**ĐẠI HỌC QUỐC GIA TP. HỒ CHÍ MINH**

**TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN**

**KHOA CÔNG NGHỆ THÔNG TIN**



**BÁO CÁO BÀI TẬP 1**

**SIMPLECODE**

**SCANNER & PARSER**

**Môn học :** Nguyên lý các ngôn ngữ lập trình

**GVHD :** Phạm Trọng Nghĩa

**LỚP :** Cử nhân Tài năng – Khóa 2015

**NGƯỜI THỰC HIỆN : 1512002 – Lê Dương Tuấn Anh**

***Tp. Hồ Chí Minh - Tháng 4/2019***

SIMPLE CODE – BÀI TẬP 1

SCANNER & PARSER

C:\Users\tdqua_000\Dropbox\SS-Slides\DeCuong-CDIO\Template CDIO v4.2\Templates\Hinh anh\LogoTruong.png

Khoa Công nghệ Thông tin

Đại học Khoa học Tự nhiên, ĐHQG-HCM

Tháng 4/2019

**MỤC LỤC**

[1 Tổng quan & Milestone 4](#_Toc499856999)

[Thông tin nhóm 4](#_Toc499857000)

[Milestone 4](#_Toc499857001)

[2 Nội dung 1 – Thuật giải Robinson 5](#_Toc499857002)

[2.1 Nội dung thuật giải Robinson 5](#_Toc499857003)

[2.2 Mô tả cài đặt thuật toán 6](#_Toc499857004)

[2.3 Testcases demo 7](#_Toc499857005)

[Testcase 1 7](#_Toc499857006)

[Testcase 2 7](#_Toc499857007)

[3 Nội dung 2 – Prolog & Bài toán Logic 8](#_Toc499857008)

[3.1 Prolog & Logic bậc nhất 8](#_Toc499857009)

[Ngôn ngữ Prolog và SWI-Prolog 8](#_Toc499857010)

[Logic bậc nhất 9](#_Toc499857011)

[Lập trình với SWI-Prolog 11](#_Toc499857012)

[Facts trong Prolog 13](#_Toc499857013)

[Rules trong Prolog 14](#_Toc499857014)

[Một số ví dụ minh họa 15](#_Toc499857015)

[Đệ quy trong Prolog 20](#_Toc499857016)

[3.2 Sử dụng SWI-Prolog xây dựng cây phả hệ 24](#_Toc499857017)

[Source Code xây dựng cơ sở tri thức 24](#_Toc499857018)

[Tập câu hỏi cho hệ tri thức 27](#_Toc499857019)

# Tổng quan & Milestone

### Thông tin cá nhân

|  |  |  |
| --- | --- | --- |
| MSSV | Họ tên | Email |
| 1512002 | Lê Dương Tuấn Anh | [1512002@student.hcmus.edu.vn](mailto:1512002@student.hcmus.edu.vn)  [leduongtuananh97@gmail.com](mailto:leduongtuananh97@gmail.com) |

### Công cụ hỗ trợ

* Python 3
* Java Runtime Environment.
* ANTLR3 & ANTLR3 Runtime for Python3

# Nội dung bài tập

## ANTLR – ANother Tool for Language Recognition

**ANTLR** là một trình tạo bộ phân tích cú pháp từ một văn phạm cho trước, được sử dụng bởi rất nhiều lĩnh vực khác nhau, chẳng hạn **công cụ tìm kiếm ở trang Twitter dùng ANTLR để xây dựng luật truy vấn, cho 2 tỉ truy vấn mỗi ngày, hoặc dùng chung với SQL Developer IDE bởi Oracle. NetBeans IDE cũng dùng ANTLR để tạo bộ phân tích cú pháp C, v.v…**

ANTLR có khả năng đọc gần như trong thời gian tuyến tính (linear approximate lookahead), có thể phân tích cú pháp & ngữ nghĩa và phân tích cây cú pháp. ANTLR có thể phân tích cú pháp của văn phạm phi ngữ cảnh, là một dạng **LL-Parser (Top-Down Parser).** Từ phiên bản 4 trở đi, **ANTLR là Adaptive LL-Parser (có khả năng xử lí một số trường hợp đệ quy trái, là một nhược điểm của LL-Parser).**

ANTLR cung cấp rất nhiều thư viện trên nhiều ngôn ngữ khác nhau, trong số đó, nổi tiếng nhất là Java & Python Runtime.

## Cài đặt & sử dụng ANTLR Python Runtime

Sau khi cài đặt Python3 & Java Runtime Environment, ta dùng lệnh pip (được cài đặt kèm Python) để cài đặt antlr như sau:

pip install antlr4-python3-runtime

*Lưu ý: Dùng phiên bản mới nhất là 4.*

Tải file **java đã được biên dịch của ANTLR4** tại <https://www.antlr.org/download/antlr-4.7.2-complete.jar>. Đây được dùng để chuyển đổi từ file văn phạm (\*.g4) sang các nền tảng đích (ở đây Python). Sau khi thực thi bằng Java file này, chương trình tạo ra một số thư viện Python. Thư viện này chứa các class hoặc hàm cần thiết để xử lí văn phạm.

Lệnh để tạo trình phân tích cú pháp, đích là Python3 (có sẵn trong file run.bat):

java -Xmx500M -cp **<antlr\_file>.jar** org.antlr.v4.Tool -Dlanguage=Python3 **<grammar\_file>.g4**

*Trong đó:*

***antlr\_file.jar*** *là file đã được download ở trên.*

***grammar\_file.g4*** *là file chứa văn phạm.*

Sau khi chạy xong lệnh trên, tại thư mục tạo thêm các file chứa tokens, class và hàm tương ứng để có thể hỗ trợ tạo cây cú pháp, phân tích cú pháp và ngữ nghĩa.

## Xây dựng văn phạm từ đề bài

Văn phạm xây dựng cho ANTLR cũng khá tương đồng với đề bài đã cho. Cách xây dựng luật Lexer của ANTLRc có thể xem tại: <https://github.com/antlr/antlr4/blob/master/doc/lexer-rules.md>.

**Một số lưu ý mà cá nhân đã nhận thấy trong quá trình tạo bộ luật cho ANTLR**:

- Có một số luật bị đệ quy trái (left-recursive), trong một số trường hợp ANTLR vẫn không thể xử lí được (mặc dù ANTLR4 đã khắc phục được hầu hết). Trong các trường hợp này cần khử đệ quy trái. Một số luật đệ quy trái được ANTLR mô tả rất rõ tại: <https://github.com/antlr/antlr4/blob/master/doc/left-recursion.md>.

- Một số từ như **type, id** ANTLR đã dùng riêng, vì vậy, trong bài tập đã chuyển thành **DATA\_TYPE & IDENTIFIER.**

- Trong bộ luật có định nghĩa: **mặc định các khoảng trắng, tab character ‘\t’, kí tự xuống hàng ‘\r’, ‘\n’, sẽ được bỏ qua** (*ANTLR hỗ trợ từ khóa skip cho việc bỏ qua)*. Tuy nhiên trong một số trường hợp ta vẫn cần giữ nguyên khoảng cách cho một số từ khóa đặc biệt, chẳng hạn **class <space> Program; int <space> a.** Các trường hợp này cần khai báo **<space>+** *(tức ít nhất có 1 khoảng trắng trong các trường hợp này).*

- Trong một số trường hợp đặc biệt, cá nhân vẫn chưa thể giải quyết hết trường hợp lỗi phân tích nhầm các kí tự trắng, dẫn đến sinh ra lỗi. Việc loại trừ lỗi này cũng khá khó khăn. Việc xây dựng bộ dịch lỗi sẽ nói rõ ở phần sau.

**Bộ luật có tên là SimpleCode, lưu tại SimpleCode.g4.**

## Cách sử dụng & mô tả

Chương trình chạy với cú pháp sau:

python baitap1.py **<type> <input\_file>**

*Trong đó*

***type:*** *0 nếu dùng để phân tích từ vựng và 1 nếu phân tích cú pháp*

***input\_file:*** *file văn bản chứa đoạn code được viết bởi ngôn ngữ SimpleCode.*

Trong file python, ta dùng các thư viện chính: SimpleCodeLexer, SimpleCodeListener, SimpleCodeParser với chức năng như sau:

- **SimpleCodeLexer**: Chứa văn phạm mà máy hiểu được từ file SimpleCode.g4. Khi đưa 1 đoạn code nào đó vào thư viện này, ANTLR sẽ cố gắng **ghép các chuỗi con trong đoạn code đã cho với các luật Lexer và tìm Token tương ứng**. **Các chuỗi con được phân tích sẽ ghép với các Lexer Rule, được khai báo bởi các từ bắt đầu bằng chữ in hoa trong file SimpleCode.g4**, chẳng hạn: *CLASS, DIGIT, ALPHA, IDENTIFIER*.

- **SimpleCodeParser**: Các luật phân tích cú pháp được mô tả bởi các từ bắt đầu bằng chữ thường (thường là camelCase). ANTLR sẽ phân tích 1 câu dựa vào token tương ứng và xem nó gắn được với luật nào. Lúc đó ANTLR sẽ có khả năng nhận dạng được câu đó.

- **SimpleCodeListener**: Đây là điểm mới của ANTLR4. Sau khi quá trình phân tích hoàn thành, ta cần đi từ nút gốc (khởi điểm của code, ở đây là luật program) bằng lệnh **parser.program()**. ANTLR4 hỗ trợ ta các phương thức để có thể “lắng nghe” ANTLR4 đang phân tích tới nút nào của cây phân tích cú pháp, lỗi nếu có, v.v… Do cuối cùng, ANTLR4 sẽ trả về 1 cây phân tích cú pháp, nên ta không cần in quá trình phân tích tại bước này. Tuy nhiên, với type = 1 (phân tích cú pháp và in lỗi nếu có), ta cần thêm vào một lớp ErrorListener (được cung cấp bởi ANTLR4) để có thể theo dõi các lỗi và xuất nếu có. Có nhiều cách để trình phân tích cú pháp nhận được, tuy nhiên nhóm chọn cách “thủ công nhất” là thêm trực tiếp vào biến private của parser:

**parser.\_listeners = [ MyErrorListener() ]**

## Quá trình phân tích từ vựng

Sau khi tạo được cây phân tích cú pháp, ta cần duyệt trên cây này để có thể xuất ra dòng, lexer & token tương ứng. Để việc xuất từ trên xuống dưới đúng theo thứ tự xuất hiện lệnh, ta cần in theo thứ tự **left-mid-right.** Việc này được mô tả trong hàm **flattenTree**:

**def** flattenTree(parent, lexers):

**if** (int(sys.argv[1]) **==** 1):

**return**

**for** i **in** range(parent.getChildCount()):

child **=** parent.getChild(i)

**if** (**not** isinstance(child, ErrorNodeImpl)):

**if** (isinstance(child, TerminalNodeImpl)) **and** (child.getText().strip() **!=** ''):

printOutChildNode(child, lexers)

**else**:

flattenTree(child, lexers)

Hàm chỉ in khi đây là nút lá và đây không phải nút lỗi, đồng thời bỏ qua các lexer nếu đó là lexer chuỗi rỗng. Cách thức in ra thông tin của nút theo như đề cho (chỉ in ra token nếu nó là **'IDENTIFIER','INTLITERAL','CHARLITERAL','STRINGLITERAL','BOOLEANLITERAL'**).

filterList **=** ['IDENTIFIER','INTLITERAL','CHARLITERAL','STRINGLITERAL','BOOLEANLITERAL']

**def** printOutChildNode(child, lexers):

**global** filterList, fWrite

line **=** child.getSymbol().line

lexer **=** child.getText()

token **=** lexers.ruleNames[child.getSymbol().type **-** 1]

**if** (token **in** filterList):

fWrite.write('{0} {1} {2}\n'.format(line, token, lexer))

**else**:

fWrite.write('{0} {1}\n'.format(line, lexer))

## Quá trình phân tích cú pháp

ANTLR4 hỗ trợ khá nhiều loại lỗi khác nhau (<https://www.antlr.org/api/Java/org/antlr/v4/runtime/BaseErrorListener.html>), đồng thời cho phép người dùng có thể chỉnh sửa cho hợp lí. *Trong bài tập này sẽ chỉ tập trung vào các lỗi cú pháp (syntaxError). Các báo cáo về văn phạm nhập nhằng, không phân tích được với bộ LL-Parser, hoặc ngữ cảnh đặc biệt (ContextSensitivity) sẽ không đề cập đến.*

### Testcase 1

|  |  |
| --- | --- |
| Input.txt | 1512538\_1512002.txt |
| **KB**  ~A|B  C|~D  ~E|F  F  **ENDKB**  E | **E**  ~A | B, C | ~D, ~E | F, F, ~E  **FALSE** |

### Testcase 2

|  |  |
| --- | --- |
| Input.txt | 1512538\_1512002.txt |
| **KB**  ~A|B  C|~D  H|~B|~A  ~E|F  F|E  A|C|~E  F|G|~H  ~G|E|F  **ENDKB**  E&A | **E&A**  ~A | B, C | ~D, H | ~B | ~A, ~E | F, F | E, A | C | ~E, F | G | ~H, ~G | E | F, ~E | ~A  C | ~D, ~E | F, F | E, A | C | ~E, F | G | ~H, ~G | E | F, ~E | ~A, H | ~A | ~A  C | ~D, A | C | ~E, F | G | ~H, ~G | E | F, ~E | ~A, H | ~A | ~A, F | F  C | ~D, F | G | ~H, ~E | ~A, H | ~A | ~A, F | F, ~G | F | A | C  C | ~D, ~E | ~A, F | F, ~G | F | A | C, ~A | ~A | F | G  C | ~D, F | F, ~A | ~A | F | G, ~G | F | C | ~E  C | ~D, F | F, F | C | ~E | ~A | ~A | F  **FALSE** |

