Block 1 – Knowledge Representation and Search

Chapter 1

Rule-based systems (RBS). Representation in RBS: facts and rules. Pattern-matching.

Block1: Chapter 1- Index

1. Knowledge Representation
2. Rule-based systems (RBS)
3. CLIPS: a tool for building RBS
4. Working memory in CLIPS
5. Rule base in CLIPS
6. Pattern-matching
7. Exercises

Bibliography

- Capítulo 3: *Sistemas Basados en Reglas*. Inteligencia Artificial. Técnicas, métodos y aplicaciones. McGraw Hill, 2008.
- CLIPS User's guide
- CLIPS Basic Programming guide

1. Knowledge Representation

Knowledge representation: specific knowledge about the problem (domain-dependent knowledge) + general knowledge about how to solve the problem.

Two types of knowledge: declarative and procedural

Declarative description

- Set of clauses describing properties: WHAT something is (e.g.: list of ingredients for a cake)
- Knowledge representation is independent of how it will be later used (separation of knowledge and control)
- Types of declarative knowledge: relational, inheritable, deducible

Procedural description

- Set of procedures to describe how to use knowledge: HOW to do something (e.g.: recipe for baking a cake)
- Procedural knowledge includes information about how to use the knowledge (knowledge and control are mixed)

Network representation (declarative): semantic networks, conceptual graphs, concept maps
Structural representation (declarative & procedural): scripts, frames & demons, objects
Logic-based representation (declarative): first-order logic, logic programming
Rule-based representation (declarative & procedural): rules, production systems (Rule-Based Systems, Knowledge-Based Systems)

2. Rule-Based systems (RBS)

Also called Production Systems: systems based on rules

RBS are composed of three elements:

- Rule base (includes rules) which represent how the world changes (problem actions)
- Working memory (includes facts) which represent the elements of the problem
- Inference engine: control mechanism of a RBS

A special type of if-then rule:

- $p1 \wedge p2 \wedge \ldots \wedge pn \Rightarrow a1 \wedge a2 \wedge \ldots \wedge an$
- Antecedent (preconditions or premise): a conjunction of conditions (statements in predicate logic, conditions on objects/variables, …)
- Consequent (effects, conclusion or action) : a conjunction of actions (the effects or new resulting states are expressed through actions). An action can be:
  - *add* a fact to the working memory
  - *remove* a fact from the working memory (**Different from logic!!!!**)
    - Negation by failure (what's not in the working memory, it is false!)
  - *query* the user
  - *external actions* (print, read, ….)

2. Rule-Based Systems: components

Working memory

- Also called short-term memory, database or Fact Base (declarative knowledge).
- Contains ***facts*** about the world which can be observed directly or inferred from a rule. Facts are temporary knowledge which may be modified by the rules (add new facts, delete facts)
- Examples: 'the red block is on top of the green block', 'the list contains the elements a b c d and e', 'the tank is empty'

Rule Base

- Also called Knowledge Base (procedural knowledge)
- Contains ***rules,*** each rule is a step in a problem solving process (inference step). A rule is a collection of conditions (preconditions) and the operations to be taken if the conditions are met (effects).
- Rules are persistent knowledge about the domain. Typically only modified from the outside of the system, e.g. by an expert on the domain.
- Rules are used to represent the problem actions. Example ('pick-up a block')

