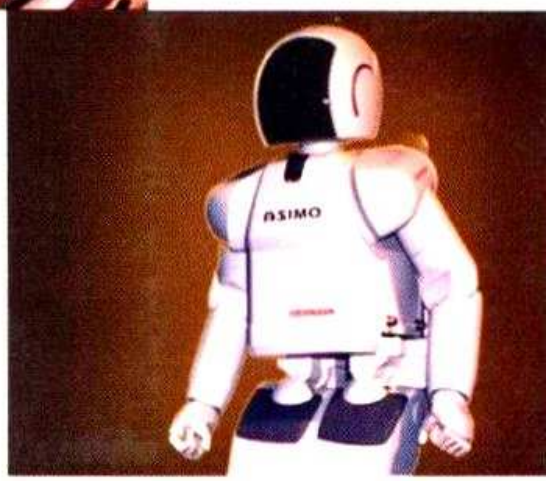
PHAN HUY KHÁNH

Lập trình lôgic trong PROLOG



NHÀ XUẤT BẢN ĐẠI HỌC QUỐC GIA HÀ NỘI

TS. PHAN HUY KHÁNH

LẬP TRÌNH LÔGIC TRONG PROLOG

NHÀ XUẤT BẢN ĐẠI HỌC QUỐC GIA HÀ NỘI

Mục lục

Lời nói đầu

Cuốn sách này nhằm cung cấp cơ sở lý thuyết và những phương pháp lập trình cơ bản nhất của môn học "Lập trình lôgic" (Programming in Logic). Người đọc sẽ được làm quen với một số kỹ thuật lập trình lôgic được ứng dụng tương đối phổ biến và chủ yếu trong lĩnh vực trí tuệ nhân tạo (Artificial Intelligence) như công nghệ xử lý tri thức, máy học, hệ chuyên gia, xử lý ngôn ngữ tự nhiên, trò chơi, v.v...

Cuốn sách gồm năm chương, trong mỗi chương, tác giả đều cố gắng đưa vào nhiều ví dụ minh họa. Nội dung các chương như sau:

Chương 1 giới thiệu ngôn ngữ lập trình Prolog. Người đọc được làm quen với các kiểu dữ liệu của Prolog, khái niệm luật, sự kiện đích và viết được các chương trình Prolog đơn giản.

Chương 2 trình bày các mức nghĩa khác nhau của một chương trình Prolog: nghĩa lôgic, nghĩa khai báo và nghĩa thủ tục, cách Prolog trả lời các câu hỏi, cách Prolog làm thoả mãn các đích.

Chương 3 trình bày các phép toán số học, phép so sánh các đối tượng và định nghĩa các hàm sử dụng phép đệ quy trong Prolog.

Chương 4 trình bày cấu trúc danh sách và các phép xử lý cơ bản trên danh sách của Prolog.

Chương 5 trình bày kỹ thuật lập trình nâng cao với Prolog.

Phần phụ lục giới thiệu ngôn ngữ lập trình SWI-Prolog, hướng dẫn cách cài đặt sử dụng phần mềm này và một số chương trình ví dụ tiêu biểu viết trong SWI Prolog đã chạy có kết quả.

Cuốn sách này dùng làm giáo trình cho sinh viên ngành Tin học và những bạn đọc muốn tìm hiểu thêm về kỹ thuật lập trình cho lĩnh vực trí tuệ nhân tạo.

Trong quá trình biên soạn, tác giả đã nhận được từ các bạn đồng nghiệp nhiều đóng góp bổ ích về mặt chuyên môn, những động viên khích lệ về mặt tinh thần, sự giúp đỡ về biên tập để cuốn sách được ra đời. Tác giả xin được bày tỏ lòng biết ơn sâu sắc.

Tác giả chân thành cảm ơn mọi ý kiến phê bình đóng góp của bạn đọc gần xa về nội dung của cuốn sách này.

