ĐẠI HỌC ĐÀ NẴNG
TRƯỜNG ĐẠI HỌC BÁCH KHOA
KHOA CÔNG NGHỆ THÔNG TIN

GIÁO TRÌNH

HỆ CHUYÊN GIA

PGS.TS. PHAN HUY KHÁNH

ĐÀ NẴNG 9-2004

Mục lục

CHƯƠNG 1

Mở đầu

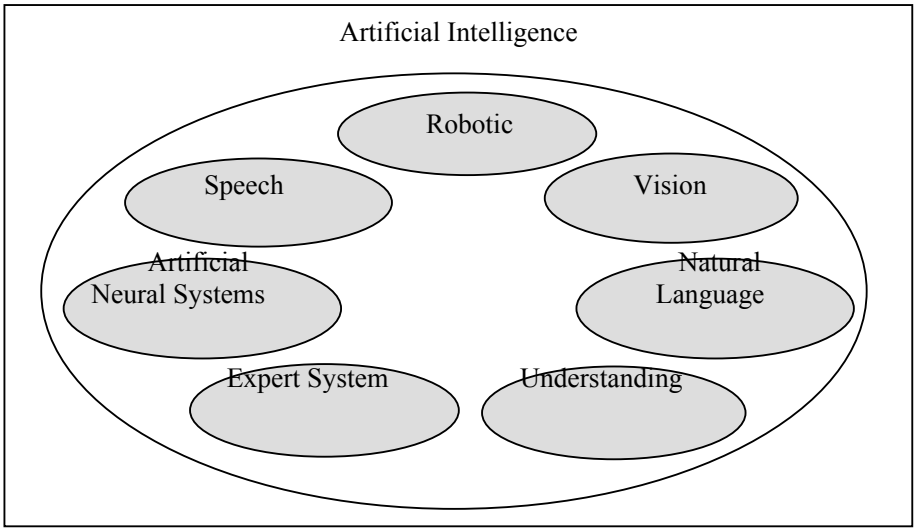
« *When I examine myself and my methods of thought,*
I come to the conclusion that the gift of fantasy has meant more
to me than my talent for absorbing positive knowledge ».
Albert Einstein

I. Giới thiệu hệ chuyên gia

I.1. Hệ chuyên gia là gì ?

Theo E. Feigenbaum : «*Hệ chuyên gia (Expert System) là một chương trình máy tính thông minh sử dụng tri thức (knowledge) và các thủ tục suy luận (inference procedures) để giải những bài toán tương đối khó khăn đòi hỏi những chuyên gia mới giải được*».

Hệ chuyên gia là một hệ thống tin học có thể mô phỏng (emulates) năng lực quyết đoán (decision) và hành động (making abilily) của một chuyên gia (con người). Hệ chuyên gia là một trong những lĩnh vực ứng dụng của *trí tuệ nhân tạo* (Artificial Intelligence) như hình dưới đây.



Hình 1.1. Một số lĩnh vực ứng dụng của trí tuệ nhân tạo

Hệ chuyên gia sử dụng các tri thức của những chuyên gia để giải quyết các vấn đề (bài toán) khác nhau thuộc mọi lĩnh vực.

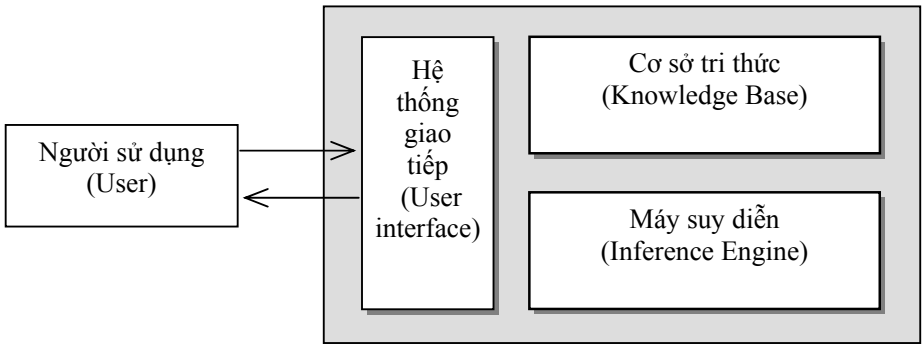
Tri thức (knowledge) trong hệ chuyên gia phản ánh sự tinh thông được tích tụ từ sách vở, tạp chí, từ các chuyên gia hay các nhà bác học. Các thuật ngữ hệ chuyên gia, *hệ thống dựa trên tri thức* (knowledge–based system) hay *hệ chuyên gia dựa trên tri thức* (knowledge–based expert system) thường có cùng nghĩa.

Một hệ chuyên gia gồm ba thành phần chính là *cơ sở tri thức* (knowledge base), *máy suy diễn* hay *môtơ suy diễn* (inference engine), và h*ệ thống giao tiếp với người sử dụng* (user

interface). Cơ sở tri thức chứa các tri thức để từ đó, máy suy diễn tạo ra câu trả lời cho người sử dụng qua hệ thống giao tiếp.

Người sử dụng (user) cung cấp *sự kiện* (facts) là những gì đã biết, đã có thật hay những thông tin có ích cho hệ chuyên gia, và nhận được những câu trả lời là những lời khuyên hay những gợi ý đúng đắn (expertise).

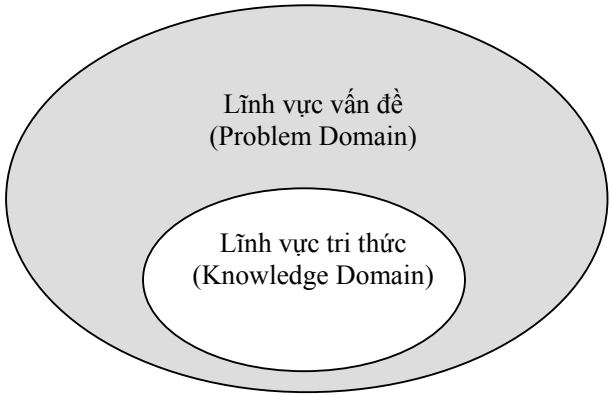
Hoạt động của một hệ chuyên gia dựa trên tri thức được minh họa như sau :



Hình 1.2. Hoạt động của hệ chuyên gia

Mỗi hệ chuyên gia chỉ đặc trưng cho một *lĩnh vực vấn đề* (problem domain) nào đó, như y học, tài chính, khoa học hay công nghệ, v.v..., mà không phải cho bất cứ một lĩnh vực vấn đề nào.

Tri thức chuyên gia để giải quyết một vấn đề đặc trưng được gọi là *lĩnh vực tri thức* (knowledge domain).



Hình 1.3. Quan hệ giữa lĩnh vực vấn đề và lĩnh vực tri thức

Ví dụ : hệ chuyên gia về lĩnh vực y học để phát hiện các căn bệnh lây nhiễm sẽ có nhiều tri thức về một số triệu chứng lây bệnh, lĩnh vực tri thức y học bao gồm các căn bệnh, triệu chứng và chữa trị.

Chú ý rằng lĩnh vực tri thức hoàn toàn nằm trong lĩnh vực vấn đề. Phần bên ngoài lĩnh vực tri thức nói lên rằng không phải là tri thức cho tất cả mọi vấn đề.

Tùy theo yêu cầu người sử dụng mà có nhiều cách nhìn nhận khác nhau về một hệ chuyên gia.

Loại người sử dụng	*Vấn đề đặt ra*
Người quản trị	Tôi có thể dùng nó để làm gì ?
Kỹ thuật viên	Làm cách nào để tôi vận hành nó tốt nhất ?

Nhà nghiên cứu	Làm sao để tôi có thể mở rộng nó ?
Người sử dụng cuối	Nó sẽ giúp tôi cái gì đây ? Nó có rắc rối và tốn kém không ? Nó có đáng tin cậy không ?

I.2. Đặc trưng và ưu điểm của hệ chuyên gia

Có bốn đặc trưng cơ bản của một hệ chuyên gia :

- *Hiệu quả cao* (high performance). Khả năng trả lời với mức độ tinh thông bằng hoặc cao hơn so với chuyên gia (người) trong cùng lĩnh vực.
- *Thời gian trả lời thoả đáng* (adequate response time). Thời gian trả lời hợp lý, bằng hoặc nhanh hơn so với chuyên gia (người) để đi đến cùng một quyết định. Hệ chuyên gia là một hệ thống thời gian thực (real time system).
- *Độ tin cậy cao* (good reliability). Không thể xảy ra sự cố hoặc giảm sút độ tin cậy khi sử dụng.
- *Dễ hiểu* (understandable). Hệ chuyên gia giải thích các bước suy luận một cách dễ hiểu và nhất quán, không giống như cách trả lời bí ẩn của các hộp đen (black box).

Những ưu điểm của hệ chuyên gia :

- *Phổ cập* (increased availability). Là sản phẩm chuyên gia, được phát triển không ngừng với hiệu quả sử dụng không thể phủ nhận.
- *Giảm giá thành* (reduced cost).
- *Giảm rủi ro* (reduced dangers). Giúp con người tránh được trong các môi trường rủi ro, nguy hiểm.
- *Tính thường trực* (Permanance). Bất kể lúc nào cũng có thể khai thác sử dụng, trong khi con người có thể mệt mỏi, nghỉ ngơi hay vắng mặt.
- *Đa lĩnh vực* (multiple expertise). chuyên gia về nhiều lĩnh vực khác nhau và được khai thác đồng thời bất kể thời gian sử dụng.
- *Độ tin cậy* (increased relialility). Luôn đảm bảo độ tin cậy khi khai thác.
- *Khả năng giảng giải* (explanation). Câu trả lời với mức độ tinh thông được giảng giải rõ ràng chi tiết, dễ hiểu.
- *Khả năng trả lời* (fast reponse). Trả lời theo thời gian thực, khách quan.
- *Tính ổn định, suy luận có lý và đầy đủ mọi lúc mọi nơi* (steady, une motional, and complete response at all times).
- *Trợ giúp thông minh như một người hướng dẫn* (intelligent -tutor).
- *Có thể truy cập như là một cơ sở dữ liệu thông minh* (intelligent database).

I.3. Sự phát triển của công nghệ hệ chuyên gia

Sau đây là một số sự kiện quan trọng trong lịch sử phát triển của công nghệ hệ chuyên gia (expert system technology).

Năm	*Các sự kiện*
1943	Dịch vụ bưu điện ; mô hình Neuron của (Mc Culloch and Pitts Model)
1954	Thuật toán Markov (Markov Algorithm) điều khiển thực thi các luật
1956	Hội thảo Dartmouth ; lý luận logic ; tìm kiếm nghiệm suy (heuristic search) ; thống

	nhất thuật *ngữ trí tuệ nhân tạo* (AI: Artificial Intelligence)
1957	Rosenblatt phát minh khả năng nhận thức ; Newell, Shaw và Simon đề xuất giải bài toán tổng quát (GPS: General Problem Solver)
1958	Mc Carthy đề xuất ngôn ngữ trí tuệ nhân tạo LISA (LISA AI language)
1962	Nguyên lý Rosenblatt's về chức năng thần kinh trong nhận thức (Rosenblatt's *Principles of Neurodynamicdynamics* on Perceptions)
1965	Phương pháp hợp giải Robinson. Ứng dụng logic mờ (fuzzy logic) trong suy luận về các đối tượng mờ (fuzzy object) của Zadeh. Xây dựng hệ chuyên gia đầu tiên về nha khoa DENDRAL (Feigenbaum , Buchanan , et.al)
1968	Mạng ngữ nghĩa (semantic nets), mô hình bộ nhớ kết hợp (associative memory model) của Quillian
1969	Hệ chuyên gia về Toán học MACSYMA (Martin and Moses)
1970	Ứng dụng ngôn ngữ PROLOG (Colmerauer, Roussell, et, al.)
1971	Hệ chuyên gia HEARSAY I về nhận dạng tiếng nói (speech recognition). Xây dựng các luật giải bài toán con người (*Human Problem Solving* popularizes rules (Newell and Simon)
1973	Hệ chuyên gia MYCIN về chẩn trị y học (Shortliffe, et,al.)
1975	Lý thuyết khung (frames), biểu diễn tri thức (knowledge representation) (Minsky)
1976	Toán nhân tạo (AM: Artificial Mathematician) (Lenat). Lý thuyết Dempster–Shafer về tính hiển nhiên của lập luận không chắc chắn (Dempster–Shafer theory of Evidence for reason under uncertainty). Ứng dụng hệ chuyên gia PROSPECTOR trong khai thác hầm mỏ (Duda, Har)
1977	Sử dụng ngôn ngữ chuyên gia OPS (OPS expert system shell) trong hệ chuyên gia XCON/R1 (Forgy)
1978	Hệ chuyên gia XCON/R1 (McDermott, DEC) để bảo trì hệ thống máy tính DEC (DEC computer systems)
1979	Thuật toán mạng về so khớp nhanh (rete algorithm for fast pattern matching) của Forgy ; thương mại hoá các ứng dụng về trí tuệ nhân tạo
1980	Ký hiệu học (symbolics), xây dựng các máy LISP (LISP machines) từ LMI.
1982	Hệ chuyên gia về Toán học (SMP math expert system) ; mạng nơ-ron Hopfield (Hopfield Neural Net) ; Dự án xây dựng máy tính thông minh thế hệ 5 ở Nhật bản (Japanese Fifth Generation Project to develop intelligent computers)
1983	Bộ công cụ phục vụ hệ chuyên gia KEE (KEE expert system tool) (intelli Corp)
1985	Bộ công cụ phục vụ hệ chuyên gia CLIPS (CLIPS expert system tool (NASA)

I.4. Các lĩnh vực ứng dụng của hệ chuyên gia

Cho đến nay, hàng trăm hệ chuyên gia đã được xây dựng và đã được báo cáo thường xuyên trong các tạp chí, sách, báo và hội thảo khoa học. Ngoài ra còn các hệ chuyên gia được sử dụng trong các công ty, các tổ chức quân sự mà không được công bố vì lý do bảo mật. Bảng dưới đây liệt kê một số lĩnh vực ứng dụng diện rộng của các hệ chuyên gia.

Lĩnh vực	*Ứng dụng diện rộng*
Cấu hình (Configuration)	Tập hợp thích đáng những thành phần của một hệ thống theo cách riêng
Chẩn đoán (Diagnosis)	Lập luận dựa trên những chứng cứ quan sát được
Truyền đạt	Dạy học kiểu thông minh sao cho sinh viên có thể hỏi

(Instruction)	*vì sao* (*why*?), *như thế nào* (*how*?) và *cái gì nếu* (*what if*?) giống như hỏi một người thầy giáo
Giải thích (Interpretation)	Giải thích những dữ liệu thu nhận được
Kiểm tra (Monitoring)	So sánh dữ liệu thu lượm được với dữ liệu chuyên môn để đánh giá hiệu quả
Lập kế hoạch (Planning)	Lập kế hoạch sản xuất theo yêu cầu
Dự đoán (Prognosis)	Dự đoán hậu quả từ một tình huống xảy ra
Chữa trị (Remedy)	Chỉ định cách thụ lý một vấn đề
Điều khiển (Control)	Điều khiển một quá trình, đòi hỏi diễn giải, chẩn đoán, kiểm tra, lập kế hoạch, dự đoán và chữa trị

Sau đây là một số hệ chuyên gia (xem thêm phần phụ lục C cuối giáo trình) :

Bảng 1 Ngành hoá học (Chemistry)	
CRYSALIS	Interpret a protein'n 3-D structure
DENDRAL	Interpret molecular structure
TQMSTUNE	Remedy Triple Quadruple Mass Spectrometer (keep it tuned)
CLONER	Design new biological molecules
MOLGEN	Design gene - cloning experiments
SECS	Design complex organic molecules
SPEX	Plan molecular biology experiments
Bảng 2 Ngành điện tử (Electronics)	
ACE	Diagnosis telephone network faults
IN -ATE	Diagnosis oscilloscope faults
NDS	Diagnosis national communication net
EURISKO	Design 3-D micro-electronics
PALLADIO	Design and test new VLSI cicuits
REDESIGN	Redesign digital circuits to new
CADHELP	Instruct for computer aided design
SOPHIE	Instruct circuit fault diagnosis
Bảng 3 Ngành địa chất (Geology)	
DIPMETER	Interpret dipmeter logs
LITHO	Interpret oil well log data
MUD	Diagnosis / remedy drilling problems
PROSPECTOR	Interpret geologic data for minerals
Bảng 4 Công nghệ (Engineering)	
REACTOR	Diagnosis / remedy reactor accidents
DELTA	Diagnosis / remedy GE locomotives
STEAMER	Instruct operation - steam power-plant

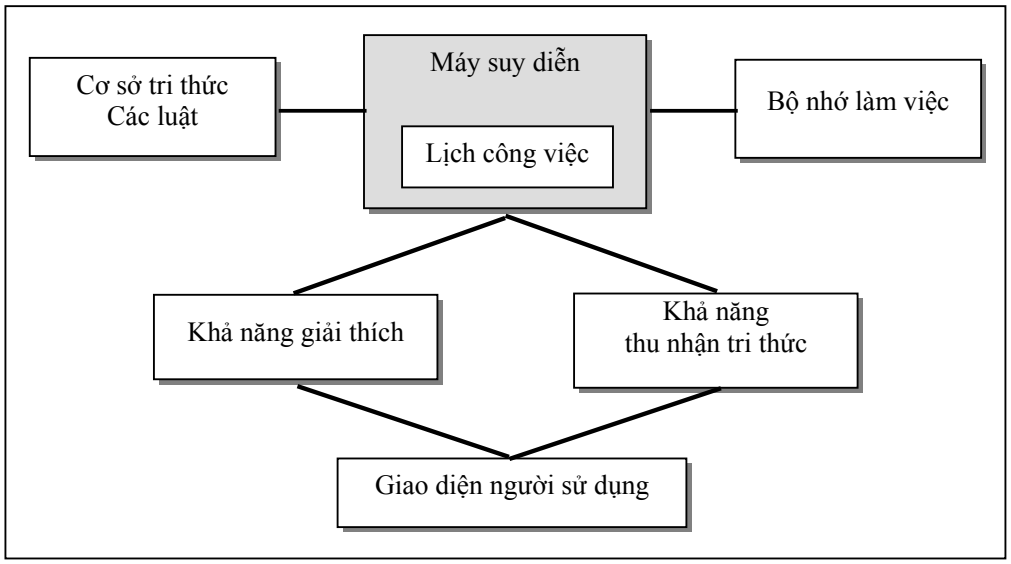
Bảng 5 Ngành y học (Medicine)	
PUFF	Diagnosis lung disease
VM	Monitors intensive - care patients
ABEL	Diagnosis acid - base / electrolytes
AI/COAG	Dianosis blood disease

AI/ RHEUM	Diagnosis rheumatoid disease
CADUCEUS	Diagnosis internal medicine disease
ANNA	Monitor digitalis therapy
BLUE BOX	Diagnosis / remedy depression
MYCIN	Diagnosis / remedy bacterial infections
ONCOCIN	Remedy / manage chemotherapy patient
ATTENDING	Instruct in anesthetic manegement
GUIDON	Instruct in bacterial infections
Bảng 6 Máy tính điện tử (Computer systems)	
PTRANS	Prognosis for managing DEC computers
BDS	Diagnosis bad parts in switching net
XCON	Configune DEC computer systems
XSEL	Configure DEC computer sales order
XSITE	Configure customer site for DEC computers
YES/MVS	Monitor / control IBM MVS opeating system
TIMM	Diagnosis DEC computer

II. Kiến trúc tổng quát của các hệ chuyên gia

II.1. Những thành phần cơ bản của một hệ chuyên gia

Một hệ chuyên gia kiểu mẫu gồm bảy thành phần cơ bản như sau :



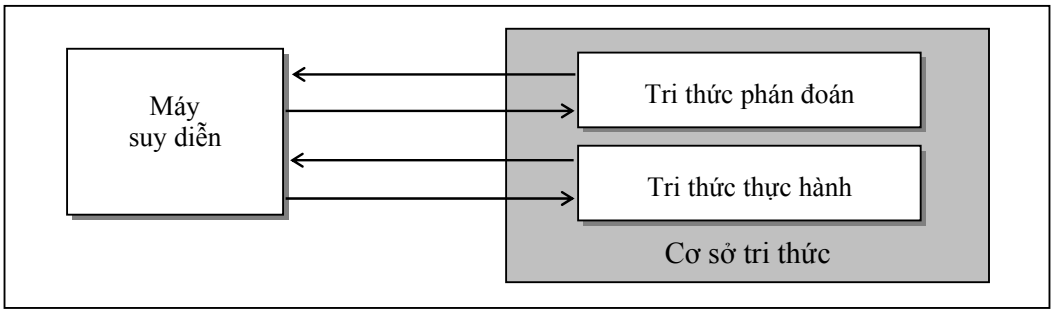
Hình 1.4. Những thành phần cơ bản của một hệ chuyên gia

- *Cơ sở tri thức* (knowledge base). Gồm các phần tử (hay đơn vị) tri thức, thông thường được gọi là *luật* (rule), được tổ chức như một cơ sở dữ liệu.
- *Máy duy diễn* (inference engine). Công cụ (chương trình, hay bộ xử lý) tạo ra sự suy luận bằng cách quyết định xem những luật nào sẽ làm thỏa mãn các sự kiện, các đối tượng. , chọn ưu tiên các luật thỏa mãn, thực hiện các luật có tính ưu tiên cao nhất.
- *Lịch công việc* (agenda). Danh sách các luật ưu tiên do máy suy diễn tạo ra thoả mãn các sự kiện, các đối tượng có mặt trong bộ nhớ làm việc.

- *Bộ nhớ làm việc* (working memory). Cơ sở dữ liệu toàn cục chứa các sự kiện phục vụ cho các luật.
- *Khả năng giải thích* (explanation facility). Giải nghĩa cách lập luận của hệ thống cho người sử dụng.
- *Khả năng thu nhận tri thức* (explanation facility). Cho phép người sử dụng bổ sung các tri thức vào hệ thống một cách tự động thay vì tiếp nhận tri thức bằng cách mã hoá tri thức một cách tường minh. Khả năng thu nhận tri thức là yếu tố mặc nhiên của nhiều hệ chuyên gia.
- *Giao diện người sử dụng* (user interface). Là nơi người sử dụng và hệ chuyên gia trao đổi với nhau.

Cơ sở tri thức còn được gọi là *bộ nhớ sản xuất* (production memeory) trong hệ chuyên gia. Trong một cơ sở tri thức, người ta thường phân biệt hai loại tri thức là *tri thức phán đoán* (assertion knowledge) và *tri thức thực hành* (operating knowledge).

Các tri thức phán đoán mô tả các tình huống đã được thiết lập hoặc sẽ được thiết lập. Các tri thức thực hành thể hiện những hậu quả rút ra hay những thao tác cần phải hoàn thiện khi một tình huống đã được thiết lập hoặc sẽ được thiết lập trong lĩnh vực đang xét. Các tri thức thực hành thường được thể hiện bởi các biểu thức dễ hiểu và dễ triển khai thao tác đối với người sử dụng.



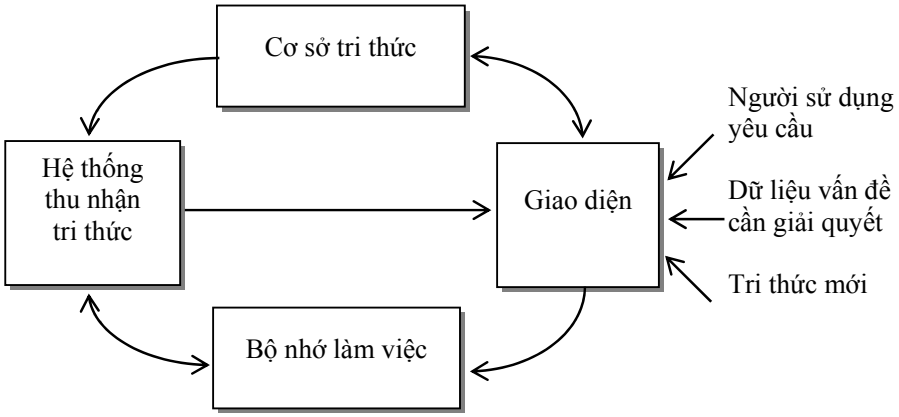
Hình 1.5. Quan hệ giữa máy suy diễn và cơ sở tri thức

Từ việc phân biệt hai loại tri thức, người ta nói máy suy diễn là công cụ triển khai các cơ chế (hay kỹ thuật) tổng quát để tổ hợp các tri thức phán đoán và các tri thức thực hành. Hình trên đây mô tả quan hệ hữu cơ giữa máy suy diễn và cơ sở tri thức.

II.2. Một số mô hình kiến trúc hệ chuyên gia

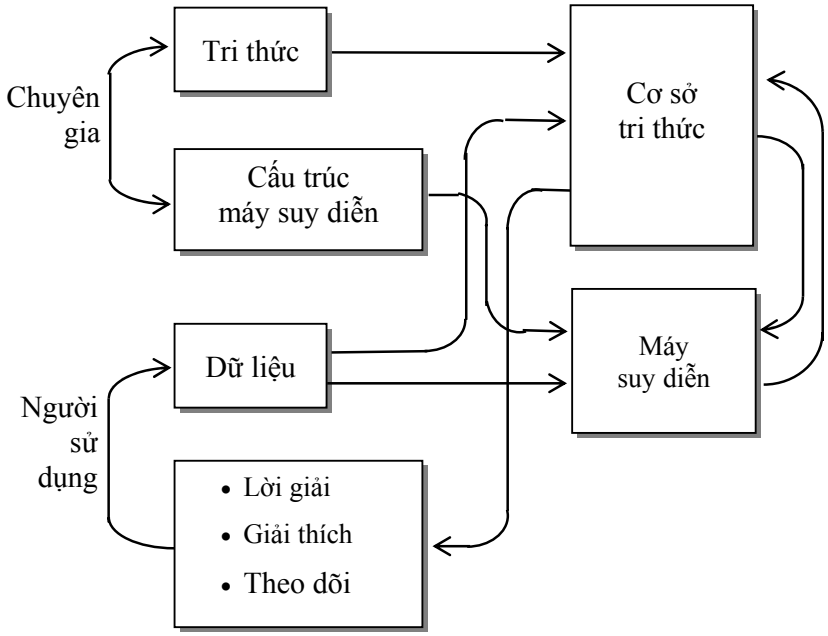
Có nhiều mô hình kiến trúc hệ chuyên gia theo các tác giả khác nhau. Sau đây là một số mô hình.

a. Mô hình J. L. Ermine



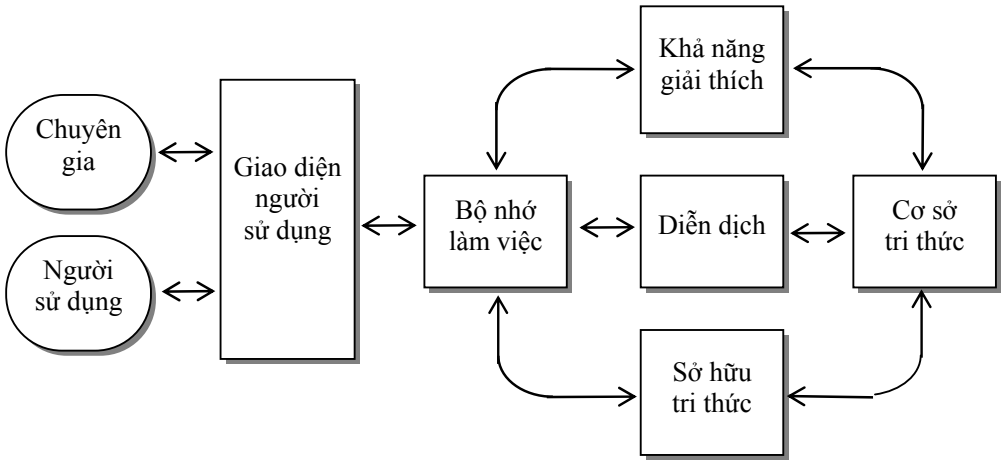
Hình 1.6. Kiến trúc hệ chuyên gia theo J. L. Ermine

b. Mô hình C. Ernest



Hình 1.7. Kiến trúc hệ chuyên gia theo C. Ernest

c. Mô hình E. V. Popov



Hình 1.8. Kiến trúc hệ chuyên gia theo E. V. Popov

II.3. Biểu diễn tri thức trong các hệ chuyên gia

Tri thức của một hệ chuyên gia có thể được biểu diễn theo nhiều cách khác nhau. Thông thường người ta sử dụng các cách sau đây :

- Biểu diễn tri thức bởi các luật sản xuất
- Biểu diễn tri thức nhờ mệnh đề logic
- Biểu diễn tri thức nhờ mạng ngữ nghĩa
- Biểu diễn tri thức nhờ ngôn ngữ nhân tạo

Ngoài ra, người ta còn sử dụng cách biểu diễn tri thức nhờ các sự kiện không chắc chắn, nhờ bộ ba : đối tượng, thuộc tính và giá trị (O-A-V: Object-Attribute-Value), nhờ khung (frame), v.v... Tuỳ theo từng hệ chuyên gia, người ta có thể sử dụng một cách hoặc đồng thời cả nhiều cách.

II.3.1. Biểu diễn tri thức bởi các luật sản xuất

Hiện nay, hầu hết các hệ chuyên gia đều là các hệ thống dựa trên luật, bởi lý do như sau :

- *Bản chất đơn thể* (modular nature). Có thể đóng gói tri thức và mở rộng hệ chuyên gia một cách dễ dàng.
- *Khả năng diễn giải dễ dàng* (explanation facilities). Dễ dàng dùng luật để diễn giải vấn đề nhờ các tiền đề đặc tả chính xác các yếu tố vận dụng luật, từ đó rút ra được kết quả.
- *Tương tự quá trình nhận thức của con người.* Dựa trên các công trình của Newell và Simon, các luật được xây dựng từ cách con người giải quyết vấn đề. Cách biểu diễn luật nhờ IF THEN đơn giản cho phép giải thích dễ dàng cấu trúc tri thức cần trích lọc.

Luật là một kiểu sản xuất được nghiên cứu từ những năm 1940. Trong một hệ thống dựa trên luật, công cụ suy luận sẽ xác định những luật nào là tiên đề thỏa mãn các sự việc.

Các luật sản xuất thường được viết dưới dạng IF THEN. Có hai dạng :

IF < điều kiện > THEN < hành động >

hoặc

IF < điều kiện > THEN < kết luận > DO < hành động >

Tuỳ theo hệ chuyên gia cụ thể mà mỗi luật có thể được đặt tên. Chẳng hạn mỗi luật có dạng *Rule: tên*. Sau phần tên là phần IF của luật.

Phần giữa **I**F và THEN là *phần trái luật* (LHS: Left - Hand -Side), có nội dung được gọi theo nhiều tên khác nhau, như *tiền đề* (antecedent), *điều kiện* (conditional part), *mẫu so khớp* (pattern part),

Phần sau THEN là kết luận hay *hậu quả* (consequent). Một số hệ chuyên gia có thêm phần hành động (action) được gọi là *phần phải luật* (RHS: Right - Hand -Side).

Ví dụ :

Rule: Đèn đỏ
IF
Đèn đỏ sáng
THEN
Dừng

Rule: Đèn-xanh
IF
Đèn xanh sáng
THEN
Đi

Trong ví dụ trên, Đèn đỏ sáng và Đèn xanh sáng là những điều kiện, hay những khuôn mẫu. Sau đây là một số ví dụ khác :

Rule: Điều trị sốt
IF
Bệnh nhân sốt
THEN
cho uống thuốc Aspirin

Hệ thống chẩn đoán xe máy (OPS5)
IF
Máy xe không nổ khi khởi động
THEN
Dự đoán: Xe bị panne sức nén. Pittong, bạc xéc-măng và lòng xy lanh sai tiêu chuẩn, dễ tạo thành những khe hở nhỏ làm cho pittong không còn kín nên hoà khí không được nén lên đầy đủ. Xử lý : nên điều chỉnh hoặc thay mới pittong, bạc xéc-măng và lòng xy lanh cho đúng tiêu chuẩn
IF
máy xe nổ không ổn định, OR
máy xe nổ rồi lại tắt, AND
bugi khô
THEN
Dự đoán : Xe đã bị nghẹt xăng. Xử lý : nên xúc rửa bình xăng và bộ khoá xăng của xe.

MYCIN hệ thống chẩn đoán bệnh viêm màng não và hiện tượng có vi khuẩn bất thường trong máu (nhiễm trùng)
IF
Tại vị trí vết thương có máu, AND
Chưa biết chắc chắn cơ quan bị tổn thương, AND

Chất nhuộm màu âm tính, AND
Vi khuẩn có dạng hình que, AND
Bệnh nhân bị sốt cao

THEN
Cơ quan có triệu chứng (0.4) nhiễm trùng

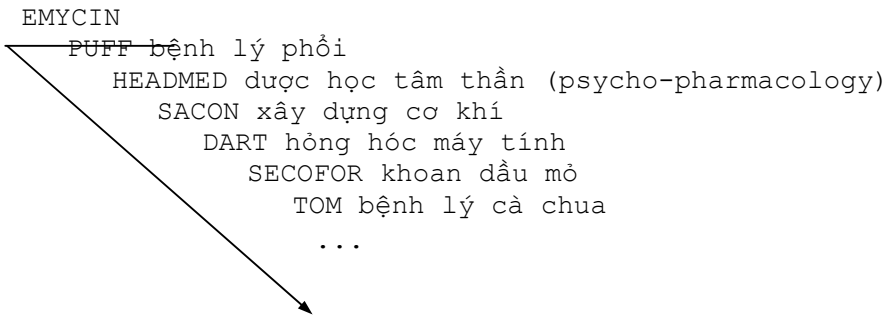
II.3.2. Bộ sinh của hệ chuyên gia

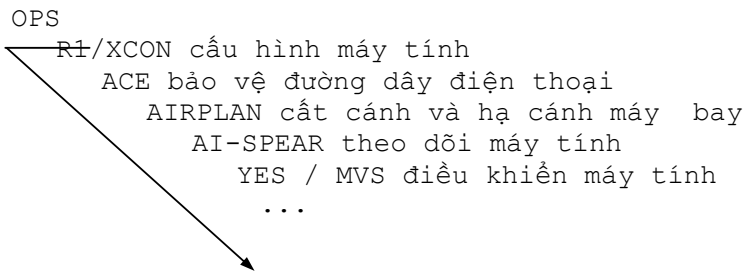
Bộ sinh của hệ chuyên gia (expert-system generator) là hợp của :
– một *máy suy diễn*,
– một *ngôn ngữ* thể hiện tri thức (bên ngoài)
– và một *tập hợp các cấu trúc và các quy ước* thể hiện các tri thức (bên trong).

Theo cách nào đó, các cấu trúc và các quy ước này xác định một cơ sở tri thức rỗng (hay rỗng bộ phận). Nhờ các tri thức chuyên môn để định nghĩa một hệ chuyên gia, người ta đã tạo ra bộ sinh để làm đầy cơ sở tri thức.

Chẳng hạn, EMYCIN là tên của bộ sinh của hệ chuyên gia MYCIN và được tiếp tục áp dụng cho một số lĩnh vực.

Hệ chuyên gia R1 được xây dựng từ bộ sinh OPS (là hệ thống luật được phát triển bởi Charles Forgy năm 1975 tại Carnegie-Mellon University). Sau đây là một số hậu duệ của EMYCIN và OPS :





Nhờ bộ sinh, mỗi hệ hệ chuyên gia có thể chứa từ hàng trăm đến hàng ngàn luật. Bảng dưới đây thống kê số luật của một số hệ chuyên gia :

Hệ chuyên gia	*Lĩnh vực*	*Năm xuất hiện*	*Số luật*
MYCIN	Y học	1974	500
PROSPECTOR	Địa chất	1979	1 600
R1/XCON	Tin học	1980	> 7 000
LITHO	Địa chất	1982	500
SPHINX	Y học	1984	400
TOM	Nông học	1984	200

Một trong những nét hấp dẫn của tiếp cận hệ chuyên gia là khả năng «học» (learn) của hệ thống nhằm thường xuyên sửa đổi và hoàn thiện cơ sở tri thức vốn có. Sơ đồ dưới đây cho biết sự tiến triển của hai hệ chuyên gia nổi tiếng của Mỹ là MYCIN và R1 :

MYCIN	1974	:	200	luật	hiện nay	:	500 luật
R1	1980	:	800				
	1981	:	1 000				
	1982	:	1 500				
	1983	:	2 000				
	1984	:	> 3 000				
	1985	:	> 7 000				

II.3.3. «Soạn thảo kết hợp» các luật

Nói chung, tuỳ theo hệ chuyên gia mà những quy ước để tạo ra luật cũng khác nhau. Sự giống nhau cơ bản giữa các hệ chuyên gia về mặt ngôn ngữ là cách *soạn thảo kết hợp* (associative writing) các luật.

Ở đây, thuật ngữ *soạn thảo kết hợp* được chọn để gợi lên khái niệm về chế độ *truy cập kết hợp* (associative access) liên quan đến chế độ *lưu trữ kết hợp* (associative memory) là chế độ mà thông tin cần tìm kiếm được đọc không chỉ căn cứ vào địa chỉ đơn vị nhớ cụ thể mà còn căn cứ vào một phần nội dung của thông tin cần tìm kiếm chứa trong đó.

Soạn thảo kết hợp các luật gồm những quy ước như sau :

1. Mỗi luật do chuyên gia cung cấp phải định nghĩa được các *điều kiện khởi động* (tác nhân) hay *tiền đề* của luật, nghĩa là các tình huống (được xác định bởi các quan hệ trên tập hợp dữ liệu đã cho) và *hậu quả* của luật, để luật này có thể áp dụng.

Theo cách dùng thông thường, người ta đặt tên riêng cho luật để chọn áp dụng, hoặc cung cấp một nhóm các *sự kiện* (fact) tương thích với điều kiện khởi động của luật.

2. Trong luật, không bao giờ người ta chỉ định một luật khác bởi tên riêng. Ví dụ : luật R sau đây tuân thủ hai đặc trưng :
IF *bệnh nhân sốt* AND *tốc độ lắng huyết cầu trong máu tăng lên*
THEN *bệnh nhân nhiễm bệnh virut*

Từ nội dung luật R, người ta có thể vận dụng như sau :

- Khi xảy ra tình huống bệnh nhân bị sốt và tốc độ lắng huyết cầu trong máu tăng lên, thì "*bệnh nhân sốt*" và "*tốc độ lắng huyết cầu trong máu tăng lên*" là những điều kiện để khởi động luật. Hậu quả của luật là "*bệnh nhân nhiễm bệnh virut*". Như vậy, việc áp dụng luật sẽ dẫn đến một sự kiện mới được thiết lập từ đây trở đi : "*bệnh nhân nhiễm bệnh virut*".
- Khi muốn tạo sự kiện "bệnh nhân bị nhiễm bệnh virut", thì điều kiện khởi động luật là "*bệnh nhân nhiễm bệnh virut*". Hậu quả của luật sẽ là "*bệnh nhân sốt*" và "*tốc độ lắng huyết cầu trong máu tăng lên*". Từ đây, luật sẽ khởi động các sự kiện mới vừa được thiết lập "*bệnh nhân sốt*" và "*tốc độ lắng huyết cầu trong máu tăng lên*".

Cách biểu diễn các điều kiện khởi động trong luật phù hợp với cách tư duy tự nhiên của các chuyên gia. Do vậy, người ta dễ dàng thể hiện cũng như sửa đổi các tri thức tiếp nhận.

Như vậy, người ta không nhất thiết phải đặt tên cho luật để có thể gọi đến khi cần, mà có thể khai thác thông tin từ các điều kiện khởi động của luật. Chẳng hạn từ luật R trên đây :

- Nếu tìm được các luật có khả năng thiết lập sự kiện "*bệnh nhân nhiễm bệnh virut*", người ta sẽ để ý đến phần **then** của chúng như là các điều kiện khởi động. Luật R là một trong các luật có điều kiện khởi động tương ứng với lời gọi "*bệnh nhân nhiễm bệnh virut*".

– Nếu tìm được các luật có khả năng đưa ra sự kiện "*bệnh nhân sốt*", chỉ cần để ý đến phần **if** của chúng như là các điều kiện khởi động. Luật R là một trong các luật có điều kiện khởi động tương ứng với lời gọi "*bệnh nhân sốt*".

Việc so sánh giữa điều kiện khởi động các luật và các sự kiện được xét tại một thời điểm đã cho (tuỳ theo trường hợp, các sự kiện giả sử đã được thiết lập hay sẽ thiết lập) cho phép *lọc* (filter) các luật để giữ lại một số luật nào đó. Phần điều kiện khởi động của luật thường được gọi là *bộ lọc*, hay mẫu so khớp của luật đó.

Trong Tin học cổ điển, mỗi thủ tục (đóng vai trò là một đơn vị tri thức) thường được xác định và được gọi bởi tên của thủ tục. Lúc này, nếu muốn thêm vào hay lấy ra một thủ tục, người ta cần dự kiến các thay đổi trong toàn bộ thủ tục khác sử dụng đến thủ tục muốn thêm vào hay lấy ra này.

Ngược lại, về nguyên tắc, việc soạn thảo kết hợp cho phép tạo ra một luật mà không cần để ý đến sự hiện diện của các luật khác. Với mỗi luật, dù là của ai, một khi được đưa vào trong cơ sở tri thức, thì chỉ cần để ý đến các biểu thức *điều kiện* để xác định nếu luật đó là áp dụng được và do vậy, có thể gọi tới nó hay không. Người ta cũng xem rằng các sự kiện được đưa vào như là hậu quả của một luật có thể giúp để gọi đến các luật khác nhờ các bộ lọc của chúng.

Như vậy, phương pháp soạn thảo kết hợp cho phép bổ sung và loại bỏ dễ dàng các luật mà không cần xem xét hậu quả của việc bổ sung và loại bỏ đó. Phương pháp soạn thảo kết hợp có vị trí quan trọng trong các hệ thống dựa trên luật của các hệ chuyên gia. Đó là *các hệ thống suy diễn định hướng bởi các bộ lọc* (PDISPattern-Directed Inference Systems).

II.3.4. Các phương pháp biểu diễn tri thức khác

a. Biểu diễn tri thức nhờ mệnh đề logic

Người ta sử dụng các ký hiệu để thể hiện tri thức và các phép toán lôgic tác động lên các ký hiệu để thể hiện suy luận lôgic. Kỹ thuật chủ yếu thường được sử dụng là lôgic vị từ (predicate logic) mà ta sẽ đề cập đến ở chương sau.

Các ví dụ dưới đây minh hoạ cách thể hiện các phát biểu (cột bên trái) dưới dạng vị từ (cột bên phải) :

Phát biểu	*Vị từ*
Tom là đàn ông	MAN(tom)
Tom là cha của Mary	FATHER(tom, mary)
Tất cả mọi người đều chết	MAN(X) → MORTAL(X) với quy ước MAN(X) có nghĩa «X là một người» và MORTAL(X) có nghĩa «X chết». MAN và MORTAL được gọi là các vị từ đối với biến X.

Các vị từ thường có chứa hằng, biến hay hàm. Người ta gọi các vị từ không chứa biến (có thể chứa hằng) là các *mệnh đề* (preposition). Mỗi vị từ có thể là một *sự kiện* (fact) hay một luật. Luật là vị từ gồm hai vế trái và phải được nối nhau bởi một dấu mũi tên (→). Các vị từ còn lại (không chứa mũi tên) được gọi là các sự kiện. Trong ví dụ trên đây, MAN và FATHER là các mệnh đề và là các sự kiện. Còn MAN(X) → MORTAL(X) là một luật.

Ví dụ : Từ các tri thức sau :

Marc có tóc vàng hoe, còn Jean có tóc màu nâu. Pierre là cha của Jean. Marc là cha của Pierre. Jean là cha của René. Marc là con của Georges.

Giả sử X, Y và là Z những người nào đó, nếu Y là con của X thì X là cha của Y. Nếu X là cha của Z và Z là cha của Y thì X là ông của Y.

ta có thể biểu diễn thành các sự kiện và các luật như sau :

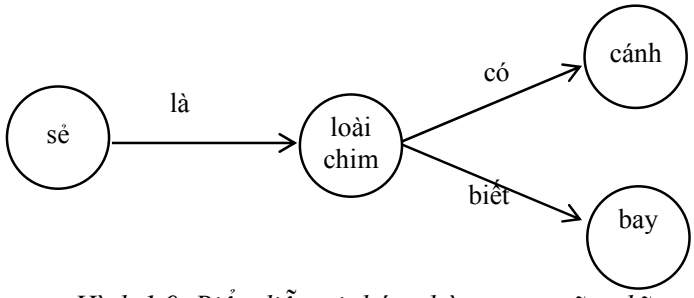
1. BLOND (marc)
2. BROWN (jean)
3. FATHER (pierre, jean)
4. FATHER (marc, pierre)
5. FATHER (jean, rené)
6. SON (marc, georges)
7. FATHER (X, Y) ← SON (Y, X)
8. GRANDFATHER (X, Y) ← FATHER (X, Z), FATHER (Z, Y)

Người ta gọi tập hợp các sự kiện và các luật là một cơ sở tri thức.

b. Biểu diễn tri thức nhờ mạng ngữ nghĩa

Trong phương pháp này, người ta sử dụng một đồ thị gồm các *nút* (node) và các *cung* (arc) nối các nút để biểu diễn tri thức. Nút dùng để thể hiện các đối tượng, thuộc tính của đối tượng và giá trị của thuộc tính. Còn cung dùng để thể hiện các quan hệ giữa các đối tượng. Các nút và các cung đều được gắn nhãn.

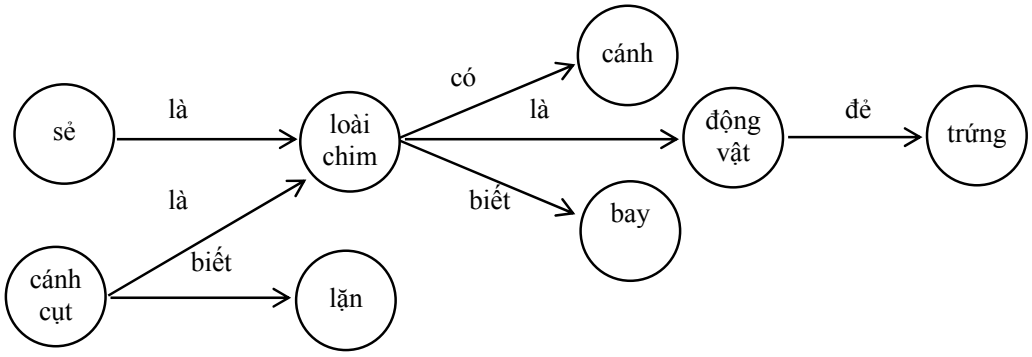
Ví dụ để thể hiện tri thức "*sẻ là một loài chim có cánh và biết bay*", người ta vẽ một đồ thị như sau :



Hình 1.9. Biểu diễn tri thức nhờ mạng ngữ nghĩa

Bằng cách thêm vào đồ thị các nút mới và các cung mới, người ta có thể mở rộng một mạng ngữ nghĩa. Các nút mới được thêm thể hiện các đối tượng tương tự (với các nút đã có trong đồ thị), hoặc tổng quát hơn. Chẳng hạn để thể hiện "*chim là một loài động vật đẻ trứng*" và "*cánh cụt là loài chim biết lặn*", người ta vẽ thêm như sau :

Một trong những tính chất quan trọng của mạng ngữ nghĩa là tính thừa kế. Khi sử dụng mạng ngữ nghĩa để biểu diễn tri thức, người ta phải xây dựng các phép toán tương ứng.



Hình 1.10. Mở rộng mạng ngữ nghĩa biểu diễn tri thức

c. Biểu diễn tri thức nhờ ngôn ngữ nhân tạo

Nói chung, theo quan điểm của người sử dụng, ngôn ngữ tự nhiên sẽ là phương cách thuận tiện nhất để giao tiếp với một hệ chuyên gia, không những đối với người quản trị hệ thống (tư cách chuyên gia), mà còn đối với người sử dụng cuối. Hiện nay đã có những hệ chuyên gia có khả năng đối thoại trên ngôn ngữ tự nhiên (thông thường là tiếng Anh) nhưng chỉ hạn chế trong lĩnh vực ứng dụng chuyên môn của hệ chuyên gia

Hình dưới đây thể hiện một đơn vị tri thức (luật) trong hệ chuyên gia MYCIN dùng để chẩn đoán các bệnh virut. Cột bên trái là một luật được viết bằng tiếng Anh, cột bên phải là *mã hoá nhân tạo* của luật đó.

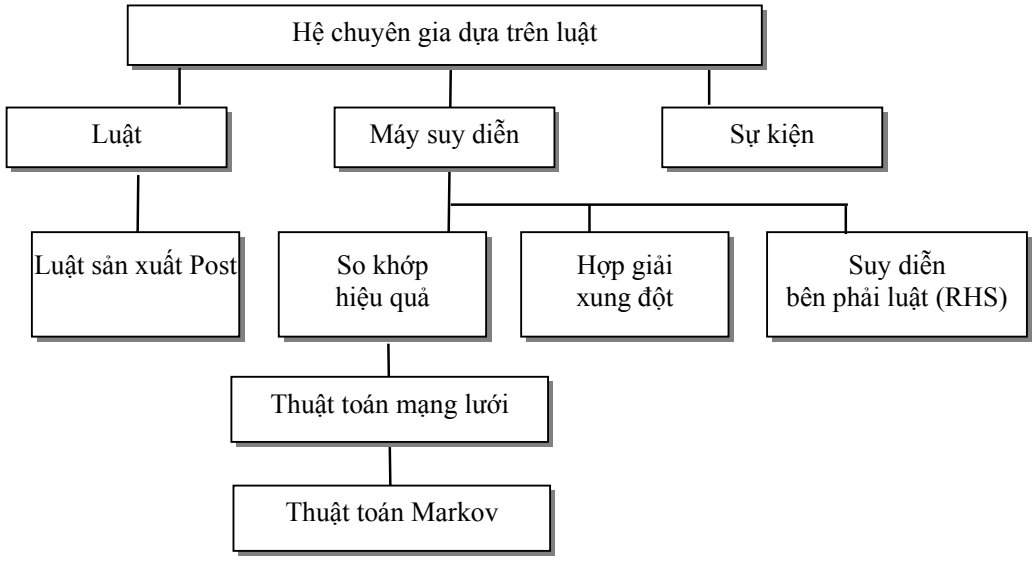
Nếu 1) Màu của cơ thể là gram dương và nếu 2) Hình thái của cơ thể là bị nhiễm trùng và nếu 3) Kiểu phát triển của cơ thể là khuẩn lạc thì tồn tại một khả năng (0.7) là cơ thể bị nhiễm khuẩn cầu chùm	(($AND (SAME CNTXT GRAM GRAM+) (SAME CNTXT MORPH COCCI) (SAME CNTXT DEVEL COLONY) (CONCLUDE CNTXT IDENT STAPHYLOCOCCUS MEASURE 0.7))

Hình 1.11. Biểu diễn tri thức nhờ ngôn ngữ nhân tạo trong MYCIN

II.4. Kỹ thuật suy luận trong các hệ chuyên gia

Có nhiều phương pháp tổng quát để suy luận trong các chiến lược giải quyết vấn đề của hệ chuyên gia. Những phương pháp hay gặp là *suy diễn tiến* (foward chaining), *suy diễn lùi* (backward chaining) và phối hợp hai phương pháp này (mixed chaining). Những phương pháp khác là *phân tích phương tiện* (means-end analysis), *rút gọn vấn đề* (problem reduction), *quay lui* (backtracking), *kiểm tra lập kế hoạch* (plan-generate-test), *lập kế hoạch phân cấp* (hierachical planning)...

