

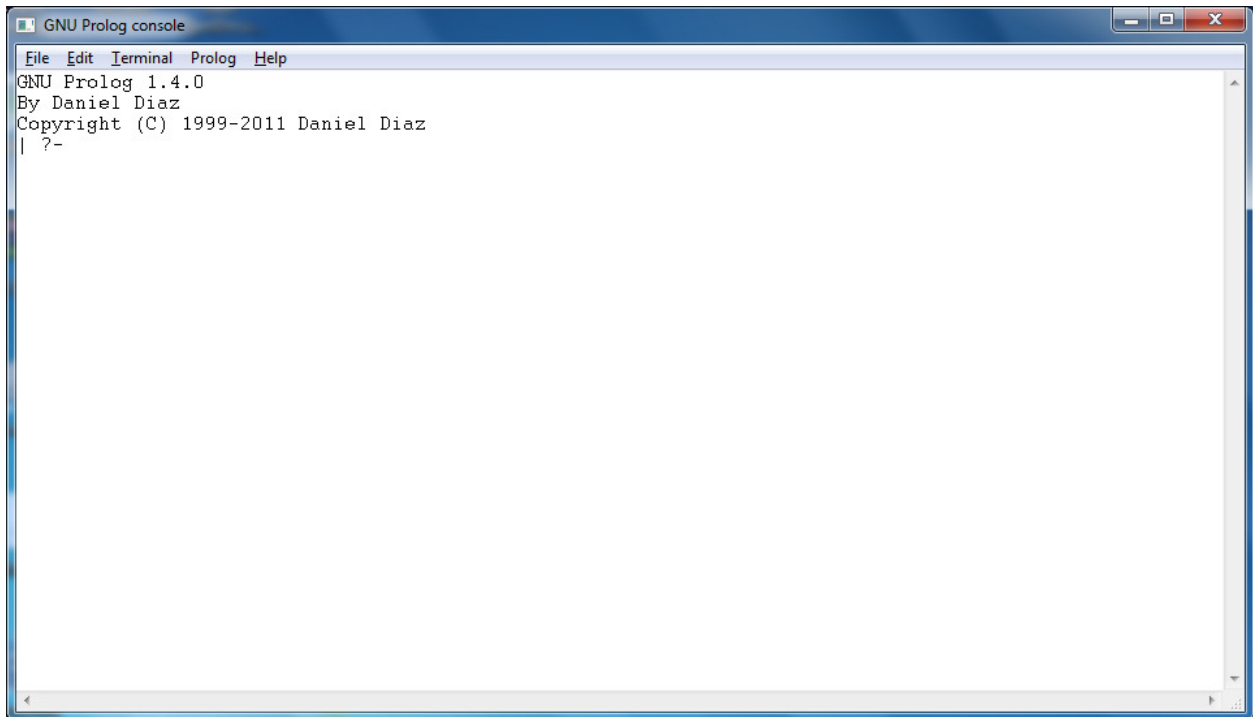
HƯỚNG DẪN LẬP TRÌNH PROLOG

Phạm Nguyên Khang

1. Bắt đầu

1.1. Tải về và cài đặt

Phiên bản Prolog được sử dụng trong tài liệu này là GNU Prolog 1.4.0 của tác giả Daniel Diaz. Chương trình có thể được tải về từ <http://www.gprolog.org/#download>. Chọn phiên bản phù hợp với hệ thống và thực hiện cài đặt. Giao diện của GNU Prolog như hình 1.



Hình 1. Giao diện của GNU Prolog

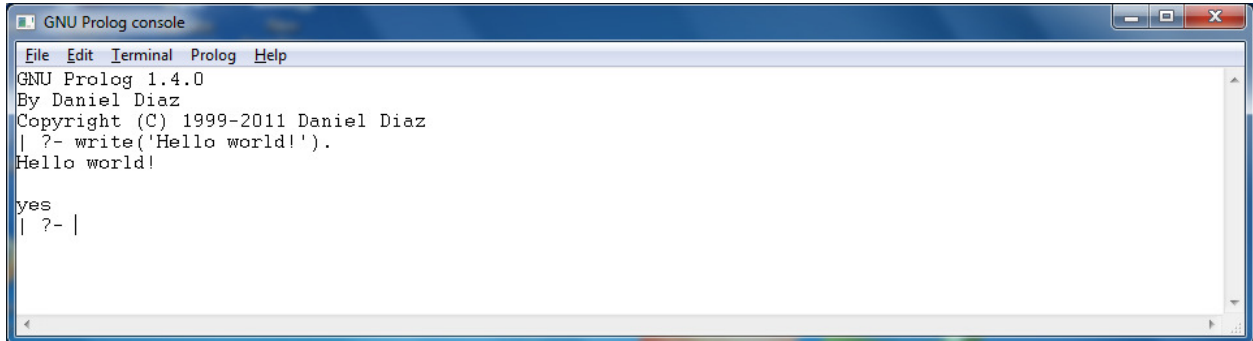
Có 2 cách để lập trình thực thi chương trình trên Prolog:

- Lập trình trực tiếp:
 - Gõ **[user]**, từ dấu nhắc của Prolog và ấn phím ENTER.
 - Lần lượt gõ các mệnh đề.
 - Sau khi kết thúc ấn tổ hợp phím Ctrl + D
- Soạn thảo trên một trình soạn thảo khác & nạp chương trình:
 - Sử dụng một chương trình soạn thảo văn bản bất kỳ, viết và lưu chương trình. Có thể sử dụng trình soạn thảo Notepad++ (<http://notepad-plus-plus.org/>).
 - Nạp chương trình lên Prolog
 - Từ dấu nhắc, gõ **consult('ten_file')**.

- Hoặc chọn menu **File/Consult...**

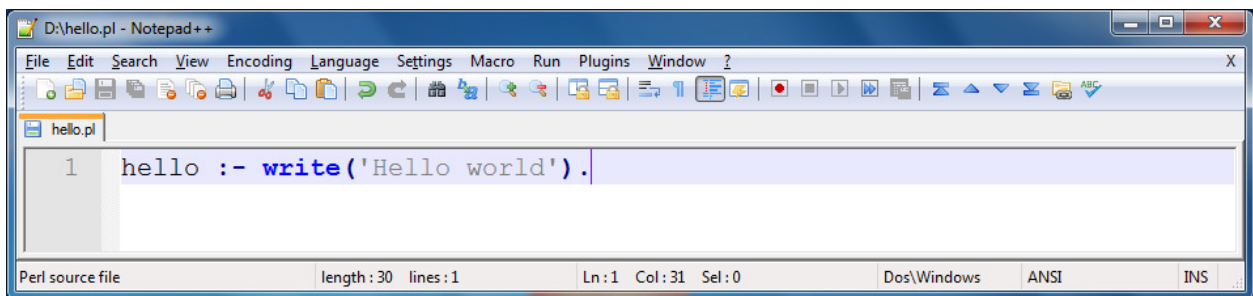
1.2. Chương trình Prolog đầu tiên

Từ dấu nhắc, gõ **write('Hello world!')**. Chương trình sẽ in chuỗi **Hello world!** Lên màn hình.



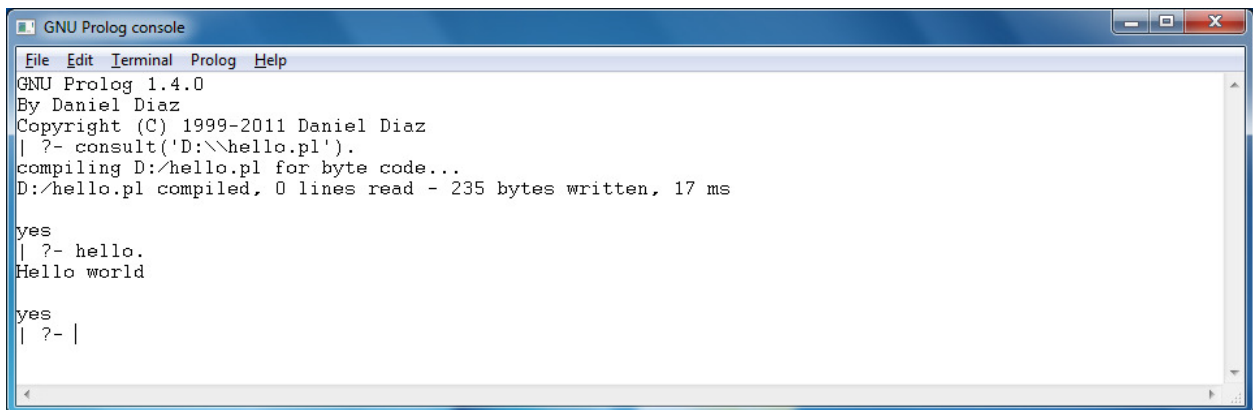
Hình 2. In chuỗi **Hello world!** ra màn hình.

Giờ ta thử một cách khác để lập trình. Mở Notepad++ hoặc bất kỳ trình soạn văn bản nào mà bạn có, soạn một chương trình có nội dung như sau:



Hình 3. Soạn chương trình Prolog đầu tiên.

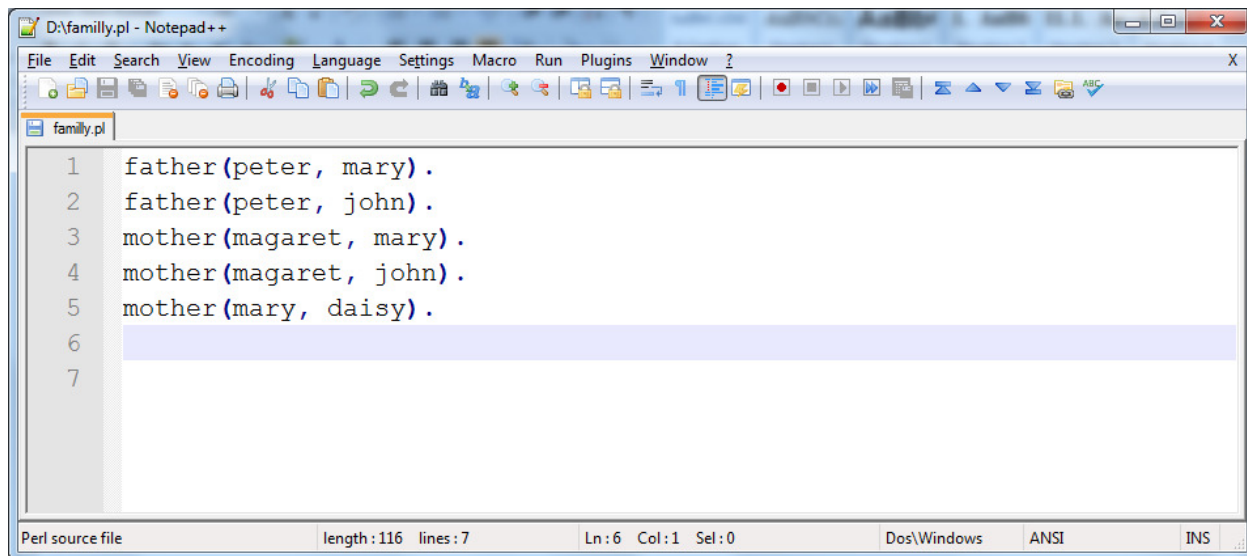
Lưu chương trình vào file **D:\hello.pl**. Để nạp chương trình lên Prolog từ dấu nhắc gõ: **consulte('D:\\hello.pl')** hoặc **consulte('D:/hello.pl')**. Để thực thi những gì ta viết trong chương trình, gõ: **hello.** (có dấu chấm cuối chữ hello). Kết quả chương trình như hình 4.



Hình 4. Kết quả của chương trình 'hello.pl'.

1.3. Truy vấn trên Prolog

Soạn thảo chương trình có nội dung như hình 5 và nạp chương trình lên Prolog.

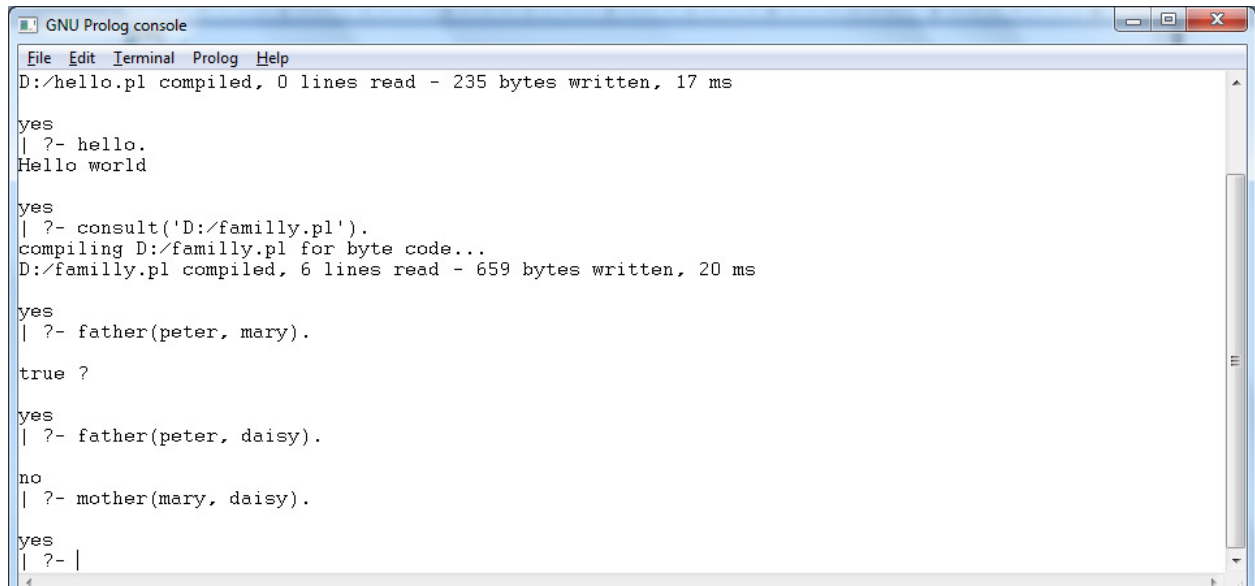


Hình 5. Chương trình 'family.pl'.

