Dokumentacja końcowa projektu z Podstaw Sztucznej Inteligencji:

0. Spis treści:

1. Skład zespołu:

- Kurek Wojciech
- Belniak Michał
- Szachewicz Jakub

2. Zadanie:

- PW.W.2

3. Treść zadania:

3.1 Maszyna wnioskująca

Napisać program, który przyjmuje:

- formuły koniunkcyjno-implikacyjne bez negacji w rachunku predykatów pierwszego rzędu
- tezę do udowodnienia.

Następnie, sprowadza formuły do postaci klauzul, dalej do formuł koniunkcyjnoimplikacyjnych i przeprowadza wnioskowanie wstecz dowodzące zadanej tezy.
Program może nie obsługiwać funkcji, czyli może nie akceptować formuł postaci PRED(F(x)). Wynik wnioskowania zostaje zademonstrowany graficznie.

4. Podstawowe wymagania i uściślenia:

Projekt zakłada napisanie programu odczytującego formuły koniunkcyjno-implikacyjne z pliku tekstowego oraz dający w rezultacie swojego działania tekstowo-graficzną reprezentację wynikowego grafu powstałego w ramach procedury wnioskowania wstecz przeprowadzonego na podstawie odczytanych formuł koniunkcyjno-implikacyjnych.

5. Sposób reprezentacji danych:

Formuły koniunkcyjno-implikacyjne mają być zapisane za pomocą specjalnej notacji tekstowej pozwalającej reprezentować m.in. symbol implikacji (=>) oraz symbol koniunkcji (^).

6. Koceptualne rozwiązanie problemu:

Częścią składową programu jest parser umożliwiający odzwierciedlenie tekstowej reprezentacji koniunkcyjno-implikacyjnych w ramach modelu obiektowego.

Każda poprawnie zdefiniowana formuła ma zostać zmapowana na obiekt klasy `Clausure`. Obiekty klasy `Clausure` stanowić mają bazę wiedzy wykorzystywaną w procesie wnioskowani wstecz. Częścią progamu jest obiektowa reprezentacja poszczególnych części formuły koniunkcyjno-implikacyjnej. Uwzględnioną są typy podstawowe takich jak model.Constant, model.Variable.

Predykat jest reprezentowany przez klasę model.Predicate zawierającą w sobie obiekty typu model.Constant i model.Variable.

Nazwę predykatu traktujemy jako stałą (model.Constant). Reprezentację obiektową posiadają również operatory – AND i NOT (operator implikacji traktowane są oddzielnie).

Ich implementacja zakłada możliwość pobierania pierwszego operandu operacji oraz pozostałych operandów operatora. Operator implikacji jest zdefiniowany w postaci klasy Clausure, zakłada on jako pierwszy operand obiekt klasy model.Predicate, drugi operand może być operacją lub predykatem.

7. Sposób działania algorytmu:

7.1 Struktury danych:

Implementacja algorytmu wnioskowania wstecz zakłada wykorzystanie struktury grafu.

7.2 Podejście:

Wykorzystywany jest mechanizm rekurencji.

7.3 Działanie

Kolejne stany algorytmu w których dla określonego (zaprzeczonego) predykatu poszukujemy formuły koniunkcyjno-implikacyjnej z unifikowalną z nim konkluzją reprezentujemy jako węzeł w grafie.
W kolejnym kroku algorytmu analizujemy ogon zdania, które zostało poddane unifikacji. W sytuacji, gdy ogon zdania był w postaci koniunkcji predykatów to też reprezentujemy go początkowo jako jeden węzeł, który później będzie rozbijany na następne węzły grafu poprzez kolejne wydzielanie pierwszego operandu z koniunkcji predykatów.
Z każdym węzłem jest związany zestaw niewykorzystanych jeszcze obiektów typu Clausure (baza wiedzy), sumaryczny zestaw podstawień użyty do uzyskania danego węzła oraz id obiektu Clausure w wyniku unifikacji którego powstał ten węzeł.

8. Wynik działania algorytmu:

8.1 Wyjście:

Program na wyjściu drukuje wynikowy graf powstały w ramach przebiegu algorytmu.