Dưới đây là nền tảng của công nghệ hệ chuyên gia hiện đại (foundation of modern rele-based expert system).



Hình 1.12. Nền tảng của công nghệ hệ chuyên gia dựa trên luật hiện đại

II.4.1. Phương pháp suy diễn tiến

Suy diễn tiến (forward charning) là lập luận từ các sự kiện, sự việc để rút ra các kết luận. Ví dụ : Nếu thấy trời mưa trước khi ra khỏi nhà (sự kiện) thì phải lấy áo mưa (kết luận).

Trong phương pháp này, người sử dụng cung cấp các sự kiện cho hệ chuyên gia để hệ thống (máy suy diễn) tìm cách rút ra các kết luận có thể. Kết luận được xem là những thuộc tính có thể được gán giá trị. Trong số những kết luận này, có thể có những kết luận làm người sử dụng quan tâm, một số khác không nói lên điều gì, một số khác có thể vắng mặt.

Các sự kiện thường có dạng :

Atthibute = value

Lần lượt các sự kiện trong cơ sở tri thức được chọn và hệ thống xem xét tất cả các luật mà các sự kiện này xuất hiện như là tiền đề. Theo nguyên tắc lập luận trên, hệ thống sẽ lấy ra những luật thoã mãn. Sau khi gán giá trị cho các thuộc tính thuộc kết luận tương ứng, người ta nói rằng các sự kiện đã được thoã mãn. Các thuộc tính được gán giá trị sẽ là một phần của kết quả chuyên gia. Sau khi mọi sự kiện đã được xem xét, kết quả được xuất ra cho người sử dụng.

II.4.2. Phương pháp suy diễn lùi

Phương pháp suy diễn lùi tiến hành các lập luận theo chiều ngược lại (đối với phương pháp suy diễn tiến). Từ một giả thuyết (như là một kết luận), hệ thống đưa ra một tình huống trả lời gồm các sự kiện là cơ sở của giả thuyết đã cho này.

Ví dụ nếu ai đó vào nhà mà cầm áo mưa và áo quần bị ướt thì giả thuyết này là trời mưa. Để củng cố giả thuyết này, ta sẽ hỏi người đó xem có phải trời mưa không ? Nếu người đó trả lời có thì giả thuyết trời mưa đúng và trở thành một sự kiện. Nghĩa là trời mưa nên phải cầm áo mưa và áo quần bị ướt.

Suy diễn lùi là cho phép nhận được giá trị của một thuộc tính. Đó là câu trả lời cho câu hỏi « *giá trị của thuộc tính A là bao nhiêu ?* » với A là một đích (goal).

Để xác định giá trị của A, cần có các nguồn thông tin. Những nguồn này có thể là những câu hỏi hoặc có thể là những luật. Căn cứ vào các câu hỏi, hệ thống nhận được một cách trực tiếp từ người sử dụng những giá trị của thuộc tính liên quan. Căn cứ vào các luật, hệ thống suy diễn có thể tìm ra giá trị sẽ là kết luận của một trong số các kết luận có thể của thuộc tính liên quan, v.v...

Ý tưởng của thuật toán suy diễn lùi như sau. Với mỗi thuộc tính đã cho, người ta định nghĩa nguồn của nó :

- Nếu thuộc tính xuất hiện như là tiền đề của một luật (phần đầu của luật), thì nguồn sẽ thu gọn thành một câu hỏi.
- Nếu thuộc tính xuất hiện như là hậu quả của một luật (phần cuối của luật), thì nguồn sẽ là các luật mà trong đó, thuộc tính là kết luận.
- Nếu thuộc tính là trung gian, xuất hiện đồng thời như là tiền đề và như là kết luận, khi đó nguồn có thể là các luật, hoặc có thể là các câu hỏi mà chưa được nêu ra.

Nếu mỗi lần với câu hỏi đã cho, người sử dụng trả lời hợp lệ, giá trị trả lời này sẽ được gán cho thuộc tính và xem như thành công. Nếu nguồn là các luật, hệ thống sẽ lấy lần lượt các luật mà thuộc tính đích xuất hiện như kết luận, để có thể tìm giá trị các thuộc tính thuộc tiền đề. Nếu các luật thoã mãn, thuộc tính kết luận sẽ được ghi nhận.

II.4.3. Các hệ thống sản xuất (production systems)

a. Các hệ thống sản xuất Post

Hệ thống sản xuất được Post sử dụng trong *logic ký hiệu* (symbolic logic) từ những năm 1943. Theo ông, rất nhiều hệ thống toán học và logic được viết dưới dạng các luật sản xuất (production rule). Các luật còn được gọi là *quy tắc viết lại* (rewrite rules) thường được dùng để định nghĩa văn phạm của một ngôn ngữ. Các ngôn ngữ lập trình thường được định nghĩa từ dạng Backus - Naur (BNF).

Ý tưởng cơ bản của Post là xuất phát từ một xâu vào (input string), được gọi là *tiền đề* (antecedent), sản xuất ra một xâu kết quả mới khác (consequent). Mỗi sản xuất có dạng :

$$< \textit{xâu tiền đề} > \rightarrow < \textit{xâu kết quả} >$$

Dấu mũi tên $\rightarrow$ chỉ ra rằng xâu vào bên trái được *chuyển* (transformation) thành xâu kết quả bên phải.

Ví dụ :

Để đi qua các ngã ba, ngã tư trong thành phố :

Đèn đỏ sáng $\rightarrow$ *Dừng*
Đèn xanh sáng $\rightarrow$ *Đi*

Để chữa trị bệnh sốt :

Bệnh nhân sốt $\rightarrow$ *Cho uống thuốc Aspirin*

Các luật có thể có nhiều tiền đề :

Bệnh nhân sốt AND *Sốt trên 39* ^0C $\rightarrow$ *Đi khám bác sĩ*

Chú ý phép AND không phải là một phần của xâu mà cho phép nối kết nhiều tiền đề lại với nhau.

Một hệ thống sản xuất Post gồm một nhóm các luật sản xuất, chẳng hạn (chú ý các số thứ tự đặt trong dấu ngoặc chỉ dùng để trình bày) :

(1) *Car won't start* $\rightarrow$ *Check battery*
(2) *Car won't start* $\rightarrow$ *Check gas*
(3) *Check battery* AND *Battery bad* $\rightarrow$ *Replace battery*
(4) *Check gas* AND *No gas* $\rightarrow$ *Fill gas tank*

Nếu đưa vào xâu *Car won't start*, thì các luật (1) và (2) có thể được áp dụng để sinh ra các xâu *Check battery* và *Check gas*. Tuy nhiên, không tồn tại cơ chế để có thể áp dụng đồng thời cả hai cho xâu vào này. Chỉ có thể áp dụng được một luật trong hai, hoặc không. Nếu đưa vào xâu *Battery bad* và *Check battery* thì luật 3 có thể được áp dụng để sinh ra xâu *Replace battery*.

Không đặt ra thứ tự các luật trong hệ thống. Sau khi đảo thứ tự, chẳng hạn (4) (2) (1) (3) thì hệ thống giữ nguyên giá trị :

(4) *Check gas* AND *No gas* $\rightarrow$ *Fill gas tank*
(2) *Car won't start* $\rightarrow$ *Check gas*
(1) *Car won't start* $\rightarrow$ *Check battery*
(3) *Check battery* AND *Battery bad* $\rightarrow$ *Replace battery*

Mặc dù các sản xuất Post được sử dụng trong hệ chuyên gia nhưng chúng không thuận tiện cho việc viết các trình ứng dụng. Hạn chế chủ yếu của các sản xuất Post khi lập trình là không có các *chiến lược điều khiển* (control strategy) để định hướng sử dụng luật... Một hệ thống Post cho phép áp dụng luật cho một xâu vào theo cách tuỳ ý mà không chỉ ra cụ thể

làm thế nào để luật được áp dụng. Chính sự lựa chọn luật một cách ngẫu nhiên như vậy làm thời gian tìm kiếm trở nên đáng kể trong các hệ thống có nhiều luật.

b. Các thuật toán Markov

Để cải tiến việc áp dụng các luật sản xuất, năm 1954, Markov đã đề xuất một cấu trúc điều khiển cho hệ thống sản xuất. Một *thuật toán Markov* (Markov algorithm) là một nhóm các sản xuất có thứ tự được áp dụng theo một thứ tự ưu tiên cho một xâu vào. Nếu luật có ưu tiên cao nhất không được áp dụng, thì qui tắc tiếp theo sẽ được áp dụng và cứ thế tiếp tục. Thuật toán Markov dừng nếu :

(1) sản xuất cuối cùng không được áp dụng cho xâu, hoặc

(2) nếu sản xuất đó là cuối một giai đoạn được áp dụng.

Thuật toán Markov cũng có thể được áp dụng cho một xâu con (substring) của một xâu, bắt đầu từ bên trái :

Ví dụ : Cho luật AB $\rightarrow$ HIJ

Khi đó, áp dụng cho xâu vào GABKAB sẽ tạo ra xâu mới GHIJKAB. Từ đó, ta nhận được tiếp tục xâu mới GHIJKHIJ.

Ký tự đặc biệt ε biểu diễn xâu rỗng (null string), là xâu không có ký tự nào.

Ví dụ : Luật A $\rightarrow$ ε

Là xóa tất cả các xuất hiện của A trong một xâu.

Các ký hiệu đặc biệt khác có vai trò như biến biểu diễn một ký tự bất kỳ được viết bởi các chữ cái thường a, b, c...

Ví dụ , luật A x B $\rightarrow$ B x A

Cho phép nghịch đảo các ký tự A và B.

Các chữ cái Hy lạp α, β dùng để chỉ các dấu đặc biệt của xâu. Ở đây, các chữ cái Hy lạp dùng để phân biệt với bảng chữ cái đang sử dụng.

Một ví dụ về thuật toán Markov là di chuyển chữ cái đầu tiên đến vị trí cuối cùng của một xâu vào. Những luật được ưu tiên áp dụng cao nhất là (1), thấp hơn là (2), rồi (3), v.v... Các luật được cho lần lượt theo độ ưu tiên giảm dần như sau :

(1) □xy $\rightarrow$ y□x

(2) □ $\rightarrow$ □

(3) □ $\rightarrow$ □

Cho xâu vào ABC, quá trình di chuyển được cho trong bảng sau :

Luật	*Thành công* (S) *hoặc thất bại* (F)	*Xâu kết quả*
1	F	ABC
2	F	ABC
3	S	□ABC
1	S	B□AC
1	S	BC□A
1	F	BC□A
2	S	BCA

Chú ý rằng ký hiệu □ hoạt động như là một biến trung gian trong ngôn ngữ lập trình. Tuy nhiên, thay vì nhận một giá trị, biến □ đóng vai trò giữ vị trí đánh dấu quá trình thay đổi xâu vào. Một khi công việc kết thúc, □ bị loại bỏ bởi luật 2.

c. Thuật toán mạng lưới (rete algorithm)

Chú ý rằng thuật toán Markov sử dụng chiến lược điều khiển tất định (definite control strategy) để áp dụng các luật có độ ưu tiên cao hơn trước tiên. Chừng nào mà luật có độ ưu tiên cao nhất không được áp dụng, thì thuật toán Markov sẽ tìm một luật khác có độ ưu tiên thấp hơn để áp dụng. Mặc dù thuật toán Markov có thể được sử dụng chủ yếu trong một hệ chuyên gia, nó vẫn không có hiệu quả trong những hệ thống có nhiều luật.

Vấn đề về *hiệu suất* (efficient) trở nên quan trọng khi người ta cần tạo ra các hệ chuyên gia giải quyết các bài toán thực tiễn chứa từ hàng trăm đến hàng ngàn luật. Một hệ chuyên gia là không hiệu quả nếu người sử dụng phải chờ đợi rất nhiều thời gian để nhận được một câu trả lời từ hệ thống. Vấn đề là cần có một thuật toán biết được tất cả các luật và có thể chọn ra các luật cần thiết để áp dụng thay vì thử lần lượt các luật.

Một giải pháp cho vấn đề này là thuật toán mạng lưới do Charles L. Forgy đề xuất tại trường Đại học Carnegie, Mellon, Hoa Kỳ vào năm 1979 trong luận văn tiến sĩ của ông về OPS (Official Production System).

Thuật toán mạng lưới cho phép so khớp (pattern mattching) rất nhanh để nhận được câu trả lời tức thời bằng cách lưu giữ thông tin của các luật trong một mạng lưới (network). Thay vì so khớp lặp đi lặp lại các sự kiện mỗi lần áp dụng một luật trong mỗi chu trình nhận thức (recognize-act cycle), thuật toán mạng lưới chỉ nhìn những thay đổi khi so khớp trong mỗi chu trình.

III. Thiết kế hệ chuyên gia

III.1. Thuật toán tổng quát

Thuật toán tổng quát để thiết kế một hệ chuyên gia gồm các bước như sau :

```
Begin
    Chọn bài toán thích hợp
    Phát biểu và đặc tả bài toán
    If Hệ chuyên gia giải quyết thoả mãn bài toán và có thể sử dụng Then
        While Bản mẫu chưa được phát triển hoàn thiện Do
        Begin
            Thiết kế bản mẫu
            Biểu diễn tri thức
            Tiếp nhận tri thức
            Phát triển hoàn thiện bản mẫu
        End
        Hợp thức hoá bản mẫu
        Triển khai cài đặt
        Hướng dẫn sử dụng
        Vận hành
        Bảo trì và phát triển
    Else
        Tìm các tiếp cận khác thích hợp hơn
    EnIf
    Kết thúc
End
```

Hình 1.13. Thiết kế một hệ chuyên gia

Để thiết kế một hệ chuyên gia, trước tiên cần có sự *lựa chọn một bài toán thích hợp* (selecting the appropriate problem). Tương tự các dự án phần mềm, để triển khai thiết kế một hệ chuyên gia, cần phải có các yếu tố về nhân lực, tài nguyên và thời gian. Những yếu tố này ảnh hưởng đến giá thành của một hệ chuyên gia.

Người ta thường đặt ra các câu hỏi sau đây :

Tại sao cần xây dựng (building) *một hệ chuyên gia* ?

Câu hỏi này thường xuyên được đặt ra cho bất kỳ dự án nào. Có thể trả lời ngay là do những đặc trưng và ưu điểm của các hệ chuyên gia. Trước khi bắt đầu, cần xác định rõ đâu là bài toán, ai là chuyên gia, và ai là người sử dụng.

Trả tiền (pay-off) *là gì* ?

Khi quyết định xây dựng một hệ chuyên gia (câu hỏi 1) cần một sự đầu tư về nhân lực, tài nguyên, thời gian và tiền bạc. Do vậy người sử dụng hệ chuyên gia phải trả tiền, tuỳ theo tính hiệu quả hay ưu điểm của hệ chuyên gia sử dụng. Tuy nhiên, nếu không có ai sử dụng hệ chuyên gia, thì sẽ không có ai trả tiền để bù lại chi phí và có lãi. Do hệ chuyên gia là một công nghệ mới, câu hỏi này khó trả lời hơn và có nhiều rủi ro hơn so với lập trình thông thường.

Sử dụng những công cụ (tools) *nào để xây dựng một hệ chuyên gia* ?

Hiện nay có rất nhiều công cụ để xây dựng các hệ chuyên gia. Mỗi công cụ đều có những ưu điểm và nhược điểm nhất định. Những công cụ phổ biến là CLIPS và OPS5, ngoài ra có ART, ART-IM, Eclipse, Cognate...

Chi phí (cost) *để xây dựng một hệ chuyên gia là bao nhiêu* ?

Chi phí hay giá thành để xây dựng một hệ chuyên gia phụ thuộc vào nguồn nhân lực, tài nguyên và thời gian hoàn thiện nó. Bên cạnh chi phí về phần cứng, phần mềm, còn chi phí về đào tạo (training). Ví dụ ở Mỹ, chi phí để đào tạo sử dụng thành thạo một hệ chuyên gia có thể lên tới 2.500USD/tuần lễ/người.

Sau bước lựa chọn, phát biểu và đặc tả bài toán là các bước phát triển hệ chuyên gia. Sau đây ta sẽ xem xét các hệ chuyên gia được phát triển như thế nào.

III.2. Các bước phát triển hệ chuyên gia

Hệ chuyên gia được phát triển như thế nào ?

Trong phạm vi rộng (large extent), việc phát triển một hệ chuyên gia phụ thuộc vào nguồn tài nguyên cung cấp. Tuy nhiên, giống như các dự án khác, việc phát triển còn phụ thuộc vào cách tổ chức quản lý quá trình phát triển như thế nào.

*a. Quản lý dự án **(Project Management)***

Quản lý dự án, chủ đề tiếp cận hệ chuyên gia, bao gồm các công đoạn như sau :

Quản lý hoạt động (Activity Management), gồm :

- *Lập kế hoạch* (planning)
  - định nghĩa các hoạt động (define activities)
  - xác định hoạt động ưu tiên (specify priority of activities)
  - nhu cầu tài nguyên (resource requirement)
  - ghi nhớ các sự kiện (milestones)
  - xác định thời gian (duration)
  - phân công trách nhiệm (responsabilities)
- *Lập biểu công việc* (scheduling)
  - ấn định điểm bắt đầu và điểm kết thúc dự án
  - giải quyết xung đột khi gặp các việc cùng mức ưu tiên

- *Phân bổ thời gian* (chronicling) - kiểm tra thực hiện dự án (monitor project performance)
- *Phân tích* (analysis) - phân tích các hoạt động về lập kế hoạch, lập biểu công việc và phân bổ thời gian hoạt động

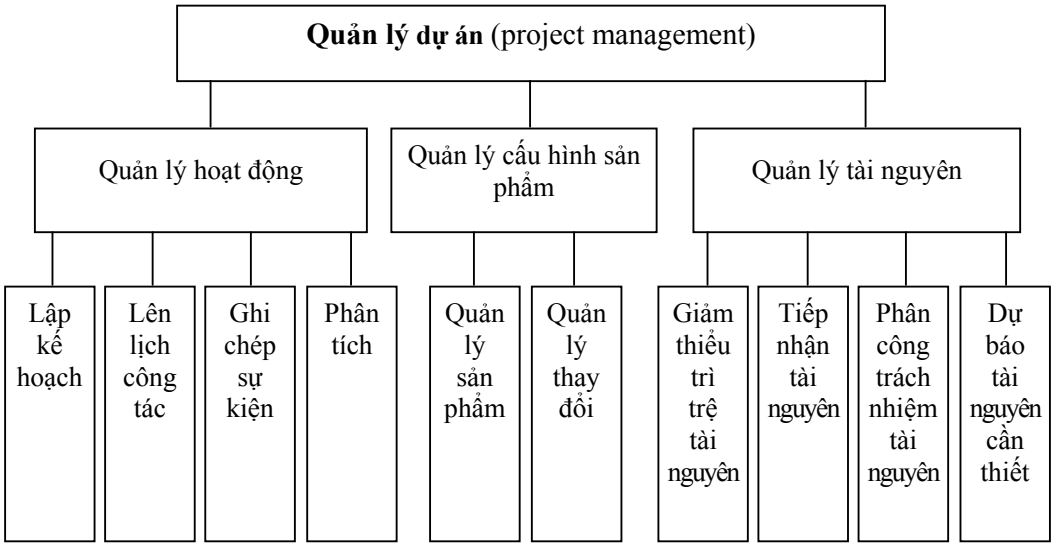
Quản lý cấu hình sản phẩm (Product Configuration Management) :

- *Quản lý sản phẩm* (product management) - quản lý các phiên bản khác nhau của các sản phẩm
- *Quản lý thay đổi* (change management) - quản lý các giải pháp sửa đổi sản phẩm và ước lượng ảnh hưởng của thay đổi sản phẩm
  - phân công người sửa đổi hệ thống
  - cài đặt phiên bản mới

Quản lý tài nguyên (Resource Management) :

- Dự báo nhu cầu tài nguyên (forecast needs for resource)
- Thu nhận tài nguyên (acquire resources)
- Phân công trách nhiệm để sử dụng tối ưu nguồn tài nguyên (assign responsabilities for optimium use of resources)
- Phân bổ tài nguyên để giảm thiểu tắc nghẽn (provide critical resources to minimize bottle-necks)

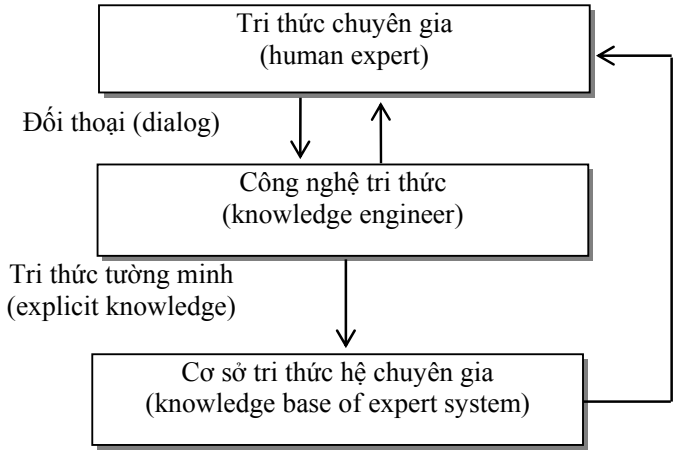
Hình dưới đây mô tả quá trình quản lý dự án phát triển một hệ chuyên gia.



Hình 1.14. Quản lý dự án phát triển một hệ chuyên gia

b. Tiếp nhận tri thức

Các bước tiếp nhận tri thức cho một hệ hệ chuyên gia như sau : Đầu tiên, công nghệ tri thức thu nhận tri thức nhờ đối thoại trực tiếp với tri thức con người (chuyên gia). Sau đó, tri thức được biểu diễn (theo một cách nào đó) tường minh trong cơ sở tri thức. Các chuyên gia đánh giá hệ chuyên gia, trao đổi qua lại với công nghệ tri thức cho đến khi hệ chuyên gia hoàn toàn thỏa mãn yêu cầu.



Hình 1.15. Tiếp nhận tri thức trong một hệ chuyên gia

c. Vấn đề phân phối (The Delivery Problem)

Hệ thống được phân phối như thế nào ?

Vấn đề phân phối một hệ thống phụ thuộc chủ yếu vào số lượng các hệ chuyên gia sẽ được phát triển. Tốt nhất là hệ chuyên gia có thể chạy trên các thiết bị phần cứng chuẩn. Tuy nhiên, một số hệ chuyên gia đòi hỏi phải có bộ xử lý LISP, từ đó làm tăng giá thành sản phẩm.

Nói chung, một hệ chuyên gia cần phải được tích hợp (integrated) với những chương trình đã có sẵn để có thể dùng lời gọi thủ tục từ một ngôn ngữ lập trình thông thường và hệ thống có thể hỗ trợ quá trình này.

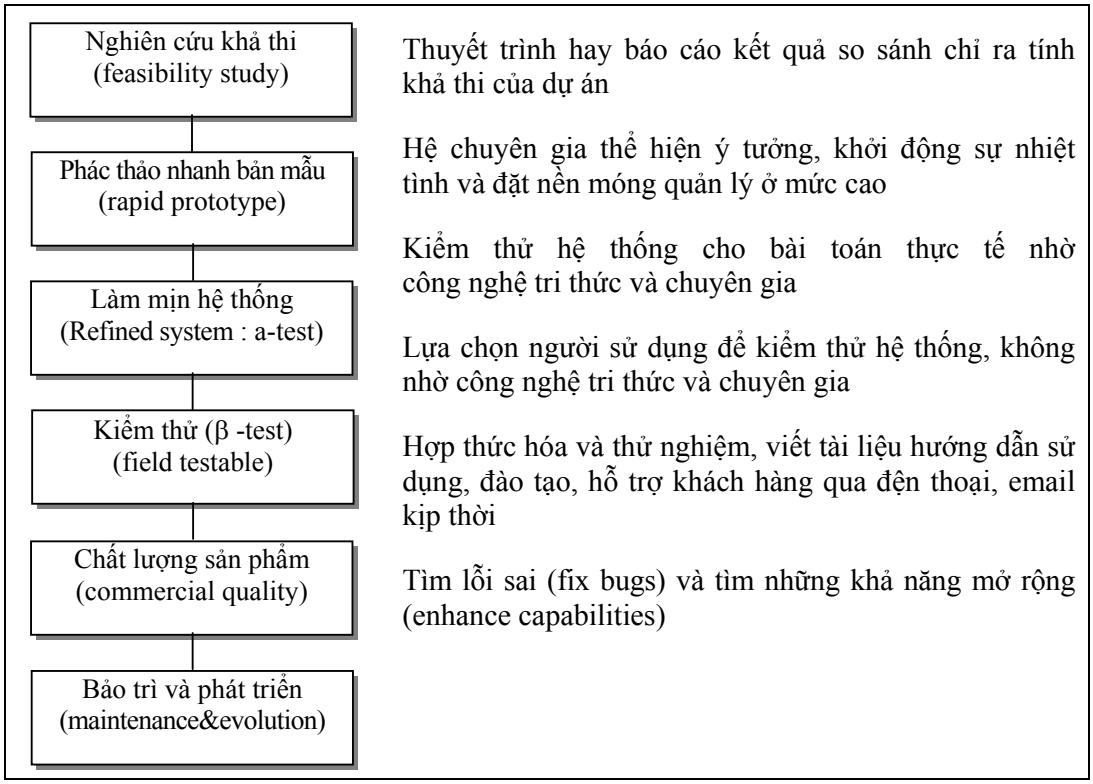
d. Bảo trì và phát triển

Hệ thống được bảo trì (maintenance) *và tiến triển* (evolve) *như thế nào* ?

Các hệ chuyên gia đòi hỏi các hoạt động bảo trì và phát triển không hạn chế (open-ended) so với các chương trình thông thường. Bởi vì các hệ chuyên gia không dựa trên các thuật toán, mà thành tích (performance) của chúng phụ thuộc vào tri thức. Vấn đề là phải thường xuyên bổ sung tiếp nhận các tri thức mới và thay đổi các tri thức cũ để đổi mới hệ thống (system improves).

Trong một sản phẩm có chất lượng thương mại (commercial quality product), cần phải thu thập một cách có hệ thống và có hiệu quả các báo cáo sai sót hệ thống do người sử dụng phát hiện. Nếu việc thu thập và khắc phục lỗi không được ưu tiên trong quá trình nghiên cứu thì phải được ưu tiên trong hệ thống chất lượng thương mại. Việc bảo trì chỉ được thực hiện tốt khi thu thập đầy đủ các báo cáo sai sót.

Hình 1.16. trình bày các giai đoạn cơ bản để phát triển một hệ chuyên gia.



Hình 1.16. Các giai đoạn phát triển một hệ chuyên gia

Sự phát triển một hệ hệ chuyên gia cũng tác động nhiều trong một hệ thống chất lượng thương mại. Người ta luôn mong muốn nhận được những thành công một khi hệ chuyên gia được phân phối đến người dùng.

III.3. Sai sót trong quá trình phát triển hệ chuyên gia

Các sai sót chủ yếu trong quá trình phát triển hệ chuyên gia được phân ra thành nhiều giai đoạn (hình 1.17.).

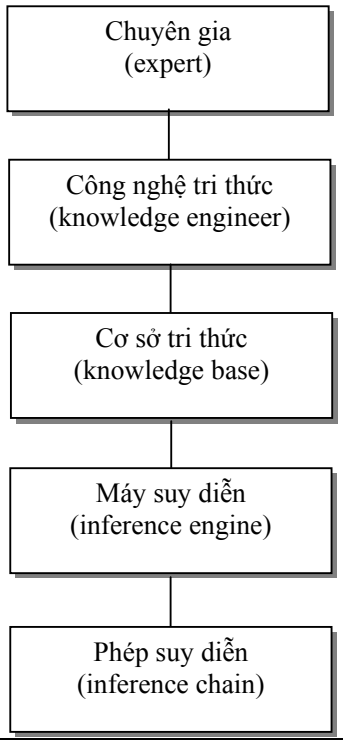
Sai sót trong tri thức chuyên gia. Chuyên gia là nguồn tri thức của một hệ chuyên gia. Nếu tri thức chuyên gia không đúng và không đầy đủ, hậu quả sai sót sẽ ảnh hưởng suốt quá trình phát triển hệ thống. Ví dụ : để hạn chế những sai sót có thể, NASA đã sử dụng *bảng kỹ thuật bay* (Flight Technique Panels) trong các chuyến bay vũ trụ. Các bảng này gồm những người sử dụng hệ thống, các chuyên gia lĩnh vực độc lập, những người phát triển hệ thống, những người quản trị nhằm bảo đảm tính đầy đủ và bao trùm hết mọi lĩnh vực phát triển.

Sai sót ngữ nghĩa. Xảy ra do hiểu sai tri thức đưa vào hệ chuyên gia. Ví dụ, giả sử một chuyên gia nói : « *You can extinguish a fire with water* » và công nghệ tri thức lại hiểu câu này là « *All fires can be extinguished by water* ».

Sai sót cú pháp. Do biểu diễn sai dạng các luật và các sự kiện, hoặc do sai sót ngữ nghĩa, hoặc sai sót trong tri thức chuyên gia ở các bước trước.

Sai sót máy suy diễn. Là một chương trình nên máy suy diễn có thể gặp lỗi khi thực hiện và có thể xác định được nguyên nhân. Tuy nhiên, việc xác định lỗi trong một số hệ chuyên gia vẫn gặp khó khăn do công cụ phần mềm sử dụng.

Ngoài ra, người ta cũng gặp phải sai sót khi suy diễn và những sai sót không biết được.



- Sai sót trong tri thức của chuyên gia : thiếu sót nhầm lẫn...
- Sai sót về mặt ngữ nghĩa giữa công nghệ tri thức và chuyên gia.
- Suy luận (elicitation) không đầy đủ về tri thức từ chuyên gia
- Sai sót về mặt cú pháp
- Sai sót do thiếu sót và nhầm lẫn tri thức trong các luật và các sự kiện, tính không chắc chắn
- Lỗi của máy suy diễn, lỗi phần mềm công cụ hệ chuyên gia
- Lỗi suy diễn do xác định sai độ ưu tiên của các luật, tương tác giữa các luật, sai trong cơ sở tri thức, suy luận không nhất quán...

Hình 1.17. Sai sót và nguyên nhân sai sót trong các hệ chuyên gia

Bài tập chương 1

1. Đọc kỹ giáo trình và tài liệu tham khảo để hiểu cac khái niệm đã trình bày.

2. Tự cho một số ví dụ về các phương pháp biểu diễn tri thức. Nhận xét.

CHƯƠNG 2

Biểu diễn tri thức nhờ logic vị từ bậc một

" The most important thing I have learned over the years is the difference between taking one's work seriously and taking one's self seriously. The first is imperative, and the second disastrous ".
Margaret Fontey

I. Ngôn ngữ vị từ bậc một

I.1. Các khái niệm

I.1.1. Cú pháp của ngôn ngữ vị từ bậc một

Trong *ngôn ngữ vị từ bậc một* (first–order predicate language), bằng cách sử dụng một bảng ký hiệu đặc biệt, người ta đưa vào các khái niệm *hạng* (term), *nguyên tử* (atom), *trực kiện* (literal) và *công thức chỉnh* (well–formed formula) để xây dựng các *biểu thức đúng* (correct expressions).

1. Bảng ký hiệu

Bảng ký hiệu để xây dựng các biểu thức đúng gồm :

- Các *dấu phân cách* (separator signs) là dấu phẩy (**,**), dấu mở ngoặc (**(**) và dấu đóng ngoặc (**)**).
- Các *hằng* (constant) có dạng chuỗi sử dụng các chữ cái in thường **a**..**z**.
  Ví dụ : a, block.
- Các *biến* (variable) có dạng chuỗi sử dụng các chữ cái in hoa **A**..**Z**.
  Ví dụ : X, NAME.
- Các *vị từ* (predicate) được viết tương tự các *biến*, là các chuỗi sử dụng các chữ cái in hoa **A**..**Z**.
  Ví dụ : ISRAINING, ON(table), P(X, blue), BETWEEN(X, Y, Z).

Khi cần thao tác trên một vị từ nào đó, cần phải ghi rõ *bậc* (arite) hay *số các đối* (argument) của vị từ đó. Bậc là một số nguyên dương. Ví dụ, trong một ứng dụng nào đó, bậc của các vị tự ISRAINING, ON, P và BETWEEN lần lượt là 0, 1, 2 và 3. Khi bậc có giá trị cố định là 0, vị từ còn được gọi là *mệnh đề* (proposition). Chẳng hạn ISRAINING, EMPTY là các mệnh đề.

- Các *hàm* (function), có cách viết tương tự các hằng, sử dụng các chữ in thường **a**..**z**.
  Mỗi hàm cũng có bậc (hay số lượng các đối) cố định, là một số nguyên dương.
  Ví dụ f(X), weight(elephan), successor(M, N) là các hàm có bậc lần lượt là 1, 1, và 2.
  Người ta quy ước rằng các hằng là những hàm bậc không (nil). Ví dụ a, elephan, block là các hằng.
- Các phép nối logic (logical connector) là $\neg$, $\wedge$, $\vee$, $\rightarrow$ và $\leftrightarrow$ tương ứng với các phép phủ định, và, hoặc, kéo theo và kéo theo lẫn nhau.

- Dấu ∃ là *lượng tử tồn tại* (existential quantifier) và ∀ là *lượng tử toàn thể* (universal quantifier).

2. Hạng (term)

Hạng được tạo thành từ hai luật sau :

- Các hằng và các biến là các hạng.
- Nếu f là một hàm có bậc $n \geq 1$ và nếu $t_1,..., t_n$ đều là các hạng, thì hàm f ($t_1,..., t_n$) cũng là một hạng.

***Ví dụ* :**

Các hàm successor(X, Y) hay weight(b) hay successor(b, wight(Z)) đều là các hạng, nhưng P(X, blue) không phải là hạng vì P là một vị từ hay weight (P(b)) cũng không phải là hạng vì P(b) không phải là một hạng (không thể làm đối cho một hàm).

3. Nguyên tử (atom)

Nguyên tử được tạo thành từ hai luật sau :

- Các mệnh đề (vị từ bậc 0) là các nguyên tử.
- Nếu P là một vị từ bậc n ($n \geq 1$) và nếu $t_1,..., t_n$ đều là các hạng, thì P($t_1,..., t_n$) cũng là một nguyên tử.

***Ví dụ* :**

P(X, blue), EMPTY, BETWEEN(table, X, sill(window)) là các nguyên tử.

Còn successor (X, Y), sill (window) ây thì không phải nguyên tử.

4. Các công thức chỉnh

Các *công thức chỉnh* (viết tắt *CTC*) được tạo thành từ ba luật sau :

- Các nguyên tử là các *CTC*.
- Nếu G và H biểu diễn các *CTC*, thì (¬G), (G ∧ H), (G ∨ H), (G → H) và (G ↔ H) cũng là các *CTC* do được tạo thành từ các phép nối lôgich giữa G và H.
- Nếu G là một *CTC* và X là một biến, thì (∃X)G và (∀X)G cũng là các *CTC*.

(∃X)G được đọc là *tồn tại một biến* X *sao cho G được thoả mãn*.

(∀X)G được đọc là *với mọi biến* X *thì G đều được thoả mãn*.

***Ví dụ* :**

Các công thức sau đây là chỉnh :

(∃X) (∀Y) ((P(X, Y) □ Q(X, Y) → R(X))

((¬(P(a) → P(b))) → ¬P(b))

còn (¬(f(a)) : *phủ định của một hàm*,

và f (P(a)) : *hàm có đối là một vị tư*, đều không phải là *CTC*.

Chú ý :

- Một *CTC* được gọi là một *trực kiện* (literal) hay một *trị đúng* nếu nó là một nguyên tử hay có dạng (¬G), với G là một nguyên tử.
- Trong một *CTC*, trước hoặc sau các ký tự nối, ký tự phân cách, các hằng, các biến, các hàm, các vị từ, người ta có thể đặt tùy ý các dấu cách (space hay blank).

Từ nay về sau ta quy ước rằng, trong một công thức, nếu có một biến được lượng tử hóa, tức là biến xuất hiện ngay theo sau ký hiệu ∃ hay ∀ thì từ đó trở đi, tất cả các vị trí đứng sau của cùng biến này cũng được lượng tử hóa.

Một *CTC* có thể chứa các biến không được lượng tử hóa, chúng được gọi là những *biến tự do* (free variable). Ví dụ : P(X) và (∃Y) Q(X, Y) là các *CTC* có chứa biến tự do X.

- Logic vị từ được gọi là «bậc một» (first–order) vì trong định nghĩa các *CTC* không chứa các lượng tử cho vị từ hay cho hàm.
  Ví dụ : (∀P)P(a) và (∀f) (∀f) (∀X) P(f (X), b)
  không phải là những *CTC* logic vị từ bậc một, mà có bậc cao hơn (higher-order).

5. Biểu diễn và sử dụng tri thức (knowledge)

Thực tế, các *CTC* dùng để diễn tả các nghĩa. Ví dụ *CTC* dưới đây :

$(\forall X)\ (MAN(X) \rightarrow M(X))$

thể hiện câu «tất cả mọi người đều chết» bằng cách quy ước rằng MAN(X) có nghĩa «X là một người» và M (X) có nghĩa «X chết».

Không phải luôn luôn dễ dàng dùng một *CTC* để biểu diễn một tri thức diễn tả theo ngôn ngữ tự nhiên (natural language). Chẳng hạn, để diễn tả rằng «nếu hai vật bằng nhau thì chúng có cùng tính chất», người ta có thể viết :

$(\forall P)\ (\forall X)\ (\forall Y)\ (EQUAL(X, Y) \rightarrow (P(X) \leftrightarrow P(Y)))$

Nhưng biểu thức trên không phải là logic vị từ bậc một vì có lượng tử ∀ áp dụng cho một ký tự vị từ là P.

I.1.2. Các luật suy diễn (inference rule)

Một *luật suy diễn* là cách biểu diễn sao cho từ một hoặc nhiều *CTC*, có thể suy dẫn (derive) thành các *CTC* khác. Chẳng hạn các luật suy diễn sau đây :

- Luật suy diễn *modus ponens* : Từ hai *CTC* lần lượt là G và (G → H), có thể suy dẫn ra *CTC* H (ở đây vẫn quy ước rằng các tên như G, H phải được thay thế bởi các *CTC* mà chúng biểu diễn).
- Luật suy diễn *modus tollens* : Từ các *CTC* là (¬H) và (G → H), ta suy dẫn ra được (¬G). Người ta viết quy ước hai luật suy diễn trên như sau :

$$\frac{G \rightarrow H \quad G}{\therefore H} \qquad \frac{\neg H \quad G \rightarrow H}{\therefore \neg G}$$

modus ponens *modus tollens*

- Luật suy diễn *chuyên dụng* (universal specialization), nếu từ một *CTC* có dạng :

  $(\forall X)\ G(X)$

  và từ một hằng bất kỳ, chẳng hạn «a», có thể suy dẫn thành *CTC* :

  G(a)

  nghĩa là mọi vị trí X trong G được thay thế bởi a.

Cho trước một tập hợp cố định các luật suy diễn, người ta có thể xem xét họ các bài toán sau : Từ một tập hợp các *CTC* đã chọn, bằng cách áp dụng một số hữu hạn lần nào đó các luật suy diễn, có thể nhận được một *CTC* đã cho trước hay không ?

Các *CTC* được chọn lúc đầu được gọi là các *tiên đề* (axiom). Các *CTC* nhận được bằng cách áp dụng các luật suy diễn được gọi là các *định lý* (theorem). Một dãy các áp dụng các luật suy diễn từ các tiên đề dẫn đến định lý là một phép *chứng minh* (proving) của định lý.

Một số kỹ thuật *hợp giải vấn đề* (problem resolution) thuộc lĩnh vực «Trí tuệ nhân tạo» như tìm kiếm trong không gian các trạng thái, có thể được xem như việc tìm kiếm một chứng minh cho một định lý đã cho. Theo nghĩa không gian các trạng thái, tập hợp các tiên đề có thể xem là một trạng thái đầu, các luật suy diễn đóng vai trò là các phép chuyển trạng thái, các trạng thái đích sẽ là tập hợp các *CTC* trong đó có chứa định lý cần chứng minh.

I.1.3. Ngữ nghĩa của ngôn ngữ vị từ bậc một

Sau đây, ta sẽ nghiên cứu cách sử dụng các *CTC* để biểu diễn và suy luận trên các *giá trị chân* (truth value) của các tri thức đã có để tìm được giá trị chân của các tri thức khác.

a. Diễn giải (Interpretation)

Một diễn giải của một *CTC* G, ký hiệu I, được xác định từ năm bước sau đây :

- Chọn một *miền diễn giải* (interpretation domaim) ký hiệu là D với $D \neq \varnothing$, nghĩa là một tập hợp khác rỗng các phần tử.
- Gán (assignation) cho mỗi hằng của G một phần tử của D.
- Gán cho mỗi mệnh đề (hay vị từ có bậc 0) một phần tử của tập hợp giá trị {*true*, *false*}. Để đơn giản ta ký hiệu F là trị *false* và T là trị *true*.
- Gán cho mỗi vị từ bậc n ($n \geq 1$) một ánh xạ từ D^n lên { T, F } :

  $P(X_1,... X_n) : D^n \rightarrow \{ T, F \}$
- Gán cho mỗi hàm bậc n ($n \geq 1$) một ánh xạ từ D^n lên D :

  $P(X_1,... X_n) : D^n \rightarrow D.$

Người ta nói rằng đã có một diễn giải I từ G lên D :

$$G \xrightarrow{I} D, \quad D \neq \varnothing$$

Ví dụ : Cho các miền diễn giải D_1, D_2, D_3 và các *CTC*

$G_1 : (\forall X)\ P(X)$

$G_2 : (\forall X)\ (\exists Y)\ Q(X, Y)$

$G_3 : (\forall X)\ (R(X) \rightarrow T\ (f\ (X), a))$

Ta xây dựng các diễn giải I_i của G_i lên D_i, i = 1.. 3, như sau :

I_1 : $D_1 = \{1, 2\}$

P(1)	P(2)
F	T

I_2 : $D_2 = \{1, 3\}$

Q(1, 1)	Q(1, 3)	Q(3, 1)	Q(3, 3)
F	T	F	F

I_3 : $D_3 = \{4, 5\}$

a
4

f(4)	f(5)
5	4

R(4)	R(5)
T	F

T(4, 4)	T(4, 5)	T(5, 4)	T(5, 5)
T	F	T	T

Chú ý : Đôi khi, người ta nói một diễn giải là không đầy đủ nếu trong đó, các phép gán cần thiết chỉ được đặc tả từng phần.

b. Giá trị một công thức theo diễn giải

Cho một diễn giải I của một miền D cho một công thức G.

- Nếu G là một mệnh đề, khi đó, giá trị gán cho G do định nghĩa của I được gọi là *giá trị của G theo I*.
- Nếu G là một trực kiện mà không phải là một mệnh đề, khi đó, với mỗi phép lựa chọn C các giá trị trong D cho các biến của G (nếu tồn tại), ta nhận được một giá trị *true* hay *false* theo cách định nghĩa I. Giá trị này được gọi là *giá trị của G theo I đối với lựa chọn C các giá trị của các biến*.

Chẳng hạn, trong công thức G_3 ở trên được diễn giải theo I_3, nguyên tử T(f(X), a) nhận giá trị T nếu X được gán phần tử 4 của D_3, và cũng nhận giá trị T nếu X nhận một giá trị khác (giả sử 5) của D_3.

- Nếu G có dạng $(\forall X)G'$, ta định nghĩa giá trị của G theo I là T (*true*) nếu giá trị của G' theo I cho mọi giá trị của biến X (trong D) là T, nếu không là F (*false*). Chẳng hạn, giá trị của G1 được diễn giải theo I là F.
- Nếu G có dạng $(\exists X)G'$, ta định nghĩa giá trị của G theo I là T (*true*) nếu giá trị của G' theo I đối với ít nhất một giá trị của biến X (trong D) là T, nếu không là F (*false*).

Chẳng hạn, giá trị của Q(X, Y) được diễn giải theo I_2 là T khi gán 1 cho X và 3 cho Y. Từ đó suy ra rằng giá trị của $(\exists Y)Q(X, Y)$ theo I_2, khi X nhận giá trị 1, là T. Ngược lại, giá trị của G_2 theo I_2 là F.

- Nếu G có dạng $(\neg G')$, người ta định nghĩa giá trị của $\neg$ G' theo I, khi giá trị này của G' theo I được định nghĩa, căn cứ theo bảng sau :

Giá trị của G' theo I	Giá trị của $(\neg G')$ theo I (giả sử G)
T	F
F	T

- Nếu G có dạng $(G' \vee G'')$, hoặc $(G' \wedge G'')$, hoặc $(G' \rightarrow G'')$, hoặc $(G' \leftrightarrow G'')$, người ta định nghĩa giá trị của G theo I, khi các giá trị của G' và G'' theo I được định nghĩa, căn cứ theo các bảng chân lý (truth table) tương ứng sau :

G'	G''	$(G' \vee G'')$
F	F	F
F	T	T
T	F	T
T	T	T

G'	G''	$(G' \wedge G'')$
F	F	F
F	T	F
T	F	F
T	T	T

G’	G’’	(G’ → G’’)
F	F	T
F	T	T
T	F	F
T	T	T

G’	G’’	(G’ ↔ G’’)
F	F	T
F	T	F
T	F	F
T	T	T

Chẳng hạn, giá trị của G_3 theo I_3 là T.

Khi một công thức G là T theo một diễn giải I, người ta nói rằng diễn giải I là một *mô hình* của G.

Chú ý rằng giá trị của một công thức theo một diễn giải đã cho được định nghĩa theo cách qua lại tương hỗ ngay khi tất cả các biến được lượng tử hoá.

I.2. Các tính chất

I.2.1. Tính hợp thức / không hợp thức, tính nhất quán / không nhất quán

Một công thức được gọi là *hợp thức* (valid) nếu và chỉ nếu mọi diễn giải đều cho giá trị T. Nếu không, nó được gọi là *không hợp thức* (non−valid).

Một công thức được gọi là *không nhất quán* (inconsistent) nếu và chỉ nếu với mọi diễn giải đều cho giá trị F. Nếu không, nó được gọi là *nhất quán* (inconsistent).

Ví dụ **:**

Cho G_1 : $(\forall X)\ (P(X) \vee (\neg Q(X)))$

G_2 : $(\forall X)\ (P(X) \vee (\neg P(X)))$

G_3 : $((\forall X)\ (P(X)) \wedge ((\exists Y)\ (\neg P(Y)))$

Công thức G_1 là nhất quán vì diễn giải I'_1 sau đây trả về cho nó giá trị T :

I'_1 : D = {1} P(1) □ T Q(1) □ T

Tuy nhiên, nó là không hợp thức vì diễn giải I''_1 sau đây trả về giá trị F :

I''_1 : D = {1} P(1) □ F Q(1) □ T

Công thức G_2 là hợp thức vì nếu không, giả sử I là một diễn giải thuộc miền D làm sai G_2, khi đó tồn tại một giá trị «a» của X, lấy trong D, sao cho $(P(a) \vee (\neg P(a)))$ là F, mà điều này không thể xảy ra do cách định nghĩa các phép ∨ và ∧. Như vậy, G_2 phải là hợp thức.

Công thức G_3 là không hợp thức vì nếu không, giả sử I là một mô hình của G, I phải làm thoả mãn $(\exists Y)\ (\neg P(Y))$, khi đó tồn tại một giá trị «a» trong D, sao cho P(a) có giá trị F, nhưng $(\forall X)\ (P(X)$ không thể thoả mãn trên D. Như vậy, G_3 phải là không hợp thức.

Chú ý **:**

- Một số tác giả gọi các công thức hợp thức là các *hằng đúng* (tautology) và các công thức không nhất quán là các *mâu thuẫn* (contradiction).
- Cách viết công thức có thể gây nhầm lẫn. Chẳng hạn công thức :
  $(\forall X)\ (MAN(X) \rightarrow MORTAL(X))$.
  gợi ý rằng « *tất cả mọi người đều chết* » và công thức là hợp thức. Thực tế, công thức này là nhất quán, nhưng không hợp thức, vì giá trị trả về của công thức phụ thuộc vào diễn giải theo biến X.

I.2.2. Tính không quyết định được và tính nửa quyết định được

Khi một công thức không chứa các biến, người ta có thể sử dụng các bảng chân lý để tiến hành một số hữu hạn các phép toán nhằm xác định một công thức đó là hợp thức hay không, có nhất quán hay không. Vấn đề trở nên vô cùng phức tạp khi các công thức có chứa biến và các dấu lượng tử.

Người ta đã chỉ ra rằng trong logic vị từ bậc một, không thể tìm được một thuật toán tổng quát để quyết định xem với chỉ một số hữu hạn phép toán, một công thức bất kỳ nào đó đã cho có là hợp thức hay không. Do vậy, người ta gọi logic vị từ bậc một là *không quyết định được* (indecidability) (theo định lý về tính không quyết định được của A. Church xây dựng năm 1936).

Tuy nhiên, người ta có thể xây dựng các thuật toán tổng quát để quyết định tính hợp thức của một số họ các *CTC*. Đặc biệt, tồn tại các thuật toán đảm bảo tính hợp thức ngay từ đầu khi ứng dụng một *CTC* hợp thức nào đó, bằng cách dừng lại sau khi áp dụng một số hữu hạn (nhưng không bị chặn trên) các phép toán để kết luận rằng công thức đã cho là hợp thức. Một thuật toán như vậy khi áp dụng cho một công thức không hợp thức có thể không bao giờ dừng. Chính vì vậy mà người ta nói logic vị từ bậc một là *nửa quyết định được* (half–decidability).

