Konstrukcja systemu, który opierając się na pytaniach w języku naturalnym i bazie relacji (implementacja w języku Prolog) potrafi odpowiadać, wyciągając logiczne wnioski. Konstrukcja bazy relacji.

Jakub Usyk, Amadeusz Waląg, Antoni Zub.

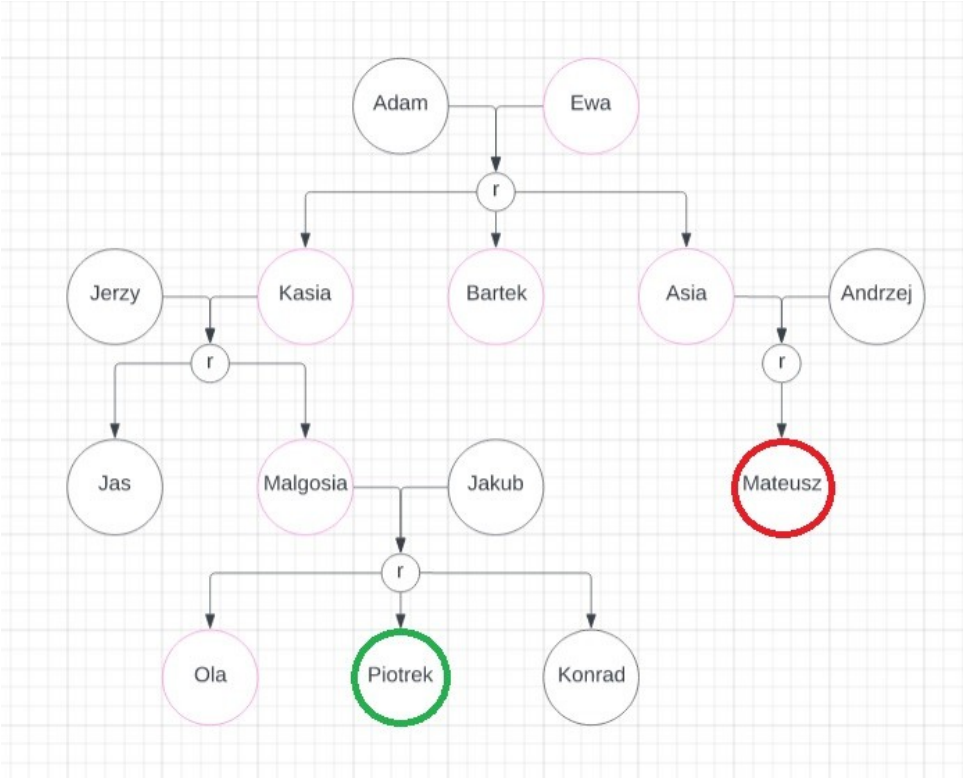
Wstęp

Prawdopodobnie każdy choć raz spotkał się z sytuacją, w której na spotkaniu w gronie rodzinnym, jeden z członków rodziny zaczyna opowiadać historię w stylu:

“Syn Asi, siostry babci Kasi od strony Twojej matki, co jest żoną tego Andrzeja..., poszedł na takie same studia co ty.

Aby lepiej zobrazować powyższy problem, stworzyliśmy graf.

Zakładamy, że jesteśmy Piotrkiem (oznaczonym kolorem zielonym), a Mateusz (oznaczony kolorem czerwonym) jest głównym bohaterem opowiadania członka rodziny.



Opis projektu

Projekt powstał w celu rozwiązania problemu ukazanego we wstępie niniejszego raportu. Opracowana została baza relacji w rodzinie. Wprowadzone zostały cztery relacje:

Dwie dzielące ze względu na płeć osoby – man lub woman:

Pozostałe dzielące ze względu na relację rodzicielską - father lub mother:

```
def male(x="X"): #male(X)
    return "(male({0}))".format(x)

def female(x="X"): #female(X)
    return "(female({0}))".format(x)

def father(x="X", y="Y"):  #father(X, Y)
    return "(father({0}, {1}))".format(x, y)

def mother(x="X", y="Y"):  #mother(X, Y)
    return "(mother({0}, {1}))".format(x, y)
```

Zdefiniowane relacje przyporządkowane zostały członkom rodziny przedstawionej na diagramie [1]:

Relacja podziału ze względu na płeć	Relacja podziału ze względu na typ sprawowanego rodzicielstwa
prolog.assertz("male(adam)") prolog.assertz("female(ewa)") prolog.assertz("male(jerzy)") prolog.assertz("female(kasia)") prolog.assertz("male(bartek)") prolog.assertz("female(asia)") prolog.assertz("male(andrzej)") prolog.assertz("male(jas)") prolog.assertz("female(malgosia)") prolog.assertz("male(jakub)") prolog.assertz("male(mateusz)") prolog.assertz("female(ola)") prolog.assertz("male(piotrek)") prolog.assertz("male(konrad)")	prolog.assertz("father(adam, kasia)") prolog.assertz("father(adam, bartek)") prolog.assertz("father(adam, asia)") prolog.assertz("mother(ewa, kasia)") prolog.assertz("mother(ewa, bartek)") prolog.assertz("mother(ewa, asia)") prolog.assertz("father(jerzy, jas)") prolog.assertz("father(jerzy, malgosia)") prolog.assertz("mother(kasia, jas)") prolog.assertz("mother(kasia, malgosia)") prolog.assertz("father(andrzej, mateusz)") prolog.assertz("mother(asia, mateusz)") prolog.assertz("father(jakub, ola)") prolog.assertz("father(jakub, piotrek)") prolog.assertz("father(jakub, konrad)") prolog.assertz("mother(malgosia, ola)") prolog.assertz("mother(malgosia, piotrek)") prolog.assertz("mother(malgosia, konrad)")

Następnie zostały zdefiniowane następujące zapytania, które odpowiadają zapytaniu kto jest kim dla kogo. Przykłady takich zapytań występujące w kodzie:

```
def daughter(x="X", y="Y"):  #daughter(X, Y)
    return "(" + female(x) + "," + mother(y, x) + ";" + female(x) + "," + father(y, x) + ")"

def brother(x="X", y="Y"):  #brother(X, Y)
    return "(" + mother("B", x) + "," + mother("B", y) + "," + male(x) + "," + negate(x, y) + ";" + father("B", x) + "," + father("B", y) +
"," + male(x) + "," + negate(x, y) + ")"

def sister(x="X", y="Y"):  #sister(X, Y)
    return "(" + mother("S", x) + "," + mother("S", y) + "," + female(x) + "," + negate(x, y) + ";" + father("S", x) + "," + father("S", y)
+ "," + female(x) + "," + negate(x, y) + ")"

def grandfather(x="X", y="Y"):  #sister(X, Y)
    return "(" + mother("Z", y) + "," + daughter("Z", x) + "," + male(x) + ";" + father("Z", y) + "," + son("Z", x) + "," + male(x) + ")"

def grandmother(x="X", y="Y"):  #sister(X, Y)
    return "(" + mother("Z", y) + "," + daughter("Z", x) + "," + female(x) + ";" + father("Z", y) + "," + son("Z", x) + "," + female(x) +
")"
```

Zapytanie te są wykonywane dla podanego argumentu – imienia osoby, w zdefiniowanych funkcjach, takich jak:

```
def queryForDaughter(parentName): #+
    return prolog.query(daughter("X", parentName))

def queryForSon(parentName): #+
    return prolog.query(son("X", parentName))

def queryForBrother(name): #+
    return prolog.query(brother("X", name))

def queryForSister(name): #+
    return prolog.query(sister("X", name))

def queryForGrandfather(grandChild): #+
    return prolog.query(grandfather("X", grandChild))

def queryForGrandmother(grandChild): #+
    return prolog.query(grandmother("X", grandChild))
```

W celu umożliwienia zadawania użytkownikowi programu pytań w sposób dla niego zrozumiały, stworzony został bot. Bot ten odpowiada na pytania zbliżone do schematu:

Who is [name] [father/mother/daughter/son/brother/sister/grandfather/grandmother/uncle/aunt/cousin]

W tym celu, pobiera imię z zapytania i wywołuje wcześniej zdefiniowaną metodę podając jej jako argument pobrane imię. Następnie przetwarza otrzymany wynik i wyświetla go użytkownikowi.

W przypadku, gdy bot nie jest w stanie dopasować odpowiedniego wywołanie do zadanego pytanie, wyświetla użytkownikowi wiadomość o treści: "[FamilyBot]: Sorry, I don't understand".

Jeśli bot stwierdzi, że osoba o którą zostanie zapytany nie posiada danego członka rodziny, odpowie w następujący sposób: "[FamilyBot]: X has no such family members".

Podsumowanie

Projekt został w całości wykonany i przetestowany ze skutkiem pozytywnym. Udało się również odpowiedzieć na pytanie o kim mowa we wstępie, pytając bota w następujący sposób:

```
[You]: who is asia son
[FamilyBot]: Mateusz is asia son
```

Bibliografia

Projekt został stworzony przy pomocy źródeł zapewnionych przez prowadzącego. Były to:

Prolog: https://www.openbookproject.net/py4fun/prolog/prolog1.html

PySwip: https://github.com/yuce/pyswip