Istnieje konieczność zapewnienia możliwości znakowej reprezentacji poszczególnych węzłów grafu wynikowego. Wynik programu ma być wyświetlony w jak najprostszej i czytelnej dla użytkownika końcowego formie.

9. Uruchomienie:

9.1 Argumenty uruchomienia

Program przyjmuje dane od użytkownika w formie **paramterów uruchomieniowych** cztery parametry.

- Baza wiedzy w postaci formuł koniunkcyjno-implikacyjnych. Baza wiedzy jest przekazywana w formie pliku, do którego ścieżkę przekazujemy programowi parametrem `-knowledgeBase`. Kolejne formuły pisane jedna pod drugą. Implikacja oznaczona poprzez symbol **=>**.
- Użyte w bazie wiedzy stałe (w tym nazwy predykatów). Stałe są przekazywane w formie pliku, do którego ścieżkę przekazujemy programowi parametrem `-const` lub `-constants`. Kolejne stałe są pisane w pliku jedna pod drugą.
- Użyte w bazie wiedzy zmienne. Zmienne są przekazywane w formie pliku, do którego ścieżkę przekazujemy programowi parametrem `-var` lub `-variables`. Kolejne zmienne są pisane w pliku jedna pod drugą.
- Teza do udowodnienia przez algorytm.Teza jest przekazywana w formie pliku, do którego ścieżkę przekazujemy programowi parametrem `-arg` lub `-argument`. Waże by stałe oraz nazwa predykatu użye w tezie były również uwzględnione w plikach ze stałymi.

9.2 Przykład uruchomienia algorytmu:

```
$ java -jar ./MaszynaWnioskujaca.jar -knowledgeBase knowledge_base.txt -var
variables.txt -const constants.txt -arg argument.txt
```

11. Przykłady działania

Przykład 6:

"The law says that it is a crime for an American to sell weapons to hostile nations. The country Nono, an enemy of America, has some missiles, and all of its missiles were sold to it by Colonel West, who is American."

They said: "**Prove that Colonel West is a criminal**"

Knowledge Base:

● . . . it is a crime for an American to sell weapons to hostile nations:

$$American(x) \land Weapon(y) \land Sells(x, y, z) \land Hostile(z) \Rightarrow Criminal(x)$$

● Nono . . . has some missiles

$$Owns(Nono,M1) \text{ and } Missile(M1)$$

● . . . all of its missiles were sold to it by Colonel West

$$\forall x\ Missile(x) \land Owns(Nono, x) \Rightarrow Sells(West, x, Nono)$$

● Missiles are weapons:

$$Missile(x) \Rightarrow Weapon(x)$$

● An enemy of America counts as "hostile":

$$Enemy(x, America) \Rightarrow Hostile(x)$$

● West, who is American . . .

$$American(West)$$

● The country Nono, an enemy of America . . .

$$Enemy(Nono, America)$$

Do udowodnienia:

Criminal(West)

Pliki przekazywane programowi:

argument.txt

```
CRIMINAL(WEST)
```

knowledge_base.txt

```
(AMERICAN(x0)^WEAPON(x1)^SELLS(x0,x1,x2)^HOSTILE(x2))=>CRIMINAL(x0)
(MISSILE(x3)^OWNS(NONO,x3))=>SELLS(WEST,x3,NONO)
MISSILE(x4)=>WEAPON(x4)
ENEMY(x5,AMERICA)=>HOSTILE(x5)
AMERICAN(WEST)
ENEMY(NONO,AMERICA)
OWNS(NONO,M)
MISSILE(M)
```