Bây giờ, ta có thể truy vấn Prolog theo kiểu **đúng/sai** để kiểm tra xem *peter* có phải là cha của *mary* không; *peter* có phải là cha của *daisy* không, *mary* có phải là mẹ của *daisy* không. Từ đầu nhắc lần lượt gõ (ấn phím ENTER sau mỗi câu truy vấn):

- father(peter, mary).
- father(peter, daisy).
- mother(mary, daisy).

Kết quả thu được như hình 6.



```
GNU Prolog console
File Edit Terminal Prolog Help
D:/hello.pl compiled, 0 lines read - 235 bytes written, 17 ms

yes
| ?- hello.
Hello world

yes
| ?- consult('D:/family.pl').
compiling D:/family.pl for byte code...
D:/family.pl compiled, 6 lines read - 659 bytes written, 20 ms

yes
| ?- father(peter, mary).

true ?

yes
| ?- father(peter, daisy).

no
| ?- mother(mary, daisy).

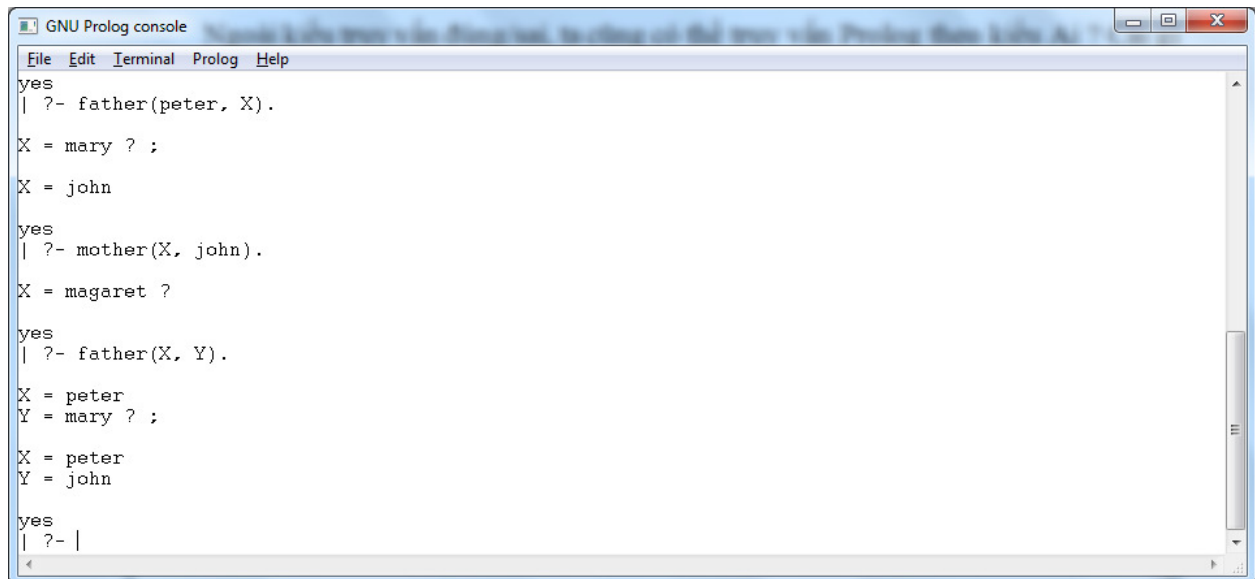
yes
| ?- |
```

Hình 6. Truy vấn đúng/sai với Prolog.

Ngoài kiểu truy vấn đúng/sai, ta cũng có thể truy vấn Prolog theo kiểu Ai ? Cái gì ? Ví dụ: ta có thể “hỏi” Prolog xem *peter* là cha của **ai** ?; **ai** là mẹ của *john* ? hoặc phức tạp hơn: **ai** là cha của **ai** ? Lần lượt gõ các câu truy vấn sau:

- father(peter, X).
- mother(X, john).
- father(X, Y).

Kết quả như hình 7. Với những câu hỏi có nhiều câu trả lời, để xem câu trả lời kế tiếp, gõ dấu chấm phẩy (;) sau mỗi câu trả lời.



```
GNU Prolog console
File Edit Terminal Prolog Help
yes
| ?- father(peter, X).
X = mary ? ;
X = john

yes
| ?- mother(X, john).
X = magaret ?

yes
| ?- father(X, Y).
X = peter
Y = mary ? ;
X = peter
Y = john

yes
| ?- |
```

Hình 7. Truy vấn “ai ?/cái gì ?” với Prolog.

Tổng kết:

- Tải GNU Prolog và cài đặt theo hướng dẫn
- Có 2 cách để lập trình trên Prolog
 - Trực tiếp
 - Gián tiếp trên một trình soạn thảo khác sau đó nạp chương trình bằng lệnh consult.
- Có 2 kiểu truy vấn trên Prolog
 - Truy vấn đúng/sai, kết quả trả về là đúng hoặc sai
 - Truy vấn ai? cái gì?, kết quả trả về là giá trị bất kỳ

2. Nguyên lý lập trình trên Prolog

2.1. Vị từ (predicate) - Tư duy lập trình và định nghĩa vấn đề trên Prolog

Đối với Prolog, một chương trình có thể hiểu như là các tri thức được người lập trình cung cấp cho hệ thống Prolog. Nhờ vào các kiến thức được cung cấp, hệ thống có thể trả lời được các câu hỏi được đặt ra, và câu trả lời có thể đạt được nhờ cơ chế suy luận của hệ thống dựa trên những kiến thức được cung cấp ban đầu.

Đơn vị kiến thức mà người lập trình cung cấp cho Prolog gọi là các *vị từ* (predicate).

Vị từ dùng để biểu diễn các khái niệm mà người lập trình muốn hệ thống dùng để suy luận để đạt được các kiến thức khác mà mình mong muốn.

Về mặt kỹ thuật, vị từ có thể được xem như hàm, nhưng giá trị trả về chỉ có thể là các giá trị luận lý - đúng hoặc sai. Và giá trị trả về này chỉ có thể sử dụng để suy luận, Prolog không có cơ chế định nghĩa chồng cho các hàm như các ngôn ngữ thủ tục khác (overriding function: cơ chế cho phép nhiều hàm có cùng tên nhưng khác tham số), chính điều này sẽ làm những người quen với việc lập trình thủ tục gặp khó khăn khi bước đầu lập trình với Prolog.

Công việc đầu tiên khi lập trình trên Prolog là định nghĩa các vị từ - các khái niệm mà mình cần cung cấp cho chương trình.

Xét các ví dụ sau:

VD1: Biểu diễn tri thức *Mọi người đều phải chết. Socrates là người.*

Yêu cầu: Chúng ta muốn hệ thống phải có khả năng suy luận và trả lời được các vấn đề liên quan đến các khái niệm trên: ai là người, ai không là người, ai phải chết, ai không phải chết. Ở đây chúng ta có một sự suy luận thông minh đặc trưng cho sức mạnh của Prolog: hệ thống sẽ tự động suy luận rằng Socrates phải chết (điều không được cung cấp ban đầu).

Để biểu diễn các vấn đề trên bằng ngôn ngữ Prolog, chúng ta cần phải xác định cần phải biểu diễn những khái niệm gì.

Trong vấn đề này chúng ta có hai khái niệm cần biểu diễn: một thực thể nào đó có thể là người (hoặc không), và một thực thể nào đó có thể chết. Như vậy chúng ta biểu diễn vấn đề đầu tiên theo kiểu ngôn ngữ Prolog có dạng như sau:

nguoi(X)

với X là một biến hay một hằng nào. Chúng ta sẽ trở lại vấn đề này sau.

Như vậy chúng ta vừa định nghĩa một khái niệm: một đối tượng X nào đó có thể là người, một đối tượng nào đó khác thì không phải là người.

Nếu hiểu như một sự định nghĩa hàm trong các ngôn ngữ thủ tục, chúng ta có thể xem như định nghĩa một hàm mang tên *nguoi*, hàm này có 1 tham số là X (kiểu bất kỳ, GNU Prolog không cần khai báo kiểu của các biến), và kết quả của hàm này, không cần phải khai báo thuộc về kiểu gì. Đối với Prolog, kết quả của một vị từ chỉ có thể là đúng hoặc sai, nên kiểu của nó chỉ có thể là bool. Nhiệm vụ của Prolog là phải trả lời được với giá trị X nhập vào nào, thì hàm này cho ra kết quả đúng hoặc sai, tức là đối tượng X ấy có phải là người hay không. Prolog chỉ có thể làm được điều này nếu như chúng ta cung cấp cho hệ thống một cơ chế suy luận đúng đắn, tức là giải thích được cho Prolog hiểu như thế nào là người?

Tương tự như vậy, chúng ta định nghĩa về vấn đề một thực thể nào đó phải chết bằng vị từ sau:

chet(X)

Như vậy với bài toán đã nêu, chúng ta sẽ đặt ra hai vị từ:

nguoi(X)

chet(X)

VD2: *Tính giá trị giai thừa của một số nguyên bất kỳ.*

Bài toán trên không cho biết dữ kiện ban đầu. Chúng ta phải cung cấp các dữ kiện ban đầu, để Prolog có thể dựa vào đó để suy luận, để từ đó hệ thống có thể giải quyết được yêu cầu của chúng ta. Việc cung cấp dữ kiện ban đầu cho hệ thống là rất quan trọng quyết định vấn đề giải quyết yêu cầu của chúng ta.

Một trong những cách giải quyết có thể được lựa chọn là chúng ta sẽ cho hệ thống biết giá trị giai thừa của toàn bộ số nguyên: giai thừa của 0 là 1, giai thừa của 1 là 1, giai thừa của 2 là 2, giai thừa của 3 là 6, giai thừa của 4 là 24... Dễ dàng nhận thấy rằng cách này là không khả thi, và trong thực tế, con người cũng không tiếp thu tri thức theo cách này.

Chúng ta có thể cung cấp dữ kiện cho hệ thống theo cách khác: giai thừa của một số là tích các số từ 1 đến số đó.

Như vậy với cách giải quyết này, chúng ta có hai khái niệm cần phải cung cấp: giai thừa của một số là gì, và tích của các số nguyên tính từ 1 đến một số là gì?

Cách đặt vấn đề này có thể giải quyết được bài toán, tuy nhiên chúng ta có thể đặt vấn đề theo một cách khác đơn giản, và hợp với tinh thần của Prolog hơn: giai thừa của 0 là 1, và giai thừa của một số lớn hơn 0 là giai thừa của số liền trước nó nhân với chính nó.

Với cách đặt vấn đề này, chúng ta chỉ có một khái niệm phải biểu diễn: giai thừa của một số là gì? (thật ra chúng ta còn một số khái niệm phải đưa ra: một số đứng trước một số là gì, nhân hai số nghĩa là gì, tuy nhiên Prolog đã cung cấp các toán tử để giải quyết vấn đề này. Hiểu theo một nghĩa nào đó, các vấn đề trên là các tiên đề, không cần phải giải thích với hệ thống.)

Nếu quen với ngôn ngữ lập trình thủ tục, chúng ta có khuynh hướng khai báo vị từ diễn tả khái niệm giai thừa như sau:

giaithua(X)

Ở đây cách đặt vấn đề như vậy là không thích hợp với ngôn ngữ Prolog, vì

- Một vị từ chỉ có thể trả lời là đúng hoặc sai, trong khi chúng ta đang mong muốn kết quả trả về theo cách khai báo này một số.\
- Ngôn ngữ Prolog không có định nghĩa chồng hàm, nghĩa là kết quả của hàm (vị từ) không thể dùng như một tham số cho một vị từ khác, trong khi chúng ta đang định dùng kết quả của hàm này để tính tiếp giá trị cho một hàm khác. (Chúng ta định dùng hàm này để tính giai thừa của $n - 1$, rồi nhân tiếp cho n để ra kết quả cuối cùng).

Vị từ thích hợp sẽ được khai báo như sau:

giaithua(X, Y)

Điều này, hiểu theo ngôn ngữ thủ tục, nghĩa là chúng ta khai báo một hàm có tham số là hai số nguyên, và kết quả trả về sẽ là đúng hoặc sai.

Điều chúng ta muốn diễn tả có nghĩa là: giai thừa của một số nguyên (integer) sẽ là một số nguyên khác.

Nếu chúng ta giải thích được cho Prolog hiểu giai thừa của một số nguyên sẽ được tính như thế nào, hệ thống sẽ có khả năng trả lời cho cả *câu hỏi thuận* (giai thừa của một số nguyên là gì), *câu hỏi nghịch* (số nguyên nào có giai thừa bằng số nguyên này), và *ngghi vấn* (giai thừa của một số nguyên X có phải là số nguyên Y hay không).

