

HỌC VIỆN CÔNG NGHỆ BƯU CHÍNH VIỄN

THÔNG



# NHẬP MÔN TRÍ TUỆ NHÂN TẠO

cuu duong than cong . com

*(Dùng cho sinh viên hệ đào tạo đại học từ xa)*

Lưu hành nội bộ

cuu duong than cong . com

HÀ NỘI - 2007

**NHẬP MÔN**

**TRÍ TU** [cuu duong than cong . com](https://fb.com/tailieudientuontt) **Ệ**  
**NHÂN TẠO**

**Biên soạn : PGS.TS. NGUYỄN QUANG HOAN**

[cuu duong than cong . com](https://fb.com/tailieudientuontt)

## LỜI NÓI ĐẦU

Trí tuệ nhân tạo (hay AI: Artificial Intelligence), là nỗ lực tìm hiểu những yếu tố trí tuệ. Lý do khác để nghiên cứu lĩnh vực này là cách để ta tự tìm hiểu bản thân chúng ta. Không giống triết học và tâm lý học, hai khoa học liên quan đến trí tuệ, còn AI cố gắng thiết lập các yếu tố trí tuệ cũng như tìm biết về chúng. Lý do khác để nghiên cứu AI là để tạo ra các thực thể thông minh giúp ích cho chúng ta. AI có nhiều sản phẩm quan trọng và đáng lưu ý, thậm chí ngay từ lúc sản phẩm mới được hình thành. Mặc dù không dự báo được tương lai, nhưng rõ ràng máy tính điện tử với độ thông minh nhất định đã có ảnh hưởng lớn tới cuộc sống ngày nay và tương lai phát triển của văn minh nhân loại.

Trong các trường đại học, cao đẳng, Trí tuệ nhân tạo đã trở thành một môn học chuyên ngành của sinh viên các ngành Công nghệ Thông tin. Để đáp ứng kịp thời cho đào tạo từ xa, Học viện Công nghệ Bưu chính Viễn thông biên soạn tài liệu này cho sinh viên, đặc biệt hệ Đào tạo từ xa học tập. Trong quá trình biên soạn, chúng tôi có tham khảo các tài liệu của Đại học Bách khoa Hà nội [1] giáo trình gần gũi về tính công nghệ với Học viện. Một số giáo trình khác của Đại học Quốc gia thành phố Hồ Chí Minh [], tài liệu trên mạng và các tài liệu nước ngoài bằng tiếng Anh [] cũng được tham khảo và giới thiệu để sinh viên đào tạo từ xa đọc thêm.

Tài liệu này nhằm hướng dẫn và giới thiệu những kiến thức cơ bản, các khái niệm, định nghĩa tóm tắt. Một số thuật ngữ được chú giải bằng tiếng Anh để học viên đọc bằng tiếng

Anh dễ dàng, tránh hiểu nhầm khi chuyển sang tiếng Việt.

Tài liệu gồm các chương sau:

- Chương 1 : Khoa học Trí tuệ nhân tạo: tổng quan
- Chương 2 : Các phương pháp giải quyết vấn đề
- Chương 3 : Biểu diễn tri thức và suy diễn
- Chương 4 : Xử lý ngôn ngữ tự nhiên
- Chương 5 : Các kỹ thuật trí tuệ nhân tạo hiện đại

Còn nhiều vấn đề khác chưa đề cập được trong phạm vi tài liệu này. Đề nghị các bạn đọc tìm hiểu thêm sau khi đã có những kiến thức cơ bản này.

Nhiều cố gắng để cập nhật kiến thức nhưng thời gian, điều kiện, khả năng có hạn nên tài liệu chắc chắn còn nhiều thiếu sót. Chúng tôi mong nhận được nhiều ý kiến đóng góp để tài liệu được hoàn thiện hơn cho các lần tái bản sau.

## CHƯƠNG 1: KHOA HỌC TRÍ TUỆ NHÂN TẠO: TỔNG QUAN

**Học xong phần này sinh viên có thể nắm được:**

1. Ý nghĩa, mục đích môn học; lịch sử hình thành và phát triển. Các tiền đề cơ bản của Trí tuệ nhân tạo (TTNT)
2. Các khái niệm cơ bản, định nghĩa của TTNT.
3. Các lĩnh vực nghiên cứu và ứng dụng cơ bản. Những vấn đề chưa được giải quyết trong TTNT

### 1.1 LỊCH SỬ HÌNH THÀNH VÀ PHÁT TRIỂN

Trong phần này chúng tôi nỗ lực giải thích tại sao chúng tôi coi trí tuệ nhân tạo là một bộ

môn đáng nghiên cứu nhất; và nỗ lực của chúng tôi nhằm giải thích trí tuệ nhân tạo là gì. Đây có phải là bộ môn hấp dẫn khi nghiên cứu không.

Trí tuệ nhân tạo hay AI (Artificial Intelligence) là một trong những ngành tiên tiến nhất. Nó chính thức được bắt đầu vào năm 1956, mặc dù việc này đã bắt đầu từ 5 năm trước. Cùng với ngành di truyền học hiện đại, đây là môn học được nhiều nhà khoa học đánh giá: “là lĩnh vực tôi thích nghiên cứu nhất trong số những môn tôi muốn theo đuổi”. Một sinh viên vật lý đã có lý khi nói rằng: tất cả các ý tưởng hay đã được Galileo, Newton, Einstein tìm rồi; một số ý tưởng khác lại mất rất nhiều năm nghiên cứu trước khi có vai trò thực tiễn. AI vẫn là vấn đề để trông từ thời Einstein.

Qua hơn 2000 năm, các triết gia đã cố gắng để hiểu cách nhìn, học, nhớ và lập luận được hình thành như thế nào. Sự kiện những chiếc máy tính có thể sử dụng được vào đầu những năm 50 của thế kỉ XX đã làm các nhà tri thức thay đổi hướng suy nghĩ. Rất nhiều người cho rằng: “những trí tuệ siêu điện tử” mới này đã cho ta dự đoán được tiềm năng của trí tuệ. AI thực sự khó hơn rất nhiều so với ban đầu mọi người nghĩ.

Hiện nay AI đã chuyển hướng sang nhiều lĩnh vực nhỏ, từ các lĩnh vực có mục đích chung chung như nhận thức, lập luận, tư duy logic đến những công việc cụ thể như đánh cờ, cung cấp định lý toán học, làm thơ và chuẩn đoán bệnh. Thường, các nhà khoa học trong các lĩnh vực khác cũng nghiêng về trí tuệ nhân tạo. Trong lĩnh vực này họ thấy các phương tiện làm việc, vốn từ vựng được hệ thống hoá, tự động hoá: các nhiệm vụ trí tuệ là công việc mà họ sẽ có thể cống hiến cả đời. Đây thực sự là một ngành rất phổ biến.

### 1.1.1. Tư duy như con người: phương pháp nhận thức

Nếu muốn một chương trình máy tính có khả năng suy nghĩ như con người, chúng ta phải tìm hiểu con người đã tư duy như thế nào? Có một số tiêu chí xác định như thế nào là suy nghĩ kiểu con người. Chúng ta cần xem công việc bên trong của bộ óc con người. Có hai phương pháp để thực hiện điều này: thứ nhất là thông qua tư duy bên trong - phải nắm bắt được suy nghĩ của con người khi làm việc - thứ hai thông qua thí nghiệm tâm lý. Khi chúng ta đã có được đầy đủ lý thuyết về tư duy thì chúng ta có thể chương trình hoá nó trên máy tính. Nếu đầu vào/ra của chương trình và thời gian làm việc phù hợp với con người thì những chương trình tự động này có thể hoạt động theo con người. Ví dụ, Newell và Simon đã phát triển phương pháp giải quyết vấn đề GPS- General Problem Solver (Newell and Simon 1961). Đây là phương pháp đối lập với các

nghiên cứu đương thời (như Wang (1960)) ông quan tâm đến việc có được những giải pháp đúng đắn, không quan tâm đến việc con người phải làm như thế nào.

### 1.1.2. Các qui tắc tư duy

Triết gia Aristote là người đầu tiên hệ thống hoá “tư duy chính xác”. Phép tam đoạn luận của ông đưa ra kết luận đúng nếu cả tiền đề chính và tiền đề thứ là đúng. Chẳng hạn: “nếu Sô-crát là con người, mọi con người đều chết, như vậy Sô-crát sẽ chết”.

Môn tư duy logic phát triển vào cuối thế kỉ XIX đầu XX. Năm 1965 các chương trình cung cấp cho chúng ta đủ những thông tin, chi tiết về một vấn đề trong tư duy logic và tìm ra phương pháp giải. Nếu vẫn còn vấn đề chưa có cách giải thì chương trình sẽ không ngừng tìm kiếm cách giải. Môn logic truyền thống trong AI là điều mong mỏi để có được một chương trình mô tả hệ

thông trí tuệ.

### 1.1.3. Khởi nguồn của AI (1943 - 1956)

Những công việc đầu tiên của AI được Warren McCulloch và Walter Pitts (1943) thực hiện. Họ đã nghiên cứu ba cơ sở lý thuyết: triết học cơ bản và chức năng của các nơ ron thần kinh; phân tích về các mệnh đề logic là của Russell và Whitehead và cuối cùng là thuyết dự đoán của Turing. Họ đã đề ra mô hình nơ ron nhân tạo, trong đó mỗi nơ ron được đặc trưng bởi hai trạng thái “bật”, “tắt”. McCulloch và Pitts cũng đã phát hiện: mạng nơ ron có khả năng học. Donald Hebb (1949) sử dụng luật học đơn giản tượng trưng cho việc truyền thông tin giữa các nơ ron.

Đầu những năm 1950, Claude Shannon (1950) và Alan Turing (1953) đã viết chương trình đánh cờ theo cách mà Von Neuman sáng chế ra máy tính. Cùng lúc đó, hai sinh viên khoa toán trường đại học Princeton, Marvin Minsky và Dean Edmond đã xây dựng hệ thống máy tính nơ ron đầu tiên vào năm 1951 được gọi là SNARC. Nó sử dụng khoảng 3000 bóng điện tử chân không và thiết bị cơ khí tự động tính giá trị thắng dư từ chùm B-24 để mô phỏng mạng với 40 nơ ron. Nhóm thạc sĩ của Minsky nghi ngờ rằng liệu đây có được coi là một phần của toán học, nhưng Neuman một thành viên của nhóm đã cho biết rằng “nếu bây giờ nó không phải là một phần của toán học thì một ngày nào đó nó sẽ là như thế”. Thật mỉa mai, sau này Minsky lại chính là người chứng minh học thuyết này và đã bác bỏ nhiều hệ thống nghiên cứu về mạng nơ ron trong suốt những năm 1970.

### *Lòng say mê và tôn trọng lớn ngay từ rất sớm (1952-1969)*

Năm 1958 McCarthy đã định nghĩa ngôn ngữ bậc cao Lisp, và trở thành ngôn ngữ lập trình cho AI. Lisp là ngôn ngữ lập trình lâu đời thứ hai mà hiện nay vẫn sử dụng. Với Lisp, McCarthy đã có phương tiện ông cần, nhưng để đáp ứng được yêu cầu và tài nguyên tính toán là một vấn đề quan trọng. Cũng vào năm 1958, McCarthy xuất bản bài báo “Các chương trình với cách nhìn nhận chung”. Trong bài báo này, ông bàn về chương trình tư vấn, một chương trình giả định được coi là hệ thống AI hoàn thiện đầu tiên. Giống học thuyết logic và cách chứng minh các định lý hình học, chương trình của McCarthy được thiết kế nhằm sử dụng kiến thức để nghiên cứu cách giải quyết vấn đề. Không như các chương trình khác, chương trình này là một bộ phận kiến thức của toàn bộ thế giới quan. Ông chỉ ra rằng làm thế nào để những điều rất đơn giản lại làm cho chương trình có thể khái quát được một kế hoạch đến sân bay và lên máy bay. Chương trình này cũng được thiết kế để nó có thể chấp nhận vài chân lý mới về quá trình thực hiện bình thường. Chính vì vậy, chương trình này có được những khả năng thực hiện trong các chương trình mới mà không cần lập trình lại.

Năm 1963, McCarthy đã có các nghiên cứu về sử dụng logic để xây dựng chương trình người tư vấn. Chương trình này được phát triển do khám phá của Robinson về phương pháp cải cách. Những công việc đầu tiên tạo ra những hệ thống mới của McCulloch và Pitts làm cho chúng phát triển. Các phương pháp nghiên cứu của Hebb đã được Widrow ủng hộ (Widrow và Hoff, 1960; Widrow, 1962). Họ đã đặt tên mạng nơ ron là mạng của ông, và cũng được Frank Rosenblatt (1962) củng cố. Rosenblatt chứng minh rằng thuật toán mà ông nghiên cứu có thể thêm vào những khả năng của nhận thức phù hợp với bất cứ dữ liệu đầu vào nào. Những nhà

nghiên cứu AI cũng đã dự đoán về những thành công sau này. Herbert Simon đã phát biểu (1957): Không phải mục đích của tôi là làm các bạn ngạc nhiên, nhưng cách đơn giản nhất để có thể khái quát là hiện nay trên thế giới, máy móc có thể suy nghĩ, có thể học và sáng tạo được. Hơn nữa, khả năng của nó là làm việc với tiến độ cao- trong tương lai rõ ràng – cho đến khi vấn đề chúng ta có thể giải được, sẽ cùng tồn tại với tư duy của con người có thể áp dụng được. Năm 1958, ông dự đoán trong 10 năm nữa, một máy tính có thể vô địch trong môn cờ vua, và các định lý toán học mới sẽ được máy chứng minh.

## 1.2. CÁC TIỀN ĐỀ CƠ BẢN CỦA TTNT

Toàn cảnh về phương pháp giải quyết vấn đề hình thành trong thập kỉ đầu nghiên cứu AI là mục đích nghiên cứu nỗ lực liên kết các bước lập luận cơ bản với nhau để tìm ra phương pháp hoàn thiện. Các phương pháp này được coi là các phương pháp kém vì sử dụng thông tin kém về lĩnh vực. Đối với nhiều lĩnh vực phức tạp, thì các phương pháp thực hiện lại rất kém. Cách duy nhất quanh vấn đề là sử dụng kiến thức phù hợp hơn để có bước lập luận rộng hơn và để giải quyết các

trường hợp nảy sinh nhất định trong các lĩnh vực nhỏ chuyên môn. Chúng ta chắc sẽ nói rằng giải quyết các vấn đề khó thì hầu như phải biết trước đáp án.

Chương trình DENDRAL (Buchanan, 1969) là một ví dụ sớm tiếp cận phương pháp này. Nó được phát triển tại Stanford, đây chính là nơi Ed Feigenbaum (một sinh viên chính qui của Herbert Simon). Bruce Buchanan (một triết gia chuyển sang làm nghiên cứu máy tính) và Joshua Lederberg (nhà nghiên cứu di truyền đoạt giải Nobel) đã hợp nhau lại để cùng suy luận, giải quyết vấn đề có cấu trúc phân tử từ những thông tin do máy đo quang phổ cung cấp. Dữ liệu đưa vào chương trình gồm các cấu trúc cơ bản của phân tử (Ví dụ  $C_6H_{12}NO_2$ ), và rất nhiều dải quang phổ đưa ra hàng loạt đoạn phân tử khác nhau khái quát chung khi nó cùng một lúc đưa ra các dòng điện tử. Ví dụ dải quang phổ chứa đựng một điểm nhọn tại  $m=15$  tương ứng với một dải của đoạn methyl ( $CH_3$ ).

Phiên bản sơ khai của chương trình khái quát được toàn bộ cấu trúc có thể bên trong bằng phân tử và sau đó phỏng đoán bằng cách quan sát mỗi dải quang phổ, so sánh nó với quang phổ thực tế. Như chúng ta nghĩ thì điều này trở nên nan giải đối với các phân tử có kích thước đáng kể. Các nhà nghiên cứu DENDRAL khuyên các nhà phân tích được khoa và cho thấy rằng họ nghiên cứu bằng cách tìm kiếm các phần bên trên của điểm nhọn trong dải quang phổ, điều đó đưa ra gợi ý chung về các cấu trúc nhỏ bên trong phân tử. Ví dụ, qui luật sau đây được sử dụng để nhận ra một nhóm nhỏ xeton ( $C=O$ )

Nếu có hai đỉnh  $x_1, x_2$  như sau:

(a)  $x_1+x_2 = M+28$  ( $M$  là khối lượng của phân tử)

(b)  $x_1-28$  là một đỉnh

(c)  $x_2-28$  là một đỉnh

(d) Có ít nhất một đỉnh  $x_1$  hoặc  $x_2$  là đỉnh cao. Sau đó có một nhóm nhỏ xeton. 5

Khi nhận ra phân tử chứa một cấu trúc nhỏ đặc biệt, số lượng thành phần tham gia có thể bị giảm xuống nhanh chóng. Nhóm DENDRAL kết luận rằng hệ thống mới là rất mạnh bởi vì: toàn bộ kiến thức có liên quan đến giải quyết công việc đã được phác thảo sơ qua từ cấu trúc chung trong [thành phần quang phổ đoán trước] để có những cấu trúc đặc biệt

Tầm quan trọng của DENDRAL là nó là hệ thống cảm nhận kiến thức thành công đầu tiên. Các chuyên gia của lĩnh vực này đi sâu từ số lượng lớn các qui luật có mục đích đặc biệt. Các hệ thống sau này cũng không kết hợp lại thành chủ đề chính của phương pháp chuyên gia của McCarthy - phần hoàn toàn tách biệt của kiến thức (trong cấu trúc của qui luật) và thành phần lập luận.

Với bài học này, Feigenbaum và các thành viên khác tại Stanford bắt đầu lập dự án chương trình Heuristic, để đầu tư mở rộng vào các phương pháp mới của hệ chuyên gia nhằm áp dụng vào các lĩnh vực khác nhau. Những nỗ lực chính sau đó là chuẩn đoán y học. Feigenbaum, Buchanan và Edward Shortliffe đã phát triển hệ chuyên gia MYCIN để chẩn đoán bệnh nhiễm trùng máu. Với khoảng 450 luật, hệ chuyên gia MYCIN có thể thực hiện tốt hơn nhiều bác sĩ mới. Nó có hai sự khác biệt cơ bản với hệ chuyên gia DENDRAL. Thứ nhất: không giống như các luật DENDRAL, không một mẫu lý thuyết chung nào tồn tại mà có thể suy luận từ các luật của hệ MYCIN. Các luật phải có câu chất vấn của chuyên gia, người có nhiệm vụ tìm chúng từ kinh nghiệm. Thứ hai: các luật phản ánh mối liên quan không chắc chắn với kiến thức y học. MYCIN kết hợp với hệ vi phân của biến số được coi là các nhân tố phù hợp tốt (ở mọi lúc) với phương pháp mà các bác sĩ tiếp cận với các triệu chứng trong quá trình chuẩn đoán.

chuẩn  
nghiên

Cách tiếp cận khác để  
đoán y học cũng được  
cứu. Tại trường đại

học Rutgers, những máy tính trong ngành sinh hoá của Saul Amarel bắt đầu tham vọng nhằm cố gắng chuẩn đoán bệnh tật dựa trên kiến thức được biểu đạt rõ ràng của những chiếc máy phân tích quá trình bệnh tật. Trong khi đó, một số nhóm lớn hơn tại MIT và trung tâm y tế của Anh đang tiếp tục phương pháp chuẩn đoán và điều trị dựa trên học thuyết có tính khả thi và thực tế. Mục đích của họ là xây dựng các hệ thống có thể đưa ra các phương pháp chẩn đoán y học. Về y học, phương pháp Stanford sử dụng các qui luật do các bác sĩ cung cấp ngay từ đầu đã được chứng minh là phổ biến hơn. Nhưng hệ chuyên gia PROSPECTOR (Duda 1979) được công bố cho mọi người bằng cách giới thiệu thiết bị khoan thăm quặng

Một vài ngôn ngữ dựa vào logic như ngôn ngữ Prolog phổ biến ở châu Âu, và PLANNER ở Mỹ. Các ngôn ngữ khác, theo sau các ý tưởng của Minsky (1975) chấp nhận phương pháp tiếp cận cấu trúc, thu thập các chứng cứ về đối tượng và các loại sự kiện.

### 1.3. CÁC KHÁI NIỆM CƠ BẢN

#### 1.3.1. Trí tuệ nhân t

ạo(AI) là

gì? Chúng ta có thể nói: “Tuyệt thật, đây là một chương trình được thực hiện bằng những suy

diễn thông minh, vì thế cần phải tiếp tục và mọi người cần bổ sung cho nó”. Nhưng theo sự phát triển của khoa học cho thấy: sẽ có ích nếu ta đi đúng hướng. Định nghĩa về AI đã có tới tám cuốn sách đề cập. Những định nghĩa đó đưa ra trên hai nhận định chính:

- Thứ nhất: quan tâm chủ yếu đến quá trình tư duy và lập luận
- Thứ hai: vấn đề ít được quan tâm hơn, đó là hoạt động.

Một hệ thống được coi là hợp lý nếu như nó thực hiện đúng. Điều này sẽ đưa ngành AI đến 4 mục tiêu.(xem Bảng 1.1).

Chúng ta sẽ đi vào chi tiết của từng hướng theo các phát biểu sau đây: 6



**“Nh ng n l c thú v m i ây l t o ra máy tính... nh ng máy móc có trí tu , hi u theo c nghĩã y l n ngh a bóng”.** (Haugeland, 1985)

**“[s t ng hoá c a] các ho t ng ã giúp chúng ta k t h p nh ng t duy c a con ng i v i công vi c c ng nh quy t nh, gi i quy t v n , h c t p...”** (Bellman 1978)

**“Ngh thu t sáng t o máy móc l th c hi n ch c n ng hình th nh t duy khi con ng i l m vi c”**  
(Kurzweil, 1990)

**“Vi c nghiên c u l m cách n o b t máy tính l m nh ng vi c m cùng m t lúc con ng i có th l m t t h n.”**  
(Rich and Knight, 1991)

**d ng máy vi tính”** (Charniak and McDermott, 1985) **“Nghiên c u máy tính l m cho máy tính có kh n ng c m nh n, l p lu n v l m vi c”.**

**(Winston, 1992)**

**“Trong l nh v c nghiên c u l tìm ra cách gi i thích v t c nh ng h nh ng có t duy trong “l nh v c x lý tính toán”.**  
(Schalkoff, 1990)

**“Trong ng nh khoa h c máy tính có liên quan n s t ng hoá nh ng n.”**  
(Luger and Stubblefield, 1993)

**“Vi c nghiên c u c s trí tu thông qua s**

**Hình 1.1 Những định nghĩa về AI được chia thành 4 nhóm:**

Hệ thống tư duy như con người	Hệ thống tư duy có lập luận
Hệ thống hoạt động như con người	Hệ thống hoạt động có lập luận

**Hoạt động nh** **ur con người:**  
**phương pháp trắc nghiệm Turing**

Phương pháp trắc nghiệm Turing được Alan Turing (1950) đưa ra . Đây là phương pháp nhằm định nghĩa một hoạt động gọi là thông minh. Turing cho rằng: hoạt động trí tuệ là khả năng có được như con người trong những công việc cần tri thức, đủ để đánh lừa người thẩm vấn mình. Nói khái quát, phương pháp trắc nghiệm của ông là: máy tính sẽ bị một người hỏi thông qua giao tiếp gõ chữ qua vô tuyến. Kết thúc thí nghiệm sẽ là lúc người hỏi không còn câu nào để hỏi hoặc cả người và máy đều hoàn thành. Để lập chương trình cho máy tính qua được quá trình kiểm tra cần hoàn thành nhiều việc. Máy tính cần có các khả năng sau:

- Xử lý ngôn ngữ tự nhiên để giao tiếp tốt bằng tiếng Anh (hoặc ngôn ngữ khác)

- Biểu diễn tri thức, lưu trữ thông tin được cung cấp trước hoặc trong quá trình thẩm vấn.
- Tự động lập luận để sử dụng thông tin đã được lưu nhằm trả lời câu hỏi và phác thảo kết luận mới.
- Máy học: dễ thích nghi với môi trường mới, kiểm tra và chấp nhận những mẫu mới.
- Đối với AI, không cần có sự cố gắng cao mới qua được quá trình kiểm tra của Turing. Khi các chương trình AI giao tiếp trực tiếp với con người thì việc hoạt động được giống như người là vấn đề thiết yếu. Quá trình trình diễn và lý giải những hệ thống như thế có thể hoặc không cần dựa vào con người.

### 1.3.2. Tri thức là gì?

Tri thức là sự hiểu biết bằng lý thuyết hay thực tế về một chủ đề hay lĩnh vực. Tri thức là tổng của những cái đang biết hiện nay; tri thức là sức mạnh. Những người có tri thức tốt là những chuyên gia (expert).

So với chương trình truyền thống (được cấu tạo từ hai “chất liệu” cơ bản là **dữ liệu** và **thuật toán**), chương trình trí tuệ nhân tạo được cấu tạo từ hai thành phần là **cơ sở tri thức** (*knowledge base*) và **động cơ suy diễn** (*inference engine*).

### 1.3.3. Cơ sở tri thức (Knowledge Base: KB)

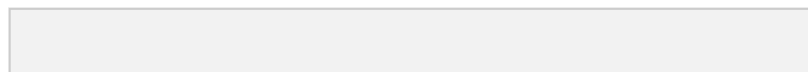
#### Định nghĩa:

Cơ sở tri thức là tập hợp các tri thức liên quan đến vấn đề mà chương trình quan tâm giải quyết. Cơ sở tri thức chứa các kiến thức được sử dụng để giải quyết các vấn đề (bài toán) trong trí tuệ nhân tạo.

### 1.3.4. Hệ cơ sở tri thức

Trong hệ cơ sở tri thức chứa hai chức năng tách biệt nhau, trường hợp đơn giản gồm hai khối: khối tri thức hay còn gọi là cơ sở tri thức; khối điều khiển hay còn gọi là động cơ suy diễn. Với các hệ thống phức tạp, bản thân động cơ suy diễn cũng có thể là một hệ cơ sở tri thức chứa các siêu tri thức (tri thức về các tri thức). Hình dưới đây mô tả cấu trúc chương trình truyền thống (bên trái) và cấu trúc chương trình trí tuệ nhân tạo (bên phải).

## DỮ LIỆU DỮ LIỆU CƠ SỞ TRI THỨC



THUẬT

TOÁN ĐỘNG CƠ SUY

DIỄN

**Động cơ suy diễn:** là phương pháp vận dụng tri thức trong cơ sở tri thức để giải quyết vấn đề.

## 1.4 CÁC LĨNH VỰC NGHIÊN CỨU VÀ ỨNG DỤNG CƠ BẢN

### 1.4.1 Lý thuyết giải bài toán và suy diễn thông minh

Lý thuyết giải bài toán cho phép viết các chương trình giải câu đố, chơi các trò chơi thông qua các suy luận mang tính người. Hệ thống giải bài toán GPS do Newel, Shaw và Simon đưa ra rồi được hoàn thiện năm 1969 là một mốc đáng ghi nhớ. Trước năm 1980, Buchanal và Luckham cũng hoàn thành hệ thống chứng minh định lý. Ngoài ra các hệ thống hỏi đáp thông minh như SI, QA2, QA3,... cho phép lưu trữ và xử lý khối lượng lớn các thông tin. Chương trình của McCarthy về các phương án hành động có khả năng cho các lời khuyên.

### 1.4.2 Lý thuyết tìm kiếm may rủi

Việc tìm kiếm lời giải cũng là việc bài toán. Lý thuyết tìm kiếm nhờ may rủi gồm các phương pháp và kỹ thuật tìm kiếm với sự hỗ trợ của thông tin phụ để giải bài toán một cách hiệu quả. Công trình đáng kể về lý thuyết này là của G.Pearl vào năm 1984.

### 1.4.3 Các ngôn ngữ về Trí Tuệ Nhân Tạo

Để xử lý các tri thức người ta không thể chỉ sử dụng các ngôn ngữ lập trình dùng cho các xử lý dữ liệu số mà cần có các ngôn ngữ khác. Các ngôn ngữ chuyên dụng này cho phép lưu trữ và xử lý các thông tin kí hiệu. Dùng các ngôn ngữ này cũng là cách để trả lời câu hỏi “thế nào” (what). rồi tới câu hỏi “làm sao vậy”(how). Một số ngôn ngữ được nhiều người biết đến là: • Các ngôn ngữ IPL.V, LISP.

• Ngôn ngữ mạnh hơn như PLANNER, PROLOG. Ngay trong một ngôn ngữ cũng

có nhiều thể hệ với những phát triển đáng kể.

### 1.4.4 Lý thuyết thể hiện tri thức và hệ chuyên gia

Theo quan điểm của nhiều chuyên gia công nghệ thông tin, trí tuệ nhân tạo là khoa học về thể hiện tri thức và sử dụng tri thức. Người ta nhận xét về phương pháp thể hiện tri thức như sau:

- Lược đồ dùng thể hiện tri thức trong chương trình
  - Mạng ngữ nghĩa, logic vị từ , khung, mạng là các phương pháp thể hiện tri thức một cách thông dụng.
  - Dùng khung để thể hiện tri thức chắc chắn là phương pháp có nhiều hứa hẹn trong các năm gần đây.

Việc gắn liền cách thể hiện và sử dụng tri thức là cơ sở hình thành hệ chuyên gia. Vậy nên

phải kết hợp các quá trình nghiên cứu các quy luật, thiết kế và xây dựng hệ chuyên gia. Tuy nhiên

cho đến nay, đa số các hệ chuyên gia mới thuộc lĩnh vực y học.

### 1.4.5 Lý thuyết nhận dạng và xử lý tiếng nói

Giai đoạn phát triển đầu của trí tuệ nhân tạo gắn liền với lý thuyết nhận dạng. Các phương pháp nhận dạng chính được giới thiệu gồm:

- Nhận dạng dùng tâm lý học
- Nhận dạng hình học
- Nhận dạng theo phương pháp hàm thế.
- Dùng máy nhận dạng

9

CuuDuongThanCong.com <https://fb.com/tailieudientucntt>

Ứng dụng của phương pháp này trong việc nhận dạng trong chữ viết, âm thanh, hình ảnh... cho đến nay đã trở nên quen thuộc. Người ta đã có hệ thống xử lý hình ảnh ba chiều, hệ thống tổng hợp tiếng nói.

Do khối lượng đồ sộ của tri thức về lý thuyết nhận dạng, các chương trình sau chưa đề cập đến các phương pháp nhận dạng được.

#### 1.4.6 Người máy

Cuối những năm 70, người máy trong công nghiệp đã đạt được nhiều tiến bộ “ Khoa học người máy là nối kết thông minh của nhận thức với hành động”. Người máy có bộ cảm nhận và các cơ chế hoạt động được nối ghép theo sự điều khiển thông minh. Khoa học về cơ học và trí tuệ nhân tạo được tích hợp trong khoa học về người máy. Các đề án trí tuệ nhân tạo nghiên cứu về người máy bắt đầu từ đề án “mắt – tay”. Trong thực tế, người máy được dùng trong các nhiệm vụ chuyên sâu, thuộc các dây truyền công nghiệp.

Nội dung về khoa học người máy sẽ được trình bày trong tài liệu riêng, không thuộc các chương của tài liệu này.

#### 1.4.7 Tâm lý học xử lý thông tin

Các kết quả nghiên cứu của tâm lý học giúp trí tuệ nhân tạo xây dựng các cơ chế trả lời theo hành vi, có ý thức. Nó giúp thực hiện các suy diễn mang tính người.

Hệ thống chuyên gia thương mại đầu tiên, R1, bắt đầu hoạt động tại công ty thiết bị kỹ thuật số (McDemott, 1982). Chương trình giúp sắp xếp cấu hình cho các hệ thống máy tính mới và trước năm 1986, nó đã tiết kiệm cho công ty khoảng 40 triệu dollar mỗi năm. Đến trước năm 1988, nhóm nghiên cứu AI của DEC đã có 40 hệ thống chuyên gia được triển khai. Du pont có 100 chiếc đi vào sử dụng và 500 chiếc được phát triển, tiết kiệm được khoảng 10 triệu dollar mỗi năm. Dường như mỗi một công ty chính của Mỹ đều có một nhóm AI của riêng công ty và cùng sử dụng hoặc đầu tư vào công nghệ hệ chuyên gia.

Năm 1981, Nhật bản thông báo về dự án “Thế hệ thứ năm”, kế hoạch 10 năm xây dựng những chiếc máy tính thông minh chạy Prolog giống như những chiếc máy chạy chương trình mã máy. Ý tưởng đó với khả năng thực hiện hàng triệu phép tính mỗi giây, máy tính có thể thuận lợi trong việc lưu trữ hàng loạt luật có sẵn. Dự án được đưa ra nhằm máy tính có thể giao tiếp bằng ngôn ngữ tự nhiên cùng một số các tham vọng khác.

Dự án “thế hệ 5” thúc đẩy niềm đam mê vào AI, bằng cách tận dụng nỗi lo lắng của người Nhật, các nhà nghiên cứu, các tổng công ty có thể hỗ trợ chung cho việc đầu tư tương tự như ở nước Mỹ.

Tổng công ty công nghệ máy tính và siêu điện

tử (MMC) đã được hình thành như một công ty liên kết nghiên cứu dự án của Nhật. Ở Anh, bài báo Alvey làm phục hồi số vốn đã bị bài báo Lighthill làm hụt. Trong cả hai trường hợp, thì AI đều là một phần trong nỗ lực lớn bao gồm cả thiết kế con chip và nghiên cứu giao diện với con người.

Bùng nổ công nghiệp AI cũng bao gồm cả các công ty như tập đoàn Carnegie, Inference, Intellicorp, và Teknowledge các công ty này yêu cầu các công cụ phần mềm để xây dựng hệ chuyên gia và các công ty phần cứng như Lisp Machine, công ty thiết bị Texas, Symbolic và Xerox đã xây dựng hệ thống làm việc tối ưu để phát triển các chương trình Lisp. Trên 100 công ty lắp ráp hệ thống công nghiệp robot. Nói chung ngành công nghiệp đi từ mức chỉ bán được vài triệu trong năm 1980 lên 2 tỉ dollar năm 1988.

Mặc dù khoa học máy tính bỏ quên lĩnh vực mạng nơ ron sau khi cuốn sách “khả năng nhận thức” của Minsky và Papert ra đời, nhưng các lĩnh vực khác vẫn tiếp tục, đặc biệt là vật lý. Một số

10

CuuDuongThanCong.com <https://fb.com/tailieudientucntt>

lượng lớn các nơ ron đơn giản đã có thể coi như một số nguyên tử trong chất rắn. Các nhà vật lý học như Hopfield (1982) đã sử dụng các kỹ thuật cơ học thống kê dẫn tới các ý tưởng thụ thai chéo quan trọng. Các nhà triết học David Rumelhart và Geoff Hinton nghiên cứu các mẫu mạng nơ ron trí nhớ. Vào những năm 1980, có ít nhất bốn nhóm khác nhau nghiên cứu lại thuật toán Back-propagation. Thuật toán này được công bố lần đầu vào năm 1969 bởi Bryson và Ho. Thuật toán được áp dụng rất nhiều trong khoa học máy tính và tâm lý học, và phổ biến kết quả trong cuốn “xử lý phân tán song song” (Rumelhart và McClelland, 1986).

Những năm gần đây, chúng ta đã chứng kiến sự thay đổi rất lớn trong nội dung và phương pháp nghiên cứu AI. Nó trở nên phổ biến khi dựa trên các lý thuyết có sẵn. Trong những năm 1970, một số lớn các kiến trúc và các phương pháp đã buộc phải thử. Rất nhiều trong số này, thậm chí là *ad hoc* và fragile và được tượng trưng ở một vài ví dụ được chọn là đặc biệt. Trong những năm gần đây, các phương pháp dựa trên mô hình Markov ẩn (HMMs) đã thống trị lĩnh vực này, hai khía cạnh của HMMs có liên quan đến những vấn đề bàn luận hiện tại. Đầu tiên, chúng được dựa trên lý thuyết toán học chính xác. Điều này cho phép các nhà nghiên cứu tiếng nói xây dựng các kết quả toán học của một vài thập kỷ đã được phát triển ở một số lĩnh vực khác. Thứ hai, chúng đã được sinh ra bởi một quá trình xử lý trên tập dữ liệu tiếng nói. Chắc chắn rằng thực hiện đó là dạng thô, và trong các thử nghiệm HMMs khắt khe không rõ ràng đã tiến triển đều đặn.

Lĩnh vực khác xem ra có lợi ích từ sự chính thức hoá là lập kế hoạch. Công việc sớm được thực hiện bởi Austin Tate (1977), sau đó là David Chapman (1987) đã có kết quả trong sự tổng hợp của các chương trình lập kế hoạch vào một khung làm việc đơn giản. Đã có một vài lời

khuyến rằng nên xây dựng trên mỗi cái khác nhau hơn là bắt đầu từ con số không tại mỗi thời điểm.

Kết quả của các hệ thống lập kế hoạch đó chỉ thực sự tốt trong các thế giới thu hẹp, trong năm 1970 nhiệm vụ lập lịch cho công việc của nhà máy. Xem Chương 11 và 12 để biết thêm chi tiết.

Cuốn *Tranh luận theo xác suất* trong các hệ thống thông minh đã đánh dấu một sự tán thưởng của lý thuyết quyết định và xác suất trong AI, tiếp theo sự hồi sinh của một sự thu nhỏ lí thú theo bài báo “Trong biện hộ của xác suất” của Peter Cheeseman (1985). Tin tưởng rằng hình thức mạng là phát minh đã cho phép tranh luận hiệu quả về chứng minh của sự phối hợp không chắc chắn. Cách tiếp cận lớn này vượt qua được vấn đề các hệ thống lập luận có khả năng trong những năm 1960 đến 1970... Chương 14 tới 16 bàn tới lĩnh vực này.

