

บทที่ 6

State Space Graphs

6.1 โครงสร้างและกลวิธี (Structures and strategies)

เราได้พูดถึง predicate calculus ไปแล้วในแง่ของการใช้ predicate calculus เป็น repredication language ในงานด้าน AI Well-formed predicate calculus เป็นรูปแบบที่ดีที่จะใช้ในการอธิบายถึง objects และ relations ในปัญหาที่เรากำลังสนใจ และยังมี inference rules ซึ่งช่วยให้เราสามารถสร้างความรู้ใหม่ขึ้นมาได้จากรายละเอียดที่เรามีอยู่ โดยการอนุมานนี้จะมีการกำหนดขอบเขต (space) ซึ่งจะใช้ในการค้นหาข้อสรุป (solution) ของปัญหานั้น เราจะมาพูดถึง ทฤษฎี state space search.

ในการออกแบบและสร้างกลไกของการ search ให้ประสบผลสำเร็จนั้น โปรแกรมเมอร์จะต้องสามารถวิเคราะห์และทำนายพฤติกรรม (behavior) ของการ search นั้นได้ นั่นคือ ต้องมีการพิจารณารายละเอียดต่อไปนี้

problem solver นั้นจะให้ข้อสรุปได้จริงหรือเปล่า

problem solver นั้นต้องทำงานเสร็จจริง หรือติดอยู่ใน loop หรือเปล่า

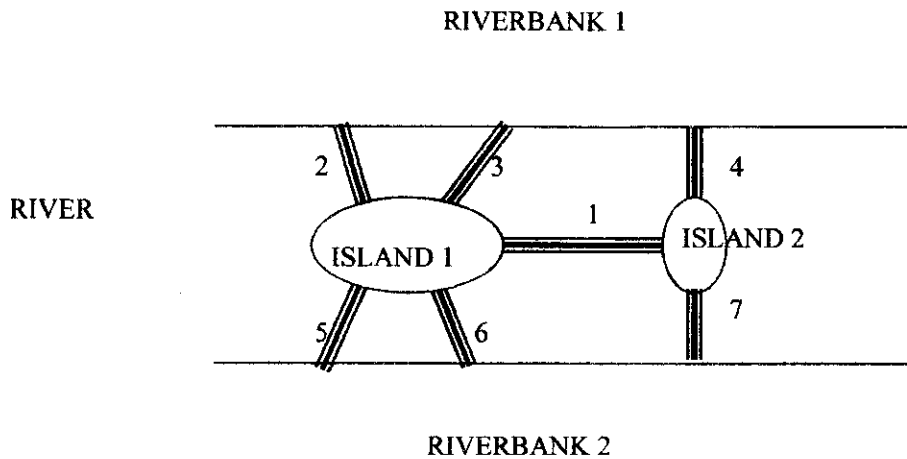
ข้อสรุป (solution) ที่ได้ จะต้องแน่ใจได้ว่าเป็น optimal solution

ความซับซ้อนของการ search โดยพิจารณาเวลาที่ใช้ หรือ หน่วยความจำที่ใช้

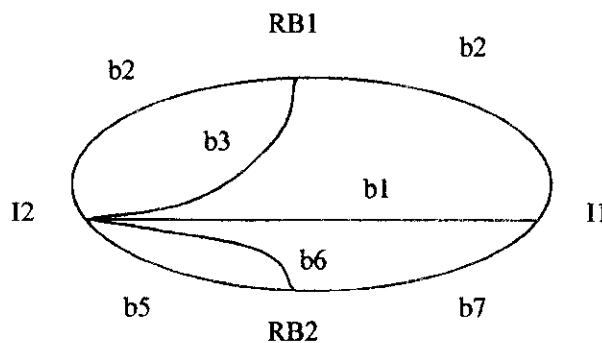
ทำอย่างไร interpreter จะทำให้การ search นั้นยุ่งยากน้อยลง

ทำอย่างไร interpreter จึงจะได้รับการออกแบบให้ทำงานกับ representation ได้ดี ที่สุด

ทฤษฎีของ state space search เป็นสิ่งที่จะนำมาใช้ตอบคำถามเหล่านี้ได้อย่างดีโดยเราจะมี การอธิบายปัญหาในรูปของ state space graph จากนั้นเราใช้ graph theory ในการวิเคราะห์โครงสร้างและความซับซ้อนของปัญหา และวิธีที่จะแก้ปัญหที่เกิดขึ้น graph theory เป็นเครื่องมือที่ดีที่สุดที่จะใช้ในการให้เหตุผลเกี่ยวกับ โครงสร้างของ object และ relations โดยการใช้กราฟนั้นเริ่มขึ้น จากปัญหา “bridges of Konigsberg” ใน คริสตศวรรษที่ 18



ปัญหาก็คือจะเป็นไปได้อย่างไรที่จะให้คนๆ หนึ่งเริ่มเดินจากจุดหนึ่ง แล้วเดินไปรอบเมือง โดยข้ามสะพานแต่ละแห่งเพียงครั้งเดียว แล้วกลับมายังที่เดิมได้
Euler ได้แสดงปัญหานี้ใหม่ในรูปของ graph



ในการพิจารณาเรื่องการเดิน Euler ได้พิจารณาจาก degree ของ node (คือจำนวน arc ที่มีในแต่ละ node นั้นเอง) ว่าเป็น odd หรือ even degree

odd degree node : คือ node ที่มีจำนวน arc เป็นเลขคี่

even degree node : คือ node ที่มีจำนวน arc เป็นเลขคู่จากการพิจารณา degree ของ node
 ทำให้ Euler สรุปได้ว่า การเดินรอบเมืองตามกฎเกณฑ์ที่กำหนดไว้ จะเป็นได้เมื่อ มีจำนวน odd degree node อยู่ใน graph เป็นจำนวน 0 หรือ 2 node นอกเหนือจากนี้การเดินทางดังกล่าวจะไม่มีวันสำเร็จ

ปัญหานี้ได้ถูกเรียกว่า **Euler path** :

เราสามารถใช้นำ predicate calculus แสดงปัญหา Königsberg ได้เช่นกัน โดย จะอยู่ในรูปของ

CONNECT(I1,I2,b1)

CONNECT(I1,I2,b1)

CONNECT(RB1,I1,b2)

CONNECT(I1,RB1,b2)

CONNECT(RB1,I1,b3)

CONNECT(I1,RB1,b3)

CONNECT(RB1,I2,b4)

CONNECT(I2,RB1,b4)

CONNECT(RB2,I1,b5)

CONNECT(I1,RB2,b5)

CONNECT(RB2,I1,b6)

CONNECT(I1,RB2,b6)

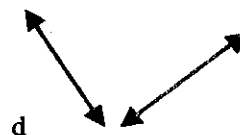
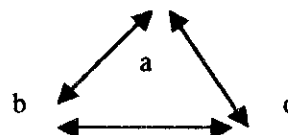
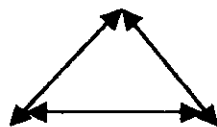
CONNECT(RB2,I2,b7)

CONNECT(I2,RB2,b7)

ทฤษฎีกราฟ

กราฟ (graph) ประกอบด้วย set ของ nodes และ set arcs ซึ่งทำหน้าที่เชื่อม node 2 ตัวเข้าด้วยกัน ในการสร้าง state space model ในการแก้ปัญหา node ของกราฟ จะหมายถึง state ที่จะเกิดขึ้นในการทำ problem-solving ส่วน arc จะหมายถึง ทางเดินจาก state หนึ่งไปยังอีก state หนึ่งซึ่งจะเกิดขึ้นตาม logical inference ของปัญหานั้นนั่นเอง

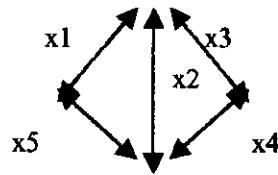
Labeled graph : จะหมายถึง กราฟที่มีการให้รายละเอียดแก่ node ที่มีอยู่ในกราฟ



unlabeled graph

labeled graph

the arcs of the graph may be labeled



Directed graph : จะหมายถึง กราฟที่มีการให้ทิศทางกับ arc โดยการใช้ลูกศรเป็นสิ่งที่กำหนดทิศทาง

Path : หมายถึง ทางเดินของกราฟจากจุดกำหนดไปยังจุดปลายทาง เช่น $[a,b,c,d]$

Rooted graph : คือกราฟที่มี node หนึ่งทำหน้าที่เป็น root คือเป็น node ที่ทำให้เกิด path ไปยัง node อื่นๆ ในกราฟ

Tree : คือ กราฟซึ่งมี arc เพียงเส้นเดียวเชื่อมระหว่าง node 2 ตัว

Connected graph : คือ กราฟที่มี node 2 ตัวมี path เชื่อมกันได้

6.2 State space representation of problems

ในการแสดงปัญหาที่เกิดขึ้น โดยใช้ state space graph นั้น

node ของกราฟจะหมายถึง state ที่เป็นวิธีการแก้ปัญหาอย่างหนึ่ง ส่วน arc จะหมายถึง ขั้นตอนในการแก้ปัญหา

initial state จะหมายถึง information ที่กำหนดให้ในตอนเริ่มต้นซึ่งทำหน้าที่เป็น root และ graph ยังระบุ goal condition ซึ่งก็คือ solution ของปัญหานั้นเอง

State space search ก็คือ กระบวนการในการหา solution path จาก state space ไปยัง goal นั้นเอง

state space จะประกอบด้วยองค์ประกอบ 4 ส่วนคือ $[N,A,S,GD]$

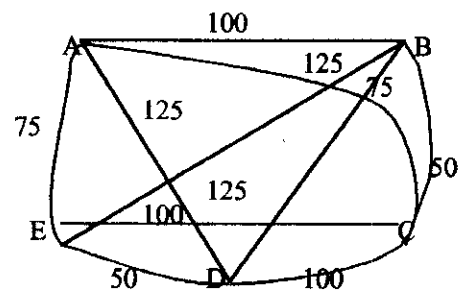
N – คือ set ของ nodeS ที่มีในกราฟ

A – คือ set ของ arcs ที่เชื่อม nodes ในกราฟ

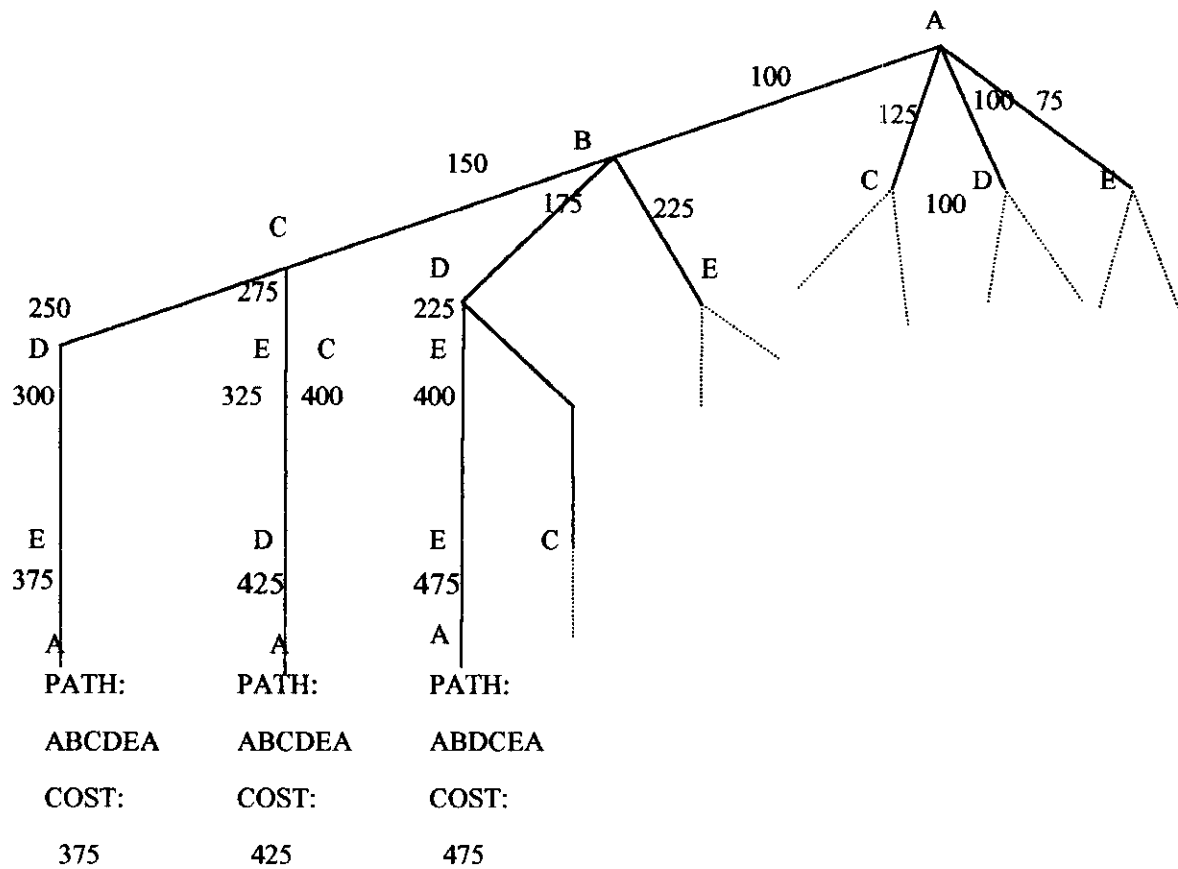
S – คือ subset ของ N ที่มี start state เป็นสมาชิก (ต้องไม่เป็น empty)

Solution path : คือ ทางเดินที่เริ่มจาก node ใน S ไปยัง node ใน GD

ตัวอย่าง : traveling salesman



an instance of the problem:



ตัวอย่างรูปนี้แสดงให้เห็นถึง exhaustive search ก็คือต้อง search ถึง $(N-1)!$ (N คือจำนวนเมืองที่มีกราฟ)

6.3 Strategies for state space search

การ search ใน state space ทำได้ 2 แนวทาง นั่นคือ การเริ่มจาก data ที่มีอยู่ มุ่งไปหา goal ที่ต้องการ หรือ จาก goal เดิมกลับมาหา data

data-driven search : เรียกอีกอย่างว่า **backward chaining**

เราเริ่มจากการพิจารณา goal นี้ และมีเงื่อนไข (condition) ใดบ้างที่จะทำให้กฎ เหล่านั้นเป็นจริง conditions ดังกล่าวจะกลายมาเป็น goal ใหม่ที่เรียกว่า subgoal ซึ่งจะนำไปใช้ในการหา subgoal ตัวต่อไป จนกระทั่งเราย้อนกลับไปถึง data ของปัญหาที่เรากำลังแก้อยู่ ทั้ง data-driven search และ goal-driven search นี้ถึงแม้จะมีวิธีการที่ ต่างกันแต่ก็ใช้ใน state space เดียวกัน การเลือกใช้ search นั้นจะพิจารณาจากลักษณะของปัญหา, ความยุ่งยากของกฎที่ใช้, รูปร่างของ state space และความพร้อมของ fact ที่จะใช้ในการแก้ปัญหา (ซึ่งจะแตกต่างกันไปตามชนิดของปัญหา)