I.2.3. Công thức tương đương

Hai *CTC* G và H được gọi là tương đương nếu và chỉ nếu chúng có cùng giá trị (T hoặc F) cho mọi diễn giải. Người ta viết : với mọi diễn giải I, I(G) = I(H).

***Ví dụ* :**

$(P(a) \rightarrow Q(b))$ và $((\neg P(a) \vee Q(b))$ là tương đương.

Có thể kiểm tra lại kết quả bằng bảng chân lý.

Hình 2.1 dưới đây là danh sách các công thức tương đương với quy ước rằng :

- G, H, K là các *CTC* bất kỳ,
- G(X), H(X) là các *CTC* với X là biến tự do,
- □ biểu diễn một *CTC* hợp thức,
  ∇ biểu diễn một *CTC* không nhất quán.

Công thức tương đương		*Được gọi là*
$(G \rightarrow H)$	$((\neg G) \vee H)$	
$(G \leftrightarrow H)$	$((G \rightarrow H) \wedge (H \rightarrow G))$	
$(\neg (\neg G))$	G	
$(\neg(G \wedge H))$	$((\neg G) \vee (\neg H))$	Luật De Morgan
$(\neg(G \vee H))$	$((\neg G) \wedge (\neg H))$	
$((G \wedge (H \vee K))$	$((G \wedge H) \vee (G \wedge K))$	Luật phân phối
$((G \vee (H \wedge K))$	$((G \vee H) \wedge (G \vee K))$	
$(G \wedge H)$	$(H \wedge G)$	Luật giao hoán
$(G \vee H)$	$(H \vee G)$	
$((G \vee H) \vee K)$	$(G \vee (H \vee K))$	Luật kết hợp cho phép loại bỏ dấu ngoặc
$((G \wedge H) \wedge K))$	$(G \wedge (H \wedge K))$	
$(G \rightarrow H)$	$((\neg H) \rightarrow (\neg G))$	Luật đối vị

Công thức tương đương		*Được gọi là*
$(G \wedge \Box)$	G	
$(G \wedge \nabla)$	∇	
$(G \vee \Box)$	$\Box$	
$(G \vee \nabla)$	G	
$(G \vee (\neg G))$	$\Box$	
$(G \wedge (\neg G))$	∇	
$(\forall X)(G(X))$	$(\forall Y)(G(Y))$	Luật dùng chung các biến
$(\exists X)(G(X))$	$(\exists Y)(G(Y))$	
$\neg((\forall X)G(X))$	$(\exists Y)(\neg G(Y))$	
$\neg((\exists X)G(X))$	$(\forall Y)(\neg G(Y))$	
$(\forall X)(G(X) \wedge H(X))$	$((\forall X)G(X) \wedge (\forall Y)H(Y))$	
$(\exists X)(G(X)) \vee H(X))$	$((\exists X)G(X) \vee (\exists Y)H(Y))$	

Hình 2..1 Bảng các công thức tương đương

I.2.4. Hậu quả logic

Công thức G được gọi là *hậu quả logic* từ các công thức $H_1,..., H_n$ nếu và chỉ nếu mọi mô hình của $H_1,..., H_n$ là một mô hình của G.

Ví dụ **:**

P(a) là hậu quả logic của $(\forall X)$ P(X)

$(\forall X)$ Q(X) là hậu quả logic của $(\forall X)$ $((\neg P(X)) \vee Q(X))$ và $(\forall X)$ P(X)

Dễ dàng chỉ ra rằng G là hậu quả logic của $H_1,..., H_n$ nếu và chỉ nếu :

$((H_1 \wedge ... \wedge H_n) \rightarrow G)$ là hợp thức, hay nếu và chỉ nếu $(H_1 \wedge ... \wedge H_n) \vee (\neg G))$ là không nhất quán.

I.3. Quan hệ giữa định lý và hậu quả logic

Ta thấy rằng việc định nghĩa các luật suy diễn, rồi đưa ra các định lý và chứng minh là độc lập với các khái niệm diễn giải (đưa vào các giá trị *true* và *false*), tương đương và hậu quả logic.

I.3.1. Nhóm các luật suy diễn «đúng đắn» (sound)

Khi các định lý, nhận được bằng cách áp dụng một nhóm các luật suy diễn đã cho, là hậu quả logic một cách hệ thống từ một tập hợp các tiên đề bất kỳ nào đó, người ta nói rằng nhóm các luật suy diễn này là đúng đắn.

Ví dụ, dễ dàng chỉ ra rằng các luật suy diễn *modus ponens* và *chuyên dụng* đã nói trước đây là đúng đắn.

I.3.2. Nhóm các luật suy diễn «đầy đủ»

Một nhóm các luật suy diễn đã cho là đầy đủ đối với phép suy diễn (deduction complete) nếu với bất kỳ một tập hợp các *CTC*, mọi hậu quả logic của chúng đều được dẫn đến từ chúng như những định lý, nghĩa là bởi áp dụng một số hữu hạn lần các luật suy diễn của nhóm.

Ví dụ, nhóm các luật suy diễn chỉ được rút ra từ luật *modus ponens* sẽ không là đầy đủ đối với phép suy diễn, $(\forall X)\ (G(X) \vee H(X))$ là một *CTC* hậu quả logic của hai *CTC* $(\forall X)\ G(X)$ và $(\forall X)\ H(X)$. Nhưng công thức đầu tiên chỉ có thể được suy diễn từ hai công thức này bởi *modus ponens* mà thôi.

I.3.3. Vì sao cần «đúng đắn» hay «đầy đủ» ?

Trong hệ thống hợp giải càc bài toán, logic vị từ bậc một thường dùng để biểu diễn những khẳng định là đúng hay sai trong những miền ứng dụng chuyên biệt. Người ta quan tâm đến phép suy diễn các *CTC*.

Nếu ta lấy các luật suy diễn trong những hệ thống này, thì một cách tự nhiên, chúng phải tạo thành nhóm «đúng đắn».

Rõ ràng người ta mong muốn nhóm các luật phải đầy đủ. Nghĩa là mọi hậu quả logic của các tiên đề phải là một định lý và do vậy phải được làm rõ bởi dẫn xuất các luật suy diễn. Tuy nhiên trong thực tế, không phải luôn luôn như vậy.

Sau đây ta sẽ chỉ ra một luật suy diễn quan trọng là *phép hợp giải* (principle of resolution), hay nói gọn là *hợp giải* (resolution).

II. Phép hợp giải

Phép hợp giải là một luật suy diễn áp dụng vào một tập hợp các *CTC* hay một tập hợp các mệnh đề (clauses) đã cho.

II.1. Biến đổi các mệnh đề

Mệnh đề là tất cả các *CTC* được xây dựng từ phép hoặc (disjonction) của các trực kiện (literals). Ví dụ :

$R(Z, a, g(X)) \vee (\neg T(U)) \vee (\neg V(b, k(c))$.

II.1.1. Dạng chuẩn trước của một công thức chỉnh

Dạng chuẩn trước (prenex normal form) của một *CTC* đã cho được suy ra bởi áp dụng liên tiếp bốn phép biến đổi sau đây :

a. Loại bỏ các phép nối $\rightarrow$ và $\leftrightarrow$

Sử dụng các luật tương đương :

$(G \rightarrow H)$ và $((\neg G) \vee H)$
$(G \leftrightarrow H)$ và $((G \rightarrow H) \wedge (H \rightarrow G))$

b. Ghép các phép nối $\neg$ với các nguyên tử liên quan

Sử dụng các luật tương đương :

$(\neg\ (\neg G))$	và	G	
$(\neg(G \vee H))$	và	$((\neg G) \wedge (\neg H))$	Luật De Morgan
$(\neg(G \wedge H))$	và	$((\neg G) \vee (\neg H))$	
$\neg((\exists X)\ P(X))$	và	$(\forall X)\ (\neg P(X))$	
$\neg((\forall X)\ P(X))$	và	$(\exists X)\ (\neg P(X))$	

c. Phân biệt các biến

Dùng luật tương đương để tác động dấu lượng tử lên một biến sơ khởi :
$(\forall X)\ P(X)$ và $(\forall Y)\ P(Y)$

(∃X) P(X) và (∃Y) P(Y) Luật dùng chung các biến

Chú ý :

Các phép biến đổi **a**, **b**, **c** dẫn *CTC* đã cho thành một *CTC* mới tương đương.

d. Dịch chuyển các dấu lượng tử

Khi dịch chuyển các dấu lượng tử trong một *CTC*, ta nhận được một *CTC* mới tương đương. Bởi vì sau bước **c**, không còn sự xung đột giữa các nhãn biến được lượng tử hoá.

Sau bốn bước biến đổi, ta nhận được công thức có *dạng chuẩn trước* tương đương với *CTC* đã cho. Phần tiếp theo của các dấu lượng tử được gọi là *tiền tố* (prefix) và phần còn lại được gọi là *ma trận* (matrix).

Có nhiều dạng chuẩn trước khác nhau cho cùng một *CTC* bằng cách áp dụng các luật tương đương nhưng vẫn tuân theo nội dung của các bước **a**, **b**, **c**, **d**.

***Ví dụ* :**

Cho *CTC* G :

$((\forall X)\ ((P(X) \wedge Q(X, a)) \rightarrow$
$(R(X, b) \wedge ((\forall Y)\ ((\forall Z)\ R(Y, Z) \rightarrow T(X, Y)))))) \vee ((\forall X)\ S(X))$

Sau bước **a**, ta nhận được :

$((\forall X)\ (\neg(P(X) \wedge Q(X, a)) \vee$
$(R(X, b) \wedge ((\forall Y)\ (\neg\ (\forall Z)\ R(Y, Z) \vee T(X, Y)))))) \vee ((\forall X)\ S(X))$

Sau các bước **b** và **c**, ta nhận được (với quy ước dấu $\neg$ áp dụng cho nguyên tử theo sau) :

$((\forall X)\ (\neg P(X) \wedge \neg Q(X, a)) \vee$
$(R(X, b) \wedge ((\forall Y)\ (((\exists Z)\ \neg R(Y, Z) \vee T(X, Y)))))) \vee ((\forall U)\ S(U))$

Sau bước **d**, ta nhận được dạng chuẩn trước của G :

$((\forall X)\ (\forall Y)\ (\exists Z)\ (\forall U)$
$(((\neg P(X) \vee \neg Q(X, a)) \vee (R(X, b) \wedge (\neg R(Y, Z) \vee T(X, Y)))) \vee (S(U))$

hay áp dụng tính chất kết hợp của phép hoặc $\vee$:

$((\forall X)\ (\forall Y)\ (\exists Z)\ (\forall U)$
$(\neg P(X) \vee \neg Q(X, a) \vee (R(X, b) \wedge (\neg R(Y, Z) \vee T(X, Y))) \vee (S(U))$

II.1.2. Chuyển qua "dạng mệnh đề" của công thức chỉnh

Từ dạng chuẩn trước G' của *CTC* G, ta có thể tạo ra *dạng mệnh đề* (form of clauses) G" từ G bằng cách áp dụng 5 bước chuyển đổi sau đây. Trước tiên cần chú ý rằng nếu các công thức G và G' luôn luôn tương đương với nhau thì các công thức G và G'' có thể không tương đương với nhau.

Ta chỉ xét G không nhất quán tương đương với G" không nhất quán. Quan hệ này vẫn đủ cho phép tạo cơ sở cho việc chứng minh tự động sau này.

a. Loại bởi các dấu lượng tử tồn tại

Cho công thức dạng (∃X) G(X) và giả sử G(X) được tạo thành từ một hoặc nhiều công thức chỉ được lượng tử hoá toàn thể (∀) đối với các biến $Y_1,..., Y_n$ mà thôi. Ta sẽ loại bỏ lượng tử tồn tại (∃X), sau đó thay thế mỗi vị trí của X trong G(X) bởi một hàm có dạng $f(Y_1,..., Y_n)$.

Chú ý rằng hàm này phải chứa tất cả các biến được lượng tử hoá toàn thể nằm bên trái lượng tử tồn tại trong công thức (∃X) G(X). Đây là một hàm cho biết có sự tương ứng giữa

các nhóm giá trị của Y_1,..., Y_n và các giá trị của X. Giá trị của X được chỉ định bởi lượng tử tồn tại ∃.

Những hàm như vậy được gọi là *các hàm Skolem* (Skolem functions). Vì rằng ta không biết gì khác ngoài các tham biến của các hàm này, ta phải sử dụng một ký hiệu gốc để biểu diễn chúng mỗi lần cần thay thế một lượng tử toàn thể khi nó xuất hiện.

Khi không có dấu «∀» nào bên trái của «∃» đã cho, hàm Skolem sẽ không có tham biến, và được gọi là một *hằng Skolem* (Skolem constant).

***Ví dụ* :**

(∃X) P(X)	trở thành	P(a)
(∀X) (∃Y) FOLLOW(Y, X)	trở thành	(∀X) FOLLOW(f(X), X)

Dạng chuẩn trước của công thức cho trong ví dụ ở mục II.1.1 trở thành :

((∀X) (∀Y) (∀U)

(¬P(X) ∧ ¬Q(X, a) ∨ (R(X, b) ∧ (¬R(Y, g(X, Y)) ∨ T(X, Y))) ∨ (S(U))

b. Loại bỏ tất cả các dấu lượng tử

Sau bước **a** trên đây, công thức chỉ còn các dấu lượng tử toàn thể. Với giả thiết rằng tất cả các biến đều được lượng tử hoá toàn thể, ta có thể loại bỏ chúng.

Ví dụ sau cùng ở mục trên đây trở thành :

(¬P(X) ∨ ¬Q(X, a) ∨ (R(X, b) ∧ (¬R(Y, g(X, Y)) ∨ T(X, Y))) ∨ (S(U))

c. Chuyển qua «dạng chuẩn hội»

Trong bước này, người ta sử dụng các luật kết hợp và phân phối đối với các phép toán logic ∧ và ∨ để rút gọn *CTC* nhận được chỉ còn toàn là phép hội (conjonction) của các phép hoặc (disjonction) của các mệnh đề hay các trực kiện (literal).

***Ví dụ* :**

$\neg P(X) \vee Q(X, a) \wedge \neg R(Y, f(X), b)$ trở thành

$(\neg P(X) \vee Q(X, a)) \wedge (\neg P(X) \vee \neg R(Y, f(X), b))$

Ví dụ ở mục trên đây trở thành :

$(\neg P(X) \vee \neg Q(X, a) \vee R(X, b) \vee S(U))$

$\wedge (\neg P(X) \vee \neg Q(X, a) \vee \neg R(Y, g(X, Y)) \vee T(X, Y) \vee (S(U))$

d. Loại bỏ tất cả các dấu phép toán logic

Phép hội của các mệnh đề nhận được từ bước trên đây được xem như một tập hợp của các mệnh đề một cách truyền thống. Chẳng hạn trong bước trên, ta nhận được tập hợp hai mệnh đề là :

$\{ (\neg P(X) \vee \neg Q(X, a) \vee R(X, b) \vee S(U)),$
$(\neg P(X) \vee \neg Q(X, a) \vee \neg R(Y, g(X, Y)) \vee T(X, Y) \vee (S(U)) \}$

e. Phân biệt các biến của các mệnh đề

Các mệnh đề trong tập hợp trên đây phải được đặt lại tên biến sao cho không có sự trùng tên. Muốn vậy, ta sử dụng các luật tương đương tổng quát :

$(\forall X)\ (G(X) \wedge H(X))$ và $((\forall X)\ G(X) \wedge (\forall Y)\ H(Y))$

Chẳng hạn ta có :

$\{ (\neg P(X) \vee \neg Q(X, a) \vee R(X, b) \vee S(U)),$
$(\neg P(Y) \vee \neg Q(Y, a) \vee \neg R(Z, g(Y, Z)) \vee T(Y, Z) \vee (S(V)) \}$

II.1.3. Quan hệ giữa *CTC* và các dạng mệnh đề của chúng

Cho các *CTC* $G_1,..., G_p$. Giả sử $G''_1,..., G''_p$ là các dạng mệnh đề tương ứng với $G_1,..., G_p$ nhận được từ các bước **a**.. **e** trong mục **II.1.2** trên đây, hoặc nhận được từ các bước **a**.. **d** trong mục **II.1.1** để tạo ra các dạng chuẩn trước $G'_1,..., G'_p$. Giả sử mỗi G''_i có dạng :

$G''_i = \{ C'_1,..., C_i^{k_i} \}$

Ta thấy rằng $\{ G_1,..., G_p \}$ là không nhất quán nếu và chỉ nếu :

$\bigcup_{i=1,p} G''_i$ là không nhất quán.

Một kết quả khác tương tự là mọi công thức là hậu quả logic của $\{ G_1,..., G_p \}$ thì cũng là hậu quả logic của $\bigcup_{i=1,p} G''_i$.

***Chú ý* :**

1) Trong trường hợp tổng quát, người ta có thể biến đổi một *CTC* đã cho thành nhiều dạng mệnh đề khác nhau. Người ta có thể loại bỏ một lần các lượng tử tồn tại (bước **II.1.2**.**a**) trước khi chuyển qua trái tất cả các lượng tử (bước **II.1.1**.**d**). Cách này có thể làm giảm số lượng tham đối của các hàm Skolem xuất hiện.

   Chẳng hạn, giả sử sau bước **II.1.1**.**c**, ta được *CTC* :

   $((\forall X)\ P(X) \wedge (\exists Y)\ Q(Y))$, bước **II.1.1**.**d** dẫn đến :
   $(\forall X)\ (\exists Y)\ (P(X) \wedge Q(Y))$, tiếp theo, bước **II.1.2**.**a** dẫn đến :
   $(\forall X)\ (P(X) \wedge Q(f(X)))$, cuối cùng, ta được tập hợp dạng mệnh đề :
   $\{ P(X), Q(f(Z)) \}$

trong khi đó, nếu thực hiện bước **II.1.2**.**a** rồi bước **II.1.1**.**d,** ta nhận được :

   $(\forall X)\ (P(X) \wedge Q(a))$, từ đó ta được tập hợp :
   $\{ P(X), Q(a) \}$

   Các kết quả trên đây có nghĩa dù dạng mệnh đề nhận được là như thế nào.

2) Khi áp dụng logic vị từ bậc một, thông thường người ta muốn một *CTC* H là hậu quả logic của các *CTC* $G_1,..., G_n$. Trong mục **I.2.3**, ta đã chỉ ra rằng điều này tương đương với sự không nhất quán của *CTC* :

   $K = G_1 \wedge ... \wedge G_n \wedge \neg H$

   Từ các kết quả trước đây, ta muốn tìm kiếm các dạng mệnh đề $G_1,..., G_n$ và $\neg H$ một cách độc lập, sau đó tích hợp chúng lại, thay vì tìm kiếm trực tiếp một dạng mệnh đề của K theo các bước **II.1.1** rồi các bước **a**.. **e** trong **II.1.2**.

3) Nếu G" là một dạng mệnh đề của G, thì G chỉ tương đương với G" khi G và G" là không nhất quán. Nếu G là nhất quán, thì khi đó, một cách tổng quát, G không tương đương với G".

***Ví dụ* :**

Cho G là *CTC* : $(\exists X)\ P(X)$ và giả sử G" là mệnh đề P(a). Nếu ta diễn giải trên miền D = {1, 2} cho bởi :

a □ 1 và

P(1)	P(2)
F	T

thì ta nhận thấy rằng diễn giải này làm G đúng và làm G" sai.

Từ nhận xét này, người ta đặt *CTC* dưới dạng mệnh đề G và $\neg H$, mà không phải dưới dạng G và H, để chứng minh rằng H là hậu quả logic của G.

II.1.4. Phép hợp giải đối với các mệnh đề cụ thể

Người ta nói một trực kiện là *cụ thể* (concrete) nếu nó không chứa các biến.

Chẳng hạn, $\neg P(a)$, $Q(a, f(b))$ là các trực kiện cụ thể, nhưng $\neg P(X)$, $Q(a, f(Y))$ không phải là các trực kiện cụ thể.

Một *mệnh đề cụ thể* là phép hoặc của các trực kiện cụ thể.

Cho hai mệnh đề cụ thể :

$G = G_1 \vee ... \vee G_n$ và

$H = \neg G_1 \vee H_2 \vee ... \vee H_m$

với G_i và H_i là các trực kiện cụ thể. Các trực kiện G_1 và $\neg G_1$ có mặt trong G và H tương ứng, được gọi là *các trực kiện bù nhau* (complementary literals).

Xuất phát từ các *mệnh đề cha* (parent clauses) là G và H, luật suy diễn, hay *phép hợp giải*, sẽ tạo ra một mệnh đề:

$K = G_2 \vee ... \vee G_n \vee H_2 \vee ... \vee H_m$

K được gọi là *mệnh đề kết quả* (resolvent clause) hay *kết quả hợp giải* (resolvent) của G và H. Người ta cũng nói rằng G và H được *hợp giải với nhau* (resolved) để tạo thành K. Một kết quả hợp giải là sự loại bỏ các trực kiện bù nhau và tuyển với tất cả các trực kiện khác của các mệnh đề cha.

Một số trường hợp đặc biệt :

- Q là một kết quả hợp giải của các mệnh đề cụ thể P và $\neg P \vee Q$(hay P và $P \rightarrow Q$). Phép hợp giải này thực chất là luật suy diễn *modus ponens* (trên các mệnh đề cụ thể).
- $\neg G \vee K$ (hay $G \rightarrow K$) là kết quả hợp giải của các mệnh đề cụ thể $\neg G \vee H$ và $\neg H \vee K$ (hay $G \rightarrow H$ và $H \rightarrow K$). Quy tắc suy diễn này là một trường hợp đặc biệt của phép hợp giải còn được gọi là *phép liên kết* (chỉ đối với các mệnh đề cụ thể).
- Các mệnh đề cụ thể G và $\neg G$ được hợp giải với nhau để tạo thành *mệnh đề rỗng* (empty clause)

***Chú ý* :**

- Hai mệnh đề cụ thể không có kết quả hợp giải. Chẳng hạn G và $\neg H$, với G và H là các nguyên tử khác nhau.
- Hai mệnh đề cụ thể có thể có nhiều kết quả hợp giải. Chẳng hạn hai mệnh đề $G \vee H \vee K$ và $\neg G \vee \neg H \vee L$ được hợp giải thành $H \vee \neg H \vee K \vee L$ hay thành $G \vee \neg G \vee K \vee L$ là những mệnh đề tương đương.
- $\neg P$ là một kết quả hợp giải của các mệnh đề cụ thể $\neg Q$ và $\neg P \vee Q$(hay $\neg Q$ và $P \rightarrow Q$). Phép hợp giải này thực chất là luật suy diễn *modus tollens*.

Trước khi định nghĩa tổng quát phép hợp giải áp dụng cho các mệnh đề không phải luôn luôn cụ thể, cần phải định nghĩa một cơ chế tạo sinh các mệnh đề. Đó là phép *hợp nhất*.

II.2. Phép hợp nhất (unification)

II.2.1. Khái niêm

Giả sử cho các mệnh đề $\neg G(X) \vee H(X)$ và $G(f(Y))$. Nếu mệnh đề thứ nhất được thay thế bởi $\neg G(f(Y)) \vee H(f(Y))$, thì ta có thể mở rộng phép hợp giải cho các mệnh đề cụ thể : loại bỏ các trực kiện bù nhau $\neg G(f(Y))$ và $G(f(Y))$ để nhận được mệnh đề $H(f(Y))$. Như vậy, phép

hợp nhất cho phép biến đổi các mệnh đề sao cho có dạng trực kiện bù nhau bằng cách áp dụng các *phép thế* (substitutions).

a. Phép thế

Phép thế là một tập hợp hữu hạn các cặp $\mathbf{t_i} \square \mathbf{V_i}$, trong đó t_i là các *hạng*, còn V_i là các *biến* phân biệt. Nếu I = 0, ta nói phép thế là *rỗng*.

Người ta nói áp dụng một phép thế s = { $t_i \square V_i$ } cho một *biểu thức* bất kỳ E (E là một hạng hoặc một *CTC*) cho trước là xác định một trường hợp của E theo s. Đó là việc thay thế tất cả các vị trí ban đầu của mỗi biến V_i trong E bởi t_i.

***Ví dụ* :**

Cho E = G(f(X), a, Y) và các phép thế :

s_1 = { Z|X, U|Y } s_2 = { b|X }

s_3 = { Y|X, g(X)|Y } s_4 = { a|X, k(c)□Y }

Ta có :

Es_1 = G(f(Z), a, U) Es_2 = G(f(b), a, Y)

Es_3 = G(f(Y), a, g(X)) Es_4 = G(f(a), a, k(c))

***Chú ý* :**

- Để chuyển E thành Es_3, chỉ có các vị trí ban đầu của X và Y trong E là được thay thế (không phải các vị trí xuất hiện trong khi áp dụng s_3). Kết quả của phép thế này là *độc lập* với thứ tự áp dụng các phần tử của phép thế.
- Es_4 là một trường hợp cụ thể của E bởi phép thế s_4.
- Các hạng t_i và các biến V_i được gọi lần lượt là hạng và biến của phép thế.

Phép tổ hợp hai phép thế $s_1$và s_2, người ta viết quy ước $s_1 \circ s_2$, là phép thế nhận được bằng cách như sau :

a) Ap dụng s_2 cho các hạng của s_1.

b) Loại bỏ khỏi s_2 các cặp $t_j \square V_j$ sao cho V_j là một biến của s_1 (hay sao cho $\exists t_i$ với $t_i \square V_j \in s_1$).

c) Tập hợp tất cả các cặp nhận được từ hai bước a) và b) trên đây.

***Ví dụ* :**

{ a □ X, g(Y, Z, U) □ V } ∘ { b □ X, c □ Y, f(X) □ Z, k(d) □ V, f(X) □ W }

= { a □ X, g(c, f(X), U) □ V, c □ Y, f(X) □ Z, f(X) □ W }

Người ta chứng minh được rằng :

$(Es_1)\ s_2 = E(s_1 \circ s_2)$

Người ta cũng chứng minh được rằng phép thế có tính kết hợp, hay :

$(s_1 \circ s_2) \circ s_3 = s_1 \circ (s_2 \circ s_3)$

Trong trường hợp tổng quát, $s_1 \circ s_2 \neq s_2 \circ s_1$.

b. Bộ hợp nhất (unifier)

Người ta nói rằng một tập hợp $\{E_i\}_i$ các biểu thức (hạng hay công thức) là *hợp nhất được* (unifiable) bởi s hay nói rằng s là bộ hợp nhất của $\{E_i\}_i$ nếu và chỉ nếu mọi phép thế s cho E_i, ký hiệu E_is, là giống nhau. Khi đó, ta ký hiệu biểu thức (duy nhất) sinh ra bởi bộ hợp nhất s là $\{E_i\}$s.

***Ví dụ* :**

s = { a □ X, c □ Y, c □ V, b □ Z, b □ U, g(b) □ W }

là một hợp nhất của :

$\{E_i\}_i$ = { G(X, f(Y), g(b)), G(X, f(c), g(Z)), G(X, f(c), g(U)), G(X, f(V), W) }

vì rằng mọi biểu thức đều trở thành :

G(a, f(c), g(b))

Có thể có nhiều bộ hợp nhất cho cùng một tập hợp các biểu thức đã cho.

Người ta gọi r là *bộ hợp nhất tổng quát hơn* (**mgu** – more general unifier) của một tập hợp các biểu thức $\{E_i\}_i$ sao cho với mọi bộ hợp nhất s khác của $\{E_i\}_i$, tồn tại một phép thế s' sao cho s = r∘s'.

Người ta chứng minh được rằng với mọi tập hợp E hợp nhất được, sẽ tồn tại một **mgu** và, nếu r_1 và r_2 là hai **mgu** của $\{E_i\}$, thì $\{E_i\}r_1$ và $\{E_i\}r_2$ là giống nhau cho tên các biến.

***Ví dụ* :**

Trong ví dụ trên đây, **mgu** của { E_1, E_2, E_3 } là :

r = { c □ Y, c □ V, b □ Z, b □ U, g(b) □ W }

Chú ý rằng s = r∘{ a □ X }.

c. Thuật toán hợp nhất

Sau đây ta xây dựng thuật toán đệ quy UNIFY(E, □) với □ là phép thế rỗng. Thuật toán tạo ra một **mgu** cho một tập hợp hữu hạn E các biểu thức hợp nhất được. Nếu E không là hợp nhất được, thuật toán dừng và thông báo.

UNIFY(□, □)

1. **if** □ là một phần tử duy nhất (mọi phần tử giống hệt nhau)
   then begin write('□ là **mgu'); stop end**
   else
2. Tạo tập hợp ***D*** các xung đột của □
3. **If** tồn tại hai phần tử V và t của ***D*** sao cho V là một biến, t là một hạng và V không có mặt trong t
   then begin □ ← □ ∘ { t □ V }
   □ ← □ { t □ V }
   UNIFY(□, □)
   end
4. **write('**Tập hợp đã cho không là hợp nhất được**'); stop**

Hình 2.2. Thuật toán hợp nhất

Thuật toán sử dụng tập hợp các xung đột, ký hiệu ***D***, của một tập hợp □ các biểu thức (hạng hay công thức). Tập hợp ***D*** này được xây dựng bằng cách quét đồng thời từ trái qua phải mọi phần tử của □ cho đến khi gặp ký hiệu ở vị trí đầu tiên làm xuất hiện sự sai khác

giữa các phần tử này. Sau đó, bằng cách trích mỗi phần tử của □ một biểu thức (cần phải là hạng hay công thức ngay khi tất cả các hàm và tất cả các vị từ được viết bởi các ký hiệu phân biệt). Biểu thức này bắt đầu từ vị trí xung đột của ký hiệu. Tập hợp các biểu thức như vậy tạo thành ***D***.

Chẳng hạn, cho □ = { G(X, f(a, Y), G(X, b), G(X, f(a, G(Z))) }

Một xung đột xuất hiện ở vị trí ký hiệu thứ năm. Như vậy :

D = { f(a, Y), b, f(a, g(Z)) }

***Ví dụ* 1 :**

Cho E = { P(a, X, f(g(Y))), P(Z, f(Z), f(U)) }.

Gọi UNIFY(E, □), khi đó □ ← E, □ ← □

Bước 1 và 2 : ***D*** ← { a, Z }

Bước 3 : với V ← Z, t ← a

□ ← □ ∘ { a | Z } = { a | Z }

□ ← □ { a | Z } = { P(a, X, f(g(Y))), P(a, f(a), f(U)) }. Gọi UNIFY(□, □)

Bước 1 và 2 : ***D*** ← { X, f(a) }

Bước 3 : với V ← X, t ← f(a)

□ ← { a | Z } ∘ { f(a) □ X} = { a | Z, f(a) □ X }

□ ← □ { f(a) | X } = { P(a, f(a), f(g(Y))), P(a, f(a), f(U)) }

Gọi : UNIFY(□, □)

Bước 1 và 2 : ***D*** ← { U, g(Y) }

Bước 3 : với V ← U, t ← g(Y)

□ ← { a | Z, f(a) | X } ∘ { g(Y) □ U} = { a | Z, f(a) □ X, g(Y) □ U }

□ ← { P(a, f(a), f(g(Y))) }

Gọi : UNIFY(□, □)

Bước 1: Tìm thấy **mgu** là □ = { a | Z, f(a) □ X, g(Y) □ U }

***Ví dụ* 2 :**

Cho E = { Q(f(a), g(X)), Q(Y, Y) }. Gọi UNIFY(E, □), khi đó □ ← E, □ ← □

Bước 1 và 2 : ***D*** ← { f(a), Y }

Bước 3 : với V ← Y, t ← f(a)

□ ← □ ∘ { f(a) | Y } = { f(a) | Y }

□ ← □ { f(a) | Y } = { Q(f(a), g(X)), Q(f(a), f(a)) }

Gọi : UNIFY(□, □)

Bước 1 và 2 : ***D*** ← { g(X), f(a) }

Bước 4: Tập hợp E không là hợp nhất được.

***Ví dụ* 3 :**

Cho E = { G(X, f(b, X)), G(X, f(Z, g(X)))}

Ta phân biệt các biến để có : E' = { G(X, f(b, X)), G(Y, f(Z, g(Y))) }

Sau phép thế { X □ Y } ta được :

E' ° { X □ Y } = { G(X, f(b, X)), G(X, f(Z, g(X))) }

Ap dụng phép thế { b □ Z } ta được :

E' ∘ { X □ Y, b □ Z } = { G(X, f(b, X)), G(X, f(b, g(X))) }

Nhưng ta không thể thế g(X) cho X : phép hợp nhất không thực hiện được.

Chú ý trong thuật toán hợp nhất **:**

– Ở bước 3 trong thuật toán hợp nhất UNIFY(□, □), đòi hỏi V phải không xuất hiện trong t : phép thế t cho V sẽ trở nên phức tạp không xác định được, không có kết quả, các biểu thức vẫn luôn phải hợp nhất.
  Chẳng hạn :
  { f(X), X } được áp dụng cho G(X, a) và G(f(X), a)
  sẽ chuyển chúng thành G(f(X), a) và G(f(f(X)), a), v.v...

– Việc kiểm tra «V có mặt trong t không ?» làm cho độ phức tạp tính toán của thuật toán trở nên đáng kể. Trong trường hợp xấu nhất, là có bậc luỹ thừa đối với số phần tử của E.
  Chẳng hạn :
  $E = \{ P(X_1, X_2,..., X_n), P(f(X_0, X_0), f(X_1, X_1), f(X_2, X_2),..., f(X_{n-1}, X_{n-1})) \}$
  Với lời gọi đệ quy thứ nhất :
  $\square = \{ f(X_0, X_0) \square X_1 \}$
  $\square = \{ P(f(X_0, X_0), X_2,..., X_n),$
  $P(f(X_0, X_0), f(f(X_0, X_0), f(X_0, X_0)), f(X_2, X_2),..., f(X_{n-1}, X_{n-1})) \}$
  Với lời gọi đệ quy thứ hai :
  $\square = \{ f(X_0, X_0) \square X_1, f(f(X_0, X_0), f(X_0, X_0)) \square X_2 \}$
  Với lời gọi đệ quy thứ ba, tập hợp □ được thêm phần tử :
  $f(f(f(X_0, X_0), f(X_0, X_0)), f(f(X_0, X_0), f(X_0, X_0))) \square X_3$

và cứ thế tiếp tục...

Với lời gọi đệ quy thứ n, tham đối cuối cùng của phần tử thứ hai của □ sẽ có $2^{n+1}-1$ xuất hiện của hàm f. Việc kiểm tra «V có mặt trong t không ?» sẽ cần vượt qua tất cả các xuất hiện của f với thời gian là $O(2^n)$.

– Tồn tại các thuật toán hợp nhất có độ phức tạp tính toán nhỏ, có thời gian tuyến tính. Tuy nhiên, những thuật toán này chỉ có ý nghĩa nếu bậc của các vị từ và của các hạng cần hợp nhất là lớn.

II.2.2. Hợp giải các mệnh đề bất kỳ

Giả sử cho hai mệnh đề cha G và H không có biến chung (theo bước **e**, mục II.1.2) và có thể tiến hành hợp giải. Khi đó, tồn tại kết quả hợp giải của G và H. Gọi G, H và kết quả hợp giải của chúng là các tập hợp các trực kiện $\{G_i\}$ và $\{H_j\}$.

Giả sử tồn tại một tập hợp con $\{G_{i'}\}$ của $\{G_i\}$ (i' nhận một số giá trị của i), và một tập hợp con $\{ H_{j'}\}$ của $\{H_j\}$ sao cho tập hợp các trực kiện :

$\boldsymbol{L} = \{ G_{i'}\} \cup \{ \neg H_{j'} \}$

là hợp nhất được.

Giả sử r là một **mgu** của $\boldsymbol{L}$, một kết quả hợp giải của G và H là một tập hợp các trực kiện được xác định như sau :

$(\{G_i\} - \{G_{i'}\})\, r \cup (\{H_j\} - \{H_{j'}\})\, r$

Ví dụ **:** Gỉa sử

$G = P(X, f(a)) \vee P(X, f(Y)) \vee Q(Y)$ và $H = \neg P(Z, f(a)) \vee \neg Q(Z)$

– Một **mgu** của $G_1 = P(X, f(a))$ và $\neg H_1 = P(Z, f(a))$ là $r_1 = \{X \square Z\}$
  Từ đó kết quả hợp giải là $P(X, f(Y)) \vee Q(Y) \vee \neg Q(X)$

– Một **mgu** của $G_2 = P(X, f(Y))$ và $\neg H_1$ là $r_2 = \{X \square Z, a \square Y\}$
  Từ đó kết quả hợp giải là $P(X, f(a)) \vee Q(a) \vee \neg Q(X)$

– Một **mgu** của $G_3 = Q(Y)$ và $\neg H_2 = Q(Z)$ là $r_3 = \{Y \square Z\}$
  Từ đó kết quả hợp giải là $P(X, f(a)) \vee P(X, f(Y)) \vee P(Y, f(a))$
– Một **mgu** của G_1, G_2 và $\neg H_1$ là $r_4 = \{X \square Z, a \square Y\} = r_2$
  Từ đó kết quả hợp giải là $Q(a) \vee \neg Q(X)$.

Ta thấy có bốn kết quả hợp giải khác nhau. Trong các trường hợp khác, không tồn tại kết quả hợp giải.

Tuy nhiên, trước khi kết luận rằng không tồn tại kết quả hợp giải, ít ra cũng phải kiểm tra các mệnh đề đã cho có các biến phân biệt nhau. Chẳng hạn :

$G(X, a)$ và $\neg G(f(X), X)$

không có kết quả hợp giải. Nếu viết lại mệnh đề thứ hai thành $\neg G(f(Y), Y)$ thì xuất hiện một mpu $\{ f(X) \square X, a \square Y\}$, từ đó có thể suy ra tồn tại một kết quả hợp giải là mệnh đề rỗng.

Như vậy, sự phân biệt các tên biến của các mệnh đề cha chỉ hợp pháp lúc gọi thuật toán hợp nhất. Việc tìm một kết quả hợp giải giữa $G(X, X)$ và $\neg G(f(X), X)$ đòi hỏi phải viết lại mệnh đề thứ hai thành $\neg G(f(Y), Y)$. Lúc này, phép thế $\{ f(X) \square X \}$ dẫn đến kết quả là hai mệnh đề $G(f(Y), f(Y))$ và $\neg G(f(Y), Y)$ không hợp nhất được.

II.2.3. Một cách trình bày khác của phép hợp giải

Phép hợp giải còn được trình bày theo một cách khác, gọi là phép *nhân tử hoá* (factorisation).

Trước tiên, người ta đưa vào *phép hợp giải nhị phân* (binary resolution principle) như sau. Cho G và H là hai mệnh đề có các biến phân biệt, được ký hiệu bởi các tập hợp các trực kiện $\{G_i\}$ và $\{H_j\}$.

Giả sử G_p và H_p là hai trực kiện của $\{G_i\}$ và $\{H_j\}$ một cách tương ứng sao cho G_p và $\neg H_q$ là hợp nhất được. Giả sử r là một **mgu** của G_p và $\neg H_q$. Người ta định nghĩa *kết quả hợp giải nhị phân* (binary resolvent) của $\{G_i\}$ và $\{H_j\}$ là tập hợp các trực kiện :

$$(\{G_i\} - \{G_p\})\, r \cup (\{H_j\} - \{H_q\})\, r$$

Ta thấy rằng *kết quả hợp giải nhị phân* là một trường hợp đặc biệt của phép hợp giải đã giới thiệu ở mục II.2.2 trên đây.

Bây giờ ta đưa vào một luật suy diễn thứ hai được gọi là *phép nhân tử hoá*. Giả sử G là một mệnh đề gồm một tập hợp các trực kiện.

Giả sử $\{L_i\}_{i = 1..\, p}$ là hai hay nhiều trực kiện của G hợp nhất được bởi một **mgu** r. Xuất phát từ G, bởi *phép nhân tử hoá*, người ta có thể suy ra mệnh đề *nhân tử* (factor) của G như sau :

$$(G - \{L_i\}_{i = 2..\, p})\, r$$

Nghĩa là ta đã cho phát sinh Gr nhưng trộn lẫn tất cả các trực kiện $L_i r$ thành một trực kiện duy nhất.

***Ví dụ* :**

Từ $G = P(X) \vee P(Y) \vee Q(b)$, ta cho phát sinh nhân tử $P(X) \vee Q(b)$.

Cuối cùng, ta đưa vào phép hợp giải (không nhị phân) như là một luật suy diễn mới bằng cách tổ hợp các phép triển khai trong luật hợp giải nhị phân và trong luật nhân tử hoá. Xuất phát từ hai mệnh đề G và H, phép hợp giải cho phép phát sinh, hoặc một kết quả hợp giải nhị phân của G và H, hoặc một kết quả hợp giải nhị phân của G và một nhân tử của H, hoặc một

kết quả hợp giải nhị phân của H và một nhân tử của G, hoặc một kết quả hợp giải nhị phân của một nhân tử của G và một nhân tử của H.

Các kết quả hợp giải nhận được theo định nghĩa thứ nhất (mục II.2.2) cũng có thể nhận được theo định nghĩa thứ hai vừa trình bày.

II.3. Các tính chất tổng quát của phép hợp giải

a. Một luật đúng đắn

Ta có thể chứng minh dễ dàng rằng phép hợp giải là một luật đúng đắn. Nghĩa là mọi kết quả hợp giải là hậu quả logic của hai mệnh đề cha.

Từ đó suy ra rằng, nếu một dãy các hợp giải dẫn đến một mệnh đề rỗng, thì tập hợp G đã cho ban đầu là không nhất quán. Nếu không, sẽ tồn tại một diễn giải I làm thoả mãn G và do đó, các mệnh đề cha H_1 và H_2 là rỗng và vừa là hậu quả logic. Nếu r là **mgu** dùng để hợp giải H_1 và H_2, thì diễn giải I phải thoả mãn H_1r và H_2r, và do H_1r sẽ là phần bù của H_2r nên diễn giải I không thể tồn tại. Vậy G là không nhất quán.

***Chú ý* :**

Cho G là tập hợp các mệnh đề và giả sử dãy $G_1',..., G_p'$ là các mệnh đề phân biệt sao cho :

G_p' là mệnh đề rỗng,

$\forall i = 1..p$, hoặc G_i' thuộc về G, hoặc G_i' là một kết quả hợp giải của hai mệnh đề cha đứng trước đó trong dãy.

Một dãy $\{G_i'\}$ như vậy để chứng minh tính không nhất quán của G được gọi là một *bác bỏ* (refutation) bởi hợp giải của G.

b. Tính hoàn toàn của phép hợp giải đối với phép bác bỏ

Người ta chỉ ra rằng nếu một tập hợp các mệnh đề là không nhất quán, thì sẽ tồn tại một dãy hữu hạn các áp dụng của phép hợp giải cho các mệnh đề này và cho các kết quả hợp giải xuất hiện, để từ đó dẫn đến mệnh đề rỗng.

Thật vậy, khi một *CTC* là hậu quả logic của một tập hợp G các *CTC*, điều này tương đương với tính không nhất quán của một dạng mệnh đề $G \wedge \neg H$. Như vậy, nếu một *CTC* H là hậu quả logic của một tập hợp G các *CTC*, thì sẽ tồn tại một phép bác bỏ cho tất cả các dạng $G \wedge \neg H$.

Đây là *tính hoàn toàn* (complétude) đối với phép bác bỏ : nhóm các luật suy diễn được rút gọn thành một luật duy nhất, như vậy phép hợp giải là *đầy đủ đối với phép bác bỏ*.

Ta có thể giải thích tính hoàn toàn của phép hợp giải đối với phép bác bỏ theo cách sau :

Gọi $\boldsymbol{R}$ (G) là hợp của một tập hợp mệnh đề G với tất cả các kết quả hợp giải của mọi cặp các mệnh đề có thể.

$\boldsymbol{R}^{p+1}$(G) là tập hợp các mệnh đề $\boldsymbol{R}(\boldsymbol{R}^{p}$(G)).

Tính hoàn toàn của phép hợp giải đối với phép bác bỏ có nghĩa rằng : $\forall$G tập hợp không nhất quán các mệnh đề, $\exists$n là một số nguyên dương sao cho mệnh đề rỗng thuộc về $\boldsymbol{R}^{n}$ (G).

***Chú ý* :**

- Thông thường, người ta rút gọn khi nói về tính hoàn toàn của phép hợp giải. Chú ý rằng không phải là tính hoàn toàn đối với sự suy diễn mà ta đã định nghĩa trước đây (nếu H là một hậu quả logic của một nhóm G các *CTC*, tính chất được thoả mãn bởi hợp giải

sẽ không là «tồn tại một một dãy hữu hạn các áp dụng của luật suy diễn xuất phát từ G để tạo ra H». Trước tiên, luật chỉ áp dụng đối với các mệnh đề có dạng $G \wedge \neg H$ mà không phải đối với *CTC* bất kỳ của G, để từ đó dẫn đến mệnh đề rỗng, mà không phải là H).

- Phép hợp giải nhị phân không là hoàn toàn đối với phép bác bỏ.

  Thật vậy, xét tập hợp không nhất quán các mệnh đề :

  $$E = \{ P(X) \vee P(Y), \neg P(W) \vee \neg P(Z) \}$$

  Từ đây ta áp dụng phép hợp giải nhị phân và nhận được các kết quả hợp giải nhị phân luôn luôn có hai trực kiện : ta sẽ không bao giờ nhận được mệnh đề rỗng.

- Trái lại, tập hợp các đôi luật hợp giải nhị phân và nhân tử hoá tạo thành một hệ thống đầy đủ đối với phép bác bỏ. Trong ví dụ trên, tính không nhất quán của E có thể được chứng minh bởi áp dụng hai luật này. Chẳng hạn, thực hiện hai phép bác bỏ rồi một phép hợp giải. Cần nhớ rằng tính không nhất quán của E cũng có thể được chứng minh bởi áp dụng chỉ mỗi phép hợp giải (không phải nhị phân).

- Sau đây là một ví dụ chỉ ra rằng không phải phép hợp giải nhị phân, cũng không phải hệ thống hai luật hợp giải nhị phân và nhân tử hoá, cũng không phải phép hợp giải (không phải nhị phân) là đầy đủ đối với phép suy diễn :

Cho G(X) và H(X) là hậu quả logic của { G(X), H(X) }, nhưng không có luật nào trong hệ thống ba luật suy diễn trên đây áp dụng được cho { G(X), H(X) }. Do vậy không có luật nào là đầy đủ đối với phép suy diễn.

III. Các hệ thống bác bỏ bởi hợp giải

Nếu G là một tập hợp các mệnh đề và phép hợp giải là một luật suy diễn đúng đắn và đầy đủ (đối với phép bác bỏ), ta có thể khẳng định rằng :

«G không nhất quán» cũng có nghĩa là «tồn tại một phép bác bỏ của G bởi hợp giải».

Tương tự, nếu G là tập hợp các *CTC* : «*CTC* H là hậu quả logic của G» tương đương với «tồn tại một phép bác bỏ của tất cả các dạng mệnh đề $G \wedge \neg H$».

***Ví dụ* :**

Xét các *CTC* : $(\forall X)\,(\exists Y)\,(\neg P(X) \rightarrow Q(Y)$ et $(\forall Z)\,(P(b) \rightarrow R(a, z))$,
nếu ta muốn chứng minh rằng : $\neg R(a, b) \rightarrow (\exists U)\,Q(U)$,

thì ta có thể tìm kiếm phép hợp giải từ tập hợp các mệnh đề :

$$\{ P(X) \square Q(f(X)), \neg P(b) \square R(a, Z), \neg R(a, b), \neg Q(U) \}$$

Sau đây ta sẽ tìm các phương pháp luận của nguyên lý hợp giải để xây dựng phép bác bỏ.

III.1. Thủ tục tổng quát bác bỏ bởi hợp giải

Giả sử rằng ta muốn chứng minh rằng *CTC* H là hậu quả lôgic của tập hợp các mệnh đề :

$$G = \{ G_i \}$$

Thủ tục bác bỏ tổng quát *không đơn định* (non-deterministic) như sau :

CHỨNG–MINH–BỞI–HỢP–GIẢI (G, H)
1. Tạo C là tập hợp các dạng mệnh đề đối với các *CTC* của G
2. Thêm vào tập hợp C một dạng mệnh đề ¬H
3. **while** C ≠ □ **do** ; □ là mệnh đề rỗng

begin
Chọn hai mệnh đề phân biệt p và q của C (3.1)

```
    if      p và q là các kết quả hợp giải          (3.2)
    then    chọn p hoặc q de thêm vào C
end
```

Hình 2.3. Thủ tục tổng quát bác bỏ bởi hợp giải

Chú ý **:**

- Căn cứ vào các tính chất của *tính hoàn toàn đối với phép bác bỏ* của nguyên lý hợp giải, nếu H thực sự hậu quả lôgic của G, thì sẽ tồn tại ít nhất một dãy các lựa chọn (bước 3.1) dẫn đến mệnh đề rỗng và do đó thủ tục dừng.
- Thủ tục trên đây không đảm bảo rằng một dãy như vậy nếu quả thật tồn tại thì sẽ tìm thấy. Các giai đoạn 3.1 và 3.2 sẽ không đơn định và rất thô sơ. Chẳng hạn, thủ tục này giải nhiều lần cùng một cặp mệnh đề từ cùng trực kiện.

III.2. Chiến lược hợp giải

Thủ tục tổng quát CHỨNG–MINH–BỞI–HỢP–GIẢI có thể được làm mịn hơn tại các bước 3.1 và 3.2. Theo cách chọn lựa các mệnh đề và các trực kiện của chúng, người ta có thể định nghĩa nhiều *chiến lược bác bỏ bởi hợp giải*. Một trong những chiến lược này lả hạn chế hay sắp xếp các lựa chọn mệnh đề và trực kiện để hợp giải tuỳ theo tiêu chuẩn đặc thù của chúng.

Một chiến lược hợp giải được gọi là *đầy đủ đối với phép bác bỏ* nếu tồn tại một phép bác bỏ bởi hợp giải của tập hợp G ban đầu, nghĩa là tồn tại một dãy hợp giải dẫn đến mệnh đề rỗng, thì cũng tồn tại một hợp giải làm thoả mãn các tiêu chuẩn lựa chọn hay thứ tự riêng cho chiến lược này.

Người ta cũng biểu diễn tính *hoàn toàn đối với phép bác bỏ* của một chiến lược hợp giải S theo cách sau. Ký hiệu :

$\boldsymbol{R}_c(G)$ phép hợp của tập hợp mệnh đề G với tất cả các kết quả hợp giải (resolvent) của tất cả các cặp mệnh đề của G, có thể nhận được bằng cách tuân theo các tiêu chuẩn biểu diễn bởi C, gắn với chiến lược S.

