# Introduction to AI
# First-order Logic

Project 2

21CLC05 - University of Science - VNUHCM

21127004 - 21127081 - 21127135 - 21127325

✦

## 1. Report

*This will be a long read so please make use of bookmarks for ease of navigation.*

| Priority | No. | Task | Status |
|---|---|---|---|
| Required | 1.1 | Learn the Prolog language, write report on main features and give examples. | Done |
| | 1.2 | Learn a Prolog environment, write report on how to implement Prolog and give examples. | Done |
| | 1.3 | Solving deductive problems using SWI-Prolog, build a Knowledge Base from the British Royal Family Tree, give at least 20 questions, and compile the outputs. | Done |
| | 2 | Build a Knowledge Base from any topic containing at least 50 predicates, give at least 20 questions, and compile the outputs. | Done |
| | 3 | Build a logical inference program using one of the deductive methods, verify the results with 1.3 and 2. | |
| Bonus | a | | |
| | b | | |
| | c | | |

**Project Completion Rate:** 100%.

| Student ID | Full name | Tasks | Contribution |
|---|---|---|---|
| 21127004 | Trần Nguyễn An Phong | 1.1, 1.2, 3 | 100% |
| 21127081 | Nguyễn Minh Khôi | 1.1, 1.2, 2 | 100% |
| 21127135 | Diệp Hữu Phúc | 1.1, 1.2, 1.3 | 100% |
| 21127325 | Phan Đặng Anh Khôi | 1.1, 1.2, 3 | 100% |

# 2. Working with the Prolog tool (40%)

## Learn the Prolog language

“Prolog is a logic programming language associated with artificial intelligence and computational linguistics.” – Wikipedia

Unlike many other programming languages, Prolog is intended primarily as a declarative programming language. In Prolog, program logic is expressed in terms of relations, and a computation is carried out by running a query over these relations.

A typical Prolog program requires a Knowledge Base, which is composed of predefined clauses, i.e., facts and rules. The Knowledge Base can, then, be queried against to generate outputs. If our query is already in the Knowledge Base or it is implied by Knowledge Base, the result might be true or a list of all satisfied terms, otherwise we get false.

| **Query** | **Result** |
|---|---|
| ?- sister('zara_phillips', 'peter_phillips'). | true. |
| ?- nephew(X, 'princess_anne'). | X = prince_william ;<br>X = prince_harry ;<br>X = james_viscount_severn ;<br>false. |
| ?- granddaughter('isla_phillips', 'timothy_laurence'). | false. |

### Terms

- The sole data type of Prolog.
- Terms are either atoms, numbers, variables, or compound terms.

| **Terms** | **Definition** | **Example** |
|---|---|---|
| **Atoms** | General-purpose name with no inherent meaning. | x, red, ‘Hello world’ |
| **Numbers** | Floats or integers. | 6.9, 420, 727 |
| **Variables** | Strings consisting of letters, numbers, and underscores. Must begin with an upper-case letter or underscore. | X, _360, Hello_world |
| **Compound terms** | - Composed of an atom called a *functor* and several *arguments*, which are terms.<br>- The number of arguments is called the *arity*. An atom can be seen as a compound term with arity zero.<br>- Lists and Strings are special cases of compound terms. | foo(bar),<br>point(X, Y, Z),<br>"Hello world",<br>.(1,.(2,.(3,[]))),<br>[1, 2, 3] |

## Facts and rules

Prolog programs describe relations, which are defined by means of clauses. There are two types of clauses: facts and rules. A **rule** is of the form,

$$Head :- Body.$$

which is read as "Head is true if Body is true". A rule's body consists of calls to predicates, which are called the rule's goals. The built-in logical operator $,/2$ denotes conjunction of goals, and $;/2$ denotes disjunction. Conjunctions and disjunctions can only appear in the body, not in the head of a rule.

**Facts** can be interpreted as clauses with empty bodies or $true/0$ as the bodies. Below are some examples of both,

| | |
|---|---|
| **Facts** | program.<br>hello_world.<br>hello(world) :- true.<br>hello(program, world). |
| **Rules** | learn(Stud, Subj) :- is_student(Stud), is_subject(Subj).<br>execute(Program) :- is_program(Program).<br>rule(X, Y) :- head(X), body(Y).<br>grandparent(GP, GC) :- parent(GP, X), parent(X, GC). |

## Execution

"Prolog's execution strategy can be thought of as a generalization of function calls in other languages, one difference being that multiple clause heads can match a given call. In that case, the system creates a choice-point, unifies the goal with the clause head of the first alternative, and continues with the goals of that first alternative. If any goal fails in the course of executing the program, all variable bindings that were made since the most recent choice-point was created are undone, and execution continues with the next alternative of that choice-point. This execution strategy is called chronological backtracking." – Wikipedia

*Well, that sure was a lot of texts!* Fortunately, we are going to meticulously walk through an example, so all you need to retain from that information dump is the strategy being coined **chronological backtracking**.

| | |
|---|---|
| **Knowledge Base** | `music(pulse).`<br>`music(tantantan).`<br>`vocal(tantantan).`<br>`song(X) :- music(X), vocal(X).` |
| **Query** | `?- song(JSong).`<br>`JSong = tantantan.` |

1. Firstly, Prolog searches for all clauses containing the predicate **song/1**.

$$song(X) :- music(X), vocal(X).$$

2. It will then try to unify **song(X)** and **song(JSong)** by generating a common variable, e.g., **Var**. This could also be depicted as the binding of two variables **X** and **JSong**. Therefore, the query will be evaluated as,

$$song(Var) :- music(Var), vocal(Var).$$

3. Proving the query is equivalent to proving the body of that clause with the appropriate variable bindings in place, i.e., the conjunction. In this case, it is **music(Var), vocal(Var)**, and the first goal needs to be proven is **music(Var)**. Prolog proceeds to look up in the Knowledge Base all clauses with the predicate **music/1** and find,

$$music(pulse).$$

4. Similar to before, it will attempt to unify **music(Var)** and **music(pulse)**. In this case, the unification is a binding of a variable **Var** and a constant, i.e., a fact, **pulse/0**, which gives us the new body,

$$music(pulse), vocal(pulse).$$

5. Since **music(pulse)** is a known fact in the Knowledge Base, the next goal to be proven is **vocal(pulse)**. And as we can expect, **vocal(pulse)** doesn't exist in the Knowledge Base so the query for it fails. Thus, Prolog backtracks to the previous body,

$$music(Var), vocal(Var).$$

6. It continues with the look up for the predicate **music** and discovers **music(tantantan)**. Likewise, a unification for **music(Var)** and **music(tantantan)** takes place and we have,

$$music(tantantan), vocal(tantantan).$$

7. Once again, Prolog seeks **music(tantantan)** and succeeds. Hence, it moves on to **vocal(tantantan)**, thankfully this time, **vocal(tantantan)** is also a recognized fact.
8. Because all goals, i.e., the body, could be proven, the query for **song(X)** succeeds. As the query contains a variable **X**, all bindings are reported to the user, thus, we are presented with,

$$JSong = tantantan.$$

## Loops and recursion

"Iterative algorithms can be implemented by means of recursive predicates." – Wikipedia

