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 1 – Giới thiệu

1. Trí tuệ nhân tạo là gì?

Để hiểu trí tuệ nhân tạo (artificial intelligence) là gì chúng ta bắt đầu với khái niệm sự bay nhân tạo (flying machines), tức là cái máy bay.

Đã từ lâu, loài người mong muốn làm ra một cái máy mà có thể di chuyển được trên không trung mà không phụ thuộc vào địa hình ở dưới mặt đất, hay nói cách khác là máy có thể bay được. Không có gì ngạc nhiên khi những ý tưởng đầu tiên làm máy bay là từ nghiên cứu cách con chim bay. Những chiếc máy biết bay được thiết kế theo nguyên lý "vỗ cánh" như con chim chỉ có thể bay được quãng đường rất ngắn và lịch sử hàng không thực sự sang một trang mới kể từ anh em nhà Wright thiết kế máy bay dựa trên các nguyên lý của khí động lực học (aerodynamics).

Các máy bay hiện nay, như đã thấy, có sức trở rất lớn và bay được quãng đường có thể vòng quanh thế giới. Nó không nhất thiết phải có nguyên lý bay của con chim nhưng vẫn bay được như chim (dáng vẻ), và còn tốt hơn chim.

Quay lại câu hỏi Trí tuệ nhân tạo là gì. Trí tuệ nhân tạo là trí thông minh của máy do con người tạo ra. Ngay từ khi chiếc máy tính điện tử đầu tiên ra đời, các nhà khoa học máy tính đã hướng đến phát hiển hệ thống máy tính (gồm cả phần cứng và phần mềm) sao cho nó có khả năng thông minh như loài người. Mặc dù cho đến nay, theo quan niệm của người viết, ước mơ này vẫn còn xa mới thành hiện thực, tuy vậy những thành tựu đạt được cũng không hề nhỏ: chúng ta đã làm được các hệ thống (phần mềm chơi cờ vua chạy trên siêu máy tinh GeneBlue) có thể thắng được vua cờ thế giới; chúng ta đã làm được các phần mềm có thể chứng minh được các bài toán hình học; v.v. Hay nói cách khác, trong một số lĩnh vực, máy tính có thể thực hiện tốt hơn hoặc tương đương con người (tất nhiên không phải tất cả các lĩnh vực). Đó chính là các hệ thống thông minh.

Có nhiều cách tiếp cận để làm ra trí thông minh của máy (hay là trí tuệ nhân tạo), chẳng hạn là nghiên cứu cách bộ não người sản sinh ra trí thông minh của loài người như

thế nào rồi ta bắt chước nguyên lý đó, nhưng cũng có những cách khác sử dụng nguyên lý hoàn toàn khác với cách sản sinh ra trí thông minh của loài người mà vẫn làm ra cái máy thông minh như hoặc hơn người; cũng giống như máy bay hiện nay bay tốt hơn con chim do nó có cơ chế bay không phải là giống như cơ chế bay của con chim.

Như vậy, trí tuệ nhân tạo ở đây là nói đến khả năng của máy khi thực hiện các công việc mà con người thường phải xử lý; và khi dáng vẻ ứng xử hoặc kết quả thực hiện của máy là tốt hơn hoặc tương đương với con người thì ta gọi đó là máy thông minh hay máy đó có trí thông minh. Hay nói cách khác, đánh giá sự thông minh của máy không phải dựa trên nguyên lý nó thực hiện nhiệm vụ đó có giống cách con người thực hiện hay không mà dựa trên kết quả hoặc dáng vẻ ứng xử bên ngoài của nó có giống với kết quả hoặc dáng vẻ ứng xử của con người hay không.

Các nhiệm vụ của con người thường xuyên phải thực hiện là: ***giải bài toán*** (tìm kiếm, chứng minh, lập luận), ***học***, ***giao tiếp***, ***thể hiện cảm xúc***, ***thích nghi với môi trường xung quanh***, v.v., và dựa trên kết quả thực hiện các nhiệm vụ đó để kết luận rằng một ai đó có là thông minh hay không. Môn học Trí tuệ nhân tạo nhằm cung cấp các phương pháp luận để làm ra hệ thống có khả năng thực hiện các nhiệm vụ đó: giải toán, học, giao tiếp, v.v. bất kể cách nó làm có như con người hay không mà là kết quả đạt được hoặc dáng vẻ bên ngoài như con người.

Trong môn học này, chúng ta sẽ tìm hiểu các phương pháp để làm cho máy tính biết cách giải bài toán, biết cách lập luận, biết cách học, v.v.

2. Lịch sử

Vào năm 1943, Warren McCulioch và Walter Pitts bắt đầu thực hiện nghiên cứu ba cơ sở lý thuyết cơ bản: triết học cơ bản và chức năng của các noron thần kinh; phân tích các mệnh đề logic; và lý thuyết dự đoán của Turing. Các tác giả đã nghiên cứu đề xuất mô hình noron nhân tạo, mỗi noron đặc trưng bởi hai trạng thái "bật", "tắt" và phát hiện mạng noron có khả năng học.

Thuật ngữ “Trí tuệ nhân tạo” (Artificial Intelligence - AI) được thiết lập bởi John McCarthy tại Hội thảo đầu tiên về chủ đề này vào mùa hè năm 1956. Đồng thời, ông cũng đề xuất ngôn ngữ lập trình Lisp – một trong những ngôn ngữ lập trình hàm tiêu biểu, được sử dụng trong lĩnh vực AI. Sau đó, Alan Turing đưa ra "Turing test" như là một phương pháp kiểm chứng hành vi thông minh.

Thập kỷ 60, 70 Joel Moses viết chương trình Macsyma - chương trình toán học sử dụng cơ sở tri thức đầu tiên thành công. Marvin Minsky và Seymour Papert đưa ra các chứng minh đầu tiên về giới hạn của các mạng nơ-ron đơn giản. Ngôn ngữ lập trình logic Prolog ra đời và được phát triển bởi Alain Colmerauer. Ted Shortliffe xây dựng thành công một số hệ chuyên gia đầu tiên trợ giúp chẩn đoán trong y học, các hệ thống này sử dụng ngôn ngữ luật để biểu diễn tri thức và suy diễn.

Vào đầu những năm 1980, những nghiên cứu thành công liên quan đến AI như các hệ chuyên gia (expert systems) – một dạng của chương trình AI mô phỏng tri thức và các kỹ năng phân tích của một hoặc nhiều chuyên gia con người

Vào những năm 1990 và đầu thế kỷ 21, AI đã đạt được những thành tựu to lớn nhất, AI được áp dụng trong logic, khai phá dữ liệu, chẩn đoán y học và nhiều lĩnh vực ứng dụng khác trong công nghiệp. Sự thành công dựa vào nhiều yếu tố: tăng khả năng tính toán của máy tính, tập trung giải quyết các bài toán con cụ thể, xây dựng các mối quan hệ giữa AI và các lĩnh vực khác giải quyết các bài toán tương tự, và một sự chuyển giao mới của các nhà nghiên cứu cho các phương pháp toán học vững chắc và chuẩn khoa học chính xác.

3. Các lĩnh vực của AI

- *Lập luận, suy diễn tự động:* Khái niệm lập luận (reasoning), và suy diễn (reference) được sử dụng rất phổ biến trong lĩnh vực AI. Lập luận là suy diễn logic, dùng để chỉ một tiến trình rút ra kết luận (tri thức mới) từ những giả thiết đã cho (được biểu diễn dưới dạng cơ sở tri thức). Như vậy, để thực hiện lập luận người ta cần có các phương pháp lưu trữ cơ sở tri thức và các thủ tục lập luận trên cơ sở tri thức đó.

- *Biểu diễn tri thức:* Muốn máy tính có thể lưu trữ và xử lý tri thức thì cần có các phương pháp biểu diễn tri thức. Các phương pháp biểu diễn tri thức ở đây bao gồm các ngôn ngữ biểu diễn và các kỹ thuật xử lý tri thức. Một ngôn ngữ biểu diễn tri thức được đánh giá là "tốt" nếu nó có tính biểu đạt cao và các tính hiệu quả của thuật toán lập luận trên ngôn ngữ đó. Tính biểu đạt của ngôn ngữ thể hiện khả năng biểu diễn một phạm vi rộng lớn các thông tin trong một miền ứng dụng. Tính hiệu quả của các thuật toán lập luận thể hiện chi phí về thời gian và không gian dành cho việc lập luận. Tuy nhiên, hai yếu tố này dường như đối nghịch nhau, tức là nếu ngôn ngữ có tính biểu đạt cao thì thuật toán lập luận trên đó sẽ có độ phức tạp lớn (tính hiệu quả thấp) và ngược lại (ngôn ngữ đơn giản, có tính biểu đạt thấp thì thuật toán lập luận trên đó sẽ có hiệu quả cao). Do đó, một thách thức lớn trong lĩnh vực AI là xây dựng các ngôn ngữ biểu diễn tri thức mà có thể cân bằng hai yếu tố này, tức là ngôn ngữ có tính biểu đạt đủ tốt (tùy theo từng ứng dụng) và có thể lập luận hiệu quả.
- *Lập kế hoạch*: khả năng suy ra các mục đích cần đạt được đối với các nhiệm vụ đưa ra, và xác định dãy các hành động cần thực hiện để đạt được mục đích đó.
- *Học máy*: là một lĩnh vực nghiên cứu của AI đang được phát triển mạnh mẽ và có nhiều ứng dụng trong các lĩnh vực khác nhau như khai phá dữ liệu, khám phá tri thức,…
- *Xử lý ngôn ngữ tự nhiên*: là một nhánh của AI, tập trung vào các ứng dụng trên ngôn ngữ của con người. Các ứng dụng trong nhận dạng tiếng nói, nhận dạng chữ viết, dịch tự động, tìm kiếm thông tin,…
- *Hệ chuyên gia*: cung cấp các hệ thống có khả năng suy luận để đưa ra những kết luận. Các hệ chuyên gia có khả năng xử lý lượng thông tin lớn và cung cấp các kết luận dựa trên những thông tin đó. Có rất nhiều hệ chuyên gia nổi tiếng như các hệ chuyên gia y học MYCIN, đoán nhận cấu trúc phân tử từ công thức hóa học DENDRAL, …
- *Robotics*

- …

4. Nội dung môn học

Giáo trình này được viết với các nội dung nhập môn về AI cho các sinh viên chuyên ngành Tin học và Công nghệ thông tin. Các tác giả có tham khảo một số tài liệu, giáo trình của các trường Đại học Quốc gia Hà nội, Đại học Bách khoa Hà nội, … Nội dung gồm các phần sau:

Chương 1. Giới thiệu: trình bày tổng quan về AI, lịch sử ra đời và phát triển và các lính vực ứng dụng của AI.

Chương 2. Các phương pháp tìm kiếm lời giải: trình bày các kỹ thuật tìm kiếm cơ bản được áp dụng để giải quyết các vấn đề và được áp dụng rộng rãi trong các lĩnh vực của trí tuệ nhân tạo.

Chương 3. Các giải thuật tìm kiếm lời giải cho trò chơi: trình bày một số kỹ thuật tìm kiếm trong các trò chơi có đối thủ.

Chương 4. Các phương pháp lập luận trên logic mệnh đề: trình bày cú pháp, ngữ nghĩa của logic mệnh đề và một số thuật toán lập luận trên logic mệnh đề.

Chương 5. Các phương pháp lập luận trên logic vị từ cấp một: trình bày cú pháp, ngữ nghĩa của logic vị từ cấp một và một số thuật toán lập luận cơ bản trên logic vị từ cấp một.

Chương 6. Prolog: Giới thiệu chung về ngôn ngữ Prolog, cú pháp, ngữ nghĩa và cấu trúc chương trình trong Prolog, một số phiên bản mới của Prolog như SWI Prolog,…

Chương 7. Lập luận với tri thức không chắc chắn: Giới thiệu về tri thức không chắc chắn và một số cách tiếp cận biểu diễn và xử lý tri thức không chắc chắn.

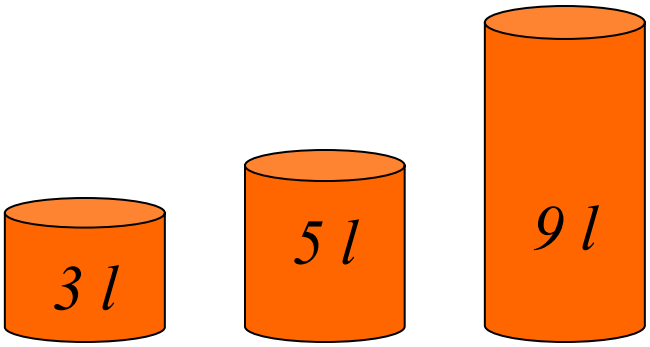
Chương 8. Học mạng noron nhân tạo: Giới thiệu về phương pháp và các kỹ thuật cơ bản trong lập luận sử dụng mạng noron nhân tạo.

Chương 2 – Bài toán và phương pháp tìm kiếm lời giải

1. Bài toán và các thành phần của bài toán

Chương này giới thiệu các giải thuật máy tính có thể giải các bài toán mà thông thường đòi hỏi trí thông minh của con người, như bài toán đong nước, bài toán 8 sô trên bàn cờ, bài toán tìm đường như mô tả bên dưới đây. Để thiết kế giải thuật chung giải các bài toán này, chúng ta nên phát biểu bài toán theo dạng 5 thành phần: Trạng thái bài toán, trạng thái đầu, trạng thái đích, các phép chuyển trạng thái, lược đồ chi phí các phép chuyển trạng thái (viết gọn là chi phí).

a. Bài toán đong nước



Sử dụng ba can 3 lít, 5 lít và 9 lít, làm thế nào để đong được 7 lít nước.

Bài toán này được phát biểu lại theo 5 thành phần như sau:

- Trạng thái: Gọi số nước có trong 3 can lần lượt là a, b, c ($a \le 3, b \le 5, c \le 9$), khi đó bộ ba (a, b, c) là trạng thái của bài toán
- Trạng thái đầu: (0, 0, 0) // cả ba can đều rỗng
- Trạng thái đích (-, -, 7) // can thứ 3 chứa 7 lít nước
- Phép chuyển trạng thái: từ trạng thái (a,b,c) có thể chuyển sang trạng thái (x,y,z) thông qua các thao tác như làm rỗng 1 can, chuyển từ can này sang can kia đến khi hết nước ở can nguồn hoặc can đích bị đầy.
- Chi phí mỗi phép chuyển trạng thái: mỗi phép chuyển trạng thái có chi phí là 1.

Một lời giải của bài toán là một dãy các phép chuyển trạng thái (đường đi) từ trạng thái đầu đến trạng thái đích. Bảng dưới đây là 2 lời giải của bài toán trên:

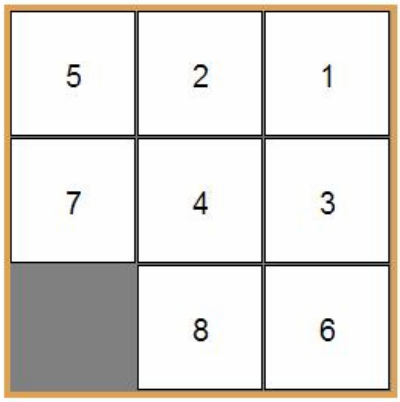
a	b	c		
0	0	0		
3	0	0		
0	0	3		
3	0	3		
0	0	6		
3	0	6		
0	3	6		
3	3	6		
1	5	6		
0	5	7	← Đích	

Lời giải 1 (chi phí: 9)

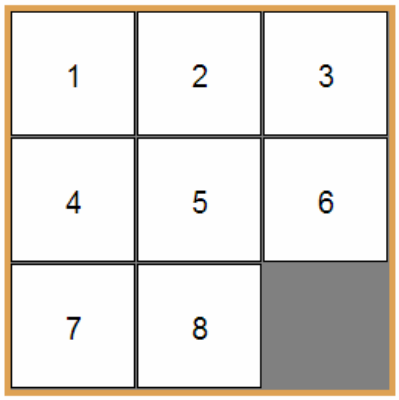
← Đầu →	a	b	c
	0	0	0
	0	5	0
	3	2	0
	3	0	2
	3	5	2
Đích →	3	0	7

Lời giải 2 (chi phí: 5)

b. Bài toán di chuyển 8 số trên bàn cờ



Trạng thái đầu



Trạng thái đích

Cho bàn cờ kích thước 3 x 3, trên bàn cờ có 8 quân cờ đánh số từ 1 đến 8 (hình vẽ). Trên bàn cờ có một ô trống. Chúng ta có thể chuyển một quân cờ có chung cạnh với ô trống sang ô trống. Hãy tìm dãy các phép chuyển để từ trạng thái ban đầu về trạng thái mà các quan cờ được xếp theo trật tự như Trạng thái đích của hình trên.