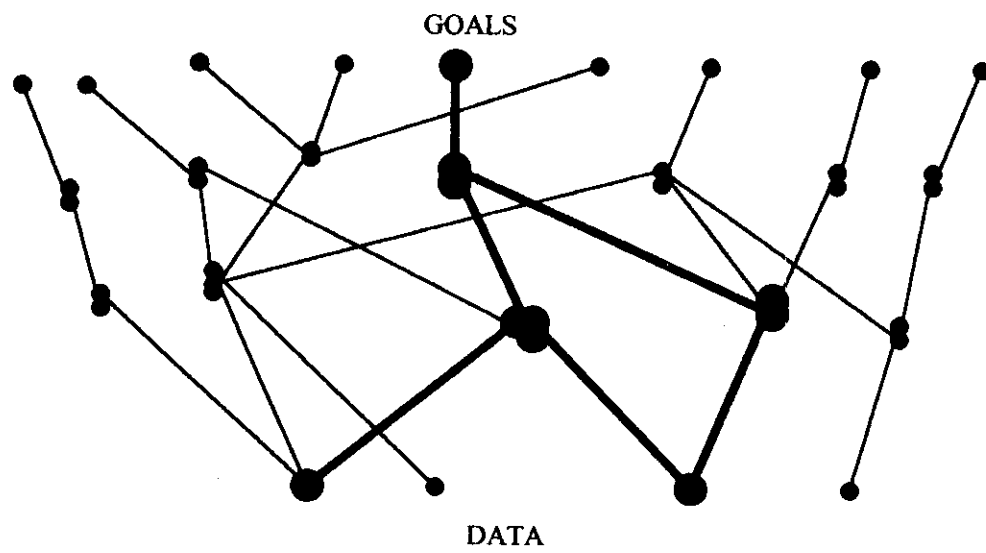
goal-driven search จะเหมาะแก่การใช้เมื่อ

1. มีการระบุ goal ไว้ใน problem statement แล้ว หรือ goal นั้นสามารถ นำมากำหนดเป็น formula ได้ง่าย (เพราะ goal ก็คือ ทฤษฎีที่ต้องการ การพิสูจน์นั่นเอง)
2. มีกฎ (rule) หลายกฎที่ตรงกับ facts ที่มีใน problem และสามารถนำมาใช้ สร้าง goal ได้ง่ายขึ้น
3. ไม่มีการกำหนด problem data มาให้

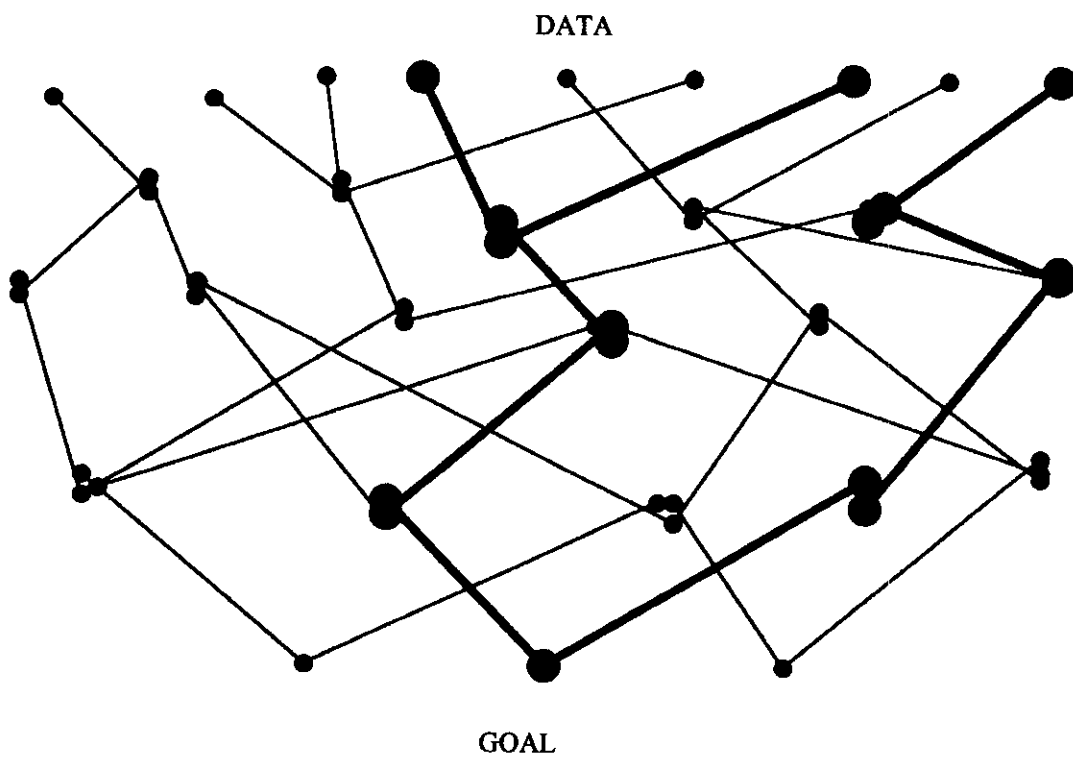
ส่วน **data-driven approach** นั้นจะเหมาะแก่การใช้เมื่อ

1. มีการกำหนด data ทั้งหมด หรือเกือบทั้งหมดให้กับ problem statement
2. มี goal ที่อาจเป็นไปได้จำนวนมาก แต่มีแนวทางที่จะใช้ fact เพื่อจะนำมาซึ่ง ข้อมูลที่ต้องการ นั้นน้อย
3. การกำหนด goal นั้นทำได้ยาก

GOAL-DRIVEN



DATA-DRIVEN



ในการแก้ปัญหานั้นไม่ว่าจะเป็นการใช้ goal-driven search หรือ data-driven search ก็ตาม ผู้แก้ปัญหามองหา path ที่ถูกต้องใน state space graph ซึ่งก็คือ path ที่เดินจาก start state ไปยัง goal state นั้นเอง แต่การที่จะหา path นี้พบในการพยายามครั้งแรกเลยนั้นเป็นไปได้ยาก ดังนั้นจึงเกิดกรณีที่เรากลับไปตาม path หนึ่งแล้วพบว่า path ที่ไม่ถูกต้อง เราเลยต้องย้อนกลับมาเริ่มใหม่ใน path ใหม่ที่เราคาดว่าจะจะเป็น path ที่ถูกต้อง ซึ่งก็อาจเป็นกรณีที่เรายังต้องย้อนกลับไปเริ่มใหม่อีกก็ได้เช่นกัน เราจะทำเช่นนี้ไปเรื่อยๆ จนพบ path ที่ถูกต้องจริงๆ

backtracking คือ เทคนิคที่ใช้พิจารณาเรื่องการย้อนกลับไปเริ่มต้นใหม่ในการค้นหา path แต่เป็นการทำงานอย่างมีรูปแบบที่ถูกต้อง โดยจะเริ่มจาก start state จนพบ goal state หรือไม่ก็ dead end เราเรียกลักษณะการค้นหา solution path นี้ว่า **backtracking search**

อัลกอริทึม ของการทำ backtracking มีดังนี้

กำหนดให้

- SL – เป็น State List คือ list ที่เก็บ state ที่กำลังถูกค้นหา path ถ้าพบ goal สิ่งที่อยู่ใน SL ก็คือบรรดา state ที่เป็น solution path นั้นเอง
- NSL – New State List – เก็บ state ที่กำลังรอการตรวจสอบ
- DE – Dead End – คือ List ของ state ที่ไม่สามารถไปถึง goal ได้
- CS – Current State – คือ state ที่กำลังถูกพิจารณา

FUNCTION BACKTRACK

BEGIN

SL := [START]; NSL := [START]; DE := []; CS := START; %INITIALZE:

WHILE NSL $\neq$ [] DO

 BEGIN

 IF CS = GOAL THEN RETURN (s1);

 IF CS HAS N CHILDREN (EXCLUDING nodes ALLREADY IN DE,SL,AND NSL
 THEN

 BEGIN

 WHILE SL IS NOT EMPTY AND CS = FIRST ELEMENT OF SL DO

 BEGIN

 ADD CS TO DE;

 REMOVE FIRST ELEMENT FROM SL;

 REMOVE FIRST ELEMENT FROM NSL;

 CS := FIRST ELEMENT OF NSL;

 END;

 ADD CS TO SL;

 END

 ELSE

 BEGIN

 PLACE CHILDREN OF CS (EXCEPT NODES ALLREADY ON DE,SL,NSL) ON NSL;

 CS := FIRST ELEMENT OF NSL;

 ADD CS TO SL

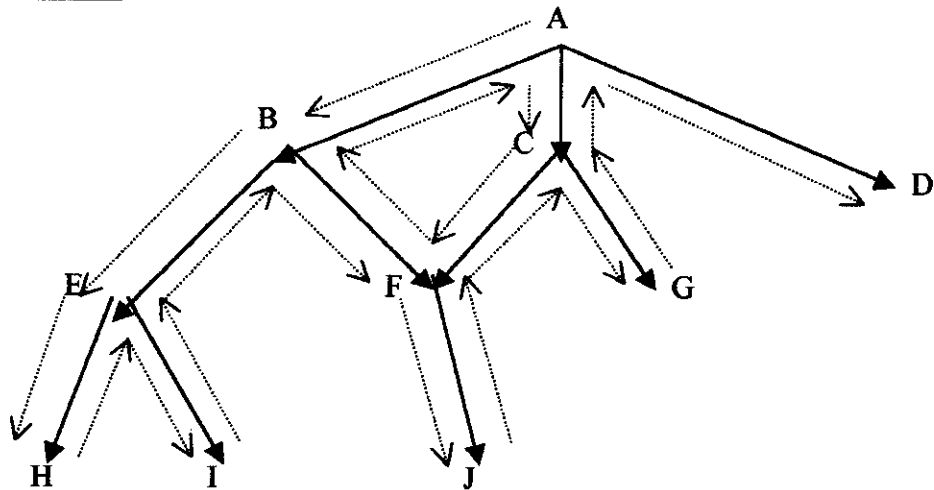
 END

 END

 RETURN FAIL

END.

ตัวอย่าง



INITIAL : SL = [A]; NSL = [A]; DE = []; CS = A

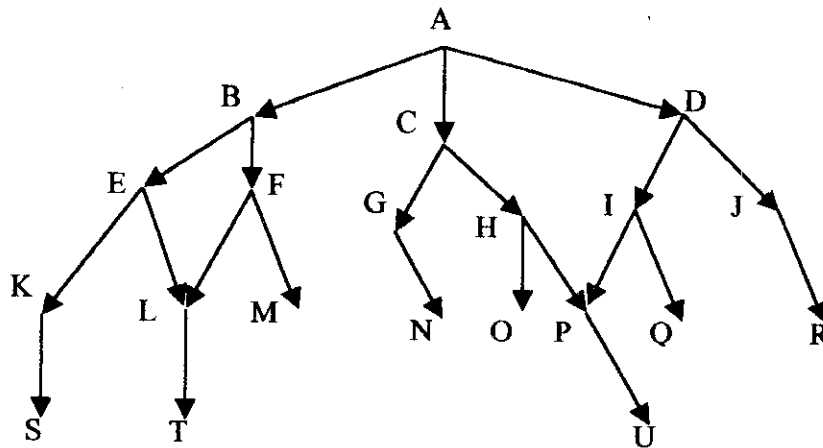
AFTER ITERATION	CS	SL	NSL	DE
0	A	[A]	[A]	[]
1	B	[B,A]	[B,C,D,A]	[]
2	E	[E,B,A]	[E,F,B,C,D,A]	[]
3	H	[H,E,B,A]	[H,I,E,F,B,C,D,A]	[]
4	I	[I,E,B,A]	[I,E,F,B,C,D,A]	[H]
5	F	[F,B,A]	[F,B,C,D,A]	[E,I,H]
6	J	[J,F,B,A]	[J,F,B,C,D,A]	[E,I,H]
7	C	[C,A]	[C,D,A]	[B,F,J,E,I,H]
8	G	[G,C,A]	[G,C,D,A]	[B,F,J,E,I,H]

backtrack คือว่าเป็น basic algorithm สำหรับการค้นหา state space graph

Depth-and breath-first search

ในการ search นั้นนอกจากจะมีการกำหนดทิศทางในการค้นหาแบบ data-driven หรือ goal-driven search แล้วเรายังเพิ่มกลไกในการ search ให้แน่นอนขึ้นได้อีกโดยการพิจารณาจากลำดับของการพิจารณา state ที่อยู่ใน state space graph ของเราทำให้เกิดวิธีการค้นหาแบบ depth-first และ breath-first search

ตัวอย่าง



Depth-first search :

A,B,E,K,S,L,T,F,M,C,G,N,H,O,P,U,D,I,Q,J,R

Breath-first search :

A,B,C,D,E,F,G,H,I,J,K,L,M,N,O,P,Q,R,S,T,U

จะเห็นว่า algorithm ของการทำ backtrack สามารถนำมาใช้กับการค้นหาแบบ depth-first search ได้ algorithm ของการทำ breath-first-search นั้นสามารถทำได้โดยการใช้ list 2 ตัว (open และ close) เก็บรายละเอียดเกี่ยวกับ state ในขณะที่มีการ search เข้าไปใน state space

OPEN – เป็น list ที่เก็บ state ที่ถูกตรวจสอบ

CLOSE – เป็น list ที่เก็บ state ที่ผ่านการตรวจสอบแล้ว

PROCEDURE BREADTH_FIRST_SEARCH

INITIAL : OPEN = [START]; CLOSE = []

WHILE OPEN $\neq$ [] DO

BEGIN

REMOVE THE LEFT-MOST START FROM OPEN, CALL IT X;

IF X IS GOAL THEN RETURN(SUCCESS);

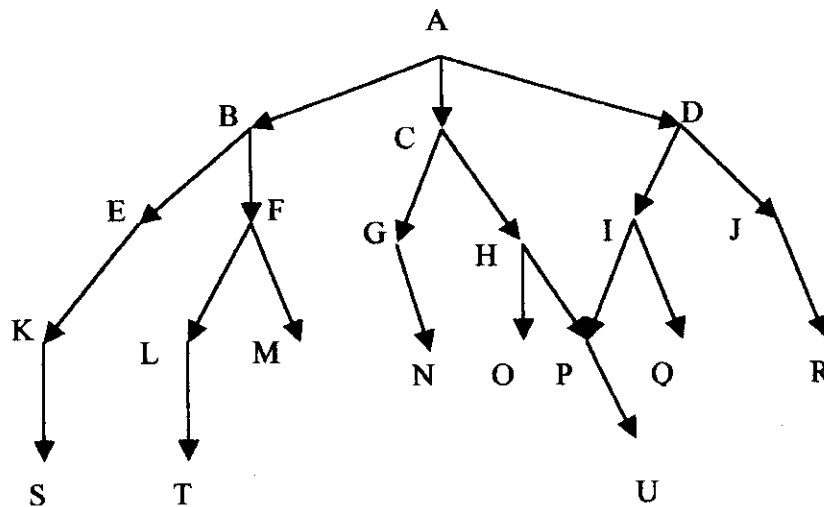
GENERATE ALL CHILDREN OF X;

PUT X ON CLOSE;

ELIMINATE ANY CHILDREN OF X ALLREADY ON EITHER OPEN OR CLOSE,
AS THESE WILL CAUSE LOOPS IN THE search;

PUT THE RAMANING DESCENDENTS, IN ORDER OF DISCOVERY, ON THE
RIGHT END OF OPEN;

END.



