

# Vietnam National University - HCM City

## University of Science



### Project 02:

### First-order logic

*Course: CSC14003 - Artificial Intelligence*

Class: 21CLC03

Student: 21127076 – Doãn Anh Khoa  
21127240 – Nguyễn Phát Đạt  
21127322 – Hoàng Xuân Khôi  
21127388 – Tăng Đức Phong

Lecturer: Nguyễn Ngọc Thảo  
Nguyễn Trần Duy Minh  
Lê Ngọc Thành  
Nguyễn Hải Đăng

TPHCM, April 2023

## Table of Contents

I.	Information .....	2
II.	Assignment plan .....	2
III.	Self-assessment .....	2
IV.	Works .....	3
	Working with the Prolog tool: .....	3
	Implement Prolog on <b>SWI-Prolog</b> .....	4
	Five illustrative examples:.....	7
1.	British Royal family problem .....	9
	Defined predicate based on predefined predicates in the knowledge base: .....	9
	Test cases for British royal family problem: .....	12
2.	Problem about the relationship between the characters in the Detective Conan manga.....	13
	Test cases for Detective Conan problem: .....	16
3.	Implement logic deductive system with Python programming language.....	18
3.1.	Overview .....	18
3.2.	Defined syntax and some important notes .....	18
3.3.	Code structures .....	19
3.4.	Algorithms and explanation .....	21
3.5.	Result of each query for each knowledge base: .....	23
3.6.	Some comments on the results .....	26
3.7.	Conclusions .....	26
V.	File submission .....	26
VI.	References .....	27

## I. Information

**Class:** 21CLC03

ID	Name
21127076	Doãn Anh Khoa
21127240	Nguyễn Phát Đạt
21127322	Hoàng Xuân Khôi
21127388	Tăng Đức Phong

## II. Assignment plan

Work		Student
<b>1.1</b>	Write report about prolog and its main features, how to implement prolog language, British royal family problem and Detective Conan problem.	21127388
<b>1.2</b>	British royal family problem (predicates and questions)	21127076 - 21127322
<b>1.2</b>	Detective Conan problem (predicates and questions)	
<b>1.3</b>	Implement logic deductive system and write report for it	21127240

## III. Self-assessment

**Completion:** 98.5%

Requirement	Completion
British royal family problem (predicates and questions)	100%
Detective Conan problem (predicates and questions)	100%
Logic deductive system in Python	95% (may produce unexpected results)

## IV. Works

### Working with the Prolog tool:

Prolog (**PRO**gramming in **LOG**ics) is a logical and declarative programming language. It plays an important role in artificial intelligence and computational linguistics. Prolog is intended primarily as a declarative programming language. In prolog, logic is expressed as relations (represented as Facts and Rules). Formulation or Computation is carried out by running a query over these relations.

Some key features of prolog and examples:

+ **Facts and rule-based system:** Prolog uses fact and rule-based system to store and reason without knowledge. Facts are statements about the world, while rules are used to deduce new information from existing facts using logical inferences.

```
male(john).
female(jane)
parent(john, mary).
parent(jane, mary).
```

In this example, we have four facts that define the relationship between some individuals. We have a male John, a female Jane, and then we define that John and Jane are the parents of Mary.

+ **Unification:** The fundamental idea is whether can the given terms be made to represent the same structure.

```
father(A, B) :- parent(A, B), male(A).
```

In this example, we define a rule which states that A is the father of B if A is a parent of B and A is male. Here, unification is used to match the variables A and B to the appropriate facts in the knowledge base.

+ **Backtracking:** When a task fails, prolog traces backward and tries to satisfy previous task with a different choice.

```
male(john).
male(ethan).

list_male :- male(X), write(X), nl, fail.
```

In this example, the fail predicates causes the prolog to backtrack and search for alternative solutions which will result in a list of male.

+ **Recursion:** Prolog supports recursive programming, which means functions or predicates can call themselves.

factorial(0, 1).

factorial(N, X) :- N > 0, N1 is N - 1, factorial(N1, X1), X is N \* X1

Here, we've defined a predicate "factorial" that calculates the factorial of a number recursively using the mathematical formula  $n! = n * (n - 1)!$

=> Overall, Prolog is designed to work with first-order logic and allows developers to write complex logic systems using very few lines of code.

## Implement Prolog on SWI-Prolog

The SWI-Prolog tool is an open-source implementation of the Prolog language that provides a variety of features, including a compiler, an interpreter, and a development environment.

First, we need to go to download page ([swi-prolog](https://www.swi-prolog.org/download/stable)) and choose the appropriate platform/version to download the installer.

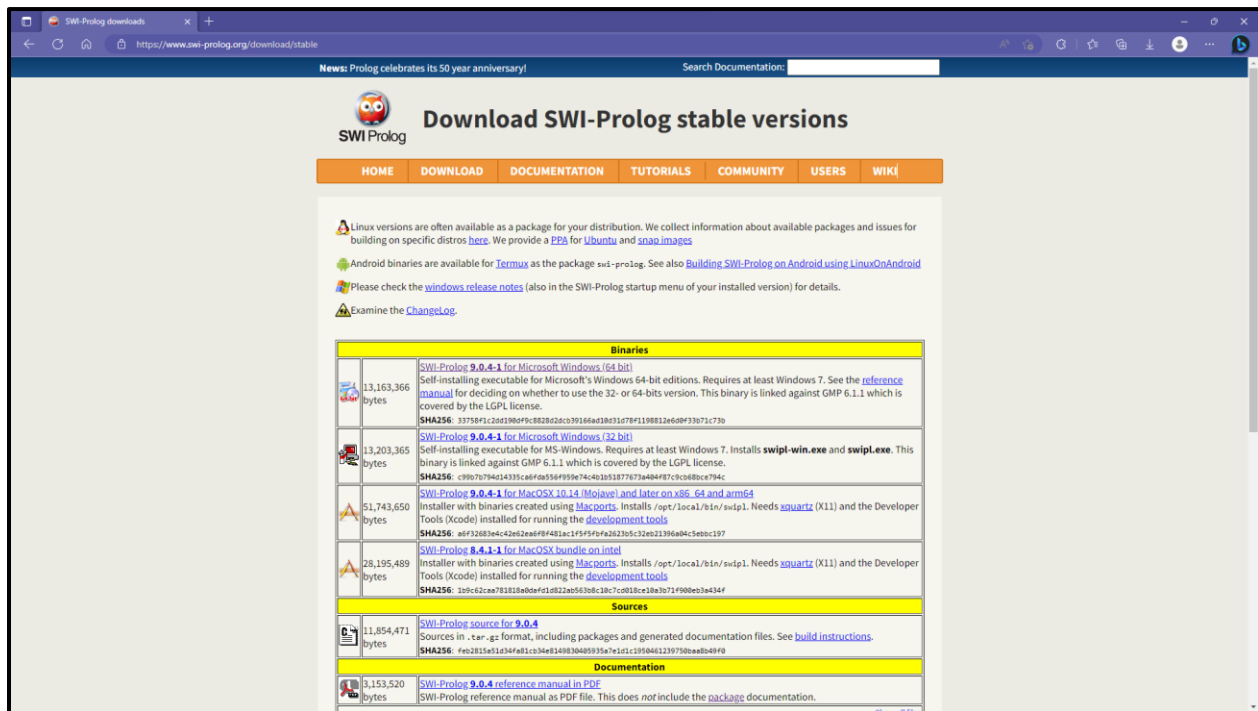


Figure 1: SWI-Prolog download page

After downloading, we run the installer and follow the instructions in the installer wizard to complete the installation process.

Once the installation is complete, you can launch SWI-Prolog by running the application shortcut 'swipl' on the desktop or by running 'swipl.exe' from the installation folder (\bin).



