

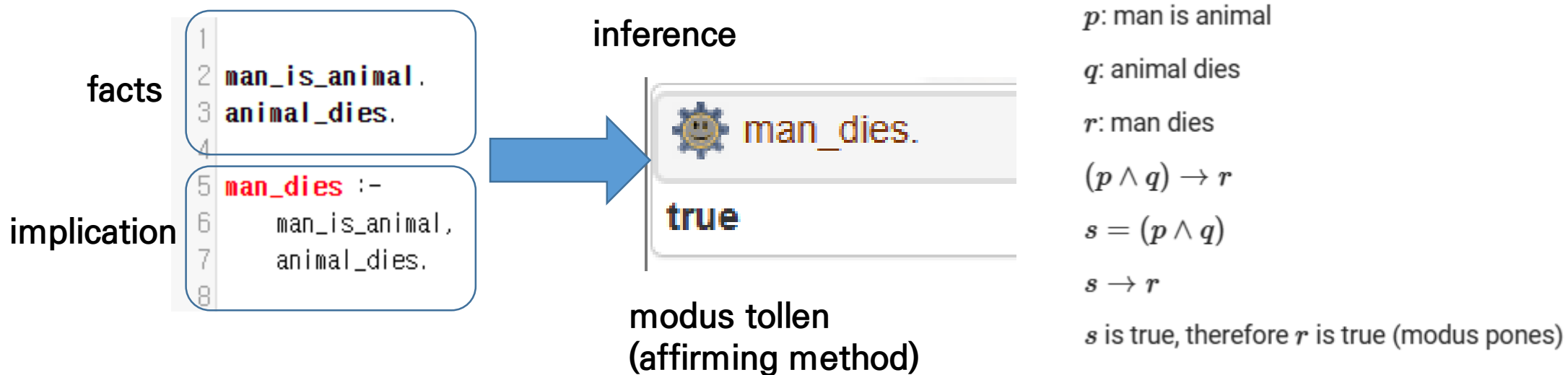
# 이산수학 (1~4강) 배운 거 활용해 보기

동명대학교 게임공학과

2024. 1학기

# 연역법

- 연역: 주어진 사실과 고리에 입각하여 추론을 통해 새로운 사실을 도출
- Prolog
  - Modus ponens (긍정 법칙) 예시



# 연역법

- Modus tollen (부정 법칙)

```
bird(sparrow).  
bird(macpie).  
bird(seagull).
```

```
insect(beatle).
```

```
mammal(human).  
mammal(elephant).  
mammal(bat).
```

```
havewings(beatle).  
havewings(bat).
```

```
flyingAnimal(X) :-  
    ( insect(X); mammal(X)), havewings(X) );  
bird(X).
```

```
flies(X) :-  
    bird(X);  
    flyingAnimal(X).
```

modus tollen  
(denying method)

```
not_bird(X) :-  
    \+ flies(X).
```

# 연역법

- Modus tollens (부정 법칙)

 `flyingAnimal(X).`

`X = beetle`

`X = bat`

 `flyingAnimal(X).`

`X = beetle`

`X = bat`

`X = sparrow`

`X = macpie`

`X = seagull`

 `bird(X).`

`X = sparrow`

`X = macpie`

`X = seagull`

 `not_bird(human).`

`true`

Modus tollen에 의해 확정적으로 새가 아니라고 하는 것

 `not_bird(bat).`

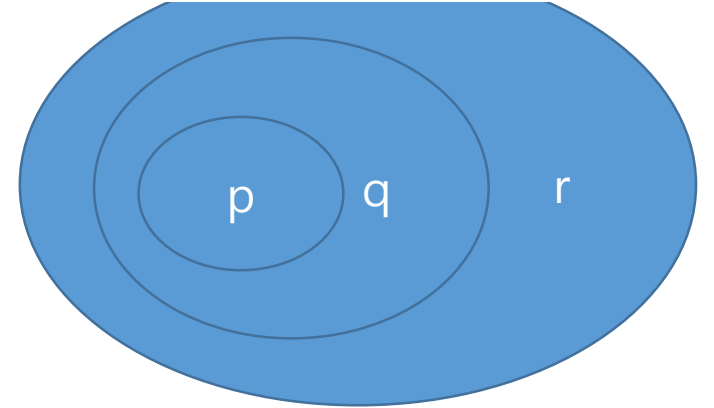
`false`

 새라는 의미가 아니고 확정적으로 새가 아니라 할 수 없다는 것

# 연역법

- 삼단 논법(Hypothetical syllogism - 조건 추론)

$$\begin{pmatrix} p \rightarrow q \\ q \rightarrow r \end{pmatrix} \Rightarrow p \rightarrow r$$



% Facts

is\_parent(john, mary).

is\_parent(mary, tom).