variables.txt

```
x0
x1
x2
x3
x4
x5
```

constants.txt

```
AMERICAN
WEAPON
SELLS
HOSTILE
CRIMINAL
MISSILE
ENEMY
OWNS
NONO
M
WEST
AMERICA
```

Wynik działania algorytmu:

```
Baza wiedzy:
K1: AMERICAN(x0_0) ^ WEAPON(x1_1) ^ SELLS(x0_0, x1_1, x2_2) ^ HOSTILE(x2_2) =>
CRIMINAL(x0_0)
K2: MISSILE(x3_3) ^ OWNS(NONO, x3_3) => SELLS(WEST, x3_3, NONO)
K3: MISSILE(x4_4) => WEAPON(x4_4)
K4: ENEMY(x5_5, AMERICA) => HOSTILE(x5_5)
K5: AMERICAN(WEST)
K6: ENEMY(NONO, AMERICA)
K7: OWNS(NONO, M)
K8: MISSILE(M)
K9: WEAPON(x1_7) ^ SELLS(x0_7, x1_7, x2_7) ^ HOSTILE(x2_7) ^ ~CRIMINAL(x0_7) =>
~AMERICAN(x0_7)
K10: AMERICAN(x0_8) ^ SELLS(x0_8, x1_8, x2_8) ^ HOSTILE(x2_8) ^ ~CRIMINAL(x0_8) =>
~WEAPON(x1_8)
K11: AMERICAN(x0_9) ^ WEAPON(x1_9) ^ HOSTILE(x2_9) ^ ~CRIMINAL(x0_9) => ~SELLS(x0_9,
x1_9, x2_9)
K12: AMERICAN(x0_10) ^ WEAPON(x1_10) ^ SELLS(x0_10, x1_10, x2_10) ^ ~CRIMINAL(x0_10) =>
~HOSTILE(x2_10)
K13: OWNS(NONO, x3_11) ^ ~SELLS(WEST, x3_11, NONO) => ~MISSILE(x3_11)
K14: MISSILE(x3_12) ^ ~SELLS(WEST, x3_12, NONO) => ~OWNS(NONO, x3_12)
K15: ~WEAPON(x4_13) => ~MISSILE(x4_13)
K16: ~HOSTILE(x5_14) => ~ENEMY(x5_14, AMERICA)

Teza: CRIMINAL(WEST)

Zbiór podstawień: {x0_0/WEST, x5_5/NONO, x4_4/M, x3_3/x4_4, x1_1/x4_4, x2_2/NONO}

Użyte klauzule:
K6: ENEMY(NONO, AMERICA)
K4: ENEMY(x5_5, AMERICA) => HOSTILE(x5_5)
K7: OWNS(NONO, M)
K8: MISSILE(M)
K2: MISSILE(x3_3) ^ OWNS(NONO, x3_3) => SELLS(WEST, x3_3, NONO)
K8: MISSILE(M)
K3: MISSILE(x4_4) => WEAPON(x4_4)
K5: AMERICAN(WEST)
K1: AMERICAN(x0_0) ^ WEAPON(x1_1) ^ SELLS(x0_0, x1_1, x2_2) ^ HOSTILE(x2_2) =>
CRIMINAL(x0_0)

Graf wnioskowania:
(CRIMINAL(WEST))
    ^
    |
(AMERICAN(WEST) ^ WEAPON(M) ^ SELLS(WEST, M, NONO) ^ HOSTILE(NONO))
    ^
    |
(AMERICAN(WEST))
    ^
    |
(MISSILE(M))
    ^
    |
(MISSILE(M) ^ OWNS(NONO, M))
    ^
    |
(MISSILE(M))
    ^
    |
(OWNS(NONO, M))
    ^
    |
(ENEMY(NONO, AMERICA))
```

Przykład 3

“Każdy kocha wspinaczkę lub narciarstwo. Miłośnicy wspinaczki nie lubią deszczu, miłośnicy narciarstwa zaś lubią śnieg. Abacki lubi to, czego nie lubi Babacki, zaś Babacki to, czego nie lubi Abacki. Abacki lubi deszcz oraz śnieg.”

Powiedzieli: „**Udowodnij, że Babacki kocha Wspinaczkę**”

Knowledge Base:

● Każdy kocha wspinaczkę lub narciarstwo

$\sim Kocha(x,Wspin) \Rightarrow Kocha(x,Narty)$

● Miłośnicy wspinaczki nie lubią deszczu,

$Kocha(x,Wspin) \Rightarrow \sim Lubi(x,Deszcz)$

● . . . miłośnicy narciarstwa zaś lubią śnieg.