Tuy nhiên mục đích của chúng ta chỉ cung cấp các dữ kiện để hệ thống có thể trả lời *câu hỏi thuận* (và có thể trả lời thêm *câu hỏi nghi vấn*) mà thôi.

Tóm tắt:

- Lập trình trên Prolog là cung cấp cho hệ thống các khái niệm và diễn giải các khái niệm đó.
- Các khái niệm được cung cấp qua các vị từ.
- Các vị từ có thể xem như các hàm như chỉ trả về giá trị đúng hoặc sai.
- Việc hệ thống có thể trả lời được những câu hỏi nào liên quan đến khái niệm đã cung cấp phụ thuộc vào việc chúng ta diễn giải các khái niệm trên cho hệ thống

2.2. Mệnh đề (clauses) – Cách giải thích các vấn đề trên Prolog

Sau khi đã cung cấp cho hệ thống các khái niệm cần thiết, chúng ta cần phải giải thích các khái niệm mình đã cung cấp, Prolog sẽ dùng các lời giải thích này để thực hiện việc suy luận và trả lời câu hỏi của chúng ta.

Các lời giải thích này được gọi là các mệnh đề (clauses). Có hai dạng mệnh đề: *sự kiện* (fact), và *luật* (rule).

Sự kiện là những điều mà chúng ta công nhận là đúng. Luật là những quy tắc mà chúng ta xác định điều kiện đúng cho chúng.

VD3: hãy viết phần clause cho vị từ *nguoi* đã định nghĩa trong VD1.

Dữ kiện ban đầu chỉ cung cấp cho chúng ta một vấn đề liên quan đến người: *Socrates là*

người. Theo như cách tư duy trong không gian của bài toán, chỉ có một con người duy nhất: *Socrates*. Không ai khác là người.

Như vậy chúng ta sẽ viết phần clause cho vị từ này như sau:

nguoi(socrates).

Chúng ta vừa viết một sự kiện: *socrates là người là điều chắc chắn đúng*. Bất kỳ ký hiệu nào có tên là *socrates* thì *socrates là người* là điều chắc chắn đúng, không cần phải có một điều kiện ràng buộc nào kèm theo.

Lưu ý:

1. Có hai cách viết dạng hằng (literal) cho ký hiệu (symbol) trên Prolog:
 - Một danh biểu (tên biến, tên hàm, tên vị từ) mở đầu bằng ký tự thường (*socrates*, *sOCRATES* ...)
 - Một chuỗi ký hiệu đặt trong cặp dấu nháy đơn ‘,’ (*‘socrates’*, *‘SOCRATES’*, *‘sOCRATES’*, *‘Socrates’*, ...)
2. Một mệnh đề luôn kết thúc bằng dấu chấm (.)

VD4: hãy viết phần mệnh đề cho vị từ *chet* trong VD1.

Dữ kiện ban đầu chỉ cung cấp cho chúng ta một sự kiện liên quan đến vấn đề này: X sẽ phải chết nếu đó là người. Điều này sẽ xác định một quy tắc:

X sẽ chỉ phải chết, tức vị từ sẽ trả về kết quả true, nếu X là người. Vấn đề X nào là người và X nào không là người chúng ta đã đưa ra khái niệm và giải thích cho Prolog trong các ví dụ 1 và 3. Như vậy phần mệnh đề sẽ được viết như sau:

chet(X) :- nguoi(X).

Mệnh đề trên có nghĩa là nếu X là người thì X chết.

Mệnh đề dạng luật (rule) sẽ bao gồm hai phần, nằm ở hai bên cặp ký hiệu ":-". Phần bên trái cho biết vị từ đang được đề cập và các tham số tương ứng. Phần bên phải, xác định điều kiện trả lời đúng cho luật trên, bao gồm các lời gọi các vị từ khác, được ngăn cách bởi ký hiệu ',' (dấu phẩy), gọi là các mệnh đề con (sub-clause). Trong ví dụ trên, chỉ có một mệnh đề con. Một luật chỉ trả lời đúng nếu tất cả các mệnh đề con bên vế phải đều trả lời đúng.

Trong ví dụ trên, chúng ta có một biến X. Tất cả các tham số mở đầu bằng ký tự hoa đều được Prolog hiểu là biến. Biến này là tham số của vị từ *chet*. Kết quả sẽ trả về đúng nếu tất cả mệnh đề con bên vế phải đều trả lời là đúng. Trong trường hợp này, chỉ có một mệnh đề con xác định xem X có phải là người không. Như vậy chúng ta đã biểu diễn được khái niệm một symbol sẽ phải chết nếu symbol đó là người, tức là tất cả những dữ kiện ban đầu được cung cấp.

VD5: Hãy viết phần clause cho vị từ *giaithua* ở VD2.

Từ các dữ kiện được cung cấp (do chúng ta tự cung cấp cho mình để giải bài toán), chúng ta thấy có một sự kiện chắc chắn đúng: *giai thừa của 0 là 1*, và có một luật suy diễn: *giai thừa của n là (n - 1)! * n*.

Chúng ta sẽ viết phần mệnh đề cho vị từ này như sau:

giaithua(0,1).

giaithua(X,Y) :- X > 0, X1 is X-1, giaithua(X1,Y1), Y is Y1*X.

Trước khi hiểu những điều được mô tả trong các ví dụ trên, chúng ta sẽ có một số nhận xét như sau:

1. Trước tiên, chúng ta thấy vị từ *giaithua* được biểu diễn bằng hai mệnh đề: một sự kiện và một luật.
Khi viết nhiều mệnh đề cho một vị từ, các mệnh đề phải được viết liên tiếp nhau (không được xen mệnh đề của vị từ khác vào).
2. Mệnh đề con đầu tiên $X > 0$, được đưa vào để đảm bảo rằng ta chỉ tính giai thừa của các số dương.
3. Hai mệnh đề con tiếp theo: $X1 = X - 1$, *giaithua(X1,Y1)* biểu diễn cho công việc tính giai thừa của $X - 1$. Tuy nhiên chúng ta không được viết **giaithua(X - 1, Y1)**. Vì Prolog chỉ chấp nhận tham số cho các vị từ phải là biến hoặc hằng chứ không được sử dụng biểu thức. Muốn sử dụng giá trị của biểu thức làm tham số đầu vào cho một mệnh đề khác, ta có thể sử dụng một biến tạm (trong trường hợp này là $X1$) và “gán” giá trị của nó bằng biểu thức thông qua phép gán **is**.
4. Chúng ta thấy sự xuất hiện của ký hiệu **is** và sẽ hiểu như mệnh đề con $X1 \text{ is } X - 1$ là phép gán.
5. Phần vị từ trên biểu diễn cho việc sử dụng kỹ thuật lập trình đệ quy, sẽ là sức mạnh lập trình chủ yếu của Prolog. Xem thêm về phần lập trình đệ quy trên Prolog trong các phần sau.

Tóm tắt

- Các khái niệm được mô tả qua các vị từ sẽ được giải thích bằng các mệnh đề.
- Có hai loại mệnh đề: sự kiện và luật.
- Tham số được truyền trong lời gọi các mệnh đề con phải là biến.
- Các kỹ thuật chủ yếu để lập trình trên Prolog là hợp nhất và đệ quy.

2.3. Thực thi chương trình - Đặt câu hỏi và nhận câu trả lời

Đến đây chúng ta đã có thể viết và thực thi các chương trình trên. Sử dụng cách lập trình trực tiếp hoặc gián tiếp được trình bày trong phần 1 để lập trình.

VD6: Viết chương trình hoàn chỉnh cho VD1. Nội dung chương trình nhập hoàn chỉnh cho VD1 như sau:

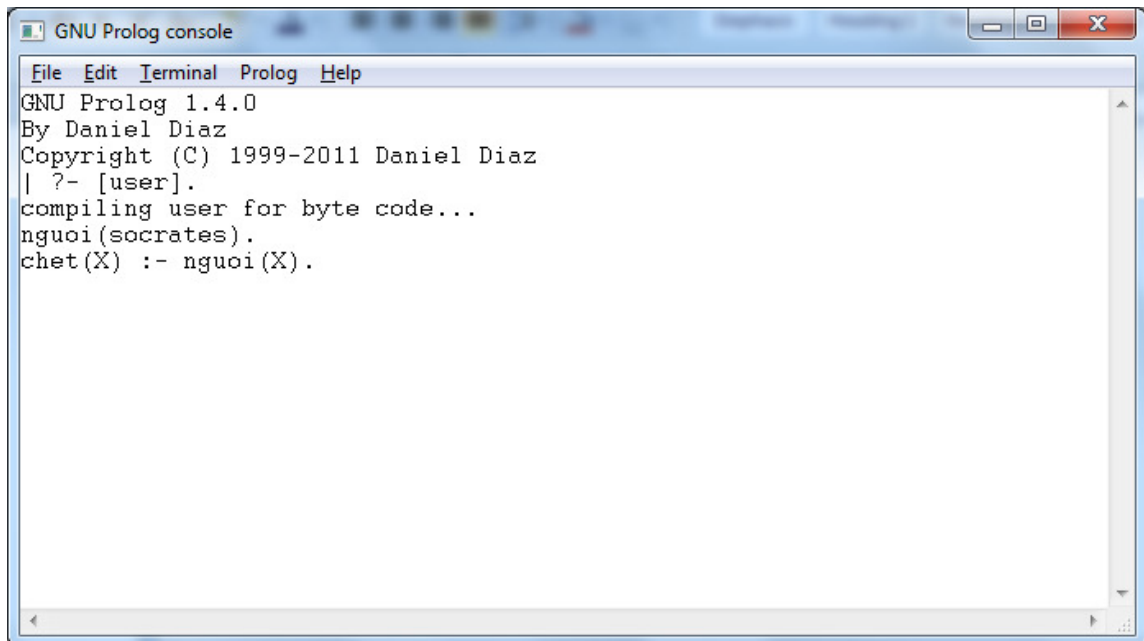
```
nguai(socrates) .  
chet(X) :-nguai(X) .
```

Giả sử ta lập trình trực tiếp. Từ dấu nhắc lệnh, gõ **[user]**.

GNU Prolog sẽ hiện ra dòng thông báo:

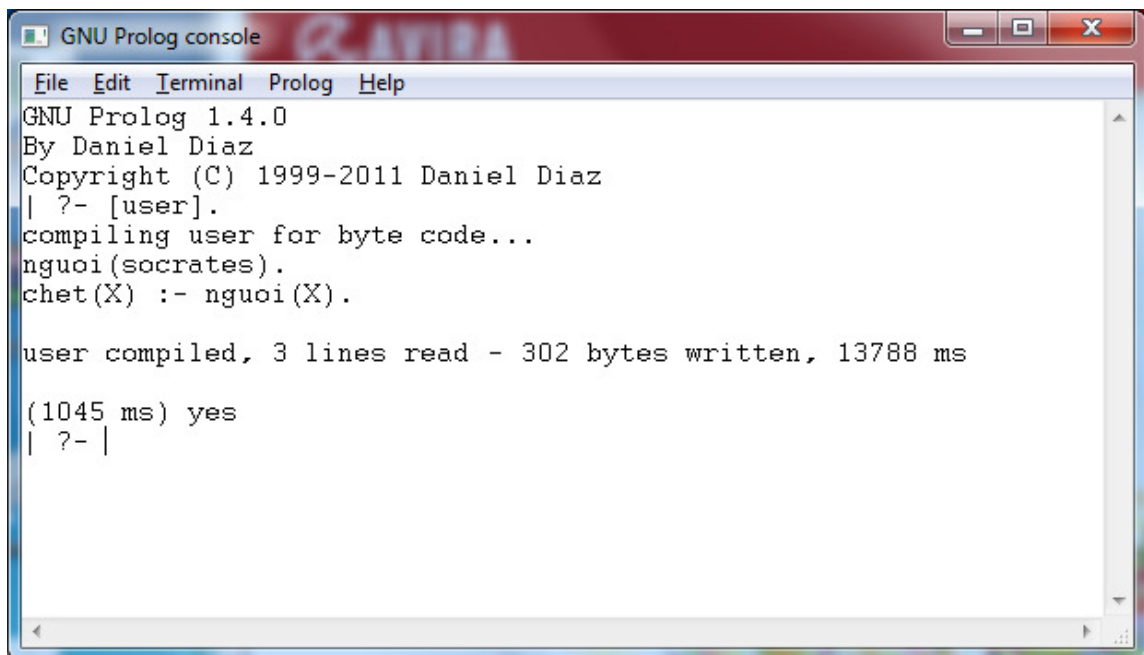
compiling user for byte code...

Sau đó nhập chương trình vào:



```
GNU Prolog 1.4.0
By Daniel Diaz
Copyright (C) 1999-2011 Daniel Diaz
| ?- [user].
compiling user for byte code...
nguoi(socrates).
chet(X) :- nguoi(X).
```

Kết thúc việc soạn thảo bằng phím **Ctrl + D**



```
GNU Prolog 1.4.0
By Daniel Diaz
Copyright (C) 1999-2011 Daniel Diaz
| ?- [user].
compiling user for byte code...
nguoi(socrates).
chet(X) :- nguoi(X).

user compiled, 3 lines read - 302 bytes written, 13788 ms

(1045 ms) yes
| ?- |
```

Để thực thi chương trình, người sử dụng nhập yêu cầu (câu hỏi) của mình cho hệ thống. Yêu cầu này gọi là goal.

