Departamento de Sistemas Informáticos y Computación
Escuela Técnica Superior de Ingeniería Informática
Universitat Politècnica de València

COLLECTION OF EXERCISES
INTELLIGENT SYSTEMS

Block 1: Knowledge Representation

September 2019

MULTIPLE CHOICE QUESTIONS

1) Given a RBS composed of a single rule:

```
(defrule R1
   ?f <- (list ?x $?y ?x $?y ?x)
=>
   (retract ?f)
   (assert (list $?y)))
```

, and WMinitial={(list a b c b c b c b c b a b c b c b c b c b a)}, how will the **final state** of the WM be?

A. {(list a)}
B. {(list a) (list)}
C. {(list b a b)}
D. {(list c b c)}

---

2) Given a RBS that solves a particular problem, the Working Memory represents:

A. The static data of the problem.
B. The dynamic data of the initial state of the problem.
C. The static and dynamic data of the initial state of the problem.
D. None of the above is true. The WM contains the facts and the rules of the problem.

---

3) We have various boxes of different size on a table and we want to have them stacked on a pile from the largest box (at the bottom) to the smallest box (at the top) (see the 'final' rule to check the goal we are aimed to). We design a RBS with the initial WM:

```
(deffacts data (table 2 5 1 6 8 7 4 pile))
```

, and the following rules:

```
(defrule table-to-pile
  (table  $?rest1  ?x  $?rest3  pile  $?rest1 ?y)
  (test (> ?y ?x))
 =>
  (assert (table $?rest1 $?rest3  pile  $?rest1 ?y ?x)))

(defrule table-to-empty-pile
   (table $?rest1 ?x $?rest3  pile)
 =>
   (assert (table $?rest1 $?rest3 pile ?x)))
```

```
(defrule final
   (table pile 1 2 4 5 6 7 8)
 =>
   (halt))
```

Which of the following assertions is true?

A. The RBS works properly and it returns the correct solution that we want.
B. The RBS works properly, it returns the correct solution that we want and the 'final' rule is not necessary because the inference process will end when the table has no more boxes on it.
C. The RBS only works properly if the boxes on the table are ordered from the smallest to the largest; for instance: (deffacts data (table 1 2 4 5 6 7 8 pile))
D. None of the above assertions is true.

---

4) Given the following RBS that computes the factorial of a number, which of the following assertions is true?

```
(deffacts factorial (number 5) (fact 1))

(defrule fact
  ?f1 <- (number ?n1)
  ?f2 <- (fact ?n2)
=>
  (retract ?f1 ?f2)
  (assert (number (- ?n1 1)))
  (assert (fact (* ?n2 ?n1 ))))
```

A. The RBS works properly and it returns the correct solution.
B. Assertion A is not true because a test (test (> ?n1 1)) is required in the LHS of the rule to work properly.
C. The modification proposed in assertion B is not enough because a 'stop' rule is also necessary: (defrule stop (declare (salience 100)) (number 1) => (halt)).
D. None of the above assertions is true.

---

5) Given the fact:

(school class 1 boys 15 girls 18 class 2 boys 21 girls 14 class 3 boys 16 girls 17)

, where the numeric value after the symbol 'class' is the class identifier and the numeric values after the symbols 'boys' and 'girls' denote the number of boys and girls in the respective class.

Indicate the correct pattern to obtain **ONLY** the identifier of any of the classes and the number of girls in such class.

A. (school $? class ?c $? girls ?g $?)
B. (school $? class ?c boys ? girls ?g $?)
C. (school $? class ? boys $? girls ?g $?)
D. (school class ?c boys ? girls ?g)

---

6) Given a RBS with WMinitial={(list 23 14 56 33)}, and the rules:

```
(defrule R1
  (declare (salience 100))
   ?f <- (list $?x ?z ?y $?w)
   (test (< ?z ?y))
 =>
   (assert (list $?x ?z ?y $?w)))
```

```
(defrule R2
   (declare (salience 150))
     ?f <- (list $?x ?z ?y $?w)
     (test (>= ?z ?y))
=>
     (assert (list $?x ?z ?y $?w)))
```

```
(defrule final
   (declare (salience 200))
      (list $?list)
 =>
     (halt))
```

, the contents of the Agenda (Conflict Set) after the first *pattern-matching* is:

A. One instance of R1, one instance of R2 and one instance of the rule 'final'
B. Two instances of R2 and one instance of R1
C. Two instances of R2, one instance of R1 and one instance of the rule 'final'
D. One instance of the rule 'final'

---

7) Given the RBS in question 6 and assuming the strategy of the Agenda is BREADTH-FIRST, which rule instance is first selected by the CLIPS inference engine to be executed? Mark the assertion that is TRUE:

A. It selects an instance of the rule R1.
B. It selects an instance of the rule R2.
C. It selects an instance of the rule 'final'.
D. None of the above.

---

8) Given a RBS composed of a single rule:

```
(defrule R1
   ?f <- (lista ?x $?y ?x $?z)
=>
   (retract ?f)
   (assert (lista $?y ?x $?z))
   (printout t "The list has changed " crlf))
```

, and WMinitial={(lista a b a b a)}, after executing the RBS, how many times the message "The list has changed " will be shown on the screen?

A. 1
B. 2
C. 3
D. 4

---

9) We have a fork lift to pick up objects from a ground floor and deliver them to the other floors of a building. In a particular problem instance there are two objects, A and B, whose destinations are the second and third floor, and weigh 2 Kg. and 8 Kg., respectively. The fork lift is at the ground floor and it cannot carry objects for more than 40 Kg. Which of the following representations **is NOT appropriate** to implement a graph search in a state-based representation?

A. (fork-lift floor 0 load object A 2 2 object B 3 8 max-weight 40 level 0)
B. (fork-lift floor 0 load object A 2 2 object B 3 8) (max-weight 40)
C. (fork-lift floor 0 load object A 2 2 object B 3 8 level 0) (max-weight 40)
D. (fork-lift floor 0) (load object A 2 2 object B 3 8 level 0) (max-weight 40)

---

10) Given the following RBS, how many rule instances will be inserted in the Agenda in the first inference cycle?

```
(defrule R1
   (lista $?x1 ?y $?x2 ?y $?x3)
=>
   (assert (lista $?x1 ?y $?x3)))

(deffacts inicio
     (lista 2 3 1 2 3 2 1))
```

A. 4
B. 5
C. None
D. 3

11) Given the following RBS, which of the following assertions is **CORRECT?**:

```
(defrule R1
    (declare (salience 100))
      ?f <- (lista $?x ?y)
            (test (> ?y 5))
=>
      (retract ?f)
      (assert (lista $?x)))

(defrule R2
  (declare (salience 200))
      ?f <- (lista ?y $?x)
            (test (> ?y 5))
=>
      (retract ?f)
      (assert (lista $?x)))

(deffacts inicio
      (lista 3 7 1 5 9))
```

A. Only when the strategy of the Agenda is BREADTH, the first rule instance to be executed will be an instance of R1
B. Only when the strategy of the Agenda is DEPTH, the first rule instance to be executed will be an instance of R2
C. An instance of R1 will be always executed in the first place
D. An instance of R2 will be always executed in the first place

---

12) Let the fact (heap A B A A B B A heapA heapB) be the initial state of a RBS. The fact represents an initial heap that contains blocks of type A and B and the goal is to put each block in its corresponding heap; i.e., in heapA or in heapB. Which of the following rules **DOES NOT** take a block A and moves it to heapA so that the problem can be solved?

A.
```
(defrule move-to-heap-A
      (heap $?x A $?y heapA $?z)
=>
      (assert (heap $?x $?y heapA A $?z)))
```

B.
```
(defrule move-to-heap-A
      (heap $?x ?b $?y heapA $?z)
      (test (eq ?b A))
=>
     (assert (heap $?x $?y heapA A $?z)))
```

C. (defrule move-to-heap-A
(heap $?x ?b $?y heapA $?z)
(test (eq ?b A))
=>
(assert (heap $?x ?b $?y heapA ?b $?z)))

D. (defrule move-to-heap-A
(heap $?x ?b $?y heapA $?z)
(test (eq ?b A))
=>
(assert (heap $?x $?y heapA ?b $?z)))

---

13) A given warehouse has two distinctive areas: a load area and an unload area. In each area, we can find several heaps of blocks of type A, B or C. Heaps are identified with an integer number from 1 to 5. The goal of the problem is to put blocks from the load area in a truck and take them to the unload area. Let be the initial fact:

(warehouse area load heap 1 A B C heap 2 B C B heap 3 A area unload heap 4 A B A heap 5 B A B B A)

Assuming that we wish to instance in variable ?p only the identifier of a heap of the load area whose first block is of type A, which of the following patterns **IS NOT** valid for this purpose?

A. (warehouse area load $?c heap ?p A $?r area unload $?d)
B. (warehouse area load $?c heap ?p A $?r)
C. (warehouse $?c heap ?p A $?r area unload $?d)
D. (warehouse $?c heap ?p A $?r 4 $?d)

---

14) Given WMinitial= {(elemento e) (lista e a e b c d e f)} and the following rules:

(defrule R1
; (declare (salience 10))
(elemento ?e)
(lista $?a ?e $?b)
=>
(assert (lista ?e $?a $?b)))

(defrule R2
; (declare (salience -30))
(lista ?a $?x ?a)
(elemento ?a)

```
=>
        (assert   (lista $?x)))
```

Which of the following assertions is **CORRECT**? (NOTE: the semicolon (;) before the (declare (salience …)) commands indicate the command is commented)

A. The final state will depend on the search strategy (breadth, depth, uniform cost, etc.)
B. No rule instance is ever triggered in this RBS
C. The final state would depend on the rules priority (salience …) if the (declare (salience …)) commands were not commented
D. The final state is the same regardless of the search strategy

---

15) Given the LHS of the rule:

```
(defrule R1
        (list $? ?x $? ?y)
        (test (< ?x ?y))
=>
    …
```

, and the fact: (list 1 3 2 1 3 6), how many instances of the rule will be inserted in the Agenda?

A. 0
B. 1
C. 5
D. More than 5

---

16) Given the WMinitial={(lista1 b a a a c c a c b b c)(lista2 a c)} and the following rule, indicate the final WM that will be reached among the given options.

```
(defrule  R1
  ?f <- (lista1  $?x ?a ?a $?y)
        (lista2  $? ?a $?)
  =>
   (retract ?f)
   (assert (lista1 $?x ?a $?y)))
```

A. {(lista1 b b b c) (lista2 a c)}
B. {(lista1 b a c a c b b c) (lista2 a c)}
C. {(lista1 b b b c)}
D. {(lista1 b a c a c b c) (lista2 a c)}

---

17) Given the LHS of the rule:

```
(defrule R2
   ?f <- (lista  $? ?b $?x ?b $?x)
    =>
   ...
```

and the fact (lista c c d c c d c c d), how many rule instances will be inserted in the Agenda?

A. 1
B. 2
C. 3
D. 4

---

18) Let be the WMinitial={(lista 5 7 3 1 6 4) (maximo 0)} and the following rule to calculate the maximum value of a list of numbers

```
(defrule R4
    ?f1 <- (lista $?a ?b $?c)
    ?f2 <- (maximo ?x)
    (test (>  ?b ?x))
=>
     (assert (lista $?a $?c))
     (assert (maximo ?b)))
```

Assuming we wish to obtain a final WM (after successively executing the rule R4) in which a fact of type (maximo …) appears only once with the maximum value of the list of numbers, which of the following assertions is **TRUE**?

A. The rule is correct
B. It is necessary to add (retract ?f1)
C. It is necessary to add (retract ?f2)
D. The modification of answer C is not sufficient and it is also necessary to add (retract ?f1)

---

19) Given the fact (problem tower a b c name A tower a name B tower name C), which of the following patterns is suitable to retrieve the name of a tower that contains only one element?

A. (problem $?x tower ?a $?y name ? $?z)
B. (problem $?x tower ?a name ?z $?x)
C. (problem $?x tower ?a name ?z $?)
D. (problem $? tower ?a name ?)

20) Let be a RBS to calculate the Fibonacci number of an integer n>0, for example, n=5 (the Fibonacci number is stored in fib-1), which of the following assertions is **TRUE**?

```
(defrule Fibonacci_number
  ?f1 <- (number ?n1)
  ?f2 <- (fib-1 ?n2)
  ?f3 <- (fib-2 ?n3)
=>
  (retract ?f1 ?f2 ?f3)
  (assert (number (- ?n1 1)))
  (assert (fib-1 (+?n2 ?n3)))
  (assert (fib-2 ?n2 )))

(deffacts fibonacci (number 5) (fib-1 1) (fib-2 0))
```

NOTE: The Fibonnaci number is calculated as: f(n)=f(n-1)+f(n-2) with f(0)=0 and f(1)=1.

A. The RBS works correctly
B. The above assertion is not true because a test condition (test (> ?n1 1)) is needed in the LHS of the rule to work properly
C. The above modification is not sufficient and it is also necessary to add a halt rule: (defrule stop (declare (salience 100)) (number 1) => (halt)).
D. None of the above assertions is true

---

21) Given the fact:

(market street 1 fruit 20 fish 0 street 2 fruit 10 fish 10 street 3 fruit 16 fish 4)

, where the numeric value after the symbol 'street' is the street identifier and the numeric values after the symbols 'fish' and 'fruit' indicate the number of stalls of fish and fruit in the respective street. Indicate the correct pattern to obtain **ONLY** the identifier of any of the streets and the number of fish stalls in such street.