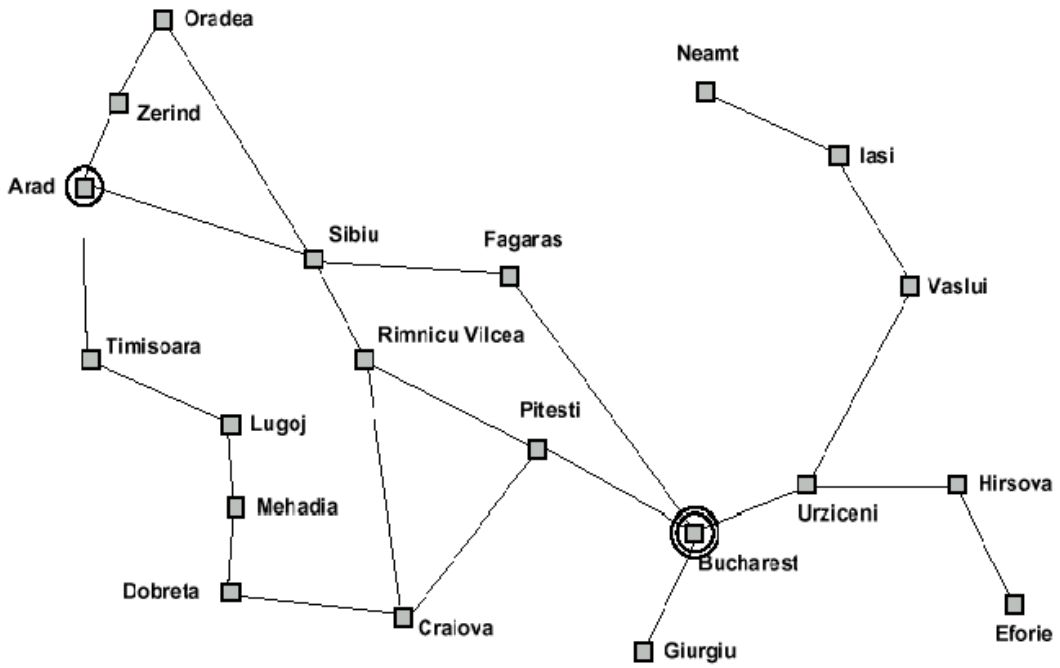
Bài toán di chuyển 8 số trên bàn cờ có thể phát biểu dưới dạng 5 thành phần như sau:

- Biểu diễn trạng thái: mảng 2 chiều kích thước 3x3, phần tử của mảng lưu số hiệu quân cờ (từ 0 đến 9, 0 là vị trí trống). Cũng có thể biểu diễn trạng thái bàn cờ bằng mảng một chiều gồm 9 phần tử: ba phần tử đầu tiên biểu diễn các ô thuộc dòng đầu tiên của bàn cờ, ba phần tử tiếp biểu diễn các quân cờ thuộc dòng thứ hai, ba phần tử cuối cùng biểu diễn các quân cờ thuộc dòng cuối cùng. Ở đây chúng tôi sử dụng mảng hai chiều 3x3 để cho giống với bàn cờ trên thực tế.
- Trạng thái đầu (hình vẽ trên)
- Trạng thái đích (hình vẽ trên)
- Phép chuyển trạng thái: đổi chỗ ô có số hiệu 0 với một trong các ô có cùng cạnh.
- Chi phí: mỗi phép chuyển có chi phí 1.

Lời giải của bài toán là dãy các phép chuyển từ trạng thái đầu đến trạng thái đích. Một lời giải của bài toán là: UP, UP, RIGHT, DOWN, LEFT, UP, RIGHT, RIGHT, DOWN,

LEFT, LEFT, UP, RIGHT, DOWN, RIGHT, DOWN (chú ý: up, down, right, left là biểu diễn sự dịch chuyển ô trống lên trên, xuống dưới, sang phải, sang trái)

c. Bài toán tìm đường đi



Một ôtô robot tìm đường đi từ thành phố Arad đến thành phố Bucharest. Biết rằng xe robot này không có bản đồ đầy đủ như trên hình vẽ trên, nhưng khi nó đến một thành phố mới, nó có bộ cảm biến đọc được biển chỉ đường đến các thành lân cận, trên biển chỉ đường có khoảng cách.

Bài toán tìm đường có thể phát biểu theo 5 thành phần như sau:

- Trạng thái: vị trí của ôtô robot (tên thành phố)
- Trạng thái đầu: Thành phố Arad
- Trạng thái đích: Thành phố Bucharest
- Phép chuyển trạng thái: từ thành phố sang thành phố lân cận
- Chi phí: khoảng cách giữa 2 thành phố trong phép chuyển trạng thái

Lời giải của bài toán là dãy các phép chuyển từ trạng thái đầu đến trạng thái đích, hay là đường đi từ thành phố đầu đến thành phố đích. Một ví dụ của lời giải bài toán là: Arad → Sibiu → Fagaras → Bucharest.

2. Giải thuật tổng quát tìm kiếm lời giải

a. Không gian trạng thái của bài toán

Mỗi bài toán với 5 thành phần như mô tả ở trên, chúng ta có thể xây dựng được một cấu trúc đồ thị với các nút là các trạng thái của bài toán, các cung là phép chuyển trạng thái. Đồ thị này được gọi là không gian trạng thái của bài toán. Không gian trạng thái có thể là vô hạn hoặc hữu hạn. Ví dụ, với bài toán di chuyển 8 số trên bàn cờ, không gian trạng thái có số lượng là 8! (8 giai thừa) trạng thái.

Lời giải của bài toán là một đường đi trong không gian trạng thái có điểm đầu là trạng thái đầu và điểm cuối là trạng thái đích. Nếu không gian trạng thái của bài toán là nhỏ, có thể liệt kê và lưu vừa trong bộ nhớ của máy tính thì việc tìm đường đi trong không gian trạng thái có thể áp dụng các thuật toán tìm đường đi trong lý thuyết đồ thị. Tuy nhiên, trong rất nhiều trường hợp, không gian trạng thái của bài toán là rất lớn, việc duyệt toàn bộ không gian trạng thái là không thể. Trong môn học Trí tuệ nhân tạo này, chúng ta sẽ tìm hiểu các phương pháp tìm kiếm lời giải trong các bài toán có không gian trạng thái lớn.

b. Giải thuật tổng quát tìm kiếm lời giải của bài toán

Với các bài toán có 5 thành phần ở trên, chúng ta có giải thuật chung để tìm kiếm lời giải của bài toán. Ý tưởng là sinh ra các lời giải tiềm năng và kiểm tra chúng có phải là lời giải thực sự của bài toán. Một lời giải tiềm năng là một đường đi trong không gian trạng thái của bài toán có nút đầu là trạng thái đầu và mỗi cung của đường đi là một phép chuyển hợp lệ giữa các trạng thái kề với cung đó. Lời giải thực sự của bài toán là lời giải tiềm năng có nút cuối cùng là trạng thái đích. Các lời giải tiềm năng là các đường đi có cùng nút đầu tiên và dãy các cung là dãy các phép chuyển hợp lệ từ trạng thái đầu đó. Các lời giải tiềm năng có thể tổ chức theo cây, gốc của cây là trạng thái đầu, cây được

phát triển bằng cách bổ sung vào các nút liền kề với trạng thái đầu, sau đó liên tiếp bổ sung vào các con của các nút lá, …

Lược đồ chung để tìm lời giải của bài toán 4 thành phần trên là xây dựng cây lời giải tiềm năng (hay là cây tìm kiếm) và kiểm tra lời giải tiềm năng có là lời giải thực sự của bài toán hay không. Các bước của giải thuật chung là như sau: xây dựng cây tìm kiếm mà nút gốc là trạng thái đầu, lặp lại 2 bước: kiểm tra xem trạng thái đang xét có là trạng thái đích không, nếu là trạng thái đích thì thông báo lời giải, nếu không thì mở rộng cây tìm kiếm bằng cách bổ sung các nút con là các trạng thái láng giềng của trạng thái đang xét. Giải thuật chung được trình bày trong bảng sau:

Đầu vào của giải thuật là bài toán (problem) với 5 thành phần (biểu diễn trạng thái tổng quát, trạng thái đầu, trạng thái đích, phép chuyển trạng thái, chi phí phép chuyển trạng thái) và một chiến lược tìm kiếm (strategy); đầu ra của giải thuật là một lời giải của bài toán hoặc giá trị failure nếu bài toán không có lời giải. Giải thuật sinh ra cây các lời giải tiềm năng, nút gốc là trạng thái đầu của bài toán, mở rộng cây theo chiến lược (strategy) đã định trước đến khi cây chứa nút trạng thái đích hoặc không thể mở rộng cây được nữa.

```
Function General_Search(problem, strategy) returns a solution, or failure
    cây-tìm-kiếm ← trạng-thái-đầu;
    while (1)
    {
      if (cây-tìm-kiếm không thể mở rộng được nữa) then return failure
      nút-lá ← Chọn-1-nút-lá(cây-tìm-kiếm, strategy)
      if (node-lá là trạng-thái-đích) then return Đường-đi(trạng-thái-đầu, nút-lá)
      else mở-rộng(cây-tìm-kiếm, các-trạng-thái-kề(nút-lá))
    }
```

Trong giải thuật chung này, chiến lược tìm kiếm (strategy) sẽ quyết định việc chọn nút lá nào trong số nút lá của cây để mở rộng cây tìm kiếm, ví dụ như nút lá nào xuất hiện trong cây sớm hơn thì được chọn trước để phát triển cây (đây là chiến lược tìm kiếm theo chiều

rộng), hoặc nút lá nào xuất hiện sau thì được chọn để mở rộng cây (đây là chiến lược tìm kiếm theo chiều sâu). Chiến lược tìm kiếm có thể được cài đặt thông qua một cấu trúc dữ liệu để đưa vào và lấy ra trạng thái lá của cây tìm kiếm. Hai cấu trúc dữ liệu cơ bản là hàng đợi và ngăn xếp. Hàng đợi sẽ lưu các trạng thái lá của cây và trạng thái nào được đưa vào hàng đợi trước sẽ được lấy ra trước, còn ngăn xếp là cấu trúc dữ liệu lưu trạng thái lá của cây tìm kiếm và việc chọn nút lá của cây sẽ theo kiểu vào trước ra sau. Bảng dưới đây là chi tiết hóa thuật toán tìm kiếm lời giải ở trên với chiến lược tìm kiếm được thể hiện thông qua cấu trúc dữ liệu hàng đợi (queue) hoặc ngăn xếp (stack). Trong giải thuật chi tiết hơn này, cây tìm kiếm được biểu diễn bằng mảng một chiều father, trong đó father(i) là chỉ nút cha của nút i. Thủ tục path(node,father) dùng để lần ngược đường đi từ trạng thái node về nút gốc (trạng thái đầu) (node được truyền giá trị là trạng thái đích khi thủ tục path được gọi).

```
Function General_Search(problem, Queue/Stack) returns a solution, or failure
    Queue/Stack ← make_queue/make_stack(make-node(initial-state[problem]));
    father(initial-state[problem]) = empty;
    while (1)
      if Queue/Stack is empty then return failure;
      node = pop(Queue/Stack) ;
      if test(node,Goal[problem]) then return path(node,father);
      expand-nodes ←adjacent-nodes(node, Operators[problem]);
      push(Queue/Stack, expand-nodes );
      foreach ex-node in expand-nodes
            father(ex-node) = node;
    end
```

```
Function path(node,father[]) : print the solution
    n ← node
  while (n # empty)
      cout<< n <<“ <-- ” ;
       n = father[n];
    end
```

c. Cây tìm kiếm:

Trong quá trình tìm kiếm lời giải, chúng ta thường áp dụng một chiến lược để sinh ra các lời giải tiềm năng. Các lời giải tiềm năng được tổ chức thành cây mà gốc là trạng thái đầu của bài toán, các mức tiếp theo của cây là các nút kề với các nút ở mức trước. Thông thường thì cây tìm kiếm được mở rộng đến nó chứa trạng thái đích là dừng.

3. Đánh giá giải thuật tìm kiếm

Một giải thuật tìm kiếm lời giải của bài toán phụ thuộc rất nhiều vào chiến lược tìm kiếm (hay là cấu trúc dữ liệu để lưu các nút lá của cây trong quá trình tìm kiếm). Để đánh giá giải thuật tìm kiếm người ta đưa ra 4 tiêu chí sau:

1. Tính đầy đủ: giải thuật có tìm được lời giải của bài toán không nếu bài toán tồn tại lời giải?
2. Độ phức tạp thời gian: thời gian của giải thuật có kích cỡ như thế nào đối với bài toán?
3. Độ phức tạp không gian: Kích cỡ của bộ nhớ cần cho giải thuật? Trong giải thuật tổng quát ở trên, kích cỡ bộ nhớ chủ yếu phụ thuộc vào cấu trúc dữ liệu lưu các trạng thái lá của cây tìm kiếm
4. Tính tối ưu: Giải thuật có tìm ra lời giải có chi phí tối ưu (nhỏ nhất hoặc lớn nhất tùy theo ngữ cảnh của bài toán)?

Độ phức tạp thời gian và độ phức tạp không gian của giải thuật tìm kiếm lời giải của bài toán có thể đánh giá dựa trên kích thước đầu vào của giải thuật. Các tham số kích thước đầu vào có thể là:

- b – nhân tố nhánh của cây tìm kiếm: số nhánh tối đa của một nút, hay là số phép chuyển trạng thái tối đa của một trạng thái tổng quát
- d – độ sâu của lời giải có chi phí nhỏ nhất
- m – độ sâu tối đa của cây tìm kiếm (m có thể là vô hạn)

Trong các giải thuật tìm kiếm lời giải đề cập đến ở chương này, chúng ta sẽ đánh giá ưu, nhược điểm của từng giải thuật dựa trên 4 tiêu chí trên.

4. Các giải thuật tìm kiếm không có thông tin phản hồi (tìm kiếm mù)

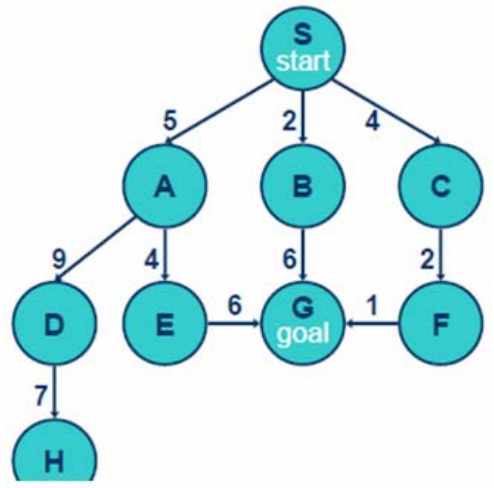
Các giải thuật tìm kiếm không sử dụng thông tin phản hồi (hay là giải thuật tìm kiếm mù) là các giải thuật chỉ sử dụng thông tin từ 5 thành phần cơ bản của bài toán (trạng thái tổng quát, trạng thái đầu, trạng thái đích, phép chuyển trạng thái, chi phí). Ý tưởng chung cơ bản của các giải thuật này là sinh ra cây lời giải tiềm năng (cây tìm kiếm) một cách có hệ thống (không bỏ sót và không lặp lại). Phần này sẽ giới thiệu các giải thuật tìm kiếm theo chiều rộng, tìm kiếm theo chiều sâu, tìm kiếm theo chiều sâu có giới hạn, tìm kiếm sâu dần. Các giải thuật này đều theo giải thuật chung đã giới thiệu bên trên, chỉ khác nhau ở chiến lược tìm kiếm hay là cấu trúc dữ liệu để lưu giữ và lấy ra các nút lá của cây tìm kiếm.

a. Tìm kiếm theo chiều rộng

Giải thuật tìm kiếm lời giải theo chiều rộng là cài đặt cụ thể của giải thuật chung tìm kiếm lời giải, trong đó có sử dụng cấu trúc dữ liệu kiểu hàng đợi (queue) để lưu giữ các trạng thái nút lá của cây tìm kiếm. Các nút lá sinh ra trong quá trình thực thi giải thuật sẽ được cập nhật vào một hàng đợi theo nguyên tắc nút nào được đưa vào hàng đợi trước sẽ được lấy ra trước trong quá trình mở rộng cây. Chi tiết của giải thuật được cho trong bảng bên dưới.