Cũng tương tự như cuộc cách mạng trong lĩnh vực người máy, khả năng của máy tính, máy

học (bao gồm cả các mạng neural) và sự trình diễn tri thức. Một cách hiểu tốt nhất của các vấn đề và sự phức tạp các thuộc tính, phối hợp cùng với sự ngẫu biến đã gia tăng trong toán học, đã có sự

chỉ đạo về lịch nghiên cứu công việc có khả năng và các phương

pháp dạng thô. Có lẽ được khuyến khích bởi sự tiến triển trong giải quyết các vấn đề con của AI, các nhà nghiên cứu đã bắt đầu tìm kiếm “yếu tố đầy đủ” cho vấn đề. Công việc của Allen Newel, John Laird và Paul Rosenbloom ở SOAR (Newel, 1990; Laird 1987) là những ví dụ hiểu biết tốt nhất của một yếu tố hoàn chỉnh về kiến trúc trong AI. Cũng gọi là hành động có mục đích “trong những hoàn cảnh xác định” của các yếu tố đưa vào trong các môi trường thực tế với các đầu vào cảm biến liên tục. Nhiều kết quả lý thú đã được tìm thấy trong công việc; bao gồm sự thực rằng trước đó các lĩnh vực con riêng biệt của AI cần tái tạo lại cái gì đó khi mà các kết quả của họ là cùng chỗ trong thiết kế một yếu tố riêng rẽ.

## 1.5 NHỮNG VẤN ĐỀ CHƯA ĐƯỢC GIẢI QUYẾT TRONG TRÍ TUỆ NHÂN TẠO

Kiện tướng cờ vua quốc tế Amold Denker, nghiên cứu các quân cờ trên bàn cờ trước mặt ông ta. Ông ta không hy vọng là hiện thực: ông phải từ bỏ cuộc chơi. Đối thủ của ông, HITECH, đã trở thành chương trình máy tính đầu tiên đánh thắng một kiện tướng cờ trong một ván chơi (Berliner, 1989).

“Tôi muốn đi từ Boston tới San Francisco” một người du lịch nói trong micro. “Bạn sẽ đi vào thời gian nào?” là câu hỏi lại. Người du lịch giải thích rằng cô ấy muốn đi vào ngày 20 tháng 10, trên chuyến rẻ nhất có thể, và trở về vào ngày Chủ nhật. Một chương trình giao tiếp bằng tay có thể hiểu được hành động nói của người là PEGASUS, đó là kết quả khiêm tốn dùng để đặt chỗ chuyến đi du lịch với giá 894 dollar bằng xe khách đường dài. Mặc dù vậy chương trình nhận biết tiếng nói có quá 10 từ bị sai, nó có thể là sự tổng hợp từ các lỗi nhỏ bởi vì sự hiểu của nó từ các hội thoại là đưa vào cùng một lúc (Zue 1994).

Một phân tích từ nơi điều khiển các nhiệm vụ của phòng thí nghiệm Jet Propulsion bắt đầu xu hướng thanh toán. Một thông báo màu đỏ xuất hiện trên màn hình chỉ ra rằng “sự không bình thường” với người du hành trên tàu không gian, đó là một nơi nào đó trong vùng xung quanh sao Hải vương. May thay, vấn đề phân tích có thể đúng từ mặt đất. Những người điều khiển tin tưởng rằng có vấn đề phải được bỏ qua đó là MARVEL, một hệ chuyên gia thời gian thực có các màn hình, dòng dữ liệu thô được chuyển từ tàu không gian, các công việc điều khiển chương trình và phân tích cảnh báo đối với những vấn đề nghiêm trọng

### TỔNG KẾT

Chương này đưa ra các định nghĩa về AI và thiết lập lại các cơ sở của nó, đó là sự phát triển. Một số các điểm quan trọng đáng lưu ý như sau:

Người ta nghĩ về AI có khác nhau. Có hai câu hỏi quan trọng đó là: bạn có quan tâm đến suy nghĩ hoặc hành vi? và Bạn có muốn hình mẫu con người hoặc từ một ý tưởng chuẩn mực? Các nhà triết học (quay trở lại năm 400 tr.CN) đưa ra ý kiến cho rằng não bộ cũng như một chiếc máy, rằng nó được điều khiển bằng tri thức đã được mã hoá, và ý nghĩ có thể mang theo thói quen giúp đỡ những hành động đúng đắn.

Một số nhà toán học đã cung cấp những công cụ các lệnh tính toán logic chắc chắn cũng tốt như là không chắc chắn, các lệnh không chính xác. Họ cũng đặt cơ sở làm việc cho các thuật toán. Ngành tâm lý học cũng cố thêm ý tưởng rằng loài người và động vật có thể đưa ra cách xử lý thông tin máy móc.

Ngành ngôn ngữ học trình bày rằng ngôn ngữ đủ để dùng trong mô hình này.

Ngành công nghiệp máy tính cung cấp các ứng dụng của AI. Các chương trình AI có xu hướng khá lớn, và họ không làm việc được nếu máy tính không có đủ tốc độ và bộ nhớ cần thiết. Lịch sử của AI có các chu kỳ thành công, tối ưu hoá đặt không đúng chỗ, và kết quả dẫn đến giảm lòng nhiệt tình và chi phí. Cũng đã có những bước lặp chỉ ra được các cách tiếp cận mới và trau dồi có hệ thống trong số các cách đó.

Những tiến triển gần đây trong học thuyết căn bản về sự thông minh đã tiến bộ cùng với khả năng của các hệ thống thực tế.

### **Những điểm chú ý về tiểu sử và lịch sử**

12

CuuDuongThanCong.com <https://fb.com/tailieudientucntt>

Cuốn sách Artificial Intelligence của Daniel Crevier (1993) đưa ra lịch sử khá hoàn chỉnh của lĩnh vực này, và Age of Intelligent Machines của Raymond Kurzweil (1990) về AI trong ngữ cảnh của khoa học máy tính và lịch sử trí tuệ. Các văn bản của Dianne Martin (1993) cũng công nhận rằng từ rất sớm các máy tính là có khả năng bởi sức mạnh thần kỳ của trí tuệ.

Phương pháp luận trạng thái của AI được Herb Simon bàn tới trong The Sciences of the Artificial (1981), đó là các lĩnh vực nghiên cứu được quan tâm cùng với các đồ tạo tác phức tạp. Nó cũng lý giải tại sao AI có thể có tầm nhìn trên cả hai lĩnh vực toán học và khoa học.

Artificial Intelligence: The very Idea bởi John Haugeland (1985) đã đưa ra sự tường thuật có thể đọc được của triết học và các vấn đề của AI. Khoa học nhận thức được mô tả tốt nhất bởi Johnson Laird, Máy tính và não bộ: giới thiệu về khoa học nhận thức. Baker (1989) gồm cả phân cú pháp của ngôn ngữ học hiện đại, cùng Chierchia và McConnel-Ginet (1990) bao gồm cả ngữ nghĩa, Allen (1995) bao gồm cả ngôn ngữ học từ quan điểm của AI.

Feigenbaum và Feldman đã làm việc với AI từ rất sớm, cuốn sách của họ có tựa đề Máy tính và suy nghĩ. Cuốn Xử lý thông tin ngữ nghĩa của Minsky và một loạt bài viết về Trí tuệ máy của Donald Michie. Một số lượng lớn các trang báo được tập hợp lại trong Sự hiểu biết trong AI (Webber và Nilsson, 1981).

Các cuộc hội thảo xuất hiện gần đây bàn về vấn đề chính của AI, đó là: thống nhất cứ hai năm một lần diễn ra hội thảo quốc tế AI, gọi tắt là IJCA (International Joint Conference AI), và hội thảo diễn ra hàng năm ở mức quốc gia về AI, và AAAI là tổ chức đứng ra bảo trợ cho AI. Các tạp chí chuyên ngành chung về AI là AI, Computation Intelligence, tổ chức IEEE Transactions on

Pattern Analysis and Machine Intelligence, và tạp chí điện tử Journal of Artificial Intelligence

Research. Các sản phẩm thương mại đưa ra trong các tạp chí như AI Expert và PC AI. Tổ chức xã hội chuyên nghiệp về AI là American Association for AI (AAAI), ACM Special Interest Group in AI (SIGART) và AISB (Society for AI and Simulation of Behaviour). Tạp chí về AI của AAAI và SIGART Bullentin có rất nhiều các đề tài và thầy hướng dẫn tốt như là thông báo của các cuộc hội



thảo và thảo luận.

Ở Việt Nam gần đây có tổ chức các Hội nghị Khoa học: Hệ mờ mạng nơ ron; Hội thảo Quốc gia về Hệ mờ do viện Toán học, Viện Công nghệ Thông tin thuộc viện Khoa học Công nghệ Quốc gia tổ chức hàng năm.

## BÀI TẬP VÀ CÂU HỎI

1. Chúng tôi đưa ra định nghĩa của AI theo hai khía cạnh, con người, ý tưởng và hành động. Nhưng có nhiều khía cạnh khác có giá trị đáng xét đến. Một trong số đó là sự chọn lựa giữa: sự

phần khích của chúng tôi về các kết quả lí thuyết hoặc các ứng dụng thực hành. Một khía cạnh nữa là chúng tôi có dự kiến có thể nhận ra các máy tính của chúng tôi có thông minh hay không. Đã có 8 định nghĩa tiêu biểu trong Hình 1.1 và có 7 định nghĩa theo bốn khía cạnh chúng tôi vừa đề cập và bạn có cảm thấy các định nghĩa là hữu ích sau đây

AI là ...

- a. “một tập hợp các thuật toán dễ tính toán, thích hợp tính gần đúng cho các bài toán đặc biệt khó.” (Partridge, 1991)
- b. “sự tham gia vào xây dựng một hệ thống kí hiệu vật lí sao cho có thể vượt qua trắc nghiệm của Turning.”
- c. “lĩnh vực khoa học máy tính nghiên cứu về các máy có thể hành động thông minh ra sao.” (Jackson, 1986)

13

CuuDuongThanCong.com <https://fb.com/tailieudientucntt>

d. “một lĩnh vực nghiên cứu đó là xoay quanh các kĩ thuật tính toán, cho phép thực hiện các công việc đòi hỏi thông minh thực sự khi có con người tham dự.” (Tanimoto, 1990) e. “một sự đầu tư rất lớn của trí tuệ của tự nhiên và các nguyên lí, các máy móc yêu cầu sự am hiểu hoặc tái tạo nó.” (Sharples, 1989)

f. “Lợi ích của máy tính là làm được mọi thứ, xem nó như là thông minh.” (Rowe, 1988) 2.

Nghiên cứu tài liệu AI để tìm ra các công việc nào dưới đây có thể giải quyết được bằng máy tính:

- a. Trò chơi bóng bàn
- b. Lái xe ở trung tâm Cairo
- c. Khám phá và chứng minh các lý thuyết toán học mới.
- d. Viết một truyện cười
- e. Đưa ra một lời khuyên khá hợp lý trong phạm vi liên quan đến luật pháp.
- f. Dịch tiếng Anh sang tiếng Việt theo thời gian thực.

3. Thực tế, hư cấu và dự đoán:

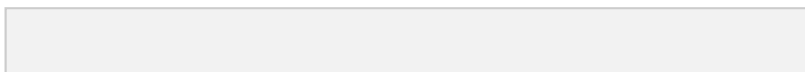
- a. Tìm một bản công bố của một nhà triết học hoặc nhà khoa học có uy tín cho rằng hiệu quả của độ chắc chắn sẽ không bao giờ được trình diễn bởi máy tính, rằng ở đó bây giờ đã được trình diễn.
- b. Tìm một công bố của một nhà khoa học về máy tính có uy tín cho rằng hiệu quả của độ đo chắc chắn được trình diễn bởi thời điểm từ khi nó hợp qui cách.

c. So sánh độ chính xác của những dự đoán trong các lĩnh



vực khác nhau, chẳng hạn như y sinh, công nghệ nano, hoặc điện gia dụng.

4. Cho rằng “những chiếc máy tính là không thông minh - chúng chỉ có thể làm được những gì mà lập trình viên bảo chúng” là câu lệnh phản trước thì đúng và hàm ý sau đó không đúng?



## CHƯƠNG 2: CÁC PHƯƠNG PHÁP GIẢI QUYẾT VẤN ĐỀ

### 2.1. GIẢI QUYẾT VẤN ĐỀ KHOA HỌC VÀ TRÍ TUỆ NHÂN TẠO

Trong phần này, chúng ta chỉ ra một agent có thể hành động như thế nào bằng cách đặt ra mục tiêu và xem xét chuỗi các hành động mà có thể đạt được những mục tiêu này. Một mục tiêu và tập các phương tiện để đạt được mục tiêu được gọi là *vấn đề*. Quá trình khám phá các phương tiện có thể làm được gì gọi là *tìm kiếm*. Chúng ta cho thấy tìm kiếm có thể thực hiện và những giới hạn của nó..

#### Giải quyết bài toán bằng cách tìm kiếm

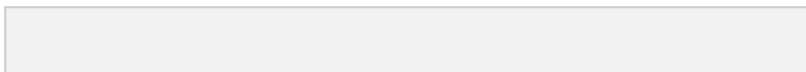
Chúng ta xem một agent quyết định phải làm gì? như thế nào? bằng cách xem xét có hệ thống kết quả các chuỗi hành động mà nó thực hiện.

Trong chương này, chúng ta mô tả một agent dựa trên mục đích gọi là **agent giải quyết bài toán**. Các agent giải quyết vấn đề sẽ quyết định phải làm gì bằng cách tìm kiếm chuỗi các hành động dẫn đến trạng thái mong muốn. Chương này phân tích các thuật toán.

### 2.2. GIẢI QUYẾT VẤN ĐỀ CỦA CON NGƯỜI

Hãy tưởng tượng các agent của chúng ta ở trong thành phố Arad, Rumani đang thực hiện một

chuyến du lịch. Agent đã có vé để bay đến Bucarét vào ngày hôm



sau. Vé máy bay không thể trả lại được, visa của agent chuẩn bị hết hạn, và kể từ ngày mai sẽ không còn chỗ trong 6 tuần tới. Phạm vi thực hiện của agent chứa nhiều yếu tố khác ngoài chi phí tiền vé máy bay và có một điều không mong muốn là có thể bị trục xuất. Chẳng hạn, agent muốn cải thiện nước da rám nắng của mình, học thêm tiếng Rumani, đi chơi đâu đó vv... Tất cả những yếu tố này có thể gợi ra vô số các hành động.

Agent đưa ra mục tiêu: lái xe tới Bucarét, và xem những thành phố nào cần phải đi, xuất phát từ Arad. Có ba con đường ra khỏi Arad, một đường đến Sibiu, một đường đến Timisoara và một đến Zerind. Tất cả các con đường này đều không đến Bucaret, vì vậy trừ khi agent nắm rõ bản đồ Rumani, agent sẽ không biết phải đi con đường nào tiếp theo. Nói cách khác, agent không biết hành động nào là tốt nhất trong các hành động. Nếu agent không có các kiến thức trợ giúp, nó sẽ bị tắc (không tìm ra được đường đi tiếp theo). Cách tốt nhất nó có thể làm là chọn ngẫu nhiên một trong các hành động.

Giả thiết agent có một bản đồ Rumani, hoặc trên giấy hoặc trong trí nhớ. Mục đích của bản đồ là

cung cấp cho agent các thông tin về các trạng thái mà nó có thể đến

và những hành động mà nó có thể thực hiện. Agent có thể sử dụng thông tin này để xem xét các đoạn của hành trình mang tính giả thiết là: khi nó tìm ra một con đường trên bản đồ từ Arad tới Bucaret, nó có thể đạt mục tiêu bằng cách thực hiện các hành động tương ứng với các chặng của hành trình. Sau đó agent lựa chọn các giá trị chưa biết để quyết định phải làm gì bằng cách kiểm tra chuỗi các hành động khác nhau dẫn đến các trạng thái đã biết; sau đó chọn hành động tốt nhất. Quá trình tìm kiếm một chuỗi các hành động như vậy được gọi là tìm kiếm. Giải thuật tìm kiếm coi một vấn đề như dữ liệu vào và đáp số là một giải pháp dưới dạng chuỗi hành động. Khi một giải pháp được tìm thấy, các hành động mà nó đề xuất có thể được tiến hành. Điều này được gọi là giai đoạn thực hiện

Trong phần này, chúng ta sẽ tìm hiểu quá trình xác định bài toán chi tiết hơn. Trước tiên, ta xem xét khối lượng kiến thức mà agent có thể có sử dụng để hướng đến các hành động của nó và trạng thái mà nó phải đi qua. Điều này phụ thuộc vào sự nhận thức của agent với môi trường của nó như thế nào thông qua kết quả giác quan và hành động của nó. Chúng ta biết có bốn loại bài toán khác nhau: bài toán một trạng thái đơn giản; bài toán đa trạng thái; bài toán ngẫu nhiên và bài toán thăm dò

## 2.3. PHÂN LOẠI VẤN ĐỀ. CÁC ĐẶC TRƯNG CƠ BẢN CỦA VẤN ĐỀ

### Những vấn đề (bài toán) xác định rõ ràng và các giải pháp

Một vấn đề là một tập hợp các thông tin mà agent sử dụng để quyết định phải làm gì. Chúng ta bắt đầu bằng cách phân loại các thông tin cần thiết dùng cho định nghĩa bài toán đơn trạng thái. Các yếu tố cơ bản của việc định nghĩa một bài toán là các trạng thái và các hành động. Để xác định được chúng một cách chính xác, chúng ta cần các yếu tố sau:

**Trạng thái ban đầu của agent.** Tập các hành động có thể đối với agent. Thuật ngữ thao tác (operation) được sử dụng để mô tả một hành động trong ngữ cảnh là trạng thái nào nó sẽ đến nếu thực hiện hành động trong từ một trạng thái đặc biệt. (Một công thức sử dụng hàm  $S$ . Cho trước trạng thái  $x$ ,  $S(x)$  cho trạng thái có thể đi tới từ  $x$  bằng bất cứ một hành động đơn).

**Định nghĩa: không gian trạng thái của vấn đề:** là tập các trạng thái có thể đạt được bằng

chuỗi hành động bất kỳ xuất phát từ trạng thái ban đầu. Một hành trình trong không gian trạng thái là tập các hành động tùy ý xuất phát từ trạng thái này đến trạng thái khác. Yếu tố tiếp

theo của vấn đề như sau: tiêu chuẩn kiểm tra trạng thái hiện thời là trạng thái đích (mục tiêu)? Việc kiểm tra đơn giản chỉ là để nhằm kiểm tra xem chúng ta đã đi tới một trong các trạng thái mục tiêu hay chưa. Thành thạo mục tiêu được xác định bởi một thuộc tính trừu tượng thay vì một tập đếm được các trạng thái. Chẳng hạn, trong môn đánh cờ, mục tiêu là đi tới một trạng thái gọi là “chiếu tướng”, khi tướng của đối phương sẽ bị ăn bất kể đối phương đi như thế nào ở bước kế tiếp.

Cuối cùng, chọn giải pháp thích hợp nhất, dù có nhiều giải pháp tới đích. Ví dụ, chúng ta có thể thích những hành trình có ít hành động hoặc các hành động có chi phí thấp. Hàm chi phí đường đi là hàm được gán chi phí cho một đường đi. Trong tất cả các trường hợp chi phí của đường đi là tổng các chi phí của các hành động đơn dọc theo đường đi. Hàm chi phí đường đi thường được ký hiệu là hàm  $g$ . Trạng thái ban đầu, tập toán tử, thủ tục kiểm tra mục tiêu và hàm chi phí đường đi xác định một vấn đề. Về mặt tự nhiên, chúng ta có thể xác định một kiểu dữ liệu để biểu diễn các vấn đề:

Kiểu dữ liệu **Bi toán**

Các thành phần: **Trạng thái ban đầu, các toán tử, kiểm tra mục tiêu, hàm chi phí**

## Giải quyết vấn đề

Hiệu quả của tìm kiếm có thể đo được theo ít nhất ba chỉ tiêu Thứ nhất, có tìm thấy một giải pháp nào không? Thứ hai, đó có phải là một giải pháp tốt không (giải pháp có chi phí đường đi

thấp)? Thứ ba, chi phí tìm kiếm với thời gian tìm kiếm và bộ nhớ yêu cầu để tìm một giải pháp là bao nhiêu? Chi phí toàn bộ của việc tìm kiếm là tổng chi phí đường đi và chi phí tìm kiếm ( $S$ ). Đối với vấn đề tìm đường đi từ Arad đến Bucarét, chi phí đường đi tỷ lệ thuận với tổng độ dài của đường, cộng thêm chi phí do sự cố dọc đường. Chi phí tìm kiếm phụ thuộc vào các tình huống. Trong môi trường tĩnh, nó bằng không vì phạm vi thực hiện là độc lập với thời gian. Nếu phải cập nhật đến Bucarét, môi trường là bán động bởi vì việc cân nhắc lâu hơn sẽ làm chi phí nhiều hơn. Trong trường hợp này, chi phí tìm kiếm có thể biến thiên xấp xỉ tuyến tính với thời gian tính toán (ít nhất với một khoảng thời gian nhỏ). Do đó, để tính toán tổng chi phí, chúng ta cần phải bổ sung thêm các giá trị là dặm và mili giây. Điều này không phải luôn dễ dàng bởi vì không có một “tỷ lệ trao đổi chính thức” giữa hai đại lượng này. Agent bằng cách nào đó phải quyết định những tài nguyên nào sẽ dành cho việc tìm kiếm và những tài nguyên nào dành cho việc thực hiện. Đối với những vấn đề có không gian trạng thái rất nhỏ, dễ dàng tìm ra giải pháp với chi phí đường đi thấp nhất. Nhưng đối với những vấn đề phức tạp, lớn, cần phải thực hiện một sự thỏa hiệp- agent có thể tìm kiếm trong một thời gian dài để tìm ra giải pháp tối ưu hoặc agent có thể tìm kiếm trong một thời gian ngắn hơn và nhận được một giải pháp với chi phí đường đi cao hơn một chút.

## Chọn lựa trạng thái và hành động

Bây giờ chúng ta có các định nghĩa mới, chúng ta hãy bắt đầu sự điều tra các vấn đề của chúng ta với một vấn đề khá dễ như sau: “Lái xe từ Arad đến Bucarét sử dụng các con đường trên bản đồ” Một không gian trạng thái có xấp xỉ 20 trạng thái, mỗi trạng thái được xác định bởi vị trí, được ghi rõ là một thành phố. Như vậy, trạng thái ban đầu là “ở Arad” và kiểm tra mục tiêu là “đây có phải là Bucarét không?”. Các toán hạng tương ứng với việc lái dọc theo các con đường giữa các thành phố.

### Các bài toán ví dụ

Phạm vi của các môi trường nhiệm vụ mà có thể được mô tả đặc điểm bởi các vấn đề được định nghĩa rõ ràng là rất rộng lớn. Chúng ta có thể phân biệt giữa cái gọi là **các bài toán trò chơi**, mà nhằm để minh họa hay thực hành rất nhiều các phương pháp giải quyết vấn đề, và cái gọi là **các bài toán thuộc thể giới thực** mà sẽ là các vấn đề khó khăn hơn và mọi người thực sự quan tâm đến các giải pháp để giải quyết các vấn đề này. Trong phần này, chúng ta sẽ đưa ra ví dụ cho cả hai loại vấn đề đó. Các vấn đề đồ chơi tất nhiên có thể mô tả một cách chính xác, ngắn gọn. Điều đó có nghĩa là các nhà nghiên cứu khác nhau có thể dễ dàng sử dụng các vấn đề này để so sánh việc thực hiện của các giải thuật. Ngược lại, các vấn đề thể giới thực lại không thể có một sự miêu tả một cách đơn giản, nhưng chúng ta sẽ cố gắng đưa ra một cách chung nhất về sự mô tả chính xác các vấn đề này.

### Các bài toán Trò chơi

#### *Trò chơi 8 quân cờ (Cờ ta canh)*

Một ví dụ của trò chơi 8 quân cờ được chỉ ra trong hình 2.1, gồm một bảng kích thước 3 x 3 với 8 quân cờ được đánh số từ 1 đến 8 và một ô trống. Một quân cờ đứng cạnh ô trống có thể đi vào ô trống. Mục tiêu là tiến tới vị trí các quân cờ như ở trong hình bên phải. Một mẹo quan trọng cần chú ý là thay vì dùng các toán tử như “chuyển quân cờ số 4 vào ô trống”, sẽ tốt hơn nếu có các toán tử như “ô trống thay đổi vị trí với quân cờ chuyển sang bên trái của nó”, bởi vì loại toán tử thứ hai này sẽ ít hơn. Điều đó sẽ giúp chúng ta có các khái niệm như sau:

- Các trạng thái: một sự mô tả trạng thái chỉ rõ vị trí của mỗi quân cờ trong 8 quân cờ ở một trong 9 ô vuông. Để có hiệu quả, sẽ có ích nếu trạng thái bao gồm cả vị trí của ô trống.
- Các toán tử: ô trống di chuyển sang trái, phải, lên trên, đi xuống.

17

CuuDuongThanCong.com <https://fb.com/tailieudientucntt>

- Kiểm tra mục tiêu: trạng thái khớp với hình dạng chỉ ra ở hình 3.4
- Chi phí đường đi: mỗi bước đi chi phí là 1, vì vậy chi phí đường đi bằng độ dài của đường đi.

Trạng thái đầu Trạng thái đích


Hình 2.1 Một ví của trò chơi 8 quân cờ

Trò chơi 8 quân cờ thuộc về loại trò chơi trượt khối. Lớp trò chơi này được biết đến có mức độ hoàn thành NP, vì vậy chúng ta không mong muốn tìm được các phương pháp tốt hơn các thuật toán tìm kiếm đã được mô tả trong chương này và trong các chương tiếp theo. Trò chơi 8 quân cờ và sự mở rộng của nó, trò chơi 15 quân cờ là những vấn đề kiểm tra tiêu chuẩn đối với các giải thuật tìm kiếm trong AI.

### Bài toán

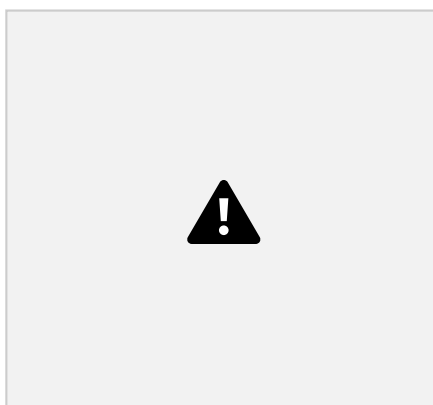
### 8 quân hậu

Mục tiêu của bài toán 8 quân hậu là đặt 8 con hậu trên một bàn cờ vua sao cho không con nào ăn con nào. (Một con hậu sẽ ăn bất cứ con nào nằm trên cùng hàng, cùng cột hoặc cùng đường chéo với nó). Hình 2.2 chỉ ra một giải pháp cố gắng để giải quyết bài toán nhưng không thành công: con hậu ở cột bên phải nhất bị con hậu ở trên cùng bên trái chiếu.

Mặc dầu các giải thuật đặc biệt hiệu quả tồn tại để giải quyết bài toán này và tập các bài toán tổng quát n con hậu, nó thực sự vẫn là vấn đề rất thú vị dùng để kiểm tra các giải thuật tìm kiếm. Có hai loại phương pháp chính. Phương pháp gia tăng bao gồm việc đặt các con hậu từng con một, trong khi phương pháp trạng thái hoàn thành lại bắt đầu với 8 con hậu trên bàn cờ và tiến hành di chuyển các con hậu. Trong cả hai phương pháp, người ta không quan tâm đến chi phí đường đi do chỉ tính đến trạng thái cuối cùng: các giải thuật do đó chỉ được so sánh về chi phí tìm kiếm. Như vậy, chúng ta có việc kiểm tra mục tiêu và chi phí đường đi như sau:

**Hình 2.2** Gần như giải pháp đối với 8 con hậu. (Giải pháp thực sự được dành cho bạn đọc tự làm như một bài tập.)

là một bài toán



### Hình 2.2 Một giải pháp đối với bài toán 8 con hậu

- ◊ Kiểm tra mục tiêu: 8 con hậu trên bàn cờ, không con nào ăn con nào
- ◊ Chi phí đường đi: bằng không
- ◊ Cũng có các trạng thái và các toán tử có thể khác

.Hãy xem xét sự công thức hoá

- ◊ Các trạng thái: bất cứ sự sắp xếp của từ 0 đến 8 con hậu trên bàn cờ
- ◊ Các toán tử: thêm một con hậu vào bất cứ ô nào

Trong cách công thức hoá này, chúng ta có 648 dãy có thể để thử. Một sự lựa chọn khôn khéo sẽ sử dụng thực tế là việc đặt một con hậu vào ô mà nó đã bị chiếu sẽ không thành công bởi vì việc

đặt tất cả các con hậu còn lại sẽ không giúp nó khỏi bị ăn(bị con hậu khác chiếu). Do vậy chúng ta có thể thử cách công thức hoá sau:

♦ Các trạng thái: là sự sắp xếp của 0 đến 8 con hậu mà không con nào ăn con nào ♦ Các toán tử: đặt một con hậu vào cột trống bên trái nhất mà nó không bị ăn bởi bất cứ con hậu nào khác.

Dễ thấy rằng các hành động đã đưa ra chỉ có thể tạo nên các trạng thái mà không có sự ăn lẫn nhau; nhưng đôi khi có thể không có hành động nào. Tính toán nhanh cho thấy chỉ có 2057 khả năng có thể để xếp thử các con hậu. Công thức hoá đúng đắn sẽ tạo nên một sự khác biệt rất lớn đối với kích thước của không gian tìm kiếm. Các sự xem xét tương tự cũng sẽ được áp dụng cho cách công thức hoá trạng thái đầy đủ. Chẳng hạn, chúng ta có thể thiết lập vấn đề như sau:

♦ Các trạng thái: là sự sắp xếp của 8 con hậu, mỗi con trên một cột.

♦ Các toán tử: di chuyển bất cứ con hậu nào bị chiếu tới một ô khác trên cùng cột. Cách công thức này sẽ cho phép các giải thuật cuối cùng tìm ra một giải pháp, nhưng sẽ là tốt hơn nếu di chuyển tới ô bị chiếu.

### **Các bài toán thế giới thực**

#### ***Tìm kiếm đường đi***

Chúng ta đã xem việc tìm kiếm đường đi được định nghĩa như thế nào bằng các thuật ngữ chỉ các vị trí xác định và các phép di chuyển dọc theo các đường nối giữa chúng. Các giải thuật tìm kiếm đường đi được sử dụng trong rất nhiều các ứng dụng, như tìm đường đi trong các mạng máy tính, trong các hệ thống tư vấn du lịch tự động, và trong các hệ thống lập kế hoạch cho các chuyến du lịch bằng máy bay. ứng dụng cuối cùng có lẽ phức tạp hơn, bởi vì du lịch bằng máy

bay có chi phí đường đi rất phức tạp, liên quan đến vấn đề tiền nong, chất lượng ghế ngồi, thời

gian trong ngày, loại máy bay, các giải thưởng cho các lộ trình bay thường xuyên, v.v Hơn nữa, các hành động trong bài toán không có kết quả được biết đầy đủ: các chuyến bay có thể đến chậm hay đăng ký trước quá nhiều, có thể bị mất liên lạc, sương mù hoặc sự bảo vệ khẩn cấp có thể gây ra sự trì hoãn.

#### ***Bài toán người bán hàng rong và các chuyến du lịch***

Hãy xét bài toán kinh điển sau: “Thăm tất cả các thành phố mỗi thành phố thăm ít nhất một lần, khởi hành và kết thúc ở Bucaret”. Điều này rất giống với tìm kiếm đường đi, bởi vì các toán tử vẫn tương ứng với các chuyến đi giữa các thành phố liền kề. Nhưng đối với bài toán này, không gian trạng thái phải ghi nhận nhiều thông tin hơn. Ngoài vị trí của agent, mỗi trạng thái phải lưu lại được các thành phố mà agent đã đi qua. Như vậy, trạng thái ban đầu sẽ là “ở Bucaret: đã thăm{Bucaret}.”, một trạng thái trung gian điển hình sẽ là “ở Vaslui: đã thăm {Bucaret, 19

CuuDuongThanCong.com <https://fb.com/tailieudientucntt>

Urziceni, Vaslui}.” và việc kiểm tra mục tiêu sẽ kiểm tra xem agent đã ở Bucaret chưa và tất cả 20 thành phố đã được viếng thăm xong toàn bộ chưa.

Bài toán người bán hàng rong (TSP) là một bài toán du lịch nổi tiếng trong đó mỗi thành phố phải được viếng thăm đúng chính xác một lần. Mục đích là tìm hành trình ngắn nhất. 6 Bài toán có độ phức tạp NP(Karp,1972), nhưng đã có một sự cố gắng rất lớn nhằm cải thiện khả năng của các thuật toán TSP. Ngoài các chuyến đi đã được lập kế hoạch cho người bán hàng rong, những thuật toán này đã được sử dụng cho các nhiệm vụ như lập kế hoạch cho sự dịch chuyển của các mũi khoan trên trên bảng mạch một cách tự động.

**Bài toán hành trình ngắn nhất - ứng dụng nguyên lý tham lam (Greedy)** Bài toán: Hãy tìm một hành trình cho người giao hàng đi qua  $n$  điểm khác nhau, mỗi điểm đi qua một lần và trở về điểm xuất phát sao cho tổng chiều dài đoạn đường cần đi là ngắn nhất. Giả sử rằng có con đường nối trực tiếp từ giữa hai điểm bất kỳ.

Tất nhiên là có thể giải bài toán này bằng cách liệt kê tất cả các con đường có thể đi, tính chiều dài của mỗi con đường đó rồi tìm con đường có chiều dài ngắn nhất. Tuy nhiên cách giải này có độ phức tạp  $O(n!)$  Do đó, khi số đại lý tăng thì số con đường phải xét sẽ tăng lên rất nhanh. Một cách đơn giản hơn nhiều cho kết quả tương đối tốt là ứng dụng thuật toán heuristic ứng dụng nguyên lý tham lam Greedy. Tư tưởng của thuật giải như sau:

- Điểm khởi đầu, ta liệt kê tất cả các quãng đường từ điểm xuất phát cho đến  $n$  đại lý rồi chọn đi theo con đường ngắn nhất.

- Khi đã đi đến một đại lý chọn đi đến đại lý kế tiếp cũng theo nguyên tắc trên.

Nghĩa là liệt kê tất cả các con đường từ đại lý ta đang đứng đến những đại lý chưa đến. Chọn con đường ngắn nhất. Lặp lại quá trình này cho đến lúc không còn đại lý nào để đi

### **Bài toán phân việc - ứng dụng của nguyên lý thứ tự**

Bài toán: Một công ty nhận được hợp đồng gia công  $m$  chi tiết máy  $J_1, J_2, \dots, J_m$ . Công ty có  $n$  máy gia công lần lượt là  $P_1, P_2, \dots, P_n$ . Mọi chi tiết đều có thể gia công trên bất kỳ máy gia công nào. Một khi đã gia công một chi tiết trên một máy, công việc sẽ tiếp tục cho đến lúc hoàn thành, không thể bị cắt ngang. Để gia công một việc  $J_i$  trên một máy bất kỳ ta cần dùng thời gian tương ứng là  $t_i$ . Nhiệm vụ của công ty là làm sao để gia công xong toàn bộ  $n$  chi tiết trong thời gian sớm nhất.

Chúng ta xét bài toán trong trường hợp có 3 máy  $P_1, P_2, P_3$  và sáu công việc với thời gian là  $t_1 = 2, t_2 = 5, t_3 = 8, t_4 = 1, t_5 = 5, t_6 = 1$ . Có một giải pháp được đặt ra là: Tại thời điểm  $t = 0$ , ta tiến hành gia công chi tiết  $J_2$  trên máy  $P_1$ ,  $J_5$  trên máy  $P_2$  và  $J_1$  tại  $P_3$ . Tại thời điểm  $t = 2$  công việc  $J_1$  được hoàn thành, trên máy  $P_3$  ta tiếp chi tiết  $J_4$ . Trong gia công  hai máy  $P_1$  và  $P_2$  vẫn đang thực hiện công việc đầu tiên của mình ... Theo vậy sau đó máy  $P_3$  sẽ tiếp tục hoàn thành nốt các công việc  $J_6$  và  $J_3$ . Thời gian hoàn thành công việc là 12. Ta thấy phương án này đã thực hiện công việc một cách không tốt. Các máy  $P_1$  và  $P_2$  có quá nhiều thời gian rảnh.

Thuật toán tìm phương án tối ưu  $L_0$  cho bài toán này theo kiểu vét cạn có độ phức tạp cỡ  $O(m^n)$  (với  $m$  là số máy và  $n$  là số công việc). Bây giờ ta xét đến một thuật giải Heuristic rất đơn giản (độ phức tạp  $O(n)$ ) để giải bài toán này:

- Sắp xếp các công việc theo thứ tự giảm dần về thời gian gia công
- Lần lượt sắp xếp các việc theo thứ tự đó vào máy còn dư nhiều thời gian nhất Với tư tưởng như vậy ta hoàn toàn có thể đưa ra một phương án tối ưu  $L^*$ , thời gian thực hiện công việc bằng 8, đúng bằng thời gian thực hiện công việc  $J_3$

Điều khiển robot là sự tổng quát hoá của bài toán tìm đường đi đã được miêu tả lúc trước. Thay vì một tập các lộ trình rời rạc, một robot có thể di chuyển trong một không gian liên tiếp với (về mặt nguyên lý) một bộ vô hạn các hành động và trạng thái có thể. Để đơn giản, robot tròn di



chuyển trên một mặt phẳng, không gian bản chất là hai chiều. Khi robot có cánh tay và chân mà phải được điều khiển, không gian tìm kiếm trở nên đa chiều. Cần có các kỹ thuật tiên tiến để biến không gian tìm kiếm trở nên hữu hạn. Ngoài sự phức tạp của bài toán còn ở chỗ các robot thật sự phải xử lý các lỗi trong việc đọc thông tin sensor và các bộ điều khiển động cơ.