# Nội dung 2 – Prolog & Bài toán Logic

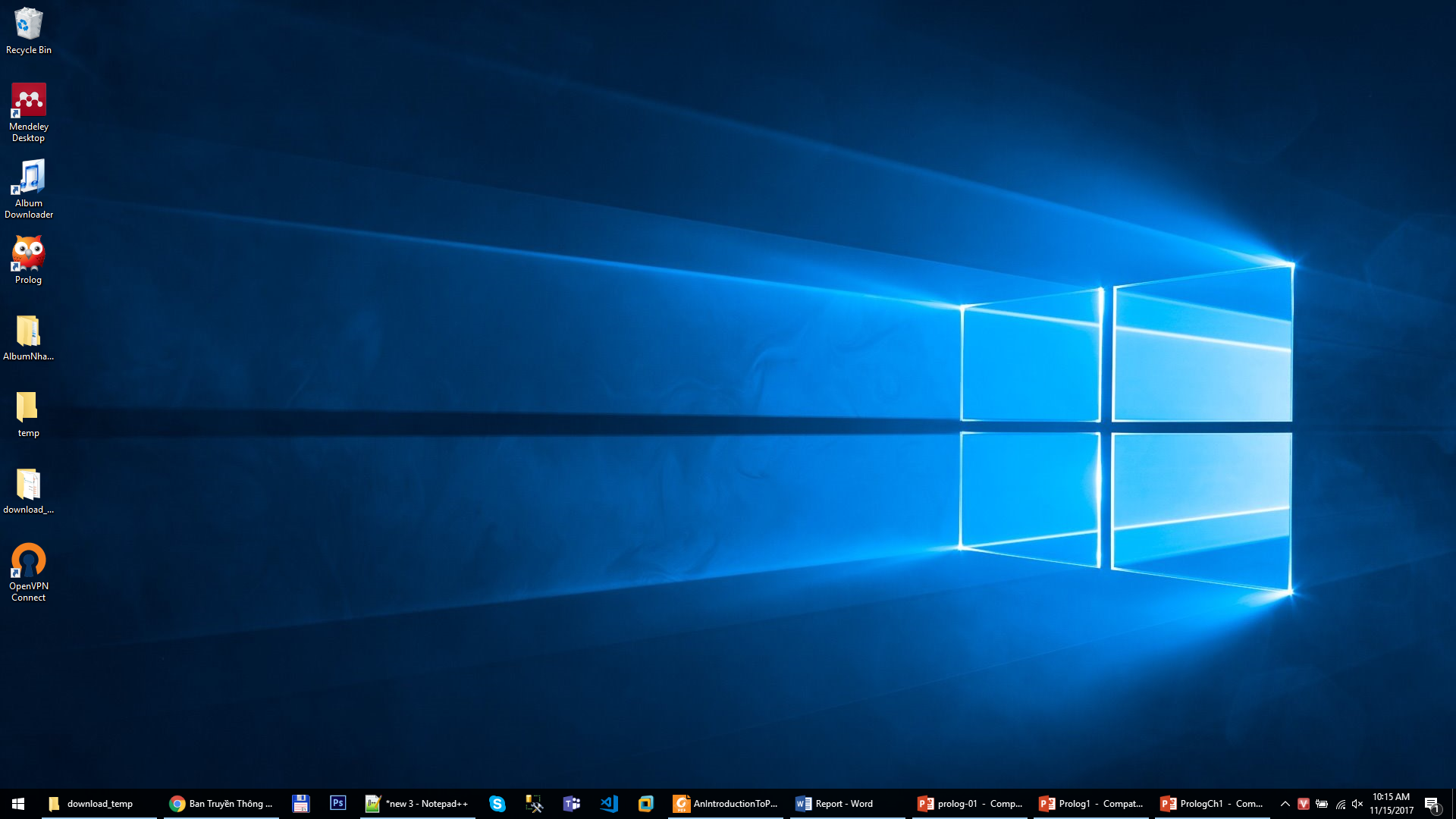
## Prolog & Logic bậc nhất

### Ngôn ngữ Prolog và SWI-Prolog

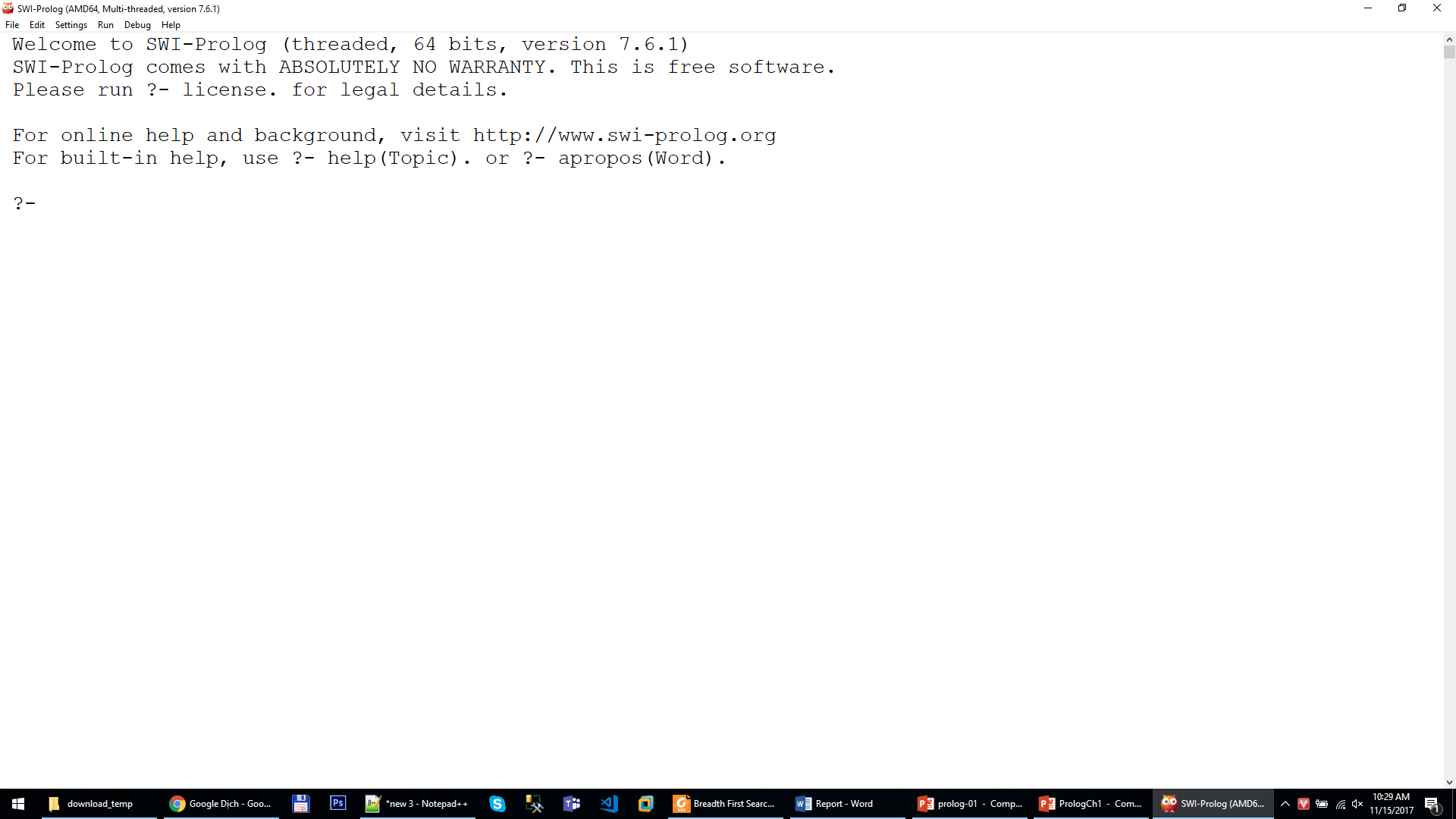
**Prolog** *(viết tắt của “Programming in Logic”)* được thiết kế vào những năm 1970 bởi **Alain Colmerauer và đồng nghiệp** với một ý tưởng mới ở thời điểm hiện tại: **Sử dụng suy luận Logic để diễn tả tri thức và lập trình**. Rõ hơn, **Prolog** sử dụng các ***“kiến thức thực tế” (facts)*** và ***“quy tắc” (rules)*** để ***rút ra cấu trúc của nó*** từ các nghiên cứu lý thuyết của các nhà Logic học trước đó như *Herbrand, Robinson.*

Lúc đầu Prolog được viết nhằm hỗ trợ việc *xử lý ngôn ngữ tự nhiên (Natural Language Processing),* nhưng do sự đơn giản, nó đã phổ biến trên nhiều lĩnh vực: **Formal logic and associated forms of programming (Hình thức hóa Logic và kiểu mẫu lập trình), Reasoning modeling (Lập mô hình suy luận), Database programming (Lập trình cơ sở dữ liệu), Planning (Bài toán lập lịch), v.v…**

Trong đồ án này, để lập trình Prolog, chúng ta sử dụng **SWI-Prolog** (<http://www.swi-prolog.org>).



*Biểu tượng phần mềm SWI-Prolog*



*Giao diện phần mềm SWI-Prolog*

### Logic bậc nhất

**Logic bậc nhất** *(hay còn gọi là logic vị từ)* cho phép chúng ta ***nhắc đến các đối tượng, tính chất và quan hệ giữa chúng***. Trong đó, **vị từ (predicate)** giống như **một hàm (function)** trả về giá trị **đúng/sai (boolean)**. Vị từ rất hữu ích trong việc **đại diện các tham số một cách hình tượng hóa**. Ngoài ra, Logic bậc nhất có thể biểu diễn một số câu mà Logic mệnh đề **không thể biểu diễn được**, chẳng hạn:

*A là cha của B nên A phải là giới tính Nam.*

Vị từ đôi lúc cũng là một phần trong một câu nào đó, chẳng hạn:

**P(X): “X đem theo áo mưa”** /\*P(X) là một hàm có tham số là X\*/

**r: Ngoài trời đang mưa lớn** /\*r là một sự thật hiển nhiên\*/

=> **r** 🡪 **P(X)**: “**Nếu ngoài trời đang mưa lớn**, **thì X đem theo áo mưa**”.

Nếu vị từ thể hiện tính chất hoặc mối quan hệ giữa các đối tượng, thì **lượng từ (Quantifiers)** giúp phát biểu **số lượng đối tượng**.

Ví dụ: *Đôi lúc ta cần diễn tả một số mệnh đề* ***liên quan đến tất cả các đối tượng*** *mà chúng ta đang có. Chẳng hạn:* ***Có tồn tại ai trong lớp này là Nam hay không****?*

Có 2 lượng từ trong Logic bậc nhất bao gồm:

1. Lượng từ “tồn tại” (Existential quantifier):

“Tồn tại một người X đem theo áo mưa”:

2. Lượng từ “với mọi” (Universal quantifier):

Giả sử có các lượng từ:

**DVCV(x)** = **“x là động vật có vú”**.

**CAHEO(x) = “x là cá heo”.**

**F là tất cả những loài động vật sống dưới biển.**

**(**: **Tất cả những loài sống dưới biển** mà **là cá heo** thì **đều là động vật có vú**!

Từ khái niệm “kéo theo” của Toán rời rạc, Logic bậc nhất hỗ trợ các chứng minh và suy dẫn dựa trên các *phương pháp hợp giải mệnh đề, tam đoạn luận, thuật giải Robinson, v.v…*

Vấn đề đặt ra: **Cần biến đổi Logic bậc nhất thành dạng mệnh đề để có thể khai báo trong Prolog.** Cách để chuyển đổi như sau**:**

**Bước 1:** Loại bỏ bằng cách dùng tương đương:

**Bước 2:** Thu giảm phép NOT (~) bằng các luật Toán rời rạc:

+ Phủ định kép:

+ De Morgan:

+ Lượng từ tương đương:

**Bước 3:** Chuẩn hóa các biến để mỗi lượng từ chỉ ảnh hưởng lên một biến duy nhất:

**Bước 4:** Do lúc này không có sự nhập nhằng giữa các lượng từ, nên có thể chuyển toàn bộ lượng từ & biến qua bên trái (không thay đổi thứ tự.

**Bước 5:** Loại bỏ lượng từ tồn tại:

+ Sử dụng hàm **Skolem**:

+ Ở bước 4 ta không thay đổi thứ tự nhằm mục đích: Các biến của lượng từ “tồn tại” sẽ được thay bằng hàm mới theo những tên biến của lượng từ “với mọi” trước nó.

+ Lúc này, tất cả các lượng từ đều là “với mọi”. Ta bỏ hết tất cả các lượng từ này, xem như mặc định tất cả các biến đều là “với mọi”. Chỉ giữ lại các phép **AND/OR**.

**Bước 6:** Chuyển về hội chuẩn **CNF (Conjunctive Normal Form)**:

+ CNF là một chuỗi các mệnh đề được nối với nhau bằng quan hệ **AND**. Từng mệnh đề được nối với nhau bởi quan hệ **OR**.

+ Sử dụng phép phân phối giữa AND và OR:

**Bước 7:** Tách riêng từ mệnh đề ra để khai báo trong Prolog.

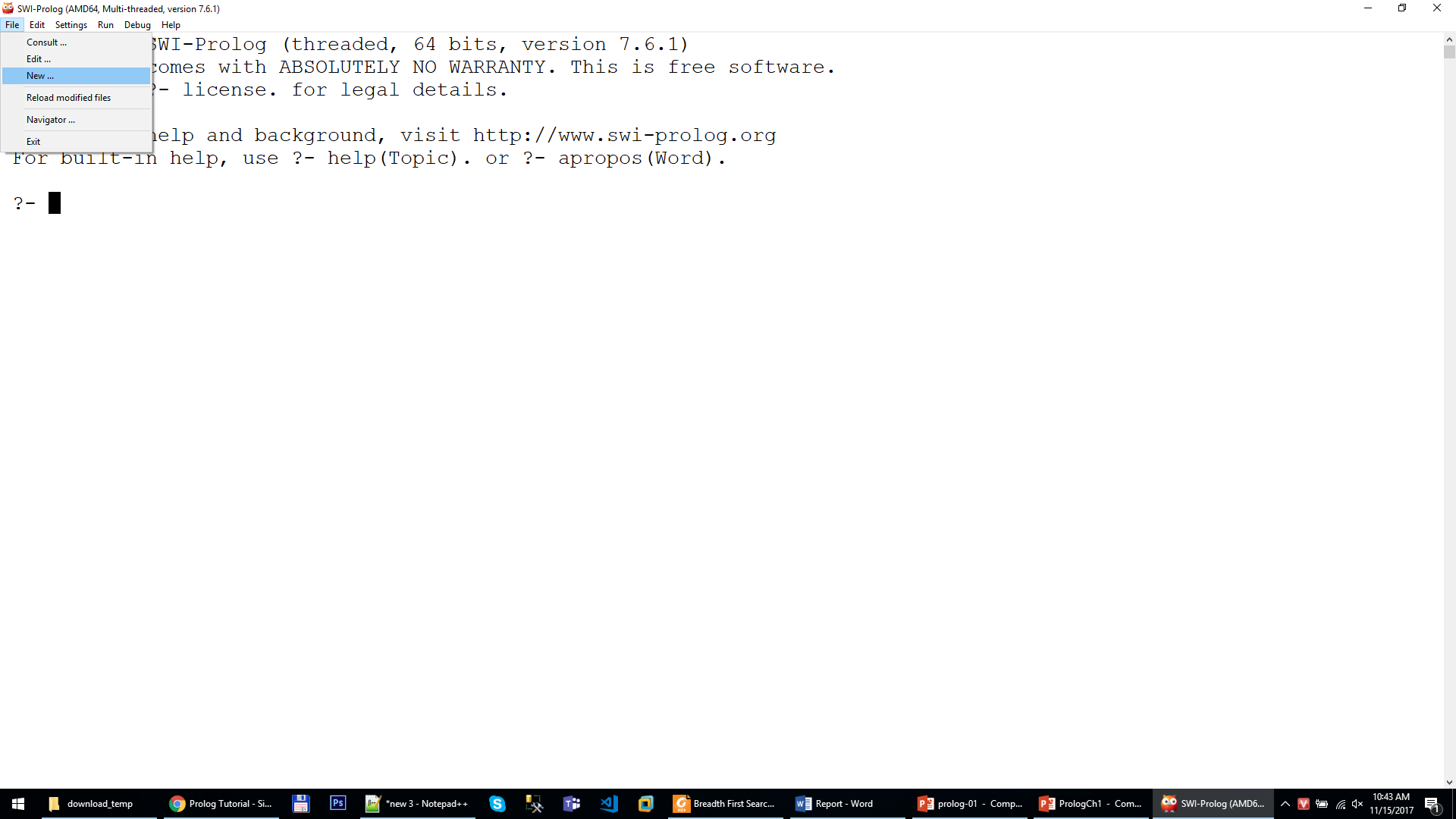
Tuy nhiên, để đơn giản, các ví dụ trong phần tiếp theo sẽ không cần chứng minh, mà tự suy ra theo những thực tế gần gũi hơn.

### Lập trình với SWI-Prolog

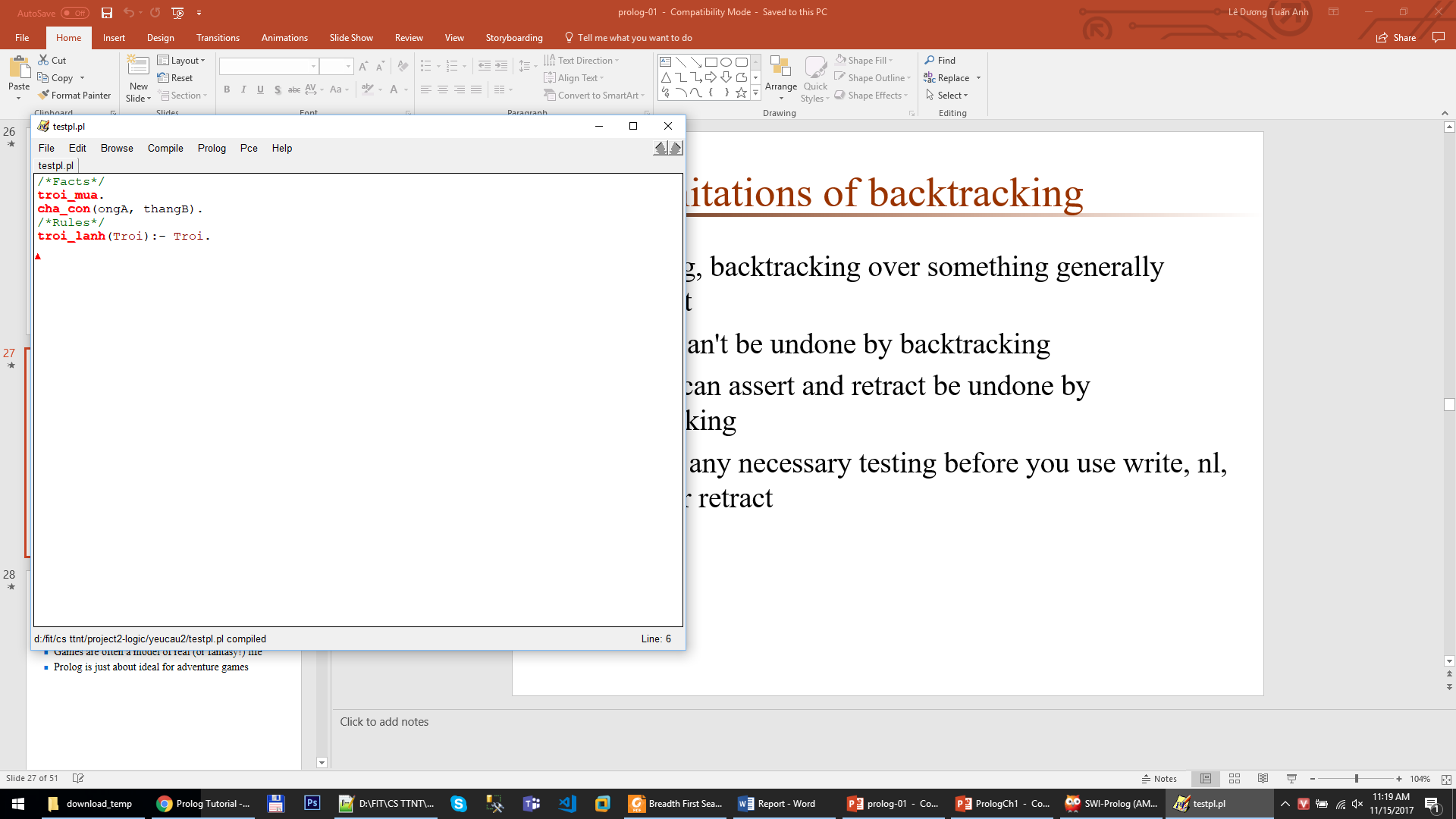
Đối với SWI-Prolog gồm 2 phần chính:

* **Phần lập trình:**+ Tạo 1 file **\*.pl** bằng cách vào **File -> New… -> Lưu file**.