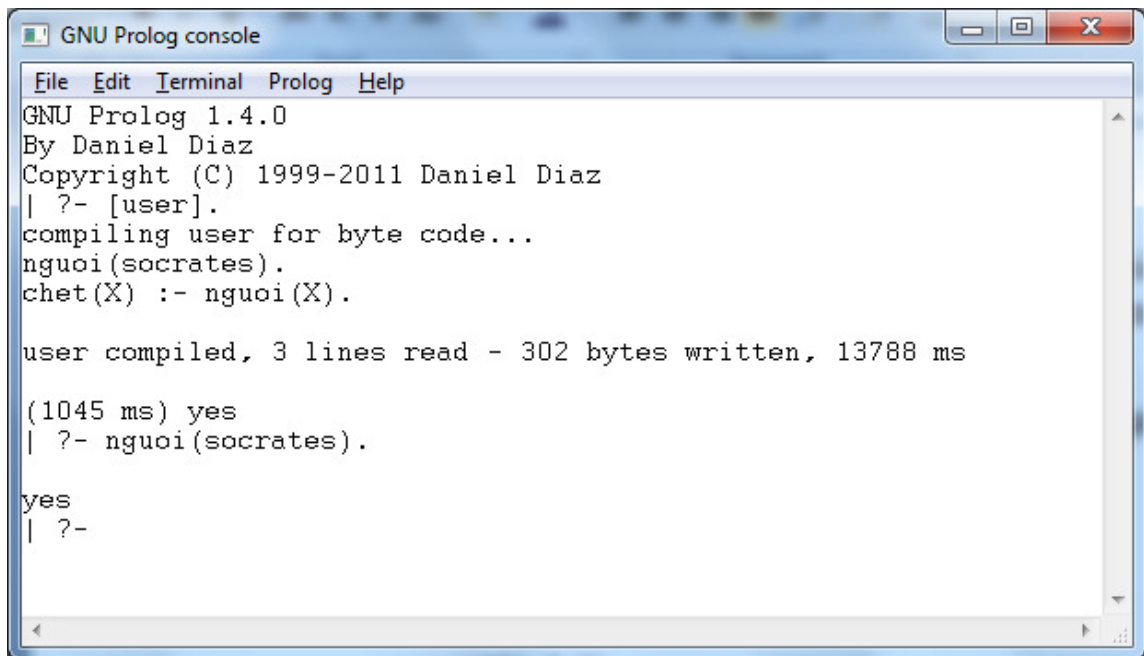
Câu hỏi chúng ta đặt ra cho hệ thống phải chỉ dựa vào các tri thức mà chúng ta đã cung cấp cho hệ thống. Chúng ta đã cung cấp cho hệ thống các khái niệm *nguoi* và *chet*, như vậy chúng ta chỉ có thể đặt các câu hỏi liên quan đến hai khái niệm này.

Ngay sau dấu nhắc, chúng ta có thể nhập câu hỏi như sau:

nguoi(socrates).

Dựa trên tinh thần của của khái niệm, câu phát biểu của chúng ta có nghĩa là "Socrates là người", nên hệ thống sẽ hiểu rằng chúng ta muốn đặt một câu hỏi nghi vấn "Socrates là người phải không ?"

Sau khi ấn Enter, chúng ta sẽ thấy hệ thống có ngay câu trả lời: **yes**



```
GNU Prolog console
File Edit Terminal Prolog Help
GNU Prolog 1.4.0
By Daniel Diaz
Copyright (C) 1999-2011 Daniel Diaz
| ?- [user].
compiling user for byte code...
nguoi(socrates).
chet(X) :- nguoi(X).

user compiled, 3 lines read - 302 bytes written, 13788 ms

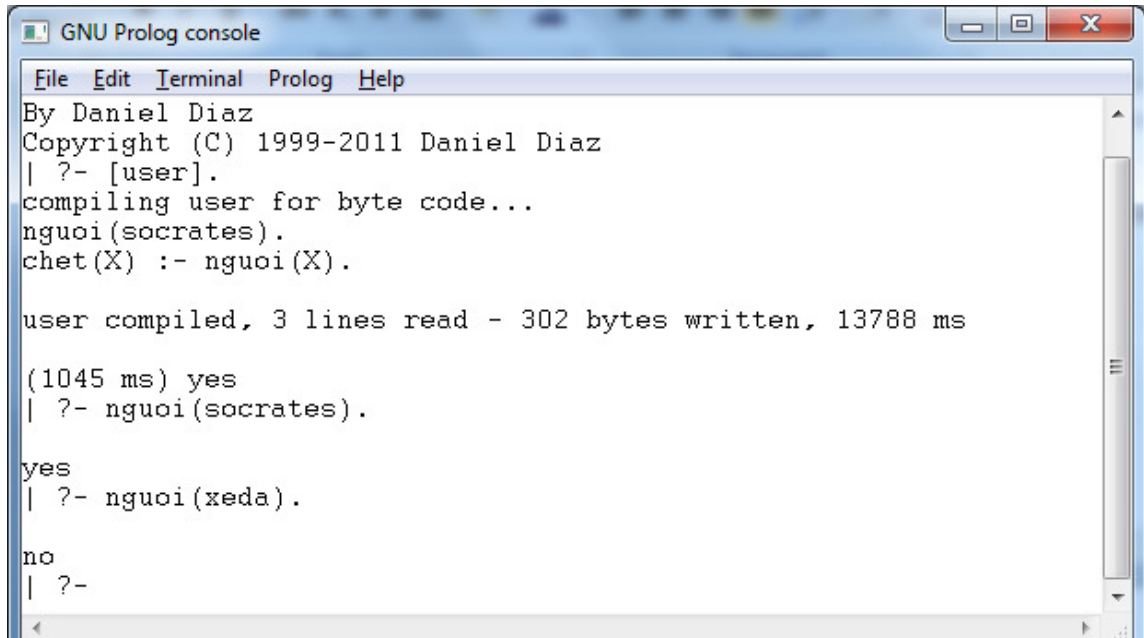
(1045 ms) yes
| ?- nguoi(socrates).

yes
| ?-
```

Thay bằng một tên khác, ví dụ:

nguoi(xeda).

Hệ thống sẽ trả lời No.



```
GNU Prolog console
File Edit Terminal Prolog Help
By Daniel Diaz
Copyright (C) 1999-2011 Daniel Diaz
| ?- [user].
compiling user for byte code...
nguoi(socrates).
chet(X) :- nguoi(X).

user compiled, 3 lines read - 302 bytes written, 13788 ms

(1045 ms) yes
| ?- nguoi(socrates).

yes
| ?- nguoi(xeda).

no
| ?-
```

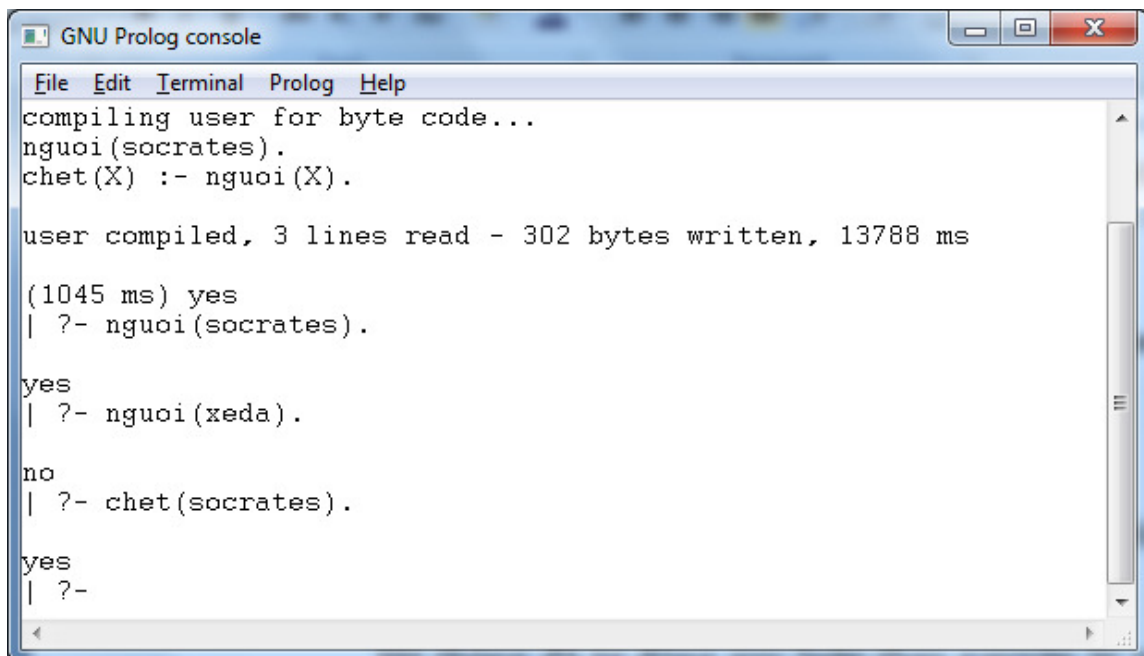
Chúng ta thấy các câu trả lời của hệ thống dựa trên kiến thức mà chúng ta đã cung cấp. Dựa vào những gì mà chúng ta đã cung cấp, hệ thống chỉ biết có một người là Socrates, tất cả những symbol khác đều không phải là người.

Tuy nhiên, với cơ chế suy luận mà chúng ta cung cấp, hệ thống có thể suy luận ra những điều chưa được cung cấp sẵn. Đây chính là điểm tạo nên sức mạnh lập trình của Prolog.

Nhập vào goal như sau:

chet(socrates).

Câu trả lời là: **yes.**



```
GNU Prolog console
File Edit Terminal Prolog Help
compiling user for byte code...
nguoi(socrates).
chet(X) :- nguoi(X).

user compiled, 3 lines read - 302 bytes written, 13788 ms

(1045 ms) yes
| ?- nguoi(socrates).

yes
| ?- nguoi(xeda).

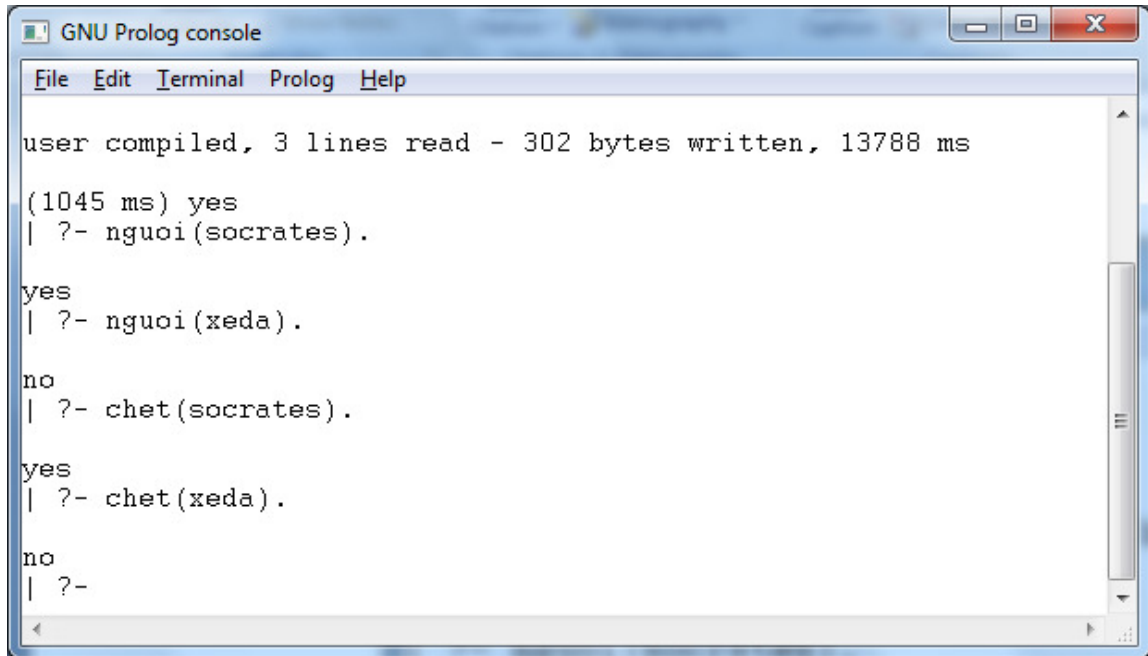
no
| ?- chet(socrates).

yes
| ?-
```

Với một tên người khác:

chet(xeda).

Câu trả lời là: **no**.



```
GNU Prolog console
File Edit Terminal Prolog Help

user compiled, 3 lines read - 302 bytes written, 13788 ms

(1045 ms) yes
| ?- nguoi(socrates).

yes
| ?- nguoi(xeda).

no
| ?- chet(socrates).

yes
| ?- chet(xeda).

no
| ?-
```

Hệ thống đã tự động suy luận theo nguyên lý mà chúng ta muốn nó phải "học": ai là người thì người đó phải chết.