A. (market $? street ? fruit $? fish ?n $?)
B. (market $? street ?c $? fish ?n $?)
C. (market street ?c fruit ? fish ?n)
D. (market $? street ?c fruit ? fish ?n $?)

---

22) Let be a RBS composed of WMinitial={(lista 2 1 6 2 3)}, and the following rules:

```
(defrule R1
   ?f <- (lista $?x ?z ?y $?w)
   (test (< ?z ?y))
=>
(assert (lista $?x ?z ?y $?w)))
```

```
(defrule R2
   ?f <- (lista $?x ?z ?y $?w)
   (test (> ?z ?y))
=>
(assert (lista $?x ?z ?y $?w)))
```

What are the contents of the Agenda after the first pattern-matching?

A. An instance of R1 and two instances of R2
B. Two instances of R1 and one instance of R2
C. Two instances of R1 and two instances of R2
D. No rule instance is generated

---

23) Given a RBS composed of the following rule:

```
(defrule rule-1
   ?f <- (lista ?y $?x ?y $?x ?y)
=>
   (retract ?f)
   (assert (lista $?x)))
```

, and the initial Working Memory {(lista 1 2 3 2 3 2 3 2 3 2 1 2 3 2 3 2 3 2 3 2 1)}, which is the final state of the WM?

A. {(lista 3 2 3)}
B. {(lista 1) (lista)}
C. {(lista 2 1 2)}
D. {(lista 1)}

---

24) The pattern format (list [name^{s} age^{s}]^{m}) represents the name and age of a number of people. Given a particular list of persons, we wish to count the number of them whose age is between 18 and 65 years old. In order to do this, we have the fact that represents the particular list of people, an initial fact (counter 0) that is used to count the number of people and the rule shown below. Mark the **CORRECT** answer.

```
(defrule count
   ?f1 <-(list  $?x1 ?num $?x2)
   ?f2 <- (counter ?cont)
   (test (numberp ?num))   ;; numberp returns TRUE if ?num is a number
   (test (and (>= ?num 18)(<= ?num 65)))
 =>
```

```
(retract ?f2)
(assert (counter(+ ?cont 1))))
```

A. The RBS works properly and returns the desired outcome.
B. It is only necessary to add the instruction (assert (lista $?x1 $?x2)) in the RHS of the rule for the RBS to work properly.
C. It is necessary to add the instructions (retract ?f1) and (assert (lista $?x1 $?x2)) in the LHS of the rule for the RBS to work properly.
D. None of the above.

---

25) Given the initial WM={(lista b a a a c a c b b c) (lista1 a c d e f g)} and the rule shown below, mark the answer that represents the final state of the WM.

```
(defrule R1
  ?f <- (lista  $?x ?a ?a $?y)
        (lista1  $? ?a $?)
  =>
   (retract ?f)
   (assert (lista $?x ?a $?y)))
```

A. {(lista b b b c) (lista1 a c d e f g)}
B. {(lista b b b c)}
C. {(lista b a c a c b b c) (lista1 a c d e f g)}
D. {(lista b a c b b c) (lista1 a c d e f g)}

---

26) Let be an initial WM={(lista 5 7 3 1 6 4) (minimum 9999)} and the rule shown below. Assuming our goal is to obtain a final WM (after successively executing the rule REGLA) in which a fact of the type (minimum …) appears only once containing the minimum value of the list, which of the following assertions is **TRUE** to accomplish our goal?

```
(defrule REGLA
    ?f1 <- (lista $?a ?b $?c)
    ?f2 <- (minimum ?x)
    (test (<  ?b ?x))
 =>
    (assert (lista $?a $?c))
    (assert (minimum ?b)))
```

A. The rule is correct
B. It is necessary to add (retract ?f2)
C. It is necessary to add (retract ?f1)
D. It is necessary to add (retract ?f1) and (retract ?f2)

27) Indicate the final **CORRECT** outcome after executing the following RBS with an initial WM={(lista 34 77 34)}:

```
(defule R1
   (declare (salience 25))
     ?f <- (lista $?x1 ?num $?x2)
 =>
    (retract ?f)
    (printout t "Message 1" crlf))
```

```
(defrule R2
    (declare (salience 10))
      ?f <- (lista ?num $?x ?num)
    =>
    (retract ?f)
     (printout t "Message 2" crlf))
```

```
(defrule R3
    =>
    (printout t "Message 3" crlf))
```

A. It will show three times the message "Message 1".
B. It will show once "Message 1" and once "Message 2".
C. It will show once "Message 1" and once "Message 3".
D. It will show once "Message 1", once "Message 2" and once "Message 3".

---

28) The following fact represents a number of heaps along with the blocks contained in each heap. The number after the symbol 'heap' denotes the heap identifier; the blocks comprised by the heap appear after the identifier, being the first block the one on the top of the heap. Mark the **CORRECT** option that instantiates the block at the bottom of any heap for the fact shown below:

(problem heap 1 A F G J K heap 2 B D heap 3 C H I L heap 4)

A. (problem $?x1 heap ?num $?x2 ?y heap $?x3) (test (not (member heap $?x1)))
B. (problem $?x1 heap ?num $?x2 ?y heap $?x3) (test (not (member heap $?x2)))
C. (problem $?x1 heap ?num $?x2 ?y $?x3)(test (not (member heap $?x3)))
D. None of the above.

---

29) Let be a RBS formed by WMinitial={(lista 2 1 5 3)}, and the following rules:

```
(defrule R1
     (declare (salience 100))
  ?f <- (lista $?x ?z ?y $?w)
        (test (< ?z ?y))
=>
 (assert (lista $?x ?z ?y $?w)))
```

```
(defrule R2
  (declare (salience 150))
   ?f <- (lista $?x ?z ?y $?w)
         (test (>= ?z ?y))
  =>
   (assert (lista $?x ?z ?y $?w)))
```

```
(defrule final
 (declare (salience 200))
   (lista $?list)
=>
   (halt))
```

after the first *pattern-matching,* how would the activations be ordered in the Agenda?

A. One instance of rule R1, one instance of rule R2 and one instance of the final rule
B. Two instances of the rule R2, one instance of the rule R1, one instance of the final rule
C. One instance of the final rule, two instances of the rule R2, one instance of the rule R1.
D. One instance of the final rule

---

30) Let be a RBS formed by WMinitial={(lista 2 1 6 2 3) (elemento 5)}, and the rule shown below. Which is the content of the final WM?

```
(defrule REGLA
  ?f <- (lista $?x ?z $?w)
        (elemento  ?y)
        (test (< ?z ?y))
 =>
   (assert (lista $?x  $?w))
   (retract ?f))
```

A. {(lista 6) (elemento 5)}
B. {(lista 2 1 2 3) (elemento 5)}
C. {(lista) (elemento 5)}
D. {(lista 2 2) (elemento 5)}

---

31) Let be the following rule that calculates the Greatest Common Divisor (GCD) of two positive integer numbers. Mark the **CORRECT** answer:

```
(defrule GCD
    ?a <- (num ?n1)
    ?b <- (num ?n2)
    (test (> ?n1 ?n2))
=>
    (retract ?a)
    (assert (num (- ?n1 ?n2))))
```

A. The rule correctly calculates the GCD and the final WM will contain a fact 'num' with the GCD value
B. A stop rule with no salience is needed to prevent the RBS from an endless execution
C. A stop rule with salience is needed to prevent the RBS from an endless execution
D. None of the above answers is correct

---

32) Let be a RBS whose initial WM is (list b a c c a b b a rest), and contains the rule:

```
(defrule R1
    ?a <- (list $?x ?y ?y $?x $?z rest $?m)
  =>
    (retract ?a)
    (assert (list $?x $?x $?z rest $?m ?y)))
```

The contents of the final WM is:

A. (list b a a b b a rest c)
B. (list b a rest c a b)
C. (list b a b a rest c b)
D. None of the above answers

---

33) Given the fact (owners cars a b c owner P cars d owner Q cars e f owner R), which describes the cars and then the owner of these cars, which of the following patterns would be used to obtain the name of an owner who has only one car?

A. (owners $?x cars ?a owner $?z)
B. (owners $? cars ? owner ?z $?)
C. (owners $?x cars ?a owner ?z $?x)
D. (owners $? cars ? owner ?z)

---

34) Let be a RBS whose initial WM is {(list A B C A B C C B A C B A)}, and contains the single rule:

```
(defrule R1
  ?f1 <- (list $?x1 ?y $?x2 ?y $?x3)
         (test (> (length $?x2) 0))
         (test (not (member ?y $?x2)))
 =>
       (retract ?f1)
       (assert (list $?x1 ?y ?y $?x3)))
```

The contents of the final WM is:

A. A list that only contains the letter ‘A’
B. A list that only contains the letter ‘B’
C. A list that only contains the letter ‘C’
D. It will depend on whether the used control strategy is breadth or depth

---

35) Given the fact (prueba 1 2 3 4 5 6 7 8 9 10) and the rule:

```
(defrule R1
    ?f1 <-  (prueba $?a  $?c)
=>
    (retract ?f1)
    (assert (lista $?c)))
```

After the first pattern-matching:

A. No rule instances will be produced
B. 9 rule instances will be produced
C. 10 rule instances will be produced
D. 11 rule instances will be produced

---

36) Given the initial WM, WM={(list 3 5 2 5 3 4 2 9 8 8 9 6) (num 5) (repetitions 0)}, and the following rule that calculates the number of times an element is repeated in a list of numbers.

```
(defrule R1
    ?f1 <- (list $?a ?b $?c)
    ?f2 <- (num ?x)
    ?f3 <- (repetitions ?z)
    (test (= ?b ?x))
=>
     (assert (list $?a $?c))
      (assert (repetitions (+ 1 ?z))))
```

Assuming we want to get a final WM (after the successive application of the rule), in which the fact (repetitions …) appears only once and the value of this fact is the number of times that the number ?x of the pattern (num ?x) appears in the list, mark the **CORRECT** answer to accomplish our objective.

A. The rule is correct
B. It is necessary to add (retract ?f1)
C. It is necessary to add (retract ?f1) and (retract ?f3)
D. It is necessary to add (retract ?f3)

---

37) Let be a RBS composed of WMinitial={list 2 1 5 3)}, and the following rules:

```
(defrule R1
   (declare (salience 200))
      ?f <- (list $?x ?z ?y $?w)
            (test (< ?z ?y))
 =>
     (assert (list $?x ?z ?y $?w)))
```

```
(defrule R2
    (declare (salience 50))
       ?f <- (list $?x ?z ?y $?w)
             (test (>= ?z ?y))
=>
       (assert (list $?x ?z ?y $?w)))
```

```
(defrule final
  (declare (salience 150))
    (list $?list)
 =>
    (halt))
```

after the first pattern-matching, how rules would be ordered in the Agenda?

A. One instance of R1, one instance of the final rule, and two instances of R2
B. Two instances of R2, one instance of R1, and one instance of the final rule
C. Once instance of the final rule, two instances of R2, and one instance of R1
D. Once instance of the final rule

---

38) Given the following LHS of a rule:

```
(defrule r1
        (list $?x $?w ?y ?z $?x)
=>
```

And the fact: (list a b a b c c), how many instances of the rule will be included in the Agenda?:

A. 0
B. 1
C. 3
D. 5

EXERCISES
(open answer questions)

Exercise 1

A hungry chimpanzee is in a room where there is bunch of bananas hanging from the ceiling. In order to reach the bananas, the chimpanzee has to pile up 5 boxes that are spread all over the room. Each box has a different size and they must be stacked from the largest to the smallest size for the robot catch the bananas. The problem consists in finding the sequence of actions that allow the chimpanzee satisfy its appetite. Design a RBS that solves this problem by searching in a state space and helps the chimpanzee pile up the boxes.

1. Specify the WM structure. Show an example of an initial WM and the final WM
2. Write the necessary production rules to solve the problem

Solution:

Para representar el estado utilizaremos el siguiente patrón:

(HABITACION APILADAS x_1 x_2 … x_5 100 DESAPILADAS y_1 y_2 … y_5)

donde: $x_i \in \{1,2,3,4,5\}$
$y_i \in \{1,2,3,4,5\}$

y el patrón (FINAL x) para controlar la parada / x ∈ {SI,NO}

Los símbolos numéricos 1, 2, 3, 4 y 5 representan cada una de las cajas; concretamente indican el tamaño de cada caja (1: caja más pequeña, 5: caja más grande). El símbolo 100 que aparece después de la representación de las cajas apiladas sirve para denotar el 'fin' de las cajas apiladas, es decir, representa la base de la pila.

El estado inicial vendrá representado por los hechos:

(FINAL NO)
(HABITACION APILADAS 100 DESAPILADAS 1 5 3 2 4)

o cualquier otra secuencia de las cajas [1,…, 5] desapiladas.