*(hoặc* ***File -> Edit*** *để mở file đã có sẵn…)*

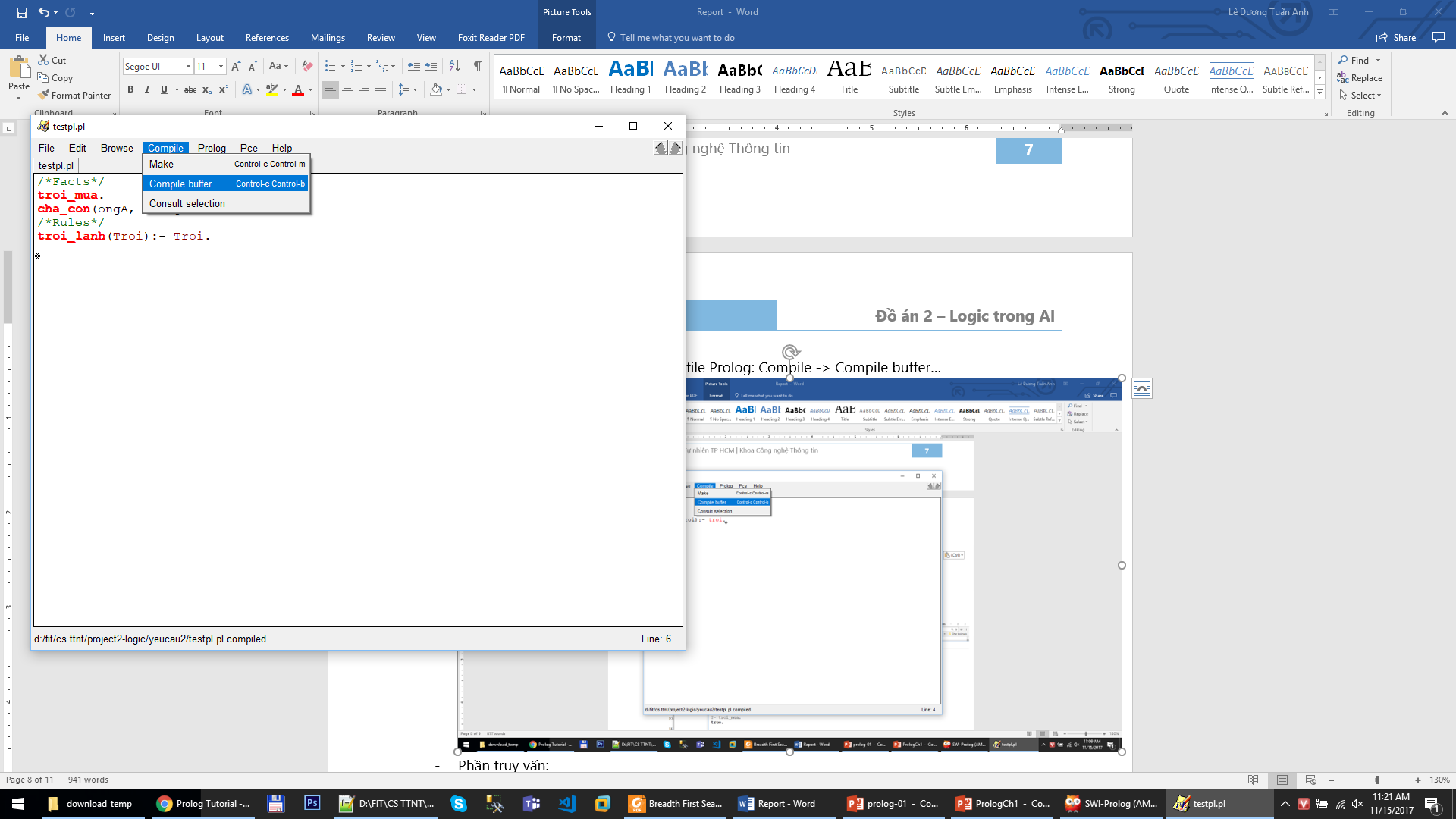


+ Tất cả các ***Facts/Rules*** cần được khai báo trong file này:



*Giao diện editor của SWI-Prolog*

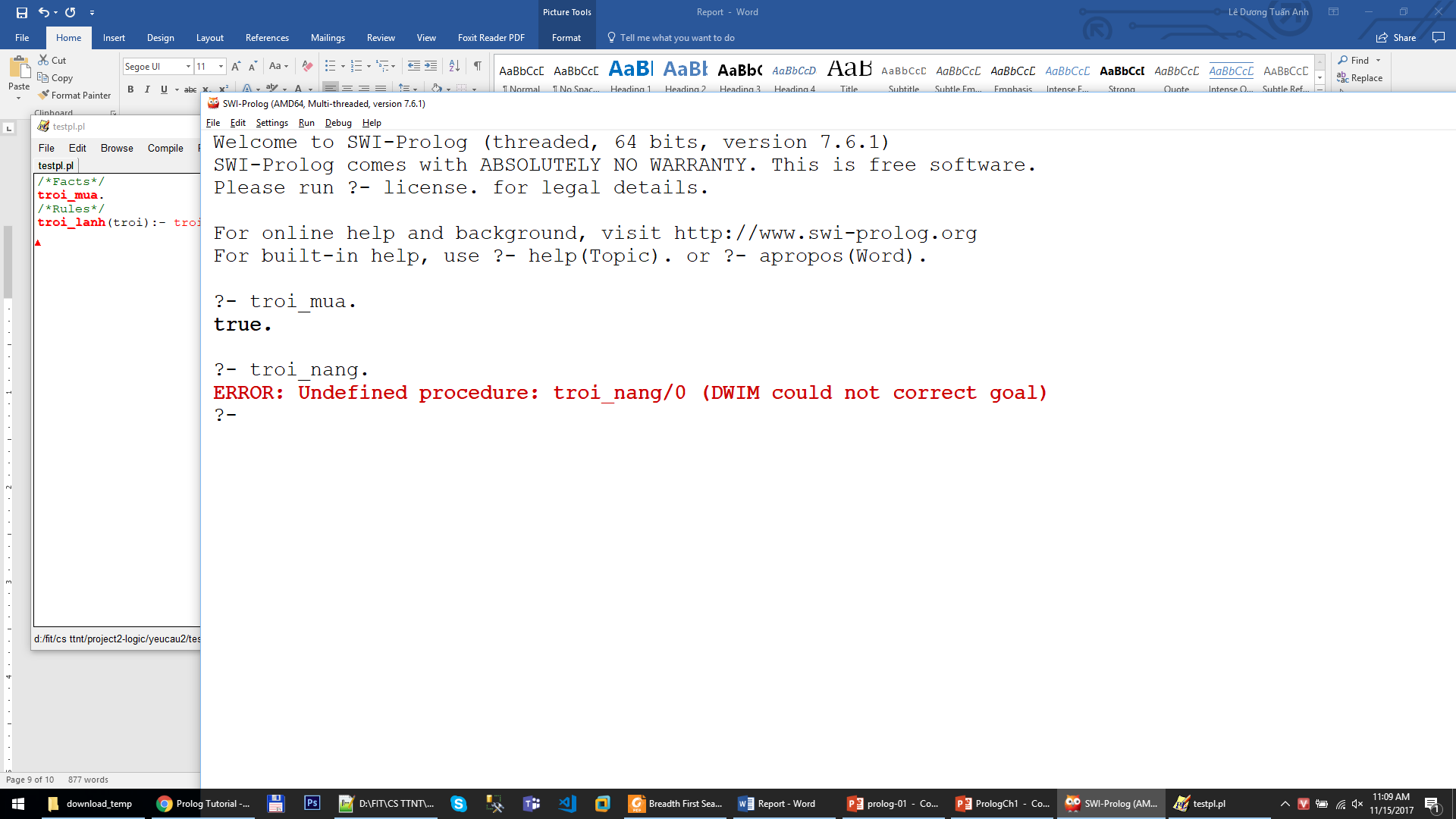
+ Sau khi nhập xong, cần biên dịch file Prolog: **Compile -> Compile buffer…**

****

* **Phần truy vấn:**

+ **?-** là phần bắt đầu của *câu lệnh truy vấn Prolog.*

+ Để truy vấn ta cần nhập đối tượng/luật mà chúng ta đã khai báo. Chẳng hạn:



*Lưu ý: Trong SWI-Prolog nếu ta hỏi một facts mà chưa được khai báo, SWI sẽ tự hiểu đó là một Rules (Procedure) và báo lỗi Undefined.*

### Facts trong Prolog

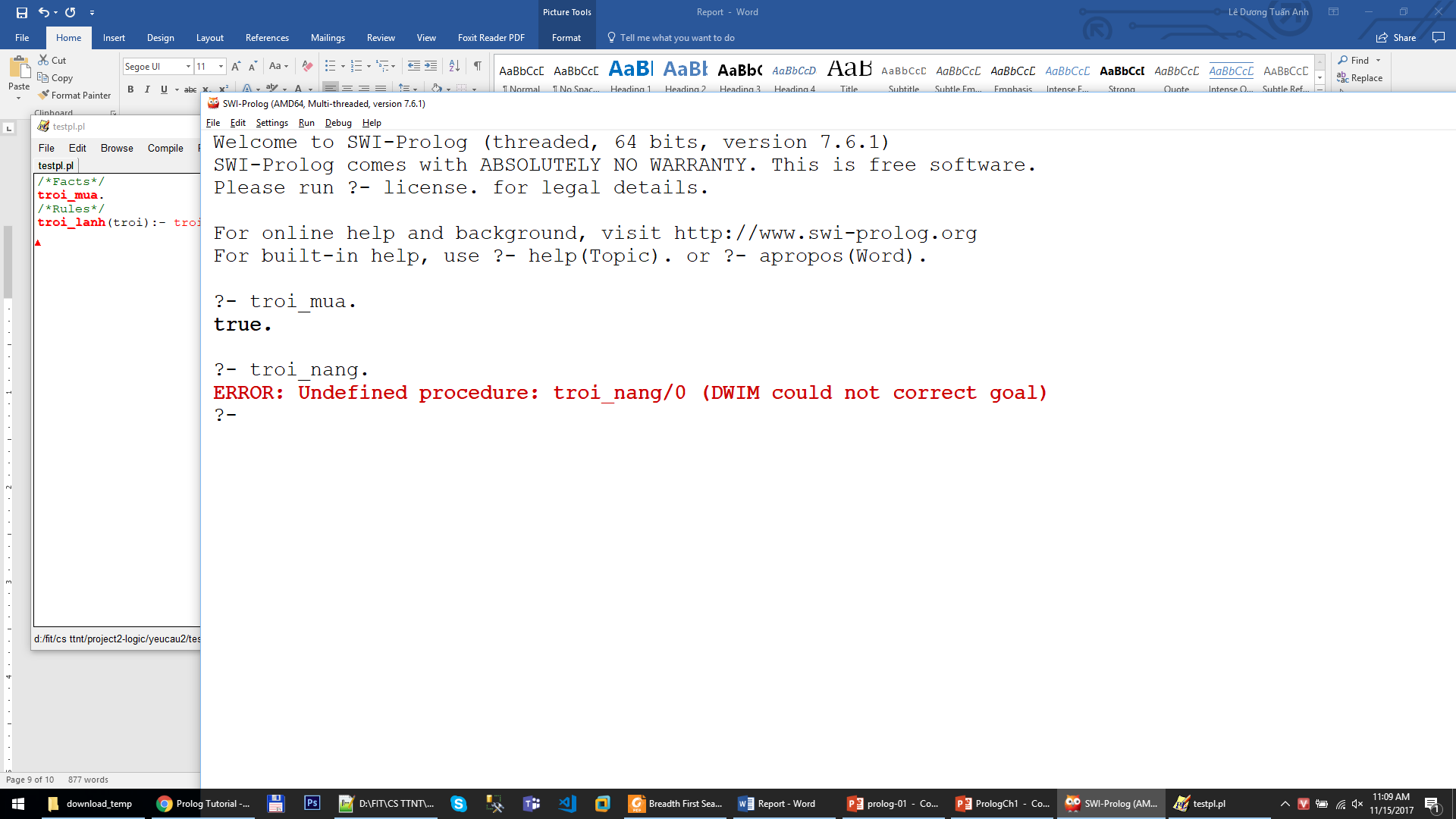
Trong Prolog, chúng ta cần khai báo các “facts” để mô tả các phát biểu. Facts có thể là một đối tượng cụ thể hoặc có thể là mối quan hệ giữa các đối tượng.

Chẳng hạn, chúng ta mô tả đối tượng *“trời mưa”* để diễn tả trời đang mưa bằng cách nhập:

**troi\_mua.**

*Lưu ý: cần có dấu chấm (.) ở cuối câu.*

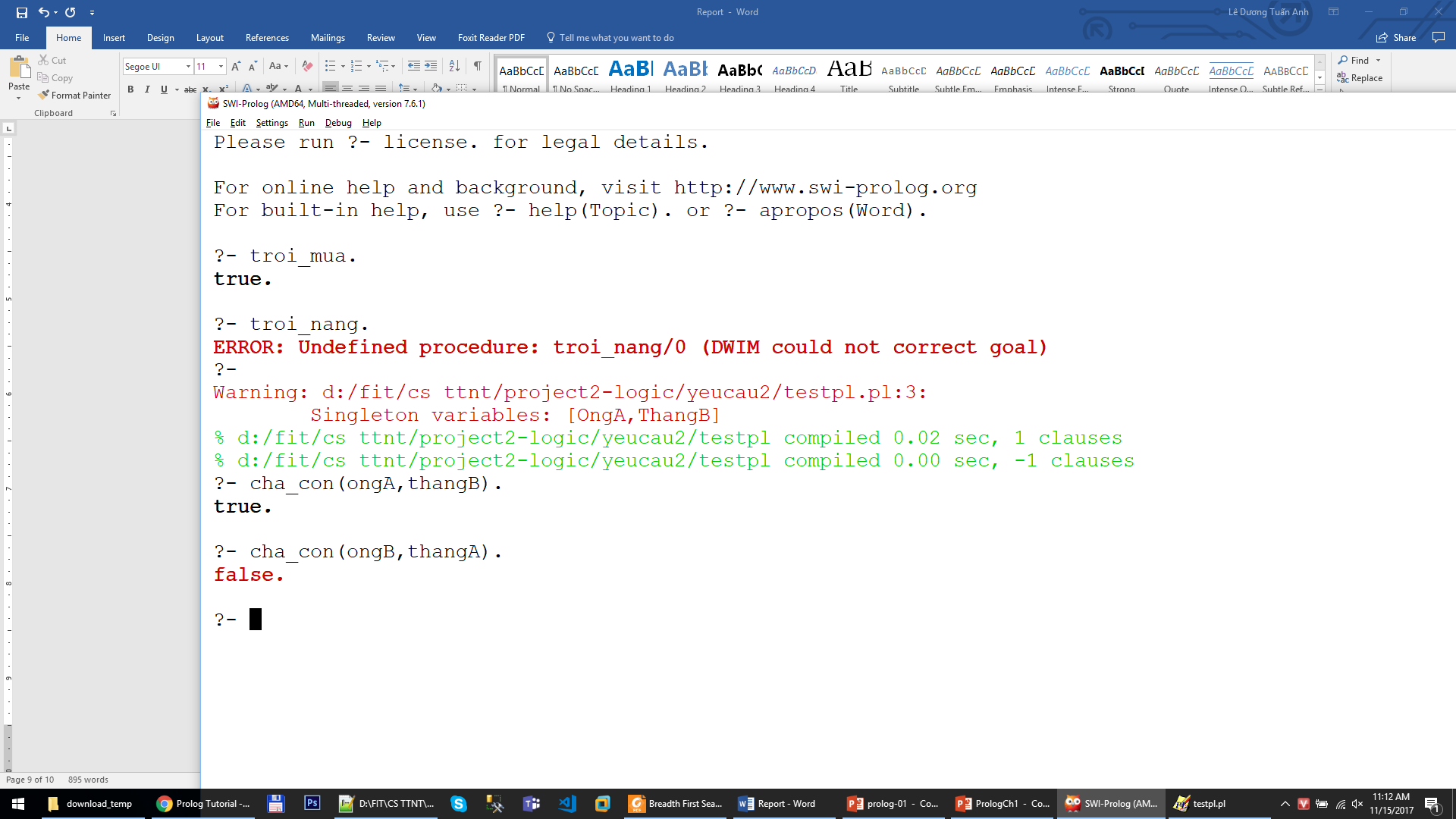
Sau đó ta có thể hỏi bằng câu truy vấn: **?- troi\_mua**. SWI sẽ trả về là **true.**



Nếu Facts được định nghĩa có tham số, chẳng hạn:

**cha\_con(ongA,thangB).** */\*Ông A là cha của thằng B\*/*

Thì ta có thể hỏi nó như một hàm Logic:



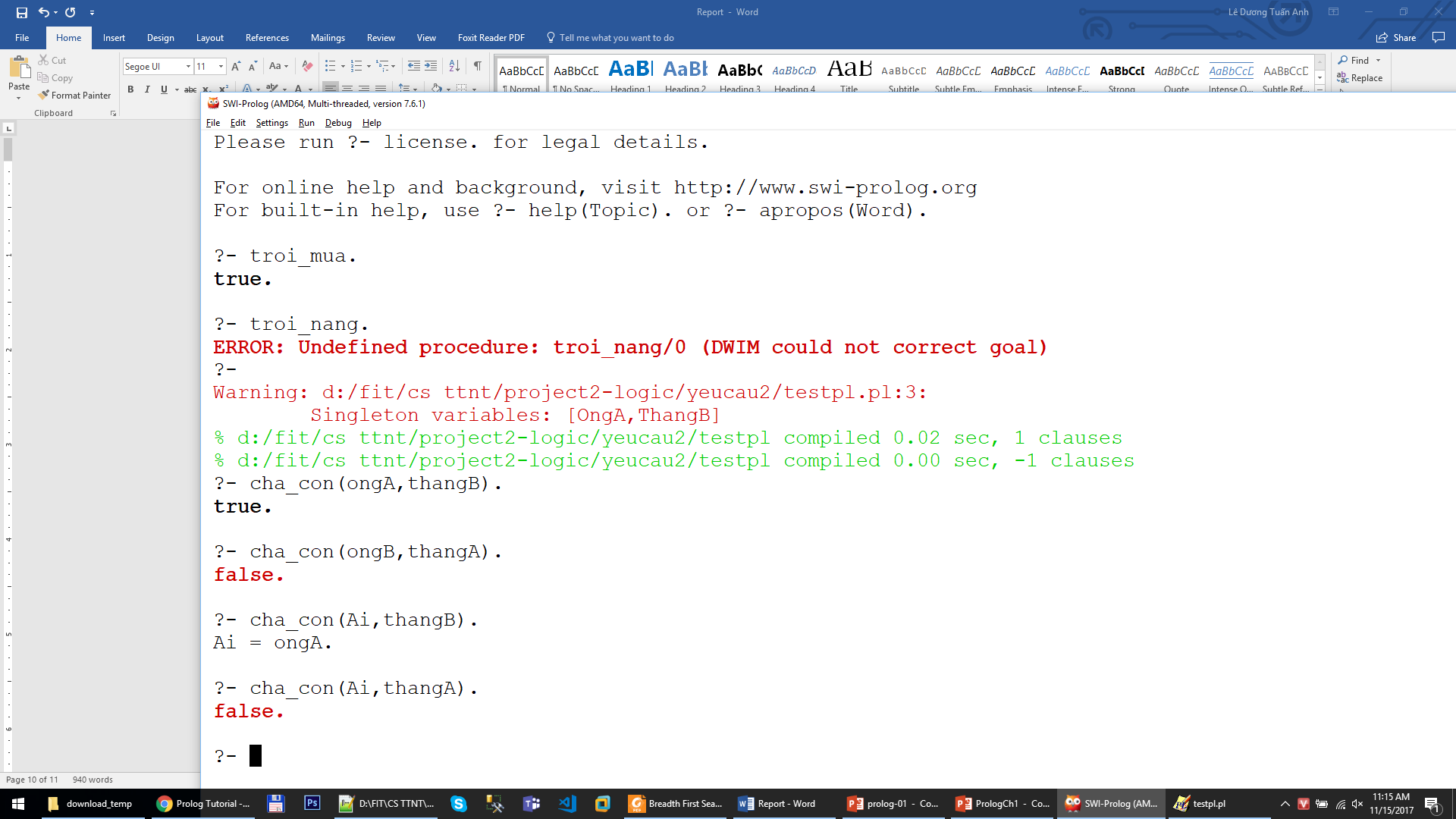
Đối với các Facts có tham số, ta có thể hỏi bằng cách dùng biến đại diện. Chẳng hạn:

**cha\_con(Ai, thangB).***/\* Ai là cha thằng B\*/*

SWI sẽ trả về:

**Ai = ongA.**

Nếu hỏi không có trong dữ liệu, sẽ trả về false:



*Lưu ý: Các biến bắt buộc bắt đầu bằng chữ cái in hoa. Các Facts/Rules bắt đầu bắt chữ cái thường*

### Rules trong Prolog

Trong Prolog, “Rules” được dùng để khai báo những điều kiện, ràng buộc giữa những đối tượng với nhau. Mỗi luật có thể có nhiều biến thể, gọi là mệnh đề.

