Giáo trình

TRÍ TUỆ NHÂN TẠO

ARTIFICIAL INTELLIGENCE

Phạm Thọ Hoàn, Phạm Thị Anh Lê

Khoa Công nghệ thông tin

Trường Đại học Sư phạm Hà Nội

Hà nội, 2011

MỤC LỤC

Chương 5 – Các phương pháp lập luận trên logic cấp một 22

Chương 4 – Các phương pháp lập luận trên logic mệnh đề

1. *Lập luận và Logic*

Loài người thông minh vì biết lập luận. Liệu máy tính có khả năng ***lập luận*** được (như con người) không? Để trả lời câu hỏi này, chúng ta trước hết hãy cho biết thế nào là lập luận.

Lập luận là hành động sinh ra một phát biểu đúng mới từ các phát biểu đúng có trước. Hay nói cách khác, một người hoặc một hệ thống được gọi là biết lập luận nếu nó chỉ ra rằng một phát biểu nào đó có đúng (true) khi cho trước một tập các phát biểu đúng hay không? Các phát biểu phải tuân theo một tập các qui tắc nhất định (ngữ pháp) và cách xác định một phát biểu là đúng (true) hay là sai (false). Một tập các qui tắc qui định ngữ pháp và cách xác định ngữ nghĩa đúng/sai của các phát biểu gọi là logic. Như vậy logic là một ngôn ngữ mà mỗi câu trong ngôn ngữ đó có ngữ nghĩa (giá trị) là đúng hoặc sai, và vì vậy có thể cho phép chúng ta lập luận, tức là một câu mới có giá trị đúng không khi cho các câu trước đó là đúng hay không. Các câu cho trước được gọi là cơ sở tri thức (Knowledge base - KB), câu cần chứng minh là đúng khi biết KB đúng gọi là câu truy vấn (query - q). Nếu q là đúng khi KB là đúng thì ta nói rằng KB suy diễn ra q (ký hiệu là KB $\models$ q).

Trong chương này và các chương tiếp theo, chúng ta sẽ xây dựng các thuật giải cho phép lập luận tự động trên các logic khác nhau. Các thuật giải này giúp máy tính có thể lập luận, rút ra phát biểu mới từ các phát biểu cho trước.

2. *Logic mệnh đề: cú pháp, ngữ nghĩa*

Logic đơn giản nhất là logic mệnh đề. Các phát biểu (câu) trong logic mệnh đề được hình thành từ các ký hiệu mệnh đề (mỗi ký hiệu có nghĩa là một mệnh đề và vì vậy có thể nhận giá trị đúng hoặc sai tùy theo mệnh đề đó là đúng hay sai trong thế giới thực) và các ký hiệu liên kết $\neg$ (với ngữ nghĩa là phủ định), $\wedge$ (và), $\vee$ (hoặc), $\Rightarrow$ (kéo theo), $\Leftrightarrow$ (tương đương). Cú pháp và ngữ nghĩa của logic mệnh đề như sau:

2.1 ***Cú pháp:***

- Các ký hiệu:
  - ✓ Hằng: true, false
  - ✓ Ký hiệu: P, Q, … Mỗi ký hiệu gọi là ký hiệu mệnh đề hoặc mệnh đề
  - ✓ Các kết nối logic: $\neg$, $\wedge$, $\vee$
  - ✓ Các ký hiệu "(" và ")"
- Qui tắc xây dựng câu: Có hai loại câu: câu đơn và câu phức
  - ✓ true và false là các câu (true là câu đơn hằng đúng, false là câu hằng sai).
  - ✓ Mỗi ký hiệu mệnh đề là một câu, ví dụ P, Q là các câu (Câu đơn)
  - ✓ Nếu A và B là các câu thì các công thức sau cũng là câu (các câu phức):

    $\neg A$

    $(A \wedge B)$

    $(A \vee B)$

    $(A \Rightarrow B)$

    $(A \Leftrightarrow B)$
- Các khái niệm và qui ước khác: Sau này, để cho gọn, ta bỏ đi các dấu "(", ")" không cần thiết. Nếu câu chỉ có một ký hiệu mệnh đề thì ta gọi câu đó là câu đơn hoặc câu phân tử. Các câu không phải là câu đơn thì gọi là câu phức. Nếu P là ký hiệu mệnh đề thì P và $\neg$P gọi là các literal, P là literal dương còn $\neg$P là literal âm. Các câu phức dạng A1 $\vee$ A2 $\vee \ldots \vee$An, trong đó các Ai là các literal, được gọi là các câu tuyển (clause).

2.2 ***Ngữ nghĩa:*** Qui định cách diễn dịch và cách xác định tính đúng (true) hay sai (false) cho các câu.

- true là câu luôn có giá trị đúng, false là câu luôn có giá trị sai

- Mỗi ký hiệu biểu diễn (ánh xạ với) một phát biểu/mệnh đề trong thế giới thực; ký hiệu mệnh đề có giá trị là đúng (true) nếu phát biểu/mệnh đề đó là đúng, có giá trị là sai (false) nếu phát biểu/mệnh đề đó là sai, hoặc có giá trị chưa xác định (true hoặc false)
- Các câu phức biểu diễn (ánh xạ với) một phủ định, mối quan hệ hoặc mối liên kết giữa các mệnh đề/phát biểu/câu phức trong thế giới thực. Ngữ nghĩa và giá trị của các câu phức này được xác định dựa trên các câu con thành phần của nó, chẳng hạn:
    - ✓ $\neg$A có nghĩa là phủ định mệnh đề/ câu A, nhận giá trị true nếu A là false và ngược lại
    - ✓ A $\wedge$ B có nghĩa là mối liên kết "A và B", nhận giá trị true khi cả A và B là true, và nhận giá trị false trong các trường hợp còn lại.
    - ✓ A $\vee$ B biểu diễn mối liên kết "A hoặc B", nhận giá trị true khi hoặc A hoặc B là true, và nhận giá trị false chỉ khi cả A và B là false.
    - ✓ (A $\Rightarrow$ B) biểu diễn mối quan hệ "A kéo theo B", chỉ nhận giá trị false khi A là true và B là false; nhận giá trị true trong các trường hợp khác
    - ✓ (A $\Leftrightarrow$ B) biểu diễn mối quan hệ "A kéo theo B" và "B kéo theo A"

  Như vậy, việc xác định tính đúng/sai của một ký hiệu mệnh đề (mệnh đề đơn) là dựa trên tính đúng sai của sự kiện hoặc thông tin mà nó ám chỉ, còn việc xác định tính đúng sai của mệnh đề phức phải tuân theo các qui tắc trên. Trong nhiều trường hợp, chúng ta (cần chỉ) biết tính đúng/sai của các câu phức, còn tính đúng/sai của các câu đơn là không cần biết hoặc có thể lập luận ra từ các các câu phức đã biết đúng/sai và các qui tắc chuyển đổi tính đúng/sai giữa các câu đơn và câu phức theo các qui tắc trên.

2.3 Các ví dụ:

Gọi A là mệnh đề "tôi chăm học", B là mệnh đề "tôi thông minh", C là mệnh đề "tôi thi đạt điểm cao môn Trí tuệ nhân tao"; Ta có thể biểu diễn các câu sau trong logic mệnh đề:

- "Nếu tôi chăm học thì tôi thi đạt điểm cao môn Trí tuệ nhân tạo": $A \Rightarrow C$
- "Tôi vừa chăm học lại vừa thông minh": $A \wedge B$
- "Nếu tôi chăm học hoặc tôi thông minh thì tôi thi đạt điểm cao môn Trí tuệ nhân tạo": $A \vee B \Rightarrow C$

2.4 Các câu hằng đúng:

Trong logic mệnh đề, ta có:

- ✓ $\neg\neg A \Leftrightarrow A$ (luật phủ định kép)
- ✓ $A \vee \neg A$ (luật loại trừ)
- ✓ $(A \Leftrightarrow B) \Leftrightarrow (A \Rightarrow B) \wedge (B \Rightarrow A)$
- ✓ $(A \Rightarrow B) \Leftrightarrow \neg A \vee B$
- ✓ $\neg (A \vee B) \Leftrightarrow \neg A \wedge \neg B$ (luật DeMorgan đối với phép $\vee$)
- ✓ $\neg (A \wedge B) \Leftrightarrow \neg A \vee \neg B$ (luật DeMorgan đối với phép $\wedge$)
- ✓ $C \vee (A \wedge B) \Leftrightarrow (C \vee A) \wedge (C \vee B)$ (luật phân phối phép $\vee$ đối với phép $\wedge$)
- ✓ $C \wedge (A \vee B) \Leftrightarrow (C \wedge A) \vee (C \wedge B)$ (luật phân phối phép $\wedge$ đối với phép $\vee$)
- ✓ $(A \wedge (A \Rightarrow B)) \Rightarrow B$ (Tam đoạn luận)
- ✓ Luật phân giải (xem mục 4)

3. Bài toán lập luận và các giải thuật lập luận trên logic mệnh đề

Như đã nói trong phần 1 của Chương này, lập luận là trả lời câu hỏi một câu *q* có là đúng khi cho cơ sở tri thức (là một câu phức là hội của tập các câu cho trước) là đúng hay không ($KB \models q$)? Một cách đơn giản nhất là chúng ta lập bảng giá trị chân lý cho

KB và cho *q* và kiểm tra xem tất cả các trường hợp làm cho KB nhận giá trị true cũng làm cho q nhận giá trị true không? Nếu có thì ta kết luận $KB \models q$, ngược lại thì kết luận là không. Phương pháp suy luận này gọi là phương pháp liệt kê và có thể thuật toán hóa được (chi tiết xem trong mục 6 của Chương này).

Một cách tiếp cận khác để trả lời cho câu hỏi $KB \models q$ là sử dụng các luật hằng đúng của logic mệnh đề (xem trong mục 2.4). Ban đầu KB bao gồm tập các câu (hội của các câu), chúng ta áp dụng các luật của logic mệnh đề trên tập các câu này để sinh ra câu mới, rồi bổ sung câu mới này vào KB, lặp lại áp dụng luật của logic và sinh ra câu mới, v.v., đến khi nào xuất hiện câu q trong KB thì dừng lại (khi đó $KB \models q$) hoặc không thể sinh ra câu mới nào nữa từ KB (khi này ta kết luận *KB* không suy ra được *q*) Lời giải cho bài toán suy diễn theo cách này là một đường đi từ trạng thái đầu đến trạng thái đích của bài toán tìm đường sau:

- ✓ *Trạng thái đầu*: KB
- ✓ *Các phép chuyển trạng thái*: các luật trong logic mệnh đề, mỗi luật *x* áp dụng cho *KB* sinh ra câu mới *x(KB)*, bổ sung câu mới này vào *KB* được trạng thái mới $KB \wedge x(KB)$
- ✓ *Trạng thái đích*: trạng thái *KB* chứa *q*
- ✓ *Chi phí cho mỗi phép chuyển*: 1

Vì số luật hằng đúng trong logic mệnh là tương đối lớn nên nhân tố nhánh của bài toán trên cũng là lớn (tất cả các cách áp dụng các luật trên tập con tất cả các câu của KB), vì vậy không gian tìm kiếm lời giải của bài toán trên là rất lớn. Để hạn chế không gian tìm kiếm lời giải của bài toán, chúng ta biểu diễn KB và q bằng chỉ các câu dạng chuẩn hội (xem mục 4), khi đó chúng ta chỉ cần áp dụng một loại luật là luật phân giải trên KB và mỗi phép chuyển là một phép phân giải hai câu có chứa ít nhất một literal là phủ định của nhau trong KB, kết quả của phép phân giải hai câu dạng chuẩn hội lại là một câu dạng chuẩn hội và được bổ sung vào KB, lặp lại áp dụng luật phân giải trên KB đến khi nào KB chứa câu *q* thì dừng. Chi tiết

thuật toán suy diễn dựa trên luật phân giải $KB \models q$ được trình bày trong mục 7 của Chương này (thực tế thì thuật toán suy diễn phân giải trả lời bài toán tương đương $(KB \wedge \neg q) \models []$.)

Giải thuật suy diễn phân giải là giải thuật đầy đủ trong logic mệnh đề, tức là với mọi câu q mà kéo theo được từ KB (q đúng khi KB đúng) thì sử dụng giải thuật suy diễn phân giải đều có thể suy diễn được KB $\models q$ (tức là không có câu nào kéo được từ KB là không suy diễn phân giải được); bởi vì bất cứ câu trong logic mệnh đề đều có thể biểu diễn được bằng câu dạng chuẩn hội (xem mục 4).

Do liên tục phải bổ sung các câu mới vào KB và lặp lại tìm kiếm các cặp câu có thể phân giải với nhau được nên nhân tố nhánh của cây tìm kiếm lời giải tăng dần theo độ sâu của cây tìm kiếm. Vì vậy không gian và thời gian của giải thuật sẽ tăng rất nhanh, giải thuật phân giải làm việc không hiệu quả. Để khắc phục nhược điểm này, người ta tìm cách biểu diễn KB dạng các câu Horn và áp dụng chỉ một loại luật (tam đoạn luận, xem mục 5) để suy diễn (tam đoạn luận áp dụng trên 2 câu dạng Horn và sinh ra câu mới cũng là câu dạng Horn). Thuật giải suy diễn tiến/lùi trên cơ sở tri thức dạng Horn trình bày chi tiết trong mục 8, nó có độ phức tạp tuyến tính đối với số câu trong KB. Tuy nhiên thuật giải suy diễn tiến/lùi lại là không đầy đủ trong logic mệnh đề, bởi vì có những câu trong logic mệnh đề không thể biểu diễn được dưới dạng Horn để có thể áp dụng được giải thuật suy diến tiến/lùi.

4. *Câu dạng chuẩn hội và luật phân giải*

- Câu dạng chuẩn hội là câu hội của các câu tuyển (clause). Như trên đã nói, câu tuyển là câu dạng $A1 \vee A2 \vee \ldots \vee An$, trong đó các Ai là các ký hiệu mệnh đề hoặc phủ định của ký hiệu mệnh đề. Vậy câu dạng chuẩn hội có dạng:

$$(\underbrace{A_{11} \vee A_{12} \vee \ldots \vee A_{1n}}_{\text{clause}}) \wedge (\underbrace{A_{21} \vee A_{22} \vee \ldots \vee A_{2m}}_{\text{clause}}) \wedge \ldots \wedge (\underbrace{A_{k1} \vee A_{k2} \vee \ldots \vee A_{kr}}_{\text{clause}})$$

Với A_{ij} là các literal (là ký hiệu mệnh đề hoặc phủ định của ký hiệu mệnh đề).