El estado final vendrá representado por los hechos:

(FINAL SI)
(HABITACION APILADAS 1 2 3 4 5 100 DESAPILADAS)

La BH (o Working Memory –WM) estará constituida por el siguiente conjunto de reglas:

```
(defrule apilar
        (FINAL NO)
        ?f <- (HABITACION APILADAS ?x $?y DESAPILADAS $?z ?z1 $?z2)
        (test (< ?z1 ?x))
=>
```

```
        (assert (HABITACION APILADAS ?z1 ?x $?y DESAPILADAS $?z $?z2)))


(defrule desapilar
        (FINAL NO)
        ?f <- (HABITACION APILADAS ?x $?y DESAPILADAS $?z ?z1 $?z2)
        (test (> ?z1 ?x))
=>
        (assert (HABITACION APILADAS $?y DESAPILADAS $?z ?z1 $?z2 ?x)))


(defrule final
        ?f <- (FINAL NO)
        (HABITACION APILADAS 1 2 3 4 5 100 DESAPILADAS)
=>
    (retract ?f)
    (assert (FINAL SI))
    (printout t "Solucion encontrada " crlf))
```

En este caso no es necesario asociar una mayor prioridad a las reglas 'apilar' y 'desapilar' porque si éstas no se activan es que las cajas ya están apiladas en el orden correcto.

Exercise 2

A robot is in a room where there are several boxes spread over the floor which must be stack in two different columns. Each box is labelled with a number (1 or 2) indicating the column where the box must be stack on. The problem restrictions are:

- In each move, the robot can stack a maximum of 3 boxes if they belong to the same column or stack one box in each column
- The difference in the number of boxes in both columns must never be greater than 2. For instance, if there are three boxes in COLUMN1 and two boxes in COLUMN2 the robot cannot take two boxes to COLUMN1 (we assume the initial number of boxes in the room satisfy this restriction).

By using the CLIPS language, outline a RBS that solves the above problem. The necessary elements are:

1. Specify the Fact Base (Working Memory) structure. Write down an example of an initial FB and the final FB
2. The necessary production rules to solve the problem

Solution:

Patrón:

(robot cajas 1 x^s 2 y^s pila 1 z^s 2 w^s) donde

$x^S \in$ INTEGER indica el número de cajas que hay en el suelo que se tienen que apilar en la pila 1

$y^S \in$ INTEGER indica el numero de cajas que hay en el suelo que se tienen que apilar en la pila 2

$z^S \in$ INTEGER es el numero de cajas de la pila1

$w^S \in$ INTEGER es el numero de cajas de la pila2

BHinicial= {(robot cajas 1 4 2 5 pila 1 0 2 0)} /* hay 4 cajas de la pila1 y 5 cajas de la pila 2*/
BHfinal= {(robot cajas 1 0 2 0 pila 1 4 2 5)}

De acuerdo a la representación escogida, lo más cómodo es plantear las siguientes reglas:

- mover-X-cajas-a-pila1
- mover-X-cajas-a-pila2
- mover-1-caja-a-cada-pila

aunque mediante variables multicampo podría emplearse una única regla para mover X cajas a cualquier columna. Sin embargo, la representación de la LHS de la regla sería bastante más complicada.

```
(defrule mover-X-cajas-a-pila1                    ;;de pila1 a pila2
        (mover $?y ?x $?z)
        (robot cajas 1 ?c1 2 ?c2 pila 1 ?p1 2 ?p2)
        (test (and (> ?c1 0)(<= ?x ?c1)))
        (test (<= (- (+ ?p1 ?x) ?p2) 2))))
=>
         (assert (robot cajas 1 (- ?c1 ?x) 2 ?c2 pila 1 (+ ?p1 ?x) 2 ?p2)))


(defrule mover-1-caja-a-cada-pila
        (robot cajas 1 ?x cajas 2 ?y pila 1 ?px 2 ?py)
        (test (and (>= ?x 1) (>= ?y 1)))
=>
        (assert (robot cajas 1 (- ?x 1) 2 (- ?y 1) pila 1 (+ ?px 1) 2 (+ ?py 1)))


(defrule final
     (declare (salience 100))
        (robot cajas 1 0 2 0 $?)
 =>
      (halt)
      (printout t "Solution found " crlf))
```

Exercise 3

Given a fact which represents a list of integer and non-ordered numbers, write a single rule (if possible) to get the initial list ordered. For example:

WMinitial: (lista 4 5 3 46 12 10)
WMfinal: (lista 3 4 5 10 12 46)

Show the rule instances that result from applying the rule to the initial WM.

Solution:

```
(defrule ordenar
        ?a <- (lista $?x ?y ?z $?w)
        (test (< ?z ?y))
=>
        (retract ?a)
        (assert (lista $?x ?z ?y $?w)))
```

Applicable instances with the initial WM:

1) Initial fact with ?y=5, ?z=3
2) Initial fact with ?y=46, ?z=12
3) Initial fact with ?y=12, ?z=10

Exercise 4

Write a simple RBS (only one rule, if possible) to find out how many correct numbers a board of the Loteria Primitiva has (this is a weekly lottery in Spain where players choose six numbers out of a total of 49). The initial WM contains two facts, one fact shows the winning number combination and the other shows the six numbers chosen by the player (you can use a global counter to store the number of correct numbers). Here is one example of initial FB:

WMinitial = {(winning-board 2 5 8 13 24 35) (chosen-numbers 3 5 15 24 26 37)}

Let's suppose the player does not introduce numbers in increasing order, would the RBS you've designed above work equally in this case? Why?

Solution:

```
(defglobal ?*num-aciertos* = 0)

(defrule cuenta-aciertos
  (boleto-ganador $?x ?y $?z)
  (combinacion $?a ?y $?b)
=>
 (bind ?*num-aciertos* (+ ?*num-aciertos* 1))
 (printout t "El numero de aciertos hasta el momento es " ?*num-aciertos* crlf))


(deffacts datos
   (boleto-ganador 2 4 8 12 22 34)
   (combinacion 42 13 22 30 2 5))
```

Sí, el SBR funcionaría de igual modo porque lo que hace la regla es detectar cualquier combinación de dos números iguales de los hechos (boleto-ganador ……) y (combinacion ……).

Exercise 5

Five members of a family are on one side of a bridge and need to cross to the other side. Each person crosses the bridge in one direction or another. Since they have to cross the bridge at night they must use a flashlight. There is only one flashlight available, which lasts a maximum of 30 seconds. Each family member crosses the bridge at a different speed and therefore it takes a different time (grandfather: 12 seconds, father: 8 sec., mother: 6 sec., daughter: 3 sec., son: 1 sec.). The flashlight is consumed each time a family member crosses the bridge and it cannot be recharged at any time. A **maximum of two people** can cross the bridge at the same time. Since each person moves at a different speed, if two of them cross at the same time they move at the speed of the slowest person. For instance:

> Initial state: The five members are on the right side of the bridge
> Action 1: The grandfather crosses the bridge to the left side with the son (the flashlight consumes 12 sec.)
> Action 2: The son crosses to the right side (the flashlight consumes 1 sec.)
> Action 3: The father and the son cross to the left side together (the flashlight consumes 8 sec.)

Design a RBS in CLIPS to find a solution for this problem, if a solution exists. The only permitted action is to cross the bridge (in one direction and/or another, one or two people, etc.)

1) Specify clearly the patterns used to define the fact base. Show the initial and the final fact base.
2) Write the production rules. In the RHS of the rules you can only use the commands “assert”, “retract”, “printout” and/or “halt”. In the LHS you can only use pattern-matching templates and “test”.

Solution 1:

1) Base de Hechos

Una posibilidad de patrón es:

(familia Abuelo derecha| izquierda 12 Padre derecha| izquierda 8 Madre derecha| izquierda 6
Hijo derecha| izquierda 1 Hija derecha| izquierda 3 linterna derecha| izquierda x),

donde x={0..30}

También podría haberse separado la información (estática) de los tiempos de trayecto en un patrón separado.

Base de hechos inicial: (familia Abuelo derecha 12 Padre derecha 8 Madre derecha 6 Hijo derecha 1 Hija derecha 3 linterna derecha 30)

Base de hechos meta: (familia Abuelo izquierda 12 Padre izquierda 8 Madre izquierda 6 Hijo izquierda 1 Hija izquierda 3 $?A)

2) Reglas (incorporando el nivel):

(deffacts problema-puente
(familia Abuelo derecha 12 Padre derecha 8 Madre derecha 6 Hijo derecha 1 Hija derecha 3 linterna derecha 30 nivel 0))

(defrule cruzar-I-2-PERSONAS
?f1 <- (familia $?A ?miembro1 derecha ?x $?B ?miembro2 derecha ?y $?C linterna derecha ?z nivel ?n)
(test (>= ?z (max ?x ?y)))
=>
(assert (familia $?A ?miembro1 izquierda ?x $?B ?miembro2 izquierda ?y $?C linterna izquierda (- ?z (max ?x ?y)) nivel (+ ?n 1))))

(defrule cruzar-D-1PERSONA
?f1 <- (familia $?A ?miembro izquierda ?x $?C linterna izquierda ?z nivel ?n)
(test (>= ?z ?x))
=>
(assert (familia $?A ?miembro derecha ?x $?C linterna derecha (- ?z ?x) nivel (+ ?n 1))))

(defrule meta
(declare (salience 100))
(familia Abuelo izquierda ? Padre izquierda ? Madre izquierda ? Hijo izquierda ? Hija izquierda ? linterna ? ?z nivel ?n)
=>
(printout t "Meta Alcanzada en nivel " ?n " queda en linterna " ?z crlf)
(halt))

Nota: Este diseño, adecuado al objetivo deseado, es uno de los distintos diseños alternativos posibles. No sería adecuado plantear una regla para cada combinación posible de paso.

Una posible solución en el nivel 7, con un consumo de 29 seg es: (Izq: Hijo, Hija), (der: Hija) (Izq: Abuelo Padre) (der: hijo) (Izq: hijo hija) (der: hijo) (Izq: madre hijo)

Solution 2:

Another design valid for any initial position of the family (left side or right side of the river). We also represent the times of each person in separate facts.

(deffacts data
(familia Abuelo derecha Padre derecha Madre derecha Hijo derecha Hija derecha linterna derecha 30)
(time Abuelo 12)
(time Padre 8)
(time Madre 6)
(time Hija 3)
(time Hijo 1)
(crossing derecha izquierda)
(crossing izquierda derecha))

```
(defrule crossing-two-people
  ?f1 <- (familia $?A ?m1 ?side $?B ?m2 ?side $?C linterna ?side ?dur nivel ?n)
     (time ?m1 ?t1)
     (time ?m2 ?t2)
     (crossing ?side ?dest)
     (test (>= ?dur (max ?t1 ?t2)))
 =>
    (assert (familia $?A ?m1 ?dest $?B ?m2 ?dest $?C linterna ?dest (- ?dur (max ?t1 ?t2)) nivel (+ ?n 1)))


(defrule crossing-one-person
  ?f1 <- (familia $?A ?m1 ?side $?B linterna ?side ?dur nivel ?n)
     (time ?m1 ?t1)
     (crossing ?side ?dest)
     (test (>= ?dur ?t1))
 =>
    (assert (familia $?A ?m1 ?dest $?B linterna ?dest (- ?dur ?t1) nivel (+ ?n 1))))
```

Exercise 6

Let a RBS be composed of WMinitial={(lista 3 6 8 5 10)} and whose RB is made up of the following rules:

```
(defrule R1
     ?f <- (lista ?x ?y $?z)
          (test (< ?x ?y))
=>
      (retract ?f)
      (assert (lista ?y $?z))
```

```
(defrule R2
     ?f <- (lista ?y $?x)
          (test (and (<= (length $?x) 3) (>= (length $?x) 1)))
=>
          (retract ?f)
          (assert (lista $?x)))
```

, assuming we apply a breadth-first strategy (more priority to older rule instances, starting by R1), which will be the final Working Memory?. Show the trace that follows from the inference process.

Solution:

Base de Hechos (BH)	**Conjunto Conflicto (CC)**
H1: (lista 3 6 8 5 10) (1*)	R1: H1 (1*)
H2: (lista 6 8 5 10) (2*)	R1: H2 (2*) R2: H2 (se elimina al eliminar H2)
H3: (lista 8 5 10) (3*)	R2: H3 (3*)
H4: (lista 5 10) (4*)	R1: H4 (4*) R2: H4 (se elimina al eliminar H4)
H5: (lista 10)	No hay nuevas instancias

==========

Nota: El número entre paréntesis indica el orden de disparo de la regla y la eliminación del hecho correspondiente como consecuencia del disparo de la regla. (*) indica que el hecho o la instancia de la regla se eliminan de la BH o CC respectivamente.

BHfinal= {(lista 10)}

Exercise 7

Let a RBS composed of WMinitial= {(lista 1 2 3 4)} and whose RB is made up of the following rules:

```
(defrule R1
      ?f <- (lista ?x $?z)
=>
   (retract ?f)
   (assert (lista $?z))
   (assert (elemento ?x)))
```

```
(defrule R2
   ?f <- (elemento ?x)
         (elemento ?y)
         (test (< ?x ?y))
=>
         (retract ?f)
         (assert (lista-new ?x ?y)))
```

, assuming we apply a breadth-first strategy (more priority to older rule instances, starting by R1):

a) Which will be the final WM? Show the trace that follows from the inference process.
b) If the initial WM were WM= {(lista 1 2 2 4)}, how many facts (lista-new ....) would be in the final WM? Which ones?
c) Assuming again the initial WM WMinitial= {(lista 1 2 3 4)}, if we eliminate the command (retract ?f) from R1, which changes regarding the facts (lista-new ...) will be in the final WM?
d) Assuming again the initial WM WMinitial= {(lista 1 2 3 4)}, if we eliminate the command (retract ?f) from R2, which changes regarding the facts (lista-new ...) will be in the final WM?