Đà Nẵng, ngày 27/05/2004

Tác giả

Chương 1

Mở đầu về ngôn ngữ Prolog

"A program is a theory (in some logic)
and computation is deduction from the theory"
J. A. Robinson

"Program = data structure + algorithm"
N. Wirth

"Algorithm = logic + control"
R. Kowalski

1.1 Giới thiệu ngôn ngữ Prolog

1.1.1 Prolog là ngôn ngữ lập trình lôgic

Prolog là ngôn ngữ được sử dụng phổ biến nhất trong dòng các ngôn ngữ lập trình lôgic (Prolog có nghĩa là Programming in Logic). Ngôn ngữ Prolog do giáo sư người Pháp Alain Colmerauer và nhóm nghiên cứu của ông đề xuất lần đầu tiên tại trường Đại học Marseille đầu những năm 1970. Đến năm 1980, Prolog nhanh chóng được áp dụng rộng rãi ở châu Âu, được người Nhật chọn làm ngôn ngữ phát triển dòng máy tính thế hệ 5. Prolog đã được cài đặt trên các máy vi tính Apple II, IBM-PC, Macintosh.

Prolog còn được gọi là ngôn ngữ *lập trình ký hiệu* (symbolic programming) tương tự các ngôn ngữ *lập trình hàm* (functional programming), hay *lập trình phi số* (non-numerical programming). Prolog rất thích hợp để giải quyết các bài toán liên quan đến các *đối tượng* (object) và mối *quan hệ* (relation) giữa chúng.

Prolog được sử dụng phổ biến trong lĩnh vực trí tuệ nhân tạo. Nguyên lý lập trình lôgic dựa trên các *mệnh đề Horn* (Horn clauses). Một mệnh đề Horn biểu diễn một sự kiện hay một sự việc nào đó là đúng hoặc không đúng, xảy ra hoặc không xảy ra (có hoặc không có, v.v...).

Ví dụ 1.1: Sau đây là một số mệnh đề Horn:

1. ***Nếu*** *một người già mà (**và**) khôn ngoan **thì** người đó hạnh phúc.*
2. *Jim là người hạnh phúc.*
3. ***Nếu*** *X là cha mẹ của Y **và** Y là cha mẹ của Z **thì** X là ông bà của Z.*
4. *Tom là ông bà của Sue.*
5. *Tất cả mọi người đều chết*
   (hoặc ***Nếu*** *ai là người **thì** ai đó phải chết).*
6. *Socrat là người.*

Trong các mệnh đề Horn ở trên, các mệnh đề 1, 3, 5 được gọi là các *luật* (rule), các mệnh đề còn lại được gọi là các *sự kiện* (fact). Một chương trình lôgic có thể được xem như là một cơ sở dữ liệu gồm các mệnh đề Horn, hoặc dạng luật, hoặc dạng sự kiện, chẳng hạn như tất cả các sự kiện và luật từ 1 đến 6 ở trên. Người sử dụng (NSD) gọi chạy một chương trình lôgic bằng cách đặt *câu hỏi* (query/ question) truy vấn trên cơ sở dữ liệu này, chẳng hạn câu hỏi:

Socrat có chết không (tương đương với khẳng định "*Socrat chết*" là đúng hay sai)?

Một hệ thống lôgic sẽ thực hiện chương trình theo cách "suy luận"-tìm kiếm dựa trên vốn "hiểu biết" đã có là chương trình - cơ sở dữ liệu, để minh chứng câu hỏi là một khẳng định *đúng* (Yes) hoặc *sai* (No). Với câu hỏi trên, hệ thống tìm kiếm trong cơ sở dữ liệu khẳng định *Socrat chết bằng cách xem ai đó là Socrat* và "tìm thấy" luật 5 thoả mãn (vế ***thì***). Vận dụng luật 5, hệ thống nhận được *Socrat là người* (vế ***nếu***) chính là sự kiện 6. Từ đó, câu trả lời sẽ là:

```
Yes
```

có nghĩa sự kiện Socrat chết là đúng.