- Với một câu bất kỳ trong logic mệnh đề, liệu có thể biểu diễn dưới dạng chuẩn hội như trên được không? Câu trả lời là có. Với câu *s*, chúng ta liệt kê tất cả các ký hiệu mệnh đề xuất hiện trong nó, lập bảng giá trị chân lý để đánh giá *s*, khi đó s là hội các tuyển mà mỗi tuyển sẽ tương ứng với dòng làm cho s bằng ~~true~~ false. Với mỗi tuyển (tương ứng với một dòng), nếu cột của ký hiệu mệnh đề trên dòng đó có giá trị true thì ký hiệu mệnh đề sẽ là literal ~~dương~~ âm, còn nếu giá trị là false thì ký hiệu mệnh đề sẽ là literal ~~âm~~ dương trong câu tuyển. Ví dụ, chúng ta muốn biết dạng chuẩn hội của câu sau:

$$\neg C \Rightarrow A \wedge B$$

Trong câu trên, có 3 ký hiệu mệnh đề là A, B, C. Ta lập bảng giá trị chân lý và chuyển sang dạng chuẩn hội như bảng sau:

A	**B**	**C**	**$\neg C \Rightarrow A \wedge B$**	**Clause**	Dạng chuẩn hội: $\neg C \Rightarrow A \wedge B$ $= (A \vee B \vee C) \wedge (A \vee \neg B \vee C) \wedge (\neg A \vee B \vee C)$
F	F	F	F	$A \vee B \vee C$	
F	F	T	T		
F	T	F	F	$A \vee \neg B \vee C$	
F	T	T	T		
T	F	F	F	$\neg A \vee B \vee C$	
T	F	T	T		
T	T	F	T		
T	T	T	T		

- Với cách chuyển một câu sang dạng chuẩn hội như dung bảng giá trị chân lý ở trên, chúng ta khẳng định bất kỳ câu nào cũng có thể chuyển sang dạng chuẩn hội được. Ngoài phương pháp sử dụng bảng chân lý, chúng ta có thể áp dụng 4 qui tắc

sau đây (theo thứ tự được liệt kê) để chuyển bất kỳ câu nào sang dạng chuẩn hội được.

- ✓ QT1: Loại bỏ $\Leftrightarrow$: thay thế $\alpha \Leftrightarrow \beta$ bằng $(\alpha \Rightarrow \beta) \wedge (\beta \Rightarrow \alpha)$.
- ✓ QT2: Loại bỏ $\Rightarrow$: Thay thế $\alpha \Rightarrow \beta$ bằng $\neg\alpha \vee \beta$
- ✓ QT3: chuyển hoặc loại bỏ dấu $\neg$ đặt trước các ký hiệu bằng các luật deMorgan và luật phủ định kép $\neg(\alpha \vee \beta) = \neg\alpha \wedge \neg\beta$; $\neg(\alpha \wedge \beta) = \neg\alpha \vee \neg\beta$; $\neg\neg\alpha = \alpha$.
- ✓ QT4: Áp dụng luật phân phối của phép $\wedge$ đối với phép $\vee$

Chẳng hạn, chúng ta cần chuyển câu trong ví dụ trên sang dạng chuẩn hội, bằng cách áp dụng lần lượt các qui tắc trên:

$\neg C \Rightarrow A \wedge B$

$= \neg(\neg C) \vee (A \wedge B)$ (QT2)

$= C \vee (A \wedge B)$ (QT3)

$= (C \vee A) \wedge (C \vee B)$ (QT4)

Chúng ta có thể dừng lại dạng chuẩn hội này, hoặc cũng có thể chứng minh tiếp rằng công thức này và công thức thu được từ phương pháp lập bảng ở trên là tương đương.

- Luật phân giải (resolution):
  - ✓ Luật phân giải:

    Nếu chúng ta có hai clause sau là đúng:

    $(P_1 \vee P_2 \vee \ldots P_i \vee \ldots \vee P_n) \wedge$

    $(Q_1 \vee Q_2 \vee \ldots Q_j \vee \ldots \vee Q_m)$

    và P_i, Q_j là các literal phủ định của nhau $(P_i = \neg Q_j)$

    thì chúng ta cũng có clause sau là đúng

$$(P_1 \vee P_2 \vee \dots P_{i-1} \vee P_{i+1} \vee \dots \vee P_n \vee Q_1 \vee Q_2 \vee \dots Q_{j-1} \vee Q_{j+1} \vee \dots \vee Q_m)$$

(Clause mới là tuyển các literal trong hai clause ban đầu nhưng bỏ đi P_i và Q_j)

- ✓ Kết quả của phép phân giải cũng là một clause (tuyển các literal), hay nói cách khác phép phân giải có tính đóng, phân giải của các clause là một clause. Đây là tính chất rất quan trong trong việc xây dựng giải thuật suy diễn tự động trình bày phía dưới.

- Câu dạng chuẩn tuyển (tham khảo thêm): Câu dạng chuẩn tuyển là câu tuyển của các hội. Giống như cấu trúc của câu dạng chuẩn hội, câu dạng chuẩn tuyển cũng có cấu trúc như vậy, nhưng chúng ta đổi chỗ dấu $\vee$ bởi dấu $\wedge$ và ngược lại. Với bất kỳ một câu trong logic mệnh đề, chúng ta cũng có thể biểu diễn nó dưới dạng chuẩn tuyển. Tuy nhiên chúng ta không có luật đóng liên quan đến tuyển của hai câu hội để sinh ra câu hội mới như luật phân giải của hai câu tuyển.

5. *Câu dạng Horn và tam đoạn luận*

- Câu dạng Horn: Như trên ta đã chỉ ra rằng tất cả các câu trong logic mệnh đề đều có thể biểu diễn được dưới dạng chuẩn hội, tức là hội của các clause, mỗi clause có dạng: $P_1 \vee P_2 \vee \dots P_i \vee \dots \vee P_n$, với P_i là các literal. Nếu trong clause mà có nhiều nhất một literal dương (tức là không có ký hiệu phủ định đằng trước) thì clause đó gọi là câu dạng Horn. Như vậy câu dạng Horn là câu có một trong ba dạng:

  $\neg P_1 \vee \neg P_2 \vee \dots \vee \neg P_n$ (không có literal dương nào)

  hoặc P (có một literal dương và không có literal âm nào)

  hoặc $\neg P_1 \vee \neg P_2 \vee \dots \vee \neg P_n \vee Q$ (có một literal dương là Q và ít nhất một literal âm)

  với $P_1, P_2, \dots, P_n$ và Q là các ký hiệu mệnh đề.

  Nếu chuyển các câu dạng Horn sang dạng luật thì chúng có dạng như sau:

  $$\neg(P_1 \wedge P_2 \wedge \dots \wedge P_n)$$

hoặc P

hoặc $P_1 \wedge P_2 \wedge \ldots \wedge P_n \Rightarrow Q$ (có một literal dương là Q)

Trong đó câu dạng thứ hai và câu ba gọi là câu Horn dương (có đúng 1 literal dương) thường được sử dụng biểu diễn tri thức trong cơ sở tri thức KB, câu dạng thứ nhất chỉ xuất hiện trong biểu diễn các câu truy vấn.

- Tam đoạn luận (hay luật Modus ponens):

Nếu chúng ta có các câu Horn dương sau là đúng:

P_1,

P_2,

…

P_n và

$P_1 \wedge P_2 \wedge \ldots \wedge P_n \Rightarrow Q$

thì câu Q là đúng

- Kết quả luật Modus ponens từ hai câu dạng Horn dương sinh ra câu Q cũng có dạng Horn dương. Vì vậy phép suy diễn tam đoạn luận là đóng trong các câu dạng Horn, kết quả tam đoạn luận từ hai câu dạng Horn là câu dạng Horn. Tương tự như tính chất đóng của phép phân giải trong trong các câu dạng chuẩn hội, tính chất đóng của phép suy luận này là rất quan trọng trong việc thiết kế các giải thuật suy diễn tự động đề dựa trên tam đoạn luận và các câu Horn (xem phần phía dưới).
- Không giống như câu dạng chuẩn hội, không phải câu nào trong logic mệnh đề đều có thể biểu diễn dạng Horn được. Chính vì thế mà thuật giải suy diễn dựa trên tam đoạn luận chỉ là đầy đủ trong ngôn ngữ các câu Horn chứ không đầy đủ trong logic mệnh đề.

6. Thuật toán suy diễn dựa trên bảng giá trị chân lý

Trong các phần còn lại của Chương này, chúng ta sẽ xây dựng các giải thuật cài đặt cho máy tính để nó biết lập luận. Giải thuật lập luận tự động là giải thuật chỉ ra rằng nếu KB (cơ sở tri thức) là đúng thì câu truy vấn q có đúng hay không?

Phương pháp lập luận đầu tiên là dựa liệt kê các tất cả các trường hợp có thể có của tập các ký hiệu mệnh đề, rồi kiểm tra xem liệu tất cả các trường hợp làm cho KB đúng xem q có đúng không. Chi tiết thuật giải như bảng sau:

```
Function Suydien_Lietke(KB, q) return true or false
    symbols=get_list_of_symbols(KB,q);
    n= symbols.size();
    int bộ_giá_trị[n]; //dùng để lưu bộ các giá trị logic (true:1, false:0)
    for (i=1; i≤2^n; i++) {
        bộ_giá_trị [1,..,n]=generate(i); // sinh ra bộ thứ i
        if (evaluate(KB, bộ_giá_trị)==true && evaluate(q, bộ_giá_trị)=false)
            return false
    return true;
```

Thuật giải trên là sinh ra toàn bộ bảng giá trị chân lý để đánh giá KB và q, nếu chỉ cần một trường hợp KB đúng mà q sai thì q sẽ kết luận KB không suy diễn được ra q.

Giải thuật trên có độ phức tạp thời gian là $2^n * m$, với n là số ký hiệu có trong KB,q và m độ dài câu trong KB.

7. *Thuật toán suy diễn dựa trên luật phân giải*

Để khắc phục nhược điểm độ phức tạp thời gian của giải thuật suy diễn dựa trên liệt kê ở trên, chúng ta đưa ra thuật giải nhanh hơn, thời gian thực hiện nhanh hơn.

Giải thuật dựa trên thực hiện liên tiếp các luật phân giải trên các câu dạng chuẩn hội. Để chứng minh $KB \models q$ ta sẽ chứng minh điều tương đương là ($KB \wedge \neg q \models []$), tức là như chúng ta vẫn gọi là chứng minh bằng phản chứng: giả sử q không đúng ($\neg q$), khi đó $KB \wedge \neg$q sẽ dẫn đến mâu thuẫn, tức là ($KB \wedge \neg q) \models []$.

Chúng ta sẽ chuyển ($KB \wedge \neg q$) về dạng chuẩn hội, tức là hội các clause, hay chúng ta chuyển KB và ¬q thành hội các clause, sau đó áp dụng liên tiếp luật phân giải (mục 4) trên các cặp clause mà có ít nhất một literal đối của nhau để sinh ra một clause mới, clause mới này lại bổ sung vào danh sách các clause đã có rồi lặp lại áp dụng luật phân giải. Giải thuật dừng khi có câu [] được sinh ra (khi đó ta kết luận $KB \models q$) hoặc không có clause nào được sinh ra (khi đó ta kết luận KB không suy diễn được ra q). Chi tiết thuật giải cho trong hình ở trang sau.

Giải thuật phân giải là giải thuật đầy đủ vì tất cả các câu trong logic mệnh đề đều có thể biểu diễn được dưới dạng hội của các clauses (dạng chuẩn hội). Tuy nhiên mỗi lần phân giải sinh ra clause mới thì lại bổ sung vào danh sách các clauses để thực hiện tìm kiếm các cặp clauses phân giải được với nhau; vì vậy số lượng clauses ở lần lặp sau lại tăng lên so với lần lặp trước, dẫn đến việc tìm kiếm các clauses phân giải được với nhau là khó khăn hơn.

Giải thuật phân giải trình bày như trên là giải thuật suy phân giải tiến, có nghĩa là từ trạng thái đầu $KB \wedge \neg q$ thực hiện các thao tác chuyển trạng thái (áp dụng luật phân giải trên cặp các clauses để sinh ra clauses mới và bổ sung vào danh sách các clauses hiện có) để sinh ra trạng thái mới, đến khi nào trạng thái mới chứa câu [] (trạng thái đích) thì dừng hoặc không sinh ra trạng thái mới được nữa.

Một cách khác để thực hiện suy diễn phân giải $KB \models q$ là xuất phát từ clause $\neg q$ (coi như trạng thái đích) ta thực hiện phân giải với các clauses khác trong KB để sinh ra clauses mới, rồi từ các clauses mới này thực hiện tiếp với các clauses khác của KB để sinh ra clauses mới hơn, đến khi nào [] được sinh ra hoặc không sinh ra được clause mới thì dừng. Nói cách khác là chỉ thực hiện phân giải các clauses liên quan đến q.

Giải thuật phân giải lùi sẽ làm việc hiệu quả hơn giải thuật phân giải tiến (chi tiết cài đặt coi như là bài tập).

```
Function Resolution(KB, q) return true or false
    clauses=get_list_of_clauses(KB ∧ ¬q);
    new={};
    do
      for each Ci, Cj in clauses
                new_clause= resol(Ci,Cj);
                if new_clause=[] return true;
                new=new ∪ new_clause;
        if new⊆clauses return false;
        clauses=clauses ∪ new;
```

8. *Thuật toán suy diễn tiến, lùi dựa trên các câu Horn*

Như ta đã thấy trong mục 5, luật Modus ponens là đóng trong các câu dạng Horn dương, có nghĩa là nếu hai câu dạng Horn dương thỏa mãn các điều kiện của luật Modus ponens thì sẽ sinh ra câu dạng Horn dương mới. Nếu chúng ta biểu diễn được KB và q bằng các câu dạng Horn dương thì có thể sử dụng luật Modus ponens để suy diễn.

Khi KB biểu diễn bằng hội các câu Horn dương, chúng ta các câu Horn dương này thành 2 loại: (1) câu có đúng một literal dương mà không có literal âm nào, đây là các câu đơn hay là các ký hiệu mệnh đề; (2) câu có đúng một literal dương và có ít nhất một literal âm, đây là các câu kéo theo mà phần thân của phép kéo theo chỉ là một ký hiệu mệnh đề.

