Trí Tuệ Nhân Tạo

Nguyễn Nhật Quang

quangnn-fit@mail.hut.edu.vn

Viện Công nghệ Thông tin và Truyền thông
Trường Đại học Bách Khoa Hà Nội

Năm học 2009-2010

Nội dung môn học:

- Giới thiệu về Trí tuệ nhân tạo
- Tác tử
- Giải quyết vấn đề: Tìm kiếm, Thỏa mãn ràng buộc
- **Logic và suy diễn**
  - **Lập trình logic Prolog**

    (Dựa trên bài giảng csc.villanova.edu/~dmatusze/8310summer2001/index.html/prolog1.ppt)
- Biểu diễn tri thức
- Suy diễn với tri thức không chắc chắn
- Học máy
- Lập kế hoạch

Lập trình logic

- Một chương trình logic biểu diễn một cơ sở tri thức (một tập các mệnh đề logic)
- Các mệnh đề logic phải ở dạng chuẩn Horn

  $p_1 \wedge p_2 \wedge \ldots \wedge p_n \rightarrow q_1 \vee q_2 \vee \ldots \vee q_m$
- Nếu n=0, m=1, thì q_1 là một **sự kiện (fact)**
- Nếu n≥1, m=1, thì ($p_1 \wedge p_2 \wedge \ldots \wedge p_n \rightarrow q_1$) là một **luật (rule)**
- Nếu n≥1, m>1, thì tương đương với luật ($p_1 \wedge p_2 \wedge \ldots \wedge p_n \wedge \neg q_1 \wedge \neg q_2 \wedge \ldots \wedge \neg q_{m-1} \rightarrow q_m$)

Ngôn ngữ lập trình logic Prolog

- Prolog = **Prog**ramming **Log**ic
- Một ngôn ngữ lập trình logic được sử dụng rất phổ biến
- Trong ngôn ngữ Prolog
  - Các luật và các sự kiện là các tiên đề (axioms)
  - Câu hỏi, được đưa ra bởi người dùng, là định lý cần chứng minh
- Prolog áp dụng phương pháp suy diễn quay lui (Back Chaining) để chứng minh
- Để chứng minh P(a)
  - Tìm sự kiện P(t) hoặc luật ($Q_1 \wedge Q_2 \wedge \ldots \wedge Q_n \rightarrow P(t)$)
  - Nếu tìm được sự kiện P(t), thay thế t=a, định lý được chứng minh
  - Nếu tìm được luật ($Q_1 \wedge Q_2 \wedge \ldots \wedge Q_n \rightarrow P(t)$), thì cần tiếp tục chứng minh các giả thiết
  - Nếu có các biến, tìm cách thay thế các biến này bằng các giá trị, sao cho đích chứng minh được thỏa mãn

Phần mềm SWI-Prolog

- SWI-Prolog là một công cụ lập trình Prolog được sử dụng rất phổ biến, có các phiên bản chạy trên các hệ điều hành Windows, MacOS, và Linux
- Bản quyền sử dụng phần mềm SWI-Prolog là miễn phí cho mục đích học tập và nghiên cứu
- SWI-Prolog có thể tải về từ địa chỉ: http://www.swi-prolog.org/

Ví dụ về chương trình Prolog (1)

- Suy diễn logic
  - Tuấn là một sinh viên của HUT
  - Mọi sinh viên của HUT đều học môn Toán rời rạc
  - Tuấn có học môn Toán rời rạc không?
- Chương trình Prolog
  - `studentHUT(tuan).`
  - `studDiscretMath(X) :- studentHUT(X).`
  - `?- studDiscretMath(tuan).`

Ví dụ về chương trình Prolog (2)

- Sự kiện: Tuấn là một sinh viên của HUT
  - studentHUT(tuan).
- Luật: Mọi sinh viên của HUT đều học môn Toán rời rạc
  - studDiscretMath(X) :- studentHUT(X).
- Câu hỏi (của người dùng): Tuấn có học môn Toán rời rạc không?
  - ?- studDiscretMath(tuan).
- Các câu hỏi có cùng dạng như các sự kiện