Figure 2: Shortcut icon for 'swipl'

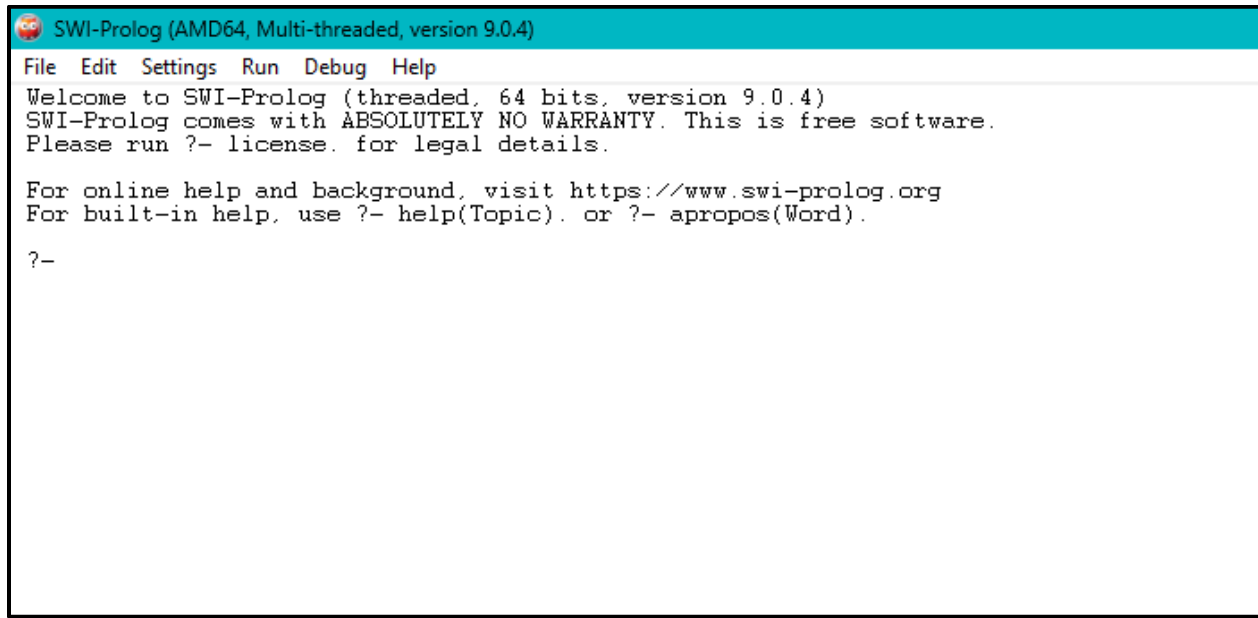


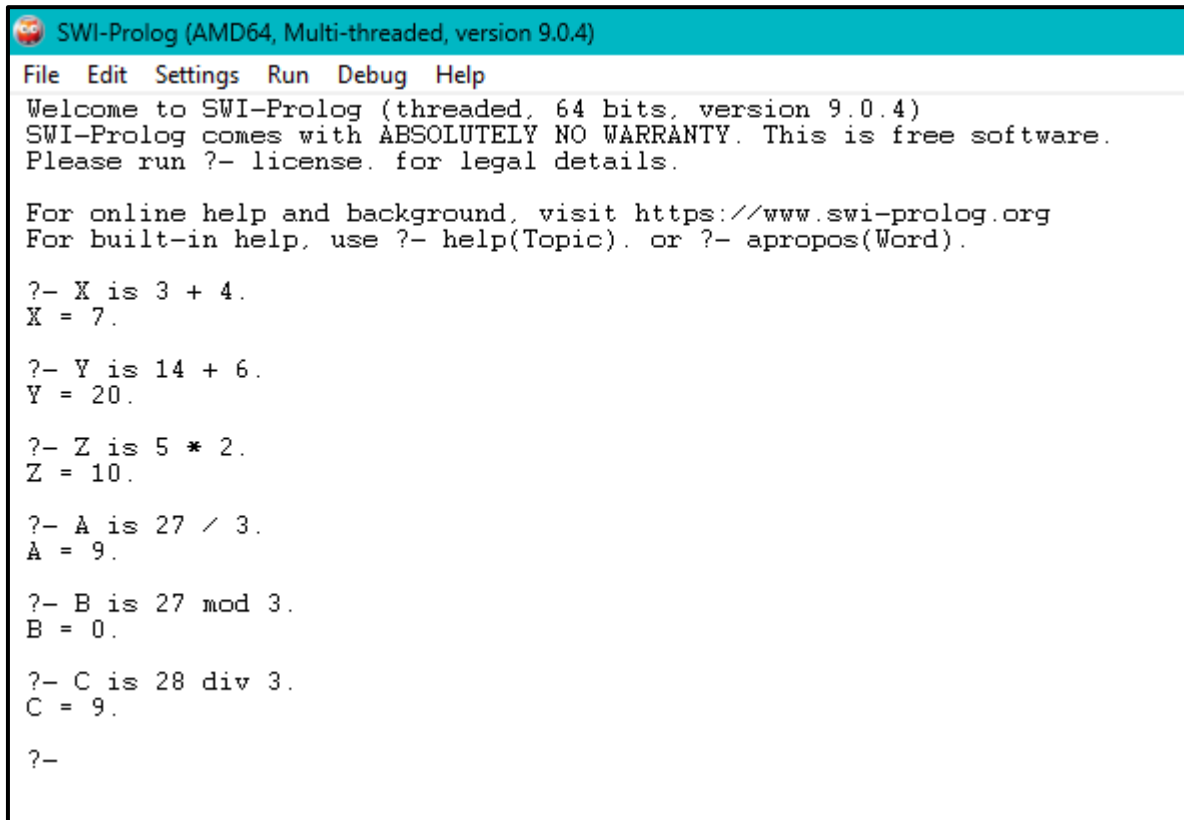
Figure 3: swi-prolog program startup console

SWI-Prolog syntax is similar to other Prolog dialects, with some variations and additional features. Here are some basic syntax rules in SWI-Prolog:

- Atoms are sequences of letters, digits and underscore characters starting with a lowercase letter or a single quote. For example, 'hello', 'Good Morning', 'pi'.
- Variables are denoted by a capital letter, or an underscore followed by a combination of letters, digits, and underscores. For example, 'X', 'Y\_1', 'Z'.
- Predicates are written as a sequence of atoms or variables separated by commas and enclosed in parentheses. For example, 'father(john, peter)', 'animal(X, dog)'.
- Clauses have the form 'Head :- Body', where Head is a predicate and Body is a conjunction of literals (atomic formulas and negated atoms). For example, 'siblings(X, Y) :- parent(Z, X), parent(Z, Y)'.

- Query is the action of asking the program about the information which is available within its database. When a Prolog program is loaded, we will get the query prompt '?-'. For example, 'male(John).' is our database then we perform a query '?-male(John)', and the program will return the result 'yes'.
- Predicate calculus symbol  $\rightarrow$ (if),  $\neg$ (not),  $\vee$ (or),  $\wedge$ (and) in Prolog are ':-', 'not()', ';;', ';;' respectively. For example, 'father(X, Y) :- not(female(X)), parent(X, Y)'
- A cut (!) inside a rule/clause will prevent Prolog from backtracking any predicates behind the cut which can be used to improve the efficiency of Prolog programs by limiting the number of solutions that are generated or reducing the search space. However, it can lead to incorrect or incomplete solutions. Example use of cut, 'predicate(X) :- one(X), !, two(X).'
- Anonymous variables (\_) can be used to ignore a certain variable or to match any term of a certain argument position as they never bound to a value and can be used multiple times in a predicate. For example, 'parent(\_, john).' the first argument is an anonymous variable indicating that we don't care who is the parent of john is.
- Negation is denoted by the prefix operator '\+' and 'not()'. For example, 'female(X)', 'not(female(X))'.
- Lists are written as comma-separated sequences of elements enclosed in square brackets. For example, '[1, 2, 3]', '[hello, world]', '[Head|Tail]', '[A\_]'
- Comment symbol (%) is used to add comments within the code, which are ignored during program execution. For example, '% this a comment in SWI-Prolog'.
- Arithmetic expressions use infix notation and standard operators such as '+', '-', '\*', '/', 'mod', and 'div'. For example, 'add(A, B, C) :- C is A + B'.
- There are predefined control structures like 'if-then-else', 'while', 'for', and 'repeat'. For example, 'greater\_than\_twenty(X) :- (X > 20 -> write('X is greater than 20')); write('X is less than or equal to 20'))'.
- SWI-Prolog supports other features such as modules, attributed variables, and meta-programming using the 'call' and 'apply' predicates, among others.