Phép AND được biểu thị bởi dấu phẩy “,”. Phép OR được biểu thị bởi dấu chấm phẩy “;”.

Chẳng hạn, để diễn tả rằng *Tất cả những người cha đều là con trai*

**dan\_ong(X):- cha(X).**

Như vậy, nếu **cha(X)** đúng thì **dan\_ong(X)** đúng.



Ngoài ra, có một số luật cần phải có 2 điều kiện trở lên. Chẳng hạn: ***X là mẹ ruột của Y*** *khi* ***X là phụ huynh của Y*** *và* ***X là phụ nữ*.**

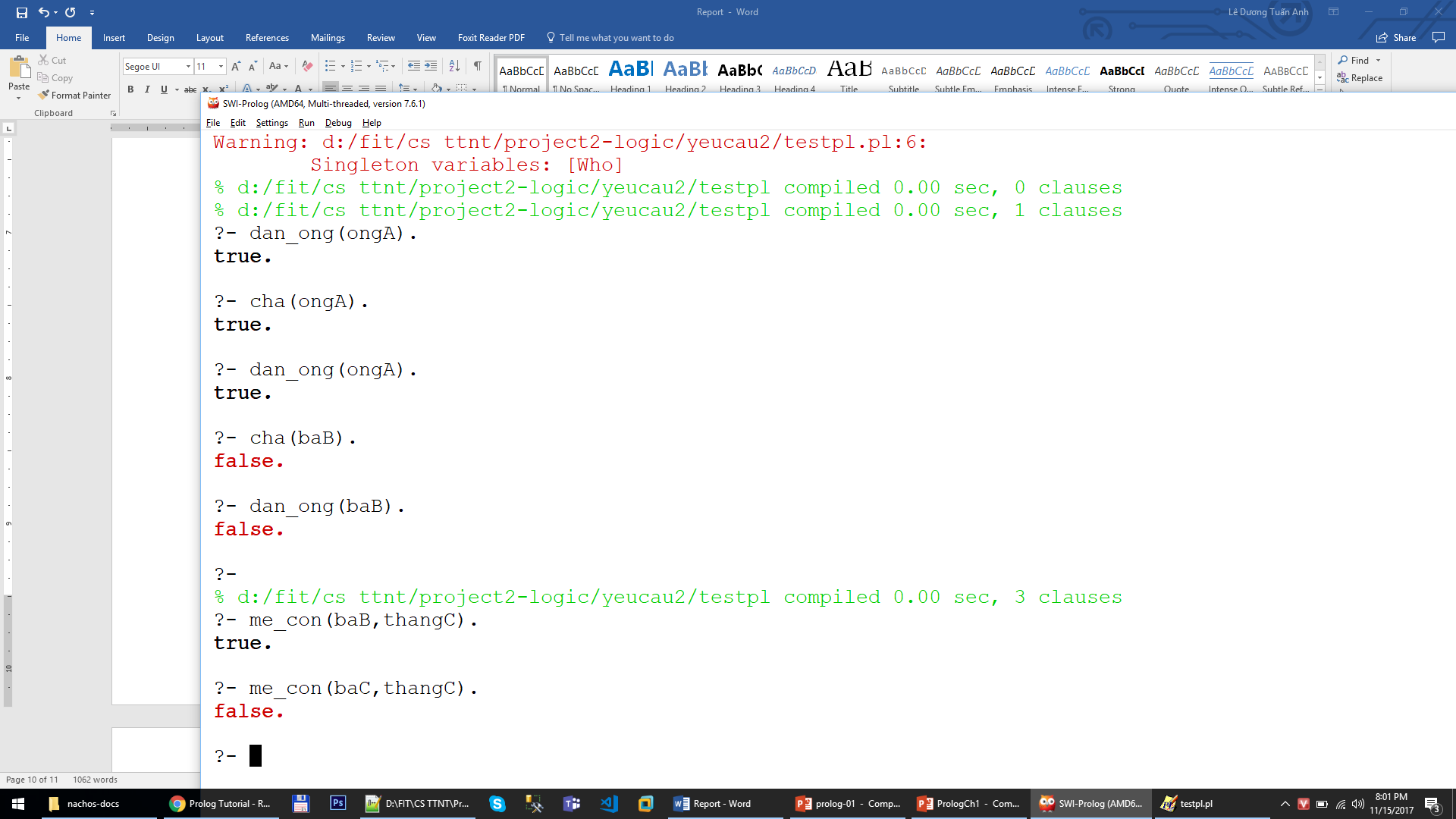
**/\*Facts\*/**

phu\_nu(baB).

phu\_huynh(baB, thangC).

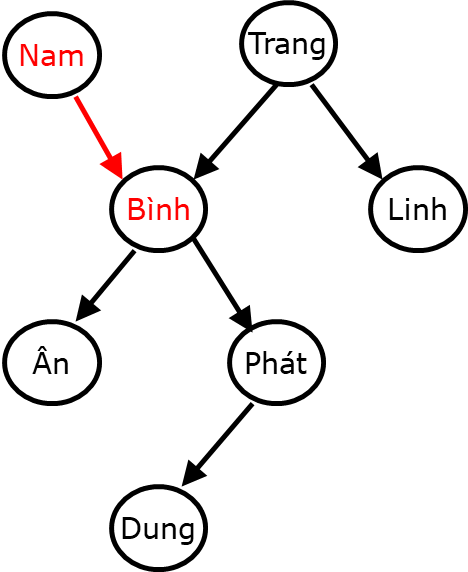
**/\*Rule\*/**

me\_con(X,Y):-phu\_huynh(X,Y), phu\_nu(X).



### Một số ví dụ minh họa

Cho cây phả hệ của một gia đình như sau:



**Các lệnh để định nghĩa các Facts theo Problog:**

phu\_huynh(nam, binh). **/\*phu\_huynh(X, Y): X là phụ huynh của Y**

phu\_huynh(trang, binh).

phu\_huynh(trang, linh).

phu\_huynh(binh, an).

phu\_huynh(binh, phat).

phu\_huynh(phat, dung).

**Một số câu truy vấn mẫu:**

*+ X có phải là phụ huynh của Y hay không?* ?- parent(X, Y).

?- phu\_huynh(binh, phat).

?- phu\_huynh(binh, nam).

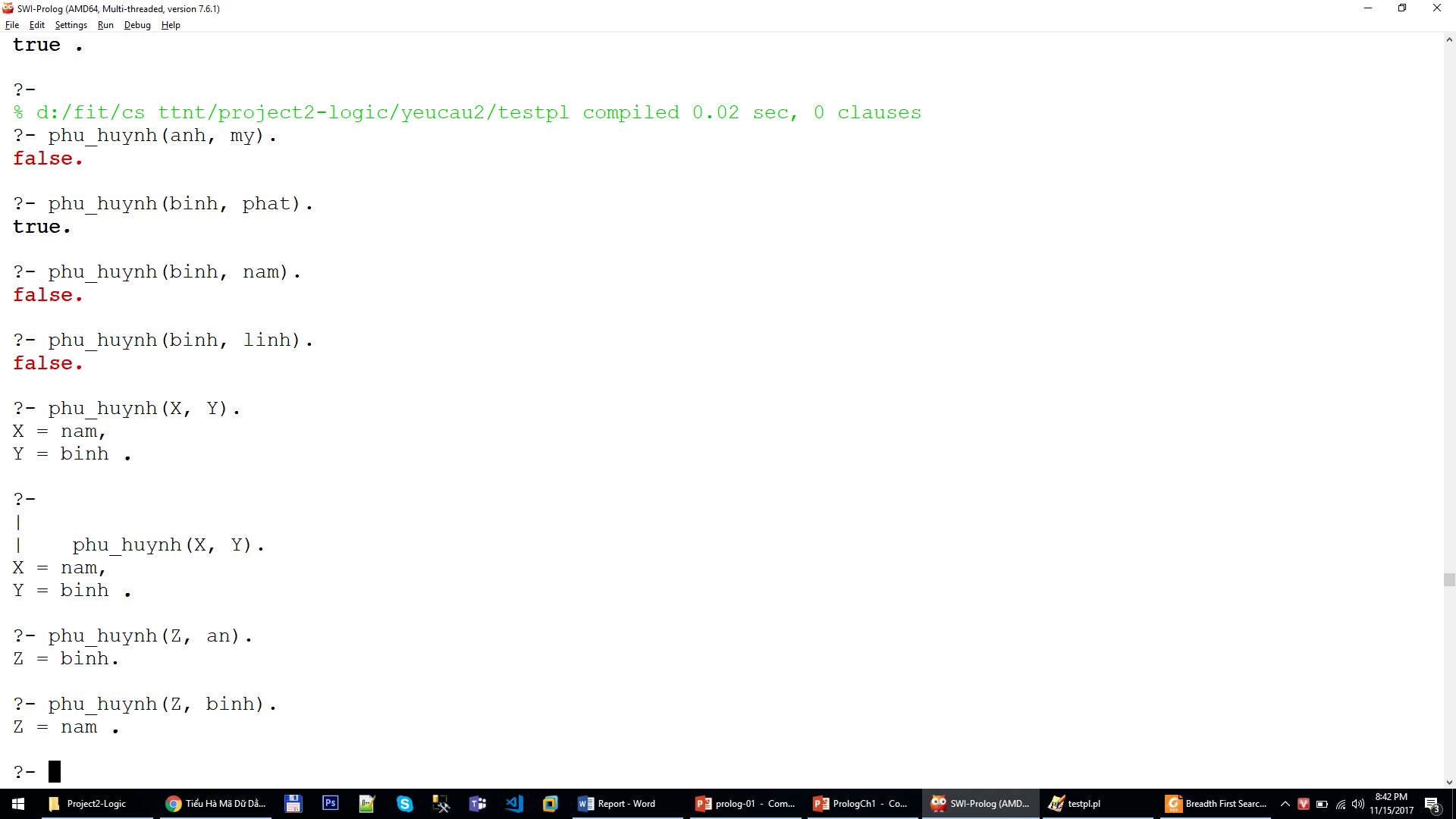
?- phu\_huynh(binh, linh).



*+ Ai là phụ huynh của X?* ?- parent(Z, X).

?- phu\_huynh(Z, an).

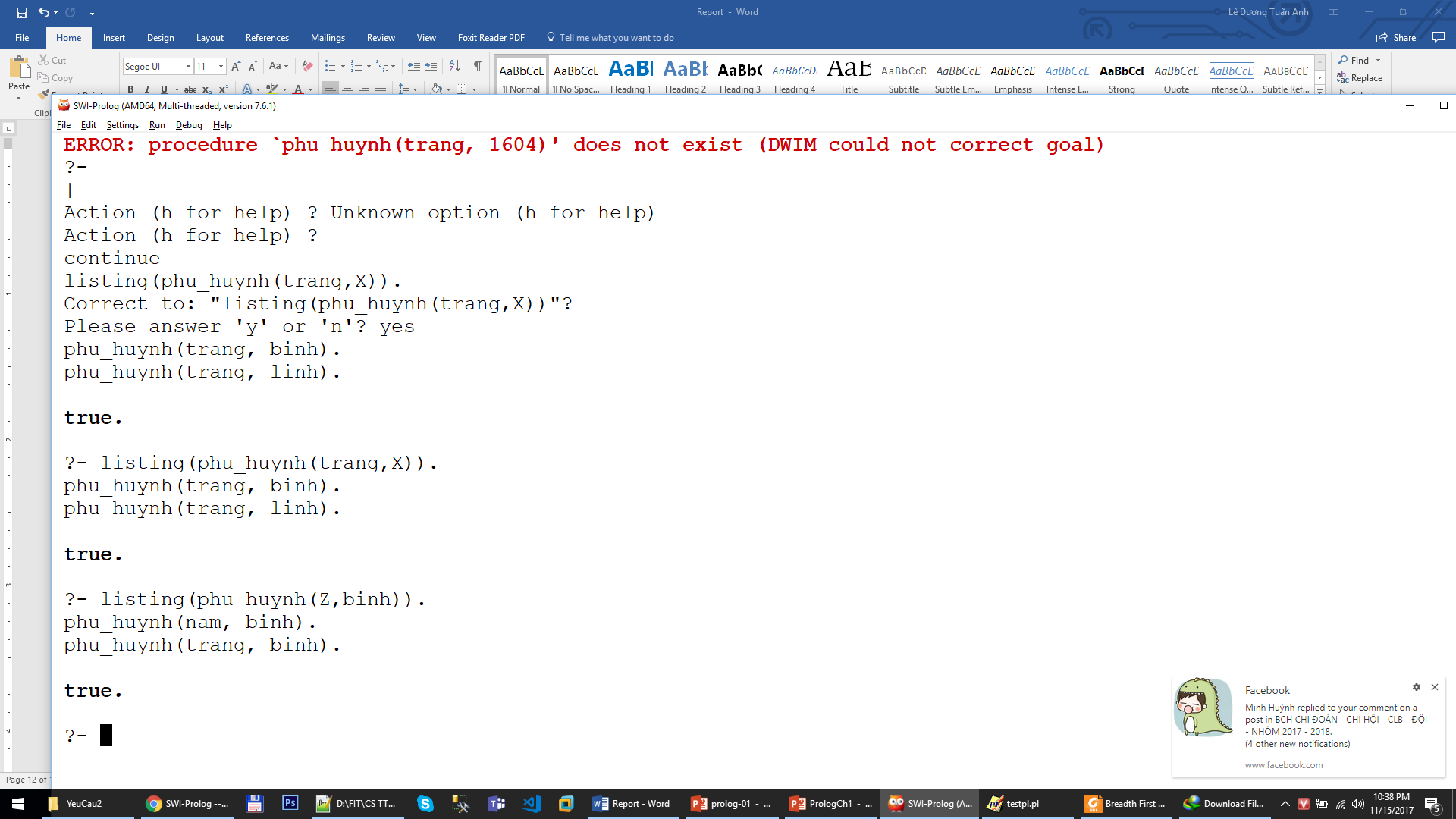
?- phu\_huynh(Z, binh).

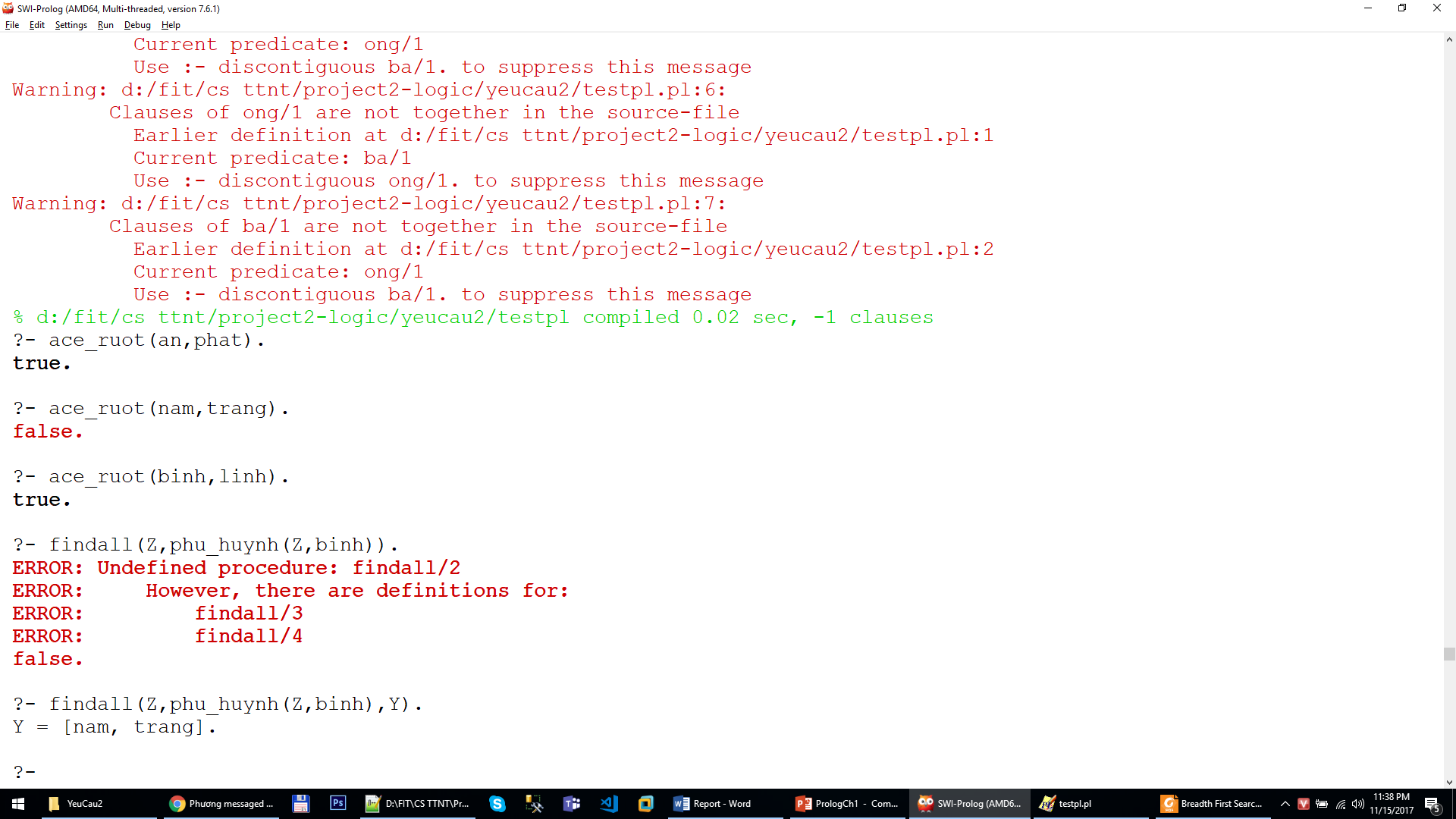


*Lưu ý:*

*Trong trường hợp có nhiều đáp án, nếu chỉ truy vấn như trên* ***Prolog sẽ chỉ in ra một phương án*** *là* ***dữ liệu đầu tiên Prolog tìm thấy trong hệ tri thức của nó****.*

