DSIIC
DEPARTAMENTO DE SISTEMAS
INFORMÁTICOS Y COMPUTACIÓN

UNIVERSITAT
POLITÈCNICA
DE VALÈNCIA

Escuela Técnica Superior de Ingeniería **Informática** etsinf

Dpto. Sistemas Informáticos y Computación
Escuela Técnica Superior de Ingeniería Informática
UNIVERSITAT POLITÈCNICA DE VALÈNCIA

INTELLIGENT SYSTEMS

CLIPS seminar (V6.3)

(C Language Integrated Production System)

CLIPS V6.3
(C Language Integrated Production System)

1. Introduction

CLIPS is a tool for Expert Systems originally developed by Software Technology Branch of the NASA/Lyndon B. Johnson Space Center. CLIPS is designed for the construction of Rule-Based Systems (RBS) and ease the software development that requires to model expert knowledge in a specific problem.

There are three ways to represent knowledge in CLIPS:

- **rules**: specially targeted to represent heuristic knowledge based on the experience
- **functions**: to represent procedural knowledge
- **object-oriented programming (OOP)**: mainly to represent procedural knowledge. CLIPS supports the 6 features generally accepted by OOP: classes, message passing, abstraction, encapsulation, heritage and polimorfism.

CLIPS supports 3 programming paradigms:

- **rule-based programming**: rules+ facts
- **procedural programming:** functions
- **object-oriented programming :** objetcs+message passing

Out of these three paradigms, we will only used rule-based programming and, in very specific cases, procedural programming.

Features of CLIPS:

- CLIPS is a tool written in C
- Full integration with other llanguages such as C and ADA
- A CLIPS process can be called from a procedural language. CLIPS carries out its action and then, returns the control to the calling program.
- Procedural code can be incorporated as external functions in CLIPS.
- Rules can "pattern-match" on objects and facts given place to an integrated system.
- CLIPS follows a syntax style similar to LISP that uses parenthesis as delimiters

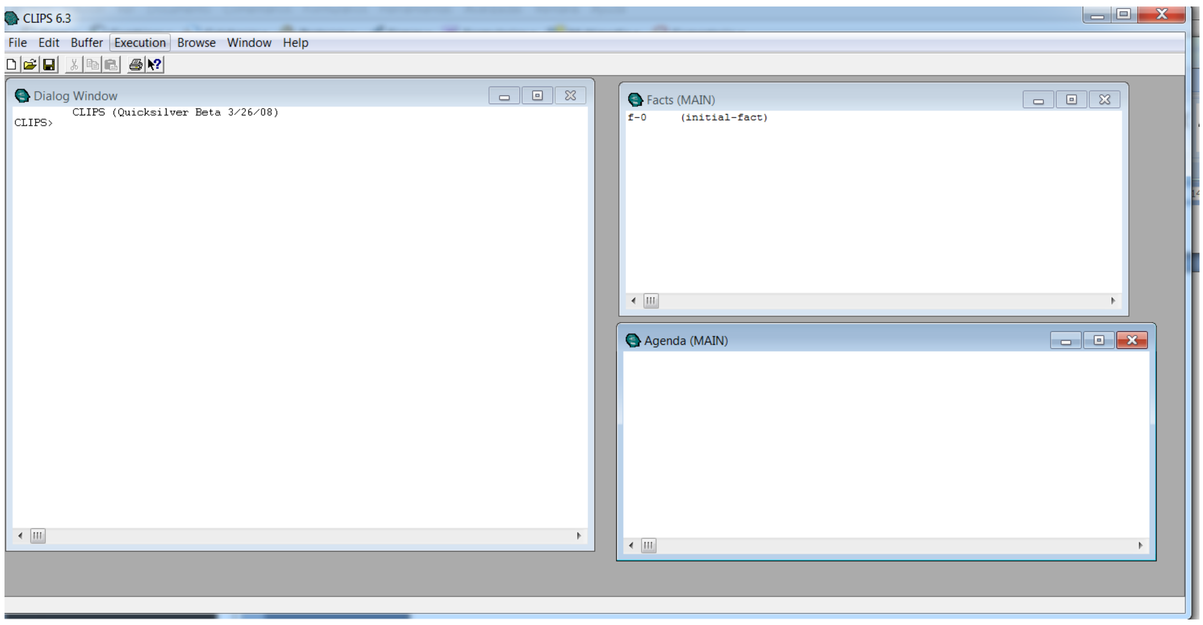
CLIPS is a Production System that is componsed of the following modules:

- **Working memory** (facts + instances of objects)
- **Agendas** (rules)
- **Inference engine** (control)

2. CLIPS interface

The CLIPS interface is composed of three main windows:

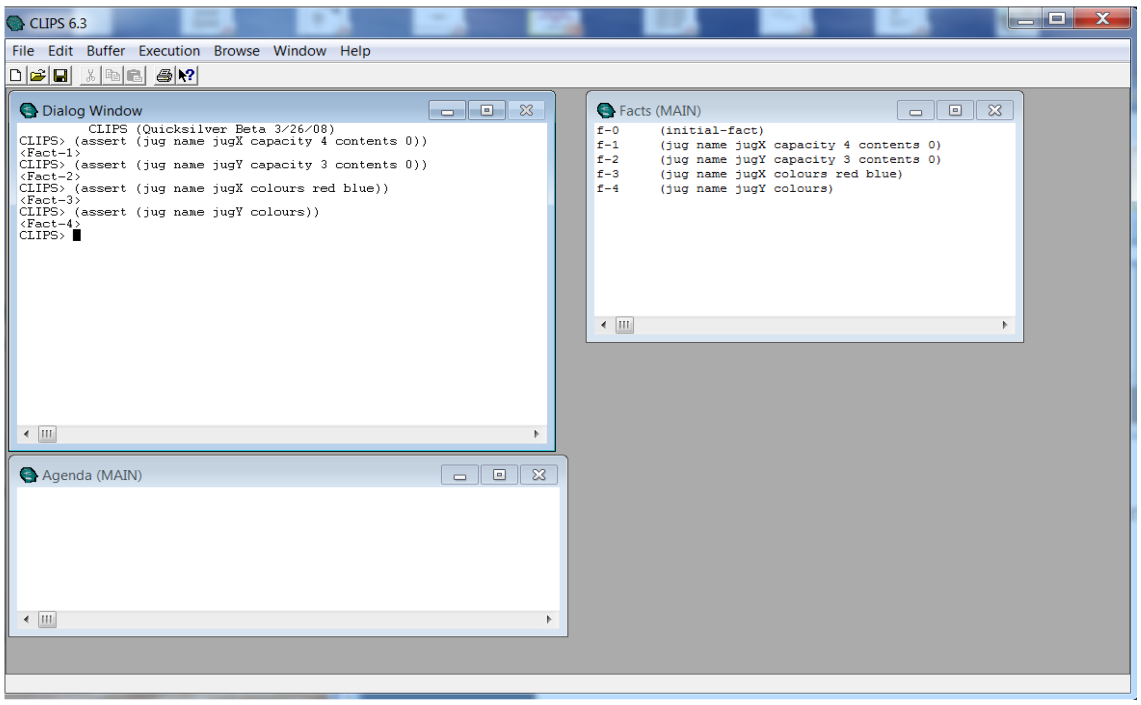
- **Dialog Window**. CLIPS command shell where commands can be typed in to execute commands that CLIPS evaluates.
- **Facts**. It contains the facts of the problem.
- **Agenda.** Agenda or conflict set where rule instances or activations are stored.



```
         CLIPS (Quicksilver Beta 3/26/08)
CLIPS>
```

```
f-0     (initial-fact)
```