Có hai cách cài đặt thuật giải suy diễn dựa trên luật Modus ponens trên các câu Horn dương. Cách thứ nhất là bắt đầu từ các ký hiệu mệnh đề được cho là đúng trong KB,

áp dụng liên tiếp các luật Modus ponens trên các câu kéo theo trong KB để suy diễn ra các ký hiệu mới, đến khi nào danh sách các hiệu được suy diễn ra chứa ký hiệu đích q thì dừng và thông báo suy diễn thành công. Nếu danh sách các ký hiệu suy diễn không chứa q và cũng không thể sinh tiếp được nữa thì thông báo suy diễn thất bại. Cách suy diễn này gọi là suy diễn tiến (hay suy diễn tam đoạn luận tiến để phân biệt với suy diễn phân giải tiến ở trên).

Chi tiết giải thuật cho trong bảng ở phía dưới. Giải thuật sử dụng danh sách các ký hiệu mệnh đề được xác định là true, true_symbols , danh sách này khởi tạo từ các ký hiệu độc lập trong KB, sau đó bổ sung khi một ký hiệu mệnh đề được suy diễn ra là true, đến khi nào danh sách chưa ký hiệu truy vấn q thì dừng hoặc không bổ sung được ký hiệu nào nữa vào danh sách này.

Cách cài đặt thứ hai là xuất phát từ đích q, chúng ta xem có bao nhiêu câu Horn kéo theo nào trong KB có q là phần đầu của luật kéo theo, chúng ta lại kiểm tra xem các ký hiệu mệnh đề nằm trong phần điều kiện của các luật này (các đích trung gian) xem có suy diễn được từ KB không, cứ áp dụng ngược các luật đến khi nào các đích trung gian được xác nhận là đúng trong KB thì kết luận suy diễn thành công, hoặc kết luận không thành công khi có tất cả các nhánh đều không chứng minh được các đích trung gian không suy diễn được từ KB. Giải thuật này gọi là giải thật suy diễn lùi (hoặc là giải thuật suy diễn tam đoạn luận lùi).

Function Forward_Horn(KB, q) *return* true or false

Input: - KB tập các câu Horn dương, đánh số $clause_1$, .., $clause_n$

- q: câu truy vấn dạng câu đơn (ký hiệu mệnh đề)

Output: true or false

Các biến địa phương:

- Int count[0.. n], count[i] là số ký hiệu xuất hiện trong phần điều kiện của $clause_i$.
- Bool proved[danhsach_kyhieu]: proved[kyhieu]=1 nếu kyhieu đã được chứng minh là suy diễn được từ KB, ngược lại =0; ban đầu khởi tạo=0 với mọi ký hiệu
- working_symbols: danh sách ký hiệu đang xem xét, khởi đầu bằng danh sách các ký hiệu độc lập trong KB

while working_symbols *is not* empty

p= pop(working_symbols);

if (!proved[p])

proved[p]=1;

for each $clause_i$ whose p appears

count[$clause_i$] = count[$clause_i$] -1;

if count[$clause_i$]==0

if head[$clause_i$]==q *return* true;

push (head[$clause_i$], working_symbols);

return false;

9. *Kết chương*

Logic mệnh đề là ngôn ngữ để biểu diễn các mệnh đề. Có hai loại mệnh đề: mệnh đề đơn và mệnh đề phức. Mệnh đề đơn tương ứng với một phát biểu nào đó (một sự kiện hoặc thông tin) và có thể phán xét xem nó đúng hay sai dựa trên phát biểu đó là đúng hay sai. Mệnh đề phức biểu diễn mối quan hệ hoặc mối liên kết (phủ định, hội, tuyển, kéo theo, tương đương) giữa các mệnh đề con của nó. Logic qui định tính đúng hay sai của mệnh đề phức dựa trên tính đúng/sai của các mệnh đề con và dựa trên kiểu của mối quan hệ/liên kết đó (là $\neg$, $\wedge$, $\vee$, $\Rightarrow$, hay là $\Leftrightarrow$). Chính vì việc gán cho các câu (mệnh đề đơn hoặc mệnh đề phức) hoặc giá trị đúng (true) hoặc giá trị sai (false) theo các qui tắc của logic giúp chúng ta phán xét được rằng một mệnh đề này là đúng khi cho biết tập các mệnh đề cho trước là đúng, hay là $KB \models q$. Lập luận là trả lời cho câu hỏi: cho *KB* đúng thì *q* có đúng không?.

Trong Chương này chúng ta đã tìm hiểu một số thuật giải lập luận (input là KB và q, output là true hoặc false). Các giải thuật lập luận gồm: lập luận bằng liệt kê, lập luận dựa trên luật phân giải, lập luận dựa trên luật Modus ponens. Giải thuật lập luận bằng liệt kê các giá trị chân lý của các ký hiệu mệnh đề xuất hiện trong KB và q có ưu điểm là không đòi hỏi dạng cấu trúc đặc biệt nào cho các câu KB và q, nhưng lại có độ phức tạp thời gian là hàm mũ đối với số các ký hiệu mệnh đề. Giải thuật dựa trên luật phân giải thì yêu cầu KB và $\neg$q phải có dạng chuẩn hội, tức là chúng ta phải thực hiện chuyển KB và $\neg$q thành dạng chuẩn hội rồi mới áp dụng giải thuật. May thay, tất cả các câu trong logic mệnh đề đều có thể chuyển được về dạng chuẩn hội. Còn giải thuật lập luận dựa trên luật Modus ponens thì yêu cầu KB và q phải có dạng câu Horn. Không phải tất cả các câu trong logic mệnh đề đều chuyển về dạng Horn được. Tuy nhiên nếu KB và q ở dạng Horn thì các giải thuật suy diễn tiến hoặc lùi dựa trên Modus ponens lại làm việc rất hiệu quả.

Các giải thuật lập luận ở trên khi cài đặt cho máy tính sẽ giúp máy tính có khả năng lập luận được.

Chương 5 – Các phương pháp lập luận trên logic cấp một

Trong Chương trước chúng ta đã tìm hiểu logic mệnh đề, một ngôn ngữ đưa ra các qui tắc xác định ngữ pháp và ngữ nghĩa (tính đúng/sai) các câu. Câu đơn giản nhất trong logic mệnh đề là các ký hiệu mệnh đề, nó biểu diễn cho các sự kiện hoặc thông tin trong thế giới thực. Câu phức tạp hơn liên kết các câu đơn bằng các phép nối logic ($\neg$, $\wedge$, $\vee$, $\Rightarrow$, $\Leftrightarrow$) biểu diễn mệnh đề phức, mô tả quan hệ hoặc liên kết các mệnh đề đơn. Như vậy, logic mệnh đề chỉ có thể biểu diễn được các MỆNH ĐỀ và các liên kết hoặc quan hệ giữa các MỆNH ĐỀ. Vì vậy sức mạnh biểu diễn của logic mệnh đề chỉ giới hạn trong thế giới các mệnh đề. Nó không quan tâm đến nội dung các mệnh đề như thế nào. Vì thế mà logic mệnh đề có những hạn chế trong việc biểu diễn và suy diễn. Ví dụ, nếu chúng ta cho cơ sở tri thức phát biểu trong ngôn ngữ tự nhiên như sau:

An là sinh viên.

Mọi sinh viên đều học giỏi.

Với cơ sở tri thức như vậy ta có thể suy diễn ra rằng "An học giỏi". Tuy nhiên nếu sử dụng logic mệnh đề thì câu "An là sinh viên" có thể biểu diễn bằng một ký hiệu mệnh đề P1; còn câu "Mọi sinh viên đều học giỏi" thì thông thường biểu diễn bằng một ký hiệu mệnh đề, chẳng hạn Q. Mệnh đề mà chúng ta cần suy diễn "An học giỏi" ký hiệu bởi T1. Khi đó cơ sở tri thức có dạng:

P1

Q

và mệnh đề cần truy vấn là T1. Vì logic mệnh đề không quan tâm đến nội dung bên trong các mệnh đề nên chúng ta không thể thực hiện suy diễn $\{P1 \wedge Q\} \models T1$ được vì chúng chẳng liên quan gì với nhau. Nếu chúng ta biết được danh sách tất cả các sinh viên, chẳng hạn {An, Bình, …, Yến} thì chúng ta có thể chuyển câu "Mọi sinh viên đều học giỏi" thành câu phức "[An là sinh viên thì An học giỏi] VÀ [Bình là sinh viên thì Bình học

giỏi] VÀ …VÀ [Yến là sinh viên thì Yến học giỏi]" thì câu đó sẽ biểu diễn được thành câu phức trong logic mệnh đề dạng:

$$(P1 \Rightarrow T1) \wedge (P2 \Rightarrow T2) \wedge \ldots \wedge (Pn \Rightarrow Tn)$$

Với P1,T1 là ký hiệu mệnh đề đã nói ở trên; P2 là mệnh đề "Bình là sinh viên", T2 là "Bình học giỏi", …, Pn là "Yến là sinh viên" và Tn là "Yến học giỏi".

Khi đó, sử dụng mệnh đề P1 đã biết là đúng thì ta áp dụng luật Modus ponens trong logic mệnh đề thì suy diễn ra được T1.

Với cách biểu diễn câu "Mọi sinh viên đều học giỏi" bằng $(P1 \Rightarrow T1) \wedge (P2 \Rightarrow T2) \wedge \ldots \wedge (Pn \Rightarrow Tn)$ trong logic mệnh đề ta có thể "Modus ponens" với câu trước đó là P1 để sinh ra T1. Tuy nhiên khi đó số câu có trong cơ sở tri thức sẽ là rất lớn (có bao nhiêu sinh viên thì có bấy nhiêu câu $Pi \Rightarrow Ti$), và khi đó các thuật toán suy diễn tự động sẽ trở nên không hiệu quả. Và quan trọng hơn câu có tính chất phổ biến "Mọi sinh viên đều học giỏi" không thể nào biểu diễn thành dạng liệt kê cho từng sinh viên được. Logic mệnh đề thiếu các câu mô tả đặc trưng cho một lớp các đối tượng (cũng giống như nếu một ngôn ngữ lập trình mà thiếu các câu lệnh lặp như for, while mà chỉ có các kiểu lệnh đơn lẻ và rẽ nhánh), vì thế mà sức mạnh biểu diễn của nó rất hạn chế.

Trong chương này, chúng ta sẽ xem xét logic cấp một, hay logic vị từ, một mở rộng của logic mệnh đề mà cho phép biểu diễn những mệnh đề mang tính phổ quát ("với mọi") và những mệnh đề mang tính đặc thù ("tồn tại") một cách dễ dàng. Để làm được điều đó, chúng ta phân tích mệnh đề thành dạng (chủ ngữ - vị từ) hoặc (chủ ngữ - vị từ - tân ngữ) và chuyển chủ ngữ và tân ngữ thành đối tượng (hoặc biến) của vị từ. Vì vậy mà câu đơn của logic cấp một có dạng Vị_từ(chủ_ngữ) hoặc Vị_từ(chủ_ngữ, tân ngữ); chẳng hạn "An là sinh viên" biểu diễn là Sinhvien(An); "An yêu Bình" biểu diễn là Yeu(An,Binh). Chính vì thế mà ta gọi nó là logic vị từ. Từ các câu đơn như vậy ta xây dựng các câu phức sử dụng các ký hiệu ($\neg, \wedge, \vee, \Rightarrow, \Leftrightarrow$) và $\forall, \exists$ (hai ký hiệu này không có trong logic mệnh đề). Quan trọng hơn, làm thế nào chúng ta xây dựng các thuật giải lập luận tự động, giải thuật cài đặt cho máy tính để nó có thể chứng minh được $KB \models q$, với KB và q là các

câu trong logic vị từ cấp một, tương tự như các giải thuật phân giải, giải thuật suy diễn tiến, lùi trong logic mệnh đề.

1. Cú pháp – ngữ nghĩa

1.1 Cú pháp

- Các ký hiệu:
  - ✓ Ký hiệu hằng:
    - Hằng của ngôn ngữ: true, false
    - Hằng do người sử dụng đặt cho tên đối tượng cụ thể: An, Binh,..., a,b,c, … (đối tượng là các chủ ngữ hoặc tân ngữ trong mệnh đề).
  - ✓ Ký hiệu biến (thường là biến đối tượng, đại diện cho chủ ngữ hoặc tân ngữ): x,y,z,t,u, …
  - ✓ Ký hiệu vị từ: P, Q, … hoặc Sinhvien, Yeu, father, …(mỗi ký hiệu tương ứng vị từ trong mệnh đề). Mỗi ký hiệu vị từ là câu đơn trong logic cấp một và có ngữ nghĩa true hay false.
  - ✓ Ký hiệu hàm: sin, cos, log, father, … Chú ý hàm father (father(An)=Binh) khác với vị từ father (father(An,Binh)) ở chỗ hàm thì trả về giá trị còn vị từ thì trả về true/false. Việc xác định một cái tên là hàm hay vị từ tùy vào sự xuất hiện của nó trong câu và các tham số của nó.
  - ✓ Ký hiệu kết nối logic: $\neg, \wedge, \vee, \Rightarrow, \Leftrightarrow$
  - ✓ Ký hiệu lượng tử: $\forall, \exists$
  - ✓ Các ký hiệu “(“ và ”)” ,”,”
- Qui tắc xây dựng câu: Có 2 loại câu: câu đơn và câu phức. Chúng được định nghĩa đệ qui như sau:
  - ✓ Câu đơn: true và false là các câu (true là câu đơn hằng đúng, false là câu hằng sai).

- ✓ Câu đơn: ***Ký_hiệu_vị_từ(hạng_thức_1, hạng_thức_2, …, hạng_thức_k) là một câu (câu đơn)***, trong đó *hạng_thức_i* là biểu thức của các đối tượng, cú pháp của *hạng thức* được xây dựng từ các ký hiệu hằng, biến và hàm như sau:
  - Các ký hiệu hằng và các ký hiệu biến là một hạng thức
  - Nếu $t_1, t_2, ..,t_n$ là các hạng thức và f là một ký hiệu hàm gồm n tham số thì $f(t_1, t_2, ..,t_n)$ cũng là một hạng thức

  Ví dụ về các câu đơn là:

  love(An,Binh)

  father(An,Nhan)

  sinhvien(Hoa)
- ✓ Câu phức: Nếu A, B là các câu và x là một ký hiệu biến thì các công thức sau cũng là câu:

  $\neg A$

  $(A \wedge B)$

  $(A \vee B)$

  $(A \Rightarrow B)$

  $(A \Leftrightarrow B)$

  $\forall x, A$

  $\exists x, A$

- ➢ Các khái niệm và qui ước khác:
  - ✓ Nếu một hạng thức không chứa biến thì gọi là hạng thức nền
  - ✓ Một câu đơn cũng có tên gọi là câu phân tử hay công thức phân tử

- ✓ Một câu đơn hoặc phủ định của một câu đơn thì gọi là literal
- ✓ Trong công thức có ký hiệu lượng tử ($\forall x$, A hoặc $\exists x$, A) các biến x trong A gọi là biến buộc (biến lượng tử), biến nào trong A không phải là biến lượng tử thì gọi là biến tự do. Các câu mà không có biến tự do gọi là câu đóng. Trong môn học này, chúng ta chỉ quan tâm đến các câu đóng (chỉ các câu đóng mới xác định được tính đúng/sai của nó, xem phần ngữ nghĩa bên dưới)
- ✓ Miền giá trị của một biến là tập hợp các giá trị/đối tượng mà biến đó có thể nhận.