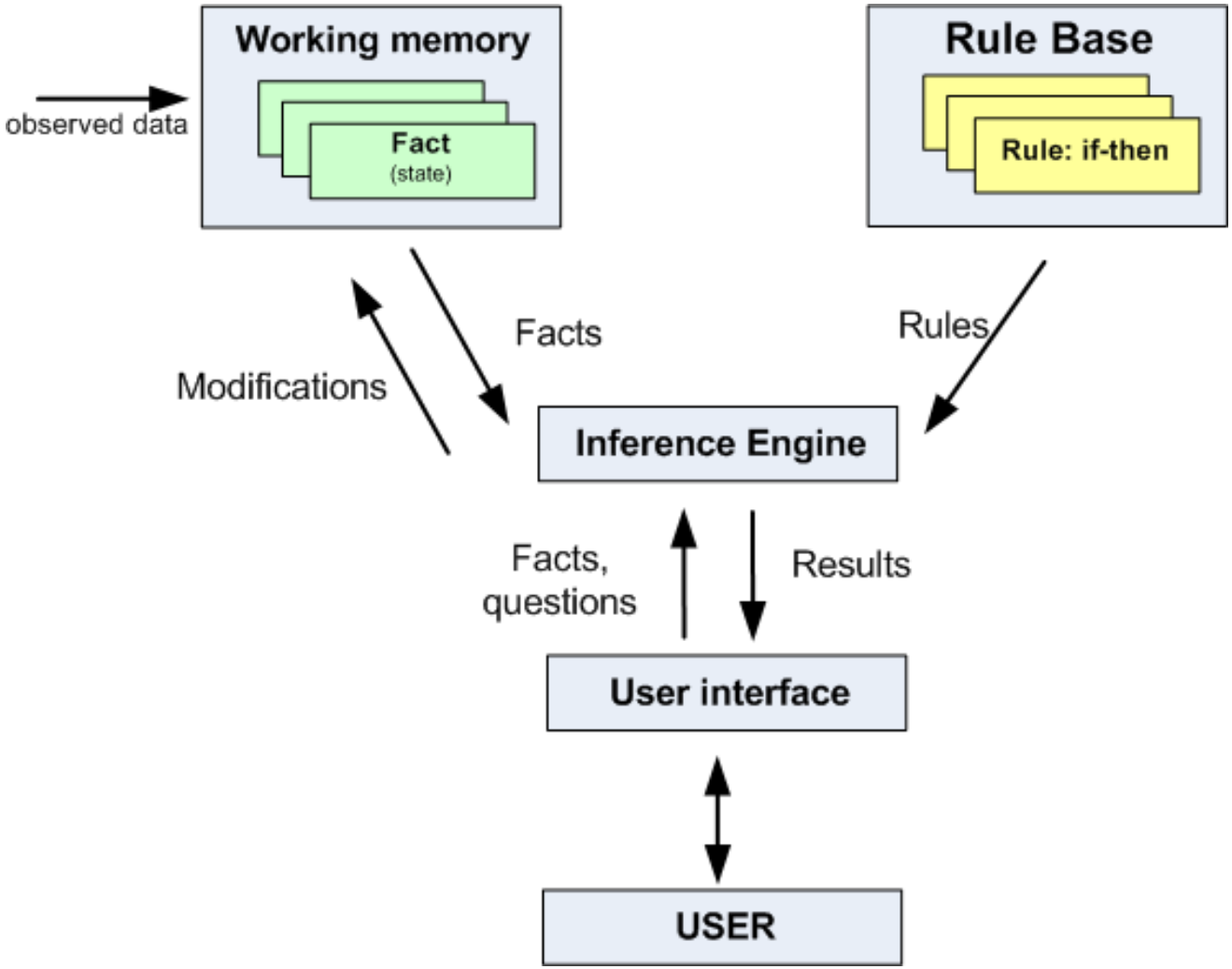
if the red block is on top of the green block and
there is no block on top of the red block
=>
add (the red block is held by the crane)
remove (the red block is on top of the green block)
add (there is no block on top of the green block)

2. Rule-Based Systems: components

Inference engine

- Processes the information in the Working Memory and Rule Base
- It is the domain-independent reasoning mechanism for RBS
- Selects a rule from the Rule Base to apply.
- An inference engine is defined by:
  - The type of reasoning chaining
  - The process of pattern-matching
  - The control mechanism for selecting and executing rules
- Two types of inference engine:
  - Forward chaining or data-driven reasoning (rules match the antecedent and infer the consequent)
  - Backward chaining or goal-driven reasoning (rules match the consequent and prove the antecedent)

2. Rule-Based Systems: architecture



3. CLIPS: a tool for building RBS

We will design RBS by using CLIPS

CLIPS (C Language Implementation Production System) is an expert system shell, a tool for building RBS, originally developed by the Software Technology Branch (STB), NASA/Lyndon B. Johnson Space Center.