### ***Sắp xếp dãy***

Sự lắp ráp tự động các đối tượng phức tạp được thực hiện bởi rôbốt lần đầu tiên đã được tiến hành bởi robot Freddy (Michie, 1972). Sự phát triển kể từ đó khá chậm chạp nhưng chắc chắn nó rất cần cho những nơi lắp ráp phức tạp như lắp ráp điện tử. Trong các bài toán lắp ráp, vấn đề là tìm một thứ tự để lắp ráp các phần của một sản phẩm. Nếu như lựa chọn sai một thứ tự, sẽ không thể lắp thêm một số các bộ phận của sản phẩm nếu như không phải dỡ bỏ một số bộ phận đã lắp ráp lúc trước đó. Kiểm tra một bước trong dãy để đảm bảo tính khả thi là một bài toán tìm kiếm hình học phức tạp rất gần với điều khiển robot. Như vậy, việc sinh ra những bước kế vị hợp lý là khâu đắt nhất trong dây truyền lắp ráp và việc sử dụng các giải thuật tốt để làm giảm việc tìm kiếm là điều cần thiết.

## **2.4 CÁC PHƯƠNG PHÁP BIỂU DIỄN VẤN ĐỀ**

### ***Tìm kiếm các giải pháp***

Chúng ta đã xem xét cách làm thế nào để xác định một vấn đề, và làm thế nào để công nhận một giải pháp. Phần còn lại – tìm kiếm một giải pháp – được thực hiện bởi một phép tìm kiếm trong không gian trạng thái. ý tưởng là để duy trì và mở rộng một tập các chuỗi giải pháp cục bộ. Trong phần này, chúng ta chỉ ra làm thế nào để sinh ra những chuỗi này và làm thế nào để kiểm soát được chúng bằng cách sử dụng các cấu trúc dữ liệu hợp lý.

### ***Khởi tạo các chuỗi hành động***

Ví dụ để giải quyết bài toán tìm đường đi từ Arad đến Bucaret, chúng ta bắt đầu với trạng thái đầu là Arad. Bước đầu tiên là kiểm tra xem nó có phải là trạng thái đích hay không. Rõ ràng là không, nhưng việc kiểm tra là rất quan trọng để chúng ta có thể giải quyết những việc bị chơi xỏ như “bắt đầu ở Arad, đi đến Arad”. Do nó không phải là trạng thái đích, chúng ta cần phải xem xét một số trạng thái khác. Điều này được thực hiện bằng cách áp dụng các toán tử cho trạng thái hiện thời, do đó xây dựng nên một tập các trạng thái mới. Quá trình này được gọi là sự mở

rộng trạng thái. Trong trường hợp này, chúng ta có ba trạng thái mới, “ở Sibiu”, “ở Timisoara” và “ở

Zerind” bởi vì có một đường đi một bước trực tiếp từ Arad đến ba thành phố này. Nếu như chỉ có duy nhất một khả năng, chúng ta sẽ chọn khả năng đó và tiếp tục đi tiếp. Nhưng bất cứ khi nào mà có nhiều khả năng lựa chọn, chúng ta phải quyết định sẽ chọn phương án nào để đi tiếp. Đây chính là vấn đề cốt yếu của việc tìm kiếm – lựa chọn một vị trí và để các lựa chọn còn lại cho việc lựa chọn sau này nếu như sự lựa chọn đầu tiên không đưa đến một giải pháp. Giả sử chúng ta chọn Zezind. Chúng ta kiểm tra xem nó đã phải là trạng thái đích chưa (nó chưa phải trạng thái đích), và sau đó mở rộng nó để có “ở Arad” và “ở Oradea”. Như thế chúng ta có thể chọn một trong hai trạng thái này, hoặc là quay lại và chọn Sibiu hay Timisoara. Chúng ta tiếp tục chọn, kiểm tra và mở rộng cho đến khi tìm được một đường đi, hoặc cho đến khi không còn trạng



thái nào nữa để mở rộng. Việc lựa chọn trạng thái nào để mở rộng trước tiên do chiến lược tìm kiếm quyết định.

## 2.5. CÁC PHƯƠNG PHÁP GIẢI QUYẾT VẤN ĐỀ CƠ BẢN

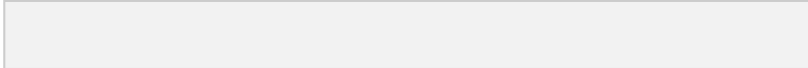
### *Các chiến lược tìm kiếm*

Công việc chủ yếu của việc tìm kiếm đã chuyển sang việc tìm kiếm một chiến lược tìm kiếm đúng đắn đối với một vấn đề. Trong sự nghiên cứu của chúng ta về lĩnh vực này, chúng ta sẽ đánh giá các chiến lược bằng các thuật ngữ của bốn tiêu chuẩn sau:

♦ **Tính hoàn thành:** chiến lược có bảo đảm tìm thấy một giải pháp khi có một vấn đề ♦ **Độ phức tạp thời gian:** chiến lược mất bao lâu để tìm ra một giải pháp? ♦ **Độ phức tạp không gian** (dung lượng bộ nhớ): chiến lược đó cần bao nhiêu dung lượng bộ nhớ cần thiết để thực hiện việc tìm kiếm.

♦ **Tính tối ưu:** chiến lược có tìm được giải pháp có chất lượng cao nhất khi có một số các giải pháp khác nhau?

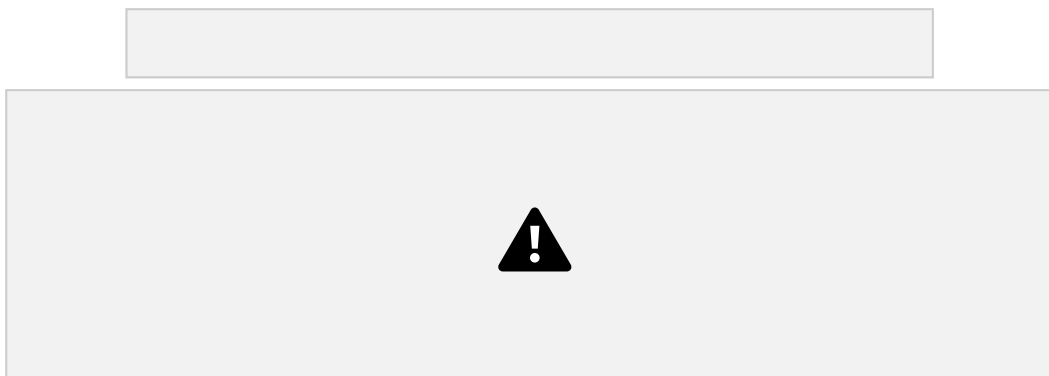
Phần này sẽ nói đến 6 chiến lược tìm kiếm mà được sử dụng dưới tiêu đề **tìm kiếm không đủ thông tin (uninformed search)**. Thuật ngữ này có nghĩa là không có các thông tin về số các bước hay chi phí đường đi từ trạng thái hiện tại cho tới đích – tất cả những gì chúng có thể làm là phân biệt một trạng thái đích với một trạng thái không phải là trạng thái đích. Tìm kiếm không có thông tin đầy đủ đôi khi còn được gọi là **tìm kiếm mù (blind search)**.

<b>Tìm</b>	<b>kiếm theo chiều rộng</b>
	Một chiến lược tìm kiếm đơn giản là tìm kiếm theo chiều rộng. Trong chiến lược này, nút gốc được mở rộng trước tiên, sau đó đến lượt tất cả các nút mà được sinh ra bởi nút gốc được mở rộng, và sau đó tiếp đến những nút kế tiếp của chúng và cứ như vậy. Nói một cách tổng quát, tất cả các nút ở độ sâu $d$ trên cây tìm kiếm được mở rộng trước các nút ở độ sâu $d+1$ . Tìm kiếm theo chiều rộng có thể được thực hiện bằng cách gọi giải thuật <code>general-search</code> với một hàm hàng đợi mà đưa các trạng thái mới được sinh ra vào cuối của hàng đợi, đứng sau tất cả các trạng thái mà đã được sinh ra trước nó:

**Function** *Tìm-kiếm-rộng(problem)*

**Returns** *một giải pháp hoặc thất bại*

**Return** *General-search (problem, xếp vào cuối hàng)*



**Hình 2.3. Tìm kiếm trên một cây nhị phân đơn giản**

Tìm kiếm theo chiều rộng là một chiến lược rất có phương pháp (có hệ thống) bởi vì nó xem xét

tất cả các đường đi có độ dài bằng 1 trước, sau đó đến tất cả những đường đi có độ dài

bằng 2, và cứ như vậy. Hình 2.3 chỉ ra quá trình của sự tìm kiếm trên một cây nhị phân đơn giản. Nếu như có một giải pháp, tìm kiếm theo chiều rộng đảm bảo sẽ tìm được nó, và nếu có nhiều giải pháp, tìm kiếm theo chiều rộng sẽ luôn tìm ra trạng thái đích nông nhất trước tiên. Dưới thuật ngữ của 4 tiêu chuẩn, tìm kiếm theo chiều rộng là hoàn thành, và nó được cung cấp một cách tối ưu chi phí đường dẫn bằng một hàm tăng của độ sâu các nút.

Chúng ta phải xem xét thời gian và dung lượng bộ nhớ nó sử dụng để hoàn thành một cuộc tìm kiếm. Để làm điều này, chúng ta xem xét một không gian trạng thái có tính giả thiết trong đó mỗi trạng thái có thể được mở rộng để tạo ra các trạng thái mới  $b$ . Chúng ta nói rằng yếu tố phân nhánh của những trạng thái này (và của cây tìm kiếm) là  $b$ . Gốc của cây tìm kiếm sinh ra  $b$  nút ở mức đầu tiên, mỗi nút đó lại sinh ra thêm  $b$  nút, và sẽ có cả thảy  $b^2$  nút ở mức thứ hai. Mỗi một nút này lại sinh ra thêm  $b$  nút, tạo ra  $b^3$  nút ở mức thứ ba, và cứ như vậy. Bây giờ chúng ta giả sử rằng giải pháp cho bài toán này có độ dài đường đi là  $d$ , như vậy số nút tối đa được mở rộng trước khi tìm thấy một giải pháp là :

$$1 + b + b^2 + b^3 + \dots + b^d$$

Đây là số nút tối đa, nhưng giải pháp có thể được tìm thấy ở bất cứ điểm nào thuộc mức có độ sâu là  $d$ . Do đó, trong trường hợp tốt nhất, số lượng các nút sẽ ít hơn.

### Tìm kiếm với chi phí thấp nhất

Phép tìm kiếm theo chiều rộng tìm được trạng thái đích, nhưng trạng thái này có thể không phải là giải pháp có chi phí thấp nhất đối với một hàm chi phí đường đi nói chung. Tìm kiếm với chi phí thấp nhất thay đổi chiến lược tìm kiếm theo chiều rộng bằng cách luôn luôn mở rộng nút có

chi phí thấp nhất (được đo bởi công thức tính chi phí được đi

$g(n)$ ), thay vì mở rộng nút có độ

sâu nông nhất. Dễ thấy rằng tìm kiếm theo chiều rộng chính là **tìm kiếm với chi phí thấp nhất** với  $g(n) = \text{DEPTH}(n)$ .

Khi đạt được những điều kiện nhất định, giải pháp đầu tiên được tìm thấy sẽ đảm bảo là giải pháp rẻ nhất, do nếu như có một đường đi khác rẻ hơn, nó đã phải được mở rộng sớm hơn và như vậy nó sẽ phải được tìm thấy trước. Việc quan sát các hành động của chiến lược sẽ giúp giải thích điều này. Hãy xem xét bài toán tìm đường đi. Vấn đề là đi từ S đến G, và chi phí của mỗi toán tử được ghi lại. Đầu tiên chiến lược sẽ tiến hành mở rộng trạng thái ban đầu, tạo ra đường đi tới A, B và C. Do đường đi tới A là rẻ nhất, nó sẽ tiếp tục được mở rộng, tạo ra đường đi SAG mà thực sự là một giải pháp, mặc dù không phải là phương án tối ưu. Tuy nhiên, giải thuật không công nhận nó là một giải pháp, bởi vì nó chi phí là 11, và nó bị che bởi đường đi SB có chi phí là 5 ở trong hàng đợi. Dường như thật là xấu hổ nếu sinh ra một giải pháp chỉ nhằm chôn nó ở sâu trong hàng đợi, nhưng điều đó là cần thiết nếu chúng ta muốn tìm một giải pháp tối ưu chứ không đơn thuần là

tìm bất cứ giải pháp nào. Bước tiếp theo là mở rộng SB, tạo ra SBG, và nó là đường đi rẻ nhất

còn lại trong hàng đợi, do vậy mục tiêu được kiểm tra và đưa ra một giải pháp. Phép tìm kiếm chi phí ít nhất tìm ra giải pháp rẻ nhất thoả mãn một yêu cầu đơn giản: Chi phí của một đường đi phải không bao giờ giảm đi khi chúng ta đi dọc theo đường đi. Nói cách khác, chúng ta mong muốn rằng

$$g(\text{Successor}(n)) \geq g(n) \text{ với mọi nút } n.$$

Giới hạn đối với chi phí đường đi không được giảm thực sự sẽ là vấn đề cần quan tâm nếu chi phí đường đi của một nút là tổng chi phí của các toán tử mà tạo nên đường đi. Nếu như mọi toán tử có một chi phí không âm, thì chi phí của đường đi không bao giờ có thể giảm đi khi chúng ta đi dọc theo đường đi và phép tìm kiếm với chi phí giống nhau có thể tìm được đường đi rẻ nhất mà không cần kiểm tra hết toàn bộ cây. Nhưng nếu một số toán tử có chi phí âm thì chẳng có một

23

CuuDuongThanCong.com <https://fb.com/tailieudientucntt>

cách tìm kiếm nào khác ngoài một phép tìm kiếm toàn bộ tất cả các nút để tìm ra giải pháp tối ưu, bởi vì chúng ta sẽ không bao giờ biết được rằng khi nào một đường đi sẽ chuyển sang một bước với chi phí âm cao và do đó trở thành đường đi tốt nhất trong tất cả các đường đi, bất kể là nó dài bao nhiêu và chi phí thế nào.

### Tìm kiếm theo chiều sâu

Tìm kiếm theo chiều sâu luôn luôn mở rộng một trong các nút ở mức sâu nhất của cây. Chỉ khi phép tìm kiếm đi tới một điểm cắt (một nút không phải đích mà không có phần mở rộng), việc tìm kiếm sẽ quay lại và mở rộng đối với những nút nông hơn. Chiến lược này có thể được thực hiện bởi General-search với một hàm hàng đợi mà luôn đưa các trạng thái mới được sinh ra vào trước hàng đợi. Do nút được mở rộng là sâu nhất, các nút kế tiếp của nó thậm chí sẽ sâu hơn và khi đó sẽ trở thành sâu nhất. Quá trình tìm kiếm được minh hoạ trong hình 2.4.



**Hình 2.4. Tìm kiếm theo chiều sâu**

Phép tìm kiếm theo chiều sâu yêu cầu dung lượng bộ nhớ rất khiêm tốn. Như hình vẽ cho thấy, nó chỉ cần phải lưu một đường duy nhất từ gốc tới nút lá, cùng với các nút anh em với các nút trên đường đi chưa được mở rộng còn lại. Đối với một không gian trạng thái với hệ số rẽ nhánh  $b$  và độ sâu tối đa  $m$ , phép tìm kiếm theo chiều sâu chỉ yêu cầu lưu trữ  $bm$  nút, ngược lại so với  $bd$  nút mà phép tìm kiếm theo chiều rộng yêu cầu trong trường hợp mục tiêu nông nhất ở độ sâu  $d$ .

Độ phức tạp thời gian của phép tìm kiếm sâu là  $O(bm)$ . Đối với những vấn đề mà có rất nhiều giải pháp, phép tìm kiếm sâu có thể nhanh hơn tìm kiếm rộng, bởi vì nó có một cơ hội tốt tìm ra một giải pháp chỉ sau khi khám phá một phần nhỏ của toàn bộ không gian. Tìm kiếm

theo chiều rộng sẽ vẫn phải tìm tất cả các đường đi có độ sâu  $d-1$  trước khi xem xét bất cứ đường đi nào có độ sâu  $d$ . Phép tìm kiếm theo chiều sâu vẫn cần thời gian  $O(bm)$  trong trường hợp tồi nhất.

Mặt hạn chế của phép tìm kiếm sâu là nó có thể bị tắc khi đi theo một đường sai. Rất nhiều bài toán có các cây tìm kiếm rất sâu, thậm chí vô hạn, vì vậy tìm kiếm sâu sẽ không bao giờ có thể quay trở lại được một trong các nút gần đỉnh của cây sau khi có một sự lựa chọn sai. Phép tìm kiếm sẽ luôn luôn tiếp tục đi xuống mà không quay trở lại, thậm chí khi có một giải pháp ở mức rất nông tồn tại. Như vậy đối với những bài toán này, phép tìm kiếm sâu sẽ hoặc là bị sa lầy trong một vòng lặp vô hạn và không bao giờ đưa ra một giải pháp, hoặc là cuối cùng nó có thể đưa ra một đường đi giải pháp mà dài hơn so với phương án tối ưu. Điều đó có nghĩa là phép tìm kiếm theo chiều sâu là không hoàn thành và không tối ưu. Bởi vì điều này, cần tránh sử dụng phép tìm kiếm sâu cho các cây tìm kiếm có độ sâu tối đa là vô hạn hoặc rất sâu.

24

CuuDuongThanCong.com <https://fb.com/tailieudientucntt>

Việc thực hiện phép tìm kiếm sâu với general-search là khá tầm thường:

**Function** *tìm kiếm sâu(bài toán)*

**returns** *một giải pháp hoặc thất bại*

**returns** *General-search(bài toán, xếp ở đâu hàng đợi)*

Người ta thường thực hiện phép tìm kiếm sâu cùng với một hàm đệ qui mà gọi tới chính nó ở lần lượt mỗi con của nó. Trong trường hợp này, hàng đợi được lưu trữ hoàn toàn trong không gian địa phương của mỗi lần gọi trong ngăn xếp gọi.

### Tìm kiếm theo độ sâu giới hạn

Tìm kiếm theo độ sâu giới hạn tránh các bẫy mà tìm kiếm theo chiều sâu mắc phải bằng cách đặt một giới hạn đối với độ sâu tối đa của đường đi. Giới hạn này có thể được thực hiện với một giải thuật tìm kiếm theo độ sâu giới hạn đặc biệt hoặc sử dụng các giải thuật tìm kiếm tổng quát với các toán tử theo dõi độ sâu. Chẳng hạn, trên bản đồ Rumania, có 20 thành phố, vì vậy chúng ta biết rằng nếu như có một giải pháp, thì nó phải có độ dài nhiều nhất là bằng 19. Chúng ta có thể thực hiện việc giới hạn độ sâu bằng cách sử dụng các toán tử dưới dạng “Nếu bạn ở thành phố A và vừa đi một đoạn đường ít hơn 19 bước, thì khởi tạo một trạng thái mới ở thành phố B với độ dài đường đi lớn hơn 1”. Với tập các toán tử mới này, chúng ta đảm bảo tìm thấy giải pháp nếu nó tồn tại, nhưng chúng ta vẫn không đảm bảo tìm thấy giải pháp ngắn nhất trước tiên: phép

tìm kiếm theo chiều sâu giới hạn là hoàn thành nhưng không tối ưu.

Nếu chúng ta chọn một giới hạn

độ sâu mà quá nhỏ, thì phép tìm kiếm theo chiều sâu giới hạn thậm chí không hoàn thành. Độ phức tạp về không gian và thời gian của phép tìm kiếm theo chiều sâu giới hạn tương đương với phép tìm kiếm sâu. Nó mất  $O(bl)$  thời gian và  $O(bl)$  không gian, với  $l$  là giới hạn độ sâu.

### Tìm kiếm lặp sâu dần (Iterative Deepening Search)

Thành phần khó khăn của tìm kiếm theo độ sâu giới hạn đem lại một giới hạn khá tốt. Chúng ta lấy 19 như là một giới hạn độ sâu “hiển nhiên” cho bài toán Rumania, nhưng thực ra nếu chúng ta nghiên cứu kỹ bản đồ, chúng ta sẽ thấy rằng bất cứ thành phố nào cũng có thể đi đến được từ bất kỳ thành phố nào khác với nhiều nhất là 9 bước. Con số này, được biết đến như là đường kính của không gian trạng thái, cho chúng ta một giới hạn độ sâu tốt hơn, và đưa đến một

phép tìm kiếm theo độ sâu giới hạn hiệu quả hơn. Tuy nhiên, đối với hầu hết các bài toán, chúng ta chỉ biết một giới hạn độ sâu tốt sau khi đã giải quyết xong bài toán.

**Function** tìm kiếm -lặp -sâu dần(bài toán) **returns** một dãy giải pháp

**Inputs:** bài toán, một vấn đề cần giải quyết

**For** độ sâu = 0 to  $\infty$  **do**

**If** tìm kiếm -độ sâu -giới hạn(bài toán, độ sâu) thành công

**then returns** kết quả

**End**

**Return** thất bại

**Hình 2.5 Giải thuật tìm kiếm lặp sâu dần**

25

CuuDuongThanCong.com <https://fb.com/tailieudientucntt>

Phép tìm kiếm lặp sâu dần là một chiến lược né tránh vấn đề lựa chọn giới hạn độ sâu tốt nhất và cố gắng thử tất cả các giới hạn độ sâu có thể: đầu tiên thử độ sâu bằng 0, sau đó độ sâu bằng 1, tiếp theo là 2, và cứ như vậy. Về mặt hiệu quả, việc lặp sâu dần kết hợp lợi ích của cả hai phép tìm kiếm theo chiều sâu và tìm kiếm theo chiều rộng. Đây là phương pháp tối ưu và đầy đủ, giống như phép tìm kiếm theo chiều rộng, nhưng chỉ yêu cầu bộ nhớ rất ít như phép tìm kiếm sâu yêu cầu. Thứ tự mở rộng các trạng thái tương tự với tìm kiếm rộng, ngoại trừ việc một số trạng thái được mở rộng nhiều lần.

Phép tìm kiếm lặp sâu dần có thể dường như là hơi lãng phí, bởi vì có rất nhiều trạng thái được mở rộng nhiều lần. Tuy nhiên, đối với hầu hết các bài toán, tổng chi phí của sự mở rộng nhiều lần này thực ra khá nhỏ. Bằng trực giác, có thể thấy rằng trong một cây tìm kiếm theo luật số mũ, hầu hết tất cả các nút là ở mức thấp nhất, vì vậy việc các mức ở bên trên được mở rộng nhiều lần sẽ không thành vấn đề lắm. nhắc lại rằng số lần mở rộng trong phép tìm kiếm theo độ sâu giới hạn tới độ sâu d với hệ số phân nhánh b là:

$$1 + b + b^2 + \dots + bd^{d-1} + bd^d + bd^{d+1} + \dots$$

Cụ thể, cho b=10, và d=5 thì số lần mở rộng là :

$$1 + 10 + 100 + 1000 + 10.000 + 100.000 = 111.111$$

Trong phép tìm kiếm lặp sâu dần, các nút ở mức dưới cùng được mở rộng một lần, những nút ở trên mức dưới cùng được mở rộng hai lần, và cứ như vậy đến gốc của cây tìm kiếm sẽ được mở rộng d+1 lần. Do đó tổng số lần mở rộng trong một phép tìm kiếm lặp sâu dần là :

$$(d+1)1 + (d)b + (d-1)b^2 + \dots + 3bd^{d-1} + 2bd^d + 1bd^{d+1} + \dots$$

Và cụ thể lại cho b=10, và d=5 thì số lần mở rộng là :

$$6 + 50 + 400 + 3000 + 20.000 + 100000 = 123.456$$

Như vậy chúng ta thấy, một phép tìm kiếm lặp sâu dần từ độ sâu 1 xuống tới độ sâu d chỉ mở rộng nhiều hơn khoảng 11% số nút so với phép tìm kiếm theo chiều rộng hay phép tìm kiếm theo chiều sâu tới độ sâu d khi hệ số phân nhánh b=10. Hệ số phân nhánh càng cao, tổng số các trạng thái được mở rộng lặp lại (nhiều lần) càng thấp, nhưng thậm chí khi hệ số phân nhánh là 2, phép tìm kiếm lặp sâu dần chỉ mở rộng số trạng thái nhiều gấp hai so với một phép tìm kiếm theo chiều

rộng đầy đủ. Điều đó có nghĩa rằng độ phức tạp thời gian của phép tìm kiếm lặp sâu dần vẫn là  $O(bd)$ , độ phức tạp không gian là  $O(bd)$ . Nói chung, lặp sâu dần là phép tìm kiếm được tham khảo đến khi có một không gian tìm kiếm lớn và độ sâu của giải pháp là không biết trước.

### Tìm kiếm tiến lùi

Ý tưởng của phép tìm kiếm tiến lùi là thực hiện đồng thời hai phép tìm kiếm: tìm kiếm từ trạng thái đầu  về phía trước và tìm kiếm  ngược lại từ trạng thái đích, và  dừng lại khi hai phép tìm kiếm này gặp nhau.

Đối với những bài toán mà hệ số rẽ nhánh là  $b$  ở cả hai hướng, phép tìm kiếm tiến lùi có thể đưa lại một sự khác biệt rất lớn. Nếu chúng ta vẫn giả sử rằng có một giải pháp ở độ sâu  $d$ , thì giải pháp sẽ được tìm thấy sau  $O(2bd/2) = O(bd/2)$  bước, bởi vì mỗi phép tìm kiếm tiến và lùi chỉ phải đi một nửa quãng đường. Xét ví dụ cụ thể, với  $b=10$  và  $d=6$ , phép tìm kiếm rộng sinh ra 1.111.111 nút, trong khi đó phép tìm kiếm tiến lùi thành công khi mỗi hướng ở độ sâu bằng 3, tại thời điểm 2.222 nút đã được tạo ra. Điều này về mặt lý thuyết nghe rất hấp dẫn. Chúng ta cần phải xem xét một số vấn đề trước khi thực hiện giải thuật

26

CuuDuongThanCong.com <https://fb.com/tailieudientucntt>

Câu hỏi chính là, tìm kiếm lùi từ đích có nghĩa là như thế nào? Chúng ta định nghĩa các nút tổ tiên (predecessor) của một nút  $n$  là tất cả các nút mà có nút  $n$  là nút kế vị (successor). Phép tìm kiếm lùi có nghĩa là sinh ra những nút tổ tiên liên tiếp bắt đầu từ nút đích.

Khi tất cả các toán tử là có thể đảo ngược, các tập kế vị và tập tổ tiên là giống hệt nhau. Tuy nhiên, đối với một số bài toán, việc tính toán các phần tử tổ tiên là rất khó khăn. Con số  $O(bd/2)$  của độ phức tạp thừa nhận rằng quá trình kiểm tra sự giao nhau của hai biên giới có thể được thực hiện trong một khoảng thời gian không đổi (như vậy, nó không phụ thuộc vào số trạng thái). Điều này thường có thể thu được với một bảng băm. Để hai phép tìm kiếm cuối cùng sẽ gặp nhau, tất cả các nút của ít nhất một trong hai phép tìm kiếm phải được lưu giữ trong bộ nhớ (giống như với trường hợp của phép tìm kiếm rộng). Điều này có nghĩa là độ phức tạp không gian của phép tìm kiếm tiến lùi không có đủ thông tin là  $O(bd/2)$ .

### So sánh các chiến lược tìm kiếm

Tiêu chuẩn	Tìm kiếm theo chiều rộng	Tìm kiếm chi phí thấp nhất	Tìm kiếm theo độ sâu	Tìm kiếm theo độ sâu giới hạn	Tìm kiếm lặp sâu dần	Tìm kiếm tiến lùi
Thời gian	$b^d$	$b^d$	$b^m$	$b^l$	$b^d$	$b^{d/2}$
Không gian	$b^d$	$b^d$	$bm$	$bl$	$bd$	$b^{d/2}$
Có tối ưu không?	Có	Có	Không	Không	Có	Có
Có hoàn thành?	Có	Có	Không		Có	Có

Đánh giá các chiến lược tìm kiếm, b là hệ số phân nhánh, d là độ sâu của giải pháp; m là độ sâu tối đa của cây tìm kiếm; l là giới hạn độ sâu						

Cho đến lúc này, chúng ta đã xét tất cả các vấn đề ngoại trừ còn một trong những vấn đề phức tạp nhất của quá trình tìm kiếm, đó là: khả năng lãng phí thời gian do việc mở rộng các trạng thái mà đã gặp và đã được mở rộng trước đó trên một số đường đi. Đối với một số bài toán, khả năng này không bao giờ xảy ra, mỗi trạng thái chỉ được đi đến theo một con đường. Việc xác định chính xác vấn đề bài toán 8 con hậu rất có tác dụng, đó là mỗi trạng thái chỉ có thể nhận được thông qua một đường đi.

Đối với rất nhiều bài toán, các trạng thái bị lặp lại là điều không thể tránh khỏi. Điều này có

ở tất cả các bài toán mà các toán tử là có thể đảo ngược, như các bài toán tìm đường đi và bài toán những nhà truyền giáo và những kẻ ăn thịt người. Các cây tìm kiếm cho những bài toán này là vô hạn, nhưng nếu chúng ta tĩa bớt một số trạng thái lặp lại, chúng ta có thể cắt cây tìm kiếm xuống còn kích thước hữu hạn, chỉ sinh ra một phần của cây mà mở rộng đồ thị không gian trạng thái. Thậm chí khi cây là hữu hạn, việc tránh các trạng thái lặp có thể dẫn đến một sự suy giảm theo cấp số mũ chi phí tìm kiếm. Một ví dụ kinh điển được mô tả ở hình 2.4. Không gian chỉ chứa  $m+1$  trạng thái, với  $m$  là độ sâu tối đa. Do cây bao gồm mỗi đường đi có thể trong không gian, nó có  $2m$  nhánh.

## 2.6. GIẢI QUYẾT VẤN ĐỀ VÀ CÁC KỸ THUẬT HEURISTIC

### *Các phương pháp tìm kiếm có đầy đủ thông tin*

Phần trước đã chỉ ra rằng các chiến lược tìm kiếm không đầy đủ thông tin có thể tìm thấy giải pháp đối với các bài toán bằng cách tạo ra một cách có hệ thống các trạng thái mới và kiểm tra chúng với mục tiêu. Điều không may là, những chiến lược này rõ ràng là không có hiệu quả trong hầu hết các trường hợp. Phần này cho một chiến lược tìm kiếm có thêm hiểu biết (có đủ thông tin) - một chiến lược sử dụng các tri thức đặc thù đối với bài toán - có thể tìm các giải pháp một cách hiệu quả hơn như thế nào. Phần này cũng chỉ ra các bài toán tối ưu có thể được giải quyết.

### Phương pháp tìm kiếm tốt nhất (Best-first)

**Function** best-first-search(*problem*, *hàm định giá*) return một dãy giải pháp

**Input** : *problem*, một bài toán

*Hàm định giá*, một hàm giá trị

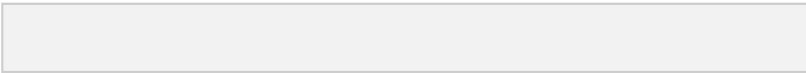
*Hàm hàng đợi* – một hàm mà sắp thứ tự các nút theo hàm giá



## Hình 2.6 Một ph ương pháp

**tìm kiếm tốt nhất sử dụng các giải thuật tìm kiếm tổng quát** Trong chương trước, chúng ta đã tìm thấy một số cách để áp dụng các tri thức cho qui

trình xác định rõ ràng chính xác một vấn đề bằng các thuật ngữ về trạng thái và các toán tử. Tuy nhiên, khi chúng ta được đưa cho một bài toán mà được chỉ rõ cụ thể, các sự lựa chọn của chúng ta là có giới hạn. Nếu chúng ta sử dụng giải thuật tìm kiếm tổng quát, thì cách duy nhất có thể áp dụng được tri thức là hàm "hàng đợi", hàm quyết định nút nào sẽ được mở rộng tiếp theo. Thông thường, tri thức để quyết định điều này được cung cấp bởi một hàm định giá trả về một số có nghĩa mô tả sự mong muốn được mở rộng nút. Khi các nút được xếp thứ tự để nút nào có định giá tốt nhất sẽ được mở rộng trước. Chiến lược như vậy được gọi là phép tìm kiếm tốt nhất (best-first). Nó có thể được cài đặt trực tiếp với tìm kiếm tổng quát, như hình 2.6.

Tên gọi  "tìm kiếm tốt nhất" là phép tìm kiếm quan trọng nhưng không chính xác. Nếu chúng ta mở rộng nút tốt nhất trước tiên, đó không phải là phép tìm kiếm - đó là một cách đi thẳng đến mục tiêu. Điều có thể làm là chọn nút tỏ ra là tốt nhất theo hàm giá. Nếu hàm giá là rõ, thì nút này sẽ là nút tốt nhất. Trong thực tế, hàm giá thỉnh thoảng có sai sót và việc tìm kiếm bị lạc đường. Tuy nhiên, chúng ta sẽ dùng tên "tìm kiếm tốt nhất", vì tên "tìm kiếm về ngoài tốt nhất" có vẻ không tiện.

Khi có một họ giải thuật tìm kiếm tổng quát với các hàm theo thứ tự khác nhau, tồn tại một họ các giải thuật tìm kiếm tốt nhất với các hàm giá khác nhau. Vì mục đích là tìm kiếm các giải pháp có chi phí thấp, những giải thuật này sử dụng phương pháp đánh giá các chi phí của giải pháp và cố gắng tối thiểu nó. Chúng ta có một phương pháp đo: sử dụng chi phí

28

CuuDuongThanCong.com <https://fb.com/tailieudientucntt>

đường đi  $g$  để quyết định đường đi nào sẽ mở. Tuy nhiên, phương pháp này không tìm trực tiếp về phía đích. Để làm chụm phép tìm kiếm, phương pháp kết hợp một số cách đánh giá chi phí đường đi từ một trạng thái tới trạng thái đích gần nhất. Xét hai phương pháp cơ bản. Phương pháp thứ nhất mở nút gần đích nhất. Phương pháp thứ hai mở nút ở đường đi có chi phí ít nhất.

**Tối thiểu hoá chi phí đánh giá để đi tới mục tiêu: phép tìm kiếm tham lam** Một trong những chiến lược tìm kiếm tốt nhất trước đơn giản nhất là tối thiểu chi phí ước lượng để đi tới mục tiêu. Đó là, nút mà trạng thái của nó được đánh giá là gần với trạng thái mục tiêu nhất luôn luôn được mở rộng trước. Đối với hầu hết các bài toán, chi phí của việc đi tới đích từ một trạng thái nào đó có thể được ước lượng nhưng không thể xác định chính xác. Một hàm mà tính toán những ước lượng chi phí như vậy được gọi là hàm heuristic và thường được biểu diễn bằng chữ cái  $h$ :



$h(n)$  = chi phí ước lượng của đường đi rẻ nhất từ trạng thái ở nút  $n$  tới trạng thái đích. Một phép tìm kiếm tốt nhất trước mà sử dụng  $h$  để lựa chọn nút mở rộng tiếp theo được gọi là phương pháp tìm kiếm tham lam (greedy search), vì các lý do mà chúng ta sẽ thấy rõ ràng sau đây. Cho một hàm heuristic  $h$ , phép tìm kiếm tham lam có thể được thực hiện như sau:

**function** greedy-search(*problem*) **returns** một giải pháp hoặc thất bại  
**return** best-first-search(*problem*,  $h$ )

Nói một cách chính thức,  $h$  có thể là bất cứ hàm nào. Chúng ta sẽ chỉ yêu cầu là  $h(n) = 0$  nếu  $n$  là một mục tiêu.

Để hình dung một hàm heuristic là như thế nào, chúng ta cần chọn một bài toán điển hình, bởi vì các hàm heuristic chuyên xác định các bài toán đặc biệt. Chúng ta hãy quay trở lại với bài toán tìm đường đi từ Arad đến Bucaret.

Một hàm heuristic tốt đối với những bài toán tìm đường đi giống như thế này là khoảng cách đường thẳng (straight-line distance hay SLD) tới mục tiêu. Tức là,

$$hSLD(n) = \text{khoảng cách đường thẳng giữa } n \text{ và vị trí đích.}$$

Với phép heuristic khoảng cách -đường thẳng, nút đầu tiên sẽ mở rộng từ Arad sẽ là Sibiu, bởi vì

nó gần Bucaret hơn so với Zerind và Timisoara. Nút mở rộng tiếp theo sẽ là Fagaras, do nó là nút

gần nhất. Fagaras sẽ sinh ra Bucaret, và là đích. Đối với bài toán này, phép heuristic dẫn tới chi phí tìm kiếm tối thiểu: nó tìm một giải pháp mà không cần mở một nút nào không nằm trên đường đi giải pháp (đường đi tới đích). Tuy nhiên, nó không phải là hoàn toàn tối ưu: đường đi mà nó tìm ra đi qua Sibiu và Fagaras tới Bucaret dài hơn đường đi xuyên qua Rimnicu Vilcea và Pitesti (rồi tới Bucaret) là 32 km. Con đường này nó không tìm ra bởi vì Fagaras gần với Bucaret theo khoảng cách đường thẳng hơn so với Rimnicu, vì vậy nó được mở trước. Chiến lược ưu tiên chọn khả năng có “miếng ngoạm lớn nhất” (tức là đi bước đầu tiên đi được xa nhất) không quan tâm đến các chi phí còn lại để đi đến đích, không đếm xỉa đến việc bước đi này có phải là tốt nhất xét về toàn cục hay không – chính vì thế nó được gọi là “phương pháp tìm kiếm tham lam”. Mặc dù tham lam được coi là một trong 7 lỗi nặng, nhưng các giải thuật tham lam thường tỏ ra khá hiệu quả. Chúng có thiên hướng tìm giải pháp nhanh chóng, mặc dù như đã chỉ ra trong ví dụ vừa

rồi, chúng không phải luôn luôn tìm ra các giải pháp tối ưu: cần phải có sự phân tích một cách kỹ các giải pháp toàn cục, chứ không chỉ một sự lựa chọn tốt nhất tức thì.