Solution:

a)

Base de Hechos (BH)	**Conjunto Conflicto (CC)**
H1: (lista 1 2 3 4)(1*)	R1: H1 (1*)
==========	
H2: (lista 2 3 4) (2*)	R1: H2 (2*)
H3: (elemento 1) (4*)	
==========	
H4: (lista 3 4) (3*)	R1: H4 (3*)
H5: (elemento 2) (6*)	R2: H3, H5 (4*)
==========	
H6: (lista 4) (5*)	R1: H6 (6*)
H7: (elemento 3) (7*)	R2: H3, H7 (5*, se elimina al eliminar H3)
	R2: H5, H7 (7*)
==========	
H8: (lista-new 1 2)	No se producen nuevas instancias

H9: (lista)	R2: H5, H10 (8*, se elimina al eliminar H5)
H10: (elemento 4)	R2: H7, H10 (9*)
H11: (lista-new 2 3)	No se producen nuevas instancias
H12: (lista-new 3 4)	No se producen nuevas instancias

Nota: El número entre paréntesis indica el orden de disparo de la regla y la eliminación del hecho correspondiente como consecuencia del disparo de la regla. (*) indica que el hecho o la instancia de la regla se eliminan de la BH o CC respectivamente.

BHfinal= {(lista), (elemento 4), (lista-new 1 2), (lista-new 2 3), (lista-new 3 4)}

b) Asumiendo que no se permite la duplicidad de hechos, los hechos lista-new que habría son:
(lista-new 1 2), (lista-new 2 4)

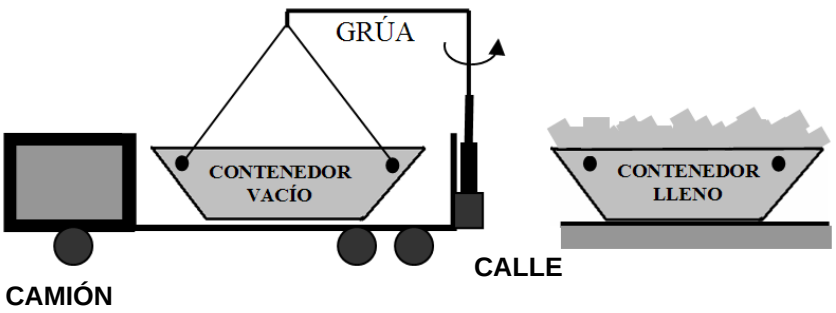
c) Ninguno. Al no eliminar el hecho (lista …) que ha provocado la instanciación de R1 no se produce ningún cambio porque dicho hecho no volverá a instanciar de nuevo regla-1 (no se puede producir una instanciación de una misma regla, con un mismo hecho asignando los mismos valores a las variables del patrón).

d) Se producirían todas las combinaciones posibles con los hechos (lista-new …). Esto es,
(lista-new 1 2) **(lista-new 1 3) (lista-new 1 4)** (lista-new 2 3) **(lista-new 2 4)** (lista-new 3 4). (los hechos que están en negrita serían los nuevos hechos que aparecerían si se eliminara el comando (retract ?f) de R2.

Exercise 8

A crane-truck is used to collect and replace the containers full of rubble (constructions rests) on the street. The truck (CAMION) collects the full container (CONTENEDOR LLENO) and leaves an empty container (CONTENEDOR VACÍO) in the same place. The crane-truck transports the empty container and has a crane (GRUA) on it.

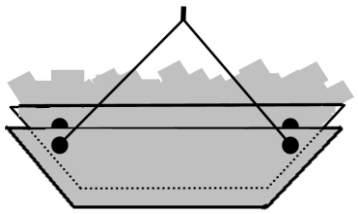
The crane can hook a container (either full of rubble, empty or having another container in it), pick up the container and place the container on the truck or on the street. There is not enough room to have directly two containers in the truck or on the street, unless one is into the other.



The possible actions for the crane are:

Pick-up-container: the crane picks up a container C which can be empty, full or have another container inside (see figure). The container can be situated in the truck or on the street and the crane must be free in order to pick up the container.

Drop-container: The crane drops the held container C (full, empty or with another container inside) in a place (street or truck). ***There is only room for one container in both the truck and the street unless one container is inside the other*** (see figure).



Stack-container: The crane stacks the held container C1 (full or empty) inside container C2, which must be necessarily empty. The crane must be free and container C2 can be in the truck or on the street.

Unstack-container: The crane unstacks container C2 (full or empty) which is inside container C1. After this operation, container C1 will remain empty. The crane must be free and the stacked containers can be on the street or in the truck.

Do not write any other actions than those described here. You don't have to consider the operations to grasp and release container, i.e those actions which refer to tie up and untie the crane wires to a container as we will assume these operations are included in the pick-up and drop container.

A possible initial situation is "***There is an empty container (Container 1) in the truck and a full container (Container 2) on the street***". Write out a RBS in CLIPS to obtain the necessary sequence of actions which achieves the following final situation "***The full container is in the truck and the empty one on the street***".

1. Specify clearly the **patterns** used to define the Fact Base.
2. Show the initial and the final fact base
3. Specify the production rules

Do not write any type of actions other than those specified in the problem description. Do not write procedural code in the rules RHS.

Rules must be general enough (as many rules as possible actions, approximately). Do not write specific rules for a particular container or for concrete situations.

Solución

Especificación de la Base de Hechos:

(grua libre|ocupado camion libre|ocupado calle libre|ocupado
contenedor Contenedor1 camion|grua|calle vacio|C2
contenedor Contenedor2 camion|grua|calle|C1 lleno)

Base de Hechos inicial:

(grua vacia camion ocupado calle ocupado contenedor
Contenedor1 camion vacio contenedor Contenedor2 calle lleno)

Base de Hechos Objetivo:

(grua $?x contenedor ?C1 camion lleno $?y)
(grua $?z contenedor ?C2 calle vacio $?q)

La especificación en CLIPS de los hechos y reglas sería:

(deffacts inicio
(grua vacia camion ocupado calle ocupado contenedor Contenedor1 camion vacio contenedor Contenedor2 calle lleno))

(defrule coger-contenedor
?f1 <- (grua vacia $?x ?lugar ocupado $?y contenedor ?C1 ?lugar ?lleva $?z)
=>
(assert (grua llena $?x ?lugar libre $?y contenedor ?C1 grua ?lleva $?z)))

(defrule desapilar-contenedor
(grua vacia $?x contenedor ?C1 ?lugar ?C2 contenedor ?C2 ?C1 ?lleva)
=>
(assert (grua llena $?x contenedor ?C1 ?lugar vacio contenedor ?C2 grua ?lleva)))

(defrule dejar-contenedor
(grua llena $?x ?lugar libre $?y contenedor ?C1 grua ?lleva $?z)
=>
(assert (grua vacia $?x ?lugar ocupado $?y contenedor ?C1 ?lugar ?lleva $?z)))

(defrule apilar-contenedor
(grua llena $?x contenedor ?C1 ?lugar vacio contenedor ?C2 grua ?lleva)
=>
(assert (grua vacia $?x contenedor ?C1 ?lugar ?C2 contenedor ?C2 ?C1 ?lleva)))

(defrule objetivo
(declare (salience 100))
(grua $?x contenedor ?C1 camion lleno $?y)
(grua $?z contenedor ?C2 calle vacio $?q)
=>
(halt))

Exercise 9

Given a list of integer numbers, we want to obtain a new list where those numbers whose values do not coincide with their ordering positions are replaced by 0 (we assume the first number in the list takes up

position number 1). For example, given the list (list 15 2 4 6 5 3 7 8 15) the new list would be (list 0 2 0 0 5 0 7 8 0). Write a single rule in CLIPS to perform this task.

Solución:

```
(deffacts lista (lista 1 2 4 6 5 3 7 8 9))                ;Es un ejemplo de lista

(defrule uno
      ?f <- (lista $?x ?v $?y)
            (and  (test (<> ?v 0))                        ;Debe controlarse, para que pueda acabar!!
                  (test (<> (length $?x) (- ?v 1))))
=>
      (retract ?f)
      (assert (lista $?x 0 $?y)))
```

Exercise 10

Given a fact that represents a list of integer and non-ordered numbers, which may contain repeated numbers several times, write a single rule (if possible) to get the same initial list of numbers without repetitions.

Example:

WMinitial (lista 1 3 4 5 6 7 4 6 4 4 3 4 5 6 4 5 6 4 8 5 7 3 2 4 4 4 1 5 6 7 1 2 3 2 3 4 6 5 3 4 5 1 4 3)
WMfinal: (lista 2 7 6 5 4 3 1 8)

Solution:

Esta es una posible regla. Hay otras alternativas.

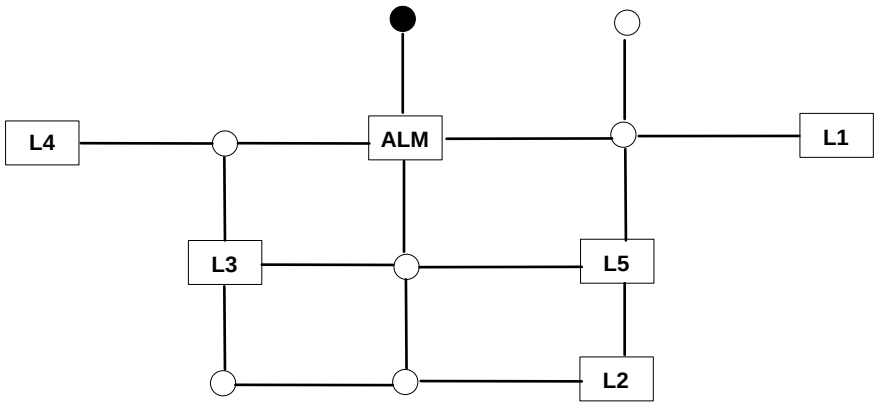
```
(defrule examen
   ?f1 <- (lista $?a ?x $?b ?x $?c)
   =>
   (retract ?f1)
   (assert (lista ?x $?a $?b $?c)))


(deffacts hechos
      (lista 1 3 4 5 6 7 4 6 4 4 3 4 5 6 6 4 5 6 4 8 5 7 3 2 4 4 4 1 5 6 7 1 2 3 2 3 4 6 5 3 4 5 1 4 3))
```

Exercise 11

We have 5 different stalls that order soft drink bottles and a distributor robot that delivers the bottles to the stalls. The robot uses a trolley for the delivering, which has a maximum capacity of 3 bottles. The robot must physically reach the stall in order to deliver the order in such a stall.

There exists a network of connections as the one shown in the figure. L1,…, L5 stand for the 5 stalls. ALM is the store where bottles are stored. The rest of nodes are intermediate points where the robot can be located. The robot can move between any pair of locations shown in the figure. The robot moves to any location directly connected to its current position in one move action.



When the robot is at the store (ALM), it can refill the trolley with new soft drink bottles. We will assume the robot always refills the trolley to the maximum of its capacity.

An example of initial state of the problem is: the robot is at any of the locations in the figure and the trolley is empty; one or several stalls place an order of a number of bottles. Stalls can order any number of bottles and the robot must deliver the order in one delivering or until out of stock. The final situation is as follows: all of the orders must have been delivered, the robot is back at the store (ALM) and the trolley is full of bottles.

Example of initial situation:

- The robot is located at the point shaded in black
- Stall L1 orders two bottles, L3 orders one bottle and L5 orders five bottles

Design a RBS to solve this problem with any possible initial configuration

- Specify the structure of the facts (patterns) that you will use in your design
- Specify the set of rules according to the design of the patterns

Solution

En el diseño del patrón se ha incluido la información de 'nivel' para poder mostrar el nivel del nodo solución, esto es, la longitud de la solución. También se incluye la variable ?*nod-gen* que almacena el número de nodos expandidos o reglas disparadas.

```
(defglobal ?*prof* = 20)
(defglobal ?*nod-gen* = 0)

(deffacts inicio
      (conexion Almacen p1)
      (conexion Almacen p4)
      (conexion Almacen p5)
      (conexion p2 p4)
      (conexion p4 L1)
      (conexion p3 Almacen)
      (conexion p3 L4)
      (conexion p5 L3)
      (conexion p3 L3)
      (conexion p5 L5)
      (conexion p4 L5)
      (conexion p8 L3)
       (conexion p8 p7)
       (conexion p5 p7)
       (conexion p7 L2)
       (conexion L5 L2)
      (robot situacion p1 botellas 0 solicitudes sol L1 2 sol L3 1 sol L4 5 nivel 0))

(defrule carga-robot
          (robot situacion Almacen botellas ?b $?x nivel ?n)
          (test (< ?b 3))
          (test (< ?n ?*prof*))
          =>
          (assert (robot situacion Almacen botellas 3 $?x nivel (+ 1 ?n)))
          (bind ?*nod-gen* (+ ?*nod-gen* 1)))



(defrule servir-peticion-completa
          (robot situacion ?s botellas ?b solicitudes $?x sol ?s ?bs $?y nivel ?n)
          (test (>= ?b ?bs))
          (test (< ?n ?*prof*))
          =>
          (assert (robot situacion ?s botellas (- ?b ?bs) solicitudes $?x $?y nivel (+ 1 ?n)))
          (bind ?*nod-gen* (+ ?*nod-gen* 1)))


(defrule servir-peticion-parcial
          (robot situacion ?s botellas ?b solicitudes $?x sol ?s ?bs $?y nivel ?n)
          (test (< ?b ?bs))
          (test (> ?b 0))
          (test (< ?n ?*prof*))
          =>
          (assert (robot situacion ?s botellas 0 solicitudes $?x sol ?s (- ?bs ?b) $?y nivel (+ 1 ?n)))
          (bind ?*nod-gen* (+ ?*nod-gen* 1)))
```

```
(defrule desplazarse
        (robot situacion ?sit $?x nivel ?n)
        (or (conexion ?sit ?dest)(conexion ?dest ?sit))
        (test (< ?n ?*prof*))
        =>
        (assert (robot situacion ?dest $?x nivel (+ 1 ?n)))
        (bind ?*nod-gen* (+ ?*nod-gen* 1)))


(defrule objetivo
        (declare (salience 100))
        (robot situacion Almacen botellas 3 solicitudes nivel ?n)
        =>
        (printout t "Solucion encontrada en el nivel: " ?n crlf)
        (printout t "NUMERO DE NODOS EXPANDIDOS O REGLAS DISPARADAS " ?*nod-gen* crlf)
        (halt))


(deffunction inicio ()
    (reset)
        (printout t "Profundidad Maxima:= " )
        (bind ?*prof* (read))
        (printout t "Tipo de Busqueda " crlf "    1.- Anchura" crlf "    2.- Profundidad" crlf )
        (bind ?a (read))
        (if (= ?a 1)
            then    (set-strategy breadth)
            else   (set-strategy depth))
    (printout t " Ejecuta run para poner en marcha el programa " crlf))
```

Exercise 12

A bingo board consists of 5 rows each containing 5 numbers. The fact base of a Production System contains a fact which represents a winner line, for example, (winner-line 12 21 34 56 77). Write a fact to represent the bingo board and a production rule to determine if there exists a winner line on the bingo board.