TRACE: BREADTH_FIRST_SEARCH

1. OPEN = [A]; CLOSE = []
2. OPEN = [B,C,D]; CLOSE = [A]
3. OPEN = [C,D,E,F]; CLOSE = [B,A]
4. OPEN = [D,E,F,G,H]; CLOSE = [C,B,A]

5. OPEN = [E,F,G,H,I,J]; CLOSE = [D,C,B,A]
6. OPEN = [F,G,H,I,K,L]; CLOSE = [E,D,C,B,A]
7. OPEN = [G,H,I,J,K,L,M]; CLOSE = [F,E,D,C,B,A]
8. OPEN = [H,I,J,K,L,M,N]; CLOSE = [G,F,E,D,C,B,A]
9. :
ทำเช่นนี้ไปเรื่อยๆ จนกระทั่งพบ goal หรือ OPEN = []

PROCEDURE DEPTH_FIRST_SEARCH

INITIAL : OPEN = [START]; CLOSE = []

WHILE OPEN $\neq$ [] DO

BEGIN

REMOVE THE NEXT state FROM THE LEFT OF OPEN, CALL IT X;

IF X IS GOAL THEN RETURN(SUCCESS);

GENERATE ALL POSSIBLE CHILDREN OF X;

PUT X ON CLOSE;

ELIMINATE ANY CHILDREN OF X ALREADY ON EITHER OPEN OR CLOSE,
AS THESE WILL CAUSE LOOPS IN THE search;

PUT THE REMAINING CHILDREN OF X, IN ORDER OF DISCOVERY, ON THE
LEFT END OF OPEN

END.

TRACE :

1. OPEN = [A]; CLOSE = []
2. OPEN = [B,C,D]; CLOSE = [A]
3. OPEN = [E,F,C,D]; CLOSE = [B,A]
4. OPEN = [K,L,F,C,D]; CLOSE = [E,B,A]
5. OPEN = [S,L,F,C,D]; CLOSE = [K,E,B,A]
6. OPEN = [L,F,C,D]; CLOSE = [S,K,E,B,A]

7. OPEN = [T,F,C,D]; CLOSE = [L,S,K,E,B,A]
 8. OPEN = [F,C,D]; CLOSE = [T,L,S,K,E,B,A]
 9. OPEN = [M,C,D]; CLOSE = [F,T,L,S,K,E,B,A]
 10. OPEN = [C,D]; CLOSE = [M,F,T,L,S,K,E,B,A]
 11. OPEN = [G,H,D]; CLOSE = [C,M,F,T,L,S,K,E,B,A]
 :
 UNTIL OPEN = [] OR GOAL IS FOUND

6.4 And/Or graph

ใน state space graph ที่เราสร้างขึ้นมานั้น node แต่ละตัวจะหมายถึง state ที่อาจทำให้เราไปหาข้อสรุปของปัญหาได้ จากนั้นเราสามารถใช้ predicate calculus แสดงให้เห็นถึง state ต่างๆ ที่จะเป็นองค์ประกอบของกระบวนการแก้ปัญหาได้ ส่วน inference rule ก็คือ arc ที่เชื่อม node ในกราฟนั่นเอง ดังนั้น ปัญหาที่อยู่ในรูปของ predicate calculus อาจจะหา solution ได้โดยการ search เช่นกัน

ตัวอย่าง ของการแสดง proposition calculus ในรูปของ graph

$$q \Rightarrow p$$

$$r \Rightarrow p$$

$$v \Rightarrow q$$

$$s \Rightarrow r$$

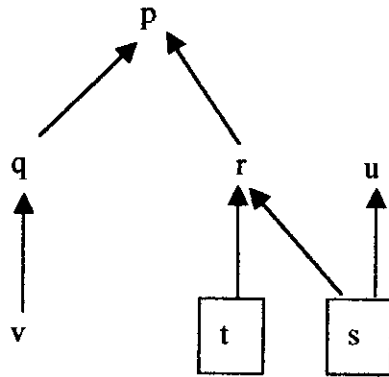
$$t \Rightarrow r$$

$$s \Rightarrow u$$

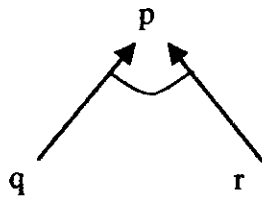
s

t

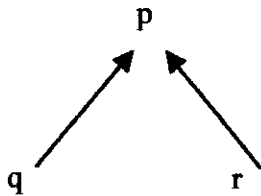
จะสามารถแสดงในรูปกราฟได้เป็น



ซึ่งกราฟนี้จะแสดงถึงความสัมพันธ์ในรูปของ implication $\Rightarrow$ ซึ่งอยู่ในรูปแบบของ directed graph และการ search จะเป็นแบบ data-driven search ในความเป็นจริง เรายังมี and และ or operation ที่ควรจะแสดงได้เช่นกันในรูปของ graph จึงได้มีการกำหนด **And/Or graph** ขึ้น ดังรูปแบบต่อไปนี้

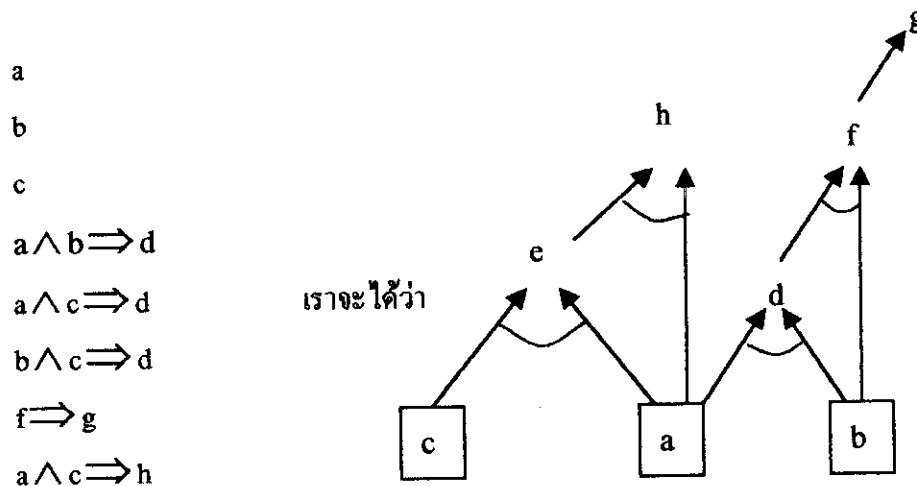


หมายถึง $q \wedge r \Rightarrow p$



หมายถึง $q \vee r \Rightarrow p$

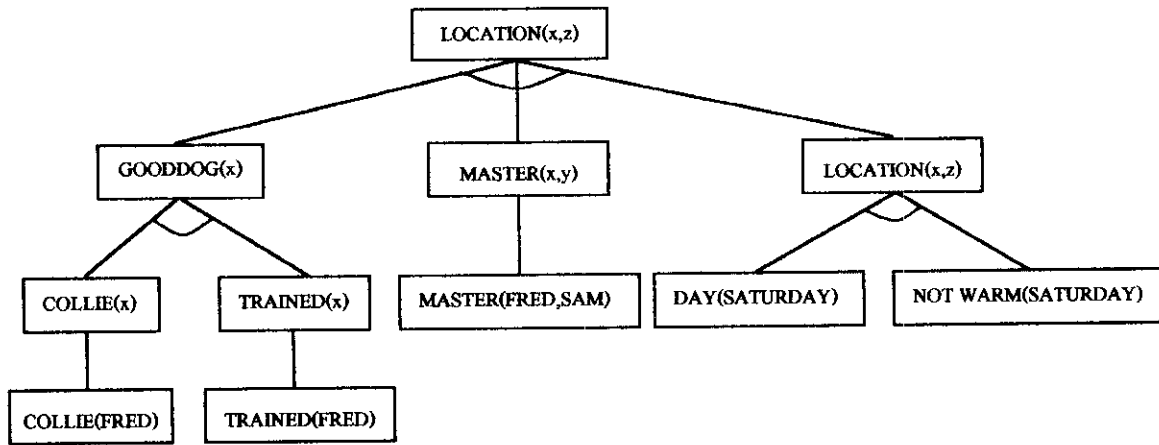
ตัวอย่าง ของการใช้ And/Or graph โดยพิจารณา proposition calculus
ต่อไปนี้



ตัวอย่าง การสร้าง And/Or graph จาก predicate calculus ต่อไปนี้

1. FRED IS A COLLIE. : COLLIE(FRED).
2. SAM IS FRED'S MASTER. : MASTER(FRED,SAM).
3. IT IS SATURDAY. : DAY(SATURDAY).
4. IT IS COLD ON SATURDAY. : $\neg$ WARM(SATURDAY).
5. FRED IS A TRAINED DOG. : TRAINED(FRED).
6. SPANIELS OR COLIES THAT ARE TRAINED ARE GOOD DOGS. :
 $\forall x [\text{SPANIEL}(x) \vee (\text{COLLIE}(x) \wedge \text{TRAINED}(x)) \Rightarrow \text{GOODDOG}(x)]$.
7. IF A DOG IS GOOD DOG AND HAS A MASTER THEN HE WILL BE WITH HIS
 MASTER. : $\forall x \forall y \forall z [\text{GOODDOG}(x) \wedge \text{MASTER}(x,y) \wedge \text{LOCATION}(y,z)$
 $\Rightarrow \text{LOCATION}(x,z)]$.
8. IF IT IS SATURDAY AND WARM, THEN SAM IS AT THE PARE. :
 $\text{DAY}(\text{SATURDAY}) \wedge \text{WARM}(\text{SATURDAY}) \Rightarrow \text{LOCATION}(\text{SAM},\text{PARK})$.
9. IF IT IS SATURDAY AND NOT WARM, THEN SAM IS AT THE MUSEUM.
 $\text{DAY}(\text{SATURDAY}) \wedge \neg \text{WARM}(\text{SATURDAY}) \Rightarrow \text{LOCATION}(\text{SAM},\text{MUSEUM})$.

goal ของปัญหานี้อยู่ในรูป expression ของ $\exists x \text{LOCATION}(\text{FRED}, x)$ หมายถึง “FRED อยู่ที่ไหน” นั่นเอง การ search ของปัญหานี้จะใช้ And/Or graph และจะใช้ goal-driven search



SUBSTITUTIONS = {FRED/x,SAM,y,MUDEUM/z}

6.5 Production Systems:PS

PS เป็น computation model ที่Newell ได้เสนอแนวความคิดเกี่ยวกับเรื่อง PS ขึ้นราวปี ค.ศ. 1967 เพื่อใช้เป็นเครื่องมือในการศึกษาถึงกระบวนการคิดของมนุษย์ ปัจจุบัน PS ได้กลายมาเป็น knowledge representation scheme ที่ได้รับความนิยมอย่างมากในงานด้าน expert system

โครงสร้างของ PS

PS ประกอบด้วยองค์ประกอบสำคัญ 3 ส่วน คือ

1. Production memory:PM

PM เก็บรายละเอียดเกี่ยวกับความรู้ โดยความรู้ดังกล่าวจะถูกอธิบายไว้ในรูปของ productions หรือ rules ซึ่งมีรูปแบบเป็น

“CONDITION $\rightarrow$ CONCLUSION”

ส่วนของ condition ของ production นี้บางทีก็เรียกว่า assumption, Left-Hand Side, LHS part หรือ if part โดย condition นี้จะอธิบายถึง pre-condition ที่จะต้องเกิดขึ้น ก่อนที่จะมีการนำ production ไปใช้

ส่วนของ conclusion ของ production นี้บางครั้งก็เรียกว่า action part, Right-Hand Side part, RHS part หรือ then part โดย condition นี้จะอธิบายถึงการทำงานขององค์ประกอบของ production เช่น การแสดง output หรือ การเก็บข้อมูลเข้าไปใน working memory

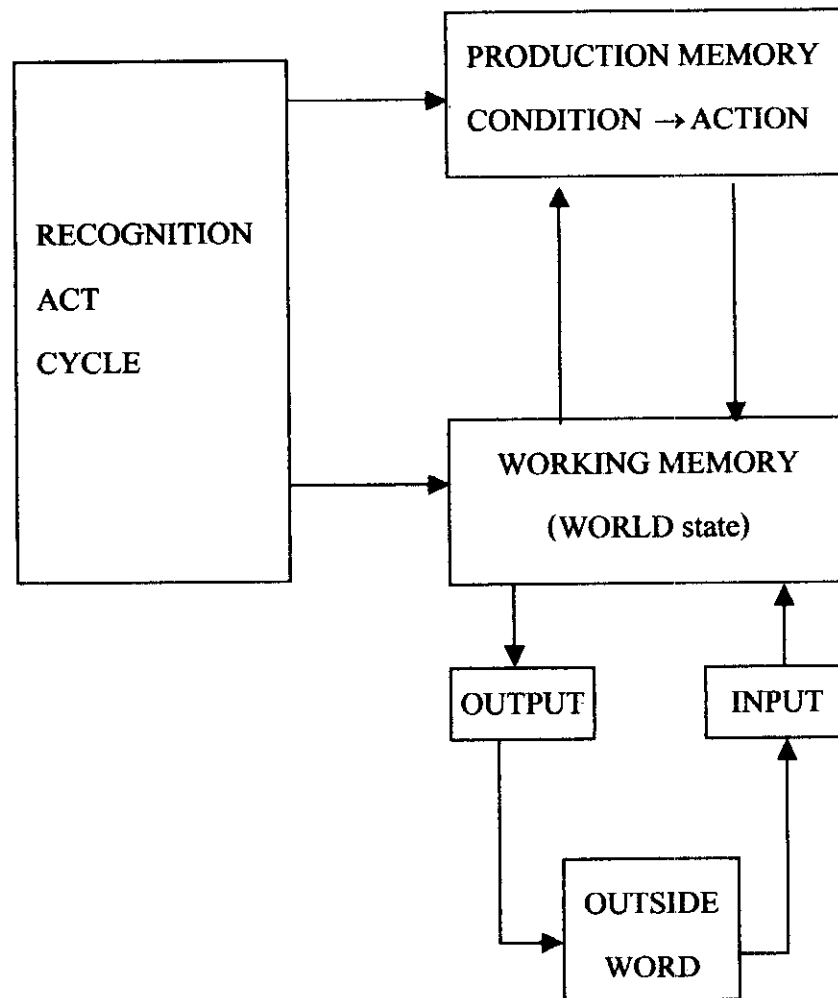
2. Working Memory: WM

WM เก็บข้อมูลซึ่งถูก PM เรียกไปใช้หรือข้อมูลที่ถูก production เปลี่ยนแปลงไป WM เรียกอีกอย่างได้ว่า “database” หรือ “context”