*Để in được nhiều đáp án, dùng lệnh* ***listing()****,* ***setof(X, …..[X}, Y)*** *hoặc* ***findall(X, ….[X], Y) – Y là danh sách chứa tất cả các X thỏa mãn*** *để liệt kê, chẳng hạng:*

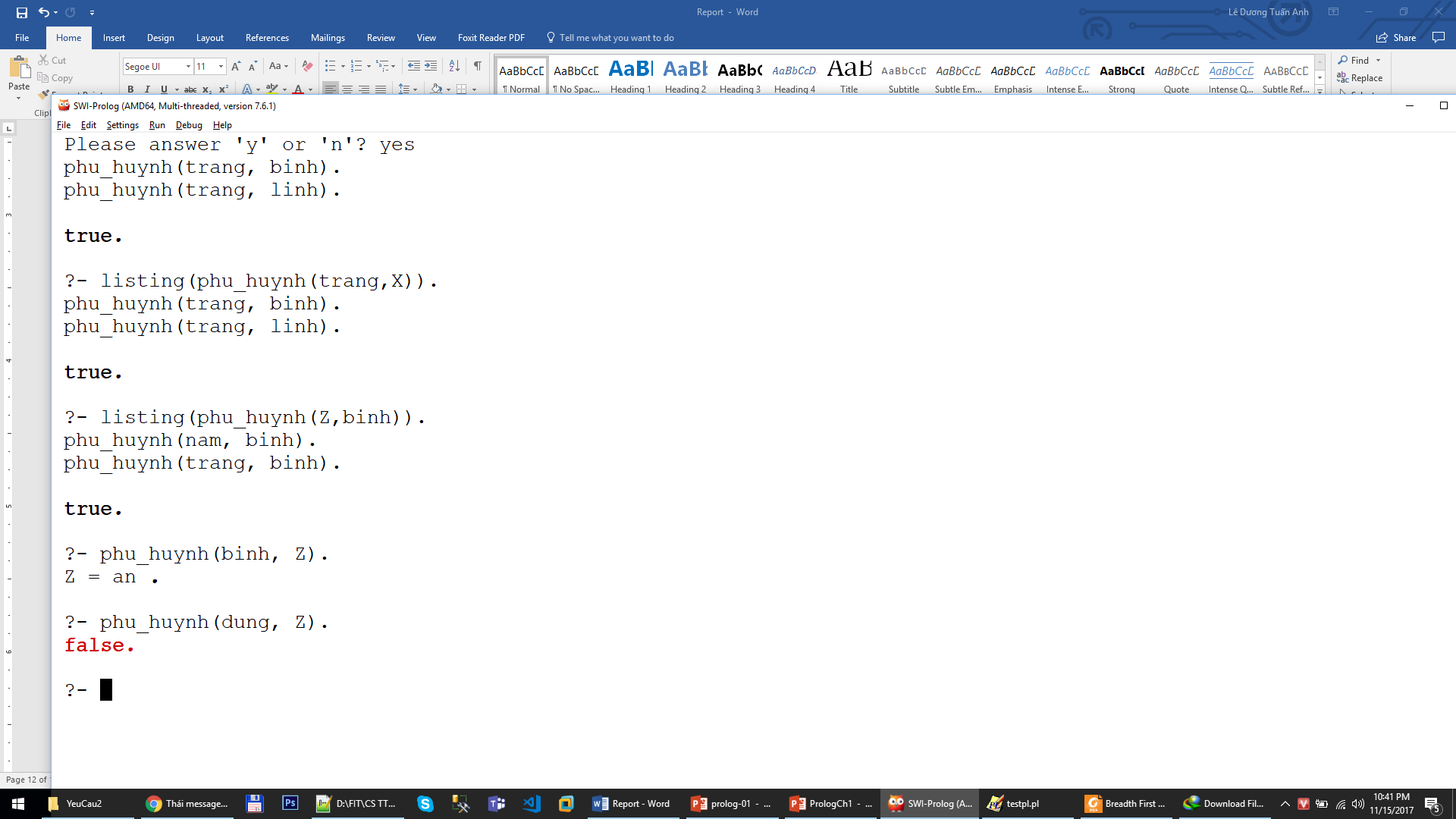




*+ X là phụ huynh của ai?* ?- phu\_huynh(X, Z).

+ phu\_huynh(binh, Z).

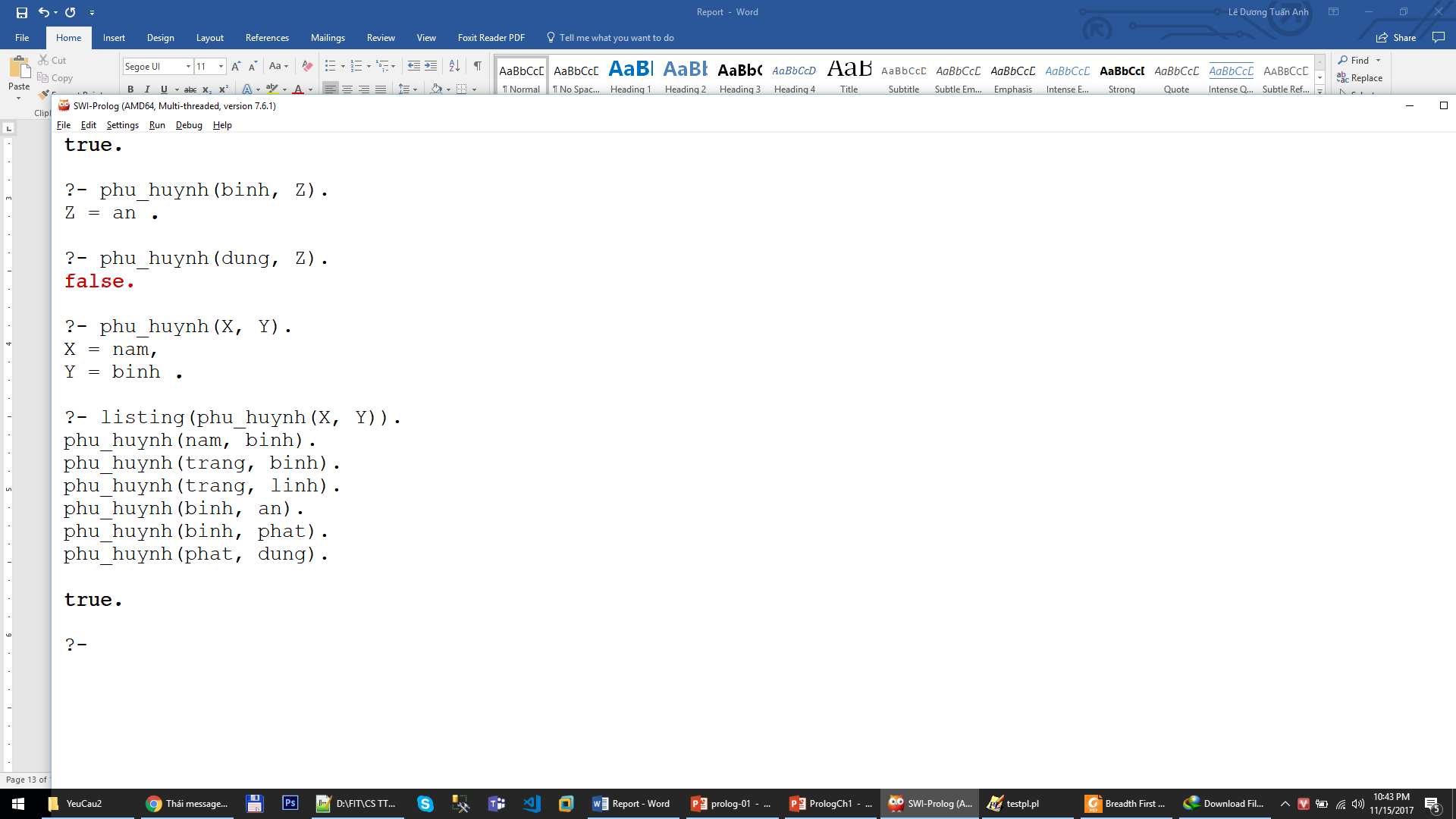
+ phu\_huynh(dung, Z).



*+ Ai là phụ huynh của ai?* ?- phu\_huynh(X, Y).

?- phu\_huynh(X, Y).

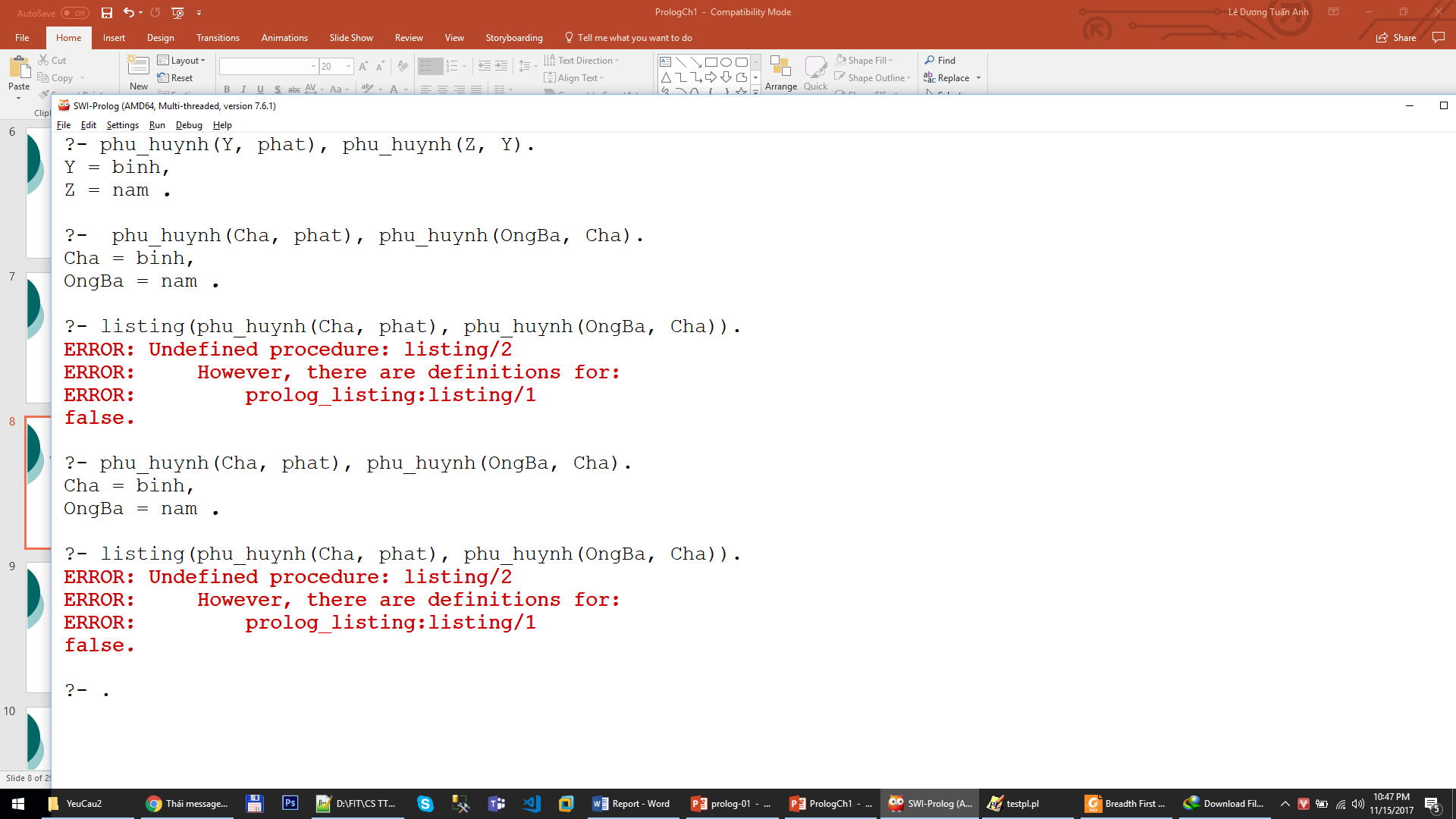
?- listing(phu\_huynh(X, Y)).



*+ Ai là ông/bà của X?* ?- phu\_huynh(Y, X), phu\_huynh(Z, Y).

?- phu\_huynh(Cha, phat), phu\_huynh(OngBa, Cha).

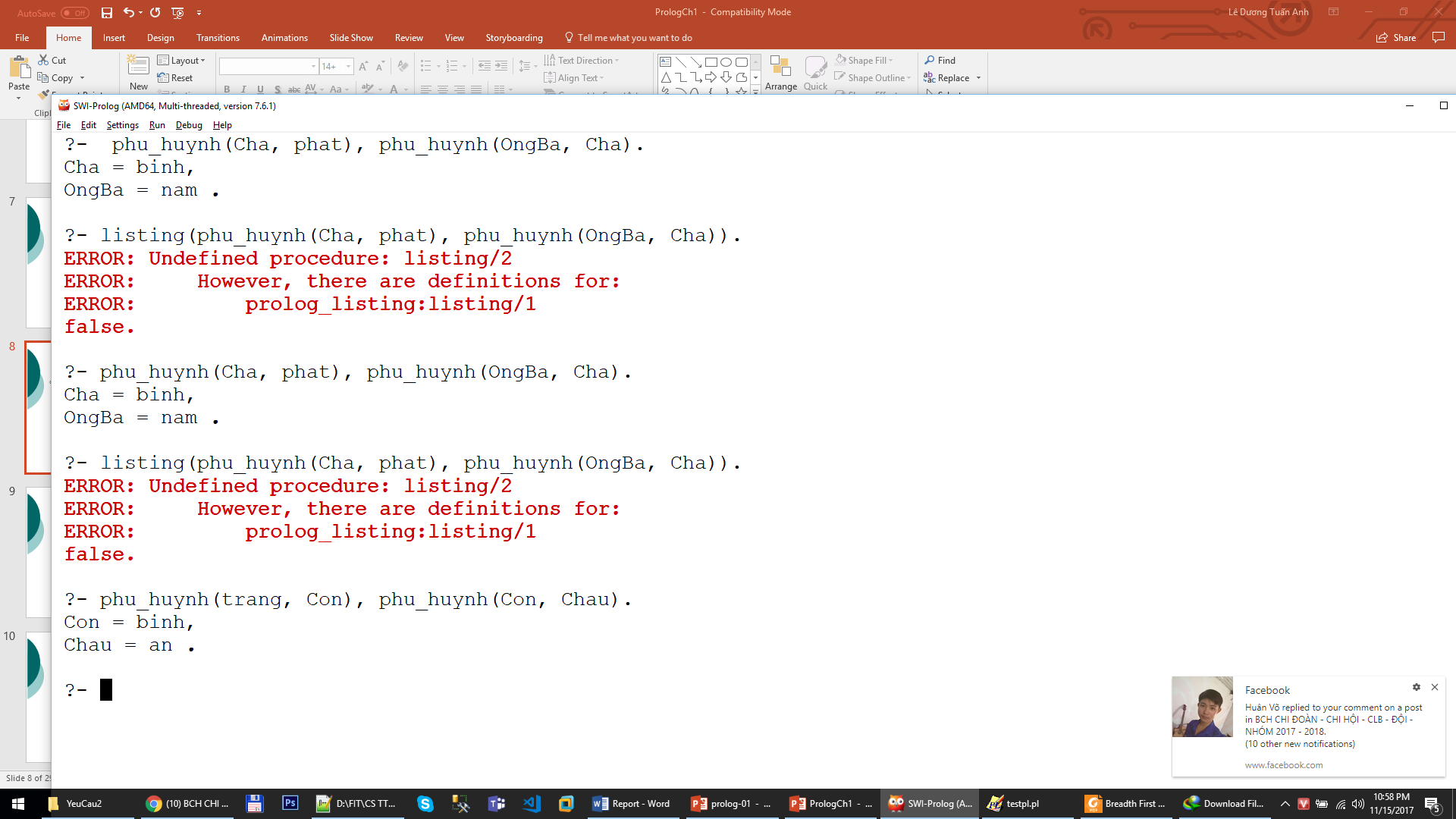
/\*Tìm ông/bà của Phát\*/



*+ Ai là cháu của X?* ?- phu\_huynh(X, Con), phu\_huynh(Con, Cháu).

?- phu\_huynh(trang, Con), phu\_huynh(Con, Chau).