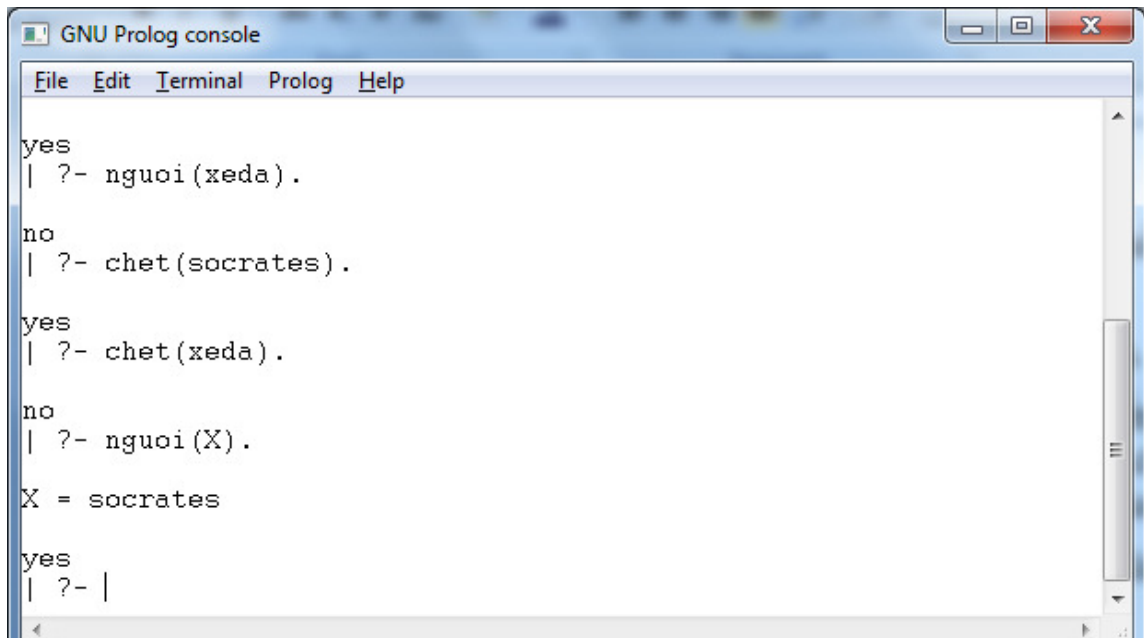
Ngoài những câu hỏi dạng Yes/No, Prolog có thể trả lời các câu hỏi yêu cầu tìm đáp số. Chúng ta nhập vào một goal như sau:

nguoi(X).

Đến đây, trong câu hỏi của chúng ta có một biến: **X** (nhắc lại: mọi danh biểu mở đầu là ký tự hoa đều là biến). Khi trong câu hỏi của chúng ta chứa một (hoặc nhiều) biến, hệ thống sẽ tìm các giá trị có thể có của biến để cho câu phát biểu của ta là đúng.

Hiểu ở mức ý niệm, câu hỏi của chúng ta là: ai là người? Kết quả trả lời của câu hỏi (ai) sẽ được chứa trong biến X.

Câu trả lời sẽ là: X = socrates



```
GNU Prolog console
File Edit Terminal Prolog Help

yes
| ?- nguoi(xeda).

no
| ?- chet(socrates).

yes
| ?- chet(xeda).

no
| ?- nguoi(X).

X = socrates

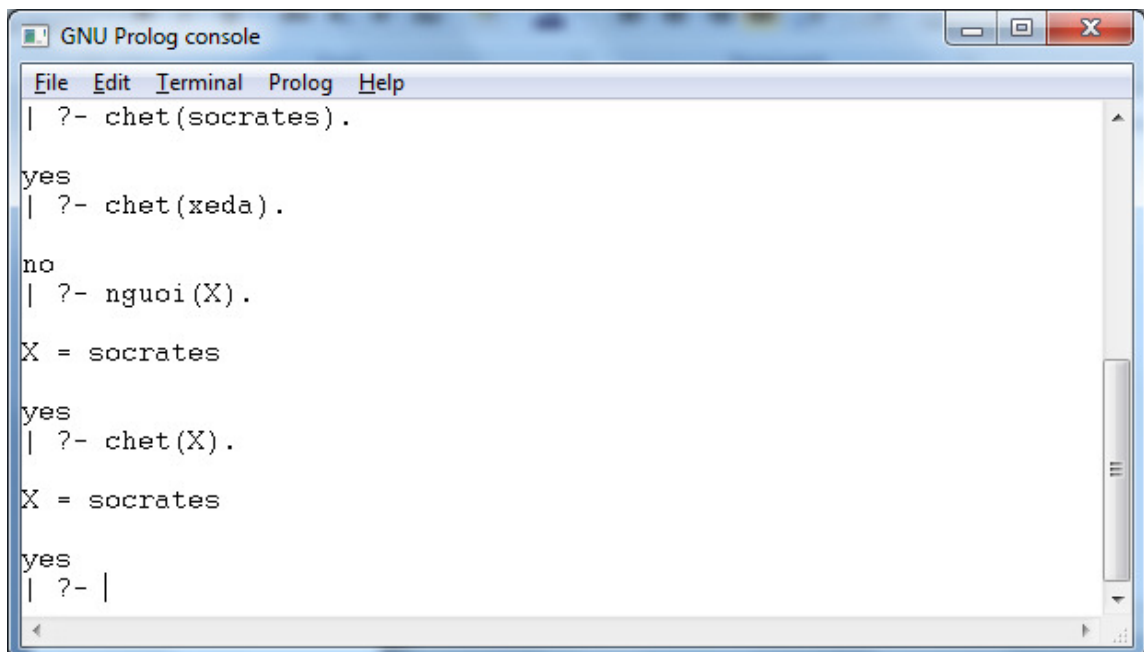
yes
| ?- |
```

Tương tự như trên, hệ thống sẽ dựa vào cơ chế suy luận đã được cung cấp để tìm ra lời giải với những câu hỏi dành cho các vị từ có các mệnh đề tương ứng là các luật.

Nhập vào goal như sau:

chet(X)

Hệ thống sẽ trả lời X = Socrates.



```
GNU Prolog console
File Edit Terminal Prolog Help

| ?- chet(socrates).

yes
| ?- chet(xeda).

no
| ?- nguoi(X).

X = socrates

yes
| ?- chet(X).

X = socrates

yes
| ?- |
```

VD7: Hoàn chỉnh và thực thi chương trình cho VD2

Chương trình hoàn chỉnh như sau:

```
giaithua(0,1) .  
giaithua(X,Y) :- X > 0, X1 is X -1, giaithua(X1,Y1), Y is X*Y1.
```

Chúng ta lưu ý là khi kết thúc mỗi mệnh đề đều có ký hiệu '.'

Chúng ta có thể đặt cho hệ thống câu hỏi dạng nghi vấn như sau:

giaithua(3, 6).

Hiểu theo ngôn ngữ tự nhiên sẽ là: có phải giai thừa của 3 là 6 hay không?

Câu trả lời là: Yes

Hoặc chúng ta có thể đặt câu hỏi:

giaithua(3, 8)

Câu trả lời sẽ là: No.

Chúng ta sẽ đặt câu hỏi theo dạng tìm lời giải (câu hỏi ai ?/cái gì ?):

giaithua(3, X)

Câu trả lời sẽ là $X = 6$

Chúng ta cũng có thể đặt câu hỏi ngược:

giaithua(X, 6)

Ý tưởng của câu hỏi sẽ là: giai thừa của số nào sẽ bằng 6. Tuy nhiên do chúng ta không cung cấp cho hệ thống cơ chế suy luận để trả lời câu hỏi này. Hệ thống sẽ không tìm được câu trả lời.

Tất nhiên chúng ta có thể đặt câu hỏi như sau:

giaithua(X, Y)

Cả hai tham số đều là biến. Như vậy câu hỏi có thể hiểu là: số nào (X) giai thừa thì thành một số khác (Y). Câu hỏi gần như vô nghĩa và những câu trả lời của hệ thống cũng sẽ chẳng mang một ý nghĩa thực sự có nghĩa nào.

Tóm tắt:

- Chương trình Prolog sẽ hoạt động theo cơ chế tương tác. Người sử dụng sẽ cung cấp yêu cầu, gọi là goal, và hệ thống sẽ trả lời các yêu cầu này.
- Nếu goal không chứa biến thì hệ thống sẽ kiểm tra phát biểu của chúng ta là đúng hoặc sai, ngược lại, hệ thống sẽ tìm các giá trị của các biến làm cho phát biểu của ta là đúng.

2.4. Phép hợp nhất (Unification) - Cơ chế tìm câu trả lời của Prolog

2.4.1. Phép hợp nhất

Công việc quan trọng nhất của Prolog trong việc tìm câu trả lời là thực hiện việc hợp nhất. Phép hợp nhất được biểu diễn bằng dấu $=$, và nó có hai thành phần, tạm gọi là vế trái vế phải. Phép hợp nhất sẽ trả về kết quả true hoặc false

Có các trường hợp hợp nhất sau:

- a. Cả hai vế đều là hằng hoặc biểu thức chứa toàn hằng. Nếu giá trị của hai vế là bằng nhau thì phép hợp nhất thành công (đáp số là true), ngược lại phép hợp nhất sẽ thất bại (kết quả là false).

Hợp nhất	Kết quả
$7 = 7$	true
$7 = 8$	false
$'abc' = 'abc'$	true
$'abcd' = 'abc'$	false
$7 = 6 + 1$	false
$7 \text{ is } 6 + 1$	true
$6 \text{ is } (7 + 1)$	false
$1 + 1 = 2$	false
$X \text{ is } 2 + 5, X \text{ is } 3 + 4$	true
$X \text{ is } 1 + 1, X = 2$	true
$1 + 1 \text{ is } 2$	false
$1 + 3 ::= 2 + 2$	true

Chú ý:

- Đối với biểu thức số học, GNU Prolog sẽ không tự động tính giá trị của nó khi hợp nhất, do đó $1 + 1 = 2$ sẽ cho kết quả false.
- Để tính giá trị của các biểu thức số học ta phải sử dụng toán tử is và gán kết quả cho một biến tạm nào đó. Ví dụ: $X \text{ is } (1 + 1), X = 2$ sẽ cho kết quả true. Nếu chỉ có một biểu thức và một hằng ta có thể không cần biến tạm. Ví dụ: $3 \text{ is } (8 - 5)$ sẽ cho kết quả true.
- Mệnh đề *Result is Expression* sẽ trả về **true** nếu *Result* có thể hợp nhất được với *kết quả* của biểu thức *Expression*.
- Để hợp nhất hai biểu thức ta sử dụng phép toán $::=$
- b. Một trong hai vế là hằng hoặc trong biểu thức chứa toàn hằng, vế kia là biến hoặc biểu thức có chứa biến.
 - Trường hợp 1: tất cả các biến đều đã có giá trị (gọi là các biến ở tình trạng bound), chúng ta quay về trường hợp a. Ví dụ:
 - $7 = X$ cho kết quả false nếu X đã có giá trị là 6
 - $7 \text{ is } X + 1$ cho true nếu X đã có giá trị là 6
 - $Y = 'Socrates'$ cho true nếu Y đã có giá trị là 'Socrates'

- Trường hợp 2: Nếu có biến chưa có giá trị (gọi là biến ở tình trạng unbound), Prolog sẽ gán giá trị cho biến sao cho hai vế có giá trị như nhau và trả về kết quả là true. Nếu không tìm giá trị như vậy, phép hợp nhất sẽ cho kết quả là false. Ví dụ:
 - $7 = X$ cho kết quả true nếu X chưa có giá trị. Sau phép hợp nhất này, X sẽ có giá trị là 7.
- c. Cả hai vế đều là biến hoặc các biểu thức có chứa biến
 - Trường hợp 1: tất cả các biến đều có chứa giá trị, chúng ta sẽ quay về trường hợp a. Ví dụ:
 - $X = Y$ cho kết quả true nếu cả X và Y đều đã có giá trị và những giá trị này bằng nhau.
 - $X \text{ is } Y - 1$ cho kết quả true nếu X và Y đều đã có giá trị và X nhỏ hơn Y 1 đơn vị.
 - Trường hợp 2: tất cả các biến của một vế đều đã có giá trị, chúng ta sẽ quay về trường hợp b.
 - $X = Y$ cho kết quả true nếu X chưa có giá trị và Y đã có giá trị, sau phép hợp nhất, X sẽ nhận giá trị của Y .
 - $X \text{ is } Y - 1$ cho kết quả true nếu X chưa có giá trị, Y đã có giá trị. Sau phép hợp nhất, X sẽ có giá trị bằng giá trị của Y trừ đi 1.
 - $Y = 5, X = Y - 1$ cho kết quả true và $X = 5 - 1$ (chú ý GNU Prolog không định trị $5-1$ để được 4 mà gán cả biểu thức $5-1$ cho X).
 - $Y = 5, X = Y - 1, 4 \text{ is } X$ sẽ cho kết quả true.
 - Trường hợp 3: cả hai vế đều còn chứa biến ở tình trạng unbound. Ví dụ:
 - $X = Y$ cho kết quả true và $X = Y$ (X tham chiếu đến Y).
 - $X = Y - 1$ cho kết quả true và $X = Y - 1$. Chú ý: GNU Prolog không định trị tự động biểu thức mà gán X bằng cả biểu thức $Y - 1$.

2.4.2. Cơ chế tìm câu trả lời của Prolog

Nếu chúng ta đặt ra cho Prolog một câu hỏi, Prolog sẽ thực hiện công việc so trùng (match), tức là tìm mệnh đề đầu tiên đề cập đến khái niệm mà chúng ta muốn hỏi.

Trở lại VD6, sau khi đã hoàn tất chương trình, chúng ta đặt ra goal như sau:

nguo(*'Socrates'*).

