Introduction to Prolog

ดร.บุญเสริม กิจศิริกุล
ภาควิชาวิศวกรรมคอมพิวเตอร์
จุฬาลงกรณ์มหาวิทยาลัย

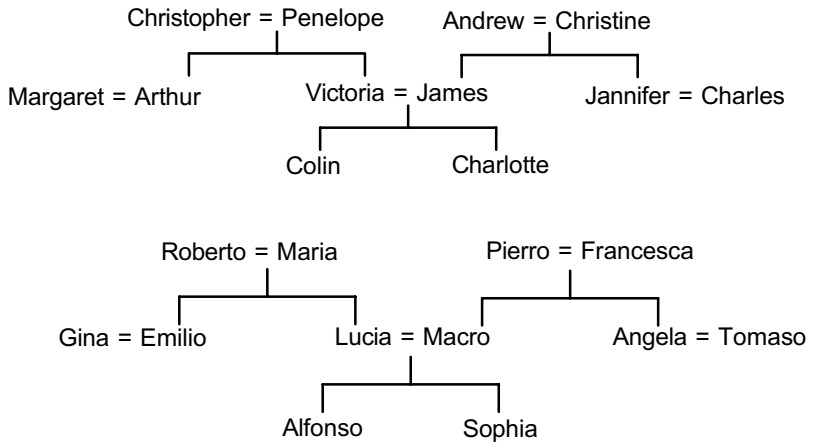
1. Basic Construct

Prolog --- description languageใช้อธิบายobjects และrelationshipsระหว่างobjectsนั้น ๆ

โปรแกรมPrologประกอบด้วย

- factsที่เกี่ยวกับobjectsและrelationsนั้น ๆ
- rulesที่เกี่ยวกับobjectsและrelationsนั้น ๆ
- queriesที่เกี่ยวกับobjectsและrelationsนั้น ๆ

Two Family Trees



A = B หมายถึง A แต่งงานกับ B

1.1 Fact

```
parent(christopher,arthur).
parent(christopher,victoria).
parent(penelope,arthur).
parent(penelope,victoria).
parent(victoria,colin).
parent(victoria,chalotte).
parent(james,colin).
parent(james,chalotte).
    ...
```

```
male(christopher).
male(james).
male(colin).
    ...
female(penelope).
female(margaret).
female(victoria).
    ...
```

parent(christopher,arthur)

- predicate name : parent
- arguments : christopher, arthur

1.2 Constant

- atom : christopher arthur
- number : 123 99.99

1.3 Query

- ?- เป็นเครื่องหมายแสดงquery

ex)

?- parent(christopher,arthur).
yes

?- parent(andrew,james).
no

1.4 Variable, Term, Substitution

variableใช้แทนsingle entity,ไม่ใช่storage location ในmemory

ex) **?- parent(christopher,X).**
X = arthur;
X = victoria;
no

- ; เป็นเครื่องหมายสั่งให้Prologหาคำตอบอื่น

Term

- constants, variables เป็น terms
- compound terms (structures) เป็น terms

Compound term ประกอบด้วย

- functor
- arguments ซึ่งเป็น terms

Substitution คือเซ็ตของคู่ลำดับ { X = t }

X : variable, t : term

• A เป็น *instance* ของ B ถ้ามี substitution θ บางตัวที่ทำให้ A = Bθ

ex)

A = parent(penelope,arthur)
B = parent(X,Y)
θ = { X = penelope, Y = arthur }
A = Bθ

X is *instantiated* to penelope

1.5 Conjunction

, เป็นเครื่องหมายแสดง logical and

ex)

?- parent(penelope,X), parent(X,Y).

หา X และ Y ซึ่ง X มีparentเป็นpenelope

และ X เป็นparentของ Y

• 1 queryอาจประกอบด้วย *goal* เดียวหรือหลาย *goals*ก็ได้

1.6 Rule (Clause)

H :- B1, B2, ... , Bn.

• :- เป็นเครื่องหมายแสดง'if'

grandparent(X,Y) :- parent(X,Z),parent(Z,Y).