Examples of how the command `assert` is used.



```
         CLIPS (Quicksilver Beta 3/26/08)
CLIPS> (assert (jug name jugX capacity 4 contents 0))
<Fact-1>
CLIPS> (assert (jug name jugY capacity 3 contents 0))
<Fact-2>
CLIPS> (assert (jug name jugX colours red blue))
<Fact-3>
CLIPS> (assert (jug name jugY colours))
<Fact-4>
CLIPS> █
```

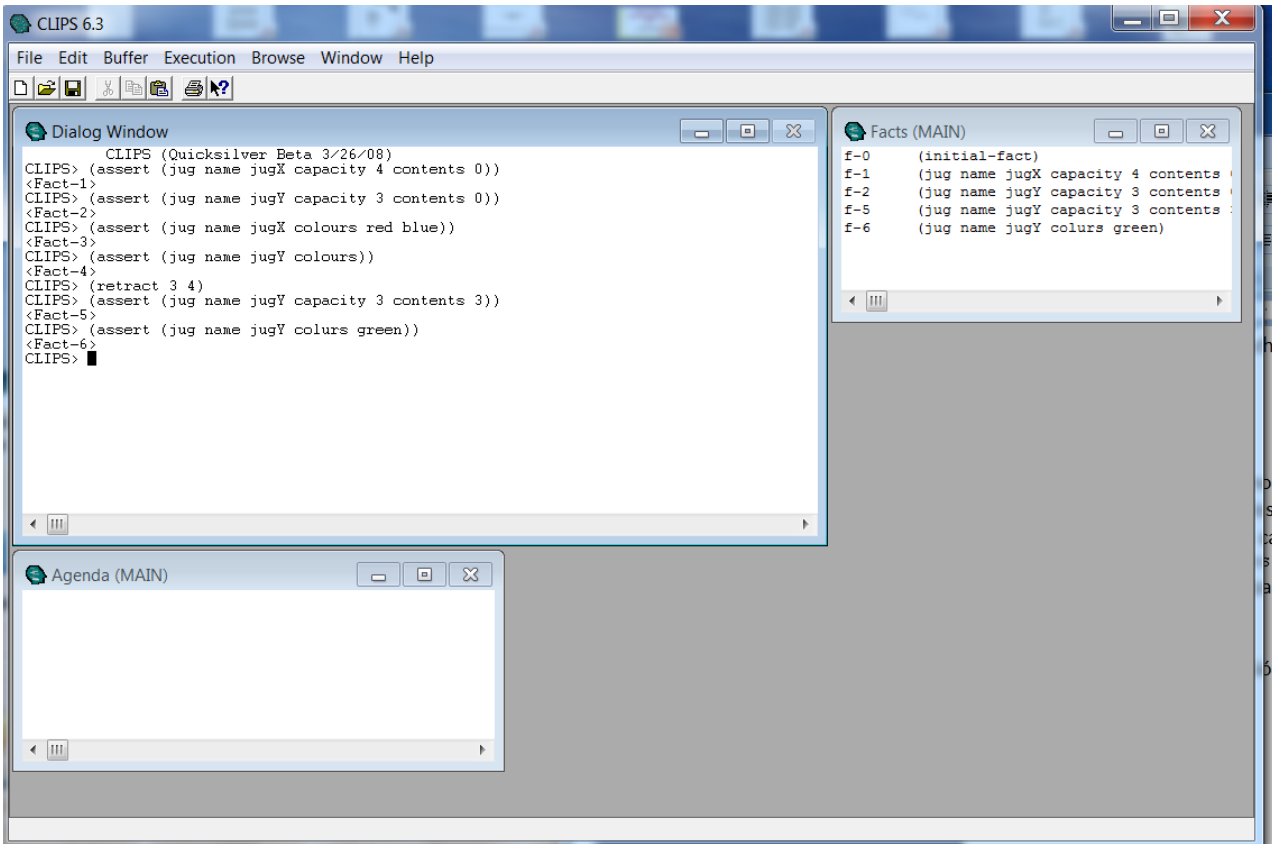
```
f-0     (initial-fact)
f-1     (jug name jugX capacity 4 contents 0)
f-2     (jug name jugY capacity 3 contents 0)
f-3     (jug name jugX colours red blue)
f-4     (jug name jugY colours)
```

- The command **`assert`** is used to insert a fact into the Working Memory (WM). The syntax is `(assert <fact>+)`.
- Facts represent a set of variables, properties and values.
- A fact is composed of a unlimited number of fields separated by blanks and enclosed by parenthesis. Fields may have or not a name assigned
- All facts have an identifier or index of the form **`f-n`** (see window **Facts**) where '**n**' is the index of the fact (`fact-index`).
- CLIPS defines by default an initial fact `(initial-fact)` when the program is loaded.
- All facts are inserted into the list ***Facts Window*** (WM).

Ordered facts:

- Fields does not have a name assigned
- The order of the fields is important
- Types of fields: `float, integer, symbol, string, external-address, fact-address, instance-name, instance-address`
- The first field of a fact usually describes the relation among the fields.