Chúng ta sẽ minh họa việc tìm kiếm lời giải bằng giải thuật tìm kiếm theo chiều rộng bằng ví dụ cụ thể như sau. Giả sử bài toán có không gian các trạng thái đầy đủ như hình vẽ ngay sau bảng giải thuật (trang sau), với trạng thái đầu là S, trạng thái đích là G và các phép chuyển trạng thái là các cung nối giữa các trạng thái. Giải thuật bắt đầu xét với hàng đợi chứa trạng thái đầu S, lấy trạng thái ở đầu hàng đợi ra kiểm tra xem nó có là trạng thái đích, nếu là đích thì in lời giải, nếu không thì bổ sung các trạng thái con của nó vào hàng đợi.

```
Function Breadth-Search(problem, Queue) returns a solution, or failure
    Queue ← make-queue(make-node(initial-state[problem]));
    father(initial-state[problem]) = empty;
    while (1)
      if Queue is empty then return failure;
      node = pop(Queue) ;
      if test(node,Goal[problem]) then return path(node,father);
      expand-nodes ←adjacent-nodes(node, Operators[problem]);
      push(Queue, expand-nodes );
      foreach ex-node in expand-nodes
            father(ex-node) = node;
    end
```

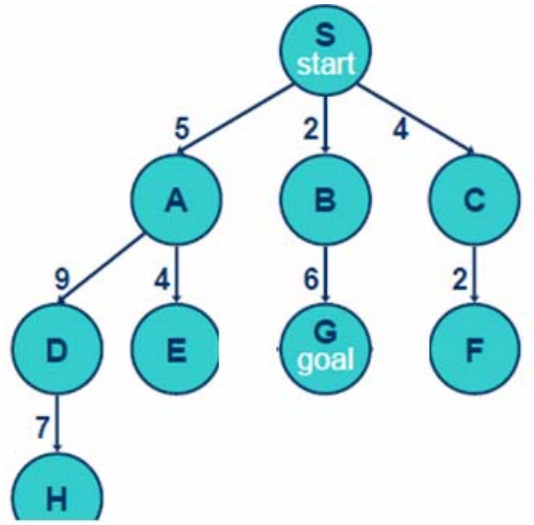


Không gian đầy đủ các trạng thái của bài toán

Bảng phía dưới là diễn biến các biến chính của giải thuật: biến trạng thái đang xét – node, biến hàng đợi – Queue, biến lưu thông tin về cây tìm kiếm – Father. Giải thuật kết thúc với 8 vòng lặp khi trạng thái đang xét node = G và khi đó lời giải của bài toán là đường đi G ← B ← S.

node	Queue	Father
	S	
S	A, B, C	Father[A,B,C]=S
A	B, C, D, E	Father[D,E]=A
B	C,D,E,G	Father[G]=B
C	D, E, G, F	Father[F]=C
D	E,G, F, H	Father[H]=D
E	G, F, H	
G	F, H	

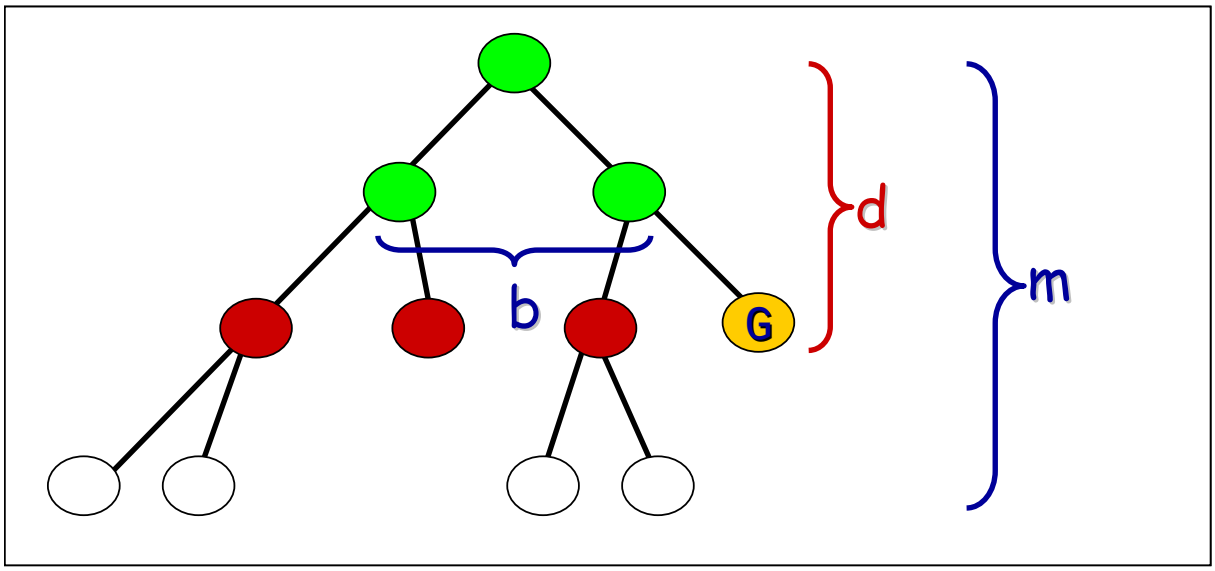
Giá trị các biến trong giải thuật theo chiều rộng



Cây tìm kiếm của giải thuật theo chiều rộng

Đánh giá giải thuật tìm kiếm theo chiều rộng:

- ✓ Tính đầy đủ: giải thuật sẽ cho lời giải của bài toán nếu bài toán tồn tại lời giải và nhân tố nhánh b là hữu hạn
- ✓ Độ phức tạp thời gian: $1+b+b^2+\ldots+b^d$ (số vòng lặp khi gặp trạng thái đích) = $O(b^d)$
- ✓ Độ phức tạp không gian: số lượng ô nhớ tối đa sử dụng trong giải thuật (chủ yếu là biến Queue, xem hình vẽ dưới): b^d
- ✓ Tính tối ưu: giải thuật tìm kiếm theo chiều rộng sẽ tìm ra lời giải với ít trạng thái trung gian nhất.



Hàng đợi trong giải thuật tìm kiếm theo chiều rộng chỉ chứa các nút lá của cây tìm kiếm, vì vậy có kích thước là b^d.

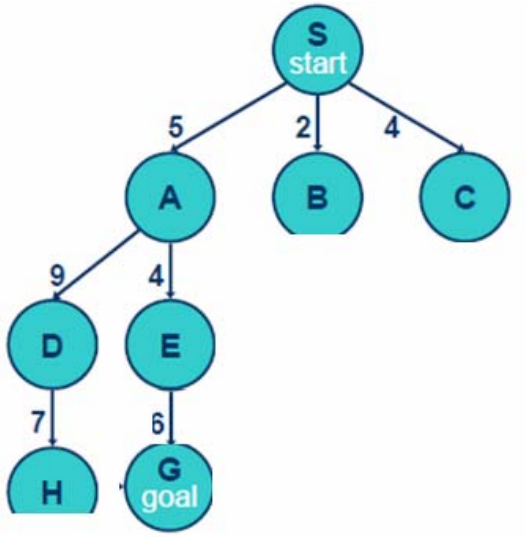
b. Tìm kiếm theo chiều sâu

Giải thuật tìm kiếm theo chiều sâu hoàn toàn tương tự như giải thuật tìm kiếm theo chiều rộng, chỉ khác ở chỗ thay vì sử dụng cấu trúc dữ liệu hàng đợi, ta sử dụng cấu trúc dữ liệu ngăn xếp (Stack) để lưu giữ các trạng thái lá của cây tìm kiếm. Đối với cấu trúc dữ liệu ngăn xếp, các trạng thái đưa vào sau cùng sẽ được lấy ra trước để mở rộng cây tìm kiếm. Giải thuật và diễn biến các biến chính trong giải thuật được trình bày trong các bảng và hình vẽ dưới đây. Kết quả của giải thuật là lời giải G ← E ← A ← S.

```
Function Depth-Search(problem, Stack) returns a solution, or failure
    Stack ← make-queue(make-node(initial-state[problem]));
    father(initial-state[problem]) = empty;
    while (1)
      if Stack is empty then return failure;
      node = pop(Stack) ;
      if test(node,Goal[problem]) then return path(node,father);
      expand-nodes ←adjacent-nodes(node, Operators[problem]);
      push(Stack, expand-nodes );
      foreach ex-node in expand-nodes
            father(ex-node) = node;
    end
```

node	Stack	father
	S	
S	A, B, C	Father[A,B,C]=S
A	D, E, B, C	Father[D,E]=A
D	H, E, B, C	Father[H]=D
H	E, B, C	
E	G, B, C	Father[G]=E
G		

Giá trị các biến trong giải thuật theo chiều sâu



Cây tìm kiếm của giải thuật theo chiều

Đánh giá giải thuật tìm kiếm theo chiều sâu:

- ✓ Tính đầy đủ: giải thuật không chắc chắn cho lời giải của bài toán trong trường hợp không gian trạng thái của bài toán là vô hạn

- ✓ Độ phức tạp thời gian: $O(b^m)$
- ✓ Độ phức tạp không gian: O(b.m)
- ✓ Tính tối ưu: giải thuật tìm kiếm theo chiều sâu không cho lời giải tối ưu.

c. Tìm kiếm theo chiều sâu có giới hạn

Giải thuật tìm kiếm theo chiều sâu ở trên có ưu điểm là nó có thể sinh ra lời giải nhanh chóng mà không tốn kém bộ nhớ của máy tính. Tuy nhiên nếu không gian trạng thái của bài toán là vô hạn thì rất có thể nó không tìm được lời giải của bài toán khi hướng tìm kiếm không chứa trạng thái đích. Để khắc phục nhược điểm này, chúng ta có thể đặt giới hạn độ sâu trong giải thuật: nếu độ sâu của trạng thái đang xét vượt quá ngưỡng nào đó thì chúng ta không bổ sung các nút kề với trạng thái này nữa mà chuyển sang hướng tìm kiếm khác. Chi tiết của giải thuật được cho trong bảng dưới đây, trong đó chúng ta đưa thêm biến mảng một chiều depth[i] lưu độ sâu của trạng thái i.

```
Function Depth-Limitted-Search(problem, maxDepth)
    returns a solution, or failure
   ------------------------------------------------------------------
    Stack ← make-queue(make-node(initial-state[problem]));
    father(initial-state[problem]) = empty;
    depth(initial-state[problem]) = 0;
    while (1)
      if Stack is empty then return failure;
      node = pop(Stack) ;
      if test(node,Goal[problem]) then return path(node,father);
      if (depth(node) < maxDepth)
            expand-nodes ←adjacent-nodes(node, Operators[problem]);
            push(Stack, expand-nodes );
            foreach ex-node in expand-nodes
                  father(ex-node) = node;
    end
```

d. Tìm kiếm sâu dần

Giải thuật tìm kiếm với chiều sâu có giới hạn ở trên phụ thuộc vào giới hạn độ sâu lựa chọn ban đầu. Nếu biết trước trạng thái đích sẽ xuất hiện trong phạm vi độ sâu nào đó của cây tìm kiếm thì chúng ta đặt giới hạn độ sâu đó cho giải thuật. Tuy nhiên nếu chọn độ sâu tối đa không phù hợp, giải thuật tìm kiếm theo chiều sâu có giới hạn sẽ không tìm được lời giải của bài toán. Chúng ta có thể gọi thực hiện giải thuật tìm kiếm lời giải ở độ sâu khác nhau, từ bé đến lớn. Giải thuật bổ sung như sau:

```
Function Iterative-deepening-Search(problem) returns a solution, or failure
    for depth = 0 to ∞ do
      result ← Depth-Limited-Search(problem, depth)
      if result succeeds then return result
    end
    return failure
```

Chương 3 –Các phương pháp tìm kiếm heuristic

1. Giải thuật tìm kiếm tốt nhất đầu tiên (best first search)

Các giải thuật trong mục 4 ở trên có chung đặc điểm là tìm kiếm lời giải một cách có hệ thống: xây dựng tất cả không gian lời giải tiềm năng theo cách vét cạn, không bỏ sót và không lặp lại. Trong rất nhiều trường hợp, các giải thuật như vậy không khả thi vì không gian trạng thái bài toán quá lớn, tốc độ xử lý và bộ nhớ của máy tính không cho phép duyệt các lời giải tiềm năng. Để hạn chế không gian cây các lời giải tiềm năng, chúng ta đưa ra một hàm định hướng việc mở rộng cây tìm kiếm. Theo cách này, chúng ta sẽ mở rộng cây theo các nút lá có nhiều tiềm năng chứa trạng thái đích hơn các nút lá khác.

Ví dụ, đối với bài toán 8 số, chúng ta đưa ra một hàm định hướng mở rộng cây như sau: giả sử n là một trạng thái bàn cờ (một sự sắp xếp 8 quân cờ trên bàn cờ 3x3), hàm định hướng h định nghĩa như sau:

h(n) = tổng khoảng cách Manhatan các vị trí của từng quân cờ trên bàn cờ n với vị trí của nó trên bàn cờ đích.

Chẳng hạn, nếu n là trạng thái đầu như trong hình của mục 1.b, h(n) có thể xác định như sau:

Quân cờ	*Vị trí trên n*	*Vị trí trên bàn cờ đích*	*Khoảng cách (số lần dịch chuyển khi bàn cờ không có quân cờ khác)*
Trạng thái n là trạng thái đầu của bài toán 8 số trong mục 1.b			
1	(3,3)	(1,3)	2
2	(2,3)	(2,3)	0
3	(3,2)	(3,3)	1

4	(1,1)	(1,2)	1
5	(1,3)	(2,2)	2
6	(3,1)	(3,2)	1
7	(1,2)	(1,1)	1
8	(2,1)	(2,1)	0
h(n) = 2 + 0 + 1 + 1 + 2 + 1 + 1 + 0 = 8			

Hàm h(n) như mô tả ở trên phản ánh sự "khác nhau" giữa trạng thái n với trạng thái đích, h(n) càng nhỏ thì n càng "giống" với trạng thái đích, khi n trùng với trạng thái đích thì h(n) = 0.

Khi không gian bài toán quá lớn, việc mở rộng cây theo chiến lược theo chiều rộng hoặc theo chiều sâu dẫn đến cây tìm kiếm quá lớn mà không chứa lời giải của bài toán. Khi đó chúng ta cần mở rộng cây theo hướng các nút lá có nhiều triển vọng chứa trạng thái đích, và hàm h(n) sẽ giúp chúng ta mở rộng cây. Chúng ta sẽ mở rộng cây theo hướng các nút lá có hàm h(n) nhỏ nhất. Khi đó h được gọi là thông tin phản hồi của quá trình mở rộng cây là có hợp lý hay không (vì thế mà các phương pháp tìm kiếm trong mục này gọi là tìm kiếm có phản hồi - informed search, chúng cũng có tên là tìm kiếm heuristic - dựa trên hàm đánh giá hợp lý h).