Five illustrative examples:



```

SWI-Prolog (AMD64, Multi-threaded, version 9.0.4)
File Edit Settings Run Debug Help
Welcome to SWI-Prolog (threaded, 64 bits, version 9.0.4)
SWI-Prolog comes with ABSOLUTELY NO WARRANTY. This is free software.
Please run ?- license. for legal details.

For online help and background, visit https://www.swi-prolog.org
For built-in help, use ?- help(Topic). or ?- apropos(Word).

?- X is 3 + 4.
X = 7.

?- Y is 14 + 6.
Y = 20.

?- Z is 5 * 2.
Z = 10.

?- A is 27 / 3.
A = 9.

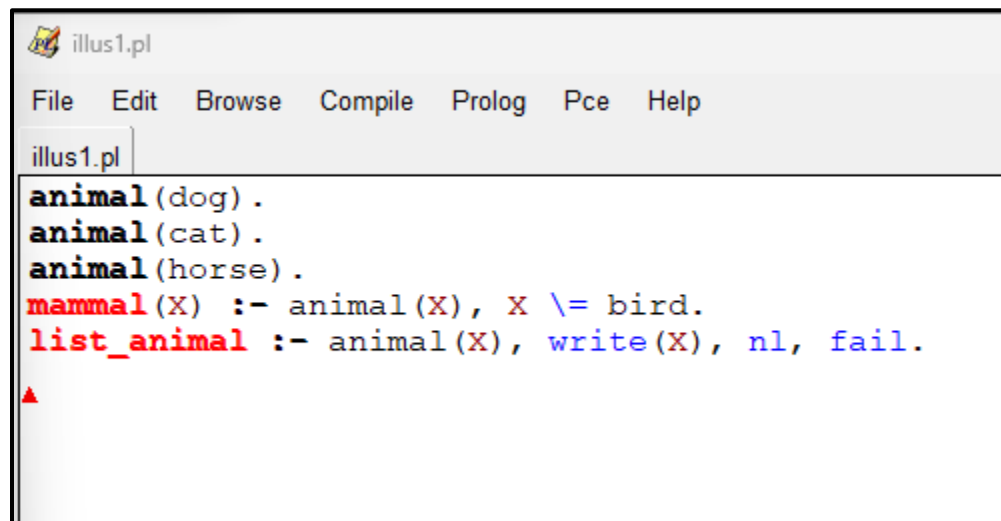
?- B is 27 mod 3.
B = 0.

?- C is 28 div 3.
C = 9.

?-

```

Figure 4: Simple arithmetic calculation.



```

illus1.pl
File Edit Browse Compile Prolog Pce Help
illus1.pl
animal(dog) .
animal(cat) .
animal(horse) .
mammal(X) :- animal(X), X \= bird.
list_animal :- animal(X), write(X), nl, fail.
▲

```

Figure 5: Defining facts and rules.



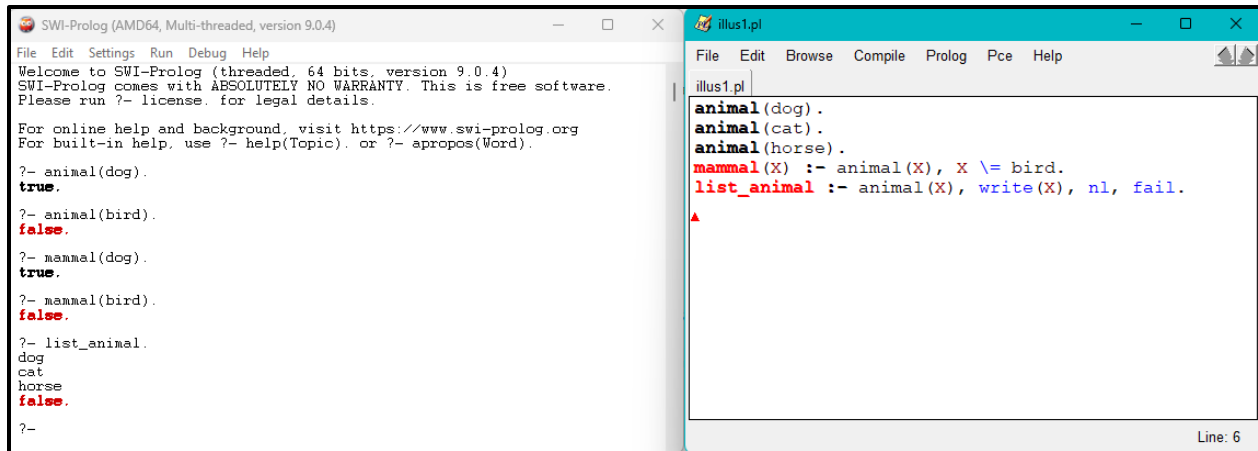


Figure 6: Querying facts and rules.

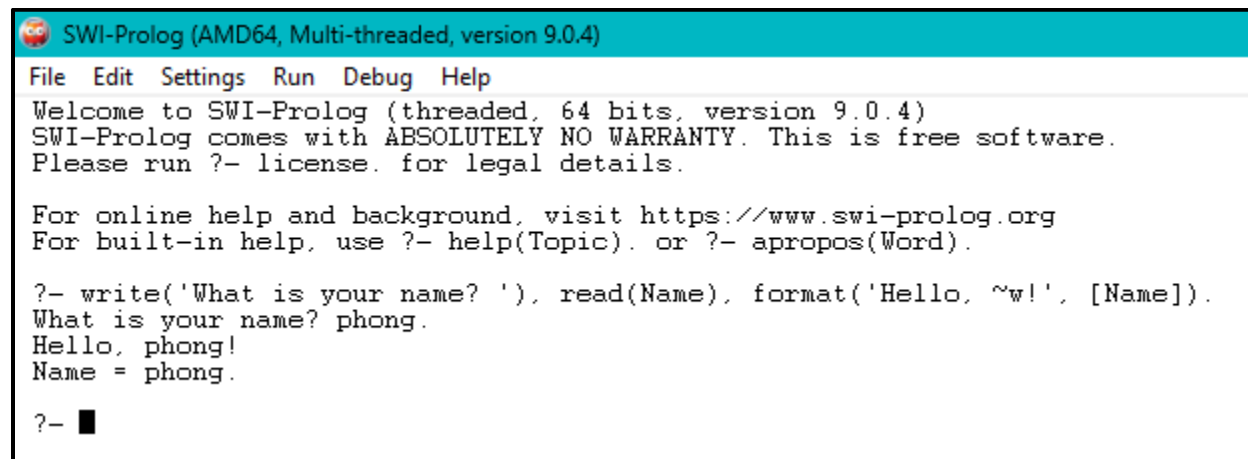


Figure 7: User input.