This concept is best explained through concrete examples and that is exactly what we are going to do. Nonetheless, we won't be bothered re-explaining some of the trivial steps that had already been done on the **Execution** section above.

| | |
|---|---|
| **Knowledge Base** | `route(home, park).`<br>`route(park, bus).`<br>`route(bus, hcmus).`<br>`goto(Src, Dest) :- route(Src, Dest).`<br>`goto(Src, Dest) :- route(Src, X), goto(X, Dest).` |
| **Query** | `?- goto(home, hcmus).`<br>`true ;`<br>`false.` |

1. Firstly, Prolog searches for all clauses containing the predicate **goto/2**. It catches the first instance with the body of just **route/2** and a unification takes place.

$$goto(home, hcmus) :- route(home, hcmus).$$

2. Expectedly, this will fail since **route(home, hcmus)** is not a fact and the body, therefore, can't be proven. Consequently, Prolog will try the next instance of **goto/2**.

$$goto(home, hcmus) :- route(home, X), goto(X, hcmus).$$

3. Now to prove the first goal of the body, a search for all clauses with predicate **route/2** begins. We discovers **route(home, park)** which is a known fact, and so Prolog binds $\boldsymbol{X = park}$, and the goal is successfully proven. Additionally, we obtain a new body.

$$route(home, park), goto(park, hcmus).$$

4. While retaining the binding, Prolog moves on to the next goal of the new body. *This is where our recursion will occur*.
5. Firstly, Prolog searches for all clauses containing the predicate **goto/2**. It catches the first instance with the body of just **route/2** and a unification takes place.

$$goto(park, hcmus) :- route(park, hcmus).$$

6. Expectedly, this will fail since **route(park, hcmus)** is not a fact and the body, therefore, can't be proven. Consequently, Prolog will try the next instance of **goto/2**.

$$goto(park, hcmus) :- route(park, X), goto(X, hcmus).$$

7. Now to prove the first goal of the body, a search for all clauses with predicate **route/2** begins. We discovers **route(park, bus)** which is a known fact, and so Prolog binds $\boldsymbol{X = bus}$, and the goal is successfully proven. Additionally, we obtain a new body.

$$route(park, bus), goto(bus, hcmus).$$

8. While retaining the binding, Prolog moves on to the next goal of the new body. *This is where our recursion will occur*.
9. Firstly, Prolog searches for all clauses containing the predicate **goto/2**. It catches the first instance with the body of just **route/2** and a unification takes place.

$$goto(bus, hcmus) :- route(bus, hcmus).$$

10. *Unexpectedly*, this will succeed since **route(bus, hcmus)** is a known fact and the body, therefore, can be proven. Consequently, this means the last goal of our body at step 3 has been successfully shown to be true.

$$goto(home, hcmus) :- route(home, park), route(park, hcmus).$$

11. Since all goals could be proven, our initial query **goto(home, hcmus)** succeeds. As the query contains no variables, no bindings are reported to the user and Prolog only outputs **true**.

We are perfectly aware of the **false.** which is often returned on the last line. To loosely explain, the contents above the **false.** are everything satisfying the query that Prolog could find. The **false.** appears this way can be interpreted as Prolog telling you it cannot find anything else.

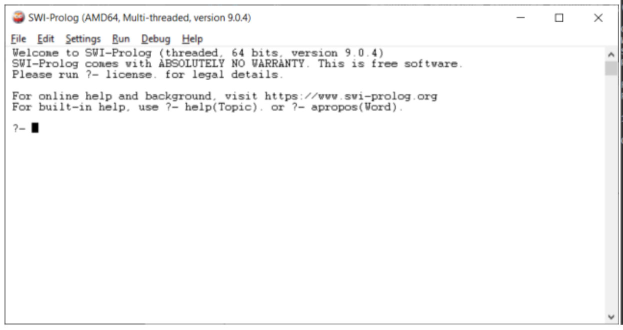
## Learn a Prolog programming environment

Our environment of choice is SWI-Prolog, specifically the stable version for 64-bit Windows. We won't delve further into the installation process since the installer is already very intuitive. Below is the download link,

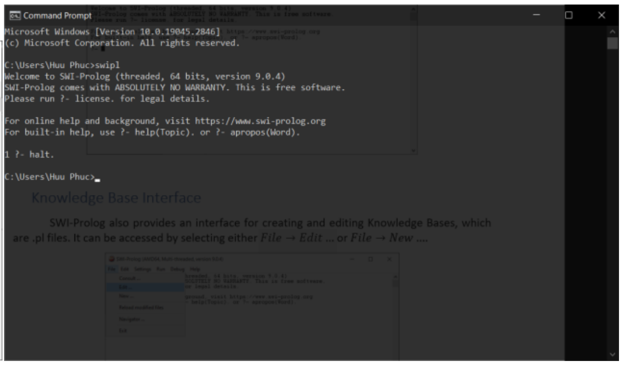
- https://www.swi-prolog.org/download/stable

### Query Interface

Although SWI-Prolog has a main interface for writing queries, this environment can essentially be initiated on any command prompt by inputting **swipl**. Additionally, a session can be ended by doing **halt.** query.