CLIPS provides:

- *facts* for building the Working Memory (Fact Base)
- *rules* for building the Rule Base
- support for procedural knowledge and object-oriented representations
- a forward-chaining inference engine based on Rete match algorithm
- different strategies for solving conflicts (selecting a rule instance)

4. Working Memory (Fact Base) in CLIPS

Facts are the elementary information items. The Working Memory (WM) is a list of facts.

A fact is a list of atomic values that are referenced positionally (ordered fact)

Ordered fact: symbol followed by a sequence of zero or more fields separated by spaces and delimited by an opening parenthesis on the left and a closing parenthesis on the right. The first field of an ordered fact specifies a "relation" that applies to the remaining fields in the ordered fact.

<fact> ::= (<symbol> <constant>*)

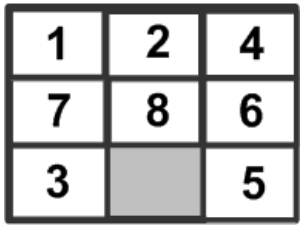
<constant> ::= <symbol> | <string> | <integer> | <float> | <instance-name>

Examples of facts:

(empty)
(on blockA blockB)
(on blockB table)
(grocery-list bread milk eggs)
(person name "John" age 24 activity journalist)
(water-jug jug X contents 0 capacity 4 jug Y contents 0 capacity 3)
(puzzle 1 2 4 7 8 6 0 3 5)

4. Working memory in CLIPS

Construct to define an initial group of facts : **(deffacts <deffacts-name> [<comment>] <fact>*)**

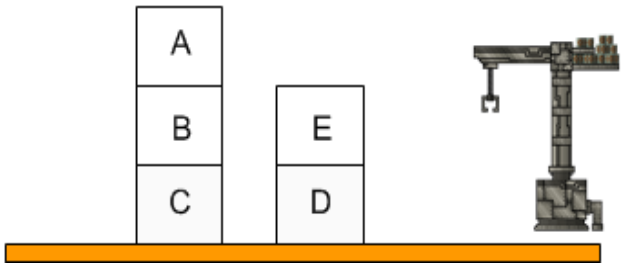


(deffacts puzzle_Rep1
(puzzle 1 2 4 7 8 6 3 0 5)}

WM={ f-1: (puzzle 1 2 4 7 8 6 3 0 5)}

(deffacts puzzle_Rep2
(puzzle 1 1 1)
(puzzle 1 2 2)
(puzzle 1 3 4)
(puzzle 2 1 7)
…)

WM={ f-1: (puzzle 1 1 1)
f-2: (puzzle 1 2 2)
f-3: (puzzle 1 3 4)
f-4: (puzzle 2 1 7)
… }



(deffacts blocks_Rep1
(tower maximum height 3)
(blocks tower 1 A B C tower 2 E D hoist nothing))

(deffacts blocks_Rep2
(tower maximum height 3)
(clear A)
(on A B)
(on B C)
(on C table)
(hoist nothing) ….)

5. Rule Base in CLIPS

The Rule Base contains a list of rules

Rules consist of two parts:

- The antecedent, premise or condition (Left-Hand Side –LHS- of the rule): it represents the conditions that must hold in the current state for the rule to be applicable
- The consequent, conclusion or action (Right-Hand Side –RHS- of the rule): it represents the actions to execute over the current state provided that the conditions of the LHS of the rule are satisfied (applicable rule), and the rule is selected for execution

```
(defrule <rule name> ["comment"]
   <conditional-element>*          ; left-hand side (LHS)  of the rule
                                   ; conditions to be satisfied
=>
   <action>*)       ; right-hand side (RHS)  of the rule
                     ; actions to be performed when the rule fires (the rule is
                    ;  selected for execution)
)
```

5. Rule base: LHS of rules

Conditional elements in the LHS of a rule can be of two types:

- Pattern: restriction which is used to determine which facts satisfy the condition specified by the pattern. Patterns amounts to the use of ***first-order predicate logic***. Patterns are matched against facts.
- Test: a query, a boolean expression that evaluates to TRUE or FALSE

```
(defrule fill-up-jug-X
      (capacity jug X ?cap)
      (contents jug X ?x jug Y ?y)
      (test (< ?x ?cap))
 =>
    ....
```

```
(defrule fill-X-from-Y
      (contents jug X ?x jug Y ?y)
      (capacity jug X ?cap)
      (test (< ?x ?cap))
      (test (>= (+ ?x ?y) ?cap))
 =>
    ....
```

All patterns and tests in the LHS must fulfill for the rule to be applicable!