Phép tìm kiếm tham lam tương tự phép tìm kiếm theo độ sâu ở điểm là nó ưu tiên đi theo một đường đơn tới đích, nhưng nó sẽ quay lui khi gặp đường cụt. Nó có những nhược điểm giống với phương pháp tìm kiếm sâu - không tối ưu, và không hoàn thành vì có thể rơi vào một đường vô hạn và không bao giờ quay lại để chọn khả năng khác. Độ phức tạp thời gian trong trường hợp tồi nhất của phép tìm kiếm tham lam là  $O(b^m)$ , với  $m$  là độ sâu tối đa của không gian tìm kiếm. Bởi vì phép tìm kiếm tham lam lưu trữ tất cả các nút trong bộ nhớ, độ phức tạp không gian của nó tương tự như độ phức tạp thời gian. Với một hàm heuristic tốt, độ phức tạp không gian và độ phức tạp thời gian có thể giảm đáng kể. Lượng giảm phụ thuộc vào bài toán cụ thể và chất lượng của hàm  $h$ .

## Tối thiểu hoá tổng chi phí đường đi: Thuật toán tìm kiếm A\*

Phương pháp tìm kiếm heuristic tối thiểu hoá chi phí dự tính tới đích,  $h(n)$ , và do đó giảm chi phí tìm kiếm đáng kể. Điều không may là, đó không phải là phương pháp tối ưu cũng như hoàn thành. Mặt khác, phép tìm kiếm theo chi phí ít nhất lại tối thiểu hoá chi phí đường tính đến thời điểm hiện tại,  $g(n)$ ; Đó là phương pháp tìm kiếm tối ưu và hoàn thành, nhưng có thể rất không hiệu quả. Sẽ rất tốt nếu chúng ta kết hợp cả hai phương pháp này để lợi dụng điểm mạnh của cả hai phương pháp. Rất may là chúng ta có thể làm được chính xác điều đó, kết hợp hai hàm định giá đơn giản bằng cách cộng chúng lại:

$$f(n) = g(n) + h(n) .$$

Do  $g(n)$  đưa ra chi phí đường đi từ nút đầu tới nút  $n$ , và  $h(n)$

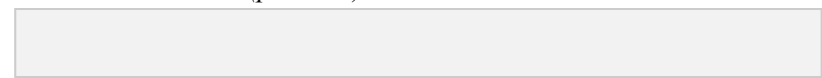
là chi phí ước tính của đường đi rẻ nhất từ  $n$  đến đích, có :

$$f(n) = \text{chi phí ước tính của giải thuật tốt nhất đi qua } n$$

Như thế, nếu chúng ta cố gắng tìm giải pháp rẻ nhất, nút cần mở rộng trước hợp lý một cách hợp lý nhất là nút có giá trị thấp nhất của  $f$ . Điều thú vị về phương pháp này là phương pháp này còn hơn cả sự hợp lý. Thực tế chúng ta có thể chứng minh rằng nó là hoàn thành và tối ưu, với một hạn chế đơn giản đối với hàm  $h$ .

Hạn chế là cần chọn một hàm  $h$  mà không vượt quá chi phí đi tới đích. Một hàm  $h$  như vậy được gọi là một heuristic có thể chấp nhận. Những heuristic có thể chấp nhận là theo quan điểm của những người lạc quan, vì họ nghĩ chi phí của việc giải quyết vấn đề là ít hơn thực tế. Sự lạc quan này cũng sẽ chuyển hàm  $f$ : Nếu  $h$  là chấp nhận được,  $f(n)$  không bao giờ vượt quá chi phí thực tế của giải pháp  $n$ . Phép tìm kiếm tốt nhất sử dụng  $f$  như là một hàm giá và một hàm  $h$  chấp nhận được với tên **phương pháp tìm kiếm A\***.

**Function**  $A^*\text{-search}(\text{problem})$

 **return** một giải pháp hoặc thất bại **Return**  $\text{best-first-search}(\text{problem}, g+h)$

### Hình 2.7.

Ví dụ rõ ràng về phép heuristic chấp nhận được là khoảng cách đường thẳng hSLD mà chúng ta sử dụng để đi đến Bucaret. Khoảng cách đường thẳng là chấp nhận được bởi vì đường đi ngắn nhất giữa bất cứ hai điểm là một đường thẳng. Trong hình 2.7, chúng ta chỉ ra một số bước đầu tiên của phép tìm kiếm A\* tới Bucaret sử dụng phép heuristic hSLD. Chú ý rằng phép tìm kiếm A\* ưu tiên mở rộng từ Rimnicu Vilcea hơn so với mở rộng từ Fagaras. Mặc dù thậm chí

Fagaras gần Bucaret hơn, đường đi tới Fagaras không hiệu quả bằng đường đi tới Rimnicu trong việc tiến gần tới Bucaret. Bạn đọc có thể mong muốn tiếp tục ví dụ này để xem điều gì sẽ xảy ra tiếp theo.

### Sự hoạt động của phép tìm kiếm A\*

Trước khi chúng ta chứng minh tính hoàn thành và tính tối ưu của A\*, chúng ta nên đưa ra một bức tranh trực giác về hoạt động của phương pháp tìm kiếm này (Hình 2.8). Một minh họa không thể thay thế cho một bằng chứng, nhưng nó thường dễ nhớ và có thể sử dụng tạo ra các chứng cứ khi có yêu cầu. Trước tiên, một sự quan sát ban đầu: nếu như bạn kiểm tra các cây tìm kiếm, bạn

sẽ chú ý một hiện tượng thú vị: Dọc theo bất cứ đường đi nào từ gốc, chi phí  $f$  không bao giờ tăng. Điều này không phải là ngẫu nhiên. Nó là đúng đối với hầu như tất cả các heuristic chấp nhận được. Người ta nói một heuristic như vậy là đưa ra sự đơn điệu (monotonicity1).

Nếu heuristic là một trong những heuristic kỳ cục mà không phải là đơn điệu. Chúng ta có thể sửa chữa nhỏ để phục hồi tính đơn điệu. Xét hai nút  $n$  và  $n'$ , với  $n$  là nút cha của  $n'$ . Giả sử  $g(n) = 3$  và  $h(n) = 4$ , ta có,  $f(n) = g(n) + h(n) = 7$  – như vậy ta biết rằng giá trị thực của một giải pháp tới  $n$  ít nhất là 7. Cũng giả sử  $g(n') = 4$  và  $h(n') = 2$ , do vậy  $f(n') = 6$ . Rõ ràng, đây là một ví dụ về một heuristic không đơn điệu. Rất may là, từ thực tế rằng bất cứ đường đi nào đến  $n'$  thì cũng là đường đi đến  $n$ , chúng ta có thể thấy rằng giá trị 6 là không có ý nghĩa gì, bởi vì chúng ta đã biết chi phí thực tế ít nhất là 7. Như vậy, chúng ta nên kiểm tra, mỗi lần chúng ta tạo ra một nút mới, để xem chi phí  $f$  của nó có nhỏ hơn chi phí  $f$  của nút cha của nó hay không: nếu nhỏ hơn, chúng ta sẽ sử dụng chi phí  $f$  của nút cha của nó:

$$f(n') = \max(f(n), g(n') + h(n')).$$

Theo cách này, ta bỏ qua các giá trị dẫn sai đường có thể xảy ra với một heuristic không đơn điệu. Công thức này gọi là **cực đại đường đi**. Nếu sử dụng công thức đó, thì  $f$  luôn không giảm dọc theo bất cứ đường đi từ gốc, giá trị  $h$  được cung cấp là chấp nhận được.



**Hình 2.8** Các giai đoạn của phép tìm kiếm A\* để đi đến Bucharest. Các nút được gán nhãn với  $f$   
**Độ phức tạp của thuật toán A\***

$= g + h$ . Các giá trị  $h$  là các khoảng cách đường thẳng tới Bucharest lấy từ giả thiết 31

Phương pháp tìm kiếm A\* là hoàn thành, tối ưu và hiệu quả một cách tối ưu trong số tất cả các thuật toán như vậy. Điều đó không có nghĩa là A\* là câu trả lời cho tất cả các yêu cầu tìm kiếm. Đối với hầu hết các bài toán, số nút trong không gian tìm kiếm đường viền mục tiêu là cấp số mũ theo độ dài của giải pháp. Mặc dù không chứng minh, nó đã được chỉ ra rằng độ tăng theo cấp số mũ sẽ xảy ra trừ phi sai số trong hàm heuristic không tăng nhanh hơn logarit của chi phí đường đi

thực tế. Theo ký hiệu toán học, điều kiện đối với độ tăng nhỏ hơn cấp số mũ là :

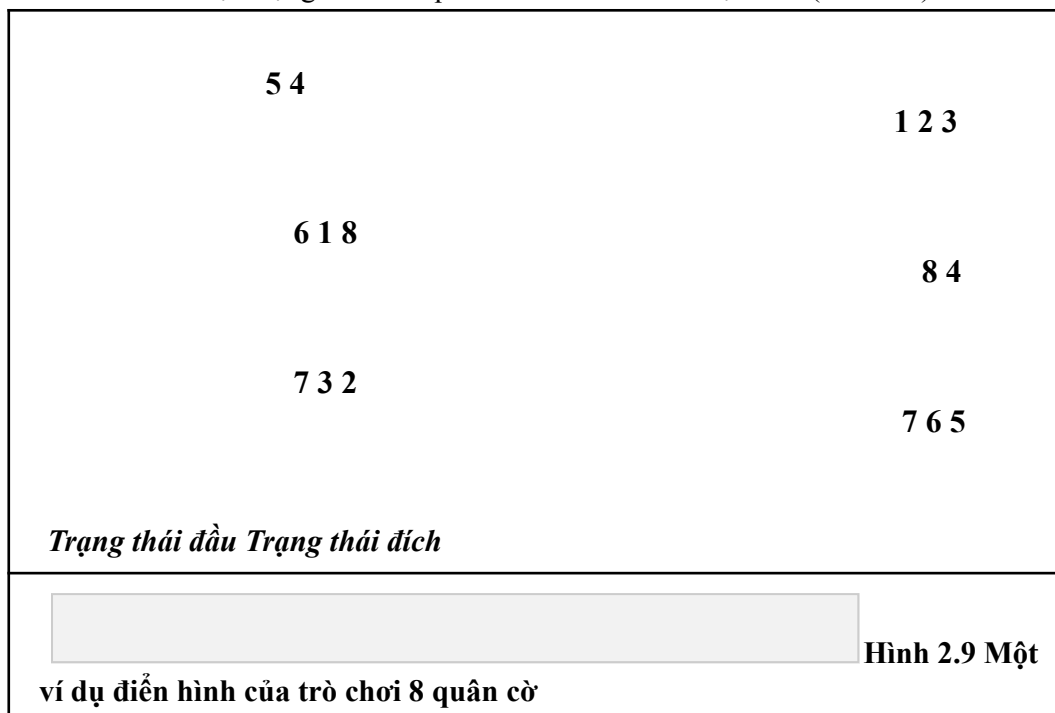
$$|h(n) - h^*(n)| \leq O(\log^*(n)),$$

với  $h^*(n)$  là chi phí thực tế của việc đi từ  $n$  đến mục tiêu. Đối với hầu hết tất cả các heuristic trong thực tế sử dụng, sai số ít nhất cũng tỷ lệ với chi phí đường đi, và độ tăng theo cấp số mũ cuối cùng sẽ vượt quá bất cứ khả năng của máy tính nào. Tất nhiên, việc sử dụng một heuristic tốt vẫn cho chúng ta một tiết kiệm rất lớn so với các phép tìm kiếm không đủ thông tin. Trong phần tiếp theo, chúng ta sẽ xem xét đến vấn đề thiết kế các heuristic tốt.

Tuy nhiên, thời gian tính toán không phải là mặt trở ngại chính của  $A^*$ . Do nó lưu trữ tất cả các nút được tạo ra trong bộ nhớ,  $A^*$  thường bị vượt ra khỏi bộ nhớ rất lâu trước khi nó hết thời gian. Các giải thuật phát triển gần đây đã vượt qua trở ngại về dung lượng bộ nhớ mà không phải hi sinh tính tối ưu hay tính hoàn thành.

## Các hàm heuristic

Cho đến lúc này, chúng ta mới chỉ xem xét một ví dụ về một heuristic: khoảng cách đường thẳng đối với các bài toán tìm đường đi. Trong phần này, chúng ta sẽ xét các hàm heuristic đối với trò chơi số 8. Điều này sẽ soi rọi về yếu tố tự nhiên của các hàm heuristic nói chung. Trò chơi số 8 là một trong những bài toán tìm kiếm theo phương pháp heuristic sớm nhất. Như đã đề cập trong phần 2.5, mục tiêu của trò chơi này là di các con cờ theo chiều ngang hoặc chiều dọc vào ô trống cho đến khi thu được trạng thái các quân cờ như mô hình mục tiêu (hình 2.9).



Trò chơi số 8 ở mức độ khó vừa phải nên là một trò chơi rất thú vị. Một giải pháp điển hình gồm khoảng 20 bước, mặc dù tất nhiên con số này biến đổi phụ thuộc vào trạng thái đầu. Hệ số rẽ nhánh khoảng bằng 3 (khi ô trống ở giữa, có bốn khả năng di chuyển; khi nó ở góc bàn cờ, có hai khả năng di chuyển; và khi nó ở trên các cạnh, có 3 khả năng đi). Điều này có nghĩa là một phép tìm kiếm vét cạn tới độ sâu 20 sẽ xem xét khoảng  $320 = 3,5 \times 10^9$  trạng thái. Bằng cách theo dõi các trạng thái lặp lại, chúng ta có thể giảm số trạng thái này xuống đáng kể, bởi vì chỉ có  $9! = 362880$  các sự sắp xếp khác nhau của 9 ô vuông. Đây vẫn là một số rất lớn các trạng thái, vì thế

bước tiếp theo là tìm một hàm heuristic tốt. Nếu chúng ta muốn tìm kiếm các giải thuật ngắn nhất, chúng ta cần một hàm heuristic mà không bao giờ ước đoán vượt quá số các bước đi tới mục tiêu. Sau đây là hai “ứng cử viên”:

- $h_1$  = số lượng quân cờ mà ở sai vị trí. Đối với hình 2.9, 7 trong số 8 quân cờ ở sai vị trí, vì vậy trạng thái đầu sẽ có  $h_1 = 7$ .  $h_1$  là một hàm heuristic chấp nhận được, bởi vì rõ ràng là bất cứ quân cờ nào mà đang ở sai vị trí phải di chuyển ít nhất một lần.
- $h_2$  = tổng số khoảng cách của các quân cờ so với vị trí mục tiêu. Bởi vì các quân cờ không thể đi theo các đường chéo, khoảng cách mà chúng ta tính tổng sẽ là tổng của các khoảng cách theo chiều ngang và theo chiều dọc. Khoảng cách này đôi khi được gọi là “khoảng cách khối thành phố”(city block distance) hoặc khoảng cách **Manhattan**  $h_2$  là chấp nhận được, vì bất cứ nước đi nào cũng chỉ có thể di chuyển một quân cờ một bước gần hơn tới đích. Các quân cờ từ 1 đến 8 trong trạng thái đầu cho ta một khoảng cách Manhattan là:

$$h_2 = 2 + 3 + 3 + 2 + 4 + 2 + 0 + 2 = 18$$

### **Hiệu quả (tác dụng) của độ chính xác heuristic trong khi thực hiện**

Một cách để xác định chất lượng của một hàm heuristic là hệ số phân nhánh hiệu quả  $b^*$ . Nếu tổng số các nút được mở rộng bởi  $A^*$  đối với một bài toán nào đó là  $N$ , và độ sâu giải pháp là  $d$ , thì  $b^*$  là hệ số phân nhánh mà một cây đồng dạng có độ sâu  $d$  sẽ phải có để chứa được  $N$  nút. Như vậy,

$$N = 1 + b^* + (b^*)^2 + \dots + (b^*)^d$$

Chẳng hạn, nếu  $A^*$  tìm thấy một giải pháp ở độ sâu bằng 5 sử dụng 52 nút, thì hệ số phân nhánh hiệu quả là 1,91. Thông thường, hệ số phân nhánh hiệu quả đưa ra bởi một heuristic cho trước là khá ổn định đối với đa số các bài toán. Do đó, các phép đo thử nghiệm giá trị  $b^*$  trong một tập nhỏ các bài toán có thể đưa ra một chỉ dẫn tốt khi xét tổng thể hàm heuristic. Một hàm heuristic được thiết kế tốt sẽ có giá trị  $b^*$  gần với 1, cho phép giải quyết một số lượng lớn các bài toán. Để kiểm tra các hàm heuristic  $h_1$  và  $h_2$ , chúng ta ít khi sinh ra 100 bài toán, mỗi bài toán với các độ dài giải pháp là 2, 4, ..., 20, và giải quyết chúng với phép tìm kiếm  $A^*$  với  $h_1$  và  $h_2$ , cùng với phép tìm kiếm lặp sâu dần không đầy đủ thông tin. Hình 2.8 đưa ra số trung bình các nút được mở rộng bởi chiến lược tìm kiếm và hệ số phân nhánh hiệu quả. Kết quả cho thấy là  $h_2$  tốt hơn  $h_1$  và phép tìm kiếm thiếu thông tin tồi hơn nhiều.

### **Xây dựng các hàm heuristic**

Chúng ta đã thấy rằng, cả  $h_1$  và  $h_2$  là những heuristic khá tốt đối với trò chơi số 8, và  $h_2$  thì tốt hơn  $h_1$ . Nhưng chúng ta không biết làm thế nào để phát minh ra một hàm heuristic. Làm sao một người có thể có được một heuristic như  $h_2$ ? Một máy tính có thể phát minh một cách máy móc ra được một heuristic như vậy không?

$h_1$  và  $h_2$  là các đánh giá (ước đoán) đối với độ dài đường đi còn lại trong trò chơi số 8, nhưng chúng cũng có thể được xem là các độ dài đường đi có độ chính xác tuyệt vời đối với những kiểu đơn giản hoá của trò chơi này. Nếu như qui tắc của trò chơi được thay đổi để một quân cờ có thể di chuyển đến bất cứ chỗ nào, thay vì chỉ có thể đi đến ô trống ngay cạnh nó, thì  $h_1$  sẽ đưa ra một cách chính xác số các bước của giải pháp gần nhất. Tương tự, nếu một quân cờ có thể đi một ô

theo tất cả các hướng, thậm chí đi vào ô đã bị chiếm bởi một quân cờ khác, thì h2 sẽ đưa ra con số chính xác các bước đi của giải pháp ngắn nhất. Một bài toán với ít ràng buộc hơn đối với các toán tử được gọi là một bài toán giải trí (relaxed problem). *Thường xảy ra trường hợp*

CuuDuongThanCong.com <https://fb.com/tailieudientucntt>

*là chi phí đường đi của một giải pháp đúng đối với một bài toán giải trí là một heuristic tốt đối với bài toán gốc (ban đầu).*

Nếu việc định nghĩa vấn đề được viết dưới dạng một ngôn ngữ chính thức, có thể xây dựng các bài toán thư giãn một cách tự động. 3 Ví dụ, nếu các toán tử của trò chơi số 8 được miêu tả như sau:

Quân cờ A có thể đi từ ô A đến ô B nếu A là cạnh B và B trống, thì chúng ta có thể tạo ra ba bài toán giải trí bằng cách bỏ đi một hoặc nhiều hơn các điều kiện:

- (a) Quân cờ A có thể đi từ ô A đến ô B nếu A ở cạnh B.
- (b) Quân cờ A có thể đi từ ô A đến ô B nếu B là trống.
- (c) Quân cờ A có thể đi từ ô A đến ô B.

Gần đây, một chương trình được gọi là ABSOLVER đã được viết mà có thể tạo ra các heuristic một cách tự động từ các khái niệm xác định bài toán, sử dụng phương pháp “bài toán thư giãn” và rất nhiều các kỹ thuật khác (Prieditis, 1993). ABSOLVER sinh ra một heuristic mới cho trò chơi số 8 tốt hơn bất cứ heuristic đang tồn tại nào, và tìm ra heuristic hữu ích đầu tiên cho trò chơi nổi tiếng là hình khối lập phương Rubik.

Một vấn đề của việc tạo ra các hàm heuristic mới là chúng ta thường thất bại trong việc có được một heuristic “tốt nhất một cách rõ ràng”. Nếu chúng ta có một tập các heuristic chấp nhận được  $h_1 \dots h_m$  cho một bài toán, và không có hàm nào vượt trội hơn hàm nào, chúng ta nên chọn heuristic nào? Như chúng ta sẽ thấy, chúng ta không cần thiết phải lựa chọn. Chúng ta có thể có heuristic tốt nhất, bằng cách định nghĩa:

$$h(n) = \max(h_1(n), \dots, h_m(n)).$$

Heuristic tổ hợp này sử dụng bất cứ hàm nào chính xác nhất đối với nút trong câu hỏi. Do các heuristic thành phần là chấp nhận được,  $h$  cũng chấp nhận được. Hơn nữa,  $h$  vượt trội hơn so với tất cả các heuristic thành phần tạo nên nó.

Một cách khác để phát minh ra một heuristic tốt là sử dụng thông tin thống kê. Điều này có thể thu được bằng cách chạy một phép tìm kiếm đối với một số các bài toán đào tạo, như 100 mô hình ngẫu nhiên của trò chơi số 8 được chọn, và thu thập các thống kê. Ví dụ, chúng ta có thể tìm thấy rằng, khi  $h_2(n) = 14$ , thì 90% của quãng đường thực sự để tới đích là 18. Như vậy khi gặp những bài toán “thực sự”, chúng ta có thể sử dụng 18 làm giá trị quãng đường bất cứ khi nào  $h_2(n)$  cho giá trị 14. Tất nhiên, nếu chúng ta sử dụng các thông tin theo xác suất như thế này, chúng ta đang từ bỏ sự bảo đảm về tính có thể chấp nhận được, nhưng tính trung bình chúng ta có lẽ sẽ mở rộng ít nút hơn.

Thông thường có thể lấy ra các đặc điểm của một trạng thái mà đóng góp cho hàm định giá heuristic của nó, thậm chí nếu rất khó nói chính xác sự đóng góp là gì. Chẳng hạn, mục tiêu trong đánh cờ là chiếu tướng đối phương, và các đặc điểm liên quan như số quân mỗi loại của mỗi bên, số quân mà bị ăn bởi quân của đối thủ, v. v. Thông thường, hàm định giá được giả định là tổ hợp tuyến tính của các giá trị đặc điểm. Thậm chí nếu chúng ta không biết các đặc điểm quan

trọng như thế nào, hay thậm chí một đặc điểm là tốt hay xấu, ta vẫn có thể sử dụng một giải thuật học tập để thu được các hệ số hợp lý cho mỗi đặc điểm. Ví dụ, trong đánh cờ, một chương trình có thể học hỏi được rằng con hậu của một người nên có hệ số dương lớn, trong khi một con tốt của đối thủ nên có một hệ số âm nhỏ.

Một yếu tố khác mà chúng ta chưa xem xét đến là chi phí tìm kiếm của việc chạy thật sự hàm heuristic trên một nút. Chúng ta vừa giả định rằng chi phí của việc tính toán hàm heuristic

34

CuuDuongThanCong.com <https://fb.com/tailieudientucntt>

tương đương với chi phí mở rộng một nút. do vậy tối thiểu hoá số lượng nút mở rộng là một điều tốt. Nhưng nếu hàm heuristic phức tạp đến nỗi mà tính toán giá trị của nó cho một nút mất khoảng thời gian để mở rộng hàng trăm nút thì chúng ta cần phải cân nhắc. Cuối cùng, rất dễ để có một hàm heuristic mà chính xác tuyệt đối – nếu chúng ta cho phép hàm heuristic thực hiện, ví dụ, một phép tìm kiếm theo chiều rộng “kín đáo”. Điều đó sẽ tối thiểu hoá số lượng các nút được mở rộng bởi phép tìm kiếm thực sự, nhưng nó sẽ không tối thiểu hoá chi phí tìm kiếm tổng thể. Một hàm heuristic tốt phải vừa chính xác vừa hiệu quả.

## 2.7. CÁC PHƯƠNG PHÁP GIẢI QUYẾT VẤN ĐỀ KHÁC

### Phương pháp tìm kiếm $A^*$ lặp sâu dần ( $IDA^*$ )

**function**  $IDA^*(problem)$  return một dãy giải pháp

**Inputs:** *problem*, một bài toán, vấn đề

**Các biến địa phương:** *giới hạn-f*, giới hạn chi phí *f* hiện thời

*root*, một nút

*root* - Make-node(trạng thái đầu[*problem*])

*giới hạn-f* - chiphi- *f*(*root*)

**loop do**

*giải pháp*, *giới hạn-f* - DFS-contour(*root*, *giới hạn-f*)

**if** *giải pháp* khác rỗng **then return** *giải pháp*

**if** *giới hạn-f* =  $\infty$  **then return** thất bại; end

**function** dfs-contour(*node*,

*giới hạn-f*) **returns** một dãy giải

pháp và một giới hạn chi phí *f* mới. **Input:** *node*, một nút

*Giới hạn -f*, giới hạn chi phí *f* hiện thời

**Các biến địa phương:** *next-f*, giới hạn chi phí *f* của contour tiếp theo, ban đầu là  $\infty$

**If** chiphi- *f*[*node*] > *giới hạn-f* **then return** rỗng, chiphi- *f*[*node*]

**If** goal-test[*problem*](state[*node*]) **then return** *node*, *giới hạn-f*

**For** mỗi nút *s* trong successor(*node*) **do**

*Giải pháp*, *f*-mới – dfs-contour(*s*, *giới hạn-f*)

**If** *giải pháp* khác rỗng **then return** *giải pháp*, *giới hạn-f*

*Next-f*  $\leftarrow$  MIN(*next-f*, *new-f*); end

**Return** rỗng, *next-f*

### Hình 2.10. Giải thuật tìm kiếm IDA

Trong ph  ần trước, chúng ta đã



chỉ ra rằng lặp sâu dần là một kỹ thuật rất hiệu quả để giảm bộ nhớ. Chúng ta có thể lại thử phép đó lần nữa, biến phép tìm kiếm  $A^*$  trở thành  $A^*$  lặp sâu dần hay phương pháp IDA\* (Hình 2.10). Trong giải thuật này, mỗi phép lặp là một phép tìm kiếm theo chiều sâu, cũng như trong phương pháp tìm kiếm lặp sâu dần bình thường. Phép tìm kiếm theo chiều sâu được sửa đổi để sử dụng một giới hạn chi phí  $f$  thay vì một giới hạn độ sâu. Như vậy, mỗi vòng lặp mở rộng các nút bên trong đường viền của chi phí  $f$  hiện tại, nhìn vào trong đường viền để tìm đường viền tiếp theo ở đâu. Khi phép tìm kiếm bên trong một đường viền đã cho đã hoàn thành, vòng lặp mới lại bắt đầu sử dụng một chi phí  $f$  mới cho đường viền tiếp theo. IDA\* là hoàn thành và tối ưu với cùng một dự báo cho trước như phép tìm kiếm  $A^*$ , nhưng do nó là phương pháp tìm kiếm theo chiều sâu, nó chỉ yêu cầu không gian bộ nhớ tỷ lệ với đường đi dài nhất mà nó khám

35

CuuDuongThanCong.com <https://fb.com/tailieudientucntt>

phá. Nếu  $\bar{D}$  là chi phí nhỏ nhất và  $f^*$  là chi phí giải pháp tối ưu, thì trong trường hợp tồi nhất, IDA\* sẽ yêu cầu  $bf^*/\bar{D}$  nút lưu trữ. Trong hầu hết các trường hợp,  $bd$  là sự ước đoán rất tốt đối với yêu cầu về dung lượng lưu trữ


### Phương pháp tìm kiếm SMA\*

Những khó khăn của IDA\* về không gian bài toán nào đó có thể tìm ra được là sử dụng quá ít bộ nhớ. Giữa các vòng lặp, nó chỉ lưu lại một số duy nhất, đó là giới hạn chi phí  $f$  hiện thời. Do nó không thể nhớ được các bước đi của nó, IDA\* buộc phải lặp lại chúng. Điều này càng đúng trong các không gian trạng thái mà là các bản đồ thay vì các cây. IDA\* có thể được sửa đổi để kiểm tra đường đi hiện thời đối với những trạng thái lặp lại, nhưng nó không thể tránh được các trạng thái lặp lại được tạo ra bởi các đường đi thay đổi.

Trong phần này, chúng ta miêu tả giải thuật SMA\* mà có thể sử dụng tất cả bộ nhớ sẵn có để tiến hành việc tìm kiếm. Việc sử dụng bộ nhớ nhiều hơn chỉ có thể cải thiện được hiệu quả tìm kiếm – một giải thuật có thể luôn bỏ qua không gian phụ, nhưng thường tốt hơn là nhớ một nút thay vì tạo ra nó khi cần thiết. SMA\* có các thuộc tính sau đây:

- Nó sẽ sử dụng tất cả dung lượng bộ nhớ được tạo ra dành cho nó.
- Nó tránh các trạng thái lặp lại chừng nào mà bộ nhớ còn cho phép.
- Nó là hoàn thành nếu bộ nhớ có sẵn là hiệu quả để lưu trữ đường đi giải pháp nông nhất.

Nó là tối ưu nếu có đủ bộ nhớ để cất giữ đường đi giải pháp tối ưu nông nhất. Trái lại, nó trả

 về giải pháp tốt nhất có thể có được trong phạm vi

bộ nhớ cho phép.

Khi có đủ bộ nhớ cho cây tìm kiếm toàn bộ, phép tìm kiếm là hiệu quả một cách tối ưu. Vấn đề không giải quyết được là không rõ SMA\* có phải luôn hiệu quả một cách tối ưu trong các giải pháp cho bởi cùng các thông tin heuristic và cùng dung lượng bộ nhớ. Việc thiết kế SMA\* là đơn giản, ít nhất là về tổng quát. Khi cần phải tạo ra một nút tiếp theo nhưng không còn bộ nhớ, nó sẽ cần phải tạo ra không gian nhớ trong hàng đợi. Để làm điều này, nó bỏ qua một nút trong hàng đợi. Những nút mà bị bỏ rơi trong hàng đợi theo cách này được gọi là những nút bị bỏ quên. Nó ưu tiên bỏ qua những nút mà không có triển vọng – tức là những nút có chi phí  $f$  cao. Để tránh khám phá lại những cây con mà đã bị bỏ rơi khỏi bộ nhớ, nó lưu lại trong các nút tổ tiên những thông tin về chất lượng của đường đi tốt nhất trên cây con bị bỏ qua. Theo cách này, nó chỉ tái sinh ra các cây con khi tất cả các đường đi khác đã được chỉ ra là tồi hơn đường đi mà nó vừa bỏ



qua. Một cách nói khác là nếu tất cả các hậu duệ của một nút  $n$  bị bỏ quên, thì chúng ta sẽ không biết đi   đường nào từ  $n$ , nhưng chúng ta   vẫn có vẫn có ý tưởng về giá trị   của việc đi khỏi  $n$  đến bất cứ chỗ nào.

### ***Các giải thuật cải tiến lặp***

Chúng ta đã thấy trong chương 2 rằng một số bài tập nổi tiếng (ví dụ, bài toán 8 con hậu) có thuộc tính là sự miêu tả trạng thái có chứa tất cả các thông tin cần thiết cho một giải pháp. Đường đi mà dẫn tới đích là không liên quan. Trong những trường hợp như vậy, các giải thuật cải tiến lặp thường cung cấp các giải pháp có tính thực tế nhất. Chẳng hạn, chúng ta bắt đầu với 8 con hậu trên bàn cờ, hoặc tất cả các dây đều đi qua các kênh nào đó. Theo cách này thì, chúng ta có thể di chuyển các con hậu xung quanh để giảm số con hậu bị chiếu; hoặc chúng ta có thể di chuyển một cái dây từ một kênh này đến một kênh khác để giảm sự tắc nghẽn. *ý tưởng chung là bắt đầu với một mô hình đầy đủ và thay đổi mô hình để cải thiện chất lượng của nó.*

36

CuuDuongThanCong.com <https://fb.com/tailieudientucntt>

Cách tốt nhất để hiểu các giải thuật cải tiến lặp là xét tất cả các trạng thái được bày ra trên bề mặt của một phong cảnh. Độ cao của bất cứ điểm nào trên phong cảnh tương ứng với hàm giá của trạng thái ở điểm đó. Ý tưởng cải tiến lặp là di chuyển quanh phong cảnh để cố gắng tìm các đỉnh cao nhất, mà là các giải pháp tối ưu. Các giải thuật cải tiến lặp thường chỉ theo sát trạng thái hiện thời, và không nhìn về phía trước vượt qua những lân cận tức thì của trạng thái đó. Điều này cũng giống với việc cố gắng tìm đỉnh của ngọn núi Everest trong sương mù dày đặc trong khi phải chịu đựng chứng hay quên. Tuy nhiên, đôi khi các giải thuật cải tiến lặp là một phương pháp của sự lựa chọn các bài toán thực tế và học búa.

Các giải thuật cải tiến lặp được chia thành hai lớp chính. Các giải thuật **trèo núi (hay còn gọi “gradient hạ xuống”** nếu chúng ta xem hàm định giá như là chi phí thay vì chất lượng) luôn cố gắng thay đổi để cải tiến trạng thái. Các giải thuật **rèn luyện tái tạo** thỉnh thoảng tạo thay đổi mà làm cho mọi thứ tồi tệ hơn, ít nhất là tạm thời.

### ***Phép tìm kiếm leo núi (Hill-climbing)***

Giải thuật tìm kiếm leo núi được chỉ ra ở hình 2.11. Nó đơn giản là một vòng lặp mà di chuyển liên tục theo hướng làm tăng giá trị. Giải thuật không duy trì một cây tìm kiếm, vì vậy cấu trúc dữ liệu nút chỉ cần ghi lại trạng thái và giá trị của nó, mà chúng ta biểu diễn bằng giá trị. Một khái niệm quan trọng là khi có nhiều hơn một nút kế tiếp tốt nhất để chọn lựa, giải thuật có thể lựa chọn trong số chúng một cách ngẫu nhiên. Qui tắc đơn giản này có ba nhược điểm nổi tiếng như sau:

- **Các giá trị cực đại địa phương:** một giá trị cực đại địa phương, trái ngược với một giá trị cực đại toàn cục, là một đỉnh mà thấp hơn đỉnh cao nhất trong không gian trạng thái.

  Khi ở trên một đại lượng cực đại địa phương, giải thuật sẽ dừng lại thậm chí mặc dù giải pháp vẫn còn lâu mới tối ưu.

- **Các cao nguyên:** một cao nguyên là một khu vực của không gian trạng thái mà hàm định giá là phẳng tuyệt đối. Phép tìm kiếm sẽ thực hiện một bước đi ngẫu nhiên.
- **Các đỉnh chóp:** một đỉnh chóp có thể có các bên sườn vòng và dốc, vì vậy phép tìm kiếm đi đến đỉnh của chóp một cách dễ dàng, nhưng đỉnh có thể dốc rất ít về phía một đỉnh khác.

Trừ phi ở đó có các toán tử mà di chuyển trực tiếp dọc theo đỉnh của hình chóp, phép tìm kiếm có thể dao động từ bên này qua bên kia, khiến cho sự tiến chuyển rất ít.

Trong mỗi trường hợp, giải thuật đi đến một điểm mà tại đó nó không tiến triển gì thêm. Nếu điều này xảy ra, một điều chắc chắn phải làm là bắt đầu lại từ một điểm khởi đầu khác. Phép **leo núi- bắt đầu lại -ngẫu nhiên** làm như sau: nó thực hiện một loạt các phép tìm kiếm leo núi từ các trạng thái ban đầu được tạo ra một cách ngẫu nhiên, thực hiện mỗi phép tìm kiếm cho đến khi nó dừng lại hoặc không có sự tiến triển rõ rệt. Nó lưu lại kết quả tốt nhất tìm được của bất cứ phép tìm kiếm nào. Nó có thể sử dụng một số bước lặp hỗn hợp, hoặc có thể tiếp tục cho đến khi kết quả lưu được tốt nhất chưa được cải thiện đối với một số phép lặp nào đó.

Rõ ràng, nếu cho phép đủ số lần lặp, phép tìm kiếm leo núi bắt đầu lại ngẫu nhiên cuối cùng sẽ tìm ra giải pháp tối ưu. Sự thành công của phép tìm kiếm leo núi phụ thuộc rất nhiều vào hình dạng của "bề mặt" không gian trạng thái: nếu như chỉ có một vài giá trị cực đại địa phương, phép tìm kiếm leo núi bắt đầu lại ngẫu nhiên sẽ rất nhanh chóng tìm thấy một giải pháp tốt. Một bài toán thực sự có một bề mặt mà trông rất giống một con nhím. Nếu bài toán là hoàn thành trong thời gian NP, thì rất có thể chúng ta không thể làm tốt hơn thời gian theo cấp số mũ. Tiếp theo là phải có số các cực đại địa phương theo cấp số mũ mà giải thuật sẽ mắc kẹt vào đó. Tuy nhiên, thông thường, một giải pháp tốt hợp lý có thể được tìm thấy sau một số ít lần lặp.