Examples of how the command `retract` is used:



```
          CLIPS (Quicksilver Beta 3/26/08)
CLIPS> (assert (jug name jugX capacity 4 contents 0))
<Fact-1>
CLIPS> (assert (jug name jugY capacity 3 contents 0))
<Fact-2>
CLIPS> (assert (jug name jugX colours red blue))
<Fact-3>
CLIPS> (assert (jug name jugY colours))
<Fact-4>
CLIPS> (retract 3 4)
CLIPS> (assert (jug name jugY capacity 3 contents 3))
<Fact-5>
CLIPS> (assert (jug name jugY colurs green))
<Fact-6>
CLIPS>
```

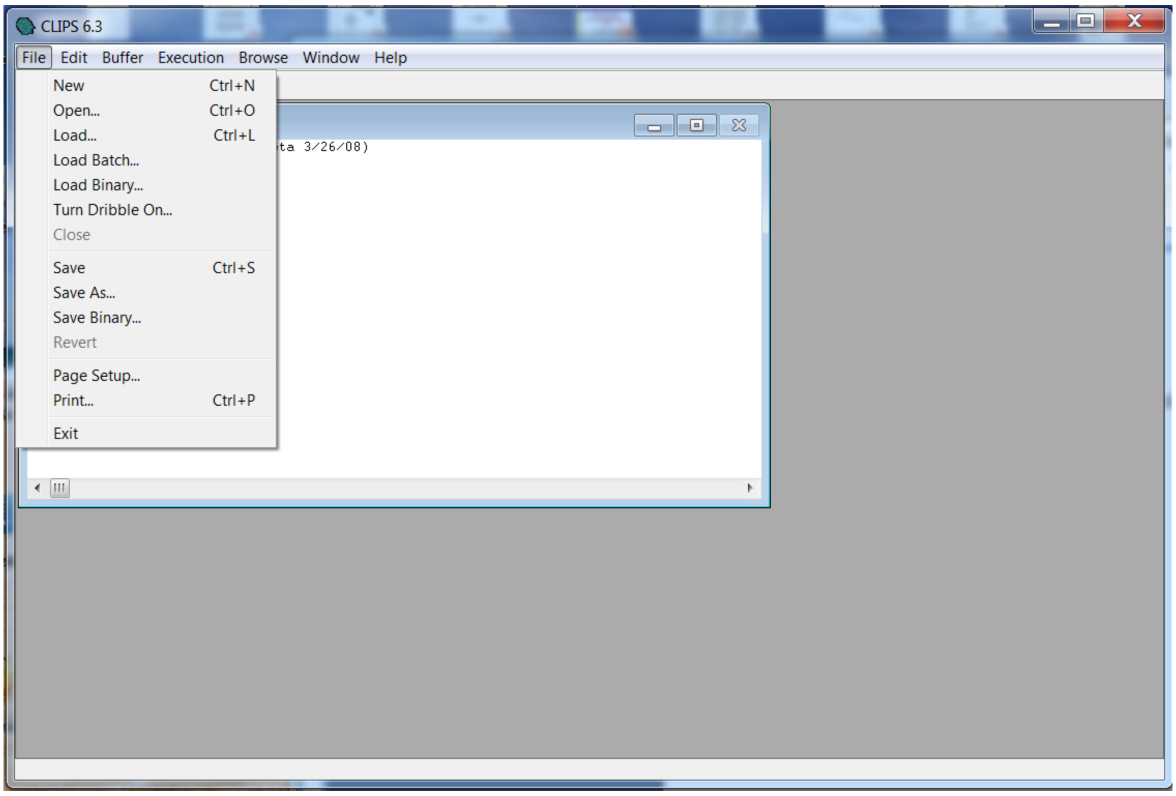
```
f-0      (initial-fact)
f-1      (jug name jugX capacity 4 contents
f-2      (jug name jugY capacity 3 contents
f-5      (jug name jugY capacity 3 contents
f-6      (jug name jugY colurs green)
```

- The command `retract` is used to delete a fact from the WM. The syntax of this command is `(retract <fact-index>+)`.

- When a fact is deleted, the rest of facts preserve their index or identifier.
- When a new fact is inserted in the WM, it is assigned the following index to the last inserted fact.

2.1 CLIPS working environment

Write your RBS in a text file. You can use any text editor or the CLIPS editor to do so (option **New** from the menu **File**).

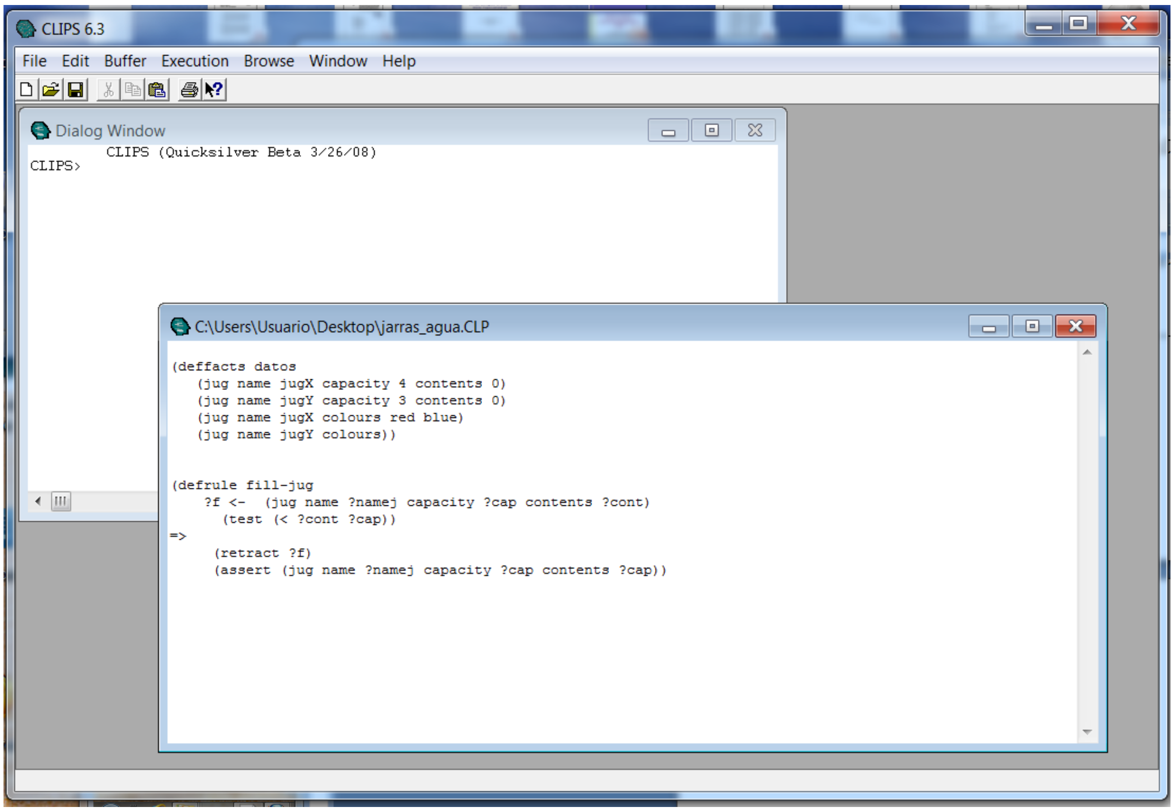


```
ta 3/26/08)
```

Next, an example of a RBS only containing a set of initial facts is shown. These initial facts are defined by the constructor `deffacts`. In addition, rules can be defined by the constructor `defrule`.