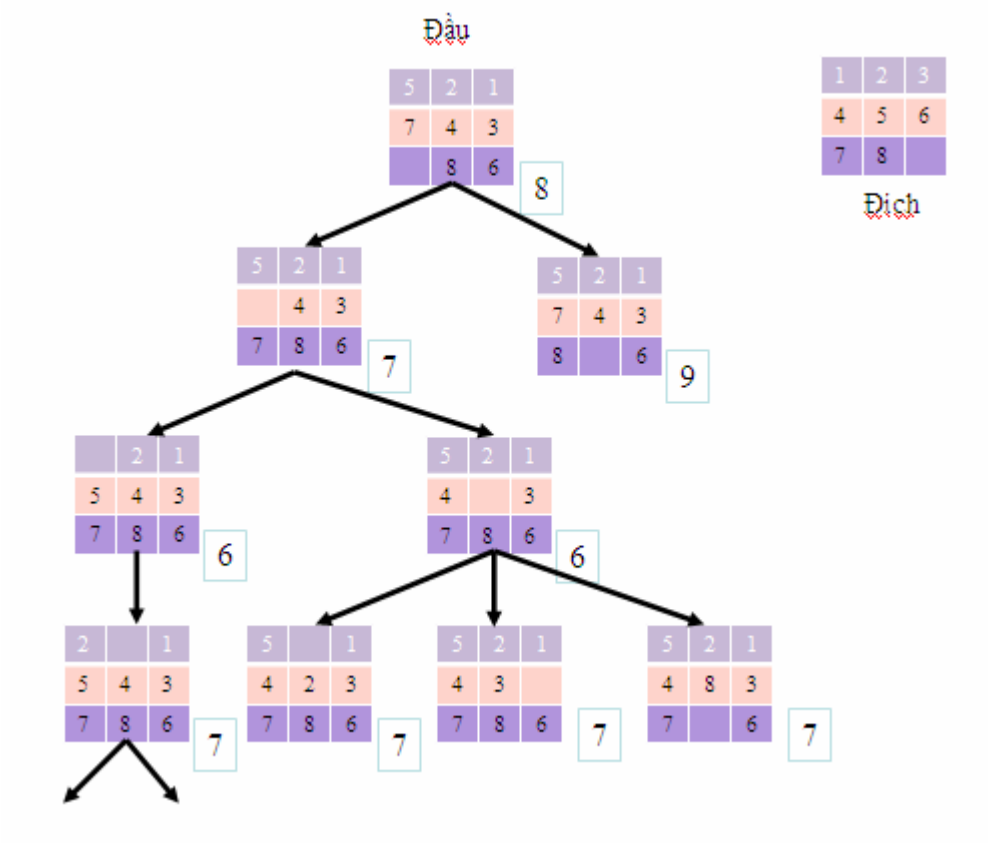
Để mở rộng cây theo nút lá có giá trị h nhỏ nhất, chúng ta sử dụng một cấu trúc dữ liệu là danh sách (list) có sắp xếp theo giá trị h. Giải thuật chi tiết được trình bày trong bảng sau (được gọi là giải thuật Best-First-Search):

```
Function Best-First-Search(problem, list, h) returns a solution, or failure
    list ← make-list(make-node(initial-state[problem]));
    father(initial-state[problem]) = empty;
    while (1)
      if list is empty then return failure;
      node = pop(list) ; // node with max/min h
      if test(node,Goal[problem]) then return path(node,father);
      expand-nodes ←adjacent-nodes(node, Operators[problem]);
      push(list, expand-nodes ,h);
      foreach ex-node in expand-nodes
            father(ex-node) = node;
    end
```

```
Function push(list, expand-nodes ,h);
    Chèn các nodes trong expand-nodes vào list sao cho mảng list sắp theo thứ tự
    tăng/giảm theo hàm h
```

Chú ý rằng, cấu trúc giải thuật này giống với các giải thuật tìm kiếm theo chiều rộng hay theo chiều sâu, chỉ khác ở chỗ, thay vì sử dụng hàng đợi hay ngăn xếp để lưu giữ các trạng thái lá của cây tìm kiếm, chúng ta sử dụng danh sách sắp xếp theo giá trị hàm h. Danh sách sắp xếp tăng hay giảm phụ thuộc vào hàm h và ngữ cảnh của bài toán, ví dụ bài toán 8 số và hàm h định nghĩa ở trên, danh sách cần sắp xếp theo thứ tự tăng dần để khi lấy phần tử ở đầu danh sách ta cẽ được nút lá "gần" với đích nhất.

Hình vẽ sau minh họa việc mở rộng cây tìm kiếm khi sử dụng giải thuật trên:



Cây có gốc là trạng thái đầu với giá trị h(đầu) = 8. Từ trạng thái gốc có hai phép chuyển: chuyển ô trống đổi vị trí cho ô số 7 (hàm h giảm đi 1) và đổi vị trí ô trống cho ô số 8 (hàm h tăng lên 1). Lúc này danh sách sắp xếp có 2 nút lá tương ứng với hai trạng thái có hàm h=7 và h=9. Trong 2 nút lá này, giải thuật sẽ chọn nút có giá trị hàm h nhỏ hơn (h=7) để mở rộng cây. Tiếp tục mở rộng cây theo hướng nút lá có giá trị h nhỏ nhất (trong trường hợp có nhiều nút lá cùng có giá trị nhỏ nhất thì chọn nút lá nào xuất hiện trước) thì ta được một phần của cây như trong hình vẽ trên.

2. Các biến thể của giải thuật best first search

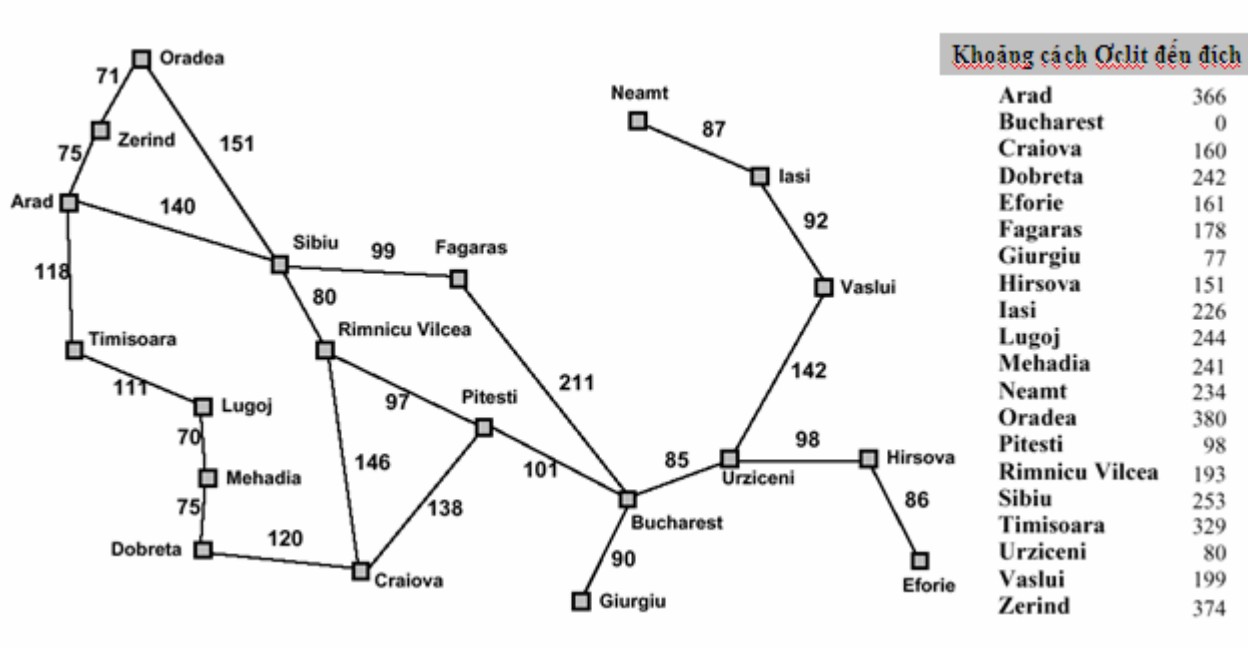
Ý tưởng của giải thuật tìm kiếm tốt nhất đầu tiên (best first search) là mở rộng cây tìm kiếm theo hướng ưu tiên các nút lá có triển vọng chứa trạng thái đích (dựa trên hàm đánh giá h). Giải thuật best-first-search có các biến thể sau:

- Khi hàm h(n) là chi phí của dãy phép chuyển từ trạng thái đầu đến trạng thái n thì giải thuật best-first-search có tên gọi khác là giải thuật tìm kiếm đều (uniform search). Trong

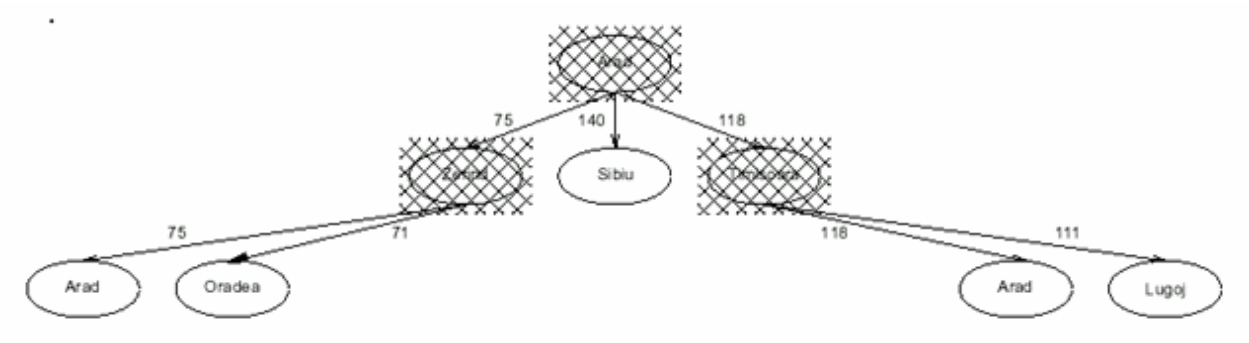
trường hợp này, cây tìm kiếm sẽ mở rộng đều về tất cả các hướng theo vết dầu loang từ trạng thái đầu. Khi hàm chi phí của dãy phép chuyển là số các đỉnh trung gian thì giải thuật uniform search trở thành giải thuật tìm kiếm theo chiều rộng. Giải thuật uniform search sẽ cho lời giải với chi phí nhỏ nhất, tuy nhiên cây tìm kiếm sinh ra trong giải thuật này thường có kích thước rất lớn.

- Khi h(n) là ước lượng chi phí/khoảng cách từ n đến đích (ví dụ như khoảng cách Manhatan trong bài toán 8 số ở trên) thì giải thuật best-first-search được gọi là giải thuật tham ăn (greedy search). Giải thuật tham ăn sẽ chọn nút lá n "gần" đến đích nhất trong số các nút lá của cây tìm kiếm để mở rộng cây, và nó không quan tâm đến chi phí từ trạng thái đầu đến n. Do vậy giải thuật có xu hướng cho ra kết quả trong thời gian nhanh nhất, nhưng không phải lúc nào cũng là lời giải ngắn nhất.

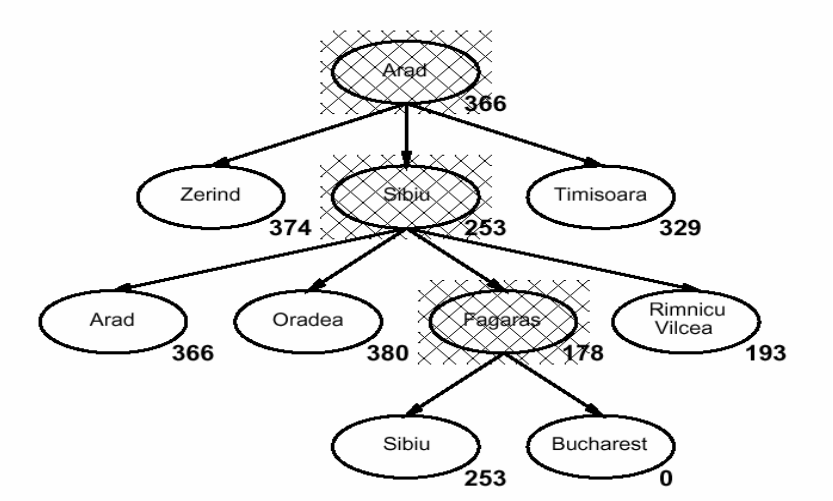
- Khi h(n) = f(n) + g(n), trong đó f(n) là hàm chi phí/khoảng cách từ trạng thái đầu đến n và g(n) là hàm ước lượng chi phí/khoảng cách từ n đến trạng thái đích, và nếu g(n) là ước lượng dưới của hàm chi phí/khoảng cách thực sự từ n đến trạng thái đích thì giải thuật best-first-search được gọi là giải thuật A*. Giải thuật A* là giải thuật trung hòa giữa hai giải thuật uniform và giải thuật greedy ở trên. A* cho lời giải có chi phí nhỏ nhất (bạn đọc có thể tìm hiểu chứng minh điều này ở các tài liệu khác) và cây tìm kiếm có kích thước vừa phải.



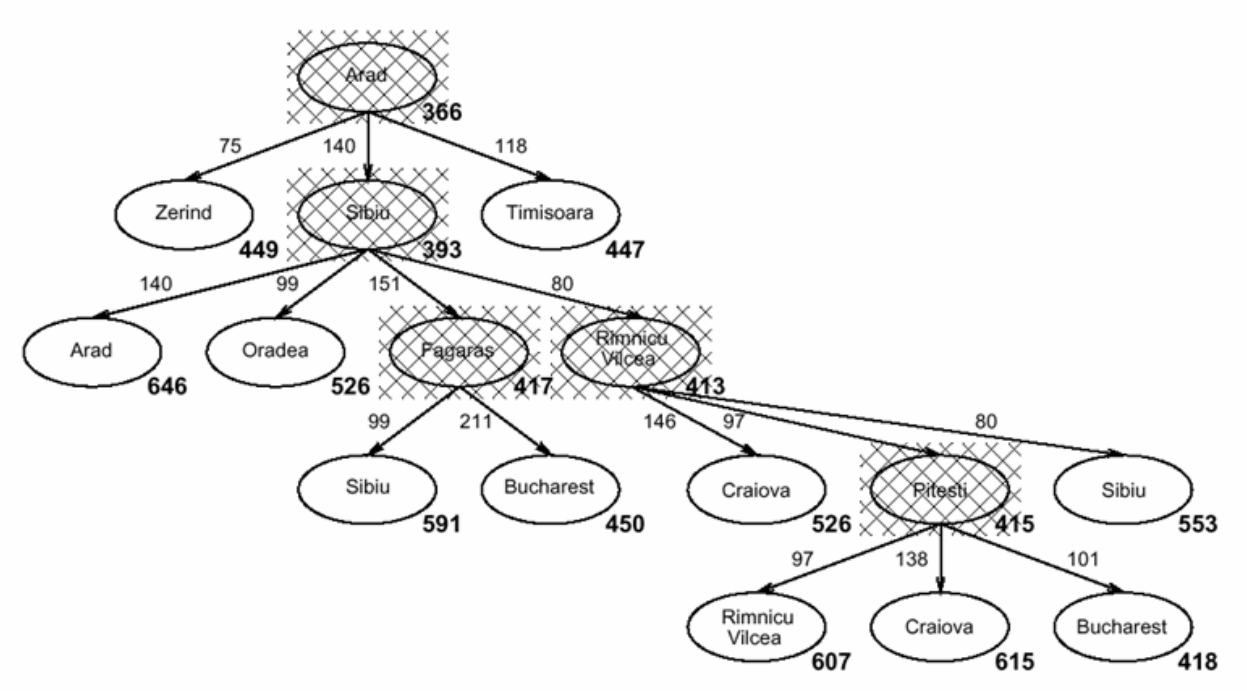
Ví dụ, đối với bài toán tìm đường đi từ thành phố Arad đến thành phố Bucharest đã mô tả trong 1.b, nếu chúng ta sử dụng khoảng cách Ơclit (khoảng cách theo đường chim bay) từ mỗi thành phố đến đích (xem hình vẽ trên) thì các giải thuật uniform, greedy và A* sẽ cho các cây tìm kiếm như sau:



Một phần cây tìm kiếm của giải thuật Uniform search



Cây tìm kiếm của giải thuật Greedy search



Cây tìm kiếm của giải thuật A*

3. Các giải thuật khác

*** Tìm kiếm leo đồi:**

Ý tưởng: Tìm kiếm theo chiều sâu kết hợp với hàm đánh giá. Mở rộng trạng thái hiện tại và đánh giá các trạng thái con của nó bằng hàm đánh giá heuristic. Tại mỗi bước, nút lá "tốt nhất" sẽ được chọn để đi tiếp.