SWI-Prolog (AMD64, Multi-threaded, version 9.0.4)
File Edit Settings Run Debug Help

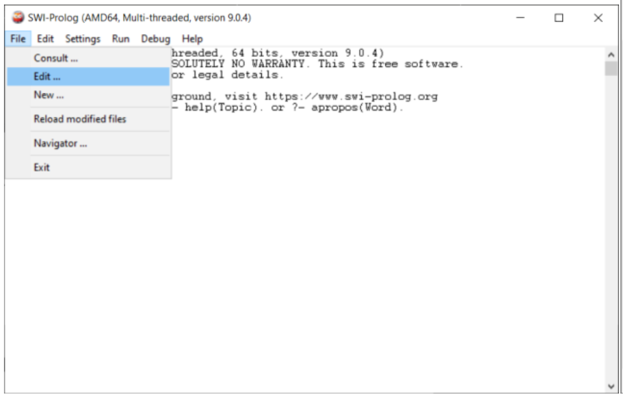
```
Welcome to SWI-Prolog (threaded, 64 bits, version 9.0.4)
SWI-Prolog comes with ABSOLUTELY NO WARRANTY. This is free software.
Please run ?- license. for legal details.

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

?- 
```



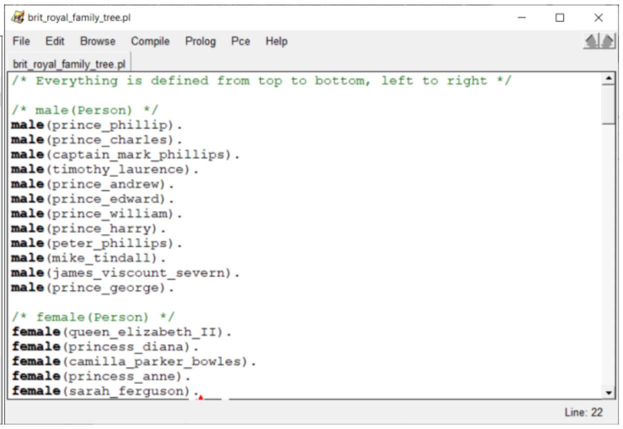
```
Microsoft Windows [Version 10.0.19045.2846]
(c) Microsoft Corporation. All rights reserved.

C:\Users\Huu Phuc>swipl
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).

1 ?- halt.

C:\Users\Huu Phuc>
```

### Knowledge Base Interface

SWI-Prolog also provides an interface for creating and editing Knowledge Bases, which are .pl files. It can be accessed by selecting either $File \rightarrow Edit$ … or $File \rightarrow New$ ….

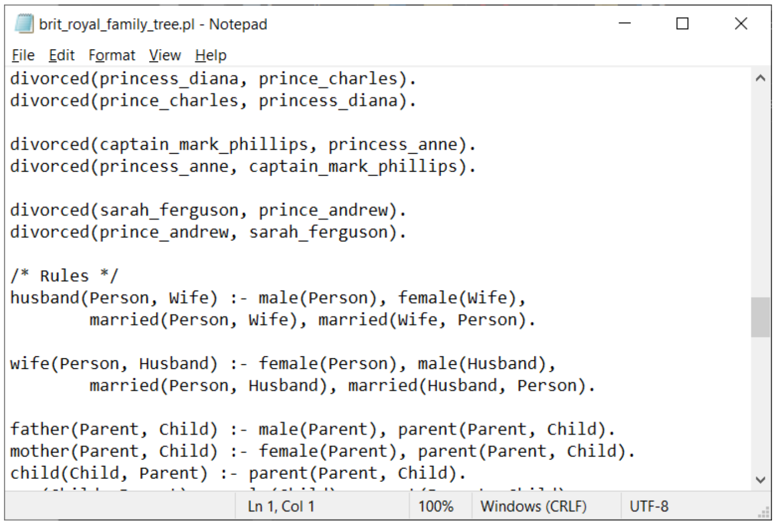

SWI-Prolog (AMD64, Multi-threaded, version 9.0.4)
File Edit Settings Run Debug Help
Consult ...
Edit ...
New ...
Reload modified files
Navigator ...
Exit

```
hreaded, 64 bits, version 9.0.4)
SOLUTELY NO WARRANTY. This is free software.
or legal details.

ground, visit https://www.swi-prolog.org
- help(Topic). or ?- apropos(Word).
```


brit_royal_family_tree.pl
File Edit Browse Compile Prolog Pce Help
brit_royal_family_tree.pl
Line: 22

```
/* Everything is defined from top to bottom, left to right */

/* male(Person) */
male(prince_phillip).
male(prince_charles).
male(captain_mark_phillips).
male(timothy_laurence).
male(prince_andrew).
male(prince_edward).
male(prince_william).
male(prince_harry).
male(peter_phillips).
male(mike_tindall).
male(james_viscount_severn).
male(prince_george).

/* female(Person) */
female(queen_elizabeth_II).
female(princess_diana).
female(camilla_parker_bowles).
female(princess_anne).
female(sarah_ferguson).
```

However, the .pl files can be modified with pretty much any text editors in existence. As for the format, each line denotes a clause, i.e., a rule or a fact, that must end with a dot.


brit_royal_family_tree.pl - Notepad
File Edit Format View Help
Ln 1, Col 1
100%
Windows (CRLF)
UTF-8

```
divorced(princess_diana, prince_charles).
divorced(prince_charles, princess_diana).

divorced(captain_mark_phillips, princess_anne).
divorced(princess_anne, captain_mark_phillips).

divorced(sarah_ferguson, prince_andrew).
divorced(prince_andrew, sarah_ferguson).

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

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

father(Parent, Child) :- male(Parent), parent(Parent, Child).
mother(Parent, Child) :- female(Parent), parent(Parent, Child).
child(Child, Parent) :- parent(Parent, Child).
```

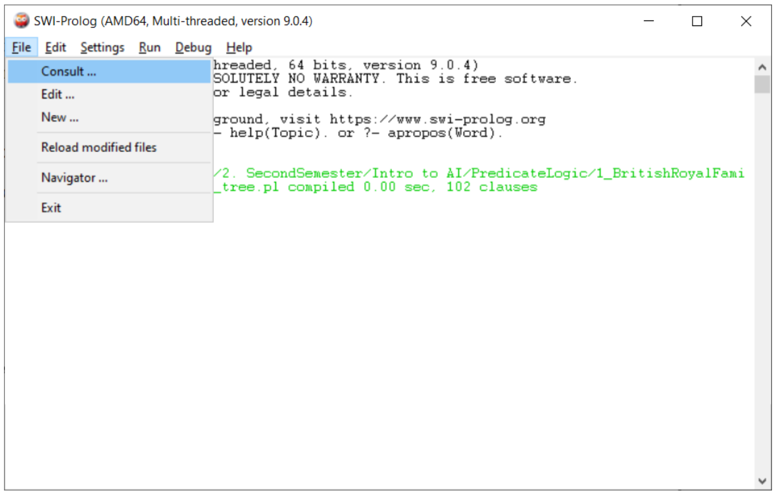
Apart from using the interface and text editors, it is possible to construct or modify a Knowledge Base with queries. Considering it will be quite a hefty write for us, we will just leave the link to the source documentation here for your perusal.