The constructor `deffacts` defines all initial facts of the problem. The constructor `deffacts` simply put these facts in the WM, which is the structure defined to insert the facts into when the RBS is executed. The syntax is

```
(deffacts <statement-name>
        (<fact-1>)
        (<fact-2>)
          .........   )
```


CLIPS 6.3
File Edit Buffer Execution Browse Window Help
Dialog Window
C:\Users\Usuario\Desktop\jarras_agua.CLP

```
        CLIPS (Quicksilver Beta 3/26/08)
CLIPS>
```

```
(deffacts datos
   (jug name jugX capacity 4 contents 0)
   (jug name jugY capacity 3 contents 0)
   (jug name jugX colours red blue)
   (jug name jugY colours))


(defrule fill-jug
    ?f <-  (jug name ?namej capacity ?cap contents ?cont)
      (test (< ?cont ?cap))
=>
     (retract ?f)
     (assert (jug name ?namej capacity ?cap contents ?cap))
```

The three steps to start a RBS are:

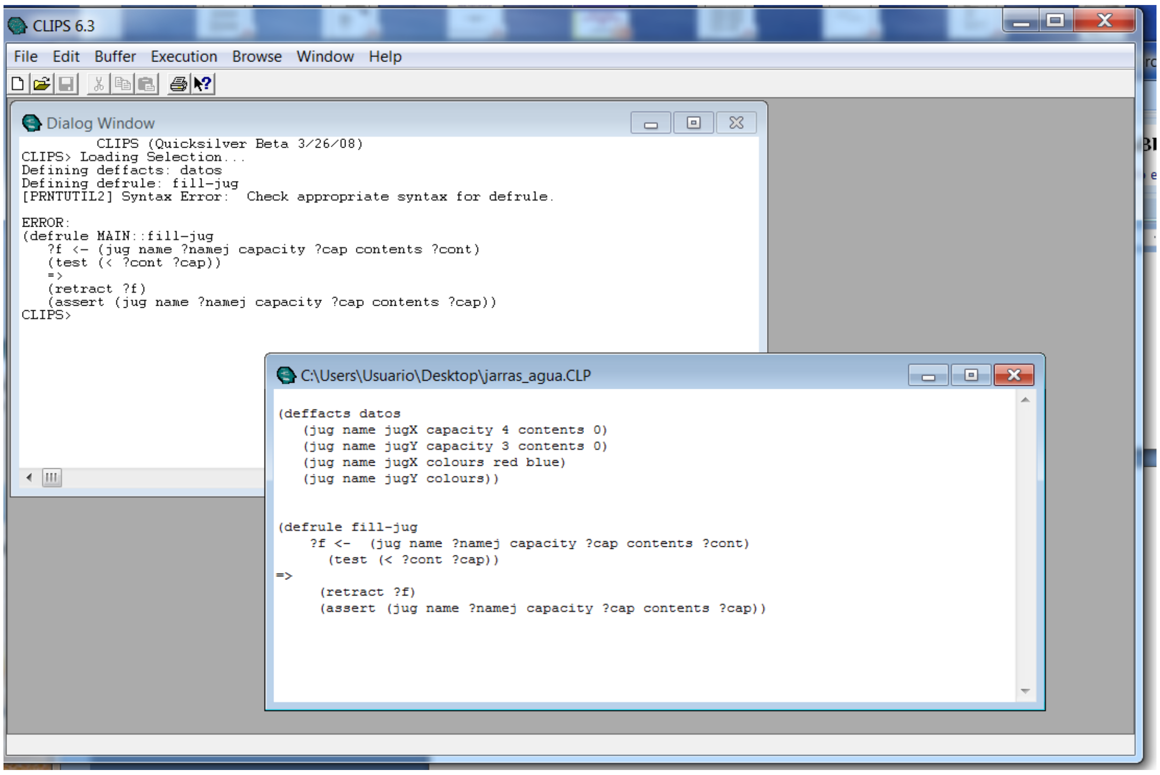
- **Load the file**. This operation is carried parsing the file to check that no syntax errors are found.
- **Reset CLIPS**. Empty the current content of the WM and the Agenda; it loads the defined facts in the structure deffacts, loads the rules defined by defrule statements and performs the first matching phase.
- **RBS execution**. This operation starts the Inference Engine and executes the SBR, applying subsesequent pattern-matching cycles of the RETE algorithm..

Loading a file

Once a file has been saved, the next operation is to load this file. This can be done in two ways:

- Using the option **Load …** from the **File** menu
- Using the option **Load Buffer** from the **Buffer** menu

This operation checks that all CLIPS structures in the file are correctly written. In the next example, it can be observed that there is an error in the definition of the rule; more precisely, the error is found at the end of the rule, where there is a missing closing parenthesis.



CLIPS 6.3

Dialog Window

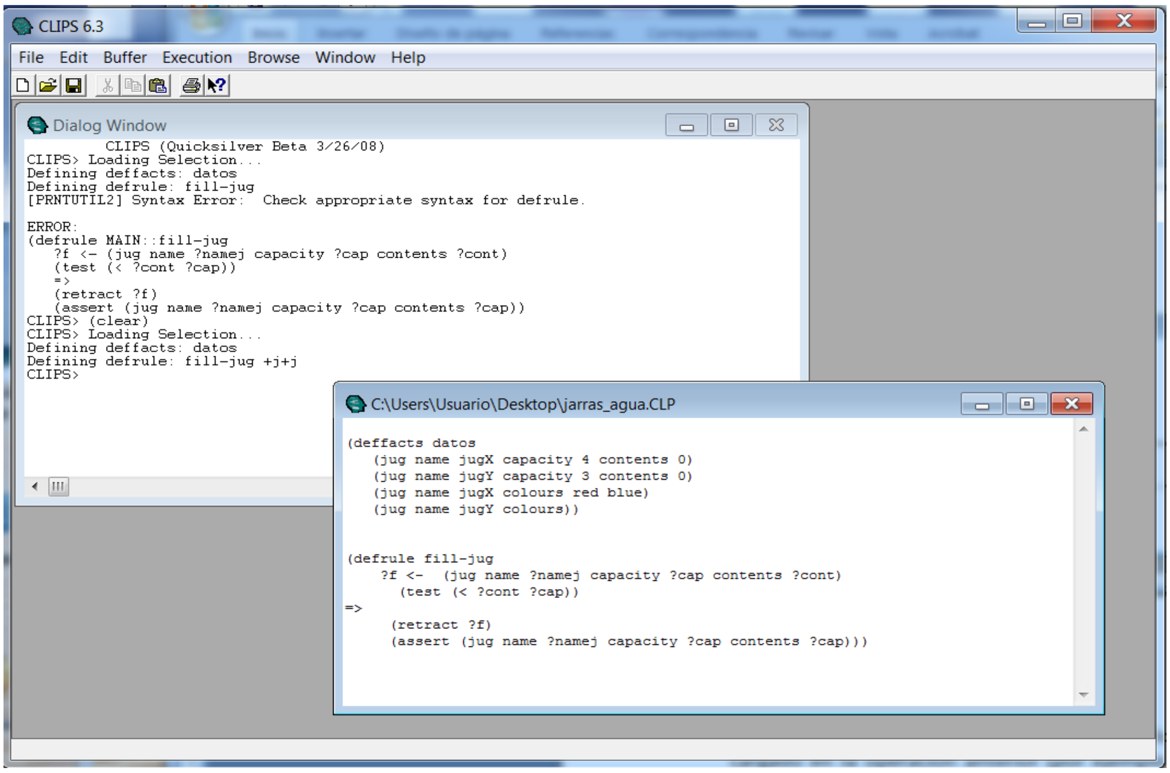
```
         CLIPS (Quicksilver Beta 3/26/08)
CLIPS> Loading Selection...
Defining deffacts: datos
Defining defrule: fill-jug
[PRNTUTIL2] Syntax Error:  Check appropriate syntax for defrule.

ERROR:
(defrule MAIN::fill-jug
   ?f <- (jug name ?namej capacity ?cap contents ?cont)
   (test (< ?cont ?cap))
   =>
   (retract ?f)
   (assert (jug name ?namej capacity ?cap contents ?cap))
CLIPS>
```