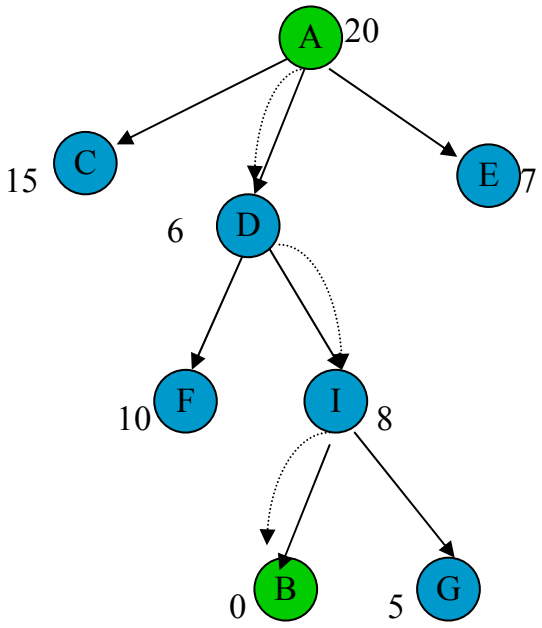
Procedure Hill-Climbing_search;

Begin

1. Khởi tạo ngăn xếp S chỉ chứa trạng thái đầu;
2. **Loop do**
   - 2.1 **If** S rỗng **then** {thông báo thất bại; stop};
   - 2.2 Lấy trạng thái u ở đầu ngăn xếp S;
   - 2.3 **If** u là trạng thái kết thúc **then**
     {thông báo thành công; stop};
   - 2.4 **For** mỗi trạng thái v kề u **do** đặt v vào danh sách L;
   - 2.5 Sắp xếp L theo thứ tự tăng dần của hàm đánh giá sao cho trạng thái tốt nhất ở đầu danh sách L;
   - 2.6 Chuyển danh sách Lvào ngăn xếp S;

End;

Ví dụ : Với ví dụ đồ thị không gian trạng thái như hình 2.2 thì cây tìm kiếm leo đồi tương ứng như hình 2.4 :



Cây tìm kiếm leo đồi

Hạn chế của thuật toán :

- Giải thuật có khuynh hướng bị sa lầy ở những cực đại cục bộ:

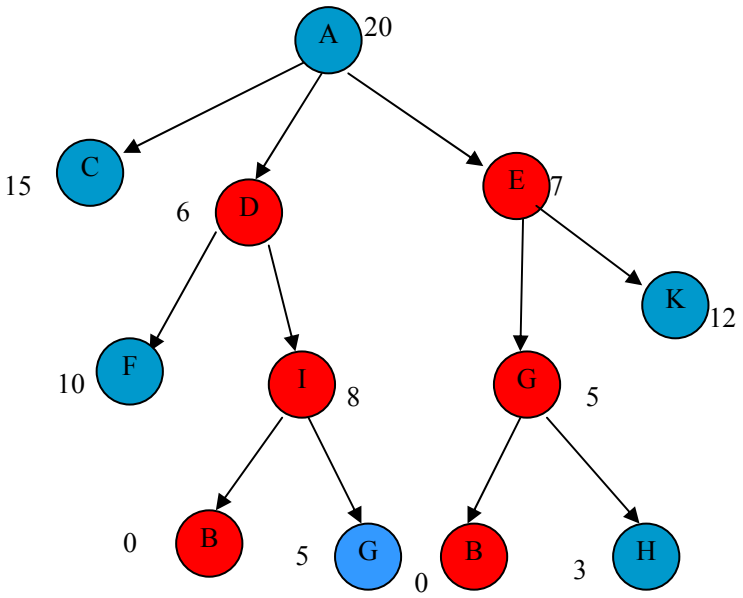
  + Lời giải tìm được không tối ưu

  + Không tìm được lời giải mặc dù có tồn tại lời giải

- Giải thuật có thể gặp vòng lặp vô hạn do không lưu giữ thông tin về các trạng thái đã duyệt.

*** Tìm kiếm Beam**

Để hạn chế không gian tìm kiếm, người ta đưa ra phương pháp tìm kiếm Beam. Đây là phương pháp tìm kiếm theo chiều rộng nhưng có hạn chế số đỉnh phát triển ở mỗi mức. Trong tìm kiếm theo chiều rộng, tại mỗi mức ta phát triển tất cả các đỉnh, còn tìm kiếm Beam thì chọn *k* đỉnh tốt nhất để phát triển. Các đỉnh này được xác định bởi hàm đánh giá. Ví dụ, với đồ thì không gian trạng thái như hình 2.2 và lấy *k*=2 thì cây tìm kiếm Beam như hình 2.5. Các đỉnh được chọn ở mỗi mức là các đỉnh được tô màu đỏ:



Cây tìm kiếm Beam

*** Tìm kiếm nhánh cận**

Ý tưởng : thuật toán tìm kiếm leo đồi kết hợp với hàm đánh giá *f(u)*. Tại mỗi bước, khi phát triển trạng thái *u*, chọn trạng thái con v tốt nhất (*f(v)* nhỏ nhất) của *u* để phát triển ở bước sau. Quá trình tiếp tục như vậy cho đến khi gặp trạng thái *w* là đích, hoặc *w* không có đỉnh kề, hoặc *w* có *f(w)* lớn hơn độ dài đường đi tối ưu tạm thời (đường đi đầy đủ ngắn nhất trong số những đường đi đầy đủ đã tìm được). Trong các trường hợp này, chúng ta không phát triển đỉnh *w* nữa, tức là cắt bỏ những nhánh xuất phát từ *w*, và quay lên cha của *w* để tiếp tục đi xuống trạng thái tốt nhất trong số những trạng thái còn lại chưa được phát triển.

Procedure Branch-and-Bound;

Begin

1. Khởi tạo ngăn xếp S chỉ chứa trạng thái đầu;

Gán giá trị ban đầu cho cost; */*cost là giá trị đường đi tối ưu tạm thời*/*

2. **Loop do**

2.1 **If** S rỗng **then** {thông báo thất bại; stop};

2.2 Lấy trạng thái u ở đầu ngăn xếp S;

2.3 **If** u là trạng thái kết thúc **then**

if g(u)<=cost then {cost ←g(u); quay lại 2.1};

2.4 **if** f(u)>cost **then** quay lại 2.1;

2.5 **For** mỗi trạng thái v kề u **do**

{g(v) ←g(u)+k(u,v);

f(v) ←g(v) +h(v);

đặt v vào danh sách L1};

2.6 Sắp xếp L theo thứ tự tăng dần của hàm f;

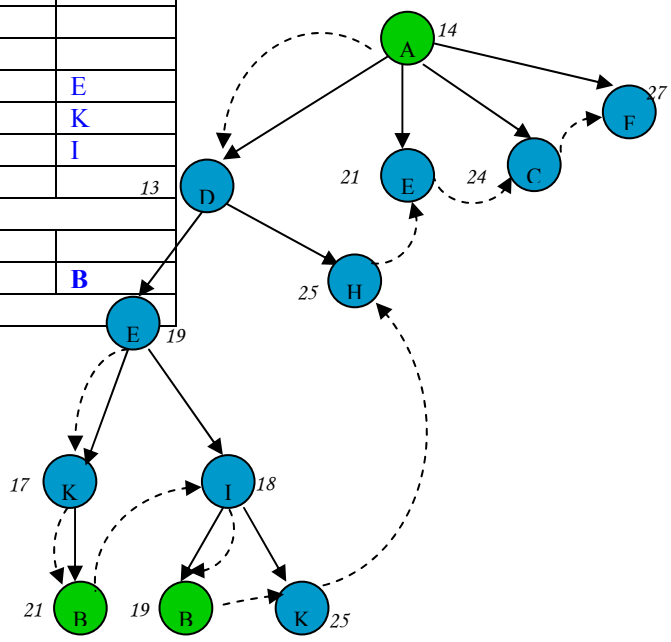
2.7 Chuyển danh sách Lvào ngăn xếp S;

End;

Ví dụ : Với đồ thị không gian trạng thái như hình 2.7, đỉnh xuất phát A và đỉnh đích B. Áp dụng thuật toán nhánh – cận, ta xây dựng được cây tìm kiếm như hình 2.9 và giá trị của hàm *f* tại các đỉnh được tính như bảng 2.2:

Đỉnh phát triển (u)	**Đỉnh con (v)**	**g(v)**	**f(v)**	**Đỉnh chọn**
A	C	9	9+15=24	
	D	7	7+6=13	D
	E	13	13+8=21	
	F	20	20+7=27	
D	H	7+8=15	15+10=25	
	E	7+4=11	11+8=19	E
E	K	11+4=15	15+2=17	K
	I	11+3=14	14+4=18	I
K	B	15+6=21	21+0=21	
B	**cost := 21**			
I	K	14+9=23	23+2=25	
	B	14+5=19	19+0=19	**B**
B	**cost := 19**			

Tính giá trị hàm f cho thuật toán nhánh-cận



Cây tìm kiếm nhánh-cận

Nhận xét : Thuật toán nhánh-cận cũng là thuật toán đầy đủ và tối ưu nếu h(u) là hàm đánh giá thấp và có độ dài các cung không nhỏ hơn một số dương δ nào đó

Chương 4 – Các giải thuật tìm kiếm lời giải cho trò chơi

Chương trình chơi cờ đầu tiên được viết bởi Claude Shannon vào năm 1950 đã là một minh chứng cho khả năng máy tính có thể làm được những việc đòi hỏi trí thông minh của con người. Từ đó người ta nghiên cứu các chiến lược chơi cho máy tình với các trò chơi có đối thủ (có hai người tham gia). Việc giải quyết bài toán này có thể đưa về bài toán tìm kiếm trong không gian trạng thái, tức là tìm một chiến lược chọn các nước đi hợp lệ cho máy tính. Tuy nhiên, vấn đề tìm kiếm ở đây phức tạp hơn so với vấn đề tìm kiếm trong chương trước, vì người chơi không biết trước đối thủ sẽ chọn nước đi nào tiếp theo. Chương này sẽ trình bày một số chiến lược tìm kiếm phổ biến như Minimax, phương pháp cắt cụt α-β.

1. Cây trò chơi đầy đủ

Các trò chơi có đối thủ có các đặc điểm: hai người thay phiên nhau đưa ra các nước đi tuân theo các luật của trò chơi (các nước đi hợp lệ), các luật này là như nhau đối với cả hai người chơi, chẳng hạn các trò chơi cờ: cờ vua, cờ tướng, cờ ca rô (tic-tăc-toe), …. Ví dụ, trong chơi cờ vua, một người điều khiển quân Trắng và một người điều khiển quân Đen. Người chơi có thể lựa chọn các nước đi theo các luật với các quân tốt, xe, mã,… Luật đi quân tốt Trắng, xe Trắng, mã Trắng,… giống luật đi quân tốt Đen, xe Đen, mã Đen,…Hơn nữa, cả hai người chơi đều biết đầy đủ các thông tin về tình thế cuộc chơi. Thực hiện trò chơi là người chơi tìm kiếm nước đi *tốt nhất* trong số rất nhiều nước đi hợp lệ, tại mỗi lượt chơi của mình, sao cho sau một dãy nước đi đã thực hiện người chơi phải thắng cuộc.

Vấn đề chơi cờ có thể được biểu diễn trong không gian trạng thái, ở đó, mỗi trạng thái là một tình thế của cuộc chơi (sự sắp xếp các quân cờ trên bàn cờ):

- Trạng thái xuất phát là sự sắp xếp các quân cờ của hai bên khi bắt đầu cuộc chơi (chưa ai đưa ra nước đi)
- Các toán tử biến đổi trạng thái là các nước đi hợp lệ

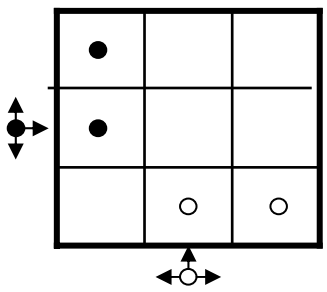
- Các trạng thái kết thúc là các tình thế mà cuộc chơi dừng, thường được xác định bởi một số điều kiện dừng (chẳng hạn, quân Trắng thắng hoặc quân Đen thắng hoặc hai bên hòa nhau)
- Hàm kết cuộc: mang giá trị tương ứng với mỗi trạng thái kết thúc. Chẳng hạn, trong cờ vua, hàm kết cuộc có giá trị là 1 tại các trạng thái mà Trắng thắng, -1 tại các trạng thái mà Trắng thua và 0 tại các trạng thái hai bên hòa nhau. Trong các trò chơi tính điểm khác thì hàm kết cuộc có thể nhận các giá trị nguyên trong đoạn [-m, m], với m là một số nguyên dương nào đó.

Như vậy, trong các trò chơi có đối thủ, người chơi (điều khiển quân Trắng – gọi tắt là Trắng) luôn tìm một dãy các nước đi xen kẽ với các nước đi của đối thủ (điều khiển quân Đen – gọi tắt là Đen) để tạo thành một đường đi từ trạng thái ban đầu đến trạng thái kết thúc là thắng cho Trắng.

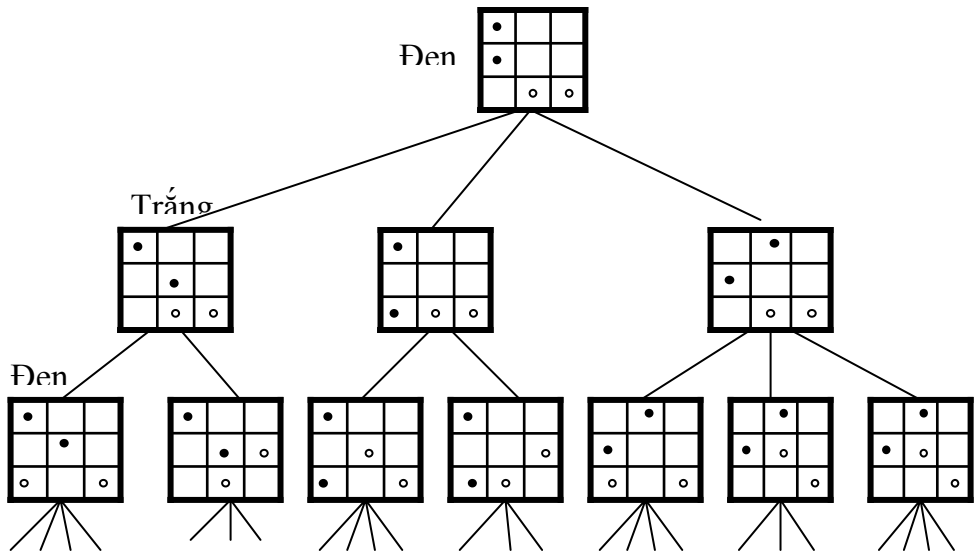
Không gian tìm kiếm đối với các trò chơi này có thể được biểu diễn bởi *cây trò chơi* như sau: gốc của cây ứng với trạng thái xuất phát, các đỉnh trên cây tương ứng với các trạng thái của bàn cờ, các cung (u, v) nếu có biến đổi từ trạng thái u đến trạng thái v. Các đỉnh trên cây được gán nhãn là đỉnh Trắng (Đen) ứng với trạng thái mà quân Trắng (Đen) đưa ra nước đi. Nếu một đỉnh u được gán nhãn là Trắng (Đen) thì các đỉnh con v của nó là tất cả các trạng thái nhận được từ u do Trắng (Đen) thực hiện một nước đi hợp lệ nào đó. Do đó, các đỉnh trên cùng một mức của cây đều có nhãn là Trắng hoặc đều có nhãn là Đen, các lá của cây ứng với trạng thái kết thúc.

