邏輯推論

笛摩根定律

- $\overline{x|y} = \bar{x}\&\bar{y}$
- $\overline{(x\&y)} = \bar{x}|\bar{y}$

x	y	x|y	$\overline{x|y}$	$\bar{x}\,\bar{y}$	$\bar{x}\&\bar{y}$
0	0	0	1	1	1
0	1	1	0	1	0
1	0	1	0	0	0
1	1	1	0	0	0

1:25 - 1:28
1:55 - 2:10
課前 2:19 -

布林邏輯：變數是布林值

- P & P ⇒ Q ⇒ Q
  (And、OR)
- A & B & C ⇒ D | E
- -(A & B) ⇔ -A | -B

prolog search eniger, langure

謂詞邏輯：是函數但傳回來的是布林值，有類函數的概念

變數可以不綁定

- Parent (x, y) ⇐ Father (x, y)　族譜
- Parent (John, Johnson)
- Ancestor (x, y) ⇐ Parent (x, y)
- Ancestor (x, y) ⇐ Ancestor (x, y) & Parent (x, y)

一階邏輯：加上 ∀ (對於所有)、∃ (存在) 運算，函數不可以是變數，內容可以是變數 x, y, ...

- ∀People (x) ⇒ Mortal (x)　人都會死
- People (Socrate)　蘇格拉底是人
- Mortal (Socrate)　蘇格拉底會死

二階邏輯：函數本身是變數

- ∃P (P(x) ∧ P(y))
- ∀P ∀x (x ∈ P | x ∉ P)
- ∀P (P(0) ∧ ∀y (P(y) ⇒ P(succ(y)) ⇒ ∀y P(y))

布林邏輯推論：狼羊過河問題 ...

kb.py

```
import re

class KB:
    def __init__(self): # 物件的建構函數
        self.rules = [] # 所有規則
```

```
    self.facts = {} # 所有已被滿足的事實

def load(self, code): # 載入知識庫
    lines = re.split(r'[\.]+ ?', code)
    print(lines)
    for line in lines:
        if len(line.strip()) > 0:
            self.addRule(line)

def isFact(self, term): # 判斷 term 是否為事實
    if len(term) == 0:
        return True
    return self.facts.get(term) != None

# check 函數的作用
#     以 鳥類 <= 會飛 & 生蛋. 為例
#     rule['terms'] = ['會飛' , '生蛋']
#     只要 ['會飛' , '生蛋'] 都被滿足了， check 就會傳回 true
#     此時 forwardChaining 就會把結論 鳥類 加入事實庫。

def check(self, rule): # 檢查規則 rule 是否所有前提都被滿足
    for term in rule['terms']:
        if self.isFact(term.strip()):
            continue
        else:
            return False
    return True

def addFact(self, term): # 把 term 加入事實庫
    self.facts[term] = True
    print("addFact({})".format(term))

def addRule(self, line): # 剖析規則
    m = re.match(r"^([^<=]*)(<=(.*))?$", line)
    head = "" if m.group(1)==None else m.group(1).strip()
    terms= "" if m.group(3)==None else m.group(3).strip().split(r"&")
    print("rule:head={} terms={}".format(head, terms))
    rule = {
      'head': head,
      'terms':terms,
      'satisfy':False
    }
    self.rules.append(rule)

def forwardChaining(self): # 前向推論的演算法
    while True:                    一直先檢查事實是否成立
        anySatisfy = False         所有事實都成立，就把結論推出來

        for rule in self.rules:     # 對於每一條規則
            if not rule['satisfy']: # 如果該規則還沒被滿足
```

```
            if self.check(rule): # 就檢查該規則的前提是否全都滿足
                self.addFact(rule['head']) # 若是就將結論加入事實庫
                rule['satisfy'] = True # 設定該規則已被滿足
                anySatisfy = True # 這次的推理至少有一條新規則被滿足了。

        if not anySatisfy: # 若沒有新規則被滿足，推理就結束了。
            break

    print("facts=", self.facts.keys())
```

Prolog: 邏輯推論引擎、語言

安裝 > choco install swi-prolog

啟動 $ swipl

退出 ?- halt. 所有語句的結尾都用一個點

?- write ("Hello world").

八皇后問題: ?- [nqueens].

nqueens.pl

```
% Every way of arranging N queens in a NxN board so that
% no two queens threaten each other
```

N皇后

```
queens(N, Qs) :-  印出n皇后的解
  numlist(1, N, P),
  findall(Q, (permutation(P, Q), not_diagonal(Q, P)), Qs).

not_diagonal(X, N) :-
  maplist(plus, X, N, Z1),
  maplist(plus, X, Z2, N),
  is_set(Z1),
  is_set(Z2).
```

fact.pl ?- [fact].

階層 ?- face (5, R)

```
% https://github.com/abdulmlik/simple-Prolog-Examples/blob/master/fact.pl
% Author: Abdulmalik Ben Ali
% Date: 5/15/2017
```

4 = 5-1

```
fact(1,1):-!.  N是整數  n1 = 4   face (4, R1)  ...
fact(N,R):- integer(N),N1 is N -1 , fact(N1,R1) , R is R1 *N.
```