1. A pattern fulfills if at least one fact in the WM matches the pattern
2. A test fulfills if it is evaluated to TRUE

5. Rule Base in CLIPS: RHS of rules

The RHS of a rule represents **procedural knowlege** (executable commands).

- assert: adds facts in the WM **(assert <fact>+)**
- retract: deletes facts from the WM **(retract <fact-index>+)**

WM= { f-1: (block A colour red) f-2: (block B colour blue) f-3: (available paint green) f-4: (available paint red) }

```
(defrule paint-block
    ?fa <- (block A colour red)
    ?fb <- (available paint green)
  =>
    (retract ?fa ?fb)
    (assert (block A colour green))
    (printout t "Block painted in green " crlf))
```

?fa and ?fb are the variables that store the index of the facts that match the patterns (?fa=1, ?fb=3)

when the rule is fired and the RHS is executed, the command 'retract' deletes the facts with indexes ?fa and ?fb, the facts that match the two patterns, respectively

WM= {f-2: (block B colour blue) f-4: (available paint red) f-5: (block A colour green)}

6. Pattern-matching

Pattern-matching (*syntactic unification*). It is the process of matching patterns of the LHS of rules against facts in the Working Memory. CLIPS inference engine automatically matches patterns against the current state of the Working Memory and determines which rules are applicable (rules whose LHS is satisfied).

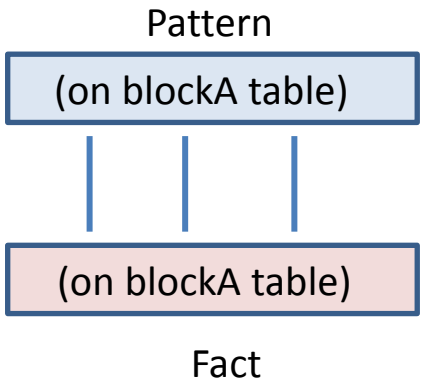
Patterns can contain:

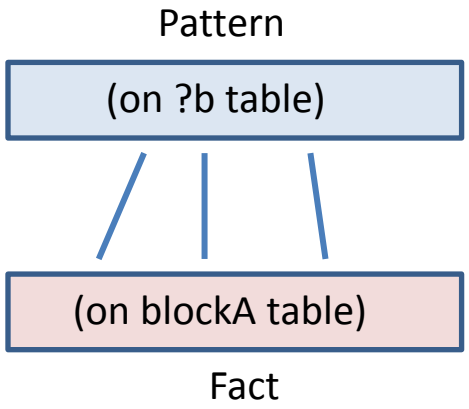
- Constants
- Single-valued variables (or single-field variables): variable name beginning with a question mark '?'. Single-valued variables are bound to a single value.
- Multi-valued variables (or multi-field variables): variable name beginning with the sign '$?'. Multi-valued variables are bound to 0 or more values.
- Wildcards. Their name is simply the symbol '?' or '$?'. These are special variables which can be bound to any constant or set of constants but without holding their bindings (they avoid keeping the variable value if this is not necessary)