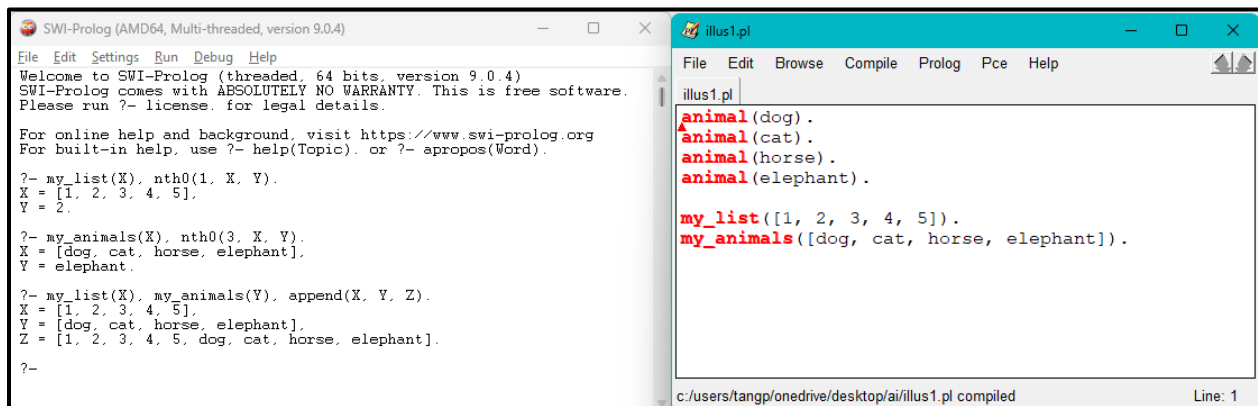


Figure 8: List manipulation.

## 1. British Royal family problem

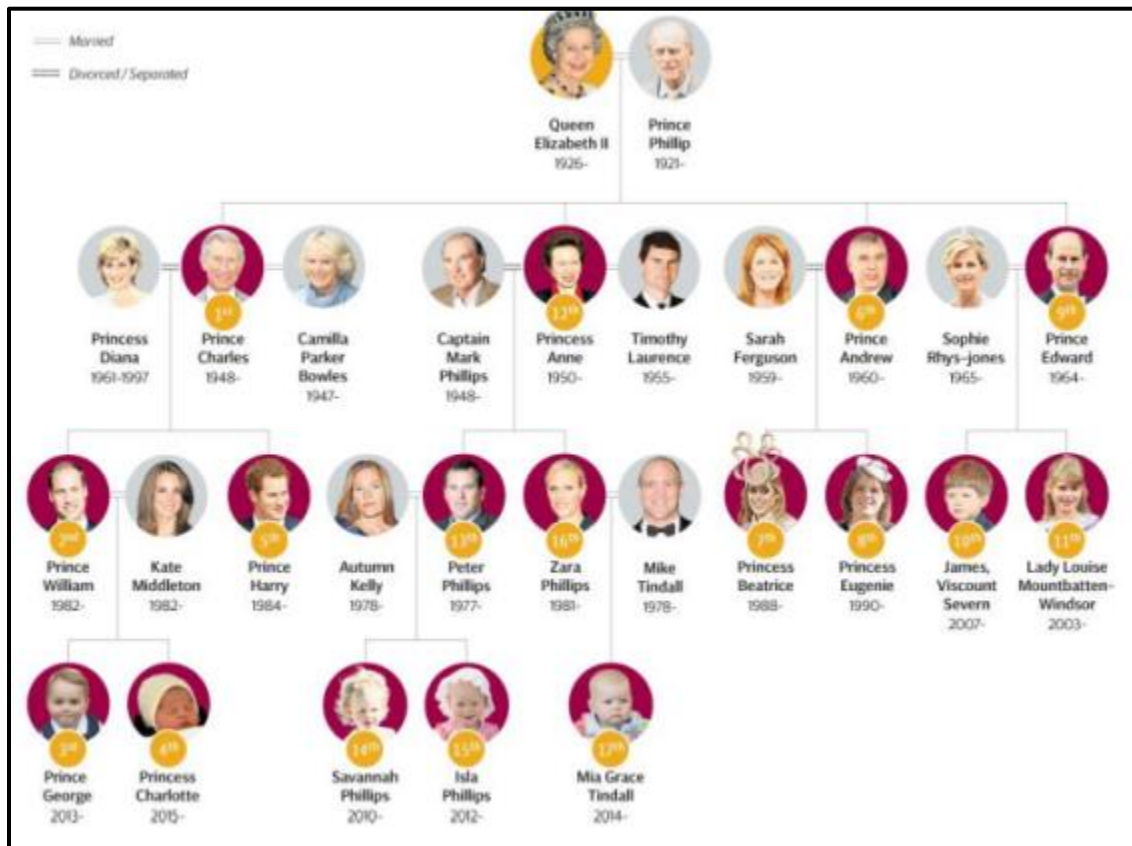


Figure 9: British Royal family tree

Defined predicate based on predefine predicates in the knowledge base:

**Facts of the Knowledges base based on the following predicates:**

- parent(Parent,Child)
- male(Person)
- married(Person, Person)
- female(Person)
- divorced(Person, Person)

**Some rules define from the above predicates:**

*Person is Wife's husband if Person is male, and Wife is female, and Person and Wife are married.*

- husband(Person, Wife) :- male(Person), female(Wife), married(Person, Wife).

*Person is Husband's wife if Husband is male, and Person is female, and Husband and Person are married.*

- wife(Person, Husband) :- male(Husband), female(Person), married(Person, Husband).

*Parent is the father of Child if Parent is male, and Parent is the parent of Child.*

- father(Parent, Child) :- male(Parent), parent(Parent, Child).

*Parent is the mother of Child if Parent is female, and Parent is the parent of Child.*

- mother(Parent, Child) :- female(Parent), parent(Parent, Child).

*Child is the child of Parent if Parent is the parent of Child.*

- child(Child, Parent) :- parent(Parent, Child).

*Child is Parent's son if Child is Parent's child and Child is male.*

- son(Child, Parent) :- child(Child, Parent), male(Child).

*Child is Parent's daughter if Child is Parent's child and Child is female.*

- daughter(Child, Parent) :- child(Child, Parent), female(Child).

*GP is grandparent of GC if GP is the parent of P, and P is the parent of GC.*

- grandparent(GP, GC) :- parent(GP, P), parent(P, GC).

*GF is grandfather of GC if GF is male, and GF is grandparent of GC.*

- grandfather(GF, GC) :- male(GF), grandparent(GF, GC).

*GM is grandmother of GC if GM is female, and GF is grandparent of GC.*

- grandmother(GM, GC) :- female(GM), grandparent(GM, GC).

*GC is grandchild of GP if GP is grandparent of GC.*

- grandchild(GC, GP) :- grandparent(GP, GC).

*GS is grandson of GP if GS is male, and GS is grandchild of GP.*

- grandson(GS, GP) :- male(GS), grandchild(GS, GP).

*GD is granddaughter of GP if GD is female, and GD is grandchild of GP.*

- granddaughter(GD, GP) :- female(GD), grandchild(GD, GP).

*Person1 and Person2 are siblings if Person1's parent is P, and Person2's parent is P, and Person1 and Person2 are not the same person.*

- sibling(Person1, Person2) :- parent(P, Person1), parent(P, Person2), Person1 \= Person2.

*Person is Sibling's brother if Person is male, and Person and Sibling are siblings.*

- brother(Person, Sibling) :- male(Person), sibling(Person, Sibling).

*Person is Sibling's sister if Person is female, and Person and Sibling are siblings.*

- sister(Person, Sibling) :- female(Person), sibling(Person, Sibling).

*Person is NieceNephew's aunt if Person is Sibling's sister, and Sibling is the parent of NieceNephew.*

- aunt(Person, NieceNephew) :- sister(Person, Sibling), parent(Sibling, NieceNephew).