Chạy chương trình Prolog (1)

- Sử dụng một chương trình soạn thảo (vd: Notepad) để tạo ra chương trình (cơ sở tri thức)
- Ghi lại chương trình trong một tập tin định dạng văn bản (text only) sử dụng đuôi của tập tin là .pl
- Ví dụ một chương trình Prolog:

```
studentHUT(tuan).

studDiscretMath(X) :- studentHUT(X).
```

Chạy chương trình Prolog (2)

- Để chạy chương trình Prolog:
  - (Với hệ điều hành Windows), kích đúp lên tập tin chương trình, hoặc
  - Chạy phần mềm SWI-Prolog, và tham vấn (consult) tới tập tin chương trình

```
?- consult('C:\\PrologPrograms\\myPrologProg.pl').
```

- Sau đó, đưa ra câu hỏi mong muốn

```
?- studDiscretMath(tuan).
```

- Prolog đưa ra câu trả lời

```
Yes
```

Prolog – Chứng minh định lý

- Prolog trả về giá trị Yes có nghĩa là "chứng minh được"
- Prolog trả về giá trị No có nghĩa là "không thể chứng minh"

```
?- stud_DiscretMath(hai).
No
```

- *Closed world assumption*: Chương trình Prolog biết tất cả về những gì nó cần biết
- Prolog cung cấp các giá trị cho các biến khi cần, để có thể hoàn chỉnh một chứng minh

```
?- stud_DiscretMath(X).
X = tuan
```

Cú pháp: Các cấu trúc

- **Một cấu trúc (structure)** gồm một tên (name) và không, một, hoặc nhiều tham số (arguments)
- Bỏ đi cặp dấu ngoặc (), nếu như không có tham số
- Các ví dụ của cấu trúc
  - sunshine
  - man(socrates)
  - path(garden, south, sundial)

Cú pháp: Các mệnh đề cơ sở

- Một cấu trúc chính là một **mệnh đề cơ sở (base clause)**
- Một mệnh đề cơ sở biểu diễn một **sự kiện (fact)**
- Các ví dụ của mệnh đề cơ sở
  - john.
  - mary
  - bill
  - loves(john, mary).
  - loves(mary, bill).

Cú pháp: Biểu diễn luật

- Một **luật** được biểu diễn bao gồm
  - Một cấu trúc (biểu diễn mệnh đề kết luận của luật – mệnh đề THEN)
  - Ký hiệu **:-**
  - Một danh sách các cấu trúc (biểu diễn mệnh đề giả thiết của luật – mệnh đề IF), ngăn cách nhau bởi dấu **,**
- Các ví dụ của luật
  - mortal(X) :- man(X).
  - happy(X) :- healthy(X), wealthy(X), wise(X).
- Dấu **,** giữa các cấu trúc có nghĩa như toán tử logic AND

Cú pháp: Các vị từ logic

- Một **vị từ (predicate)** là một tập hợp các mệnh đề với cùng tên và một số (>=1) các tham số
- Ví dụ: Các mệnh đề biểu diễn vị từ **loves**

```
loves(john, mary).
loves(mary, bill).
loves(chuck, X) :- female(X), rich(X).
```

Cú pháp: Chương trình

- **Một chương trình (program)** là một tập hợp các mệnh đề
- Các mệnh đề trong chương trình có thể được sắp xếp theo bất kỳ trật tự (thứ tự) nào
- Các mệnh đề của một vị từ được sử dụng (được xét đến) theo đúng trật tự của chúng trong chương trình

Cú pháp: Các biến và hằng

- Các biến bắt đầu bằng chữ cái in hoa hoặc ký tự đặc biệt

  Vd: `X, Socrates, _result`

- Các hằng *không* bắt đầu bằng chữ cái in hoa hoặc ký tự đặc biệt

  Vd: `x, socrates`

- Các hằng chứa các ký tự đặc biệt, hoặc bắt đầu với chữ cái in hoa, thì phải đặt vào trong cặp ký tự nháy đơn ''