- https://www.swi-prolog.org/pldoc/man?section=dynpreds

## Consulting a Knowledge Base

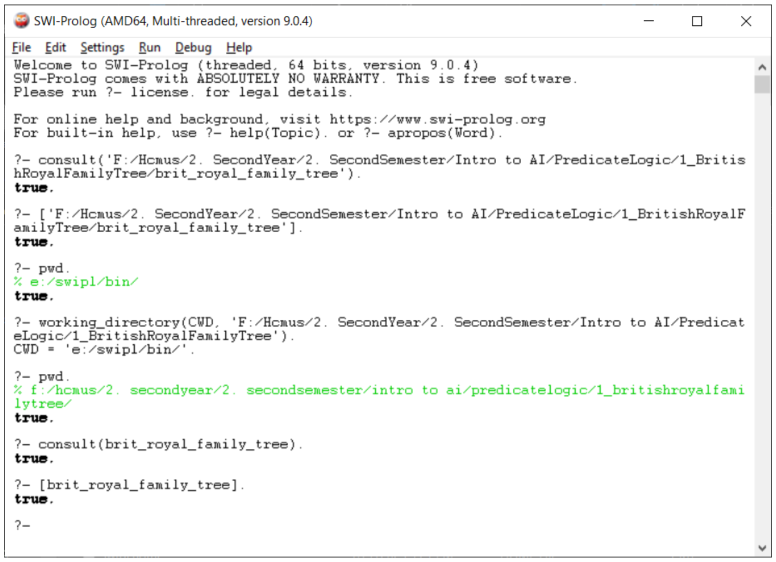
There are multiple methods to load a Knowledge Base.

- Method 1: Using $File \rightarrow Consult$ ….



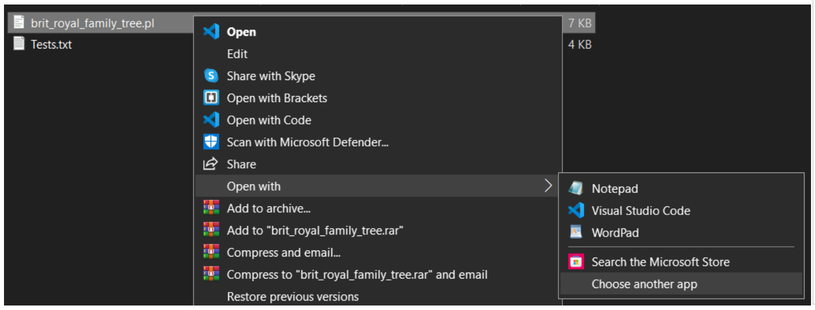
```
hreaded, 64 bits, version 9.0.4)
SOLUTELY NO WARRANTY. This is free software.
or legal details.

ground, visit https://www.swi-prolog.org
- help(Topic). or ?- apropos(Word).

/2. SecondSemester/Intro to AI/PredicateLogic/1_BritishRoyalFami
_tree.pl compiled 0.00 sec, 102 clauses
```

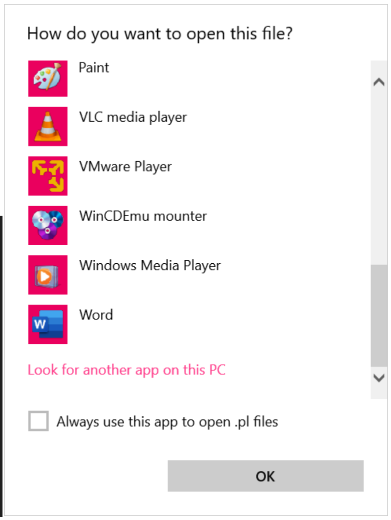
- **Method 2:** Using **consult('path:/to/pl/file')** or **['path:/to/pl/file']**.
- **Method 3:** Changing directory with **working_directory(CWD, 'path:/to/dir/with/pl/file')** then **consult(plFile)** or **[plFile]**.

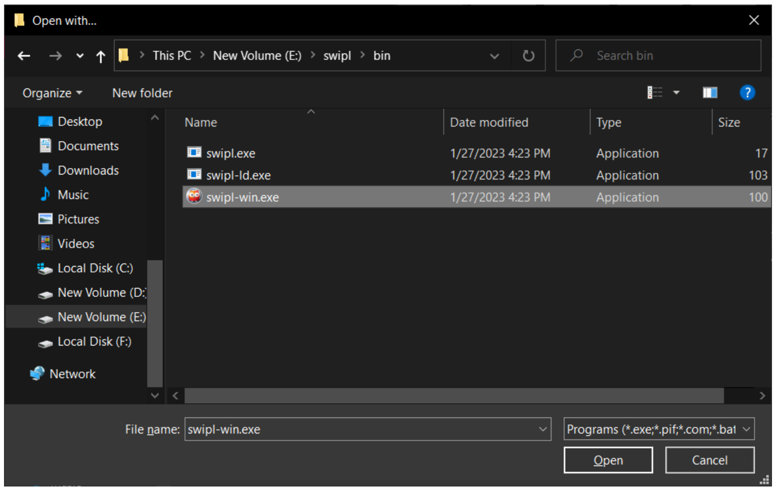


```
Welcome to SWI-Prolog (threaded, 64 bits, version 9.0.4)
SWI-Prolog comes with ABSOLUTELY NO WARRANTY. This is free software.
Please run ?- license. for legal details.

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

?- consult('F:/Hcmus/2. SecondYear/2. SecondSemester/Intro to AI/PredicateLogic/1_Britis
hRoyalFamilyTree/brit_royal_family_tree').
true.

?- ['F:/Hcmus/2. SecondYear/2. SecondSemester/Intro to AI/PredicateLogic/1_BritishRoyalF
amilyTree/brit_royal_family_tree'].
true.

?- pwd.
% e:/swipl/bin/
true.

?- working_directory(CWD, 'F:/Hcmus/2. SecondYear/2. SecondSemester/Intro to AI/Predicat
eLogic/1_BritishRoyalFamilyTree').
CWD = 'e:/swipl/bin/'.

?- pwd.
% f:/hcmus/2. secondyear/2. secondsemester/intro to ai/predicatelogic/1_britishroyalfami
lytree/
true.

?- consult(brit_royal_family_tree).
true.

?- [brit_royal_family_tree].
true.

?-
```

- **Method 4:** Opening the .pl file directly with **swipl-win.exe**.


brit_royal_family_tree.pl
Tests.txt
Open
Edit
Share with Skype
Open with Brackets
Open with Code
Scan with Microsoft Defender...
Share
Open with
Add to archive...
Add to "brit_royal_family_tree.rar"
Compress and email...
Compress to "brit_royal_family_tree.rar" and email
Restore previous versions
7 KB
4 KB
Notepad
Visual Studio Code
WordPad
Search the Microsoft Store
Choose another app