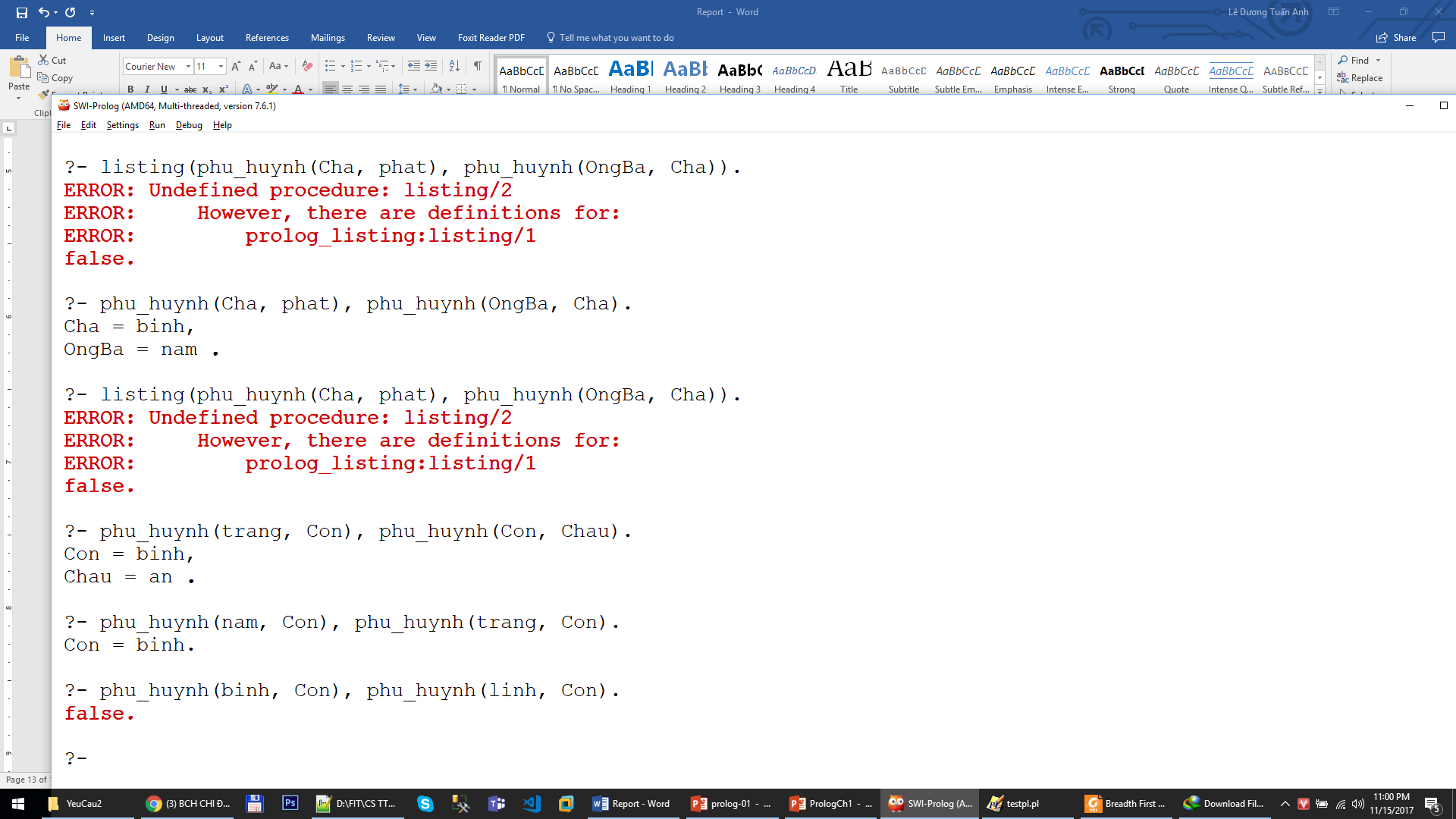
/\*Tìm cháu của Trang\*/



*+ X và Y có con với nhau hay không?* ?- phu\_huynh(X, Con), phu\_huynh(Y, Con).

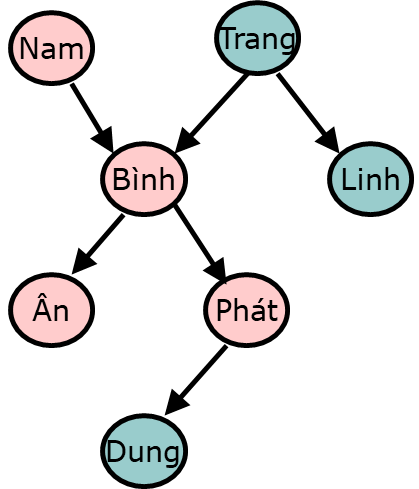
?- phu\_huynh(nam, Con), phu\_huynh(trang, Con).

?- phu\_huynh(binh, Con), phu\_huynh(linh, Con).



Giả sử, ta thêm vào các **Facts** là giới tính của từng người. Cũng với cây phả hệ trên:

*(Nút hồng là giới tính Nam, nút xanh là giới tính Nữ)*



**Ta thêm vào trong file Prolog các Facts sau:**

ong(nam).

ba(linh).

ong(an).

ong(phat).

ba(trang).

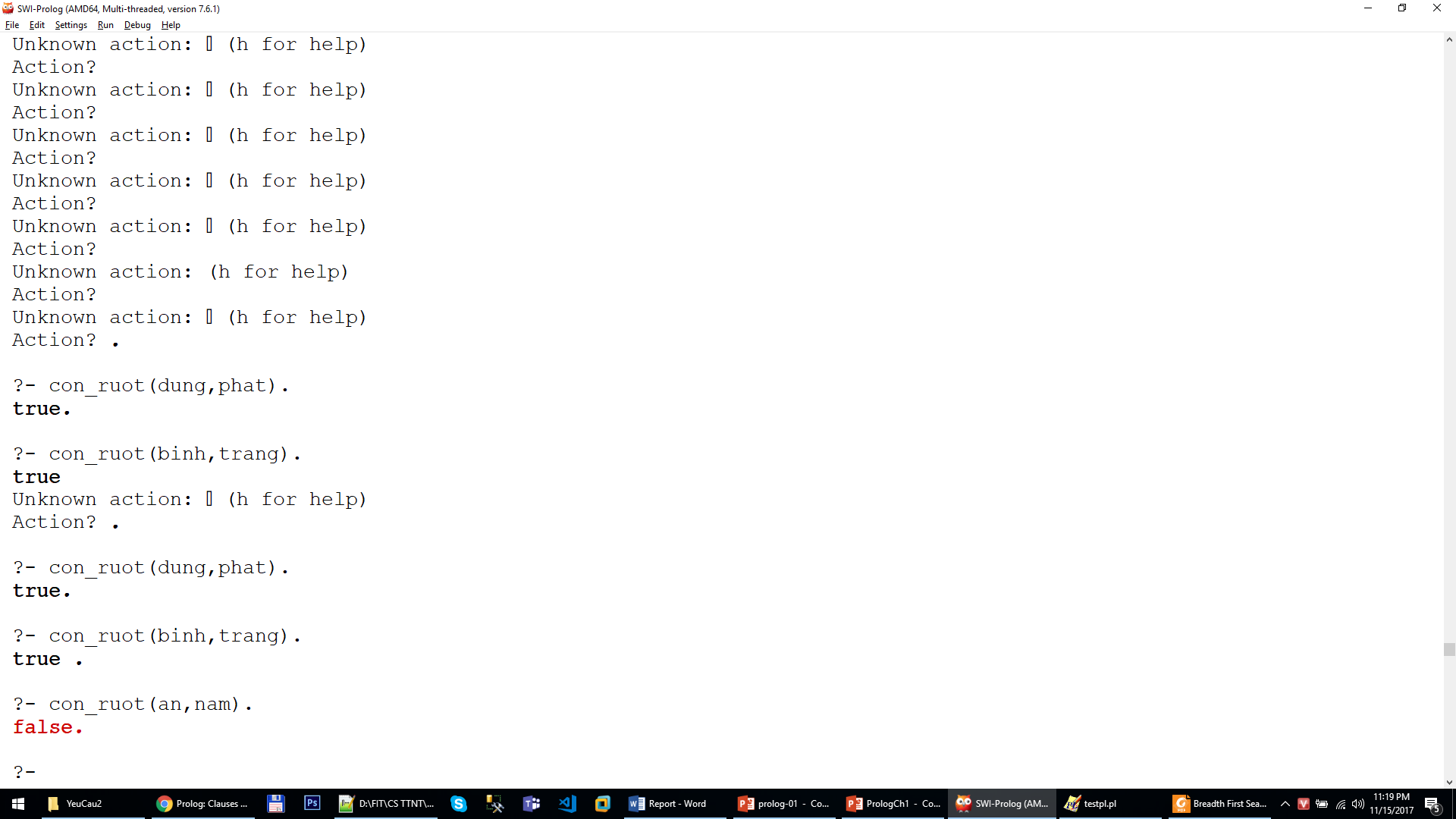
ong(binh).

ba(dung).

Lúc này ta có thể kết hợp các Facts để tạo ra các luật, chẳng hạn như:

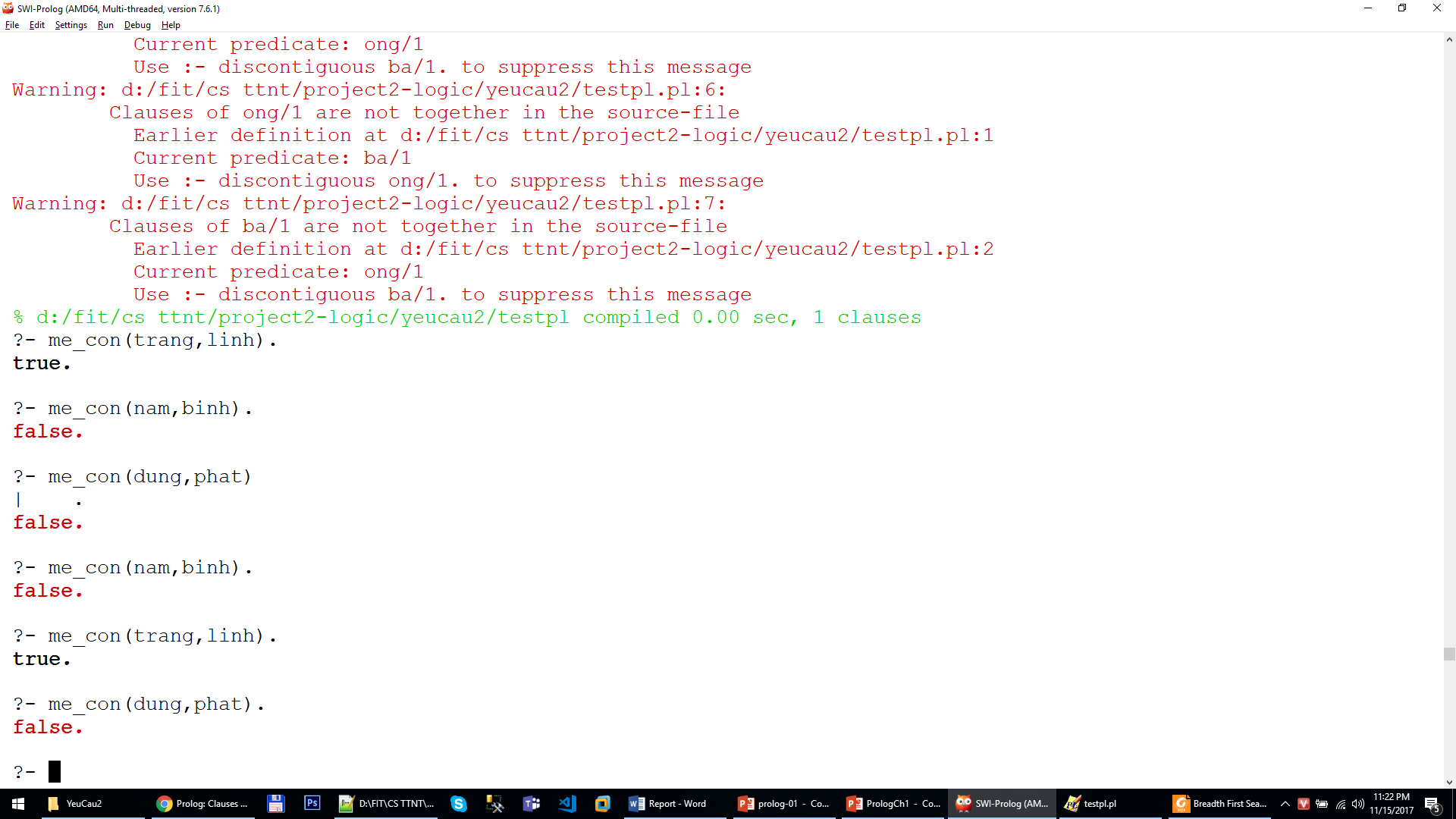
***+ Y là con ruột của X nếu X là phụ huynh của Y.***

**con\_ruot(Y, X):- phu\_huynh(X, Y).**



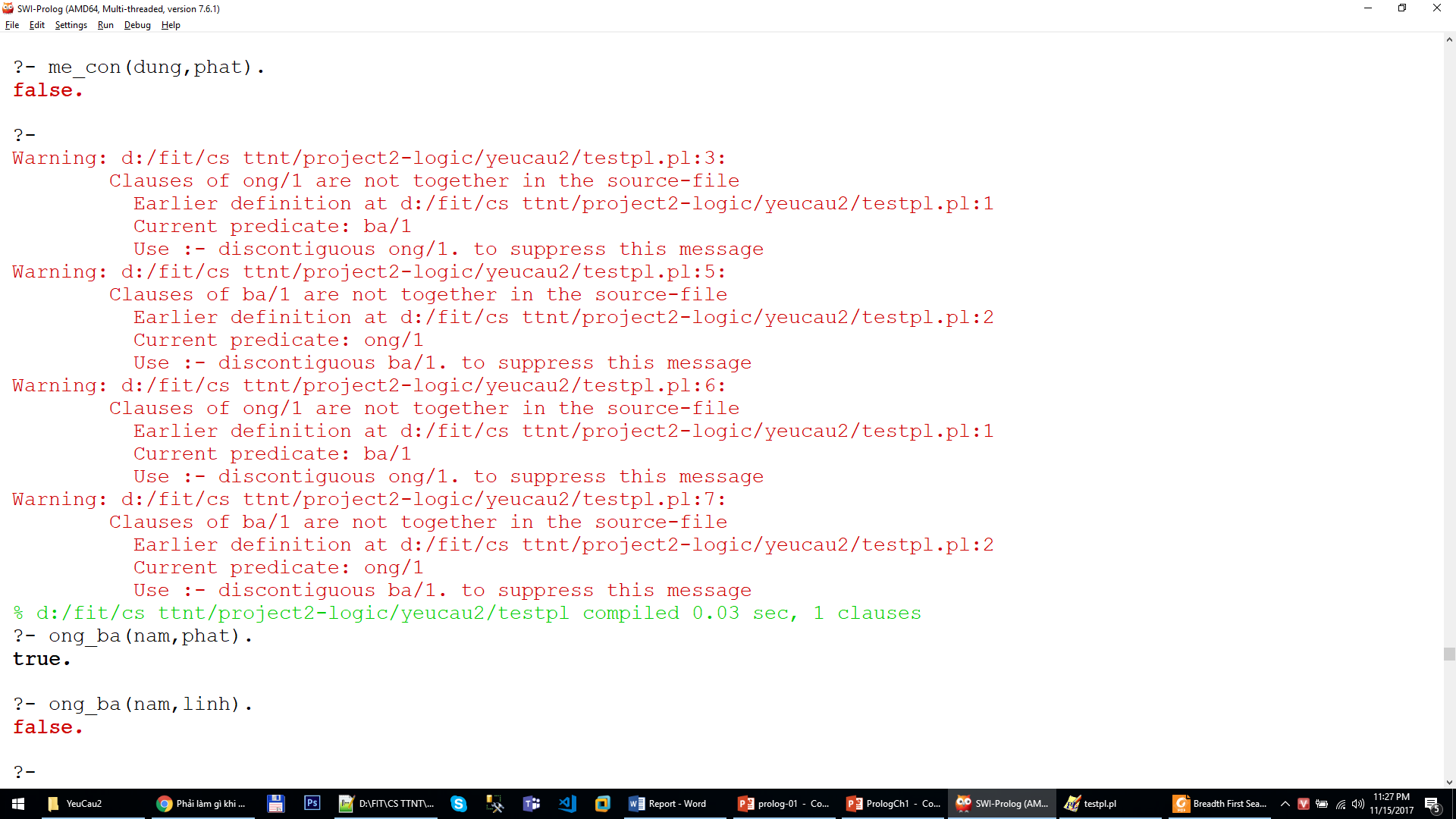
***+ X là mẹ của Y nếu X là phụ huynh của Y và X là phụ nữ***

**me\_con(X, Y):- phu\_huynh(X, Y), ba(X).**



***+ X là ông/bà của Y khi tồn tại Z mà X là phụ huynh của Z và Z là phụ huynh của Y***

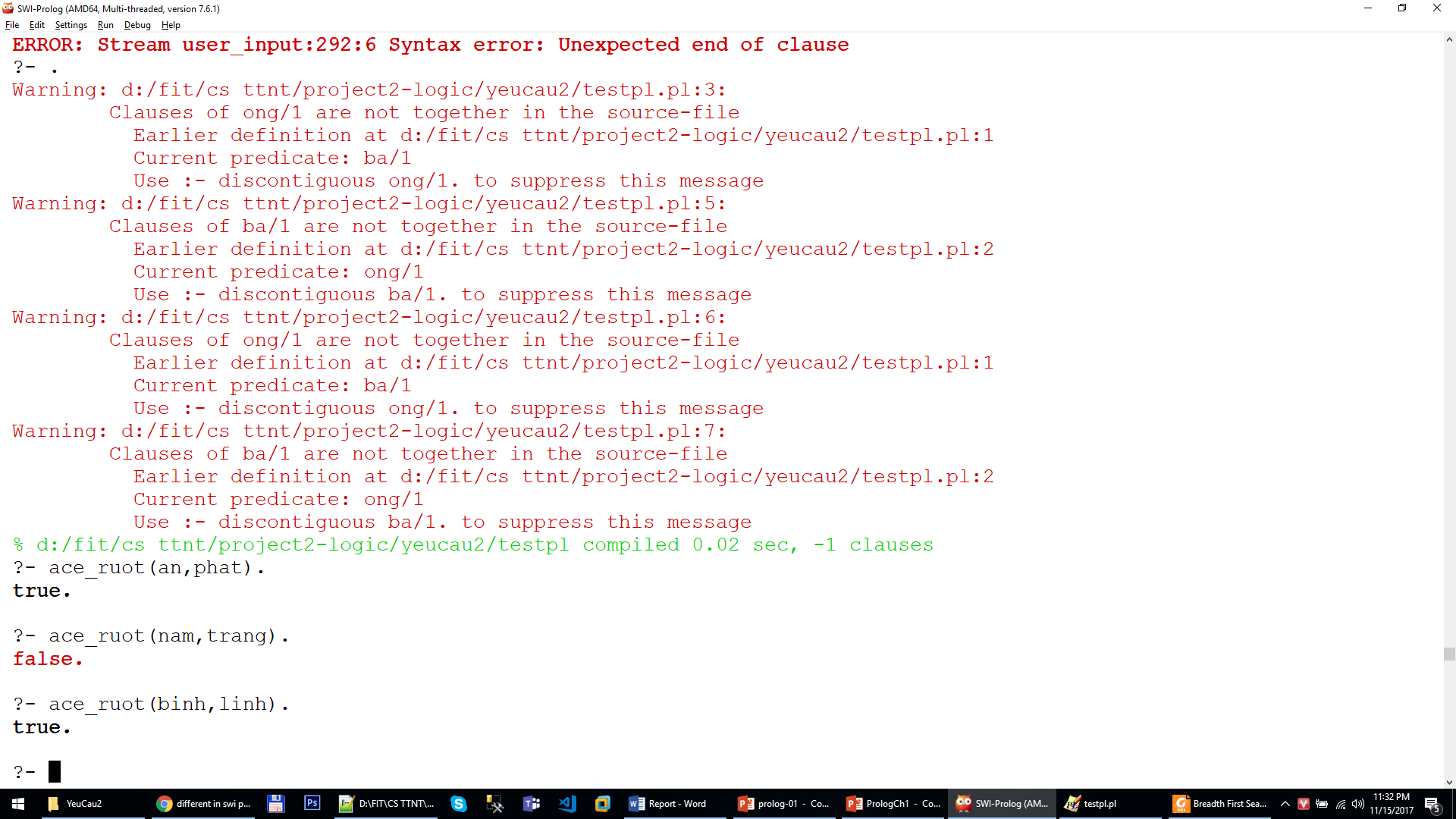
**ong\_ba(X, Y):- phu\_huynh(X, Z), phu\_huynh(Z, Y).**



***+ X là anh/chị/em ruột với Y khi X và Y có chung phụ huynh***

**ace\_ruot(X,Y) :- phu\_huynh(Z,X), phu\_huynh(Z,Y),different(X,Y).**

*dif(X,Y) dùng để kiểm tra X khác Y (do nếu bỏ đi, luôn tồn tại Y=X để thỏa mãn!).*



### Đệ quy trong Prolog

Đối với những truy vấn có rất nhiều cấp độ cần phải trả lời, ta cần sử dụng **thủ thuật đệ quy**. Prolog cũng hỗ trợ cơ chế này.