37

CuuDuongThanCong.com <https://fb.com/tailieudientucntt>

toán

Các biến cục bộ: *current*, một nút

Next, một nút

*Current* ← make-node(initial-state[problem])

**Loop do**

*Next* ← một nút con cháu có giá trị cao nhất

của nút *current* (hiện tại) If value[next] <

value[current] then return *current* *Current* ←

*next*

**End**

Function Hill-climbing(problem) return một

trạng thái giải pháp Inputs : *problem*, một bài **Hình 2.11. Giải thuật tìm kiếm trèo núi.**

### ***Giới hạn của việc tìm kiếm***

Phương pháp dễ hiểu nhất để điều khiển khối lượng việc tìm kiếm là thiết lập một độ sâu giới hạn, để việc kiểm tra ngưỡng (giới hạn) tiến hành đối với tất cả các nút ở độ sâu *d* hay bên dưới độ sâu *d*. Độ sâu được chọn để khối lượng thời gian sử dụng sẽ không vượt quá những gì mà luật chơi cho phép. Khi thời gian hết, chương trình sẽ quay lại bước đi mà được lựa chọn bởi phép tìm kiếm kết thúc sâu nhất.

Những phương pháp này có một số những hậu quả tai hại do tính chất gần đúng của hàm định giá. Rõ ràng, cần có một hàm kiểm tra giới hạn phức tạp. Hàm giá chỉ nên áp dụng cho các vị trí

thụ động, tức là dường như sẽ không phô bày sự “đúng đưa chuyển động dữ dội” (sự biến động lớn) về mặt giá trị trong tương lai gần. Ví dụ trong đánh cờ, các vị trí mà dễ ăn có thể được tạo ra là không thụ động đối với một hàm định giá mà chỉ tính đến vật chất. Các vị trí không thụ động có thể được mở rộng hơn nữa cho đến khi chạm tới các vị trí thụ động. Việc tìm kiếm bổ sung này được gọi là một phương pháp **tìm kiếm thụ động**; đôi khi nó được giới hạn chỉ để xem xét các kiểu đi nào đó, như bước đi ăn (quân đối phương), mà sẽ nhanh chóng giải quyết sự không chắc chắn ở vị trí.

### ***Trình độ phát triển của các chương trình trò chơi***

Việc thiết kế các chương trình trò chơi có hai mục đích: cả hai đều nhằm hiểu rõ hơn

việc làm thế nào để chọn các hành động trong các miền giá trị phức tạp với các kết quả không chắc chắn và để phát triển các hệ thống hiệu suất cao đối với trò chơi được nghiên cứu. **Cờ vua**

Cờ vua thu hút được sự quan tâm lớn nhất trong trò chơi. Mặc dù không đạt tới như lời khẳng định của Simon năm 1957 rằng trong vòng 10 năm nữa, các máy tính sẽ đánh bại bất cứ nhà vô địch thế giới nào, nhưng giờ đây các máy tính gần như đã sắp đạt được mục tiêu đó. Trong môn cờ vua tốc độ, các máy tính đã đánh bại nhà vô địch thế giới, Gary Kasparov, trong các trò chơi 5 phút và 25 phút, nhưng trong các trò chơi đầy đủ, máy tính chỉ xếp trong top 100 tay cờ giỏi nhất thế giới. Hình dưới đây cho thấy tỷ lệ của các nhà vô địch cờ vua là người và máy tính trong những năm qua.

38

CuuDuongThanCong.com <https://fb.com/tailieudientucntt>



Bước nhảy đầu tiên trong việc thực hiện không phải

xuất phát từ những giải thuật tốt hơn hay các hàm định giá mà là từ phần cứng. Belle, máy tính chuyên dùng cho việc chơi cờ vua đầu tiên (Condon và Thomson, 1982) đã sử dụng các mạch tích hợp để cài đặt các định giá vị trí và các thể hệ di chuyển, làm cho nó có thể tìm kiếm khoảng vài triệu vị trí chỉ để đi một nước.

**Hình 2.12 Tỷ lệ của các nhà vô địch cờ vua: con người và máy tính.**

Hệ thống HITECH là một máy tính có mục đích đặc biệt được thiết kế bởi người xứng đáng cựu vô địch thế giới Hans Berliner và sinh viên của ông Carl Ebeling, có thể tính toán nhanh các hàm

định giá rất phức tạp. Tạo ra khoảng 10 triệu trên mỗi nước đi và sử dụng việc định giá chính xác nhất các vị trí đã được phát triển. HITECH đã trở thành vô địch thế giới về máy tính năm 1985 và là chương trình đầu tiên đánh bại thần đồng của nhân loại, Arnold Denker năm 1987. Vào thời điểm đó nó đứng trong 800 người chơi cờ giỏi nhất thế giới.

Hệ thống tốt nhất hiện thời là Deep Thought 2 được sản xuất bởi IBM. Mặc dù Deep Thought 2 sử dụng hàm định giá đơn giản, nó kiểm tra khoảng một nửa tỷ vị trí cho mỗi nước đi, đạt đến độ sâu 10 hoặc 11 (nó đã từng tìm được 37 nước chiếu tướng hết cờ). Tháng 2 năm 1993, Deep Thought 2 thi đấu với đội Olympic của Đan mạch và thắng 3-1, đánh bại một đại kiện tướng và hoà với một đại kiện tướng khác. Hệ số FIDE của nó là 2600, xếp trong số 100 người chơi cờ giỏi nhất thế giới.

### **Cờ đam**

Bắt đầu vào năm 1952, Arthur Samuel của IBM làm việc trong thời gian rỗi của ông, đã xây dựng một chương trình chơi cờ đam (loại cờ gồm 24 quân cờ cho 2 người chơi – ND) mà tự học hàm định giá của nó bằng cách tự chơi với nó hàng nghìn lần.

Chương trình của Samuel bắt đầu như một người mới học việc, nhưng chỉ sau một vài ngày tự chơi với chính nó đã có thể đấu trong những cuộc thi lớn của loài người. Khi một người thấy rằng công cụ tính toán của Samuel (một chiếc máy tính IBM 704) có 10.000 từ trong bộ nhớ chính, bằng từ để lưu trữ dữ liệu và một chu kỳ thời gian khoảng hầu như một miligiây, điều đó cho thấy đây là một trong những thành tích vĩ đại của AI.

Có rất ít những người khác có thể cố gắng làm được tốt hơn cho đến khi Jonathan Schaeffer và các đồng nghiệp viết trình Chinook, mà chạy trên một máy tính thông thường sử dụng phép tìm kiếm alpha-beta, nhưng sử dụng một số kỹ thuật, bao gồm cơ sở dữ liệu giải pháp tuyệt vời cho tất cả các vị trí 6-quân cờ, và gây ra sự tàn phá thế trận dẫn đến chấm dứt ván cờ. Chinook đã chiến thắng trong giải Mỹ mở rộng 1992 và trở thành chương trình đầu tiên mà thử thách một

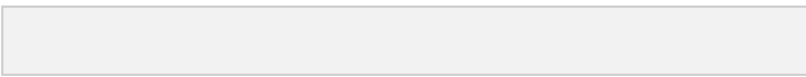

cách chính thức những nhà vô địch thế giới. Sau đó nó gặp phải một vấn đề, dưới cái tên Marion Tinsley. Tiền sỹ Tinsley đã là nhà vô địch thế giới suốt hơn 40 năm, chỉ thua có 3 trận trong suốt khoảng thời gian đó. Trong trận đầu tiên với Chinook, Tinsley đã chịu thua ván thứ 4 và thứ năm của mình, những đã thắng chung cuộc 21.5–18.5. Gần đây, giải vô địch thế giới tháng 8 năm 1994 giữa Tinsley và Chinook kết thúc sớm khi Tinsley phải xin rút lui vì lý do sức khoẻ. Chinook chính thức trở thành nhà vô địch thế giới.

### ***Trò chơi là các bài toán tìm kiếm***


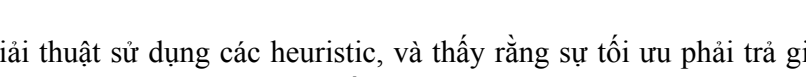
Chơi trò chơi cũng là một trong những khu vực cốt lõi của các nỗ lực trong lĩnh vực trí tuệ nhân tạo. Năm 1950, hầu như ngay khi máy tính trở nên có thể lập trình được, các chương trình chơi cờ được viết bởi Shannon (người phát minh ra lý thuyết thông tin) và bởi Alan Turing. Kể từ đó, đã có những phát triển rất mạnh mẽ về các tiêu chuẩn của việc chơi, đạt tới điểm mà các hệ thống hiện thời có thể thử thách các nhà vô địch của loài người mà không sợ xấu hổ.

Các nhà nghiên cứu đầu tiên đã chọn cờ vì một số lý do. Một máy tính chơi cờ sẽ là một chứng cứ sinh tồn của một máy cơ khí làm một điều gì đó mà cần sự thông minh. Hơn nữa, sự đơn giản của luật chơi, và thực tế rằng trạng thái thế giới có thể nắm bắt được đầy đủ đối với chương trình có nghĩa là rất dễ để biểu diễn trò chơi như là một cuộc tìm kiếm trong một không gian các vị trí trò chơi có thể. Sự biểu diễn trò chơi của máy tính thực ra có thể chỉnh sửa theo bất cứ chi tiết thích

đáng hợp lý nào – không giống như sự miêu tả của bài toán về một cuộc chiến tranh.

Nhưng những gì làm cho trò chơi thực sự khác biệt là chúng thường quá khó để giải quyết. Ví dụ, đánh cờ,  có một hệ số phân nhánh trung bình khoảng 35, và mỗi  bên thường đi khoảng 50 nước trong một ván cờ, dẫn đến cây tìm kiếm có khoảng  $35^{100}$  nút (mặc dù “chỉ có” khoảng  $10^{40}$  vị trí hợp lý khác nhau). Trò chơi cờ ca rô (Tic-Tac-Toe) khá buồn tẻ đối với những người lớn thông minh bởi vì rất dễ để quyết định bước đi đúng. Sự phức tạp của các trò chơi đưa ra một kiểu không chắc chắn hoàn toàn mới mà chúng ta chưa được biết; sự không chắc chắn xuất hiện không phải vì có thông tin bị mất, mà do chúng ta không có đủ thời gian để tính toán một dãy chính xác của bất cứ nước đi nào. Thay vào đó, chúng ta phải dự đoán tốt nhất dựa trên kinh nghiệm của mình, và hành động trước khi chúng ta biết chắc được cần phải hành động như thế nào. Dưới viễn cảnh này, các trò chơi rất giống với thế giới thực hơn so với các bài toán tìm kiếm tiêu chuẩn mà chúng ta đã xét cho tới nay.

## Kết luận

Chương  này áp dụng các heuristic  để làm giảm chi phí tìm kiếm. Chúng ta đã xem xét một số giải thuật sử dụng các heuristic, và thấy rằng sự tối ưu phải trả giá cao dưới dạng chi phí tìm kiếm, thậm chí với các heuristic tốt.

- Phép tìm kiếm best-first là phép tìm kiếm tổng quát khi các nút có chi phí ít nhất (dựa theo một số tính toán) được mở rộng trước tiên.
- Nếu chúng ta tối thiểu hoá chi phí ước tính để đi tới mục tiêu  $h(n)$ , chúng ta có phương pháp tìm kiếm tham lam. Thời gian tìm kiếm thường giảm đi so với một giải thuật không đầy đủ thông tin, nhưng giải thuật này là không tối ưu và không hoàn thành.
- Tối thiểu hoá  $f(n) = g(n) + h(n)$  để kết hợp điểm mạnh của phép tìm kiếm thiếu thông tin và phép tìm kiếm hấu ăn. Nếu chúng ta sử dụng các trạng thái lặp lại và đảm bảo rằng  $h(n)$  không bao giờ ước lượng vượt quá, chúng ta có phép tìm kiếm  $A^*$ .

- $A^*$  là hoàn thành, tối ưu và hiệu quả một cách tốt nhất trong số tất cả các giải thuật tìm kiếm tối ưu. Độ phức tạp không gian của nó vẫn là một trở ngại lớn.
- Độ phức tạp thời gian của các giải thuật heuristic phụ thuộc vào chất lượng của các hàm heuristic. Các heuristic tốt có thể thỉnh thoảng được xây dựng bằng cách kiểm tra sự xác định bài toán hoặc bằng cách tổng quát hoá từ kinh nghiệm với các lớp bài toán.
- Chúng ta có thể giảm yêu cầu đối với dung lượng bộ nhớ cho phép tìm kiếm  $A^*$  với các giải thuật có bộ nhớ giới hạn như  $IDA^*(A^* \text{ lặp sâu dần})$  và  $SMA^*(A^* \text{ có bộ nhớ giới hạn đơn giản hoá})$ .
- Các giải thuật cải tiến lặp chỉ lưu trữ một trạng thái đơn trong bộ nhớ, nhưng có thể bị sa lầy ở những cực đại địa phương. Phép tìm kiếm rèn luyện tái tạo đưa ra một cách để thoát khỏi cực đại địa phương, và là phương pháp hoàn thành, tối ưu khi được cho một lịch trình dài và đủ gọn.

## BÀI TẬP

2.1 Giả sử rằng chúng ta chạy một giải thuật tìm kiếm hấu ăn với  $h(n) = -g(n)$ . Phép tìm kiếm hấu ăn sẽ cạnh tranh với kiểu tìm kiếm nào?

2.2 Hãy phát minh ra một hàm heuristic cho trò chơi số 8 mà thỉnh thoảng ước lượng vượt quá, và chỉ ra làm thế nào nó có thể dẫn đến một giải pháp gần tới ưu đối với một bài toán cụ thể. 2.3.

Hãy chứng minh rằng nếu hàm heuristic  $h$  tuân theo bất đẳng thức tam giác, thì chi phí  $f$  dọc

theo bất cứ đường đi nào trên cây tìm kiếm là không giảm. (Bất đẳng thức tam giác phát biểu rằng tổng số các chi phí từ A đến B và từ B đến C không được nhỏ hơn chi phí đi trực tiếp từ A đến C).

2.4. Phép tìm kiếm A\* tiến lùi có thể là một phương pháp tốt không? Nó có thể áp dụng được trong những điều kiện nào?

2.5. Hãy miêu tả một không gian tìm kiếm trong đó phép tìm kiếm lặp sâu dần thực hiện kém hơn nhiều so với phương pháp tìm kiếm theo chiều sâu.

2.6. Hãy viết giải thuật cho phương pháp tìm kiếm tiến lùi, bằng giả mã hoặc bằng một ngôn ngữ lập trình. Giả sử rằng mỗi phép tìm kiếm sẽ là một phép tìm kiếm theo chiều rộng, và phép tìm kiếm tiến và phép tìm kiếm lùi thay nhau mở rộng một nút ở một thời điểm

## CHƯƠNG 3: BIỂU DIỄN TRI THỨC VÀ SUY DIỄN

*Trong chương này chúng ta sẽ trình bày các đặc trưng của ngôn ngữ biểu diễn tri thức. Chúng ta sẽ nghiên cứu logic mệnh đề, một ngôn ngữ biểu diễn tri thức rất đơn giản, có khả năng biểu diễn hẹp, nhưng thuận lợi cho ta làm quen với nhiều khái niệm quan trọng trong logic, đặc biệt trong logic vị từ cấp một sẽ được nghiên cứu trong các chương sau.*

### 3.1 Nhập môn

### 3.2 Tri thức và dữ liệu



### 3.3 Phân loại tri thức

### 3.4 Bản chất của các tri thức chuyên gia

### 3.5 Các phương pháp biểu diễn tri thức

### 3.6 Cơ chế suy diễn

### 3.7 Các hệ cơ sở tri thức và các hệ chuyên gia

### 3.8 Các ngôn ngữ lập trình thông minh

## 3.1 NHẬP MÔN

Con người sống trong môi trường có thể nhận thức được thế giới nhờ các giác quan (tai, mắt và [redacted] các giác quan khác), sử dụng [redacted] các tri thức tích lũy được và [redacted] nhờ khả năng lập luận, suy diễn, con người có thể đưa ra các hành động hợp lý cho công việc mà con người đang làm. Một mục tiêu của Trí tuệ nhân tạo ứng dụng là thiết kế các **Agent thông minh** (intelligent agent) cũng có khả năng đó như con người. Chúng ta có thể hiểu Agent thông minh là bất cứ cái gì có thể nhận thức được môi trường thông qua các **bộ cảm nhận** (sensors) và đưa ra hành động hợp lý đáp ứng lại môi trường thông qua **bộ phận hành động** (effectors). Các robots, các softbot (software robot), các hệ chuyên gia,... là các ví dụ về Agent thông minh. Các Agent thông minh cần phải có tri thức về thế giới hiện thực mới có thể đưa ra các quyết định đúng đắn.

## 3.2 TRI THỨC VÀ DỮ LIỆU

Thành phần trung tâm của agent **dựa trên tri thức** (knowledge-based agent), còn gọi là **hệ dựa trên tri thức** (knowledge-based system) hoặc đơn giản là hệ tri thức trong đó chứa cơ sở tri thức (Knowledge Base: viết tắt tiếng Anh: KB; viết tắt tiếng Việt: CSTT).

tập hợp [redacted] Cơ sở tri thức là một tập hợp [redacted] các tri thức được biểu diễn dưới [redacted] dạng nào đó. Mỗi khi nhận được các thông tin đưa vào, Agent cần có khả năng suy diễn để đưa ra các câu trả lời, đưa ra các hành động hợp lý. Nhiệm vụ này được thực hiện bởi bộ suy diễn-thành phần cơ bản khác của các hệ tri thức. Như vậy, hệ tri thức bao hàm một CSTT và được trang bị một thủ tục suy diễn. Mỗi khi tiếp nhận các sự kiện từ môi trường, thủ tục suy diễn thực hiện quá trình liên kết các sự kiện với các tri thức trong CSTT để rút ra các câu trả lời, hoặc các hành động hợp lý mà Agent cần thực hiện. Khi thiết kế một Agent giải quyết vấn đề nào đó thì CSTT sẽ chứa các tri thức về đối tượng cụ thể đó. Để máy tính có thể sử dụng, xử lý tri thức, cần biểu diễn tri thức dưới dạng thuận tiện. Đó là mục tiêu của biểu diễn tri thức.

Tri thức là một khái niệm trừu tượng. Chúng ta không cố gắng đưa ra một định nghĩa chính xác ở đây mà muốn so sánh nó với hai khái niệm có liên quan là thông tin và dữ liệu. Karan Sing đã phát biểu: “Chúng ta ngập chìm trong thông **biến thông tin** nhưng lại khát tri thức”.

Trong ngữ cảnh của khoa học máy tính “dữ liệu là nguyên liệu thô để xử lý” là các con số, chữ cái, hình ảnh, âm thanh... Thông tin là tất cả những gì con người có thể cảm nhận qua các giác quan (chính xác, xem khái niệm Entropy là độ đo thông tin, độ đo về các tin tức mới đối với một người nào đó). Nếu so về số lượng: dữ liệu nhiều hơn thông tin; thông tin nhiều hơn tri thức. Chúng ta có thể mô tả chúng theo dạng hình chóp.



### 3.3 PHÂN LOẠI TRI THỨC

Người ta thường phân loại tri thức thành các dạng sau:

#### Tri thức sự kiện

**Định nghĩa:** Tri thức sự kiện là một khẳng định về một sự kiện, hiện tượng hay một khái niệm nào đó trong một hoàn cảnh không gian hoặc thời gian nhất định.

**Ví dụ:** khẳng định về hiện tượng: "Mặt trời lặn ở phương Tây". Khái niệm về: "tam giác đều: là tam giác có ba góc bằng nhau".

#### Tri thức mô tả

**Định nghĩa:** Tri thức sự kiện là một khẳng định về một sự kiện, hiện tượng hay một khái niệm nào đó trong một hoàn cảnh không gian hoặc thời gian nhất định.

**Ví dụ:** khẳng định về hiện tượng: "Mặt trời lặn ở phương Tây". Khái niệm về: "tam giác đều: là tam giác có ba góc bằng nhau".

#### Tri thức thủ tục

**Định nghĩa:** Tri thức thủ tục là tri thức mô tả cách giải quyết một vấn đề, quy trình xử lý các công việc, lịch trình tiến hành các thao tác ... Các dạng của tri thức thủ tục thường dùng là các luật, chiến lược, lịch trình

**Ví dụ:** IF xe máy không khởi động được

THEN đầu tiên kiểm tra bugi

#### Tri thức heuristic

**Định nghĩa:** Tri thức **heuristic** là tri thức nông cạn do không đảm bảo hoàn toàn chính xác hoặc tối ưu theo một nghĩa nào đó về cách giải quyết vấn đề. Tri thức **heuristic** thường được coi là một mẹo nhằm dẫn dắt tiến trình lập luận

**Ví dụ:** một số giải thuật tìm đường đi ngắn nhất, giải thuật  $A^*$  có thể được coi là lời giả của một vấn đề tốt nhưng chưa hẳn tối ưu.

Ngoài ra người ta còn phân chia ra tri thức meta: tri thức tham chiếu đến các tri thức khác;

tri thức có cấu trúc: tri thức về các quan hệ giữa các khái niệm, quan hệ giữa các đối tượng...

### 3.4. BẢN CHẤT CỦA CÁC TRI THỨC CHUYÊN GIA

#### Chuyên gia (Expert).

Nói chung, chuyên gia là người có đầy đủ kỹ năng, kiến thức sâu (cả về luật và các sự kiện) về một lĩnh vực nào đó; người có thể làm những việc mà người khác ít khả năng làm được.

#### Hệ chuyên gia

Hệ chuyên gia (đơn giản) là chương trình máy tính có thể thực hiện các công việc, vấn đề trong thuộc lĩnh vực hẹp ở mức tương tự như một người chuyên gia [19].

Hầu hết các hệ chuyên gia là các hệ dựa luật. Hiện nay một số các hệ chuyên gia thành công trong các lĩnh vực: bán hàng, kỹ nghệ, y học và địa chỉ (tìm kiếm mỏ), các hệ điện lực và khai mỏ. Để hiểu rõ bản chất tri thức của chuyên gia, chúng ta quan sát một hệ chuyên gia gồm các thành phần nào. Nói chung hệ chuyên gia bao gồm các phần cơ bản như sau

### 3.5. CÁC PHƯƠNG PHÁP BIỂU DIỄN TRI THỨC

Trong phần này, chúng ta sẽ tập trung nghiên cứu logic vị từ cấp một (first-order predicate logic hoặc first-order predicate calculus) - một ngôn ngữ biểu diễn tri thức, bởi vì logic vị từ cấp một có khả năng biểu diễn tương đối tốt, và hơn nữa nó là cơ sở cho nhiều ngôn ngữ biểu diễn tri thức khác. Nhưng trước hết chúng ta sẽ nghiên cứu logic mệnh đề (propositional logic hoặc propositional calculus). Nó là ngôn ngữ rất đơn giản, có khả năng biểu diễn hạn chế, song thuận tiện cho ta đưa vào nhiều khái niệm quan trọng trong logic.

#### 3.5.1 Biểu diễn tri thức bằng Logic mệnh đề

**Định nghĩa:** Logic mệnh đề là công cụ toán logic, trong đó các mệnh đề được mã hoá (gán) cho một biến, hoặc hằng; còn vác biểu thức là sự liên kết có nghĩa giữa các biến hằng với một số toán tử nhất định.

**Ví dụ:** Mệnh đề “Nếu trời mưa (A) thì đất ướt (B)” được mô tả:  $A \Rightarrow B$

Tri thức được mô tả dưới dạng các mệnh đề trong **ngôn ngữ biểu diễn tri thức**. Mỗi câu có thể xem như sự mã hóa một sự hiểu biết của ta về thế giới thực. Ngôn ngữ biểu diễn tri thức (cũng như mọi ngôn ngữ hình thức khác) gồm hai thành phần cơ bản là **cú pháp** và **ngữ nghĩa**. • Cú pháp của một ngôn ngữ bao gồm các ký hiệu và các quy tắc liên kết các ký hiệu (các luật cú pháp) để tạo thành các câu (công thức) trong ngôn ngữ. Các câu ở đây là biểu diễn ngoài, cần phân biệt với biểu diễn bên trong máy tính. Các câu sẽ được chuyển thành các cấu trúc dữ liệu thích hợp được cài đặt trong một vùng nhớ nào đó của máy tính, đó là biểu diễn bên trong. Bản thân các câu chưa chứa đựng một nội dung nào cả, chưa mang một ý nghĩa nào cả. • Ngữ nghĩa của ngôn ngữ cho phép ta xác định ý nghĩa của các câu trong một miền nào đó của thế giới hiện thực. Chẳng hạn, trong ngôn ngữ các biểu thức số học, dãy ký hiệu  $(x+y)*z$  là một câu viết đúng cú pháp. Ngữ nghĩa của ngôn ngữ này cho phép ta hiểu rằng, nếu  $x, y, z$ , ứng với các số nguyên, ký hiệu  $+$  ứng với phép toán cộng, còn  $*$  ứng với phép chia, thì biểu thức  $(x+y)*z$  biểu diễn quá trình tính toán: lấy số nguyên  $x$  cộng với số nguyên  $y$ , kết quả được nhân với số nguyên  $z$ .

• Ngoài hai thành phần cú pháp và ngữ nghĩa, ngôn ngữ biểu diễn tri thức cần được cung cấp **cơ chế suy diễn**. Một luật suy diễn (rule of inference) cho phép ta suy ra một công thức từ một tập nào đó các công thức. Chẳng hạn, trong logic mệnh đề, luật modus ponens cho phép từ hai công thức  $A$  và  $A \Rightarrow B$  suy ra công thức  $B$ . Chúng ta sẽ hiểu **lập luận** hoặc **suy diễn** là một quá trình áp dụng các luật suy diễn để từ các tri thức trong cơ sở tri thức và các sự kiện ta nhận được các tri thức mới. Như vậy chúng ta xác định:

**Ngôn ngữ biểu diễn tri thức = Cú pháp + Ngữ nghĩa + Cơ chế suy diễn.**

Một ngôn ngữ biểu diễn tri thức tốt cần có khả năng biểu diễn rộng, tức là mô tả được mọi điều mà chúng ta muốn. Nó cần hiệu quả để đi tới các kết luận; thủ tục suy diễn đòi hỏi ít thời

gian tính toán và không gian nhớ. Người ta mong muốn ngôn ngữ biểu diễn tri thức gần với ngôn ngữ tự nhiên.

### 3.5.1.1. Cú pháp

Cú pháp của logic mệnh đề rất đơn giản. Nó cho phép xây dựng các công thức. Cú pháp của logic mệnh đề gồm tập các *ký hiệu* và *tập các luật xây dựng công thức*.

- **Các ký hiệu**

Hai hằng logic: **True** và **False**.

Các ký hiệu mệnh đề (còn được gọi là các biến mệnh đề):  $P, Q, \dots$

Các phép kết nối logic:  $\wedge, \vee, \neg, \Rightarrow, \Leftrightarrow$ .

Các dấu mở ngoặc (“ và đóng ngoặc ”).

- **Các quy tắc xây dựng các công thức**

Các biến mệnh đề là công thức. Nếu  $A$  và  $B$  là công thức thì:

$(A \wedge B)$  (đọc “ $A$  hội  $B$ ” hoặc “ $A$  và  $B$ ”)

$(A \vee B)$  (đọc “ $A$  tuyển  $B$ ” hoặc “ $A$  hoặc  $B$ ”)

$(\neg A)$  (đọc “phủ định  $A$ ”)

$(A \Rightarrow B)$  (đọc “ $A$  kéo theo  $B$ ” hoặc “nếu  $A$  thì  $B$ ”)

$(A \Leftrightarrow B)$  (đọc “ $A$  và  $B$  kéo theo nhau”)

là các công thức.

Để ngắn gọn, ta bỏ đi các cặp dấu ngoặc khi không cần thiết. Ví dụ, thay cho  $((A \vee B) \wedge C)$ , ta viết  $(A \vee B) \wedge C$ .

Các công thức là các ký hiệu mệnh đề sẽ được gọi là các **câu đơn** hoặc **câu phân tử**. Các công thức không phải là câu đơn sẽ được gọi là câu phức hợp. Nếu  $P$  là ký hiệu mệnh đề thì  $P$  và  $\neg P$  được gọi là **literal**,  $P$  là **literal dương**, còn  $\neg P$  là **literal âm**. Câu phức hợp có dạng  $A_1 \vee \dots \vee A_m$  trong đó  $A_i$  là các literal sẽ được gọi là **câu tuyển** (clause).

### 3.5.1.2 Ngữ nghĩa:

Ngữ nghĩa của logic mệnh đề cho phép ta **xác định** ý nghĩa của các công thức trong thế giới hiện thực nào đó. Điều đó được thực hiện bằng cách kết hợp mỗi ký hiệu mệnh đề với sự kiện nào đó trong thế giới hiện thực.

Chẳng hạn, ký hiệu mệnh đề  $P$  có thể ứng với sự kiện “Paris là thủ đô nước Pháp” hoặc bất kỳ một sự kiện nào khác. Bất kỳ một sự kết hợp các ký hiệu mệnh đề với các sự kiện trong thế giới thực được gọi là một **minh họa** (interpretation). Chẳng hạn minh họa của ký hiệu mệnh đề  $P$  có thể là một sự kiện (mệnh đề) “Paris là thủ đô nước Pháp”. Một sự kiện chỉ có thể đúng hoặc sai. Chẳng hạn, sự kiện “Paris là thủ đô nước Pháp” là đúng, còn sự kiện “Số Pi là số hữu tỉ” là sai.

Một cách chính xác hơn, ta hiểu một minh họa là một cách gán cho mỗi ký hiệu mệnh đề một giá trị chân lý **True** hoặc **False**. Trong một minh họa, nếu ký hiệu mệnh đề  $P$  được gán giá trị chân lý **True/False** ( $P$ : **True**/  $P$ : **False**) thì ta nói mệnh đề  $P$  **đúng/sai** trong minh họa đó. Trong

một minh họa, ý nghĩa của các câu phức hợp được xác định bởi ý nghĩa của các kết nối logic. Chúng ta xác định ý nghĩa của các kết nối logic trong các bảng chân lý (xem hình 3.1)

45

CuuDuongThanCong.com <https://fb.com/tailieudientuontt>

P	Q	$\neg P$	$P \wedge Q$	$P \vee Q$	$P \Rightarrow Q$	$P \Leftrightarrow Q$
False	False	True	False	False	True	True
False	True	True	False	True	True	False
True	False	False	False	True	False	False
True	True	False	True	True	True	True

**Hình 3.1 Bảng chân lý của các kết nối logic**

Ý nghĩa của các kết nối logic  $\wedge$ ,  $\vee$  và  $\neg$  được xác định như ý nghĩa của các từ “và”, “hoặc là” và “phủ định” trong ngôn ngữ tự nhiên. Chúng ta cần giải thích thêm về ý nghĩa của phép kéo theo  $P \Rightarrow Q$  (P kéo theo Q). Ở đây: P là giả thiết, Q là kết luận. Trục quan cho phép ta xem rằng, khi P là đúng và Q là đúng thì câu “P kéo theo Q” là đúng, còn khi P là đúng Q là sai thì câu “P kéo theo Q” là sai. Nhưng nếu P sai và Q đúng, hoặc P sai Q sai thì “P kéo theo Q” là đúng hay sai? Nếu xuất phát từ giả thiết sai, thì không khẳng định gì về kết luận. Không có lý do để nói rằng nếu P sai và Q đúng hoặc P sai và Q sai thì “P kéo theo Q” là sai. Do đó, trong trường hợp P sai thì “P kéo theo Q” là đúng dù Q là đúng hay Q là sai.

Bảng chân lý cho phép ta xác định ngữ nghĩa các câu phức hợp. Chẳng hạn ngữ nghĩa của các câu  $P \wedge Q$  trong minh họa  $\{P \leftarrow \text{True}, Q \sigma \text{False}\}$  là **False**. Việc xác định ngữ nghĩa của một câu  $(P \vee Q) \wedge \neg S$  trong minh họa được tiến hành như sau: đầu tiên ta xác định giá trị chân lý của  $P \vee Q$  và  $\neg S$ , sau đó ta sử dụng bảng chân lý của  $\wedge$  để xác định giá trị  $(P \vee Q) \wedge \neg S$ . Một công thức được gọi là **thoả được** (satisfiable) nếu nó đúng trong một minh họa nào đó. Chẳng hạn công thức  $(P \vee Q) \wedge \neg S$  là thoả được vì nó có giá trị **True** trong minh họa  $\{P \sigma \text{True}, Q \sigma \text{False}, S \sigma \text{True}\}$ .

Một công thức được gọi là **vững chắc** (valid) nếu nó đúng trong mọi minh họa. Chẳng hạn câu  $P \vee \neg P$  là vững chắc (luôn bằng 1: **True**).

Một công thức được gọi là **không thoả được**, nếu nó là sai trong mọi minh họa. Chẳng hạn công thức  $P \wedge \neg P$  (luôn bằng 0: **False**).

Chúng ta sẽ gọi một **mô hình** (model) của một công thức là một minh họa sao cho công thức là đúng trong minh họa này. Như vậy một công thức **thoả được** là công thức có một mô hình. Chẳng hạn, minh họa  $\{P \sigma \text{False}, Q \sigma \text{False}, S \sigma \text{True}\}$  là một mô hình của công thức  $(P \Rightarrow Q) \wedge S$ .

Bảng cách lập bảng chân lý (phương pháp bảng chân lý) ta có thể xác định được một công thức có **thoả được** hay không. Trong bảng này, mỗi biến mệnh đề đứng đầu một

cột, công thức cần kiểm tra đứng đầu một cột, mỗi dòng tương ứng với một minh họa. Chẳng hạn hình 3.2 là bảng chân lý cho công thức  $(P \Rightarrow Q) \wedge S$ . Trong bảng chân lý này ta cần đưa vào các cột phụ

ứng với các công thức con của các công thức cần kiểm tra để việc tính giá trị của công thức này được dễ dàng. Từ bảng chân lý ta thấy rằng công thức  $(P \Rightarrow Q) \wedge S$  là **thoả được** nhưng không **vững chắc**.

46

CuuDuongThanCong.com <https://fb.com/tailieudientucntt>

P	Q	S	$P \Rightarrow Q$	$(P \Rightarrow Q) \wedge S$
False	False	False	True	False
False	False	True	True	True
False	True	False	True	False
False	True	True	True	True
True	False	False	False	False
True	False	True	False	False
True	True	False	True	False
True	True	True	True	True

**Hình 3.2 Bảng chân lý cho công thức  $(P \Rightarrow Q) \wedge S$**

Cần lưu ý rằng, một công thức chứa  $n$  biến, thì số các minh họa của nó là  $2^n$ , tức là bảng chân lý có  $2^n$  dòng. Như vậy việc kiểm tra một công thức có **thoả được** hay không bằng phương pháp bảng chân lý, đòi hỏi thời gian mũ. Cook (1971) đã chứng minh rằng, vấn đề kiểm tra một công thức trong logic mệnh đề có **thoả được** hay không là vấn đề NP-đầy đủ.

Chúng ta sẽ nói rằng một tập công thức  $G = \{G_1, \dots, G_m\}$  là **vững chắc (thoả được, không thoả được)** nếu hội của chúng  $G_1 \wedge \dots \wedge G_m$  là **vững chắc (thoả được, không thoả được)**. Một

mô hình của tập công thức  $G$  là mô hình của công thức  $G_1 \wedge \dots \wedge G_m$ .

### 3.5.2 Dạng chuẩn tắc

Trong mục này chúng ta sẽ xét việc chuẩn hóa các công thức, đưa các công thức về dạng thuận lợi cho việc lập luận, suy diễn. Trước hết ta sẽ xét các phép biến đổi tương đương. Sử dụng các phép biến đổi này, ta có thể đưa một công thức bất kỳ về dạng chuẩn tắc.

#### 3.5.2.1 Sự tương đương của các công thức

Hai công thức  $A$  và  $B$  được xem là **tương đương** nếu chúng có cùng một giá trị chân lý trong mọi minh họa. Để chỉ  $A$  tương đương với  $B$  ta viết  $A \equiv B$ . Bằng phương pháp bảng chân lý, dễ dàng

chứng minh được sự tương đương của các công thức sau đây:

$$A \Rightarrow B \equiv \neg A \vee B$$

$$A \Leftrightarrow B \equiv (A \Rightarrow B) \wedge (B \Rightarrow A)$$

$$\neg(\neg A) \equiv A$$

• **Luật De Morgan**

$$\neg(A \vee B) \equiv \neg A \wedge \neg B$$

$$\neg(A \wedge B) \equiv \neg A \vee \neg B$$

• **Luật giao hoán**

$$A \vee B \equiv B \vee A$$

$$A \wedge B \equiv B \wedge A$$

• **Luật kết hợp**

47

CuuDuongThanCong.com <https://fb.com/tailieudientucntt>

$$(A \vee B) \vee C \equiv A \vee (B \vee C)$$

$$(A \wedge B) \wedge C \equiv A \wedge (B \wedge C)$$

• **Luật phân phối**

$$A \wedge (B \vee C) \equiv (A \wedge B) \vee (A \wedge C)$$

$$A \vee (B \wedge C) \equiv (A \vee B) \wedge (A \vee C)$$

### 3.5.2.2 Dạng chuẩn tắc

Các công thức tương đương có thể xem như các biểu diễn khác nhau của cùng một sự kiện. Để dễ dàng viết các chương trình máy tính thao tác trên các công thức, chúng ta sẽ chuẩn hóa các công thức, đưa chúng về dạng biểu diễn chuẩn được gọi là **dạng chuẩn hội**. Một công thức ở dạng chuẩn hội nếu nó là hội của các câu tuyển. Nhớ lại rằng, câu tuyển có dạng  $A_1 \vee \dots \vee A_m$  trong đó các  $A_i$  là **literal**. Chúng ta có thể biến đổi một công thức bất kỳ về công thức ở dạng chuẩn hội bằng cách áp dụng thủ tục sau.

- Bỏ các dấu kéo theo ( $\Rightarrow$ ) bằng cách thay  $(A \Rightarrow B)$  bởi  $(\neg A \vee B)$ .
- Chuyển các dấu phủ định ( $\neg$ ) vào sát các ký hiệu mệnh đề bằng cách áp dụng luật De Morgan và thay  $\neg(\neg A)$  bởi  $A$ .
- Áp dụng luật phân phối, thay các công thức có dạng  $A \vee (B \wedge C)$  bởi  $(A \vee B) \wedge (A \vee C)$ .