Head of the clause : **grandparent(X,Y)**

Body of the clause : **parent(X,Z), parent(Z,Y)**

<u>ความหมายของrule 'P :- Q, R.'</u>

(1) declarative meaning : interpreting the rule as a logical axiom

- P is true if Q and R are true
- grandfather(X,Y) is true if parent(X,Z) and parent(Z,Y) are true.
- forall X, Y and Z, X is a grandparent of Y if X is a parent of Z and Z is a parent of Y

(2) procedural meaning:

- To solve problem P, first solve the subproblem Q and then the subproblem R
- To answer a query *Is X a grandparent of Y,* answer the query *Is X a parent of Z* and *Is Z a parent of Y*

2.1 Equality and matching

?- X = Y.

match X และ Y โดยพยายามทำให้ X equal Y

ถ้าสำเร็จแสดงว่า X match Y ได้

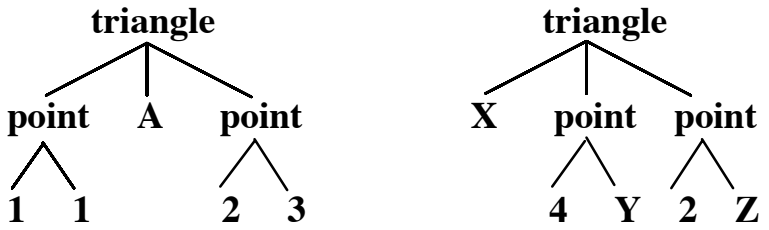
ถ้าไม่สำเร็จแสดงว่า X match Y ไม่ได้

กฎสำหรับmatch X กับ Y

- If X is an uninstantiated variable and Y is instantiated to any term, then X and Y are equal and X is instantiated to that term
- Integers and atoms are equal to themselves
- Two structures are equal if they have the same functor and number of arguments, and all corresponding arguments are equal

ex)

triangle(point(1,1),A,point(2,3)) =
triangle(X,point(4,Y),point(2,Z))



X = point(1,1)
A = point(4,Y)
Z = 3

2.2 backtracking

Clause 'A :- B1, B2, ... , Bn.'

สามารถมองในรูปของconventional languageเป็น

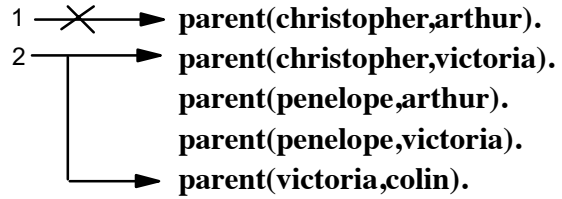
Procedure A;
call B1,
call B2,
...
call Bn.

แต่สิ่งที่ต่างกันคือในPrologจะมีmechanismที่เรียกว่า *backtracking*

- ใน conventional language, เมื่อ Bi ไม่สามารถ processต่อไปได้ จะเกิดerror
  แต่ในProlog, computationจะถูก undone ที่ Bi และdifferent computation pathจะถูก execute
- backtracking ก็คือmechanismนี้ :
  ลืมสิ่งที่ทำแล้วลองพยายาม satisfy goal โดยหา alternationที่จะsatisfy goalนั้น

ex) Find X who is a child of 'christopher'
and a parent of 'colin'

?- parent(christopher,X), parent(X,colin).



Execution of Prolog program

To execute a list of goals G1,G2,...,Gn ,the procedure 'execute' does:

(1) If the goal list is empty then terminate with success

(2) If the goal list is not empty then continue

(3) Scan the clauses in the program from top to bottom until the first clause C (H :- B1,...,Bm) is found such that the head of C matches G1.
If no such clause then terminate with failure.

Find substitution θ such that $H\theta = G1\theta$.

Replace G1 in the goal list with $B1\theta,...,Bm\theta$,

obtaining the new goal list

$B1\theta,..., Bm\theta, G2\theta,..., Gn\theta$

(If C is a fact then the new goal list is shorter than the original one)

(4) Execute (recursively with this procedure) this new goal list. If the execution of the new goal list terminates with success then the execution of the original list also terminates with success. If not, then abandon this new goal and go back to (3). Continue scanning the next clause.

2.3 Example

ปัจหา :

ลิงตัวหนึ่งอยู่ที่ประตูในห้อง กลางห้องมีกล้วยแขวนอยู่ที่เพดาน ลิงหิวมากและอยากไปหยิบกล้วยแต่มันสูงไม่พอที่จะเอื้อมถึง ที่หน้าต่างในห้องมีกล่อง ๆหนึ่ง ลิงสามารถเดินบนพื้น,ปีนกล่อง,ผลักกล่องไปมารอบห้องและหยิบกล้วยถ้ากล่องอยู่ใต้กล้วย

คำถาม :

ลิงหยิบกล้วยได้หรือไม่

ให้predicate state**แสดง**

1. **ตำแหน่งที่ลิงอยู่** 2. **ลิงยืนบนพื้นหรือกล่อง**
3. **ตำแหน่งที่กล่องตั้งอยู่** 4. **ลิงมีกล้วยหรือไม่**

state**เริ่มแรก** : state(atdoor,onfloor,atwindow,hasnot)

state**ที่ต้องการ** : state(_,_,_,has)