Ví dụ: trò chơi Dodgem:

Có hai quân Trắng và hai quân Đen được xếp vào bàn cờ 3x3. Ban đầu các quân cờ được xếp như hình bên. Quân Đen có thể đi đến ô trống bên phải, ở trên hoặc ở dưới. Quân Trắng có thể đi đến ô trống bên trên, bên trái hoặc bên phải. Quân Đen nếu ở cột ngoài cùng bên phải có thể đi ra khỏi bàn cờ, quân Trắng nếu ở hàng trên cùng có thể đi ra khỏi bàn cờ. Ai đưa được cả hai quân của mình ra khỏi bàn cờ hoặc tạo ra tình thế mà đối phương không đi được là thắng cuộc.



Trò chơi Dodgem



Cây trò chơi Dodgem với Đen đi trước

2. *Giải thuật Minimax*

Quá trình chơi cờ là quá trình mà Trắng và Đen thay phiên nhau đưa ra các nước đi hợp lệ cho đến khi dẫn đến trạng thái kết thúc cuộc chơi. Quá trình này biểu diễn bởi đường đi từ nút gốc tới nút lá trên cây trò chơi. Giả sử tại một đỉnh u nào đó trên đường đi, nếu u là đỉnh Trắng (Đen) thì cần chọn một nước đi nào đó đến một trong các đỉnh con Đen (Trắng) v của u. Tại đỉnh Đen (Trắng) v sẽ chọn đi tiếp đến một đỉnh con Trắng (Đen) w của v. Quá trình này tiếp tục cho đến khi đạt đến một đỉnh lá của cây.

Chiến lược tìm nước đi của Trắng hay Đen là luôn tìm những nước đi dẫn tới trạng thái tốt nhất cho mình và tồi nhất cho đối thủ. Giả sử Trắng cần tìm nước đi tại đỉnh u, nước đi tối ưu cho Trắng là nước đi dẫn tới đỉnh con v sao cho v là tốt nhất trong số các đỉnh con của u. Đến lượt Đen chọn nước đi từ v, Đen cũng chọn nước đi tốt nhất cho mình. Để chọn nước đi tối ưu cho Trắng tại đỉnh u, cần xác định giá trị các đỉnh của cây trò chơi gốc u. Giá trị của các đỉnh lá ứng với giá trị của hàm kết cuộc. Đỉnh có giá trị càng lớn càng tốt cho Trắng, đỉnh có giá trị càng nhỏ càng tốt cho Đen. Để xác định giá trị các đỉnh của cây trò chơi gốc u, ta đi từ mức thấp nhất (các đỉnh lá) lên gốc u. Giả sử cần xác định giá trị của đỉnh v mà các đỉnh con của nó đã xác định. Khi đó, nếu v là đỉnh Trắng

thì giá trị của nó là giá trị lớn nhất trong các đỉnh con, nếu v là đỉnh Đen thì giá trị của nó là giá trị nhỏ nhất trong các đỉnh con.

Sau đây là thủ tục chọn nước đi cho Trắng tại đỉnh u Minimax(u, v), trong đó v là đỉnh con được chọn của u:

```
Procedure Minimax(u, v);
begin
    val ←-∝;
    for  mỗi w là đỉnh con của u  do
    if  val(u) <= MinVal(w)  then
            {val ← MinVal(w);  v ←  w}
end;
-------------------------------------------------
Function MinVal(u); {hàm xác định giá trị cho các đỉnh Đen}
begin
    if  u là đỉnh kết thúc  then  MinVal(u) ← f(u)
    else MinVal(u) ← min{MaxVal(v) | v là đỉnh con của u}
end;
-------------------------------------------------
Function MaxVal(u); { hàm xác định giá trị cho các đỉnh Trắng}
begin
    if  u là đỉnh kết thúc  then  MaxVal(u) ← f(u)
    else MaxVal(u) ← max{MinVal(v) | v là đỉnh con của u}
end;
```

Trong các thủ tục và hàm trên, f(u) là giá trị của hàm kết cuộc tại đỉnh kết thúc u.

Thuật toán Minimax là thuật toán tìm kiếm theo chiều sâu. Về lý thuyết, chiến lược Minimax cho phép tìm nước đi tối ưu cho Trắng. Tuy nhiên trong thực tế, ta không có đủ thời gian để tính toán nước đi tối ưu này. Bởi vì thuật toán tính toán trên toàn bộ cây trò

chơi (xem xét tất cả các đỉnh của cây theo kiểu vét cạn). Trong các trò chơi hay thì kích thước của cây trò chơi là cực lớn. Chẳng hạn, trong cờ vua, chỉ tính đến độ sâu 40 thì cây trò chơi đã có đến 10^{120} đỉnh. Nếu cây có độ cao m và tại mỗi đỉnh có b nước đi thì độ phức tạp về thời gian của thuật toán Minimax là $O(b^m)$.

Trong thực tế, các trò chơi đều có giới hạn về thời gian. Do đó, để có thể tìm nhanh nước đi tốt (không phải tối ưu) thay vì sử dụng hàm kết cuộc và xét tất cả các đỉnh của cây trò chơi, ta sử dụng hàm đánh giá và chỉ xem xét một bộ phận của cây trò chơi.

3. Giải thuật Minimax với độ sâu hạn chế

a) Hàm đánh giá

Hàm đánh giá eval cho mỗi đỉnh u là đánh giá "mức độ lợi thế" của trạng thái u. Giá trị của eval(u) là số dương càng lớn thì trạng thái u càng có lợi cho Trắng, giá trị của eval(u) là số dương càng nhỏ thì trạng thái u càng có lợi cho Đen, eval(u)=0 thì trạng thái u không có lợi cho đối thủ nào, eval(u)=+∞ thì u là trạng thái thắng cuộc cho Trắng, eval(u)=-∞ thì u là trạng thái thắng cuộc cho Đen.

Hàm đánh giá đóng vai trò rất quan trọng trong các trò chơi, nếu hàm đánh giá tốt sẽ định hướng chính xác việc lựa chọn các nước đi tốt. Việc thiết kế hàm đánh giá phụ thuộc vào nhiều yếu tố: các quân cờ còn lại của hai bên, sự bố trí các quân cờ này,… Để đưa ra hàm đánh giá chính xác đòi hỏi nhiều thời gian tính toán, tuy nhiên, trong thực tế người chơi bị giới hạn thời gian đưa ra nước đi. Vì vậy, việc đưa ra hàm đánh giá phụ thuộc vào kinh nghiệm của người chơi. Sau đây là một số ví dụ về cách xây dựng hàm đánh giá:

Ví dụ 1: Hàm đánh giá cho cờ vua. Mỗi loại quân được gán một giá trị số phù hợp với "sức mạnh" của nó. Chẳng hạn, quân tốt Trắng (Đen) được gán giá trị 1 (-1), mã hoặc tượng Trắng (Đen) được gán giá trị 3 (-3), xe Trắng (Đen) được gán giá trị 5 (-5) và hậu Trắng (Đen) được gán giá trị 9 (-9). Hàm đánh giá của một trạng thái được tính bằng cách lấy tổng giá trị của tất cả các quân cờ trong trạng thái đó. Hàm đánh giá này được gọi là hàm tuyến tính có trọng số, vì có thể biểu diễn dưới dạng:

$$s_1w_1 + s_2w_2 + \ldots + s_nw_n$$

Trong đó, w_i là giá trị của quân cờ loại i, s_i là số quân loại đó.

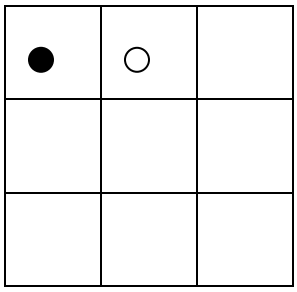
Đây là cách đánh giá đơn giản, vì nó không tính đến sự bố trí của các quân cờ, các mối tương quan giữa chúng.

Ví dụ 2: Hàm đánh giá trạng thái trong trò chơi Dodgem. Mỗi quân Trắng được gán giá trị tương ứng với các vị trí trên bàn cờ như trong hình bên trái. Mỗi quân Đen được gán giá trị ở các vị trí tương ứng nhu hình bên phải:

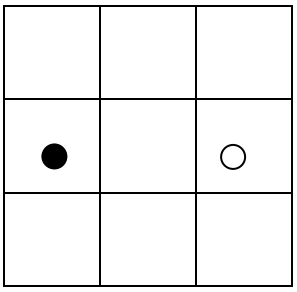
30	35	40
15	20	25
0	5	10

-10	-25	-40
-5	-20	-35
0	-15	-30

Ngoài ra, nếu quân Trắng cản trực tiếp một quân Đen, nó được thêm 40 điểm, nếu cản gián tiếp được thêm 30 điểm (xem hình dưới). Tương tự, nếu quân Đen cản trực tiếp quân Trắng nó được thêm -40 điểm, cản gián tiếp được thêm -30 điểm.

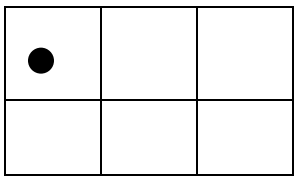


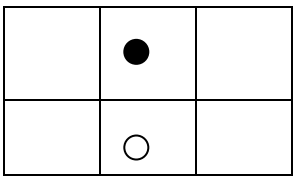
Trắng cản trực tiếp Đen được thêm 40 điểm

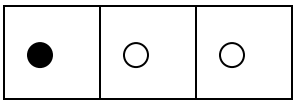


Trắng cản gián tiếp Đen được thêm 30 điểm

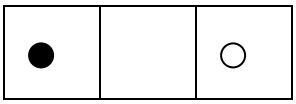
Áp dụng cách tính hàm đánh giá nêu trên, ta tính được giá trị của các trạng thái ở các hình dưới như sau:







Giá trị hàm đánh giá:75=
(-10+0+5+10)+(40+30)



Giá trị hàm đánh giá:-5=
(-25+0+20+10)+(-40+30)

b) Thuật toán

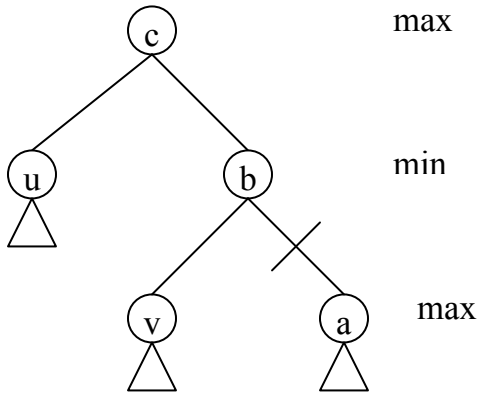
Để hạn chế không gian tìm kiếm, khi xác định nước đi cho Trắng tại u, ta chỉ xem xét cây gốc u tại độ cao h nào đó. Áp dụng thủ tục Minimax cho cây trò chơi gốc u, độ cao h và sử dụng hàm đánh giá để xác định giá trị cho các lá của cây.

```
Procedure Minimax(u, v, h);
begin
    val ←-∝;
    for  mỗi w là đỉnh con của u  do
    if  val(u) <= MinVal(w, h-1)  then
            {val ← MinVal(w, h-1);  v ←  w}
end;
-------------------------------------------------
Function MinVal(u, h); {hàm xác định giá trị cho các đỉnh Đen}
begin
    if  u là đỉnh kết thúc or h = 0 then  MinVal(u, h) ← eval(u)
    else MinVal(u, h) ← min{MaxVal(v, h-1) | v là đỉnh con của u}
end;
-------------------------------------------------
Function MaxVal(u, h); { hàm xác định giá trị cho các đỉnh Trắng}
begin
    if  u là đỉnh kết thúc or h =0  then  MaxVal(u, h) ← eval(u)
    else MaxVal(u, h) ← max{MinVal(v, h-1) | v là đỉnh con của u}
end;
```

4. Giải thuật Minimax với cắt tỉa alpha-beta

Trong chiến lược Minimax với độ sâu hạn chế thì số đỉnh của cây trò chơi phải xét vẫn còn rất lớn với *h*>=3. Khi đánh giá đỉnh *u* tới độ sâu *h*, thuật toán Minimax đòi hỏi phải đánh giá tất cả các đỉnh của cây gốc *u* với độ sâu *h*. Tuy nhiên, phương pháp cắt cụt alpha-beta cho phép cắt bỏ những nhánh không cần thiết cho việc đánh giá đỉnh *u*. Phương pháp này làm giảm bớt số đỉnh phải xét mà không ảnh hưởng đến kết quả đánh giá đỉnh *u*.

Ý tưởng: Giả sử tại thời điểm hiện tại đang ở đỉnh Trắng *a*, đỉnh *a* có anh em là *v* đã được đánh giá. Giả sử cha của đỉnh *a* là *b*, *b* có anh em là *u* đã được đánh giá, và cha của *b* là *c* như hình sau:



Cắt bỏ cây con gốc a nếu eval(u)>eval(v)

Khi đó ta có giá trị đỉnh Trắng *c* ít nhất là giá trị của *u*, giá trị của đỉnh Đen *b* nhiều nhất là giá trị của *v*. Do đó, nếu eval(u) > eval(v) ta không cần đi xuống để đánh giá đỉnh *a* nữa mà vẫn không ảnh hưởng đến đánh giá đỉnh *c*. Hay nói cách khác, ta có thể cắt bỏ cây con gốc *a*.

Lập luận tương tự cho trường hợp *a* là đỉnh Đen, trường hợp này nếu eval(u)<eval(v) ta cũng cắt bỏ cây con gốc *a*.

Để cài đặt kỹ thuật này, đối với các đỉnh nằm trên đường đi từ gốc tới đỉnh hiện thời, ta sử dụng tham số α để ghi lại giá trị lớn nhất trong các giá trị của các đỉnh con đã đánh giá

của một đỉnh Trắng, tham số β để ghi lại giá trị nhỏ nhất trong các giá trị của các đỉnh con đã đánh giá của một đỉnh Đen.

Thuật toán:

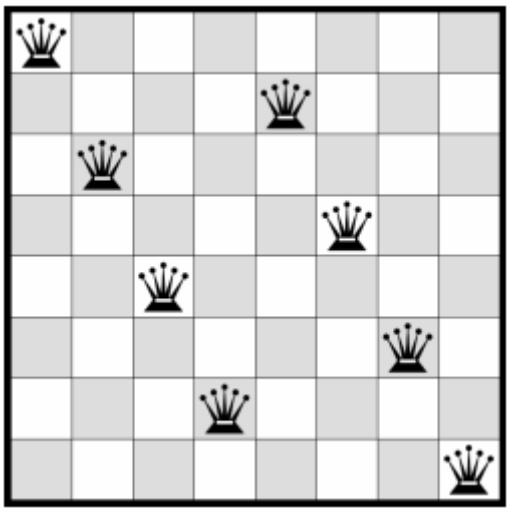
```
Procedure   Alpha_beta(u, v);
begin
      α←-∝; β←-∝;
      for  mỗi w là đỉnh con của u  do
      if  α <= MinVal(w, α, β)  then
            {α ← MinVal(w, α, β);  v ←  w}
end;
-------------------------------------------------
Function MinVal(u, α, β); {hàm xác định giá trị cho các đỉnh Đen}
begin
      if  u là đỉnh kết thúc or u là lá của cây hạn chế then
          MinVal(u, α, β) ← eval(u)
      else for  mỗi đỉnh v là con của u  do
            {β ← min{β, MaxVal(v, α, β)} ;
           If α >= β  then  exit};
      /*cắt bỏ các cây con từ các đỉnh v còn lại */
      MinVal(u, α, β) ← β;
end;
-------------------------------------------------
Function MaxVal(u, α, β); { hàm xác định giá trị cho các đỉnh Trắng}
begin
      if  u là đỉnh kết thúc or là lá của cây hạn chế  then
            MaxVal(u, α, β) ← eval(u)
      Else for mỗi đỉnh v là con của u  do
```

```
            α ← max{α, MinVal(v, α, β)} ;
            If α >= β then exit};
        /*cắt bỏ các cây con từ các đỉnh v còn lại */
        MaxVal(u, α, β) ← α
end;
```

Chương 5 – Các phương pháp tìm kiếm lời giải thỏa mãn các ràng buộc

1. Các bài toán thỏa mãn các ràng buộc

a. Bài toán 8 quân hậu



Hãy đặt trên bàn cờ 8 quân hậu sao cho không có hai quân hậu nào cùng hang hoặc cùng cột hoặc cùng đường chéo.