  Vd: `'C:\\My Documents\\examples.pl'`

- Để hiển thị ký tự nháy đơn, thì phải sử dụng: '' hoặc \'

Cú pháp: Chú thích

- Được đặt trong cặp dấu: /*…*/
- Hoặc được đặt sau dấu %
- Ví dụ
  - parent(x,y). % sự kiện mô tả x là cha mẹ của y

Cú pháp: Các lỗi hay gặp

- Phân biệt giữa chữ hoa và chữ thường là rất quan trọng
- Không được có khoảng trắng (space characters) giữa tên và danh sách tham số của một cấu trúc
  - Biểu diễn hợp lệ: `man(socrates).`
  - Biểu diễn không hợp lệ: `man (socrates).`
- Kết thúc của mỗi mệnh đề phải có dấu chấm (.)
  - Có thể đặt dấu chấm ở dòng kế tiếp

Suy diễn lùi trong Prolog

- Giả sử chúng ta có cơ sở tri thức (chương trình Prolog):

```
loves(chuck, X) :- female(X), rich(X).
female(jane).
female(mary).
rich(mary).
```

- Giả sử người dùng đặt câu hỏi

```
loves(chuck, X)
```

- Quá trình suy diễn lùi (Back chaining) của Prolog:
  - female(X) = female(jane), X = jane.
    rich(jane). : thất bại (không thể chứng minh được!)
  - female(X) = female(mary), X = mary.
    rich(mary). : thành công (chứng minh được!)

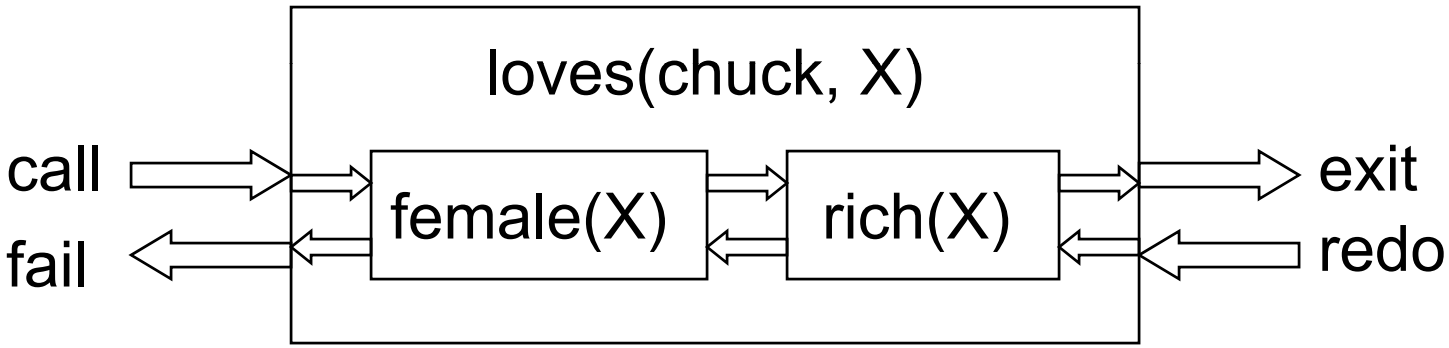
Các lời gọi

- Mỗi lời gọi (call) trong chương trình Prolog được thể hiện bởi:

- Mỗi cấu trúc (trong chương trình Prolog) có bốn cổng (ports): call, exit, redo, fail
- Cổng exit kết nối với cổng call
- Cổng fail kết nối với cổng redo