$Kocha(x,Narty) \Rightarrow Lubi(x,Snieg)$

● Abacki lubi to, czego nie lubi Babacki,

$Lubi(Abacki, x) \Rightarrow \sim Lubi(Babacki,x)$

● . . . zaś Babacki to, czego nie lubi Abacki.

$\sim Lubi(Babacki, x) \Rightarrow Lubi(Abacki,x)$

● Abacki lubi deszcz. . .

$Lubi(Abacki, Deszcz)$

● Abacki lubi (. . .) snieg.

$Lubi(Abacki, Snieg)$

Do udowodnienia:

$Kocha(Babacki, Wspin)$

Pliki przekazywane programowi:

argument.txt

```
KOCHA(Babacki,Wspin)
```

knowledge_base.txt

```
~KOCHA(x0,Wspin)=>KOCHA(x0,Narty)
KOCHA(x1,Wspin)=>~LUBI(x1,Descz)
KOCHA(x2,Narty)=>LUBI(x2,Snieg)
LUBI(Abacki,x3)=>~LUBI(Babacki,x3)
~LUBI(Babacki,x4)=>LUBI(Abacki,x4)
LUBI(Abacki,Descz)
LUBI(Abacki,Snieg)
```

variables.txt

```
x0
x1
x2
x3
x4
```

constants.txt

```
KOCHA
LUBI
~KOCHA
~LUBI
Wspin
Narty
Descz
Snieg
Abacki
Babacki
```

Rezulatat działania programu:

```
Baza wiedzy:
K1: ~KOCHA(x0_0, Wspin_4) => KOCHA(x0_0, Narty_5)
K2: KOCHA(x1_1, Wspin_4) => ~LUBI(x1_1, Descz_6)
K3: KOCHA(x2_2, Narty_5) => LUBI(x2_2, Snieg_7)
K4: LUBI(Abacki_8, x3_3) => ~LUBI(Babacki_9, x3_3)
K5: ~LUBI(Babacki_9, x4_4) => LUBI(Abacki_8, x4_4)
K6: LUBI(Abacki_8, Descz_6)
K7: LUBI(Abacki_8, Snieg_7)
K8: ~KOCHA(x0_6, Narty_5) => KOCHA(x0_6, Wspin_4)
K9: LUBI(x1_7, Descz_6) => ~KOCHA(x1_7, Wspin_4)
K10: ~LUBI(x2_8, Snieg_7) => ~KOCHA(x2_8, Narty_5)
K11: LUBI(Babacki_9, x3_9) => ~LUBI(Abacki_8, x3_9)
K12: ~LUBI(Abacki_8, x4_10) => LUBI(Babacki_9, x4_10)

Teza: KOCHA(Babacki_9, Wspin_4)

Zbiór podstawień: {x0_0/Abacki_8, x2_2/Abacki_8, x3_3/Snieg_7, x1_7/Abacki_8,
x0_6/Babacki_9, x2_8/Babacki_9}

Użyte klauzule:
K6: LUBI(Abacki_8, Descz_6)
K9: LUBI(x1_7, Descz_6) => ~KOCHA(x1_7, Wspin_4)
K1: ~KOCHA(x0_0, Wspin_4) => KOCHA(x0_0, Narty_5)
K3: KOCHA(x2_2, Narty_5) => LUBI(x2_2, Snieg_7)
K4: LUBI(Abacki_8, x3_3) => ~LUBI(Babacki_9, x3_3)
K10: ~LUBI(x2_8, Snieg_7) => ~KOCHA(x2_8, Narty_5)
K8: ~KOCHA(x0_6, Narty_5) => KOCHA(x0_6, Wspin_4)

Graf wnioskowania:
(KOCHA(Babacki_9, Wspin_4))
    ^
    |
(~KOCHA(Babacki_9, Narty_5))
    ^
    |
(~LUBI(Babacki_9, Snieg_7))
    ^
    |
(LUBI(Abacki_8, Snieg_7))
    ^
    |
(KOCHA(Abacki_8, Narty_5))
    ^
    |
(~KOCHA(Abacki_8, Wspin_4))
    ^
    |
(LUBI(Abacki_8, Descz_6))
```