How do you want to open this file?
Paint
VLC media player
VMware Player
WinCDEmu mounter
Windows Media Player
Word
Look for another app on this PC
Always use this app to open .pl files
OK



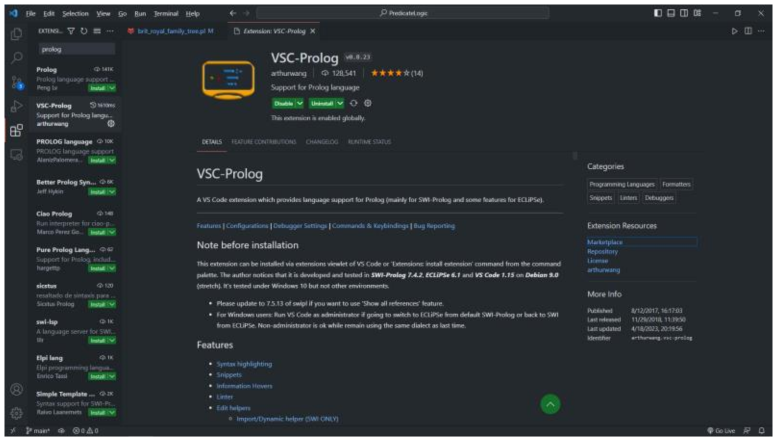
| Name | Date modified | Type | Size |
|---|---|---|---|
| swipl.exe | 1/27/2023 4:23 PM | Application | 17 |
| swipl-ld.exe | 1/27/2023 4:23 PM | Application | 103 |
| swipl-win.exe | 1/27/2023 4:23 PM | Application | 100 |

File name: swipl-win.exe
Programs (*.exe;*.pif;*.com;*.bat

## Integrating with Visual Code

In our humble opinion, the base interfaces of SWI-Prolog are not very aesthetically pleasing. It is that, combined with the annoying act of having to jump between multiple windows during test compilation or debugging, gives enough reasons to seek out the reliable VSCode.

All we need to do for the integration is a decent Prolog extension. We recommend VSC-Prolog since it has the nicest-looking syntax highlighting.



VSC-Prolog v0.8.23

arthurwang | 128,541 | ★★★★☆ (14)

Support for Prolog language

This extension is enabled globally.

VSC-Prolog

A VS Code extension which provides language support for Prolog (mainly for SWI-Prolog and some features for ECLiPSe).

Features | Configurations | Debugger Settings | Commands & Keybindings | Bug Reporting

Note before installation

This extension can be installed via extensions viewlet of VS Code or 'Extensions: install extension' command from the command palette. The author notices that it is developed and tested in ***SWI-Prolog 7.4.2***, ***ECLiPSe 6.1*** and ***VS Code 1.15*** on ***Debian 9.0*** (stretch). It's tested under Windows 10 but not other environments.

- Please update to 7.5.13 of swipl if you want to use 'Show all references' feature.
- For Windows users: Run VS Code as administrator if going to switch to ECLiPSe from default SWI-Prolog or back to SWI from ECLiPSe. Non-administrator is ok while remain using the same dialect as last time.

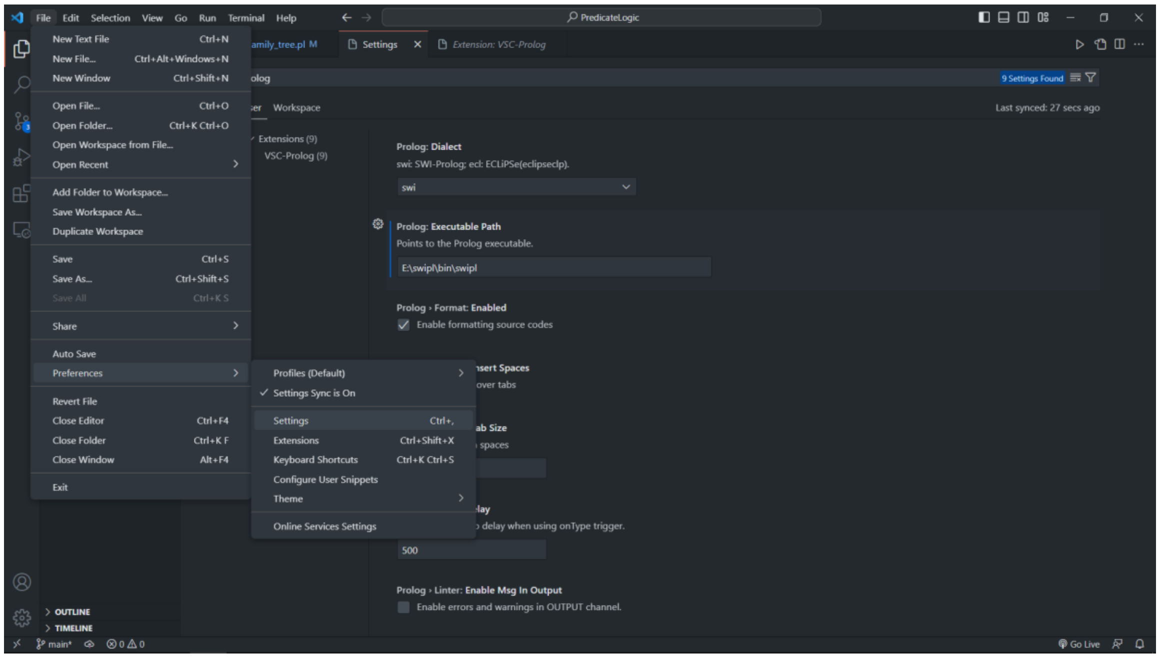
Features

- Syntax highlighting
- Snippets
- Information Hovers
- Linter
- Edit helpers
  - Import/Dynamic helper (SWI ONLY)

| | |
|---|---|
| Published | 8/12/2017, 16:17:03 |
| Last released | 11/29/2018, 11:39:50 |
| Last updated | 4/18/2023, 20:19:56 |
| Identifier | arthurwang.vsc-prolog |

If you proceed with this extension, for the linters to work properly,

1. Go into $File \rightarrow Preferences \rightarrow Settings$.
2. Search for **prolog**.
3. Change the **Executable Path** to where your **swipl.exe** is located.



Prolog: Dialect
swi: SWI-Prolog; ecl: ECLiPSe(eclipseclp).
swi

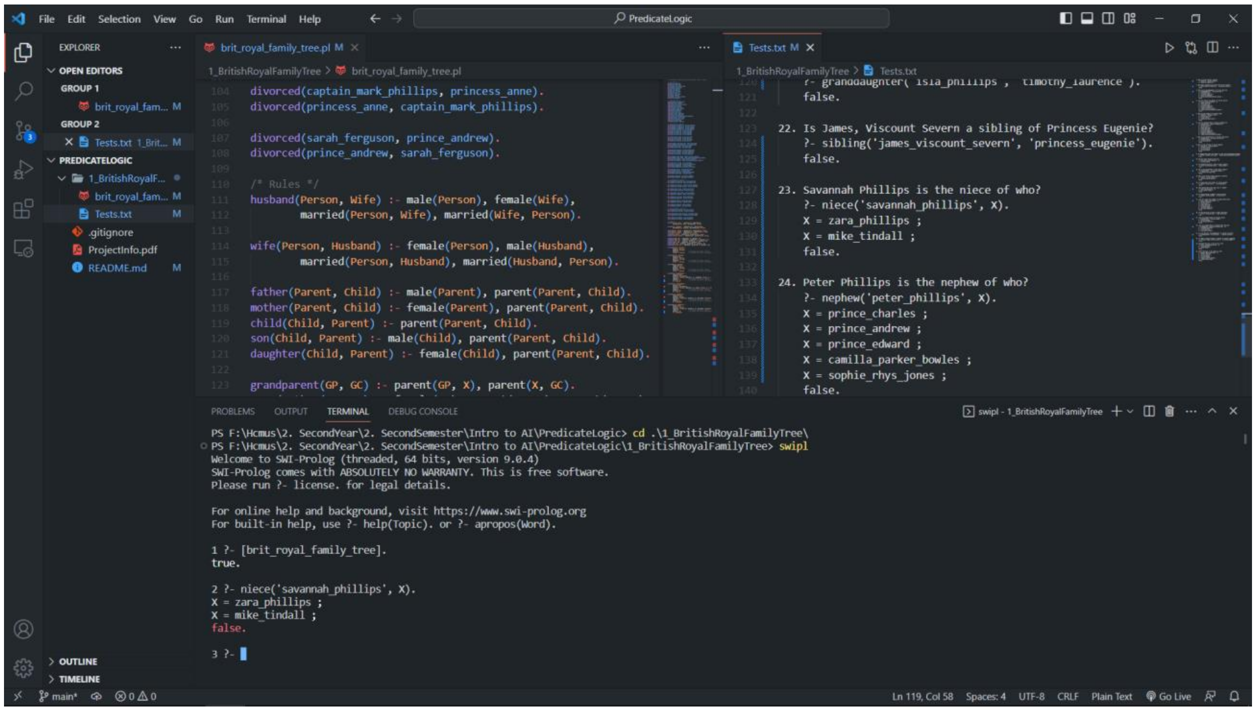
Prolog: Executable Path
Points to the Prolog executable.
E:\swipl\bin\swipl

Prolog › Format: Enabled
✓ Enable formatting source codes

Prolog › Linter: Enable Msg In Output
Enable errors and warnings in OUTPUT channel.

## Writing some queries

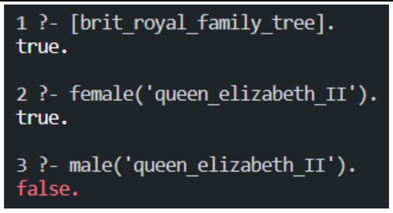
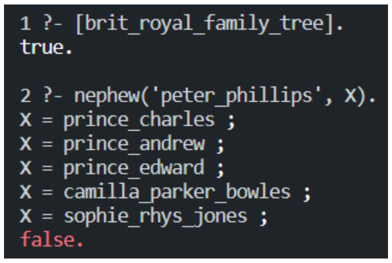
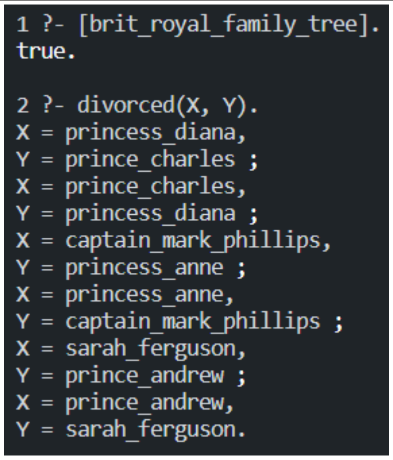
Below is a picture depicting our workflow.

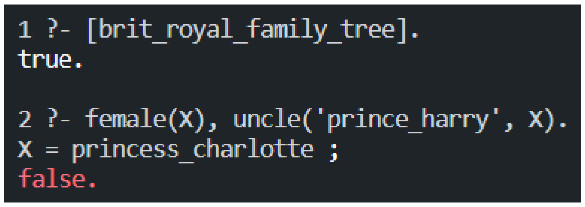
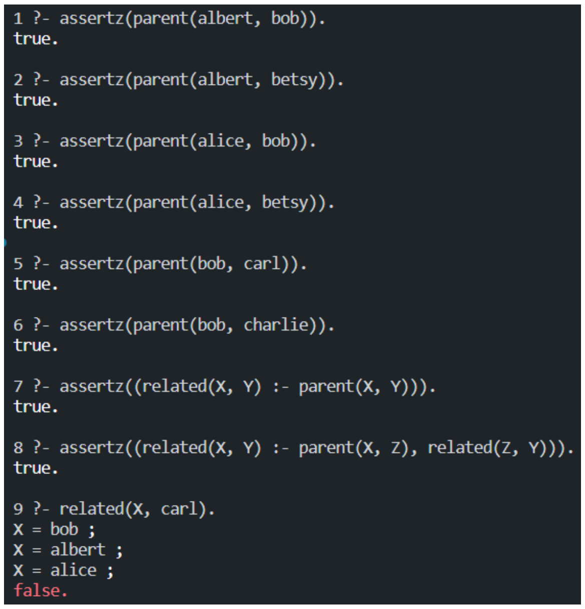


```
divorced(captain_mark_phillips, princess_anne).
divorced(princess_anne, captain_mark_phillips).

divorced(sarah_ferguson, prince_andrew).
divorced(prince_andrew, sarah_ferguson).

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

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

father(Parent, Child) :- male(Parent), parent(Parent, Child).
mother(Parent, Child) :- female(Parent), parent(Parent, Child).
child(Child, Parent) :- parent(Parent, Child).
son(Child, Parent) :- male(Child), parent(Parent, Child).
daughter(Child, Parent) :- female(Child), parent(Parent, Child).

grandparent(GP, GC) :- parent(GP, X), parent(X, GC).
```

```
    ?- granddaughter('isla_phillips', 'timothy_laurence').
    false.

22. Is James, Viscount Severn a sibling of Princess Eugenie?
    ?- sibling('james_viscount_severn', 'princess_eugenie').
    false.

23. Savannah Phillips is the niece of who?
    ?- niece('savannah_phillips', X).
    X = zara_phillips ;
    X = mike_tindall ;
    false.

24. Peter Phillips is the nephew of who?
    ?- nephew('peter_phillips', X).
    X = prince_charles ;
    X = prince_andrew ;
    X = prince_edward ;
    X = camilla_parker_bowles ;
    X = sophie_rhys_jones ;
    false.
```

```
PS F:\Hcmus\2. SecondYear\2. SecondSemester\Intro to AI\PredicateLogic> cd .\1_BritishRoyalFamilyTree\
PS F:\Hcmus\2. SecondYear\2. SecondSemester\Intro to AI\PredicateLogic\1_BritishRoyalFamilyTree> swipl
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).

1 ?- [brit_royal_family_tree].
true.

2 ?- niece('savannah_phillips', X).
X = zara_phillips ;
X = mike_tindall ;
false.

3 ?-
```