Solution:

Patrón para el cartón: (carton linea $x1^m$ linea $x2^m$ linea $x3^m$ linea $x4^m$ linea $x5^m$)
$x1, x2, x3, x4, x5 \in$ INTEGER

```
(defrule bingo
   (linea-ganadora $?x)
   (carton $?a linea $?x $?b)
 =>
   (printout t "Linea ganadora " crlf))
```

```
(deffacts datos
   (linea-ganadora 12 21 34 56 77)
   (carton linea 1 2 3 4 5 linea 23 44 55 66 77 linea 12 21 34 56 77 linea 6 7 8 9 10 linea 18 28 38 48 58))
```

Exercise 13

Let a RBS be composed of WMinitial={(elem e)(lista a b c d e f)}, and whose RB is:

```
(defrule R1
   ?l1 <- (elem ?e)
   ?l2 <- (lista $?a ?e $?b)
         =>
   (assert (lista ?e $?a $?b))
   (assert (cambio)))
```

```
(defrule R2
   ?l1 <- (cambio)
   (lista $?l)
         =>
   (retract ?l1))
```

assuming we apply a breadth-first strategy (more priority to older rule instances, starting by R1), which will be the final WM? Show the trace that follows from the inference process.

Solution:

Working Memory (WM)	Conflict Set (CS)
H1: (elem e) H2: (lista a b c d e f)	R1: H1, H2 (?e=e, $?a=a b c d, $?b=f)
==========	
H1, H2 H3: (lista e a b c d f) H4: (cambio)	R1: H1, H3 (?e=e, $?a={} $?b=a b c d f) R2: H4, H2 ($?l= a b c d e f) R2: H4, H3 ($?l= e a b c d f)
==========	
H1, H2, H3, H4 no se generan los hechos de los comandos assert porque son los mismos que H3 y H4	R2: H4, H2 ($?l= a b c d e f) R2: H4, H3 ($?l= e a b c d f) – se elimina al eliminar la instancia anterior el hecho H4 -
==========	
H1, H2, H3	
==========	

BH final= {H1: (elem e), H2: (lista a b c d e f), H3: (lista e a b c d f)}

Exercise 14

A “matrioska” doll or “Russian nested doll” is a set of dolls of decreasing sizes placed one inside the other. Each doll is identical in shape and painting, just slightly smaller than the one it is nested in, and so on up to a particular number of dolls.

We have **N** dolls, each of a different size, such that a doll of size 's' can be placed inside any other doll of size greater than 's'. We have also a table with a fixed number **M** of holes or positions to accommodate a maximum of **M blocks of dolls**. A block of dolls is a composition of one or more dolls (one inside the other) always satisfying the size restriction.

Initially, the N dolls are distributed in blocks of dolls along the M holes available on the table. The number of holes initially occupied may be lower than M. The goal is to form a single block with the N dolls in a hole of the table, if possible, taking into account the maximum number of holes to work with is M. In order to accomplish the goal, we have a robot which can perform the following moves:

1) There are two possible ways for the robot to grasp a block of dolls:
    a. Grasp the entire block of dolls in a hole on the table, or
    b. Open the lid of the most external doll in a block and take the dolls inside, i.e. take the x-1 dolls inside the block.

2) Once the robot is grasped a block of dolls (origin block), the robot can only:
    a. Insert the origin block into another block of dolls (destination block) as long as the size of the external doll of the origin block is smaller than the size of the most internal doll in the destination block, or
    b. Drop the origin block onto another hole of the table as long as the hole is empty

For example, if there is a 6-doll block (as the one shown in the figure) onto a hole on the table (dolls would be one inside the other, from the smallest to the biggest) then the robot could grasp the entire 6-doll block, or it could open the lid of the biggest doll and pick up the 5-doll interior block. Once the robot is grasped a block of dolls, it could insert such block into another block -if size restrictions are met, or it could drop the block on an empty hole of the table.

Here is an **example of initial situation**: we have 11 dolls and a table with 4 holes that allows for a maximum of 4 blocks of dolls. The 11 dolls are initially distributed in 3 blocks as follows: dolls of size 1, 2, 5, 7, 8 and 11 are placed in one block; dolls of size 3, 4 and 9 are placed in a second block of dolls; dolls of size 6 and 10 are placed in a third block.

Following the CLIPS syntax, design a RBS that returns, for a given initial state of N dolls and M holes on the table, ***whether the problem is solvable or not.***

1) Specify clearly the patterns to define the fact base (for any initial situation).

2) According to your specification, describe the initial fact base for the example shown above and the final fact base.
3) Write the production rules. In the LHS of the rule you can only use pattern-matching templates and tests. In the RHS of the rule you can only use commands "assert", "retract", "halt" and "printout". You are not allowed to use functions.
4) The RBS must return a message displaying whether it is possible or not to form a single block of dolls for a given initial situation.

NOTE. All rules must be general and no specifically designed for the initial or final state of a particular problem instance.

Solution

1) Especificación patrones

(maxbloques mb^s)

$mb^s \in$ INTEGER; indica el máximo número de bloques que se pueden formar en la mesa, o sea el máximo número de huecos que hay en la mesa. Este patrón representa un hecho estático ya que el máximo número de bloques o huecos de la mesa es fijo durante toda la ejecución de un problema.

(elementos x_1^s x_2^s … x_n^s)

$x_i^s \in$ INTEGER indica la i-muñeca, más concretamente el tamaño de la i-muñeca hasta un total de N muñecas. Este patrón representa también un hecho estático ya que el número de muñecas se mantiene fijo durante toda la ejecución de un problema.

(problema <bloque x^m finbloque>^{m} nbloques y^s robot z^m)

Este es el patrón principal y representa la información dinámica del problema. Se compone de:

a) un número variable de elementos **<bloque x^m finbloque>** que representa los bloques de muñecas que hay en la mesa; este número no podrá ser nunca inferior a 1 ni exceder el valor indicado en el hecho correspondiente al patrón (maxbloques mb^s). Si un robot coge el bloque entero de muñecas representado en **<bloque x^m finbloque>**, todos los elementos de **<bloque x^m finbloque>** se eliminarán del hecho principal.
b) $x^m \in 2^{[1\ ..\ N]}$ indica las muñecas (concretamente tamaños de muñecas) que hay en ese bloque. Este valor puede ser cualquier combinación de números enteros de 1 hasta N (máximo número de muñecas), siempre respetando la restricción de los tamaños.
c) $y^s \in$ INTEGER representa el número de bloques o huecos de la mesa ocupados en dicho estado
d) $z^m \in$ {EMPTY, $2^{[1\ ..\ N]}$} representa si el robot está libre o no; si está libre, z^m tomará el valor EMPTY; en caso contrario el valor de z^m será el bloque de muñecas que el robot tiene agarrado.

2) BH inicial y BH final

BHinicial={
(maxbloques 4)

(elementos 1 2 3 4 5 6 7 8 9 10 11)
(problema bloque 1 2 5 7 8 11 finbloque bloque 3 4 9 finbloque bloque 6 10 finbloque nbloques 3 robot empty) }

BHfinal={
(maxbloques 4)
(elementos 1 2 3 4 5 6 7 8 9 10 11)
(problema bloque 1 2 3 4 5 6 7 8 9 10 11 finbloque nbloques 1 robot empty) }

3) 4) Reglas incluido mensajes que tiene que devolver el SP

```
(deffacts datos
  (maxbloques 4)
  (elementos 1 2 3 4 5 6 7 8 9 10 11)
  (problema bloque 1 2 5 7 8 11 finbloque bloque 3 4 9 finbloque bloque 6 10 finbloque nbloques 3 robot empty))


(defrule sacar-bloque
   (problema $?x bloque $?b ?c finbloque $?y robot empty)
   (test (> (length $?b) 0))
    (test (not (member bloque $?b)))
 =>
   (assert (problema $?x bloque ?c finbloque $?y robot $?b)))


(defrule coger-bloque-entero
     (problema $?x bloque $?b finbloque $?y nbloques ?np robot empty)
     (test (> (length $?b) 0))
     (test (not (member bloque $?b)))
 =>
    (assert (problema $?x $?y nbloques (- ?np 1) robot $?b)))


(defrule meter-bloque
   (problema $?x bloque ?c1 $?b finbloque $?y robot $?r ?c2)
     (test (neq ?c2 empty))
     (test (not (member bloque $?b)))
     (test (< ?c2 ?c1))
 =>
    (assert (problema $?x bloque $?r ?c2 ?c1 $?b finbloque $?y robot empty)))


  (defrule hacer-bloque
    (problema $?x nbloques ?np robot $?r ?c2)
    (maxbloques ?maxp)
    (test (< ?np ?maxp))
    (test (neq ?c2 empty))
```

```
    =>
     (assert (problema $?x bloque $?r ?c2 finbloque nbloques (+ ?np 1) robot empty)))

(defrule final1
   (declare (salience 100))
    (elementos $?elem)
    ?f1 <- (problema $?x bloque $?elem finbloque $?y)
   =>
     (printout t "Todas las muñecas estan apiladas  " crlf)
     (halt))


 (defrule final2
   (declare (salience -5))
   =>
    (printout t "El problema no tiene solucion " crlf)
    (halt))
```

Exercise 15

Consider a RBS composed of WM={(S 0 list 2 3 1 20)} and whose Rule Base only contains the following rule:

```
(defrule exam
   ?f <- (S ?s list $?x ?y ?z $?w)
         (test (< ?z ?y))
=>
         (assert (S (+ ?s 1) list $?x ?z ?y $?w)))
```

1. Assuming a forward-chaining reasoning and a breadth-first strategy to order activations in the Conflict Set (more priority to older facts and activations), show the trace that follows from the inference process and indicate the contents of the final WM.

2. Assume that there is a command (retract ?f) in the RHS of the rule; show the new rule and final state of the WM.

Solución:

a)

Base de Hechos (BH)	**Conjunto Conflicto (CC)**
H1: (S 0 list 2 3 1 20)	exam: H1,?y=3, ?z=1, $?x=(2), $?w=(20) (1*)
==========	
H2: (S 1 list 2 1 3 20)	exam: H2, ?y=2, ?z=1, $?x=(), $?w=(3 20) (2*)

H3: (S 2 list 1 2 3 20)	No se producen más activaciones. De todas las posibles activaciones el patrón de la regla con el hecho H3, ninguna instanciación satisface el test.

La BH final contendrá tres hechos: BHfinal={(S 0 list 2 3 1 20) (S 1 list 2 1 3 20) (S 2 list 1 2 3 20)}

b) La traza será la misma excepto que se elimina de la BH el correspondiente hecho. De este modo, en cada ciclo del motor de inferencia, la BH contendrá únicamente un hecho. El contenido final de la BH será el hecho H3. BH final= BHfinal={(S 2 list 1 2 3 20)}.

Exercise 16

We have three towers (A1, A2 and A3) each containing a number of disks. Disks are all of different size. We will assume that the largest disk is of size 99. There exists a fourth tower (B) which is initially empty.

A disk d can be put in towers A1, A2 and A3 provided that the size of d is smaller than the below disk or that the tower is empty. However, a disk d' can only be put in tower B if the size of d' is greater than the below disk or tower B is empty.

We have a robot-arm to move one disk from one tower to another. The robot-arm can only access the top disk of any tower.

Design a RBS to have all disks properly ordered in tower B (from the largest to the smallest disk) for any possible initial situation; that is, for any number of disks in towers A1, A2 and A3 and tower B initially empty. An example of initial situation could be:

1) tower A1 contains disks 6 9 10
2) tower A2 contains disks 3 7 8
3) tower A3 contains disks 1 2 4
4) tower B is empty

, where numbers identify the disk size. In the final state, all disks should be ordered in tower B: {10 9 8 7 6 4 3 2 1}.

The design of a RBS that performs the minimal search possible will be valued.