$\boldsymbol{R}_c^{p1}(G) = \boldsymbol{R}_c(\boldsymbol{R}_c^{p}(G))$

Chiến lược hợp giải S là đầy đủ nếu và chỉ nếu với mọi tập hợp không nhất quán các mệnh đề, tồn tại một số nguyên dương n sao cho mệnh đề rỗng □ thuộc $\boldsymbol{R}_c^{p}(G)$:

Chú ý **:**

- Tính hoàn toàn của một *chiến lược hợp giải* (đối với phép bác bỏ) không đồng nghĩa với tính hoàn toàn của một *nguyên lý hợp giải* (đối với phép bác bỏ).
- Các chiến lược không đầy đủ có thể có ích lợi thực tiễn.
- Một chiến lược là đầy đủ không có nghĩa là khi làm thoả mãn các tiêu chuẩn lựa chọn hay thứ tự tìm thấy một cách tất yếu, với một số hữu hạn các mệnh đề, mệnh đề rỗng xuất phát từ tất cả tập hợp G không nhất quán các mệnh đề. Sau đây, khi một chiến lược có tính chất này, người ta nói là *đầy đủ trực tiếp*.

III.2.1. Đồ thị định hướng, đồ thị tìm kiếm và đồ thị bác bỏ

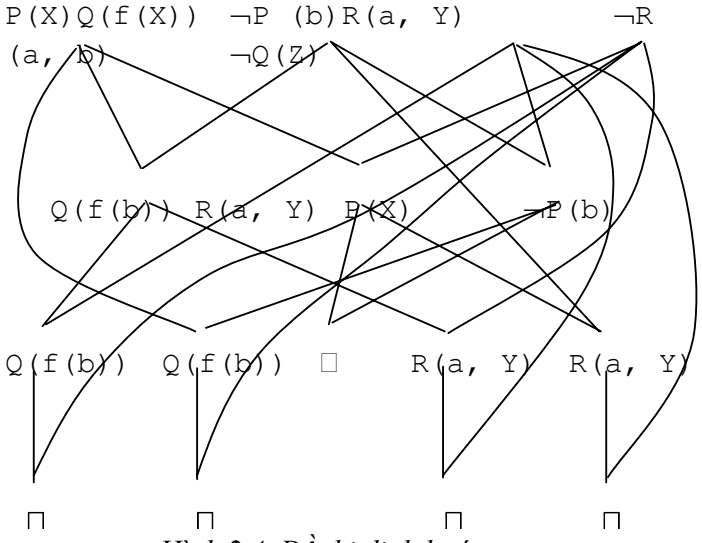
Với mọi hệ thống các luật suy diễn, cho trước một hệ tiên đề, người ta có thể biểu diễn tập hợp các định lý được suy ra (và cách suy ra chúng) dưới dạng một đồ thị được gọi là *đồ thị định hướng*.

Trong trường hợp của nguyên lý hợp giải, các đỉnh là các mệnh đề và các cung nối các mệnh đề cha đến các mệnh đề kết quả resolvents.

Ví dụ, cho tập hợp các mệnh đề :

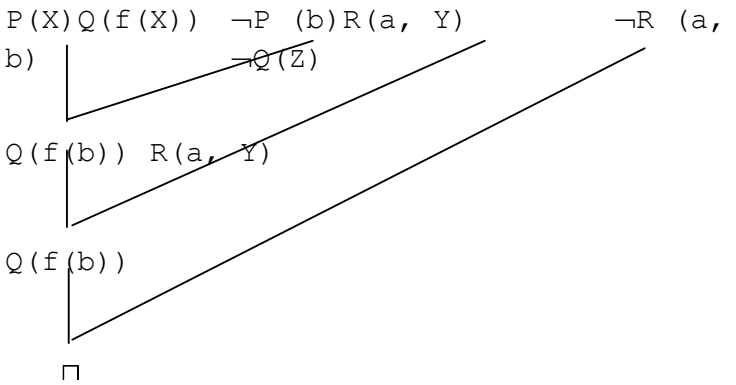
{ P(X) □ Q(f(X)), ¬P(b) □ R(a, Y), ¬R(a, b), ¬Q(Z) }

ta có thể vẽ đồ thị định hướng như sau :



Hình 2.4. Đồ thị định hướng

Ký hiệu □ chỉ mệnh đề rỗng □. Để đồ thị được rõ ràng, người ta đặt song song các kết quả hợp giải được dẫn đến từ nhiều cặp mệnh đề cha và không vẽ mũi tên cho các cung. Từ đồ thị định hướng này, ta có thể trích ra các đồ thị bác bỏ, chẳng hạn đồ thị sau :



Hình 2.5. Đồ thị bác bỏ

Sau đây ta sẽ xem xét một cách ngắn gọn một số chiến lược tương ứng với nhiều cách tìm kiếm các đồ thị bác bỏ bên trong một đồ thị định hướng. Phần đồ thị định hướng rõ ràng để tìm ra một đồ thị bác bỏ được gọi là *đồ thị tìm kiếm*.

III.2.2. Chiến lược hợp giải bởi bác bỏ theo chiều rộng

(breadth first resolution)

Trong chiến lược này, người ta gán cho mỗi mệnh đề một độ sâu có giá trị :

0 đối với các mệnh đề ban đầu,

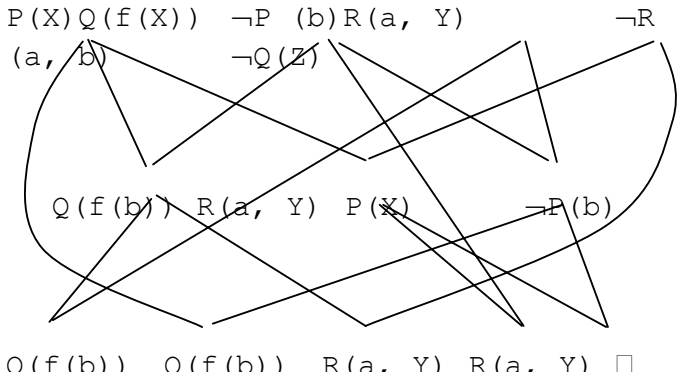
1 đối với các kết quả hợp giải của các cặp mệnh đề có độ sâu 0,

...

n đối với các kết quả hợp giải của các cặp mệnh đề có độ sâu n−1.

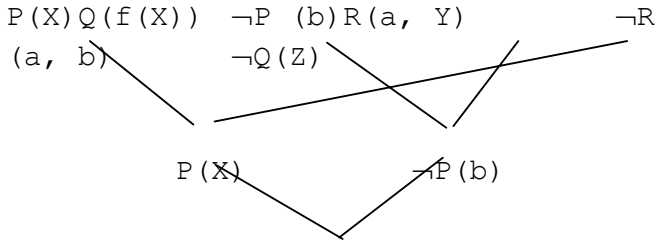
Cách người ta đặt độ sâu là làm sao cho không tạo ra một mệnh đề có độ sâu p khi mà có khả năng tạo ra một mệnh đề có độ sâu p̃1. Rõ ràng rằng khi một kết quả hợp giải xuất hiện lần đầu tiên, thì nó sẽ có độ sâu sâu nhất có thể nhận được từ hai mệnh đề cha.

Ví dụ trên đây có thể áp dụng chiến lược hợp giải bởi bác bỏ theo chiều rộng để dẫn đến tại thời điểm dừng một đồ thị tìm kiếm như sau :



Hình 2.6. Đồ thị tìm kiếm

Từ đồ thị tìm kiếm này, ta có thể trích ra đồ thị bác bỏ sau :



Hình 2.7. Đồ thị bác bỏ

Cần chú ý rằng :

- Chiến lược này không cấm việc tạo ra một phần tử nào đó của đồ thị định hướng của tập hợp mệnh đề G, nhưng chi phối tính rõ ràng của đồ thị định hướng bằng cách làm xuất hiện tập hợp $\boldsymbol{R}^{p}(G)$ trước $\boldsymbol{R}^{p1}(G)$. Chiến lược này là đầy đủ.
- Mặt khác, từ G hữu hạn, người ta suy ra rằng $\boldsymbol{R}^{p}(G)$ là hữu hạn, $\forall p = 1..n$. Như vậy, chiến lược hợp giải bởi bác bỏ theo chiều rộng là đầy đủ trực tiếp : nếu G không nhất quán, thì chiến lược này sẽ làm xuất hiện mệnh đề rỗng.
- Người ta nhận được một phép bác bỏ với mệnh đề rỗng có độ sâu sâu nhất có thể.

III.2.3. Chiến lược hợp giải bởi bác bỏ với «tập hợp trợ giúp»

Cho G là một tập hợp không thoả mãn các mệnh đề và T là một tập hợp con của G, sao cho tập hợp các mệnh đề G̃T là nhất quán.

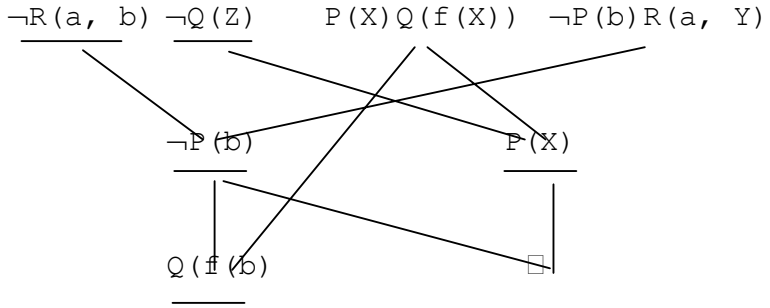
Theo định nghĩa, tập hợp trợ giúp của G quan hệ với T (giả thiết thoả mãn tính chất trước đây) gồm các mệnh đề của đồ thị định hướng,hoặc thuộc về T, hoặc là con cháu của T.

Một chiến lược hợp giải (đối với một tập hợp mệnh đề G đã cho) sử dụng tập hợp trợ giúp (set of support resolution) quan hệ với T (GT giả sử nhất quán) sao cho với mỗi hợp giải, ta lấy một mệnh đề cha trong tập hợp trợ giúp quan hệ với T.

Một chiến lược như vậy không hạn chế tính rõ ràng của đồ thị định hướng như chiến lược theo chiều rộng, làm thu hẹp đồ thị tìm kiếm được sinh ra. Ta chứng minh được rằng các chiến lược này là đầy đủ.

Ta có thể tạo ra các kết quả hợp giải mà không thu hẹp đồ thị hơn nữa, bằng cách kiểm tra độ sâu theo cách của chiến lược theo chiều rộng. *Trong những điều kiện như vậy*, các chiến lược hợp giải với tập hợp trợ giúp là đầy đủ trực tiếp.

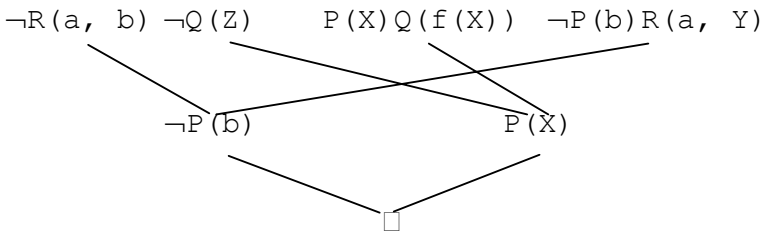
Sau đây, ta sẽ biểu diễn một đồ thị tìm kiếm nhận được theo cách này.



Hình 2.8. Đồ thị tìm kiếm

Xuất phát từ ví dụ đầu mục III.2.1. Lúc đầu, tập hợp trợ giúp chỉ có các mệnh đề ¬R(a, b) và ¬Q(Z). Các mệnh đề của đồ thị tìm kiếm thuộc về tập hợp trợ giúp được gạch chân.

Từ đó, đồ thị bác bỏ được trích ra như sau :



Hình 2.9. Đồ thị bác bỏ

Chú ý rằng trong ví dụ này, đồ thị tìm kiếm được chứa trong đồ thị nhận được bởi chiến lược theo chiều rộng, và cũng là đồ thị bác bỏ.

Để tiến hành chiến lược tập hợp trợ giúp, cần chọn T sao cho $\tilde{G}T$ là nhất quán. Sau đây là ba phương pháp đơn giản :

1. Định nghĩa T như là tập hợp các mệnh đề không chứa các trực kiện phủ định hay trực kiện âm (một trực kiện phủ định, negative literal, là một nguyên tử có dấu phủ định ¬ đặt trước).

   $\tilde{G}$ không thể rỗng, nếu không một diễn giải làm cho các trực kiện của T có giá trị T (true) sẽ là một mô hình của G, mô hình này không nhất quán.

   Mỗi mệnh đề của G-T sẽ chứa một trực kiện phủ định. Tập G-T sẽ nhất quán vì rằng một diễn giải làm cho tất cả các trực kiện phủ định của G-T có giá trị T sẽ là một mô hình của G-T.

2. Định nghĩa T như là tập hợp các mệnh đề không chứa các trực kiện khẳng định (hay trực kiện dương). Như vậy $\tilde{G}$ là nhất quán theo cách suy luận tương tự trên đây.

3. Định nghĩa T như là tập hợp các mệnh đề nhận được bằng cách phủ định *CTC* cần chứng minh. Trong trường hợp này, $\tilde{G}$ là tập hợp các tiên đề mà dễ dàng được xem là nhất quán.

Nếu một tập hợp mệnh đề G là không nhất quán, thì G phải chứa ít nhất một mệnh đề mà các trực kiện đều dương (nếu không, mọi trực kiện của G sẽ chứa ít nhất một trực kiện âm, ta có thể xây dựng một mô hình của G như đã chỉ ra ở phương pháp 1). Từ đó, ta có thể áp dụng chiến lược tập hợp trợ giúp theo phương pháp 1 trên đây. Tương tự như vậy, G phải chứa ít nhất một mệnh đề mà các trực kiện đều âm, và do đó, ta có thể áp dụng chiến lược tập hợp trợ giúp theo phương pháp 2. Ta có thể kiểm tra trong ví dụ trên đây rằng tập hợp T đã chọn thoả mãn phương pháp 2 và cả phương pháp 3.

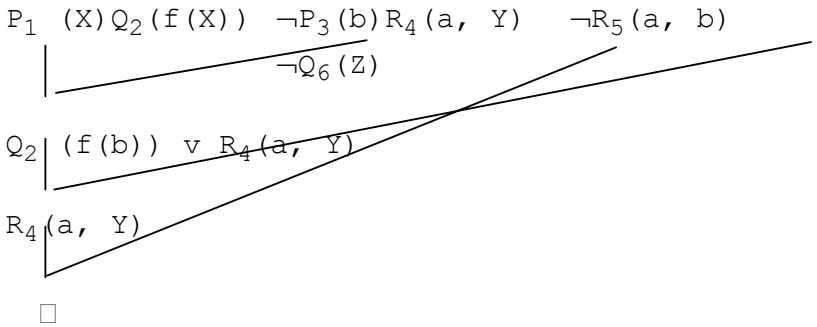
III.2.4. Chiến lược hợp giải bởi bác bỏ dùng «khoá»
(lock resolution)

Người ta đánh số một cách ngẫu nhiên tất cả các trực kiện của các mệnh đề cho trước. Phép hợp giải của hai mệnh đề chỉ có thể thực hiện được khi số thứ tự của chúng là nhỏ nhất. Các trực kiện của các mệnh đề kết quả (resolvents) thừa kế số thứ tự của các mệnh đề cha (trong trường hợp xảy ra trùng khả năng, người ta lấy lại số thấp nhất).

Chiến lược dùng khoá là đầy đủ.

Mặt khác, có thể sắp đặt, mà không thu hẹp hơn nữa, việc tạo sinh các kết quả hợp giải bằng cách kiểm tra độ sâu của chúng nhờ chiến lược theo chiều rộng. *Trong những điều kiện như vậy*, các chiến lược khoá là đầy đủ trực tiếp.

Tiến hành đánh số một cách ngẫu nhiên các trực kiện của các mệnh đề đã cho trong ví dụ đầu mục III, ta xây dựng được đồ thị tìm kiếm như sau :



Hình 2.10. Đồ thị tìm kiếm

Ở đây, ta thấy đồ thị bác bỏ phủ hoàn toàn lên đồ thị tìm kiếm.

III.2.5. Chiến lược hợp giải bởi bác bỏ là «tuyến tính»

Cho G là tập hợp mệnh đề không nhất quán, chọn C_0 là mệnh đề sao cho $\tilde{G}\{C_o\}$ tạo thành tập hợp mệnh đề nhất quán, khi đó C_0 được gọi là mệnh đề trung tâm xuất phát.

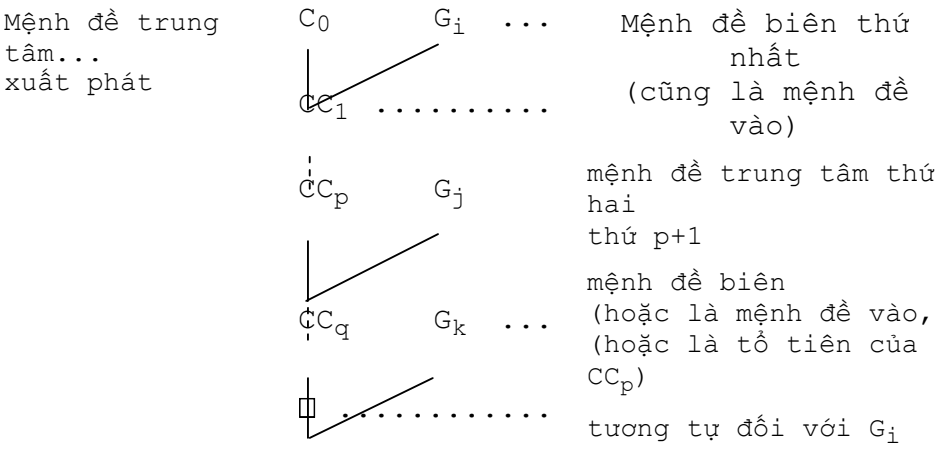
Mọi kết quả hợp giải cho phép sẽ có C_0 là tổ tiên và được gọi là các *mệnh đề trung tâm* (central clauses).

Chiến lược tuyến tính (linear resolution) chỉ cho phép tiến hành hợp giải giữa một mệnh đề trung tâm CC (lúc đầu C_0 là trung tâm) và một *mệnh đề biên* CB được chọn từ :

- Hoặc trong số các mệnh đề của G (được gọi là các mệnh đề vào).
- Hoặc trong số kết quả hợp giải tổ tiên (là các mệnh đề trung tâm) của CC.

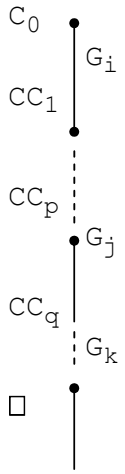
Chú ý rằng một mệnh đề trung tâm chỉ có thể được hợp giải với một mệnh đề trung tâm khác nếu một trong chúng là tổ tiên của mệnh đề kia.

Phép bác bỏ được vẽ theo sơ đồ sau :



Hình 2.11. Sơ đồ bác bỏ

Hay, bằng cách thay thế tên các mệnh đề biên trên các cung nối các mệnh đề trung tâm, ta nhận được đồ thị như sau :

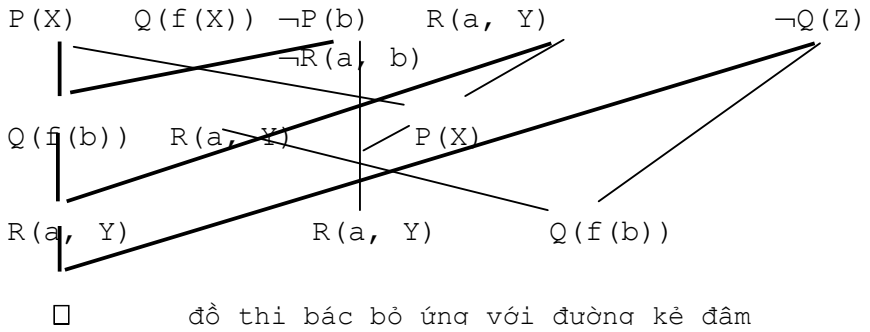


Hình 2.12. Sơ đồ chiến lược tuyến tính

Do đồ thị có dáng một đường thẳng nên người ta gọi đây là chiến lược *tuyến tính* (linear strategy). Tính chất tuyến tính ở chỗ hai mệnh đề trung tâm chiều có thể được hợp giải khi một trong chúng là tổ tiên của mệnh đề kia.

Ví dụ **:**

- Đồ thị tìm kiếm cho ở mục III.2.2 bởi áp dụng chiến lược theo chiều rộng không thể sinh ra bởi chiến lược tuyến tính, dù với cách chọn C_0 nào trong số các mệnh đề ban đầu. Đồ thị bác bỏ cũng không phải là tuyến tính.
- Tương tự như vậy đối với đồ thị tìm kiếm và đồ thị bác bỏ cho ở mục III.2.3 bởi áp dụng chiến lược tập hợp trợ giúp.
- Đồ thị tìm kiếm ở mục III.2.4 được tạo bởi áp dụng chiến lược dùng khoá được xem là kết quả của việc áp dụng chiến lược tuyến tính xuất phát từ mệnh đề :
  $C_0 = P(X) \vee Q(f(X))$ hay $C_0 = \neg P(b) \vee R(a, Y)$
  Lúc này, đồ thị bác bỏ nhận được là lẫn lộn với đồ thị tìm kiếm.
- Sau đây là đồ thị tìm kiếm nhận được từ một chiến lược tuyến tính xuất phát từ mệnh đề $C_0 = P(X) \vee Q(f(X))$ nhưng không lẫn lộn với đồ thị bác bỏ :



Hình 2.13. Đồ thị tìm kiếm chiến lược tuyến tính

Đồ thị tìm kiếm trên đây nhận được bằng cách lái theo chiều rộng việc áp dụng chiến lược tuyến tính xuất phát từ P(X) □ Q(f(X)).

Các chiến lược tuyến tính là đầy đủ.

Nếu ta sắp đặt việc tạo ra các mệnh đề trung tâm nhờ chiến lược theo chiều rộng, thì chúng là đầy đủ trực tiếp.

Nhớ lại rằng tính hoàn toàn căn cứ vào giả thiết mệnh đề trung tâm xuất phát C_0 sao cho $G - \{C_0\}$ là nhất quán. Thông thường, người ta chọn C_0 là một trong các mệnh đề lấy ra từ phép phủ định của *CTC* mà người ta muốn chứng minh, nghĩa là phủ định của *sự giả định* hay *sự phỏng đoán* (conjecture).

Ta có thể giả thiết rằng $G - \{C_0\}$ là một tập hợp mệnh đề nhất quán, chỉ gồm các mệnh đề dạng tiên đề và chỉ có một số mệnh đề có tính phỏng đoán. Nếu phủ định sự phỏng đoán chỉ nhận biết được một mệnh đề, thì $G - \{C_0\}$ phải là nhất quán vì rằng đó là tập hợp các tiên đề.

Bây giờ giả thiết rằng phủ định sự phỏng đoán chỉ nhận biết được hai mệnh đề, chẳng hạn C_1 và C_2. Sự phỏng đoán sẽ có dạng : $\neg C_1 \vee \neg C_2$. Nếu $G - \{C_0\}$ là không nhất quán, ta có $\neg C_2$ là hậu quả logic của các tiên đề, có nghĩa rằng sự phỏng đoán $\neg C_1 \vee \neg C_2$ là không chính xác : ta có thể đề nghị sự phỏng đoán $\neg C_2$ là thích hợp hơn cả.

Nếu vẫn còn hoài nghi về kết quả khắc phục này, người ta có thể vận dụng chiến lược tuyến tính xuất phát từ, một cách liên tiếp, mỗi mệnh đề có được từ phỏng đoán.

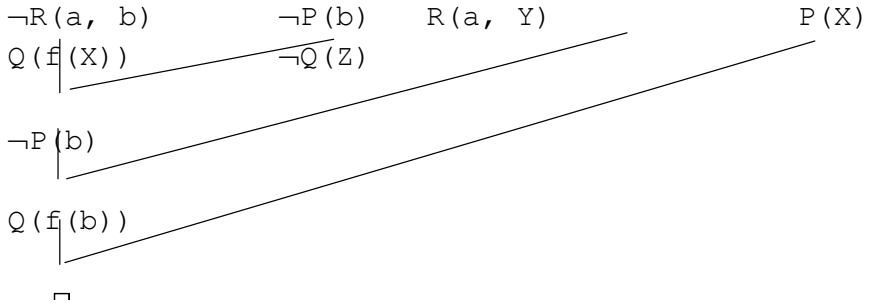
Quay lại ví dụ vừa rồi, từ 4 mệnh đề ban đầu đã cho :

{ P(X) □ Q(f(X)), ¬P(b) □ R(a, Y), ¬R(a, b), Q(Z) }

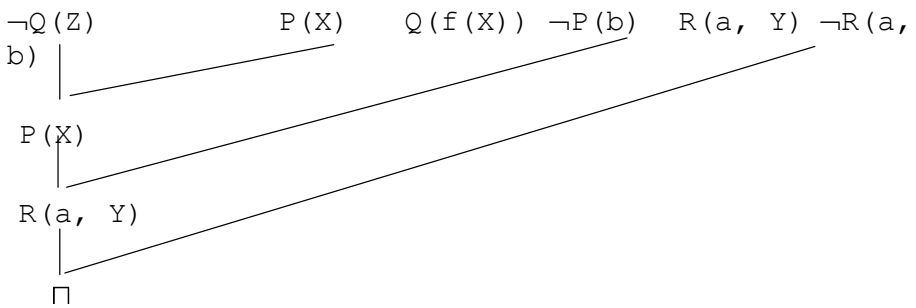
ta thấy hai mệnh đề cuối cùng là kết quả của phép phủ định phỏng đoán :

¬R(a, b) → (∃U) Q(U)

Nếu lấy mệnh đề xuất phát là C_0 = ¬R(a, b) và nếu ta áp dụng chiến lược tuyến tính và theo chiều rộng, thì ta nhận được :



Tương tự, xuất phát từ C_0 = ¬Q(Z), ta có :



Hình 2.14. Đồ thị tìm kiếm chiến lược tuyến tính

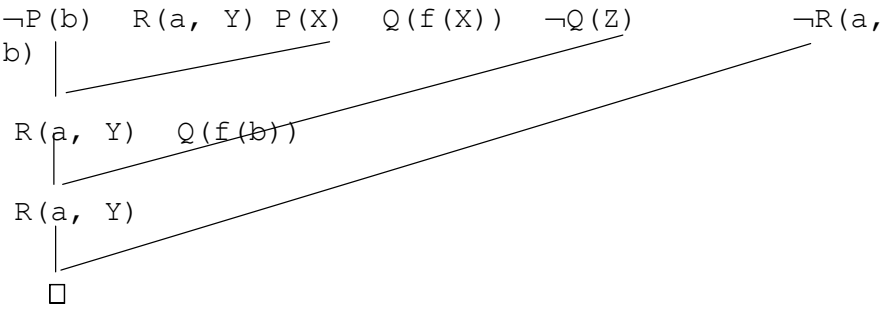
***Chú ý* :**

Trên đây ta đã giới thiệu một bác bỏ tuyến tính xuất phát từ mệnh đề :

C_0 = P(X) ∨ Q(f(X))

mà mệnh đề này không là kết quả của phép phủ định phỏng đoán. Thành công của bác bỏ, ngay cả khi tính đến tính không nhất quán của G đã không được bảo đảm ngay từ đầu. Thực tế, người ta có thể kiểm tra sau đó rằng tập hợp G của bốn mệnh đề ban đầu được cho là nhất quán một cách tối thiểu, nghĩa là dù bất kỳ mệnh đề C nào của G, tập hợp G $-\{C_0\}$ là nhất quán.

Từ nhận định này, ta có thể tìm thấy một bác bỏ tuyến tính xuất phát từ mệnh đề cuối cùng như sau :



Hình 2.15. Đồ thị bác bỏ chiến lược tuyến tính

III.2.6. Chiến lược bác bỏ bởi hợp giải là «tuyến tính theo đầu vào»

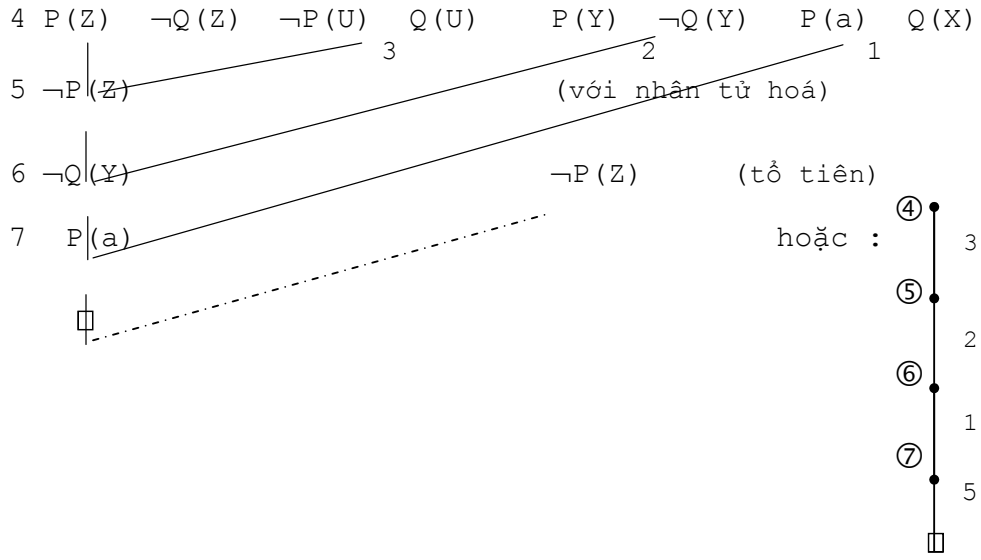
Chiến lược bác bỏ bởi hợp giải là «tuyến tính theo đầu vào» (linear input resolution) thực chất là chiến lược tuyến tính (chọn một mệnh đề C_0 sao cho tập hợp $G - \{C_0\}$ là nhất quán), nhưng người ta yêu cầu thêm điều kiện chỉ lấy các mệnh đề biên trong số các mệnh đề vào đã cho.

Ví dụ, bốn phép bác bỏ tuyến tính trong mục III.2.5 vừa rồi cũng là bác bỏ tuyến tính theo đầu vào.

Nói chung, chiến lược tuyến tính theo đầu vào không là đầy đủ. Ví dụ sau đây là một minh chứng. Cho G là tập hợp các mệnh đề vào :

$G = \{P(a) \vee Q(X), P(Y) \vee \neg Q(Y), \neg P(Z) \vee \neg Q(Z), \neg P(U) \vee Q(U)\}$

Sử dụng chiến lược tuyến tính theo đầu vào, xuất phát từ $\neg P(Z) \vee \neg Q(Z)$, ta nhận được đồ thị bác bỏ như sau :



Hình 2.16. Đồ thị bác bỏ chiến lược tuyến tính theo đầu vào

Trong ví dụ này, ta không thể nhận được phép bác bỏ tuyến tính theo đầu vào. Mỗi mệnh đề vào thuộc G đều có hai trực kiện. Để từng đôi trực kiện cùng được loại bỏ với nhau đối với phép hợp giải, cần thực hiện phép nhân tử hoá trong quá trình hợp giải, điều này là không thể xảy ra đối với mỗi một trong bốn mệnh đề vào.

Tuy nhiên, chiến lược tuyến tính theo đầu vào là đầy đủ (chỉ với hợp giải nhị phân mà không nhân tử hoá) nếu áp dụng cho tập hợp không nhất quán các *mệnh đề Horn* (người ta sẽ

nói : *đầy đủ đối với mệnh đề Horn*). Các mệnh đề Horn là những mệnh đề chỉ chứa nhiều nhất một trực kiện dương, như :

$\neg R(a, b), \neg P(b) \vee R(a, Y)$.

Chú ý rằng cả 4 mệnh đề trong ví dụ ở mục III.2.5 đều không phải là mệnh đề Horn. Dẫu sao ta cũng đã tạo ra 4 phép bác bỏ tuyến tính theo đầu vào : các chiến lược không đầy đủ theo lý thuyết, nhưng thực tiễn, chúng vẫn có hiệu quả.

III.2.7. Chiến lược hợp giải «LUSH»

Chiến lược hợp giải «LUSH» là sự vận dụng của «chiến lược tuyến tính theo đầu vào» với ưu điểm thu hẹp các hợp giải cho phép. Trong mệnh đề trung tâm hiện hành, người ta chọn một cách ngẫu nhiên một trực kiện và chỉ để ý đến những hợp giải chứa trực kiện cố định này.

Chiến lược hợp giải «LUSH» là đầy đủ (chỉ với hợp giải nhị phân) đối với các mệnh đề Horn.

***Chú ý* :**

«LUSH» viết tắt từ cụm từ «Linear resolution with Unrestricted Selection function for Horn clauses». Ở đây, «Unrestricted» có nghĩa là việc chọn trực kiện trong mỗi mệnh đề trung tâm là tuỳ ý (nhưng không phải là tuỳ tiện).

***Ví dụ* :**

Xét các tiên đề sau đây :

a_1 : $(\forall X)$ EQUAL(X, X) thể hiện phép *phản xạ* của quan hệ EQUAL.

a_2 : $(\forall U)(\forall V)(\forall W)((EQUAL(U, V) \wedge EQUAL(WV)) \rightarrow EQUAL(U, W)$
thể hiện phép *nửa–phản xạ* (half–transitivity).

Ta muốn chứng minh rằng :

$(\forall X)(\forall Y)$ EQUAL (X, Y) $\rightarrow$ EQUAL (Y, X) có nghĩa là *đối xứng* (symmetry).

Thật vậy, ta có thể chứng minh trực tiếp : từ a_2, bằng cách lấy V=U, ta có :

$(\forall U)(\forall W)$ (EQUAL (U, U) $\wedge$ EQUAL (W, U)) $\rightarrow$ EQUAL (U, W)).

Nhưng từ a_1, ta được :

$(\forall U)(\forall W)$ EQUAL (W, U) $\rightarrow$ EQUAL (U, W) *dpcm*.

Bây giờ ta có thể dùng phép bác bỏ để chứng minh. Ta cần chứng minh tính không nhất quán của :

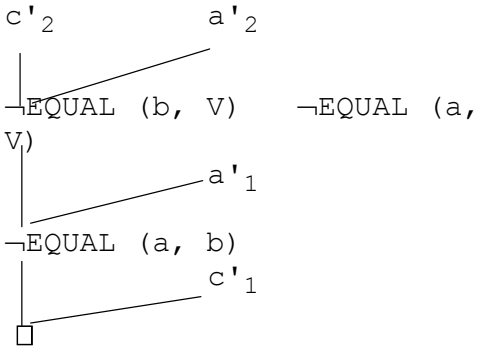
a'_1 : EQUAL (X, X)

a'_2 : $\neg$EQUAL (U, V) $\square\neg$EQUAL (W, V) $\square$EQUAL (U, W)

c'_1 : EQUAL (a, b)

c'_2 : $\neg$EQUAL (b, a)

Sau đây là đồ thị hợp giải nhị phân theo đầu vào :



Hình 2.17. Đồ thị hợp giải nhị phân theo đầu vào

***Chú ý* :**

- Trong mục III.2.3, ta đã chỉ ra rằng khi một tập hợp mệnh đề đã cho là không nhất quán thì phải tồn tại một mệnh đề không có trực kiện dương, và cần phải có một mệnh đề không có trực kiện dương tham gia vào mọi phép chứng minh tính không nhất quán bằng hợp giải. Trong ví dụ trên đây, ta đã phải chọn mệnh đề xuất phát là c'_2 vì tập hợp các mệnh đề còn lại tạo thành một tập hợp nhất quán : toàn bộ trực kiện đều dương.
- Khi áp dụng các mệnh đề Horn, xuất phát từ một mệnh đề không có trực kiện dương, ta chỉ có thể tạo ra được các kết quả hợp giải cũng không có trực kiện dương.
- Khi một tập hợp các mệnh đề Horn là không nhất quán, thì cần phải có một *mệnh đề đơn vị* (unitary) (là mệnh đề chỉ có một trực kiện duy nhất) trong chúng là dương. Trong trường hợp này, nói chung sẽ tồn tại một phép bác bỏ (bởi hợp giải nhị phân) sao cho mỗi lần hợp giải, một trong các mệnh đề cha phải là một mệnh đề đơn vị dương.

III.3. Ví dụ minh hoạ : bài toán tìm người nói thật

Trên một hòn đảo nọ có hai loại người : những *người thành thật* luôn luôn chỉ nói sự thật và những *người dối trá* luôn luôn chỉ nói dối. Mỗi người dân của đảo hoặc là một người thành thật, hoặc là một người dối trá.

Một người nước ngoài đến đảo và gặp ba người dân của đảo là A, B và C. Ông ta hỏi A : «Ông là một người thành thật hay là một người dối trá ?». A trả lời ấp úng nên người nước ngoài nọ không hiểu gì. Ông ta quay sang hỏi B : «Anh ta đã nói gì vậy ?». B trả lời : «Anh ta đã nói rằng anh ta là người dối trá». Lúc này, C nói xen vào : «Ông đừng tin B, anh ta nói dối đấy !».

Hãy xác định xem B và C là những người thành thật hay là dối trá ?

Lời giải :

Ta sẽ biểu diễn các câu hỏi và câu trả lời trên đây bởi các công thức logic. Gọi S (sincere), L (lie) và SAY là các vị từ với quy ước như sau :

- S(X) chỉ rằng X luôn là người thành thật,
- L(X) chỉ rằng X luôn là người dối trá,
- SAY(X, Y) chỉ rằng X nói Y.

Ta có :

1. Mọi người dân trên đảo là thành thật hoặc dối trá :
   $(\forall X)\ S(X)) \vee L(X)$

2. Không có người nào lại vừa thành thật lại vừa dối trá :
   $(\forall X)\ \neg S(X) \vee \neg L(X)$
3. Nếu X là thành thật và nếu anh ta nói Y, thì Y là đúng :
   $(\forall X)\ (\forall Y)\ (S(X) \wedge SAY(X, Y)) \rightarrow Y)$
4. Nếu X là dối trá và nếu anh ta nói Y, thì Y là sai :
   $(\forall X)\ (\forall Y)\ (L(X) \wedge SAY(X, Y)) \rightarrow \neg Y)$
5. Nếu Y là đúng và nếu X nói Y, thì X là thành thật :
   $(\forall X)\ (\forall Y)\ (Y \wedge SAY(X, Y)) \rightarrow S(X)$
6. Nếu Y là sai và nếu X nói Y, thì X là dối trá :
   $(\forall X)\ (\forall Y)\ (\neg Y) \wedge SAY(X, Y)) \rightarrow L(X)$
7. Câu trả lời của A : SAY(a, không_thể_hiểu_được))
8. Câu trả lời của B : SAY(b, SAY(a, L(a)))
9. Câu trả lời của C : SAY(c, L(b))

Các hằng *a, b, c* biểu diễn ba người dân trên đảo ; hằng *không_thể_hiểu_được* biểu diễn câu nói lắp của A. Ta thấy có thể có nhiều cách để biểu diễn các vị từ. Trong các công thức 3, 4, 5, 6 biến Y có thể được xem như là một ký hiệu vị từ. Trong công thức 8, các ký hiệu vị từ SAY và L được xem như các ký hiệu hàm, tương tự đối với L trong công thức 9.

Để làm việc với logic các vị từ bậc một, ta sẽ chỉ dùng một ký hiệu vị từ duy nhất là C. Ký hiệu C(X) có nghĩa đúng (Correct) là X. Có thể bỏ các vị từ S, L và SAY bằng cách thay vào các hàm là *s, l* và *say* (tuy nhiên vẫn có thể giữ vị từ S mà không thay nó bởi hàm *s*). Lúc này ta nhận được 9 công thức mới như sau :

1. $(\forall X)\ C(s(X)) \veebar C(s(X))$
2. $(\forall X)\ \neg C(s(X)) \veebar \neg C(s(X))$
3. $(\forall X)\ (\forall Y)\ (C(s(X)) \wedge C(say(X, Y))) \rightarrow C(Y)$
4. $(\forall X)\ (\forall Y)\ (C(l(X)) \wedge C(say(X, Y))) \rightarrow \neg C(Y)$
5. $(\forall X)\ (\forall Y)\ (C(Y) \wedge C(say(X, Y))) \rightarrow C(s(X))$
6. $(\forall X)\ (\forall Y)\ (\neg C(Y) \wedge C(say(X, Y))) \rightarrow C(s(X))$
7. C(say(a, không_thể_hiểu_được))
8. C(SAY(b, say(a, l(a))))
9. C(SAY(c, l(b)))

Bây giờ ta chuyển 9 công thức này thành các mệnh đề, trừ mệnh đề đầu tiên đã là mệnh đề Horn, lần lượt được đánh số như sau :

1. $C(s(X)) \veebar C(l(X))$
2. $\neg C(s(X)) \veebar \neg C(l(X))$
3. $\neg C(s(X)) \veebar \neg C(say(X, Y)) \veebar C(Y)$
4. $\neg C(l(X)) \veebar \neg C(say(X, Y)) \veebar \neg C(Y)$
5. $\neg C(Y) \veebar \neg C(say(X, Y)) \veebar C(s(X))$
6. $C(Y) \veebar \neg C(say(X, Y))) \veebar C(l(X))$
7. C(say(a, không_thể_hiểu_được))
8. C(say(b, say(a, l(a))))
9. C(say(c, l(b)))

Để tiến hành quá trình bác bỏ, ta cần chọn một phỏng đoán và thêm các mệnh đề nhận được bởi phủ định phỏng đoán này vào 9 mệnh đề trên.

Trước hết, ta phỏng đoán rằng B là một người dối trá, nghĩa là C(l(b)), khi đó cần thêm vào mệnh đề phủ định ¬C(l(b)). Tiếp theo, ta cần chọn một chiến lược tổng quát để tiến hành bác bỏ bởi hợp giải từ các chiến lược :

– xây dựng tập hợp trợ giúp (tập hợp nào ?),

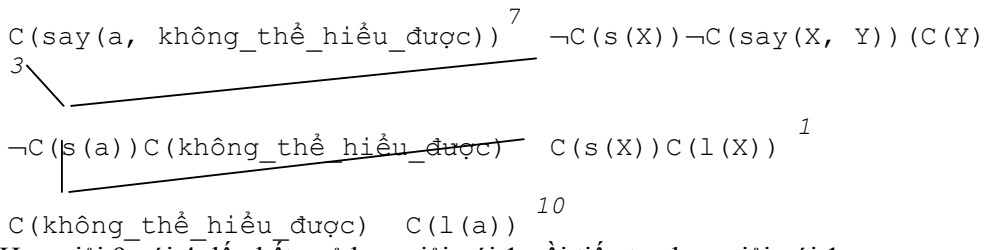
– dùng khoá (cách đánh số các trực kiện như thế nào ?),
– tuyến tính (mệnh đề nào được chọn làm mệnh đề xuất phát ?), v.v...

Mặt khác, cần chỉ rõ chiến lược hợp giải là theo chiều rộng hay chiều sâu ?

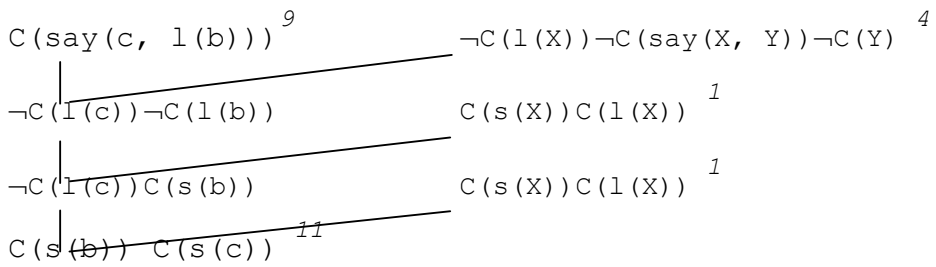
Dù chọn chiến lược nào và với lý do nào, việc tổ hợp các hợp giải cũng gặp rất khó khăn.

Giả sử mệnh đề ¬C(l(b)) được đánh số là 0, sau đây là một quá trình bác bỏ :

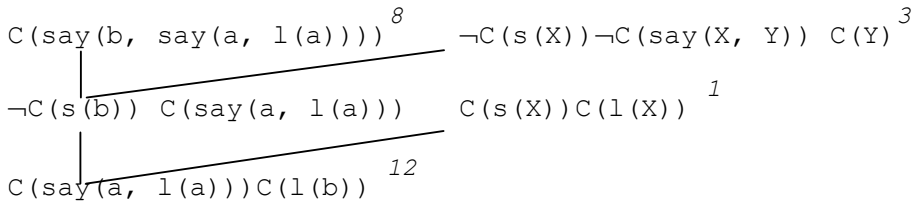
1. Hợp giải 7 với 3, rồi lấy kết quả hợp giải với 1 :



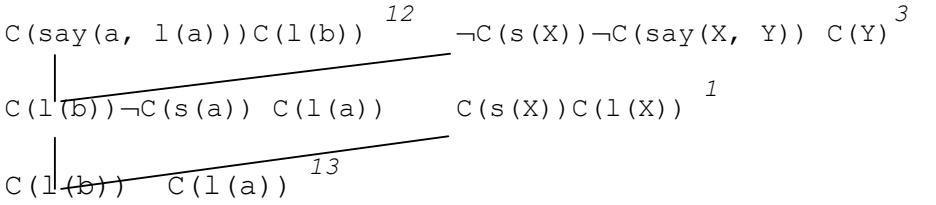
2. Hợp giải 9 với 4, lấy kết quả hợp giải với 1, rồi tiếp tục hợp giải với 1 :



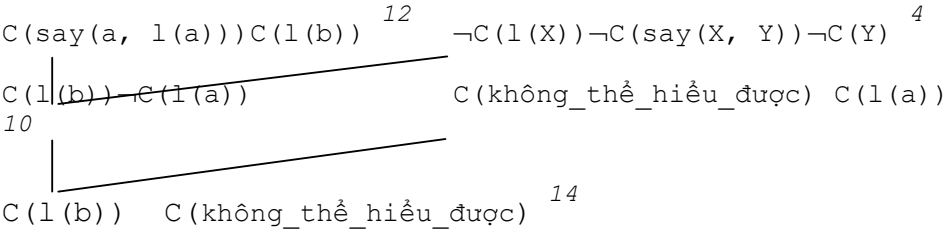
3. Hợp giải 8 với 3, lấy kết quả hợp giải với 1 :



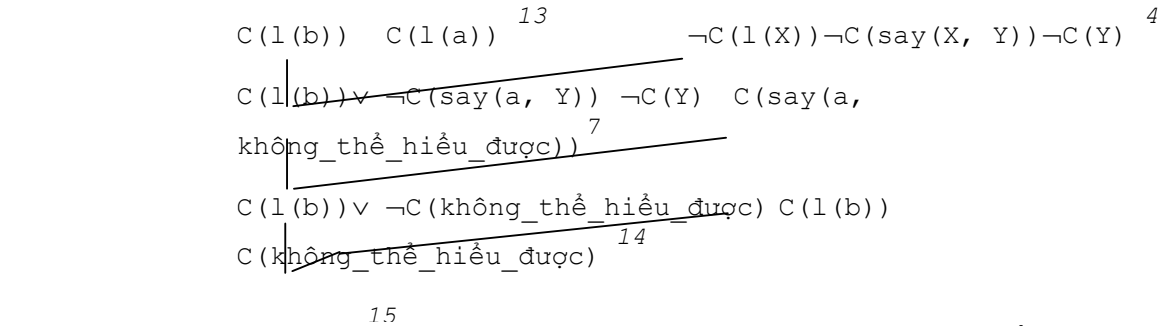
4. Hợp giải 12 với 3, lấy kết quả hợp giải với 1 (sử dụng nhân tử hoá) :



5. Hợp giải 12 với 4, lấy kết quả hợp giải với 10 :



6. Hợp giải 13 với 4, lấy kết quả hợp giải với 7, rồi với 4 (nhân tử hoá) :



7. Hợp giải 15 với 0, tức C(l(b)) với ¬C(l(b)), ta nhận được mệnh đề rỗng □.

Vậy ta đã chứng minh được rằng B là một người dối trá.

Ta nhận thấy rằng chiến lược bác bỏ trên đây không là tuyến tính, mà tương thích với chiến lược tập hợp trợ giúp. Các mệnh đề 2, 5 và 9 đã không được sử dụng. Chẳng hạn để xác định B là người như thế nào thì mệnh đề 9 với sự có mặt của C sẽ trở nên thừa.

Việc đưa ra các hợp giải khác nhau và các kết quả hợp giải khác nhau để có được phép bác bỏ cuối cùng là bổ ích. Chẳng hạn, kết quả hợp giải từ 8 và 3 có nghĩa rằng hoặc B không phải là người thành thật, hoặc quả đúng A đã nói rằng anh ta là người nói dối.

Phép hợp giải của 15 với 2, rồi lấy kết quả hợp giải với 11 dẫn đến C(s(c)). Nếu tiếp tục phỏng đoán rằng C là người nói dối, thì ta cần thêm vào mệnh đề ¬C(s(c) để hợp giải với C(s(c)), kết quả là một mệnh đề rỗng. Việc chứng minh C là người nói dối không sử dụng đến mệnh đề 5.

Cũng cần chú ý rằng thay vì sử dụng phép bác bỏ với phỏng đoán đã cho, ta có thể đưa ra các hợp giải bằng cách theo dõi sự không xuất hiện mệnh đề rỗng mà xuất hiện một trong bốn trực kiện C(l(c)), C(s(b)), ¬C(l(c)), ¬C(s(b)).

Thực tế, dạng thức vừa vận dụng trên đây không trung thành với phát biểu liên quan đến cách biểu diễn thái độ của A. Theo mệnh đề 7 thì ta đã xử lý câu trả lời của A như là một hằng, là *không_thể_hiểu_được*. Để tuân thủ điều này, ta đã cố tình không nói «A đã nói rằng anh ta là người thành thật», tức s(a), hay không nói «A đã nói rằng anh ta là người nói dối», tức l(a), vì rằng hằng *không_thể_ hiểu_được* không tương thích với các hạng s(a) hay l(a).

Tuy nhiên, mặc dù với hạn chế bổ sung như vậy, ta vẫn có thể nhận được mệnh đề rỗng, tức chứng minh được điều phỏng đoán. Để biểu diễn câu phát biểu một cách sát thực hơn, ta có thể thay thế mệnh đề 7 bởi mệnh đề 7' :

7'. C(say(a, l(a))) ∨ C(say(a, s(a)))

với giả thiết rằng dù là người thành thật hay là người nói dối, A chỉ trả lời trực tiếp vào câu hỏi của người lạ, mà không trả lời thật thà hay dối trá khác với kiểu «tôi là người thành thật», hay «tôi là người người nói dối», như là «ông là nước ngoài», hay «ông không phải là nước ngoài», hay là một câu nào đó đại loại như «B là người này người kia».