3. Recognize-act cycle :

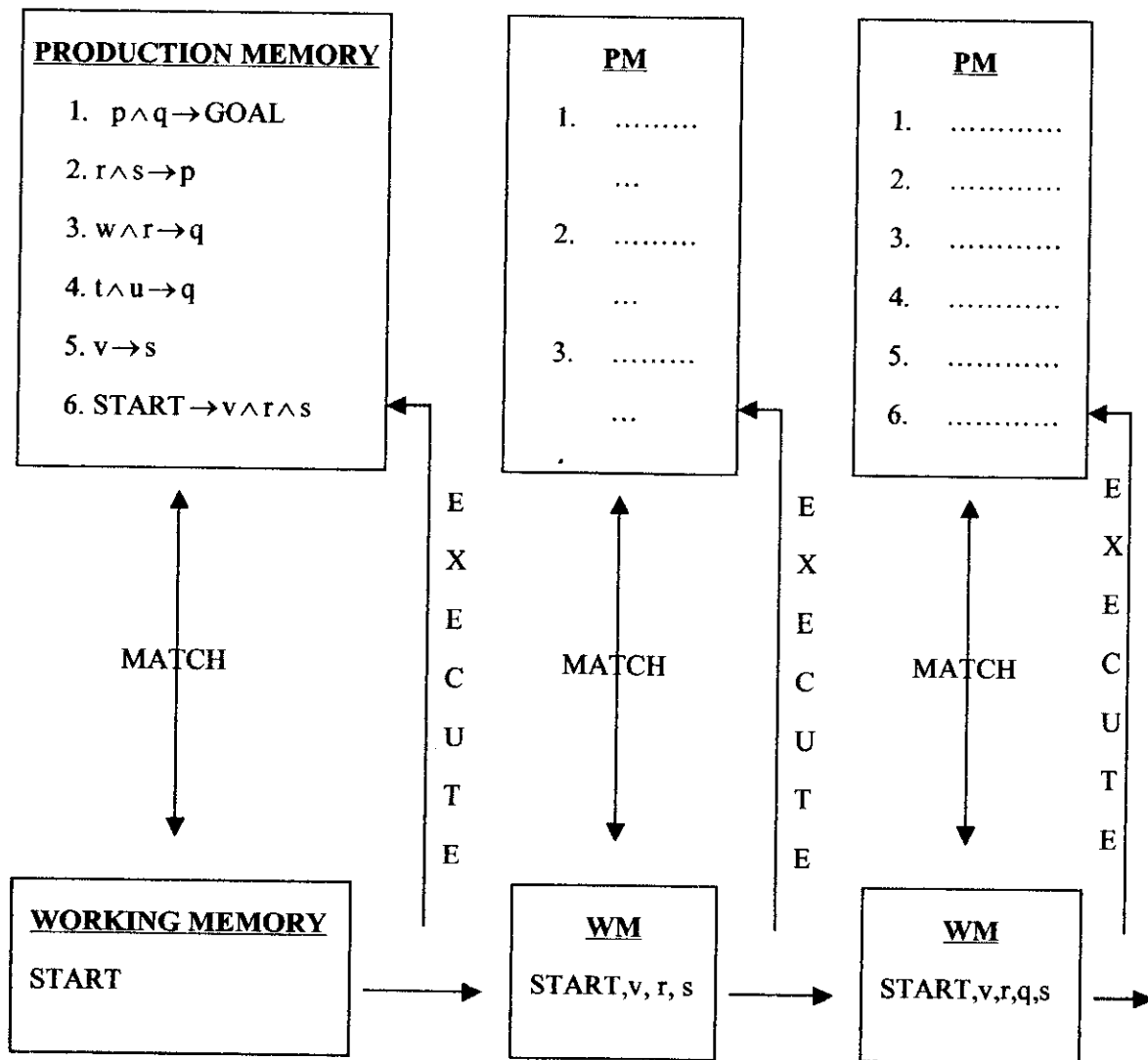
เป็นส่วนควบคุมการทำงานของ PS โดยการควบคุมจะมีรูปแบบดังนี้

- current state ของกระบวนการแก้ปัญหาจะถูกเก็บไว้ใน WM
- WM จะเป็นตัวเก็บขั้นตอนเริ่มต้นที่เกิดขึ้นในการแก้ปัญหา
- รูปแบบที่มีใน WM จะถูกนำไปหาความสัมพันธ์กับ productions ที่อยู่ใน PM ซึ่งทำให้เกิด **conflict set** คือ subset ของ productions ที่สัมพันธ์กับสิ่งที่อยู่ใน WM
- productionS ที่อยู่ใน conflict set จะถูกกำหนดให้เป็น enable
- productionS ใน conflict set ตัวใดตัวหนึ่งจะถูกเลือกมาใช้ในการทำงาน ซึ่งเรียกว่าเป็นการ fired นั่นคือ ส่วน conclusion ของ production ที่ถูก **fired** จะเกิดการทำงาน ซึ่งทำให้เกิดการเปลี่ยนแปลงรายละเอียดใน WM
- การทำงานในลักษณะนี้จะเกิดขึ้นเป็นรอบๆ ไปจนกระทั่งไม่มี production ใดที่จะสัมพันธ์กับรายละเอียดที่มีใน WM จึงหยุดการทำงาน



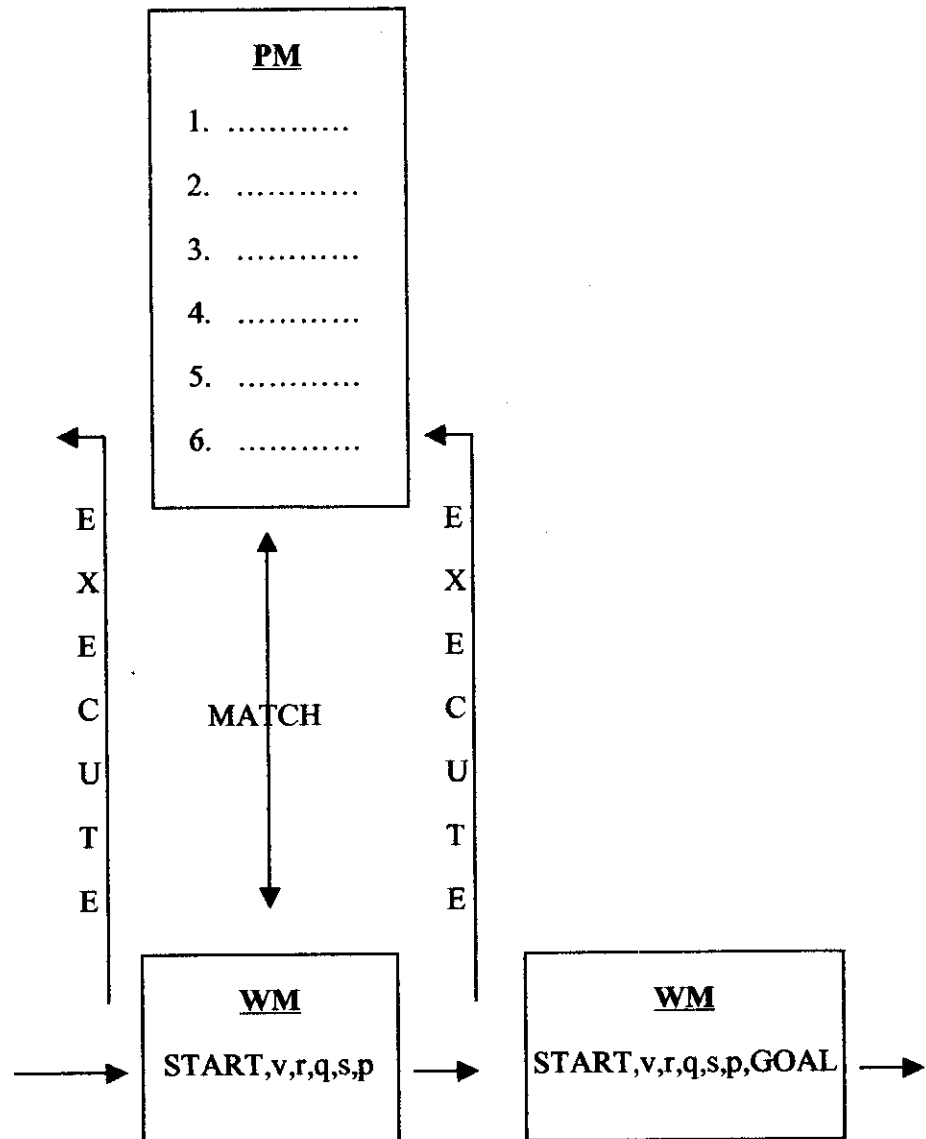
- รูปแบบโครงสร้างของ PS

ตัวอย่าง การทำ data-driven search ใน PS

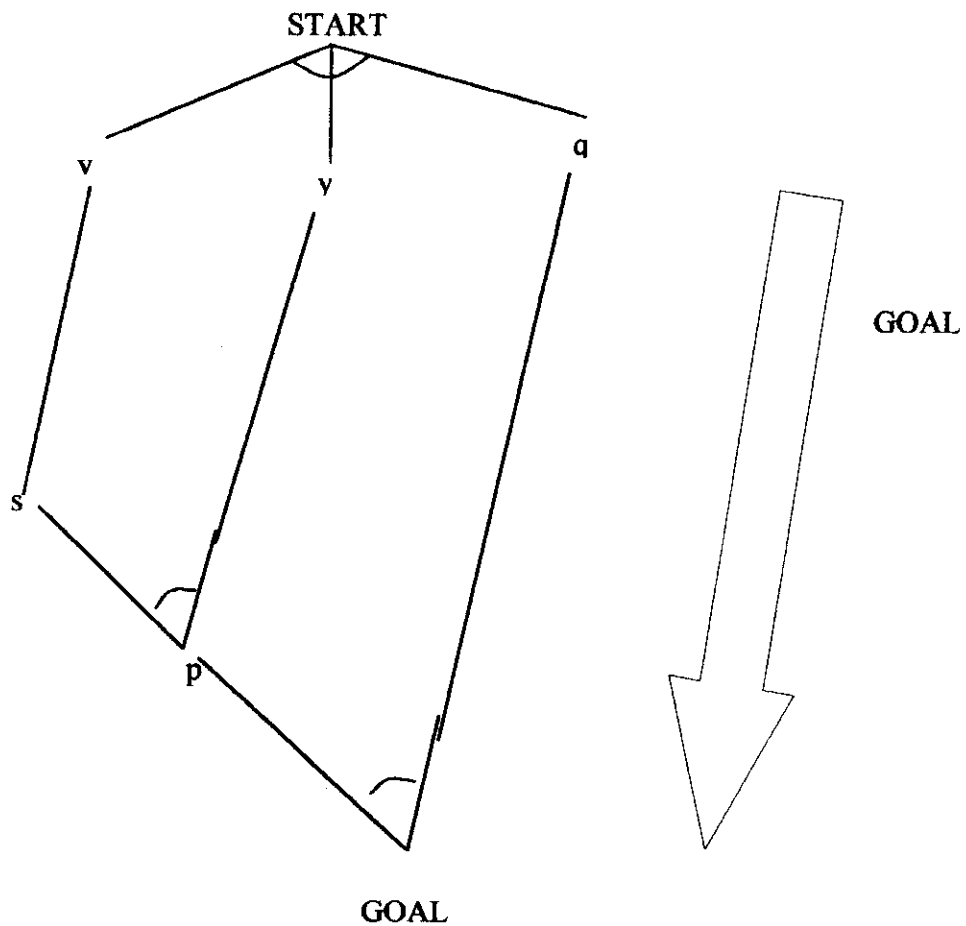


TRACE EXECUTION :

ITERATION#	WM	CONFLICT SET	RULE FIRED
0	START	6	6
1	START,v,r,q	6,5	5
2	START,v,r,q,s	6,5,2	2
3	START,v,r,q,s,p	6,5,2,1	1
4	START,v,r,q,p,GOAL	6,5,2,1	HALT



Space searched by execution :



ตัวอย่าง การทำ goal-driven search ใน PS

PRODUCTION SET

1. $p \wedge q \rightarrow \text{GOAL}$
2. $r \wedge s \rightarrow p$
3. $w \wedge r \rightarrow q$
4. $t \wedge u \rightarrow q$
5. $v \rightarrow s$
6. $\text{START} \rightarrow v \wedge r \wedge s$

TRACE OF EXECUTION :

ITERATION#	WORKING MEMORY	CONFLICT SET	RULE FIRED
0	GOAL	1	1
1	GOAL,p,q	1,2,3,4	2
2	GOAL,p,q,r,s	1,2,3,4,5	3
3	GOAL,p,q,r,s,w	1,2,3,4,5	4
4	GOAL,p,q,r,s,w,t,u	1,2,3,4,5	5
5	GOAL,p,q,r,s,w,t,u,v	1,2,3,4,5,6	6
6	GOAL,p,q,r,s,w,t,u,v, START	1,2,3,4,5,6	HALT

บทที่ 7

PROLOG

7.1 ความเป็นมาของภาษาโปรล็อก

ในทศวรรษหลังสุดนี้ราคาของ Hardware ลดลงครึ่งหนึ่งทุกๆ 4 ปี ขณะที่ราคาของ Software เพิ่มขึ้นตลอด และในปัจจุบันค่าใช้จ่ายส่วนใหญ่ของระบบจะเป็นของ Software ราคาของ Software จะประมาณ 10 % ของระบบทั้งหมดในปี 1970 , 50 % ในปี 1975 และมากกว่า 80 % ในปี 1985 ปัจจุบันเดี่ยวนี้นับเพิ่มขึ้นอย่างฉับพลัน

ผลงานของการวิจัยหลายปีก็คือ โปรล็อก โปรล็อกที่เป็นทางการรุ่นแรกถูกพัฒนาโดยมหาวิทยาลัย มาร์เซิล ประเทศฝรั่งเศส โดยอเลกซันดร์ คอมเมอร์เรอร์ ในตอนต้นของทศวรรษ 1970 เป็นเครื่องมืออำนวยความสะดวกสำหรับการเขียนโปรแกรมในแนวทางตรรกศาสตร์ โปรล็อกมีสมรรถภาพและประสิทธิภาพมากกว่าภาษาอื่นๆ เช่น ภาษาปาสคาล และ ภาษาเบสิก คำว่าโปรล็อกมาจาก PROgramming in LOGic

ในวันนี้ โปรล็อกเป็นเครื่องมือที่สำคัญในการพัฒนางานด้านปัญญาประดิษฐ์ และงานด้านระบบผู้เชี่ยวชาญ ซึ่งง่ายต่อผู้ใช้ และ โปรล็อกเป็นโปรแกรมที่ชาญฉลาด ทำให้ โปรล็อกเป็นที่นิยม

ไม่เหมือนตัวอย่างใน Pascal โปรแกรมภาษาโปรล็อกจะให้รายละเอียดของปัญหาโดยใช้จำนวนของกฎเกณฑ์และความจริง และถ้าถึงปัญหาที่ต้องการคำตอบที่ต้องการค้นหาคำตอบโดยเอากฎเกณฑ์และความจริงมาแสดง

ใน Pascal มักจะแสดงวิธีการทำงานที่แน่นอน แต่โปรแกรมเมอร์ภาษาโปรล็อกจะอธิบายว่าอะไรที่ต้องทำ ระบบของโปรล็อกจะจัดการว่าจะทำอะไรเพราะว่าการกำหนดจุดหมาย ซึ่งถ้ากำหนดในภาษาปาสคาลและภาษาเบสิกก็จะเกิดข้อผิดพลาด

โปรล็อกจะสอนโปรแกรมเมอร์ให้สร้างโครงสร้างของรายละเอียดของปัญหาต่างๆ ให้ดี เพื่อให้โปรล็อกสามารถทำได้เสมือนเครื่องมือเฉพาะด้าน

ในปี 1980 ทางการญี่ปุ่นได้ประกาศแผนงานออกแบบและสร้างคอมพิวเตอร์ในยุค ที่ 5 โดยใช้โปรล็อกเป็นภาษาพื้นฐานในการพัฒนาเทคโนโลยีทางด้าน Hardware และ Software

Turbo Prolog สามารถทำอะไรได้บ้าง

- Prototype ของโปรแกรมประยุกต์ จุดเริ่มต้นของการนำความคิดไปใช้กับชีวิตจริง
- ควบคุมและตรวจสอบกระบวนการในอุตสาหกรรม Turbo Prolog จัดเตรียมการติดต่อกับ I/O Ports ที่สมบูรณ์
- สามารถนำไปใช้กับฐานข้อมูลเชิงสัมพันธ์ประเภทพลวัต
- ตัวแปลภาษาเป็นแบบภาษามนุษย์
- สร้าง Software ของการติดต่อกับภาษาธรรมชาติ
- สร้างระบบผู้เชี่ยวชาญ
- สร้างสัญลักษณ์ของการแก้ปัญหาทางคณิตศาสตร์
- งานทางด้านปัญญาประดิษฐ์

Turbo Prolog แตกต่างจากภาษาอื่นอย่างไร

Turbo Prolog เป็นภาษาเชิงอธิบาย จะต้องบอกขั้นตอนว่าคอมพิวเตอร์จะแก้ปัญหายังไร โปรแกรม Turbo Prolog ประกอบด้วยรายละเอียดของปัญหารายละเอียดนี้จะมี 3 ส่วน ส่วนที่ 1 และส่วนที่ 2 จะเป็นส่วนของการประกาศการติดต่อและส่วนที่ 3 จะเป็นส่วนของอนุประโยคซึ่งจะรวมกฎเกณฑ์และความจริง

1. ชื่อและโครงสร้างของ object ในปัญหานั้นๆ
2. ชื่อของความสัมพันธ์ที่อยู่ระหว่าง object
3. ความสัมพันธ์ของกฎเกณฑ์และข้อเท็จจริง

รายละเอียดของการเขียนโปรแกรม Turbo Prolog จะใช้ความสัมพันธ์ระหว่างข้อมูลเข้าและข้อมูลออก คำตอบจะมาจากข้อมูลเข้า Turbo Prolog จะใช้กฎเกณฑ์และความจริง โปรแกรม Turbo Prolog จะมีรายละเอียดของอนุประโยคทางตรรกความจริงดังนี้

It is raining to day.