*Person is NieceNephew's aunt if Parent is the mother of NieceNephew, and Parent and Husband are siblings, and Husband and Person are married, and Person is not Parent's sister.*

- aunt(Person, NieceNephew) :- mother(Parent, NieceNephew), sibling(Parent, Husband), married(Husband, Person), not(sister(Person, Parent))

*Person is NieceNephew's uncle if Person is Sibling's brother, and Sibling is the parent of NieceNephew.*

- uncle(Person, NieceNephew) :- brother(Person, Sibling), parent(Sibling, NieceNephew).

*Person is NieceNephew's uncle if Parent is the father of NieceNephew, and Parent and Wife are siblings, and Person and Wife are married, and Person is not Parent's brother.*

- uncle(Person, NieceNephew) :- father(Parent, NieceNephew), sibling(Parent, Wife), married(Person, Wife), not(brother(Person, Parent)).

*Person is AuntUncle's niece if AuntUncle is Person's uncle, and Person is female.*

- niece(Person, AuntUncle) :- uncle(AuntUncle, Person), female(Person).

*Person is AuntUncle's niece if AuntUncle is Person's aunt, and Person is female.*

- niece(Person, AuntUncle) :- aunt(AuntUncle, Person), female(Person).

*Person is AuntUncle's nephew if AuntUncle is Person's uncle, and Person is male.*

- nephew(Person, AuntUncle) :- uncle(AuntUncle, Person), male(Person).

*Person is AuntUncle's nephew if AuntUncle is Person's aunt, and Person is male.*

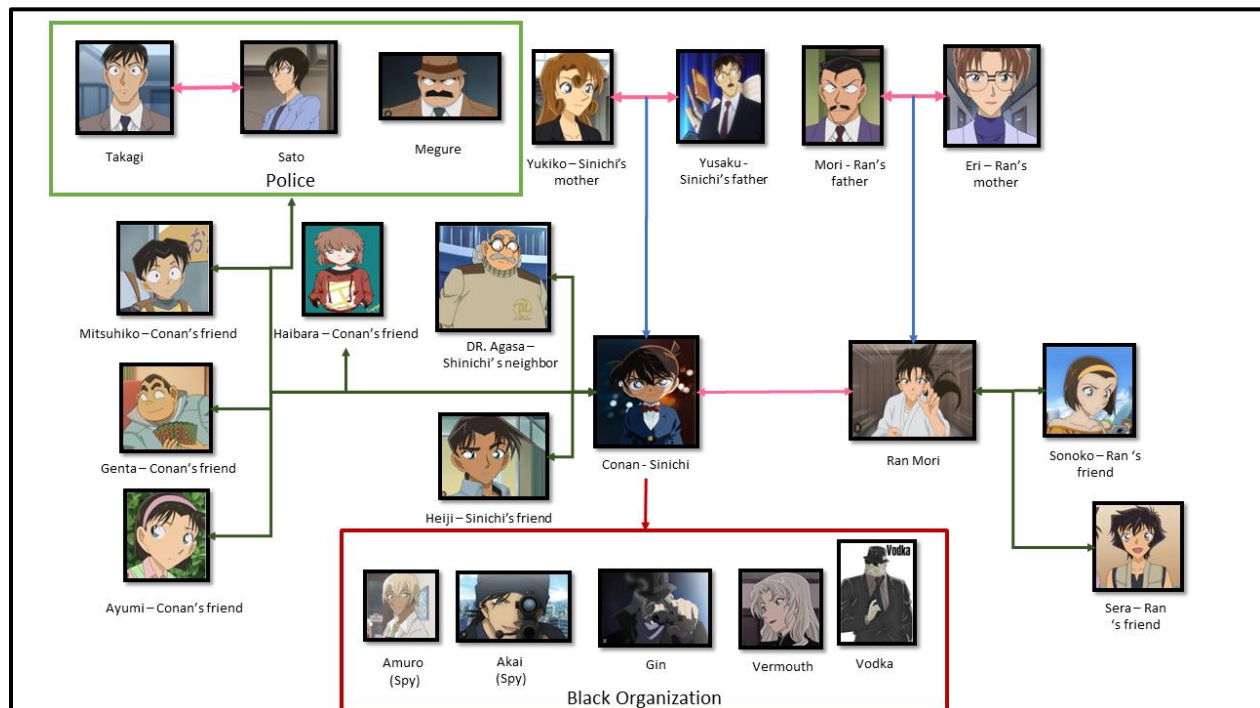
- nephew(Person, AuntUncle) :- aunt(AuntUncle, Person), male(Person).

Test cases for British royal family problem:

No.	Question	Prolog code	Answer
1.	Who is Andrew's mother?	mother(X, andrew).	X = elizabeth.
2.	Was Elizabeth the wife of Mia Grace Tindall	divorced(elizabeth, mia).	false.
3.	Is Price Harry the son of Prince Charles?	son(harry, charles).	true.
4.	Is Isla Phillip son of Autumn Kelly?	son(isla, autumn).	false.
5.	Is Mia Grace Tindall the grandchild of Prince Phillip?	grandchild(mia, phillip).	false.
6.	Are Price Harry and Peter Phillip siblings?	sibling(harry, peter).	false.
7.	Are Diana and Charles divorced?	divorced(diana, charles).	true.
8.	Is Charlotte married?	married(charlotte, X).	false.
9.	List the couples that married.	married(X, Y).	(elizabeth, phillip), (charles, camilla), (anne, timothy), (edward, sophie), (william, kate), (peter, autumn), (zara, mike).
10.	List the couples that divorced.	divorced(X, Y).	(dianam, charles), (mark, anne), (andrew, sarah).
11.	List the couples that married and have children.	married(X, Y), child(Z, X), child(Z, Y).	(elizabeth, phillip), (edward, sophie), (william, kate), (peter, autumn), (zara, mike).
12.	List the couples that are divorced but have children.	divorced(X, Y), child(Z, X), child(Z, Y).	(diana, charles), (mark, anne), (andrew, sarah).
13.	List pair of siblings.	sibling(X, Y).	(charles, anne, andrew, edward),

			(william, harry), (peter, zara), (beatrice, eugenie), (james, louise), (george, charlotte), (savannah, isla)
14.	Is Autumn Mia's aunt?	aunt(autumn, mia).	true.
15.	Is Peter William's uncle?	uncle(peter, william).	false.
16.	Is Mike Diana's niece?	niece(mike, diana).	false.
17.	List all of Peter's nieces.	niece(X, peter).	mia.
18.	List all of Zara's granddaughter.	granddaughter(X, zara).	false.
19.	Who are Elizabeth's grandchild and also Sarah's child?	grandchild(X, elizabeth), child(X, sarah).	beatrice, augenie.
20.	List all children of the divorced couple.	child(X, Y), divorced(Y, Z).	william, harry, peter, zara, beatrice, eugenie.
21.	List all Elizabeth's grandchildren that married.	grandchild(X, elizabeth), married(X, Y).	peter, zara, william.

## 2. Problem about the relationship between the characters in the Detective Conan manga.



**Facts of the Knowledges base based on the following predicates:**

- male(Person).
- female(Person).
- child(Person).
- friend(Person1, Person2).
- love(Person1, Person2).
- parent(Person1, Person2).
- police(Person).
- blackOrganization(Person).
- detective(Person).

**Some rules define from the above predicates:**

*X is a spy if X is a police officer, and X is in black organization.*

- spy(X) :- police(X), blackOrganization(X).

*X coop with Y if X is a police officer, and Y is a detective.*

- coop(X, Y) :- police(X), detective(Y).

*X coop with Y if Y is a police officer, and X is a detective.*

- coop(X, Y) :- police(Y), detective(X).

*X coop with Y if X and Y are friends, and X is not a child, and X and Y are not the same person.*

- coop(X, Y) :- friend(X, Y), not(child(X)), X \= Y.

*X coop with Y if X and Y are friends, and Y is not a child, and X and Y are not the same person.*

- coop(X, Y) :- friend(X, Y), not(child(Y)), X \= Y.

*X coop with Y if Y is the parent of X.*

- coop(X, Y) :- parent(Y, X).

