實際操作和了解 propositional.wumpus.txt 的程式

1. 找到其中可以合併而且不影響執行程式結果的規則
    - 備註：這一件工作對初學者來說不簡單
2. 能不能改寫程式，讓程式可以肯定地說 (3,1) 有 Pit？而不只是現在所說的 may have

1. 可找到以下可合併的寫法：

第 1 段

原寫法：

```
(defrule b21rp11
    (b21 true)
    (p11 unknown)
=>
    (assert (p11 maybe))
)

(defrule b21rp22
    (b21 true)
    (p22 unknown)
=>
    (assert (p22 maybe))
)

(defrule b21rp31
    (b21 true)
    (p31 unknown)
=>
    (assert (p31 maybe))
)
```

修改後：

```
(defrule b21r
    (b21 true)
=>
    (assert (p11 maybe))
    (assert (p22 maybe))
    (assert (p31 maybe)))
```

第 2 段

原寫法：

```
(defrule b12rp11
    (b12 true)
    (p11 unknown)
=>
    (assert (p11 maybe))
)

(defrule b12rp22
    (b12 true)
    (p22 unknown)
=>
    (assert (p22 maybe))
)

(defrule b12rp13
    (b12 true)
    (p13 unknown)
=>
    (assert (p13 maybe))
)
```

修改後：

```
(defrule b12r
    (b12 true)
=>
    (assert (p11 maybe))
    (assert (p22 maybe))
    (assert (p13 maybe)))
```

第 3 段

原寫法：

```
(defrule nb12rp11
    (b12 false)
=>
    (assert (p11 no))
)

(defrule nb12rp22
    (b12 false)
=>
    (assert (p22 no))
)

(defrule nb12rp13
    (b12 false)
=>
    (assert (p13 no))
)
```

修改後：

```
(defrule nb12
    (b12 false)
=>
    (assert (p11 no))
    (assert (p22 no))
    (assert (p13 no)))
```

因為最後有三組判斷式：

```
(defrule p31no
    (p31 no)
    ?f <- (p31 maybe)
=>
    (retract ?f)
)

(defrule p22no
    (p22 no)
    ?f <- (p22 maybe)
=>
    (retract ?f)
)
```

2. 綜合上述，與題目，重構改寫如下：

```
;; breeze = [true, false]
;; stench = [true, false]
;; PIT = [unknown, yes, no, maybe]
;; Wumpus = [unknown, yes, no, maybe]

(clear)

(deffacts initial
    (b11 false)
    (s11 false)
    (b21 true)
    (b12 false)
    (s21 false)
    (s12 true)
    (p11 no)
    (w11 no)
    (p21 unknown)
    (p31 unknown)
    (p22 unknown)
    (p12 unknown)
    (p13 unknown)
    (w21 unknown)
    (w31 unknown)
    (w22 unknown)
    (w12 unknown)
    (w13 unknown))

(reset)

(defrule r:yes-p31
    (p31 yes)
    =>
    (assert (b21 true))
    (assert (b32 true))
    (assert (b41 true)))

(defrule r:no-p11
    (p11 no)
    ?f <- (p11 maybe)
    =>
    (retract ?f))

(defrule r:no-p22
    (p22 no)
    ?f <- (p22 maybe)
    =>
    (retract ?f))

(defrule r:no-p31
    (p31 no)
    ?f <- (p31 maybe)
    =>
```

```
    (retract ?f))

(defrule r:false-b12
    (b12 false)
    =>
    (assert (p11 no))
    (assert (p22 no))
    (assert (p13 no)))

(defrule r:true-b12
    (b12 true)
    =>
    (assert (p11 maybe))
    (assert (p22 maybe))
    (assert (p13 maybe)))

(defrule r:true-b21
    (b21 true)
    =>
    (assert (p11 maybe))
    (assert (p22 maybe))
    (assert (p31 maybe)))

(defrule r:check-p31
    (b21 true)
    (b12 false)
    (p22 no)
    ?m <- (p31 maybe)
    =>
    (retract ?m)
    (assert (p31 yes)))
```

在 CLIPS 環境中執行下列指令，觀察所得，說明(了解)為什麼程式會有這樣的輸出。

```
(assert (numlist 1 2 3 4 5))
(defrule prac1
    (numlist $?p ?q $?r ?s)
    =>
    (printout t "p= " $?p " q= " ?q " r= " $?r " s= " ?s crlf)))
```

這段輸出為：

```
CLIPS> (run)
p= () q= 1 r= (2 3 4) s= 5
p= (1) q= 2 r= (3 4) s= 5
p= (1 2) q= 3 r= (4) s= 5
p= (1 2 3) q= 4 r= () s= 5
```

因為 `$?` 是 multifield wildcard，可以一次 match 0 ~ 多個 pattern，因此 `$?p` 一開始時，會是空 list，而 `$?r` 會是 `2 3 4` 三個值 (`1` 則是 match 給了 `?q` 了)

在 CLIPS 環境中執行以下指令，注意執行過程中 facts window 的變化

```
(assert (stack A B C D E))
(defrule splitStack
    ?c <- (yourChoice ?middle)
    ?oldStack <- (stack $?top ?middle $?bottom)
  =>
    (retract ?c ?oldStack)
    (assert (stack $?top))
    (assert (stack ?middle $?bottom)))
(assert (yourChoice D))
```

1. 以上這一個操作把原來的 stack 分成兩個 stacks
2. 如果最後一個指令改用 (assert (yourChoice B)) 的話，會看到甚麼 facts？

改寫如下：

```
(clear)

(assert (stack A B C D E))

(defrule splitStack
    ?c <- (yourChoice ?middle)
    ?oldStack <- (stack $?top ?middle $?bottom)
  =>
    (retract ?c ?oldStack)
    (assert (stack $?top))
    (assert (stack ?middle $?bottom))
    (printout t "final facts: " crlf)
    (facts))

(assert (yourChoice D))

(printout t "init facts: " crlf)
(facts)

(run)
```

如果改 (assert (yourChoice B))，結果會是：

1. $middle = B
2. $?top = A
3. $?bottom = C D E
4. retract ?c = 1, ?oldStack = 2 之後
5. assert (stack A), (stack B C D E) 進 f-3, f-4
6. 結果為：

```
f-0     (initial-fact)
f-3     (stack A)
f-4     (stack B C D E)
```

實際執行結果如下：

```
CLIPS> (clear)
==> f-0     (initial-fact)
CLIPS> (assert (stack A B C D E))
==> f-1     (stack A B C D E)
<Fact-1>
CLIPS> (defrule splitStack
    ?c <- (yourChoice ?middle)
    ?oldStack <- (stack $?top ?middle $?bottom)
  =>
    (retract ?c ?oldStack)
    (assert (stack $?top))
    (assert (stack ?middle $?bottom))
    (printout t "final facts: " crlf)
    (facts))
CLIPS> (assert (yourChoice B))
==> f-2     (yourChoice B)
<Fact-2>
CLIPS> (printout t "init facts: " crlf)
init facts:
CLIPS> (facts)
f-0     (initial-fact)
f-1     (stack A B C D E)
f-2     (yourChoice B)
For a total of 3 facts.
CLIPS> (run)
FIRE    1 splitStack: f-2,f-1
<== f-2     (yourChoice B)
<== f-1     (stack A B C D E)
==> f-3     (stack A)
==> f-4     (stack B C D E)
final facts:
f-0     (initial-fact)
f-3     (stack A)
f-4     (stack B C D E)
For a total of 3 facts.
CLIPS>
```

In [ ]: