TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN TPHCM

KHOA CÔNG NGHỆ THÔNG TIN

**BÁO CÁO MÔN HỌC**

**CƠ SỞ TRÍ TUỆ NHÂN TẠO**

Đồ án 2

**LOGIC BẬC NHẤT**

Nhóm sinh viên thực hiện: Vũ Lê Thế Anh 1612838

Nguyễn Lê Hồng Hạnh 1612849

# **Tìm hiểu ngôn ngữ Prolog**

## **Giới thiệu:**

Prolog là một ngôn ngữ lập trình logic [logic programming] và tính toán trên ký hiệu [symbolic computation], được biết đến với khả năng mạnh mẽ trong việc giải quyết những vấn đề liên quan đến ký hiệu [symbol] và suy diễn [inference], rất quan trọng trong lĩnh vực trí tuệ nhân tạo [Artificial Intelligence].

Một chương trình Prolog bao gồm một chuỗi các luật [rule] và các sự thật [fact], gọi là các mệnh đề Horn [Horn clause], tạo thành một cơ sở tri thức [knowledge base]. Một chương trình được chạy bằng cách đưa ra một truy vấn [query] và kiểm tra xem nó có thể được chứng minh (hay phản chứng) dựa trên cơ sở tri thức cho trước.

## **Sự thật:**

Trong Prolog, chúng ta có thể biểu diễn một phát biểu [statement] bằng một sự thật. Sự thật có thể bao gồm một phần tử cụ thể hoặc một mối quan hệ [relation] giữa các phần tử.

### **Sự thật nguyên tố:**

Sự thật nguyên tố giống như một mệnh đề trong logic mệnh đề. Ví dụ, ta có thể biểu diễn việc cảm thấy vui bằng cách viết chương trình:

happy.

Lúc này, ta có thể đưa ra câu truy vấn kiểm tra xem trời có nắng thật không:

?- happy.

Từ thời điểm này, khi có kí hiệu ‘?-’ ở đầu dòng lệnh, ta hiểu đó là một câu truy vấn, ngược lại, đó là một câu chương trình.

Câu truy vấn trên sẽ trả về true (hoặc yes), do có thể tìm thấy sự thật happy trong cơ sở tri thức.

Mặt khác, giả sử câu truy vấn được đặt ra là:

?- sad.

Câu truy vấn này sẽ trả về kết quả là false (hoặc no). Lưu ý rằng điều này không đồng nghĩa với việc ‘cảm thấy buồn’ là một mệnh đề sai (như ý nghĩa của false trong logic), nó chỉ có nghĩa là không thể suy diễn điều đó từ những thông tin đã biết. Cụ thể, cơ sở tri thức hiện tại chỉ có happy.

Cú pháp khai báo một sự thật là một chuỗi ký tự gồm ký số, ký tự chữ hoặc dấu gạch dưới (\_), bắt đầu bằng một ký tự thường và kết thúc bằng dấu chấm (.).

Một số ví dụ của sự thật (trong một thế giới giả định):

the\_anh\_sieng\_hoc.

hong\_hanh\_luoi\_bieng.

the\_anh\_thich\_an\_kem\_9k\_cua\_Baskin\_Robin.

Một số ví dụ sai cú pháp:

The\_Anh. (ký tự chữ hoa ở đầu)

\_hong\_hanh. (ký tự \_ ở đầu)

vlta (thiếu dấu chấm)

### **Sự thật với đối số:**

Các sự thật phức tạp hơn bao gồm một mối quan hệ giữa các phần tử mà sự thật đó nói tới. Các phần tử này được gọi là đối số [argument]. Một sự thật có thể có 0 hay nhiều đối số (sự thật nguyên tố chính là sự thật với 0 đối số). Cấu trúc khai báo tổng quát của sự thật có đối số:

<vị từ>(<đối số>{,<đối số>}).

Trong đó, vị từ [predicate] phải bắt đầu bằng ký tự chữ thường. Ta lấy ví dụ:

like(theanh,honghanh).

Sự thật trên biểu diễn mối quan hệ quý mến giữa theanh và honghanh. Lưu ý rằng mối quan hệ này là một chiều. Ví dụ, nếu ta định nghĩa like(A, B) là người A quý người B thì với chương trình có sự thật trên, khi đặt câu hỏi liệu honghanh có quý theanh, hay truy vấn:

?- like(honghanh, theanh).

Câu truy vấn sẽ trả về false (không) do trong cơ sở tri thức chỉ chứa việc theanh quý honghanh.

Chiều của mối quan hệ là do người xây dựng cơ sở tri thức định nghĩa ban đầu. Do đó, việc định nghĩa phải rõ ràng và đảm bảo cách hiểu thống nhất.

Để ví dụ về sự thật có đối số, xét chương trình:

hoc(theanh,csttnt).

hoc(theanh,matma).

hoc(honghanh,csttnt).

hoc(honghanh,dohoa).

Trong đó hoc(A, B) mang hàm nghĩa người A học lớp B.

Từ đó có các truy vấn và kết quả (/\*..\*/ là các bình luận, chứa kết quả):

?- hoc(theanh,csttnt). /\* true \*/

?- hoc(honghanh,matma). /\* false \*/

?- hoc(quoccuong,matma). /\* false \*/

?- hoc(theanh, cosotrituenhantao). /\* false \*/

Truy vấn cuối lại một lần nữa nói lên vấn đề về ngữ nghĩa của kết quả “false”.