*X coop with Y if X is the parent of Y.*

- coop(X, Y) :- parent(X, Y).



*X coop with Z if Z is the parent of Y, and X and Y are friends.*

- $\text{coop}(X, Z) :- \text{parent}(Z, Y), \text{friend}(X, Y).$

*X coop with Z if X is the parent of Y, and Z and Y are friends.*

- $\text{coop}(X, Z) :- \text{parent}(X, Y), \text{friend}(Z, Y).$

*X coop with Y if X is a detective, and Y is a detective, and X and Y are not the same person.*

- $\text{coop}(X, Y) :- \text{detective}(X), \text{detective}(Y), X \neq Y.$

*X coop with Y if X is in black organization, and Y is in black organization, and X and Y are not the same person.*

- $\text{coop}(X, Y) :- \text{blackOrganization}(X), \text{blackOrganization}(Y), X \neq Y.$

*X confronts Y if X is a police officer, and Y is in black organization.*

- $\text{confront}(X, Y) :- \text{police}(X), \text{blackOrganization}(Y), X \neq Y.$

*X confronts Y if Y is a police officer, and X is in black organization.*

- $\text{confront}(X, Y) :- \text{police}(Y), \text{blackOrganization}(X), X \neq Y.$

*X confronts Y if X is a detective, and Y is in black organization.*

- $\text{confront}(X, Y) :- \text{detective}(X), \text{blackOrganization}(Y).$

*X confronts Y if Y is a detective, and X is in black organization.*

- $\text{confront}(X, Y) :- \text{detective}(Y), \text{blackOrganization}(X).$

*X confronts Y if X is a spy, and Y is in black organization.*

- $\text{confront}(X, Y) :- \text{spy}(X), \text{blackOrganization}(Y), X \neq Y.$

*X confronts Y if Y is a spy, and X is in black organization.*

- $\text{confront}(X, Y) :- \text{spy}(Y), \text{blackOrganization}(X), X \neq Y.$

*X protects Y from Z if X is police officer, and Y is female, and Y is not in black organization, and X and Y are not the same person, and Z is in black organization.*

- $\text{protect}(X, Y, Z) :- \text{police}(X), \text{female}(Y), \text{not}(\text{blackOrganization}(Y)), X \neq Y, \text{blackOrganization}(Z).$

*X protects Y from Z if X is a detective, and Y is female, and Y is not in black organization, and X and Y are not the same person, and Z is in black organization.*



- `protect(X, Y, Z) :- detective(X), female(Y), not(blackOrganization(Y)), X \= Y, blackOrganization(Z).`

*X protects Y from Z if X is police officer, and Y is a child, and Z is in black organization.*

- `protect(X, Y, Z) :- police(X), child(Y), blackOrganization(Z).`

*X protects Y from Z if X is a detective, and Y is a child, and Z is in black organization.*

- `protect(X, Y, Z) :- detective(X), child(Y), blackOrganization(Z).`

*X protects Y if X is in black organization, and Y is in black organization, and X and Y are not the same person.*

- `protect(X, Y) :- blackOrganization(X), blackOrganization(Y), X \= Y.`

Test cases for Detective Conan problem:

No.	Question	Prolog code	Answer
1.	Is Conan a man?	<code>male(conan).</code>	true.
2.	Is Ayumi a child?	<code>child(ayumi).</code>	true.
3.	Who is Conan's girlfriend?	<code>love(conan, X).</code>	X = ran.
4.	Is Haibara Conan's girlfriend?	<code>love(conan, haibara).</code>	false.
5.	Can Conan cooperate with Gin?	<code>coop(conan, gin).</code>	false.
6.	Are Conan and Vodka confronting each other?	<code>confront(conan, vodka).</code>	true.
7.	Is Conan protecting Ran from Kogoro?	<code>protect(conan, ran, kogoro).</code>	false.
8.	Who is Conan's parent?	<code>parent(X, conan).</code>	yusaku, yukiko.
9.	List all the detectives.	<code>detective(X).</code>	conan, heiji, kogoro, sera.
10.	List all the member of Black Organization.	<code>blackOrganization(X).</code>	gin, vodka, vermou, akai, amuro.
11.	List all the people that can cooperate with Conan.	<code>coop(conan, X).</code>	megure, takagi, sato, akai, amuro, heiji, haibara, sonoko, ran, yusaku, yukiko, mori, eri, kogoro, sera.

12.	List all the people that can cooperate with Gin.	coop(gin, X).	vodka, vermouth, akai, amuro.
13.	List all pair of people that can cooperate together, and they are member of black organization.	coop(X, Y), blackOrganization(X), blackOrganization(Y).	(gin, vodka), (gin, vermouth), (gin akai), (gin, amuro), (vodka, vermouth), (vodka, akai), (vodka, amuro), (vermouth, akai), (vermouth, amuro), (akai, amuro).
14.	List all the people that confront Conan.	confront(conan, X).	gin, vodka, vermouth, akai, amuro.
15.	List all the people that confront Vermouth.	confront(X, vermouth).	megure, takagi, sato, akai, amuro, conan, heiji, kogoro.
16.	List all pair of couples.	love(X, Y).	(conan, ran), (yusaku, yukiko), (kogoro, eri), (heiji, kazuha), (sato, takagi).
17.	List all the children that megure protect from Gin in Black Organization.	protect(megure, X, gin), child(X).	ayumi, genta, mitsuhiko.
18.	Who are Conan's friends?	friend(conan, X).	heiji, haibara, sonoko, ayumi, genta, mitsuhiko, ran.
19.	List spy in Black Organization.	spy(X).	akai, amuro.
20.	List all people that heiji can protect Vodka from B.O.	protect(heiji, X, vodka).	ran, eri, haibara, sonoko, sato, yukiko, sera, kazuha, ayumi, genta, mitsuhiko

### 3. [Implement logic deductive system with Python programming language.](#)

#### 3.1. Overview

The code implemented using *Python* programming language to build a logical inference program using the following deductive method: **backward reasoning** to reduce the time run by each query.

#### 3.2. Defined syntax and some important notes

Logical operator with precedence from highest to lowest: similar to SWI prolog but the not operator: '+' is excluded (may produce syntax error if used).

English	First order logic	PROLOG
Not	$\neg$	not()
Different from	$\neq$	\=
Equal to	=	=
And	$\wedge$	,
Or	$\vee$	;
If	$\rightarrow$	:-

Note that we suggest that each atom or variable shouldn't exist in its sentence or term since most of the operations work on the string representation of them, so the replace() or find()... methods work correctly.

Moreover, there is a *keyword*: “\_VarAtom” that used for replacing the atoms (constant) of the query to make finding the substitutions easier by using the result returned by the new query to find answer for the original query. Example: The query: loves(X, bob) will be replace as loves(X, \_VarAtom**bob**). => Each query or sentence in knowledge bases must not contain the above substring.

Each logic sentence must obey the following syntax besides the syntax defined by SWI prolog:

- Each operator except ',' and ';' must have exactly 2 space characters: one before and one after them. Example: sth(X) :- func(Y).
- The **and** operator (,) and **or** operator (;) must follow right after the term and one space required after them. Example: sth(X, Y), func(Z).
- Each sentence in the knowledges base (\*.pl file) must end by a period '.' and must be in separated lines.
- The not() operator can be treated as a term. Example: not(hate(adam, eve)).

When the user enters a query sentence:

- If the sentence contains any variable, the result is all substitutions possible for the sentence. Note that it will automatically produce *False* indicating that it have produce all possible substitutions.
- If the sentence doesn't contain any variable, it will produce *True* if the sentence is true in the knowledge bases, otherwise it will produce *False*. Note that, for any sentence that not included or unknown in the knowledges base, the code simply assume that this is *False*.