Prolog sẽ tìm mệnh đề đầu tiên có liên quan đến khái niệm *nguo*. Hiển nhiên, mệnh đề đầu tiên (và duy nhất có liên quan đến khái niệm này là:

nguo(*'Socrates'*).

Như vậy, khi đã có goal (*nguo('Socrates')*) và tìm thấy mệnh đề liên quan (*nguo('Socrates')*), Prolog sẽ tiến hành tìm kiếm lời giải, công việc này tiến hành bằng cách tạo mối liên kết giữa các tham số ở phần goal và các tham số ở phần mệnh đề. Có các trường hợp sau:

- a. Cả hai tham số này đều là các biến unbound, trong trường hợp này Prolog sẽ xem cả hai tham số là một.
- b. Ở tất cả các trường hợp khác, Prolog sẽ tiến hành phép hợp nhất giữa hai loại tham số.

Sau khi đã tạo mối quan hệ giữa các tham số ở phần goal và phần clause, Prolog sẽ tiến hành các mệnh đề con (nếu mệnh đề này một luật). Nếu tất cả các sub-clause thành công và các biến ở phần goal đã ở tình trạng bound (tức là đã có giá trị), Prolog sẽ thông báo lời giải.

Nếu là câu hỏi thuộc dạng Yes/No như ví dụ trên, tức là câu hỏi không chứa biến, Prolog sẽ trả lời Yes nếu công việc hợp nhất như đã nói ở phần b thành công và các sub-clause đều thành công (nếu mệnh đề so trùng là một luật).

Quay trở lại với ví dụ của chúng ta, ở đây tham số của Goal là một hằng ("Socrates"), và tham số của mệnh đề tương ứng cũng là một hằng ("Socrates"), hai hằng này hợp nhất thành công, và kết quả trả lời là Yes.

Nếu chúng ta đặt ra câu hỏi khác:

nguo(i('Xeda')).

Prolog cũng chỉ tìm thấy một mệnh đề liên quan đến khai niệm này (nguo(i("Socrates"))), và vì sự hợp nhất giữa hai hằng "Socrates" và "Xeda" thất bại, đáp số sẽ trả lời là No.

Chúng ta xét trường hợp câu hỏi của chúng ta có chứa biến:

nguo(i(X)).

Hệ thống sẽ tìm thấy mệnh đề có liên quan đến vấn đề này (nguo(i("Socrates"))), và tiến hành hợp nhất giữa X và "Socrates", và vì X chưa có giá trị (unbound) nên phép hợp nhất thành công, X có giá trị là "Socrates".

Vì việc hợp nhất giữa các tham số giữa phần goal và phần clause đã thành công, đây là một sự kiện nên không cần phải thực hiện phần sub-clause, và sau khi hợp nhất, tất cả các biến cần tìm đã có giá trị (ở đây chỉ có một biến là X), nên hệ thống sẽ công bố đã tìm ra lời giải và in ra giá trị của X (X = "Socrates").

Chúng ta xét trường hợp khi ở câu hỏi phần goal so trùng với một luật:

chet(Y).

Chúng ta hoàn toàn có thể đặt câu hỏi là chet(X), nhưng chúng ta sẽ đặt tên biến khác để tiện phân biệt giữa biến trong câu hỏi của goal và tham số cục bộ ở mệnh đề. Câu hỏi trong goal được so trùng với mệnh đề sau:

chet(X): - ngo(i(X)).

Vì hai biến X(tham số của mệnh đề) và Y(tham số của goal) đều chưa chứa giá trị, hệ thống sẽ xem cả hai biến là một, tức là, khi X có được giá trị thì Y cũng có giá trị đó và ngược lại.

Do đây là một luật, nên hệ thống sẽ tiến hành thực hiện các sub-clause. Hệ thống sẽ thực hiện sub-clause đầu tiên ngo(i(X)).

Quá trình thực hiện các sub-clause ở vế phải sẽ được thực hiện như sau:

- a. Nếu sub-clause này có tham số là biến unbound, Prolog sẽ tìm giá trị của biến này để sub-clause có giá trị Yes, nếu không tìm được giá trị như vậy, sub-clause sẽ thất bại.
- b. Nếu sub-clause có tham số đều là biến bound (đã có giá trị) hoặc là hằng, Prolog sẽ kiểm tra xem sub-clause có trả về giá trị Yes hay không, nếu không, sub-clause sẽ thất bại.

Các sub-clause sẽ được tiến hành từ trái qua phải, và nếu có một sub-clause thất bại, mệnh đề được so trùng sẽ thất bại.

Trong trường hợp trên, khi tiến hành sub-clause `nguoi(X)`, do biến `X` là unbound, nên chúng ta rơi vào trường hợp a, hệ thống sẽ tìm giá trị của `X` cho sub-clause trên là đúng. Cách tìm kiếm câu trả lời cho sub-clause này hoàn toàn giống như cách hệ thống tìm câu trả lời khi chúng ta đặt câu hỏi này trong phần goal, và như vậy `X` sẽ có giá trị là "Socrates" sau khi sub-clause này thực hiện xong.

Do `X` và `Y` được xem như một, nên khi `X` có giá trị là "Socrates" thì `Y` cũng có giá trị này.

Do tất cả các sub-clause đã thực hiện xong, và `Y` đã có giá trị, nên Prolog công bố là đã tìm ra lời giải và in ra giá trị của `Y`.

Tóm tắt:

- Phép hợp nhất là nền tảng của mọi hoạt động của Prolog để tìm ra lời giải
- Để trả lời câu hỏi, Prolog so trùng câu hỏi với mệnh đề và tạo mối liên quan giữa các tham số.
- Prolog tìm ra lời giải khi thực hiện thành công một mệnh đề và tất cả các biến nếu có trong các tham số của goal đều đã có giá trị.

2.5. Quay lui - Không chế số lượng lời giải - Vị từ cắt và fail

VD8: Xem lại VD6

```
nguoi('Socrates').  
chet(X) :- nguoi(X).
```

Nạp chương trình và truy vấn:

`nguoi(X), write('Ket qua: '), write(X).`

Trong câu truy vấn này ta gọi vị từ `write` để in giá trị `X`. Vị từ `write` sẽ cho kết quả đúng nếu các tham số nhập vào là đều là biến ở trạng thái bound hoặc là hằng.

2.5.1. Quay lui (back-tracing) trên Prolog

Hợp nhất (unification) là nền tảng cho cơ chế suy luận của Prolog, tuy nhiên, để tìm ra lời giải đúng, Prolog phải sử dụng cơ chế quay lui, khi giá trị đầu tiên được gán cho tham số không phải là lời giải.

Chúng ta xét ví dụ sau:

VD9:

```
nguoi('Socrates').  
nguoi('Xeda').  
vua('Xeda').  
sungsuong(X) :- nguoi(X), vua(X).
```

Trong ví dụ này, ngoài khái niệm về *người*, chúng ta đưa thêm khái niệm về *vua* và *sung sướng*. Diễn giải những thông tin trong các dữ kiện trên thành ngôn ngữ tự nhiên, chúng ta có được các điều sau: “Thế giới mà chúng ta sống có hai người là Socrates và Xeda. Chúng ta có một vua là Xeda, và một thực thể nào đó chỉ sung sướng nếu thực thể đó vừa người vừa là vua.”

Lưu ý rằng trong ví dụ trên, các mệnh đề liên quan đến cùng một vị từ phải viết liên tiếp nhau.

Xét khi hệ thống trả lời câu hỏi sau:

sungsuong(X).

Trước tiên hệ thống sẽ so trùng goal trên với mệnh đề

sungsuong(X) :- nguoi(X), vua(X).

Lưu ý rằng vào lúc này chúng ta có hai biến X: một biến X là tham số của goal và một biến X là tham số của mệnh đề. Về nguyên tắc, hai biến X này hoàn toàn khác nhau nhưng theo phép hợp nhất biến – biến, do lúc này cả hai đều chưa có giá trị nên 2 biến này sẽ cùng tham chiếu đến một giá trị như nhau.

Sau đó Prolog sẽ tiến hành các sub-clause. Ở sub-clause đầu tiên, nguoi(X), tương tự như VD6, Prolog sẽ tìm được câu trả lời là X = Socrates.

Khi thực hiện sub-clause thứ hai, vua(X), do X đã có giá trị (Socrates), Prolog sẽ kiểm tra xem giá trị này có làm giá trị của mệnh đề là true hay không.

Như các ví dụ trên, việc tiến hành trả lời một sub-clause cũng tương tự như khi trả lời một goal, Prolog lại so trùng sub-clause với một mệnh đề cùng tên. Prolog tìm thấy một mệnh đề liên quan đến vua là vua('Xeda') và tiến hành hợp nhất giữa X và Xeda. Do X đã có giá trị là Socrates, việc hợp nhất thất bại.

Tuy nhiên khi sub-clause này thất bại, không có nghĩa rằng Prolog sẽ vội kết luận rằng mệnh đề này thất bại. Ở đây công việc tìm kiếm câu trả lời thất bại sau khi biến X được gán giá trị và chuyển từ trạng thái bound sang unbound. Hệ thống sẽ quay lại thời điểm biến X được gán giá trị (khi trả lời sub-clause nguoi(X)), X được chuyển lại sang tình trạng unbound, và cố gắng tìm kiếm một giá trị khác của X để cho mệnh đề con này vẫn đúng. Công việc này được gọi là quay lui (back-tracing).

Do việc so trùng sub-clause này với mệnh đề nguoi('Socrates') thất bại, hệ thống sẽ so trùng với mệnh đề khác. Nếu không còn mệnh đề nào khác liên quan đến sub-clause, việc thực hiện mệnh đề mới thật sự thất bại, tuy nhiên ở đây hệ thống tìm thấy một mệnh đề khác liên quan đến khái niệm này là nguoi('Xeda'). Việc hợp nhất giữa X và 'Xeda' lại được thực hiện, X sẽ có

giá trị là Xeda và sau đó, khi lại tiếp tục thực hiện sub-clause vua(X) thì chúng ta sẽ dễ dàng thấy rằng sub-clause lần này được thực hiện thành công. Prolog đã tìm ra lời giải, tuy nhiên, ở trường hợp này, ngoài sự hợp nhất, Prolog còn sử dụng thêm một “vũ khí” mới, đó là phép quay lui.

2.5.2. Không chế số lượng lời giải

VD10. Chúng ta xét ví dụ sau:

```
nguai ( 'Socrates' ) .  
nguai ( 'Xeda' ) .  
chet (X) :- nguoi (X) .
```

Ví dụ không có gì phức tạp, so với VD6, chúng ta chỉ thêm một người mới là Xeda.

Khi truy vấn với câu hỏi:

nguai(X).

Chúng ta có hai lời giải:

X = Socrates

X = Xeda

Chúng ta cảm thấy hai đáp số này là hiển nhiên. Với GNU Prolog, sau khi in ra câu trả lời đầu tiên, Prolog sẽ dừng lại chờ chúng ta. Nếu chúng ta muốn xem tiếp câu trả lời kế tiếp, ta gõ dấu chấm phẩy (;). Nếu muốn dừng lại ta gõ phím ENTER.

Để không chế số lượng lời giải theo ý mình trong chương trình, chúng ta sử dụng hai vị từ đặc biệt là *cut* và *fail*, như các ví dụ sau:

VD11: Nạp lại chương trình trong VD10, sử dụng vị từ *fail* để in ra tất cả các lời giải trong trường hợp muốn hỏi ai là người ? Ta truy vấn:

```
nguai(X), nl, write(X), fail.
```

- *nl* là vị từ đặc biệt, luôn trả về kết quả là đúng, và chỉ đơn giản là xuống dòng trước khi in thông tin mới ra cửa sổ giao tiếp.
- Vị từ *fail* là một vị từ đặc biệt, luôn luôn trả về kết quả là sai.

Như vậy để in ra tất cả các kết quả, chúng ta dùng một thủ thuật (trick) thường gặp khi lập trình trên Prolog: sau khi đã tìm thấy lời giải cho sub-goal nguoi(X) và in giá trị này ra bằng lời gọi vị từ write(X), chúng ta gọi vị từ *fail* để nhận được kết quả là sai. Do cơ chế back-tracing đã nói ở trên, Prolog lại quay lại thời điểm gọi sub-goal nguoi(X) để tìm lời giải khác và in ra. Quá trình này cứ tiếp tục cho đến khi không thể tìm thấy thêm một lời giải nào khác. Bằng cách này, chúng ta đã in ra tất cả các lời giải cho câu hỏi nguoi(X), tuy nhiên lưu ý rằng với goal chính thì không tìm ra lời giải (do chúng ta luôn gọi vị từ fail cho sub-goal cuối cùng).

VD12: Viết lại chương trình trong ví dụ 10, sử dụng vị từ cut để in ra một lời giải đầu tiên.

```

nguo(i('Socrates')).
nguo(i('Xeda')).
chet(X) :- nguo(i(X), !.
chet('Milou').