1.2 **Ngữ nghĩa** (qui định cách diễn dịch và xác định tính đúng/sai cho các câu)

- ✓ Một câu đơn đóng (không chứa biến) là tương ứng với một mệnh đề (phát biểu, sự kiện, thông tin) nào đó trong thế giới thực, câu đơn có giá trị chân lý true hay false tùy theo mệnh đề (phát biểu, sự kiện, thông tin) mà nó ám chỉ là đúng hay sai trong thực tế.
- ✓ Câu phức là câu biểu diễn (ánh xạ với) một phủ định, mối quan hệ hoặc mối liên kết giữa các mệnh đề/phát biểu/câu con hoặc một sự phổ biến hoặc đặc thù của mệnh đề/phát biểu trong thế giới thực. Ngữ nghĩa và giá trị chân lý của các câu phức này được xác định dựa trên các câu con thành phần của nó, chẳng hạn:
  - $\neg$A có nghĩa là phủ định mệnh đề/ câu A, nhận giá trị true nếu A là false và ngược lại
  - A $\wedge$ B có nghĩa là mối liên kết "A và B", nhận giá trị true khi cả A và B là true, và nhận giá trị false trong các trường hợp còn lại.
  - A $\vee$ B biểu diễn mối liên kết "A hoặc B", nhận giá trị true khi hoặc A hoặc B là true, và nhận giá trị false chỉ khi cả A và B là false.

- ($A \Rightarrow B$) biểu diễn mối quan hệ "A kéo theo B", chỉ nhận giá trị false khi A là true và B là false; nhận giá trị true trong các trường hợp khác
- ($A \Leftrightarrow B$) biểu diễn mối quan hệ "A kéo theo B" và "B kéo theo A"
- $\forall x$ A biểu diễn sự phổ biến của A, nhận giá trị true tất cả các câu sinh ra từ A bằng cách thay x bởi một giá trị/đối tượng cụ thể thuộc miền giá trị biến x đều là true, ngược lại thì câu phổ biến này nhận giá trị false
- $\exists x$ A biểu diễn sự tồn tại của A, nhận giá trị true khi có một giá trị x0 trong miền giá trị của biến x làm cho A true, false trong các trường hợp còn lại.

Như vậy, việc xác định tính đúng/sai của một câu đơn (vị từ) là dựa trên tính đúng sai của sự kiện hoặc thông tin mà nó ám chỉ, còn việc xác định tính đúng sai của câu phức phải tuân theo các qui tắc trên. Trong nhiều trường hợp, chúng ta (cần chỉ) biết tính đúng/sai của các câu phức, còn tính đúng/sai của các câu đơn là không cần biết hoặc có thể lập luận ra từ các các câu phức đã biết đúng/sai và các qui tắc chuyển đổi tính đúng/sai giữa các câu đơn và câu phức theo các qui tắc trên.

1.3 Các ví dụ:

Các câu trong ngôn ngữ tự nhiên có thể biểu diễn trong logic vị từ cấp một:

- ✓ "An là sinh viên" Sinhvien(An)
- ✓ "Nam là cha của Hoàn" Cha(Nam,Hoàn)
- ✓ "Mọi sinh viên đều học giỏi" $\forall x$ Sinhvien(x) $\Rightarrow$ Hocgioi(x)
  (chú ý $\forall$ thường đi với $\Rightarrow$. Khác với $\forall x$ Sinhvien(x) $\wedge$ Hocgioi(x))
- ✓ "Trong sinh viên có bạn học giỏi" $\exists x$ Sinhvien(x) $\wedge$ Hocgioi(x)
  (chú ý $\exists$ thường đi với $\wedge$. Khác với $\exists x$ Sinhvien(x) $\Rightarrow$ Hocgioi(x).

1.4 Các câu hằng đúng (có giá trị chân lý luôn bằng true)

Ngoài các công câu hằng đúng trong logic mệnh đề, chúng ta thêm các câu hằng đúng liên quan đến các lượng tử như sau:

- $\forall x\, P(x) \Leftrightarrow \forall y\, P(y)$ (qui tắc đổi tên)
- $\exists x\, P(x) \Leftrightarrow \exists y\, P(y)$ (qui tắc đổi tên)
- $\forall x \forall y\, P(x,y) \Leftrightarrow \forall y \forall x\, P(x,y)$ (qui tắc giao hoán)
- $\exists x \exists y\, P(x,y) \Leftrightarrow \exists y \exists x\, P(x,y)$ (qui tắc giao hoán)
- $\forall x\, P(x) \Leftrightarrow \neg \exists x\, \neg P(x)$ (chuyển đổi giữa $\forall$ và $\exists$)
- $\exists x\, P(x) \Leftrightarrow \neg \forall x\, \neg P(x)$ (chuyển đổi giữa $\forall$ và $\exists$)
- $\neg(\forall x\, P(x)) \Leftrightarrow \exists x\, \neg P(x)$ (DeMorgan)
- $\neg(\exists x\, P(x)) \Leftrightarrow \neg \forall x\, \neg P(x)$ (DeMorgan)
- $\forall x\, P(x) \Rightarrow P(a)$, với a là giá trị thuộc miền giá trị của X (loại bỏ $\forall$)
- $\exists x\, P(x) \Rightarrow P(e)$, với e là một giá trị vô danh, không có trong cơ sở tri thức (loại bỏ $\exists$)
- $P(a) \Rightarrow \exists x\, P(x)$ (đưa ký hiêu $\exists$ vào)
- Luật phân giải tổng quát (xem mục 3 của Chương này)
- Modus Ponens tổng quát (xem mục 4 của Chương này)

2. Lập luận trong logic vị từ cấp một

- Ví dụ: Xem xét bài toán lập luận (hay chứng minh) được phát biểu trong ngôn ngữ tự nhiên như sau:

  Cho:

  "An là con trai. Thủy là con gái. Tóc của con gái dài hơn tóc của con trai"

  Hãy chứng minh:

"Tóc của Thủy dài hơn tóc của An"

Bài toán này có thể biểu diễn trong logic vị từ cấp một như sau:

Cho các câu sau (cơ sở tri thức - KB) là đúng:

$$\text{Contrai(An)} \quad (1)$$

$$\text{Congai(Thuy)} \quad (2)$$

$$\forall x \forall y\ \text{Contrai}(x) \wedge \text{Congai}(y) \Rightarrow \text{Tocdaihon}(y,x) \quad (3)$$

Chúng ta cần chứng minh (câu truy vấn q):

Tocdaihon(Thuy,An).

Đây là một lời giải của bài toán trên (lời giải là dãy các bước áp dụng luật logic vị từ cấp một để đưa cơ sở tri thức về điều cần chứng minh):

Bước 1: Từ (1) và (2) ta áp dụng luật đưa $\wedge$ vào $(A,B \Rightarrow A \wedge B)$:

$$\text{Contrai(An)} \wedge \text{Congai(Thuy)} \quad (4)$$

Bước 2: Áp dụng luật loại bỏ $\forall$ trong (3) với {x/An, y/Thuy} ta được:

$$\text{Contrai(An)} \wedge \text{Congai(Thuy)} \Rightarrow \text{Tocdaihon(Thuy,An)} \quad (5)$$

Bước 3: Áp dụng luật Modus ponens cho (4) và (5) ta có:

$$\text{Tocdaihon(Thuy,An)} \quad (6)$$

Đến đây ta được điều phải chứng minh.

- Cũng giống như trong logic mệnh đề, bài toán lập luận (chứng minh $KB \models q$) có thể xem là bài toán tìm đường đi như sau:
  - ✓ *Trạng thái đầu*: KB
  - ✓ *Các phép chuyển trạng thái*: mỗi phép chuyển trạng thái là một lần áp dụng luật trong logic vị từ cấp một (nhiều hơn các luật của logic mệnh đề) trên tập câu trong KB. Mỗi luật *l* áp dụng cho *KB* sinh ra câu mới *l(KB)*, bổ sung câu mới này vào *KB* được trạng thái mới $KB \wedge l(KB)$

- ✓ *Trạng thái đích*: trạng thái *KB* chứa *q*
- ✓ *Chi phí cho mỗi phép chuyển*: 1

- Bài toán trên có thể tìm được lời giải bằng cách áp dụng các thuật toán tìm kiếm như đã trình bày trong các chương đầu của giáo trình này về tìm kiếm. Tuy nhiên không gian tìm kiếm lời giải của bài toán này là rất lớn. Cũng giống như trong logic mệnh đề, nếu cơ sở tri thức (KB) và câu truy vấn (q) được biểu diễn bằng (hoặc có thể chuyển được sang) các câu có dạng thích hợp, thì chúng ta có thể chỉ cần áp dụng một loại luật của logic mệnh đề để chứng minh rằng $KB \models q$. Cụ thể là nếu KB và q biểu diễn được bằng các câu Horn thì chỉ cần áp dụng liên tiếp các luật Modus ponens là chứng minh được $KB \models q$ (xem mục 5,7,8); còn nếu KB và q biểu diễn bằng các câu dạng chuẩn hội thì ta chỉ cần liên tiếp áp dụng các luật phân giải là thực hiện được việc suy diễn $KB \models q$ (xem mục 4,6).

3. *Phép đồng nhất hai vị từ, thuật giải đồng nhất*

- *Phép đồng nhất là gì*? Khi áp dụng luật trong logic vị từ, ta thường xuyên gặp phải việc đối sách các vị từ trong hai câu xem chúng có thể đồng nhất được với nhau không (tức là chúng sẽ hoàn toàn như nhau trên một bộ giá trị nào đó. Chẳng hạn ở bước 2 trong chứng minh ví dụ trên, khi áp dụng Luật loại bỏ ký hiệu ∀ trong câu có tính phổ biến (3) để được câu cụ thể trên bộ giá trị (x=An, y=Thuy), ta phải đối sánh các cặp vị từ <Contrai(An) và Contrai(x)>, <Congai(Thuy) và Congai(y)> để tìm ra giá trị x=An, y=Thuy để cho các cặp vị từ đó là hoàn toàn như nhau (để có áp dụng các luật tiếp theo). Việc đối sánh hai vị từ để tìm ra một bộ giá trị cho các biến sao cho hai vị từ là đồng nhất được gọi là phép đồng nhất. Vậy phép đồng nhất là thao tác thực hiện trên hai vị từ (hoặc phủ định của vị từ) và cho kết quả là sự thay thế các biến xuất hiện trong các vị từ bằng các hạng thức (các giá trị) để hai vị từ đó là như nhau.

- Ví dụ:
  - ✓ Đồng nhất (Contrai(An), Contrai(y)) = {y/An}

- ✓ Đồng nhất (Yêu(An,x), Yêu(y,Binh)) = {x/Binh; y/An}
- ✓ Đồng nhất (Yêu(An,x), Yêu(y, Emgai(Hoa)) = {x/Emgai(Hoa); y/An} (chú ý: trong trường hợp này Emgai(x) là một hàm – em gái của x, không phải là vị từ)
- ✓ Đồng nhất (Yeu(An,x), Yeu(An,y)) = {x, y/x }
- ✓ Đồng nhất (Ban(An,x), Ban(y, Emgai(y))={x/Emgai(An); y/An}
- ✓ Đồng nhất (P(a,X), P(X,b)) = **failure**
- ✓ Đồng nhất[**parents(x, father(x), mother(Jane))**, **parents(Bill, father(y), mother(y))]= failure**

➢ Giải thuật đồng nhất:

- ✓ Input: hai literal *p* và *q*.
- ✓ Output: Sự thay thế gán giá thay thế các biến *theta*

```
Procedure Đồng_nhất(p, q, theta) return true or false
    (r,s)=hạng thức đầu tiên không nhất quán giữa (p,q);
    if ((r,s)=empty) return theta; thành công
    if (là_biến(r))
        theta = theta ∪ {r/s}
        Đồng_nhất(thaythe(theta,p), thaythe(theta,q), theta)
    elseif (là_biến(s))
        theta = theta ∪ {s/r}
        Đồng_nhất(thaythe(theta,p), thaythe(theta,q), theta)
    else return failure
```

4. *Câu dạng chuẩn hội, luật phân giải tổng quát*

- Câu dạng chuẩn hội: Cũng giống như trong logic mệnh đề, câu dạng chuẩn hội trong logic vị từ cấp một có dạng sau (là hội của các tuyển)

$$(\underbrace{A_{11} \vee A_{12} \vee \ldots \vee A_{1n}}_{\text{clause}}) \wedge (\underbrace{A_{21} \vee A_{22} \vee \ldots \vee A_{2m}}_{\text{clause}}) \wedge \ldots \wedge (\underbrace{A_{k1} \vee A_{k2} \vee \ldots \vee A_{kr}}_{\text{clause}})$$

với A_{ij} là các literal (là ký hiệu vị từ hoặc phủ định của ký hiệu vị từ).

(chính xác hơn phải có thêm các lượng từ $\forall$ cho tất cả các biến trong câu)

- Chuyển câu bất kỳ sang dạng chuẩn hội: một câu bất kỳ trong logic vị từ cấp một đều có thể biểu diễn sang dạng chuẩn hội. Để chuyển một câu sang dạng chuẩn hội, ta áp dụng các qui tắc sau đây:
  - ✓ QT1: Loại bỏ $\Leftrightarrow$: thay thế $\alpha \Leftrightarrow \beta$ bằng $(\alpha \Rightarrow \beta) \wedge (\beta \Rightarrow \alpha)$.
  - ✓ QT2: Loại bỏ $\Rightarrow$: Thay thế $\alpha \Rightarrow \beta$ bằng $\neg\alpha \vee \beta$
  - ✓ QT3: chuyển hoặc loại bỏ dấu $\neg$ đặt trước các ký hiệu bằng các luật deMorgan và luật phủ định kép $\neg(\alpha \vee \beta) = \neg\alpha \wedge \neg\beta$; $\neg(\alpha \wedge \beta) = \neg\alpha \vee \neg\beta$; $\neg\neg\alpha = \alpha$; $\neg\forall x P(x) = \exists x \neg P(x)$; $\neg\exists x P(x) = \forall x \neg P(x)$
  - ✓ QT4: Chuẩn hóa các biến: các biến lượng từ không được trùng tên, ví dụ $\forall x P(x) \vee \exists x Q(x)$ chuyển thành $\forall x P(x) \vee \exists y Q(y)$
  - ✓ QT5: chuyển các lượng từ về đầu câu, ví dụ $\forall x P(x) \vee \exists y Q(y)$ chuyển thành $\forall x \exists y\, P(x) \vee Q(y)$
  - ✓ QT6: loại bỏ $\exists$ bằng giá trị vô danh: ví dụ $\exists x$ Rich(x) trở thành Rich(c) với c là ký hiệu hằng vô danh, không trùng với các ký hiệu có trong cơ sở tri thức. Chú ý khi $\exists$ đặt bên trong $\forall$, phải sử dụng hàm vô danh; ví dụ: $\forall x \exists y \forall z$ P(x,y,z) trở

thành $\forall x \forall z\ P(x,f(x),z)$ với f là ký hiệu hàm vô danh, không trùng với ký hiệu hàm khác trong cơ sở tri thức.