Bài toán 8 quân hậu có thể biểu diễn bởi 5 thành phần như sau:

- Trạng thái: mảng một chiều 8 phần tử HAU[0,1,…,7], phần tử HAU[i] biểu diễn dòng đặt con hậu cột i. Ví dụ HAU[i]=j có nghĩa là con hậu cột I đặt ở dòng j.
- Trạng thái đầu: Một mảng ngẫu nhiên 8 phần tử, mỗi phần tử nhận giá trị từ 0 đến 7
- Trạng thái đích: Gán các giá trị khác nhau phạm vi từ 0 đến 7 cho các phần tử của mảng sao cho i-HAU[i] ≠ j-HAU[j] (không nằm trên cùng đường chéo phụ) và i+HAU[i] ≠ j + HAU[j] (không nằm trên cùng đường chéo chính).
- Chi phí: không xác định

Trong bài toán này, trạng thái đích là không tường minh mà được xác định bởi tập các ràng buộc. Khác với các bài toán trước, lời giải của bài toán này không phải là đường đi từ trạng thái đầu đến trạng thái đích mà là một phép gán các giá trị cho các biến mô tả trong trạng thái của bài toán sao cho phép gán thỏa mãn các ràng buộc của trạng thái đích.

Để giải các bài toán thỏa mãn các ràng buộc, chúng ta không cần xác định 5 thành phần như các bài toán trong các chương trước, mà chúng ta cần quan tâm đến các thành phần sau:

- Tập các biến mô tả trạng thái của bài toán: HAU[0], HAU[1], .., HAU[7] trong bài toán 8 quân hậu (HAU[i] là số hiệu dòng đặt con hậu ở cột I, ví dụ HAU[0]=0 có nghĩa là con hậu cột đầu tiên (cột 0) sẽ đặt ở dòng đầu tiên (dòng 0).
- Miền giá trị cho các biến: HAU[i] Є {0, 1, 2, 3, 4, 5, 6, 7}
- Tập ràng buộc: với i≠j thì HAU[i] ≠HAU[j] (không có hai con hậu cùng hàng ngang), i-HAU[i] ≠ j-HAU[j] (không có hai con hậu nào cùng đường chéo phụ); i+HAU[i] ≠ j+HAU[j] (không có hai con hậu nào cùng đường chéo chính)

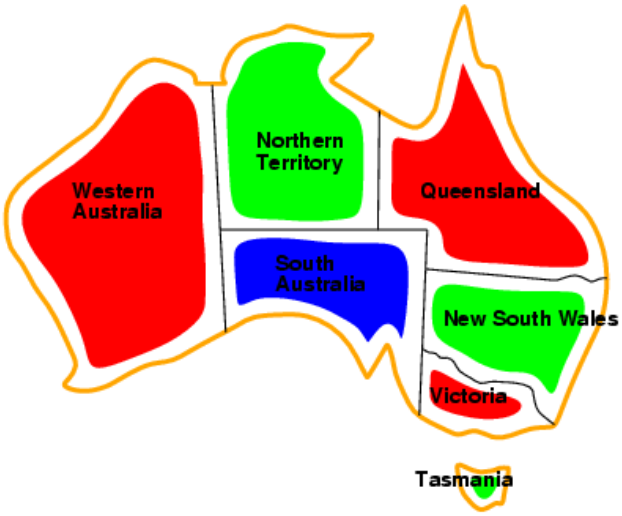
Lời giải của bài toán là một phép gán giá trị trong miền giá trị cho các biến sao cho thỏa mãn các ràng buộc của bài toán.

b. Bài toán tô màu đồ thị

Sử dụng ba màu để tô bản đồ các tỉnh của một nước sao cho các tỉnh kề nhau thì có màu khác nhau. Ví dụ, nước Australia có 7 bang như hình vẽ, chỉ sử dụng ba màu: đỏ, xanh lơ và xanh da trời để tô màu 7 bang của nước Australia sao cho không có hai bang nào kề nhau lại có màu giống nhau. Bài toán này có thể mô tả bằng 3 thành phần như sau:

- Tập các biến: WA, NT, Q, NSW, V, SA, T (các biến là các ký tự đầu của tên các bang)
- Miền giá trị: 7 biến có thể nhận các giá trị trong tập {đỏ, xanh lá cây, xanh da trời}

- Tập ràng buộc: WA≠NT, WA≠SA, NT≠SA, NT≠Q, SA≠Q, SA≠NSW, SA≠V, Q≠NSW, NSW≠V



Lời giải của bài toán tô màu đồ thị là phép gán các giá trị {đỏ, xanh da trời, xanh lá cây} cho tập 7 biến thỏa mãn tập các ràng buộc.

c. Bài toán giải mã các ký tự

Tìm các chữ số thích hợp cho các ký tự để phép tính sau là đúng:

$$\begin{array}{r} \quad T\ W\ O \\ +\ T\ W\ O \\ \hline F\ O\ U\ R \end{array}$$

Bài toán giải mã các ký tự được mô tả bằng 3 thành phần sau:

- Tập các biến: T, W, O, F, U, R, N1, N2, N3 (N1, N2, N3 là 3 số nhớ của phép cộng ở các vị trí hàng đơn vị, hàng chục, hàng trăm)
- Miền giá trị: Các biến có thể nhận các giá trị: {0, 1, .., 9}
- Ràng buộc: T, W, O, F, U, R phải khác nhau đôi một; O + O = X +10.N1; N1 + W + W = U + 10.N2; N2 + T + T = O + 10.N3; F=N3; T≠0; F≠0

Lời giải của bài toán là một phép gán các chữ số từ 0 đến 9 cho các biến và thỏa mãn tập các ràng buộc.

2. Giải thuật quay lui vét cạn

Việc giải bài toán thỏa mãn các ràng buộc là tìm ra một phép gán giá trị cho tập các biến của bài toán sao cho tập các ràng buộc được thỏa mãn. Giả sử bài toán cần gán giá trị cho n biến, chúng ta có thể tìm lời giải của bài toán bằng các bước mô tả như sau:

- Bắt đầu bằng phép gán rỗng, chưa gán giá trị cho biến nào cả { }.

- Nếu tất cả các biến đã được gán giá trị, in ra lời giải và thoát khỏi chương trình

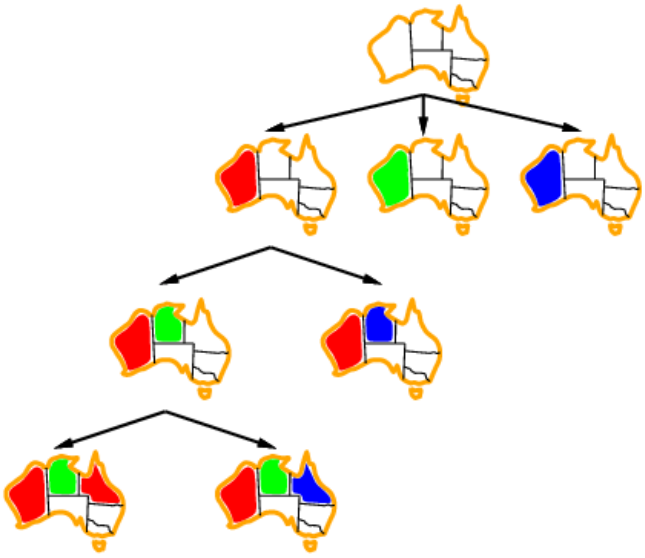
- Tìm giá trị để gán cho biến chưa có giá trị mà không xung đột với các các biến đã được gán trước đó (xung đột hay không là dựa trên tập ràng buộc). Nếu không tìm được giá trị thỏa mãn các ràng buộc cho biến đang xét thì hủy bỏ phép gán giá trị cho biến liền trước đó và tìm giá trí mới cho nó.

- Nếu biến đầu tiên không còn giá trị phù hợp để gán thì bài toán không có lời giải.

Giải thuật gán giá trị cho n biến như trên gọi là giải thuật quay lui vét cạn hay thử và sai (backtracking). Trong giải thuật, mỗi bước thực hiện một phép gán với cách làm giống nhau và lời giải của bài toán chỉ xuất hiện ở bước gán cho biến cuối cùng. Giải thuật trên có thể cài đặt đệ quy như sau:

```
Function Backtracking-Search(problem) returns a solution, or failure
    Return RescusiveBacktracking({},problem);
-------------------------------------------------------
Function RescusiveBacktracking(assignment, problem) returns a solution, or failure
    if (length(assignment)==n) return assignment ;
    var ← Chọn_biến_chưa_gán(problem, assignment);
    for each value in Miền_giá_trị(var,problem)
            if KiemTraNhấtQuán(assignment U{var=value}, problem)
                assignment= assignment U{var=value}
                RescusiveBacktracking(assignment, problem);
                assignment= assignment - {var=value}
    return failure;
```

Bản chất của giải thuật RescusiveBacktracking là phép duyệt theo chiều sâu có thêm bước kiểm tra sự thỏa mãn của các ràng buộc ở mỗi bước. Thứ tự việc gán giá trị cho các biến trong bài toán tô màu đồ thị có thể biểu diễn bằng đồ thị sau:



Một phần đồ thị biểu diễn thứ tự phép gán giá trị
cho các biến của giải thuật Backtracking

3. *Các cải tiến của giải thuật quay lui*

Trong giải thuật RescusiveBacktracking ở trên, thứ tự các biến có thể ảnh hưởng đến thời gian và không gian bộ nhớ của giải thuật. Chúng ta có thể thay đổi thứ tự các biến để gán giá trị, và khi biến được chọn, chúng ta có thể chọn giá trị nào trước các giá trị khác trong các giá trị hợp lệ để gán cho biến đó. Đôi khi thứ tự các biến và thứ tự các giá trị gán cho các biến làm tăng đáng kể hiệu quả của giải thuật.

a) Nguyên tắc chọn biến tiếp theo

Vì lời giải của bài toán chỉ xuất hiện ở mức độ sâu n trong giải thuật đệ qui, vì vậy ResicusiveBacktracking ưu tiên phát triển theo chiều sâu để tìm ra phép gán đầy đủ (tức là lời giải) của bài toán trong thời gian nhanh nhất. Khi một số biến được gán giá trị, miền giá trị của các biến còn lại cũng sẽ bị co hẹp lại do tập các ràng buộc chi phối. Vì

thế, để có thể tìm kiếm được phép gán có độ sâu n nhanh nhất mà không bị hủy bỏ để gán lại giá trị cho biến thì có 2 nguyên tắc sau:

- Nguyên tắc 1: Lựa chọn biến mà miền giá trị hợp lệ còn lại là ít nhất (biến có ít lựa chọn nhất nên được chọn trước để làm giảm độ phức tạp của cây tìm kiếm)
- Nguyên tắc 2: Lựa chọn biến tham gia vào nhiều ràng buộc nhất (gán cho biến khó thỏa mãn nhất)

Trong hai nguyên tắc trên, nguyên tắc thứ nhất được ưu tiên cao hơn và được áp dụng trong suốt quá trình thực hiện của giải thuật. Đối với phép chọn biếu đầu tiên hoặc trong trường hợp có nhiều biến có cùng số giá trị ít nhất thì nguyên tắc thứ hai sẽ được sử dụng để lựa chọn biến tiếp theo.

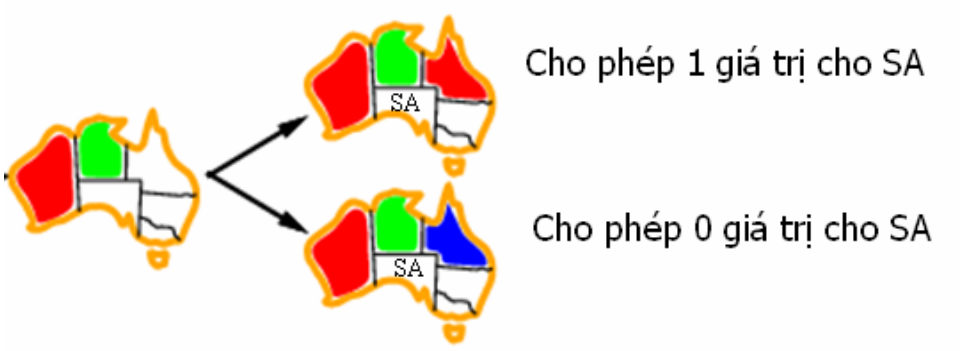
Ví dụ, đối với bài toán tô màu đồ thị, ban đầu chúng ta chọn biến SA để gán giá trị vì SA tham gia vào nhiều mối ràng buộc hơn (nguyên tắc 2). Khi chọn màu biến cho SA thì các biến WA, NT, Q, NSW,V sẽ được chọn ở bước gán tiếp theo do chỉ còn 2 lựa chọn là hai màu còn lại (nguyên tắc 1), trong 5 biến này ta lại lấy biến NT, Q hoặc NSW vì nó tham gia vào nhiều ràng buộc hơn (có thể chọn 1 trong ba biến này ngẫu nhiên). Cứ như vậy chúng ta sẽ chọn thứ tự các biến còn lại dựa trên Nguyên tắc 1, nếu có nhiểu biến cùng thỏa mãn nguyên tắc 1 thì chọn trong chúng biến thỏa mãn Nguyên tắc 2.

b) Nguyên tắc chọn thứ tự giá trị gán cho biến

Một khi một biến được lựa chọn để gán giá trị thì sẽ có nhiều giá trị có thể gán cho biến đó. Việc lựa chọn thứ tự giá trị gán cho biến có tác động không nhỏ trong việc tìm ra lời giải đầu tiên. Trong trường hợp bài toán cần tìm tất cả lời giải hoặc bài toán không có lời giải thì thứ tự các giá trị gán cho biến không có tác dụng.

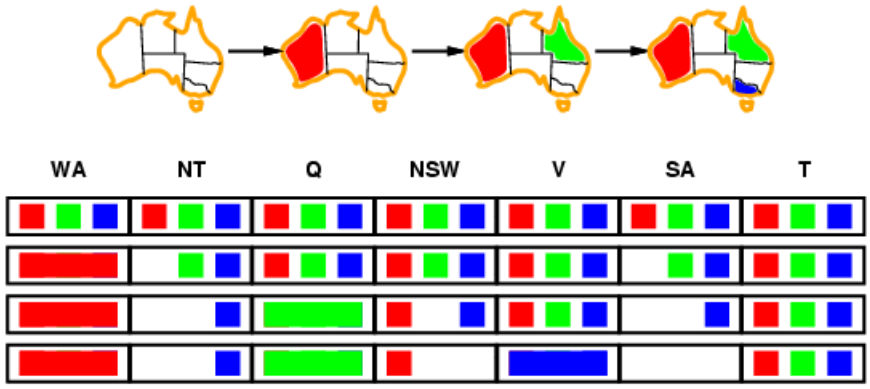
Trong trường hợp bài toán yêu cầu tìm ra một lời giải và chúng ta mong muốn tìm ra lời giải trong thời gian nhanh nhất thì chúng ta sẽ lựa chọn giá trị cho biến đang xét sao cho nó ít ràng buộc đến các biến còn lại nhất. Ví dụ: nếu ta đã chọn WA=đỏ, NT=xanh da trời và chúng ta đang xem xét gán giá trị cho biến Q. Có 2 giá trị có thể gán cho Q mà không bị xung đột với hai phép gán trước: đỏ và xanh da trời. Trong 2 cách này thì nếu gán xanh

da trời thì làm cho biến SA không còn giá trị để gán, còn nếu gán màu đỏ thì sẽ có 1 giá trị có thể gán cho biến SA. Vậy trong trường hợp này ta sẽ gán màu đỏ cho biến Q để tăng khả năng tìm được lời giải đầu tiên.