ให้predicate move**แสดงการเปลี่ยนจาก**state**หนึ่งไปอีก**state

move(State1,MoveOp,State2)

โดยที่ MoveOp **อาจเป็น**

1 grasp banana 2 climb box 3 push box 4 walk around

เช่น move(state(middle,onbox,middle,hasnot),
grasp, state(middle,onbox,middle,has))

ให้predicate canget**แสดงว่าที่**state**หนึ่ง ๆลิงหยิบกล้วยได้หรือไม่**

```
canget(state(_,_,_,has))
canget(S1) :- move(S1,M,S2), canget(S2).
move(state(middle,onbox,middle,hasnot), grasp,
        state(middle,onbox,middle,has)).
move(state(P,onfloor,P,H), climb, state(P,onbox,P,H)).
move(state(P1,onfloor,P1,H), push(P1,P2),
        state(P2,onfloor,P2,H)).
move(state(P1,onfloor,B,H), walk(P1,P2),
       state(P2,onfloor,B,H)).

?- canget(state(atdoor,onfloor,atwindow,hasnot)).
yes
```

3. Recursive Programming

- Recursive programming เป็นเครื่องมือสำคัญในการเขียนProlog programs

ex) **predesessor(X,Y) :- parent(X,Y).**
predesessor(X,Y) :- parent(X,Z), parent(Z,Y).
predesessor(X,Y) :- parent(X,Z),
parent(Z,W),
parent(W,Y).
. . .
→ **predessessor(X,Y) :- parent(X,Y).**
predessessor(X,Y) :- parent(X,Z), predesessor(Z,Y).

3.1 Arithmetic

Natural Number

- Natural numberแสดงได้โดยใช้constant '0'และ successor function 's'

0, s(0), s(s(0)), . . .

โดยที่ $s^n(0)$ หมายถึง n

Program:

natural_number(0).
natural_number(s(X)) :- natural_number(X).

Addition

plus(X,Y,Z) <--- Z เป็นผลบวกของ X และ Y

Program:

plus(0,X,X) :- natural_number(X).
plus(s(X),Y,s(Z)) :- plus(X,Y,Z).

Multiplication

times(X,Y,Z) <--- Z เป็นผลคูณของ X และ Y

Program:

times(0,X,0).
times(s(X),Y,Z) :- times(X,Y,W), plus(W,Y,Z).

3.2 List

- List เป็นdata structureที่สำคัจในProlog

list	head	tail	formal object
[a]	a	[]	.(a,[])
[a,b,c]	a	[b,c]	.(a,.(b,.(c,[])))
[]	-	-	[]
[[a,b],c]	[a,b]	[c]	.(.(a,.(b,[])),.(c,[]))
[X|Y]	X	Y	.(X,Y)
[X,Y|Z]	X	[Y|Z]	.(X,.(Y,Z))

ex)

สมมุติว่าโปรแกรมเป็น

p([1,2,3]).

และqueryเป็น '?- p([X|Y])' แล้ว เราจะได้ว่า

?- p([X|Y]).
X = 1,
Y = [2,3]

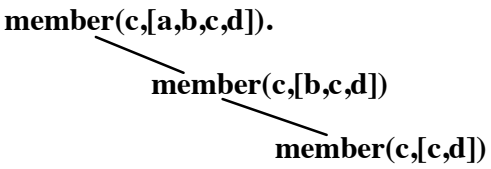
membership of a list --- ทดสอบว่าtermหนึ่งๆเป็นสมาชิกของlistที่กำหนดให้หรือไม่

Program:

```
member(X,[X|_]).
member(X,[_|Y]) :- member(X,Y).
```

Query:

```
?- member(c,[a,b,c,d]).
   yes
```



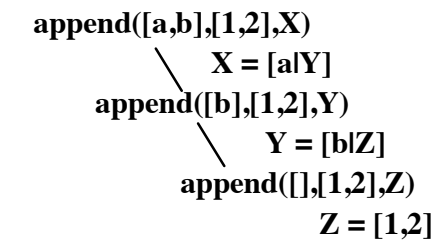
append --- ต่อ2 listเข้าด้วยกัน

Program:

```
append([],L,L).
append([A|L1],L2,[A|L3]) :- append(L1,L2,L3).
```

Query:

```
?- append([a,b],[1,2],X).
   X = [a,b,1,2]
```



4. Negation and Cut

4.1 Negation

- โปรแกรมประกอบด้วยrulesและfactsซึ่งอธิบายว่าอะไรหรือความสัมพันธ์ใด *เป็นจริง*
- *not* ใช้แสดงnegation
- not(G)[1]จะเป็นจริงถ้าGไม่เป็นจริงตามโปรแกรม (negation as failure)
- not(G) จะไม่เป็นจริงถ้าGเป็นจริงตามโปรแกรม

