości wyprowadzenia słowa pustego.



Rysunek 2: Wizualizacja metody rezolucji dla prostego przykładu

Wynikiem wyprowadzenia widocznego na obrazku 2 jest słowo puste, zatem postawiona teza *Jeśli Janek kiepsko sypia, to nie ma w domu myszy* jest prawdziwa.

8.2 Rachunek zdań

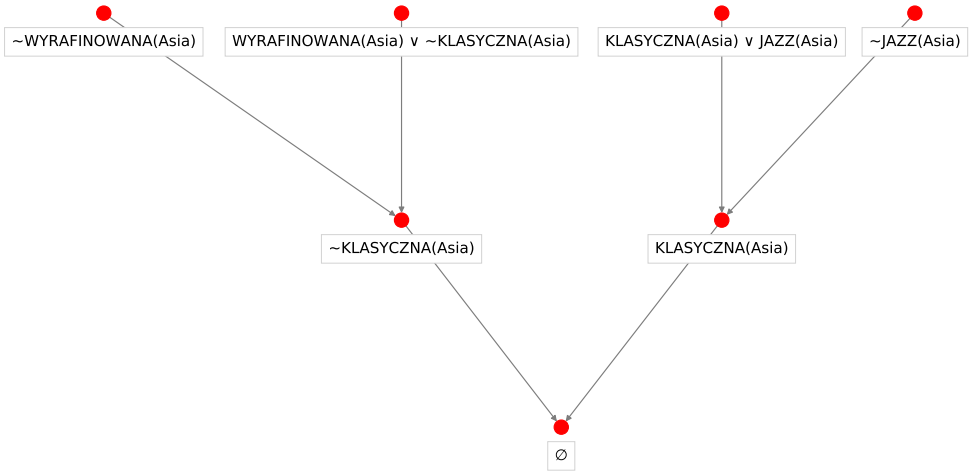
> Asia lubi muzykę klasyczną lub jazz. Jeżeli Asia lubi muzykę klasyczną, to jest wyrafinowana. Asia nie lubi jazzu.

Korzystając z metody rezolucji oraz z informacji zawartych w powyższym tekście określona zostanie prawdziwość stwierdzenia *Czy Asia jest wyrafinowana?*.

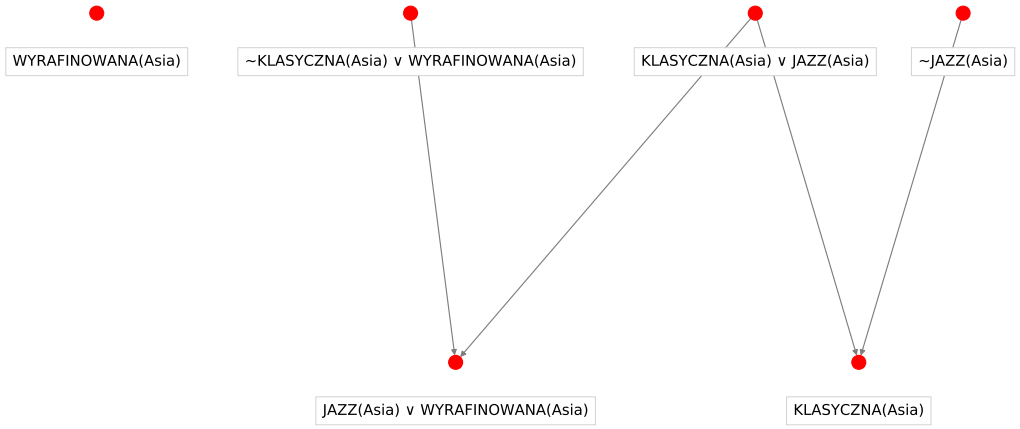
Na podstawie opisu w języku naturalnym tworzone są klauzule:

1. $KLASYCZNA(Asia) \vee JAZZ(Asia)$
2. $\neg KLASYCZNA(Asia) \vee WYRAFINOWANA(Asia)$
3. $\neg JAZZ(Asia)$

Na rysunku 3 zostało przedstawione drzewo wywodu dla zaprezentowanego przykładu i zanegowanej tezy $\neg WYRAFINOWANA(Asia)$. W przypadku, gdy zadamy tezę, która jest nieprawdziwa, a mianowicie $WYRAFINOWANA(Asia)$ otrzymamy drzewo wywodu, które nie zawiera pustej klauzuli, co ukazano na rysunku 4.



Rysunek 3: Wizualizacja metody rezolucji przeprowadzonej dla rachunku zdań



Rysunek 4: Wizualizacja metody rezolucji w przypadku nieprawdziwej tezy

8.3 Baza wiedzy z dodatkowymi informacjami

> Wszyscy ludzie, którzy nie są biedni i są zdolni, są szczęśliwi. Ci, którzy czytają są zdolni. Szczęśliwi ludzie mają ciekawe życie. Ludzie, którzy mają ciekawe życie, lub nie czytają mają dużo znajomych. Osoby, które mają dużo znajomych często się z nimi spotykają. Ci, którzy spotykają się ze znajomymi są szczęśliwi.
>
> Jan nie potrafi czytać, ale jest zamożny. Tomek nie potrafi czytać i jest biedny. Adam jest zdolny, ale nie jest szczęśliwy.

Korzystając z metody rezolucji oraz z informacji zawartych w powyższym tekście określona zostanie prawdziwość stwierdzenia *Czy istnieje osoba, która czyta, jest zdolna, szczęśliwa oraz ma znajomych?*.

8.3.1 Wydzielone klauzule oraz teza

Na podstawie opisu w języku naturalnym tworzone są klauzule:

1. $BIEDNY(x) \vee \neg ZDOLNY(x) \vee SZCZESLIWY(x)$
2. $\neg CZYTA(x) \vee ZDOLNY(x)$
3. $\neg SZCZESLIWY(x) \vee CIEKAWE(x)$
4. a) $\neg CIEKAWE(a) \vee ZNAJOMI(a)$
   b) $\neg CZYTA(a) \vee ZNAJOMI(a)$
5. $\neg ZNAJOMI(b) \vee SPOTYKA(b)$
6. $\neg SPOTYKA(c) \vee SZCZESLIWY(c)$
7. $\neg CZYTA(Jan) \wedge \neg BIEDNY(Jan)$
8. $\neg CZYTA(Tomek) \wedge BIEDNY(Tomek)$
9. $\neg BIEDNY(Adam) \wedge CZYTA(Adam)$

Zanegowana teza:
$\neg ZNAJOMI(x) \wedge \neg CZYTA(x) \wedge \neg ZDOLNY(x) \wedge \neg SZCZESLIWY(x)$