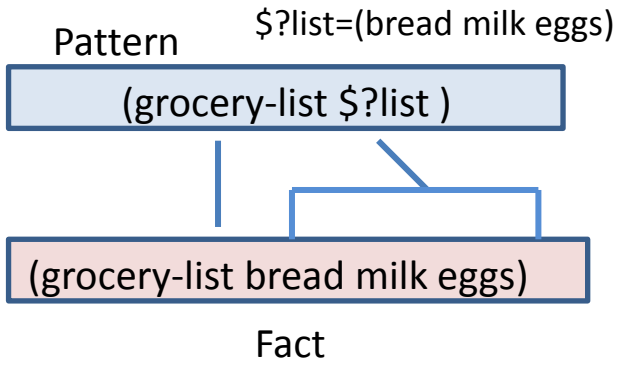
Matching

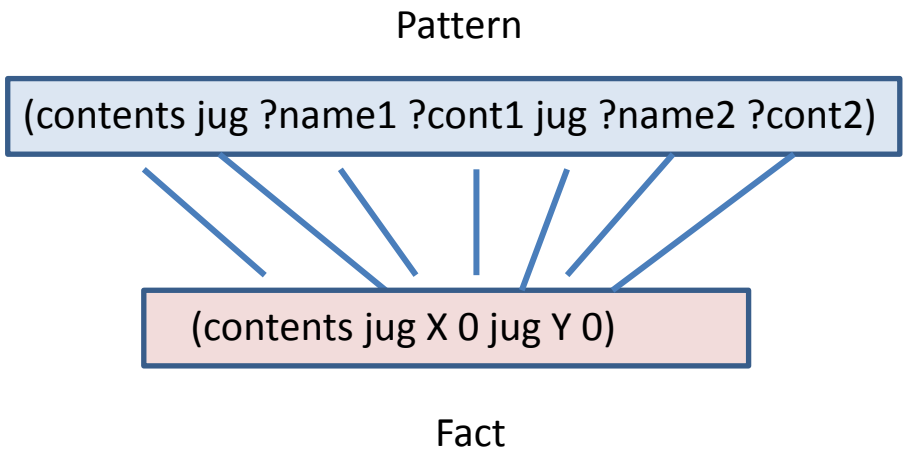
Element in the pattern	Element in the fact
Constant	Constant
Single-valued variable (?x)	One single element
Multi-valued variable ($?x)	Zero or more elements
Single-valued wildcard (?)	One single element without holding the binding
Multi-valued wildcard ($?)	Zero or more elements without holding the bindings