- ✓ QT7: bỏ qua các ký hiệu lượng tử $\forall$
- ✓ QT8: Áp dụng luật phân phối của phép $\wedge$ đối với phép $\vee$

Ví dụ: Biểu diễn các câu sau thành các câu trong logic vị từ và chuyển chúng về dạng chuẩn hội:

"Tất cả con chó đều sủa về ban đêm. Hễ nhà ai có mèo thì nhà người đó đều không có chuột. Những ai khó ngủ thì đều không nuôi bất cứ con gì mà sủa về ban đêm. Bà Bình có mèo hoặc có chó"

$$\forall x\ (\text{Là\_Chó}(x) \Rightarrow \text{Sủa\_về\_đêm}(x)) \quad (1)$$

$$\forall x \forall y\ (\text{Có}(x,y) \wedge \text{Là\_Mèo}(y) \Rightarrow \neg \exists z\ (\text{Có}(x,z) \wedge \text{Là\_Chuột}(z))) \quad (2)$$

$$\forall x\ (\text{Khó\_ngủ}(x) \Rightarrow \neg\ \exists z(\text{Có}(x,z) \wedge \text{Sủa\_về\_đêm}(z)) \quad (3)$$

$$\exists x\ (\text{Có}(\text{BBinh},x) \wedge (\text{Là\_Mèo}(x) \vee \text{Là\_Chó}(x)) \quad (4)$$

Áp dụng các qui tắc (QT) ở trên, ta chuyển sang các câu dạng clause như sau:

(1) Tương đương với: $\neg \text{Là\_Chó}(x) \vee \text{Sủa\_về\_đêm}(x)$

(2) $\forall x \forall y\ (\neg\ (\text{Có}(x,y) \wedge \text{Là\_Mèo}(y)) \vee (\neg \exists z\ (\text{Có}(x,z) \wedge \text{Là\_Chuột}(z))))$

$\forall x \forall y\ (\neg \text{Có}(x,y) \vee \neg \text{Là\_Mèo}(y) \vee (\forall z\ (\neg \text{Có}(x,z) \vee \neg \text{Là\_Chuột}(z))))$

$\forall x \forall y \forall z\ (\neg \text{Có}(x,y) \vee \neg \text{Là\_Mèo}(y) \vee (\neg \text{Có}(x,z) \vee \neg \text{Là\_Chuột}(z)))$

$\neg \text{Có}(x,y) \vee \neg \text{Là\_Mèo}(y) \vee \neg \text{Có}(x,z) \vee \neg \text{Là\_Chuột}(z)$

(3) $\forall x\ (\neg \text{Khó\_ngủ}(x) \vee (\neg\ \exists z(\text{Có}(x,z) \wedge \text{Sủa\_về\_đêm}(z)))$

$\forall x\ (\neg \text{Khó\_ngủ}(x) \vee (\forall z(\neg \text{Có}(x,z) \vee \neg \text{Sủa\_về\_đêm}(z)))$

$\forall x \forall z\ (\neg \text{Khó\_ngủ}(x) \vee \neg \text{Có}(x,z) \vee \neg \text{Sủa\_về\_đêm}(z))$

$\neg \text{Khó\_ngủ}(x) \vee \neg \text{Có}(x,z) \vee \neg \text{Sủa\_về\_đêm}(z)$

(4) $\exists x$ (Có(BBinh,x) $\wedge$ (Là_Mèo(x) $\vee$ Là_Chó(x))

Có(BBinh,a) $\wedge$ (Là_Mèo(a) $\vee$ Là_Chó(a)) (tách ra thành hai clause)

- Luật phân giải:

Nếu chúng ta có hai clause sau là đúng:

$(P_1 \vee P_2 \vee \ldots P_i \vee \ldots \vee P_n) \wedge$

$(Q_1 \vee Q_2 \vee \ldots Q_j \vee \ldots \vee Q_m)$

và có phép thay thế theta sao cho

thaythe(theta,P_i)= $\neg$thaythe(theta,Q_j)

thì chúng ta cũng có clause sau là đúng

thaythe(theta, $P_1 \vee P_2 \vee \ldots P_{i-1} \vee P_{i+1} \vee \ldots \vee P_n \vee Q_1 \vee Q_2 \vee \ldots Q_{j-1} \vee Q_{j+1} \vee \ldots \vee Q_m$)

(Clause mới là tuyển các literal trong hai clause ban đầu nhưng bỏ đi P_i và Q_j)

- Kết quả của phép phân giải cũng là một clause (tuyển các literal), hay nói cách khác phép phân giải có tính đóng, phân giải của các clause là một clause. Đây là tính chất rất quan trong trong việc xây dựng giải thuật suy diễn tự động trình bày phía dưới.

5. Câu dạng Horn và tam đoạn luận tổng quát trong logic cấp 1

- Câu dạng Horn: Tất cả các câu trong logic vị từ cấp một đều có thể biểu diễn được dưới dạng chuẩn hội, tức là hội của các clause, mỗi clause có dạng: $P_1 \vee P_2 \vee \ldots P_i \vee \ldots \vee P_n$, với P_i là các literal. Nếu trong clause mà có nhiều nhất một literal dương (tức là không có ký hiệu phủ định đằng trước) thì clause đó gọi là câu dạng Horn. Như vậy câu dạng Horn là câu có một trong ba dạng:

$\neg P_1 \vee \neg P_2 \vee \ldots \vee \neg P_n$ (không có literal dương nào)

hoặc P (có một literal dương và không có literal âm nào)

hoặc $\neg P_1 \vee \neg P_2 \vee \ldots \vee \neg P_n \vee Q$ (có một literal dương là Q và ít nhất một literal âm)

với $P_1, P_2, \ldots, P_n$ và Q là các ký hiệuvị từ.

Nếu chuyển các câu dạng Horn sang dạng luật thì chúng có dạng như sau:

$\neg(P_1 \wedge P_2 \wedge \ldots \wedge P_n)$

hoặc P

hoặc $P_1 \wedge P_2 \wedge \ldots \wedge P_n \Rightarrow Q$ (có một literal dương là Q)

Trong đó câu dạng thứ hai và câu ba gọi là câu Horn dương (có đúng 1 literal dương) và thường được sử dụng để biểu diễn tri thức trong cơ sở tri thức KB. Câu dạng thứ nhất được gọi là câu dạng Horn âm (không có literal dương nào), và phủ định câu dạng Horn âm này sẽ là hội các câu Horn dương. Câu dạng Horn âm chỉ xuất hiện trong biểu diễn các câu truy vấn (q) vì khi đó $\neg q$ sẽ là các câu Horn dương và thay vì chứng minh KB suy diễn ra q thì ta chứng minh $KB \wedge \neg q$ suy diễn ra [], khi này cơ sở tri thức $KB \wedge \neg q$ là hội các câu dạng Horn dương.

- Tam đoạn luận (hay luật Modus ponens tổng quát):

Nếu chúng ta có các câu Horn dương sau là đúng:

P'_1,

P'_2,

…

P'_n

$P_1 \wedge P_2 \wedge \ldots \wedge P_n \Rightarrow Q$

và có phép thay thế theta sao cho

thaythe(theta,P'_i)= thaythe(theta, P_i)

thì câu thaythe(theta,Q) là đúng

- Kết quả luật Modus ponens từ hai câu dạng Horn dương sinh ra câu thaythe(theta,Q) cũng có dạng Horn dương. Vì vậy phép suy diễn tam đoạn luận là đóng trong các câu dạng Horn, kết quả tam đoạn luận từ hai câu dạng Horn là câu dạng Horn. Tương tự như tính chất đóng của phép phân giải trong trong các câu dạng chuẩn hội, tính chất đóng của phép suy luận này là rất quan trọng trong việc thiết kế các giải thuật suy diễn tự động đề dựa trên tam đoạn luận và các câu Horn (xem phần phía dưới).
- Không giống như câu dạng chuẩn hội, không phải câu nào trong logic mệnh đề đều có thể biểu diễn dạng Horn được. Chính vì thế mà thuật giải suy diễn dựa trên tam đoạn luận chỉ là đầy đủ trong ngôn ngữ các câu Horn chứ không đầy đủ trong logic mệnh đề.

6. Giải thuật suy diễn phân giải

- Giải thuật suy diễn phân giải dựa trên luật phân giải: hai câu tuyển (clause) có một literal dương và một literal âm mà đồng nhất với nhau được thì sẽ sinh ra câu tuyển mới là tuyển các literal còn lại của cả hai câu sau khi bỏ đi hai literal đồng nhất này. Câu mới (là kết quả của phép phân giải) cũng là câu dạng tuyển (clause) và khi bổ sung vào KB (tức là KB $\wedge$ câu_clause_mới) thì kết quả KB cũng là dạng chuẩn hội (hội các câu tuyển). Vì vậy mà trước khi áp dụng giải thuật phân giải ta phải chuyển KB $\wedge \neg$q sang dạng chuẩn hội.
- Giống như giải thuật phân giải trong logic mệnh đề, giải thuật phân giải trong loc vị từ cấp một cũng thực hiện liên tiếp các phép phân giải hai clause trong biểu diễn dạng chuẩn hội của $KB \wedge \neg$q, bổ sung clause mới vào KB và lặp lại đến khi hoặc sinh ra câu rống ([]) hoặc không kết quả phân giải không bổ sung thêm clause nào vào KB được nữa.

```
Function Resolution(KB, q) return true or false
    KB = KB ∧ ¬q
    clauses=get_list_of_clauses(KB);
    while ([] not in KB)
      (Ci,Cj)=get_resolvable_pair(KB); // lấy hai câu mà chứa cặp literals
                                       //có thể đồng nhất với nhau được,
                                       //nhưng dấu ngược nhau
       if  (Ci,Cj)=empty return "failure“
       else
            resolvent = resolution-rule(S1, S2);
            KB = KB ∧ resolvent;
    return “success”;
```

- Mỗi lần thực hiện phép phân giải là một phép chuyển trạng thái từ KB sang trạng thái mới KB ∧ resolvent (với resolvent là kết quả của phép phân giải). Ở một trạng thái bất kỳ, có nhiều cặp clause có thể phân giải được với nhau, hay nói cách khác có nhiều phép chuyển trạng thái; việc lựa chọn phép chuyển trạng thái nào là dựa trên chiến lược lựa chọn, chúng ta có thể chọn theo chiều rộng, hoặc chọn theo chiều sâu như các chiến lược tìm kiếm theo chiều rộng hoặc theo chiều sâu như đã trình bay trong Chương Các phương pháp tìm kiếm lời giải.
- Việc chứng minh $KB \wedge \neg q \models []$ cũng có thể thực hiện bằng chiến lược chứng minh lùi (tìm kiếm lùi), xuất phát từ $\neg q$ (là đích của bài toán gốc $KB \models q$ chứ không phải đích []) ta tìm các câu trong *KB* có thể phân giải được với $\neg q$, áp dụng luật phân giải theo chiều rộng, đến khi nào [] được sinh ra thì dừng. Giải thuật phân giải theo cách này gọi là giải thuật phân giải lùi.

- Ví dụ minh họa: Giả sử chúng ta có cơ sở tri thức như cho trong ví dụ ở mục 4 trong Chương này, hãy chứng minh "Nếu bà Bình là người khó ngủ thì nhà bà ấy không có chuột". Câu cần chứng minh này tương đương với câu sau trong logic vị từ cấp một (q):

  $$\text{Khó\_ngủ(BBinh)} \Rightarrow \neg\exists z(\text{Có(BBinh,z)} \wedge \text{Là\_Chuột(z)})$$

  Và ¬q là câu:

  $$\neg(\text{Khó\_ngủ(BBinh)} \Rightarrow \neg\exists z(\text{Có(BBinh,z)} \wedge \text{Là\_Chuột(z)}))$$

  Hay các câu tương đương sau:

  $$\neg[\neg\text{Khó\_ngủ(BBinh)} \vee (\neg\exists z(\text{Có(BBinh,z)} \wedge \text{Là\_Chuột(z)}))]$$

  $$[\text{Khó\_ngủ(BBinh)} \wedge \exists z(\text{Có(BBinh,z)} \wedge \text{Là\_Chuột(z)}))]$$

  $$\text{Khó\_ngủ(BBinh)} \wedge \text{Có(BBinh,b)} \wedge \text{Là\_Chuột(b)}$$

  (với b là ký hiệu hằng vô danh)

  Khi đó $KB \wedge \neg q$ gồm các clause sau (dạng chuẩn hội):

  $$\neg\text{Là\_Chó(x)} \vee \text{Sủa\_về\_đêm(x)} \quad (1)$$

  $$\neg\text{Có(x,y)} \vee \neg\text{Là\_Mèo(y)} \vee \neg\text{Có(x,z)} \vee \neg\text{Là\_Chuột(z)} \quad (2)$$

  $$\neg\text{Khó\_ngủ(x)} \vee \neg\text{Có(x,z)} \vee \neg\text{Sủa\_về\_đêm(z)} \quad (3)$$

  $$\text{Có(BBinh,a)} \quad (4)$$

  $$\text{Là\_Mèo(a)} \vee \text{Là\_Chó(a)} \quad (5)$$

  $$\text{Khó\_ngủ(BBinh)} \quad (6)$$

  $$\text{Có(BBinh,b)} \quad (7)$$

  $$\text{Là\_Chuột(b)} \quad (8)$$

  $KB \wedge \neg q \models []$ theo các bước phân giải như sau:

- (1) và (5) {x/a}

$$Là\_Mèo(a) \vee Sủa\_về\_đêm(a) \quad (9)$$

- (2) và (8) {z/b}

$$\neg Có(x,y) \vee \neg Là\_Mèo(y) \vee \neg Có(x,b) \quad (10)$$

- (7) và (10) {x/BBinh}

$$\neg Có(BBinh,y) \vee \neg Là\_Mèo(y) \quad (11)$$

- (9) và (11) {y/a}

$$\neg Có(BBinh,a) \vee Sủa\_về\_đêm(a) \quad (12)$$

- (4) và (12)

$$Sủa\_về\_đêm(a) \quad (13)$$

- (3) và (13) {z/a}

$$\neg Khó\_ngủ(x) \vee \neg Có(x,a) \quad (14)$$

- (4) và (14) {x/BBinh}

$$\neg Khó\_ngủ(BBinh) \quad (15)$$

- (6) và (15)

$$[]$$

Dãy các bước chứng minh ở trên chỉ là một lời giải của bài toán chứng minh $KB \wedge \neg q \models []$. Bạn đọc có thể đưa ra lời giải khác.