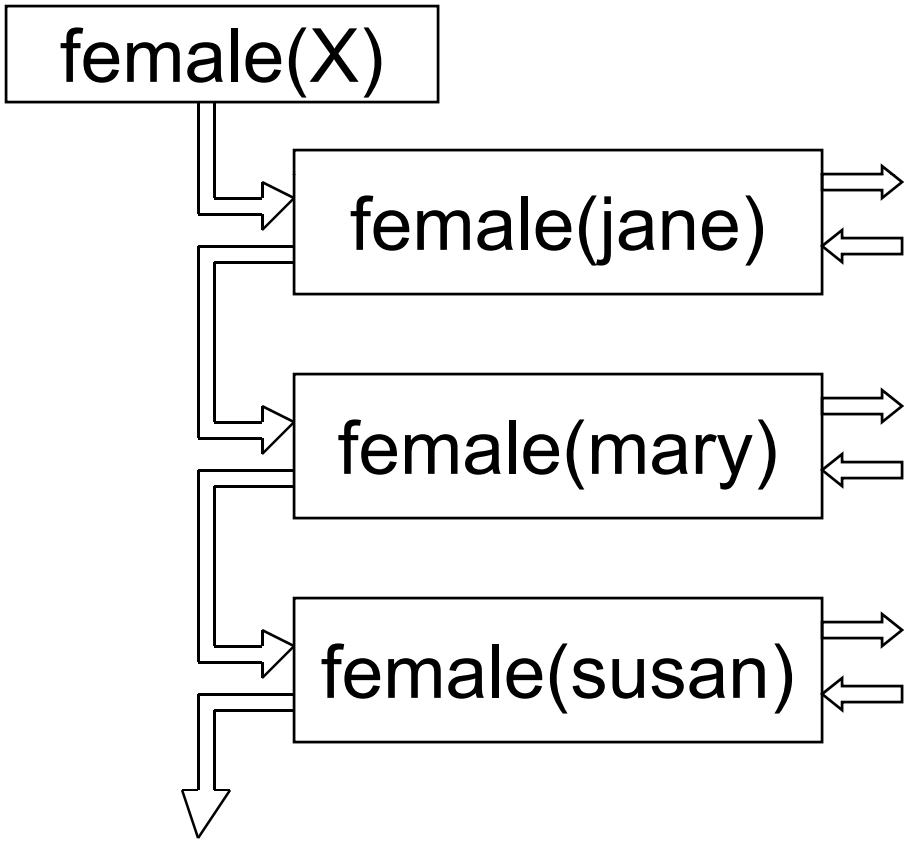
Các lời gọi lồng nhau (nested calls)

- loves(chuck, X) :- female(X), rich(X).



Các câu trả lời bổ sung

- female(jane).
  female(mary).
  female(susan).
- ?- female(X).
  X = jane
  X = mary
  X = susan
  Yes



Các cách đọc

- loves(chuck, X) :- female(X), rich(X).
- Cách đọc theo kiểu khai báo: Chuck yêu X nếu X là phụ nữ và giàu có
- Cách đọc theo kiểu thủ tục xấp xỉ: Để tìm được một đối tượng X mà Chuck yêu, thì trước tiên phải tìm được một phụ nữ X, sau đó kiểm tra xem X có giàu hay không
- Cách đọc theo kiểu khai báo thường được ưa thích hơn

Logic đơn điệu

- Logic đơn điệu (monotonic logic): Một khi chứng minh được một điều gì đó là đúng, thì điều đó sẽ mãi mãi đúng
- Logic đơn điệu không phù hợp với các bài toán (ứng dụng) thực tế
  - Nếu cái ví ở trong túi xách tay và túi xách tay ở trong xe ô-tô, thì chúng ta có thể kết luận là cái ví ở trong xe ô-tô
  - Nhưng, điều gì xảy ra nếu cái ví được lấy ra khỏi xe ô-tô?

Logic không đơn điệu

- Prolog sử dụng logic không đơn điệu (nonmonotonic logic)
  - Một mệnh đề được chứng minh là đúng ở một thời điểm trước có thể không còn (không thể được chứng minh) là đúng ở một thời điểm sau
- Trong Prolog, các sự kiện và các luật có thể được thay đổi ở một thời điểm nào đó
  - Những sự kiện và luật như vậy được gọi là động (không cố định)
- `assert(...)` : để bổ sung một sự kiện hoặc một luật vào cơ sở tri thức
- `retract(...)` : để loại bỏ một sự kiện hoặc một luật khỏi cơ sở tri thức
- `assert` và `retract` được gọi là các vị từ ngoài logic *(extralogical predicates)*

Các ví dụ về **assert** và **retract**

- assert(man(plato)).
- assert((loves(chuck,X) :- female(X), rich(X))).
- retract(man(plato)).
- retract((loves(chuck,X) :- female(X), rich(X))).