| Type | Query |
|---|---|
| **True/False** |  1 ?- [brit_royal_family_tree].<br>true.<br><br>2 ?- female('queen_elizabeth_II').<br>true.<br><br>3 ?- male('queen_elizabeth_II').<br>false. |
| **Multiple results** |  1 ?- [brit_royal_family_tree].<br>true.<br><br>2 ?- nephew('peter_phillips', X).<br>X = prince_charles ;<br>X = prince_andrew ;<br>X = prince_edward ;<br>X = camilla_parker_bowles ;<br>X = sophie_rhys_jones ;<br>false. |
| **Multiple variables** |  1 ?- [brit_royal_family_tree].<br>true.<br><br>2 ?- divorced(X, Y).<br>X = princess_diana,<br>Y = prince_charles ;<br>X = prince_charles,<br>Y = princess_diana ;<br>X = captain_mark_phillips,<br>Y = princess_anne ;<br>X = princess_anne,<br>Y = captain_mark_phillips ;<br>X = sarah_ferguson,<br>Y = prince_andrew ;<br>X = prince_andrew,<br>Y = sarah_ferguson. |

| | |
|---|---|
| **Multiple predicates** |  `1 ?- [brit_royal_family_tree].`<br>`true.`<br><br>`2 ?- female(X), uncle('prince_harry', X).`<br>`X = princess_charlotte ;`<br>`false.` |
| **Recursion** |  `1 ?- assertz(parent(albert, bob)).`<br>`true.`<br><br>`2 ?- assertz(parent(albert, betsy)).`<br>`true.`<br><br>`3 ?- assertz(parent(alice, bob)).`<br>`true.`<br><br>`4 ?- assertz(parent(alice, betsy)).`<br>`true.`<br><br>`5 ?- assertz(parent(bob, carl)).`<br>`true.`<br><br>`6 ?- assertz(parent(bob, charlie)).`<br>`true.`<br><br>`7 ?- assertz((related(X, Y) :- parent(X, Y))).`<br>`true.`<br><br>`8 ?- assertz((related(X, Y) :- parent(X, Z), related(Z, Y))).`<br>`true.`<br><br>`9 ?- related(X, carl).`<br>`X = bob ;`<br>`X = albert ;`<br>`X = alice ;`<br>`false.` |

## The British Royal Family Tree

The relevant folder is **1_BritishRoyalFamilyTree**.

- brit_royal_family_tree.pl: The Knowledge Base constructed from the British Royal Family.
- brit_royal_family_tree_image.png: The image showing the whole tree of the British Royal Family.
- Tests.txt: Records every single pair of performed query and its output. *Due to their lengthy nature, we chose not to include them in this report.*

For most of the predicates are very straightforward, we will only elaborate on the ones that, we think, are noteworthy.

### Sibling, brother, sister

Since they share basically the same implementation, let us examine **brother**.

```
brother(Person, Sibling) :-
        male(Person),
        parent(X, Person),
        parent(X, Sibling),
        female(X),              % The gender here doesn't matter,
        Person \= Sibling.      % we choose only one to avoid duplication.
```

Since we only care about whether **Person** and **Sibling** share one parent **X**, the parent's gender doesn't matter. Hence, we arbitrarily picked **female(X)**, we could also have gone with **male(X)**. One last thing is that we must make sure that two different people are really being checked, thus, **Person \= Sibling**.

Here is an example of the duplication when we removed **female(X)**.

```
?- brother('prince_george', X).
X = princess_charlotte ;
X = princess_charlotte.
```

### Aunt, uncle, niece, nephew

Since they share basically the same implementation, let us examine **aunt**.

```
aunt(Person, NieceNephew) :-
        female(Person),
        parent(X, NieceNephew),
        (parent(Y, Person); (parent(Y, Z), husband(Z, Person), X \= Z)),
        female(Y),              % The gender here doesn't matter,
        parent(Y, X),           % we choose only one to avoid duplication.
        X \= Person.
```

We consider an aunt to be either a parent's sister or uncle's wife. As we have already written the logic for the **sister** predicate, we will skip them here. On to the uncle's wife, instead of checking for a parent **Y** of both **X** and **Person**, we need a parent **Y** of both **X** and the husband **Z** of **Person**, and to also ensure that we don't check someone multiple times with **X \= Z**.

Once again, here is an example of the duplication when we removed **female(X)**.

```
?- aunt('autumn_kelly', X).
X = mia_grace_tindall ;
X = mia_grace_tindall ;
false.
```

And here is an example of what will happen if the **X \= Z** check is neglected.

```
?- aunt('autumn_kelly', X).
X = savannah_phillips ;
X = isla_phillips ;
X = mia_grace_tindall ;
false.
```