1.1.2 Cú pháp Prolog

1.1.2.1 Các thuật ngữ

Một chương trình Prolog là một cơ sở dữ liệu gồm các *mệnh đề* (clause). Mỗi mệnh đề được xây dựng từ các *vị từ* (predicate). Một vị từ là một phát biểu nào đó về các đối tượng có giá trị chân lý *đúng* (true) hoặc *sai* (fail). Một vị từ có thể có các đối là các *nguyên tử* (logic atom).

Mỗi nguyên tử (nói gọn) biểu diễn một quan hệ giữa các *hạng* (term). Như vậy, hạng và quan hệ giữa các hạng tạo thành mệnh đề.

Hạng được xem là những đối tượng "dữ liệu" trong một trình Prolog. Hạng có thể là *hạng sơ cấp* (elementary term) gồm *hằng* (constant) hay trực kiện (literal), *biến* (variable) và các *hạng phức hợp* (compound term).

Các hạng phức hợp biểu diễn các đối tượng phức tạp của bài toán cần giải quyết thuộc lĩnh vực đang xét. Hạng phức hợp là một *hàm tử* (functor) có chứa các *đối* (argument), có dạng

Tên_hàm_tử(Đối_1, ..., Đối_n)

Tên hàm tử là một chuỗi chữ cái và/hoặc chữ số được bắt đầu bởi một chữ cái thường. Các đối có thể là biến, hạng sơ cấp, hoặc hạng phức hợp. Trong Prolog, hàm tử đặc biệt "." (dấu chấm) biểu diễn cấu trúc *danh sách* (list). Kiểu dữ liệu hàm tử tương tự kiểu bản ghi (record) và *danh sách* (list) tương tự kiểu mảng (array) trong các ngôn ngữ lập trình mệnh lệnh (C, Pascal...).

Ví dụ 1.2:

```
f(b, a, b).
student(robert, 1975, info, 2, address(6, 'mai
juin', 'Caen')).
[a, b, c]
```

Mệnh đề có thể là một *sự kiện*, một *luật* (hay *quy tắc*), hay một *câu hỏi*.

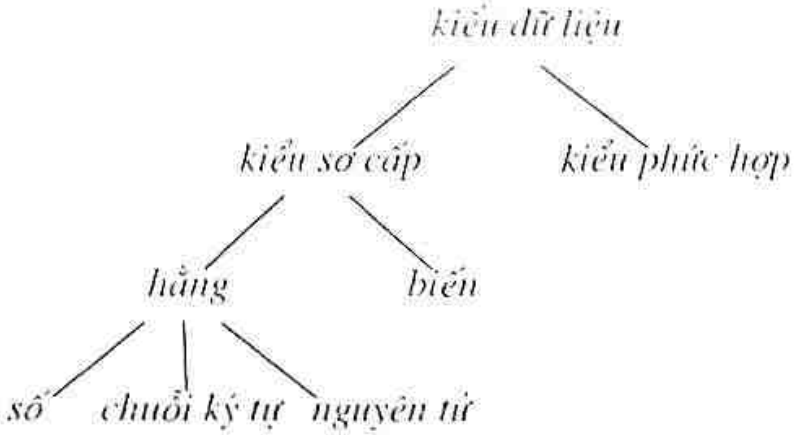
Prolog quy ước viết sau mỗi mệnh đề một dấu chấm để kết thúc như sau:

- Sự kiện: < ... >. (tương ứng với luật < ... > :- true.)
- Luật: < ... > :- < ... >.
- Câu hỏi ?- < ... >. (ở chế độ tương tác có dấu nhắc lệnh)

1.1.2.2 Các kiểu dữ liệu Prolog

Hình 1.1. biểu diễn một sự phân lớp các kiểu dữ liệu trong Prolog gồm kiểu dữ liệu sơ cấp và kiểu dữ liệu có cấu trúc. Sự phân lớp này nhận biết kiểu của một đối tượng nhờ bề ngoài cú pháp.

Cú pháp của Prolog quy định mỗi kiểu đối tượng có một dạng khác nhau. Prolog không cần cung cấp một thông tin nào khác để nhận biết kiểu của một đối tượng. Trong Prolog, NSD không cần khai báo kiểu dữ liệu.