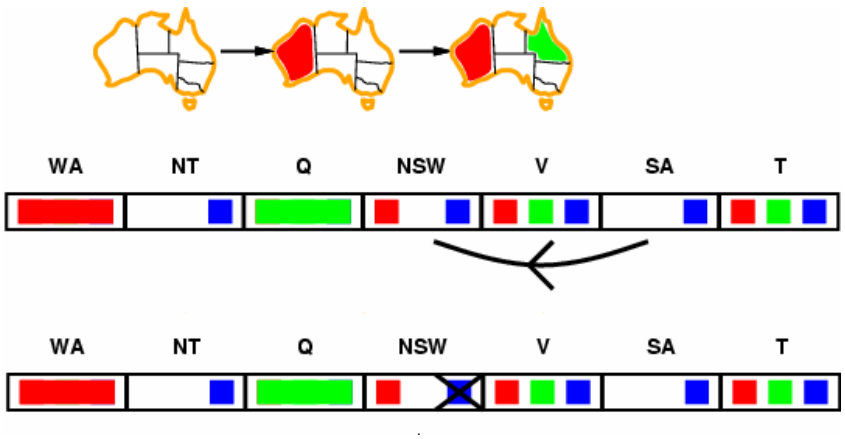
c) Kiểm tra tiến (kiểm tra trước – forward checking)

Trong nguyên tắc chọn thứ tự giá trị gán cho một biến, chúng ta cần phải kiểm tra xem giá trị định gán sẽ tác động thế nào đối với các biến chưa gán thông qua các ràng buộc. Việc hạn xác định tác động trước như vậy gọi là forward checking. Forward checking còn có tác dụng hạn chế không gian tìm kiếm (hạn chế miền giá trị cho các biến còn lại khi biến hiện tại được gán một giá trị cụ thể). Ví dụ, nếu ban đầu chúng ta gán WA màu đỏ thì miền giá trị của các bang lân cận (NT và SA) sẽ không thể là màu đỏ được nữa. Nếu gán tiếp Q là màu xanh lá cây thì NT và SA chỉ còn nhận giá trị là xanh da trời và NSW chỉ còn miền giá trị là màu đỏ (xem diễn biến miền giá trị các biến thu hẹp dần trong quá trình gán giá trị cho biến WA và Q)



d) Lan truyền ràng buộc (constraint propagation)

Trong quá trình gán giá trị cho biến, nếu một biến có mà miền giá trị của nó không còn giá trị nào hợp lệ để gán thì chúng ta phải hủy bỏ việc gán giá trị cho biến ngay trước đó và gán bằng giá trị khác. Nếu một trong các biến còn lại mà miền giá trị chỉ 1 giá trị hợp lý thì chúng ta có thể áp dụng tập các ràng buộc liên quan đến biến đó để giảm miền giá trị cho biến còn lại khác. Chẳng hạn, bằng forward checking chúng ta đã xác định được biến SA chỉ có giá trị màu xanh da trời thì chúng ta áp dụng các ràng buộc liên quan đến SA để suy ra rằng biến NSW không thể nhận giá trí màu xanh da trời. Khi đó NSW chỉ còn màu đỏ và áp dụng các ràng buộc liên quan đến NSW suy ra V không thể nhận màu đỏ, v.v. Quá trình loại bỏ miền giá trị cho các biên còn lại dựa trên các ràng buộc gọi là lan truyền ràng buộc nhằm giảm bớt không gian tìm kiếm phép gán hợp lệ.



4. Các giải thuật tối ưu địa phương

Chương 6 – 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:

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

clause clause 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=¬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,…,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:

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

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) 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 Y new_clause;
        if new⊆clauses return false;
        clauses=clauses Y 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 7 – 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 Y {r/s}
        Đồng_nhất(thaythe(theta,p), thaythe(theta,q), theta)
    elseif (là_biến(s))
        theta = theta Y {s/r}
        Đồng_nhất(thaythe(theta,p), thaythe(theta,q), theta)
    else return failure
```

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

a) 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)

b) 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)

c) 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)

d) 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 Y Q';

KB $\leftarrow$KB Y new;

return false;

8. Thuật toán suy diễn lùi dựa trên câu Horn

Chương 8 – Prolog

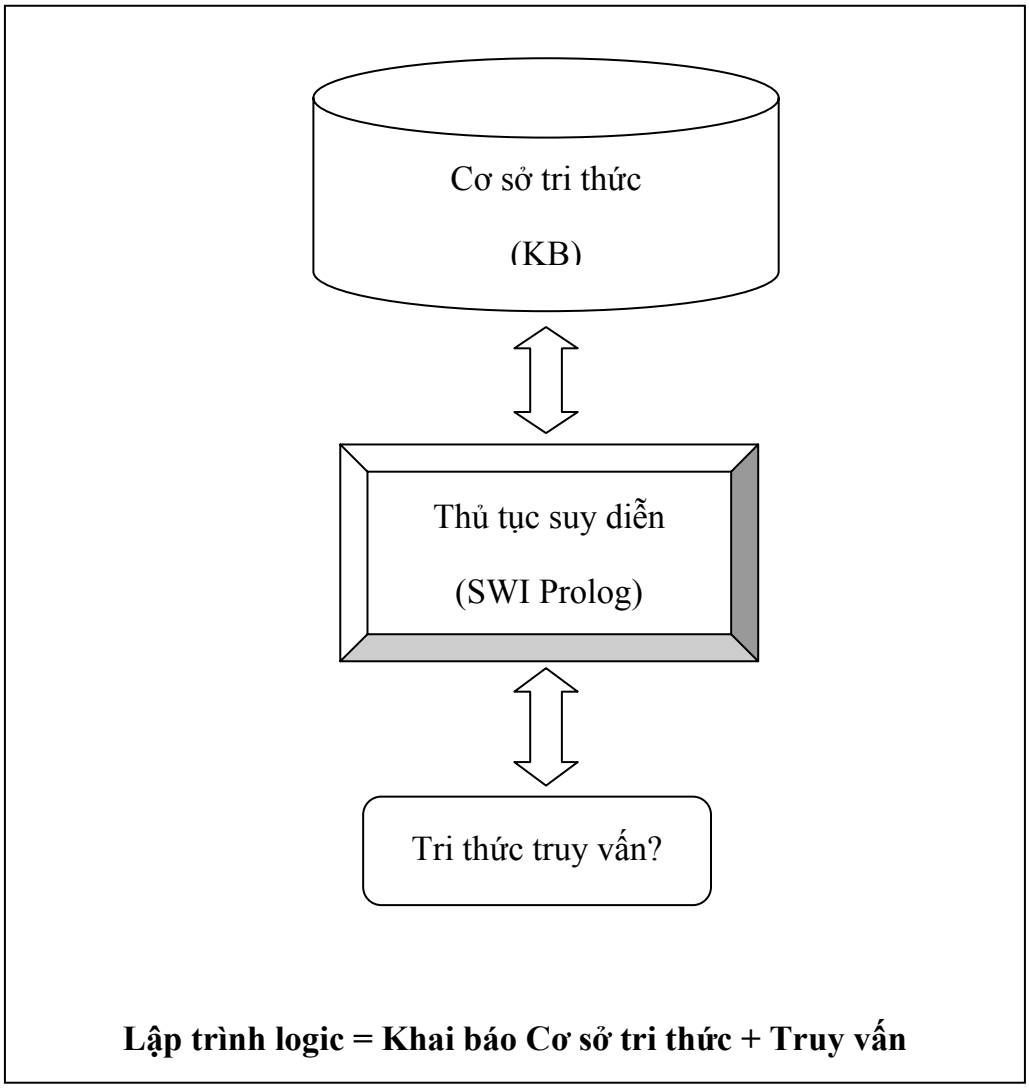
Trong Chương 4 và 5 chúng ta đã tìm hiểu logic mệnh đề và logic vị từ cấp một. Chúng ta cũng đã tìm hiểu các thuật toán lập luận tự động, chứng minh câu truy vấn q từ cơ sở tri thức KB. Có hai loại thuật toán lập luận cơ bản: (1) Lập luận trong các câu dạng chuẩn hội với luật phân giải, và (2) Lập luận 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:- p1, p2, …, pn.
{nghĩa là: if (p1 and p2 and … and pn) 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:

$$p_1, p_2, \ldots, p_n.$$

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 $\forall x,y,\ldots (\neg p_1 \vee \neg p_2 \vee \ldots \vee \neg p_n)$. (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.

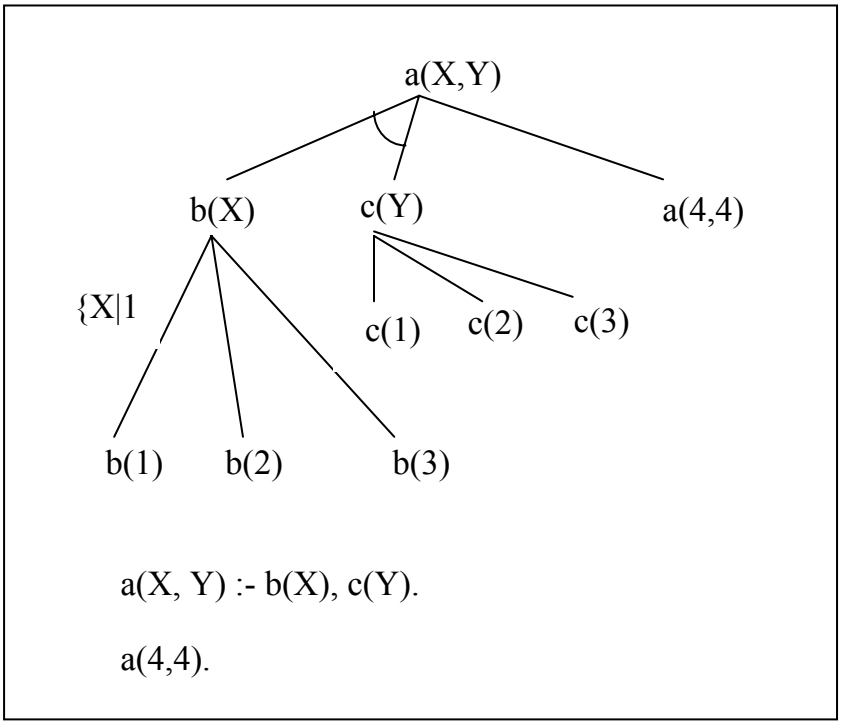
Vị từ Cắt (!):

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).

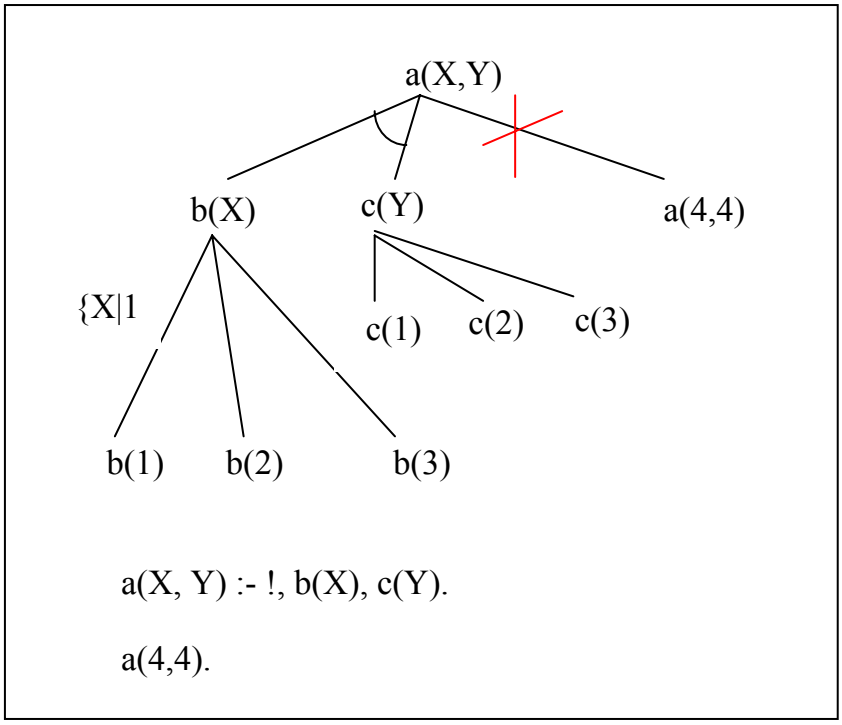
bằng một trong các câu lệnh dưới đây (chèn vị từ ngắt ! vào các vị trí khác nhau):

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

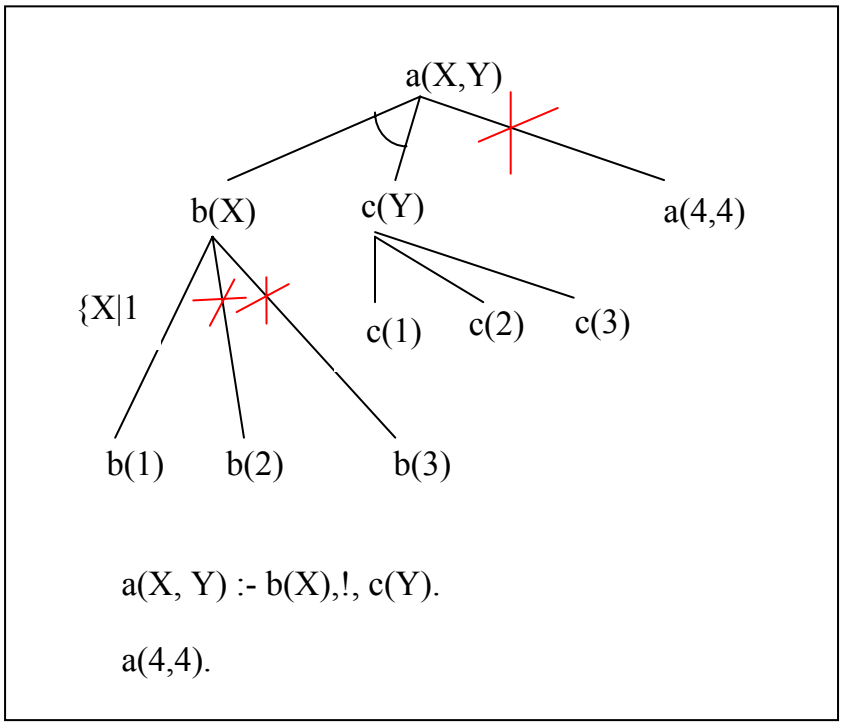
a(X, Y) :- b(X), c(Y),!. % không quay lui đối với vị từ a,b,c

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



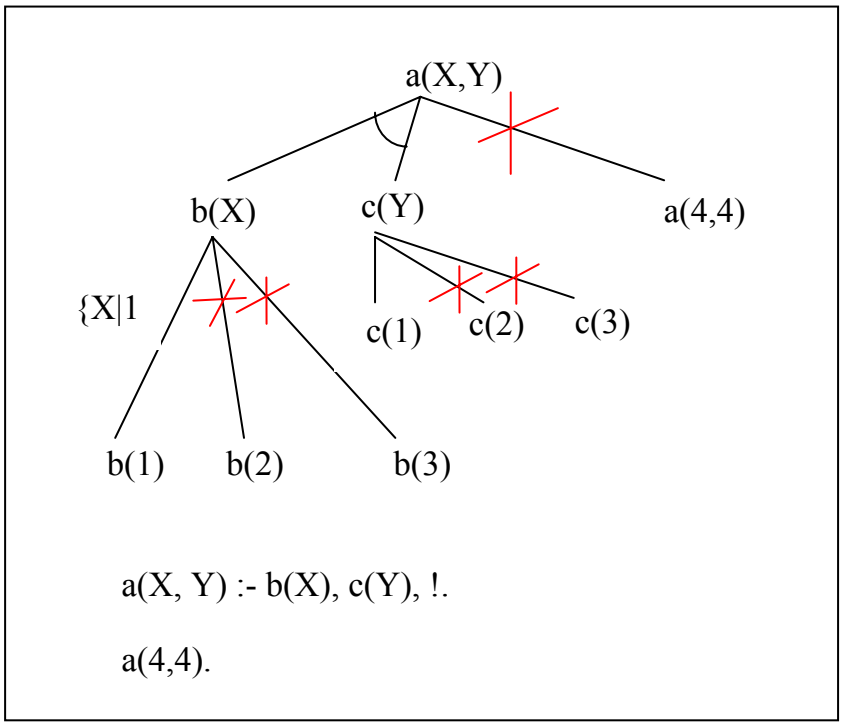
a(X, Y) :- !, b(X), c(Y).

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 9 – 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 10 – 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).