Nếu cải chính mệnh đề 7 bởi mệnh đề 7' và thay vì dùng mệnh đề :

C(say(a, không_thể_hiểu_được))

mà dùng một trong các trực kiện C(say(a, l(a))) hay C(say(a, s(a))), thì phép bác bỏ trên đây không còn duy nhất : mệnh đề rỗng không được sinh ra.

Sau đây là một phép bác bỏ khác không dùng đến mệnh đề 7' :

1. Hợp giải 2 và 3 với phép nhân tử hoá :

¬C(s(X)) ∨ ¬C(say(X, l(X))) 10'

2. Hợp giải 1 và 4, rồi lấy kết quả hợp giải với 2 (nhân tử hoá) :
$\neg C(l(X)) \vee \neg C(say(X, l(X)))$ *11'*

3. Hợp giải 10' và 1, rồi lấy kết quả hợp giải với 11' (nhân tử hoá) :
$\neg C(say(X, l(X)))$ *12'*

4. Hợp giải 6 và 8, rồi lấy kết quả hợp giải với 12' :
$C(l(b))$ *13'*

4. Hợp giải 13' và 0, ta nhận được mệnh đề rỗng □.

Vậy B là một người nói dối. Phép hợp giải đã không dùng đến mệnh đề 9 (có sự tham gia của C), 5 và 7'. Đây là kết quả của một chiến lược tuyến tính, tuy nhiên vẫn có thể áp dụng chiến lược tập hợp trợ giúp.

Hợp giải 5 với 13', cho kết quả hợp giải với 9 để nhận được C(s(c)), cuối cùng hợp giải C(s(c)) với ¬C(s(c)) nhận được mệnh đề rỗng, vậy C là người thành thật.

Các giai đoạn bác bỏ hình thức trên đây liên quan đến việc xác định B và C rất gần gũi với việc suy luận theo ngôn ngữ tự nhiên như sau :

«Không thể nào một người thành thật hay một người nói dối lại nói : "Tôi là người nói dối", vì rằng một người thành thật thì không thể nói dối, còn một người dối trá thì không khi nào dám nói lên sự thật. Do vậy, A không nói rằng anh ta là một người nói dối và B nói dối khi nói rằng A đã khẳng định rằng anh ta là người nói dối. Do vậy, B là một người nói dối. Vì C đã nói rằng B đã nói dối, nên C đã nói lên điều đúng, nên C là một người thành thật».

Bài tập chương 2

3. Cho các ví dụ sử dụng cú pháp của ngôn ngữ vị từ bậc một :

a. Bảng ký hiệu
b. Hạng (term)
c. Nguyên tử (atom)
d. Các công thức chỉnh

4. Thế nào là tính hợp thức và không hợp thức, tính nhất quán và không nhất quán của một công thức ? Thế nào làt ính không quyết định được và tính nửa quyết định được của logic vị từ bậc một

5. Cho các ví dụ về công thức chỉnh và phép biến đổi mệnh đề

6. Từ các ví dụ đã cho trong giáo trình, tự cho các ví dụ về các chiến lược hợp giải tìm kiếm và bác bỏ.

7. Tìm một ví dụ khác tương tự bài toán tìm người nói thật.

CHƯƠNG 3

Máy suy diễn

" I am only one, but still I am one. I cannot do everything,
but still I can do something; and because I cannot do everything,
I will not refuse to do something I can do ".
Edward Everett Hale

Chương trước, ta đã vận dụng logic hình thức để hợp giải bài toán đã cho, nghĩa là nghiên cứu những phương pháp phán đoán và khẳng định các lời giải, đồng thời đánh giá những tính chất và những hạn chế của các phương pháp này.

Trong chương này, dựa trên các khái niệm *công thức*, *tiên đề*, *luật suy diễn* và các mối quan hệ giữa chúng, ta sẽ nghiên cứu các *máy suy diễn* (inference engine) trong các *hệ thống dùng luật* hay *dựa trên luật* (Rules−Based Systems).

I. Nguyên lý hoạt động của các máy suy diễn

Trong các hệ thống dùng luật, mỗi luật bao gồm thông tin về *bộ khởi động* (starter), hay *điều kiện khởi động* của luật, và *thân* (body) của luật, hay thông tin về *kết quả khởi động luật* :

Luật = <bộ khởi động> + <thân>

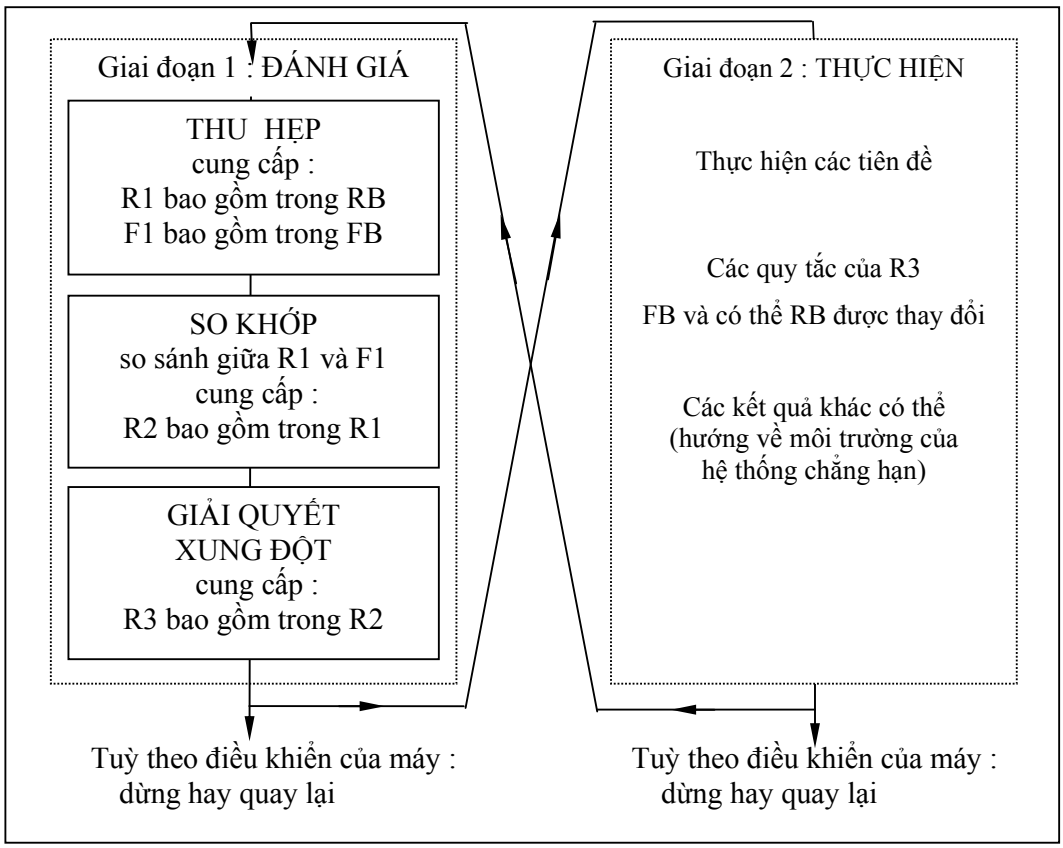
Máy suy diễn liên kết các *chu kỳ* (cycle), mỗi chu kỳ gồm hai *giai đoạn* (phase) là EVALUATION (đánh giá) và EXECUTION (thực hiện).

Khi máy được khởi động, cơ sở tri thức chứa các thông tin liên quan đến phát biểu bài toán cần giải :

- Các sự kiện đã được xác nhận và các sự kiện sẽ được thiết lập (biểu diễn bài toán hay đích),
- Những tri thức thực hành thuộc lĩnh vực tạo nên cơ sở luật.

Theo sơ đồ, ở giai đoạn EVALUATION, máy suy diễn xác định trong cơ sở luật có tồn tại các luật sẽ được khởi động căn cứ vào trạng thái hiện hành của cơ sở sự kiện không, nếu có, thì là những luật nào. Trong giai đoạn EXECUTION, máy suy diễn khởi động các luật đã được tìm thấy ở giai đoạn EVALUATION.

Hình 3.1 dưới đây mô tả chu kỳ cơ bản của một máy suy diễn. Ta quy ước gọi RB là *cơ sở luật* (Rules Base) và FB là *cơ sở sự kiện* (Facts Base) tại thời điểm bắt đầu của EVALUATION. Máy suy diễn điều khiển liên kết các *bước* (step) của giai đoạn EVALUATION, của các giai đoạn EVALUATION và EXECUTION, rồi các chu kỳ đầy đủ giữa chúng.



Hình 3.1 Chu kỳ cơ bản của một máy suy diễn

Máy suy diễn được điều khiển dừng ở giai đoạn EVALUATION hoặc ở giai đoạn EXECUTION :

- Căn cứ vào trạng thái hiện hành của cơ sở sự kiện, máy dừng ở giai đoạn EVALUATION khi không tìm thấy các luật khởi động trong cơ sở luật.
- Máy dừng ở giai đoạn EXECUTION khi một trong các luật khởi động cho kết quả dừng.

I.1. Giai đoạn đánh giá EVALUATION

Giai đoạn EVALUATION gồm ba bước : RESTRICTION (hay SELECTION), PATTERN–MATCHING (hay FILTERING) và CONFLICT-RESOLUTION.

a. Bước thu hẹp (RESTRICTION)

RESTRICTION (thu hẹp) là bước đầu tiên của giai đoạn EVALUATION, để xác định từ một trạng thái hiện hành hay quá khứ của cơ sở sự kiện (ký hiệu FB) và từ một trạng thái hiện hành hay quá khứ của cơ sở luật (ký hiệu RB), một tập hợp con F1 của FB và một tập hợp con R1 của RB sao cho có thể tiến hành so sánh được trong bước FILTERING tiếp theo.

Người ta thường dùng kỹ thuật khai thác các tri thức dựa trên sự phân bố các sự kiện và các luật theo các họ riêng biệt. Đôi khi, các tri thức cho phép phân biệt các sự kiện và các luật liên quan trực tiếp đến lĩnh vực. Ví dụ, trong một ngữ cảnh chẩn đoán bệnh, người ta có thể phân biệt được ngay các luật liên quan đến các bệnh trẻ em với các luật khác, hay phân biệt được các sự kiện liên quan đến phân tích máu với các sự kiện khác.

Như vậy, bước RESTRICTION là ưu tiên cho một nhóm nào đó các luật hay các sự kiện

đối với một hoặc nhiều chu kỳ. Thông thường, những tri thức để phân biệt các sự kiện và các luật là rất tổng quát.

Chẳng hạn, nhiều hệ thống phân biệt được các sự kiện đã thiết lập (coi như đã được xác nhận) với các sự kiện sẽ thiết lập (biểu diễn bài toán ban đầu hay đích, hay giả thuyết). Việc thu hẹp sẽ ưu tiên các sự kiện–bài toán so với các sự kiện–bài toán khác, hay ưu tiên bài toán xuất hiện gần đây nhất so với các bài toán trước đó. Sự phân biệt theo họ các sự kiện hay theo họ các luật thường được cụ thể hoá bởi định nghĩa các cấu trúc phân biệt.

b. Bước so khớp (PATTERN–MATCHING)

PATTERN–MATCHING (so khớp hay lọc) là bước thứ hai của giai đoạn EVALUATION. Máy suy diễn so sánh phần khởi động của mỗi quy tắc của R1 với tập hợp các sự kiện F1. Một tập hợp con R2 của R1 nhóm các luật tương thích với F1, nghĩa là những luật có điều kiện khởi động thoả mãn các trạng thái của F1 (tuỳ theo mỗi hệ thống mà có những tiêu chuẩn thoả mãn khác nhau). R2 được gọi là *tập hợp xung đột* (conflict set).

c. Giải quyết xung đột (CONFLICT-RESOLUTION)

Bước thứ ba của giai đoạn EVALUATION là CONFLICT-RESOLUTION. Máy suy diễn xác định các luật, giả sử là một tập hợp con R3 của R2, cần phải được khởi động. Nếu tập hợp R3 rỗng, thì giai đoạn EXECUTION của chu kỳ này không được thực thi.

Thông thường, người ta lựa chọn các luật dựa trên những tiêu chuẩn *không liên quan đến nghĩa* (meaning/signification) của luật có mối quan hệ với bối cảnh áp dụng. Ví dụ, cơ sở luật được sắp xếp ngẫu nhiên thành một danh sách và người ta chọn những luật đứng đầu tiên, hoặc chọn ưu tiên những luật ít sử dụng hơn, hoặc có thể chọn trước tiên những luật ít phức tạp nhất : ít điều kiện cần kiểm tra, ít biến (variable) cần xác định trước khi khởi động, v.v...

Đôi khi, người ta dựa trên những tiêu chuẩn *liên quan đến nghĩa* để chọn các luật có mối quan hệ với bối cảnh áp dụng. Chẳng hạn, một số luật có thể được chọn do có những dấu hiệu giải bài toán tốt hơn, hay có thể đáng tin cậy hơn, hay ít tốn kém về chi phí hơn so với những luật khác, v.v...

I.2. Giai đoạn thực hiện EXECUTION

Khi R3 rỗng, một số máy suy diễn tự động dừng : người ta nói những máy này có một *chế độ điều khiển bắt buộc* (irrevocable control regime). Một số máy khác thì lại xem xét lại tập hợp tương tranh R2 của một chu kỳ trước đó và kiểm tra khả năng khởi động của các luật khác của R2.

Tuy nhiên, nếu không có một khởi động nào cho luật được thực thi kể từ phép chọn trước đó trong R2, nghĩa là nếu kết quả của các luật này không được huỷ bỏ, trước khi khởi động các luật khác, thì người ta cũng nói rằng những máy này *hoạt động theo chế độ điều khiển bắt buộc*. Ngược lại, người ta nói chúng hoạt động *theo chế độ điều khiển bởi thăm dò* (tentative control regime) khi có sự thay thế các khởi động luật bởi các khởi động khác.

Để thể hiện một máy quay lại giải quyết các xung đột trước đó, bằng cách khởi động lại các luật, người ta nói máy hoạt động *quay lui* (to backtrack).

Sự quay lui hay không quay lui của máy lúc đầu được điều khiển ở giai đoạn RESTRICTION, tiếp theo, bởi giai đoạn CONFLICT-RESOLUTION. Trong mục tiếp theo, ta sẽ giới thiệu chi tiết hoạt động của một số máy đơn giản ở chế độ điều khiển bắt buộc hay ở chế độ điều khiển bởi thăm dò.

Trên thực tế, mỗi giai đoạn của chu kỳ cơ bản của một máy có thể dẫn đến những cách sắp đặt rất khác nhau. Hình 3.1 trên đây mô tả hoạt động của một chu kỳ. Để giải quyết một bài toán đã cho, có thể cần đến hàng ngàn chu kỳ.

Mỗi chu kỳ cơ bản của một máy suy diễn làm ta liên tưởng đến chu kỳ-lệnh của một máy tính. Nhưng mỗi chu kỳ của máy suy diễn, hay chu kỳ suy diễn, đòi hỏi hàng trăm, thậm chí hàng ngàn các chu kỳ-lệnh này. Chính vì vậy, cần có những máy tính có tốc độ lớn để có thể đạt được hàng trăm chu kỳ suy diễn trong một giây. Dự án máy tính thế hệ 5 của Nhật đề xuất những kiến trúc đặc trưng cho phép đạt được tốc độ hàng triệu hay hàng tỷ *lips* (Logical Inference Per Second, mỗi chu kỳ là một suy diễn).

II. Một số sơ đồ cơ bản để xây dựng máy suy diễn

Sau đây ta sẽ trình bày một số chiến lược liên kết các luật trong các máy suy diễn. Ta sẽ giới thiệu các «sơ đồ» (diagram) của máy hoạt động theo kiểu *suy diễn tiến* (pre-chaining), hay theo kiểu *suy diễn lùi* (back-chaining), theo *chiều sâu* (depth-first), hay theo *chiều rộng* (breadth-first), tuỳ theo *chế độ bắt buộc* hay theo *chế độ bởi thăm dò*. Một vấn đề cũng được đặt ra là *lập kế hoạch hành động* (actions planning). Riêng máy hoạt động theo chế độ bởi thăm dò sử dụng đến các *biến* (variable).

Dưới đây ta sẽ trình bày sáu sơ đồ máy. Khi khởi động một trong bốn máy đầu tiên là PREDIAGRAM–1, PREDIAGRAM–2, BACKDIAGRAM–1 và BACK DIAGRAM2, hai biến toàn cục FACTSBASE và RULESBASE được giả thiết là biểu diễn các tập hợp sự kiện và tập hợp luật tương ứng.

II.1. Một ví dụ về cơ sở tri thức

Để minh hoạ hoạt động của bốn máy vừa nói trên, ta xây dựng một cơ sở tri thức ví dụ như sau :

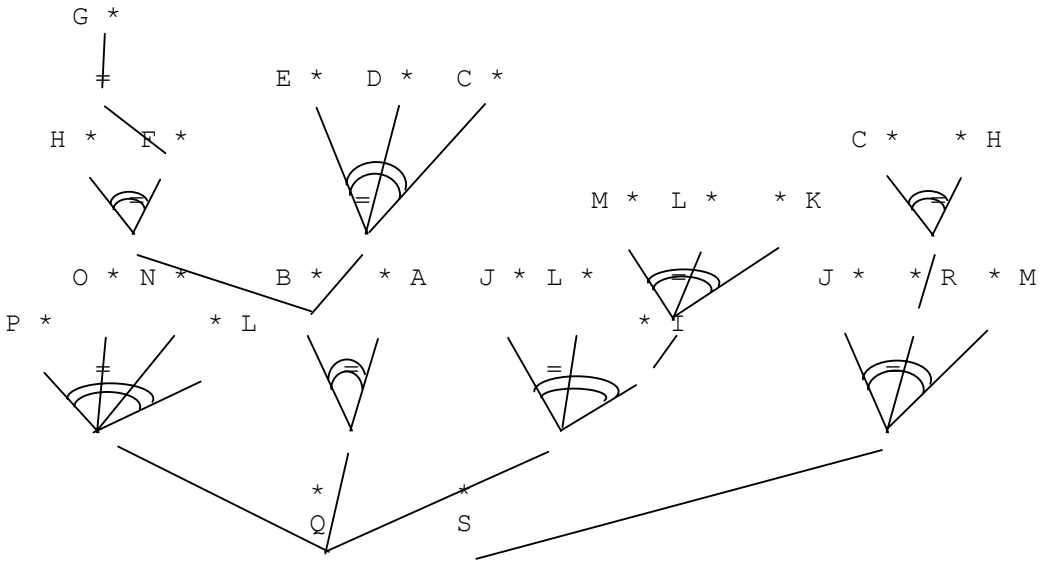
```
RULESBASE là danh sách các luật sau đây
                    :
        1   K, L, M → I
        2   I, L, J → Q
        3   C, D, E → B
        4   A, B → Q
        5   L, N, O, P → Q
        6   C, H → R
        7   R, J, M → S
        8   F, H → B
        9   G → F
FACTSBASE là danh sách : A, C, D, E, G,
                  H, K
```

Hình 3.2 Một cơ sở tri thức ký hiệu

Thành phần bên trái của mỗi luật còn được gọi là phần *tiền đề* (premise trong luật ba đoạn) của luật, là phép hội (conjunction) của các sự kiện ký hiệu. Thành phần bên phải còn được gọi là phần *kết luận* (conclusion) của luật. Ta giải thích luật 1 như sau :

- Hoặc : nếu K và L và M là các sự kiện được thiết lập, thì kết luận rằng sự kiện I được thiết lập.
- Hoặc : nếu thiết lập I, cần phải thiết lập K và L và M.

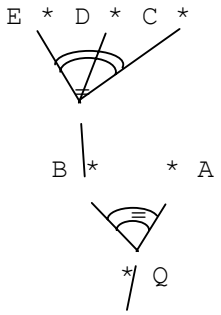
«Đồ thị VÀ-HOẶC» sau đây biểu diễn mọi liên kết có thể của các luật :



Hình 3.3 Một đồ thị VÀ-HOẶC từ cơ sở tri thức ký hiệu

Trong đồ thị VÀ-HOẶC trên đây (cũng còn được gọi là đồ thị các bài toán con), khi một nhánh HOẶC được thiết lập từ H VÀ F, một nhánh HOẶC được thiết lập từ E VÀ D VÀ C, thì B được thiết lập. Sự kiện F được thiết lập do G được thiết lập. Để cho tiện trình bày, ta đã lặp lại các nút sự kiện C, H, J và M.

Dưới đây, ta xây dựng một đồ thị con VÀ-HOẶC từ đồ thị VÀ-HOẶC trên đây để thiết lập sự kiện Q khi các sự kiện A, C, D và E được thiết lập.



Hình 3.4. Đồ thị VÀ-HOẶC thiết lập Q khi A, C, D và E được thiết lập

II.2. Tìm luật nhờ suy diễn tiến với chế độ bắt buộc đơn điệu

a. Sơ đồ PREDIAGRAM–1 : lấy ngay kết luận của mỗi luật

PREDIAGRAM–1 là sơ đồ máy suy diễn gồm hai thủ tục SETUP-A-FACT và RUN-A-CYCLE. Máy suy diễn được khởi động khi thủ tục SETUP-A-FACT được gọi. Ví dụ, để thiết lập Q, người ta gọi : SETUP-A-FACT (‘Q’).

```
    procedure SETUP-A-FACT (FACT)
1.  if FACT in FACTSBASE then return ‘success’
2.  return RUN-A-CYCLE (RULESBASE)
    procedure RUN-A-CYCLE (RULES, FACT, ARULE)
1.  if RULES = ∅ then return ‘failure’
2.  ARULE ← chọn một luật nào đó từ RULES (chẳng hạn luật gặp đầu tiên)
3.  RULES ← RULES − { ARULE }
4.  if (∀) (ARULE.premise) in FACTSBASE then
    4.1 begin
    4.2 if ARULE.conclusion = FACT then return ‘success’
    4.3 if not (ARULE.conclusion in FACTSBASE) then
        FACTSBASE ← FACTSBASE + { ARULE.conclusion }
    4.4 RULESBASE ← RULESBASE − { ARULE }
    4.5 return RUN-A-CYCLE (RULESBASE)
    4.6 end
5.  return RUN-A-CYCLE (RULES)
```

Hình 3.5. Sơ đồ máy PREDIAGRAM–1

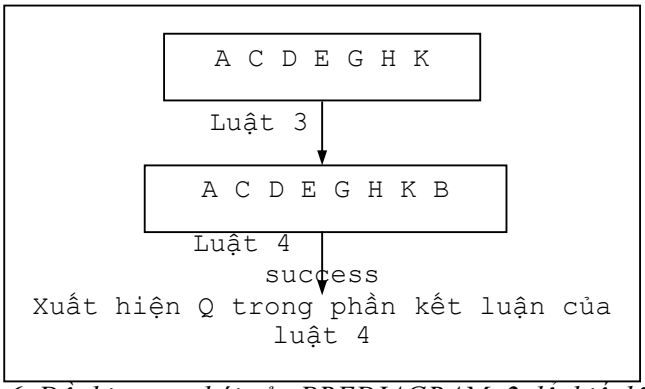
Thủ tục SETUP-A-FACT gây ra sự *suy diễn tiến* của các luật, bằng cách so sánh các phần tử thuộc tiền đề của các luật với các phần tử của cơ sở sự kiện, xem chúng như những sự kiện đã *được thiết lập*. Người ta cũng nói trong thủ tục SETUP-A-FACT, phần tiền đề của các luật (giả sử là các thành phần bên trái) cũng là phần khởi động.

Khi một luật được khởi động, phần tử kết luận được thiết lập và ngay lập tức được đưa vào trong cơ sở sự kiện FACTSBASE. Trong suốt quá trình thực hiện thủ tục, FACTSBASE chỉ chứa các sự kiện đã được thiết lập.

Chẳng hạn ta xét cơ sở sự kiện và luật đã cho trong ví dụ hình 3.2. Để có thể thiết lập sự kiện Q, ta khởi động máy PREDIAGRAM–1 ở thủ tục 3.7 bởi lời gọi SETUP-A-FACT(‘Q’).

Máy hoạt động như sau : luật 3 được khởi động để bổ sung B vào FACTSBASE, luật 4 được khởi động để bổ sung Q. Lời gọi thủ tục trả về kết quả «success».

Hình 3.4. biểu diễn việc liên kết các luật 3 và 4 như là một đồ thị VÀ-HOẶC. Hình 3.8 dưới đây biểu diễn đồ thị trạng thái của thủ tục.



Hình 3.6. Đồ thị trạng thái của PREDIAGRAM–2 để thiết lập Q

Vị trí của PREDIAGRAM–1 đối với chu kỳ cơ bản tổng quát

Bước RESTRICTION loại trừ tất cả những luật đã sử dụng : xem lệnh 4.3 trong thủ tục RUN-A-CYCLE ở hình 3.5. Các bước FILTERING và CONFLICT-RESOLUTION đan xen nhau trong các lệnh từ 1 đến 4 của RUN-A-CYCLE.

b. Sơ đồ PREDIAGRAM□ : tạo sinh và tích luỹ sự kiện theo chiều rộng

Tương tự PREDIAGRAM–1, sơ đồ máy suy diễn PREDIAGRAM2 cũng gây ra sự *suy diễn tiến* của các luật và được gọi bởi thủ tục SETUP-A-FACT ().

Cho E là một sự kiện của cơ sở sự kiện. Trong PREDIAGRAM2, ngược lại với những gì đã xảy ra trong PREDIAGRAM–1, máy tìm kiếm ưu tiên lần lượt từng luật đối với các luật tương thích với E, trước khi bổ sung các kết luận vào trong E và trước khi sử dụng chúng để khởi động các luật mới.

Ta gọi các sự kiện ở mức 0 là các sự kiện khởi đầu, các sự kiện ở mức 1 là các kết luận của các luật có thể được khởi động xuất phát từ các sự kiện khởi đầu, và một cách tổng quát, các sự kiện ở mức n1 là các sự kiện nhận được xuất phát từ các sự kiện mà ít nhất một sự kiện ở mức n, trong khi đó, các sự kiện khác ở mức nhỏ hơn hoặc bằng n. Thủ tục PREDIAGRAM2 chỉ tạo ra các sự kiện ở mức n1 khi tất cả các sự kiện ở mức n đã được tạo ra. Người ta nói rằng PREDIAGRAM2 tạo ra các sự kiện *ưu tiên theo chiều rộng*.

Những kết luận mới của tất cả các luật tương thích với E được tích luỹ liên tiếp nhờ biến NEWFACTS (lệnh 4.3). Khi tất cả các luật đã được xem xét, NEWFACTS trở nên rỗng trong FACTSBASE (lệnh 1.3).

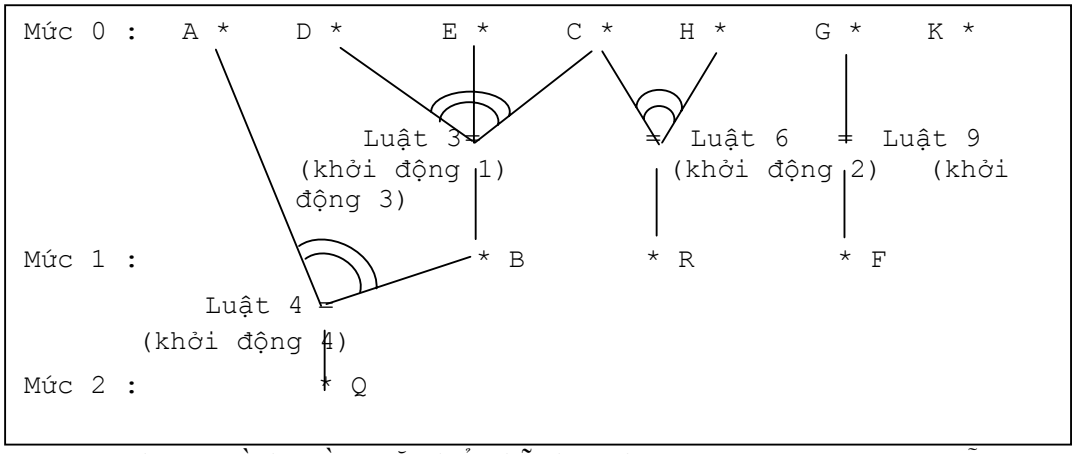
```
    procedure SETUP-A-FACT (FACT)
1.  if FACT in FACTSBASE then return 'success'
2.  return RUN-A-CYCLE (FACTSBASE, emptylist, FACT)
    procedure RUN-A-CYCLE (RULES, NEWFACTS, FACT, ARULE)
1.  if RULES = ∅ then
    1.1  begin
    1.2  if NEWFACTS = ∅ then return 'failure'
    1.3  FACTSBASE ← FACTSBASE + NEWFACTS
    1.4  return RUN-A-CYCLE (FACTSBASE, emptylist, FACT)
    1.5  end
```

2. ARULE ← chọn một luật nào đó từ RULES (chẳng hạn luật gặp đầu tiên)
3. RULES ← RULES − { ARULE }
4. if (∀) (ARULE.premise) in FACTSBASE then
   4.1 begin
   4.2 if kết luận của ARULE = FACT then return 'success'
   4.3 if not (ARULE.conclusion in FACTSBASE)
       or not (ARULE.conclusion in NEWFACTS)
       then NEWFACTS ← NEWFACTS + { ARULE.conclusion }
   4.4 RULESBASE ← RULESBASE − { ARULE }
   4.5 end
5. return RUN-A-CYCLE (RULES, NEWFACTS, FACT)

Hình 3.7. Sơ đồ máy PREDIAGRAM̃2

Thủ tục PREDIAGRAM̃2 gây ra các sự suy diễn tiến của các luật và và tạo ra các sự kiện theo chiều rộng. Để minh hoạ, ta xét cơ sở sự kiện và luật đã cho trong ví dụ ở hình 3.2. Để có thể thiết lập sự kiện Q, người ta khởi động máy PREDIAGRAM̃2 bởi lời gọi SETUP-A-FACT('Q').

Đồ thị VÀ-HOẶC trong hình sau đây biểu diễn hoạt động của máy :



Hình 3.8. Đồ thị VÀ-HOẶC biểu diễn hoạt động của máy PREDIAGRAM̃2

Xuất phát từ trạng thái đầu của cơ sở sự kiện {A, C, D, E, G, H, K}, luật 3 được khởi động làm cho NEWFACTS = {B}. Sau đó, lần lượt luật 6 rồi luật 9 được khởi động xuất phát từ cùng trạng thái đầu làm cho NEWFACTS = {B, R, F}. Lúc này không còn trạng thái nào được khởi động xuất phát từ trạng thái đầu. Sau đó, tập hợp NEWFACTS trở nên rỗng trong FACTSBASE và FACTSBASE trở thành {A, C, D, E, G, H, K, B, R, F}. Xuất phát từ trạng thái này, luật 4 được khởi động và tạo ra sự kiện Q.

Vị trí của PREDIAGRAM–2 đối với chu kỳ cơ bản tổng quát

Bước RESTRICTION một mặt, loại trừ tất cả những luật đã sử dụng, nhưng mặt khác, không cho phép sử dụng các sự kiện có mức ṇ1 chừng nào mà tất cả các sự kiện có mức nhỏ hơn hoặc bằng n chưa được tạo ra. Các bước FILTERING và CONFLICT-RESOLUTION đan xen nhau trong RUN-A-CYCLE.

Các máy ở chế độ bắt buộc, đơn điệu

Cả hai máy PREDIAGRAM–1 và PREDIAGRAM–2 đều hoạt động theo chế độ bắt buộc, vì rằng việc khởi động của một luật không bao giờ được thay thế bởi một khởi động khác và kết quả tạo ra bởi một luật trong cơ sở sự kiện không bao giờ được xét lại.

Mặt khác, các lệnh trong PREDIAGRAM–1 và PREDIAGRAM–2 chỉ cho phép bổ sung

các sự kiện đã thiết lập (lệnh 4.2), vì rằng các sự kiện đã thiết lập vẫn còn đó để tiếp tục. Người ta nói rằng những máy này là *đơn điệu*.

II.3. Tìm luật nhờ suy diễn lùi với chế độ thăm dò đơn điệu

a. Sơ đồ BACKDIAGRAM –1 : sản sinh các bài toán con theo chiều sâu

Máy suy diễn BACKDIAGRAM–1 cho trong hình 3.9. dưới đây gồm bốn thủ tục là SETUP-A-FACT, SETUP1, SETUP2 và FACTS-CONJUNCTION-SETUP.

Máy được khởi động bằng cách gọi một trong các thủ tục SETUP-A-FACT hay FACTS-CONJUNCTION-SETUP. Chẳng hạn, để thiết lập Q, ta có lời gọi : SETUP-A-FACT('Q').

Thủ tục BACKDIAGRAM–1 tạo ra sự suy diễn lùi của các luật bằng cách so sánh phần kết luận của các luật với các sự kiện cần thiết lập tại thời điểm đang xét. Đối với BACKDIAGRAM–1, phần kết luận của các luật (thành phần bên phải) là phần khởi động của chúng. Chú ý rằng tại mỗi thời điểm, thủ tục duy trì tập hợp các sự kiện cần thiết lập.

Ví dụ, lúc đầu, khi gọi SETUP-A-FACT('Q'), tập hợp các sự kiện cần thiết lập chỉ chứa Q. Nếu luật 2 được sử dụng và được khởi động thì sự kiện cần thiết lập Q sẽ được thay thế bởi I, J và L. Người ta nói rằng I, J và L là những bài toán con của bài toán Q hay bài toán Q là cha của các bài toán I, J và L.

```
    procedure SETUP-A-FACT (FACT)
1.  if FACT in FACTSBASE then return 'success'
2.  return SETUP1 (FACTSBASE)
    procedure SETUP1 (RULES)
1.  if RULES = ∅ then return 'failure'
2.  ARULE ← chọn một luật nào đó từ RULES (chẳng hạn luật gặp đầu tiên)
3.  RULES ← RULES – { ARULE }
4.  if (FACT in ARULE.conclusion) then
    if SETUP2 (ARULE) = 'success' then return 'success'
5.  return SETUP1 (RULES)
    procedure SETUP2 (RULE, THEFACTS)
1.  THEFACTS ← { F | F in ARULE.premise }
2.  return FACTS-CONJUNCTION-SETUP (THEFACTS)

    procedure FACTS-CONJUNCTION-SETUP (FACTS)
1.  if FACTS = ∅ then return 'success'
2.  AFACT ← chọn một sự kiện nào đó từ FACTS
        (chẳng hạn luật gặp đầu tiên)
3.  FACTS ← FACTS – { AFACT }
4.  if SETUP-A-FACT (AFACT) = 'failure' then return 'failure'
5.  return FACTS-CONJUNCTION-SETUP (FACTS)
```

Hình 3.9. Sơ đồ máy BACKDIAGRAM–1

Ta gọi các bài toán ban đầu ở mức 0 và các bài toán mà một cha của chúng là ở mức n, còn các cha khác có thể ở mức nhỏ hơn hoặc bằng n. Để thiết lập một sự kiện Q, BACKDIAGRAM–1 tạo sinh và giải các bài toán con tuỳ theo một chiến lược là trường hợp đặc biệt của họ các chiến lược *ưu tiên theo chiều sâu* : thủ tục sẽ giải các bài toán con có mức ưu tiên cao nhất.

Phân biệt giữa khởi động một luật và rút ra kết luận của một luật

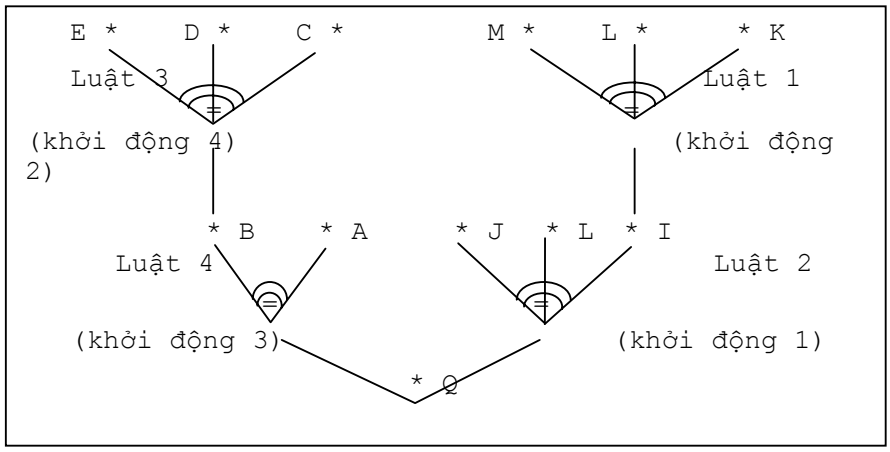
Khi một luật được sử dụng để thiết lập một sự kiện có mặt trong phần kết luận xuất phát từ các sự kiện có mặt trong phần tiền đề, giả sử đã được thiết lập, người ta nói rằng đã *rút ra kết luận của một luật*.

Khi một luật được sử dụng theo kiểu suy diễn lùi để thay thế các sự kiện có mặt trong phần tiền đề bởi sự kiện có mặt trong phần kết luận, người ta nói rằng đã *khởi động một luật*. Vấn đề là không nên nhầm lẫn việc khởi động một luật và rút ra kết luận của một luật.

Ví dụ hoạt động của thủ tục BACKDIAGRAM–1

Ta tiếp tục xét cơ sở sự kiện và luật đã cho trong ví dụ ở hình 3.2. Để có thể thiết lập sự kiện Q, người ta khởi động máy BACKDIAGRAM–1 bởi lời gọi SETUP-A-FACT (‘Q’).

Đồ thị các bài toán con dưới đây biểu diễn hoạt động của máy :



Hình 3.10. Đồ thị các bài toán con thiết lập Q của máy BACKDIAGRAM1

Khi luật 2 được khởi động, dẫn đến các sự kiện J, L và I cần được thiết lập. Trước tiên, sự kiện I được xem xét, từ đó luật 1 được khởi động dẫn đến các sự kiện M, L và K. Mặc dù sự kiện K được thiết lập, nhưng không có luật nào chứa L hay M trong phần khởi động, nên sự kiện cần được thiết lập lại quay trở lại I. Do không có luật nào chứa I trong phần khởi động, sự kiện cần được thiết lập bây giờ quay trở lại Q. Luật 4 được khởi động, dẫn đến các sự kiện A và B. Do A đã được thiết lập, máy xem xét sự kiện cần được thiết lập B. Từ đó, luật 3 được khởi động dẫn đến các sự kiện C, D và E. Do C, D và E đã được thiết lập, luật 3 ngay lập tức được ghi nhận : B được thiết lập và do đó, Q được thiết lập.

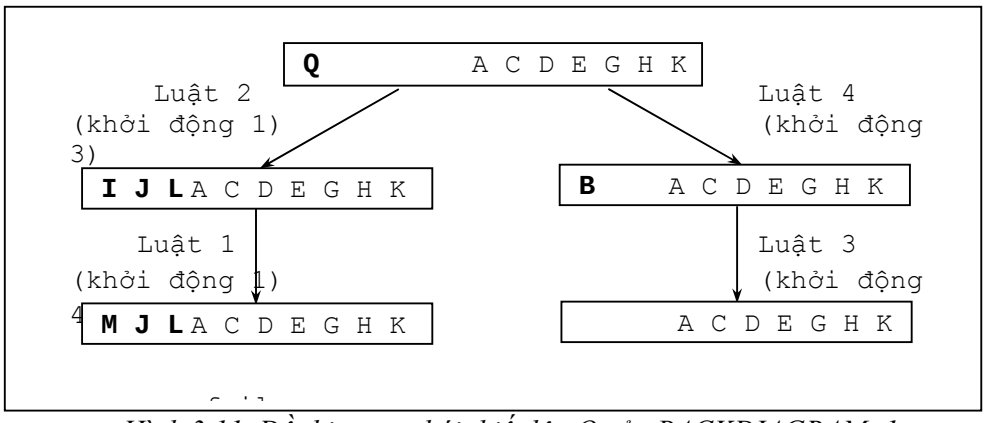
Vị trí của BACKDIAGRAM–1 đối với chu kỳ cơ bản tổng quát

Do suy diễn lùi, việc khởi động một luật dẫn đến một sự kiện Q, là *sự kiện cần được thiết lập* (hay *giả thuyết cần truy cập*, hay *bài toán*, hay *đích*) được thay thế bởi nhiều sự kiện khác cần thiết lập, chẳng hạn L, N, O và P.

Tuy nhiên, các bài toán con không có mặt một cách tường minh (explicit) trong cơ sở sự kiện : chúng chỉ được ghi nhớ trong quá trình gọi thủ tục. Những bài toán con chưa được giải quyết làm thành một phần *tiềm ẩn* (implicit) trong cơ sở sự kiện mà các sự kiện này có vai trò xác định lựa chọn các luật sẽ được sử dụng cho chu kỳ tiếp theo.

Hình 3.11. dưới đây biểu diễn quá trình khởi động các luật của thủ tục BACKDIAGRAM–1 bởi một đồ thị trạng thái để thiết lập Q.

Phần bên phải của mỗi ô tương ứng với phần tường minh của cơ sở sự kiện, gồm các sự kiện đã thiết lập. Phần bên trái tương ứng với phần ẩn, gồm các sự kiện cần thiết lập hay các bài toán chưa được giải quyết, được in chữ đậm trong hình). Các kết luận cuối cùng kéo theo bởi các luật 3 rồi 2 không được biểu diễn trong hình.



Hình 3.11. Đồ thị trạng thái thiết lập Q của BACKDIAGRAM–1

Trong BACKDIAGRAM–1, giai đoạn RESTRICTION trước tiên được cụ thể hoá bởi sự phân biệt giữa một bên là các sự kiện đã thiết lập và một bên là các bài toán. Chỉ có các bài toán là được so sánh với các luật, để có thể khởi động chúng. Nghĩa là việc RESTRICTION một mặt vào lúc đầu của mỗi chu kỳ cơ bản, chỉ đến bài toán phải được giải quyết trong chu kỳ này và một mặt, thu hẹp tập hợp các luật cần sử dụng (lệnh 3 của thủ tục SETUP1).

Về nguyên tắc, bài toán này sẽ được xem xét trong số các bài toán xuất hiện từ chu kỳ trước đó (lệnh 2 của FACTS-CONJUNCTION-SETUP, hình 3.9), loại trừ hai trường hợp sau :

- Mọi bài toán xuất hiện từ chu kỳ trước đó bây giờ được coi là đã thiết lập (trả về thông báo 'success' bởi lệnh 1 rồi lệnh 5 của thủ tục FACTS-CONJUNCTION-SETUP). Trong trường hợp này, bài toán đã làm khởi động một luật ở chu kỳ trước đó được coi là đã thiết lập (trả về lần cuối cùng thông báo 'failure' bởi thủ tục SETUP-A-FACT).
- Có thể xảy ra rằng một bài toán xuất hiện từ chu kỳ trước đó không thể được thiết lập (không thực hiện thủ tục FACTS-CONJUNCTION-SETUP do lệnh 4). Trong trường hợp này, bài toán đã làm khởi động chu kỳ trước đó vẫn còn phải được thiết lập (trả về lần cuối cùng thông báo 'failure' bởi thủ tục SETUP-A-FACT).

Trong cả hai trường hợp, về nguyên tắc, việc RESTRICTION sẽ đưa đến một bài toán lấy từ chu kỳ trước trực tiếp và cứ thế tiếp tục.

Thủ tục BACKDIAGRAM–1 dừng, hoặc do tất cả các bài toán lúc đầu được coi là đã được thiết lập, hoặc do một trong chúng không thể được thiết lập.

Tuy nhiên, BACKDIAGRAM–1 có thể không bao giờ dừng, nếu ta thêm vào cơ sở luật RULESBASE ở ví dụ (hình 3.2) một luật thứ 10 là $Q \rightarrow L$.

Các bước GIẢI_QUYẾT_TRANH_ CHẤP và FILTERING trong chu kỳ cơ bản của BACKDIAGRAM–1 đan xen nhau trong các lệnh từ 1 đến 4 của thủ tục SETUP1. Giai đoạn EXECUTION được biểu diễn bởi thủ tục SETUP2.

b. Một vài biến dạng của BACKDIAGRAM–1

BACKDIAGRAM–1 có thể được mở rộng bằng nhiều cách, chẳng hạn :

- Ghi nhớ các sự kiện đã được thiết lập và hiển thị các thông báo mỗi khi các sự kiện

mới xuất hiện.

- Tránh các vòng lặp bằng cách để riêng ra tất cả những luật đã được khởi động một lần.
- Cho phép hệ thống ước lượng một số sự kiện nhờ các câu hỏi của người dùng, thay vì liên kết các luật.

c. Sơ đồ BACKDIAGRAM –2 : tạo sinh các bài toán con theo chiều sâu trừ khi có một luật được kết luận ngay

Từ sơ đồ BACKDIAGRAM−1, bằng cách thay đổi chiến lược theo chiều sâu, ta nhận được sơ đồ BACKDIAGRAM−2 cho trong hình 3.12.

Chú ý rằng các thủ tục SETUP1, SETUP2 và FACTS-CONJUNCTION-SETUP của BACKDIAGRAM−2 và BACKDIAGRAM−1 là giống hệt nhau.

Giả sử ta có tập hợp các luật kết thúc tại sự kiện FACT (lệnh 2 của thủ tục SETUP-A-FACT), ta mong muốn chỉ xét trước tiên nếu một trong các luật này là kết thúc ngay lập tức, nghĩa là nếu các sự kiện của tiền đề của luật này đã có mặt trong FACTSBASE (lệnh 3 của thủ tục SETUP-A-FACT).

Ta xét các luật kết thúc tại sự kiện FACT tương ứng với một di chuyển theo chiều rộng trên đồ thị của hình... Đó là sau khi xem xét theo chiều rộng không thành công, người ta có xu hướng thiết lập các sự kiện thiếu từ một đến nhiều luật. Điều này tương ứng với một sự tăng tiến từ mức theo chiều sâu so với đỉnh biểu diễn FACT.

Chiến lược này được nhìn nhận như là một bước đầu tiên kể từ các chiến lược ưu tiên theo chiều sâu đến các chiến lược ưu tiên theo chiều rộng. Chiến lược này cũng tương ứng với một dạng thức giải quyết xung đột đặc biệt.

```
    procedure SETUP-A-FACT (FACT, CONFLICT)
1.  if FACT in FACTSBASE then return 'success'
2.  CONFLICT ← { R | (R in FACTSBASE) and (FACT in R.conclusion) }
3.  if DIRECTSETUP1 (CONFLICT) ='success' then return 'success'
4.  return SETUP1 (CONFLICT)
    procedure DIRECTSETUP1 (RULES)
1.  if RULES = ∅ then return 'failure'
2.  ARULE ← chọn một luật nào đó từ RULES (chẳng hạn luật gặp đầu tiên)
3.  RULES ← RULES − { ARULE }
4.  THEFACTS ← { F | F in ARULE.premise }
5.  if (THEFACTS là tập hợp con của FACTSBASE) then return 'success'
6.  return DIRECTSETUP1 (RULES)
    procedure SETUP1 (RULES, ARULE)
1.  if RULES = ∅ then return 'failure'
2.  ARULE ← chọn một luật nào đó từ RULES (chẳng hạn luật gặp đầu tiên)
3.  RULES ← RULES − { ARULE }
4.  if SETUP2 (ARULE) = 'success' then return 'success'
5.  return SETUP1 (RULES)
    procedure SETUP2 (RULE, THEFACTS)
1.  THEFACTS ← { F | F in ARULE.premise }
2.  return FACTS-CONJUNCTION-SETUP (THEFACTS)

    procedure FACTS-CONJUNCTION-SETUP (FACTS, AFACT)
1.  if FACTS = ∅ then return 'success'
2.  AFACT ← lấy một sự kiện nào đó từ FACTS (chẳng hạn luật gặp đầu tiên)
3.  FACTS ← FACTS − { AFACT }
4.  if SETUP-A-FACT (AFACT) = 'failure' then return 'failure'
5.  return FACTS-CONJUNCTION-SETUP (FACTS)
```

Hình 3.12. Sơ đồ BACKDIAGRAM –2

Các máy chế độ thăm dò nhưng đơn điệu

Cả hai thủ tục BACKDIAGRAM–2 và BACKDIAGRAM–1 đều hoạt động theo chế độ thăm dò vì rằng dãy các khởi động luật mà vô ích có thể bị loại bỏ và được thay thế bởi các khởi động khác : các sự kiện cần thiết lập được thay thế bởi các sự kiện khác. Mặc dù có sự quay lui tác động lên các sự kiện cần thiết lập, nhưng ta thấy rằng cả hai thủ tục BACKDIAGRAM–2 và BACKDIAGRAM–1 đều không lấy đi, và cũng không thêm vào, các sự kiện đã thiết lập. Cả hai thủ tục đều hoạt động theo chế độ đơn điệu.

II.4. Tìm các luật nhờ liên kết hỗn hợp, với chế độ thăm dò không đơn điệu

Hình 3.13 dưới đây thể hiện một cơ sở tri thức ký hiệu dùng để tạo ra các kế hoạch hành động (action plan) và phục vụ cho sơ đồ máy ví dụ MIXEDIAGRAM ở hình 3.15. Luật R1 được giải thích như sau : «để giải bài toán P1, cần thực hiện các hành động A1 rồi A2». Luật R4 được giải thích như sau : «để giải bài toán P3, biết rằng sự kiện F1 đã được thiết lập, chỉ cần thực hiện hành động A5 rồi giải bài toán P4».

RULESBASE :

R1 :	P1	←	A1, A2
R2 :	P1	←	A3
R3 :	P2	←	P1, A4, P3
R4 :	P3, F1	←	A5, P4
R5 :	P3, F2	←	P5
R6 :	P3	←	A4, P6
R7 :	P4, F2, F3	←	A2

FACTSBASE (các sự kiện đã thiết lập) : { F3, F4 }

PROBLEMSTACK (các sự kiện cần thiết lập) : danh sách (P2)

Bảng các hành động kết thúc (những bài toán sơ khởi)

Hành động	Sự kiện đã thêm vào	Sự kiện đã loại bỏ
A1	F1, F2	Không có
A2	Không có	F1
A3	F1	Không có
A4	Không có	Không có
A5	F2	F4

Lúc đầu, PLAN là danh sách rỗng

Hình 3.13 Một cơ sở tri thức khởi đầu của sơ đồ máy MIXEDIAGRAM

a. Liên kết hỗn hợp

Các luật được gọi bằng cách kiểm tra thành phần bên trái của chúng. Mỗi phần tử thuộc thành phần bên trái luật là một bộ lọ, rút gọn thành một ký hiệu trong ví dụ đang xét. Rõ ràng rằng bộ lọc đầu tiên dựa trên một phần đặc biệt của cơ sở sự kiện, chỉ chứa các sự kiện cần thiết lập (hay các bài toán cần giải) được gọi là PROBLEMSTACK (viết tắt PRS). Các bộ lọc khác, thông thường thuộc thành phần bên trái của luật dựa trên một phần của cơ sở sự kiện, theo quy ước, chỉ chứa các sự kiện đã thiết lập, là FACTSBASE.