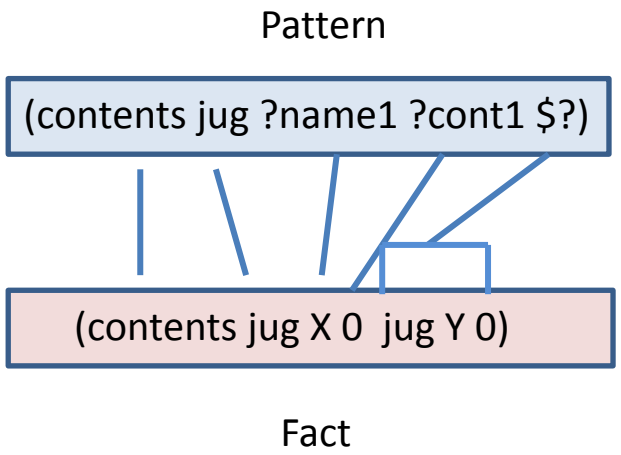
6. Pattern-matching



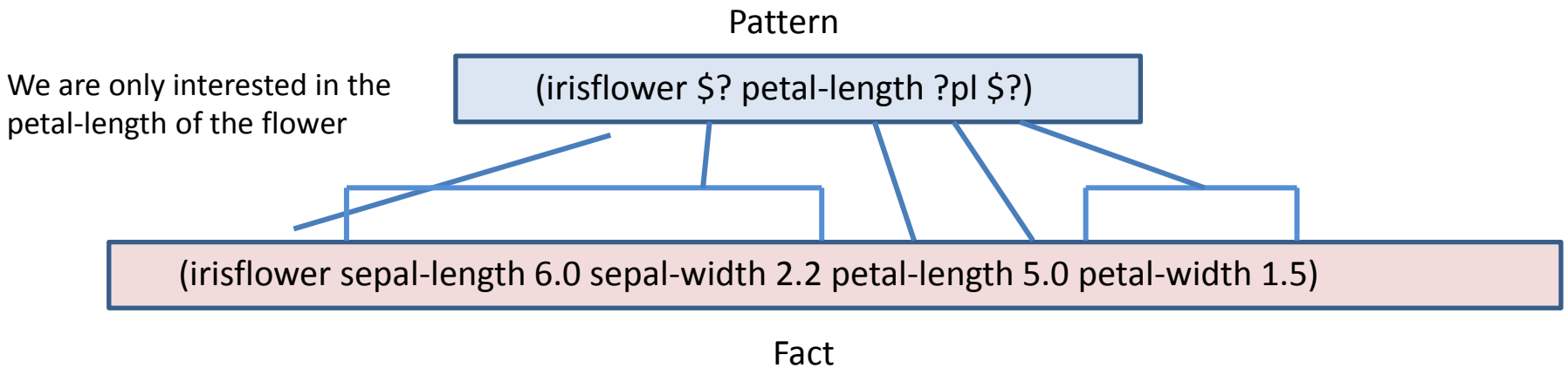




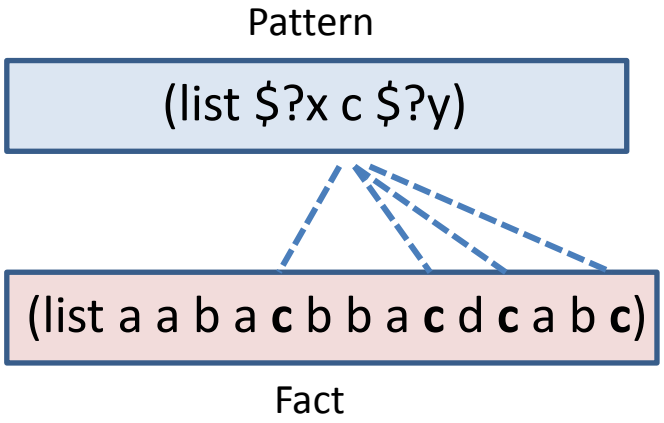




6. Pattern-matching



When two or more multi-valued variables appear in a pattern, the pattern and the fact can match more than once



#Match	$?x	$?y
1	(a a b a c b b a c d c a b)	()
2	(a a b a c b b a c d)	(a b c)
3	(a a b a c b b a)	(d c a b c)
4	(a a b a)	(b b a c d c a b c)

6. Pattern-matching and rule-matching

Choices of matching between a pattern and the facts of the Working Memory

1	pattern	fact	Only one match
2	pattern	fact 1 fact 2	Same pattern matches once with different facts
3	pattern 1 pattern 2	fact	Different patterns match the same fact
4	pattern	fact	Several matches pattern-fact due to the use of multi-valued variables

Rule-matching: the LHS of a rule matches when there is a at least a fact in the Working Memory that matches each pattern in the LHS and the tests are evaluated to TRUE

When a rule matches, it is said the rule is applicable and that ***an instance of a rule*** has been found

There can be more than one match (more than one instance) for the same rule depending on the number of facts that match each pattern

6. Rule-matching

Rule Base

```
(defrule R1
     (number 2)
=>
 ...
(defrule R2
     (number ?num)
=>
 ...
(defrule R3
     (number ?n1)
     (number ?n2)
=>
 ...
(defrule R4
     (list $? ?n1 $? ?n2 $?)
     (test (= (mod ?n1 2) 0))
     (test (<> (mod ?n2 2) 0))
=>
 ...

(defrule R5
    (number ?n1)
    (list $?list)
    (test (member ?n1 $?list))
=>
 ...
```

```
(deffacts data
    (number 2)
    (number 5)
    (number 17)
    (list 26 2 17 18))
```

Working Memory

```
f-1: (number 2)
f-2: (number 5)
f-3: (number 17)
f-4: (list 26 2 17 18)
```

Rule Instances

Rule	# Match	Facts
R1	1	f-1
R2	1	f-1: {?num=2}
	2	f-2: {?num=5}
	3	f-3: {?num=17}
R3	1	f-1, f-1: {?n1=2, ?n2=2}
	2	f-1, f-2: {?n1=2, ?n2=5}
	3	f-1, f-3: {?n1=2, ?n2=17}
	4	f-2, f-1: {?n1=5, ?n2=2}
	5	f-2, f-2: {?n1=5, ?n2=5}
	6	f-2, f-3: {?n1=5, ?n2=17}
	7	f-3, f-1: {?n1=17, ?n2=2}
	8	f-3, f-2: {?n1=17, ?n2=5}
	9	f-3, f-3: {?n1=17, ?n2=17}
R4	1	f-4: {?n1=2, ?n2=17}
	2	f-4: {?n1=26,?n2=17}
R5	1	f-1,f-4: {?n1=2, $?list=(26 2 17 18)}
	2	f-3,f-4: {?n1=17, $?list=(26 2 17 18)}