หรือรูปแบบของกฎเกณฑ์จะเป็นดังนี้

You will get wet if it is raining

and you forgot your umbrella.

Turbo Prolog สามารถพิจารณาเหตุผลจากความจริงต่อไปนี้

pohn likes malee.

tom likes sang.

และกฎเกณฑ์คือ

somsri likes X if tom likes X.

Turbo Prolog จะให้คำตอบคือ

somsri likes sang.

คุณสามารถที่จะให้ Turbo Prolog ค้นหาคำตอบจากตัวอย่างคือ

Find every person who likes sang.

และ Turbo Prolog จะใช้กฎเกณฑ์และความจริงทั้งหมดเป็นข้อมูลในการหาคำตอบ ทุกคำตอบของปัญหานั้นๆ

Turbo Prolog จะควบคุมโดยอัตโนมัติ เมื่อ Turbo Prolog ทำงานระบบจะพยายามค้นหากลุ่มของข้อมูลที่จะนำไปสู่คำตอบ ระหว่างปฏิบัติการคำตอบอาจแสดง หรือผู้ใช้อาจจะป้อนข้อมูลเข้าไปในปฏิบัติการ Turbo Prolog จะใช้ backtracking mechanism ในคำตอบแรกที่พบ Turbo Prolog จะค้นหาจุดหมายใหม่ (goal) จากคำใหม่ๆ เพื่อจะหาคำตอบใหม่ๆ

ไวยากรณ์ของ Turbo Prolog จะสั้นและง่าย ซึ่งจะเป็นไวยากรณ์ที่ง่ายมากไม่ซับซ้อนเหมือนโปรแกรมภาษารุ่นเก่าๆ

Turbo Prolog เป็นภาษาระดับสูงกว่า ปาสคาล มีรูปแบบการจดจำที่ง่ายมีประสิทธิภาพโดยใช้โครงสร้างของการเรียกตัวเอง (Recursive)

7.2 ภาษาโปรลอก

บทนี้จะเริ่มด้วยการศึกษาถึง โครงสร้างของ Turbo Prolog ส่วนแรกจะเป็นตัวอย่างที่ง่าย โดยใช้จากแบบฝึกหัดที่ง่ายๆ จากองค์ประกอบพื้นฐานของโปรแกรมโปรลอกทีละขั้นตอน

การทำงานของโปรแกรมโปรลอกจะค้นหาคำตอบแบบ backtracking และวิธีที่จะให้โปรลอกตรวจสอบว่า ถ้าบางสิ่งบางอย่างไม่จริง แต่มีข้อมูลที่เป็นไปได้ ก็จะสามารถที่จะให้คำตอบที่เป็นไปได้

โครงสร้างของภาษาโปรล็อกจะประกอบด้วยส่วนประกอบที่สำคัญ 4 ส่วนคือ

- domains
- predicates
- goal
- clauses

แต่ก่อนที่จะใช้โครงสร้างเหล่านี้จะต้องรู้จักตัวแปร, object, ความสัมพันธ์ของ object

```
/* The first Program */
```

```
domains
```

```
    name,sport = symbol
```

```
predicates
```

```
    likes(name,sport)
```

```
clauses
```

```
    likes(somporn,tennis).
```

```
    likes(korn,football).
```

```
    likes(tom,baseball).
```

```
    likes(surasri,swimming).
```

```
    likes(mana,tennis).
```

```
    likes(big,X) if likes(tom,X).
```

ในส่วนของ clauses จะเก็บกฎเกณฑ์และความจริง โดยที่ความจริง คือ

likes(somporn,tennis).	ความหมายคือ	somporn likes tennis.
------------------------	-------------	-----------------------

likes(korn,football).	ความหมายคือ	korn likes football.
-----------------------	-------------	----------------------

likes(tom,baseball).	ความหมายคือ	tom likes baseball.
----------------------	-------------	---------------------

likes(surasri,swimming).	ความหมายคือ	surasri likes swimming.
--------------------------	-------------	-------------------------

likes(mana,tennis).	ความหมายคือ	mana likes tennis.
---------------------	-------------	--------------------

likes(big,X) if likes(tom,X).	ส่วนนี้คือ กฎเกณฑ์ (rule) เมื่อ compile และ run โปรแกรมแล้ว	
-------------------------------	---	--

โปรล็อกจะพิมพ์

```
goal:
```

ให้กด Enter

คำตอบคือ likes(big,baseball)

Variables

ใน Turbo Prolog ตัวแปรจะต้องขึ้นต้นด้วยตัวพิมพ์ใหญ่ และมีความยาวรวมกันไม่เกิน 250 ตัวอักษร ตัวต่อไปจะเป็นตัวพิมพ์เล็ก, ตัวเลขหรือขีดเส้นใต้ (_) ก็ได้ แต่ถ้าบางครั้งใน predicates นั้นๆ มี object หลายๆ object แต่บางครั้งเราต้องการให้โปรล็อกไม่สนใจ object เราจะใช้ (_) แทนตัวแปร เช่น likes(_,tennis) คือทุกๆ คนที่ชอบ tennis(_) จะเรียกว่าตัวแปรที่ไม่ระบุ

Object และ Relation

คือ object และความสัมพันธ์ ซึ่งก็คือความสัมพันธ์ระหว่าง object กับ object จากตัวอย่างโปรแกรม ที่ 1 ในส่วนของ clauses

likes(somporn,tennis).

likes(korn,football). ๑

likes ในโปรล็อกจะมองว่าเป็นความสัมพันธ์ และ somporn,tennis,korn,football ก็คือ object

ประเภทของ object

object มีทั้งหมด 6 ประเภท

1. Char ใช้แทน object ประเภทตัวอักษร 1 ตัว และต้องอยู่ในเครื่องหมาย ‘ ’ เช่น ‘m’, ‘a’, ‘z’ เป็นต้น
2. integer ใช้แทนตัวเลขจำนวนเต็ม ตั้งแต่ - 32,768 ถึง 32,767
3. real ใช้แทนตัวเลขจำนวนจริง จะมีค่าตั้งแต่ $\pm 1 \times 10^{-307}$ ถึง $\pm 1 \times 10^{307}$
4. string ใช้แทน object ประเภทตัวอักษรหลายๆ ตัว ซึ่งจะต้องอยู่ในเครื่องหมาย “ ” เช่น “tennis”, “korn” เป็นต้น
5. symbol ใช้แทน object ประเภทตัวอักษรหลายๆ ตัว ตัวแรกจะต้องเป็นตัวพิมพ์ใหญ่เสมอ เช่น Tom, Baseball,Bill,Swimming เป็นต้น
6. file ใช้แทน object ประเภทชื่อของแฟ้มข้อมูล

Domains คือส่วนของการประกาศประเภทของ object เช่น

name,sport	= symbol
age	= integer
money	= real

Predicates คือส่วนที่ใช้สำหรับประกาศความสัมพันธ์ของ object กับ object ที่จะใช้ในโปรแกรม
นั้นๆ เช่น

likes(name,sport)

car(brand,year)

person(name,age,sex)

แต่บางครั้งเราก็สามารถใส่ประเภทของ object ไว้ใน predicates ได้ เช่น

likes(symbol,symbol)

car(symbol,integer)

person(symbol,integer,char)

Goal คือ จุดหมายที่ต้องการจะทราบ เช่น likes(X,tennis) ความหมายคือใครชอบ tennis ไปรลอก
ก็จะค้นหาจากข้อเท็จจริงที่อยู่ในส่วนของ clauses ให้ ถ้ามีก็จะแสดงคำตอบ เช่น
likes(somporn,tennis), likes(mana,tennis) และจะมี yes ตามมา แต่ถ้าไม่มีคำตอบก็จะมีคำว่า
no

clauses คือส่วนของอนุประโยคซึ่งจะเป็นข้อมูลของกฎเกณฑ์และความจริง และเป็นข้อมูลที่
ไปรลอกจะใช้สำหรับค้นหาคำตอบหรือจุดหมาย (goal) ด้วย เช่น

likes(somporn,tennis).

likes(korn,football).

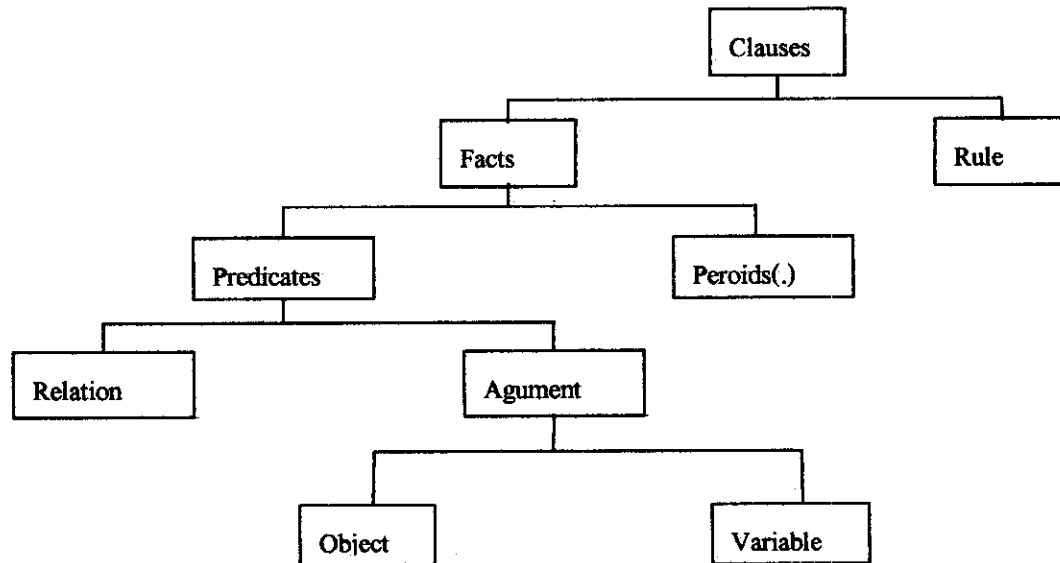
likes(tom,baseball).

likes(surasri,swimming).

likes(mana,tennis).

likes(big,X) if likes(tom,X). ส่วนนี้คือส่วนของ กฎเกณฑ์(rule)

หลังอนุประโยคของ clauses จะต้องตามด้วยจุดจุลภาค “.” เสมอ
รูปแบบของ clauses



comments ในภาษาโปรล็อกจะใช้เครื่องหมาย “/*” สำหรับค้นข้อความ และ เครื่องหมาย “*/”
สำหรับปิดข้อความที่ใช้อธิบายการทำงานของโปรแกรม

7.3 การใช้งาน Turbo Prolog

โปรแกรมจากเอกสารนี้จะใช้ Borland Turbo Prolog Version 2.0 จะเป็นโปรแกรมใน
Dos Mode

เมื่อเรียกใช้ Turbo Prolog แล้ว จะเข้าสู่ Editor ของ Turbo Prolog ซึ่งจะประกอบด้วย
หน้าต่าง 4 หน้าต่าง

หน้าต่างแรกชื่อ Editor ใช้สำหรับป้อน Source Code ของโปรแกรมโปรล็อก ซึ่งจะ
เหมือนกับ Turbo Pascal หรือ Turbo C

หน้าต่างที่สอง ชื่อ Dialog จะใช้สำหรับดูผลลัพธ์หรือป้อนจุดหมาย (goal)

หน้าต่างที่สาม ชื่อ Message จะแสดงการทำงานในขณะนั้นๆ เช่น ถ้า Save ก็จะมีชื่อเพิ่ม
ข้อมูล หรือ compile ก็จะมีชื่อเพิ่มที่ compile หรือ run ก็จะมีชื่อของ อนุประ โยคที่เรียกใช้อยู่

หน้าต่างที่สี่ ชื่อ Trace ใช้สำหรับดูการทำงานของโปรล็อกทีละขั้นตอน ถ้าต้องการจะดูการทำงานต้องไปตั้งค่าในเมนู Options ตัวเลือก compiler directives และตัวเลือก Trace ให้เลือก Trace หรือ Short Trace ซึ่ง Short Trace จะทำงานเร็วกว่า Trace ซึ่ง Trace จะตรวจทุกๆ อนุประโยคและแสดงทุกๆ อนุประโยค แต่ Short Trace จะไม่แสดงทุกๆ ประโยค เมื่อเลือก Trace หรือ Short Trace และกำหนดจุดหมายแล้วจะต้อง กดปุ่ม F10 โปรล็อกจึงจะทำงานให้

ส่วนเมนูอื่นๆ ก็จะเหมือนกับ compiler อื่นๆ ของบริษัท Borland

ตัวอย่างโปรแกรมโปรล็อก

```
/* Program Car description */

domains
  brand,color = symbol
  age,price   = integer
  mileage     = real

predicates
  car(brand,mileage,age,color,price)

clauses
  car(chrysler,130000,3,red,12000).
  car(ford,90000,4,gray,25000).
  car(datsun,80000,1,red,30000).
```

เป็นโปรแกรมเก็บข้อมูลรายละเอียดของรถยนต์

เช่นอยากทราบว่ามียี่ห้อคันที่ราคามากกว่า \$20000 เมื่อ Run แล้ว ที่ Goal: ให้พิมพ์

car(Name,Mile,Year,Colour,Cost) and Cost>20000

Name,Mile,Year,Colour,Cost ชื่อเหล่านี้โปรล็อกจะมองว่าเป็นตัวแปร ไม่ใช่ object ซึ่งจะต้องขึ้นต้นด้วยตัวพิมพ์ใหญ่เสมอ

```
/* Program display pupil play table tennis */
```

```
domains
  child = symbol
  age   = integer

predicates
  pupil(child,age)

clauses
  pupil(prapan,9).
```

pupil(john,10).
 pupil(jane,9).
 pupil(suda,9).