### 3.3. Code structures

- The code defines four constants: `CONST_ATOM`, `CONST_NUMBER`, `CONST_VARIABLE`, and `CONST_COMPLEX`, which represent different types of terms that can be used in Prolog expressions. It also defines two constants `CONST_FACT` and `CONST_RULE`, which represent two types of expressions that can be evaluated by the interpreter: facts and rules.
- Function `splitWithParenLevel()` that takes an input string `inp`, a start index `start`, an end index `end`, a delimiter string `delim`, and an optional index list `indexList` and splits `inp` into a list of substrings based on the delimiter `delim`. The delimiter is only considered if the substring is not within a pair of parentheses. The function returns the list of substrings and a list of pairs of start and end indices of the substrings in `inp`.
- The code defines several functions to determine the type of a term. The `isAtom()` function checks if a string represents an atom, which is a sequence of alphanumeric characters and underscores that starts with a lowercase letter or is enclosed in single quotes. The `isNumber()` function checks if a string represents a number. The `isVar()` function checks if a string represents a variable, which is a sequence of alphanumeric characters and underscores that starts with an uppercase letter or an underscore.
- The code also defines a class ***Term*** that represents a Prolog term. A term can be an atom, a number, a variable, or a complex term that consists of a functor and a list of arguments, where each argument is a term. The `Term` class has methods to parse an input string and create a `Term` object, update the string representation of a `Term` object, get a list of variables in a `Term` object, and assign a value to a variable in a `Term` object.

Term	
+type: str +ss: str +functor: str +arguments: list[Term]	*type represents the type of Term (atom/number/variable/complex term). *ss is string representation of Term. *functor represents the Term functor *arguments is a list of arguments in Term (complex term).

- Besides, there is a class **Relationship** that represents a binary relationship between two terms. A Relationship object has an operator and two terms. Moreover, there is a class named **Rule** that represents a Prolog rule. A Rule object has a consequent and an antecedent, where the consequent is a Term object, and the antecedent is a list of Relationship objects.

Relationships	
+operator: str +term1: Expression +term2: Expression	*operator is relation operator (';', ':-', '=', '\='). *term1 is the left part of Term. *term2 is the right part of Term, after relation operator.
Rule	
+ss: str +consequent: Term +antecedent: List[Expression]	*ss is string representation of Rule. *consequent is a Term. *antecedent is a list of Expression

- Furthermore, class **Expression** represents a logic sentence. An Expression object can be a fact or a rule. A fact is a Term object, and a rule is a Rule object. The Expression class has a method to evaluate the expression, give a list of facts, and return a list of substitutions that satisfy the expression. The method uses backtracking to generate all possible substitutions that satisfy the expression.

Expression	
+relations: list[Relationships] +contains: None/Rule/Term +type: str +ss: str	*relations is a list of Relationships in Expression. *contains can be NoneType or Rule of Term. *type represents the type of Expression (). *ss is string representations of Expression.

### 3.4. Algorithms and explanation

The algorithm that is used for answering user's query use the idea of backward chaining algorithm:

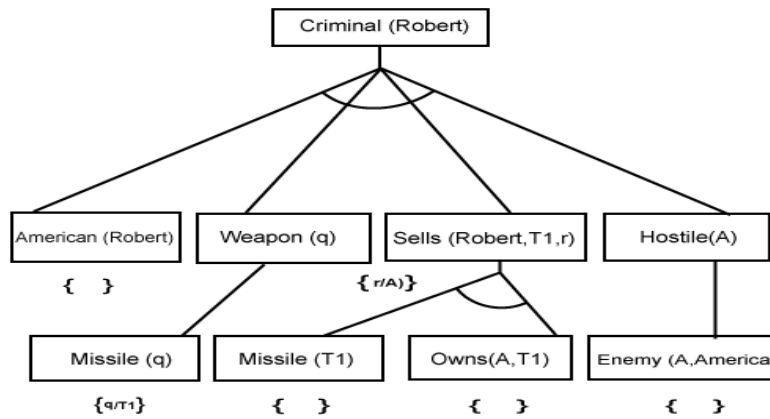


Figure 10: Backward chaining (Credit: <https://www.javatpoint.com/forward-chaining-and-backward-chaining-in-ai>)

The algorithm to check if a sentence is true works as follows:

- If the sentence is a single term (has no operator) has appeared true for any facts in the knowledges base.
- If the sentece has operator, call **recursion** on each of the subterm of the sentence and check the result of them by operator:
  - Operator and (,): return true if all of them are true.
  - Operator or (;): return true if any of them are true.
  - Operator if (:-): return true if the antecedent is false, otherwise return true if the consequent is true.
  - Operator equal to (=): return true if lhs and rhs has same values.
  - Operator different from (\=): return true if lhs and rhs has different values.
- If there is no return operation happens, check in the rules of the knowledge base, for each of the rule we do as follow:
  - If the rule's consequent can be **unify** with the goal (have same functor): call **recursion** on the antecedents. If the result of the antecedents are all true, then return true.
- If no above case happens, return false as default.

*The idea used for finding all substitutions is similar to the above algorithm. With some modifications:*

- If the sentence is a single term, we unify it with all facts in the knowledge base, if any fact can be **unify** successfully, yield the substitution returned by the unify function.
- If the sentence contains any operator, we call **recursion** on each side of the operator,:
  - Operator and (,): takes the substitutions returned, merge them together into list of substitution, with each substitution, substitute it with the goal and check if it's true, yield the substitution.
  - Operator or (;): takes the substitutions returned, merge them together into list of substitution and yield all of them.
  - Operator if (:-): yield the substitution returned by the antecedent.
  - Operator equal to (=): yield all possible equal atom (constant) in the knowledge base.
  - Operator different from (\=): yield all possible pairs of different atoms (constant) in the knowledge base.
- Check the knowledge base rules, standardize (change variable) of each rule, then check if it's the same as the goal. Call **recursion** on the antecedent and yield each substitution returned by the antecedent.

Note that, the complexity of the algorithm becomes very high ( $O(n^2)$ ) if we encounter the operator (\=) so if it appear in the operator (,), we extract them out of the goal and find substitution for the other, merge them, then extract any substitution that not satisfy the operator (\=).

Example: for the query  $\text{mother}(X, Z), \text{father}(Y, Z), X \neq Y$ . Extract  $X \neq Y$ , find all substitution for  $\text{mother}(X, Z)$  and  $\text{father}(Y, Z)$ . If  $\text{mother}(X, Z)$  return  $[\{X: a, Z: b\}, \{X: c, Z: d\}]$  and  $\text{father}(Y, Z)$  return  $[\{Y: a, Z: b\}, \{Y: e, Z: b\}]$ , then the final substitution will be  $\{X: a, Y: e, Z: b\}$ .

*Unify algorithms's pseudo code:*

```

function UNIFY( $x, y, \theta$ ) returns a substitution to make  $x$  and  $y$  identical
inputs:  $x$ , a variable, constant, list, or compound expression
          $y$ , a variable, constant, list, or compound expression
          $\theta$ , the substitution built up so far (optional, defaults to empty)

if  $\theta$  = failure then return failure
else if  $x = y$  then return  $\theta$ 
else if VARIABLE?( $x$ ) then return UNIFY-VAR( $x, y, \theta$ )
else if VARIABLE?( $y$ ) then return UNIFY-VAR( $y, x, \theta$ )
else if COMPOUND?( $x$ ) and COMPOUND?( $y$ ) then
    return UNIFY( $x$ .ARGS,  $y$ .ARGS, UNIFY( $x$ .OP,  $y$ .OP,  $\theta$ ))
else if LIST?( $x$ ) and LIST?( $y$ ) then
    return UNIFY( $x$ .REST,  $y$ .REST, UNIFY( $x$ .FIRST,  $y$ .FIRST,  $\theta$ ))
else return failure



---


function UNIFY-VAR( $var, x, \theta$ ) returns a substitution

if  $\{var/val\} \in \theta$  then return UNIFY( $val, x, \theta$ )
else if  $\{x/val\} \in \theta$  then return UNIFY( $var, val, \theta$ )
else if OCCUR-CHECK?( $var, x$ ) then return failure
else return add  $\{var/x\}$  to  $\theta$ 