```

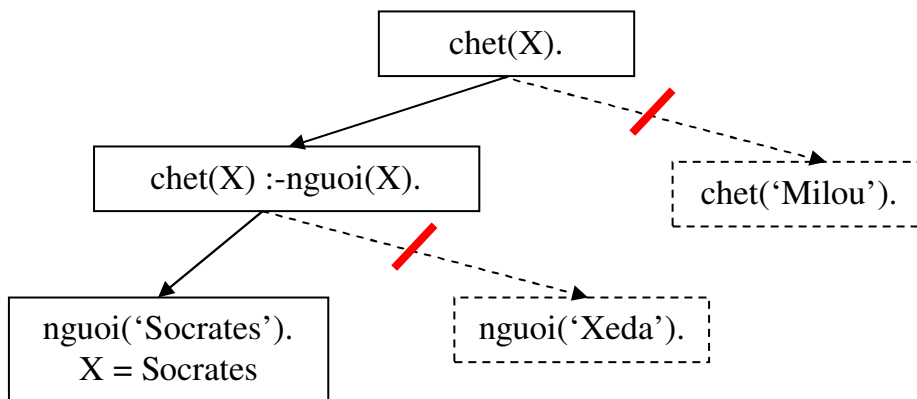
Vị từ *cắt* được viết được là ! Đây là một vị từ đặc biệt, sẽ trả lời đúng khi các sub-clause đứng trước nó đúng, và nó sẽ cắt đường quay lui của tất cả các mệnh đề con đứng trước nó, và kể cả mệnh đề chứa nó.

Nạp chương trình và truy vấn:

chet(X).

Khi sử dụng câu truy vấn **chet(X)**, khi trả lời sub-clause **nguo(i(X)**, hệ thống tìm ra một đáp số là **X = Socrates**, vị từ ! tiếp theo sẽ trả lời là thành công.

Sau đó, Prolog sẽ quay lui tìm tiếp các lời giải khác. Quay trở lại tìm xem có thể tìm được một người nào khác nữa hay không. Nhưng do ! đã cắt hết các nhánh quay lui của **nguo(i(X)** nên không thể tìm thêm được giá trị nào khác cho X nữa. Prolog tiếp tục quay lên trên tìm câu trả lời với các mệnh đề **chet** khác. Nhưng vì ! cũng đã cắt hết đường quay lui, Prolog không còn đường để quay lui nữa. Vì thế quá trình suy diễn kết thúc. Có thể minh họa điều này bằng cây suy diễn bên dưới:



Hãy sửa lại chương trình trong ví dụ 12 một chút ở vị trí cắt:

```

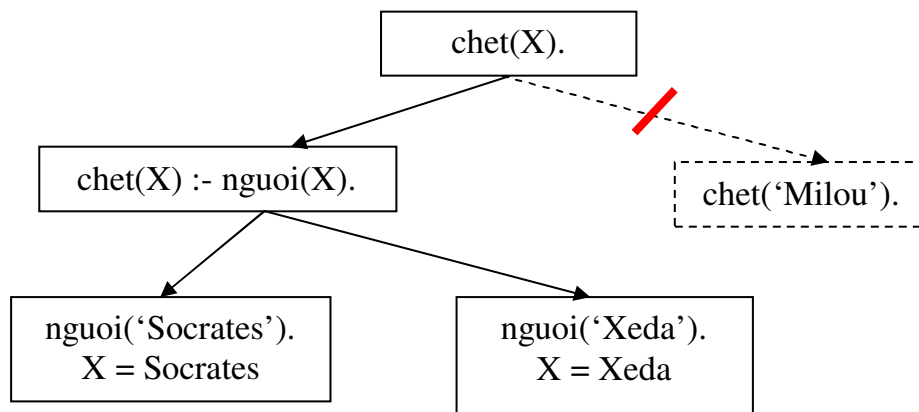
nguo(i('Socrates')).
nguo(i('Xeda')).
chet(X) :- !, nguo(i(X).
chet('Milou').

```

Nạp chương trình và truy vấn: **chet(X)**. Chương trình sẽ cho 2 lời giải:

X = Socrates và **X = Xeda**.

Vì ! chỉ cắt đường quay lui của **chet** mà không cắt đường quay lui của người nên Prolog có thể tìm tất cả các lời giải đối với mệnh đề con **nguoi(X)**. Cây suy diễn được cho như hình bên dưới:



Tóm tắt

- Prolog sẽ sử dụng cơ chế quay lui khi một biến khi chuyển từ trạng thái unbound sang bound sẽ dẫn đến sự thất bại trong việc truy tìm lời giải.
- Vị từ fail luôn trả lời là sai, có thể được sử dụng để “ép” Prolog quay lui.
- Vị từ ! sẽ trả lời và cắt đường quay lui của các mệnh đề con đứng trước nó, kể cả mệnh đề gọi nó.

2.6. Lập trình đệ quy với Prolog

Chúng ta nhớ lại rằng với VD2, chúng ta đã cố gắng né tránh cách đặt vấn đề để giải bài toán giai thừa theo cách nhân dồn các số từ 1 đến số cần tính giá trị giai thừa. Điều này sẽ dẫn đến một điểm yếu của Prolog: không cung cấp các cấu trúc điều khiển cần thiết, dẫn đến việc khó khăn khi thực hiện phép lặp. Tuy nhiên ví dụ này cũng cho thấy một kỹ thuật lập trình tạo nên sức mạnh chủ yếu của Prolog: lập trình đệ quy. Kỹ thuật này cũng phù hợp với suy nghĩ của con người khi tiếp cận giải quyết vấn đề và khiến cho việc lập trình trên Prolog có một sự uyển chuyển và nhẹ nhàng trong việc viết mã. Tuy vậy, nó tạo ra một sự khó khăn với những người quen lập trình thủ tục.

Chúng ta sẽ xem xét lại từng bước trong việc gọi đệ quy để tìm ra lời giải.

VD13: Xét từng bước quá trình gọi đệ quy và hợp nhất của VD7 với goal là `giaithua(2,X)`

Xét chương trình như sau:

```

giaithua(0,1) :- !.
giaithua(X,Y) :- X1 is X - 1, giaithua(X1,Y1), Y is X*Y1.
  
```

Chương trình này có chút thay đổi so với chương trình trước đây: ta sử dụng vị từ cắt để điều khiển việc quay lui. Chúng ta muốn khẳng định rằng: nếu số cần tìm giai thừa là 0 thì

giai thừa của nó là 1, và kết quả này là duy nhất, không cần phải tiếp tục tìm các trường hợp khác. Nghĩa là không cần quay lui để xét mệnh đề **giaithua(X, Y)** thứ 2.

Với goal là **giaithua(2, X)**, hệ thống sẽ so trùng với mệnh đề **giaithua(0,1)** là mệnh đề đầu tiên tìm thấy có liên quan đến khái niệm **giaithua**.

Hệ thống sẽ hợp nhất các tham số theo thứ tự, 2 hợp nhất với 0 và X hợp nhất với 1. Công việc hợp nhất X với 1 thành công, X có giá trị là 1, nhưng 2 hợp nhất với 0 thất bại.

Hệ thống sẽ tiếp tục quay lui tìm kiếm lời giải khác bằng cách so trùng với mệnh đề khác. Lần này hệ thống so trùng goal với mệnh đề **giaithua(X,Y)**. Khi tạo mối liên quan giữa các tham số, hệ thống hợp nhất 2 với X của mệnh đề và Y với X của goal. Chúng ta sẽ ký hiệu X^G là X tham số của goal. Do Y và X^G đều chưa có giá trị, Prolog sẽ hợp nhất hai biến này là một.

Sau đó hệ thống bắt đầu thực hiện từng sub-clause:

- **X1 is X - 2**

X1 là biến mới, và chưa có giá trị. X đã có giá trị là 2, nên $X - 1$ có giá trị là 1. Hợp nhất X1 với 1 ta sẽ có giá trị của X1 là 1. Tiếp tục với sub-clause thứ hai:

- **giaithua(X1, Y1)**

Ở đây mệnh đề giai thừa được gọi đệ quy. Lưu ý lúc này X1 đã có giá trị là 1, Y1 là biến mới chưa có giá trị, vì vậy nhiệm vụ của hệ thống là tìm giá trị của Y1 sao cho sub-clause **giaithua(X1,Y1)** cho giá trị là đúng. Và cũng như các ví dụ trên, cách thức Prolog trả lời một sub-clause cũng tương tự như khi trả lời câu hỏi từ goal, tức là lại so trùng câu hỏi với các mệnh đề đã biết :

- So trùng với **giaithua(0,1)**, Prolog tiến hành hợp nhất X1 với 0, Y1 với 1, do X1 đã có giá trị là 1, việc hợp nhất với 0 thất bại, Prolog phải so trùng với mệnh đề khác.
- So trùng với **giaithua(X,Y)**, Prolog tiến hành hợp nhất X1 với X và đồng nhất Y1 với Y. Chúng ta ký hiệu X và Y ở lần gọi đệ quy này là $X^{(2)}$ và $Y^{(2)}$ và sử dụng cách ký hiệu tương tự như vậy cho các biến còn lại ở lần gọi đệ quy này cũng như các lần gọi đệ quy tiếp theo. Như vậy $X^{(2)}$ sẽ có giá trị là 1 và Y1 sẽ có giá trị mà $Y^{(2)}$ sẽ có.

Tương tự ở lần gọi thứ nhất, các sub-clause của mệnh đề trên ở lần gọi thứ hai này sẽ lần lượt được gọi:

- $X1^{(2)}$ is $X^{(2)} - 1$, hợp nhất $X1^{(2)}$ với $X^{(2)} - 1$, ta có $X1^{(2)}$ có giá trị là 0.
- **giaithua($X1^{(2)}$, $Y1^{(2)}$)**, $X12$ đã có giá trị là 0, Prolog sẽ tìm giá trị của $Y1^{(2)}$ bằng việc tiếp tục so trùng **giaithua($X1^{(2)}$, $Y1^{(2)}$)** với các mệnh đề có liên quan: So trùng **giaithua($X1^{(2)}$, $Y1^{(2)}$)** với **giaithua(0,1)**. Do $X1^{(2)}$ đã có giá trị là 0, Prolog tiến hành hợp nhất $X1^{(2)}$ với 0 và $Y1^{(2)}$ với 1. Thực hiện tiếp sub-clause ! trả về kết quả đúng. Việc thực hiện mệnh đề **giaithua(0, 1)** thành công, và $Y1^{(2)}$ đã có giá trị là 1 nên câu hỏi **giaithua($X1^{(2)}$, $Y1^{(2)}$)** đã có đáp số. Vì từ ! sẽ ngăn chặn

việc tìm các đáp số khác, vì vậy trong trường hợp này, Prolog không tiếp tục so trùng tiếp với mệnh đề $\text{giaithua}(X,Y)$.

- $Y^{(2)}$ is $X^{(2)} * Y1^{(2)}$, lúc này $Y^{(2)}$ chưa có giá trị, $X^{(2)}$ và $Y1^{(2)}$ đã có giá trị là 1 và 1 nên Prolog sẽ hợp nhất $Y^{(2)}$ và 1. Kết quả sẽ là $Y^{(2)}$ có giá trị là 1.

Như vậy đến đây các sub-clause của mệnh đề $\text{giaithua}(X^{(2)},Y^{(2)})$ đã thực thi thành công, và $Y^{(2)}$ đã có giá trị là 1, và vì $Y1$ được đồng nhất với $Y^{(2)}$, nên $Y1$ cũng sẽ có giá trị là 1.

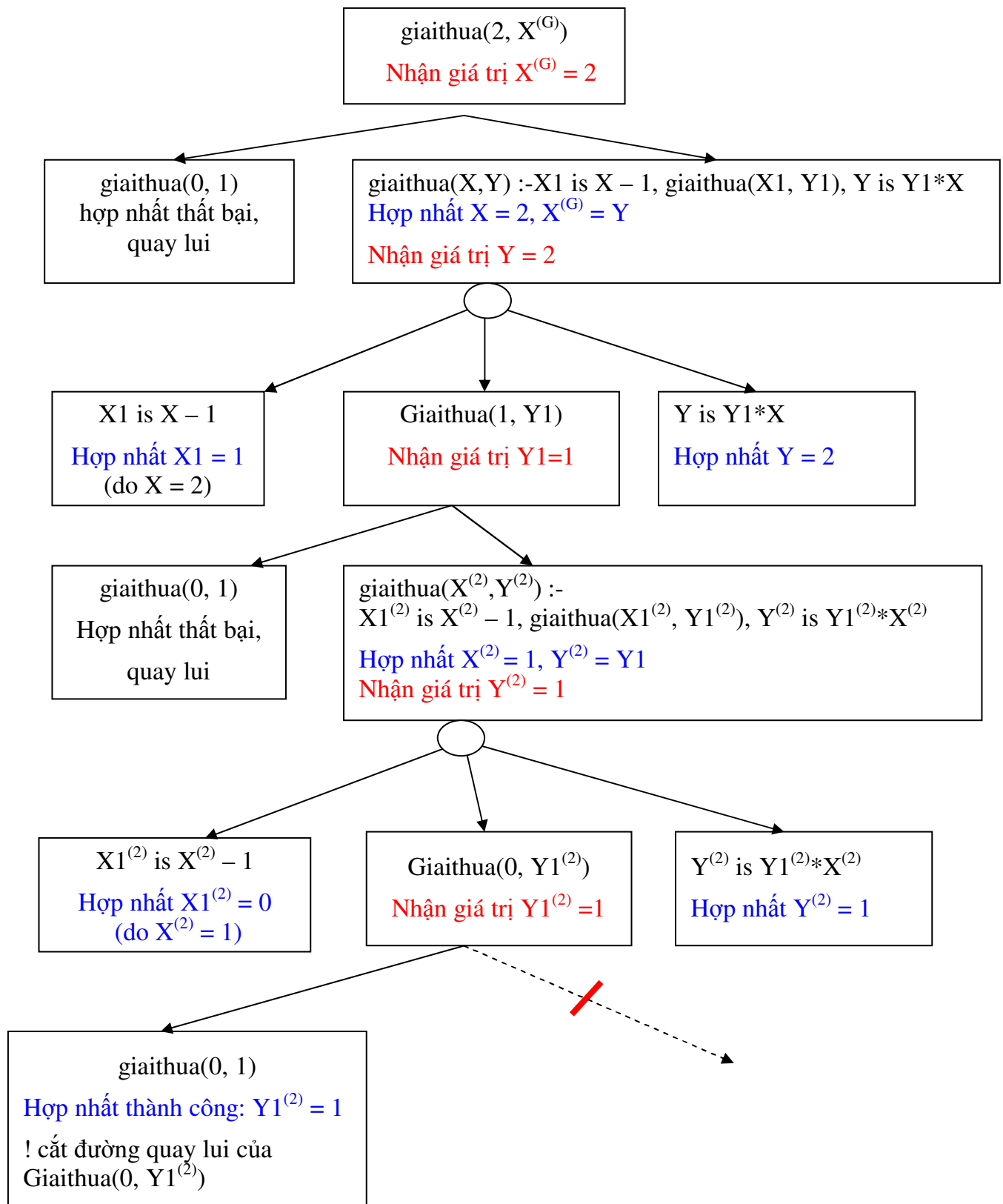
- Y is $X*Y1$, lúc này Y chưa có giá trị, X và $Y1$ đã lần lượt có giá trị là 2 và 1, nên Prolog hợp nhất Y và $2*1$, kết quả Y sẽ có giá trị là 2.