## 3. Build a Knowledge Base with Prolog (30%)

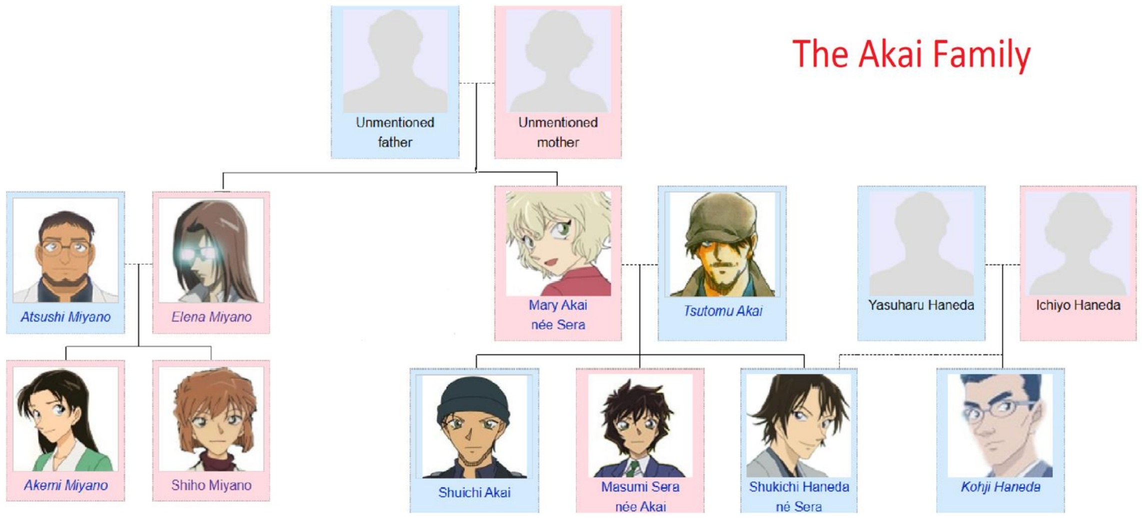
The relevant folder is **2_AkaiFamilyTree**.

- akai_family_tree.pl: The Knowledge Base constructed from the Akai Family from the universe of Case Closed, or Detective Conan.

- akai_family_tree_image.png: The image showing the whole tree of the Akai Family.
- Tests.txt: Records every single pair of performed query and its output. *Due to their lengthy nature, we chose not to include them in this report.*

## The Akai Family Tree

***Due to the limitations of our tree-drawing skill, we were not able to display every single relation. Thus, we highly urge you to view the image and the Knowledge Base, or the tests, side by side for the best experience.***



## Base predicates (Facts)

There is nothing to comment about these as they are already very self-descriptive.

1. **male(Person).**
2. **female(Person).**
3. **is_british(Person).**
4. **is_japanese(Person).**
5. **is_biochemist(Person).**
6. **was_black_organization(Person).**
7. **is_fbi_agent(Person).**
8. **is_cia_agent(Person).**
9. **is_mi6_agent(Person).**
10. **is_alive(Person).**
11. **shrank(Person).**
12. **parent(Parent, Child).**
13. **non_blood_brother(Person1, Person2).**
14. **dating(Person1, Person2).**
15. **married(Person1, Person2).**

## Derived predicates (Rules)

The predicates that are built from the above predicates, we will only elaborate on those that are more confusing.

### Family relationships

Most of the predicates below had been discussed in the **British Royal Family Tree** section of chapter 2, **Working with the Prolog tool**. Even though we experimented with some different implementations, the logic of the predicates is retained.

16. **husband(Person, Wife).**
17. **wife(Person, Husband).**
18. **father(Parent, Child).**
19. **mother(Parent, Child).**
20. **child(Child, Parent).**
21. **son(Child, Parent).**
22. **daughter(Child, Parent).**
23. **grandparent(GP, GC).**
24. **grandmother(GM, GC).**
25. **grandfather(GF, GC).**
26. **grandchild(GC, GP).**
27. **grandson(GS, GP).**
28. **granddaughter(GD, GP).**
29. **sibling(Person1, Person2).**
30. **brother(Person, Sibling).**
31. **sister(Person, Sibling).**
32. **aunt(Person, NieceNephew).**
33. **uncle(Person, NieceNephew).**
34. **niece(Person, AuntUncle).**
35. **nephew(Person, AuntUncle).**

These are the only new additions to the rudimentary family relationships, they both involve the base predicate **dating(Person1, Person2)**.

36. **boyfriend(Person, Girlfriend).**
37. **girlfriend(Person, Boyfriend).**

### Non-blood relationships

The predicates in this segment describe the relationships that necessitate the base predicate **non_blood_brother(Person1, Person2)**.

38. **auntNB(AuntNB, Person).**
39. **uncleNB(UncleNB, Person).**
40. **brotherNB(BrotherNB, Person).**
41. **sisterNB(SisterNB, Person).**

To acquire a clear meaning behind a non-blood relationship, we will set up a little scenario.

> Assume that you have a male friend, or a sworn brother, you will have to refer to his mother as aunt and his father uncle, his brother as brother and his sister as sister.

And that is basically all there is to it. Now onto implementations, since the underlying logic is pretty much the same, we will use **auntNB/2** as an example.

```
auntNB(AuntNB, Person) :-
        non_blood_brother(Person, X),
        mother(AuntNB, X).
```

The **non_blood_brother(Person, X)** will give us a friend, or sworn brother, **X** of **Person**, then we just need to check if **AuntNB** is the mother of **X**.

## Black organization

These are the predicates pertaining to the lore of the universe we are basing this Knowledge Base on, **Case Closed**, also known as **Detective Conan**.

42. **research_APTX_4869(Person).**
    - is_biochemist(Person), was_black_organization(Person).
43. **took_APTX_4869(Person).**
    - shrank(Person).
44. **infiltrator(Person).**
    - was_black_organization(Person), is_fbi_agent(Person).
45. **hunted(Person).**
    - took_APTX_4869(Person), is_alive(Person).
46. **is_police(Person).**
    - is_fbi_agent(Person); is_cia_agent(Person); is_mi6_agent(Person).
47. **police_family(Person).**
    - is_police(Person), parent(X, Person), is_police(X).
48. **scientist_family(Person).**
    - is_biochemist(Person), parent(X, Person), is_biochemist(X).
49. **has_antidote_APTX_4869(Person).**
    - research_APTX_4869(Person), is_alive(Person).

As the above predicates are already extremely self-documenting, we will only give some notes on the last one below.

50. **british_japanese(Person).**

```
british_japanese(Person) :-
        (father(X, Person), is_british(X),
        mother(Y, Person), is_japanese(Y));
        (father(X, Person), is_japanese(X),
        mother(Y, Person), is_british(Y)).
```

For a **Person** to be British Japanese, the **Person** needs to, either have a British father **X** and a Japanese mother **Y**, or a Japanese father **X** and a British mother **Y**.

## 4. Implement logic deductive system in the programming language (30%)

## 5. Reference

### Working with the Prolog tool

- https://en.wikipedia.org/wiki/Prolog
- https://sicstus.sics.se/sicstus/docs/3.12.9/html/sicstus/Compound-Terms.html
- https://youtu.be/SykxWpFwMGs
- https://www.geeksforgeeks.org/prolog-an-introduction/
- https://www.swi-prolog.org/pldoc/man?section=dynpreds

### Build a Knowledge Base with Prolog

- https://www.detectiveconanworld.com/wiki/Main_Page

### Implement logic deductive system in the programming language