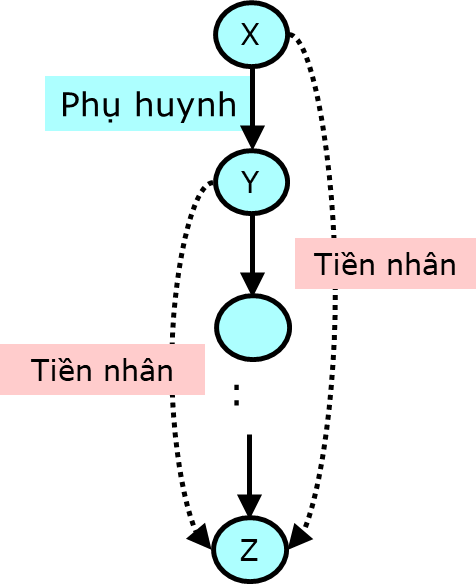
Xét truy vấn sau: **Cho biết tất cả những ai có tiền nhân là X**

**X được gọi là truyền nhân của Z nếu tồn tại Y sao cho:**

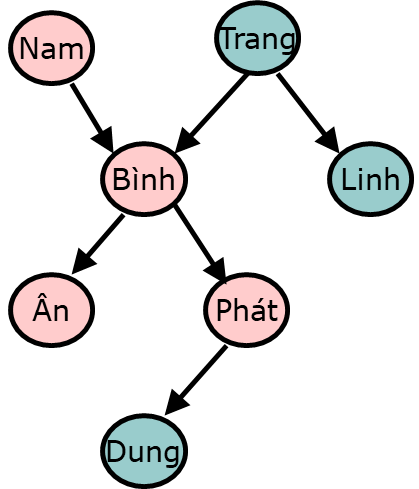
**+ X là phụ huynh của Y (1)**

**+ [và] Y là tiền nhân của Z (2)**

*Lưu ý: Nếu X là phụ huynh của Y thì Y cũng có tiền nhân là X.*



*Mô hình minh họa cho truy vấn*



*Với cây phả hệ này,* ***{Bình, Phát, Ân, Dung}*** *là những người có truyền nhân là* ***Nam***

Mô phỏng lại (1) và (2), ta có:

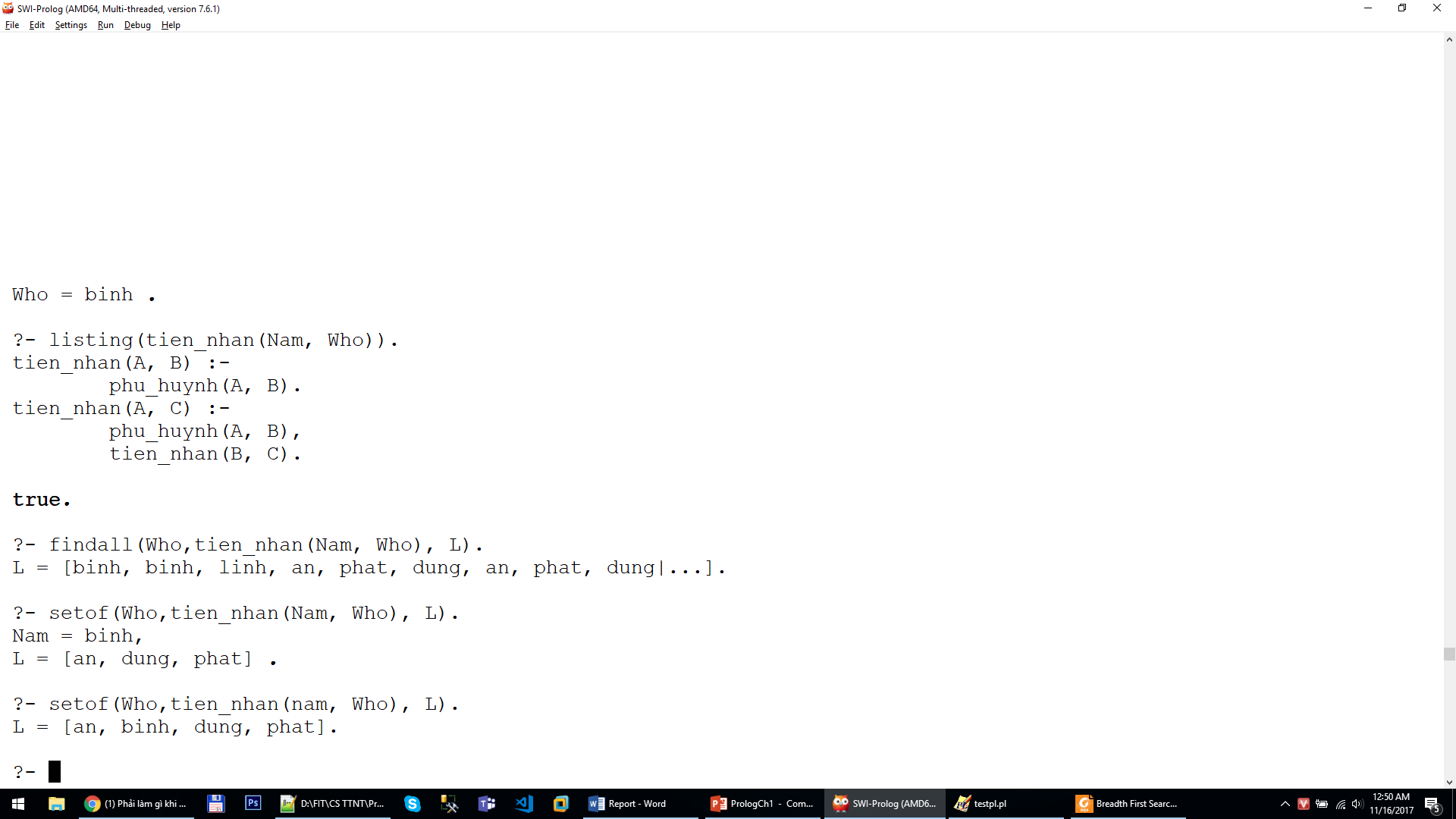
(1) <=> tien\_nhan(X, Z) :- phu\_huynh(X, Z).

(2) <=> tien\_nhan(X, Z) :- phu\_huynh(X, Y), tien\_nhan(Y, Z).

Trong đó: tien\_nhan(X, Z) đúng nếu **X là tiền nhân của Z**.

Một bộ **(1), (2)** như trên được gọi là **một thủ tục đệ quy**. Trong đó (2) đóng vai trò như **lời gọi đệ quy**, tức sẽ **tiếp tục thực hiện đến khi không còn tồn tại Y để (1) và (2) xảy ra** *(truy đến nút lá)* hoặc **(1) hoặc (2) đúng để kết thúc đệ quy**.

* Để trả lời truy vấn tương tự như truy vấn trên, Prolog đã **tìm mọi trường hợp** để làm cả (1) và (2) **thỏa mãn tất cả các điều kiện** cho đến **khi không thể thì dừng đệ quy lại**.



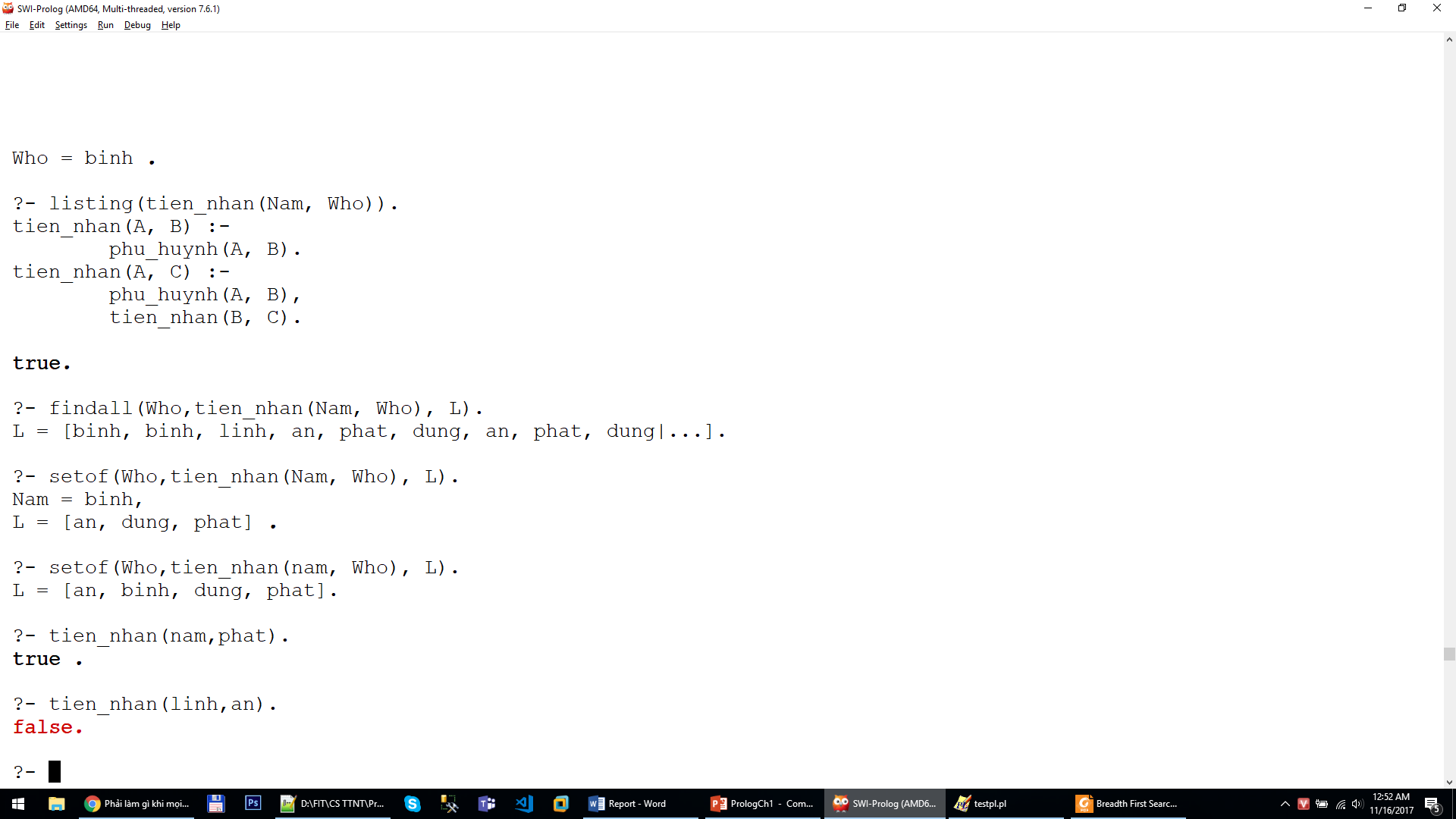
*Ví dụ: Tìm tất cả những* ***Who*** *có tiền nhân là* ***Nam*** *và lưu vào tập* ***L****.*

*hàm* ***setof()*** *tương tự findall() nhưng có tính năng lọc những trường hợp giống nhau, không cần* ***dif()***

Thử truy vấn:

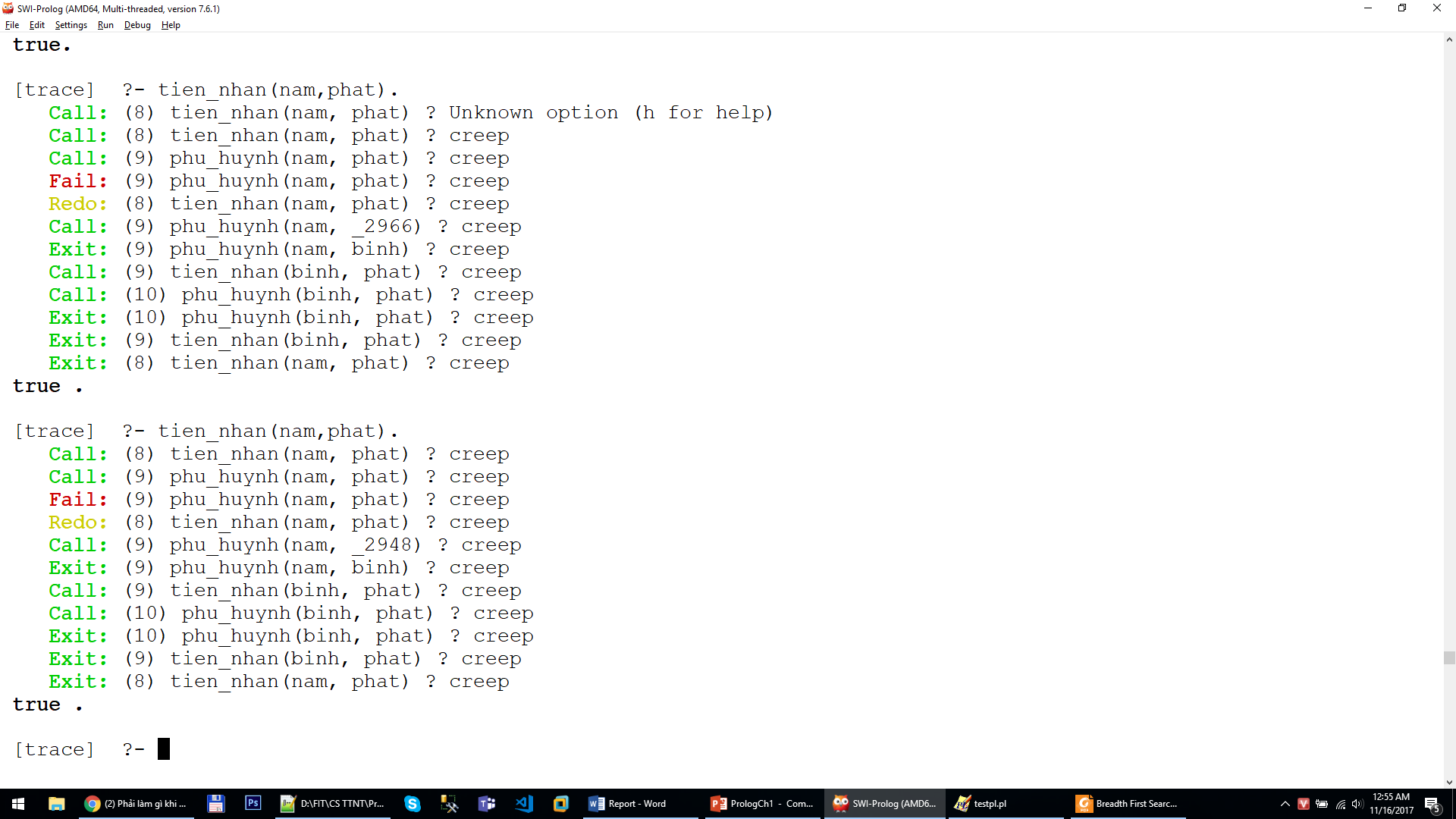
*+ Nam có phải tiền nhân của Phát hay không?:* ?- tien\_nhan(nam,phat).

*+ Linh có phải tiền nhân của Ân hay không?* ?- tien\_nhan(linh,an).