John is older than mary

Mary is older than tom

% If X is a parent of Y, then X is older than Y.

is\_older(X, Y) :- is\_parent(X, Y).

% If X is older than Y and Y is older than Z, then X is older than Z.

is\_older(X, Z) :- is\_older(X, Y), is\_older(Y, Z).

 is\_older(john, tom).  
true

# 연역법


## • 선언 삼단 논법(OR 양자택일 disjunctive dilemma)

$$\begin{pmatrix} p \vee q \\ \neg p \end{pmatrix} \Rightarrow q$$

유죄는 범죄자(p) 혹은 방조자(q)

누군가가 유죄라면  $p \vee q$   
범죄자는 아니다  $\neg p$   
→ 방조자일 수밖에 없다.

```
help_crime(X).  
X = jack  
X = cathy
```



```
commit_crime(john).  
commit_crime(bob).  
commit_crime(susan).  
commit_crime(sue).
```

```
sentenced_guilty(john).  
sentenced_guilty(susan).  
sentenced_guilty(jack).  
sentenced_guilty(cathy).
```

```
criminal(X) :-  
    commit_crime(X).
```

```
guilty(X) :-  
    % 유죄의 조건  
    criminal(X); sentenced_guilty(X).
```

```
help_crime(X) :-  
    guilty(X), \+ criminal(X).
```

# 연역법

## • 양도 법칙 (constructive dilemma)

$$\frac{\left( \begin{array}{c} (p \rightarrow q) \wedge (r \rightarrow s) \\ p \vee r \end{array} \right)}{\quad} \Rightarrow q \vee s$$

⚙️ *delicious\_or\_healthy(X).*

**X** = wopper

**X** = bigmac

**X** = shanghaiburger

**X** = pulmuone

**X** = hansalim



kind\_of\_hamburger(wopper).  
kind\_of\_hamburger(bigmac).  
kind\_of\_hamburger(shanghaiburger).

kind\_of\_pizza(mrpizza).  
kind\_of\_pizza(domino).  
kind\_of\_pizza(pizzanara).

organic(pulmuone).  
organic(hansalim).

nonorganic(shinramyon).

food(X) :-  
    kind\_of\_hamburger(X); kind\_of\_pizza(X); organic(X); nonorganic(X).

delicious(X) :-  
    kind\_of\_hamburger(X).

healthy(X) :-  
    organic(X).

delicious\_or\_healthy(X) :-  
    kind\_of\_hamburger(X); organic(X) .

# 연역법

## • 파괴적 법칙 (destructive dilemma)

$$\left( \begin{array}{c} (p \rightarrow q) \wedge (r \rightarrow s) \\ \neg q \vee \neg s \end{array} \right) \implies \neg p \vee \neg r$$

⚙️ `not_delicious_or_not_healthy(X).`

<code>X = wopper</code>
<code>X = bigmac</code>
<code>X = shanghaiburger</code>
<code>X = mrpizza</code>
<code>X = mrpizza</code>
<code>X = domino</code>
<code>X = domino</code>
<code>X = pizzanara</code>
<code>X = pizzanara</code>
<code>X = pulmuone</code>
<code>X = hansalim</code>



```
kind_of_hamburger(wopper).
kind_of_hamburger(bigmac).
kind_of_hamburger(organicburger).
kind_of_hamburger(shanghaiburger).
```

```
kind_of_pizza(mrpizza).
kind_of_pizza(domino).
kind_of_pizza(pizzanara).
```

```
organic(pulmuone).
organic(hansalim).
organic(organicburger).
```

```
nonorganic(shinramyon).
```

```
food(X) :-
    kind_of_hamburger(X); kind_of_pizza(X); organic(X).
```

```
delicious(X) :-
    kind_of_hamburger(X).
```

```
healthy(X) :-
    organic(X).
```

```
not_delicious_or_not_healthy(X) :-
    food(X), ( \+ kind_of_hamburger(X) ; \+ organic(X)).
```