Thành phần bên trái của luật tạo nên *bộ khởi động* của luật đó. Các luật được gọi theo *liên kết hỗn hợp* theo nghĩa rằng các bộ lọc của bộ khởi động có thể quan hệ đồng thời đến các sự kiện đã thiết lập và các sự kiện cần thiết lập. Khi không có các bộ lọc liên quan đến FACTSBASE (các bộ lọc sự kiện), máy hoạt động theo kiểu suy diễn lùi.

Thành phần bên phải của luật là *thân* của luật đó. Nó xác định các hành động cần thực hiện trên PRS và sau đó là trên FACTSBASE. Chẳng hạn, nếu phần tử đầu tiên của PRS là P1, luật R1 có thể được khởi động : A1 và A2 được đặt một cách tương ứng ở các vị trí đầu tiên và vị trí thứ hai của PRS, trong khi đó, P1 được lấy ra.

b. Lập hay «tạo sinh kế hoạch»

Các hành động A1, A2, A5, v.v... có thể được xem như là những bài toán giải được ngay, được gọi là các *bài toán sơ khởi* (primitive problem). Chúng cũng biểu diễn các hành động sơ cấp có thể có mặt trong một kế hoạch hành động, được gọi là các *hành động kết thúc* (terminal act). Khi một hành động như vậy được chèn vào trong một kế hoạch đang xây dựng, cần biểu diễn các kết quả của nó : thêm vào hay lấy các sự kiện ra. Thêm một sự kiện vào được giải thích như là thiết lập một sự kiện. Lấy ra một sự kiện được giải thích như là không còn được thiết lập nữa.

Một ký hiệu được xem là biểu diễn một hành động kết thúc nếu nó có mặt trong *bảng các hành động kết thúc*. Khi đó, luật R4 tương đương với :

P3, F1 ← ADD F2, MOVE F4, R4

Sau khi khởi động một luật, khi một hành động có mặt tại vị trí đầu tiên (hay là *đỉnh*) của PRS, thì nó được lấy ra ngay (từ đỉnh) để *đặt vào cuối danh sách* PLAN. Tại thời điểm này, quá trình đặt vào, lấy ra mô tả trong bảng các hành động kết thúc ở hình 3.13 trên đây được thực hiện. Chẳng hạn, khi A5 được lấy ra, F2 được thêm vào trong FACTSBASE và F4 được lấy ra.

c. Không đơn điệu

Sự khác nhau cơ bản của MIXEDIAGRAM so với các sơ đồ máy trước đây là ở khả năng điều khiển, bởi việc khởi thảo ra một luật (cùng với những thông tin liên quan như trong bảng các hành động kết thúc), việc lấy ra các kết luận trước đó đã được thiết lập bởi áp dụng các luật khác.

Khi một máy suy diễn xét lại các sự kiện đã được thiết lập, người ta nói rằng máy đó là *không đơn điệu*. Như vậy, MIXEDIAGRAM là máy không đơn điệu.

d. Khởi động ưu tiên theo độ sâu

```
    procedure SOLVE-A-PROBLEM (PROBLEM)
1.  PROBLEMSTACK ← danh sách chỉ chứa mỗi phần tử PROBLEM
2.  PLAN ← danh sách rỗng
3.  ACONTEXT ←  danh sách các phần tử
                PROBLEMSTACK, FACTSBASE, PLAN
4.  if SOLVE (ACONTEXT, RULESBASE) = 'failure' then return 'failure'
5.  Soạn thảo PLAN
6.  return 'success'
    procedure SOLVE (THECONTEXT, RULES)
1.  if PROBLEMSTACK = ∅ then return 'success'
2.  RULE-INDIC ← RUN-A-CYCLE (đỉnh của PROBLEMSTACK, RULES)
3.  if RULE-INDIC = 'failure' then return 'failure'
4.  ACONTEXT ←  danh sách các phần tử sau khi ra khỏi RUN-A-CYCLE :
                PROBLEMSTACK, FACTSBASE, PLAN
5.  if SOLVE (ACONTEXT, RULESBASE) = 'success'then return 'success'
6.  if RULE-INDIC = 'primitive' then return 'failure'
7.  PROBLEMSTACK  ← Phần tử đầu tiên của THECONTEXT
8.  FACTSBASE     ← Phần tử thứ hai của THECONTEXT
9.  PLAN          ← Phần tử thứ ba của THECONTEXT
10. RULES ← RULE-INDIC đã lấy đi phần tử đầu tiên
11. return SOLVE (ACONTEXT, RULESBASE)
    procedure RUN-A-CYCLE (PROBLEM, RULES, ARULE)
1.  if PROBLEM có mặt trong bảng các sự kiện kết thúc then
    1.1  begin
    1.2  Loại bỏ PROBLEM là đỉnh của PROBLEMSTACK
    1.3  Đặt PROBLEM vào cuối PLAN
    1.4  Thực hiện đặt vào và lấy ra đối với FACTSBASE như đạ mô tả đối với
         PROBLEM trong bảng các sự kiện kết thúc
    1.5  return 'primitive'
    1.6  end
2.  if RULES = ∅ then return 'failure'
3.  ARULE ← Luật đầu tiên của RULES
4.  if ARULE là khởi động được
         (bộ khởi động tương thích với PROBLEM và FACTSBASE) then
    4.1  begin
    4.2  Loại bỏ đỉnh của PROBLEMSTACK
    4.3  Đặt các phần tử trong thân của ARULE vào PROBLEMSTACK,
         chú ý phần tử đầu tiên nằm ở đỉnh và cứ thế tiếp tục
    4.4  return RULES
    4.5  end
5.  return RUN-A-CYCLE (PROBLEM, RULES đã lấy đi ARULE)
```

Hình 3.14. Sơ đồ MIXEDIAGRAM

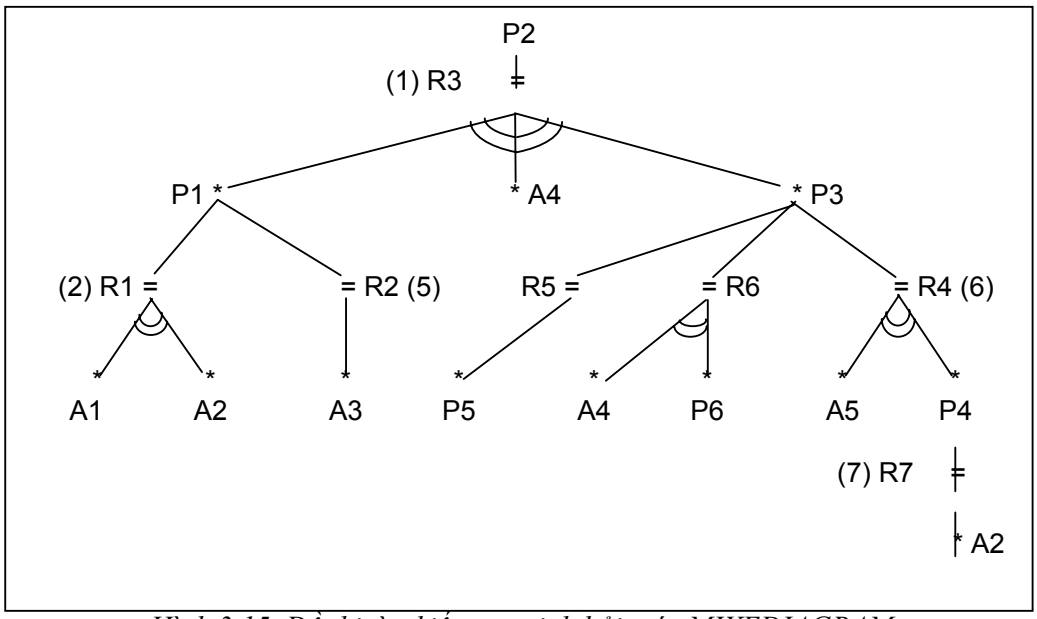
Máy MIXEDIAGRAM được mô tả trong hình 3.14 trên đây. Xuất phát từ cơ sở tri thức cho trong hình 3.13, ta có thể sử dụng lời gọi :

SOLVE-A-PROBLEM (P2)

Máy khởi động luật đầu tiên của RULESBASE có bộ khởi động tương thích với PRS và FACTSBASE. Luật này được khởi động và sẽ là luật đầu tiên của RULESBASE tương thích với tình huống được tạo ra bởi việc khởi động của luật trước đó : quá trình khởi động ưu tiên theo chiều sâu.

Nếu PRS trở nên rỗng, máy dừng lại : bài toán ban đầu đã được giải, trạng thái hiện hành của PLAN được cung cấp. Nếu PRS không rỗng, nhưng nếu không còn một luật nào khởi động được, MIXEDIAGRAM sẽ có khuynh hướng quay lui : nghĩa là nó sẽ áp dụng một luật mới cho bài toán cuối cùng không là sơ khởi đã được lấy ra từ PRS, và cứ thế tiếp tục.

Các máy BACKDIAGRAM–1 và BACKDIAGRAM–2 cũng có cơ chế quay lui. Tuy nhiên, đối với MIXEDIAGRAM, điều mới ở đây là khi xem xét lại cách một bài toán được phân tách ra, MIXEDIAGRAM lại được dẫn đến thiết lập lại bối cảnh (trạng thái của PRS, FACTSBASE, PLAN) giống hệt bối cảnh đã tồn tại ở thời điểm bài toán này được xem xét, trước khi xem xét lại lần nữa, nếu có thể, bởi luật phân tách khác. Do vậy, sơ đồ MIXEDIAGRAM dẫn đến một chương trình khác hẳn.



Hình 3.15. Đồ thị tìm kiếm tạo sinh bởi máy MIXEDIAGRAM

Hình 3.15 trên đây là đồ thị VÀ-HOẶC biểu diễn sự kết nối các luật tạo ra từ cơ sở tri thức cho trong ví dụ ở hình 3.13.

Ta có :

Các bước	Kết quả hành động	Nội dung PROBLEMSTACK	Nội dung FACTSBASE	Nội dung PLAN
1	Khởi động luật R3	(P1 A4 P3)	không thay đổi	rỗng
2	Khởi động luật R1	(A1 A2 A4 P3)	không thay đổi	rỗng
3	Lấy A1, A2, A4 từ đỉnh PRS	(P3)	{F2 F3 F4}	(A1 A2 A4)
4	Khởi động luật R5	(P5)	không thay đổi	không thay đổi
5	Không còn luật nào được khởi động, quay lại giải P3	(P3)	{F2 F3 F4}	(A1 A2 A4)
		RULES={R6 R7}		
6	Khởi động luật R6	(A4 P6)	không thay đổi	không thay đổi
7	Lấy A4 từ đỉnh PRS	(P6)	không thay đổi	(A1 A2 A4 A4)
8	Không còn luật nào được khởi động, quay lại giải P3	(P3)	{F2 F3 F4}	(A1 A2 A4)
		RULES = {R7}		
9	Không còn luật nào được khởi động, quay lại giải P1	(P1 A4 P3)	{F3 F4}	rỗng
		RULES = {R4 R3 R4 R5 R6 R7}		
10	Khởi động luật R2	(A3 A4 P3)	không thay đổi	rỗng
11	Lấy A3 và A4 từ đỉnh PRS	(P3)	{F1 F3 F4}	(A3 A4)
12	Khởi động luật R4	(A5 A4)	không thay đổi	không thay đổi
13	Lấy A5 từ đỉnh PRS	(P4)	{F1 F2 F3}	(A3 A4 A5)
14	Khởi động luật R7	(A2)	không thay đổi	không thay đổi
15	Lấy A2 từ đỉnh PRS	rỗng	{F1 F2 F3}	(A3 A4 A5 A2)
16	Máy dừng			

e. Giải thích sơ đồ MIXEDIAGRAM

Thủ tục SOLVE để rút gọn bài toán đầu (gọi là PB) khỏi PROBLEMSTACK khi thủ tục RUN-A-CYCLE được gọi. Có bốn trường hợp xảy ra :

Trường hợp 1 :

Thủ tục RUN-A-CYCLE trả về 'failure' khi PB không tương ứng với một hành động kết thúc và không được phân tách bởi một luật nào từ danh sách RULES. Trong trường hợp này, thủ tục hiện hành SOLVE dừng và trả về 'failure' ở mức cao hơn.

Tham đối thứ hai của RUN-A-CYCLE là RULE-INDIC nhận được giống như là tham đối thứ hai của SOLVE. Tại lời gọi đầu tiên của SOLVE bởi SOLVE-A-PROBLEM (lệnh 4), tham đối này có giá trị của RULESBASE. Tương tự như vậy đối với một số lời gọi của SOLVE bởi chính nó (lệnh 5). Đối với các lời gọi khác (lệnh 11 của SOLVE), tham đối này không còn là một danh sách con của RULESBASE.

Nếu RUN-A-CYCLE không trả về 'failure' là vì, hoặc PB là một hành động kết thúc, hoặc PB là một bài toán phân tách được bởi một trong các luật của danh sách cung cấp cho lời gọi RUN-A-CYCLE. Trong hai tình huống này, người ta gọi ngay lập tức thủ tục SOLVE với tham đối thứ nhất là bối cảnh của việc xử lý PB (tiến triển của FACTSBASE và/hoặc của PROBLEMSTACK) bởi RUN-A-CYCLE, và tham đối thứ hai là toàn bộ RULESBASE. Sau đây, ta gọi R là lời gọi này đối với thủ tục SOLVE.

Trường hợp 2 :

Nếu lời gọi cuối cùng R này đối với SOLVE trả về 'success', thủ tục SOLVE hiện hành trả về 'success' ở mức cao hơn. Nếu không, việc xử lý phải phân biệt tuỳ theo thủ tục RUN-A-CYCLE đã nhận biết PB như là một hành động kết thúc, nghĩa là một bài toán sơ khởi không thể phân tách được, hoặc đã được dự kiến phân tách.

Trường hợp 3 :

Thủ tục RUN-A-CYCLE sẽ trả về 'primitive' nếu PB là một hành động kết thúc. Trong

trường hợp này, sự thất bại của lời gọi R đối với SOLVE có nghĩa cần phải xem xét lại việc đặt PB vào đầu danh sách PROBLEMSTACK : lúc này, thủ tục SOLVE hiện hành trả về 'failure' ở mức cao hơn.

Trường hợp 4 :

Nếu PB không là một hành động kết thúc, thì PB phải phân tách được. Như vậy, thủ tục RUN-A-CYCLE đã khởi động một luật để phân tách PB, đặt kết quả phân tách của PB vào đầu danh sách PROBLEMSTACK và trả về danh sách con của RULESBASE có phần tử đầu tiên là luật vừa được khởi động. Trong trường hợp này, lời gọi R đối với SOLVE không thành công do có sự xuất hiện các bài toán con của PB. Vì vậy, cần thử tìm một phân tách khác đối với PB : bối cảnh đã tồn tại trước lời gọi RUN-A-CYCLE được khôi phục và xuất hiện một lời gọi mới đối với SOLVE. Tham đối thứ nhất của lời gọi này đối với SOLVE là danh sách các luật nhận được từ tham đối của SOLVE hiện hành, sau khi lấy đi luật đã thử.

f. Một vài biến tấu đơn giản khác của MIXEDIAGRAM

1. Dãy các bài toán. Tương tác

Trên nhiều điểm khác nhau, người ta có thể phỏng theo sơ đồ BACK DIAGRAM–1. Để đưa ra một dãy có thứ tự các bài toán, chỉ cần xem xét lại thủ tục SOLVE-A-PROBLEM sao cho dãy này được nạp vào PROBLEMSTACK khi mới khởi động. Ta có thể dự kiến một hoạt động tương tác : khi không có một luật nào áp dụng được cho bài toán nằm trên đỉnh của PROBLEMSTACK (xem lệnh 2 của RUN-A-CYCLE), người ta có thể quyết định, trước khi quay lui, hỏi người sử dụng xem có cần hiển thị phần tử đỉnh của PROBLEMSTACK, và hiển thị, hoặc tất cả hoặc một phần của FACTSBASE hay không ? Người sử dụng cũng có thể chỉ ra bài toán có thể giải được trong điều kiện đã định.

2. Điều khiển theo độ sâu

Vì các sự kiện đã thiết lập có thể được xem xét lại, người ta cần phải khởi động một luật nhiều lần, trong cùng một bối cảnh, nên có thể dẫn đến nguy cơ xoay vòng. Để điều khiển sự tiến triển tìm kiếm theo độ sâu, người ta có thể gắn mỗi phần tử PB của PROBLEMSTACK với một «mức» bằng mức của bài toán là bài toán đã được phân tách để xuất hiện PB cộng thêm 1. Bài toán khởi đầu sẽ được gắn mức 0. Người ta không cho phép giải quyết bài toán có mức vượt quá một giá trị giới hạn đã được ấn định trước. Chú ý rằng mức của một bài toán có thể giảm dần theo quá trình tìm kiếm.

3. Tìm kiếm tất cả các lời giải

Có thể có nhiều kế hoạch hành động cùng biểu diễn lời giải của một bài toán đã cho. Trong ví dụ ở hình 3.13, người ta có thể tìm thấy hai kế hoạch để giải P là kế hoạch (A1 A2) và kế hoạch (A3).

Nếu ta thêm một luật thứ tám là P1 ← A1, ta sẽ tìm thấy hai kế hoạch để giải P2 là (A3 A4 A5 A2) và kế hoạch (A1 A4 A5 A2).

Lúc này ta có thể thay đổi MIXEDIAGRAM sao cho khi cần, nó có thể tìm được các kế hoạch khác nhau này. Muốn vậy, ta sẽ bỏ dòng lệnh 5 của SOLVE-A-PROBLEM và thay thế lệnh 1 của SOLVE bởi :

```
1     if PROBLEMSTACK = ∅ then
1.1   begin
1.2   Soạn thảo PLAN
1.3   Yêu cầu người sử dụng lập PLAN khác (Y/N?)
1.4   if đồng ý (Y)then return 'failure'
1.5   return 'success'
1.6   end
```

Ở lệnh 1.4, người ta đã tạo ra một 'failure' để bắt buộc máy tìm kiếm một lời giải khác có thể, xuất phát từ bối cảnh hiện hành.

II.5. Sơ đồ máy sử dụng biến

(suy diễn lùi với chế độ bắt buộc và đơn điệu)

Hình 3.16 trình bày một ví dụ về tri thức và các bài toán sẽ được giải bởi máy BACKDIAGRAM–3 cho bởi hình 3.19. Chú ý rằng trong hình 3.16, các thể hiện tri thức có chứa các biến, các biến được đặt tên quy ước là x, y, z, u và v. Các ký hiệu C1, C2, C3, C4, C5, C8 biểu diễn các sự kiện đã được thiết lập, C6, C7, C9 biểu diễn luật.

Sự kiện C3 trong cơ sở thể hiện pierre là cha của jean. Sự kiện C6 nói lên rằng dù x hay y là như thế nào, ngay khi SON(y, x) được thiết lập, nghĩa là khi đã thiết lập y là con của x, thì có thể thiết lập FATHER(x, y), nghĩa là x là cha của y. Hay ngược lại : dù x hay y là như thế nào, để thiết lập FATHER(x, y), chỉ cần thiết lập SON(y, x). Sự kiện C6 thể hiện dù x, y và z là như thế nào, để thiết lập rằng x là ông của y, chỉ cần thiết lập x là cha của z và z là cha của y.

Ở đây, ta quy ước rằng các thành phần bên trái luật là kết luận của luật, các thành phần bên phải luật là tiền đề. Các luật luôn luôn chỉ chứa một kết luận duy nhất và có một hoặc nhiều tiền đề.

KNOWLEDGESBASE :

C1 BLOND (marc)
C2 BROWN (jean)
C3 FATHER (pierre, jean)
C4 FATHER (marc, pierre)
C5 FATHER (jean, rené)
C6 FATHER (x, y) ← SON (y, x)
C7 GRANDFATHER (x, y) ← FATHER (x, z), FATHER (z, y)
C8 SON (marc, georges)

PROBLEMSTACK (các sự kiện cần thiết lập) :

List (GRANDFATHER (u, v) BLOND (u))

Hình 3.16. Cơ sở tri thức và PROBLEMSTACK của BACKDIAGRAM–3

Ví dụ 1 :

Cần «*tìm x và y* sao cho x là ông của y, x có tóc hoe». Trạng thái ban đầu của PROBLEMSTACK (viết tắt PBS) cho trong hình 3.16 trên đây.

a. Hoạt động của BACKDIAGRAM–3

Hoạt động của máy BACKDIAGRAM–3 để tìm lời giải cho ví dụ 1 như sau :

- ***Chu kỳ đầu tiên :***

Máy lần lượt giải các bài toán đã được sắp xếp trong PBS. Trước tiên, máy ghép đôi biểu thức GRANDFATHER (u, v) lần lượt với các tri thức từ C1 đến C8 theo thứ tự này. Nếu C1 là một luật, thì việc ghép đôi chỉ tác động lên phần kết luận của luật. Việc ghép đôi giữa GRANDFATHER (u, v) với kết luận của C7 thành công : thật vậy, bằng cách áp dụng các thay thế «u bởi x» và «v bởi y» trong GRANDFATHER (u, v), ta nhận được hai biểu thức giống hệt nhau.

Ta nói là đã *hợp nhất* (unified) GRANDFATHER (u, v) và GRANDFATHER (x, y) bởi *phép hợp nhất* (unifier) ký hiệu UNIF như sau :

UNIF = «u bởi x» và «v bởi y» hay được viết tắt là :
UNIF = (x□u ; y□v)

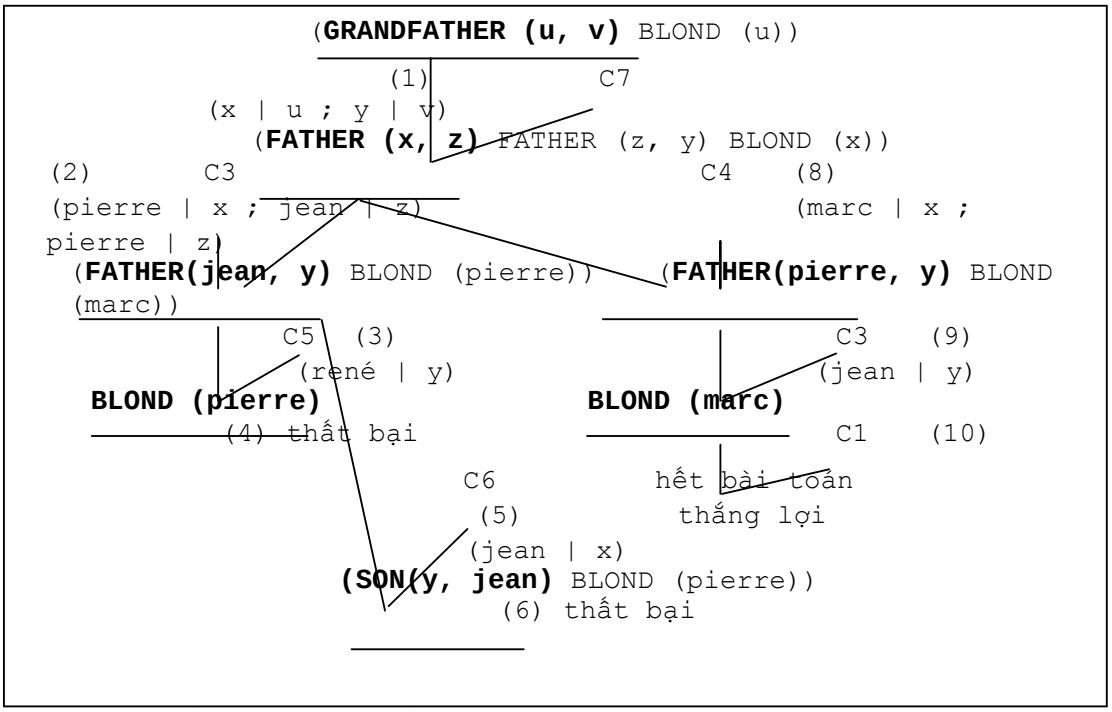
Trong trường hợp đặc biệt này, sẽ có một trong các biểu thức không bị thay đổi bởi áp dụng phép hợp nhất. Trong khi ghép đôi, máy sẽ biến đổi PBS bằng cách thay thế GRANDFATHER (u, v) bởi phần tiền đề của C7, sau khi đã áp dụng phép hợp nhất cho phần tiền đề này. Có thể phần tiền đề không bị thay đổi bởi đã áp dụng phép hợp nhất.

UNIF tiếp tục được áp dụng cho phần còn lại của PBS, hay BLOND (u). Ta nhận được :

PBS = (FATHER (x, z) FATHER (z, y) BLOND (x))

Như vậy, ta đã dẫn bài toán ban đầu («hãy xác định nếu tồn tại u, v sao cho u hoặc là ông của v, hoặc v có tóc hoe») thành bài toán «hãy xác định nếu tồn tại x, y và z sao cho x là cha của z và z là cha của y và x có tóc hoe».

Hình 3.17 dưới đây biểu diễn các chu kỳ thực hiện của BACKDIAGRAM–3 :



Hình 3.17. Cơ sở tri thức và PROBLEMSTACK của BACKDIAGRAM–3

Chú ý : Xuất phát từ các biểu thức (GRANDFATHER (u, v) BLOND (u)) và GRANDFATHER (x, y) ← FATHER (x, z), FATHER (z, y), ta đã suy diễn được :

PBS = (FATHER (x, z), FATHER (z, y) BLOND (x))

Nội dung mới của PBS chính là *kết quả hợp giải* (resolvant) từ các biểu thức của PBS và của luật C7 theo *nguyên lý hợp giải* đã giới thiệu trước đây.

□ ***Chu kỳ thứ hai :***

Phần tử đầu tiên của PBS là FATHER (x, z) sẽ được so sánh với các tri thức từ C1 đến C8 theo thứ tự này.

Máy phát hiện ra phép hợp nhất UNIF = (pierre □x, jean □ z) giữa FATHER (x, z) và C3. Khi đó, sẽ tồn tại x (x = pierre) và z (z = jean) sao cho x phải là cha của z. Bài toán đầu tiên của PBS xem như đã được giải và được lấy ra khỏi PBS. Những bài toán cần giải còn lại của PBS tiếp tục được hợp nhất bởi UNIF và dẫn đến kết quả :

PBS = (FATHER (jean, y) BLOND (pierre))

Cần để ý là so với chu kỳ trước, việc suy diễn ở chu kỳ này xảy ra như C3 là luật có phần tiền đề rỗng.

- ***Chu kỳ thứ ba :***

FATHER (jean, y) được ghép đôi với C5 nhờ phép hợp nhất UNIF = (rené □y) để nhận được :

PBS = (BLOND (pierre))

- ***Chu kỳ thứ tư :***

BLOND (pierre) không thể được ghép đôi với tri thức nào trong các tri thức từ C1 đến C8. Chu kỳ thứ tư thất bại (dừng không qua giai đoạn EXECUTION).

- ***Chu kỳ thứ năm :***

Bước RESTRICTION yêu cầu máy quay lui. Lúc này, PBS được khôi phục lại tình trạng ở thời điểm đầu chu kỳ trước đó là không thể tiếp tục (chu kỳ thứ ba) :

PBS = (FATHER (jean, y) BLOND (pierre))

Việc ghép đôi được duy trì trong chu kỳ thứ ba này (tương ứng với bước CONFLICT-RESOLUTION). Máy chuẩn bị ghép đôi lại (bước FILTERING) đối với FATHER (jean, y) ***xuất phát từ C6***. Việc ghép đối với C6 thành công, do có phép hợp nhất UNIF = (jean □ x), từ đó :

PBS = (SON (y, jean) BLOND (pierre))

- ***Chu kỳ thứ sáu :***

SON (y, jean) không thể ghép đôi với tri thức nào trong các tri thức từ C1 đến C8. Chu kỳ thứ sáu không thể kết thúc bình thường (dừng không qua giai đoạn EXECUTION).

- ***Chu kỳ thứ bảy :***

Bước RESTRICTION yêu cầu máy tiếp tục quay lui. PBS khôi phục lại tình trạng ở thời điểm đầu chu kỳ trước đó là không thể tiếp tục (chu kỳ năm) :

PBS = (FATHER (jean, y) BLOND (pierre))

Việc ghép đôi được duy trì trong chu kỳ thứ năm này (tương ứng với bước CONFLICT-RESOLUTION). Máy chuẩn bị ghép đôi lại (bước FILTERING) đối với FATHER (jean, y) ***xuất phát từ C7***. Việc ghép đôi với C7 thất bại, do đó chu kỳ thứ bảy không thể chuyển qua giai đoạn EXECUTION.

- ***Chu kỳ thứ tám :***

Máy tiếp tục quay lui. PBS được khôi phục lại tình trạng ở thời điểm đầu chu kỳ hai, nghĩa là :

PBS = (FATHER (x, z), FATHER (z, y) BLOND (x))

Việc ghép đôi được duy trì trong chu kỳ thứ hai này. Máy chuẩn bị ghép đôi lại đối với FATHER (x, z) ***xuất phát từ C4***. Việc ghép đối với C6 thành công, do có phép hợp nhất UNIF = (marc □ x ; pierre □ z), từ đó :

PBS = (FATHER (pierre, y) BLOND (marc))

- ***Chu kỳ thứ chín :***

Máy có thể ghép đôi FATHER (pierre, y) với C3 nhờ phép hợp nhất :
UNIF = (jean □ y)
Từ đó :
PBS = BLOND (marc)

- ***Chu kỳ thứ mười :***

Máy có thể ghép đôi BLOND (marc) với C1 mà không cần dùng một phép hợp nhất nào. Do PBS trở nên rỗng, nên dãy các phép thế đưa vào đối với các biến khởi động của PBS

không được sử dụng đến (không quay lui) tạo thành một lời giải cho bài toán ban đầu : tồn tại u (u = marc) và v (v = jean) sao cho u là ông của jean và u có tóc hoe.

Kết quả xử lý của sơ đồ máy BACKDIAGRAM–3 cho ở hình 3.19 đối với bài toán trên đây chính là dãy các phép thế như sau :

UNIF = (x □ u ; y □ v ; marc □ x ; jean □ y)

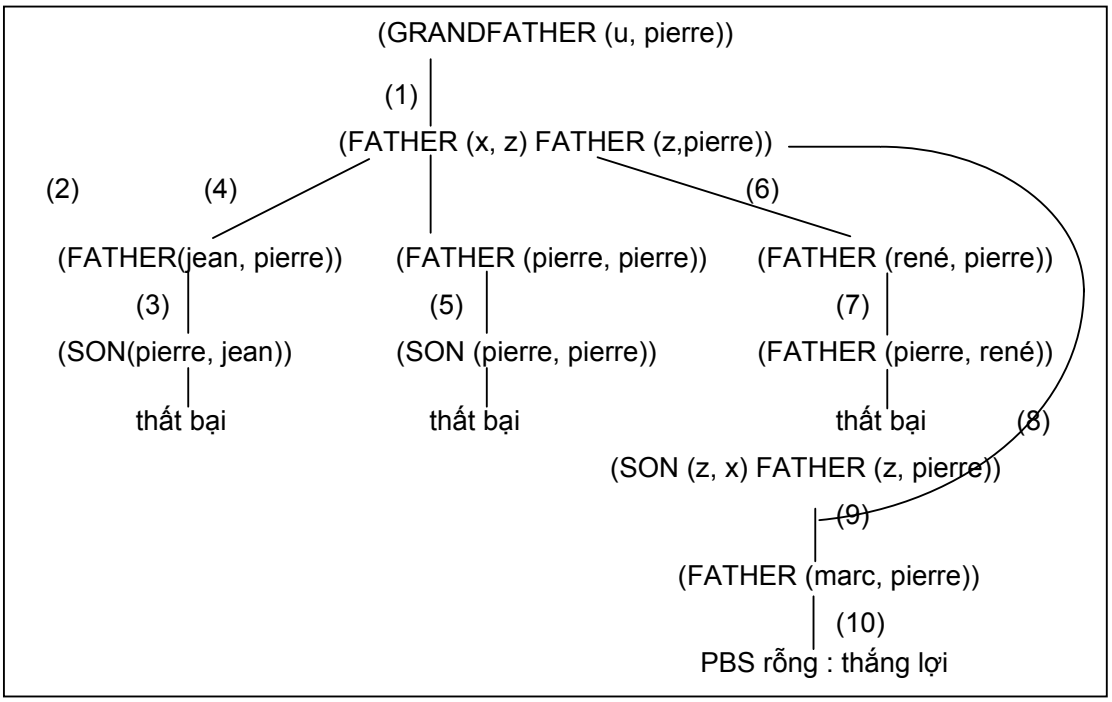
Ví dụ 2 :

Từ các tri thức đã cho ở hình 3.16, ta muốn «tìm kiếm u sao cho u là ông của pierre». Từ đó, nội dung ban đầu của PROBLEMSTACK sẽ là :

PBS = GRANDFATHER (u, pierre)

Bằng cách nhại lại (bắt chước) y hệt các bước vừa trình bày trên đây, ta nhận được kết quả hợp giải cho trong hình 3.18 dưới đây. Câu trả lời được đưa ra ở nhánh thứ tư : tồn tại u, u = georges, sao cho u là ông của pierre.

Các con số trong hình tương ứng với thứ tự của các hợp nhất.



Hình 3.18. Đồ thị hợp nhất của BACKDIAGRAM–3 để giải ví dụ 2

b. BACKDIAGRAM–3 : sơ đồ máy suy diễn kiểu Prolog

Sơ đồ máy BACKDIAGRAM–3 sử dụng các tri thức kiểu suy diễn lùi và ưu tiên theo độ sâu. Sơ đồ này không thiết lập các sự kiện mới mà chỉ đơn giản là biến đổi PBS, danh sách có thứ tự các sự kiện đang còn phải thiết lập. Do không bao giờ lấy ra các sự kiện đã thiết lập, nên cách hoạt động của máy là đơn điệu. Mặt khác, do các trạng thái trước của PBS có thể được khôi phục (quay lui), máy hoạt động theo chế độ thăm dò.

Giai đoạn SELECTION/RESTRICTION nhằm tập trung sử dụng các tri thức (luật và sự kiện) đối với phần tử đầu tiên của PBS. Tuỳ theo sự quay lui của máy, trạng thái của PBS có thể xác định được.

Ngôn ngữ thể hiện tri thức cho trong hình 3.16 và được sử dụng trong sơ đồ máy BACKDIAGRAM–3 tương tự ngôn ngữ Prolog. Các biểu thức biểu diễn tri thức là những «mệnh đề Horn». Chú ý rằng thông thường, các hệ chuyên gia chỉ thừa nhận các biến trong

các luật, mà không thừa nhận các biến trong các sự kiện như trong hai ví dụ trên.

Việc tạo sinh các kết quả hợp giải *ưu tiên theo độ sâu* có thể dẫn đến vòng lặp, nhưng có thể tránh được trong chiến lược *ưu tiên theo chiều rộng*. Chẳng hạn, vòng lặp gây ra bởi chiến lược ưu tiên theo độ sâu : nếu ta thêm phần tử C9 vào cuối danh sách các tri thức KNOWLEDGESBASE như sau :

C9 : SON (x, y) ← FATHER (y, x)

Và nếu xét lại bài toán GRANDFATHER (u, pierre) ở ví dụ 2, máy sẽ áp dụng vô hạn lần các luật C6–C9–C6–C9–...

```
Procedure JUSTIFY (PROBLEMSTACK, REMAININGBASE, aproblem,
                   aknowledge, theconclusion, substitu1, substitu2, theproblems)
1.  if PROBLEMSTACK = ∅ then return danh sách rỗng
2.  if REMAININGBASE = ∅ then 'failure'
3.  APROBLEM ← phần tử đầu tiên của PROBLEMSTACK
4.  AKNOWLEDGE ← phần tử đầu tiên của REMAININGBASE
    (tuỳ theo phép hợp nhất tên biến có thể thay đổi)
5.  THECONCLUSION ← phần kết luận của AKNOWLEDGE
6.  SUBSTITU1 ← UNIFICATION (APROBLEM, THECONCLUSION)
7.  if SUBSTITU1 ≠ 'failure' then
7.1     begin
7.2     THEPROBLEMS ← đặt có thứ tự các phần tử của PROBLEMSTACK khác
        APROBLEM vào cuối danh sách các tiền đề của AKNOWLEDGE với việc
        áp dụng liên tiếp các phép hợp nhất của danh sách có thứ tự SUBSTITU1
        cho biểu thức trước đó của THEPROBLEMS
7.3     SUBSTITU2 ← JUSTIFY (THEPROBLEMS, KNOWLEDGESBASE)
7.4     if SUBSTITU2 ≠ 'failure' then return danh sách nhận được bằng cách thay thế các phần
        tử của SUBSTITU2 theo thứ tự vào cuối của SUBSTITU1
7.5     end
8.  REMAININGBASE ← REMAININGBASE – {AKNOWLEDGE}
9.  JUSTIFY (PROBLEMSTACK, REMAININGBASE)
```

c. Giải thích sơ đồ máy BACKDIAGRAM–3

Có hai tham đối trong lời gọi thủ tục JUSTIFY. Tham đối thứ nhất là một danh sách có thứ tự các bài toán cho trước. Chú ý rằng sự xuất hiện của một biến trong biểu diễn bài toán, chẳng hạn biến x trong P(x), tương ứng với một câu hỏi về giá trị của x để có P(x). Tham đối thứ hai là một danh sách có thứ tự các tri thức cho trước lúc khởi động thủ tục. Lời gọi thủ tục có dạng :

JUSTIFY (PBS, KNOWLEDGESBASE)

trong đó PBS và KNOWLEDGESBASE có giá trị cho trong ví dụ 1 hình 3.16.

Khi thủ tục JUSTIFY kết thúc (có thể không kết thúc), JUSTIFY trả về hoặc giá trị 'failure', hoặc lời giải là một danh sách (có thể rỗng) các phép thế trên các biến của bài toán ban đầu. Ví dụ, trong ví dụ 1 hình 3.16, thủ tục JUSTIFY trả về lời giải là danh sách có thứ tự (x □ u ; y □ v ; marc □ x ; jean □ y).

Chú ý rằng lệnh 4 của thủ tục JUSTIFY dùng để kiểm tra nếu các biến có mặt trong AKNOWLEDGE là khác với các biến có mặt trong APROBLEM. Nếu có sự bằng nhau, người ta đổi tên những biến cần thiết trong AKNOWLEDGE cho đến khi thoả mãn. Ví dụ,

nếu :

AKNOWLEDGE = P(x, a) ← Q(x) và nếu APROBLEM = P(b, x)

thì người ta đổi tên biến x thành một biến gốc, giả sử y, trong AKNOWLEDGE, để nhận được : P(y, a) ← Q(y). Sự đổi tên biến là hợp lệ vì biến x (tương ứng với lượng tử toàn thể) trong AKNOWLEDGE thực tế là một biến câm, độc lập với các biến x có mặt trong các thể hiện bài toán hay trong các biểu diễn tri thức khác. Sự đổi tên biến là cần thiết để P(x, a) và P(b, x) có thể hợp nhất được.

Trong chương trước, ta đã giới thiệu chi tiết thuật toán hợp nhất hai trực kiện. Thủ tục UNIFICATION được sử dụng trên đây có thể được lập trình dựa theo thuật toán này. Ta giả thiết rằng lời gọi thủ tục chứa ba tham đối : hai tham đối là hai biểu thức cần hợp nhất, tham đối thứ ba là một danh sách rỗng. Thủ tục trả về 'failure' nếu không tồn tại phép hợp nhất giữa hai biểu thức. Thủ tục trả về kết quả là phép hợp nhất dưới dạng một danh sách (có thể rỗng) các phép thế nếu tồn tại phép hợp nhất.

Bài tập chương 3

1. Cho các câu sau :

- Jhon thích tất cả các loại thức ăn.
- Táo là một loại thức ăn.
- Thịt gà là một loại thức ăn.
- Bất cứ thứ gì mà bất cứ người nào ăn mà không chết đều là thức ăn.
- Bill ăn lạc rang và anh ta vẫn sống.
- Sue bất cứ thứ gi mà Bill ăn.

Yêu cầu :

a. Chuyển các câu trên thành các công thức chỉnh (wff) theo vị từ bậc một.
b. Chuyển các công thức chỉnh câu a. thành dạng mệnh đề.
c. Sử dụng hợp giải để chứng minh rằng Jhon thích lạc rang.
d. Sử dụng hợp giải để trả lời câu hỏi "Sue ăn thức gì ?"

2. Cho các sự kiện sau :

- Các thành viên của câu lạc bộ Đồng hương là Joe, Sally, Bill và Ellen.
- Joe là chồng của Sally.
- Bill là anh trai của Ellen.
- Vợ của mỗi thành viên đã lập gia đình trong câu lạc bộ cũng là thành viên của câu lạc bộ.
- Buổi họp mặt câu lạc bộ Đồng hương gần nhất là ở tại nhà Joe.

Yêu cầu :

a. Chuyển các sự kiện trên thành các vị từ bậc một.
b. Sử dụng hợp giải để chứng minh tính đúng đắn của hai mệnh đề dưới đây. Chú ý nếu không thể chứng minh đúng, thì hãy thêm vào các sự kiện mới để có thể chứng minh :
    - Buổi họp mặt câu lạc bộ Đồng hương gần nhất là ở tại nhà Sally.
    - Ellen chưa lập gia đình.

3. Cho các sự kiện sau :

- Steve chỉ thích các môn dễ học.
- Các môn học về lập trình đều khó.
- Tất cả những môn học trong bộ môn thể dục đều dễ học.

- Bóng chuyền là một môn thể dục

Sử dụng hợp giải để trả lời câu hỏi "Steve thích học môn gì ?"

4. Cho một cơ sở tri thức như sau :

R1. B, D, E → F
R2. D, G → A
R3. C, F → A
R4. B → X
R5. D → E
R6. A, X → H
R7. C → D
R8. X, C → A
R9. X, B → D

Yêu cầu :

a. Vẽ đồ thị và-hoặc từ cơ sở tri thức trên.
b. Vẽ đồ thị và-hoặc minh hoạ thuật toán suy diễn tiến theo chiều sâu.
c. Vẽ đồ thị và-hoặc minh hoạ thuật toán suy diễn tiến theo chiều rộng.
d. Vẽ đồ thị và-hoặc minh hoạ thuật toán suy diễn lùi

5. Làm tất cả các câu hỏi đã cho trong bài tập 4. cho các cơ sở tri thức đã cho ở các bài tập 1, 2, 3 trên đây.

CHƯƠNG 4

Hệ chuyên gia MYCIN và ngôn ngữ OPS5

" J'avais un extrème désir d'apprendre à distinguer
le vrai d'avec le faux pour pour voir clair en mes actions
et marcher avec assurance en cette vie ".

René Descartes

I. Hệ chuyên gia MYCIN

I.1. Giới thiệu MYCIN

Trong những năm 1970, một nhóm các nhà nghiên cứu khoa học tại trường Đại học Tổng hợp Stanford, Hoa Kỳ, làm việc dưới sự chủ trì của giáo sư Ed Feigenbaum, đã xây dựng giả thuyết rằng sự thông minh (intelligence) được căn cứ trên sự lưu trữ những khối lượng lớn về tri thức. Họ đã tìm ra kỹ thuật "biểu diễn tri thức" và tiến hành thực hiện dự án lập trình nghiệm suy (HPP: Heuristic Programming Project).

Trong những năm 1970-1973, họ đã xây dựng một hệ thống phân tích dữ liệu từ bộ phân tích phổ để từ đó, xây dựng hệ chuyên gia DENDRAL. Khi dự án kết thúc, họ đã tìm ra cách biểu diễn tri thức dưới dạng các luật.

Bắt đầu từ năm 1973, họ nghiên cứu về lĩnh vực liệu pháp kháng sinh (therapy antibiotic). Kết quả nghiên cứu là hệ chuyên gia MYCIN ra đời trong khoảng thời gian 1973-1978. Thế hệ tiếp theo là EMYCIN (Essential MYCIN).

Do có nhiều loại thuốc kháng sinh, kháng vi cũng như có nhiều loại vi trùng với các cách xử lý khác nhau, nên chỉ có các thầy thuốc chuyên gia thuộc lĩnh vực này mới có thể có liệu pháp chữa trị hiệu quả. Mục đích của MYCIN là :

- Là một hệ thống dễ sử dụng.
- Khả năng vận hành đáng tin cậy.
- Chứa đựng nhiều tri thức liên quan đến lĩnh vực kháng sinh, kháng vi.
- Khả năng xử lý các chỉ dẫn chữa trị không đúng hoặc không đầy đủ.
- Khả năng giải thích và chỉ dẫn chữa trị.

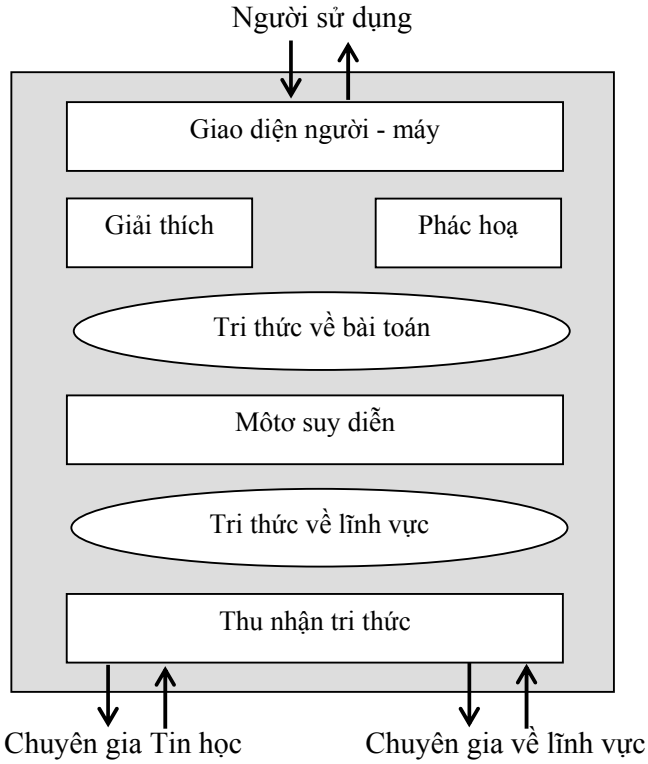
MYCIN là một chương trình tra cứu. MYCIN cung cấp cho các thầy thuốc những ý kiến chữa trị liên quan đến liệu pháp kháng sinh.

MYCIN là một hệ chuyên gia có các đặc tính :

- Lập trình nghiệm suy.
- Tri thức chuyên gia về lĩnh vực kháng sinh.
- Giải thích kiểu tương tác.
- Khả năng phán đoán.

MYCIN có khoảng 500 luật và các sự kiện rất tiêu biểu. Hoạt động của hệ thống như sau :

1. MYCIN yêu cầu thông tin về lâm sàng.
2. Bắt đầu suy luận từ những tri thức hiện có
3. Đưa ra các phán đoán và lời khuyên.
4. Trả lời các câu hỏi liên quan đến suy luận



Hình 4.1. Hoạt động của hệ chuyên gia MYCIN

I.2. Biểu diễn tri thức trong MYCIN

MYCIN biểu diễn các sự kiện bởi một bộ bốn như sau :

Sự kiện	(Context *Ngữ cảnh*	Parameter *Tham biến*	Value *Giá trị*	CF) *Hệ số*
Theo thuật ngữ hướng đối tượng	(Object	Attribute	Value	CF)

a. Ngữ cảnh

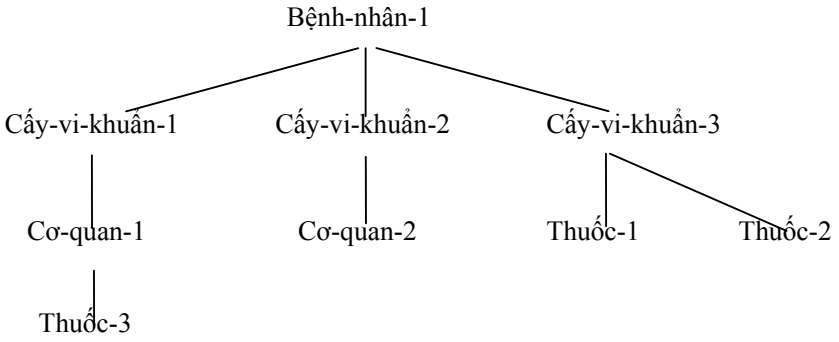
Có tất cả 10 ngữ cảnh dùng để suy luận liên quan đến các liệu pháp (therapy) chữa trị các bệnh nhiễm khuẩn trong MYCIN :

1.	Person	Bệnh nhân
2.	PERS	Những điều trị trước đây
3.	CurCuls	Những xét nghiệm đã làm (cấy khuẩn)
4.	CurDrugs	Những loại thuốc đã uống
5.	CurOrgs	Những cơ quan (bộ phận cơ thể) được cấy khuẩn
6.	OpDrugs	Những loại thuốc đã cho để chữa trị
7.	PosTher	Các liệu pháp có thể
8.	PriorCuls	Những xét nghiệm đã tiến hành trước đây
9.	PriorDrugs	Những loại thuốc đã uống trước đây
10.	PriorOrgs	Những cơ quan bị nhiễm khuẩn trước đây

Có bốn loại câu hỏi trong MYCIN :

1. Có tồn tại những quan trọng không ?
2. Có phải vi khuẩn là nguyên nhân gây ra nhiễm trùng không ?
3. Những loại thuốc nào là hiệu quả ?
4. Những loại thuốc nào là hiệu quả nhất ?

Các câu hỏi trên dẫn đến việc tạo ra một cây ngữ cảnh "động" bao gồm các tình huống ngữ cảnh. Chẳng hạn dưới đây là một cây ngữ cảnh :



Hình 4.2: Cây ngữ cảnh của MYCIN

Một cách tổng quát, MYCIN căn cứ trên những *giả thuyết* (hypothesis) chắc chắn nhất để suy luận và sử dụng các *siêu luật* (meta-rule) để *hội tụ* (focalize) việc tìm kiếm nguyên nhân.

b. Các tham biến

Các tham biến trong MYCIN đều được định kiểu và được gắn nhãn với :

Hệ thống : Y/N, NUMB, ONE_OF, ANY_OF

PROMPT (Dấu nhắc) : câu hỏi yêu cầu người sử dụng gõ vào một giá trị.