[1] บางครั้งแสดงโดย \+ (G)

ex) X แต่งงานกับ Y ได้ถ้า X กับ Y ไม่ใช่พี่น้องกัน และ X ชอบ Y

Program:

```
can_marry(X,Y) :- not(silbing(X,Y)), like(X,Y).
silbing(X,Y) :- parent(Z,X), parent(Z,Y), X \= Y.
like(arthur,margaret).
```

Query:

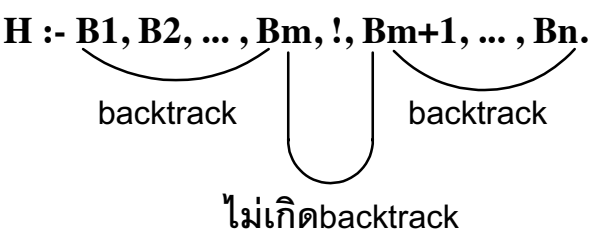
```
?- can_marry(arthur,margaret).
   yes
```

4.2 Cut

- ใช้เพื่อเปลี่ยนแปลงลำดับการทำงานของโปรแกรม

Preventing Backtracking

- Prologจะbacktrackโดยอัตโนมัติเพื่อที่จะsatisfy goalทำให้ผู้ใช้ไม่ต้องกระทำbacktrackเองอย่างexplicit
- แต่บางครั้งbacktrackingทำให้โปรแกรมขาดประสิทธิภาพ
- เพื่อควบคุมหรือป้องกันbacktrackingเราใช้cut(!)



- ก่อนที่ control จะผ่านcut backtrack อาจเกิดภายใน B1,...,Bm

หลังจากที่ทำ B1,...,Bn สำเร็จและต้องการหาคำตอบอื่น

- คำตอบอื่นจะมีได้โดยbacktrackingภายในBm+1,...,Bn
- backtrackingไม่สามารถเกิดภายในB1,...,Bm ได้อีก

- backtrackingไม่สามารถเกิดขึ้นได้แม้ว่าจะมีcluase อื่นเช่น

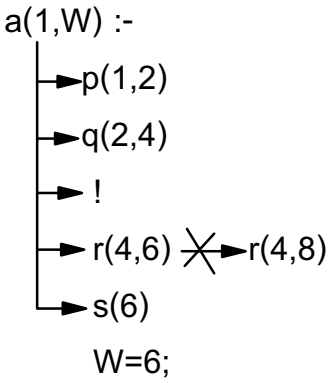
H :- C1, C2, ... , Cp

- หลังจากที่controlผ่านcutไปแล้วคำตอบอื่นทุกตัวที่อาจเกิดขึ้นโดยgoalหน้าcutและโดยclauseอื่นไม่สามารถกระทำได้

```
a(X,W) :- p(X,Y), q(Y,Z), !, r(Z,W), s(W).
a(X,W) :- t(X,W).
p(1,2).
q(2,4).
q(2,5).
r(4,6).
r(4,8).
r(5,7).
s(6).
s(7).
t(1,9).
```

query:

```
?- a(1,W).
 W = 6;
 no
```



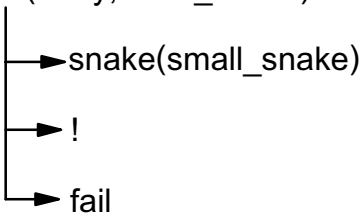
4.3 Cut–Fail Combination

- *fail* เป็นbuild-in predicateซึ่งทำหน้าที่คล้ายnot (แต่failไม่มีargument)
- ทำหน้าที่เป็นgoalซึ่งจะfailเสมอและทำให้เกิด backtracking
- นิยมใช้คู่กับcut

ex) Maryชอบสัตว์ทุกตัวที่ไม่ใช่งู

```
like(mary,X) :- snake(X), !, fail.
like(mary,X) :- animal(X).
snake(small_snake).
snake(big_snake).
```

?- like(mary,small_snake).