7. Thuật toán suy diễn tiến dựa trên câu Horn

Giải thuật suy diễn phân giải ở trên là đầy đủ trong logic vị từ cấp một, có nghĩa là giải thuật sẽ cho phép chứng minh được $KB \models q$ chỉ bằng áp dụng mỗi loại luật phân giải nếu q chứng minh được từ KB trong logic vị từ cấp một (vì ta luôn có thể chuyển $KB \wedge \neg q$ về dạng chuẩn hội các câu tuyển và vì thế chỉ cần áp dụng luật phân giải). Tuy nhiên, giải thuật phân giải phải duyệt tất cả các cặp câu tuyển có trong KB mà có thể phân giải được với nhau và chọn cách phân giải theo một chiến lược (tìm kiếm)

nào đó, sau đó bổ sung kết quả phân giải vào KB và lặp lại thực hiện tìm kiếm các câu tuyển có thể phân giải được. Giải thuật này thường không hiệu quả vì số lượng câu tuyển trong KB sẽ tăng lên sau mỗi lần lặp.

Trong mục này, chúng ta sẽ xem xét các giải thuật chứng minh hiệu quả hơn. Như đã xét trong mục 5, luật Modus ponens (hay tam đoạn luận) có tính chất đóng trong các câu Horn dương (câu tuyển có đúng một literal dương), vì thế nếu cả KB và q (hoặc $\neg q$) có thể biểu diễn được dạng câu Horn dương thì chúng ta có thể chứng minh $KB \models q$ (hoặc $KB \wedge \neg q \models []$) chỉ bằng các luật Modus ponens.

Để chứng minh $KB \models q$ (khi KB biểu diễn bằng hội các câu Horn dương), ta chia KB thành 2 loại câu: (1) câu có một literal dương và không có literal âm nào (hay gọi là các câu đơn hoặc các câu sự kiện) và (2) câu có một literal dương và có ít nhất một literal âm (hay goi là câu luật). Giải thuật suy diễn tiến thực hiện như sau: bắt đầu với tập các câu sự kiện trong KB, lặp lại việc áp dụng các luật Modus ponens tổng quát (xem mục 5) để sinh ra các câu sự kiện mới, nếu câu sự kiện mới này là q thì dừng và thông báo suy diễn thành công, nếu không thì bổ sung các câu sự kiện mới này vào tập các câu sự kiện đã biết và áp dụng các luật Modus ponens tổng quát; nếu không có câu sự kiện mới nào được sinh ra thì việc chứng minh $KB \models q$ là thất bại. Chi tiết giải thuật suy diễn tiến dựa trên các câu Horn dương và luật Modus ponens tổng quát như trang sau.

Giải thuật suy diễn tiến có một số nhược điểm, trong đó có nhược điểm là nó sẽ sinh ra rất nhiều sự kiện mà không liên quan gì đến câu truy vấn (vì bản chất của giải thuật này là tìm kiếm theo chiều rộng).

Function FOL_Forward_Horn(KB, q) *return* true or false

Input: - KB tập các câu Horn dương (câu sự kiện, câu kéo theo)

- q: câu truy vấn dạng câu đơn (ký hiệu vị từ)

Output: true or false

while new *is not* empty

new $\leftarrow$ {};

for each r in {câu kéo theo trong KB}

($P_1 \wedge P_2 \wedge \ldots \wedge P_n \Rightarrow Q$) $\leftarrow$ Phântíchcâu(r);

f*or some* $P'_1, P'_2, \ldots P'_n$ in {câu sự kiện trong KB}

if (Đồng_nhất($P_1 \wedge P_2 \wedge \ldots \wedge P_n, P'_1 \wedge P'_2 \wedge \ldots \wedge P'_n, \theta$)

Q' $\leftarrow$ thaythe(θ,Q);

if (Đồng_nhất (Q',q)) return true

else new $\leftarrow$ new $\cup$ Q';

KB $\leftarrow$ KB $\cup$ new;

return false;

8. Thuật toán suy diễn lùi dựa trên câu Horn

Chương 6 – Prolog

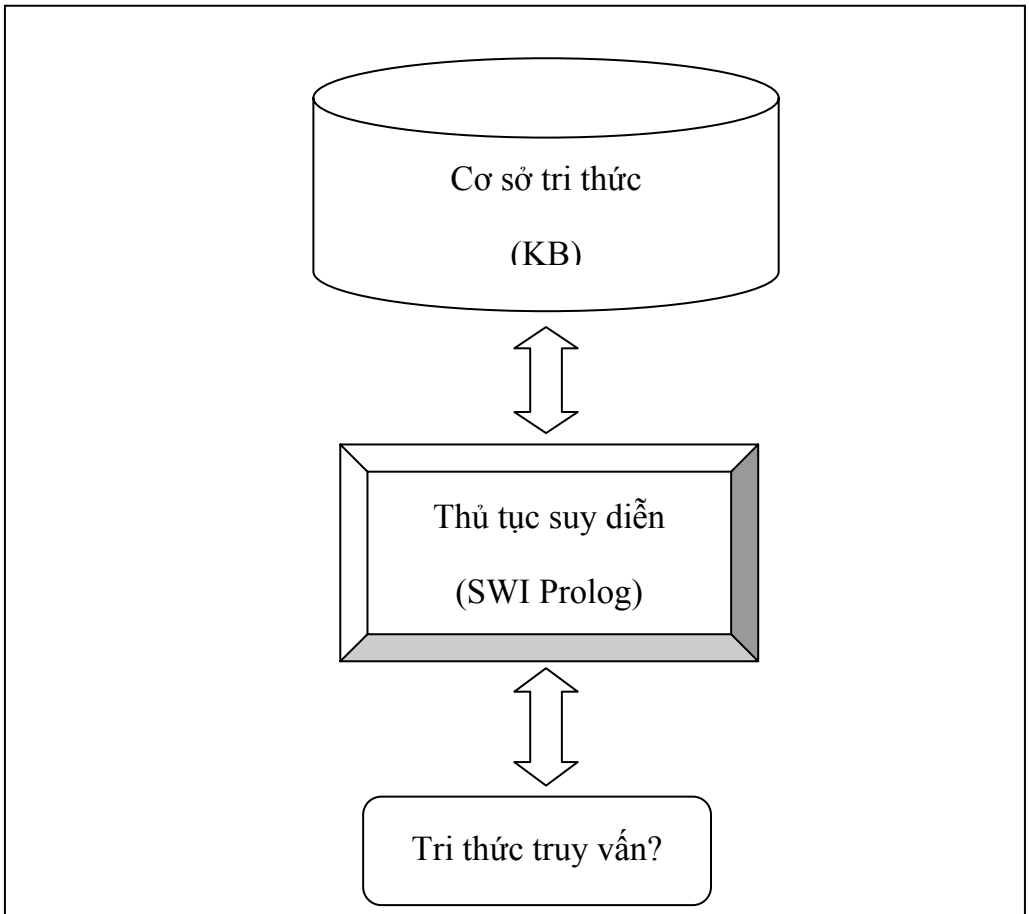
Trong Chương 4 và 5 chúng ta đã tìm hiểu logic mệnh đề và logic vị từ cấp một. Chúng ta cũng đã tìm hiểu các thuật toán lập luận tự động, chứng minh câu truy vấn q từ cơ sở tri thức KB. Có hai loại thuật toán lập luận cơ bản: (1) Lập luận trong các câu dạng chuẩn hội với luật phân giải, và (2) Lập luận trong các câu Horn với luật Modus ponens (hay tam đoạn luận). Trong Chương này, chúng ta sẽ tìm hiểu một ngôn ngữ con của Logic vị từ cấp một, **prolog – programming in logic, ngôn ngữ gồm các câu Horn trong Logic vị từ cấp một có bổ sung một số thành phần phi logic giúp cho sức mạnh biểu diễn của ngôn ngữ Prolog tốt hơn và giúp cho việc cài đặt các giải thuật suy diễn dễ dàng và hiệu quả hơn.** Rất nhiều thuật toán lập luận tự động trong Prolog đã được cài đặt cho máy tính, ví dụ như SWI Prolog phát triển bởi J. Wielemaker, SICS Prolog phát triển bởi Viện Khoa học máy tính Thụy Điển, v.v.. Ngôn ngữ Prolog mà các sản phẩm này cung cấp là tương đối giống nhau (có sai khác không đáng kể). Ngoài chức năng cơ bản là cung cấp trình biên dịch (thuật toán lập luận $KB \models q$) thì hầu hết các sản phẩm đều cung cấp bộ soạn thảo chương trình (cơ sở tri thức).

Trong Chương này, chúng ta sẽ tìm hiểu ngôn ngữ Prolog, Phần mềm SWI Prolog, và lập trình Prolog.

1. Lập trình logic, môi trường lập trình SWI Prolog

Lập trình logic:

Khác với các lập trình thủ tục (lập trình C, Pasal, Fortran, v.v.) là chỉ ra thứ tự các câu lệnh xử lý trên tập các cấu trúc dữ liệu để giải quyết bài toán sinh ra output từ input; lập trình logic là ***khai báo*** các sự kiện, tri thức (luật) đã biết (hoặc đã đúng) và sử dụng máy tính (có trang bị thuật giải suy diễn) để ***truy vấn*** một sự kiện mới hoặc tri thức mới từ các sự kiện và tri thức đã cho (xem sơ đồ bên dưới). Các loại tri thức truy vấn có thể kiểm tra một sự kiện hoặc tri thức nào đó có đúng hay không, hoặc liệt kê các bộ giá trị của các biến sao cho thỏa mãn điều kiện logic nào đó (tức là làm cho một biểu thức logic nào đó nhận giá trị *true*).



Lập trình logic = Khai báo Cơ sở tri thức + Truy vấn

Để trả lời các câu truy vấn, chúng ta cần thủ tục suy diễn (lập luận) như đã trình bày trong các chương trước. Chúng ta đã biết, khi cơ sở tri thức biểu diễn được thành hội các câu Horn thì thuật toán suy diễn sẽ rất hiệu quả (có độ phức tạp thời gian là tuyến tính đối với số câu Horn trong cơ sở tri thức). Vì thế mà hầu hết các sản phẩm cài đặt trên máy tính đều hạn chế ngôn ngữ biểu diễn tri thức dạng các câu Horn. Trong tài liệu này, chúng ta sẽ tìm hiểu một cài đặt miễn phí, SWI Prolog.

Môi trường lập trình SWI Prolog:

SWI Prolog là một cài đặt thủ tục suy diễn hỗ trợ các câu Horn có bổ sung thêm một số thành phần phi logic (các phép toán input/output, các phép toán tăng sức mạnh biểu diễn hoặc tăng tính hiệu quả của thuật toán suy diễn). Bộ suy diễn của SWI Prolog sử dụng giải thuật phân giải SLD (Selective Linear Definite clause resolution), ý tưởng chính là biểu diễn các câu Horn dạng các câu tuyển (clause) có một literal dương, rồi áp dụng giải thuật phân giải lùi. Giải thuật phân giải SLD này sẽ được mô tả chi tiết trong phần cuối của Chương này. SWI Prolog có thể download miễn phí tại địa chỉ sau:

http://www.swi-prolog.org/download/stable.

Sauk khi cài đặt và chạy chương trình SW Prolog, Hệ thống hiển thị dấu nhắc yêu cầu nhập vào câu truy vấn như sau:

```
1?- |
```

Tất nhiên, trước khi nhập câu truy vấn, chúng ta phải cho Hệ thống biết chúng ta sẽ truy vấn trên cơ sở tri thức nào. Một cơ sở tri thức là một khai báo các sự kiện và các luật về một lĩnh vực nào đó, và được lưu trong một file. Để load một file cơ sở tri thức, ta sử dụng menu File → Consult → Chọn file. Các mô tả các sự kiện và luật (các câu Horn) trong các file cơ sở tri thức được gọi là chương trình prolog. Nhiệm vụ của người lập trình logic là viết các chương trình prolog này và các câu truy vấn.

Ví dụ, ta soạn thảo một file chương trình prolog (cơ sở tri thức) có tên file là *giapha.pl* (có thể sử dụng bất cứ bộ soạn thảo văn bản nào, hoặc sử chính bộ soạn thảo do SWI Prolog cung cấp bằng cách sử dụng menu → File → New/Edit → Nhập tên file), nội dung của file như sau:

```
cha( hoan, nam ).              % cha cua hoan la nam
cha( duong, hoan ).
me( duong, hoa ).
chame( X, Y ) :- cha( X, Y ).
chame( X, Y ) :- me( X, Y ).
```

```
ongba( X, Y ) :- chame( X, Z ), chame( Z, Y ).
```

Trong môi trường SWI, chúng ta load file chương trình này (File → Consult → *giapha.pl*), sau đó chúng ta nhập các câu truy vấn từ dấu nhắc của SWI Prolog. Ví dụ các câu truy vấn và trả lời truy vấn như sau:

```
1?-chame(duong,hoa).
true

2?-ongba(X,nam).
X=duong
```

Trong các phần tiếp theo, chúng ta sẽ tìm hiểu các câu khai báo trong file chương trình và các loại câu truy vấn.

2. Ngôn ngữ Prolog cơ bản, chương trình Prolog

Qui ước đặt tên biến và tên hằng:

Prolog là ngôn ngữ cho máy tính, vì vậy nó cần một qui ước rất quan trọng trong việc đặt tên biến và tên hằng, theo đó, tên một biến phải bắt đầu bằng ký tự in hoa (chẳng hạn X, Sinhvien, v.v.), còn tên hằng phải bắt đầu bằng ký tự in thường (ví dụ: an, binh, lasinhvien, v.v.). Vì Prolog là ngôn ngữ các câu Horn trong logic vị từ cấp một nên các biến chỉ xuất hiện trong các hạng thức là tham số của các vị từ.