Hình 1.1. Các kiểu dữ liệu trong Prolog

Các kiểu dữ liệu Prolog được xây dựng từ các ký tự ASCII:

- Các chữ cái in hoa A, B, ..., Z và chữ cái in thường a, b, ..., z.
- Các chữ số 0, 1, ..., 9.
- Các ký tự đặc biệt, chẳng hạn + - * / < > = : . & _ ? .

1.1.2.3 Chú thích

Trong một chương trình Prolog, *chú thích* (comment) được đặt giữa hai cặp ký hiệu /* và */ (tương tự ngôn ngữ C). Ví dụ:

```
/************************/
```

```
/*** Đây là một chú thích ***/
/*****************************/
```

Trong trường hợp muốn đặt một chú thích ngắn sau mỗi phần khai báo Prolog cho đến hết dòng, có thể đặt trước một ký hiệu %.

Ví dụ

```
%%%%%%%%%%%%%%%%%%%%%%%%%%
% Đây cũng là một chú thích
%%%%%%%%%%%%%%%%%%%%%%%%%%
```

Prolog sẽ bỏ qua tất cả các phần chú thích trong thủ tục.

1.2 Các kiểu dữ liệu sơ cấp của Prolog

1.2.1 Các kiểu hằng

1.2.1.1 Kiểu hằng số

Prolog sử dụng cả số nguyên và số thực. Cú pháp của các số nguyên và số thực rất đơn giản, chẳng hạn như các ví dụ sau:

```
1        1515       0        -97
3.14     -0.0035    100.2
```

Tuỳ theo phiên bản cài đặt, Prolog có thể xử lý các miền số nguyên và miền số thực khác nhau. Ví dụ trong phiên bản Turbo Prolog, miền số nguyên cho phép từ `-32768` đến `32767`, miền số thực cho phép từ `±1e-307` đến `±1e+308`. Các số thực rất ít khi được sử dụng trong Prolog. Lý do chủ yếu ở chỗ Prolog là ngôn ngữ lập trình ký hiệu, phi số.

Các số nguyên thường chỉ được sử dụng khi cần đếm số lượng các phần tử hiện diện trong một danh sách Prolog dạng `[a1, a2, ..., an]`.

1.2.1.2 Kiểu hằng lôgic

Prolog sử dụng hai hằng lôgic có giá trị là `true` và `fail`. Thông thường các hằng lôgic không được dùng như tham số mà được dùng như các mệnh đề. Hằng `fail` thường được dùng để tạo sinh lời giải bài toán.

1.2.1.3 Kiểu hằng chuỗi ký tự

Các hằng là chuỗi ký tự (string) được đặt giữa hai dấu nháy kép.

`"Toto \#\{@ tata"`	chuỗi có tuỳ ý ký tự
`""`	chuỗi rỗng (empty string)
`"\""`	chuỗi chỉ có một dấu nháy kép

1.2.1.4 Kiểu hằng nguyên tử

Các hằng nguyên tử Prolog là chuỗi ký tự ở một trong ba dạng như sau:

(1) Chuỗi gồm chữ cái, chữ số và ký tự _ luôn luôn được *bắt đầu bằng một chữ cái in thường.*

`newyork`	`a_`
`nil`	`x_y`
`x25`	`tom_cruise`

(2) Chuỗi các ký tự đặc biệt:

`<--->`	`.:.`
`======>`	`::==`
`...`	

(3) Chuỗi đặt giữa hai dấu nháy đơn (quote) *được bắt đầu bằng chữ in hoa*, dùng phân biệt với các tên biến:

`'Jerry'`	`'Tom SMITH'`

1.2.2 Biến

Tên biến là một chuỗi ký tự gồm chữ cái, chữ số, *bắt đầu bởi chữ hoa hoặc dấu gạch dưới dòng:*

```
X, Y, A
Result, List_of_members
_x23, _X, _
```