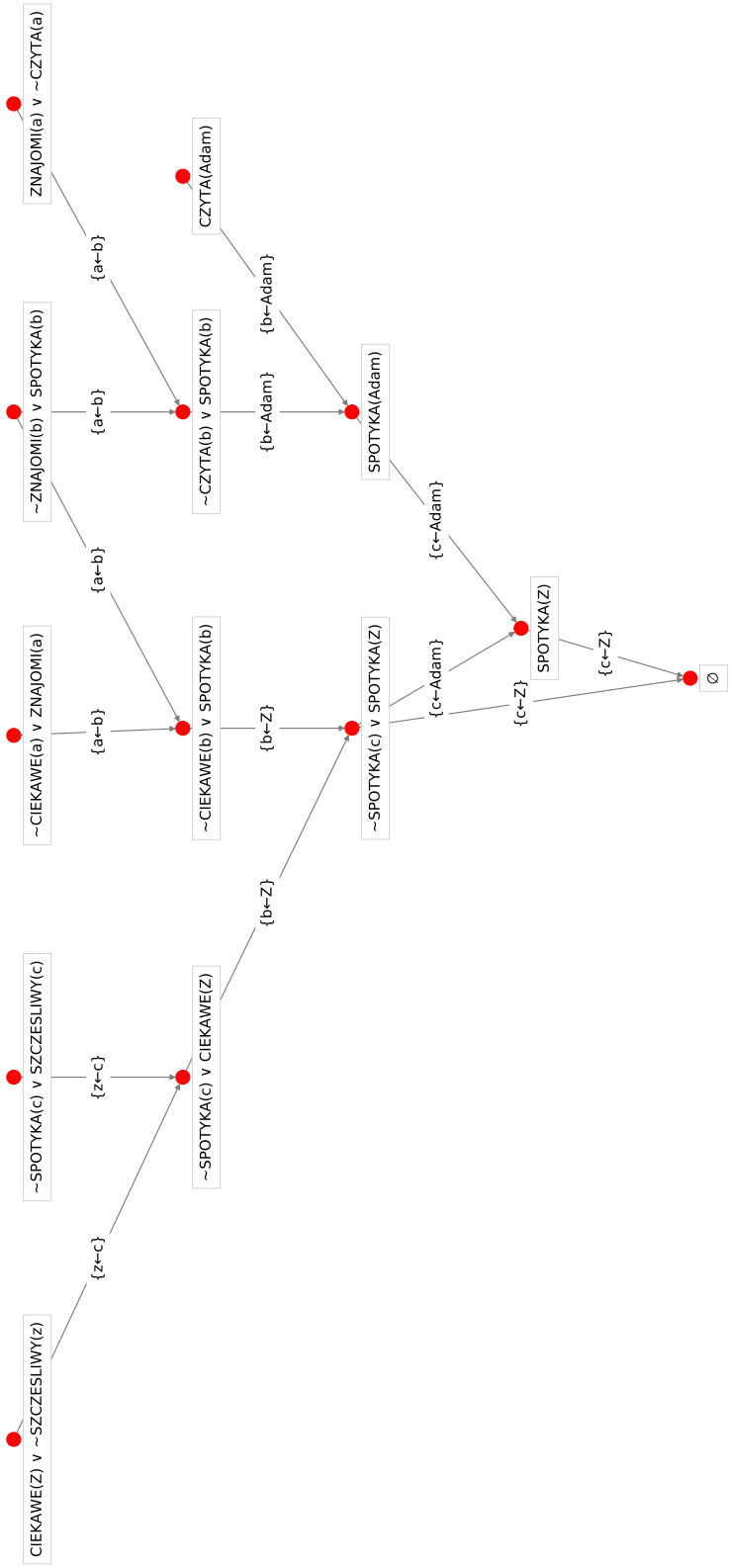
8.3.2 Drzewa wywodu

Na rysunku 5 zostało przedstawione drzewo wywodu dla zaprezentowanego przykładu.

Wynikiem powyższego wyprowadzenia jest słowo puste, zatem postawiona teza *Czy istnieje osoba, która czyta, jest zdolna, szczęśliwa oraz ma znajomych?* jest prawdziwa.

8.3.3 Uwagi

Informacje z klauzul, które nie dotyczą zadanej tezy nie zostały umieszczone w drzewie wywodu.