Solution:

(towers A1 dk_a1^{m} 100 A2 dk_a2^{m} 100 A3 dk_a3^{m} 100 B dk_b^{m}) , dk_a1, sk_a2, dk_a3, d_b $\in$ INTEGER

El entero 100 se utiliza para indicar la base de la torre, un disco 'suelo' de tamaño máximo.

```
(deffacts initialdata
    (towers A1 6 9 10 100 A2 3 7 8 100 A3 1 2 4 100 B))

(defrule move-from-A1-to-B
     (towers A1 ?da1 $?ra1 A2 ?da2 $?ra2 A3 ?da3 $?ra3 B $?rb)
     (test (<> ?da1 100))
     (test (and (< ?da1 ?da2)(< ?da1 ?da3)))
=>
     (assert (towers A1 $?ra1 A2 ?da2 $?ra2 A3 ?da3 $?ra3 B ?da1 $?rb)))


(defrule move-from-A2-to-B
     (towers A1 ?da1 $?ra1 A2 ?da2 $?ra2 A3 ?da3 $?ra3 B $?rb)
     (test (<> ?da2 100))
     (test (and (< ?da2 ?da1)(< ?da2 ?da3)))
=>
     (assert (towers A1 ?da1 $?ra1 A2 $?ra2 A3 ?da3 $?ra3 B ?da2 $?rb)))


(defrule move-from-A3-to-B
     (towers A1 ?da1 $?ra1 A2 ?da2 $?ra2 A3 ?da3 $?ra3 B $?rb)
     (test (<> ?da3 100))
     (test (and (< ?da3 ?da1)(< ?da3 ?da2)))
=>
     (assert (towers A1 ?da1 $?ra1 A2 ?da2 $?ra2 A3 $?ra3 B ?da3 $?rb)))


(defrule final
   (declare (salience 100))
     (towers A1 100 A2 100 A3 100 $?)
  =>
    (halt))
```

Con el diseño realizado, solo existirá una regla aplicable por cada hecho introducido en la BH, realizando así la mínima búsqueda posible.

La regla 'final' no sería necesaria ya que con el diseño realizado, cuando se alcanza el estado final ninguna de las reglas move-from-…. serían aplicables.

Exercise 17

Consider a RBS composed of WM={(lista a a b a) (par a 1) (par b 2)} and whose Rule Base only contains the following rule:

```
(defrule R1
    ?f <- (lista $?x ?sym $?y)
          (par ?sym ?num)
```

=>
(retract ?f)
(assert (lista $?x ?num $?y))

Solution:

Base de Hechos	Agenda (instancias de reglas)
f-1:(lista a a b a) f-2:(par a 1) f-3:(par b 2)	R1:f-1,f-2 {?sym=a, $?x=(), $?y=(a b a), ?num=1, ?f=1} R1:f-1,f-2 {?sym=a, $?x=(a), $?y=(b a), ?num=1, ?f=1} R1:f-1,f-2 {?sym=a, $?x=(a a b), $?y=(), ?num=1, ?f=1} R1:f-1,f-3 {?sym=b, $?x=(a a), $?y=(a), ?num=2, ?f=1}
f-4:(lista 1 a b a)	R1:f-4,f-2 {?sym=a, $?x=(1), $?y=(b a), ?num=1, ?f=4} R1:f-4,f-3 {?sym=b, $?x=(1 a), $?y=(a), ?num=2, ?f=4} R1:f-4,f-2 {?sym=a, $?x=(1 a b), $?y=(), ?num=1, ?f=4}
f-5:(lista 1 1 b a)	R1:f-5,f-3 {?sym=b, $?x=(1 1), $?y=(a), ?num=2, ?f=5} R1:f-5,f-2 {?sym=a, $?x=(1 1 b), $?y=(), ?num=1, ?f=5}
f-6:(lista 1 1 2 a)	R1:f-6,f-2 {?sym=a, $?x=(1 1 2), $?y=(), ?num=1, ?f=6}
f-7: (lista 1 1 2 1)	

En la primera iteración se producen 4 instancias de R1, se dispara la primera de la agenda y se borran el resto por ser dependientes del hecho que se elimina f-1. En las siguientes iteraciones se produce sucesivamente una instancia menos ya que en cada iteración se reemplaza en el hecho (lista ...) un símbolo por un número. Independientemente de la estrategia de la agenda y el orden en el que se disparen las reglas, la BH final contendrá únicamente los hechos (lista 1 1 2 1)(par a 1)(par b2).

Exercise 18

This is a problem from the blocksworld domain. Blocks are arranged on a flat surface. We have 5 blocks labeled with letters A, B, C, D and E. All blocks are of the same size. Only one block can be stacked on top of another block thus forming a tower of any number of blocks. A robot-arm is used to manipulate the blocks (one at a time). The actions that the robot-arm can realize are:

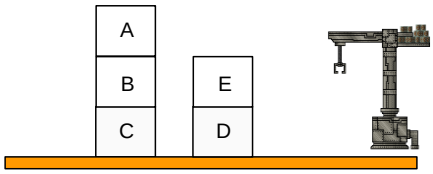
unstack(x,y): grabs the block x, which is on top of the block y. The robot-arm must be empty and the block x has no block onto it. The block y becomes clear after this action.

stack(x,y): puts the block x on top of block y. The robot-arm is holding the block x and the block y is clear. Once the block s is on top of block y, the block y is no longer clear.

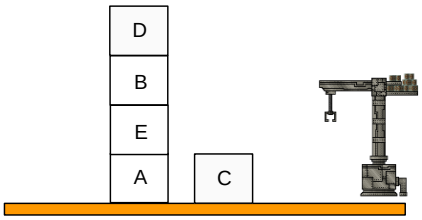
pickup(x): grabs the block x, which is on the table. The robot-arm must be empty and the block x has no block onto it.

drop(x): puts down the block x on the table. The robot-arm must be holding the block x.

Design a RBS that determines the sequence of actions of the robot-arm to reach a blocksworld configuration (goal state) from an initial blocks configuration (initial state). For example:



initial state (s0)



goal state

Solution:

Patrón: (brazo br^s [pila bl^m]^m)

donde:

$br^s \in$ {libre, A, B, C, D, …}
$bl^m \in$ {A, B, C, D, …} con la excepción de que debe haber un bloque como mínimo. Si la pila está vacía entonces no existe tal pila y el símbolo 'pila' tampoco aparecerá en el hecho.
[pila bl^m]^m : puede haber tantas pilas como queramos; de nuevo, el multi-valuado aquí es una excepción porque tiene que haber como mínimo una pila

Generamos un patrón para el estado final que es igual que el patrón anterior pero con el símbolo 'meta' delante.

```
(deffacts inicial
  (brazo libre pila a b c pila e d)
  (meta brazo libre pila d b e a pila c))


(defrule Apilar
  ?f1 <- (brazo ?x $?r1 pila ?y $?r2)   ;el brazo sostiene a ?x (bloque) e ?y esta en cabeza
         (test (not (eq ?x libre)))
=>
  (assert (brazo libre $?r1 pila ?x ?y ?r2))
  (printout t "Apilar " ?x " en " ?y crlf))


(defrule Desapilar
  ?f1 <- (brazo libre $?r1 pila ?x ?y $?r2)   ;brazo libre, ?y (bloque) sobre ?x
         (test (not (eq ?y pila)))

=>
  (assert (brazo ?x $?r1 pila ?y $?r2))
  (printout t "Desapilar " ?x crlf))
```

```
(defrule Coger
  ?f1 <- (brazo libre $?r1 pila ?x $?r2)  ;brazo libre, ?x solo en mesa
         (test (or (= (length$ $?r2) 0) (eq pila (nth$ 1 $?r2))))
=>
  (assert (brazo ?x $?r1 $?r2))
  (printout t "Coger " ?x crlf))


(defrule Dejar
  ?f1 <- (brazo ?x $?r1)    ;el brazo sostiene a ?x (bloque) y lo deja en mesa
       (test (not (eq ?x libre)))
=>
  (assert (brazo libre $?r1 pila ?x))
  (printout t "Dejar en mesa " ?x crlf))


(defrule final
  (declare (salience 100))
  (meta brazo libre $?v)
  (brazo libre $?v)
=>
  (printout t "Estado final alcanzado" crlf)
  (halt))
```

Exercise 19

A lineal puzzle is composed of three tiles labeled W1, W2 and W3, three tiles labeled B1, B2 and B3 and one empty space or blank. Let's suppose the initial puzzle configuration is the one shown in the following figure:

W1	W2	W3		B1	B2	B3

There are three legal move operators:

a) to put one tile in the blank (for instance, W3 or B1 can move to the blank)
b) to move one tile to the blank by jumping over another tile (for example, W2 or B2 would jump to the blank)
c) to move one tile to the blank by jumping over two tiles (for example, W1 or B3 would jump to the blank)

The goal is to get the next puzzle configuration:

B1	B2	B3		W1	W2	W3

The initial state can be formed by any distribution of the six tiles to the left or right of the blank, and the blank can be located at any position in the lineal puzzle.

Design a RBS to solve this problem with any initial puzzle configuration

- Specify the structure of the facts (patterns) that you will use in your design
- Specify the set of rules according to the design of the patterns

Solution:

We include in the pattern the information about the level (nivel) of the node in order to show then the length of the solution. We also include the variable ?*nod-gen* that keeps the number of expanded nodes or fired rules.

```
(defglobal ?*prof* = 20)
(defglobal ?*nod-gen* = 0)

(deffacts inicio (puzzle B1 B2 B3 V N1 N2 N3 nivel 0))

(defrule Mover-1-Izq
        (puzzle $?y ?x V $?z nivel ?n)
        (test (< ?n ?*prof*))
        =>
        (assert (puzzle $?y V ?x $?z nivel (+ 1 ?n)))
        (bind ?*nod-gen* (+ ?*nod-gen* 1)))

(defrule Mover-2-Izq
        (puzzle $?y ?x1 ?x2 V $?z nivel ?n)
        (test (< ?n ?*prof*))
        =>
        (assert (puzzle $?y V ?x2 ?x1 $?z nivel (+ 1 ?n)))
        (bind ?*nod-gen* (+ ?*nod-gen* 1)))

(defrule Mover-3-Izq
        (puzzle $?y ?x1 ?x2 ?x3 V $?z nivel ?n)
        (test (< ?n ?*prof*))
        =>
        (assert (puzzle $?y V ?x2 ?x3 ?x1 $?z nivel (+ 1 ?n)))
        (bind ?*nod-gen* (+ ?*nod-gen* 1)))

(defrule Mover-1-Der
        (puzzle $?y V ?x $?z nivel ?n)
        (test (< ?n ?*prof*))
        =>
        (assert (puzzle $?y ?x V $?z nivel (+ 1 ?n)))
        (bind ?*nod-gen* (+ ?*nod-gen* 1)))

(defrule Mover-2-Der
```

```
        (puzzle $?y V ?x1 ?x2 $?z nivel ?n)
        (test (< ?n ?*prof*))
        =>
        (assert (puzzle $?y ?x2 ?x1 V $?z nivel (+ 1 ?n)))
        (bind ?*nod-gen* (+ ?*nod-gen* 1)))

(defrule Mover-3-Der
        (puzzle $?y V ?x1 ?x2 ?x3 $?z nivel ?n)
        (test (< ?n ?*prof*))
        =>
        (assert (puzzle $?y ?x3 ?x1 ?x2 V $?z nivel (+ 1 ?n)))
        (bind ?*nod-gen* (+ ?*nod-gen* 1)))

(defrule Objetivo
        (declare (salience 100))
        (puzzle N1 N2 N3 V B1 B2 B3 nivel ?n)
        =>
        (printout t "Solucion encontrada en el nivel: " ?n crlf)
        (printout t "NUMERO DE NODOS EXPANDIDOS O REGLAS DISPARADAS " ?*nod-gen* crlf)
        (halt)
)

(deffunction inicio ()
    (reset)
        (printout t "Profundidad Maxima:= " )
        (bind ?*prof* (read))
        (printout t "Tipo de Busqueda " crlf "    1.- Anchura" crlf "    2.- Profundidad" crlf )
        (bind ?a (read))
        (if (= ?a 1)
            then    (set-strategy breadth)
            else   (set-strategy depth))
    (printout t " Ejecuta run para poner en marcha el programa " crlf))
```

Exercise 20 (exam 2013)

We have N heaps of containers in a terminal port and some of the containers must be loaded in the next ship. We distinguish between containers of type A, which must be loaded in the next ship, and containers of type B, otherwise. Given a state as the one shown in Figure 1, the goal of the RBS is to arrange the containers so that containers A do not have any container B onto them (see Figure 2).