4.4 Green Cut : Expressing Determinism

- กรณีที่ต้องการแสดงการtestแบบdeterministic
- cutประเภทนี้เรียกว่า *green cut* ซึ่งจะไม่เปลี่ยนความหมายของโปรแกรม

ex) minimum(X,Y,Z) <--- Z เป็นค่า minimum ของ X และ Y

```
minimum(X,Y,Z) :- X < Y, !, Z = X.
minimum(X,Y,Z) :- X = Y, !, Z = X.
minimum(X,Y,Z) :- X > Y, Z = Y.
```

ex) insert(X,Ys,Zs) <--- the list Zs is the result of inserting the element X into the list Ys

โดยที่ A > B หมายถึง A มากกว่า B

A =< B หมายถึง A น้อยกว่า B

```
insert(X,[],[X]).
insert(X,[Y|Ys],[Y|Zs]) :- X > Y, !, insert(X,Ys,Zs).
insert(X,[Y|Ys],[X,Y|Ys]) :- X =< Y.
```

4.5 Red Cut : Omitting Explicit Condition

- กรณีที่ต้องการละcondition
- cutประเภทนี้เรียกว่า *red cut* ซึ่งจะเปลี่ยนความหมายของโปรแกรม

ex) if_then_else(P,Q,R) <--- if P then Q else R

```
if_then_else(P,Q,R) :- P, !, Q.
if_then_else(P,Q,R) :- R.
```

⟶

```
if_then_else(P,Q,R) :- P,  Q.
if_then_else(P,Q,R) :- not(P), R.
```

ex) delete(Xs,X,Ys) <--- Ys is the result of deleting all occurrences of X from the list Xs

```
delete([X|Ys],X,Zs) :- !, delete(Ys,X,Zs).
delete([Y|Ys],X,[Y|Zs]) :- !, delete(Ys,X,Zs).
delete([],X,[]).
```

4.4 Example : logic puzzle

คน3คนซึ่งเป็นเพื่อนกันสมัครเข้าแข่งขันการเขียนโปรแกรมโดยมาทีละคนไม่พร้อมกัน ทุกคนมีชื่อต่างกัน ชอบกีฬาคนละประเภท และมีสัญชาติต่างกัน

Clue 1: Michaelชอบbasketballและทำคะแนนได้ดีกว่าคนAmerican

Clue 2: Simonซึ่งเป็นคนIsraeliทำคะแนนได้ดีกว่าคนเล่นtennis

Clue 3: คนเล่นcricketมาสมัครคนแรก

Queries: คนสัญชาติAustralianคือใคร? Richardเล่นกีฬาอะไร?

จากclue 1:

did_better(Man1,Man2,Friends), hasName(Man1,michael)

sport(Man1,basketball), nationality(Man2,american)

จากclue 2:

did_better(Man1,Man2,Friends), hasName(Man1,simon)

nationality(Man1,israeli), sport(Man2,tennis)

จากclue 3:

first(Friends,Man), sport(Man,cricket)

- ให้Manแทนfriend(Name,Country,Sport)ซึ่งแสดงความสัมพันธ์"คนชื่อ Name มีสัญชาติCountryและชอบเล่น Sport"
- ให้Friendsแทน [friend(Name1,Country1,Sport1),
  friend(Name2,Country2,Sport2),
  friend(Name3,Country3,Sport3)]

```
solve_puzzle(Puzzle,Solution) :-
    structure(Puzzle,Structure),
    clues(Puzzle,Structure,Clues),
    solve(Clues),
    queries(Puzzle,Structure,Queries,Solution),
    solve(Queries).

solve([Clue|Clues]) :- Clue, solve(Clues).
solve([]).

structure(threeMen,[friend(N1,C1,S1),friend(N2,C2,S2),friend(N3,C3,S3)]).

clues(test,Friends,
    [ ( did_better(Man1Clue1,Man2Clue1,Friends), hasName(Man1Clue1,michael),
        sport(Man1Clue1,basketball), nationality(Man2Clue1,american) ),
      ( did_better(Man1Clue2,Man2Clue2,Friends), hasName(Man1Clue2,simon),
        nationality(Man1Clue2,israeli), sport(Man2Clue2,tennis) ),
      ( first(Friends,ManClue3), sport(ManClue3,cricket) )
    ] ).
```

```
queries(threeMen,Friends,
    [ member(Q1,Friends), hasName(Q1,Name), nationality(Q1,australian),
      member(Q2,Friends), hasName(Q2,richard),    sport(Q2,Sport)
    ],
    [['The Australian is ',Name],['Richard plays ',Sport]] ).

did_better(A,B,[A,B,C]).
did_better(A,C,[A,B,C]).
did_better(B,C,[A,B,C]).

hasName(friend(A,B,C),A).
nationality(friend(A,B,C),B).
sport(friend(A,B,C),C).
first([X|Xs],X).
member(X,[X|_]).
member(X,[_|Ys]) :- member(X,Ys).
```

query:

?- solve_puzzle(threeMen,Sol).

Sol = [['The Australian is ',michael],['Richard play ',tennis]]