## **Biến:**

Lúc này, làm sao để đặt câu hỏi: “Thế Anh học môn gì?”. Giả sử chúng ta có sự thật sau trong cơ sở tri thức:

hoc(theanh,csttnt).

Để đặt câu hỏi theanh học môn gì, ta thử truy vấn:

?- hoc(theanh,mongi).

Kết quả trả về rõ ràng là false, bởi vì trong cơ sở tri thức chỉ biết theanh có quan hệ hoc với csttnt, chứ không có quan hệ hoc với mongi.

Để cho phép thực hiện những truy vấn kiểu như trên, Prolog bổ sung một khái niệm đó là biến [variable]. Quá trình đồng nhất giữa phần tử và các biến gọi là quá trình đồng nhất [unification].

Biến được phân biệt bởi ký tự chữ hoa ở đầu. Dấu gạch dưới (\_) cũng đóng vai trò như một ký tự in hoa. Một số ví dụ của biến:

X

Bien

\_bien

Với biến, ta có thể biểu diễn câu hỏi đầu đề bằng truy vấn:

?- hoc(theanh, Mon).

Truy vấn trên cho kết quả trả về là:

Mon = csttnt.

Kết quả của truy vấn cho thấy biến Mon đã được đồng nhất với csttnt.

Trường hợp truy vấn lúc này của chúng ta là:

?- hoc(honghanh,X).

Truy vấn sẽ trả về no, lý do là không đồng nhất được phần tử nào với biến X.

## **Luật**

Luật cho phép chúng ta tạo ra những phát biểu có điều kiện. Mỗi luật có nhiều biến thể, gọi là các mệnh đề [clause].

Xét phát biểu “Ai rồi cũng sẽ chết.”, phát biểu này có thể được biểu diễn bởi luật:

se\_chet(X) :- nguoi(X)

Có hai thể hiện [interpretation] của mệnh đề này. Cách thể hiện tuyên bố [declarative interpretation] là “Cho X, X sẽ chết nếu X là người.” Cách thể hiện quy trình [procedural interpretation] là “Để chứng minh đích chính là X sẽ chết, phải chứng minh đích phụ là X là người.”

Tiếp tục thêm vào cơ sở tri thức sự thật “Socrates là người.”, ta có:

se\_chet(X) :- nguoi(X)

nguoi(socrates).

Lúc này, nếu truy vấn:

?- se\_chet(socrates).

Prolog sẽ đưa ra câu trả lời là true.

Prolog đã làm điều này bằng cách nào? Để tìm ra câu trả lời cho truy vấn se\_chet(socrates). nó đã dùng luật ở trên, nói rằng để chứng minh một đối tượng là sẽ chết, trước hết phải chứng minh đối tượng đó là người. Nói cách khác, từ đích chính se\_chet(socrates)., Prolog tạo ra một đích phụ là chứng minh nguoi(socrates). Đích phụ này được thỏa mãn bởi sự thật nguoi(socrates), dẫn đến đích chính cũng thỏa mãn và Prolog trả về true.

Chúng ta cũng có thể dùng biến trong các truy vấn. Ví dụ, để tìm coi những ai sẽ chết, ta truy vấn:

?- se\_chet(P).

Và Prolog sẽ trả về:

P = socrates.

Điều này nghĩa là Prolog đã tìm được đích đến se\_chet(P) bằng cách đồng nhất biến P với socrates. Quy trình làm việc này tương tự trên: ta chứng minh đích chính se\_chet(P) bằng cách chứng minh đích phụ nguoi(P). Prolog sẽ tìm bất kì P nào là người. Nhận thấy nguoi(P) có thể đồng nhất nguoi(socrates) bằng cách gắn kết P và socrates. Gắn kết này truyền về đích chính và Prolog trả về như trên.

Một ví dụ khác về luật, xét cơ sở tri thức:

ngon(X) :- mem(X), thit(X). /\* một món là ngon nếu nó là thịt và nó mềm \*/

ngon(X) :- ngot(X), keo(X). /\* một món là ngon nếu nó là kẹo và nó ngọt \*/

ngon(kem). /\* kem thì luôn ngon \*/

Như vậy, ta có tới ba cách để kiểm tra xem một món có ngon hay không. Cả ba phương án này được biểu diễn bởi ba mệnh đề của cùng một vị từ ngon. Khi kiểm tra truy vấn, Prolog sẽ bắt đầu từ mệnh đề đầu tiên (bất kể là luật hay sự thật) của ngon và thử đồng nhất nó. Nếu không thành công, Prolog thử các mệnh đề tiếp theo cho tới khi hết mệnh đề để thử, lúc này Prolog sẽ trả về false.

Mọi biến cùng tên trong một mệnh đề (ví dụ như X ở trên) bắt buộc phải cùng mang một giá trị trong mọi nghiệm của một truy vấn. Đối với trường hợp trong các mệnh đề khác nhau, các biến này là độc lập. Ta nói các biến mang tính cục bộ của mệnh đề.

Trong trường hợp trên:

ngon(X) :- mem(X), thit(X).

ngon(X) :- ngot(X), keo(X).

được Prolog hiểu như là:

ngon(X1) :- mem(X1), thit(X1).

ngon(X2) :- mem(X2), thit(X2).

Thực tế trong quá trình thông dịch Prolog cũng gán số ngẫu nhiên vào đuôi biến để đảm bảo tính cục bộ này.

Để cho một ví dụ cụ thể, ta xét cơ sở tri thức:

ngon(X) :- mem(X), thit(X).

ngon(X) :- ngot(X), keo(X).

thit(bo\_Kobe).

thit(trau).

keo(keo\_sua).

keo(keo\_me).

mem(bo\_Kobe).

ngot(keo\_sua).

Ở ví dụ đầu tiên, ta muốn biết liệu kẹo sữa có ngon không. Ta truy vấn:

?- ngon(keo\_sua).

Prolog trả về true.

Khi thực thi truy vấn này, Prolog trước hết kiểm tra mem(keo\_sua). vốn không thành công vì chỉ có bo\_kobe là mem trong cơ sở tri thức. Điều này dẫn đến mệnh đề đầu tiên của ngon sẽ không thành công. Prolog tiếp tục xét đến mệnh đề tiếp theo. Điều này nghĩa là kiểm tra ngot(keo\_sua) rồi keo(keo\_sua). Cả hai đều thỏa theo cơ sở tri thức. Vậy kết quả trả về là true như trên.

Ở ví dụ tiếp theo, ta muốn biết tất cả những món ngon. Ta truy vấn:

?- ngon(Mon).

Prolog sẽ trả về:

Mon = bo\_Kobe

Khi thực thi truy vấn này, Prolog trước hết thử mệnh đề đầu tiên của ngon. Kết quả của việc này là yêu cầu tìm đến mem(Mon), đạt được bằng cách gắn kết Mon với bo\_Kobe. Tiếp theo cần tìm thit(bo\_Kobe), điều này thỏa mệnh đề (sự thật) trong cơ sở tri thức và đích đến ngon(bo\_Kobe) đạt được.

Ta sẽ tìm hiểu kĩ hơn về quá trình tìm kiếm này trong phần sau.

## **Tìm kiếm chứng cứ**

Ở mục này, ta sẽ tìm hiểu kĩ hơn quá trình tìm đến đích của Prolog. Giả sử ta có cơ sở tri thức như sau:

an(theanh,bo).

an(theanh,ga).

an(theanh,vit).

Để trả lời câu hỏi “Tất cả những món mà Thế Anh ăn là gì?”, ta truy vấn:

an(theanh,Mon).

Như đã trình bày ở trên, Prolog sẽ trả về:

Mon = bo

Nếu tiếp tục hỏi, sẽ nhận:

Mon = ga

Tiếp tục, nhận:

Mon = vit

Tại đây, nếu tiếp tục, ta sẽ nhận về false do không thể tìm thấy cách đồng nhất nào khác.

Cơ chế tìm kiếm nhiều lời giải này gọi là quay lui [backtracking].

Để lấy ví dụ cho cơ chế này, ta xét cơ sở tri thức:

ngon(X) :- thit(X), mem(X).

thit(trau).

thit(bo\_Kobe).

mem(bo\_Kobe).

Với truy vấn:

?- ngon(Mon)

Ta xem quy trình xử lý của Prolog:

* Đầu tiên, nó tìm trong cơ sở tri thức các mệnh đề của vị từ ngon và tìm thấy:

ngon(X) :- thit(X), mem(X).

* Prolog sẽ đồng nhất ngon(X) và ngon(Mon) (đồng nhất 2 biến) bằng cách khởi tạo chung 1 biến, giả sử là \_Var1. Câu truy vấn sẽ trở thành:

ngon(\_Var1) :- thit(\_Var1), mem(\_Var1).

* Lúc này, Prolog sẽ tìm đối tượng thỏa mãn mem(\_Var1) và thit(\_Var1), tức truy vấn:

thit(\_Var1), mem(\_Var1).

* Tiếp theo, Prolog tìm trong cơ sở tri thức các mệnh đề của vị từ thit và tìm thấy:

thit(trau).

* Prolog sẽ đồng nhất \_Var1 và trau (đồng nhất biến và tập hằng) bằng cách khởi tạo \_Var1 thành trau, câu truy vấn trở thành:

thit(trau), mem(trau).

* thit(trau) có trong cơ sở tri thức nên câu truy vấn trên trở thành:

mem(trau).

* mem(trau) không có trong cơ sở tri thức, hướng tìm thất bại, Prolog quay lui về

thit(\_Var1), mem(\_Var1).

* Prolog lại tiếp tục tìm trong cơ sở tri thức mệnh đề tiếp theo của thit, có:

thit(bo\_Kobe).

* Tiếp tục đồng nhất bằng cách khởi tạo \_Var1 bằng bo\_Kobe và truy vấn:

thit(bo\_Kobe), mem(bo\_Kobe).

* Do thit(bo\_Kobe) đã thỏa nên truy vấn:

mem(bo\_Kobe).

* mem(bo\_Kobe) có trong cơ sở tri thức, do đó hướng tìm thành công.
* Prolog trả về Mon = bo\_Kobe.

## **Đệ quy:**

Trong một số yêu cầu lập trình, chúng ta muốn tuần tự lặp lại một quy trình nào đó trên toàn cấu trúc dữ liệu, hoặc cho đến khi một điều kiện nào đó thỏa mã. Chúng ta có thể làm việc này bằng đệ quy với Prolog.

Thông thường, đệ quy được thực hiện bằng cách định nghĩa các điều kiện dừng trước là các sự thật, sau đó định nghĩa các luật để thực hiện các thao tác nào đó trước khi tự gọi lại chính mình.

Sau đây là một ví dụ, xét cơ sở tri thức:

tren\_duong(rome).

tren\_duong(Dich) :- kha\_thi(Dich, DichMoi), tren\_duong(DichMoi).

kha\_thi(nha, benxe).

kha\_thi(benxe, rome).