C:\Users\Usuario\Desktop\jarras_agua.CLP

```
(deffacts datos
   (jug name jugX capacity 4 contents 0)
   (jug name jugY capacity 3 contents 0)
   (jug name jugX colours red blue)
   (jug name jugY colours))


(defrule fill-jug
    ?f <-  (jug name ?namej capacity ?cap contents ?cont)
      (test (< ?cont ?cap))
=>
     (retract ?f)
     (assert (jug name ?namej capacity ?cap contents ?cap))
```

In this case what should be done is:

a. Correct the error in the file
b. Type the command `(clear)` to delete all stuctures previously loaded (for example, the structure `deffacts`). The command `(clear)` can be also activated using the option **Clear CLIPS** from the **Execution** menu
c. Reload the file



CLIPS 6.3

Dialog Window

```
         CLIPS (Quicksilver Beta 3/26/08)
CLIPS> Loading Selection...
Defining deffacts: datos
Defining defrule: fill-jug
[PRNTUTIL2] Syntax Error:  Check appropriate syntax for defrule.

ERROR:
(defrule MAIN::fill-jug
   ?f <- (jug name ?namej capacity ?cap contents ?cont)
   (test (< ?cont ?cap))
   =>
   (retract ?f)
   (assert (jug name ?namej capacity ?cap contents ?cap))
CLIPS> (clear)
CLIPS> Loading Selection...
Defining deffacts: datos
Defining defrule: fill-jug +j+j
CLIPS>
```

C:\Users\Usuario\Desktop\jarras_agua.CLP

```
(deffacts datos
   (jug name jugX capacity 4 contents 0)
   (jug name jugY capacity 3 contents 0)
   (jug name jugX colours red blue)
   (jug name jugY colours))


(defrule fill-jug
    ?f <-  (jug name ?namej capacity ?cap contents ?cont)
      (test (< ?cont ?cap))
=>
     (retract ?f)
     (assert (jug name ?namej capacity ?cap contents ?cap)))
```

Reset CLIPS

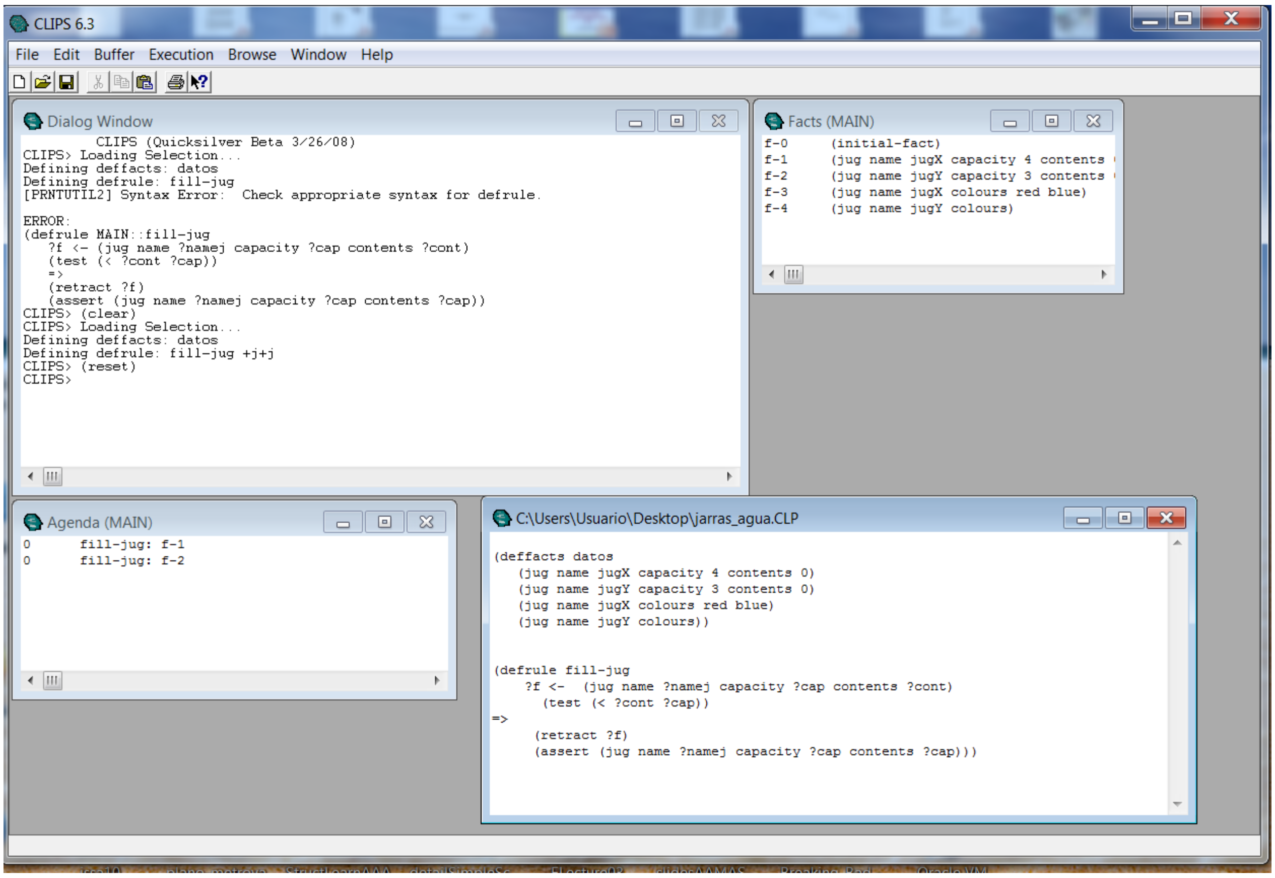
Once a file is error-free, the next operation is to load the data structures in memory. More precisely, the command `(reset)` performs the following operations:

1. Delete existing facts in the WM from previous executions.
2. Insert the initial fact `(initial-fact)`.
3. Insert the facts defined with the constructors `deffacts` in the WM
4. Performs the first phase of the matching operation preparing the possible rule activations that may exist.

The command `(reset)` can be activated:

a. Directly from the command window (Dialog Window)
b. By the **Reset** option from the **Execution** menu

As observed in the next screen, the operation `(reset)` has inserted facts in the WM and two activations in the Agenda (the rule `fill-jug` for the jug referred as to `jugX`, and the same rule `fill-jug` for the jug referred as to `jugY`).


CLIPS 6.3
File Edit Buffer Execution Browse Window Help
Dialog Window
Facts (MAIN)
Agenda (MAIN)
C:\Users\Usuario\Desktop\jarras_agua.CLP

```
         CLIPS (Quicksilver Beta 3/26/08)
CLIPS> Loading Selection...
Defining deffacts: datos
Defining defrule: fill-jug
[PRNTUTIL2] Syntax Error:  Check appropriate syntax for defrule.

ERROR:
(defrule MAIN::fill-jug
   ?f <- (jug name ?namej capacity ?cap contents ?cont)
   (test (< ?cont ?cap))
   =>
   (retract ?f)
   (assert (jug name ?namej capacity ?cap contents ?cap))
CLIPS> (clear)
CLIPS> Loading Selection...
Defining deffacts: datos
Defining defrule: fill-jug +j+j
CLIPS> (reset)
CLIPS>
```

```
f-0     (initial-fact)
f-1     (jug name jugX capacity 4 contents
f-2     (jug name jugY capacity 3 contents
f-3     (jug name jugX colours red blue)
f-4     (jug name jugY colours)
```

```
0      fill-jug: f-1
0      fill-jug: f-2
```

```
(deffacts datos
   (jug name jugX capacity 4 contents 0)
   (jug name jugY capacity 3 contents 0)
   (jug name jugX colours red blue)
   (jug name jugY colours))


(defrule fill-jug
    ?f <-  (jug name ?namej capacity ?cap contents ?cont)
      (test (< ?cont ?cap))
=>
     (retract ?f)
     (assert (jug name ?namej capacity ?cap contents ?cap)))
```

Executing CLIPS

The third and last step is to execute the RBS. This operation can be perfomed in two ways:

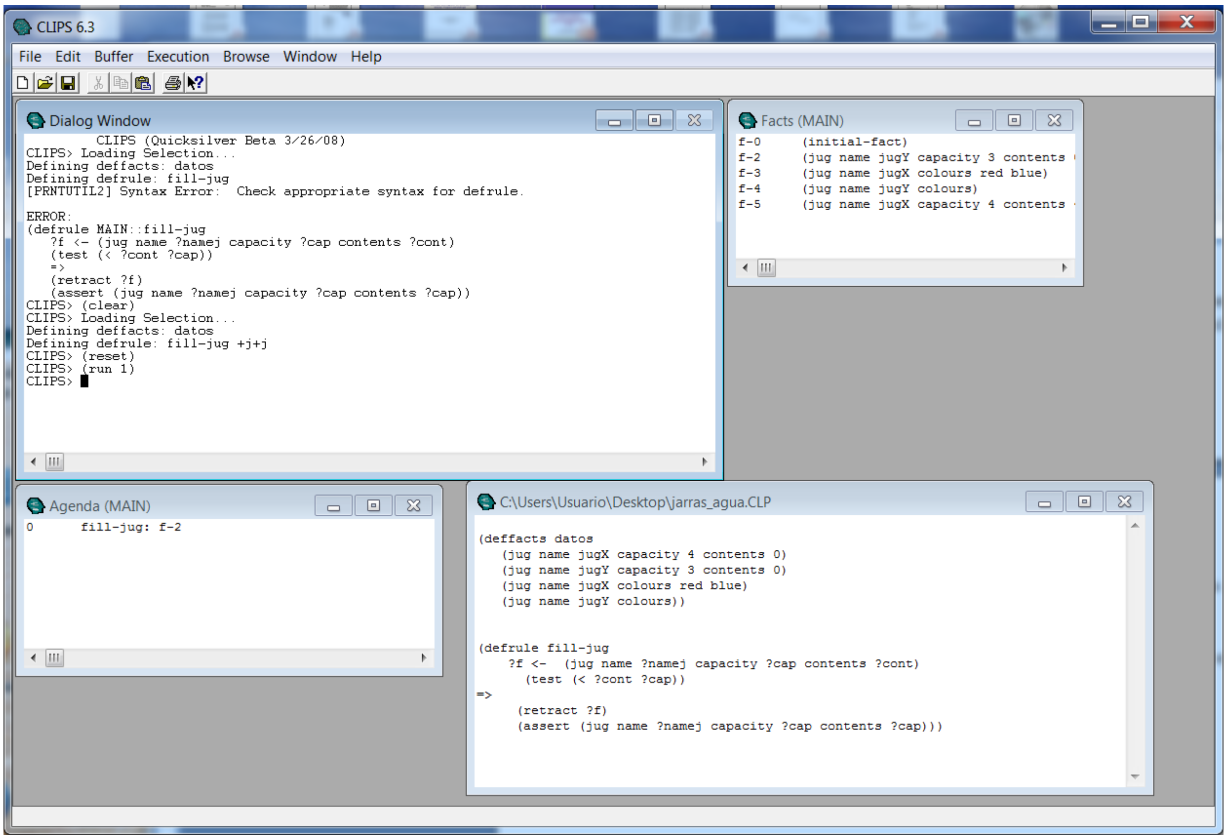
a. Typing the command `(run)` or selecting the **Run** option from the **Execution** menu.The command Run executes subsequent pattern-matching-execution cycles until the system finishes (or a rule halts the inferential process, or the Agenda is empty)

b. By the **Step** option from the **Execution** menu. This option allows to execute the RBS step by step, observing every step of the inferential process.