6. Rule-matching

Rule Base

```
(defrule R1
     (list $? ?x $?)
     (test (evenp ?x))
=>
     (assert (number even ?x))
     (printout t ?x " is an even number of the list " crlf))


(defrule R2
     (list $? ?x $?)
     (test (oddp ?x))
=>
     (assert (number odd ?x))
     (printout t ?x " is an odd number of the list " crlf))
```

```
(deffacts data
    (list 26 2 17 18))
```

Working Memory

f-1: (list 26 2 17 18)

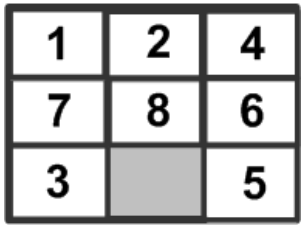
Rule Instances

Rule	# Match	Facts
R1	1 2 3	f-1: {?x=18} f-1: {?x=2} f-1: {?x=26}
R2	1	f-1: {?x=17}

Rule matching

- Four rule instances, three instances of the rule R1 and one of the rule R2
- The inference engine will select one of the rule instances
- Once selected, the rule instance is fired and the RHS of such an instance is executed

6. Rule matching



WM= { f-1: (puzzle 1 1 1) f-2: (puzzle 1 2 2) f-3: (puzzle 1 3 4) f-4: (puzzle 2 1 7) f-5: (puzzle 2 2 8) f-6: (puzzle 2 3 6) f-7: (puzzle 3 1 3) f-8: (puzzle 3 2 0) f-9: (puzzle 3 3 5)}

```
(defrule move_cell_to_left  ;; blank to right
      ?fa <- (puzzle  ?x0  ?y0  0)
      ?fb <- (puzzle  ?x1  ?y1  ?cell)
      (test (and (<  ?y0  3)(=  ?x0 ?x1)))
      (test (= ?y1 (+ ?y0 1)))
=>
      (retract ?fa ?fb)
      (assert (puzzle ?x0 ?y0 ?cell))
      (asssert (puzzle ?x1 ?y1 0)))
```

Rule instances

Rule	# Match	Facts
move_cell_to_left	1	f-8, f-9: {?x0=3, ?y0=2, ?x1=3, ?y1=3, ?cell=5} ?fa=8, ?fb=9

WM= { f-1: (puzzle 1 1 1) f-2: (puzzle 1 2 2) f-3: (puzzle 1 3 4) f-4: (puzzle 2 1 7) f-5: (puzzle 2 2 8) f-6: (puzzle 2 3 6) f-7: (puzzle 3 1 3) f-10: (puzzle 3 2 5) f-11: (puzzle 3 3 0)}

7. Exercises

Exercise 1

Let a RBS be composed of WMinitial={(lista 7 3 2 5)} and whose RB is made up of the following rule:

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

Say the set of rule instances of R1 that results from the application of the pattern-matching process.

Exercise 2

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))
  =>
  ...
```

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

Say the set of rule instances of R1 and R2 that results from the application of the pattern-matching process.

7. Exercises

Exercise 3

Let WMinitial={(lista 1 2 3 4)} be the initial WM of RBS whose RB contains the rules R1 and R2 :

```
(defrule R1
    ?f <- (lista ?x $?z)
    =>
    …
```

```
(defrule R2
    ?f <-  (elemento ?x)
           (elemento ?y)
           (test (< ?x ?y))
    =>
    ….
```

Say the set of rule instances of R1 and R2 that results from the application of the pattern-matching process.

Exercise 4

Let WMinitial={(lista a a b a) (par a 1) (par b 2)} be the initial WM of RBS whose RB contains the rule R1:

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

Say the set of rule instances of R1 that results from the application of the pattern-matching process.