Vị từ tren\_duong (vị trí hiện tại có nằm trên đường đến Rome không) được định nghĩa một cách đệ quy. Chương trình muốn xem từ một điểm bất kì Dich có thể đi đến Rome được không. Mệnh đề đầu tiên cho biết Rome luôn nằm trên đường đi đến Rome. Mệnh đề thứ hai kiểm tra xem có thể đi từ địa điểm hiện tại (Dich) đến đi điểm tiếp (DichMoi), và kiểm tra xem DichMoi có nằm trên đường dẫn đến Rome hay không.

Xét truy vấn:

?- tren\_duong(nha).

Quy trình xử lý của Prolog như sau:

* Tìm các mệnh đề của vị từ tren\_duong, bỏ qua mệnh đề đầu tiên vì rome và nha cùng là biến, không thể đồng nhất được. Xét mệnh đề tiếp:

tren\_duong(Dich) :- kha\_thi(Dich, DichMoi), tren\_duong(DichMoi).

* Đồng nhất bằng cách khởi tạo Dich bằng nha:

tren\_duong(nha) :- kha\_thi(nha, DichMoi), tren\_duong(DichMoi).

kha\_thi(nha, DichMoi), tren\_duong(DichMoi).

* Tìm đến các mệnh đề của kha\_thi và thử đồng nhất chỉ có kha\_thi(nha, benxe) là thành công bằng cách khởi tạo DichMoi bằng benxe.

kha\_thi(nha, benxe), tren\_duong(benxe).

tren\_duong(benxe).

* Lại kiểm tra tương tự:

tren\_duong(benxe) :- kha\_thi(benxe, DichMoi), tren\_duong(DichMoi).

kha\_thi(benxe, DichMoi), tren\_duong(DichMoi).

* Lúc này, DichMoi được gán là rome khi đồng nhất với kha\_thi(benxe, rome):

kha\_thi(benxe, rome), tren\_duong(rome).

tren\_duong(rome).

* Do tren\_duong(rome) có trong cơ sở tri thức, việc kiểm tra hoàn tất. Prolog trả về true (có đường từ nhà tới Rome).

# **Tìm hiểu SWI-Prolog:**

## **Cài đặt:**

Trang chủ của SWI-Prolog nằm ở: <http://www.swi-prolog.org/>

Đường dẫn để tải chương trình SWI-Prolog phiên bản ổn định [stable]: <http://www.swi-prolog.org/download/stable>. Trong đó có cả hướng dẫn tải cho các hệ điều hành khác nhau (Windows, Ubuntu, Mac,…).

Báo cáo sẽ không đi sâu vào cách cài đặt SWI-Prolog, mọi hướng dẫn trên trang chủ và trong bộ cài đặt đều đã ghi đầy đủ.

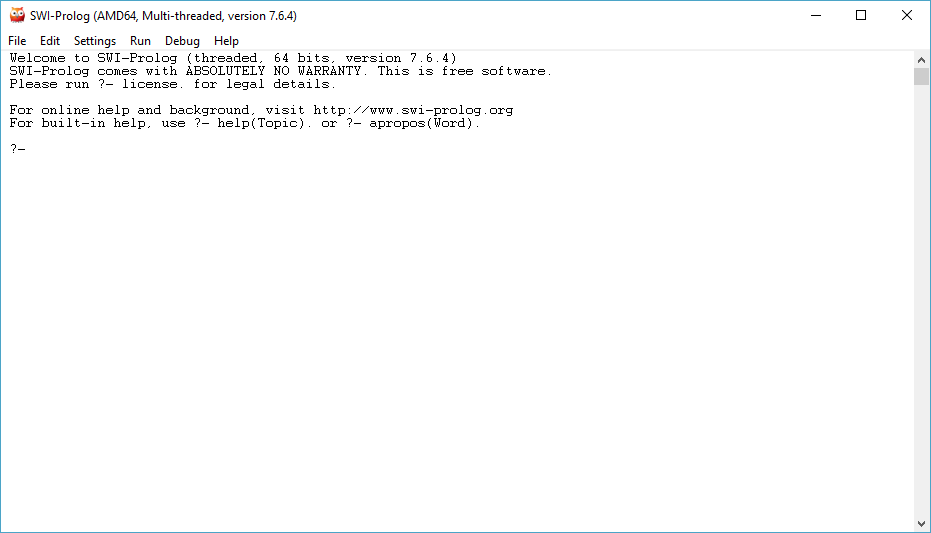
Do người viết sử dụng hệ điều hành Windows, nên báo cáo sẽ viết chủ yếu về cách triển khai trên hệ điều hành Windows. Tuy nhiên, giao diện dòng lệnh [command line] của nó là tương tự ở mọi hệ điều hành nên báo cáo vẫn có tính tham khảo.

## **Sử dụng:**

### **Giao diện chính:**

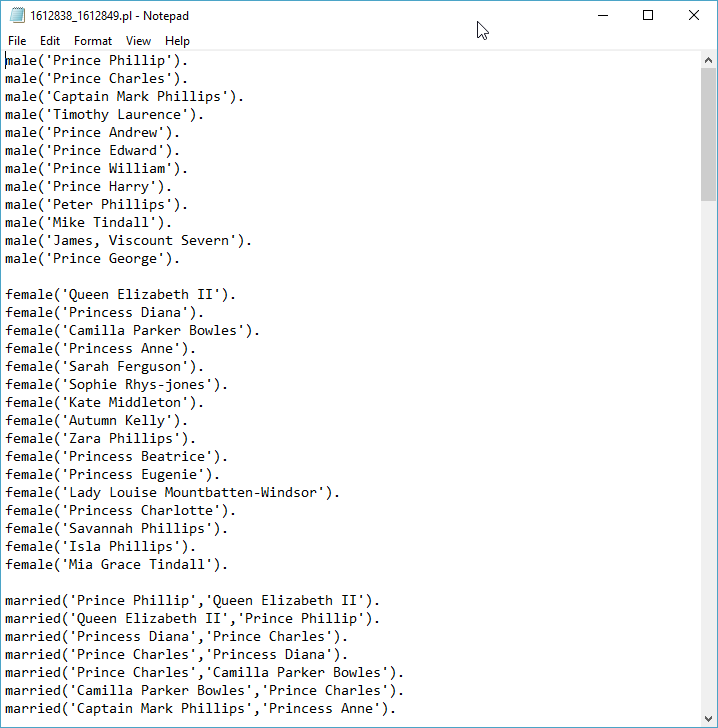
Hình dưới là giao diện chính của SWI-Prolog. Ngoại trừ thanh công cụ (File, Edit,…) là giao diện người dùng trên Windows thì phần còn lại tương tự như giao diện dòng lệnh của Command Prompt trên Windows hay Terminal trên Ubuntu, Mac,…

Người dùng có thể gõ các “câu lệnh” (thực chất là các mệnh đề truy vấn) ở sau ?-.



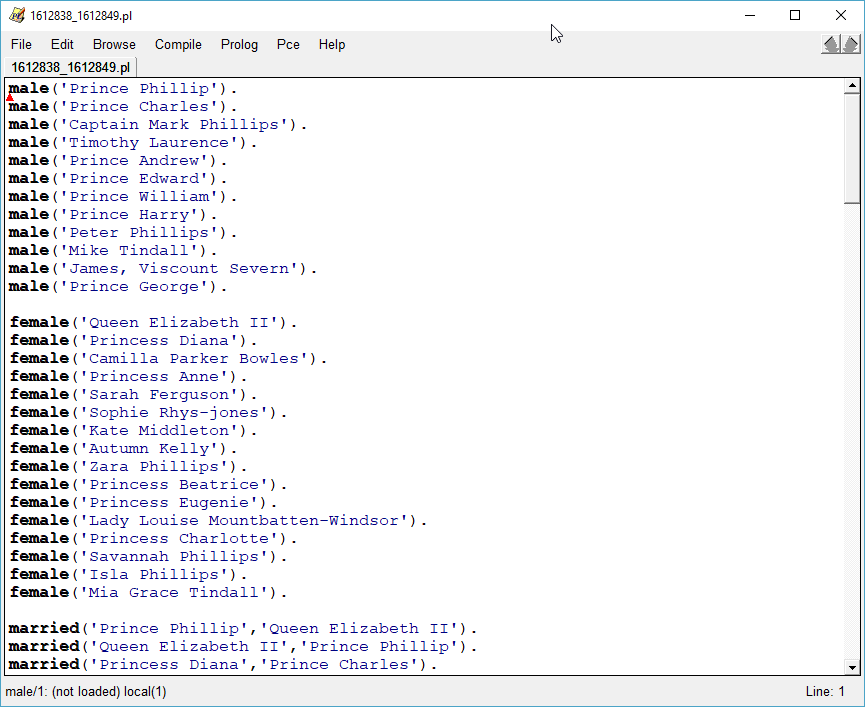
### **Cơ sở tri thức:**

Việc xây dựng cơ sở tri thức được thực hiện trên một tập tin riêng lẻ (có định dạng .pl).



Tập tin gồm nhiều dòng (là các mệnh đề được đánh dấu kết thúc bằng dấu chấm (.)). Mỗi mệnh đề là một sự thật hay một luật (phân cách giữa phần hệ quả và phần tiền đề bằng :-).

Việc chỉnh sửa tập tin có thể thực hiện bằng bất kì chương trình hiệu chỉnh văn bản [text editor] nào, ví dụ: Notepad, Notepad++, Sublime Text,… hoặc môi trường tích hợp phát triển [Intergrated Development Environment] do SWI-Prolog cung cấp:



Môi trường này được mở bằng cách: từ giao diện chính, chọn File -> New để tạo tập tin cơ sở tri thức mới hoặc File -> Edit… để chỉnh sửa tập tin cơ sở tri thức đã có.

Một cách khác để xây dựng cơ sở tri thức là sử dụng các vị từ có sẵn trong màn hình dòng lệnh:

assert(<fact>): định nghĩa một sự thật mới

assert((<rule>)): định nghĩa một luật mới (lưu ý có 2 cặp ngoặc)

Tuy nhiên, cách này không được khuyến khích do khó kiểm soát.

### **Tham vấn:**

Sau khi xây dựng cơ sở tri thức, ta sử dụng bằng cách tham vấn [consult] nó. Có ba cách để làm việc này nếu trên hệ điều hành Windows.

Cách 1: Từ giao diện chính, chọn File -> Consult… và dẫn đến tập tin chứa cơ sở tri thức.

Cách 2: Bằng dòng lệnh, gõ:

consult(<dir>) với <dir> là đường dẫn đến tập tin cơ sở tri thức đặt trong ngoặc đơn ‘…’

reconsult(<dir>) với <dir> như trên

Trong đó, trong trường hợp có hai mệnh đề cùng vị từ và cùng bậc, reconsult sẽ ghi đè lên mệnh đề cũ, còn consult sẽ tạo thành mệnh đề mới.

Cách 3: Mở trực tiếp tập tin .pl bằng Prolog.

### **Truy vấn:**

Khi đã nạp cơ sở tri thức, ta có thể bắt đầu thực hiện các truy vấn bằng cách gõ lên giao diện dòng lệnh.

## **Ví dụ:**

Sử dụng cơ sở tri thức Cây phả hệ hoàng gia Anh (trình bày ở mục sau).

Ví dụ 1: Truy vấn true/false

Truy vấn đúng trong thực tế (Isla Phillips là nữ)



Truy vấn sai trong thực tế (Prince Phillip là nam)



Truy vấn không chứa tri thức (không có Anh Hào trong cơ sở tri thức này)



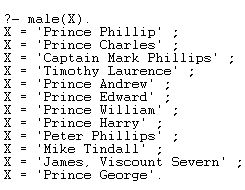
Ta thấy sự nhập nhằng bởi vì hai từ *false* trả về mang hai nghĩa khác nhau.

Ví dụ 2: Truy vấn liệt kê

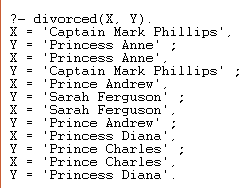
Tìm tất cả con trai:



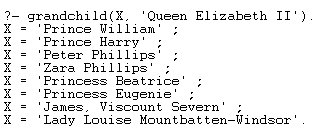
Prolog sẽ hiện 1 phép thế mỗi lần, nếu muốn tìm tất cả cần bấm phím chấm phẩy (;).



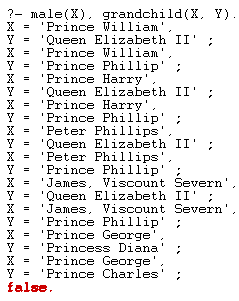
Tìm tất cả cặp đã ly hôn :



Tìm tất cả cháu (quan hệ ông/bà cháu) của nữ hoàng:



Tìm với nhiều vị từ:

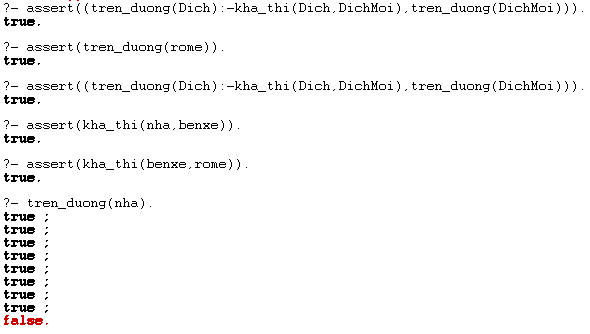


Cách tìm trên tương đương câu hỏi tìm tất cả cặp ông/bà và cháu trai.

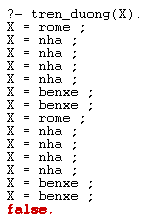
Ví dụ 3: đệ quy

Sử dụng ví dụ trình bày ở trên.

Truy vấn xem từ nhà có thể đến được Rome hay không.



Hoặc truy vấn tất cả những địa điểm trên đường đi.



Sự xuất hiện lặp lại do trong quá trình đệ quy nhiều sự thật được kiểm tra và đều đúng.

# **Cây phả hệ hoàng gia Anh:**

Tập tin ‘1612838\_1612849\_CayPhaHeHoangGiaAnh.pl’ đính kèm cho cơ sở tri thức trên cây phả hệ trong yêu cầu đồ án.

Dưới đây là 20 câu hỏi, cùng truy vấn Prolog tương ứng:

|  |  |  |
| --- | --- | --- |
| **STT** | **Ngôn ngữ tự nhiên** | **Ngôn ngữ Prolog** |
| 1 | Isla Phillips có là nữ? | female('Isla Phillips'). |
| 2 | Timothy Laurence có đang kết hôn với Princess Anne? | married('Timothy Laurence', 'Princess Anne'). |
| 3 | Prince William có là phụ huynh của Savannah Phillips? | parent('Prince William', 'Savannah Phillips'). |
| 4 | Autumn Kelly có là con của Captain Mark Phillips? | child('Autumn Kelly','Captain Mark Phillips'). |
| 5 | Mia Grace Tindall có là cháu gái của Autumn Kelly (quan hệ cô cháu)? | niece('Mia Grace Tindall', 'Autumn Kelly'). |
| 6 | Mike Tindall có là cháu của Queen Elizabeth II (quan hệ bà cháu)? | grandchild('Zara Phillips','Queen Elizabeth II'). |
| 7 | Sarah Ferguson có đã ly hôn với Timothy Laurence? | divorced('Sarah Ferguson','Timothy Laurence'). |
| 8 | Ai là mẹ của Princess Beatrice? | mother(X,'Princess Beatrice'). |
| 9 | Ai là ông của Mia Grace Tindall? | grandfather(X,'Mia Grace Tindall'). |
| 10 | Ai đang kết hôn với Prince William? | married(X,'Prince William'). |
| 11 | Princess Anne có đã từng ly hôn với ai không, cho biết người đó. | divorced(X, 'Princess Anne'). |
| 12 | Captain Mark Phillips đang kết hôn với ai? | married(X, 'Captain Mark Phillips'). |
| 13 | Kể tên tất cả những người là nam trong cây phả hệ. | male(X). |
| 14 | Kể tên tất cả con của Queen Elizabeth II. | child(X, 'Queen Elizabeth II'). |
| 15 | Kể tên tất cả cháu gái của Prince Phillip (quan hệ ông cháu). | granddaughter(X, 'Prince Phillip'). |
| 16 | Kể tên hai phụ huynh của Lady Louise Mountbatten-Windsor. | parent(X, 'Lady Louise Mountbatten-Windsor'). |
| 17 | Kể tên các cặp vợ chồng trong cây phả hệ. | married(X,Y). |
| 18 | Kể tên các cặp cô-cháu trong cây phả hệ. | aunt(X, Y). |
| 19 | Kể tên các cặp đã ly hôn trong cây phả hệ. | divorced(X, Y). |
| 20 | Kể tên các cặp anh chị em trong cây phả hệ. | sibling(X,Y). |