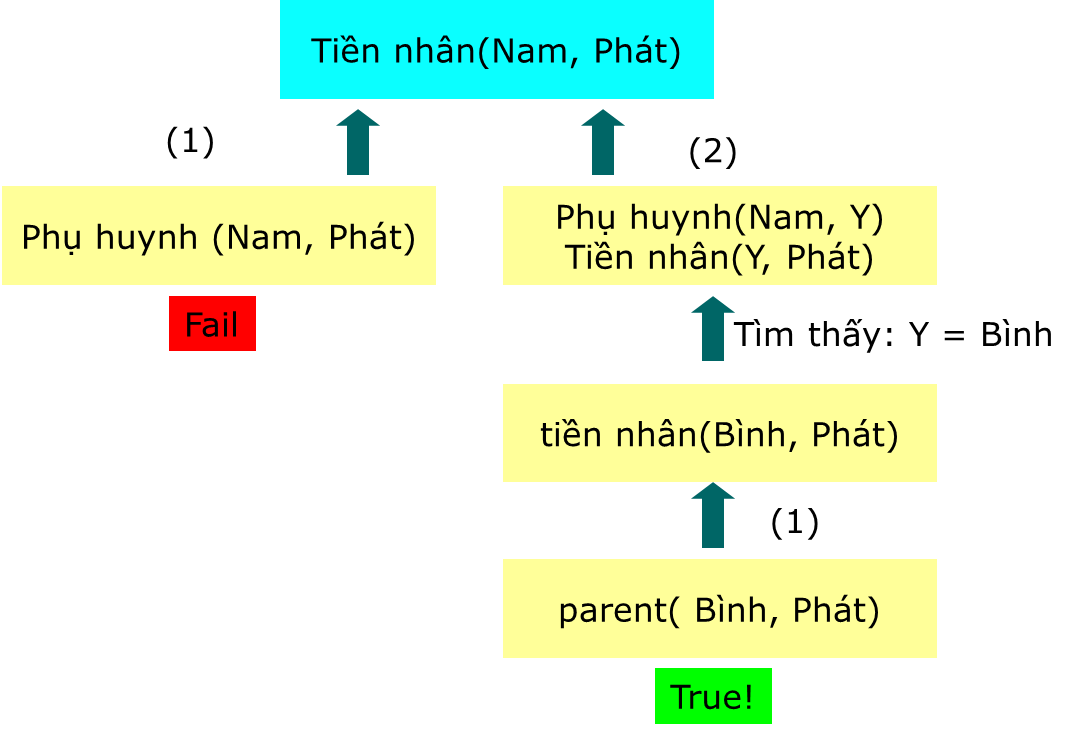
Trong SWI-Prolog, hàm **trace** hỗ trợ ta biết rõ chương trình đã suy luận như thế nào.

Bật trace, thử lại truy vấn ?- tien\_nhan(nam,phat). (\*) để biết được Prolog đã suy luận như thế nào.



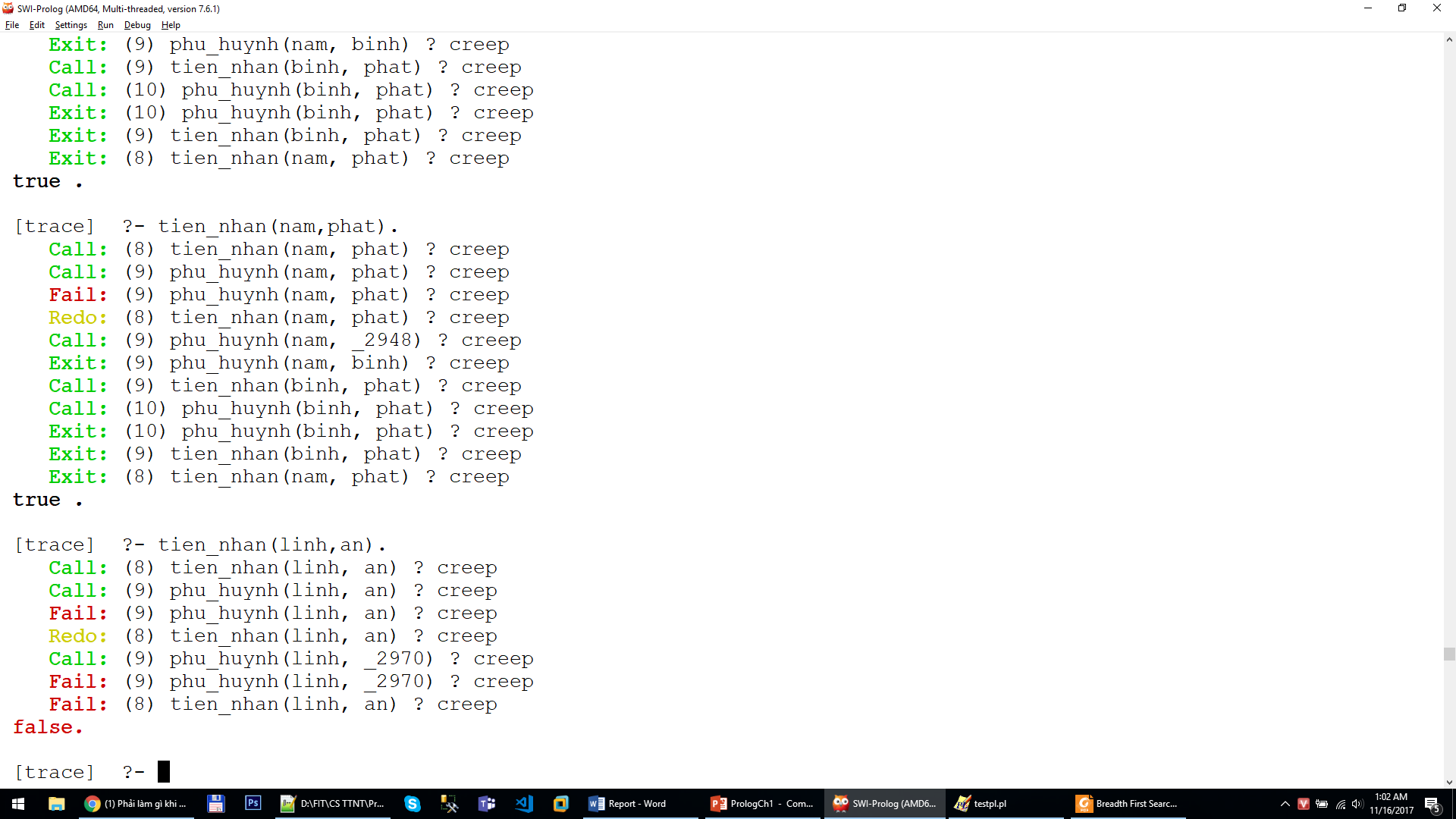
Dựa vào kết quả trên, ta thấy được:

* Đầu tiên, chương trình chọn **(1)** và chạy trước. Tuy nhiên do Nam **không phải phụ huynh của Phát** nên bị **Fail** 🡪 **Thử lại (Redo).**
* Lúc này, chương trình gọi **(2)**, tìm thấy được **Y = Bình** *(Nam là phụ huynh của Bình).* Lúc này hàm đệ quy mới được gọi là: tien\_nhan(binh, phat) (\*\*).
* Ứng với (\*\*), **(1)** được gọi lại với 2 tham số là **Bình và Phát**. Lúc này **Bình là phụ huynh của Phát** 🡪 tìm được một hàm trả về **giá trị đúng**. Lúc này, chương trình **thoát khỏi đệ quy** (*do đã tìm được đáp án*) theo thứ tự: ***hàm (1) của (\*\*) kết thúc 🡪 hàm đệ quy (\*\*) kết thúc 🡪 (\*) kết thúc.***
* Do tìm được giá trị đúng 🡪 truy vấn trả về: **True!**

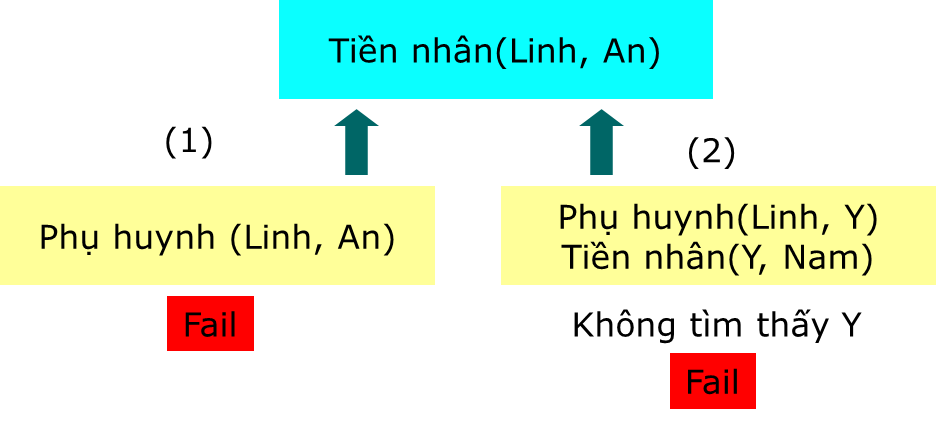


*Mô hình của truy vấn*

Đối với truy vấn ?- tien\_nhan(linh, an) (\*):



* Đầu tiên (\*) gọi **(1)**. Tuy nhiên **Linh không phải phụ huynh của Ân** nên bị **Fail** 🡪 **Thử lại (Redo)**.
* Lúc này, chương trình gọi **(2)**. Tuy nhiên, chương trình **không thể tìm được Y nào** để **Linh là phụ huynh của Y** nhằm gọi đệ quy 🡪 **Fail**. Do đã hết hàm tương ứng (\*) 🡪 Kết thúc theo thứ tự: ***Kết thúc (2) 🡪 Kết thúc (\*).***
* Do không tìm thấy được một Rule/Fact nào 🡪 Trả về **False!**



*Mô hình minh họa truy vấn*

## Sử dụng SWI-Prolog xây dựng cây phả hệ

Do ở phần 3.1 nhóm đã thể hiện Prolog trên công cụ SWI-Prolog nên ở phần này nhóm tập trung vào cây phả hệ.

### Source Code xây dựng cơ sở tri thức

Toàn bộ Source Code được đặt trong đường dẫn **YeuCau2\[AI-Proj2] England-Family-Tree.pl**

**Mô tả các vị từ mô tả quan hệ trong hình:**

- **parent(Parent,Child):** Mô tả quan hệ cha mẹ-con cái (Parent là cha mẹ của Child).

- **male(Person):** Mô tả giới tính Nam cho Person.

- **female(Person):** Mô tả giới tính Nữ cho Person.

- **married(Person, Person):** Mô tả quan hệ hôn nhân của hai người (Person đã từng làm đám cưới với Person, hiện tại có thể vẫn đang là vợ chồng hoặc đã li hôn).

- **divorced(Person, Person):** Mô tả quan hệ li hôn của hai người (Person đã từng làm đám cưới với Person, nhưng hiện tại đã li hôn).

**Mô tả các vị từ hỗ trợ xác định trên cây gia phả:**

*- Father – quan hệ cha-con: Khi là cha mẹ & người đó là Nam.*

**father**(Father,Children):- **male**(Father), **parent**(Father,Children).

*- Mother – quan hệ mẹ-con: Khi là cha mẹ & người đó là Nữ.*

**mother**(Mother,Children):- **female**(Mother), **parent**(Mother,Children).

*- Child – quan hệ con cái-cha mẹ: Khi tồn tại mối quan hệ người tại (cha mẹ-con cái)*

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

*- Son – quan hệ con trai-cha mẹ: Khi người con là Nam*

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

*- Dauget – quan hệ con gái-cha mẹ: Khi người con là Nữ*

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

*- Grandparent – quan hệ ông bà-cháu: Khi ông bà là ba mẹ của ba mẹ cháu.*

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

*- Grandfather – quan hệ ông-cháu: Khi ông/bà là Nam*

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

*- Grandmother – quan hệ bà-cháu: Khi ông/bà là Nữ*

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

*- Grandchild – quan hệ cháu-ông bà: Khi tồn tại quan hệ ngược lại ông bà-cháu*

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

*- Grandson – quan hệ ông bà-cháu trai: Khi là cháu và là Nam*

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

*- Granddaughter – quan hệ ông bà-cháu gái: Khi là cháu và là Nữ*

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

*- Spouse – quan hệ vợ/chồng: Khi đã cưới nhau nhưng chưa li hôn*

**spouse**(Husband,Wife) :- **married**(Husband,Wife),

**not**(**divorced**(Husband,Wife)).

*- Husband – quan hệ chồng-vợ: Khi đang là vợ chồng và là Nam*

**husband**(Person,Wife) :- **male**(Person), **spouse**(Person,Wife).

*- Wife – quan hệ vợ-chồng: Khi đang là vợ chồng và là Nữ*

**wife**(Person,Husband) :- **female**(Person),**spouse**(Person,Husband).

*- Sibling – quan hệ anh/chị/em ruột: Khi Person1 & Person2 có chung cha mẹ.*

**sibling**(Person1,Person2) :- **parent**(Parent,Person1),

**parent**(Parent,Person2),

**dif**(Person1,Person2).

*- Brother – quan hệ anh/em trai: Khi anh/chị/em ruột là Nam*

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

*- Sister – quan hệ chị/em gái: Khi anh/chị/em ruột là Nữ*

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

*- Aunt – quan hệ dì/cô/thím/bác gái: Khi người đó là chị/em gái của bố mẹ ruột, hoặc là vợ của chú/dượng/cậu/bác trai:*

**aunt**(Aunt,Person) **:- parent**(Parent,Person),

(**sister**(Aunt,X);(**brother**(Uncle,Parent),**wife**(Aunt,Uncle))).

*- Uncle – quan hệ chú/dượng/cậu/bác trai: Khi người đó là anh/em trai của bố mẹ ruột, hoặc là chồng của dì/cô/thím/bác gái:*

**uncle**(Uncle,Person) :- **parent**(Parent,Person),

(**brother**(Uncle,Parent);(**sister**(Aunt,Parent),**husband**(Uncle,Aunt))).

*- Nephew: Con trai của anh/chị/em của mình -> Mình là cô/chú của nó và nó là Nam.*

**nephew**(Nephew,Person) :- **male**(Nephew),

(**aunt**(Person,Nephew);**uncle**(Person,Nephew)).

*- Niece: Con gái của anh/chị/em mình -> Mình là cô/chú của nó và nó là Nữ.*

**niece**(Niece,Person) :- **female**(Niece),

(**aunt**(Person,Niece);**uncle**(Person,Niece)).

*- firstCousin: Anh em họ cùng lứa -> Có chung ông/bà, nhưng không phải là anh chị em ruột*

**firstcousin**(Person1,Person2) :- **grandparent**(GP,Person1),

**grandparent**(GP,Person2),

**dif**(Person1,Person2),

**not**(**sibling**(Person1,Person2)).

### Tập câu hỏi cho hệ tri thức

|  |  |
| --- | --- |
| Tập câu hỏi, code Prolog và đáp án | |
| Test 01 | **Sarah Ferguson có phải mẹ của Princess Eugenie không?** |
| ?- **mother**(sarah\_ferguson, princess\_eugenie). |
| true . |
| Test 02 | **Ai là cha của Prince Charles?** |
| ?- **father**(Father, prince\_charles). |
| Father = prince\_phillip . |
| Test 03 | **Kate Middleton có phải vợ của Prince Harry không?** |
| ?- **wife**(kate\_middleton, prince\_harry). |
| false . |
| Test 04 | **Liệt kê tất cả các con của Queen Elizabeth II** |
| ?- **findall**(X,parent(queen\_elizabeth\_ii, X),Child). |
| Child = [prince\_charles, princess\_anne, prince\_andrew, prince\_edward]. |
| Test 05 | **Liệt kê tất cả các cháu (ở mức cậu) trai của Captain Mark Phillips** |
| ?- **setof**(X,nephew(X, captain\_mark\_phillips),Nephew).**x** |
| false . |
| Test 06 | **Liệt kê tất cả cô/dì của Prince William** |
| ?- **setof**(X,aunt(X, prince\_william),Aunt). |
| Aunt = [princess\_anne, sophie\_rhys\_jones]. |
| Test 07 | **Timothy Laurence có phải là chú của Zara Phillips hay không?** |
| ?- **uncle**(timothy\_laurence,zara\_phillips). |
| false. |
| Test 08 | **Peter Phillips có người cô nào tên là Sarah Ferguson hay không?** |
| ?- **aunt**(sarah\_ferguson, peter\_phillips). |
| false . |
| Test 09 | **Camilla Parker Bowles có phải là dì của Prince Harry hay không?** |
| ?- **aunt**(camilla\_parker\_bowles, prince\_harry). |
| false. |
| Test 10 | **Liệt kê tất cả anh em họ hàng bậc 1 của Isla Phillips** |
| ?- **setof**(X,firstcousin(X,isla\_phillips),FCousin). |
| FCousin = [mia\_grace\_tindall]. |
| Test 11 | **Liệt kê tất cả anh/em trai họ hàng bậc 1 của Peter Phillips** |
| ?- **setof**(X,(male(X), firstcousin(X,peter\_phillips)),FCousin). |
| FCousin = [james\_viscount\_severn, prince\_harry, prince\_william]. |
| Test 12 | **Liệt kê tất cả chị/em gái họ hàng bậc 1 của Mia Grace Tindall** |
| ?- **setof**(X,(female(X), firstcousin(X,mia\_grace\_tindall)),FCousin). |
| FCousin = [isla\_phillips, savannah\_phillips]. |
| Test 13 | **Ai là bà của Mia Grace Tindall?** |
| ?- **grandmother**(GM,mia\_grace\_tindall). |
| GM = princess\_anne . |
| Test 14 | **Prince Charles có phải là bà nội của Prince George không?** |
| ?- **grandmother**(prince\_charles, prince\_george). |
| false. |
| Test 15 | **Prince William có phải vợ/chồng của Kate Middleton không?** |
| ?- **spouse**(prince\_william, kate\_middleton). |
| true . |
| Test 16 | **Zara Phillips và Prince Harry có chung ông (grandfather) hay không?** |
| ?- **grandfather**(GF, zara\_phillips), **grandfather**(GF, prince\_harry). |
| GF = prince\_phillip . |
| Test 17 | **Autumn Kelly và Peter Phillips có chung bà (grandmother) hay không?** |
| ?- **grandmother(**GM, autumn\_kelly), **grandmother**(GM, peter\_phillips). |
| false . |
| Test 18 | **Ai vừa là dì của Savannah Phillips vừa là chú của Isla Phillips?** |
| ?- **aunt(**Aunt, savannah\_phillips), **aunt**(Aunt, isla\_phillips). |
| Aunt = zara\_phillips . |
| Test 19 | **Ai vừa là cậu của Prince William vừa là cậu của Peter Phillips?** |
| ?- **uncle(**Uncle,prince\_william**), uncle(**Uncle**,**peter\_phillips**).** |
| Uncle = prince\_andrew . |
| Test 20 | **Liệt kê tất cả những người cháu của Prince Phillip chưa có gia đình.** |
| ?- **setof(GC**, (grandchild(GC,prince\_phillip), not(spouse(GC, SGC))), Z). |
| Z = [james\_viscount\_severn, lady\_louise\_mountbatten\_windsor, prince\_harry, princess\_beatrice, princess\_eugenie]. |
| Test 21 | **Captain Mark Phillips và Princess Anne có phải là vợ chồng hay không?** |
| ?- **spouse**(captain\_mark\_phillips,princess\_anne). |
| **false** . |