Select the **Step** option in our problem. Each operation Step performs the following phases:

1. select the first activation from the Agenda
2. execute the right-hand side of this activation or rule instance (RHS of the selected rule)
3. perform a new matching process.

In our small example, the execution of the first instance from the Agenda deletes the fact **`f-1`** and generates the fact **`f-5`** (representing that the jug `jugX` is now filled) but the fact **f-5** does not lead to a new rule matching, since it does not activate the only defined rule in our RBS.


CLIPS 6.3
File Edit Buffer Execution Browse Window Help
Dialog Window
Facts (MAIN)
Agenda (MAIN)
C:\Users\Usuario\Desktop\jarras_agua.CLP

```
         CLIPS (Quicksilver Beta 3/26/08)
CLIPS> Loading Selection...
Defining deffacts: datos
Defining defrule: fill-jug
[PRNTUTIL2] Syntax Error:  Check appropriate syntax for defrule.

ERROR:
(defrule MAIN::fill-jug
   ?f <- (jug name ?namej capacity ?cap contents ?cont)
   (test (< ?cont ?cap))
   =>
   (retract ?f)
   (assert (jug name ?namej capacity ?cap contents ?cap))
CLIPS> (clear)
CLIPS> Loading Selection...
Defining deffacts: datos
Defining defrule: fill-jug +j+j
CLIPS> (reset)
CLIPS> (run 1)
CLIPS> 
```

```
f-0      (initial-fact)
f-2      (jug name jugY capacity 3 contents
f-3      (jug name jugX colours red blue)
f-4      (jug name jugY colours)
f-5      (jug name jugX capacity 4 contents
```

```
0       fill-jug: f-2
```

```
(deffacts datos
   (jug name jugX capacity 4 contents 0)
   (jug name jugY capacity 3 contents 0)
   (jug name jugX colours red blue)
   (jug name jugY colours))


(defrule fill-jug
    ?f <-  (jug name ?namej capacity ?cap contents ?cont)
      (test (< ?cont ?cap))
=>
     (retract ?f)
     (assert (jug name ?namej capacity ?cap contents ?cap)))
```

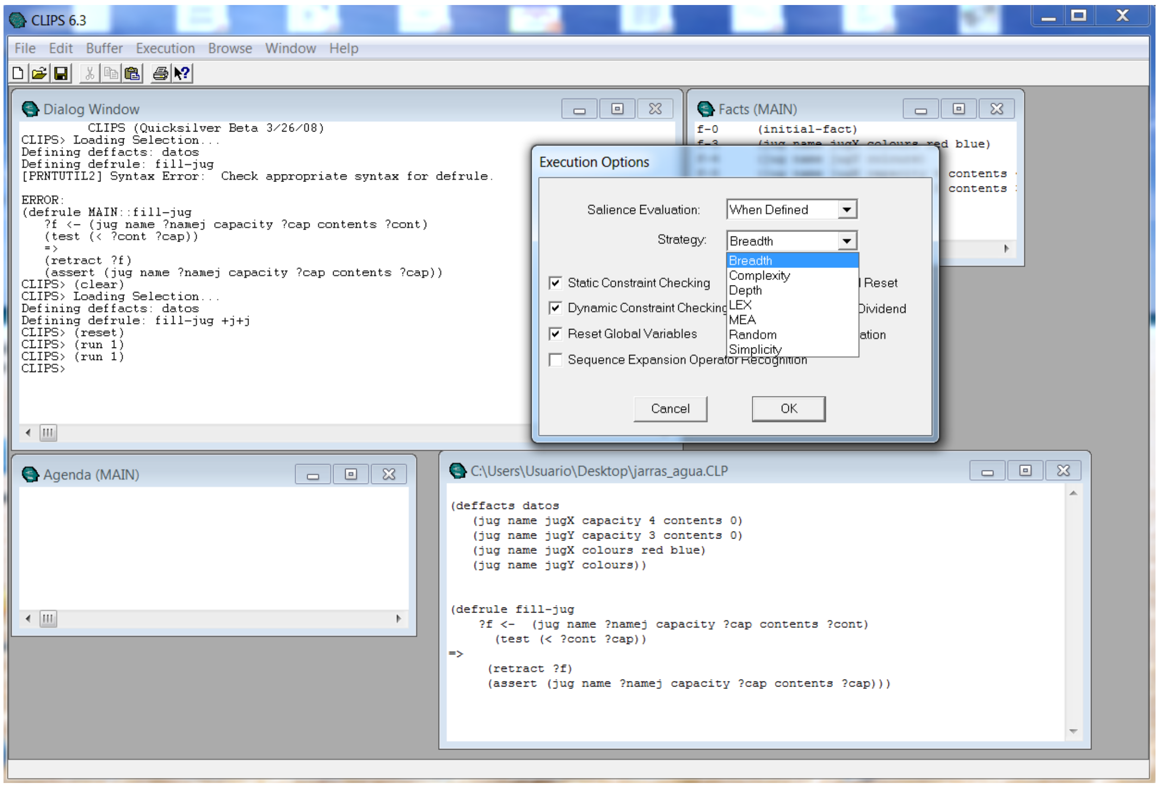
Next, we would execute **Step** from the **Execution** menu, the only instance in the Agenda would be selected, its RHS would be executed and it would generate a new fact **`f-6`** that does not activate an instance of the rule `fill-jug`, thus, the matching phase does not generate new activations. The inferential process halts at this point.

2.2 Control strategies

For a detailed definition of the ***conflict resolution strategies*** in CLIPS (refraction, no fact duplicity, etc.), see slides of Block 1, Unit 2.

Basically, in our examples, we would work with the following control (search) strategies: **Breadth**, **Depth** and eventually **Random**.

To select a control strategy in CLIPS, go to **Options** in the **Execution** menu. In this new window, click on the drop-down menu **Strategy** and there we will find the control strategy options available in CLIPS.