Przykład 7

"Wolves, foxes, birds, caterpillars, and snails are animals. Also there are some grains, and grains are plants. Every animal either likes to eat plants or animals much smaller than itself that like to eat some plants. Caterpillars and snails are much smaller than birds, which are much smaller than foxes, which are in turn much smaller than wolves.
Wolves do not like to eat foxes or grains, while birds do not like to eat snails. Snails like to eat some plants and caterpillars like to eat grains.
We know, that Animal1 is a bird, Grain1 is a grain and Animal2 is a caterpillar."

They said: "**Prove, that Animal1 eats Animal2."**

Knowledge Base:

● Wolves, (. . .) are animals.

$$Wolf(x) \Rightarrow Animal(x)$$

● . . . foxes, (. . .) are animals.

$$Fox(x) \Rightarrow Animal(x)$$

● (. . .) birds (. . .) are animals.

$$Bird(x) \Rightarrow Animal(x)$$

● (. . .) caterpillars, (. . .) are animals.

$$Caterpillar(x) \Rightarrow Animal(x)$$

● (. . .) and snails are animals.

$$Snail(x) \Rightarrow Animal(x)$$

● , and grains are plants(. . .)

$$Grain(x) \Rightarrow Plant(x)$$

● Every animal either likes to eat plants or animals much smaller than itself that like to eat some plants.

$$\text{Animal}(x) \land \text{Plant}(y) \land \sim\text{Eats}(x,y) \land \text{Animal}(z) \land \text{Smaller}(z,x) \land \text{Plant}(u) \land \text{Eats}(z,u) \Rightarrow \Rightarrow \text{Eats}(x,z)$$

● Caterpillars and snails are much smaller than birds, which are much smaller than foxes, which are in turn much smaller than wolves.

$$\text{Caterpillar}(x) \land \text{Bird}(y) \Rightarrow \text{Smaller}(x,y)$$

$$\text{Snail}(x) \land \text{Bird}(y) \Rightarrow \text{Smaller}(x,y)$$

$$\text{Bird}(x) \land \text{Fox}(y) \Rightarrow \text{Smaller}(x,y)$$

$$\text{Fox}(x) \land \text{Wolf}(y) \Rightarrow \text{Smaller}(x,y)$$

● Miłośnicy wspinaczki nie lubią deszczu,

$$\text{Bird}(x) \land \text{Grain}(y) \Rightarrow \sim\text{Eats}(x,y)$$

● caterpillars like to eat grains

$$\text{Caterpillar}(x) \land \text{Grain}(y) \Rightarrow \text{Eats}(x,y)$$

● Snails like to eat some plants,

$$\text{Snail}(x) \land \text{Plant}(y) \Rightarrow \text{Eats}(x,y)$$

● Wolves do not like to eat foxes (. . .)

$$\text{Wolf}(x) \land \text{Fox}(y) \Rightarrow \sim\text{Eats}(x,y)$$

● Wolves do not like to eat (. . .) grains,

$$\text{Wolf}(x) \land \text{Grain}(y) \Rightarrow \sim\text{Eats}(x,y)$$

● while birds do not like to eat snails.

$$\text{Bird}(x) \land \text{Snail}(y) \Rightarrow \sim\text{Eats}(x,y)$$

- Animal1 is a bird

Bird(Animal1)

- Grain1 is a grain

Grain(Grain1)

- Animal2 is a caterpillar.