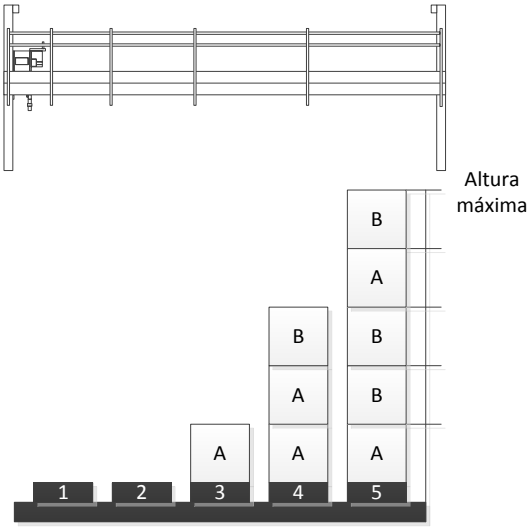


Figura 1. Situación inicial

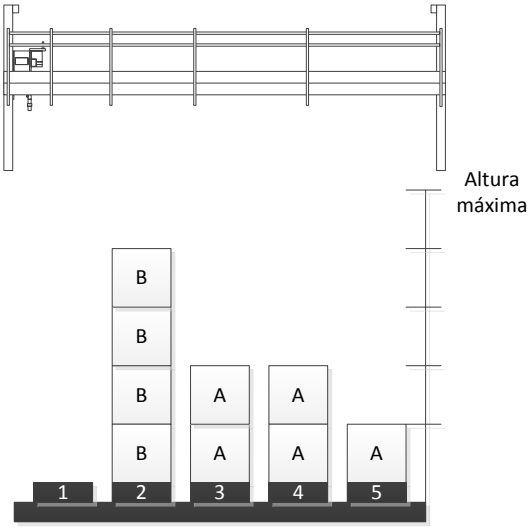


Figura 2. Una posible situación final

Containers can be moved from one heap to another by using a hoist that reaches all of the heaps. There are restrictions on the number of heaps as well as on the height of the heaps (see below).

Let's assume that we use this pattern to represent the information of a problem state:

(hoist G^s [heap N^s containers C^s B^m]^m)

, where:

- hoist, heap and containers are constant symbols
- [heap N^s containers C^s B^m] is a sub-pattern that represents the information in one heap. This sub-pattern is repeated for every heap in the state.
- G^s takes a value from the set {free, A, B}; that is, G^s shows whether the hoist is free or the type of the container it is holding
- N^s is the heap identifier; it is a natural number greater than 0 that shows the position of the heap in the tier (1 is the identifier for the heap on the left end and 'n' is the identifier for the heap on the right end as it is shown in the figures). Heaps are ordered from left to right, from the smallest to the highest identifier.
- C^s shows the number of containers in the heap
- B^m is the set of containers in the heap, the first element in the list is the top container

Additionally, we define facts to represent the restrictions on the number of heaps and the maximum height of the heaps:

(num-heaps X^s) (max-height Y^s)

where X^s e Y^s are integer values.

a) Write the Working Memory (facts) of the initial and final situation of figure 1 and 2, respectively. The representation of an empty heap is left open to each student. Take into account that the rules of sections b) and c) must then be compliant with this representation.
b) Write a rule to unstack a container B from a heap if this container is blocking a container A that is found beneath.

c) Write a rule to stack a container of either type (A or B) in a heap other than the one on the right end, and which already contains at least one container. Height of the rules must be compensated so a container will be stacked on a heap 'x' as long as the difference in height between 'x' and the heap on its right is less than two containers.

d) We want to measure the cost of all the operations necessary to reach a particular problem state. The cost of unstacking a container is 10 for containers A and 15 for containers B. The cost of stacking any container is 5. For example, the application of these costs would give a value of 95 for the state in figure 2:
    a. Unstack 4 containers B: 15*4 = 60
    b. Unstack 1 container A: 10*1 = 10
    c. Stack the 5 containers: 5*5 = 25
    d. TOTAL = 95

Make the proper modifications in the Working Memory (facts) to be able to represent this information.

NOTES:
- Rules must be valid for any heap and for states with any number of heaps
- You can make use of the following functions:
    (member$ <expression> <multifield-expression>)
    (length$ <multifield-expression>)
    (abs <numeric-expression>)

Solution:

Section a)

```
(deffacts inicio
        (hoist libre heap 1 containers 0 heap 2 containers 0 heap 3 containers 1 A heap 4 containers 3 B A A
heap 5 containers 5 B A B B A)
        (max-height 5)
        (num-heaps 5))
```

Situación final:

```
        (hoist libre heap 1 containers 0 heap 2 containers 4 B B B B heap 3 containers 2 A A heap 4
containers 2 A A heap 5 containers 1 A)
```

The facts (max-height 5) and (num-heaps 5) are static and do not change.

Section b)

```
(defrule unstack-B  ;;unstacks container B if there exists a container A beneath
        (hoist libre $?x heap ?p containers ?n B $?b $?y)
        (test (member A $?b))
        (test (= ?n (+ (length $?b) 1)))
=>
```

```
        (assert (hoist B $?x heap ?p containers (- ?n 1) $?b $?y)))
```

Section c)

```
(defrule stack
        (hoist ?c $?x heap ?p containers ?n $?b heap ?p2 containers ?n2 $?b2 $?y)
        (max-height ?max)
        (test (neq ?c libre))
        (test (> ?n 0))
        (test (= ?n (length $?b)))
        (test (= ?n2 (length $?b2)))
        (test (< ?n ?max))
        (test (< (abs (- ?n2 ?n)) 2))
=>
        (assert (hoist libre $?x heap ?p containers (+ ?n 1) ?c $?b heap ?p2 containers ?n2 $?b2 $?y)))
```

Section d)

```
(deffacts inicio
        (hoist libre heap 1 containers 0 heap 2 containers 0 heap 3 containers 1 A heap 4 containers 3 B A A
heap 5 containers 5 B A B B A cost 0)
        (max-height 5)
        (num-heaps 5)
        (cost unstack A 10 B 15)
        (cost stack A 5 B 5))
```

Exercise 21 (exam 2013)

A farmer has to cross a river with his fox, goose and grain. Each trip, his boat can only carry himself and one of his possessions. Additionally, an unguarded fox eats the goose and an unguarded goose eats the grain. Consequently, the farmer cannot leave the fox alone with the goose nor the goose alone with the grain.

We want to design a RBS to cross the farmer, fox, goose and grain from one bank to the river to the other. Assume that all of them are in bank A of the river in the initial state and have to cross to bank B.

a) Assuming the following pattern to represent the information of one state in a state-based representation problem:

   (problem farmer x^S fox x^S goose x^S grain x^S) where $x^S \in \{A, B\}$

   NOTE: Use only command 'assert' in the RHS of the rules

   a.1) By using CLIPS language, write a rule to cross the farmer alone from bank A to bank B of the river.

```
(defrule FarmerB  ; cross farmer, do not leave goose alone with the fox or the grain
  ?f <- (problem farmer A fox ?z  goose ?g grain ?t)
        (test   (or (eq ?g B)(and (eq ?t B)(eq ?z B))))
  =>
        (assert (problem farmer B fox ?z goose ?g grain ?t)))
```

a.2) By using CLIPS language, write a rule to cross the fox from bank A to bank B of the river.

```
(defrule ZorraD   ;cross fox to B, do not leave goose alone with the grain
  ?f <- (problem farmer A fox A goose ?g grain ?t)
        (test (or (eq ?g B)(eq ?t B)))
  =>
        (assert (problem farmer B fox B goose ?g grain ?t)))
```

b) Suppose the river is too wide to cross it from one bank to the other without intermediate stops. Assume we have {I1, I2, …, In} islets in the middle of the river.

b.1) without modifying the pattern in (a), add the necessary facts to the Working Memory to represent this new information so that the move rules are independent of the origin/destination (bank of the river, islet).

(problem farmer x^S fox x^S goose x^S grain x^S) $x^S \in \{A, B, I1, I2, \ldots, In\}$

and a set of facts representing the connections between the two banks of the river and the islets

(connected A I1)	(connected I1 A)
(connected I1 I2)	(connected I2 I1)
(connected I2 I3)	(connected I3 I2)
……..	…….
(connected I_{n-1} I_n)	(connected I_n I_{n-1})
(connected I_n B)	(connected B I_n)

b.2) considering the new information in b.1), modify the rule in a.2) to cross the fox from any origin to any destination, either any bank of the river or any islet.

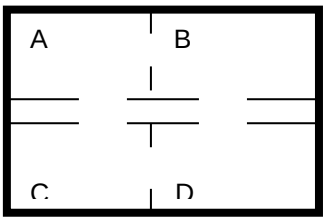
```
(defrule fox   ; cross fox from origin to destination, do not leave goose alone with the grain
  ?f <- (problem farmer ?origin fox ?origin goose ?g grain ?t)
        (test (not (and  (eq ?g ?origin )(eq ?t ?origin))))
        (connected ?origin ?destination)
  =>
        (assert (problem farmer ?destination fox ?destination goose ?g grain ?t)))
```

Exercise 21 (January 2015)

One floor of a building is distributed in 4 rooms (A, B, C and D) and a corridor which connects with the four rooms (see figure below). There are 2 robots which are responsible of delivering a cup of tea or a cup of coffee to several customers who are in the rooms. Robots are only allowed to move within a delimited area: robot 1 can only move across room A, room B and the corridor; robot 2 can only move across room C, room D and the corridor.

A vending machine of tea and coffee is found in room A. A cabinet with cups is found in room C. People who order a cup of tea or coffee can only be in rooms B or D. Robots can only hold a cup at a time. A robot X can pass on a cup (empty or full) to a robot Y if robot Y is not holding another cup.

The operators of this problem are: (a) the robot grabs an empty cup; (b) a robot pass on a cup (empty or full) to the other robot; (c) the robot fills the cup with tea or coffee according to the customer order; (d) a robot moves to an adjacent room or to the corridor; and (e) a robot delivers the cup of tea or cup of coffee ordered by the customer.



Given the following pattern to represent the dynamic information of this problem:

(problem robot 1 pos ?p1 carries ?l1 robot 2 pos ?p2 carries ?l2 order [customer ?p ?t]^m)

where, ?p1 ∈ {A, B, COR}, ?p2 ∈ {C, D, COR}, ?p ∈ {B, D}, ?l1, ?l2 ∈ {nothing, empty, tea, coffee} and ?t ∈ {tea, coffee}

a) (0.4 points) Write an initial working memory for this problem where robot 1 is in room A, robot 2 is in room D and there are two customers who have ordered a cup of coffee and a cup of tea, respectively.

(problem robot 1 pos A carries nothing robot 2 pos D carries nothing order customer B coffee customer B tea)
(connection robot 1 A COR) (connection robot 1 COR A) (connection robot 1 A B) (connection robot 1 B A) (connection robot 1 B COR) (connection robot 1 COR B)
(connection robot 2 C COR) (connection robot 2 COR C) (connection robot 2 C D) (connection robot 2 D C) (connection robot 2 D COR) (connection robot 2 COR D)

b) (0.2 points) Write an example of final working memory in which all of the orders have been served.

(problem robot 1 pos A carries nothing robot 2 pos D carries nothing order)
;; el resto de hechos son estáticos y se mantienen

c) (0.8 points) Write a single rule to move any robot between two points within its allowed area.

```
(defule move
  (local $?x robot ?rob pos ?p1 $?y)
  (connection robot ?rob ?p1 ?dest)
 =>
  (assert  (problem $?x robot ?rob pos ?dest $?y)))
```

d) (0.8 points) Write a single rule to pass on a cup (empty or full) between the two robots.

```
(defule switch_cup
 (problem robot 1 pos ?p1 carries ?l1 robot 2 pos ?p1 carries ?l2 $?y)
 (test (or (and (eq ?l1 nothing)(neq ?l2 nothing)) (and (eq ?l2 nothing)(neq ?l1 nothing))))
 =>
 (assert  (problem robot 1 pos ?p1 carries ?l2 robot 2 pos ?p1 carries ?l1 $?y)))
```

e) (0.8 points) Write a single rule to deliver the cup of tea or cup of coffee ordered by a customer.

```
(defule serve
 (problem $?x robot ?rob pos ?pos carries ?lle $?y customer ?pos ?lle $?z)
 =>
 (assert  (problem $?x robot ?rob pos ?pos carries nothing $?y $?z)))
```

Exercise 22 (January 2016)

We have three storehouses (A, B and C) in a city location, each having a set of packages to be transported to any of the other two stores. Hence, store A may have packages for B and/or C; store B may have packages for store A and/or C; and store C may have packages whose destination is store A and/or B. The problem goal is to deliver all packages to their destination store.

For the package transportation, there is only one truck available which can keep 10 packages at most. The truck can move between any pair of storehouses. When the truck is at store X, packages located at store X and whose destination is a store other than X can be loaded into the truck. Likewise, when the truck is at store X, only the packages whose final destination is X can be unloaded from the truck.

Example of initial situation:

- There are 7 packages in store A: 4 of them go to B and 3 go to C.
- There are 10 packages in store B: 7 of them go to A and 3 go to C.
- There are 6 packages in store C: 3 for A and 3 for B.
- The truck is initially in store A and it is empty.

Given the following pattern to represent the dynamic information of the problem;

(transport [store ?cit [destin ?dest ?num]^{m}]^{m} truck ?loc [?dest_pack ?num_pack]^{m} total ?tot)

, where:

?cit, ?loc ∈ {A,B,C}
?dest ∈ {A,B,C} such that ?dest ≠ ?cit ;; destination
?num ∈ INTEGER ;; number of packages to destination, even when the number is 0

?tot ∈ INTEGER ;; overall number of packages in the truck

?dest_pack ∈ {A,B,C} ;; destination (only if there are packages for such destination)
?num_pack ∈ INTEGER such that ?num_pack ≠ 0 ;; number of packages to destination (only if it is not 0)

NOTE 1: If the truck is not carrying packages for store X then the label [X 0] does not appear in the fact
NOTE 2: Given a store X, we only represent the packages to be moved to another store (we do not represent the packages of X)
NOTE 3: If necessary, you can add a static fact in any of the following questions.

a) (0.5 points) Write the initial fact base that represents the information described above.

```
(transport store A destin B 4 destin C 3 store B destin A 7 destin C 3 store C destin A 3
destin B 3 truck A total 0)
```

b) (1 point) Write a single rule that loads into the truck all the packages in store X that go to a destination Y. Assume the truck does not have a priori packages for Y. The restriction about the max capacity of the truck must be satisfied

```
(defrule load
   (transport $?x1 store ?alm $?y1 destin ?dest ?num $?y2 truck ?alm $?z total ?total)
    (test (> ?num 0))
    (test (not (member store $?y1)))
    (test (<= (+ ?total ?num) 10))
  =>
   (assert (transport $?x1 store ?alm $?y1 destin ?dest 0 $?y2 truck ?alm ?dest ?num $?z
total (+ ?total ?num))))
```

c) (0.8 points) Write a single rule that displays a message for each destination for which the truck is not carrying packages. You must show a message like: "The truck has not packages for destination XXXX ", for each destination that satisfies this condition.