- Lưu ý là đối với các luật, phải sử dụng 2 cặp ngoặc đơn ((…))
  - Để tránh xảy ra lỗi cú pháp
  - Giả sử nếu viết: assert(foo :- bar, baz).
  - Sẽ có bao nhiêu tham số đối với assert?

Sử dụng Prolog trong bài toán thực tế

- Trong các bài toán (ứng dụng) thực tế, mọi thứ có thể thay đổi (một điều đã chứng minh là đúng, thì có thể sẽ không còn đúng vào một thời điểm sau đó)
- Prolog rất phù hợp cho việc biểu diễn các thay đổi trong các bài toán thực tế
- Các ứng dụng trò chơi là một ví dụ điển hình về các bài toán mà trong đó các sự việc có thể thay đổi
- Prolog là ngôn ngữ rất phù hợp để xây dựng các chương trình trò chơi mạo hiểm

Ví dụ minh họa chương trình trò chơi

- Chạy ứng dụng SWI-Prolog…

```
Welcome to SWI-Prolog (Multi-threaded, 32 bits,
   Version 5.6.59)
Copyright (c) 1990-2008 University of Amsterdam.
SWI-Prolog comes with ABSOLUTELY NO WARRANTY.
   This is free software, and you are welcome to
   redistribute it under certain conditions.
Please visit http://www.swi-prolog.org for details.
For help, use ?- help(Topic). or ?- apropos(Word)

1 ?- consult('C:\\prolog\\dragon.pl').
   % C:\prolog\dragon.pl compiled 0.00 sec, 12,468 bytes
true.
```

Các lệnh của trò chơi

- ?- start.

Enter commands using standard Prolog syntax.
Available commands are:

start.	-- to start the game.
n. s. e. w.	-- to go in that direction.
take(Object).	-- to pick up an object.
drop(Object).	-- to put down an object.
use(Object).	-- to use an object.
attack.	-- to attack an enemy.
look.	-- to look around you again.
instructions.	-- to see this message again.
halt.	-- to end the game and quit.

Bắt đầu chương trình

- You are in a meadow. To the north is the dark mouth of a cave; to the south is a small building. Your assignment, should you decide to accept it, is to recover the famed Bar-Abzad ruby and return it to this meadow.

  true.

Đi về hướng nam

- ?- s.

  You are in a small building. The exit is to the north. The room is devoid of furniture, and the only feature seems to be a small door to the east.

  There is a flashlight here.

  true.

Lấy đồ vật, Cửa bị khóa

- ?- take(flashlight).
  OK.

  true.

- ?- e.
  The door appears to be locked.
  You can't go that way.

  true.

Mở khóa cửa, Quan sát

- ?- use(key).
  The closet is no longer locked.

  true.

- ?- look.
  You are in a big, dark cave. The air is fetid.

  There is a chest here.

  true.

Biểu diễn các sự kiện

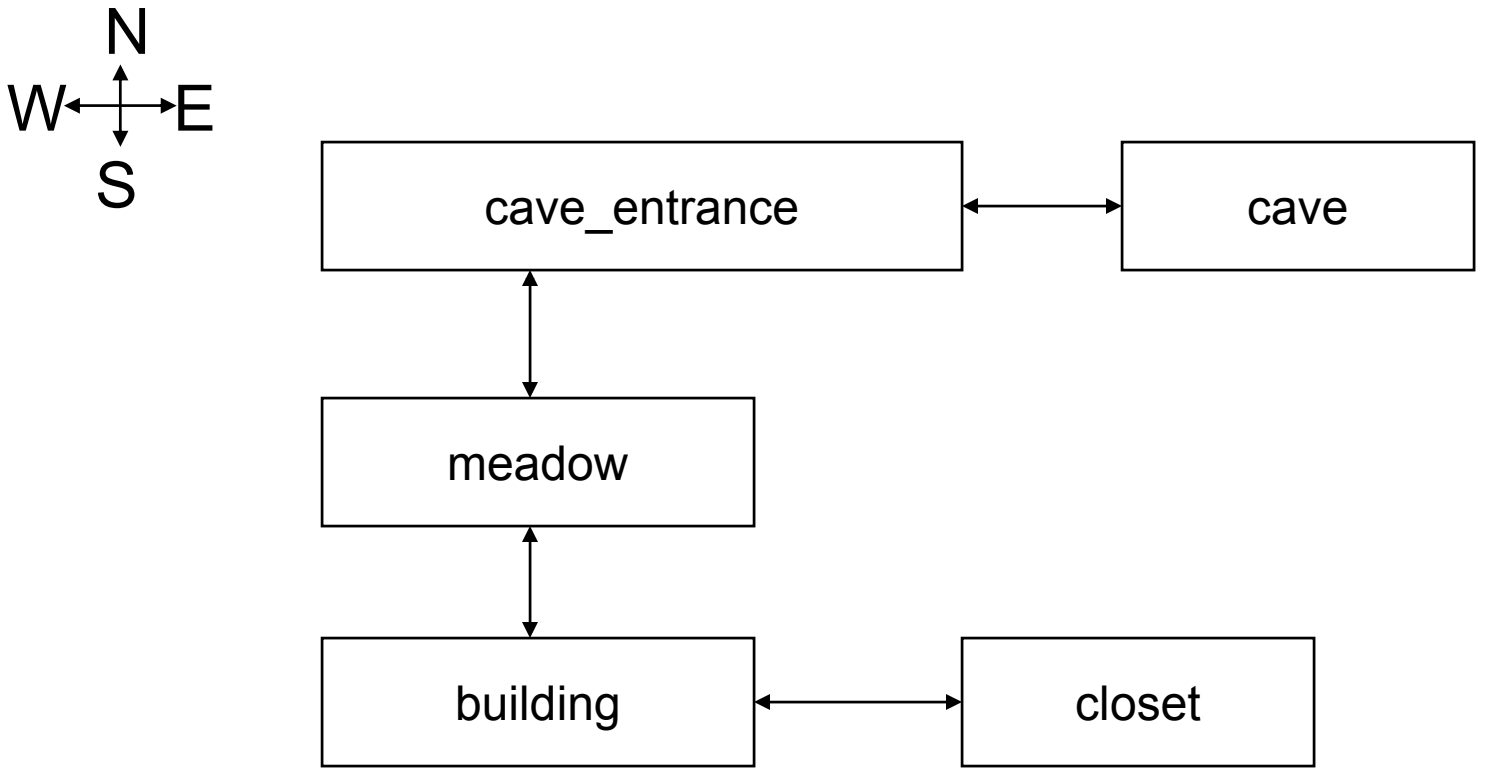
- Vị trí (nơi) hiện tại của tôi:
  - i_am_at(meadow).
- Vị trí của các đồ vật:
  - at(flashlight, building).
- Tôi đang giữ (nắm) đồ vật gì:
  - holding(key).
- Những sự kiện nào có thể được thay đổi:
  - :- dynamic i_am_at/1, at/2, holding/1.

Đầu vào và đầu ra

- Đầu vào (input) là các lệnh (các câu hỏi) trực tiếp đối với Prolog
  - take(flashlight).
- Đầu vào (output): sử dụng write(...) để đưa ra các hiển thị
- Sử dụng nl để kết thúc dòng hiển thị (để bắt đầu viết dòng hiển thị mới)
- Ví dụ:

```
describe(closet) :-
write('You are in an old storage closet.'), nl.
```

Bản đồ của trò chơi



Biểu diễn bản đồ

- path(cave, w, cave_entrance).
  path(cave_entrance, e, cave).
- path(meadow, s, building).
  path(building, n, meadow).
- Hoặc có thể biểu diễn như sau:
  - path(cave, w, cave_entrance).
    path(X, e, Y) :- path(Y, w, X).