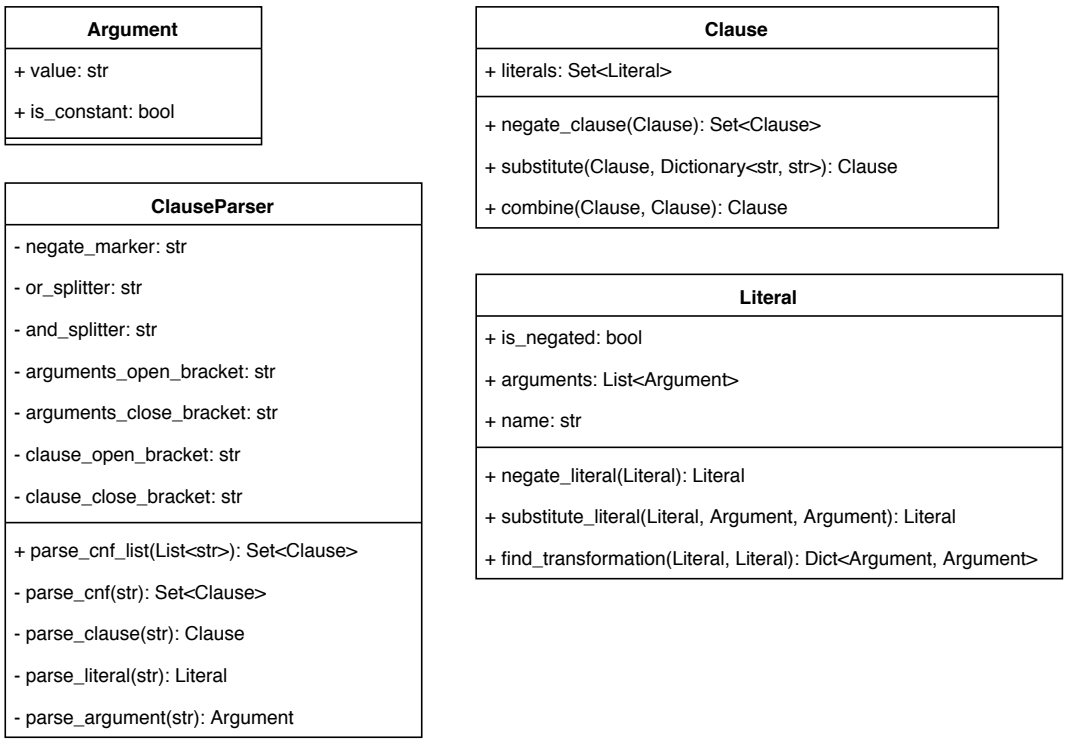
Rysunek 5: Wizualizacja metody rezolucji dla rozbudowanego przykładu

9 Architektura rozwiązania

Zaprojektowany system składa się z trzech modułów:

- Parser – przyjmuje zdania w koniunkcyjnej postaci normalnej i zwraca na ich podstawie zbiór klauzul.
- Narzędzie do przeprowadzenia rezolucji – Na podstawie bazy wiedzy oraz tezy przeprowadza dowód rezolucyjny i zwraca informację, czy zadana teza jest prawdziwa oraz drzewo wywodu w postaci grafu skierowanego.
- Narzędzie do wizualizacji – Na podstawie grafu skierowanego obrazuje drzewo wywodu. Pozwala na zapisanie grafiki między innymi w formatach *.pdf, .png*.

Klasy reprezentujące składowe koniunkcyjnej postaci normalnej formuł logicznych zostały przedstawione na diagramie 6.



Rysunek 6: Diagram klas reprezentujących obiekty użyte w systemie

9.1 Moduły

Projekt został wykonany w języku *Python* w wersji 3.6.

9.1.1 Parser

Parser pozwala na sprecyzowanie znaków specjalnych wykorzystywanych podczas wydzielania klauzul z danych wejściowych. Domyślnie wykorzystywana jest następująca konfiguracja:

- $\sim$ - znak negacji
- | - symbol alternatywy
- & - symbol koniunkcji
- (- symbol nawiasu otwierającego listę argumentów
-) - symbol nawiasu zamykającego listę argumentów
-] - symbol nawiasu otwierającego listę klauzul
- [- symbol nawiasu zamykającego listę klauzul

Każda klauzula powinna znajdować się w osobnej linii. Ponadto, zanegowana teza powinna zostać umieszczona po separatorze w postaci ---.

9.1.2 Narzędzie do wizualizacji

Do wizualizacji drzewa wywodu zostały wykorzystane biblioteki *matplotlib, graphviz*. Dzięki drugiej z nich uzyskano hierarchiczną strukturę drzewa. Możliwa jest modyfikacja stylu wykorzystywanego podczas reprezentacji graficznej grafu.

9.1.3 Narzędzie do przeprowadzania rezolucji

Do utworzenia grafu wynikowego zawierającego drzewo wywodu została wykorzystana biblioteka *networkx*.

Bibliografia

1. W. S. Franz Baader. „Handbook of Automated Reasoning; Chapter 8 - Unification-Theory”, 2001. URL: https://www.cs.bu.edu/~snyder/publications/UnifChapter.pdf.
2. D. Ślęzak. *Sztuczna inteligencja i systemy doradcze; Wykład 5 - Wnioskowanie w rachunku zdan*. URL: https://www.mimuw.edu.pl/~wjaworski/SI/wyklad5.pdf.