1.3 Sự kiện và luật trong Prolog

1.3.1 Xây dựng sự kiện

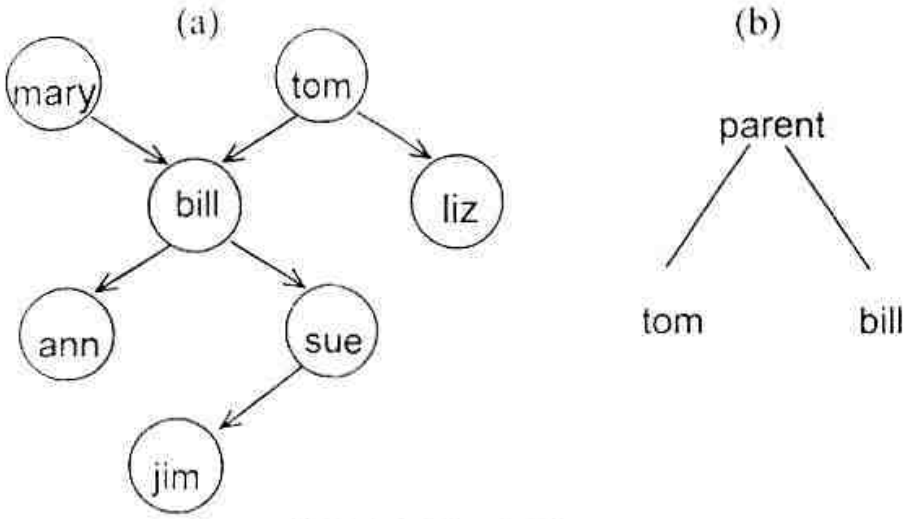
Ví dụ 1.3. Quan hệ gia đình.

Để xây dựng các sự kiện trong một chương trình Prolog, ta lấy một ví dụ về cây gia hệ như hình 1.2.

Trong cây gia hệ (a), các nút chỉ người, còn các mũi tên chỉ quan hệ *cha mẹ của* (parent of). Sự kiện *Tom là cha mẹ của Bill* được viết thành một vị từ Prolog như sau (chú ý mệnh đề được kết thúc bởi một dấu chấm):

`parent(tom, bill).` (Chú ý không có dấu cách trước dấu mở ngoặc)

Ở đây, vị từ `parent` có hai đối là `tom` và `bill`. Người ta có thể biểu diễn vị từ này bằng một cây như trong hình 1.2 (b): nút gốc là tên vị từ, các nút là các đối.



Hình 1.2. Cây gia hệ

Từ cây gia hệ trên đây, có thể tiếp tục viết các vị từ khác để nhận được một chương trình Prolog gồm 6 vị từ như sau:

```
parent(mary, bill).
parent(tom, bill).
parent(tom, liz).
parent(bill, ann).
```

```
parent(bill, sue).
parent(sue, jim).
```

Sau khi hệ thống Prolog nhận được chương trình này, thực chất là một cơ sở dữ liệu, người ta có thể đặt ra các câu hỏi liên quan đến quan hệ parent. Ví dụ câu hỏi *Bill có phải là cha mẹ của Sue* được gõ vào trong hệ thống đối thoại Prolog (dấu nhắc ?-_) như sau:

```
?- parent(bill, sue).
```

Sau khi tìm thấy sự kiện này trong chương trình, Prolog trả lời:

```
Yes
```

Ta tiếp tục đặt câu hỏi khác:

```
?- parent(liz, sue).
No
```

Bởi vì Prolog không tìm thấy sự kiện Liz là người mẹ của Sue trong chương trình. Tương tự, Prolog trả lời No cho sự kiện:

```
?- parent(tom, ben).
```

Vì tên ben chưa được đưa vào trong chương trình. Ta có thể tiếp tục đặt ra các câu hỏi thú vị khác. Chẳng hạn, ai là cha (hay mẹ) của Liz?

```
?- parent(X, liz).
```

Lần này, trước khi đưa ra trả lời Yes hoặc No, Prolog đưa ra một giá trị của X làm thoả mãn câu hỏi trên đây:

```
X = tom
```

Để biết được ai là con của Bill, ta chỉ cần viết:

```
?- parent(bill, X).
```

Với câu hỏi này, Prolog sẽ có hai câu trả lời, đầu tiên là:

```
X = ann ->;
```

Để biết được câu trả lời tiếp theo, trong hầu hết các cài đặt của Prolog, NSD phải gõ vào một dấu chấm phẩy (;) sau -> (Arity Prolog):

```
X = sue
```

Nếu đã hết phương án trả lời mà vẫn tiếp tục yêu cầu (;), Prolog trả lời No, ngược lại trả lời Yes.