Caterpillar(Animal2)

Do udowodnienia:

Eats(Animal1, Animal2)

Pliki przekazywane programowi:

argument.txt

```
EATS(Animal1,Animal2)
```

knowledge_base.txt

```
WOLF(x1)=>ANIMAL(x1)
FOX(x2)=>ANIMAL(x2)
BIRD(x3)=>ANIMAL(x3)
CATERPILLAR(x4)=>ANIMAL(x4)
SNAIL(x5)=>ANIMAL(x5)
GRAIN(x6)=>PLANT(x6)
GRAIN(x7)=>PLANT(x7)
(ANIMAL(x8)^PLANT(y8)^~EATS(x8,y8)^ANIMAL(z8)^SMALLER(z8,x8)^PLANT(u8)^EATS(z8,u8))=>EA
TS(x8,z8)
(CATERPILLAR(x9)^BIRD(y9))=>SMALLER(x9,y9)
(SNAIL(x10)^BIRD(y10))=>SMALLER(x10,y10)
(BIRD(x11)^FOX(y11))=>SMALLER(x11,y11)
(FOX(x13)^WOLF(y13))=>SMALLER(x13,y13)
(BIRD(x14)^GRAIN(y14))=>~EATS(x14,y14)
(CATERPILLAR(x15)^GRAIN(y15))=>EATS(x15,y15)
(SNAIL(x16)^PLANT(y16))=>EATS(x16,y16)
(WOLF(x17)^FOX(y17))=>~EATS(x17,y17)
(WOLF(x18)^GRAIN(y18))=>~EATS(x18,y18)
(BIRD(x19)^SNAIL(y19))=>~EATS(x19,y19)
BIRD(Animal1)
GRAIN(Grain1)
CATERPILLAR(Animal2)
```

variables.txt

```
x0
x1
x2
x3
x4
x5
x6
x7
x8
x9
x10
x11
x13
x14
x15
x16
x17
x18
x19
y8
y9
y10
y11
y13
y14
y15
y16
```

```
y17
y18
y19
z8
u8
```

constants.txt

```
WOLF
FOX
BIRD
CATERPILLAR
SNAIL
GRAIN
ANIMAL
EATS
~EATS
SMALLER
PLANT
Animal1
Animal2
Grain1
```

Rezulatat działania programu:

```
Baza wiedzy:
K1: WOLF(x1_1) => ANIMAL(x1_1)
K2: FOX(x2_2) => ANIMAL(x2_2)
K3: BIRD(x3_3) => ANIMAL(x3_3)
K4: CATERPILLAR(x4_4) => ANIMAL(x4_4)
K5: SNAIL(x5_5) => ANIMAL(x5_5)
K6: GRAIN(x6_6) => PLANT(x6_6)
K7: GRAIN(x7_7) => PLANT(x7_7)
K8: ANIMAL(x8_8) ^ PLANT(y8_19) ^ ~EATS(x8_8, y8_19) ^ ANIMAL(z8_30) ^ SMALLER(z8_30,
x8_8) ^ PLANT(u8_31) ^ EATS(z8_30, u8_31) => EATS(x8_8, z8_30)
K9: CATERPILLAR(x9_9) ^ BIRD(y9_20) => SMALLER(x9_9, y9_20)
K10: SNAIL(x10_10) ^ BIRD(y10_21) => SMALLER(x10_10, y10_21)
K11: BIRD(x11_11) ^ FOX(y11_22) => SMALLER(x11_11, y11_22)
K12: FOX(x13_12) ^ WOLF(y13_23) => SMALLER(x13_12, y13_23)
K13: BIRD(x14_13) ^ GRAIN(y14_24) => ~EATS(x14_13, y14_24)
K14: CATERPILLAR(x15_14) ^ GRAIN(y15_25) => EATS(x15_14, y15_25)
K15: SNAIL(x16_15) ^ PLANT(y16_26) => EATS(x16_15, y16_26)
K16: WOLF(x17_16) ^ FOX(y17_27) => ~EATS(x17_16, y17_27)
K17: WOLF(x18_17) ^ GRAIN(y18_28) => ~EATS(x18_17, y18_28)
K18: BIRD(x19_18) ^ SNAIL(y19_29) => ~EATS(x19_18, y19_29)
K19: BIRD(Animal1_11)
K20: GRAIN(Grain1_13)
K21: CATERPILLAR(Animal2_12)
K22: ~ANIMAL(x1_33) => ~WOLF(x1_33)
K27: ~PLANT(x6_38) => ~GRAIN(x6_38)
K29: PLANT(y8_40) ^ ~EATS(x8_40, y8_40) ^ ANIMAL(z8_40) ^ SMALLER(z8_40, x8_40) ^
PLANT(u8_40) ^ EATS(z8_40, u8_40) ^ ~EATS(x8_40, z8_40) => ~ANIMAL(x8_40)
K30: ANIMAL(x8_41) ^ ~EATS(x8_41, y8_41) ^ ANIMAL(z8_41) ^ SMALLER(z8_41, x8_41) ^
PLANT(u8_41) ^ EATS(z8_41, u8_41) ^ ~EATS(x8_41, z8_41) => ~PLANT(y8_41)
K31: ANIMAL(x8_42) ^ PLANT(y8_42) ^ ANIMAL(z8_42) ^ SMALLER(z8_42, x8_42) ^
PLANT(u8_42) ^ EATS(z8_42, u8_42) ^ ~EATS(x8_42, z8_42) => EATS(x8_42, y8_42)
K32: ANIMAL(x8_43) ^ PLANT(y8_43) ^ ~EATS(x8_43, y8_43) ^ SMALLER(z8_43, x8_43) ^
PLANT(u8_43) ^ EATS(z8_43, u8_43) ^ ~EATS(x8_43, z8_43) => ~ANIMAL(z8_43)
K33: ANIMAL(x8_44) ^ PLANT(y8_44) ^ ~EATS(x8_44, y8_44) ^ ANIMAL(z8_44) ^ PLANT(u8_44)
^ EATS(z8_44, u8_44) ^ ~EATS(x8_44, z8_44) => ~SMALLER(z8_44, x8_44)
K34: ANIMAL(x8_45) ^ PLANT(y8_45) ^ ~EATS(x8_45, y8_45) ^ ANIMAL(z8_45) ^
SMALLER(z8_45, x8_45) ^ EATS(z8_45, u8_45) ^ ~EATS(x8_45, z8_45) => ~PLANT(u8_45)
K35: ANIMAL(x8_46) ^ PLANT(y8_46) ^ ~EATS(x8_46, y8_46) ^ ANIMAL(z8_46) ^
SMALLER(z8_46, x8_46) ^ PLANT(u8_46) ^ ~EATS(x8_46, z8_46) => ~EATS(z8_46, u8_46)
```