Ví dụ: Ta chuẩn hóa công thức  $(P \Rightarrow Q) \vee \neg(R \vee \neg S)$ :

$$\begin{aligned} & (P \Rightarrow Q) \vee \neg(R \vee \neg S) \equiv (\neg P \vee Q) \vee (\neg R \wedge S) \\ & \equiv ((\neg P \vee Q) \vee \neg R) \wedge ((\neg P \vee Q) \vee S) \\ & \equiv (\neg P \vee Q \vee \neg R) \wedge (\neg P \vee Q \vee S). \end{aligned}$$

Như vậy công thức  $(P \Rightarrow Q) \vee \neg(R \vee \neg S)$  được đưa về dạng chuẩn hội  $(\neg P \vee Q \vee \neg R) \wedge (\neg P \vee Q \vee S)$ .

Khi biểu diễn tri thức bởi các công thức trong logic mệnh đề, cơ sở tri thức là một tập nào đó các công thức. Bằng cách chuẩn hoá các công thức, cơ sở tri thức là một tập nào đó các câu tuyển.

### 3.5.3. Các câu Horn:

Ở trên ta đã chỉ ra, mọi công thức đều có thể đưa về dạng chuẩn hội, tức là hội của các tuyển, mỗi câu tuyển có dạng:

$$IP_1 \vee \dots \vee IP_m \vee Q_1 \vee \dots \vee Q_n$$

trong đó

các ký

mệnh đề

$P_i, Q_i$  là

hiệu

(literal

dương) câu này tương đương với câu:  $P_1 \wedge \dots \wedge IP_m \Rightarrow Q_1 \vee \dots \vee Q_n$

Dạng câu này được gọi là **câu Kowalski** (do nhà logic Kowalski đưa ra năm 1971). Khi  $n \leq 1$ , tức là câu tuyển chỉ chứa nhiều nhất một literal dương, ta có một dạng câu đặc biệt quan trọng được gọi là **câu Horn** (mang tên nhà logic Alfred Horn, năm 1951). Nếu  $m > 0, n = 1$ , câu Horn có dạng:

$$P_1 \wedge \dots \wedge P_m \Rightarrow Q$$

Trong đó  $P_i, Q$  là các literal dương. Các  $P_i$  được gọi là các điều kiện (hoặc giả thiết), còn  $Q$  được gọi là kết luận (hoặc hệ quả). Các câu Horn dạng này còn được gọi là các luật **if-then** và được biểu diễn như sau:

48

CuuDuongThanCong.com <https://fb.com/tailieudientucntt>

**If  $P_1$  and....and  $P_m$  then  $Q$ .**

Khi  $m=0, n=1$  câu Horn trở thành câu đơn  $Q$ , hay sự kiện  $Q$ . Nếu  $m > 0, n=0$  câu Horn trở thành dạng  $IP_1 \vee \dots \vee IP_m$  hay tương đương  $l(P_1 \wedge \dots \wedge P_m)$ .

Cần chú ý rằng, không phải mọi công thức đều có thể biểu diễn dưới dạng hội của các câu Horn. Tuy nhiên trong các ứng dụng, cơ sở tri thức thường là một tập nào đó các câu Horn (tức là một tập nào đó các luật if-then).

### 3.5.4. Luật suy diễn

Một công thức  $H$  được xem là **hệ quả logic** (logical consequence) của một tập công thức  $G = \{G_1, \dots, G_m\}$  nếu trong bất kỳ minh họa nào mà  $\{G_1, \dots, G_m\}$  đúng thì  $H$  cũng đúng. Nói cách khác bất kỳ mô hình nào của  $G$  cũng là mô hình của  $H$ .

Khi có một cơ sở tri thức, ta muốn sử dụng các tri thức trong cơ sở này để suy ra tri thức mới mà nó là hệ quả logic của các công thức trong cơ sở tri thức. Điều đó được thực hiện bằng cách sử dụng **các luật suy diễn** (rule of inference). Luật suy diễn giống như một thủ tục mà chúng ta sử dụng để sinh ra một công thức mới từ các công thức đã có. Một luật suy diễn gồm hai phần: một tập các điều kiện và một kết luận. Chúng ta sẽ biểu diễn các luật suy diễn dưới dạng “phân số”, trong đó tử số là danh sách các điều kiện, còn mẫu số là kết luận của luật, tức là mẫu số là công thức mới được suy ra từ các công thức ở tử số.

Sau đây là một số luật suy diễn quan trọng trong logic mệnh đề. Trong các luật này  $\alpha, \alpha_i, \beta, \gamma$  là các công thức:

$$\frac{\alpha \quad \alpha \Rightarrow \beta}{\beta}$$

**Luật Modus Ponens**

$$\alpha \Rightarrow \beta, \alpha$$

$$\beta$$



Từ một kéo theo và giả thiết của kéo theo, ta suy ra kết luận của nó.

#### Luật Modus Tollens

$$\alpha \Rightarrow \beta, \neg \beta$$

$$\neg \alpha$$

Từ một kéo theo và phủ định kết luận của nó, ta suy ra phủ định giả thiết của kéo theo.

#### Luật bắc cầu

$$\alpha \Rightarrow \beta, \beta \Rightarrow \gamma$$

$$\alpha \Rightarrow \gamma$$

Từ hai kéo theo, mà kết luận của kéo theo thứ nhất trùng với giả thiết của kéo theo thứ hai, ta suy ra kéo theo mới mà giả thiết của nó là giả thiết của kéo theo thứ nhất, còn kết luận của nó là kết luận của kéo theo thứ hai.

#### Luật loại bỏ hội

$$\alpha_1 \wedge \dots \wedge \alpha_i \wedge \dots \wedge \alpha_m$$

$$\alpha_i$$

Từ một hội ta suy ra một nhân tử bất kỳ của hội.

#### Luật đưa vào hội

49

CuuDuongThanCong.com <https://fb.com/tailieudientucntt>

$$\alpha_1, \dots, \alpha_i, \dots, \alpha_m$$

$$\alpha_1 \wedge \dots \wedge \alpha_i \wedge \dots \wedge \alpha_m$$

Từ một danh sách các công thức, ta suy ra hội của chúng.

#### Luật đưa vào tuyển

$$\alpha_i$$

$$\alpha_1 \vee \dots \vee \alpha_i \vee \dots \vee \alpha_m$$

Từ một công thức, ta suy ra một tuyển mà một trong các hạng tử của tuyển là công thức đó.

#### Luật phân giải

$$\alpha \vee \beta, \neg \beta \vee \gamma$$

$$\alpha \vee \gamma$$

Từ hai tuyển, một tuyển chứa một hạng tử đối lập với một hạng tử trong tuyển kia, ta suy ra tuyển của các hạng tử còn lại trong cả hai tuyển.

Một luật suy diễn được xem là **tin cậy** (sound) nếu bất kỳ một mô hình nào của giả thiết của luật cũng là mô hình của kết luận của luật. Chúng ta chỉ quan tâm đến các luật suy diễn tin cậy. Bằng phương pháp bảng chân lý, ta có thể kiểm chứng được các luật suy diễn nêu trên đều là tin cậy. Bảng chân lý của luật phân giải được cho trong hình 3.3. Từ bảng này ta thấy rằng, trong bất kỳ một minh họa nào mà cả hai giả thiết  $\alpha \vee \beta, \neg \beta \vee \gamma$  đúng thì kết luận  $\alpha \vee \gamma$  cũng đúng. Do đó

luật phân giải là luật  t suy diễn tin

cây.

$\alpha$	$\beta$	$\gamma$	$\alpha \vee \beta$	$\neg \beta \vee \gamma$	$\alpha \vee \gamma$
False	False	False	False	True	False
False	False	True	False	True	True
False	True	False	True	False	False
False	True	True	True	True	True
True	False	False	True	True	True
True	False	True	True	True	True
True	True	False	True	False	True
True	True	True	True	True	True

**Hình 3.3 Bảng chân lý chứng minh tính tin cậy của luật phân giải.**

Ta có nhận xét rằng, luật phân giải là một luật suy diễn tổng quát, nó bao gồm luật Modus Ponens, luật Modus Tollens, luật bắc cầu như các trường hợp riêng. (Bạn đọc dễ dàng chứng minh được điều đó).

### ***Tiên đề, định lý, chứng minh.***

Giả sử chúng ta có một tập nào đó các công thức. Các luật suy diễn cho phép ta từ các công thức đã có suy ra công thức mới bằng một dãy áp dụng các luật suy diễn. Các công thức đã

50

CuuDuongThanCong.com <https://fb.com/tailieudientucntt>

cho được gọi là các **tiên đề**. Các công thức được suy ra được gọi là các **định lý**. Dãy các luật được áp dụng để dẫn tới định lý được gọi là một **chứng minh** của định lý. Nếu các luật suy diễn là tin cậy, thì các định lý là hệ quả logic của các tiên đề.

Ví dụ: Giả sử ta có các công thức sau:

$$Q \wedge S \Rightarrow G \vee H \quad (1)$$

$$P \Rightarrow Q \quad (2)$$

$$R \Rightarrow S \quad (3)$$

$$P \quad (4)$$

$$R \quad (5)$$

Giả sử ta cần chứng minh công thức  $G \vee H$ . Từ công thức (2) và (4), ta suy ra  $Q$  (Luật Modus Ponens). Lại áp dụng luật Modus Ponens, từ (3) và (5) ta suy ra  $S$ . Từ  $Q, S$  ta suy ra  $Q \wedge S$  (luật đưa vào hội). Từ (1) và  $Q \wedge S$  ta suy ra  $G \vee H$ . Công thức  $G \vee H$  đã được chứng minh.

Trong các hệ tri thức, chẳng hạn các hệ chuyên gia, hệ lập trình logic,..., sử dụng các luật suy diễn người ta thiết kế lên các **thủ tục suy diễn** (còn được gọi là **thủ tục chứng minh**) để từ các tri thức

trong cơ sở tri thức ta suy ra các tri thức mới đáp ứng nhu cầu của người sử dụng.

Một **hệ hình thức** (formal system) bao gồm một tập các tiên đề và một tập các luật suy diễn nào đó (trong ngôn ngữ biểu diễn tri thức nào đó).

Một tập luật suy diễn được xem là **đầy đủ**, nếu mọi hệ quả logic của một tập các tiên đề đều chứng minh được bằng cách chỉ sử dụng các luật của tập đó.

### **Phương pháp chứng minh bác bỏ**

Phương pháp chứng minh bác bỏ (refutation proof hoặc proof by contradiction) là một phương pháp thường xuyên được sử dụng trong các chứng minh toán học. Tư tưởng của phương pháp này là như sau: Để chứng minh P đúng, ta giả sử P sai (thêm  $\neg P$  vào các giả thiết) và dẫn tới một mâu thuẫn. Sau đây ta sẽ trình bày cơ sở của phương pháp chứng minh này.

Giả sử chúng ta có một tập các công thức  $G = \{G_1, \dots, G_m\}$  ta cần chứng minh công thức H là hệ quả logic của G. Điều đó tương đương với chứng minh công thức  $G_1 \wedge \dots \wedge G_m \Rightarrow H$  là vững chắc. Thay cho chứng minh  $G_1 \wedge \dots \wedge G_m \Rightarrow H$  là vững chắc, ta chứng minh  $G_1 \wedge \dots \wedge G_m \wedge \neg H$  là không thỏa mãn được. Tức là ta chứng minh tập  $G' = (G_1, \dots, G_m, \neg H)$  là không thỏa được. G sẽ không thỏa được nếu từ G' ta suy ra hai mệnh đề đối lập nhau. Việc chứng minh công thức H là hệ quả logic của tập các tiên đề G bằng cách chứng minh tính không thỏa được của tập các tiên đề được thêm vào phủ định của công thức cần chứng minh, được gọi là chứng minh bác bỏ. **3.5.5.**

### **Luật phân giải, chứng minh bác bỏ bằng luật phân giải**

Để thuận tiện cho việc sử dụng luật phân giải, chúng ta sẽ cụ thể hoá luật phân giải trên các dạng câu đặc biệt quan trọng.

Luật phân giải trên các câu tuyển

$$A_1 \vee \dots \vee A_m \vee C$$

$$\neg C \vee B_1 \vee \dots \vee B_n$$

$$A_1 \vee \dots \vee A_m \vee B_1 \vee \dots \vee B_n$$

trong đó  $A_i, B_j$  và C là các literal.

Luật phân giải trên các câu Horn:

51

CuuDuongThanCong.com <https://fb.com/tailieudientucntt>

Giả sử  $P_i, R_j, Q$  và S là các literal. Khi đó ta có các luật sau:

$$P_1 \wedge \dots \wedge P_m \wedge S \Rightarrow Q,$$

$$R_1 \wedge \dots \wedge R_n \Rightarrow S$$

$$P_1 \wedge \dots \wedge P_m \wedge R_1 \wedge \dots \wedge R_n \Rightarrow Q$$

Một trường hợp riêng hay được sử dụng của luật trên là:

$$P_1 \wedge \dots \wedge P_m \wedge S \Rightarrow Q,$$

$$S$$

$$P_1 \wedge \dots \wedge P_m \Rightarrow Q$$

Khi ta có thể áp dụng luật phân giải cho hai câu, thì hai câu này được gọi là **hai câu phân giải được** và kết quả nhận được khi áp dụng luật phân giải cho hai câu đó được gọi là **phân giải thức** của chúng. Phân giải thức của hai câu A và B được kí hiệu là  $\text{res}(A, B)$ . Chẳng hạn, hai câu tuyển phân giải được nếu một câu chứa một literal đối lập với một literal trong câu kia. Phân giải

thức của hai literal đối lập nhau ( $P$  và  $\neg P$ ) là câu rỗng, chúng ta sẽ ký hiệu câu rỗng là  $[]$ , câu rỗng không thỏa được.

Giả sử  $G$  là một tập các câu tuyển (bằng cách chuẩn hoá ta có thể đưa một tập các công thức về một tập các câu tuyển). Ta sẽ ký hiệu  $R(G)$  là tập câu bao gồm các câu thuộc  $G$  và tất cả các câu nhận được từ  $G$  bằng một dãy áp dụng luật phân giải.

Luật phân giải là luật đầy đủ để chứng minh một tập câu là không thỏa được. Điều này được suy từ định lý sau:

**Định lý**

**phân giải:**

thỏa được nếu và chỉ nếu câu rỗng  $[] \in R(G)$ .

Một tập câu tuyển là không

*procedure Resolution;*

*Input: tập  $G$  các câu tuyển ;*

*begin*

*1.Repeat*

*1.1 Chọn hai câu  $A$  và  $B$  thuộc  $G$ ;*

*1.2 if  $A$  và  $B$  phân giải được then tính  $Res(A,B)$ ;*

*1.3 if  $Res(A,B)$  là câu mới then thêm  $Res(A,B)$  vào  $G$ ;*

*until nhận được  $[]$  hoặc không có câu mới xuất hiện;*

*2. if nhận được câu rỗng then thông báo  $G$  không thỏa được*

*else thông báo  $G$  thỏa được;*

*end*

Định lý phân giải có nghĩa rằng, nếu từ các câu thuộc  $G$ , bằng cách áp dụng luật phân giải ta dẫn tới câu rỗng thì  $G$  là không thỏa được, còn nếu không thể sinh ra câu rỗng bằng luật phân giải thì  $G$  thỏa được. Lưu ý rằng, việc dẫn tới câu rỗng có nghĩa là ta đã dẫn tới hai literal đối lập nhau  $P$  và  $\neg P$  (tức là dẫn tới mâu thuẫn).

Từ định lý phân giải, ta đưa ra thủ tục sau đây để xác định một tập câu tuyển  $G$  là thỏa được hay không. Thủ tục này được gọi là thủ tục phân giải.

Dễ thấy, nếu  $G$  là tập các câu hữu hạn thì các literal có mặt trong các câu của  $G$  là hữu hạn. Do đó, số các câu tuyển thành lập được từ các literal đó là hữu hạn. Vì vậy, chỉ có một số

52

CuuDuongThanCong.com <https://fb.com/tailieudientucntt>

hữu hạn câu được sinh ra bằng luật phân giải. Thủ tục phân giải sẽ dừng lại sau một số hữu hạn bước.

Chỉ sử dụng luật phân giải ta không thể suy ra mọi công thức là hệ quả logic của một tập công thức đã cho. Tuy nhiên, sử dụng luật phân giải ta có thể chứng minh được một công thức bất kì có là hệ quả của một tập công thức đã cho hay không bằng phương pháp chứng minh bác bỏ. Vì vậy luật phân giải được xem là **luật đầy đủ cho bác bỏ**. Thủ tục chứng minh bác bỏ bằng luật phân giải xem [9, 17]

**Ví dụ:** Giả sử  $G$  là tập hợp các câu tuyển sau

$\neg A \vee \neg B \vee P$  (1)

$\neg C \vee \neg D \vee P$  (2)

$\neg E \vee C$  (3)

A (4)

E (5)

D (6)

Giả sử ta cần chứng minh P. Thêm vào G câu sau:

$\neg P$  (7)

áp dụng luật phân giải cho câu (2) và (7) ta được câu:

$\neg C \vee \neg D$  (8)

Từ câu (6) và (8) ta nhận được câu:

$\neg C$  (9)

Từ câu (3) và (9) ta nhận được câu:

$\neg E$  (10)

Tới đây đã xuất hiện mâu thuẫn, vì câu (5) và (10) đối lập nhau. Từ câu (5) và (10) ta nhận được câu rỗng [9].

Vậy P là hệ quả logic của các câu (1) --(6).

Thông thường chúng ta có thể bằng chân lý để chứng minh tính đúng đắn của một biểu thức. Nhưng phương pháp đó tỏ ra cồng kềnh và có tính “thủ công”. Thay vào đó, chúng ta có thể sử dụng hai thuật toán sau đây để chứng minh biểu thức là đúng hoặc sai

### Thuật toán Havard (1970)

**Bước 1:** Phát biểu lại giả thiết (GT) và kết luận của bài toán dưới dạng chuẩn sau:

$GT_1, GT_2, \dots, GT_n$   
 $\rightarrow KL_1, KL_2, \dots,$   
 $KL_m$

$KL_m$

Trong đó các  $GT_i$ ,  $KL_j$  được xây dựng từ các biến mệnh đề và các phép nối  $\wedge, \vee, \neg$ , **Bước 2:** Bỏ phủ định (nếu cần). Khi cần bỏ các phủ định: chuyển về  $GT_i$  sang về kết luận  $KL_j$  và ngược lại (giống như chuyển dấu âm trong đại số từ về phải sang trái và ngược lại) **Bước 3: Thay dấu “ $\wedge$ ” ở  $GT_i$  và “ $\vee$ ” ở  $KL_j$  bằng các dấu “,”**

**Bước 4:** Nếu  $GT_i$  còn dấu “ $\vee$ ” và  $KL_j$  còn dấu “ $\wedge$ ” thì tách chúng thành hai dòng con **Bước 5:**

Một dòng được chứng minh nếu tồn tại chung một mệnh đề ở cả hai vế **Bước 6:** Bài toán được chứng minh khi và chỉ khi tất cả các dòng được chứng minh. Ngược lại thì bài toán không được chứng minh.

### Thuật toán Robin son (1971)

Robison đã cải tiến thuật toán Havard. Cách thức chứng minh như sau:

**Bước 1:** Phát biểu lại giả thiết (GT) và kết luận của bài toán dưới dạng chuẩn sau:

$GT_1, GT_2, \dots, GT_n \rightarrow KL_1, KL_2, \dots, KL_m$

Trong đó các  $GT_i$ ,  $KL_j$  được xây dựng từ các biến mệnh đề và các phép nối  $\wedge, \vee, \neg$ ,

**Bước 2: Thay dấu “ $\wedge$ ” ở  $GT_i$  và “ $\vee$ ” ở  $KL_j$  bằng các dấu “,”**

**Bước 3:** Chuyển về  $KL_j$  sang về  $GT_i$  với dấu phủ định để còn một vế, tức là :

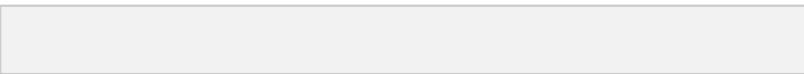
$GT_1, GT_2, \dots, GT_n, ], KL_1, ], KL_2, \dots, ]KL_m$

**Bước 4:** Xây dựng một mệnh đề mới bằng cách tuyển một cặp mệnh đề từ danh sách các mệnh đề. Nếu mệnh đề mới có các biến mệnh đề đối ngẫu thì mệnh đề đó được loại bỏ. **Bước 5: Bổ sung mệnh đề mới này vào danh sách và lặp lại bước 4**

**Bước 6:** Bài toán được chứng minh khi và chỉ khi chỉ còn hai mệnh đề đối ngẫu. Ngược lại thì bài toán không được chứng minh.

Thuật toán này thực chất là chứng minh bằng phản chứng

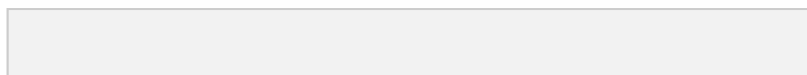
### 3.5.6 Biểu diễn tri thức bằng Logic vị từ

Logic mệnh đề cho phép ta biểu diễn các sự kiện. Mỗi kí hiệu trong logic mệnh đề được minh họa như là  một sự kiện trong thế giới hiện thực, sử dụng các kết nối logic ta có thể tạo ra các câu phức hợp biểu diễn các sự kiện mang ý nghĩa phức tạp hơn. Như vậy, khả năng biểu diễn của logic mệnh đề chỉ giới hạn trong phạm vi thế giới các sự kiện.

Thế giới hiện thực bao gồm các **đối tượng**. Mỗi đối tượng có những **tính chất** riêng để phân biệt nó với các đối tượng khác. Các đối tượng lại có **quan hệ** với nhau. Các mối quan hệ rất đa dạng và phong phú. Chúng ta có thể liệt kê rất nhiều ví dụ về đối tượng, tính chất, quan hệ. Đối tượng: một cái bàn, một cái nhà, một cái cây, một con người, một con số.... Tính chất: Cái bàn có thể có tính chất: có bốn chân, làm bằng gỗ, không có ngăn kéo. Con số có thể có tính chất là số nguyên, số hữu tỉ, là số chính phương...

Quan hệ: cha con, anh em, bè bạn (giữa con người); lớn hơn, nhỏ hơn, bằng nhau (giữa các con số); bên trong, bên ngoài nằm trên nằm dưới (giữa các đồ vật)...

Hàm: Một trường hợp riêng của quan hệ là quan hệ hàm. Chẳng hạn, vì mỗi người có một mẹ, do đó ta có quan hệ hàm ứng mỗi người với mẹ của nó.



Mục này dành cho nghiên cứu logic vị từ cấp một với tư cách là một ngôn ngữ biểu diễn tri thức. Logic vị từ cấp một đóng vai trò quan trọng trong biểu diễn tri thức vì khả năng biểu diễn của nó (nó cho phép ta biểu diễn tri thức về thế giới với các đối tượng, các thuộc tính của đối tượng và các quan hệ của đối tượng), hơn nữa, nó là cơ sở cho nhiều ngôn ngữ logic khác.

#### 3.5.6.1 Cú pháp và ngữ nghĩa của logic vị từ cấp 1

Logic vị từ cấp một là mở rộng của logic mệnh đề. Nó cho phép ta mô tả thế giới với các đối tượng, các thuộc tính của đối tượng và các mối quan hệ giữa các đối tượng. Nó sử dụng các biến (biến đối tượng) để chỉ các đối tượng trong một miền đối tượng nào đó. Để mô tả các thuộc tính của đối tượng, các quan hệ giữa các đối tượng, trong logic vị từ, người ta đưa vào các **vị từ**

(predicate). Ngoài các kết nối logic như trong logic mệnh đề, logic vị từ cấp một còn sử dụng các **lượng tử**. Chẳng hạn, lượng tử  $\forall$  (với mọi) cho phép ta tạo ra các câu nói tới mọi đối tượng trong một miền đối tượng nào đó.

##### 3.5.6.1.1 Cú pháp.

## Các ký hiệu.

Logic vị từ cấp một sử dụng các loại ký hiệu sau đây.

Các ký hiệu hằng:  $a, b, c, An, Ba, John, \dots$

Các ký hiệu biến:  $x, y, z, u, v, w, \dots$

Các ký hiệu vị từ:  $P, Q, R, S, Like, Havecolor, Prime, \dots$

Mỗi vị từ là vị từ của  $n$  biến ( $n \geq 0$ ). Chẳng hạn  $Like$  là vị từ của hai biến,  $Prime$  là vị từ một biến.

Các ký hiệu vị từ không biến là các ký hiệu mệnh đề.

Các ký hiệu hàm:  $f, g, \cos, \sin, mother, husband, distance, \dots$

Mỗi hàm là hàm của  $n$  biến ( $n \geq 1$ ). Chẳng hạn,  $\cos, \sin$  là hàm một biến,  $distance$  là hàm của ba biến.

Các ký hiệu kết nối logic:  $\wedge$  (hội),  $\vee$  (tuyển),  $\neg$  (phủ định),  $\Rightarrow$  (kéo theo),  $\Leftrightarrow$  (kéo theo nhau). Các ký hiệu lượng từ:  $\forall$  (với mọi),  $\exists$  (tồn tại).

Các ký hiệu ngăn cách: dấu phẩy, dấu mở ngoặc và dấu đóng ngoặc.

## Các hạng thức

Các hạng thức (term) là các biểu thức mô tả các đối tượng. Các hạng thức được xác định đệ quy như sau.

Các ký hiệu hằng và các ký hiệu biến là hạng thức.

Nếu  $t_1, t_2, t_3, \dots, t_n$  là  $n$  hạng thức và  $f$  là một ký hiệu hàm  $n$  biến thì  $f(t_1, t_2, \dots, t_n)$  là hạng thức. Một hạng thức không chứa biến được gọi là một **hạng thức cụ thể** (ground term).

Chẳng hạn,  $An$  là ký hiệu hằng,  $mother$  là ký hiệu hàm một biến, thì  $mother(An)$  là một hạng thức cụ thể.

## Các công thức phân tử

Chúng ta sẽ biểu diễn các tính chất của đối tượng, hoặc các quan hệ giữa các đối tượng bởi các **công thức phân tử (câu đơn)**.

Các công thức phân tử (câu đơn) được xác định đệ quy như sau.

Các ký hiệu vị từ không biến (các ký hiệu mệnh đề) là công thức phân tử.

Nếu  $t_1, t_2, \dots, t_n$  là  $n$  hạng thức và  $P$  là vị từ của  $n$  biến thì  $P(t_1, t_2, \dots, t_n)$  là công thức phân tử. Chẳng hạn,  $Hoa$  là một ký hiệu hằng,  $Love$  là một vị từ của hai biến,  $husband$  là hàm của một biến, thì  $Love(Hoa, husband(Hoa))$  là một công thức phân tử.

## Các công thức

Từ công thức phân tử, sử dụng các kết nối logic và các lượng từ, ta xây dựng nên các công thức (các câu).

Các công thức được xác định đệ quy như sau:

Các công thức phân tử là công thức.



công thức.

Nếu  $G$  là một công thức và  $x$  là biến thì các biểu thức  $(\forall x G)$ ,  $(\exists x G)$  là công thức.

Các công thức không phải là công thức phân tử sẽ được gọi là các câu phức hợp. Các công thức không chứa biến sẽ được gọi là **công thức cụ thể**. Khi viết các công thức ta sẽ bỏ đi các dấu ngoặc không cần thiết, chẳng hạn các dấu ngoặc ngoài cùng.

Lượng tử phổ dụng cho phép mô tả tính chất của cả một lớp các đối tượng, chứ không phải của một đối tượng, mà không cần phải liệt kê ra tất cả các đối tượng trong lớp. Chẳng hạn sử dụng vị từ  $\text{Elephant}(x)$  (đối tượng  $x$  là con voi) và vị từ  $\text{Color}(x, \text{Gray})$  (đối tượng  $x$  có màu xám) thì câu “tất cả các con voi đều có màu xám” có thể biểu diễn bởi công thức  $\forall x (\text{Elephant}(x) \Rightarrow \text{Color}(x, \text{Gray}))$ .

Lượng tử tồn tại cho phép ta tạo ra các câu nói đến một đối tượng nào đó trong một lớp đối tượng mà nó có một tính chất hoặc thoả mãn một quan hệ nào đó. Chẳng hạn bằng cách sử dụng các câu đơn  $\text{Student}(x)$  ( $x$  là sinh viên) và  $\text{Inside}(x, \text{P301})$ , ( $x$  ở trong phòng 301), ta có thể biểu diễn câu “Có một sinh viên ở phòng 301” bởi biểu thức  $\exists x (\text{Student}(x) \wedge \text{Inside}(x, \text{P301}))$ .

Một công thức là công thức phân tử hoặc phủ định của công thức phân tử được gọi là **literal**.

Chẳng hạn,  $\text{Play}(x, \text{Football})$ ,  $\neg \text{Like}(\text{Lan}, \text{Rose})$  là các literal. Một công thức là tuyển của các

literal sẽ được gọi là **câu tuyển**. Chẳng hạn,  $\text{Male}(x) \vee \neg \text{Like}(x, \text{Football})$  là câu tuyển. Trong

công thức  $\forall x G$ , hoặc  $\exists x G$

trong đó  $G$  là một công thức

nào đó, thì mỗi xuất hiện của biến  $x$  trong công thức  $G$  được gọi là **xuất hiện buộc**. Một công thức mà tất cả các biến đều là xuất hiện buộc thì được gọi là **công thức đóng**.

Ví dụ: Công thức  $\forall x P(x, f(a, x)) \wedge \exists y Q(y)$  là công thức đóng, còn công thức  $\forall x P(x, f(y, x))$  không phải là công thức đóng, vì sự xuất hiện của biến  $y$  trong công thức này không chịu ràng buộc bởi một lượng tử nào cả (Sự xuất hiện của  $y$  gọi là **sự xuất hiện tự do**). Sau này chúng ta chỉ quan tâm tới các công thức đóng.

#### 3.5.6.1.2. Ngữ nghĩa.

Cũng như trong logic mệnh đề, nói đến ngữ nghĩa là chúng ta nói đến ý nghĩa của các công thức trong một thế giới hiện thực nào đó mà chúng ta sẽ gọi là **một minh họa**.

Để xác định một minh họa, trước hết ta cần xác định một miền đối tượng (nó bao gồm tất cả các đối tượng trong thế giới hiện thực mà ta quan tâm).

Trong một minh họa, các ký hiệu

các đối tượng cụ thể trong miền đối tượng, các ký hiệu hàm sẽ được gắn với một hàm cụ thể nào đó. Khi đó, mỗi hạng thức cụ thể sẽ chỉ định một đối tượng cụ thể trong miền đối tượng. Chẳng hạn, nếu  $An$  là một ký hiệu hằng,  $\text{Father}$  là một ký hiệu hàm, nếu trong minh họa  $An$  ứng với một người cụ thể nào đó, còn  $\text{Father}(x)$  gắn với hàm: ứng với mỗi  $x$  là cha của nó, thì hạng thức  $\text{Father}(An)$  sẽ chỉ người cha của  $An$ .

#### Ngữ nghĩa của các câu đơn.

Trong một minh họa, các ký hiệu vị từ sẽ được gắn với một thuộc tính, hoặc một quan hệ cụ thể nào đó. Khi đó mỗi công thức phân tử (không chứa biến) sẽ chỉ định một sự kiện cụ thể. Đương nhiên sự kiện này có thể là đúng (True) hoặc sai (False). Chẳng hạn, nếu trong minh họa, ký hiệu hằng  $Lan$  ứng với một cô gái cụ thể nào đó, còn  $\text{Student}(x)$  ứng với thuộc tính “ $x$  là sinh viên” thì

câu Student (Lan) có giá trị chân lý là True hoặc False tùy thuộc trong thực tế Lan có phải là sinh viên hay không.

### Ngữ nghĩa của các câu phức hợp.

Khi đã xác định được ngữ nghĩa của các câu đơn, ta có thể xác định được ngữ nghĩa của các câu phức hợp (được tạo thành từ các câu đơn bằng các liên kết các câu đơn bởi các kết nối logic) như trong logic mệnh đề.

Ví dụ: Câu  $\text{Student}(\text{Lan}) \wedge \text{Student}(\text{An})$  nhận giá trị True nếu cả hai câu  $\text{Student}(\text{Lan})$  và  $\text{Student}(\text{An})$  đều có giá trị True, tức là cả Lan và An đều là sinh viên.

Câu  $\text{Like}(\text{Lan}, \text{Rose}) \vee \text{Like}(\text{An}, \text{Tulip})$  là đúng nếu câu  $\text{Like}(\text{Lan}, \text{Rose})$  là đúng hoặc câu  $\text{Like}(\text{An}, \text{Tulip})$  là đúng.

### Ngữ nghĩa của các câu chứa các lượng tử

Ngữ nghĩa của các câu  $\forall x G$ , trong đó  $G$  là một công thức nào đó, được xác định như là ngữ nghĩa của công thức là hội của tất cả các công thức nhận được từ công thức  $G$  bằng cách thay  $x$  bởi một đối tượng trong miền đối tượng. Chẳng hạn, nếu miền đối tượng gồm ba người  $\{\text{Lan}, \text{An}, \text{Hoa}\}$  thì ngữ nghĩa của câu  $\forall x \text{Student}(x)$  được xác định là ngữ nghĩa của câu  $\text{Student}(\text{Lan}) \wedge \text{Student}(\text{An}) \wedge \text{Student}(\text{Hoa})$ . Câu này đúng khi và chỉ khi cả ba câu thành phần đều đúng, tức là cả Lan, An, Hoa đều là sinh viên.

Như vậy, công thức  $\forall x G$  là đúng nếu và chỉ nếu mọi công thức nhận được từ  $G$  bằng cách thay  $x$  bởi một đối tượng trong miền đối tượng đều đúng, tức là  $G$  đúng cho tất cả các đối tượng  $x$  trong miền đối tượng.

Ngữ nghĩa của công thức  $\exists x G$  được xác định như là ngữ nghĩa của công thức là tuyển của tất cả các công thức nhận được từ  $G$  bằng cách thay  $x$  bởi một đối tượng trong miền đối tượng. Chẳng hạn, nếu ngữ nghĩa của câu  $\text{Younger}(x, 20)$  là “ $x$  trẻ hơn 20 tuổi” và miền đối tượng gồm ba người  $\{\text{Lan}, \text{An}, \text{Hoa}\}$  thì ngữ nghĩa của câu  $\exists x \text{Younger}(x, 20)$  là ngữ nghĩa của câu  $\text{Younger}(\text{Lan}, 20) \vee \text{Younger}(\text{An}, 20) \vee \text{Younger}(\text{Hoa}, 20)$ . Câu này nhận giá trị True nếu và chỉ nếu ít nhất một trong ba người Lan, An, Hoa trẻ hơn 20 tuổi.

Như vậy công thức  $\exists x G$  là đúng nếu và chỉ nếu một trong các công thức nhận được từ  $G$  bằng cách thay  $x$  bằng một đối tượng trong miền đối tượng là đúng.

Bằng các phương pháp đã trình bày ở trên, ta có thể xác định được giá trị chân lý (True, False)

của một công thức bất kỳ trong một minh họa. (Lưu ý rằng, ta chỉ quan tâm tới các công thức đóng).

Sau khi đã xác định khái niệm minh họa và giá trị chân lý của một công thức trong một minh họa, chúng ta có thể đưa ra các khái niệm **công thức vững chắc (thỏa được, không thỏa được)**, **mô hình** của công thức giống như trong logic mệnh đề.

### Các công thức tương đương

Cũng như trong logic mệnh đề, ta nói hai công thức  $G$  và  $H$  tương đương (viết là  $G \equiv H$ ) nếu chúng cùng đúng hoặc cùng sai trong mọi minh họa. Ngoài các tương đương đã biết trong logic mệnh đề, trong logic vị từ cấp một còn có các tương đương khác liên quan tới các lượng tử. Giả sử  $G$  là một công thức, cách viết  $G(x)$  nói rằng công thức  $G$  có chứa các xuất hiện của biến  $x$ . Khi đó công thức  $G(y)$  là công thức nhận được từ  $G(x)$  bằng cách thay tất cả các xuất hiện của  $x$

57

CuuDuongThanCong.com <https://fb.com/tailieudientucntt>

bởi  $y$ . Ta nói  $G(y)$  là công thức nhận được từ  $G(x)$  bằng cách *đặt tên lại* (biến  $x$  được đổi tên lại là  $y$ ).

Các công thức tương đương:

$$1. \forall x G(x) \equiv \forall y G(y)$$

$$\exists x G(x) \equiv \exists y G(y)$$

Đặt tên lại biến đi sau lượng tử tồn tại, nhận được công thức tương đương.

$$2. \neg (\forall x G(x)) \equiv \exists x (\neg G(x))$$

$$\neg (\exists x G(x)) \equiv \forall x (\neg G(x))$$

$$3. \forall x (G(x) \wedge H(x)) \equiv \forall x G(x) \wedge \forall x H(x)$$

$$\exists x (G(x) \vee H(x)) \equiv \exists x G(x) \vee \exists x H(x)$$

**Ví dụ:**  $\forall x \text{ Love}(x, \text{Husband}(x)) \equiv \forall y \text{ Love}(y, \text{Husband}(y))$ .

### 3.5.6.2. Chuẩn hóa và công thức

Từ các câu phân tử, bằng cách sử dụng các kết nối logic và các lượng tử, ta có thể tạo ra các câu phức hợp có cấu trúc rất phức tạp. Để dễ dàng cho việc lưu trữ các câu trong bộ nhớ, và thuận lợi cho việc xây dựng các thủ tục suy diễn, chúng ta cần chuẩn hoá các câu bằng cách đưa chúng về dạng **chuẩn tắc hội** (hội của các câu tuyến).

ta sẽ chuyển

Trong mục này chúng trình bày thủ tục một câu phức hợp

thành một câu ở dạng chuẩn tắc hội tương đương.