Chương trình Prolog, các câu Horn dương:

Chương trình prolog về cơ bản là dãy (hội) các câu Horn dương (câu tuyển có đúng 1 literal dương). Các câu này có dạng Horn dương trong prolog có dạng tổng quát như sau:

head:- p_1, p_2, …, p_n.

{nghĩa là: if (p_1 and p_2 and … and p_n) then head}

Ở đây *head*, P_1, p_2, …, p_n là các vị từ (có thể có các tham số); vị từ head gọi là phần đầu của luật, còn P_1, p_2, …, p_n gọi phần thân (phần điều kiện) của luật. Nếu n>0 thì câu Horn dương trên là câu dạng luật; còn nếu n=0 thì câu không có phần điều kiện, khi này ta có câu mô tả sự kiện và có thể viết đơn giản là:

```
head.
```

Chú ý: các câu trong chương trình prolog đều kết thúc bởi dấu chấm (“.”). Tất cả các câu đều là câu đóng, nếu có ký hiệu biến xuất hiện trong câu thì ta ngầm hiểu rằng biến đó là biến buộc, đặt dưới lượng từ $\forall$, trừ các biến chỉ xuất hiện trong phần điều kiện của câu thì biến đó được hiểu là đặt dưới lượng từ $\exists$ (thực chất thì nếu chuyển dạng câu tuyển thì $\exists$ sẽ chuyển sang $\forall$ do chuyển vế và lấy phủ định)

Vị từ, hạng thức:

Như đã giới thiệu ở trên, chương trình prolog bao gồm hai loại câu: câu sự kiện (câu đơn) và câu luật (câu phức). Các câu này được xây dựng từ các vị từ ($head, P_1, p_2, \ldots, p_n$), mỗi vị từ có cú pháp như sau:

tên _vi_tu(hang_thuc$_1$, hang_thuc$_2$, …, hang_thuc$_n$)

trong đó *tên_vị_từ* tuân theo qui tắc đặt tên hằng; các *hạng_thức*$_i$ có thể là:

- ✓ Giá trị:
  - o tên hằng ký hiệu, ví dụ như *an*, *x*, *mauxanh*, v.v.
  - o hằng xâu, ví dụ ‘Xin chao’
  - o hằng số nguyên hoặc số thực, ví dụ như *5*, *3.1416*, v.v.
- ✓ tên biến, ví dụ như *X*, *Sinhvien*, v.v. (chú ý: tên biến bắt đầu bằng ký tự viết hoa; các biến đều không có kiểu biến, nó có thể nhận bất cứ một giá trị nào; tất cả các biến đều là biến địa phương trong câu nó xuất hiện)
- ✓ cấu trúc (nhóm các hạng thức lại thành cấu trúc), ví dụ như: mau[red, green, blue], [march, 17, 2011], v.v. Hai trường hợp đặc biệt của cấu trúc là list và string sẽ được tìm hiểu sâu hơn ở các phần sau của Chương này.

Ví dụ về chương trình Prolog: chương trình lưu trong file *giapha.pl* của ví dụ trước bao gồm ba câu đầu là các câu sự kiện và 2 câu cuối là câu luật; có 4 ký hiệu vị từ là: *cha*, *me*, *chame*, *ongba*; có 4 tên hằng: *nam*, *hoan*, *hoa*, *duong*; có 3 biến: *X*,*Y*,*Z*.

3. Câu truy vấn

Câu truy vấn tổng quát có dạng tổng quát như sau:

$p_1, p_2, ..., p_n$.

Chúng ta chia câu truy vấn thành hai loại:

- ✓ Câu truy vấn không chứa biến: khi đó câu truy vấn có nghĩa là "biểu thức logic *(p_1 and p_2 and ...and p_n)* có là đúng (có giá trị true) trong cơ sở tri thức (chương trình prolog) đã cho hay không?". Chẳng hạn, câu truy vấn *chame(duong, hoa)* trong ví dụ ở Phần 1 là hỏi: "có phải *hoa* là *chame* của *duong* không?". Trong trường hợp này, SWI Prolog sẽ trả lời là *true* hoặc *false*.
- ✓ Câu truy vấn có chứa tập các biến (ví dụ X,Y, …): khác với các câu trong cơ sở tri thức (chương trình prolog) mà ở đó mặc định hiểu rằng các biến là đi với lượng từ $\forall$, các biến trong câu truy vấn lại ngầm định đi với lượng từ $\exists$, khi đó câu truy vấn có nghĩa là: "có $\exists$ X, Y, … sao cho biểu thức logic *(p_1 and p_2 and ...and p_n)* có là đúng (có giá trị true) không?". Chẳng hạn, câu truy vấn *ongba(X,nam)* trong ví dụ ở Phần 1 là hỏi: "có tồn tại *X* mà có *nam* là *ongba* của *X* không?". Trong trường hợp này SWI Prolog sẽ tìm một giá trị X sao cho ongba(X,nam) có giá trị là true. Nếu chúng ta muốn SWI Prolog tìm tất cả các giá trị X thỏa mãn ongba(X,nam), sau mỗi trả lời của Hệ thống, chúng ta ấn phím ";" thay vì ấn phím "enter".

Chú ý: câu truy vấn *q* trong Prolog có dạng như trên sẽ tương đương $\neg q$ là câu dạng Horn âm $\forall x,y,... (\neg p_1 \vee \neg p_2 \vee ... \vee \neg p_n)$. (câu tuyển không có literal dương nào).

4. Vị từ phi logic (câu phi logic)

Chương trình là dãy (thứ tự là quan trọng!) các câu sự kiện và câu luật có cú pháp như định nghĩa như ở trên. Ngoài các câu chuẩn Horn trong logic vị từ cấp một này (các câu khai báo tri thức), Prolog còn có các vị từ phi logic để điều khiển việc thực hiện suy diễn hoặc để vào/ra dữ liệu như trong các ngôn ngữ thủ tục. Ví dụ về các câu phi logic là (mặc dù là các câu phi logic, nhưng Prolog vẫn gán giá trị hằng đúng cho chúng):

```
write(hang_thuc).        % lệnh này in hang_thuc ra màn hình
nl.                      % đưa con trỏ màn hình xuống dòng mới
read(ten_bien).          % nhập giá trị từ bàn phím vào biến
X is bieu-thuc.          % gán giá trị bieu-thuc cho biến X
```

Ví dụ, chương trình Hello.pl có nội dung như sau:

```
xinchao:-write('What is your name?'), nl, read(X), write('Hello '), write(X).
```

Sau khi load chương trình Hello.pl và chạy chương trình (câu truy vấn) thì được kết quả sau:

```
1 ?- xinchao.
What is your name?
|: hoan.                % chú ý: kết thúc nhập dữ liệu bằng dấu chấm (".")
Hello hoan
true.
```

Trong các phần sau của Chương này, chúng ta sẽ gặp thêm một số câu phi logic khác nữa như câu lệnh cắt (!).

5. *Trả lời truy vấn, quay lui, cắt, phủ định*

Trả lời truy vấn – quay lui:

Để tìm hiểu các chương trình Prolog được thực thi như thế nào (trình biên dịch Prolog trả lời các câu truy vấn thế nào), chúng ta tìm hiểu ví dụ sau:

Bài toán là viết chương trình Prolog tìm số lớn nhất trong hai số. Chúng ta soạn thảo file chương trình *timsolonnhat.pl* với vị từ *bigger(N,M)* để in ra số lớn nhất như sau:

```
bigger(N,M):- N < M, write('The bigger number is '), write(M).

bigger(N,M):- N > M, write('The bigger number is '), write(N).
```

```
bigger(N,M):- N =:= M, write('Numbers are the same').
```

Sau khi load chương trình, chúng ta nhập các câu truy vấn sau (câu trả lời truy vấn xuất hiện sau mỗi truy vấn):

```
1 ?- bigger(3,5).

The bigger is 5
true.b
2 ?- bigger(8,7).

The bigger is 8
true.
3 ?- bigger(10,10).

Numbers are the same
true.
```

Để trả lời các câu truy vấn ở trên, SWI Prolog sẽ thực hiện đồng nhất câu truy vấn với các vị từ là phần đầu các luật theo thứ tự từ trên xuống dưới. Khi gặp luật có thể đồng nhất được, SWI Prolog sẽ thực hiện đồng nhất câu truy vấn với phần đầu của luật và thực hiện các lệnh trong phần thân của luật. Nếu tất cả các biến trong luật (sau khi đồng nhất) đều đã xác định được giá trị thì SWI Prolog sẽ trả về cho người dùng kết quả true và đợi tương tác với người dùng. Khi người dung muốn tìm kết quả tiếp theo, nhấn phím ";", SWI Prolog sẽ chuyển sang tìm, đồng nhất và thực hiện các luật tiếp theo.

Khi câu truy vấn đồng nhất được với một luật mà có một biến nào đó vẫn còn chưa xác định được giá trị, SWI Prolog sẽ hình thành các câu truy vấn mới là các vị từ còn chứa biến; sau đó thực hiện đệ qui việc tìm, đồng nhất và thực hiện các luật trong cơ sở tri thức theo thứ tự từ trên đối với các câu truy vấn trung gian này (đích trung gian). Việc thực hiện suy diễn lùi như thế này còn gọi là quay lui.

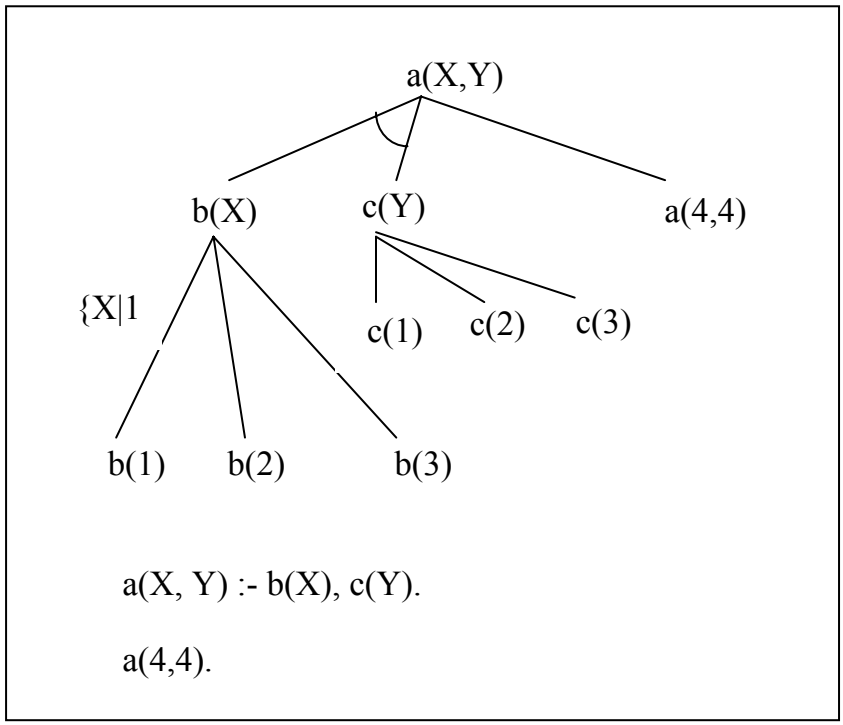
Một điểm lưu ý nữa, sau khi tìm được luật đồng nhất với câu truy vấn, SWI Prolog sẽ thực hiện phần thân của luật theo thứ tự từ trái qua phải. Vì phần thân của luật có

dạng hội các vị từ, nên khi thực hiện, nếu gặp một vị từ mà có giá trị chân lý là false thì SWI Prolog sẽ không thực hiện các vị từ sau đó.

Vị từ Cắt (!):

Khi thực hiện chương trình, SWI Prolog thực hiện từ trên xuống, từ trái qua phải, và chứng minh câu truy vấn bằng quay lui (lùi). Khi tìm được một lời giải của câu truy vấn, SWI Prolog sẽ thực hiện quay lui vét cạn để tìm lời giải tiếp theo. Trong trường hợp chúng ta chỉ cần tìm 1 lời giải, hoặc trong trường hợp chúng ta biết chắc chắn không có lời giải khi thực hiện quay lui, ta có thể đặt vị từ cắt (!) ở sau danh sách các vị từ mong muốn. Khi có vị từ cắt xuất hiện trong một câu thì SWI Prolog sẽ không thực hiện quay lui đối với các vị từ đặt trước nó. Để hiểu cơ chế ngắt quay lui của vị từ cắt (!), ta lấy ví dụ sau:

```
a(X, Y) :- b(X), c(Y).
a(4,4).
b(1).
b(2).
b(3).
c(1).
c(2).
c(3).
```



Khi thực hiện truy vấn:

```
1 ?- a(X,Y).
```

thì được kết quả như sau:

```
1 ?- a(X,Y).
X = 1,
```

Y = 1 ;

X = 1,

Y = 2 ;

X = 1,

Y = 3 ;

X = 2,

Y = 1 ;

X = 2,

Y = 2 ;

X = 2,

Y = 3 ;

X = 3,

Y = 1 ;

X = 3,

Y = 2 ;

X = 3,

Y = 3.

Bây giờ chúng ta sẽ thay thế câu lệnh đầu tiên trong chương trình

a(X, Y) :- b(X), c(Y).