# 수학적 귀납법

- 수학적 귀납법으로 증명되는 명제는 점화식으로 표현 가능
  - 점화식
    - 1에서  $n$ 까지 자연수의 합  $A(n) = A(n-1) + n$
    - $n(n+1)/2$ 
      - $n(n+1)/2 = (n-1)n/2 + n = (n^2+n) / 2 = n(n+1)/2$

% Base case

sum\_of\_natural\_numbers(1, 1).

% Inductive step


sum\_of\_natural\_numbers(N, Sum) :-

N > 1,

Prev is N - 1,

sum\_of\_natural\_numbers(Prev, PrevSum),

Sum is PrevSum + N.



sum\_of\_natural\_numbers(10000, Sum).

---

Sum = 50005000

# 수학적 귀납법

- 피보나치 수열

- 점화식

- $F(0), F(1) = 0, 1$


- $F(n) = F(n-1) + F(n-2)$

- 팩토리얼

- 점화식

- $0! = 1$

- $n! = (n-1)! * n$

 `sum_of_natural_numbers(10000, Sum).`

**Sum** = 50005000

# 대우 증명법 (contrapositive proof)

- $\text{parent}(x,y) \rightarrow \text{ischild}(y, x)$

Contrapositive rule

$$\sim \text{ischild}(x,y) \rightarrow \sim \text{parent}(x,y)$$

% Facts

```
parent(john, alice).  
parent(john, bob).  
parent(jack, mary).
```

% Rule

```
% If Y is a child of X, then X is a parent of Y.  
is_child(Y, X) :- parent(X, Y).
```

% Contrapositive Rule

```
% If Y is not a child of X, then X is not a parent of Y.  
not_parent_of(X, Y) :- \+ is_child(Y, X).
```

% Query

```
% Is it true that John is not a parent of Mary?
```

# 존재 증명법 (existence proof)

- Prolog에서는 조건을 만족하는 값을 찾는 일

## % Facts


```
likes(john, pizza).  
likes(mary, sushi).  
likes(alex, pizza).
```

## % Rule

```
% To prove there exists a person who likes pizza.  
exists_person_likes_pizza :-  
    likes(Person, pizza),  
    format('~w likes pizza.~n', [Person]).
```

## % Query

```
% Is there a person who likes pizza?
```

```
 exists_person_likes_pizza.
```

```
john likes pizza.
```

```
true
```

# 집합을 다루는 prolog 코드

```
% Define sets
set(a, [1, 2, 3, 4, 5]).
set(b, [4, 5, 6, 7, 8]).

% Union of two sets
union_set(Set1, Set2, Union) :-
    findall(X, (member(X, Set1); member(X, Set2)), Union).

% Intersection of two sets
intersection_set([], _, []).
intersection_set([X|Set1], Set2, [X|Intersection]) :-
    member(X, Set2),
    !,
    intersection_set(Set1, Set2, Intersection).
intersection_set(_|Set1, Set2, Intersection) :-
    intersection_set(Set1, Set2, Intersection).

% Difference of two sets
difference_set([], _, []).
difference_set([X|Set1], Set2, [X|Difference]) :-
    \+ member(X, Set2),
    !,
    difference_set(Set1, Set2, Difference).
difference_set(_|Set1, Set2, Difference) :-
    difference_set(Set1, Set2, Difference).
```

```
?- set(a, SetA), set(b, SetB),
    union_set(SetA, SetB, Union),
    intersection_set(SetA, SetB, Intersection),
    difference_set(SetA, SetB, Difference).
```

```
set(a, SetA), set(b, SetB), union_set(SetA, SetB, Union), intersection_set(SetA, SetB, Intersection), difference_set(SetA, SetB, Difference).
```

**Difference** = [1, 2, 3],

**Intersection** = [4, 5],

**SetA** = [1, 2, 3, 4, 5],

**SetB** = [4, 5, 6, 7, 8],

**Union** = [1, 2, 3, 4, 5, 4, 5, 6, 7, 8]