Liệt kê

- Liệt kê các mệnh đề của một vị từ là một cách để kiểm tra trạng thái hiện tại của chương trình (cơ sở tri thức)
- Cú pháp: listing(predicate)
- ?- listing(at).
  - at(key, cave_entrance).
    at(flashlight, building).
    at(sword, closet).

    true.

Di chuyển theo các hướng

- Biểu diễn việc di chuyển theo 4 hướng tương ứng với 4 lệnh **n**, **s**, **e**, **w**, sử dụng vị từ *go*
- n :- go(n).

  s :- go(s).

  e :- go(e).

  w :- go(w).

Vị từ *go*

- go(Direction) :-
  i_am_at(Here),
  path(Here, Direction, There),
  retract(i_am_at(Here)),
  assert(i_am_at(There)),
  look.

- go(_) :-
  write('You can''t go that way.').

Vị từ take

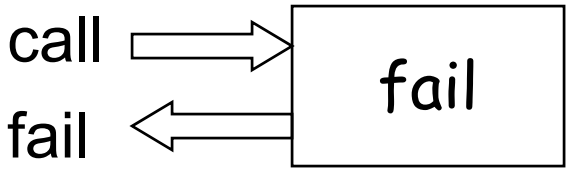
- take(X) :-
  i_am_at(Place),
  at(X, Place),
  retract(at(X, Place)),
  assert(holding(X)),
  write('OK.'),
  nl.

- take(X) :-
  holding(X),
  write('You\'re already holding it!'), nl.

- take(X) :-
  write('I don\'t see it here.'), nl.

Sử dụng từ khóa fail

- Một vị từ thất bại (fail) nếu nó không thành công (không thể chứng minh được)
- Có thể phát biểu rõ ràng một mệnh đề là thất bại, bằng cách sử dụng fail



- Sử dụng fail chỉ đảm bảo là mệnh đề hiện tại của vị từ là thất bại; chứ không bắt buộc toàn bộ vị từ là thất bại

Cắt

- Sử dụng cắt, được ký hiệu là **!** , như là một điểm kết thúc
  - Kết thúc việc đánh giá mệnh đề hiện tại ở vị trí cắt
- Ý nghĩa của cắt: Không xem xét (thêm nữa) bất kỳ mệnh đề nào khác

Sử dụng kết hợp cắt-thất bại

- Sử dụng kết hợp cắt-thất bại, ký hiệu là !, fail , sẽ có nghĩa chỉ định rằng toàn bộ vị từ là thất bại (không thể chứng minh được)
- !, fail kết thúc mệnh đề hiện thời, và chỉ định vị từ thất bại
- Theo sau !, fail sẽ không có bất kỳ mệnh đề nào của vị từ được xem xét nữa, và vì vật toàn bộ vị từ sẽ thất bại

Ví dụ về !, fail

- Biểu diễn cửa của phòng kho bị khóa
- ```
  path(building, e, closet) :-
      locked(closet),
      write('The door appears to be locked.'), nl,
      !, fail.

  path(building, e, closet).
  ```
- Nếu cửa của phòng kho không bị khóa, thì mệnh đề thứ nhất sẽ thất bại "một cách bình thường", và *mệnh đề thứ hai sẽ được xét đến*
- Nếu cửa của phòng kho không bị khóa, thì `!, fail` sẽ ngăn cản Prolog *không xét đến mệnh đề thứ hai*
  ```

Thả các đồ vật

```
drop(A) :-
        holding(A),
        i_am_at(B),
        retract(holding(A)),
        assert(at(A, B)),
        write('OK.'), nl.

drop(A) :-
        write('You aren\'t holding it!'), nl.
```