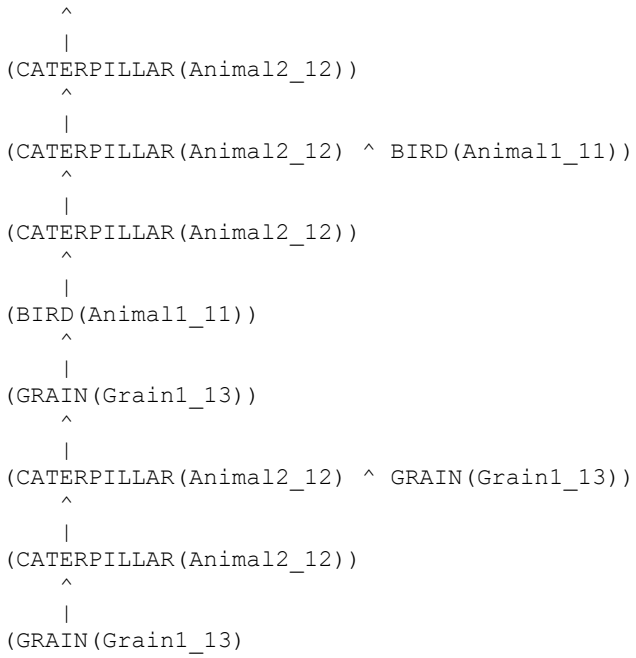
K36: BIRD(y9_47) ^ ~SMALLER(x9_47, y9_47) => ~CATERPILLAR(x9_47)
K37: CATERPILLAR(x9_48) ^ ~SMALLER(x9_48, y9_48) => ~BIRD(y9_48)
K39: SNAIL(x10_50) ^ ~SMALLER(x10_50, y10_50) => ~BIRD(y10_50)
K40: FOX(y11_51) ^ ~SMALLER(x11_51, y11_51) => ~BIRD(x11_51)
K42: WOLF(y13_53) ^ ~SMALLER(x13_53, y13_53) => ~FOX(x13_53)
K44: GRAIN(y14_55) ^ EATS(x14_55, y14_55) => ~BIRD(x14_55)
K45: BIRD(x14_56) ^ EATS(x14_56, y14_56) => ~GRAIN(y14_56)
K46: GRAIN(y15_57) ^ ~EATS(x15_57, y15_57) => ~CATERPILLAR(x15_57)
K47: CATERPILLAR(x15_58) ^ ~EATS(x15_58, y15_58) => ~GRAIN(y15_58)
K49: SNAIL(x16_60) ^ ~EATS(x16_60, y16_60) => ~PLANT(y16_60)
K50: FOX(y17_61) ^ EATS(x17_61, y17_61) => ~WOLF(x17_61)
K51: WOLF(x17_62) ^ EATS(x17_62, y17_62) => ~FOX(y17_62)
K54: SNAIL(y19_65) ^ EATS(x19_65, y19_65) => ~BIRD(x19_65)