Như vậy đến đây các sub-clause của mệnh đề $\text{giaithua}(X,Y)$ đã thực thi thành công, và Y đã có giá trị là 2, và vì X^G được đồng nhất với Y , nên X^G cũng sẽ có giá trị là 2, và lời giải của bài toán đã được tìm thấy. Do không còn đường nào khác để quay lui nên Prolog dừng lại.

Cây suy diễn được cho trong hình bên dưới.

Tóm tắt:

- Đệ quy là sức mạnh lập trình chủ yếu trên Prolog.
- Mỗi lần gọi đệ quy, các tham số và biến cục bộ trong mỗi mệnh đề sẽ được tạo mới tương ứng với lần gọi đệ quy đó.
- Có thể dùng vị từ cắt để ngăn chặn các lần gọi đệ quy thừa khi đã tìm ra đáp số.



2.7. Danh sách trên Prolog

Danh sách là kiểu dữ liệu cấu trúc đặc biệt trên Prolog. Có thể hiểu danh sách như một kiểu dãy một chiều, và phần tử của danh sách có thể thuộc về kiểu dữ liệu bất kỳ.

2.7.1. Cấu trúc của danh sách

Một danh sách trên Prolog bao gồm hai phần: phần đầu (head) là phần tử đầu tiên của danh sách và phần đuôi (tail) là danh sách các phần tử còn lại của danh sách.

Một danh sách có thể mô tả theo hai cách:

- Liệt kê các phần tử của danh sách, ví dụ: [1,2,3,4,5]
- Mô tả phần đầu và phần đuôi của danh sách, ngăn cách bởi dấu |, ví dụ: [1|[2,3,4,5]]

VD15: Mô tả một danh sách bao gồm 5 số nguyên là 1,2,3,4,5

Danh sách trên có thể mô tả theo các cách sau:

[1,2,3,4,5]

[1|[2,3,4,5]]

[1|[2|[3,4,5]]]

[1|[2|[3|[4,5]]]]]

[1|[2|[3|[4|[5]]]]]]]

[1|[2|[3|[4|[5|[]]]]]]]]

Lưu ý: danh sách rỗng có thể được mô tả như sau: []

VD16: Viết chương trình in ra phần đầu và phần đuôi của một danh sách.

Chương trình này thực chất chỉ giúp chúng ta nhìn rõ hơn khái niệm về danh sách.

Chương trình được viết như sau:

indanhsach(L,H,T) :- L = [H|T] .

Xét khi chúng ta nhập goal vào như sau:

indanhsach([1,2,3,4,5],X,Y).

Prolog sẽ so trùng goal với mệnh đề indanhsach(L,H,T), L được hợp nhất với [1,2,3,4,5], X được đồng nhất với H, Y được đồng nhất với T.

Khi thực hiện sub-clause $L = [H|T]$, L được hợp nhất với [H|T], như vậy phần đầu của L sẽ hợp nhất với H, phần đuôi sẽ hợp nhất với T. Do L đã có giá trị là [1,2,3,4,5], phần đầu của L sẽ có giá trị là 1, phần đuôi sẽ có giá trị là [2,3,4,5], vậy sau khi hợp nhất, H sẽ có giá trị là 1 và L sẽ có giá trị là [2,3,4,5]. Cũng tức là X sẽ có giá trị là 1 và Y có giá trị là [2,3,4,5]. Prolog đã tìm thấy lời giải và sẽ in ra lời giải này.

Lưu ý:

- Chương trình trên sẽ chạy sai nếu danh sách nhập vào là rỗng ([]) do lời giải của chúng ta chưa xử lý trường hợp này
- Phần mệnh đề cho vị từ `indanhsach` có thể viết lại là `indanhsach([H|T],H,T)`.

Tóm tắt

- Danh sách là kiểu dữ liệu cấu trúc đặc biệt do người dùng định nghĩa trên Prolog
- Một danh sách bao gồm hai phần: phần đầu là phần tử đầu, phần đuôi là danh sách
- các phần tử còn lại của danh sách.
- Trong trường hợp danh sách rỗng, phần đầu của danh sách sẽ không có.

2.8. Lập trình đệ quy với danh sách trên Prolog

Khi xử lý danh sách trên Prolog, người lập trình phải từ bỏ phong cách dùng vòng lặp để xử lý dãy mà phải tận dụng kỹ thuật đệ quy để tìm ra lời giải.

Chúng ta xét một số ví dụ sau đây:

VD17: Viết vị từ đếm xem một danh sách có bao nhiêu phần tử.

Đầu tiên chúng ta phải định nghĩa được công việc mà chúng ta định làm. Chúng ta sẽ viết một vị từ như sau:

dem(L, N)

Chúng ta đếm trong một danh sách có bao nhiêu phần tử. Ví dụ khi gọi goal `dem([1,2,3,4], X)`, đáp số sẽ là $X = 4$.

Tiếp theo chúng ta phải xác định một thuật giải phù hợp với tinh thần của Prolog. Không thể dùng vòng lặp, chúng ta phải xây dựng một giải thuật đệ quy. Một giải thuật đệ quy sẽ bao gồm hai phần: điều kiện dừng và lời gọi đệ quy.

Điều kiện dừng cho bài toán này rất dễ dàng: khi danh sách không có phần tử nào, thì hiển nhiên số phần tử của nó là 0.

Vậy điều kiện dừng sẽ được viết như sau:

dem([],0) :- !.

Khi trường hợp danh sách không có phần tử nào, đáp số 0 là duy nhất, vậy chúng ta có thể dùng nhất cắt để yêu cầu Prolog không tìm thêm lời giải nào khác.

Phần đệ quy hơi khó đối với những ai chưa quen với danh sách trên Prolog. Prolog chỉ cung cấp cho chúng ta cơ chế chia danh sách thành hai phần: phần tử đầu và phần đuôi danh sách các phần tử còn lại. Như vậy lời giải gần như đã tự nó hiện ra: chúng ta sẽ gọi đệ quy để đếm phần đuôi có bao nhiêu phần tử rồi cộng nó với 1 (phần tử đầu) sẽ ra số phần tử trong một danh sách. Phần này sẽ được viết như sau:

dem([H|T], X):- dem(T,X1), X is X1+1.

Tư tưởng đệ quy đã được thể hiện rất rõ ràng: đếm phần đuôi T của danh sách để có được

tổng $X1$, hợp nhất X với $X1+1$ sẽ cho đáp số cần tìm.

Tuy nhiên chúng ta thấy ở đây biến H hoàn toàn không cần dùng trong vế phải của mệnh đề. Prolog không coi đây là lỗi, nhưng sẽ "phàn nàn" về việc này. Xét về hiệu quả lập trình, không cần khai báo tên biến mới cho thành phần H trong trường hợp này. Có một nguyên tắc để nhận ra những biến "vô dụng" như vậy: đó là những biến chỉ xuất hiện 1 lần trong suốt mệnh đề. Đối với trường hợp này, ta nên dùng ký hiệu '_' thay cho tên biến để biểu diễn biến này không cần dùng trong thuật giải.

Vậy phần đệ quy sẽ được "tinh chế" như sau:

dem([_|T],X):- dem(T,X1), X is X1+1.

Như vậy toàn bộ chương trình hoàn chỉnh sẽ như sau:

```
dem([], 0) :- !.  
dem([_|T], X) :- dem(T,X1), X is X1 + 1.
```

VD18: Viết vị từ tính tổng các phần tử trong một danh sách

```
tong([], 0) :- !.  
tong([H|T], X) :- tong(T,X1), X is X1 + H.
```

Tư duy đệ quy ở đây là: chúng ta tính tổng phần đuôi của danh sách, rồi cộng nó với phần tử đầu để tính tổng danh sách.

VD19: Viết vị từ kiểm tra một số nguyên có nằm trong danh sách hay không.

Xác định vấn đề: chúng ta sẽ viết vị từ $ptu(X, L)$, khi gọi $ptu(2,[1,3,5])$ kết quả sẽ là No, ngược lại khi gọi $ptu(3,[1,3,5])$, kết quả sẽ là Yes.

Ở đây chúng ta phải xác định được các trường hợp một phần tử nằm trong một danh sách. Và chúng ta phải trình bày được các khái niệm này một cách đệ quy.

Sau đây là ý tưởng của thuật giải: có hai trường hợp để một số nguyên là phần tử của một danh sách: số nguyên này là phần tử đầu của danh sách hoặc nó là phần tử của phần đuôi danh sách.

Sau khi xác định được ý tưởng, chúng ta có bài giải như sau:

```
ptu(H, [H|_]) :- !.  
ptu(H, [_|T]) :- ptu(H, T).
```

Lưu ý:

- Chúng ta đã thay thế các biến chỉ xuất hiện một lần trong một mệnh đề bằng ký hiệu '_' như đã nói.
- Ở mệnh đề đầu đã có ký hiệu nhất cắt, nên nếu mệnh đề này đúng, bài toán đã có lời giải và không so trùng đến mệnh đề thứ hai. Như vậy mệnh đề thứ hai chỉ

được so trùng khi mệnh đề thứ nhất thất bại, và vì mệnh đề thứ nhất ứng với trường hợp số nguyên cần tìm bằng với phần tử đầu của danh sách, nên khi mệnh đề thứ nhất thất bại, tức là số nguyên cần tìm không bằng với phần tử đầu của danh sách, nên trong mệnh đề thứ hai, chúng ta không cần quan tâm đến phần tử đầu của danh sách.

VD20: Xác định phần tử thứ n của danh sách

Khi ta gọi $\text{ptn}([1,3,5,7,9],2,X)$. thì $X = 3$ (phần tử thứ 2 của danh sách).

Vì Prolog chỉ cho chúng ta truy xuất phần tử đầu tiên của danh sách, nên chúng ta phải xây dựng thuật giải đệ quy dựa trên cơ sở này: nếu n bằng 1 thì phần tử cần tìm là phần tử đầu của danh sách, ngược lại thì đó sẽ là phần tử thứ n - 1 của phần đuôi danh sách.

```
ptn([H|_], 1, H) :- !.  
ptn([_|T], N, X) :- N1 = N - 1, ptn(T, N1, X).
```

VD 21: Tạo ra một danh sách bao gồm chỉ các phần tử lẻ của danh sách ban đầu.

Goal khi gọi sẽ có dạng $\text{ptle}([1,2,3,4,5,6],L)$. thì $L = [1,3,5]$

Điều kiện dừng của bài toán rất đơn giản: khi danh sách là rỗng thì danh sách được tạo ra cũng là rỗng. Và phần đệ quy của bài toán phải dựa trên việc truy xuất từ phần tử đầu tiên của danh sách và gọi đệ quy cho phần đuôi. Chúng ta phải xét các trường hợp khác nhau của phần tử đầu.

```
ptle([], []) :- !.  
ptle([H|T], [H|T1]) :- H mod 2 = 1, ptle(T, T1), !.  
ptle([_|T], T1) :- ptle(T, T1).
```

Ở đây chúng ta lưu ý đến mệnh đề thứ ba: do hai mệnh đề trên đều có nhất cắt, nên mệnh đề thứ ba chỉ được so trùng khi hai mệnh đề trên đều sai. Mệnh đề thứ hai đã xét trường hợp khi H lẻ, vì vậy ở mệnh đề thứ ba chắc chắn H không phải là số lẻ, và vì vậy không cần phải xét đến.

Tóm tắt:

- Đệ quy là công cụ chủ yếu để xử lý danh sách trên Prolog
- Việc gọi đệ quy trên danh sách thường dựa trên cơ sở chia danh sách thành phần đầu và phần đuôi.
- Sử dụng vị từ nhất cắt hợp lý sẽ khiến các mệnh đề trở nên gọn nhẹ và logic hơn.