LABDATA (Dữ liệu) : lấy từ các phòng xét nghiệm sử dụng giá trị Y/N :
nếu Y, hệ thống yêu cầu đưa giá trị vào, nếu N, hệ thống tự tìm.

LOOKAHEAD (Tìm kiếm) : tạo ra danh sách các luật từ tham số đã cho.

TRANS (Dịch giải) : dịch kết quả ra tiếng Anh.

Có ba loại kiểu giá trị trong MYCIN :

Giá trị đơn (exclusive-single valued) : hệ thống đưa ra nhiều giá trị nhưng chỉ có duy nhất một giá trị đúng. Ví dụ : tên bệnh nhân.

Giá trị bội (non-exclusive-multi valued) : hệ thống đưa ra nhiều giá trị.
Ví dụ : thuốc chữa trị, cảnh báo dị ứng thuốc.

Giá trị nhị phân : Yes/Không.

c. Độ tin cậy (Certain Factor)

Trong MYCIN, mỗi sự kiện đều được gắn một *độ tin cậy* (hay độ chắc chắn), viết tắt CF (Certainty Factor), với quy ước $CF \in [-1, 1]$, và mỗi luật đều được gắn một *năng lực* (force), $CF \in [-1, 1]$.

d. Biểu diễn luật

MYCIN sử dụng kỹ thuật *suy diễn lùi* (backward chaining), điều khiển bởi một *đích* (goal). Các luật của MYCIN có dạng *trải ra* (abduction) như sau :

$$A \wedge B \xrightarrow{CF_R} C$$

Ở mọi thời điểm, MYCIN cần các phép chứng minh. Trong luật trên, để đạt tới đích C, cần chứng minh A và B, để chứng minh A, cần chứng minh..., và cứ thế tiếp tục. Các luật sau đây là hợp lệ :

$A \wedge B \wedge C \rightarrow D$
$A \wedge (B \vee C) \rightarrow D$
$(A \vee B \vee C) \wedge (D \vee E) \rightarrow F$

Nhưng các luật sau đây là không hợp lệ :

$A \vee B \vee C \rightarrow D$
$A \wedge (B \vee (C \wedge D)) \rightarrow E$

MYCIN có các *mẫu luật* (rule template) cho phép hệ thống suy luận trên những luật riêng đã có để tự động tạo ra các luật mới. Cho các giả thiết H_1, H_2 có độ tin cậy CF_1, CF_2 và các luật R có năng lực CF_R tương ứng. Cách MYCIN tổ hợp các giả thiết như sau :

Phép giao $\wedge$:

$$CF(H_1 \wedge H_2) = \min \{ CF(H_1), CF(H_2) \}$$

Phép hợp $\vee$:

$$CF(H_1 \vee H_2) = \max \{ CF(H_1), CF(H_2) \}$$

Các luật :

$$A \wedge B \xrightarrow{CF_R} C \qquad CF_C = CF_R . \min \{ CF_A, CF_B \}$$

$$A \wedge (B \vee C) \xrightarrow{CF_R} D \qquad CF_D = CF_R . \min \{ CF_A, \max \{ CF_B, CF_C \} \}$$

Tổ hợp các giả thiết :

$$\text{Combine}(CF_1, CF_2) = \begin{cases} CF_1 + CF_2 (1 - CF_1) & \text{nếu } CF_1 \geq 0 \text{ và } CF_2 \geq 0 \\ \dfrac{CF_1 + CF_2}{1 - \min\{|CF_1|, |CF_2|\}} & \text{nếu } CF_1 . CF_2 < 0 \\ - \text{Combine}(- CF_1, - CF_2) & \text{nếu } CF_1 \leq 0 \text{ và } CF_2 \leq 0 \end{cases}$$

Ta có tính chất sau :

(H_1, CF_1) và (H_2, CF_2) và $(H_1 = H_2) \rightarrow (H_1, CF_1 = \text{Combine}(CF_1, CF_2))$

$CF(\neg H) = - CF(H)$

I.3. Kỹ thuật suy diễn của MYCIN

a. Thủ tục MONITOR

```
Begin { Monitor }
    OK = True
    While Chưa hết điều kiện And OK Do Begin
        Lấy một điều kiện
        If Các dữ liệu của điều kiện thoả mãn Then Gọi thủ tục FindOut
        If Điều kiện sai Then OK = False
        Else Lấy điều kiện tiếp theo
    End
    If OK Then Đưa ra kết luận
    Else Thông báo luật sai
End { Monitor }
```

b. Thủ tục FINDOUT

```
Begin { FindOut }
    If Có dữ liệu Lab Then Begin
        Yêu cầu người sử dụng trả lời
        If Not Tìm thấy kết quả Then Begin
            Tìm gọi các luật liên quan đến dữ liệu
            Gọi thủ tục Monitor
        End
    End
Else Begin
        Tìm gọi các luật liên quan đến dữ liệu
        Gọi thủ tục Monitor
        If Not Tìm thấy kết quả Then
            Yêu cầu người sử dụng trả lời
        End
    End
End { FindOut }
```

c. Hệ thống giao tiếp của MYCIN

Tại mọi thời điểm, người sử dụng có thể yêu cầu hệ thống trả lời các câu hỏi bằng cách đưa vào các lệnh như sau :

WHY? (tại sao) ˜ hệ thống liệt kê các suy diễn hiện đang tiến hành
HOW? (như thế nào) ˜ hệ thống liệt kê nguồn của một sự kiện

MYCIN có thể hiểu và suy luận trên các luật, các sự kiện và ngữ cảnh viết bằng tiếng Anh quy ước (pseudo-english). Sau đây là một phiên làm việc của MYCIN :

MYCIN is a rule-based system that diagnoses bacterial infections of the blood. By asking questions and backward chaining through a rule base of about 500 rules, Mycin can recognize about 100 causes of bacterial infections, allowing Mycin to recommend effective drug prescriptions. Mycin uses backward chaining because doctors prefer to focus on one hypothesis at a time, whereas a forward chaining system can jump around seemingly at random. Here is a fragment of a dialog between Mycin and a doctor:

>> What is the patients name?
John Doe.
>> Male or Female?
Male.
>> Age?
55.
>> Let's call the most recent positive culture C1. From what site was C1 taken?
From the blood.
...
>> My recommendation is as follows: give gentamycin using a dose of 119 mg (1.7 mg/kg) q8h IV [or IM] for 10 days. Modify dose in renal failure. Also, give clindamycin using a dose of 595 mg (8.5 mg/kg) q6h IV [or IM] for 14 days.

II. Hệ sản xuất OPS5

II.1. Giới thiệu OPS5

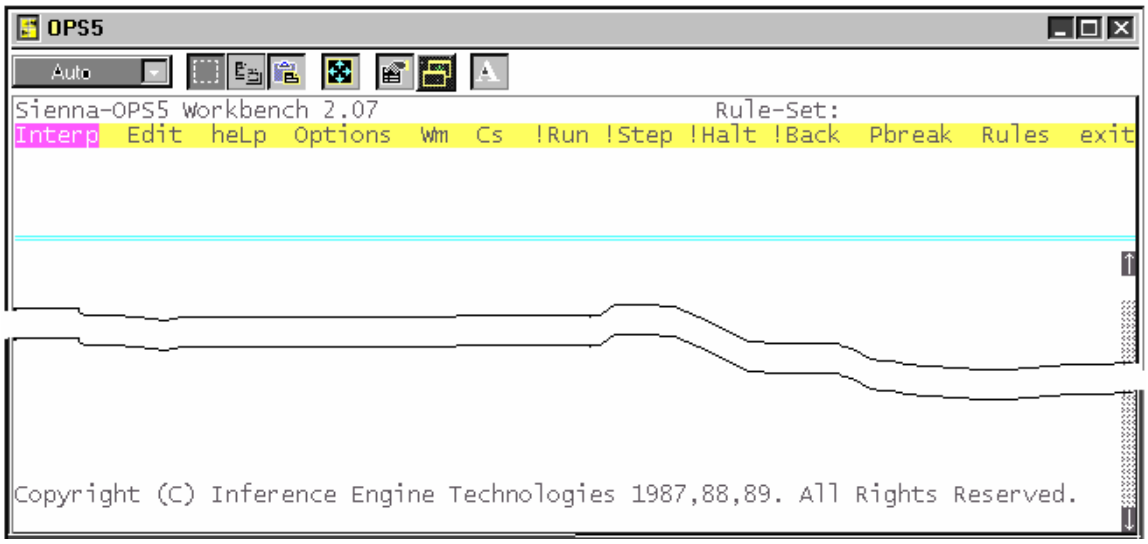
Hệ sản xuất OPS5 (viết tắt từ Official Production System) do Tiến sĩ Charles Forgy đề xuất và được phát triển vào cuối những năm 1970 tại trường Đại học Carnegie-Mellon, Hoa Kỳ. OPS5, còn được gọi là ngôn ngữ lập trình OPS5, là một trong những hệ sản xuất mạnh được sử dụng để xây dựng các hệ chuyên gia. Ví dụ R1/XCON, XSEL là các hệ chuyên gia về cấu hình và buôn bán các thiết bị điện tử, PTRANS quản lý các phân xưởng sản xuất, v.v...

Có thể nói, các mô hình hệ sản xuất được ứng dụng tương đối rộng rãi trong các lĩnh vực trí tuệ nhân tạo, hệ chuyên gia và *tâm lý nhận thức* (cognitive psychology). Sau OPS5, người ta tiếp tục phát triển OPS-83, ART (Automated Reasoning Tool), CLIPS (C Language Intergated Production System)...

OPS5 phiên bản 2.07 chạy trong hệ điều hành MS-DOS. Sau khi khởi động, trên màn hình xuất hiện các dòng giới thiệu về sản phẩm.

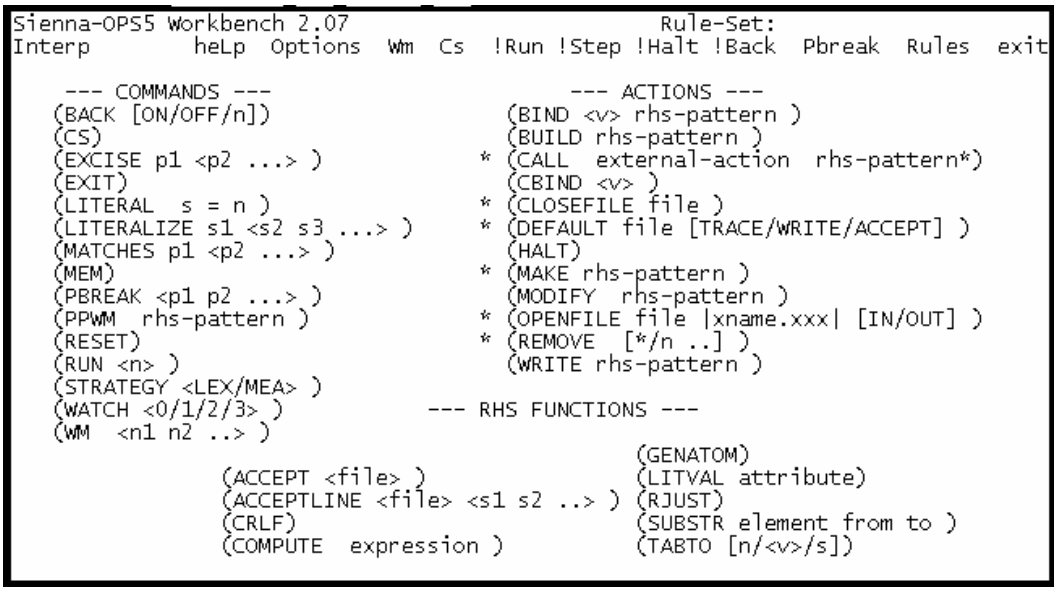
```
Sienna OPS5
WORKBENCH

STUDENT VERSION   2.07

Copyright (C) 1987,88,89 by:
Inference Engine Technologies
All rights reserved.
```

Nhấn một phím bất kỳ, cửa sổ làm việc OPS5 xuất hiện như hình dưới đây :



Hình 4.3. Cửa sổ làm việc của OPS5

Tại mỗi thời điểm, người sử dụng nhấn phím F1 hoặc gọi lệnh Help để gọi hướng dẫn của OPS5.



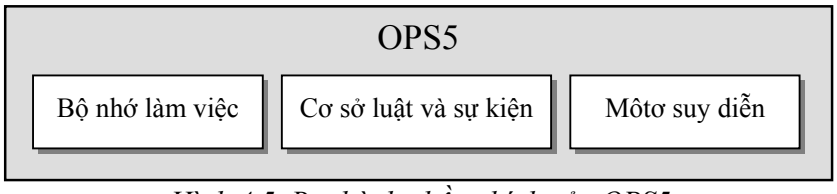
Hình 4.4. Màn hình hướng dẫn của OPS5

Phụ lục cuối giáo trình (tiếng Anh) hướng dẫn sử dụng hệ thống OPS5.

II.2. Các thành phần của OPS5

II.2.1. Các đặc trưng chính của ngôn ngữ

Ngôn ngữ OPS5 có ba thành phần chính là tập hợp các luật, hay được gọi là *cơ sở luật* (rule base), *bộ nhớ làm việc* (working memory, viết tắt WM), và máy suy diễn, được chỉ ra trong hình dưới đây.



Hình 4.5. Ba thành phần chính của OPS5

OPS5 sử dụng các luật chứa các biến lượng tử. Máy suy diễn hoạt động theo kiểu *suy diễn tiến* (forworrd-chaining), *không đơn điệu* (non-monotone), giải quyết xung đột nhờ *các tiền quan hệ* (antecedence relationships) giữa các sự kiện trong tập hợp các sự kiện, còn được gọi là *cơ sở sự kiện* (fact base).

Cơ sở luật của OPS5 sử dụng *dữ liệu định kiểu* (typed data) đã được khai báo trước. Một chương trình OPS5 gồm các thành phần :

1. Một tập hợp các khai báo các cấu trúc dữ liệu có mặt trong chương trình.
2. Định nghĩa *cơ sở luật* (viết tắt rb).
3. Định nghĩa *cơ sở sự kiện ban đầu* (viết tắt fb).

4. Một dãy các lệnh tương tác để diễn giải các luật và áp dụng chúng.

Chú ý rằng người sử dụng có thể thay đổi thứ tự các thành phần trên đây trong những trường hợp đặc biệt sau :

- Ngay sau khi người sử dụng định nghĩa một luật đầu tiên, OPS5 xem rằng các khai báo đã xong và không thể thay đổi nữa.
- Không một lệnh OPS5 nào được thực hiện trước khi ít nhất một luật được định nghĩa ; chăng hạn không một sự kiện nào được tạo ra cho fb trước định nghĩa này.
- Một luật đưa vào sau định nghĩa của fb sẽ chỉ có nghĩa đối với các sự kiện xuất hiện trong fb sau khi tạo ra luật này.

OPS5 sử dụng các luật để truy cập bộ nhớ làm việc WM, thể hiện sự "tinh thông" của hệ thống. Các luật chứa các mẫu so khớp với WM và các hành động để có thể sửa đổi WM. Ngoài ra, các luật còn có khả năng trao đổi với môi trường bên ngoài bằng cách gọi thủ tục và có khả năng tự xây dựng các luật mới.

II.2.2. Kiểu dữ liệu OPS5

Tương tự ngôn ngữ lập trình hàm Lisp, các đối tượng sơ cấp của OPS5 là *nguyên tử số* hay *nguyên tử kí hiệu*. Các ký tự đặc biệt (,) , {, }, <, > và ↑ có thể xuất hiện trong một ký hiệu nếu được đặt trước một dấu gạch đứng □̃

Các luật trong OPS5 thao tác trên các *lớp đối tượng* (class of objects). Mỗi lớp đối tượng được định nghĩa bởi *tên lớp* và một tập hợp các *thuộc tính* (attributes). Tên lớp chỉ định kiểu dữ liệu do người sử dụng tự đặt và được sử dụng sau đó trong môi trường OPS5. Mỗi thuộc tính được đặt tên phân biệt, có giá trị là một nguyên tử số hay ký hiệu. Như sẽ thấy sau này, mỗi sự kiện trong fb sẽ là phần tử của một lớp, có giá trị tương ứng với một thuộc tính nào đó, những thuộc tính khác có thể không được gán giá trị (giá trị mặc nhiên của chúng là nil).

Lệnh khai báo một lớp đối tượng của OPS5 là literalize có cú pháp như sau :

```
( literalize  classname  attribute1  attribute2  ... )
```

Trong đó, classname là tên lớp, attribute1, attribute2, ... là các thuộc tính. Khi bắt đầu sử dụng lệnh literalize, các thuộc tính chưa được gán giá trị ngay.

Ví dụ 1 :

Khi quản lý hồ sơ xin việc, ta khai báo một lớp bằng lệnh literalize như sau :

```
(literalize  hồ-sơ-xin-việc  họ-lót  tên  phái  tuổi  địa-chỉ  nghề-nghiệp)
```

Lệnh này khai báo kiểu dữ liệu hồ-sơ-xin-việc gồm các thuộc tính liên quan đến người xin việc như họ-lót, tên, phái, tuổi, địa-chỉ, nghề-nghiệp. Chú ý trong OPS5, dấu dash (-), như trong hồ-sơ-xin-việc, là để thay cho dấu cách (whitespace) vì các dấu cách như tab, spaces, linefeeds được dùng để phân cách trong một chương trình OPS5.

Ví dụ 2 :

Ta xét một lớp các dụng cụ trong một phân xưởng sản xuất, được đặt tên là tool . Mỗi dụng cụ (thuộc lớp tool) có các thuộc tính : tên dụng cụ (thể hiện chức năng của dụng cụ đó), kích thước, vị trí của dụng cụ trong phân xưởng tại một thời điểm đã cho, trọng lượng, trạng thái sử dụng, v.v...

Lớp được tool khai báo như sau :

```
( literalize tool name size weight position state )
```

Một luật có thể tham chiếu đến một đối tượng của lớp dụng cụ nhờ một *bộ lọc* (filter) như sau :

```
( tool ↑ name englishkey ↑ position rack-A ↑ size 21 )
```

Đây là một danh sách mà phần tử đầu tiên là tên lớp, phần còn lại là một dãy các cặp *thuộc tính* / *giá trị*. Các thuộc tính được phân biệt với giá trị tương ứng bởi một dấu mũi tên ↑ đứng trước (trong phiên bản OPS5 chạy trong MS-DOS, dấu ↑ được thay bằng dấu mũ ^). Thứ tự xuất hiện các thuộc tính không nhất thiết phải tuân theo khai báo lớp trước đó. Trong ví dụ trên, ↑ name englishkey cho biết thuộc tính name có giá trị là englishkey.

Giá trị của một thuộc tính hoặc là một nguyên tử số hay ký hiệu, hoặc là một vectơ các nguyên tử. Tuy nhiên, mỗi lớp chỉ có duy nhất một thuộc tính có giá trị là vectơ, gọi tắt là thuộc tính vectơ. Thuộc tính vectơ này được khai báo bởi lệnh vector-attribute, số thành phần của vectơ thay đổi tùy theo các đối tượng khác nhau của lớp, nhưng phải nhỏ hơn một giới hạn nào đó. Chẳng hạn ta có thể khai báo một thuộc tính vectơ trong lớp dụng cụ trên đây :

```
( vector-attribute position )
```

Ví dụ sau đây cho phép tham chiếu đến thuộc tính vectơ position của lớp tool, một vectơ có 2 thành phần và một vectơ có 4 thành phần :

```
( tool ↑ name pipekey ↑ position cupboard hook-45 )

( tool ↑ name crowbar ↑ position settled localcoordinate 124 66 )
```

Người sử dụng có thể xây dựng mối quan hệ giữa các lớp đã cho : giá trị một thuộc tính của một lớp này có thể là một đối tượng, hay một vectơ các đối tượng, của một lớp khác.

Ví dụ :

```
( literalize rack name content )

( vector-attribute content )
```

Lớp rack có thể có thuộc tính content là một vectơ gồm các dụng cụ như trong sự kiện sau :

```
( rack ↑ name key-rack ↑ content crowbar pipekey englishkey1 engliskey2 )
```

OPS5 còn một lệnh khai báo khác là external dùng để khai báo các hàm bên ngoài dùng cho ứng dụng (có thể được viết trong một ngôn ngữ hỗ trợ cho OPS5 như MACLISP, FRANZLISP hay BLISS, tùy theo phiên bản cài đặt của OPS5).

Chú ý rằng, so với OPS-83 (phiên bản phát triển từ OPS5), OPS5 rất yếu về định kiểu. OPS5 không kiểm tra kiểu đối tượng lúc khai báo ban đầu. OPS5 không đưa ra các thông báo lỗi khi diễn dịch chương trình OPS5 mà gặp các sai sót về kiểu. Chẳng hạn việc thực hiện một phép tính số học trên một thuộc tính kiểu ký hiệu trả về một kết quả sai mà không gây ra lỗi.

Sai sót thường hay xảy ra, kể cả đối với những người đã sử dụng thành thạo OPS5, họ có thể tham chiếu đến hay tạo ra các sự kiện mà không thuộc vào một lớp đã khai báo nào. Một thuộc tính vectơ không được khai báo đúng, có thể được OPS5 chấp nhận nhưng thường gây ra sai sót khi thực hiện. Việc tham chiếu đến một thuộc tính chưa được khai báo trong bất kỳ một lớp nào cũng gây ra một thông báo lỗi.

II.2.3. Cơ sở luật (rb)

Trong các hệ thống dựa trên luật, các luật làm nền tảng cho mọi hoạt động của hệ thống, giúp người phát triển hệ thống hiểu biết một cách tường tận về cách làm việc của hệ thống.

Các luật trong OPS5 có cấu trúc hai thành phần IF-THEN. Thành phần IF là *phần bên trái luật* (LHS), có nhiệm vụ so khớp các mẫu với các sự kiện trong WM để quyết định luật đó có được tuân thủ hay được *chọn ra* (fired) hay không. Thành phần THEN là *phần bên phải luật* (RHS), quyết định mọi hoạt động của hệ thống : trao đổi với bộ nhớ làm việc WM, vào-ra thông tin, xây dựng các luật mới và gọi thực hiện các thủ tục viết trên một ngôn ngữ khác.

Mỗi luật OPS5 được định nghĩa bởi lệnh **p** có cú pháp như sau :

```
( p  rule-name
     left-member → right-member )
```

trong đó :

rule-name	tên luật,
left-member	điều kiện, còn được gọi là các *mẫu so khớp* (patterns) để máy suy diễn tiến hành so khớp (match),
right-member	hành động (actions) được triển khai khi điều kiện thỏa mãn,
dấu →	ngăn cách phần bên trái và phần bên phải luật.

Ví dụ, để chẩn đoán một người mắc bệnh cảm cúm, ta viết luật sau :

```
(p    camcum
      ( benh      sot     sotcao
                  mui     khorat
                  dau     daudau
                  non     buonnon )
      →
      ( write(crlf) | Chan doan : Ban da bi benh cam cum | )
      ( write(crlf) | Xu ly : Ban nen cat thuoc theo bai thuoc sau :| )
      ( write(crlf) | Tia to : 10g ; Huong nhu : 10g ; Vo quyt : 10g; | )
      ( write(crlf) | Gung kho : 0,4g ; Cam thao dat :  10g. | )
      ( write(crlf) | Sac voi 400ml nuoc con lai 200ml, uong luc con am | ))
```

Luật camcum trên được chọn ra khi phần bên trái luật được so khớp thỏa mãn với các thành phần trong bộ nhớ làm việc WM, tức là có tình trạng sotcao, khorat, daudau, binon.Khi đó, phần bên phải luật sẽ được thực hiện : in ra bệnh mà bệnh nhân mắc phải và bài thuốc chữa trị.

a. Thành phần bên trái luật : left-member

Là một danh sách mô tả bộ lọc trên các sự kiện của cơ sở sự kiện fb, gồm :

- Tên một lớp đối tượng là một nguyên tử nằm đầu danh sách.
- Các thuộc tính mà mỗi thuộc tính là một ký hiệu được bắt đầu bởi dấu ↑.
- Các biến, mỗi là một ký hiệu nằm giữa các cặp < >.
- Các quan hệ đại số giữa các thuộc tính :
  = bằng nhau,
  <> khác nhau,
  và các phép tuyển or, hay phép hội and của những quan hệ này.

Ví dụ, khai báo sau đây :

```
( tool ↑ name <tool-name> ↑ position settled )
```

cho phép lọc tất cả sự kiện của lớp tool mà vị trí position (thành phần đầu tiên của thuộc tính vectơ này) có giá trị là settled. Biến <tool-name> được gán giá trị cho thuộc tính name. Khai báo sau đây :

```
( tool ↑ name englishkey ↑ size >= 12 ↑ position <place> )
```

lọc các sự kiện của lớp tool, có tên englishkey, có kích thước nhỏ hơn hoặc bằng 12. Biến <place> được gán giá trị là thành phần đầu tiên của thuộc tính position.

Khai báo :

```
( tool ↑ name << eyekey pipekey flat key >>  ↑ size { <= 9  >=13 } )
```

lọc tất cả công cụ có tên eyekey, hay pipekey, hay flat key (phép tuyển), có kích thước nằm giữa 9 và 13 (phép hội).

Một phép tuyển chỉ có thể tác động lên các hằng số, hoặc hằng ký hiệu, trong khi đó, một phép hội lại có thể tác động lên biến, chẳng hạn khai báo :

(tool ↑ size { <dim> >= 9 <= <weight> })

cho phép gán cho <dim> giá trị của ↑ size và thõa mãn điều kiện nằm giữa 9 và giá trị của biến <weight>. Phép lọc này là hợp lệ nếu biến <weight> có mặt trước quan hệ này trong thành phần bên trái luật.

OPS quan niệm một điều kiện là *chắc chắn* (positive) : điều kiện đó là *hợp lệ* (valid) nếu tồn tại một sự kiện trong fb được thừa nhận qua phép lọc. Người ta có thể khai báo một điều *phủ định* (negative) bằng cách đặt một dấu ~ trước danh sách, chẳng hạn :

- (crowbar ↑ hold <object>)

là một điều kiện hợp lệ nếu không tồn tại một sự kiện nào của lớp crowbar mà thuộc tính giữ một giá trị khác nil.

Người ta có thể kết hợp một tên với một điều kiện chắc chắn : đó là một biến cho phép tham khảo đến thành phần bên phải luật một sự kiện được lọc bởi điều kiện này. Cú pháp để kết hợp một tên với một điều kiện chắc chắn như sau :

{ <key1> (tool ↑ name englishkey ↑ position settled) }

hay :

{ (tool ↑ name englishkey ↑ position settled) <key1> }

Tóm lại, những hạn chế chính trên thành phần bên trái luật như sau

- Tên biến xuất hiện đầu tiên (từ trái qua phải) không thể đặt sau bất kỳ phép toán nào.
- Những phép toán hợp lệ là : =, <> (khác nhau), và so sánh nguyên tử số. OPS5 không cho phép thực hiện các phép toán đại số hay tính giá trị một hàm ngoại vi trong thành phần bên trái.
- Một biến có thể nhận giá trị là một sự kiện được lọc bởi một điều kiện, hoặc một giá trị của một thuộc tính vô hướng (scalar attribute) hay chỉ một thành phần của một thuộc tính vectơ.
- Điều kiện đầu tiên của thành phần bên trái không thể phủ định (-).

b. Thành phần bên phải luật right-member

Phần bên trái không trực tiếp làm thay đổi giá trị trong bộ nhớ làm việc WM, mà chỉ khi các thành phần giá trị của nó đã được so khớp phù hợp với các sự kiện tồn tại trong WM, các thành phần của phần bên phải sẽ được thực hiện.

Phần bên phải luật có thể khởi tạo, xóa bỏ hay sửa đổi các giá trị dữ liệu trong WM. OPS5 có tất cả 15 hàm mô tả sự hoạt động của phần bên phải luật.

Người ta phân biệt 5 kiểu hàm như sau :

1. *Các hàm liên quan đến bộ nhớ làm việc WM, làm thay đổi cơ sở sự kiện fb :*

make	Khởi tạo các giá trị mới vào WM.
modify	Sửa đổi các giá trị mà tồn tại trong WM.
remove	Xóa bỏ các giá trị và các thành phần trong WM.

2. *Các hàm thao tác vào-ra và truy cập tệp :*

write	Nhập dữ liệu là một chuỗi ký tự và/hoặc một biến.
accept	Nhập vào một "từ" dữ liệu là một số hoặc là một ký hiệu.
acceptline	Nhập vào một dòng dữ liệu.

default	Tạo đường dẫn vào-ra (hầu hết các shell OPS5 tự động mặc định thiết bị vào-ra).
Openline	Mở và gán một tên cho tệp dữ liệu để tham chiếu sau đó.
Closefile	Đóng một tệp đã mở.

3. *Các hàm tính toán số học, liên kết biến - giá trị và gọi hàm ngoại vi :*

compute	Thực hiện các phép toán số học.
genatom	Tạo một phần tử hay một ký hiệu mới.
litval	Trả về giá trị của một thuộc tính trong WM.
bind	Gán giá trị cho biến.
cbind	Gán một giá trị trong WM cho một biến.
substr	Trả về một chuỗi (xâu) con.
call	Gọi một chương trình con từ bên ngoài vào shell OPS5.

4. *Điều khiển mô tơ suy diễn :*

halt	Dừng máy suy diễn và dừng chương trình.

5. *Tạo các luật mới :*

build	Tạo một luật mới trong chương trình.

II.2.4. Cơ sở sự kiện (fb)

Mỗi phần tử của fb là một vec tơ gồm 127 thành phần, mỗi thành phần chỉ có thể nhận một giá trị nguyên tử (số hoặc ký hiệu). Thành phần đầu tiên nhất thiết phải là tên của một lớp đối tượng.

Trong giai đoạn khai báo, OPS5 gán cho mỗi thuộc tính một giá trị là một số nguyên từ 2..127 với vai trò là *chỉ số* (index) để được tham chiếu đến (bởi lệnh vĩ mô $litbind) mà không thay đổi được. Giá trị này có tính *toàn cục* (global) và không thuộc về một lớp nào. Khi một thuộc tính được khai báo trong nhiều lớp sẽ tương ứng với một chỉ số duy nhất chung cho tất cả các lớp này. Thuật toán gán phải thõa mãn các điều kiện sau :

- Một thuộc tính vectơ có chỉ số lớn nhất trong tất cả các lớp chứa nó : giả sử j là giá trị thuộc tính vectơ, thì các thành phần j , j+1, ..., 127 sẽ tương ứng với các giá trị của thuộc tính vectơ.
- Cùng một lớp, các thuộc tính được gán các giá trị phân biệt.

Các thành phần của một sự kiện mặc nhiên bằng nil trừ thành phần đầu tiên là tên của lớp, là những thuộc tính một cách tường minh được gán giá trị khi tạo mới hay thay đổi một sự kiện. Giá trị của một thuộc tính nhận được bằng cách *đánh chỉ số* (indexation) lên vectơ tương ứng với sự kiện. Ký tự ↑ là một ký hiệu đánh chỉ số. Sau ↑ là một số nguyên trong khoảng từ 2 đến 127, hay là một tên thuộc tính kết hợp với nó một cách trực tiếp. Nếu ↑ vắng mặt , OPS5 xem như chỉ số tương ứng này được tăng lên một (increment) để truy cập đến thành phần tiếp theo. Chẳng hạn ba điều kiện sau đây :

(tool ↑ name <tool-name> ↑ size <tool-size>)

(tool ↑ 3 <tool-name> ↑ 4 <tool-size>)

(tool ↑ name <tool-name> <tool-size>)

là tương đương, nếu các thuộc tính name và size được gán các giá trị nguyên là 3 và 4 tương ứng. Nếu thành phần thứ hai không được gán (nil) , thì ta nhận được điều kiện tương đương khác như sau :

(tool nil <tool-name> <tool-size>)

Khi cài đặt một hệ chuyên gia, người ta thường gặp những bất lợi liên quan đến tính hiệu quả của một cơ sở sự kiện fb, chẳng hạn :

- Các sự kiện là một *danh sách phẳng* (plate list) không có cấu trúc.
- Các biến chỉ có thể được gán giá trị là nguyên tử (hay là các sự kiện đầy đủ nếu đó là tên của một điều kiện). Người ta phải sử dụng nhiều biến để thu nhận nhiều giá trị gán cho một thuộc tính vectơ.
- Tính toàn cục của các thuộc tính : sự nhầm lẫn các thuộc tính giữa các lớp là một sai sót nguy hiểm vì OPS5 không cảnh báo gì. Để tránh sai sót, các thuộc tính của các lớp cần được đặt tên phân biệt. Các tên thuộc tính này sẽ giúp tiếp cận đến các thành phần của các sự kiện một cách tường minh. Chú ý nhớ sử dụng dấu ↑ để tên thuộc tính (đứng ngay sau dấu này) lấy giá trị là phần tử tiếp theo tên thuộc tính.

Cơ sở sự kiện được khởi động bởi một dãy các lệnh make, chẳng hạn:

(make tool ↑ name englishkey ↑ position rack-A ↑ size 21)

(make tool ↑ name pipekey ↑ position cupboard hook- 45)

(make tool ↑ name crowbars ↑ position settled localcoordinate 124 66)

Chú ý rằng các thuộc tính và các thành phần thuộc tính vectơ của một lớp không phải đã được định nghĩa hết cho mỗi sự kiện.

Mỗi sự kiện của fb được lưu giữ thời điểm tạo ra nhờ một bộ đếm khởi động bắt đầu từ giá trị 1 và tăng lên 1 mỗi lần một sự kiện mới được tạo ra, hay có sự thay đổi sự kiện. Như vậy, mỗi sự kiện được đặt tương ứng với một số thứ tự và người sử dụng có thể chỉ định nó. Chẳng hạn, lệnh :

(remove 1 3)

loại bỏ các sự kiện 1 và 3 của fb. Lệnh (wm) cho phép xem nội dung của fb.

II.2.5. Bộ nhớ làm việc

Bộ nhớ làm việc WM là nơi lưu trữ các giá trị sự kiện. Mỗi sự kiện trong WM được gọi là một *phần tử nhớ vận hành* (working memory element). Bộ nhớ làm việc được truy cập và hoạt động với những khả năng như sau :

- Người sử dụng có thể lưu trữ các phần tử nhớ vận hành trong suốt quá trình soạn thảo chương trình OPS5.
- Người sử dụng có thể thêm, xóa bỏ hoặc sửa đổi các sự kiện của WM trong khi chương trình vẫn đang thực hiện.
- Máy suy diễn tiến hành so khớp các thành phần bên trái luật đang xét với các phần tử nhớ vận hành.
- Trong quá trình *gỡ rối* (debugging), người sử dụng có thể xem nội dung của WM nhờ hệ thống tương tác của OPS5 (interactive shell), lệnh (wm).

a. Cấu trúc bộ nhớ làm việc

Bộ nhớ làm việc của OPS5 có cấu trúc tương tự *bản ghi* (record) trong các ngôn ngữ lập trình quen thuộc như Pascal, C, ... Tuy nhiên, cấu trúc của OPS5 đơn giản hơn do mỗi trường có thể chứa bất kỳ kiểu dữ liệu nào được xác định tại thời điểm gán giá trị. Trong khi đó, Pascal hay C bị hạn chế bởi việc gán kiểu dữ liệu ban đầu. Ngôn ngữ OPS5 yêu cầu các phần tử nhớ vận hành phải được khai báo trong phần đầu tiên của chương trình nhờ lệnh literalize.

b. Khởi tạo bộ nhớ làm việc

Để lập trình OPS5, người sử dụng cần khởi tạo bộ nhớ làm việc WM gồm các lớp dữ liệu và các sự kiện. Việc tạo ra WM ở giai đoạn này được gọi là *bước khởi tạo* (literalization section) cho một chương trình OPS5. Sau đó, sử dụng lệnh make để tạo ra các phần tử nhớ vận hành.

Ví dụ, để khởi tạo WM với lớp dữ liệu hồ-sơ-xin-việc vừa được khai báo trên đây, ta khởi tạo các giá trị thuộc tính như sau :

```
( make  hồ-sơ-xin-việc    ↑ họ-lót   hoàng-thị
                          ↑ tên   ngọ
                          ↑ phái  nữ
                          ↑ tuổi   21
                          ↑ địa-chỉ  18-ong-ich-khiêm-da-nang
                          ↑ nghề-nghiệp   giáo-viên )
```

Mỗi khi WM đã được khởi tạo, dùng lệnh :

```
( wm )
```

để xem nội dung của WM. Để xoá nội dung của WM, gọi lệnh :

```
( remove )
```

Do OPS5 làm việc ở chế độ diễn dịch nên mỗi khi chương trình được tải vào shell OPS5, nội dung trước đó của WM bị xóa sạch sẽ để được khởi tạo lại, ngay cả khi chạy lại cùng một chương trình. Khả năng này làm các chương trình OPS5 chạy nhanh hơn vì mỗi chương trình chỉ làm việc với một tập các sự kiện đã được định nghĩa riêng.

OPS5 quan niệm rằng nếu như nội dung WM không được xóa sạch thì tất cả các sự kiện vẫn còn tồn tại mỗi khi chạy chương trình. Khi đó, kết quả thu được có thể sai, hoặc xuất hiện lỗi chạy chương trình.

Tuy nhiên, người ta không khởi tạo WM ngay bên ngoài shell OPS5 mà thường kết hợp khởi tạo WM bằng lệnh make ngay trong các luật của chương trình vì điều này sẽ tạo ra một lượng lớn các sự kiện đầy đủ phong phú chứ không hạn chế gõ từng lệnh, khởi tạo từng sự kiện một ở ngoài shell OPS5**.**

II.3. Làm việc với OPS5

II.3.1. Hoạt động của máy suy diễn

Máy suy diễn OPS5 hoạt động như sau. Đầu tiên, các sự kiện về bài toán do người sử dụng cung cấp được thu nhận để đặt vào bộ nhớ làm việc WM. Sau đó, máy suy diễn tự động so khớp các thành phần bên trái luật với các sự kiện lưu trữ trong WM để quyết định luật nào sẽ sử dụng. Sau khi đã xác định được luật, máy sẽ thực hiện các hành động trong phần bên phải luật đó. Máy suy diễn sẽ tiếp tục vòng lặp của mình với các luật kế tiếp cho đến khi không còn luật nào trong chương trình nữa. Nói cách khác, khi không còn gì để so khớp nữa, máy suy diễn sẽ ngừng và chương trình kết thúc tại đó.

Sơ đồ hoạt động của máy suy diễn như sau :

```
Begin { Inference Engine }
    Nhập dữ liệu vào bộ nhớ làm việc WM
    While Chưa hết luật Do Begin
        Lấy một luật tiếp theo
        So khớp các thành phần bên trái luật với các phần tử của WM
        If  So khớp thành công
        Then  Thực hiện các lệnh bên phải của luật đó
        EndIf
    End
```

End { Monitor }

II.3.2. Tập xung đột và cách giải quyết xung đột

Thông thường, một chương trình OPS5 chỉ cho phép một luật được sử dụng trong suốt một vòng lặp của máy suy diễn. Nhưng vấn đề đặt ra là không phải lúc nào máy suy diễn cũng chỉ tìm thấy một luật duy nhất. Trong một số trường hợp, máy suy diễn có thể tìm thấy nhiều luật mà có các thành phần bên trái so khớp phù hợp với các sự kiện trong WM. Khi gặp tình huống như vậy, máy suy diễn phải lựa chọn một luật phù hợp nhất để sử dụng. Có thể tóm tắt quá trình hoạt động của máy suy diễn qua bốn bước lặp sau đây :

1. Máy suy diễn lọc toàn bộ tập hợp các xung đột.
2. *Nếu* :
   - tập hợp các xung đột là rỗng, hoặc nếu :
   - một số lượng tối đa các chu kỳ đã được thực hiện, hoặc nếu :
   - luật sử dụng trước đó là một điểm dừng hoặc chứa lệnh dừng halt nằm bên phải luật,

   Thì dừng máy suy diễn.
3. *Nếu không*, chọn một luật nào đó trong tập hợp các xung đột tuỳ theo chiến lược giải quyết xung đột đang áp dụng.
4. Sử dụng luật đã chọn : thực hiện các lệnh nằm bên phải luật.

Ở bước 3, một nhóm các luật có các thành phần bên trái được máy suy diễn so khớp phù hợp với WM được gọi là tập xung đột (conflict set). Sau khi tập xung đột được tạo ra, bước tiếp theo, máy suy diễn cần chọn một luật để thực hiện bằng cách sử dụng một trong hai *chiến lược giải quyết xung đột* (conflict resolution strategy) là LEX (LEXicographical ordering) và MEA (Means-Ends Analysis).

Các chiến lược giải quyết xung đột LEX và MEA sử dụng một *thẻ thời gian* (time tag, đôi khi còn được gọi là *thẻ tiếp cận*, access tag) là một con số gắn cho một phần tử của WM. Số này có nghĩa khi một phần tử của WM được tạo ra hay vừa mới được sửa đổi.

a. Chiến lược giải quyết xung đột LEX

LEX là chiến lược giải quyết xung đột mặc định (default) của môi trường OPS5. Mỗi lần cần gọi LEX, sử dụng lệnh :

(strategy lex)

bằng cách hoặc gõ trực tiếp tại dấu nhắc lệnh, hoặc đặt ở phần bên phải một luật. Chiến lược LEX giải quyết xung đột qua bốn bước như sau :

1. Loại bỏ (discard, hay refraction) các luật trong tập xung đột đã được chọn ra trên cùng một dữ liệu. Nếu chỉ còn lại một luật trong tập xung đột thì lấy luật đó để thực hiện, nếu không thì thực hiện bước tiếp theo.
2. Sắp xếp các thành phần bên trái theo thứ tự giảm dần của các thẻ thời gian và so sánh tất cả các thẻ thời gian đó. Sau đó so sánh các thành phần bên trái của các luật trong tập xung đột để tìm ra luật có thành phần vừa mới làm biến đổi (altered) WM. Nếu tìm được luật có các điều kiện khớp với các phần tử của WM có thẻ thời gian mới nhất thì chọn nó để thực hiện. Nếu có ràng buộc xuất hiện thì loại trừ tất cả các luật không có ràng buộc từ tập xung đột và tiếp tục bước tiếp theo.
3. Dựa trên các thuộc tính vàhoặc các biến, so sánh các điều kiện của các luật còn lại theo nguyên tắc luật nào có nhiều mẫu so khớp hơn thì luật đó quan trọng hơn. Nếu chỉ còn lại một luật trong tập xung đột thì chọn nó. Nếu có ràng buộc xuất hiện thì loại trừ tất cả các luật không có ràng buộc trong tập xung đột và tiếp tục bước tiếp theo.

4. Nếu sau khi thực hiện các bước trên mà vẫn còn nhiều luật trong tập xung đột thì chọn ngẫu nhiên (selected randomly) một luật và thực hiện nó.

b. Chiến lược giải quyết xung đột MEA

MEA không phải là chiến lược giải quyết xung đột mặc định của OPS5. Khi cần sử dụng MEA, sử dụng lệnh :

(strategy mea)

MEA giải quyết xung đột qua năm bước, trong đó bước 1, bước 4 và bước 5 tương tự chiến lược LEX :

1. Loại bỏ các luật trong tập xung đột đã được chọn ra trên cùng một dữ liệu. Nếu chỉ còn lại một luật trong tập xung đột thì lấy luật đó để thực hiện, nếu không thì thực hiện bước tiếp theo.
2. Chỉ so sánh điều kiện đầu tiên của các luật trong tập xung đột để tìm xem luật nào có dòng điều kiện đầu tiên vừa làm thay đổi WM. Việc so sánh được tiến hành cho tất cả các thẻ thời gian của các phần tử trong WM đã so khớp với điều kiện đầu tiên. Nếu tìm được luật có điều kiện đầu tiên khớp với các phần tử của WM có thẻ thời gian mới nhất thì chọn nó để thực hiện. Nếu có ràng buộc xuất hiện thì loại trừ tất cả các luật không có ràng buộc từ tập xung đột và tiếp tục bước tiếp theo.
3. So sánh từ điều kiện thứ hai trở đi của các luật trong tập xung đột để tìm xem luật nào làm thay đổi WM có thẻ thời gian mới nhất. Việc so sánh dựa theo thứ tự giảm dần của các thẻ thời gian đối với điều kiện thứ hai. Luật nào có thẻ thời gian mới nhất được chọn ra. Nếu có ràng buộc xuất hiện thì loại trừ tất cả các luật không có ràng buộc từ tập xung đột và thực hiện bước tiếp theo.
4. Dựa trên các thuộc tính vàhoặc các biến, so sánh các điều kiện của các luật còn lại theo nguyên tắc luật nào có nhiều mẫu so khớp hơn thì luật đó quan trọng hơn. Nếu chỉ còn lại một luật trong tập xung đột thì chọn nó. Nếu có ràng buộc xuất hiện thì loại trừ tất cả các luật không có ràng buộc trong tập xung đột và tiếp tục bước tiếp theo.
5. Nếu sau khi thực hiện các bước trên mà vẫn còn nhiều luật trong tập xung đột thì chọn ngẫu nhiên một luật và thực hiện nó.

c. Lựa chọn chiến lược giải quyết xung đột

Chiến lược LEX đơn giản hơn chiến lược MEA vì LEX không chú trọng đến thứ tự các điều kiện trong phần trái luật, trong khi đó, chiến lược MEA cho phép nhóm các luật thành các *đơn thể* (modules). Các đơn thể giúp giảm bớt tính phức tạp của một chương trình và do đó được sử dụng để lập trình OPS5. Chiến lược MEA cho phép điều khiển bên trong nhờ cách thao tác gián tiếp trên luật.

Chiến lược MEA cũng cho phép người lập trình so khớp một nhóm các luật với các phần tử WM tại những thời điểm xác định. Nói cách khác thì một chương trình có thể kích hoạt một nhóm các luật thông qua việc truy cập các phần tử WME, tạo ra thẻ thời gian của thành phần được truy cập mới nhất. Chiến lược MEA giúp cho việc làm giảm bớt dung lượng của tập xung đột và thời gian cần thiết để thực hiện luật.

Thông thường khi xây dựng một chương trình OPS5, người ta hay sử dụng chiến lược LEX mặc định để giải quyết xung đột. Tuy nhiên, đối với các chương trình có tập xung đột lớn, kỹ thuật chương trình phức tạp thì người ta hay sử dụng chiến lược MEA để làm cho chương trình chạy nhanh hơn, giải quyết các vấn đề có tính quy mô hơn.

II.3.3. Lệnh và phép toán của OPS5

a. Một số lệnh OPS5

Sau khi khai báo kiểu dữ liệu, định nghĩa các cơ sở luật rb và cơ sở sự kiện fb, người sử dụng có thể viết các dòng lệnh chương trình để khai thác các luật trong chế độ tương tác. Sau đây là một số lệnh thông dụng :

(run k) khởi động máy suy diễn hoạt động tối đa k chu kỳ,
(run 1) cho phép chạy từng bước,
(run) chạy không hạn chế số chu kỳ.

(back k) cho phép tìm lại trạng thái của cơ sở sự kiện fb trước k chu kỳ so với chu kỳ hiện hành, k ≤ 32 và một trong k luật trước đó đã được chọn ra không gọi đến các hàm từ bên ngoài.

(watch i) nếu i = 1, liệt kê kết quả các luật đã được chọn ra,
nếu i = 2, liệt kê tất cả các thay đổi trong cơ sở sự kiện fb.

(break r1 ... rN) tạo điểm dừng tại các luật r1, ..., rN : sau khi một trong các luật này được chọn ra, máy suy diễn dừng lại cho đến khi có sự can thiệp, bởi (run k) chẳng hạn. Lệnh này có tác dụng tương tự lệnh halt đặt bên trong vế phải luật.

(excise r1 ... rN) loại bỏ các luật r1, ..., rN khỏi cơ sở luật rb.

(pm), (wm), (cs) xem nội dung của fb, rb, wm và tập xung đột tương ứng.

b. Các phép toán của OPS5

OPS5 thực hiện các phép toán số học như cộng (+), trừ (-), nhân (*), chia (//) và lấy phần dư (\\) nhờ lệnh compute đặt ở phần bên phải luật. Lệnh compute cần có tối thiểu hai đối số và một phép toán. Các đối số có thể là các số (các hằng) hoặc các biến. Kết qủa của các phép tính toán số học được lưu trữ trong WM.

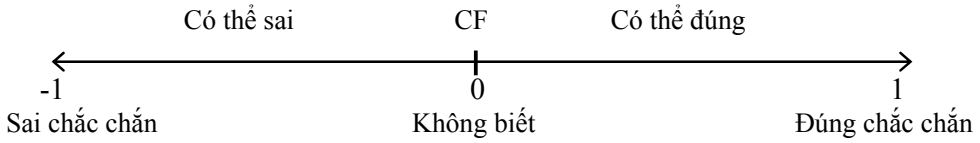
Ví dụ, điểm trung bình cho ba môn thi *Toán*, *Lý* và *Hóa* của một học sinh có tên là *Minh* được tính toán nhờ các câu lệnh OPS5 như sau :

```
( literalize ketquathi tenhs lop toan ly hoa diemtb )
( make ketquathi   ↑ tenhs minh
                   ↑ lop 10d
                   ↑ toan 8
                   ↑ ly 7 ↑
                   ↑ hoa 8
                   ↑ diemtb ( compute (( toan+ly+hoa )//3 ))))
```

Giá trị của phép toán được tính toán và được lưu trữ trong WM trước khi thực hiện lệnh make. Những lập trình viên OPS5 nhiều kinh nghiệm có thể thực hiện các phép toán phức tạp bằng cách gọi các thủ tục từ bên ngoài OPS5, chẳng hạn gọi các hàm viết trong C. Kết quả trả về có thể là các tham đối của chương trình hoặc các tệp dữ liệu kết quả.

c. Yếu tố chắc chắn

Hầu hết các hệ chuyên gia cho lời khuyên hoặc giải quyết một vấn đề nào đó với kết quả có độ chính xác được đo bởi *độ chắc chắn* CF. Giá trị của CF thay đổi từ -1, ứng với "sai chắc chắn", đến +1, ứng với "đúng chắc chắn". Giá trị CF=0 cho biết "không biết", giá trị âm thể hiện độ không tin cậy vào giả thuyết trong khi đó giá trị dương thể hiện tính chắc chắn sự tin cậy vào giả thuyết.