```

Figure 11: Unify algorithm. Credit: [1]

### 3.5. Result of each query for each knowledge base:

#### The family tree of the British Royal Family:

No.	Question	Prolog code	Answer
1.	Who is Andrew's mother?	mother(X, andrew).	X = elizabeth.
2.	Was Elizabeth the wife of Mia Grace Tindall	divorced(elizabeth, mia).	false.
3.	Is Prince Harry the son of Prince Charles?	son(harry, charles).	true.
4.	Is Isla Phillip son of Autumn Kelly?	son(isla, autumn).	false.
5.	Is Mia Grace Tindall the grandchild of Prince Phillip?	grandchild(mia, phillip).	false.
6.	Are Prince Harry and Peter siblings?	sibling(harry, peter).	false
7.	Are Diana and Charles divorced?	divorced(diana, charles).	true.
8.	Is Charlotte married?	married(charlotte, X).	false.
9.	List the couples that married.	married(X, Y).	(elizabeth, phillip), (charles, cammilla), (anne, timothy), (edward, sophie),



			(william, kate), (peter, autumn), (zara, mike).
10.	List the couples that divorced.	divorced(X, Y).	(diana, charles) (mark, anne), (andrew, sarah).
11.	List the couples that married and have children.	married(X, Y), child(Z, X), child(Z, Y).	(elizabeth, phillip), (edward, sophie), (william, kate), (peter, autumn), (zara, mike).
12.	List the couples that are divorced but have children.	divorced(X, Y), child(Z, X), child(Z, Y).	(diana, charles), (mark, anne), (andrew, sarah).
13.	List pair of siblings.	sibling(X, Y).	(charles, anne, andrew, edward), (william, harry), (peter, zara), (beatrice, eugenie), (james, louise), (george, charlotte), (savannah, isla).
14.	Is Autumn Mia's aunt?	aunt(autumn, mia).	false. (Expected value: true).
15.	Is Peter William's uncle?	uncle(peter, william).	false.
16.	Is Mike Diana's niece?	niece(mike, diana).	false.
17.	list all of Peter's nieces.	niece(X, peter).	Intractable
18.	List all of Zara's granddaughter.	granddaughter(X, zara).	false.
19.	Who is Elizabeth's grandchild and also Sarah's child?	grandchild(X, elizabeth), child(X, sarah).	beatrice, eugenie.
20.	List all children of the divorced couple.	child(X, Y), divorced(Y, Z).	william, harry, peter, zara, beatrice, eugenie.
21.	List all Elizabeth's grandchildren that married.	grandchild(X, elizabeth), married(X, Y).	peter, zara, william.

**Character relationships of Detective Conan:**

No.	Question	Prolog code	Answer
1.	Is Conan a man?	male(conan).	true.

2.	Is Ayumi a child?	child(ayumi).	true.
3.	Who is Conan's girlfriend?	love(conan, X).	ran.
4.	Is Haibara Conan's girlfriend?	love(conan, haibara).	false.
5.	Can Conan cooperate with Gin?	coop(conan, gin).	false.
6.	Are Conan and Vodka confronting each other?	confront(conan, vodka).	true.
7.	Is Conan protecting Ran from Kogoro?	protect(conan, ran, kogoro).	false.
8.	Who are Conan's parent?	parent(X, conan).	yusaku, yukiko.
9.	List all the detectives.	detective(X).	conan, heiji, kogoro, sera.
10.	List all the member of Black Organization.	blackOrganization(X).	gin, vodka, vermouth, akai, amuro.
11.	List all the people that can cooperate with Conan.	coop(conan, X).	megure, takagi, sato, akai, amuro, heiji, haibara, sonoko, ran, yusaku, yukiko, mori, eri, kogoro, sera.
12.	List all the people that can cooperate with Gin.	coop(gin, X).	vodka, vermouth, akai, amuro.
13.	List all pair of people that can cooperate together, and they are member of black organization.	coop(X, Y), blackOrganization(X), blackOrganization(Y).	Intractable.
14.	List all the people that confront Conan.	confront(conan, X).	gin, vodka, vermouth, akai, amuro.
15.	List all the people that confront Vermouth.	confront(X, vermouth).	megure, takagi, sato, akai, amuro, conan, heiji, kogoro.
16.	List all pair of couples.	love(X, Y).	(conan, ran), (yusaku, yukiko), (kogoro, eri), (heiji, kazuha), (sato, takagi).

17.	List all the children that Megure protect from Gin in Black Organization.	protect(megure, X, gin), child(X).	Intractable.
18.	Who are Conan's friends?	friend(conan, X).	heiji, haibara, sonoko, ayumi, genta, mitsuhiko, ran.
19.	List spy in Black Organization.	spy(X).	akai, amuro.
20.	List all people that Heiji can protect Vodka from B.O.	protect(heiji, X, vodka).	ayumi, genta, mitsuhiko (not enough)

### 3.6. Some comments on the results

The code can produce right answer for most of the query. However, there are some cases where the algorithm does not match the expected result, because of the following reason:

- Some cases require many rules, so it takes a long time to get the expected result.
- Some results may be wrong or missing due to algorithm incompleteness.

### 3.7. Conclusions

Due to the limited time, the algorithm has not been completed perfectly. However, its performance is acceptable in most cases.

Note that you must enter the correct syntax which is mentioned in section 3.2 (Defined syntax and some important notes) so that the algorithm can perform exactly the queries that we expected.

## V. File submission

Some text files contains descriptions about queries for each knowledges base:

- problem1.txt: file contains queries (questions in prolog code) for the British Royal family problem.
- problem2.txt: file contains queries (questions in prolog code) for Detective Conan problem.
- About the structure of the queries file, every two lines will represent a question, the first line is the question in English, and the second line is the prolog code for the question for testing purposes.

The Source folder contains:

- problem1.pl: file contains the knowledge base for the British Royal family problem.
- problem2.pl: file contains the knowledge base for Detective Conan problem.

- *term.py*: contains *Term* class definition, some predefined constants and some important methods and functions such as: `unify()`, `substitute()`...
- *expression.py*: contains *Relationship* class, *Rule* class, *Expression* class definitions, some predefined constant, and important methods and functions such as: `mergeAndSubs()`, `standardizeVariable()`, `getValue()`, `findAllSubstitute()`, `substituteExp()`...
- *prolog.py*: contains *Prolog* class representation.
- *main.py*: contains `main()` function to run code.

## VI. References

Some reference links and books that we used:

- [1] First order logic: Russell, S. J., & Norvig, P. (2010). Artificial Intelligence: A Modern Approach (3rd ed.). Prentice Hall.
- [2] SWI-Prolog: <https://www.swi-prolog.org/>
- [3] Prolog - Wikipedia: <https://en.wikipedia.org/wiki/Prolog>
- [4] Prolog | An Introduction - GeeksforGeeks: <https://www.geeksforgeeks.org/prolog-an-introduction/>
- [5] Prolog - Introduction (tutorialspoint.com):  
[https://www.tutorialspoint.com/prolog/prolog\\_introduction.htm](https://www.tutorialspoint.com/prolog/prolog_introduction.htm)
- [6] Prolog Syntax - javatpoint: <https://www.javatpoint.com/prolog-syntax>
- [7] Prolog FACTS, RULES and QUERIES:  
<http://www.cs.trincoll.edu/~ram/cpsc352/notes/prolog/factsrules.html?fbclid=IwAR1jIIJfKiT4bgxgUhaJJij5GA1dwMo1Tl4HtC6fvvaQdF1GH4mtMeOCFLY>