Do kết quả truy vấn khá dài, xin phép không trình bày ở đây. Các câu truy vấn có thể được tìm thấy ở tập tin ‘1612838\_1612849\_CayPhaHeHoangGiaAnh\_TruyVan.txt’.

# **Cơ sở tri thức từ điển đa ngôn ngữ:**

Từ vị từ cơ bản:

**word(X, Spelling, Language, PartOfSpeech, Pronunciation, Definition).**

**synonymous\_(Def1, Def2).**

**antonymous\_(Def1, Def2).**

**tense(Word1, Word2, Tense).**

**relative\_(Word1, Word2).**

**direct\_derive(Word1, Word2).**

Trong đó:

* **word** tức là có từ X, được đánh vần là Spelling, trong ngôn ngữ Language, thuộc loại từ PartOfSpeech (động từ [verb], danh từ [noun], tính từ [adjective],…), có cách phát âm Pronunciation và có nghĩa là Definition.
* **synonymous\_** ý chỉ hai định nghĩa Def1 và Def2 được coi là tương đồng
* **antonymous\_** ý chỉ hai định nghĩa Def1 và Def2 được coi là trái nghịch
* **tense** ý chỉ Word2 là Word1 trong thì Tense
* **relative\_** ý chỉ hai từ Word1 và Word2 có mối quan hệ “bà con” với nhau
* **direct\_derive** ý chỉ Word1 là con/có nguồn gốc trực tiếp của Word2

Ta xây dựng thêm các vị từ:

|  |  |  |
| --- | --- | --- |
| **#** | **Vị từ** | **Ý nghĩa** |
| 1 | **pronounce(Word, Pronunciation)** | từ Word được phát âm là Pronunciation |
| 2 | **mean(Word, Definition)** | từ Word có nghĩa là Definition |
| 3 | **type(Word, PartOfSpeech)** | từ Word thuộc loại từ PartOfSpeech |
| 4 | **in\_language(Word, Language)** | từ Word được viết bằng tiếng Language |
| 5 | **synonymous(Def1, Def2)** | hai định nghĩa Def1, Def2 là (gián tiếp/thật sự) tương đương (vd: Def1 đồng nghĩa DefX đồng nghĩa Def2 -> Def1 đồng nghĩa Def2) |
| 6 | **antonymous(Def1, Def2)** | hai định nghĩa Def1, Def2 là (gián tiếp/thật sự) trái nghĩa (vd: Def1 trái nghĩa với DefX đồng nghĩa Def2 -> Def1 trái nghĩa Def2) |
| 7 | **synonym(Word1, Word2)** | hai từ Word1, Word2 là hai từ đồng nghĩa |
| 8 | **antonym(Word1, Word2)** | hai từ Word1, Word2 là hai từ trái nghĩa |
| 9 | **homophone(Word1, Word2)** | hai từ Word1, Word2 là hai từ đồng âm |
| 10 | **homograph(Word1, Word2)** | hai từ Word1, Word2 là hai từ cùng cách đánh vần |
| 11 | **heteronym(Word1, Word2)** | hai từ Word1, Word2 là hai từ cùng cách đánh vần nhưng phát âm khác nhau |
| 12 | **same\_word\_different\_language(Word1, Word2)** | hai từ Word1, Word2 cùng chỉ một thứ nhưng ở hai ngôn ngữ khác nhau |
| 13 | **same\_language(Word1, Word2)** | hai từ Word1, Word2 thuộc cùng một ngôn ngữ |
| 14 | **same\_part\_of\_speech(Word1, Word2)** | hai từ Word1, Word2 có cùng loại từ |
| 15 | **is\_word(X)** | X là một từ |
| 16 | **is\_language(X)** | X là một ngôn ngữ |
| 17 | **is\_part\_of\_speech(X)** | X là một loại từ |
| 18 | **is\_pronunciation(X)** | X là một phiên âm |
| 19 | **is\_definition(X)** | X là một định nghĩa |
| 20 | **is\_verb(X)** | X là một động từ |
| 21 | **is\_adjective(X)** | X là một tính từ |
| 22 | **is\_noun(X)** | X là một danh từ |
| 23 | **is\_english(X)** | X là từ tiếng Anh |
| 24 | **is\_greek(X)** | X là từ tiếng Hy Lạp |
| 25 | **is\_latin(X)** | X là từ latin |
| 26 | **translate(Word1, Language1, Word2, Language2)** | Word1 trong tiếng Language1 được dịch thành Word2 trong tiếng Language2 |
| 27 | **is\_tense(X)** | X là một thì (hiện tại, quá khứ, quá khứ hoàn thành,…) |
| 28 | **same\_word\_different\_tense(Word1, Word2)** | Word1 và Word2 cùng là một từ (động từ) nhưng với thì khác nhau |
| 29 | **relative(Word1, Word2)** | hai từ Word1, Word2 có bà con thật sự |
| 30 | **verb\_relative(Word, VerbWord)** | từ Word có từ bà con thuộc loại động từ là VerbWord |
| 31 | **noun\_relative(Word, NounWord)** | từ Word có từ bà con thuộc loại danh từ là NounWord |
| 32 | **adj\_relative(Word, AdjWord)** | từ Word có từ bà con thuộc loại tính từ là AdjWord |
| 33 | **is\_base(Word)** | từ Word là một gốc từ (làm gốc để sinh ra các từ khác) |
| 34 | **indirect\_derive(Word, WordFrom)** | từ Word có nguồn gốc gián tiếp (không phải con trực tiếp) của WordFrom |
| 35 | **direct\_originate(Origin, Word)** | từ Origin là từ cha trực tiếp (ngay trước) của từ Word |
| 36 | **indirect\_originate(Origin, Word)** | từ Origin là từ cha gián tiếp của từ Word |
| 37 | **base(Word, BaseWord)** | từ Word có gốc từ là BaseWord |
| 38 | **closest\_base(Word, BaseWord)** | từ Word có gốc từ gần nhất là BaseWord (do BaseWord cũng có thể có gốc từ) |
| 39 | **same\_base(Word1, Word2)** | hai từ Word1, Word2 có cùng gốc từ |
| 40 | **common\_ancestor(Word1, Word2, WordAncestor)** | hai từ Word1, Word2 có cùng tổ tiên (có thể không là gốc từ) là WordAncestor (có thể có nhiều hơn một tổ tiên) |
| 41 | **lowest\_common\_ancestor(Word1, Word2, WordAncestor)** | hai từ Word1, Word2 có cùng tổ tiên (có thể không là gốc từ) là WordAncestor, và đó là tổ tiên gần nhất |
| 42 | **language\_oriented\_derive(Word, WordFrom, Language)** | từ Word bắt nguồn từ WordFrom viết trong ngôn ngữ Language |
| 43 | **greek\_derive(Word, WordFrom)** | từ Word bắt nguồn từ WordFrom trong tiếng Hy Lạp |
| 44 | **latin\_derive(Word, WordFrom)** | từ Word bắt nguồn từ WordFrom trong tiếng Latin |