NSD có thể đặt các câu hỏi tổng quát hơn, chẳng hạn: *ai là cha mẹ của ai?* Nói cách khác, cần tìm X và Y sao cho X là cha mẹ của Y. Ta viết như sau:

```
?- parent(X, Y).
```

Sau khi hiển thị câu trả lời đầu tiên, Prolog sẽ lần lượt tìm kiếm những cặp cha mẹ – con thoả mãn và lần lượt hiển thị kết quả nếu chừng nào NSD còn yêu cầu cho đến khi không còn kết quả lời giải nào nữa (kết thúc bởi Yes):

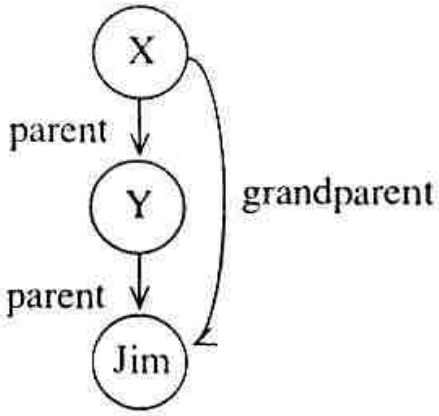
```
X = mary
Y = bill ->;
X = tom
Y = bill ->;
X = tom
Y = liz ->;
X = bill
Y = ann ->;
X = bill
Y = sue ->;
X = sue
Y = jim
Yes
```

Tuỳ theo cài đặt Prolog, NSD có thể gõ vào một dấu chấm (.) hoặc Enter để chấm dứt giữa chừng luồng trả lời.

Ta có thể tiếp tục đưa ra những câu hỏi phức tạp hơn khác, chẳng hạn *ai là ông (bà) của Jim ?* Thực tế, quan hệ ông – bà (grandparent) chưa được định nghĩa, cần phải phân tách câu hỏi này thành hai phần sơ cấp hơn:

Ai là cha (mẹ) của Jim? Giả sử có tên là Y.

Ai là cha (mẹ) của Y? Giả sử có tên là X.



Hình 1.3. Quan hệ ông bà được hợp thành từ hai quan hệ cha mẹ

Lúc này, có thể viết trong Prolog như sau:

```
?- parent(Y, jim), parent(X, Y).
```

Prolog trả lời:

```
Y = sue
X = bill
Yes
```

Câu hỏi trên đây tương ứng với câu hỏi: tìm X và Y thoả mãn:

```
parent(Y, jim)
```

và

```
parent(X, Y).
```

Nếu thay đổi thứ tự hai thành phần câu hỏi, thì nghĩa lôgic vẫn không thay đổi và Prolog trả lời cùng kết quả (có thể thay đổi về thứ tự), nghĩa là ta có thể đặt câu hỏi như sau:

```
?- parent(X, Y), parent(Y, jim).
X = bill
Y = sue
Yes
```

Bây giờ ta đặt câu hỏi *ai là cháu của Tom*?

```
?- parent(tom, X), parent(X, Y).
X = bill
```

```
Y = ann->;
X = bill
Y = sue ->;
No
```

Một câu hỏi khác có thể như sau: *Ann và Sue có cùng cha mẹ không?* nghĩa là ta diễn đạt thành hai giai đoạn:

1. Tìm X là cha mẹ của Ann.

2. X tìm thấy có cùng là cha mẹ của Sue không?

Câu hỏi và trả lời trong Prolog như sau:

```
?- parent(X, ann), parent(X, sue).
X = bill
```

Trong Prolog, câu hỏi còn được gọi là *đích* (goal) cần phải được *thoả mãn* (satisfy). Mỗi câu hỏi đặt ra đối với cơ sở dữ liệu có thể tương ứng với một hoặc nhiều đích. Chẳng hạn dãy các đích:

```
parent(X, ann), parent(X, sue).
```

tương ứng với câu hỏi là *phép hội* (conjunction) của 2 mệnh đề:

X là một cha mẹ của Ann, *và*

X là một cha mẹ của Sue.