Hình 4.5. Phạm vi giá trị của nhân tố chắn chắn CF

Trong OPS5, độ tố chắc chắn không phải là xác suất, mà là các độ đo không hình thức về sự tin tưởng hay độ tin cậy vào một vấn đề. Để thể hiện độ tin cậy này, người thiết kế chương trình thường thêm vào một sự kiện một giá trị CF phù hợp. Ví dụ, hai sự kiện *"hôm nay có khả năng trời sẽ mưa"* và *"hôm nay trời mưa, CF=0.6"* có ý nghĩa như nhau. Giá trị CF=0.6 được hiểu là *"có khả năng"*.

Người ta cũng dùng hệ số CF trong luật để thể hiện quan hệ không chắc chắn giữa điều kiện trong phần bên trái luật và hành động trong phần bên phải luật.

Ví dụ : nếu có mây đen
thì trời sẽ mưa CF=0.8

được hiểu là "*nếu có mây đen thì gần như chắn chắn trời sẽ mưa*".

II.4. Đánh giá và phát triển của OPS5

II.4.1. Đánh giá

OPS5 là một ngôn ngữ xây dựng hệ chuyên gia khá tiêu biểu. OPS5 mang đầy đủ những ưu điểm của một ngôn ngữ dựa trên luật khi thiết kế một chương trình hệ chuyên gia :

- Môi trường thiết kế cho các ngôn ngữ dựa trên luật không phức tạp do cấu trúc luật IF-THEN đơn giản, dễ hiểu và dễ sử dụng, dễ quản lý các sự kiện được lưu trữ trong bộ nhớ làm việc.
- Mỗi luật biểu diễn độc lập một lượng kiến thức riêng, thuận tiện trong việc kiểm tra, cập nhật và sử dụng.
- Sự tách biệt giữa các luật cho phép bổ sung các luật mới, tạo điều kiện mở rộng dễ dàng các tri thức trong hệ thống tri thức đang xét .
- Máy suy diễn được nhúng và ẩn trong hệ thống, người sử dụng không thể làm thay đổi máy suy diễn.

Tuy nhiên, bên cạnh những ưu điểm trên đây, các ngôn ngữ dựa trên luật cũng có những hạn chế như :

- Các luật thường theo một hướng duy nhất : theo hướng suy luận tiến hoặc theo hướng suy luận lùi.
- Hệ thống vận hành nặng nề do phải so khớp mẫu và giải quyết xung đột. Cùng giải quyết một vấn đề, OPS5 thực hiện chậm hơn so với ngôn ngữ C hoặc ngôn ngữ Pascal.
- Hệ chuyên gia càng lớn càng nhanh chóng trở nên phức tạp : với một hệ thống có nhiều luật, sự gia tăng điều khiển dẫn đến chương trình dễ bị hư hỏng và khó theo dõi.

II.4.2. Phát triển của ngôn ngữ OPS5

OPS5 đã được sử dụng thành công để xây dựng nhiều hệ chuyên gia. Tuy nhiên, nhược điểm của OPS5 là không có tính kết hợp giữa chương trình dựa trên luật với một chương trình theo dạng mệnh lệnh. Để khắc phục, Tiến sĩ Charles Forgy, tác giả của OPS5, đã phát triển OPS5 thành một ngôn ngữ mới là OPS8.

Khác với trình diễn dịch OPS5, OPS83 là một trình biên dịch (compiler), cho phép giao tiếp với các ngôn ngữ khác một cách dễ dàng, chẳng hạn ngôn ngữ C. Vì vậy, tri thức của hệ chuyên gia và giao diện người dùng được thiết kế thuận tiện và hiệu quả.

Tương tự ngôn ngữ C, Pascal,trong một chương trình OPS83, người sử dụng không những có thể thực hiện các vòng lặp FOR, WHILE, mà còn có thể gọi hàm và thủ tục. Tuy nhiên, khác với C, Pascal, ..., OPS83 là một ngôn ngữ thiết kế hệ chuyên gia dựa trên luật, nên OPS83 còn cho phép xây dựng các luật.

OPS83 là phiên bản hoàn thiện của OPS5, vừa cho phép thiết kế hệ chuyên gia dựa trên luật, vừa cho phép xây dựng một giao diện đồ hoạ linh động tạo điều kiện thuận lợi trong việc giao tiếp giữa người sử dụng và hệ chuyên gia.

Phụ lục A Hướng dẫn sử dụng OPS5

Using the Workbench

Tools in the Workbench
The Workbench has been designed to make the development of OPS5 programs easier. The Workbench display consists of a top line displaying the product name and version, and the current active rule set name. The top line also functions as a message display line.
The second line on the screen is the main choice list. The choice list allows easy selection of one of the built in tools. When the Workbench is first entered, the Interp choice is highlighted. To select a choice, use either the left or right cursor keys to move the highlight to the desired choice. Then type enter, and the highlighted choice will be activated. To exit from any active choice, simply type **Esc**. You are then free to select another choice.
All choices may also be selected from within the active interpreter by typing cntl and the capitalized letter in the desired choice. For instance, to get help, type Ctrl-l. Choice list options prefixed by an exclamation point are '**Alt**' commands and may be executed from within the interpreter by typing alt and the capitalized letter of the desired choice.
The following is a list of the possible choices:

Interp	The main Sienna OPS5 interpreter
Edit	The Sienna Editor
heLp	Sienna OPS5 language summary
Wm	Pop-up window to display working memory
Cs	Pop-up window to display the conflict set
Options	Pop-up window to display and alter options
!Run	Alt command to run OPS5
!Step	Alt command to single step OPS5
!Halt	Alt command to abort a running program
!Back	Alt command to back up 1 rule firing
Pbreak	Pop-up window to display all current break points
Rules	Pop-up window to display active rules and operators
exit	Exit the Sienna OPS5 Workbench

1. The Two Sections of the Screen.
In addition to the top two workbench lines, the main window is divided into upper and lower sections, separated by a double horizontal line. The upper window is where all screen output from a running program is displayed (via WRITE), and where all input to the program (via ACCEPT or ACCEPTLINE) is read from. The lower window is where all interaction with the interpreter occurs. This is where top-level commands are entered and where trace and debugging information is displayed.
Each window is actually the full size of a screen, with only a portion being displayed at any one time. The position of the double horizontal line may be adjusted at any time that input can be entered. To move the line up, type ctrlÄu, and to move the line down, type Ctrl-D Function key F-2 will swap the position of the line about the center of the screen.
To display the entire program output screen (upper window) with nothing else displayed, type F-4. To display the entire lower window, type F-5.

Key Summary:

Ctrl-L	Display the HELP screen
Ctrl-E	Invoke the full screen editor
Ctrl-W	Pop-up the working memory brouse window
Ctrl-C	Pop-up the conflict set browser
Ctrl-O	Pop-up the Options window
Ctrl-P	Pop-up the Breakpoint window
Ctrl-R	Pop-up the Rules window
ALT-R	Run the rules
ALT-S	Single step
ALT-H	Halt rule firings
ALT-B	Back the interpreter up one rule firing
Ctrl-D	Move the double dividing line down
Ctrl-U	Move the double dividing line up
F1	Display the OPS5 help window
F2	Flip the dividing line about the center of the screen
F4	Display the upper screen only
F5	Display the lower screen only

2. Interp: The OPS5 Interpreter

The OPS5 interpreter is where you will spend most of your time. It can be entered only by moving the highlight to the Interp choice and typing enter.

If you have had experience with LISP implementations of OPS5, then you should feel comfortable in the Sienna OPS5 interpreter environment, since all of the same top-level commands are available, and work the same way. In addition to the standard commands are a few that are unique to Sienna OPS5, and among the standard commands are some that are additionally available through pop-up windows.

Choice list functions which are preceded by an exclamation point -- !Run !Step !Halt !Back; are top level '**Alt**' commands, and produce an immediate action. To invoke one of these commands, type alt and the first letter of the desired 'alt' command. The other entries on the choice list are '**Ctrl**' commands (except for Interp and exit) which can be invoked by typing ctrl followed by the capitalized letter in the command name. The ctrl commands cause a pop-up window to be displayed containing information specific to the command. alt and ctrl commands are dispatched immediately. Even if an OPS5 command has been entered at the command prompt, if a ctrl/alt command is invoked, it is immediately executed. The command that was typed at the OPS5 prompt will not be evaluated until the enter key is pressed. Both alt and ctrl commands are further described below.

3. Edit: How and When to Enter the Editor

The editor may be invoked by any one of four methods:

1. By moving the choice menu highlight to Edit and typing enter.
2. When in the interpreter, by typing Ctrl-E.
3. When in the Rules window, by selecting the Edit option, selecting a rule, and typing enter.
4. When loading a file into the interpreter, if a syntax error is found and the second parameter of the load function is 'T', the editor is invoked automatically with the cursor placed on the line where the error was detected.

When the first two methods are used, the editor buffer which was last displayed, or last loaded, will be the active buffer. If no files have been loaded into the system, then the default scratch buffer will be active.

When the third method is used to invoke the editor, the buffer containing the selected rule is made active and the top of the screen is positioned at the beginning of the selected rule. The editor can be exited at any time by pressing esc.

4. Help: Using the Help Screen

The help screen provides one screen of help, which is available either by selecting the help option on the choice list and pressing enter, or, when the interpreter is active, by typing Ctrl-L. Information on the Help screen is divided into three categories: commands, actions, and RHS functions. The commands category includes commands that can be entered at the top-level. In addition to those listed, any action preceded by an asterisk in the actions list can also be entered at the top level. The actions category are those actions that can be entered in the right hand side of a rule. The RHS function category are functions which can be placed within most RHS actions. The notational conventions of the help screen are as follows:

n	an integer number
p1	the name of a production
s	a symbol
v	a variable
< x x ...>	optional arguments
/	select one of the listed
[..]	one of the included options is required.

The heLp screen is meant to be a reminder of the most-used language elements and gives an indication of the syntax.

Press any key to exit the help window.

5. Options: Setting OPS5 Options

The Options window may be selected by either moving the choice list highlight to Options and pressing enter, or, when in the interpreter, by typing Ctrl-O. The Options window displays the current settings of the strategy, watch level, delay level, echo status, and the default file settings.

The strategy, watch, delay and echo status may be changed in the options window. You can select the item to change by pressing either the cursor up or cursor down keys. When you have selected the item to change, one of the allowable values for that item may be selected by pressing the left or right cursor keys. The strategy, watch and delay options are also top level commands.

The default files section displays the currently active files for the trace, accept and write files. To change these values you must use the OPENFILE and DEFAULT actions at the top-level.

To leave the Options window, press the esc or enter key.

6. Wm: Displaying the Contents of Working Memory

The Wm (Working Memory) window may be displayed by selecting the Wm option from the choice list and pressing enter, or by typing Ctrl-W whenever text input can be entered, when the Cs window is displayed, or when the Rules window (but no other) is displayed.

The current contents of working memory are displayed in order of recency, with the latest addition to, or change of, working memory being displayed first.

The bottom of the window lists the active keys for the window. If there are more WMEs in working memory than can be displayed in the window, the cursor up, cursor down, page up, page down, home and end keys can be used to move through the rest of working memory. If the WMEs are too long to be displayed in the window, the left cursor and right cursor keys can be used to shift the window contents left and right.

To exit the window press esc.

7. Cs: Displaying the Conflict Set Entries

The Cs window is invoked by selecting the CS option from the choice list, by typing ctrl-c whenever text input can be entered, or when the Wm or Rules window, but no other are displayed. The Cs window displays the contents of the current conflict set, sorted in the order in which the rule instantiations would fire. Displayed with the name of the instantiated rule is the list of time tags of instantiating WMEs for the rule. Given the instantiating WME time tags, the Wm window can be invoked to look up the actual working memory elements.
If there are more conflict set entries than can be displayed in the window at one time, the cursor up, cursor down, page up, page down, home and end keys may be used to view the rest of the entries. The cursor left and cursor right keys can be used to scroll the window to the left and right as needed.
To exit the window, press esc.

8. !Run -- Using the Run Choice

The !Run choice can be selected with the choice list highlight, or can be invoked by typing Alt-R from within the interpreter. This has the same effect as typing "(RUN)" at the OPS5 prompt. The interpreter begins running and continues until no productions are instantiated or a halt action or break point is encountered.
Note that, when within the interpreter, typing alt-R causes the interpreter to be run immediately. Commands which may have been entered at the OPS5 prompt are not evaluated unless enter has been first pressed from within the interpreter.

9. !STEP -- Single Stepping Through Your Program

The !Step commands can be invoked by moving the highlight to that option and pressing enter, or by typing alt-s from within the interpreter.
The action of the !Step command is to fire 1 production from the conflict set. It is functionally equivalent to typing (RUN 1) at the OPS5 prompt, but somewhat more convenient.
Commands entered but not evaluated are not evaluated by the !Step command.

10. !HALT: How to Stop a Runaway Program

The !Halt commands can be invoked either by selecting it in the choice list menu, or by typing Alt-H while a program is running. However, since the !Halt commands is useful only while the interpreter is running, and it is not possible to select an option with the highlight during that time, the only meaningful way that the command can be invoked is by typing Alt-H.
The !Halt command is functionally equivalent to the evaluation of a Halt action in the RHS of a production. When the !Halt command is executed the interpreter is halted immediately after the current rule has completed firing, and control returns to the top level.

11. !BACK: Running Backwards -- Undoing Rule Firings

The !Back commands can be selected from the choice list by moving the highlight to the option and pressing enter, or by typing alt-b from within the interpreter.
The !Back command is functionally equivalent to typing (BACK 1) at the OPS5 prompt. The interpreter is backed up one rule firing and the system is restored to the state prior to the last rule firing.
The action of this command is immediate. Any commands entered at the OPS5 prompt but not yet evaluated by typing enter are not evaluated by invoking this command.

12. PBREAK: Listing Break Points

The Pbreak option opens a window which displays all currently set break points. The option can be selected from the menu by highlighting the option and pressing enter, or by typing CNTl-p from within the interpreter.

Break points cannot be set or cleared with this option, use the Rule window instead. If no break points have been set then the message "** no break points set **" is displayed in the window. To close the window and exit, press **Esc**. To set or clear break points use either the rules window or the pbreak command.

13. RULES: Displaying and Manipulating Rules

1. How to use the Rules screen

The Rules window can be invoked by selecting the Rules choice with the highlight and pressing enter, or by typing Ctrl-R whenever text input can be entered or while the Wm or Cs windows are displayed.

The Rules window displays a list of the compiled rules and allows one r of four operations: Edit, Matches, Pbreak, and Excise, to be applied r to the selected rule. To select a rule, use the cursor up and cursor r down keys to highlight the desired rule. When there are more rules r than can fit on one rules screen, you can also use the page up, page r down, home and end keys to view the rest of the rules. If rule names r are too long to fit within the window, the Ctrl-Cursor-Left and r Ctrl-Cursor-Right keys can be used to view the truncated portion of r the rule names.

To select one of the four operations to apply to the selected rule, r uses the Cursor-Left and Cursor-Right keys to make you choice, and r then press enter.

To summarize the Rules window keys and their actions:

Cursor up	move rules highlight bar up
Cursor down	move rules highlight down
Page up	scroll up one window
Page down	scroll down one window
Home	to top of rules
End	to bottom of rules
Ctrl cursor left	scroll window left
Ctrl cursor right	scroll window right

2. EDIT: Editing a rule

A rule may be edited by selecting the edit option along with a r particular rule. When enter is pressed Sienna Edit is entered. The r buffer that contains the rule is made the active buffer, and the top r of the buffer window is placed at the top of the selected rule. If r the rule cannot be found, then an error message is displayed.

A rule will not be found if the edit buffer containing the rule does r not exist, or if the rule text has been deleted.

3. MATCHES: Displaying WMEs that match a rule

The Matches option performs the same task as the matches command r from the OPS5 prompt. When the Matches option is used from the rules r window is used, however, a window is opened and the matches for the r rule are displayed in the window. If there is more information than r can be displayed at one time in the window, then any of the special r keys listed at the bottom of the window can be used to browse through r the display.

Some care should be exercised when using the Matches option. The r matches information, when displayed in the window, must be formatted r into a data structure in order to allow browsing through the list. If r the matches of a rule which creates large cross products of

objects is r displayed, the resulting space needed to preformat the information r could exceed available memory. If you suspect that this will be the r case, then use the matches command at the OPS5 prompt instead.
To close the Matches window, press esc.

4. PBREAK: Setting and clearing break points

The Pbreak option allows you to easily set and clear break points r on rules. To set or clear a break point, select Pbreak with the r highlight, select the rule you want to set/clear, and press the enter r key. A message will be displayed at the top of the screen.
When a rule has a break point set on it, an asterisk is visible to the r left of the rule name. When the break point is cleared, the asterisk ris cleared also.

5. EXCISE: Purging rules from the Knowledge Base

The Excise option can be used to purge a rule from the RETE network. To excise a rule, select the rule in the Rules window, select the Excise option, and press enter. A window will appear to verify that you want the rule excised. If you do, then type 'Y'. If not, just press enter or esc to cancel the operation.
Excising a rule from the RETE network does NOT delete the rule text from the edit buffer, if it exists in one.

14. EXIT: Leaving the Workbench

To exit the workbench you must select the exit option on the main choice list, and then press enter. If you are in the interpreter, the editor, or any other selection, then you must first return to the choice list by pressing esc.
Upon exit, if any edit buffers have been altered since being last saved, a warning is given along with the chance to abort the exit. You may then enter the editor and save the modified buffers by pressing F-4 within each buffer.

http://www.haley.com/3173193312476179/ReteAlgorithm.html

Phụ lục B Một số hệ chuyên gia

Historical Projects

A number of major projects are now considered completed work; their goals have been met, and our research attention has moved on to new areas.

DENDRAL (1965-83)

The DENDRAL Project was one of the earliest expert systems. DENDRAL began as an effort to explore the mechanization of scientific reasoning and the formalization of scientific knowledge by working within a specific domain of science, organic chemistry. Another concern was to use AI methodology to understand better some fundamental questions in the philosophy of science, including the process by which explanatory hypotheses are discovered or judged adequate. After more than a decade of collaboration among chemists, geneticists, and computer scientists, DENDRAL had become not only a successful demonstration of the power of rule-based expert systems but also a significant tool for molecular structure analysis, in use in both academic and industrial research labs. Using a plan-generate-test search paradigm and data from mass spectrometry and other sources, DENDRAL proposes plausible candidate structures for new or unknown chemical compounds. Its performance rivals that of human experts for certain classes of organic compounds and has resulted in a number of papers that were published in the chemical literature. Although no longer a topic of academic research, the most recent version of the interactive structure generator, GENOA, has been licensed by Stanford University for commercial use.

META-DENDRAL (1970-76)

META-DENDRAL is an inductive prograrn that automatically formulates new rules for DENDRAL to use in explaining data about unknown chemical compounds. Using the plan-generate-test paradigm, META-DENDRAL has successfully formulated rules of mass spectrometry, both by rediscovering existing rules and by proposing entirely new rules. Although META-DENDRAL is no longer an active program, its contributions to ideas about learning and discovery are being applied to new domains. Among these ideas are that induction can be automated as heuristic search; that, for efflciency, search can be broken into two steps--approximate and refined; that learning must be able to cope with noisy and incomplete data; and that learning multiple concepts at the same time is sometimes inescapable.

MYCIN (1972-80)

MYCIN is an interactive program that diagnoses certain infectious diseases, prescribes antimicrobial therapy, and can explain its reasoning in detail. In a controlled test, its performance equalled that of specialists. In addition, the MYCIN program incorporated several important AI developments. MYCIN extended the notion that the knowledge base should be separate from the inference engine, and its rule-based inference engine was built on a backward-chaining, or goal-directed, control strategy. Since it was designed as a consultant for physicians, MYCIN was given the ability to explain both its line of reasoning and its knowledge. Because of the rapid pace of developments in medicine, the knowledge base was designed for easy augmentation. And because medical diagnosis often involves a degree of uncertainty, MYCIN's rules incorporated certainty factors to indicate the importance (i.e., likelihood and risk) of a conclusion. Although MYCIN was never used routinely by physicians, it has substantially influenced other AI research. At the HPP, MYCIN led to work in TEIRESIAS, EMYCIN, PUFF, CENTAUR, VM, GUIDON, and SACON, all described below, and to ONCOCIN and ROGET. The book *Rule-Based Expert Sytem: The MYCIN*

Experiment at the Stanford Heuristic Programming Project describes the decade of research on MYCIN and its descendants.

TEIRESIAS (1974-77)

The knowledge acquisition program TEIRESIAS was built to assist domain experts in refining the MYCIN knowledge base. TEIRESIAS developed the concept of metalevel knowledge, i.e., knowledge by which a program can not only use its knowledge directly, but can examine it, reason about it, and direct its use. TEIRESIAS makes clear the line of reasoning used in making a diagnosis and aids physician experts in modifying or adding to the knowledge base. Much of this was incorporated into the EMYCIN framework. The flexibility and understandability that TEIRESIAS introduced into the knowledge base debugging process have been models for the design of many expert systems.

EMYCIN (1974-79)

The core inference engine of MYCIN, together with a knowledge engineering interface, was developed under the name EMYCIN, or "Essential MYCIN." It is a domain-independent framework that can be used to build rule-based expert systems for consultation problems such as those encountered in diagnosis or troubleshooting. EMYCIN continues to be a primary example of software that can facilitate building expert systems and has been used in a variety of domains, both medical (e.g., PUFF) and nonmedical (e.g., SACON). The system has been widely distributed in the U.S. and abroad and is the basis for the Texas Instruments software system called Personal Consultant.

PUFF (1977-79)

The PUFF system was the first program built using EMYCIN. PUFF's domain is the interpretation of pulmonary function tests for patients with lung disease. The program can diagnose the presence and severity of lung disease and produce reports for the patient's file. Once the rule set for this domain had been developed and debugged, PUFF was transferred to a minicomputer at Pacific Medical Center in San Francisco, where it is used routinely to aid with interpretation of pulmonary function tests. A version of PUFF has been licensed for commercial use.

CENTAUR (1977-80)

The CENTAUR system was designed to experiment with an expert system that combines both rule- and frame-based approaches to represent and use knowledge about medicine and medical diagnostic strategies. For purposes of comparison, CENTAUR was developed for the same task domain as PUFF, interpretation of pulmonary function tests. CENTAUR performed well, demonstrating the effectiveness of this representation and control methodology.

VM (1977-81)

The Ventilator Manager (VM) program interprets online quantitative data in the intensive care unit (ICU) and advises physicians on the management of post-surgical patients needing a mechanical ventilator to help them breathe. While based on the MYCIN architecture, VM was redesigned to allow for the description of events that change over time. Thus, it can monitor the progress of a patient, interpret data in the context of the patient's present and past condition, and suggest adjustments to therapy. VM was tested in the surgical ICU at Pacific Medical Center in San Francisco. Some of the program's concepts have been built directly into more recent respiratory monitoring devices.

GUIDON (1977-81)

GUIDON is an experimental program intended to make available to students the expertise contained in EMYCIN-based systems. GUIDON incorporates separate knowledge bases for the domain itself and for tutoring, and engages the student in a dialogue that presents dornain knowledge in an organized way over a number of sessions. Using the MYCIN knowledge base as the domain to be taught, work in GUIDON explored several issues in intelligent computer-assisted instruction (ICAI), including means for structuring and planning a dialogue, generating teaching material, constructing and verifying a model of what the student knows, and explaining expert reasoning. Although GUIDON was successful in many respects, it also revealed that the diagnostic strategies and some of the medical knowledge that were contained implicitly in the MYCIN rules had to be made explicit in order for students to understand and remember them easily. As a result, a new expert system, NEOMYCIN, has been developed.

SACON (1977-78)

SACON (for Structural Analysis CONsultant) was implemented as a test of the EMYCIN framework in an engineering context. SACON advised structural engineers on the use of MARC, a large structural analysis program, and has served as a prototype of many advisory systems.

MOLGEN (1975-84)

The MOLGEN project has applied AI methods to research in molecular biology. Initial work focused on acquiring and representing the expert knowledge needed to design and simulate experiments in the domain. This led to the development of UNITS, described below. The second phase of research resulted in two expert systems, representing distinct approaches to the design of genetic experiments. One system used "skeletal plans," which are abstracted outlines of experiment designs that can be applied to specific experimental goals and environments. The other system was based on planning with constraints, in which planning decisions are made in the spaces of overall strategy, domain-independent decisions, and domain-dependent laboratory decisions, and the interaction of separate steps or subproblems of an experiment constitute constraints on the overall problem. These two systems were later synthesized into a third system, called SPEX. Current work, known as MOLGEN-II (see the section "The Heuristic Programming Project"), is investigating the process of theory formation in molecular biology.

UNITS (1975-81)

The frame-based UNITS system was developed in the MOLGEN project as a general-purpose knowledge representation, acquisition, and manipulation tool. Designed for use by domain experts with little previous knowledge of computers, it provides an interface that allows the expert to describe both factual and heuristic knowledge. It contains both domain-independent and domain-specific components, including modified English rules for describing the procedural knowledge. UNITS has been licensed by Stanford University for commercial development.

AM (1974-80)

The AM program explored machine learning by discovery in the domain of elementary mathematics. Using a framework of 243 heuristic rules, AM successfully proposed plausible new mathematical concepts, gathered data about them, noticed regularities, and, completing this cycle, found ways of shortening the statement of those hypotheses by making new definitions. However, AM was not able to generate new heuristics. This failing was found to be inherent in the design of AM; related work on discovering new heuristics was done as part of EURISKO.

EURISKO (1978-84)

A successor to AM, EURISKO has also investigated automatic discovery, with a particular emphasis on heuristics, their representation, and the part played by analogy in their discovery. Several hundred heuristics, mostly related to functions, design, and simulation, guide EURISKO in applying its knowledge in several domains. In each domain, the program has three levels of task to perform: working at the domain level to solve problems; inventing new domain concepts; and synthesizing new heuristics that are specific and powerful enough to aid in handling tasks in the domain. EURISKO has been applied to elementary mathematics; programming, where it has uncovered several Lisp bugs; naval fleet design, where it has reigned undefeated in the Traveller Trillion Credit Squadron tournament; VLSI design, where it has come up with some novel and potentially useful three-dimensional devices; oil-spill cleanup; and a few other domains.

RLL (1978-80)

RLL (for Representation Language Language) is a prototype tool for building customized representation languages. RLL is self-descriptive, i.e., it is itself described in terms of RLL units. It has been used as the underlying language for EURISKO and other systems.

Contract Nets (1976-79)

The Contract Nets architecture is an early contribution to work on computer architectures for parallel computation. Recently, it has received much attention in the emerging literature on multiprocessor architectures for symbolic computation. In the Contract Nets architecture, problem solving is distributed among decentralized and loosely coupled processors. These processors communicate about task distribution and answers to subproblems through an interactive negotiation analogous to contract negotiation in the building trades: the "contract" is given to the processor that can handle the task at the lowest system cost, and failure to complete a task results in its reassignment to another processor.

CRYSALIS (1976-83)

The CRYSALIS project explored the power of the blackboard model in interpreting X-ray data from crystallized proteins. The overall strategy was to piece together the three-dimensional molecular structure of a protein by successively refining descriptions of the structure. Although the knowledge base was developed for only a small part of the problem, the blackboard model with its hierarchical control structure was shown to be very powerful for solving such highly complex problems. Results from CRYSALIS are currently being incorporated in other KSL work and have contributed to improved models of control.

AGE (1976-82)

The AGE (for Attempt to GEneralize) project sought to develop a software laboratory for building knowledge-based programs. AGE-1, the knowledge engineering tool that resulted, is designed for building programs that use the blackboard problem-solving framework. It can aid in the construction, debugging, and running of a program. AGE-1 has been used in a number of academic laboratories and for various applications in industry and the defense community.

QUIST (1978-81)

QUIST combines AI and conventional database technology in a system that optimizes queries to large relational databases. QUIST uses heuristics embodying semantic knowledge about the contents of the database to make inferences about the meanings of the terms in a query. It reformultes the original query into an equivalent one whose answer can be found in the database more efficiently. Then conventional query optimization techniques are used to plan an efflcient sequence of retrieval operations.

GLisp (1982-83)

GLisp is a programming language that allows programs to be written in terms of objects and their properties and behavior. The GLisp compiler converts such programs into efficient Lisp code. The compiler has been released to outside users, along with the GEV window-based data inspector, which displays data according to their GLisp description. GLisp is now being distributed from the University of Texas.

Model of Endorsement (1982-85)

The model of endorsement represents and reasons with heuristic knowledge under uncertainty. Instead of associating numerical weights with evidence, the model of endorsement discriminates kinds of evidence and distinguishes the importance of different evidence-gathering situations. Thus, this model's significance is that it examines the question of how to reason about uncertainty, as well as with it. in expert systems.

AI Handbook (1975-82)

The Handbook of Artificial Intelligence was a community effort by KSL (formerly HPP) students and researchers plus collaborators around the country. It describes the fundamental ideas, useful techniques, and exemplary programs from the first 25 years of AI research. Designed for scientists and engineers with no AI background. the three-volume *Handbook* book contains some 200 articles organized into 15 chapters. Chapters cover such topics as

General Readings

Clancey, W. J., and E. H. Shortliffe. *Readings in Medical Artificial Intelligence: The First Decade.* Reading, MA: Addison-Wesley, 1984.
Feigenbaum, E. A., and P. McCorduck. *The Fifth Generation: Artificial Intelligence and Japan's Computer Challenge to the World.* Reading, MA: Addison-Wesley, 1983.
Hayes-Roth, F., D. A. Waterman, and D. Lenat, eds. *Building Expert Sytems.* Reading, MA: Addison-Wesley, 1983.
Barr, A., E. A. Feigenbaum, and P. Cohen, eds. *The Handbook of Artificial Intelligence,* Volumes 1-3. Los Altos, CA: Kaufmann, 1981, 1982.
Feigenbaum, E. A. The art of artificial intelligence: 1. Themes and case studies of knowledge engineering. *Proceedings IJCAI-77,* pp. 1014-1029. (Also published in *AFIPS Conf Proceedings: 1978 Computer Conference.* Montvale, NJ: AFIPS Press, 1978.)

AGE

Aiello, N., C. Bock, H. P. Nii, and W. White. AGE reference manual. Memo HPP-81-24 (Knowledge Systems Laboratory), October 1981.
Aiello, N., C. Bock, H. P. Nii, and W. White. Joy of AGing: an introduction to AGl system. Memo HPP-81-23 (Knowledge Systems Laboratory), October 1981.
Nii, H. P. Introduction to knowledge engineering, blackboard model, and AGE. Memo HPP-80-29 (Knowledge Systems Laboratory), March 1980.
Nii, H. P., and N. Aiello. AGE: a knowledge-based program for building knowledge-based programs. *Proceedings IJCAI-79,* pp. 645-655.

AM

Davis, R., and D. Lenat. *Knowledge-Based Systems in Artificial Intelligence: AM and TEIRESIAS.* New York: McGraw-Hill, 1982.
Blackboard Architecture
Blackboard Architecture Hayes-Roth, B. The blackboard model of control. *Artificial Intelligence,* in press.
Hayes-Roth, B. BB-l: an environment for building blackboard systems. Memo HPP-8416 (Knowledge Systems Laboratory), 1984.

Hayes-Roth, B. The blackboard architecture: a general framework for problem-solving? Memo HPP-83-30 (Knowledge Systems Laboratory), May 1983.

CENTAUR

Aikins, J. S. Prototypical knowledge for expert systems. *Artificial Intelligence* 20(2):163-210 (1983).

Contract Nets

Smith, R. G. A framework for problem solving in a distributed processing environment. Memo HPP-78-28 (Knowledge Systems Laboratory), December 1978. Also Stanford CS Report STAN-CS-78-700, 1978.

CRYSALIS

Engelmore, R., and A. Terry. Structure and function of the CRYSALIS system. *Proceeding IJCA1-79,* pp. 25256.

DART/HELIOS

Foyster, G. HELIOS user's manual. Memo HPP-84-34 (Knowledge Systems Laboratory), August 1984.
Singh, N. MARS: a multiple abstraction rule-based simulator. Memo HPP-83-43 (Knowledge Systems Laboratory), December 1983.
Joyce, R. Reasoning about time-dependent behavior in a system for diagnosing digital hardware faults. Memo HPP-83-37 (Knowledge Systems Laboratory), August 1983.
Genesereth, M. R. Diagnosis using hierarchical design models. *Proceedings AAA1-82,* pp. 278-283.
Genesereth, M. R. The use of design descriptions in automated diagnosis. Memo HPP-81-20 (Knowledge Systems Laboratory), January 1981.

DENDRAL

[There have been more than 100 publications about DENDRAL, describing both the chemical results obtained using the program and the AI issues explored.]

Lindsay, R. K., B. G. Buchanan, E. A. Feigenbaum, and J. Lederberg. *Application of Artificial Intelligence for Chemistry: The DENDRAL Project.* New York: McGraw-Hill, 1980.
Gray, N. A. B., D. H. Smith, T. H. Varkony, R. E. Carhart, and B. G. Buchanan. Use of a computer to identify unknown compounds: the automation of scientific inference. Chapter 7 in G. R. Waller and O. C. Dermer, eds., *Biomedical Application of Mass Spectrometry.* New York: Wiley, 1980.
Buchanan, B. G., and E. A. Feigenbaum. DENDRAL and META-DENDRAL: their applications dimensions. *Artificial Intelligence* 11:5-24 (1978).

META-DENDRAL

Buchanan, B. G., and T. Mitchell. Model directed learning of production rules. In D. A. Waterman and F. Hayes-Roth, eds., *Pattern-Directed Inference System.* New York: Academic Press, 1978.

EURISKO

Lenat, D. EURISKO: a program that learns new heuristics and domain concepts. *Artificial Intelligence* 21(2):61-98 (1983).
Lenat, D. Theory formation by heuristic search. *Artificial Intelligence* 21(1):31-59 (1983).
Lenat, D. The nature of heuristics. *Artificial Intelligence* 19(2): 189-249 (1981) .

GLisp

Novak, G. S., Jr. GLisp: a high-level language for AI programming. *Proceedings AAAI-82,* pp. 238-241.
Novak, G. S., Jr. GLisp user's manual. Memo HPP-82-1 (Knowledge Systems Laboratory), January 1982.

GUIDON

Hasling, D., W. J. Clancey, and G. Rennels. Strategic explanations for a diagnostic consultation system. *International Journal of Man-Machine Studies* 20(1):3-19 (1984).
London, B., and W. J. Clancey. Plan recognition strategies in student modeling: prediction and description. *Proceedings AAAI-82,* pp. 335-338.
Clancey, W. J. lutoring rules for guiding a case method dialogue. *International Journal of Man-Machine Studies* 11:25-49 (1979).
Clancey, W. J. Dialogue management for rule-based tutorials. *Proceedings IJCAI-79,* pp. 155-161.

Intelligent Agent

Rosenschein, J., and M. R. Genesereth. Communication and cooperation. Memo HPP-84-5 (Knowledge Systems Laboratory), March 1984.
Finger, J. J., and M. R. Genesereth. RESIDUE: a deductive approach to design. Memo HPP-83-46 (Knowledge Systems Laboratory), December 1983.
MacKinlay, J. Intelligent presentation of information: the generation problem of user interfaces. Memo HPP-83-34 (Knowledge Systems Laboratory), March 1983.
Finger, J. J. Sensory planning. Memo HPP-82-12 (Knowledge Systems Laboratory), April 1982.

KBVLSI

Brown, H., C. Tong, and G. Foyster. PALLADIO: an exploratory environment for circuit design. *Computer* 16(12):41-56 (1983).

Knowledge Acquisition

Bennett, J. S. ROGET: a knowledge-based system for acquiring the conceptual structure of an expert system. *Journal of Automated Reasoning* 1(1):49-74 (1985)
Dietterich, T. G. Constraint propagation techniques for theory-driven data interpretation. Memo HPP 84-46 (Knowledge Systems Laboratory), December 1984.
Dietterich, T. G., and B. G. Buchanan. The role of the critic in learning systems. In O. Selfridge, E. Rissland, and M. Arbib, eds, *Adaptive Control of Ill-Defined Systems.* New York: Plenum, 1984. (NATO Advanced Workshop on Adaptive Control of Ill-Defined Systems; Devon, England, June 1981.) Also Memo HPP-81-19 (Knowledge Systems Laboratory), January 1981.
Buchanan, B. G., T. M. Mitchell, R. G. Smith, and C. R. Johnson, Jr. Models of learning systems. *Encyclopedia of Computer Science and Technology* 11 (1978).
Mitchell, T. M. Version spaces: an approach to concept learning. Memo HPP-79-2 (Knowledge Systems Laboratory), January 1979. Also Stanford CS Report STAN-CS-78-711, 1978.

Model of Endorsement

Cohen, P. *Heuristic Reasoning about Uncertainty: An AI Approach.* Boston: Pitman, 1985.

MOLGEN

Friedland, P., and L. Kedes. Discovering the secrets of DNA. To appear in joint issue *ACM/Computer,* October 1985.

Friedland, P., and Y. Iwasaki. The concept and implementation of skeletal plans. *Journal of Automated Reasoning* 1(2) (in press).

ach, R., Y. Iwasaki, and P. Friedland. Intelligent computational assistance for experiment design. *Nucleic Acids Research,* January 1984.

Iwasaki, Y., and P. Friedland. SPEX: a second-generation experiment design system. *Proceedings AAA1-82,* pp. 341-344.

Stefik, M. Planning with constraints. Memo HPP-80-2 (Knowledge Systems Laboratory), January 1980. Also Stanford CS Report STAN-CS-80-784, 1980.

Friedland, P. Knowledge-based experiment design in molecular genetics. *Proceedings IJCA1-79,* pp. 285-287.

MYCIN and EMYCIN

Buchanan, B. G., and E. H. Shortliffe. *Rule-Based Expert Systems: The MYCIN Experiments oJ the Stanford Heuristic Programming Project.* Reading, MA: Addison-Wesley, 1984.

van Melle, W. *System Aids in Constructing Conultation Programs: EMYCIN.* Ann Arbor, MI: UMI Research Press, 1982.

MRS

Smith, D. E., and M. R. Genesereth. Controlling infinite chains of inference. Memo HPP-84-6 (Knowledge Systems Laboratory), February 1984.

Genesereth, M. R., and D. E. Smith. Partial programs. Memo HPP-841 (Knowledge Systems Laboratory), January 1984.

Genesereth, M. R. An overview of meta-level architecture. *Proceedings AAA1-8,* pp. 119-124.

Genesereth, M. R., R. Greiner, and D. E. Smith. A meta-level representation system. Memo HPP-83-28 (Knowledge Systens Laboratory), May 1983.

NEOMYCIN

Clancey, W. J. Methodology for building an intelligent tutoring system. In W. Kintsch, J.R. Miller, and P.G. Polson, eds., *Method and Tactics in Cognitive Science.* Hillsdale, NJ: Lawrence Erlbaum Associates, 1984.

Clancey, W. J. Acquiring, representing, and evaluating a competence model of diagnosis. In M.T.H. Chi, R. Glaser, and M. Farr, eds., *The Nature of Expertise,* in preparation. Also Memo HPP-84-2 (Knowledge Systems Laboratory), February 1984.

Clancey, W. J. The epistemology of a rule-based expert system: a framework for explanation. *Artificial Intelligence* 20(3): 215-251(1983).

Clancey, W. J. The advantages of abstract control knowledge in expert system design. *Proceedings AAA1-8,* pp. 74-78.

ONCOCIN

Bischoff, M. B., E. H. Shortliffe, A. C. Scott, R. W. Carlsen, and C. D. Jacobs. Integration of a computer-based consultant into the clinical setting. *Proceedings of the Seventh Annual Symposium on Computer Applications in Medical Care,* pp. 149-152 (October 1983).

Tsuji, S., and E. H. Shortliffe. Graphical access to the knowledge base of a medical consultation system. *Proceedings of AAMSI Congress 1983,* pp. 551-555.

Langlotz, C. P., and E. H. Shortliffe. Adapting a consultation system to critique user plans. *International Journal of Man-Machine Studies* 19(5):479-496 (1983).

Gerring, P. E., E. H. Shortliffe, and W. van Melle. The interviewer reasoner model: an approach to improving system responsiveness in interactive AI systems. *AI Magazine* 3(4):24-27 (1982).
Suwa, M., A. C. Scott, and E. H. Shortliffe. An approach to verifying completeness and consistency in a rule-based expert system. *AI Magazine* 3(4):16-21 (1982).
Shortliffe, E. H., A. C. Scott, M. B. Bischoff, A. B. Campbell, W. van Melle, and C. D. Jacobs. ONCOCIN: an expert system for oncology protocol management. *Proceedings IJCA1-81,* pp. 876-881.

PATHFINDER

Horvitz, E. J., D. E. Heckerman, B. N. Nathwani, and L. M. Fagan. Diagnostic strategies in the hypothesis-directed PATHFINDER system. *First Conference on Artificial Intelligence Applications,* pp. 630-636 (IEEE Computer Society, 1984).

PIXIE

Sleeman, D. H. Basic algebra revisited: a study with 14-year-olds. *International Journal of Man-Machine Studies,* in press. Also Memo HPP-83-9 (Knowledge Systems Laboratory), February 1983.
Sleeman, D. H. A user modelling front end subsystem. *International Journal of Man-Machine Studies,* in press.
Sleeman, D. H. Inferring (mal)rules from pupils' protocols. *Proceedings of the 1982 European AI Conference,* pp. 160-164.
Sleeman, D. H. Inferring student models for intelligent computer-aided instruction. In R. S. Michalski, J. G. Carbonell, and T. M. Mitchell, eds., *Machine Learning.* Palo Alto, CA: Tioga Press, 1982.
Sleeman, D. H., and J. S. Brown. Intelligent tutoring systems: an overview. In D. H. Sleeman and J. S. Brown, eds., *Intelligent Tutoring Systems.* New York: Academic Press, 1982.

PUFF

Aikins, J. S., J. C. Kunz, E. H. Shortliffe and R. J. Fallat. PUFF: an expert system for interpretation of pulmonary function data. *Computers and Biomedical Research* 16:199-208 (1983).

QUIST

King, J.J. Query optimization by semantic reasoning. Stanford CS Report STAN-CS-81-861, 1981.

RADIX

Blum, R. L. Representation of empirically derived causal relationships. *Proceedings NCA1-8,* pp. 268-271.
Blum, R. L. Discovery, confirmation, and incorporation of causal relationships from a large time-oriented database: the RX Project. Computers and Biomedical Research 15(2):164-187 (1982).
Blum, R. L. Discovery and representation of causal relationships from a large time-oriented database: the RX Project. In D. A. B. Lindberg and P. L. Reichertz, eds., *Medical Informatics* 19 (1982).

RLL

Greiner, R., and D. B. Lenat. A representation language language. *Proceedings of AAA1-80,* pp. 165-169.

SACON

Bennett, J. S., and R. S. Engelmore. SACON: a knowledge-based consultant for structural analysis. *Proceedings IJCA1-79,* pp. 47-49.

SOAR

Rosenbloom, P. S., J. E. Laird, J. McDermott, A. Newell, and E. Orciuch. Rl-SOAR: an experiment in knowledge-intensive programming in a problem-solving architecture. *Proceedings of the IEEE Workshop in Principles of Knowledge-Based Systems,* 1984.
Laird, J. E., P. S. Rosenbloom, and A. Newell. Towards chunking as a general learning mechanism. *Proceeding AAA1-84,* pp. 188-192.

TEIRESIAS

Davis, R., and D. Lenat. *Knowledge-Based Systems in Artificial Intelligence: AM and TEIRESIAS.* New York: McGraw-Hill, 1982.

UNITS

Smith, R., and P. Friedland. Unit package user's guide. Memo HPP-80-28 (Knowledge Systems Laboratory), December 1980.
Stefik, M. An examination of a frame-structured representation system. *Proceedings IJCA1-79,* pp. 845-852.

VM

Fagan, L. M. VM: representing time-dependent relations in a medical setting. Memo HPP 831 (Knowledge Systems Laboratory), June 1980.
Osborn, J., L. M. Fagan, R. Fallat, D. McClung, and R. Mitchell. Managing the data from respiratory measurements. *Medical Instrumentation* 13:6 (1979).

Phụ lục C Tham khảo

Une nouvelle ère qui s'ouvre...

Il est difficile de parler du Système Expert sans parler d'Intelligence Artificielle, ou IA en abréviation. L'IA est une combinaison/association de la science des ordinateurs (informatique), de la physiologie et de la philosophie. Elle comprend de multitudes disciplines, depuis la vision artificielle jusqu'au système expert en passant par la programmation des jeux. L'élément en commun de toutes les disciplines de l'IA est la création des machines qui "pensent".

Pour qu'une machine soit classée "pensante", il est nécessaire de définir ce qu'est "intelligence". A quel degré d'intelligence est associé à la résolution des problèmes complexes, à la création des liens entre les faits ou encore à la généralisation d'une règle ? Et la perception, la compréhension ? Des recherches dans le domaine d'apprentissage, de langage, de la perception sensorielle ont énormément contribué à la conception des machines intelligentes.

L'un des plus grands défis d'un expert IA est de concevoir un système qui imite le comportement humain, composé de milliards de neurones, capable d'entretenir les affaires les plus complexes du monde. Le début de l'Intelligence Artificielle remonte bien avant l'électronique, à l'époque des philosophes et des mathématiciens (George BOOLE) et bien d'autres théories ou principes qui ont été utilisés comme fondement de l'IA.

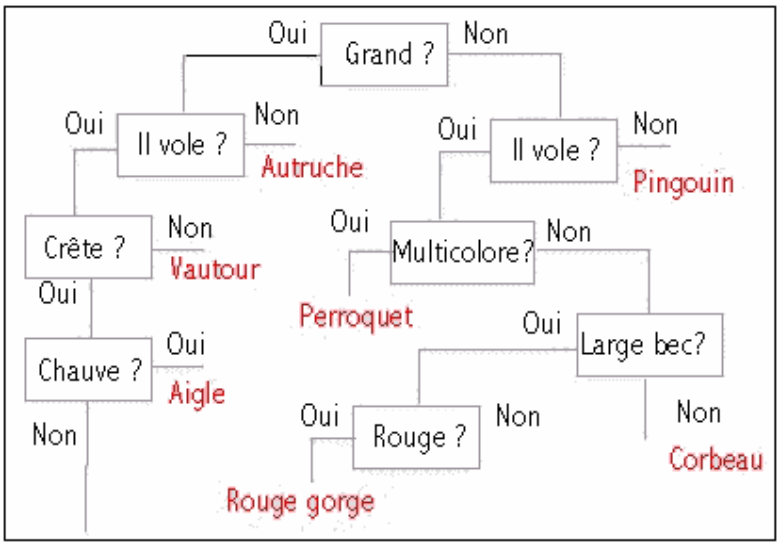
Elle intriguait les chercheurs avec l'invention de l'ordinateur en 1943. La technologie est finalement disponible et apte (semble-t-il) à simuler un comportement "intelligent". Quarante ans plus tard et après quelques marches trébuchantes, l'Intelligence Artificielle grandissait d'une dizaine de chercheurs à des milliers d'ingénieurs et de spécialistes, de programmes d'ordinateur capable de jouer aux échecs aux systèmes capables de diagnostiquer les maladies.

Un réseau neuronal

Le cerveau humain est constitué d'une toile de milliards de cellules appelées neurones, comprendre ses complexités semble l'une des dernières frontières des recherches scientifiques. Une des voies de recherche en Intelligence Artificielle consiste à construire des circuits électroniques qui sont capables de réagir comme des neurones dans un cerveau humain. Mais, la plupart des activités neuronales restent inconnues, la complexité du réseau neuronal constitue précisément ce que l'on appelle l'intelligence humaine. De par lui-même une neurone n'est pas intelligente, mais une fois assemblée les neurones sont capables de de passer les signaux électriques au travers de ses réseaux.

Système Expert, une approche descendante

Grâce aux énormes capacités de stockage d'information, un système expert est capable d'interpréter les statistiques et en déduire des règles. Un système expert fonctionne comme un détective qui cherche à résoudre une énigme. En utilisant les informations, les règles et la logique, il parvient à trouver la solution au problème.Ainsi, un système expert conçu pour identifier les oiseaux pourrait avoir un raisonnement suivant.



Un diagramme de ce type représente la logique des systèmes experts. En utilisant un jeu de règles similaires, les systèmes experts peuvent avoir des applications diverses et variées. Avec une interface homme machine améliorée, les ordinateurs trouvent encore une place plus large dans la société.

Tài liệu tham khảo

[1] Ivan Brako. *Programmation en Prolog pour l'Intelligence Artificielle*. Inter-Editions, Paris 1988.

[2] James L. Crowley. *Systèmes Experts*. Support de cours, ENSIMAG 1999.

[3] AHenry Farrenry, Malik Ghallab. *Éléments d'Intelligence Artificielle*. HERMES Paris 1990.

[4] Joseph Giarratano, Gart Riley, *Expert System. Principles and Programming*. PWS Publishing Company, 1993.

[5] James P. Ignizio, *Introduction to Expert System. The development and Implementation of Rule-Based Expert System*, McGRAW-HILL 1991.

[6] Phan Huy Khánh. *Lập trình Prolog.* Nhà Xuất bản Đại học Quốc gia Hà Nội 2004.

[7] Phan Huy Khánh.*Lập trình hàm.* Nhà Xuất bản Khoa học và Kỹ tthuật 2004.

[8] Elaine Rich, Kevin Knight. *Artificial Intelligence.* International Edition, McGRAW-HILL 1991.

[9] Porter D. Sherman, Jhon C. Martin. *An OPS5 Primer. Introduction to Rule-Based Expert System.* Pretice Hall, 1990.

[10] Nguyễn Thanh Thuỷ. *Trí tuệ nhân tạo. Các phương pháp giải quyết vấn đề và kỹ thuật xử lý tri thức*. Nhà Xuất bản Giáo dục, 1996.

[11] Đỗ Trung Tuấn. *Hệ chuyên gia*. Nhà Xuất bản Giáo dục, 1999.

[12] Đỗ Trung Tuấn. *Trí tuệ nhân tạo*. Nhà Xuất bản Giáo dục, 1998.

[13] Trần Thành Trai. *Nhập môn hệ chuyên gia*. Trung tâm Khoa học Tự nhiên và Công nghệ Quốc gia, Phân viện CNTT, tpHCM, 1995.

[14] Tài liệu tham khảo trên internet
http://www.dockitsoft.com/index.htm
http://www.cas.utk.edu/utcc/user_services/users_guides/OpenVMS_guide/vms.html
http://yoda.cis.temple.edu:8080/UGAIWWW/lectures/rete.html#6
http://www.homeoint.org/articles/kaspar/jjk2dufr.htm#TelePC
http://www.homeoint.org/articles/kaspar/jjk2dufr.htm#TelePC