We generate the static fact (destinations A B C)

```
(defrule display
   (transport $?x truck ?loc $?z total ?tot)
   (destinations $? ?d $?)
   (test (not (member ?d $?z)))
 =>
   (printout t "The truck has not packages for destination " ?d crlf))
```

d) (0.7 points) Write a single rule that unloads from the truck in store X the packages whose destination is X. The rule must be valid for any destination and the new resulting fact must not contain the label of the destination nor the number of packages; that is, once the packages are unloaded at X, the label [X 0] of the truck must not appear in the new fact.

```
(defrule unload
   (transport $?x truck ?loc $?y ?loc ?elem2 $?z total ?tot)
 =>
   (assert (transport $?x truck ?loc $?y $?z total (- ?tot ?elem2))))
```

Exercise 23 (January 2017)

We want to design a RBS for handling the collection of stickers of a number of children. A sticker is represented by an alpha-numerical identifier (e.g., A1, B3, C2, etc.). A child has a number of stickers (including repeated stickers) and the number of children is not limited in the application. An example is:

- Kid 1 has the stickers: A2 A4 A5 B1 A2 B3
- Kid 2 has the stickers: B3 A4 C2 C1 B3 C2
- Kid 3 has the stickers: C2 C4 B1 A2

The dynamic information of the problem is represented with facts that follow this pattern:

(collections [kid ?n [?id-sticker]^m fkid ?n]^m)

where:

?n ϵ INTEGER ;; kid identifier
?id-sticker ϵ {A1, A2, B1,…} ;;sticker identifier

Answer the following questions:

a) (0.3 points) Write the facts of the Working Memory that represent the above example.

```
(deffacts datos
    (collections  kid 1 A2 A4 A5 B1 A2 B3 fkid 1
            kid 2 B3 A4 C2 C1 B3 C2 fkid 2
            kid 3 C2 C4 B1 A2 fkid 3))
```

b) (1 point) Write a single rule for two kids to swap a sticker. Swapping a sticker is only allowed if kids hand over a repeated sticker of their collections and they get in return a sticker which is not in their collections.

```
(defrule intercambio
    (collections $?x kid ?n1 $?c1 ?c $?c2 ?c $?c3 fkid ?n1
                 $?y kid ?n2 $?p1 ?p $?p2 ?p $?p3 fkid ?n2 $?z)
  (test (neq ?c ?p))
  (test (and (not (member$ ?c $?p1))(not (member$ ?c $?p2))(not (member$ ?c $?p3))))
  (test (and (not (member$ ?p $?c1))(not (member$ ?p $?c2))(not (member$ ?p $?c3))))
   =>
    (assert (collections $?x kid ?n1 $?c1 ?c $?c2 ?p $?c3 fkid ?n1 $?y kid ?n2 $?p1 ?p
$?p2 ?c $?p3 fkid ?n2 $?z)))
```

Another solution:

```
(defrule intercambio
  ?f1 <- (collections $?x kid ?n1 $?c1 ?c $?c2 ?c $?c3 fkid ?n1
                 $?y kid ?n2 $?p1 ?p $?p2 ?p $?p3 fkid ?n2 $?z)
  ?f2 <- (collections $?x kid ?n1 $?todo1 fkid ?n1
                $?y kid ?n2 $?todo2 fkid ?n2 $?z)
  (test (eq ?f1 ?f2))
  (test (neq ?c ?p))
  (test (and (not (member$ ?p $?todo1))(not (member$ ?c $?todo2))))
 =>
  (assert (collections $?x kid ?n1 $?c1 ?c $?c2 ?p $?c3 fkid ?n1 $?y kid ?n2 $?p1 ?p
$?p2 ?c $?p3 fkid ?n2 $?z)))
```

c) (0.7 points) Write a single rule to display the children that have a sticker exactly three times in their respective collections. The rule must display a message per kid and sticker; example: "The kid " ?n " has the sticker " ?x " three times".

```
(defrule acaparador
  (collections $? kid ?n $?x1 ?y1 $?x2 ?y1 $?x3 ?y1 $?x4 fkid ?n $?)
  (test (and (not (member ?y1 $?x1))(not (member?y1 $?x2))
             (not (member ?y1 $?x3))(not (member ?y1 $?x4))))
 =>
 (printout t "The kid " ?n " has the sticker " ?y1 "  three times " crlf))
```

d) (1 point) Let's suppose that the WM contains facts that follow the pattern (special ?id-sticker) to indicate that the sticker identified with ?id-sticker is a special sticker. Write a single rule to calculate the number of children who have at least two special and different stickers. The result of executing the rule will be a fact that follows this format: (special-list [?n]^{m}), where ?n is the identifier of a kid who has at least two different special stickers. The kid identifier must appear only once in the list (even though the kid has several special stickers). Assume that the WM contains several facts with the format (special ?id-sticker) and the fact (special-list).

```
(defrule especiales
    (collections $? kid ?n1 $? ?a $? ?b $? fkid ?n1 $?y)
    (special ?a)
    (special ?b)
    (test (neq ?a ?b))
?r <- (special-list $?z)
    (test (not (member$ ?n1 $?z)))
 =>
   (retract ?r)
   (assert (special-list $?z ?n1)))
```

Exercise 24 (January 2019)

In an airport, several pickup luggage trains are available to take the suitcases (bags) from the check-in area to the plane assigned to the flight of the suitcases. A checked bag carries the label of its flight. Initially, the trains are not assigned to any flight. The flight assigned to a train will be the flight of the first bag loaded in the train. A train can only transport bags for one flight, and each flight is assigned to only one train.

The pattern to represent the dynamic information of a problem state is:

(airport [TRAIN num^{s} dest^{s} bag^{m}]^{m}) where

num $\in$ INTEGER;; is the train identifier

dest ∈ {none, F1, F2, F3, …} ;; is a symbol that represents the flight assigned to a train (initially, the flight of the train is unknown and the value of this field will be none).
bag ∈ {B1, B2, B3,…};; is a symbol that represents the bag identifier (initially, this field is empty)

A possible initial situation for this problem is:

- There are five bags (B1, B2, B3, B4 and B5); the first two bags are cheked for flight F14; the third bag is checked for flight F2 and the last two bags go to flight F10
- There are three available luggage trains in the airport. Initially, the trains are empty.

We want to solve this problem through a search process in a state space by designing a RBS in CLIPS:

1) (0.7 points) Write the facts of the Working Memory (WM) that represents the situation given above. Specify also the patterns that you need to represent the static information of the problem and include in the WM the facts associated to these patterns.

```
(deffacts datos
  (destino M1 F14)
  (destino M2 F14)
  (destino M3 F2)
  (destino M4 F10)
  (destino M5 F10)
  (airport TRAIN 1 none TRAIN 2 none TRAIN 3 none))
```

2) (1 point) Write a single rule to load the first bag in a train and assign the flight of the loaded bag to the train.

```
(defrule tren_vuelo
  (destino ?mal ?flight)
  (airport $?y TRAIN ?n ?dest $?z)
  (test (eq ?dest none))
  (test (not (member ?flight $?y)))
  (test (not (member ?flight $?z)))
 =>
  (assert (airport $?y TRAIN ?n ?flight ?mal $?z)))
```

3) (0.8 points) Write a single rule to load a bag in a train when the train is already assigned to a flight. You must check that the flight of the bag is the same as the flight of the train and that the bag is not already loaded in the train.

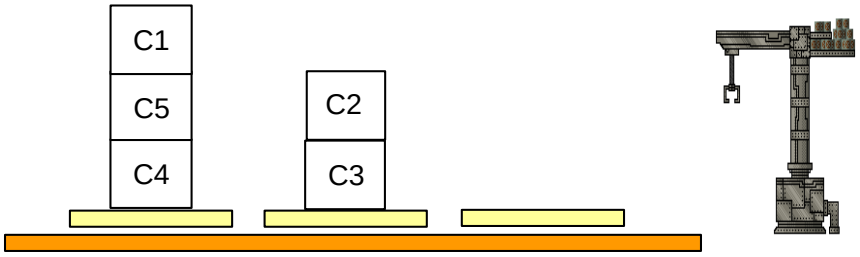
```
(defrule maleta_tren
   (destino ?mal ?flight)
   (airport $?x TRAIN ?n ?flight $?maletas)
   (test (not (member ?mal $?maletas)))
  =>
   (assert (airport $?x TRAIN ?n ?flight ?mal $?maletas)))
```

4) (0.5 points) Let be the pattern (flight f^s), where $f^s \in$ {F1, F2, F3, …} is a symbol that represents the flight identifier, and let's assume a fact that denotes a particular flight. Write a single rule that displays all the bags loaded in the train assigned to such a flight. The rule must display only one message of the type: "The bags X X X are loaded in the train Y".

```
(defrule listado_maletas
   (flight ?flight)
   (airport $? TRAIN ?n ?flight $?maletas $?resto)
   (test (not (member TRAIN $?maletas)))
   (test (or (= (length$ $?resto) 0) (eq (nth$ 1 $?resto) TRAIN)))
  =>
   (printout t "The bags " $?maletas " are loaded in the train "  ?n crlf))
```

Exercise 25 (January 2019)

In the harbor of a city there exists a series of containers stacked in several piles. An example of an initial situation is shown in the figure, where there is a maximum of three piles: containers C1, C5 and C4 are in pile 1, containers C2 and C3 are in pile 2 and the third pile is empty. Containers are arranged in a pile such that a container Ci is on top of a container Cj if the weight of Ci is smaller than the weight of Cj.



There is also a crane that can grab a tower of n containers, where n <= 3, and n can be all the containers of a pile. We will assume that the containers grabbed by the crane are in the same position as they appear in the pile. For example, if the crane grabs the tower of containers [C1 C5], container C5 will be the base of the tower and C1 will be the top of the tower. If the crane grabs

the tower of containers [C1 C5 C4], the base of the tower will be C4 and the top container of the tower will be C1.

The tower of containers grabbed by the crane can be placed in an empty pile or over a container Ci whenever the weight of the base container is smaller than the weight of Ci. For example, let's suppose the crane grabs the tower [C1 C5]; then, the crane can place [C1 C5] on top of C2 if the weight of C5 is smaller than the weight of C2.

The pattern to represent the dynamic information of a problem state is:

(harbor [pile num^{s} cont^{m} endpile]^{m} crane cra^{m}) where

num ∈ INTEGER ;; is a number that identifies the pile
cont ∈ {C1, C2, C3,…} ;; is a symbol that represents the container; this field represents the containers of the pile
cra ∈ {C1, C2, C3,…} ;; represents the containers grabbed by the crane

We want to solve this problem through a search process in a state space by designing a RBS in CLIPS:

1) (0.7 points) Write the facts of the Working Memory (WM) that represents the situation given above. Specify also the patterns that you need to represent the static information of the problem and include in the WM the facts associated to these patterns.

```
(deffacts datos
  (harbor pile 1 C1 C5 C4 endpile pile 2 C2 C3 endpile pile 3 endpile crane)
  (weight C1 10)
  (weight C5 20)
  (weight C4 30)
  (weight C2 33)
  (weight C3 35))
```

2) (0.7 points) Write a rule for a crane to grab all the containers of a pile. The rule must satisfy the restriction that the number of containers of the pile is <= 3.

```
(defrule coger_n_bloques
   (harbor $?x pile ?num $?resto endpile $?y crane)
   (test (and (<= (length$ $?resto) 3) (>= (length$ $?resto) 1)))
   (test (not (member pile $?resto)))
 =>
   (assert (harbor $?x pile ?num endpile $?y crane ?top $?resto)))
```

3) (0.8 points) Assuming the crane is holding a tower of containers, write a rule to place the tower of containers in a non-empty pile that has at least one container. The rule must satisfy the constraint on the weights explained above in the wording of the problem.

```
(defrule dejar_bloques
  (harbor $?x pile ?num ?top $?resto endpile $?y crane $?restrob ?base)
  (weight ?top ?pestop)
  (weight ?base ?pesbas)
  (test (not (member pile $?resto)))
  (test (< ?pesbas ?pestop))
 =>
  (assert (harbor $?x pile ?num $?restrob ?base ?top $?resto endpile $?y crane)))
```

4) (0.8 points) Write a rule that displays the containers of a pile. The rule must show a message like "The container Y is in pile X" for every container of a pile.

```
(defrule mostrar
  (harbor $?x pile ?num $?p1 ?con $?p2 endpile $?y crane)
  (test (not (member pile $?p1)))
  (test (not (eq ?con pile)))
  (test (not (member pile $?p2)))
 =>
  (printout t "The container " ?con " is in pile " ?num crlf))
```