Thủ tục chuẩn hoá các công thức gồm các bước sau:

#### • Loại bỏ các kéo theo

Để loại bỏ các kéo theo, ta chỉ cần thay công thức  $P \Rightarrow Q$  bởi công thức tương đương  $\neg P \vee Q$  thay  $P \Leftrightarrow Q$  bởi  $(\neg P \vee Q) \wedge (\neg Q \vee P)$

#### • Chuyển các phủ định tới các phân tử

Điều này được thực hiện bằng cách thay công thức ở vế trái bởi công thức ở vế phải trong các tương đương sau

$$\neg (\neg P) \equiv P$$

$$\neg (P \wedge Q) \equiv \neg P \vee \neg Q$$

$$\neg (P \vee Q) \equiv \neg P \wedge \neg Q$$

$$\neg (\forall x P) \equiv \exists x (\neg P)$$

$$\neg (\exists x P) \equiv \forall x (\neg P)$$

### • Loại bỏ các lượng tử tồn tại

Giả sử  $P(x,y)$  là các vị từ có nghĩa: “ $y$  lớn hơn  $x$ ” trong miền các số. Khi đó, công thức  $\forall x (\exists y (P(x,y)))$  có nghĩa là “với mọi số  $x$ , tồn tại  $y$  sao cho số  $y$  lớn hơn  $x$ ”. Ta có thể xem  $y$  trong công thức đó là hàm của đối số  $x$ . Chẳng hạn, loại bỏ lượng tử  $\exists y$ , công thức đang xét trở thành  $\forall x (P(x, f(x)))$ .

58

CuuDuongThanCong.com <https://fb.com/tailieudientucntt>

Một cách tổng quát, giả sử  $\exists y (G)$  là một công thức con của công thức đang xét và nằm trong miền tác dụng của các lượng tử  $\forall x_1, \dots, \forall x_n$ . Khi đó, có thể xem  $y$  là hàm của  $n$  biến  $x_1, \dots, x_n$  của ví dụ  $f(x_1 \dots x_n)$ . Sau đó, thay các xuất hiện của  $y$  trong công thức  $G$  bởi hạng thức  $f(x_1 \dots x_n)$  và loại bỏ các lượng tử tồn tại. Hàm  $f$  được đưa vào để loại bỏ các lượng tử tồn tại được gọi là **hàm Skolem**.

**Ví dụ:** xét công thức sau:

$$\forall x (\exists y (P(x,y) \vee \forall u (\exists v (Q(a, v) \wedge \exists y \neg R(x,y)))) (1)$$

Công thức con  $\exists y P(x,y)$  nằm trong miền tác dụng của lượng tử  $\forall x$ , ta xem  $y$  là hàm của  $x$ :  $f(x)$ . Các công thức con  $\exists v (Q(a, v))$  và  $\exists y \neg R(x,y)$  nằm trong miền tác dụng của các lượng tử  $\forall x, \forall u$  ta xem  $v$  là hàm  $g(x,u)$  và  $y$  là hàm  $h(x,u)$  của hai biến  $x, u$ . Thay các xuất hiện của  $y$  và  $v$  bởi các hàm tương ứng, sau đó loại bỏ các lượng tử tồn tại, từ công thức (1) ta nhận được công thức:

$$\forall x (P(x, f(x)) \vee \forall u (Q(a, g(x,u)) \wedge \neg R(x, h(x,u)))) (2)$$

### • Loại bỏ các lượng tử phổ dụng

Sau bước 3 trong công thức chỉ còn lại các lượng tử phổ dụng và mọi xuất hiện của các biến đều nằm trong miền tác dụng của các lượng tử phổ dụng. Ta có thể loại bỏ tất cả các lượng tử phổ dụng, công thức (2) trở thành công thức:

$$P(x, f(x)) \vee (Q(a, g(x,u)) \wedge \neg R(x, h(x,u))) (3)$$

Cần chú ý rằng, sau khi được thực hiện bước này tất cả các biến trong công thức được xem là chịu tác

dụng của các lượng tử phổ dụng.

### • Chuyển các tuyển tới các literal

Bước này được thực hiện bằng cách thay các công thức dạng:  $P \vee (Q \wedge R)$  bởi  $(P \vee Q) \wedge (P \vee R)$  và thay  $(P \wedge Q) \vee R$  bởi  $(P \vee Q) \wedge (P \vee R)$ . Sau bước này công thức trở thành hội của các câu tuyển nghĩa là ta nhận được các công thức ở dạng chuẩn tắc hội.

Chẳng hạn, câu (3) được chuyển thành công thức sau

$$(P(x, f(x)) \vee (Q(a, g(x,u)) \wedge \neg R(x, h(x,u)))) (4)$$

### • Loại bỏ các hội

Một câu hội là đúng nếu và chỉ nếu tất cả các thành phần của nó đều đúng. Do đó công thức ở dạng chuẩn tắc hội tương đương với tập các thành phần.

Chẳng hạn, câu (4) tương đương với

tập hai câu tuyển sau

$$P(f(x)) \vee (Q(a, g(x, u)))$$

$$P(f(x)) \vee \neg R(x, h(x, u)) \quad (5)$$

#### • Đặt tên lại các biến

Đặt tên lại các biến sao cho các biến trong các câu khác nhau có tên khác nhau, chẳng hạn, hai câu (5) có hai biến cùng tên là  $x$ , ta cần đổi tên biến  $x$  trong câu hai thành  $z$ , khi đó các câu (5) tương đương với các câu sau

$$P(f(x)) \vee (Q(a, g(x, u)))$$

$$P(f(x)) \vee \neg R(z, h(z, u)) \quad (5')$$

59

CuuDuongThanCong.com <https://fb.com/tailieudientucntt>

Như vậy, khi tri thức là một tập hợp nào đó các công thức trong logic vị từ, bằng cách áp dụng thủ tục trên ta nhận được cơ sở tri thức chỉ gồm các câu tuyển (tức là luôn có thể xem mỗi câu trong cơ sở tri thức là tuyển của các literal). Tương tự như logic mệnh đề, mỗi câu tuyển có thể biểu diễn dưới dạng một kéo theo; vế trái của các kéo theo là hội của các câu phân tử; vế phải là tuyển của các câu phân tử. Dạng câu này được gọi là câu Kowalski. Một trường hợp quan trọng của câu Kowalski là câu Horn (luật *if-then*).

#### 3.5.6.3 Các luật suy diễn

Trong các phần trước chúng ta đã đưa ra các luật suy diễn quan trọng trong logic mệnh đề: luật Modus Ponens, luật Modus Tolens, luật bắc cầu,... luật phân giải. Chúng ta đã chỉ ra rằng, luật phân giải là luật đầy đủ cho bác bỏ. Điều đó có nghĩa là, bằng phương pháp chứng minh bác bỏ, chỉ sử dụng luật phân giải ta có thể chứng minh được một công thức có là hệ quả logic của một tập các công thức cho trước hay không. Kết quả quan trọng này sẽ được mở rộng sang logic vị từ.

Tất cả các luật suy diễn đã được đưa ra trong logic mệnh đề đều đúng trong logic vị từ cấp một.

Bây giờ ta đưa ra một luật suy diễn quan trọng trong logic vị từ liên quan tới lượng từ phổ dụng •

#### Luật thay thế phổ dụng:

Giả sử  $G$  là một câu, câu  $\forall x G$  là đúng trong một minh hoạ nào đó nếu và chỉ nếu  $G$  đúng đối với tất cả các đối tượng nằm trong miền đối tượng của minh hoạ đó. Mỗi hạng thức  $t$  ứng với một đối tượng vì thế nếu câu  $\forall x G$  đúng thì khi thay tất cả các xuất hiện của biến  $x$  bởi hạng thức  $t$  ta nhận được câu đúng. Công thức nhận được từ công thức  $G$  bằng cách thay tất cả các xuất hiện của  $x$  bởi  $t$  được kí hiệu là  $G[x/t]$ . Luật thay thế phổ dụng (*universal instantiation*) phát biểu rằng, từ công thức

$\forall x G$

$\forall x G$

$G[x/t]$

Chẳng hạn, từ câu  $\forall x \text{Like}(x, \text{Football})$  (mọi người đều thích bóng đá), bằng cách thay  $x$  bởi  $An$  ta suy ra câu  $\text{Like}(An, \text{Football})$  (An thích bóng đá)

#### • Hợp nhất

Trong luật thay thế phổ dụng, ta cần sử dụng phép thế các biến bởi các hạng thức để nhận được các công thức mới từ công thức chứa các lượng từ phổ dụng. Ta có thể sử dụng phép thế để hợp nhất các câu phân tử (tức là để các câu trở thành đồng nhất). Chẳng hạn xét hai câu phân tử

Like(An, y), Like(x, Football). Cần lưu ý rằng hai câu này là hai câu  $\forall y$  Like(An,y) và  $\forall x$  Like(x,Football) mà để cho đơn giản ta bỏ đi các lượng tử phổ. Sử dụng phép thế  $[x/An, y/Football]$  hai câu trên trở thành đồng nhất Like(An,Football). Trong các suy diễn, ta cần sử dụng phép hợp nhất các câu bởi các phép thế. Chẳng hạn, cho trước hai câu

Friend(x,Ba)  $\Rightarrow$  Good(x) (Mọi bạn của Ba đều là người tốt)

Friend(Lan,y) (Lan là bạn của tất cả mọi người)

Ta có thể hợp nhất hai câu Friend(x,Ba)  $\Rightarrow$  Good(x) và Friend(Lan,y) bởi phép thay thế  $[x/Lan, y/Ba]$ . áp dụng luật thay thế phổ dụng với phép thay thế này ta nhận được hai câu:

Friend(Lan,Ba)  $\Rightarrow$  Good(Lan)

Friend(Lan,Ba)

Từ hai câu này, theo luật Modus Ponens, ta suy ra câu Good(Lan) (Lan là người tốt).

60

CuuDuongThanCong.com <https://fb.com/tailieudientucntt>

Một cách tổng quát, một phép thế  $\theta$  là một dãy các cặp  $x_i/t_i$ ,  $\theta = [x_1/t_1 \ x_2/t_2 \dots \ x_n/t_n]$  trong đó các  $x_i$  là các biến khác nhau, các  $t_i$  là các hạng thức và các  $x_i$  không có mặt trong  $t_i$  ( $i=1, \dots, n$ ). Áp dụng phép thế  $\theta$  vào công thức G, ta nhận được công thức  $G_\theta$ , đó là công thức nhận được từ công thức G bằng cách thay mỗi sự xuất hiện của các  $x_i$  bởi  $t_i$ . Chẳng hạn, nếu  $G = P(x,y,f(a,x))$  và  $\theta = [x/b, y/g(z)]$  thì  $G_\theta = P(b,g(z),f(a,b))$ .

Với hai câu phân tử G và H mà tồn tại phép thế  $\theta$  sao cho  $G_\theta$  và  $H_\theta$  trở thành đồng nhất ( $G_\theta = H_\theta$ ) thì G và H được gọi là **hợp nhất được**, phép thế  $\theta$  được gọi là **hợp nhất tử** của G và H. Chẳng hạn, hai câu Like(An,y) và Like(x,Football) là hợp nhất được bởi hợp nhất tử  $[x/An, y/Football]$ . Vấn đề đặt ra là, với hai câu phân tử bất kì G và H, chúng có hợp nhất được không và nếu có thì làm thế nào tìm được hợp nhất tử? Vấn đề này sẽ được nghiên cứu trong mục sau. Còn bây giờ chúng ta đưa ra các luật suy diễn quan trọng nhất, trong đó có sử dụng phép hợp nhất.

#### • Luật Modus Ponens tổng quát.

Giả sử  $P_i, P_i'$  ( $i=1, \dots, n$ ) và Q là các công thức phân tử sao cho tất cả các cặp câu  $P_i, P_i'$  hợp nhất được bởi phép thế  $\theta$ , tức là  $P_{i\theta} = P_{i'\theta}$  ( $i=1, \dots, n$ ). Khi đó ta có luật:

$$(P_1 \wedge \dots \wedge P_n \Rightarrow Q), P_1', \dots, P_n' \\ Q'$$

Trong đó  $Q' = Q_\theta$ .

Ví dụ: Giả sử ta có các câu (Student (x)  $\wedge$  Male (x)  $\Rightarrow$  Like (x,Football)) và Student(Anh), Male(Anh). Với phép thế  $\theta = [x/Anh]$ , các cặp câu Student(x), Student(Anh) và Male(x), Male(Anh) hợp nhất được. Do đó ta suy ra câu Like(Anh,Football).

#### Luật phân giải tổng quát

##### • Luật phân giải trên các câu tuyển

Giả sử ta có hai câu tuyển  $A_1 \vee \dots \vee A_m \vee C$  và  $B_1 \vee \dots \vee B_n \vee D$ , trong đó  $A_i$  ( $i=1, \dots, m$ ) và  $B_j$  ( $j=1, \dots, n$ ) là các literal, còn  $C$  và  $D$  là các câu phân tử có thể hợp nhất được bởi phép thế  $\theta$ ,  $C\theta = D\theta$ . Khi đó ta có luật:

$$A_1 \vee \dots \vee A_m \vee C, B_1 \vee \dots \vee B_n \vee D$$

$$A_1' \vee \dots \vee A_m' \vee B_1' \vee \dots \vee B_n'$$

Trong đó  $A_i' = A_i\theta$  ( $i=1, \dots, m$ ) và

$$B_j' = B_j\theta \quad (j=1, \dots, n)$$

Trong luật phân giải này (và trong các luật phân giải sẽ trình bày sau này), hai câu ở tử số (giả thiết) của luật được gọi là hai câu **phân giải được**, còn câu ở mẫu số (kết luận) của luật được gọi là **phân giải thức** của hai câu ở tử số. Ta sẽ ký hiệu phân giải thức của hai câu  $A$  và  $B$  là  $\text{Res}(A, B)$ .

Ví dụ: Giả sử ta có hai câu  $A = \text{Hear}(x, \text{Music}) \vee \text{Play}(x, \text{Tennis})$  và  $B = \neg \text{Play}(An, y) \vee \text{Study}(An)$ . Hai câu  $\text{Play}(x, \text{Tennis})$  và  $\text{Play}(An, y)$  hợp nhất được bởi phép thế  $\theta = [x|An, y|\text{Tennis}]$ . Do đó từ hai câu đã cho, ta suy ra câu  $\text{Hear}(An, \text{Music}) \vee \text{Study}(An)$ . Trong ví dụ này, hai câu

61

CuuDuongThanCong.com <https://fb.com/tailieudientucntt>

$A = \text{Hear}(x, \text{Music}) \vee \text{Play}(x, \text{Tennis})$  và  $B = \neg \text{Play}(An, y) \vee \text{Study}(An)$  là phân giải được và phân giải thức của chúng là  $\text{Hear}(An, \text{Music}) \vee \text{Study}(An)$ .

#### • Luật phân giải trên các câu Horn:

Câu Horn (luật If-Then) là các câu có dạng

$$P_1 \wedge \dots \wedge P_m \Rightarrow Q$$

trong đó  $P_i$  ( $i=1, \dots, m$ ;  $m \geq 0$ ) và  $Q$  là các câu phân tử.

Giả sử ta có hai câu Horn  $P_1 \wedge \dots \wedge P_m \wedge S \Rightarrow Q$  và  $R_1 \wedge \dots \wedge R_n \Rightarrow T$ , trong đó hai câu  $S$  và  $T$  hợp nhất được bởi phép thế  $\theta$ ,  $S\theta = T\theta$ . Khi đó ta có luật:

$$P_1 \wedge \dots \wedge P_m \wedge S \Rightarrow Q,$$

$$R_1 \wedge \dots \wedge R_n \Rightarrow T$$

$$P_1' \wedge \dots \wedge P_m' \wedge R_1' \wedge \dots \wedge R_n \Rightarrow Q$$

trong đó  $P_i' = P_i\theta$  ( $i=1, \dots, m$ ),  $R_j' = R_j\theta$  ( $j=1, \dots, n$ ),  $Q' = Q\theta$ .

Trong thực tế, chúng ta thường sử dụng trường hợp riêng sau đây. Giả sử  $S$  và  $T$  là hai câu phân tử, hợp nhất được bởi phép thế  $\theta$ . Khi đó ta có luật:

$$P_1 \wedge \dots \wedge P_m \wedge S \Rightarrow Q,$$

$$T$$

$$P_1' \wedge \dots \wedge P_m' \Rightarrow Q'$$

trong đó  $P_i' = P_i\theta$  ( $i=1, \dots, m$ ) và  $Q' = Q\theta$ .



Ví dụ: Xét hai câu  $\text{Student}(x) \wedge \text{Male}(x) \Rightarrow \text{Play}(x, \text{Football})$  và  $\text{Male}(\text{Ba})$ . Hai câu  $\text{Male}(\text{Ba})$  và  $\text{Male}(x)$  hợp nhất được với phép thế  $[x|\text{Ba}]$ , do đó từ hai câu trên ta suy ra  $\text{Student}(\text{Ba}) \Rightarrow \text{Play}(\text{Ba}, \text{Football})$ .

#### 3.5.6.4. Sử dụng logic vị từ cấp 1 để biểu diễn tri thức

Logic vị từ cấp một cho phép chúng ta biểu diễn các đối tượng trong thế giới hiện thực với các tính chất của chúng và các mối quan hệ giữa chúng. Để biểu diễn tri thức của chúng ta về một miền đối tượng nào đó trong logic vị từ cấp một, trước hết chúng ta cần đưa ra các kí hiệu: các kí hiệu

để chỉ ra  hằng (hằng đối tượng)  
các kí  các đối tượng cụ thể;  
đôi tượng bất kỳ trong miền đối tượng; các kí hiệu hàm để biểu diễn các quan hệ hàm; các kí hiệu vị từ để

biểu diễn các mối quan hệ khác nhau giữa các đối tượng. Các kí hiệu được đưa ra đó tạo thành **hệ thống từ vựng** về miền đối tượng mà chúng ta đang quan tâm. Sử dụng các từ vựng đã đưa ra, chúng ta sẽ tạo ra các câu trong logic vị từ cấp một để biểu diễn tri thức của chúng ta về miền đối tượng đó. Tập hợp tất cả các câu được tạo thành sẽ lập nên cơ sở tri thức trong hệ tri thức mà chúng ta mong muốn xây dựng. Vấn đề xây dựng cơ sở tri thức sẽ được đề cập đến trong mục sau. Sử dụng các kí hiệu hằng, các kí hiệu biến và các kí hiệu hàm, chúng ta sẽ tạo ra các hạng thức (term) để biểu diễn các đối tượng. Tuy nhiên trong rất nhiều vấn đề, để thuận lợi cho biểu diễn, chúng ta cần đến một dạng hạng thức đặc biệt, đó là danh sách. Danh sách là một cấu trúc dữ liệu được sử dụng thường xuyên nhất trong các ngôn ngữ Prolog và Lisp. Trong mục này chúng ta sẽ

trình bày sự tạo thành các hạng thức danh sách và các phép toán trên danh sách. Song trước hết chúng ta cần xác định vị từ bằng, một vị từ thường xuyên được sử dụng.

##### 3.5.6.4.1. Vị từ bằng

Vị từ bằng, kí hiệu truyền thống là  $=$ , biểu diễn mối quan hệ đồng nhất. Giả sử  $T_1$  và  $T_2$  là hai hạng thức bất kỳ, khi đó công thức phân tử  $T_1 = T_2$  là đúng trong một minh hoạ nếu trong minh hoạ đó  $T_1$  và  $T_2$  ứng với cùng một đối tượng. Chẳng hạn, câu:  $\text{Father}(\text{An}) = \text{Ba}$  là đúng trong một minh hoạ, nếu người ứng với  $\text{Father}(\text{An})$  và người ứng với  $\text{Ba}$  là một.

Sau đây là một ví dụ đơn giản về sử dụng vị từ bằng nhau. Giả sử vị từ  $\text{Sister}(x, y)$  có nghĩa là “ $x$  là chị gái của  $y$ ”, khi đó câu “ $\text{Tam}$  có ít nhất hai chị gái” có thể biểu diễn bởi công thức:  $\exists x, y$

$$(\text{Sister}(x, \text{Tam}) \wedge \text{Sister}(y, \text{Tam}) \wedge \neg (x=y))$$

Sau này chúng ta thường viết  $T_1 \neq T_2$  thay cho  $\neg (T_1 = T_2)$ .

##### 3.5.6.4.2. Danh sách và các phép toán trên danh sách

Để mô tả vấn đề, trong rất nhiều trường hợp chúng ta cần sử dụng các danh sách. Danh sách là cấu trúc dữ liệu được sử dụng rộng rãi nhất trong các ngôn ngữ xử lý các thông tin không phải là số.

Danh sách là một dãy gồm  $n$  ( $n \geq 0$ ) đối tượng bất kỳ, ở đây đối tượng có thể là đối tượng đơn hoặc đối tượng có cấu trúc, và do đó danh sách cũng là một dạng đối tượng. Chúng ta sẽ biểu diễn danh sách bởi cặp dấu ngoặc vuông, bên trong liệt kê các thành phần của danh sách, các thành phần ngăn cách nhau bởi dấu phẩy.

Sau đây là một số ví dụ về danh sách:

[ ] (danh sách rỗng)

[ spring, summer, autumn, winter ]

[ john, data(23, may, 1964), 8354268 ]

[ a, [a,c], b, [a,d,e] ]

Một danh sách không phải là danh sách rỗng có thể phân tách làm hai phần: thành phần đầu tiên của danh sách được gọi là **đầu danh sách**, phần còn lại của danh sách được gọi là **đuôi danh sách**. Chẳng hạn trong danh sách:

[ blue, red, white, yellow ]

đầu của danh sách là blue, và đuôi của danh sách là danh sách [red, white, yellow].

Chúng ta có thể biểu diễn danh sách bởi cách viết:

	[ đầu_danh_sách   đuôi_danh_sách ]
--	------------------------------------

Chẳng hạn:

[ blue, red, white, yellow ] = [ blue | [ red, white, yellow ] ]

Chúng ta cũng có thể biểu diễn danh sách bằng cách liệt kê ra một số thành phần ở đầu danh sách, theo sau là dấu gạch đứng “|”, rồi đến danh sách các thành phần còn lại. Chẳng hạn, sau đây là một số cách viết danh sách trên:

[ blue, red, white, yellow ]

= [ blue | [ red, white, yellow ] ]

= [ blue, red | [ white, yellow ] ]

= [ blue, red, white | [ yellow ] ]

= [ blue, red, white, yellow | [ ] ]

Chúng ta có thể biểu diễn danh sách bởi các hạng thức trong logic vị từ cấp một. Trong logic vị từ, một danh sách được định nghĩa như sau:

Danh sách hoặc là kí hiệu hằng [ ], hoặc là một hạng thức có dạng list(X,Y), trong đó list là kí hiệu hàm của hai biến, đối số X là một hạng thức bất kì và được gọi là đầu danh sách, đối số Y là một danh sách và được gọi là đuôi danh sách. (Trong ngôn ngữ Prolog, người ta sử dụng kí hiệu “.” thay cho kí hiệu list; tức là hạng thức list(X,Y) trong Prolog được viết là (X,Y).

Như vậy các cặp biểu diễn sau là tương đương:

Biểu diễn hạng thức Biểu diễn dấu ngoặc vuông list(X,Y) [ X | Y ]

list(X,list(Y,Z)) [ X, Y | Z ]

list(X,list(Y,list(Z, [ ]))) [ X,Y,Z ]

Trong ngôn ngữ Prolog, ta có thể sử dụng cả hai dạng biểu diễn danh sách.

**Các phép toán cơ bản trên danh sách:**

• **Quan hệ thành phần**

Quan hệ “đối tượng X là thành phần của danh sách L” được biểu diễn bởi vị từ:

--

Member(X,L)

Quan hệ này được xác định như sau:

X là thành phần của danh sách L nếu:

(1) hoặc X là đầu danh sách L

(2) hoặc X là thành phần của đuôi danh sách L

Tức là, vị từ Member(X,L) được xác định bởi công thức:

$$\text{Member}(X,L) \leftarrow (L = [X \mid L_1]) \vee (L = [Y \mid L_2] \wedge \text{Member}(X,L_2))$$

Chẳng hạn,

Member(a,[a,b,c])

Member([b,c],[a,[b,c],d])

là các quan hệ đúng, còn

là sai. Member(b,[a,[b,c],d])

### • Kết nối hai danh sách thành một danh sách

Phép toán kết nối 2 danh sách  $L_1$  và  $L_2$  thành một danh sách L được biểu diễn bởi vị từ

Conc( $L_1, L_2, L$ )

Quan hệ này được xác định như sau:

Danh sách L là kết nối của 2 danh sách  $L_1$  và  $L_2$  nếu:

(1) hoặc  $L_1 = []$  và  $L = L_2$

(2) hoặc  $L_1 \neq []$  và đầu của L là đầu của  $L_1$  và đuôi của L là kết nối của đuôi  $L_1$  và  $L_2$ .

Tức là, quan hệ Conc( $L_1, L_2, L$ ) được xác định bởi công thức:

64

CuuDuongThanCong.com <https://fb.com/tailieudientucntt>

$$\text{Conc}(L_1, L_2, L) \leftarrow (L_1 = [] \wedge L = L_2) \vee (L_1 = [X \mid L_3] \wedge L = [X \mid L_4] \wedge \text{Conc}(L_3, L_2, L_4))$$

Chẳng hạn,

Conc([a,b],[1,2,3], [a,b,1,2,3])

Conc([a,[b,c]], [a, ], d, [a,[b,c],a, ], d)

là các quan hệ đúng, nhưng:

Conc([a,b],[c,d],[a,b,a,c,d])

là sai.

### • Loại bỏ một thành phần khỏi danh sách

Việc loại bỏ một thành phần X khỏi danh sách L được xác định bởi quan hệ:

Delete(X,L,L1)

Quan hệ này được xác định như sau:

Danh sách  $L_1$  là kết quả loại thành phần X khỏi danh sách L, nếu

(1) hoặc X là đầu của L và  $L_1$  là đuôi của L

(2) hoặc đầu của  $L_1$  là đầu của L và đuôi của  $L_1$  là kết quả việc loại X khỏi đuôi của L.

Tức là,

$$\text{Delete}(X,L,L_1) \leftarrow (L = [X \mid L_1]) \vee (L = [Y \mid L_2] \wedge \text{Delete}(X,L_2,L_3) \wedge L_1 = [Y \mid L_3])$$

= $[Y, L_3]$ ) Chẳng hạn, ta có quan hệ đúng sau:

Delete(a,[a,b,a,c],[b,a,c])

Delete(a,[a,b,a,c],[a,b,c])

#### • Quan hệ danh sách con

Quan hệ Sublist( $L_1, L$ ) là đúng nếu danh sách  $L_1$  là danh sách con của danh sách  $L$ , chẳng hạn: Sublist([c,d,e],[a,b,c,d,e,f]) à quan hệ đúng; còn Sublist([b,d],[a,b,c,d,e,f]) là sai. Đương nhiên, nếu  $L_1$  là danh sách con của  $L$ , thì danh sách  $L$  có thể phân tách thành các danh sách con. Do đó ta có thể xác định quan hệ này như sau:

$L_1$  là danh sách con của danh sách  $L$ , nếu:

(1)  $L$  là kết nối của  $L_2$  và  $L_3$ , và

(2)  $L_3$  là kết nối của  $L_1$  và  $L_4$

tức là, ta có: Sublist( $L_1, L$ )  $\leq$  Conc( $L_2, L_3, L$ )  $\wedge$  Conc( $L_1, L_4, L_3$ )

Trên đây là một số phép toán cơ bản trên danh sách, để dễ dàng cho xử lý danh sách, ta cần xác định

một số phép toán khác.

**Tập hợp** là một khái niệm cơ bản trong toán học, ta có thể mô tả rất nhiều vấn đề bằng cách sử dụng tập hợp và các phép toán trên tập hợp. Tuy nhiên ta có thể biểu diễn tập hợp bởi danh sách, và các phép toán tập hợp có thể được xác định thông qua các phép toán trên danh sách. **3.5.6.5. Xây dựng cơ sở tri thức**

Như chúng ta đã biết một hệ tri thức bao gồm hai thành phần chính là cơ sở tri thức (CSTT) và thủ tục suy diễn. Như vậy để thiết kế một hệ tri thức, chúng ta cần phải xây dựng nên CSTT. CSTT bao gồm các câu (trong một ngôn ngữ biểu diễn tri thức nào đó, ở đây là logic vị từ cấp một). Các câu này biểu diễn tri thức của chúng ta về một lĩnh vực áp dụng mà chúng ta đang quan tâm. Logic vị từ cấp một là một công cụ mạnh để biểu diễn tri thức và lập luận. Song một

vấn đề đặt ra là, ta phải lựa chọn các đối tượng nào, các sự kiện nào, các quan hệ nào, các tri thức chung nào để đưa vào CSTT, sao cho với CSTT đó, thủ tục suy diễn có thể đưa ra các câu trả lời cho các câu hỏi của người sử dụng.

Quá trình xây dựng CSTT được gọi là **công nghệ tri thức** (knowledge engineering). Người kỹ sư tri thức (người kỹ sư làm việc trong lĩnh vực công nghệ tri thức) có thể nắm được các công nghệ làm ra CSTT, nhưng nói chung anh ta không hiểu biết về lĩnh vực áp dụng. Người kỹ sư tri thức cần phải phỏng vấn các chuyên gia trong lĩnh vực đó để khai thác các tri thức cần thiết đưa vào CSTT, quá trình này được gọi là **quá trình thu thập tri thức** (knowledge acquisition). Chỉ nắm được cú pháp và ngữ nghĩa của ngôn ngữ biểu diễn tri thức, chúng ta không thể xây dựng được CSTT. Người kỹ sư cần phải hiểu biết các kỹ thuật của công nghệ tri thức. Trong mục này chúng ta sẽ trình bày vắn tắt các kỹ thuật cơ bản của công nghệ tri thức.

Phương pháp luận xây dựng một CSTT bao gồm các bước chính sau đây:

Trước hết cần phải xác định CSTT mà ta xây dựng nói tới các loại đối tượng nào, các sự kiện nào, các thuộc tính nào, các quan hệ nào. Muốn vậy người kỹ sư tri thức cần phải tìm hiểu các chuyên gia trong lĩnh vực áp dụng để có sự hiểu biết đầy đủ về lĩnh vực đó. Cần nhớ rằng, chỉ khi nào đã hiểu biết tường tận về lĩnh vực áp dụng mới bắt đầu xây dựng CSTT. • **Xây dựng hệ**

## thống từ vựng.

Hệ thống từ vựng bao gồm các hằng, các hàm và các vị từ. Bước này thực hiện việc chuyển dịch các khái niệm trong lĩnh vực áp dụng thành các tên hằng, tên hàm, tên vị từ. Các tên hằng, tên hàm, tên vị từ cần được lựa chọn sao cho người đọc CSTT có thể hiểu được dễ dàng ý nghĩa của nó. Chẳng hạn, ta có thể dùng vị từ HasColor (x,y) để biểu diễn quan hệ "đối tượng x có màu y", dùng vị từ Small(x) để biểu diễn thuộc tính "đối tượng x có cỡ nhỏ". Cùng một quan hệ, ta có thể biểu diễn bởi hàm hoặc vị từ, chẳng hạn, ta có thể sử dụng hàm HusbandOf(x) để biểu diễn đối tượng là chồng của x, hoặc có thể sử dụng vị từ IsHusband(y,x) để biểu diễn quan hệ "y là chồng của x". Một ví dụ khác, nếu chúng ta quan tâm tới các đối tượng với các màu khác nhau, ta có thể sử dụng vị từ HasColor(x,y) đã đưa ra ở trên. Song nếu chỉ quan tâm tới các đối tượng với màu đỏ hay không, ta có thể dùng vị từ Red(x) (x có màu đỏ).

Như vậy, trong giai đoạn xây dựng hệ thống từ vựng, ta cần quyết định biểu diễn một quan hệ bởi hàm hay vị từ, các hàm (vị từ) là các hàm (vị từ) của các đối số nào. Ta cần chọn các tên hằng, tên hàm, tên vị từ sao cho nó nói lên được nội dung mà nó cần mô tả.

- Biểu diễn tri thức chung về lĩnh vực:

Hệ thống từ vựng chỉ là danh sách các thuật ngữ. Chúng ta cần phải sử dụng các thuật ngữ này để viết ra các công thức logic mô tả các tri thức chung của chúng ta về lĩnh vực áp dụng, và cũng là để chính xác hoá các thuật ngữ mà chúng ta đưa ra trong hệ thống từ vựng. Chẳng hạn, khi nói tới các quan hệ họ hàng, ta cần đưa vào các vị từ Sibling(x,y) biểu diễn quan hệ "x và y là anh em", vị từ Parent (u,v) biểu diễn "u là cha mẹ của v",... Sau đó ta cần đưa vào tiên đề sau đây:

$$\forall x,y(\text{Sibling}(x,y) \Leftrightarrow (x \neq y) \wedge \exists p(\text{Parent}(p,y) \wedge \text{Parent}(p,x)))$$

Câu này nói rằng, nếu x và y là anh em thì họ phải cùng cha mẹ và ngược lại. Một số lượng lớn các tri thức của con người được mô tả bởi các câu trong ngôn ngữ tự nhiên. Do đó để xây dựng CSTT chúng ta cần biết chuyển các câu trong ngôn ngữ tự nhiên thành các câu trong ngôn ngữ vị từ. Sau đây là một vài ví dụ cho ta biết cách chuyển các câu trong ngôn ngữ tự nhiên thành công thức logic (chúng tôi khuyên độc giả hãy tự mình viết ra các công thức

logic trước khi đọc tiếp). Sử dụng các vị từ Mushroom(x) (x là nấm), Purple(y) (y màu tím), Poisonnous(z) (z có độc), chúng ta hãy chuyển các câu sau đây thành các công thức logic. Mọi cây nấm tím đều có độc

$$\forall x(\text{Mushroom}(x) \wedge \text{Purple}(x) \Rightarrow \text{Poisonnous}(x))$$

Tất cả cây nấm hoặc có màu tím hoặc có độc

$$\forall x(\text{Mushroom}(x) \Rightarrow \text{Purple}(x) \vee \text{Poisonnous}(x))$$

Mọi cây nấm hoặc màu tím hoặc có độc nhưng không là cả hai

$$\forall x(\text{Mushroom}(x) \Rightarrow (\text{Purple}(x) \wedge \neg \text{Poisonnous}(x)) \vee (\neg \text{Purple}(x) \wedge \text{Poisonnous}(x)))$$

Tất cả các cây nấm tím đều có độc trừ một cây

$\exists x(\text{Mushroom}(x) \wedge \text{Purple}(x) \wedge$

$\text{Poisonous}(x)) \wedge \forall y(\text{Mushroom}(y) \wedge \text{Purple}(y) \wedge (y \neq x) \Rightarrow \text{Poisonous}(y)))$

Chỉ có hai cây nấm tím

$\exists x, y(\text{Mushroom}(x) \wedge \text{Purple}(x) \wedge \text{Mushroom}(y) \wedge \text{Purple}(y) \wedge (x \neq y) \wedge$

$\forall z, x(\text{Mushroom}(z) \wedge \text{Purple}(z) \Rightarrow (z=x) \vee (z=y)))$

Sau khi chúng ta đã viết ra các công thức logic (các tiên đề) mô tả các tri thức chung về lĩnh vực áp dụng, có thể xem như chúng ta đã xây dựng nên một CSTT. Để thuận lợi cho việc lưu trữ CSTT trong máy tính, và thuận lợi cho thủ tục suy diễn hoạt động, chúng ta có thể chuẩn hoá các câu trong CSTT.

Trong toán học, người ta cố gắng tìm được một tập các tiên đề độc lập, tức là trong đó không có tiên đề nào có thể suy ra từ các tiên đề còn lại. Tuy nhiên, trong các hệ tri thức, CSTT có thể chứa các tiên đề "thừa", chúng không làm tăng thêm các tri thức mới được suy ra, song chúng có thể làm cho quá trình suy diễn hiệu quả hơn.

### 3.5.6.5. Cài đặt cơ sở tri thức

Trong mục trước, chúng ta đã xét một số kĩ thuật để xây dựng cơ sở tri thức (CSTT). CSTT là tập các mệnh đề (tiên đề) mô tả tri thức của về một lĩnh vực nào đó. Giả thiết, CSTT đã được chuẩn hoá, tức gồm các câu tuyên (mỗi câu là tuyên của các literal). Thủ tục suy diễn, nói chung, là thủ tục chứng minh bác bỏ. Trong trường hợp CSTT là một tập các câu Horn (các luật If-Then), ta có thể sử dụng thủ tục suy diễn tiến (forward chaining) hoặc thủ tục suy diễn lùi (backward chaining) sẽ được trình bày sau để lập luận. Trong các thủ tục suy diễn, ta phải thực hiện lập công việc như sau: tìm ra các literal có thể hợp nhất được với một literal đang xét nào đó; nếu

tìm được thì áp dụng luật phân giải cho hai câu chứa hai literal hợp nhất được đó. Do đó, để cho thủ tục suy diễn thực hiện hiệu quả, cần cài đặt CSTT sao cho công việc lập lại trên được thực hiện hiệu quả. Trước hết, ta cần biểu diễn các hạng thức và các câu phân tử bằng các cấu trúc dữ liệu thích hợp.