โปรแกรมนี้จะให้จับคู่ของเด็กที่มีอายุ 9 ปีสำหรับการแข่งขันเทนนิสโดยที่เด็กทุกคนจะต้องเจอกัน 2 ครั้งใครชนะมากที่สุดจะเป็นแชมป์เมื่อพิมพ์ข้อมูลแล้ว ในส่วนของ Goal: ให้พิมพ์

pupil(Person1,9) and pupil(Person2,9) and Person1 <> Person2

หลักการการทำงานของโปรลอกจะทำงานแบบ backtracking หรือการทำงานแบบย้อนกลับ
 เริ่มต้นโปรลอกจะทำงานดังนี้

pupil(Person1,9) and pupil(Person2,9) and Person1 <> Person2

prapan	prapan	prapan	prapan	เป็นเท็จ
pupil(prapan,9)	pupil(prapan,9)			
pupil(john,10)	pupil(john,10)			
pupil(jane,9)	pupil(jane,9)			
pupil(suda,9)	pupil(suda,9)			

pupil(Person1,9) and pupil(Person2,9) and Person1 <> Person2

prapan	jane	prapan	jane	ถูกต้อง
pupil(prapan,9)	pupil(prapan,9)			
pupil(john,10)	pupil(john,10)			
pupil(jane,9)	pupil(jane,9)			
pupil(suda,9)	pupil(suda,9)			

pupil(Person1,9) and pupil(Person2,9) and Person1 <> Person2

prapan	jane	prapan	suda	ถูกต้อง
pupil(prapan,9)	pupil(prapan,9)			
pupil(john,10)	pupil(john,10)			
pupil(jane,9)	pupil(jane,9)			

pupil(suda,9)

pupil(suda,9)

เมื่อ Person2 ถึงอนุประ โยคสุดท้าย โปรดอกก็จะไปเปลี่ยนค่าของเงื่อนไขแรกซึ่งก็คือค่าของ Person1 และค่าของ Person2 ก็จะเริ่มต้นที่อนุประ โยคแรก

pupil(Person1,9) and pupil(Person2,9) and Person1 <> Person2

	jane	prapan	jane	prapan	ถูกต้อง
pupil(prapan,9)		pupil(prapan,9)			
pupil(john,10)		pupil(john,10)			
pupil(jane,9)	pupil(jane,9)				
pupil(suda,9)		pupil(suda,9)			

pupil(Person1,9) and pupil(Person2,9) and Person1 <> Person2

	jane	suda	jane	suda	ถูกต้อง
pupil(prapan,9)		pupil(prapan,9)			
pupil(john,10)		pupil(john,10)			
pupil(jane,9)	pupil(jane,9)				
pupil(suda,9)		pupil(suda,9)			

pupil(Person1,9) and pupil(Person2,9) and Person1 <> Person2

	suda	prapan	suda	prapan	ถูกต้อง
pupil(prapan,9)		pupil(prapan,9)			
pupil(john,10)		pupil(john,10)			
pupil(jane,9)		pupil(jane,9)			
pupil(suda,9)	pupil(suda,9)				

pupil(Person1,9) and pupil(Person2,9) and Person1 $\triangleleft$ Person2

	suda	jane	suda	jane ถูกต้อง
pupil(prapan,9)		pupil(prapan,9)		
pupil(john,10)		pupil(john,10)		
pupil(jane,9)		pupil(jane,9)		
pupil(suda,9)		pupil(suda,9)		

ซึ่งจะได้คำตอบทั้งหมด 6 ชุด

Person1 = prapan,	Person2 = jane
Person1 = prapan,	Person2 = suda
Person1 = jane,	Person2 = prapan
Person1 = jane,	Person2 = suda
Person1 = suda,	Person2 = prapan
Person1 = suda,	Person2 = jane

/* display Sophie could date */

domains
person = symbol

predicates
male(person)
smoker(person)
vegetarian(person)
somsri_could_date(person)

clauses
male(jack).
male(big).
male(tom).
smoker(gai).
smoker(tom).
vegetarian(jack).
vegetarian(tom).
somsri_could_date(X) if male(X) and not(smoker(X)).
somsri_could_date(X) if male(X) and vegetarian(X).

goal

somsri_could_date(X) and
write("a possible date for somsri is ",X) and nl.

โปรแกรมนี้จะให้โปรแกรมค้นหาคำตอบโดยไม่ต้องป้อนจุดหมาย(goal) หลังจาก run แล้ว
เครื่องหมายทางคณิตศาสตร์และตรรกศาสตร์

Operand1	Operator	Operand2	Result
integer	+, -, *	integer	integer
real	+, -, *	integer	real
integer	+, -, *	real	real
real	+, -, *	real	real
integer or real	/	integer or real	real

Operator Priority

Operator	Priority
ภายในวงเล็บ () ก่อน	4
mod, div	3
*, /	2
+, -	1

และจะทำจากซ้ายไปขวา

Relational Operator

=	เท่ากับ
<	น้อยกว่า
<=	น้อยกว่าหรือเท่ากับ
>	มากกว่า
>=	มากกว่าหรือเท่ากับ
<>, $\neq$	ไม่เท่ากับ

คำสั่งทางตรรกศาสตร์

and คือการ and กันของเงื่อนไข หรือ ,

or คือการ or กันของเงื่อนไข หรือ ;

แต่ not จะเป็น predicate รูปแบบของ not จะต้องใส่ไว้ภายในวงเล็บ not()

if จะใช้สำหรับตรวจสอบเงื่อนไขสามารถใช้เครื่องหมาย “:-” แทนก็ได้

```
/* Program Relation ship */
```

```
domains
```

```
person = symbol
```

```
predicates
```

```
male(person)
```

```
female(person)
```

```
father(person,person)
```

```
mother(person,person)
```

```
parent(person,person)
```

```
sister(person,person)
```

```
brother(person,person)
```

```
uncle(person,person)
```

```
grandfather(person,person)
```

```
clauses
```

```
male(pracha).
```

```
male(chalong).
```

```
male(bob).
```

```
male(itti). .
```

```
female(boonpa).
```

```
female(tan).
```

```
female(marasri).
```

```
female(sukjai).
```

```
mother(marasri,boonpa).
```

```
mother(pracha,sukjai).
```

```
father(pracha,bob).
```

```
father(bever,chalong).
```

```
father(tan,bob).
```

```
father(marasri,pracha).
```

```
parent(X,Y) if mother(X,Y).
```

```
parent(X,Y) if father(X,Y).
```

```
brother(X,Y) if
```

```
male(Y) and
```

```
parent(X,P) and
```

```
parent(Y,P) and
```

```
/* The brother of X is Y if */
```

```
/* X is a male and */
```

```
/* the parent of X is P and */
```

```
/* the parent of Y is P and */
```

X <> Y.	/* X and Y are not the same */
sister(X,Y) if	/* The sister of X is Y if */
female(Y) and	/* Y is female and */
parent(X,P) and	/* the parent of X is P and */
parent(Y,P) and	/* the parent of Y is P and */
X <> Y.	
uncle(X,U) if	/* The uncle of X is U if */
mother(X,P) and	/* the mother of X is P and */
brother(P,U).	/* the brother of P is U. */
uncle(X,U) if	/* The uncle of X is U if */
father(X,P) and	/* the father of X is P and */
brother(P,U).	/* the brother of P is U */
grandfather(X,G) if	/* The grandfather of X is G */
father(P,G) and	/* if the father of P is G */
mother(X,P).	/* and the mother of X is P */
grandfather(X,G) if	/* The grandfather of X is G */
father(X,P) and	/* if the father of X is P */
mother(P,G).	/* the father of P is G */

จงค้นหาว่า

1. Is pracha itti's brother ?
2. Who is marasri's grandfather ?
3. Who is tan's sister ?
4. What is the relationship (if any) between marasri and boonpa ?

Output Predicates จะมีอยู่ 3 ชนิดคือ write, writedevise, writef.

- รูปแบบของ write คือ

write(Arg1,Arg2,...,ArgN)

Arg คือ Variable, Object , ข้อความที่ต้องการแสดงออกทางจอภาพ ถ้าเป็นข้อความต้องอยู่ภายในเครื่องหมาย “ ”

คำสั่งที่มีเครื่องหมาย “\” นำหน้าสามารถใช้ได้เฉพาะ Predicates นี้เท่านั้น

\n : ใช้สำหรับขึ้นบรรทัดใหม่

\t : ใช้สำหรับให้เว้นระยะ 8 ตัวอักษร

```

- รูปแบบของ write(device,output)
output คืออุปกรณ์ที่ต้องการให้แสดงผล
คำสั่งนี้ใช้สำหรับแสดงผลออกที่อุปกรณ์ใดๆ เช่น
write(device(printer)) คือจะส่งให้แสดงผลออกทาง printer
write(device(screen)) คือจะส่งให้แสดงผลออกทางจอภาพ

- รูปแบบของ write(F,Var1,Var2,...,VarN)
F คือรูปแบบที่จะระบุให้คอมพิวเตอร์แสดงบนจอภาพจะมีรูปแบบดังนี้
%-m.p
- แทนคำสั่งให้แสดงผลลัพธ์ทางด้านซ้าย ปกติ ไปรอกจะแสดงผลลัพธ์ว่า
m แทนค่าที่จะระบุถึงความยาวของหน่วยย่อยของผลลัพธ์ของตัวอักษร
p แทนค่าที่จะระบุถึงจำนวนทศนิยมของตัวเลขที่จะแสดงหรือความยาว
ของผลลัพธ์ทุกตัว และ p ต้องเป็นเลขคี่
f แทนค่าจะเป็นจุดทศนิยม
e แทนค่าจะเป็นยกกำลัง
g แทนค่าเป็นเลขที่พิเศษ

Var1,Var2,...,VarN คือ ตัวแปร หรือ Object หรือ ข้อความที่ต้องการจะนำมาแสดงบน
จอภาพ
คำสั่ง m จะใช้สำหรับไปรอกจนครบบรรทัดใหม่
/* all write predicates */
predicates
run
clauses
run:-
write(device(screen),
write("Product list"),nl,
write("%-8%-3.0 = %5.2%8",item_no,42,22.5,bath),nl.

```

Input Predicates จะใช้สำหรับรับข้อมูลจาก Keyboard โดยผู้กล่าวคำสั่ง

- readln ใช้สำหรับรับข้อมูลประเภท string หรือ symbol
รูปแบบ readln(Var)
Var คือ Object หรือ Variable
เมื่อป้อนข้อมูลแล้วจะส่งค่า Enter การทำงานนี้จะสมบูรณ์
- readchar ใช้สำหรับรับข้อมูลประเภท ตัวอักษร 1 ตัว

รูปแบบ readchar(Var)
Var คือ Object ประเภท char
เมื่อป้อนข้อมูล 1 ตัวอักษรแล้วไม่ส่งค่า Enter
- readint ใช้สำหรับรับข้อมูลประเภทตัวเลขจำนวนเต็ม

รูปแบบ readint(Var)
Var คือ Object ประเภท int

- readreal ใช้สำหรับรับข้อมูลประเภทจำนวนจริง
รูปแบบ readreal(Var)

Var คือ Object ประเภท real

/* Read and write display */

domains
number,qty = integer
description = string
cost = real

predicates
go

clauses
go:-

write("Part No. : "),
readln(Number),
write("Description : "),
readln(Description),
write("Quantity on hand : "),
readln(Qty),
write("Cost : "),
readreal(Cost),

write("Part No. = ",Number),nl,
write("Description = ",Description),nl,

write("Quantity on hand = ",Qty),nl,

/* Reads User Predicates not build-in Predicates */

domains
name,thing = symbol

predicates
likes(name,thing)
reads(name)
is_inquisitive(name)

clauses
likes(korn,beer).
likes(luk,skate).
likes(suwit,computer).
likes(Z,books) if
reads(Z) and
is_inquisitive(Z).
likes(luk,books).
likes(luk,films).
reads(korn).
reads(suwit).
is_inquisitive(korn).

/* Find Telephone Number by type Name */

predicates
reference(symbol,symbol)

goal
write("Please type a name :"),
readln(The_Name),
reference(The_Name,No),
write("The Phone number is ",Phone_No),nl.
clauses
reference("Ann","01-123456").
reference("Boonsong","01-569767").
reference("Charem","01-267400").
reference("Daeng","01-191051").

Recursion คือ โปรแกรมประเภทเวียนซ้ำของ โปรแกรมนี้จะต้องการหาผลคูณในแฟก
ทอเรียล เช่น โปรแกรม Factorial ที่จะต้องหาค่าแฟกทอเรียลของค่า Factorial ค่า
เริ่มต้นของ Factorial คือ Factorial 1 = 1 การหาของ Factorial ก็คือ Factorial 5 =
5* Factorial 4 คำนวณไปเรื่อย จนถึงค่าที่ต้องการ คือ Factorial 1 ซึ่งจะมีค่าเท่ากับ 1

/* Recursion Program Factorial */

domains
n = integer
f = real

predicates
factorial(n,f)

clauses

factorial(1,1).

factorial(N,Res) if

N > 0 and

N1 = N-1 and

factorial(N1,FacN1) and

Res = N*FacN1.

เมื่อ Run แล้ว

Goal: factorial(5,X)

โปรแกรมจะแสดงคำตอบของ 5 factorial

Compound Object คือ Object ที่มี Predicates อยู่ใน Object
รูปแบบของ Predicates ใดๆ ใน คือ

Relation(Object,Object,...,Object)

รูปแบบของ Compound Object คือ

Relation(Object,Relation(Object,Object,...),...)

Object ที่อยู่ใน Object จะเรียกว่า Functor
รูปแบบของ Functor คือ

Functor(Object,Object,...)

การกำหนด Compound Object จะต้องกำหนดให้ domains เป็น
necessary = car(brand,color,year);computer(brand,cpu,speed)
brand,color,cpu,name = symbol
year,speed = integer

```

predicates
    owns(name, necessary)

/* Compound object demonstration */
domains
    person = address(name,street,city,zip)
    name,street,city,zip = string
predicates
    read(person)
run

clauses
run:-
    read(Person),nl,
    write(Person),nl,nl,
    write("Correct name and address (y/n)?"),
    readchar(Answer),
    Answer = 'y',
run:-
    write("\nRe-enter name and address"),nl,
    run.

read(address(Name,Street,City,Zip)):-
    write("Name: "),
    readln(Name),
    write("Street: "),
    readln(Street),
    write("City: "),
    readln(City),
    write("Zip: "),
    readln(Zip),
    /* Display Compound Object */

domains
    title,author = symbol
    pages = integer
    publication = book(title,pages)
predicates
    written_by(author,publication)
    long_novel(Title)
clauses
    written_by(heming,book("DR NO",210)).
    written_by(melville,book("MOBY DICK",600)).
    long_novel(Title):- written_by(_,book(Title,Length)),
        Length > 300.