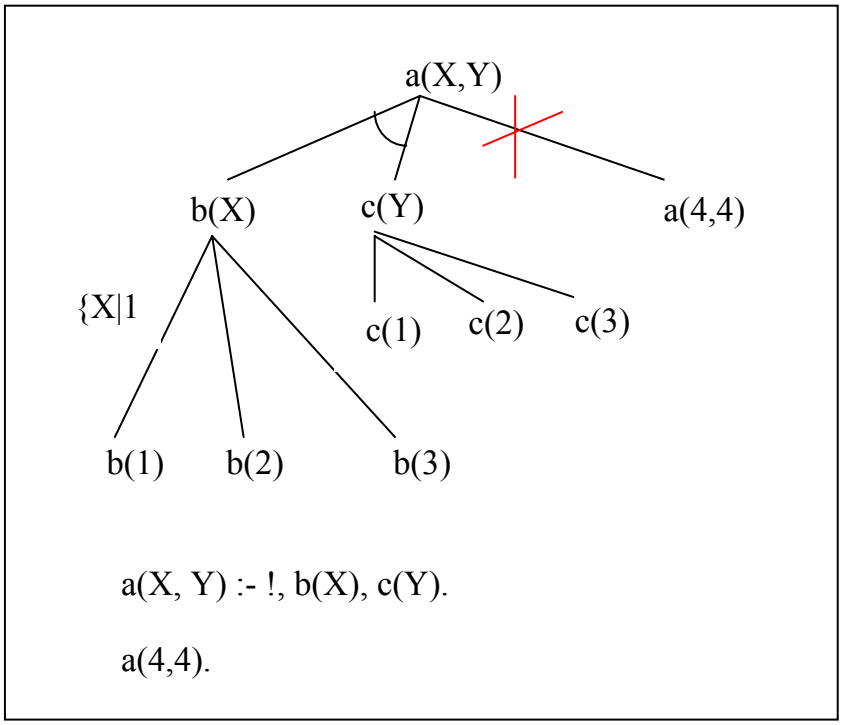
bằng một trong các câu lệnh dưới đây (chèn vị từ ngắt ! vào các vị trí khác nhau):

a(X, Y) :- !, b(X), c(Y). % không quay lui đối với vị từ a

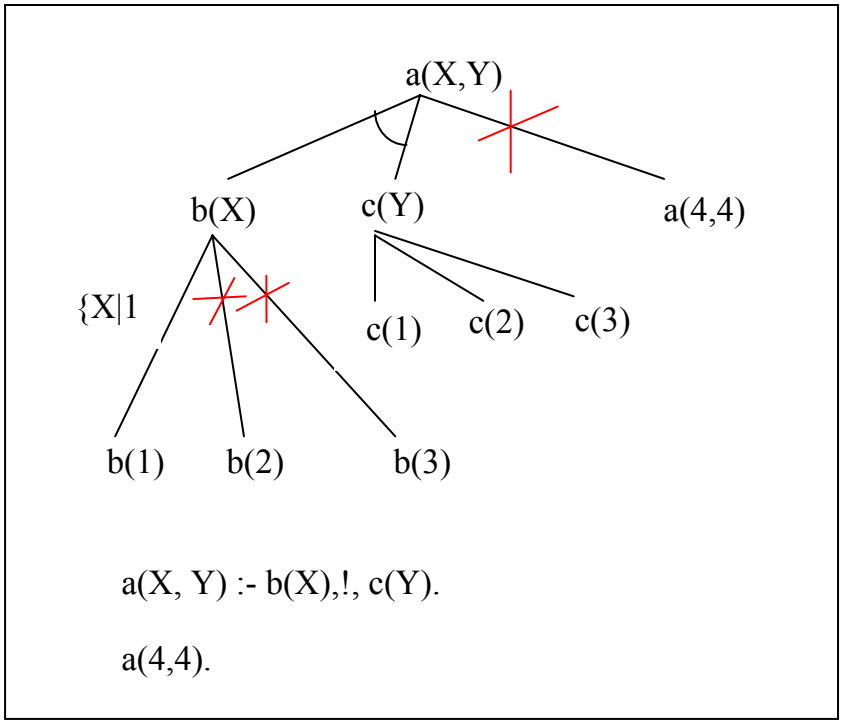
a(X, Y) :- b(X),!, c(Y). % không quay lui đối với vị từ a,b

a(X, Y) :- b(X), c(Y),!. % không quay lui đối với vị từ a,b,c

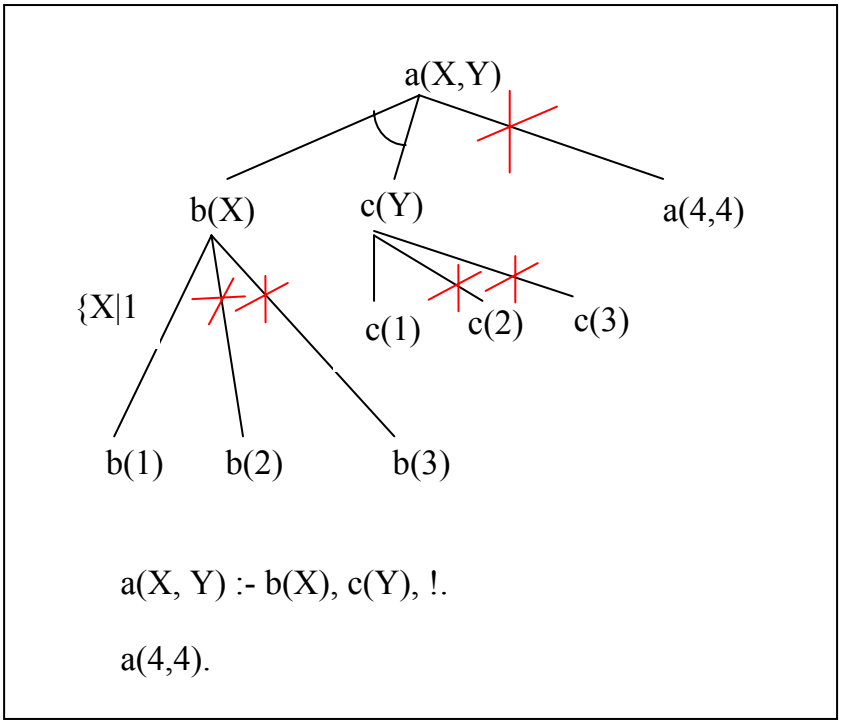
Và thực hiện lại câu truy vấn thì ta sẽ được các kết quả khác nhau như trong các hình vẽ sau.




```
a(X, Y) :- !, b(X), c(Y).
a(4,4).
```




```
a(X, Y) :- b(X),!, c(Y).
a(4,4).
```



Vị từ phủ định:

Trong SWI Prolog, vị từ not(X) có giá trị *true* khi SWI không chứng minh được X. Hay nói cách khác, những sự kiện mà SWI không chứng minh được là true thì SWI sẽ cho là sự kiện đó là false (giả thuyết đóng). Ví dụ, cho chương trình logic như sau:

```
lacontrai( binh).                % binh la con trai
lacontrai( an).
khonglacontrai( X) :- not (lacontrai(X)).
```

Nếu ta thực hiện các câu truy vấn:

```
1 ? - khonglacontrai(X).
false
```

vì SWI không tìm được đối tượng nào làm cho vị từ khonglacontrai(X) là đúng. Nhưng khi chúng ta thực hiện truy vấn sau:

```
2 ? - khonglacontrai(thanh).
```

true

kết quả cho là true vì SWI không chứng minh được lacontrai(thanh).

Vị từ not có tác dụng trong một số trường hợp, chẳng hạn bài toán kiểm tra xem một số có là số nguyên tố không, tức là số mà không chia hết cho các số nhỏ hơn nó (trừ số 1 và chính nó). Bài toán này độc giả có thể xem ở phần cuối chương này.

6. *Vị từ đệ qui*

Vị từ đệ quy là vị từ xuất hiện trong cả phần đầu và phần than của luật, hay nói cách khác, vị từ gọi chính nó. Định nghĩa vị từ đệ qui bao giờ cũng có 2 phần, phần sự kiện và phần đệ qui. Ví dụ, chương trình sau định nghĩa vị từ fibonaci(N,X) để tính phần từ thứ N trong dãy fibonaci, kết quả đưa vào biến X (dãy Fibonaci là dãy có phần tử thứ nhất bằng 0, phần tử thứ hai bằng 1, phần tử thứ ba trở đi sẽ là tổng của hai phần tử liền ngay trước).

```
fibonaci( 1,0).          % phần tử đầu tiên là 0
fibonaci( 2,1).          % phần tử thứ đầu tiên là 1
fibonaci( N,F) :- N>2, N1 is N-1, N2 is N-2, fibonaci(N1,F1), fibonaci(N2,F2), F is F1+F2.
```

Truy vấn chương trình logic này với các tham số N khác nhau ta sẽ được kết quả lưu trên biến F là phần tử thứ N của dãy. Ví dụ:

```
1 ? - fibonaci(3,F).
F=1
2 ? - fibonaci(4,F).
F=2
3 ? - fibonaci(10,F).
F=34
```

Chú ý: Vị từ fibonaci(N,F) ở trên là để định nghĩa phần tử thứ N của dãy Fibonaci và kết quả lưu trong F, vì vậy mà SWI chỉ có thể thực hiện các câu truy vấn mà ở đó tham số thứ nhất là hằng số, ví dụ câu truy vấn như fibonaci(10,F) để tìm phần tử thứ 10 của dãy; câu truy vấn như fibonaci(10,34) để kiểm tra xem phần tử thứ 10 của dãy có là 34 không; câu truy vấn fibonaci(N,34) sẽ không thực hiện được trên SWI!

7. *Cấu trúc dữ liệu trong Prolog*

Danh sách:

Danh sách là một cấu trúc dữ liệu được tạo dựng sẵn trong SWI Prolog và cũng đã có sẵn các phép toán để lấy phần tử đầu và phần đuôi danh sách. Danh sách là nhóm bất kỳ các hạng thức với nhau bằng dấu "[" và "]" và phân cách bởi dấu ",". Ví dụ [a,b,c,d] là danh sách gồm 4 phần tử. Thao tác cơ bản để thao tác với danh sách là tách phần tử đầu của danh sách. Ví dụ:

```
1 ? – [X|Y]=[a,b,c,d].
X=a,
Y=[b,c,d]
2 ? – [X,Y|Z]=[a,b,c,d].
X=a,
Y=b,
Z=[c,d]
3 ? – [X,[Y|Z]]=[a,b,c,d].
X=a,
Y=b,
Z=[c,d]
```

Ngoài thao tác cơ bản ở trên, SWI cũng đã xây dựng một số thao tác khác, ví dụ:

```
4 ? – member(b,[a,b,c,d]).     % b có phải là phần tử của danh sách [a,b,c,d] không?
true
5 ? – append([a,b,c],[d,e,f],X).          % nối hai danh sách
X = [a, b, c, d, e, f]
```

Để hiểu rõ thêm về danh sách, chúng ta xét ví dụ sau: hãy viết chương trình đảo ngược danh sách.

```
my_reverse([],[]).

my_reverse([H|T],L):- my_reverse(T,R),append(R,[H],L).
```

Câu truy vấn có thể là:

```
1 ? – my_reverse([a,b,c,d],Y).
Y=[d,c,b,a]
```

Ví dụ tiếp theo là sắp xếp danh sách theo thứ tự tăng dần. Để giải bài toán này, chúng ta sẽ xây dựng vị từ có hai tham số sapxep(X,Y), với X là danh sách cần sắp xếp, Y là kết quả danh sách đã sắp xếp. Trong ví dụ dưới đây, ta sử dụng giải thuật sắp xếp theo kiểu chèn, sử dụng biến trung gian

```
sapxep (X,Y):-i_sort(X,[],Y).

i_sort([],Y,Y).

i_sort([H|T],Z,Y):-insert(H,Z,Y1),i_sort(T,Y1,Y).

insert(X,[Y|T],[Y|NT]):-X>Y,insert(X,T,NT).

insert(X,[Y|T],[X,Y|T]):-X=<Y.

insert(X,[],[X]).
```

8. Thuật toán suy diễn trong Prolog

Chương 7 – Lập luận với tri thức không chắc chắn

Trong các chương trước, chúng ta đã tìm hiểu logic mệnh đề, logic vị từ cấp một, và prolog. Ngôn ngữ và ngữ nghĩa của các logic này chỉ giới hạn cho các câu đúng/sai. Trong thực tế, nhiều thông tin/tri thức chúng ta không hoàn toàn biết được nó là đúng hay sai và chúng ta vẫn có thể rút ra (lập luận ra) các thông tin/tri thức từ những điều ta không chắc chắn đó mặc dù các thông tin/tri thức rút ra cũng là những cái không chắc chắn.

Một ví dụ về việc lập luận với các thông tin không chắc đúng và với kết luận cũng không chắc đúng như sau. Giả sử chúng ta đã biết (qua quan sát 100 ngày gần đây) về các hoạt động của anh A với các điều kiện thời tiết khác nhau. Trong số 100 ngày, có 70 ngày trời nắng và không có gió. Anh ấy không đi chơi golf vào các ngày có gió hoặc không nắng. Trong 70 ngày nắng và không có gió thì anh ấy chỉ đi chơi golf trong 50 ngày. Việc đi chơi golf hay không phụ thuộc vào thời tiết, đôi khi đơn giản cũng chỉ vì hôm đó anh có thích hay không. Bây giờ dựa vào nhiều điều đã biết này, chúng ta phải trả lời các câu hỏi như: "ngày mai anh ấy có đi chơi golf không nếu biết rằng dự báo thời thiết ngày mai trời có thể có mưa?", hoặc "khả năng ngày mai anh ấy đi chơi golf là bao nhiêu?", hoặc là nếu biết anh ấy không đi chơi golf thì thời tiết hôm đó thế nào?", v.v. Rõ rang các thông tin/tri thức đã biết là không chắc chắn và câu truy vấn thì trả lời cũng có thể không phải là dạng chắc chắc.

Vậy làm thế nào mà máy tính có thể biểu diễn được các thông tin/tri thức không chắc chắn và lập luận để trả lời các câu truy vấn như trên. Có ba cách tiếp cận để giải quyết vấn đề biểu diễn và suy diễn các thông tin và tri thức không chắc chắn: logic mờ, lý thuyết khả năng và lý thuyết xác suất. Trong chương này, chúng ta chỉ tìm hiểu về lý thuyết xác suất, một ngôn ngữ để biểu diễn các thông tin, tri thức không chắc chắn và lý thuyết xác suất cho phép chúng ta lập luận để rút ra các thông tin và tri thức mới.

Chương 8 – Học mạng nơron nhân tạo

Hệ thống được gọi là có khả năng học (có dáng vẻ học như con người) là hệ thống có khả năng tìm ra một sự khái quát hoặc mô hình cho các dữ liệu huấn luyện (dữ liệu có gán nhãn nhận diện hoặc phân loại). Đặc trưng khái quát hoặc mô hình đó có thể được sử dụng để nhận diện hoặc phân loại dữ liệu mới. Hệ thống học thông minh là hệ thống có dáng vẻ ứng xử (hoặc kết quả nhận diện hoặc kết quả dự đoán) như đứa trẻ con học; chúng quan sát các hình ảnh của các ký tự đã được phân loại (thông qua việc nói với chúng đấy là ký tự gì - dữ liệu huấn luyện), và khái quát các đặc trưng của các loại ký tự; khi đưa hình ảnh của ký tự mới (dữ liệu kiểm tra) vào thì chúng nhận diện hoặc phân loại được ký tự đó thuộc loại nào. Hệ thống thông minh là hệ thống nhận diện đúng hoặc phân loại đúng dữ liệu kiểm tra, và khi đó hệ thống được gọi là có khả năng học (hay có dáng vẻ học).