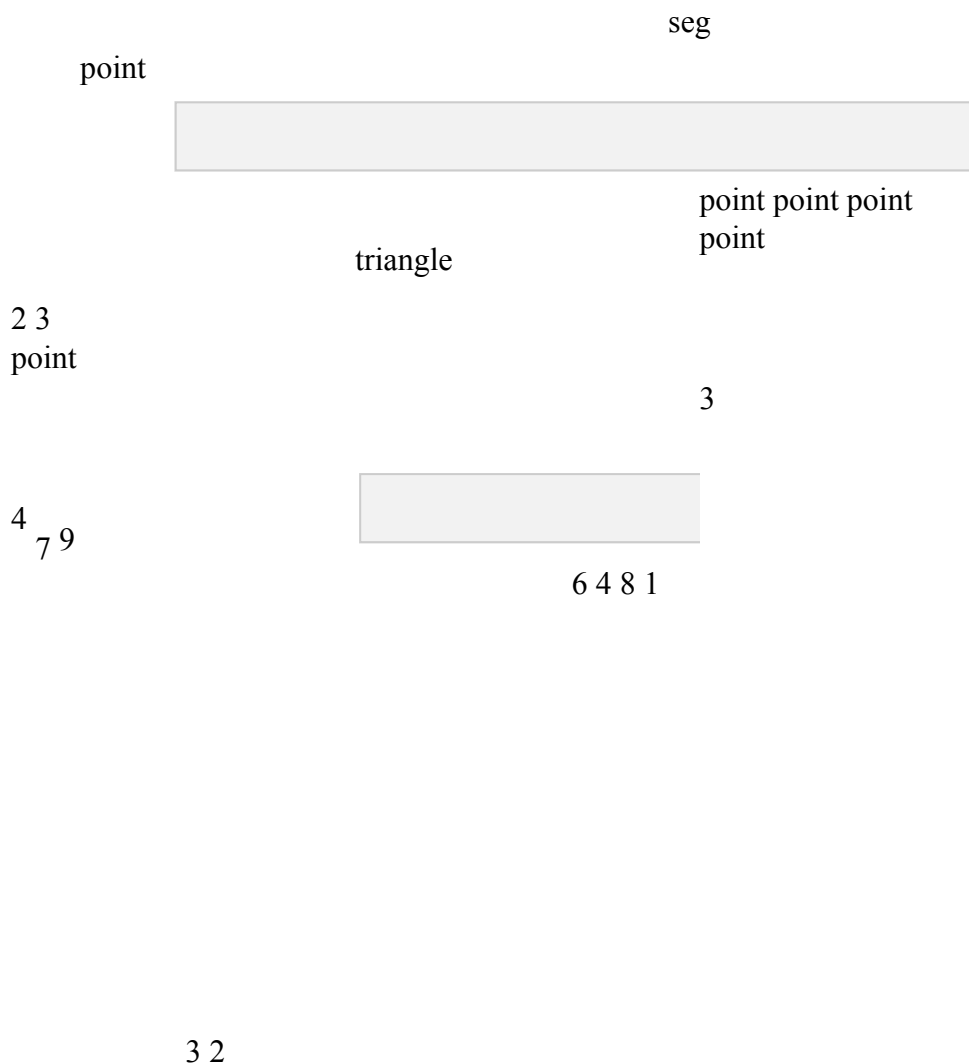
#### 3.5.6.5.1. Cài đặt các hạng thức và các câu phân tử:

Về mặt cú pháp, các hạng thức và các câu phân tử có cấu trúc giống nhau. Do đó, ta chỉ xét việc cài đặt các hạng thức. Các hạng thức biểu diễn đối tượng trong thế giới hiện thực. Từ các đối tượng đơn được biểu diễn bởi các kí hiệu, hằng, biến; ta có thể tạo ra các đối tượng có cấu trúc. Các đối tượng có cấu trúc là các đối tượng có một số thành phần. Các thành phần này lại có thể là các đối tượng có cấu trúc khác. Để kết hợp các thành phần tạo thành một đối tượng mới, sử dụng kí hiệu hàm (function). Chẳng hạn, sử dụng kí hiệu hàm *Date*, ngày 1 tháng 5 năm 2000 được

May<sup>2000</sup> Day

**Hình 3.3. Cây biểu diễn hạng thức date(Day, May,2000)**

Các ví dụ sau đây cho ta thấy cách tạo ra các hạng thức biểu diễn các đối tượng hình học trong mặt phẳng. Một điểm trong không gian hai chiều được xác định bởi hai tọa độ; một đoạn thẳng được xác định bởi hai điểm; một tam giác được xác định bởi ba điểm. Do đó, nếu ta sử dụng các kí hiệu hàm point (điểm), seg (đoạn thẳng), triangle (tam giác) thì điểm có tọa độ (2,3) được biểu diễn bởi hạng thức point (2, 3)



**Hình 3.4. Các cây biểu diễn các hạng thức**

Đoạn thẳng nối hai điểm (4, 7) và (9, 3) được biểu diễn bởi hạng thức:  
seg(point (4, 7), point (9, 3))

Tam giác có ba đỉnh là ba điểm (3, 2), (6, 4), (8, 1) được biểu diễn bởi hạng thức:

68



Triangle (point(3, 2), point(6, 4), point(8, 1))

Các hạng thức trên được biểu diễn bởi các cấu trúc cây trong hình 3.4

Một cách tổng quát, các hạng thức được biểu diễn bởi các cây (chẳng hạn, các cây trong **hình 3.4**), trong đó gốc của cây là các kí hiệu hàm. Nếu các đối số này không phải là kí hiệu hằng, hoặc kí hiệu biến thì chúng là các cây con của gốc được tạo thành theo quy tắc trên.

Bây giờ chúng ta xem xét các đối tượng danh sách được biểu diễn bởi cấu trúc cây như thế nào. Nhớ lại rằng, danh sách [spring, summer, autumn, winter]

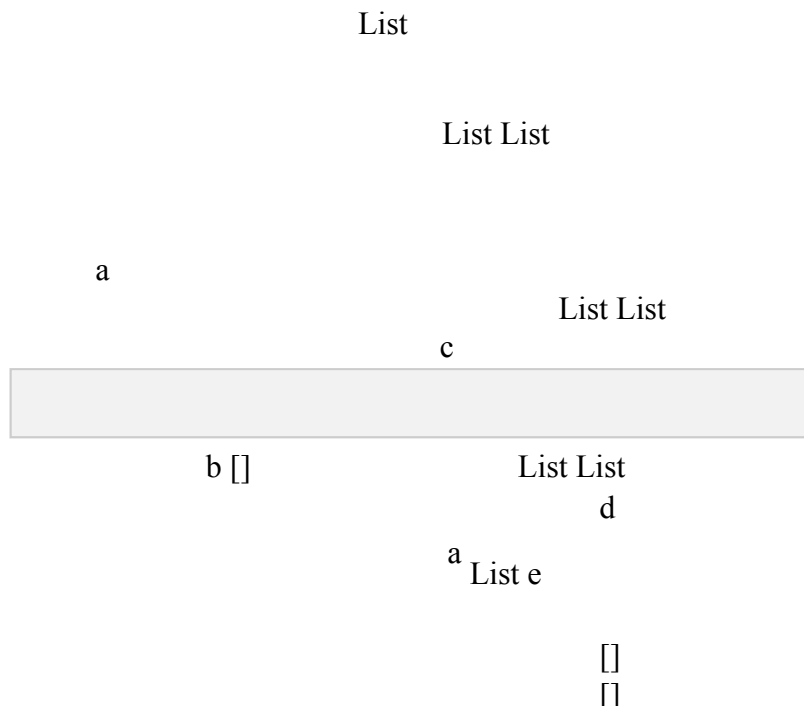
được biểu diễn bởi hạng thức: list (spring, list (summer, list (autumn, list (winter,[ ]))))

Hạng thức này được biểu diễn bởi cây trong **hình 3.5**



**Hình 3.5 Cây biểu diễn danh sách [spring, summer, autumn, winter]**

Một cách tương tự, bằng cách chuyển sang dạng hạng thức, ta có thể biểu diễn danh sách: [a,b], c, [ [a, e], d ] bởi cây trong **hình 3.6**



### Hình 3.6 Cây biểu diễn danh sách [ [a,b], c, [[a,e],d] ]

69

CuuDuongThanCong.com <https://fb.com/tailieudientucntt>

Trong cây **hình 3.6** nếu a, b, c, d, e không phải là các đối tượng đơn mà là các đối tượng có cấu trúc được biểu diễn bởi các hạng thức, thì ở vị trí của các đỉnh gắn nhãn a, b, c, d, e sẽ là các cây con biểu diễn các hạng thức đó.

Trên đây chúng ta đã chỉ ra rằng, các hạng thức (và các câu phân tử) có thể biểu diễn một cách tự nhiên bởi các cấu trúc cây.

#### 3.5.6.5.2. Cài đặt cơ sở tri thức


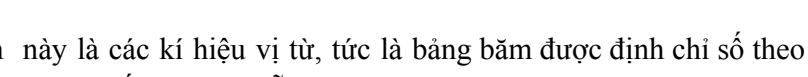
Chúng ta đã biết cách tạo ra các cấu trúc dữ liệu để biểu diễn các hạng thức và các câu phân tử. Bây giờ chúng ta nghiên cứu các kỹ thuật cài đặt cơ sở tri thức sao cho các thủ tục suy diễn có thể thực hiện có hiệu quả. Giả sử CSTT bao gồm các câu tuyển dạng

$$C = P_1 \vee \dots \vee P_m \vee [Q_1 \vee \dots \vee Q_n]$$

trong đó,  $P_i$  ( $i = 1, \dots, m$ ,  $m \geq 0$ ),  $Q_k$  ( $k = 1, 2, \dots, n$ ,  $n \geq 0$ ) là các câu phân tử. Một cách tự nhiên, mỗi câu C có thể được biểu diễn bởi bản ghi gồm hai trường:

- Danh sách các Literal dương [ $P_1, \dots, P_m$ ]
- Danh sách các Literal âm [ $Q_1, \dots, Q_n$ ]

Một cách đơn giản nhất, ta có thể cài đặt CSTT như một danh sách các câu tuyển. Tuy nhiên với cách cài đặt này, thủ tục suy diễn kém hiệu quả, bởi mỗi lần cần xét xem một câu phân tử S có hợp nhất với một thành phần nào đó của một câu trong CSTT, ta phải đi qua danh sách, xem xét từng thành phần của các câu trong danh sách cho tới khi tìm ra hoặc đi tới hết danh sách.

hơn, ta  Một giải pháp khác tốt bởi bảng  có thể cài đặt CSTT bằng bảng này là các kí hiệu vị từ, tức là bảng băm được định chỉ số theo các kí hiệu vị từ. Trong bảng băm, tại chỉ số ứng với mỗi kí hiệu vị từ ta sẽ lưu:

- Danh sách các literal dương của kí hiệu vị từ đó.
- Danh sách các literal âm.
- Danh sách các câu mà kí hiệu vị từ xuất hiện trong các literal dương của câu (Câu dương).
- Danh sách các câu mà kí hiệu vị từ xuất hiện trong các literal âm của câu (Câu âm). Ví dụ:

Giả sử CSTT chứa các câu sau:

Brother (An, Ba)

Brother (Tam, Hoa)

]Brother (Lan, Cao)

]Brother (x,y)

∨

Male(x)  $\downarrow$  Brother (x,y)  $\vee$   $\downarrow$  Male(y)  $\vee$   $\downarrow$  Brother (y,x)

Male(Cao)

Male(Ba)

$\downarrow$  Male(Lan)

Khi đó, tại các chỉ số ứng với khoá Brother và Male, bảng băm sẽ lưu giữ các thành phần được cho trong bảng sau

70

CuuDuongThanCong.com <https://fb.com/tailieudientucntt>

Khoá	Literal dương	Literal âm	Câu dương	Câu âm
Brother	Brother(An,Ba) Brother(Tam, Hoa)	$\downarrow$ Brother (Lan, Cao)	$\downarrow$ Brother(x,y) $\vee$ Male(y) $\vee$ Brother(y, x)	$\downarrow$ Brother (x,y) $\vee$ $\downarrow$ Male(y) $\vee$ Brother(y,x)  $\downarrow$ Brother (x,y) $\vee$ Male(x)
Male	Male(Cao) Male(Ba)	$\downarrow$ Male(Lan)	$\downarrow$ Brother(x,y) $\vee$ Male(x)	$\downarrow$ Brother (x,y) $\vee$ $\downarrow$ Male(y) $\vee$ Brother (y,x)

Cài đặt CSTT bởi bảng băm định chỉ số theo các kí hiệu vị từ là phương pháp rất hiệu quả cho việc tìm kiếm trên CSTT, nếu như CSTT chứa nhiều kí hiệu vị từ và với mỗi kí hiệu vị từ chỉ có một số ít các câu chứa kí hiệu vị từ đó. Tuy nhiên trong một số áp dụng, có thể có rất nhiều câu chứa cùng một kí hiệu vị từ nào đó. Chẳng hạn, Cơ Sở Tri Thức có thể chứa hàng triệu câu nói về người lao động, mỗi người lao động được biểu diễn bởi các thông tin về họ tên, ngày tháng năm sinh, số thẻ bảo hiểm, công việc (Công việc được xác định bởi nơi làm việc và tiền lương). Tức là mỗi người lao động được mô tả bằng câu có dạng:

Worker (Tom, date (3, may,1965), 012-34-567, job(UNIMEX, 300))

(câu này nói rằng, có người lao động tên là Tom, sinh ngày 3 tháng 5 năm 1965, số thẻ bảo hiểm 012-34-567, làm việc tại công ty UNIMEX với mức lương 300).

Trong các trường hợp như thế, để tìm kiếm có hiệu quả, ngoài việc xây dựng bảng băm định chỉ số theo các đối số của các kí hiệu vị từ, ta cần xây dựng các bảng băm định chỉ số theo các đối số của các vị từ. Chẳng hạn, ở đây các literal dương ứng với khoá Worker cần được tổ chức dưới dạng bảng băm định chỉ số theo khóa là số bảo hiểm y tế hoặc họ tên và ngày tháng năm sinh.

### 3.5.6.5.3 Biểu diễn tri thức bằng luật và lập luận

Với một CSTT gồm các câu trong logic vị từ cấp một, ta có thể chứng minh công thức có là hệ quả logic của CSTT hay không bằng phương pháp chứng minh bác bỏ và thủ tục phân giải. Tuy nhiên, thủ tục chứng minh này có độ phức tạp lớn và đòi hỏi chiến lược giải một cách thích hợp. Vì lý do này, các nhà nghiên cứu cố gắng tìm các tập con của logic vị từ cấp một, sao cho chúng đủ khả năng biểu diễn CSTT trong nhiều lĩnh vực, và có thể đưa ra các thủ tục suy diễn hiệu quả.

Các tập con này của logic vị từ cấp một sẽ xác định các ngôn ngữ biểu diễn tri thức đặc biệt. Trong phần này chúng ta sẽ nghiên cứu ngôn ngữ chỉ bao gồm các câu Horn (các luật *nếu - thì*).

Chỉ sử dụng các luật *nếu - thì*, ta không thể biểu diễn được mọi điều trong logic vị từ cấp một. Tuy nhiên, với các luật *nếu - thì* ta có thể biểu diễn được một khối lượng lớn tri thức trong nhiều lĩnh vực khác nhau, và có thể thực hiện các thủ tục suy diễn hiệu quả.

### Biểu diễn tri thức bằng luật sinh

Ngôn ngữ bao gồm các luật *nếu - thì* (if - then), còn gọi là các **luật sản xuất hay luật sinh** (production rule), là ngôn ngữ phổ biến nhất để biểu diễn tri thức. Nhớ rằng, các câu Horn có dạng

$$P_1 \wedge \dots \wedge P_n \Rightarrow Q$$

trong đó các  $P_i$  ( $i = 1, \dots, n$ ) và  $Q$  là các câu phân tử.

Các câu Horn còn được viết dưới dạng

71

CuuDuongThanCong.com <https://fb.com/tailieudientucntt>

nếu  $P_1$  và  $P_2 \dots$  và  $P_n$  thì  $Q$

(if  $P_1$  and ... and  $P_n$  then  $Q$ )

các  $P_i$  ( $i = 1, \dots, n$ ) được gọi là các điều kiện,  $Q$  được gọi là kết luận của luật.

Các luật *nếu - thì* có các ưu điểm sau đây

- Mỗi luật *nếu - thì* mô tả một phần nhỏ tương đối độc lập của tri thức.
- Có thể thêm và cơ sở tri thức các luật mới, hoặc loại bỏ một số luật cũ mà không ảnh hưởng nhiều tới các luật khác.
- Các hệ tri thức với cơ sở tri thức gồm các luật *nếu - thì* có khả năng đưa ra lời giải thích cho các quyết định của hệ.

Các luật *nếu - thì* là dạng biểu diễn tự nhiên của tri thức. Bằng cách sử dụng các luật *nếu - thì* chúng ta có thể biểu diễn được một số lượng lớn tri thức của con người về tự nhiên, về xã hội, kinh nghiệm của con người trong lao động, sản xuất, tri thức của các thầy thuốc, tri thức của các kỹ sư, tri thức trong các ngành khoa học: kinh tế, sinh học, hoá học, vật lý, toán học,...

Sau đây là một luật về chẩn đoán bệnh:

Nếu

1. bệnh nhân ho lâu ngày, và
2. bệnh nhân thường sốt vào buổi chiều

Thì bệnh nhân có khả năng bệnh lao

Một luật tiết:

về kinh nghiệm dự báo thời

Nếu chuồn chuồn bay thấp

thì trời sẽ mưa

Nhiều định lý trong toán học có thể biểu diễn bởi các luật. Chẳng hạn,

Nếu

1. tam giác có một góc bằng  $60^\circ$ , và

2. tam giác có hai cạnh bằng nhau  
thì tam giác đó là tam giác đều.

Trên đây chúng ta chỉ xét các luật trong đó mỗi phần kết luận của một luật xác định một khẳng định mới được suy ra khi tất cả các điều kiện của luật được thoả mãn. Trong nhiều áp dụng, cơ sở luật của hệ cần được đưa vào các luật mà phần kết luận của luật là một **hành động** hệ cần thực hiện. Gọi các luật dạng này là luật hành động (action). Một luật hành động có dạng:

nếu

1. <điều kiện 1>, và

2. <điều kiện 2>, và

....

m. <điều kiện m>

thì <hành động>

Hành động trong các luật hành động có thể là thêm vào một sự kiện mới, có thể là loại bỏ một sự kiện đã có trong bộ nhớ làm việc; hành động cũng có thể là thực hiện một thủ tục nào đó. Trong các robot được trang bị một hệ dựa trên luật, thì phần kết luận của luật có thể là một hành động nào đó mà robot cần thực hiện.

72

CuuDuongThanCong.com <https://fb.com/tailieudientucntt>

Người ta phân biệt hai dạng hệ, các hệ dựa trên luật sử dụng lập luận tiến và phần kết luận của các luật xác định các khẳng định mới được gọi là các **hệ diễn dịch** (*deduction systems*). Các hệ dựa trên luật mà phần kết luận của các luật xác định các hành động cần thực hiện được gọi là **hệ hành động dựa trên luật** (*rule-based reaction systems*).

Trong các hệ diễn dịch, chúng ta xem mọi luật đều sinh ra các khẳng định mới “có giá trị” như nhau; tức là, ta không xem khẳng định do luật này sinh ra là “tốt” hơn khẳng định do luật khác sinh ra. Do đó trong các hệ diễn dịch, khi mà nhiều luật có thể cháy được (một luật được gọi là **cháy được** nếu tất cả các điều kiện của luật được thoả mãn) ta có thể cho tất cả các luật đó cháy (một luật cháy để sinh ra khẳng định mới).

Trong các hệ hành động, khi có có nhiều hơn một luật có thể cháy, nói chung, chúng ta chỉ muốn thực hiện một trong các hành động có thể. Do đó, trong các hệ hành động, chúng ta cần có chiến lược giải quyết va chạm để quyết định cho luật nào cháy trong số các luật có thể cháy. Sau đây là một số chiến lược giải quyết va chạm.

- Sắp xếp các luật theo thứ tự ưu tiên. Trong các luật có thể cháy, luật nào có mức ưu tiên cao nhất sẽ được thực hiện.
- Sắp xếp dữ liệu. Các sự kiện trong bộ nhớ làm việc được sắp xếp theo thứ tự ưu tiên. Luật nào mà các điều kiện của nó được làm thoả mãn bởi các điều kiện có mức ưu tiên cao sẽ được thực hiện trước.
- Các luật được phân thành các nhóm. Trong mỗi nhóm luật, được chỉ ra các điều kiện để áp

dụng các luật của nhóm. Các điều kiện này liên quan đến nội dung của bộ nhớ làm việc. • Sử dụng các siêu luật

(*metarule*). Đó là các luật mà phần điều kiện của nó liên quan tới

nội dung của các luật và nội dung của bộ nhớ làm việc, còn phần kết luận của nó chỉ ra các luật có thể được áp dụng hoặc có thể bị cấm áp dụng. Các siêu luật sẽ điều khiển sự

cho phép các luật cháy.

Đương nhiên là, việc sử dụng chiến lược giải quyết va chạm nào phụ thuộc vào từng áp dụng. Tuy theo mục đích thiết kế của hệ mà ta lựa chọn chiến lược giải quyết va chạm cho thích hợp.

### Biểu diễn tri thức không chắc chắn

Trong đời sống thực tế, có rất nhiều điều mà ngay cả các chuyên gia cũng không hoàn toàn tin tưởng chúng là đúng hay sai. Đặc biệt là các kết luận trong chẩn đoán y học, trong dự báo thời tiết, trong phỏng đoán sự hỏng hóc của máy móc, chúng ta không thể tin tưởng 100% các kết luận đưa ra là đúng. Chẳng hạn, nếu xe máy đang chạy bị chết máy và kiểm tra xăng hãy còn thì có thể tin rằng 90% là do có vấn đề ở bugi. Tuy nhiên vẫn còn 10% phỏng đoán đó là sai, xe bị

chết máy do các nguyên nhân khác. Do đó trong các hệ dựa trên luật, chúng ta còn phải đưa vào

**mức độ chắc chắn** của các luật và các sự kiện trong cơ sở tri thức. Chúng ta sẽ gán cho mỗi luật hoặc sự kiện một mức độ chắc chắn nào đó, mức độ chắc chắn là một số nằm giữa 0 và 1. Cách viết

$$A_1 \wedge \dots \wedge A_n \Rightarrow B : C \quad (1)$$

có nghĩa là luật  $A_1 \wedge \dots \wedge A_n \Rightarrow B$  có độ chắc chắn là  $C$  ( $0 \leq C \leq 1$ ).

Chúng ta cần phải đưa ra phương pháp xác định mức độ chắc chắn của các kết luận được suy ra.

Trước hết chúng ta đánh giá kết luận suy ra từ luật chỉ có một điều kiện. Giả sử ta có luật  $A \Rightarrow$

$$B : C \quad (2)$$

Theo lý thuyết xác suất, ta có

73

CuuDuongThanCong.com <https://fb.com/tailieudientucntt>

$$\Pr(B) = \Pr(B | A)\Pr(A) \quad (3)$$

trong đó  $\Pr(B)$ ,  $\Pr(A)$  là xác suất của sự kiện  $B$ ,  $A$  tương ứng (tức là mức độ chắc chắn của  $B$ ,  $A$  tương ứng), còn  $\Pr(B | A)$  là xác suất có điều kiện của  $B$  khi  $A$  đã xảy ra, ở đây  $\Pr(B | A)$  là mức độ chắc chắn của luật  $A \Rightarrow B$ , tức là bằng  $C$ .

Trong trường hợp luật có  $n$  ( $n > 1$ ) điều kiện, tức là các luật dạng (1), ta xem  $A = A_1 \wedge \dots \wedge A_n$ . Trong trường hợp này, mức độ chắc chắn của  $A$ ,  $\Pr(A)$  được tính bằng các phương pháp khác nhau, tùy thuộc vào các sự kiện  $A_i$  ( $i = 1, \dots, n$ ) là độc lập hay phụ thuộc.

Giả sử các sự kiện  $A_i$  ( $i = 1, \dots, n$ ) là độc lập, khi đó

$$\Pr(A) = \Pr(A_1) \dots \Pr(A_n) \quad (4)$$

Ví dụ. Giả sử cơ sở tri thức của hệ chứa luật sau

**IF** 1.  $X$  có tiền án, và

2.  $X$  có thù oán với nạn nhân  $Y$ , và

3.  $X$  đưa ra bằng chứng ngoại phạm sai

**THEN**  $X$  là kẻ giết  $Y$ .

với mức độ chắc chắn 90%.

Giả sử ta có các sự kiện

Hung có tiền án, với mức độ chắc chắn là 1.

Hung có thù oán với nạn nhân Meo, với mức độ chắc chắn là 0,7.

Hung

đưa ra bằng chứng ngoại

phạm sai, với mức độ chắc chắn là 0,8.

Từ các sự kiện và luật trên, ta có

$$\Pr(A) = 1.0.7.0.8 = 0,56$$

$$\Pr(B) = 0,9.0,56 = 0,504$$

Như vậy mức độ chắc chắn của kết luận “Hung là kẻ giết Meo” là 50, 4%. Công thức (4) chỉ áp dụng cho các sự kiện  $A_1, \dots, A_n$  là độc lập (tức sự xuất hiện của sự kiện này không ảnh hưởng gì đến sự xuất hiện của các sự kiện khác). Nếu các sự kiện  $A_1, \dots, A_n$  là phụ thuộc, ta có thể tính mức độ chắc chắn của điều kiện của luật,  $A = A_1 \wedge \dots \wedge A_n$ , theo công thức sau:

$$\Pr(A) = \min (\Pr(A_1), \dots, \Pr(A_n)) \quad (5)$$

Chẳng hạn, với các thông tin trong ví dụ trên, từ công thức (5) ta có

$$\Pr(A) = \min(1, 0,7, 0,8) = 0,7$$

$$\text{Do đó } \Pr(B) = 0,9. 0,7 = 0,63$$

Ngoài công thức (5), người ta còn đưa ra các phương pháp khác để tính mức độ chắc chắn  $\Pr(A)$ , khi mà  $A = A_1 \wedge \dots \wedge A_n$  và các  $A_1, \dots, A_n$  không độc lập.

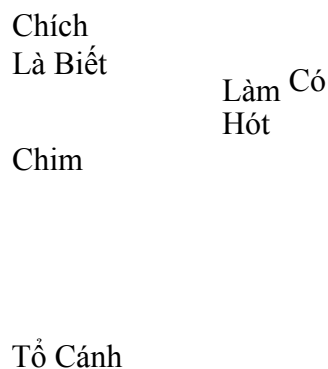
#### 3.5.6.5.4 Biểu diễn tri thức bằng mạng ngữ nghĩa

Mạng ngữ nghĩa là một phương pháp biểu diễn tri thức dễ hiểu nhất. Phương pháp biểu diễn dựa trên đồ thị, trong đó đỉnh là các đối tượng (hay khái niệm) còn các cung là các mối quan hệ giữa các đối tượng (hoặc khái niệm) đó.

**Ví dụ 1:** giữa các đối tượng và khái niệm: *chích, chim, tổ, cánh, hót* có các quan hệ như hình dưới đây

74

CuuDuongThanCong.com <https://fb.com/tailieudientucntt>



**Hình 3.7. Ví dụ đơn giản về mạng ngữ nghĩa**

**Ví dụ 2.** Cho tập quan hệ (hàm) trong tam giác:

a)  $a/\sin\alpha = b/\sin\beta$

b)  $c/\sin\gamma = b/\sin\beta$

c)  $S =$  [REDACTED]

d)  $\alpha + \beta + \gamma = \pi$

e)  $S = h_c \cdot c$



f) Nếu chúng ta đặt các quan hệ này là các nút vuông, các biến là các nút tròn; cung là các đường liên kết giữa các biến có liên quan (như S,  $h_c$ , c) trong các quan hệ (ở đây là công thức e); c); b) ). Học viên có thể dùng mạng ngữ nghĩa để mô tả các mối quan hệ giữa biến và hàm.

### 3.5.6.5.5 Biểu diễn tri thức bằng khung

Khung là một phương pháp biểu diễn tri thức có cấu trúc dữ liệu chứa tất cả tri thức liên quan đến một đối tượng cụ thể nào đó. Khung (Frame) có liên hệ chặt chẽ đến khái niệm hương đối tượng. Khung thường được dùng để biểu diễn những tri thức “chuẩn” hoặc những tri thức dựa trên kinh nghiệm hoặc các điểm đã được hiểu biết cặn kẽ.

Cấu trúc của khung gồm hai thành phần cơ bản “Slot” và “Facet”. Slot là một thuộc tính đặc tả đối tượng ví dụ khung mô tả xe hơi có hai Slot là *Trọng lượng* và *Loại động cơ*. Một Slot có thể chứa nhiều Facet. Các Facet đôi khi còn được gọi là Slot con, Ví dụ Facet của Slot Loại động cơ có thể là: *Kiểu động cơ*, *Số xi lanh*, *Khả năng tăng tốc độ*.

### 3.5.6.5.6 Các phương pháp biểu diễn tri thức khác

Một số cách biểu diễn tri thức khác có thể liệt kê như sau:

- Mô tả tri thức bằng cặp ba Đối tượng-Thuộc tính-Giá trị

(Object-Attribute-Value: O-A-V). Đây là phương pháp biểu diễn cổ điển, đơn giản. Hiệu quả của phương pháp không cao nên hiện nay ít dùng. Mặt khác, cặp ba Đối tượng-Thuộc tính-Giá trị gần giống với phương pháp mạng ngữ nghĩa.

- ③ Phương pháp mô tả tri thức bằng kịch bản (Script)
- ③ Phương pháp mô tả tri thức bằng mặt (Face)
- ③ Phương pháp mô tả tri thức bằng bảng đen (Blackboard)
- ③ Phương pháp mô tả tri thức theo thủ tục

## 3.6 CƠ CHẾ SUY DIỄN

### 3.6.1 Khái niệm về suy diễn và lập luận

Suy diễn (inference) và lập luận (reasoning) là hai khái niệm được dùng chung để chỉ một tiến trình đưa đến kết luận từ các giả thiết cho ở dạng cơ sở tri thức (sự kiện, quy luật). Các hệ tri thức mà cơ sở tri thức bao gồm các luật sẽ được gọi là các **hệ dựa trên luật** (*rule - based system*). Trong các mục còn lại của chương này chúng ta sẽ nghiên cứu các thủ tục suy diễn trong các hệ dựa trên luật.

Một khi chúng ta đã lưu trữ một cơ sở tri thức, chúng ta cần có thủ tục lập luận để rút ra các kết luận từ cơ sở tri thức. Trong các hệ dựa luật, có hai phương pháp luận lập luận cơ bản: • Lập luận tiến và

- lập luận lùi

Chúng ta sẽ phân chia cơ sở tri thức thành hai bộ phận: **cơ sở luật** và **cơ sở sự kiện** (hoặc **bộ nhớ làm việc**). Cơ sở luật bao gồm các luật có ít nhất một điều kiện, biểu diễn các tri thức chung về lĩnh vực áp dụng. Còn cơ sở luật bao gồm các câu phân tử (các luật không điều kiện) mô tả các

sự kiện mà chúng ta biết về các đối tượng trong lĩnh vực áp dụng.

### 3.6.2 Lập luận tiến

Tư tưởng cơ bản của lập luận tiến là áp dụng luật suy diễn Modus Ponens tổng quát). Trong mỗi bước của thủ tục lập luận tiến, người ta xét một luật trong cơ sở luật. Đối sánh mỗi điều kiện của luật với các sự kiện trong cơ sở sự kiện, nếu tất cả các điều kiện của luật đều được thoả mãn thì sự kiện trong phần kết luận của luật được xem là sự kiện được suy ra. Nếu sự kiện này là sự kiện mới (không có trong bộ nhớ làm việc), thì nó được đặt vào bộ nhớ làm việc. Quá trình trên được lặp lại cho tới khi nào không có luật nào sinh ra các sự kiện mới.

Như vậy quá trình lập luận tiến là quá trình xem xét các luật. Với mỗi luật, ta đi từ phần điều kiện tới phần kết luận của luật, khi mà tất cả các điều kiện của luật đều được làm thoả mãn (bởi các sự kiện trong cơ sở sự kiện), thì ta suy ra sự kiện trong phần kết luận của luật. Chính vì lẽ đó mà có tên lập luận tiến (forward chaining hoặc forward reasoning).

Quá trình lập luận tiến không định hướng tới giải quyết một vấn đề nào cả, không định hướng tới tìm ra câu trả lời cho một câu hỏi nào cả. Lập luận tiến chỉ là quá trình suy ra các sự kiện mới từ các sự kiện trong bộ nhớ làm việc. Vì vậy lập luận tiến còn được gọi là **lập luận điều khiển bởi dữ liệu** (*data - driven reasoning*), hoặc **lập luận định hướng dữ liệu** (*data - directed reasoning*).

Để thấy Ví dụ lập luận tiến. được quá trình lập luận tiến diễn ra như thế nào, chúng ta xét ví dụ sau đây. (Ví dụ này là của P. H. Winston xem [17]).

Giả sử cơ sở luật (cơ sở luật về các động vật trong sở thú) gồm các luật sau

Luật 1: nếu động vật có lông mao

thì động vật là loài có vú

Luật 2: nếu động vật có lông vũ

thì động vật là chim

Luật 3: nếu 1. động vật biết bay, và

2. động vật đẻ trứng

76

CuuDuongThanCong.com <https://fb.com/tailieudientucntt>

thì động vật là chim

Luật 4: nếu 1. động vật là loài có vú, và

2. động vật ăn thịt

thì động vật là thú ăn thịt

Luật 5: nếu 1. động vật là loài có vú, và

2. động vật có răng nhọn, và

3. động vật có móng vuốt

thì động vật là thú ăn thịt

Luật 6: nếu 1. động vật là thú ăn thịt, và

2. động vật có màu lông vàng hung, và
3. động vật có đốm sẫm

thì động vật là báo Châu Phi

Luật 7: nếu 1. động vật là thú ăn thịt, và

2. động vật có màu lông vàng hung, và
3. động vật có vằn đen

thì động vật là hổ

Luật 8: nếu 1. động vật là chim, và

2. động vật không biết bay, và

3. động vật có chân dài, và
4. động vật có cổ dài

thì động vật là đà điểu

Luật 9: nếu 1. động vật là chim, và

2. động vật không biết bay, và
3. động vật biết bơi, và
4. động vật có lông đen và trắng

thì động vật là chim cánh cụt

Giả sử một em bé quan sát một con vật có tên là Ki trong sở thú, em thấy nó có các đặc điểm sau

Ki có lông mao

Ki ăn thịt

Ki có màu lông vàng hung

Ki có đốm sẫm

Lúc này cơ sở sự kiện sẽ bao gồm các sự kiện trên.

Thủ tục lập luận tiến xem xét luật 1. Khi biến “động vật” trong luật này được thay bởi Ki, điều kiện của luật trở thành “Ki có lông mao”, đây là một sự kiện có trong bộ nhớ làm việc, do đó ta suy ra “Ki là loài có vú”. Đây là sự kiện mới, do đó nó được thêm vào bộ nhớ làm việc. Xét luật 4, thế biến “động vật” bởi Ki, thì hai điều kiện của luật trở thành:

Ki là loài có vú, và

Ki ăn thịt

Cả hai sự kiện này đều có trong bộ nhớ làm việc, do đó từ luật 4 ta suy ra “Ki là thú ăn thịt”. Sự kiện mới này lại được thêm vào bộ nhớ làm việc. Ta xét tiếp luật 6, thế biến “động vật” bởi Ki, các điều kiện của luật trở thành:

Ki là loài thú ăn thịt, và

Ki có màu lông vàng hung, và

Ki có đốm sẫm

Tất cả các điều kiện này đều đúng, do đó từ luật 6, ta suy ra “Ki là báo Châu Phi”. Như vậy từ các sự kiện đã biết về Ki, lập luận tiến đã suy ra các sự kiện mới sau Ki là loài có vú.

Ki là thú ăn thịt.

Ki là báo Châu Phi.

### 3.6.3 Lập luận lùi

Trong các hệ dựa trên luật, chúng ta còn có thể sử dụng phương pháp **lập luận lùi** (*backward chaining* hoặc *backward reasoning*).

Trong lập luận lùi, người ta đưa ra các giả thuyết cần được đánh giá. Sử dụng lập luận lùi, giả thuyết đưa ra hoặc là được chứng minh, hoặc là bị bác bỏ (bởi các sự kiện trong bộ nhớ làm việc). Cần lưu ý rằng, chúng ta nói giả thuyết được chứng minh, hoặc bị bác bỏ là muốn nói tới nó được chứng minh, hoặc bác bỏ bởi tình trạng hiện thời của bộ nhớ làm việc. Khi mà bộ nhớ làm việc thay đổi (chúng ta thêm vào hoặc loại bỏ một số sự kiện) thì một giả thuyết đã được chứng minh có thể trở thành bị bác bỏ và ngược lại.

Quá trình lập luận lùi diễn ra như sau: Ta đối sánh giả thuyết đưa ra với các sự kiện trong bộ nhớ làm việc. Nếu có một sự kiện khớp với giả thuyết, (ở đây “khớp” được hiểu là hai câu mô tả sự kiện và giả thuyết trùng nhau qua một phép thế nào đó), thì ta xem như giả thuyết là đúng. Nếu không có sự kiện nào khớp với giả thuyết, thì ta đối sánh giả thuyết với phần kết luận của các luật. Với mỗi luật mà kết luận của luật khớp với giả thuyết, ta đi lùi lại phần điều kiện của luật. Các điều kiện này của luật được xem như các giả thuyết mới. Với giả thuyết mới, ta lập lại quá trình trên.

Nếu tất cả các giả thuyết được sinh ra trong quá trình phát triển các giả thuyết bởi các luật được chọn thích hợp đều được thoả mãn (đều có trong bộ nhớ làm việc) thì giả thuyết đã đưa ra được xem là đúng. Ngược lại, dù ta áp dụng luật nào để phát triển các giả thuyết cũng dẫn tới các giả thuyết không có trong bộ nhớ làm việc và không thể quy giả thuyết này về các giả thuyết mới khác, thì giả thuyết đã đưa ra được xem là sai.

#### Ví dụ lập luận lùi.

Để làm sáng tỏ tư tưởng của lập luận lùi, xét với dụ sau. Giả sử bộ nhớ làm việc chứa các sự kiện sau.

Bibi có lông vũ

Bibi có chân dài

Bibi có cổ dài

Bibi không biết bay

Ta đưa ra giả thuyết sau đây

Bibi là đà điểu