```

List คือจำนวนพจน์หรือค่าพจน์ต่างๆ ที่นำมาเรียงต่อกัน list อาจจะเป็นตัวแปร หรือ object หรือ compound object หรือ object ชนิดอื่นๆ ก็ได้ ใน list จะมีสมาชิกที่เหมือนกันก็ได้ แต่จะแตกต่างกันที่ค่าในวงเล็บกำกับ "[]" เช่น

[computer,chemistry,math,biology,physics]
["dog", "cat", "gozyia", "ant", "fighting cock"]
[1,2,3,4,5,6]

การประกาศประเภทของ object ที่เป็น list จะต้องประกาศที่ domains
หลังประเภทของ object จะตามด้วย "***" เช่น

domains
subject = symbol*
pets = string*
digit = integer*

สำหรับ list ที่ไม่มีของในใดๆ (Empty list) จะใช้ [] แทน

predicates ที่เป็น list จะมีลักษณะเช่นเดียวกัน predicates ใดๆ ไป

โปรแกรมจะแบ่ง list ออกเป็น 2 ส่วน คือส่วนหัว Head และ ส่วนหาง Tail ส่วนหัวคือ
ข้อมูลที่เก็บของ list นั้นๆ ซึ่งข้อมูลที่เหลือคือส่วนหาง เราสามารถระบุส่วนหัวและส่วนหางของ
list ได้โดยการใส่ | คั่นระหว่างส่วนหัวและส่วนหาง เช่น

member([computer|chemistry,math,biology,physics])
หรือจะเขียนทั่วๆ ไปคือ member([Head|Tail])

จะเห็นว่า Head = computer

Tail = chemistry,math,biology,physics

/* Display first List */

domains
namelist = symbol*

predicates
member(namelist)

clauses
member([math,ee,arunya,suvit,varunya,suchai,panom]).

นางจันทิมา รุ่งเรือง

Goal : member(X)

Goal : member([X|Y])

Goal : member([X,Y|Z])

Goal : member([A,B,C,D,E,F|G])

/* Display writeln */

domains

integerlist = integer*

predicates

append(integerlist,integerlist,integerlist)

writeln(integerlist)

clauses

append([],List,List).

append([X|L1],List2,[X|L3]) if

append(L1,List2,L3).

writeln(I).

writeln([Head|Tail]) if

write(Head),nl,writeln(Tail).

Goal : append(L1,L2,[1,2,4])

Goal : append([1,2,3],[5,6],L) and writeln(L)

Goal : append([1,2],[3],L),append(L,L,LL),writeln(LL)

Fail เป็น Built-in Predicates ของ Prolog เมื่อ Prolog พบ fail ในข้อโปรแกรม Prolog ก็จะทำใน

ข้อโปรแกรม Prolog เป็นที่ทราบกันดีว่า Prolog จะหาคำตอบโดยการไล่ไปข้างหน้า (forward chaining) หรือ backward chaining ไปยังข้อโปรแกรม

Prolog

/* Fail Program 15 */

domains

name = symbol

predicates

father(name,name)

everybody

clauses

father(nut,kate).

```

father(suk,jack).
father(suk,marsu).
everybody :-
  father(X,Y) and
  write(X," is ",Y,"'s father\n") and
  fail.

```

Goal : father(X,Y)

Goal : everybody

งานโปรแกรม ถ้าไม่ fail หมายความว่า โปรแกรมจะทำงานต่อไปจนเจอ

Nut is Kat's father

ถ้าไม่ fail หมายความว่า ยังไม่เจอ

```

/* Demonstrate the use of fail */
domains
  province,places = string

```

predicates

```

location(province,places)
run

```

clauses

run:-

```

clearwindow,
location(Province,Places),
write("%-15%-20\n",Province,Places),fail.
run.

```

```

location("Chonburi","Pattaya").
location("Chiangmai","Doi Suthep").
location("Puket","Patong Beach").
location("Bangkok","Grand Palace").
location("Bangkok","Wat Podhi").

```

Goal : run

การทำงานของโปรแกรมเมื่อเจอค่า fail หมายความว่า

การทำงานของ fail หมายความว่า ไม่เจอค่า fail predicates not) and not) จะไม่ทำงานต่อไปจนเจอ

การ fail ไม่เจอ

```

/* use of not with fail predicate */
domains
  province,places = string

```

```

/* Use of cut predicate */
domains
  province,places = string
predicates
  location(province,places)
run
  check(province)
clauses
run:-
  clearwindow,
  writef("%-15%-20\n","Province","Place"),
  fail.

Goal : run
โปรแกรมนี้จะ built ใน Bangkok เพราะใช้ not(check(Province)) ซึ่ง Province คือ Bangkok

Cut เป็น build in predicate ที่ทำหน้าที่ป้องกันการ backtracking หรือย้อนกลับหลังจากที่
โปรแกรมทำงานเงื่อนไขใน clause predicate cut ไม่ทำงานเลย เครื่องหมาย "!" เมื่อ
โปรแกรมทำงานถึง cut แล้วจะผ่าน cut ได้หรือไม่คือ cut จะให้สำเร็จเสมอ

```

```

clauses
run:-
  location(province,places)
run
  check(province)
predicates
  location(province,places)
run
  check(province)
clauses
run:-
  clearwindow,
  writef("%-15%-20\n","Province","Place"),
  fail.
run:-
  location(Province,Places),
  not(check(Province)),
  writef("%-15%-20\n","Province,Places"),
  fail.
run.

```

```

check("Bangkok").
location("Chonburi","Pattaya").
location("Chiangmai","Doi Suthep").
location("Puket","Patong Beach").
location("Bangkok","Grand Palace").
location("Bangkok","Wat Podhi").

```

Predicates		Function
abs(X)	Absolute value of X	
cos(X)	Cosine of X (radians)	
sin(X)	Sine of X (radians)	
tan(X)	Tangent of X (radians)	
arctan(X)	Arc tangent of X	
exp(X)	e raised to the power of X	
ln(X)	Log of base e	
log(X)	Log of base 10	
sqr(X)	Square root of X	
X mod Y	Returns the remainder of X divided by Y	
X div Y	Returns the quotient of X divided by Y	
random(X)	Binds X to a random real number $0 \leq X < 1$	
round(X)	Rounds X	

Goal : run
if not

```
run:-  
    location(Province,Places),  
    check(Province),  
    write("%-15%-20\n",Province,Places),  
    fail.  
run.  
check("Bangkok"):-  
    !,fail.  
check(_).  
location("Chonburi","Pattaya").  
location("Chiangmai","Doi Suthep").  
location("Puket","Patong Beach").  
location("Bangkok","Grand Palace").  
location("Bangkok","Wat Podhi").
```

```
/* Program find root */
```

```
predicates
```

```
solve(real,real,real)
```

```
reply(real,real,real)
```

```
clauses
```

```
solve(A,B,C):-
```

```
    D = B*B-4*A*C, reply(A,B,D),nl.
```

```
reply(.,.,D):-
```

```
    D<0, write("No solution"),!.
```

```
reply(A,B,D):-
```

```
    D = 0, X = -B/(2*A), write("X = ",X),!.
```

```
reply(A,B,D):-
```

```
    SqrtD=sqrt(abs(D)),
```

```
    X1 = (-B+SqrtD)/(2*A),
```

```
    X2 = (-B-SqrtD)/(2*A),
```

```
    write("x1 = ",X1," and x2 = ",X2).
```

```
Goal : solve(1,2,1)
```

```
Goal : solve(1,1,4)
```

```
Goal : solve(1,-3,2)
```

การสร้างหน้าต่าง (windows)

ในโปรแกรม จะมี Predicates สำหรับสร้างหน้าต่าง หลาย Predicates
ก่อนที่จะสร้างหน้าต่างได้จะต้องทราบรายละเอียดของหน้าต่างที่จะใช้กันหน้าต่าง

ค่าของสี

Color	Foreground	Background
Black	0	0
Blue	1	16
Green	2	12
Cyan	3	48
Red	4	64
Magenta	5	80
Brown	6	96
White	7	112
Gray	8	-
Light blue	9	-
Light green	10	-
Light cyan	11	-
Pink	12	-
Light magenta	13	-
Yellow	14	-
High-intensity white	15	-

Predicate makewindow ใช้สำหรับสร้างหน้าต่าง รูปแบบคือ

makewindow(WNo,ScrAttr,FrameAttr,Header,Row,Col,Height,Width)

WNo คือเลขที่ของหน้าต่างที่ใช้จะอิงเมื่อมีการทำงานกันหน้าต่าง

ScrAttr คือ Screen Attribute ที่ป้อนค่าของค่าของสี foreground + background

FrameAttr คือ ค่าของสีของหน้าต่าง

Header คือข้อความที่แสดงการจะแสดงภายในหน้าต่าง

```
Row,Col คือ จุดที่ต้องการกำหนดตำแหน่ง Row = 0 ถึง 24 Col = 0 ถึง 79
Height คือ ความสูงของหน้าต่างที่ต้องการให้สูงกี่ Row
Width คือ ความกว้างของหน้าต่างที่ต้องการให้กว้าง Col

/* Program makewindow first */
predicates
run
clauses
run:-
makewindow(1,20,7,"A blue window",2,5,10,50),
write("The characters are red"),
makewindow(2,184,7,"A light cyan window",14,55,10,20),
write("This window is light cyan and the "),
write("letters are black and blink."),
write("Please type an integer to exit"),nl,
readint(_),
removewindow,
readint(_),
removewindow,
readint(_),
write("Notice where the integer appears").

Goal : run
```

Predicate shiftwindow ใช้สำหรับย้ายหน้าต่าง

รูปแบบ shiftwindow(WNo)

WNo คือหมายเลขของหน้าต่างที่ต้องการให้ย้ายไป

Predicate clearwindow ใช้สำหรับล้างหน้าต่างต่างๆ ถ้าไม่กำหนดขนาดของหน้าต่าง ให้ลบออกก็จะล้างจอภาพให้

รูปแบบ clearwindow

Predicate cursor ใช้สำหรับกำหนดจุดบนหน้าต่าง

รูปแบบ cursor(Row,Col)

Row คือค่าของ Row ภายในขอบเขตของหน้าต่าง

Col คือค่าของ Col ภายในขอบเขตของหน้าต่าง

```
/* Program makewindow #2 */
```

```
predicates  
start  
run(integer)  
do_sums  
set_up_windows  
clear_windows
```

```
clauses  
start:-
```

```
set_up_windows,do_sums.  
set_up_windows:-
```

```
makewindow(1,7,7,"0,0,25,80"),  
makewindow(1,7,7,"Left operand",2,5,5,25),  
makewindow(2,7,7,"2,35,5,10"),  
nl,write("PLUS"),  
makewindow(2,7,7,"Right operand",2,50,5,25),  
makewindow(3,7,7,"Gives",10,30,5,25),  
makewindow(4,7,7,"20,30,5,35").
```

```
do_sums:-
```

```
run(_),clear_windows,do_sums.  
run(Z):-
```

```
shiftwindow(1),  
cursor(2,1),readint(X),  
shiftwindow(2),  
cursor(2,10),readint(Y),  
shiftwindow(3),Z=X+Y,cursor(2,10),write(Z),  
shiftwindow(4),  
write(" Please press the space bar"),  
readchar(_),  
clear_windows:-
```

```
shiftwindow(1),clearwindow,  
shiftwindow(2),clearwindow,  
shiftwindow(3),clearwindow,  
shiftwindow(4),clearwindow.
```

```
Goal : start
```

การเขียนโปรแกรม

การเขียนโปรแกรมการหาค่าผลบวก 4 Predicates ที่เกี่ยวข้องกับ graphics, text, dot, line

Predicate graphics ใช้สำหรับเปลี่ยนให้อยู่ในกราฟ graphics mode
รูปแบบ graphics(Mode,Palette,Background)

Mode คือการเลือกแนวระนาบของจอภาพ

Palette คือสีของ foreground palette

Background คือสี background

Predicate text จะใช้สำหรับในกล่องข้อความ text mode
รูปแบบ text

Predicate dot จะใช้สำหรับการเขียนภาพและจุด หรือ pixel ซึ่งคือจุดตัดระหว่าง
RowและColซึ่งจะนับอยู่ระหว่าง 0 – 31999 และสามารถกำหนดสีของจุดได้
รูปแบบ dot(Row,Col,Color)

Predicate line จะใช้สำหรับเขียนเส้นตรงจากจุดหนึ่งไปยังจุดหนึ่ง
รูปแบบ line(Row1,Col1,Row2,Col2,Color)
Row1,Col1 คือจุดแรก
Row2,Col2 คือจุดที่สอง
Color คือสีของเส้น

การตั้งค่า Mode

Mode	Columns	Row	Description
1	320	200	Medium resolutions, 4 colors
2	640	200	Hi resolutions black and white
3	320	200	Medium resolutions, 16 colors
4	640	200	Hi resolutions, 16 colors
5	640	350	Enhanced resolutions, 13 colors

การตั้งค่า Palette

Palette	Color1	Color2	Color3
0	green	red	yellow
1	cyan	magenta	white

013131901 Background Color

0	black	8	gray
1	blue	9	light blue
2	green	10	light green
3	cyan	11	light cyan
4	red	12	light red
5	magenta	13	light magenta
6	brown	14	yellow
7	white	15	high intensity white

/* manual drawing demonstration */

domains

integer = reference integer

predicates

move(char,integer,integer,integer)

start

changeState(integer,integer)

goal

start

clauses

start:-

makeWindow(1,7,7,"directions to turtle",0,0,24,80),

write("Use theletters: u to move up",nl,

write("Use theletters: d to move down",nl,

write("Use theletters: r to move right",nl,

write("Use theletters: l to move left",nl,

write("Press <CTRL><BREAK> when done."),

readchar(),

graphics(1,1,4),

line(1000,1000,31000,31000,7),

line(1000,31000,31000,31000,7),

line(31000,31000,1000,1000,7),

line(31000,1000,1000,1000,7),

changeState(15000,15000),

changeState(X,Y):-

readchar(Z),move(Z,X,Y,X1,Y1),changeState(X1,Y1).

move(r,X,31000,X,31000):-!

move(r,X,Yold,X,Ynew):-!,Ynew=Yold+100,do(X,Yold,3).

move(l,X,1000,X,1000):-!