Teza: EATS(Animal1_11, Animal2_12)

Zbiór podstawień: {x15_14/Animal2_12, x9_9/Animal2_12, x4_4/Animal2_12, y8_19/x6_6,
u8_31/x7_7, y9_20/Animal1_11, x3_3/Animal1_11, x6_6/Grain1_13, x8_8/Animal1_11,
x14_13/Animal1_11, y15_25/x7_7, y14_24/x6_6, x7_7/Grain1_13, z8_30/Animal2_12}

Użyte klauzule:
K20: GRAIN(Grain1_13)
K21: CATERPILLAR(Animal2_12)
K14: CATERPILLAR(x15_14) ^ GRAIN(y15_25) => EATS(x15_14, y15_25)
K20: GRAIN(Grain1_13)
K7: GRAIN(x7_7) => PLANT(x7_7)
K19: BIRD(Animal1_11)
K21: CATERPILLAR(Animal2_12)
K9: CATERPILLAR(x9_9) ^ BIRD(y9_20) => SMALLER(x9_9, y9_20)
K21: CATERPILLAR(Animal2_12)
K4: CATERPILLAR(x4_4) => ANIMAL(x4_4)
K20: GRAIN(Grain1_13)
K19: BIRD(Animal1_11)
K13: BIRD(x14_13) ^ GRAIN(y14_24) => ~EATS(x14_13, y14_24)
K20: GRAIN(Grain1_13)
K6: GRAIN(x6_6) => PLANT(x6_6)
K19: BIRD(Animal1_11)
K3: BIRD(x3_3) => ANIMAL(x3_3)
K8: ANIMAL(x8_8) ^ PLANT(y8_19) ^ ~EATS(x8_8, y8_19) ^ ANIMAL(z8_30) ^ SMALLER(z8_30,
x8_8) ^ PLANT(u8_31) ^ EATS(z8_30, u8_31) => EATS(x8_8, z8_30)

Graf wnioskowania:
(EATS(Animal1_11, Animal2_12))
    ^
    |
(ANIMAL(Animal1_11) ^ PLANT(Grain1_13) ^ ~EATS(Animal1_11, Grain1_13) ^
ANIMAL(Animal2_12) ^ SMALLER(Animal2_12, Animal1_11) ^ PLANT(Grain1_13) ^
EATS(Animal2_12, Grain1_13))
    ^
    |
(BIRD(Animal1_11))
    ^
    |
(GRAIN(Grain1_13))
    ^
    |
(BIRD(Animal1_11) ^ GRAIN(Grain1_13))
    ^
    |
(BIRD(Animal1_11))
    ^
    |
(GRAIN(Grain1_13))


(CATERPILLAR(Animal2_12))
(CATERPILLAR(Animal2_12) ^ BIRD(Animal1_11))
(CATERPILLAR(Animal2_12))
(BIRD(Animal1_11))
(GRAIN(Grain1_13))
(CATERPILLAR(Animal2_12) ^ GRAIN(Grain1_13))
(CATERPILLAR(Animal2_12))
(GRAIN(Grain1_13)