Dưới đây là 20 câu hỏi, cùng truy vấn Prolog tương ứng:

|  |  |  |
| --- | --- | --- |
| **STT** | **Ngôn ngữ tự nhiên** | **Ngôn ngữ Prolog** |
| 1 | ‘English’ có phải là một ngôn ngữ? | is\_language(‘English’). |
| 2 | ‘go’ và ‘gone’ có phải là cùng một từ chỉ khác thì? | same\_word\_different\_tense(‘go’, ‘gone’). |
| 3 | ‘nation’ và ‘national’ có quan hệ bà con với nhau? | relative(‘nation’, ‘national’). |
| 4 | ‘cool’ có phải là tính từ bà con với ‘cold’? | adj\_relative(‘cool’, ‘cold’). |
| 5 | ‘cool’ trong tiếng Anh có phải là ‘chaud’ trong tiếng Pháp? | translate(‘cool’,’English’,’chaud’,’French’). |
| 6 | Cho biết cách phát âm của từ ‘chaud’. | pronounce(‘chaud’,X). |
| 7 | Liệt kê mọi động từ đã biết. | is\_verb(X). |
| 8 | Dịch từ ‘froid’ trong tiếng Pháp ra tiếng Anh. | translate(‘froid’,’French’,X,’English’). |
| 9 | Liệt kê mọi cặp từ đồng nghĩa đã biết. | synonym(X, Y). |
| 10 | Gốc từ của ‘flow’ là gì? | base(‘flow’,X). |
| 11 | ‘flow’ và ‘rhyme’ có cùng tổ tiên? Cho biết (các) tổ tiên đó. | common\_ancestor(‘flow’,’rhyme’,X). |
| 12 | Cho biết mọi từ trái nghĩa với ‘hot’. | antonym(‘hot’,X). |
| 13 | Cho biết mọi từ bắt nguồn của ‘flow’. | indirect\_originate(X, ‘flow’). |
| 14 | Cho biết từ cha trực tiếp của ‘rhyme’ và ngôn ngữ của nó. | direct\_originate(X,’rhyme’), in\_language(X, Y). |
| 15 | Cho biết tổ tiên chung gần nhất của ‘flow’ và ‘rhyme’. | lowest\_common\_ancestor(‘flow’,’rhyme’,X). |
| 16 | Cho biết các từ con (gián tiếp) của ‘\*sreu-’. | indirect\_originate(‘\*sreu-‘,X). |
| 17 | Cho biết các từ bà con của ‘nationality’ và loại từ của nó. | relative(X, ‘nationality’), type(X, Y). |
| 18 | Cho biết nghĩa của từ ‘chaud’. | synonym(‘chaud’,X). |
| 19 | Cho biết từ nguyên trong tiếng Latin của ‘rhyme’. | latin\_derive(‘rhyme’,X). |
| 20 | ‘flow’ có bắt nguồn từ ‘fluxus’? | indirect\_derive(‘flow’,’fluxus’). |

Do kết quả truy vấn khá dài, xin phép không trình bày ở đây. Các câu truy vấn có thể được tìm thấy ở tập tin ‘1612838\_1612849\_TuDien\_TruyVan.txt’.