Not	Frequency	Keyboard Character
C(low)	131	A
C sharp	139	W
D	147	S
D sharp	156	E
E	165	D
F	175	F
F sharp	185	T
G	196	G
G sharp	208	Y
A	220	H
A sharp	233	U
B	247	J
C (middle)	262	K

การสร้างเสียงในโปรแกรม Computer Piano

Frequency ที่จะส่งไปยัง Duration คือ 1/100 วินาที

sound(Duration,Frequency) predicate นี้ จะให้เสียงต่างจากกันตามค่าในตัวเลข

deep จะให้เสียง deep ขึ้น

จะมี 2 predicates คือ

การสร้างเสียงในโปรแกรม

Crit + Break

โปรแกรมใช้สำหรับเล่นเปียโน โดยให้ผู้เล่น R ใช้สำหรับไปทางขวา L ใช้สำหรับไปทางวา
ขึ้น D ใช้สำหรับเลื่อนลง ขึ้น U ใช้สำหรับเลื่อนขึ้น และถ้าจะออกจากโปรแกรมต้องกดขึ้น

```
move(I,X,Yold,X,Ynew):-!,Ynew=Yold-100,dot(X,Yold,3),
move(U,1000,Y,1000,Y):-!,
move(u,Xold,Y,Xnew,Y):-!,Xnew=Xold-100,dot(Xold,Y,3),
move(d,31000,Y,31000,Y):-!,
move(d,Xold,Y,Xnew,Y):-!,Xnew=Xold+100,dot(Xold,Y,3),
move(...,...):-!,exit
move(...,X,Y,X,Y).
```

```

/* Use sound predicate */
predicates
    piano
    tone(char,integer)
goal
    piano.
clauses
    piano:-
        readchar(Note,tone(Note,Freq),sound(5,Freq),piano.
        tone('a',137).tone('w',139).tone('s',147).
        tone('e',156).tone('f',175).tone('r',185).
        tone('g',196).tone('y',208).tone('h',220).
        tone('u',233).tone('j',247).tone('k',262).
        tone(5000). /* all other keys squeak */

```

การจะออกจากโปรแกรมนี้จะต้อง กดปุ่ม Ctri + Break

การใช้งานแฟ้มข้อมูล

การใช้งานโปรแกรมแฟ้มข้อมูลไม่มีการเก็บข้อมูลที่ใช้สำหรับโปรแกรมที่เราสร้างไว้ไว้ใน hardisk เดียวแฟ้มข้อมูลจะเก็บข้อมูลทาง keyboard ในโปรแกรมจะมีหลาย Predicates สำหรับอ่านและเขียนแฟ้มข้อมูล

Predicate readdevice ใช้สำหรับการอ่านหรือรับ input จากอุปกรณ์ที่กำหนด

รูปแบบ readdevice(SymFile)

SymFile คือ ชื่อของอุปกรณ์ที่จะรับข้อมูล จะมี keyboard และ com1

Predicate writedevice ใช้สำหรับแสดงข้อมูลออกตามอุปกรณ์ที่กำหนด

รูปแบบ writedevice(SymFile)

และทุกครั้งที่ใส่แฟ้มข้อมูลแล้วจะจบการทำงานจะต้องปิดแฟ้มข้อมูลด้วย

Predicate closefile จะใช้สำหรับปิดแฟ้มข้อมูล

รูปแบบ closefile(SymFile)

SymFile คือ ชื่อของอุปกรณ์ที่จะรับข้อมูล จะมี screen,printer และ com1

การใช้งานแฟ้มข้อมูลได้จะต้องมีการเปิดแฟ้มและปิดแฟ้มข้อมูลด้วย การเปิดแฟ้มข้อมูลก็จะ

มีหลายประเภท

Predicate openread(S,FName) จะใช้สำหรับเปิดแฟ้มเพื่ออ่านข้อมูลอย่างเดียว

Predicate openwrite(S,FName) จะใช้สำหรับเปิดแฟ้มเพื่อแก้ไขข้อมูล ถ้าชื่อที่เปิดแฟ้มข้อมูลไปซ้ำกับแฟ้มเก่า ข้อมูลในแฟ้มเก่านั้นจะถูกทับทันที

Predicate openappend(S,FName) จะใช้สำหรับเปิดแฟ้มเพื่อเพิ่มข้อมูลซึ่งข้อมูลใหม่จะ

ต่อท้ายข้อมูลเดิม

Predicate openmodify(S,FName) จะใช้สำหรับเปิดแฟ้มเพื่อการอ่านข้อมูล และปรับปรุงข้อมูล

ในแฟ้มแบบ Random File Access

S คือ parameter สำหรับชื่อคือกับ FName ซึ่งจะต้องประกาศไว้ที่ domains ในแฟ้มประเภท

file

FName คือชื่อของแฟ้มข้อมูลที่จะใช้ทำงาน จะขึ้นอยู่กับชื่อแฟ้มในแฟ้มข้อมูล " " ถ้าแฟ้มนั้นไม่อยู่ใน directory ปัจจุบัน ก็ควรจะกำหนดเส้นทาง (path) ด้วย

/* Use of openwrite predicate */

domains

file = employee_address

predicates
go

clauses

go:-

```
openwrite(employee_address,"address.dat"),
write(device(employee_address),
write("Name1
Address1
Tel.1"),nl,
write("Name2
Address2
Tel.2"),nl,
write("Name3
Address3
Tel.3"),nl,
write("Name4
Address4
Tel.4"),nl,
write("Name5
Address5
Tel.5"),nl,
closefile(employee_address),
write(device(screen)).
```

Goal : go

โปรแกรมนี้จะทำการเขียนข้อมูลในแฟ้ม address.dat โดยข้อมูลจะมี Name1 ถึง Name 5 ,Address1 ถึง Address5 , Tel1 ถึง Tel5

เนื่องจาก predicate openmodify เป็น predicate ที่ใช้สำหรับ random file access จะเห็นว่า
ฟังก์ชันสำหรับการกำหนดตำแหน่งที่จะค้นหาของกฎในแผนผังจะใช้ predicate filepos

รูปแบบ filepos(SymF,Position,Mode)
SymF ชื่อพารามิเตอร์สำหรับชื่อของ FName

Position คือตำแหน่งของแฟ้มที่ต้องการจะทำงาน

Mode คือประเภทของการเริ่มต้นทำงานของแฟ้ม จะมี 3 ค่า

0 เป็นการนับตำแหน่งจากที่เริ่มต้นของแฟ้ม

1 เป็นการนับตำแหน่งจากตำแหน่งที่อยู่ในขณะนั้น

2 เป็นการนับตำแหน่งจากท้ายสุดของแฟ้ม

ในการเก็บข้อมูลจะเก็บเป็น ตัวอักษร(byte) 1 record จะต้องมีจำนวนที่น้อยกว่าขนาดของหน่วยการคำนวณ

ว่าจะไม่ record ในอีกจะใช้จำนวนตัวอักษรตาม

/* test49 (14.3) random file access */

domains
file = employee_address

predicates

go

run

repeat

response(char)

clauses

go:-

repeat,

run,

write("More data modification (y/n)?"),nl,

response(Reply),

Reply = 'n',

write("End of modification."),nl.

go.

repeat

repeat-

repeat

run:-

openmodify(employee_address,"address.dat"),

write("Enter the number of record."),nl,

readin(Record),

Position = 46*(Record-1),

filepos(employee_address,Position,0),

readdevice(employee_address),

readin(Data),

```

write(Data,
readdevice(keyboard),
closefile(employee_address),
writedevice(screen).
response(Reply):-
readchar(Reply),
write(Reply),nl.

```

Goal : go

โปรแกรมนี้จะใช้รับพนักงาน address.dat ซึ่งสร้างโดยโปรแกรมก่อนหน้า
 โปรแกรมจะ record ของป้อนข้อมูลจากตำแหน่ง Position = 46*(Record-1) ในพ้อยท์การป้อนข้อมูล
 โปรแกรมก่อนหน้าด้วย

/* Find rout */
 ค้นหาโปรแกรม

```

domains
town = symbol
distance = integer

```

```

predicates
road(town,town,distance)
route(town,town,distance)

```

```

clauses
road(houston,tampa,200). /* There is a road 200
long from Houston to Tampa */

```

```

road(gordon,tampa,300).
road(houston,gordon,100).
road(houston,kansas_city,120).
road(gordon,kansas_city,130).

```

```

route(Town1,Town2,Distance):-
road(Town1,Town2,Distance).
route(Town1,Town2,Distance):-
road(Town1,X,Dist1),route(X,Town2,Dist2),
Distance = Dist1+Dist2.

```

/* Adventures in a dangerous cave */

```

domains
room = symbol
roomlist = room*

```

```

predicates
gallery(room,room) /* There is a gallery
between two rooms */

```

```

neighborroom(room,room) /* Necessary because it does
not matter which direction
we go along a gallery */
avoid(roomlist)
go(room,room)
route(room,room,roomlist) /* This is the route to be
followed. Roomlist consists
of a list of rooms already
visited. */
member(room,roomlist)
clauses
gallery(entry,monsters). gallery(entry,fountain).
gallery(fountain,hell). gallery(fountain,food).
gallery(exit,gold_treasure). gallery(fountain,mermaid).
gallery(robbers,gold_treasure). gallery(fountain,fobbers).
gallery(food,gold_treasure). gallery(mermaid,exit).
gallery(monsters,gold_treasure).
neighborroom(X,Y) if gallery(X,Y).
neighborroom(X,Y) if gallery(X,Y).
avoid([monsters,robbers]).
go(Here,There) if route(Here,There,[Here]).
route(exit,exit,VisitedRooms) if
member(gold_treasure,VisitedRooms) and
write(VisitedRooms) and nl.
route(Room,Way_out,VisitedRooms) if
neighborroom(Room,Nextroom) and
avoid(DangerousRooms) and
not(member(NextRoom,DangerousRooms)) and
not(member(NextRoom,VisitedRooms)) and
route(NextRoom,Way_out,NextRoom|VisitedRooms)).
member(X,[X|_])
member(X,[_|H]) if member(X,H).
/* Tower of Hanoi */
domains
loc = right ; middle ; left
predicates
hanoi(integer)
move(integer,loc,loc,loc)
inform(loc,loc)
clauses
hanoi(N):- move(N,left,middle,right).
move(1,A,_,C) :- inform(A,C),!.

```

```

move(N,A,B,C) :-
    N1 = N-1,move(N1,A,C,B),inform(A,C),move(N1,B,A,C).
inform(Loc1,Loc2):-
    write("\nMove a disk from ",Loc1," to ",Loc2).
Goal : hanoi(3)

```

```

/* Profession counsellor expert system */

```

```

domains

```

```

    personality,qualification = symbol

```

```

    profession,name,query = string

```

```

    reply = char

```

```

database

```

```

    yes(qualification)

```

```

    no(qualification)

```

```

predicates

```

```

    career(profession)

```

```

    qualification(personality)

```

```

run

```

```

run1

```

```

answer(char)

```

```

go

```

```

repeat

```

```

question(query,qualification)

```

```

clear_facts

```

```

remember(qualification,reply)

```

```

ask(query,qualification,reply)

```

```

goal
go.

clauses
go:-
    makewindow(1,20,112,"Profession Expert System",0,0,25,80),
    write("\n\n\t Profession Counsellor Expert System"),nl,nl,nl,
    write("\t\t\t Version 1.1"),nl,nl,nl,
    write("\t\t\t Please Press Enter"),nl,
    field_attr(20,42,5,206),
    readln(_),
    repeat,
    runl,
    write("\nWould you like to try again (y/n)?"),
    answer(Reply),
    Reply = 'n',
    go.

repeat,
    repeat:-
    repeat.

runl:-
    clearwindow,
    write("What is the person's age ?"),
    readln(Age),

```

```

Age<=15,!
write("Sorry ,your age is not suitable for ")nl,
write("this profession counsellor expert system.")nl.

```

```

run!:-
run.

```

```

run:-
write("What is the person's name ?"),
readln(Person),
career(Profession),!,
write(Person," can probably be a good ",Profession,".")nl,
clear_facts.

```

```

run:-
write("Sorry ,we have not enough knowledge "),
write("base to recommend you.")nl,
clear_facts.

```

```

question(_ ,Qualification):-
yes(Qualification),!.

```

```

question(Query,Qualification):-
not(no(Qualification)),

```

```

ask(Query,Qualification,Reply),
Reply = 'y'.

```

```

ask(Query,Qualification,Reply):-

```

```

write(Query),
readchar(Reply),
write(Reply),nl,
remember(Qualification,Reply):
remember(Qualification,'y'):-
asserta(yes(Qualification)).
remember(Qualification,'n'):-
asserta(no(Qualification)).

clear_facts:-
    retract(yes(_)),fail.
    retract(no(_)),fail.

clear_facts.

qualification(female):-
    question("Is Person female (y/n)? ",female).

qualification(good_health):-
    question("Does Person have good health (y/n)? ",good_health).

qualification(good_common_sense):-
    question("Does Person have good common sense (y/n)? ",good_common_sense).

```

qualification(like_office_work):-

question("Does Person like reading (y/n)? ",like_reading).

qualification(like_reading):-

question("Does Person have devotion (y/n)? ",devotion).

qualification(devotion):-

question("Does Person have good public relation (y/n)? ",good_public_relation).

qualification(good_public_relation):-

question("Is Person's good swimmer (y/n)? ",good_swimmer).

qualification(good_swimmer):-

question("Is Person an age less than 25 (y/n)? ",age_less_than_25).

qualification(age_less_than_25):-

question("Does Person have good appearance (y/n)? ",good_appearance).

qualification(good_appearance):-

question("Does Person have determination (y/n)? ",determination).

qualification(determination):-

question("Does Person have self confidence (y/n)? ",self_confidence).

qualification(self_confidence):-

question("Does Person have good at number (y/n)? ",good_at_number).

qualification(good_at_number):-

question("Does Person like office work (y/n)? ",like_office_work).

qualification(progressive_thought):-

question("Does Person have progressive thought (y/n)? ",progressive_thought).

qualification(decisive):-

question("Is Person decisive (y/n)? ",decisive).

qualification(patience):-

question("Does Person patience (y/n)? ",patience).

qualification(talkative):-

question("Is Person talkative (y/n)? ",talkative).

qualification(humorous):-

question("Is Person humorous (y/n)? ",humorous).

qualification(good_personality):-

question("Does Person have a good personality (y/n)? ",good_personality).

qualification(ambitious):-

question("Is Person ambitious (y/n)? ",ambitious).

qualification(thoughtful):-

question("Is Person thoughtful (y/n)? ",thoughtful).

career("engineer or doctor"):-
qualification(good_at_number,
qualification(self_confidence),
qualification(good_common_sense),
qualification(good_health),
qualification(determination),
qualification(devotion),
qualification(like_reading).

career("engineer or computer_specialist"):-
qualification(good_at_number),
qualification(self_confidence),
qualification(good_common_sense),
qualification(good_health),
qualification(determination),
qualification(like_reading),
qualification(like_office_work),
qualification(progressive_thought),
qualification(decisive).

career("salesman or politician"):-
qualification(good_public_relation),
qualification(determination),
qualification(patience),
qualification(talkative),
qualification(humorous),
qualification(ambitious),
qualification(good_personality).

```

career("air_hostess"):-
    qualification(female),
    qualification(good_health),
    qualification(self_confidence),
    qualification(good_appearance),
    qualification(age_less_than_25),
    qualification(good_swimmer),
    qualification(good_public_relation).

career("engineer"):-
    qualification(good_at_number),
    qualification(self_confidence),
    qualification(good_common_sense),
    qualification(good_health),
    qualification(determination).

career("doctor"):-
    qualification(good_health),
    qualification(determination),
    qualification(devotion),
    qualification(like_reading),
    qualification(self_confidence).

career("computer_specialist"):-
    qualification(good_common_sense),
    qualification(self_confidence),
    qualification(like_reading),

```

qualification(like_office_work),
qualification(progressive_thought),
qualification(decisive).

career("salesman"):-

qualification(good_public_relation),
qualification(determination),
qualification(patience),
qualification(talkative),
qualification(humorous),
qualification(good_personality).

career(politician):-

qualification(good_public_relation),
qualification(determination),
qualification(ambitious),
qualification(talkative),
qualification(humorous),
qualification(good_pwtsonality).

career(university_teacher):-

qualification(good_common_sense),
qualification(devotion),
qualification(like_reading),
qualification(self_confidence),
qualification(thoughtful),
qualification(decisive).

```
answer(Reply):-  
    readchar(Reply),  
    write(Reply),nl.
```