Nếu câu trả lời là `Yes`, thì có nghĩa đích đã được thoả mãn, hay đã *thành công*. Trong trường hợp ngược lại, câu trả lời là `No`, có nghĩa đích *không được thoả mãn*, hay đã *thất bại*.

Nếu có nhiều câu trả lời cho một câu hỏi, Prolog sẽ đưa ra câu trả lời đầu tiên và chờ yêu cầu của NSD tiếp tục.

1.3.2 Xây dựng luật

1.3.2.1 Định nghĩa luật

Từ chương trình gia hệ trên đây, ta có thể dễ dàng bổ sung các thông tin khác, chẳng hạn bổ sung các sự kiện về giới tính (nam, nữ) của những người đã nêu tên trong quan hệ `parent` như sau:

```
woman(mary).
man(tom).
man(bill).
woman(liz).
woman(sue).
woman(ann).
man(jim).
```

Ta đã định nghĩa các quan hệ *đơn* (unary) `woman` và `man` vì chúng chỉ liên quan đến một đối tượng duy nhất. Còn quan hệ `parent` là nhị phân, vì liên quan đến một cặp đối tượng. Như vậy, các quan hệ đơn dùng để thiết lập một thuộc tính của một đối tượng. Mệnh đề:

```
woman(mary).
```

được giải thích: *Mary là nữ*. Tuy nhiên, ta cũng có thể sử dụng quan hệ nhị phân để định nghĩa giới tính:

```
sex(mary, female).
sex(tom, male).
sex(bill, male).
. . .
```

Bây giờ ta đưa vào một quan hệ mới `child`, đối ngược với `parent` như sau:

```
child(liz, tom).
```

Từ đó, ta định nghĩa luật mới như sau:

```
child(Y, X) :- parent(X, Y).
```

Luật trên được hiểu là:

Với mọi X và Y,

Y là con của X nếu

X là cha (hay mẹ) của Y.

hay

Với mọi X và Y,

nếu X là cha (hay mẹ) của Y thì

Y là con của X.

Có sự khác nhau cơ bản giữa sự kiện và luật. Một sự kiện, chẳng hạn:

```
parent(tom, liz).
```

là một điều gì đó luôn đúng, không có điều kiện gì ràng buộc. Trong khi đó, các luật liên quan đến các thuộc tính chỉ được thoả mãn nếu một số điều kiện nào đó được thoả mãn. Mỗi luật bao gồm hai phần:

- phần bên phải (RHS: Right Hand Side) chỉ *điều kiện*, còn được gọi là *thân* (body) của luật, và
- phần bên trái (LHS: Left Hand Side) chỉ *kết luận*, còn được gọi là *đầu* (head) của luật.

Nếu điều kiện `parent(X, Y)` là đúng, thì `child(Y, X)` cũng đúng và là hậu quả lôgic của phép suy luận (inference).

```
child(Y, X) :- parent(X, Y).
```

đầu thân

Câu hỏi sau đây giải thích cách Prolog sử dụng các luật: Liz có phải là con của Tom không?

```
?- child(liz, tom)
```

Thực tế, trong chương trình không có sự kiện nào liên quan đến con, mà ta phải tìm cách áp dụng các luật. Luật trên đây ở dạng tổng quát với các đối tượng X và Y bất kỳ, mà ta lại cần các đối tượng cụ thể `liz` và `tom`.

Ta cần sử dụng *phép thế* (substitution) bằng cách gán giá trị `liz` cho biến Y và `tom` cho X. Người ta nói rằng các biến X và Y đã được *ràng buộc* (bound):

```
X = tom
```

và

```
Y = liz
```