CLIPS 6.3
File Edit Buffer Execution Browse Window Help
Dialog Window
Facts (MAIN)
Agenda (MAIN)
C:\Users\Usuario\Desktop\jarras_agua.CLP
Execution Options
Cancel
OK

```
         CLIPS (Quicksilver Beta 3/26/08)
CLIPS> Loading Selection...
Defining deffacts: datos
Defining defrule: fill-jug
[PRNTUTIL2] Syntax Error:  Check appropriate syntax for defrule.

ERROR:
(defrule MAIN::fill-jug
   ?f <- (jug name ?namej capacity ?cap contents ?cont)
   (test (< ?cont ?cap))
   =>
   (retract ?f)
   (assert (jug name ?namej capacity ?cap contents ?cap))
CLIPS> (clear)
CLIPS> Loading Selection...
Defining deffacts: datos
Defining defrule: fill-jug +j+j
CLIPS> (reset)
CLIPS> (run 1)
CLIPS> (run 1)
CLIPS>
```

```
f-0      (initial-fact)
f-3      (jug name jugX colours red blue)
```

Salience Evaluation: When Defined

Strategy: Breadth

Breadth
Complexity
Depth
LEX
MEA
Random
Simplicity

[✓] Static Constraint Checking
[✓] Dynamic Constraint Checking
[✓] Reset Global Variables
[ ] Sequence Expansion Operator Recognition

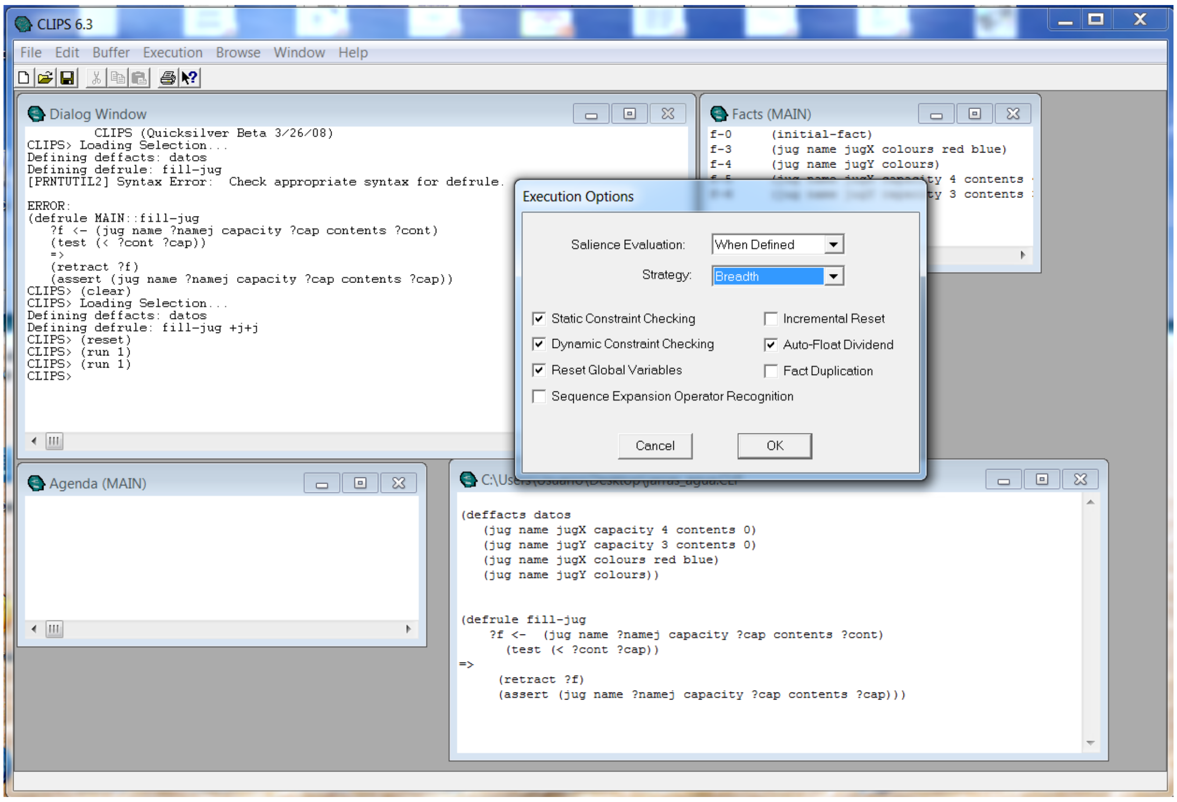
```
(deffacts datos
   (jug name jugX capacity 4 contents 0)
   (jug name jugY capacity 3 contents 0)
   (jug name jugX colours red blue)
   (jug name jugY colours))


(defrule fill-jug
    ?f <-  (jug name ?namej capacity ?cap contents ?cont)
      (test (< ?cont ?cap))
=>
     (retract ?f)
     (assert (jug name ?namej capacity ?cap contents ?cap)))
```

Moreover, it can be observed that the window **Execution Options** shows the unmarked label **Fact Duplication** (default option in CLIPS).

Reminders:

1. CLIPS uses refraction: it does not allow to active a rule more than once with the same facts (facts with the same index numbers).
2. CLIPS by default does not allow Fact Duplication

For further details, see slides of Block 1, Unit 2.



```
         CLIPS (Quicksilver Beta 3/26/08)
CLIPS> Loading Selection...
Defining deffacts: datos
Defining defrule: fill-jug
[PRNTUTIL2] Syntax Error:  Check appropriate syntax for defrule.

ERROR:
(defrule MAIN::fill-jug
   ?f <- (jug name ?namej capacity ?cap contents ?cont)
   (test (< ?cont ?cap))
   =>
   (retract ?f)
   (assert (jug name ?namej capacity ?cap contents ?cap))
CLIPS> (clear)
CLIPS> Loading Selection...
Defining deffacts: datos
Defining defrule: fill-jug +j+j
CLIPS> (reset)
CLIPS> (run 1)
CLIPS> (run 1)
CLIPS>
```

```
f-0     (initial-fact)
f-3     (jug name jugX colours red blue)
f-4     (jug name jugY colours)
                          ty 4 contents
                          ty 3 contents
```

Salience Evaluation: When Defined

Strategy: Breadth

[✓] Static Constraint Checking
[✓] Dynamic Constraint Checking
[✓] Reset Global Variables
[ ] Sequence Expansion Operator Recognition
[ ] Incremental Reset
[✓] Auto-Float Dividend
[ ] Fact Duplication

C:\Us

```
(deffacts datos
   (jug name jugX capacity 4 contents 0)
   (jug name jugY capacity 3 contents 0)
   (jug name jugX colours red blue)
   (jug name jugY colours))


(defrule fill-jug
    ?f <-  (jug name ?namej capacity ?cap contents ?cont)
      (test (< ?cont ?cap))
=>
     (retract ?f)
     (assert (jug name ?namej capacity ?cap contents ?cap)))
```

3. CLIPS inference engine

In this section, two examples or RBS are presented to analyse the behaviour of the pattern matching cycle (or pattern-matching-execution cycle) based on the RETE matching algorithm.

3.1 Sorting a list of numbers

Let be a WM containing a fact representing a list of natural numbers that is not sorted. Write a single (if possible) rule that returns as a final WM, the initial list sorted from lowest to highest:

Example: (list 4 5 3 46 12 10) return: (list 3 4 5 10 12 46)

The CLIPS file would be the following (it can be downloaded from poliformaT `SortNumbers.clp`).

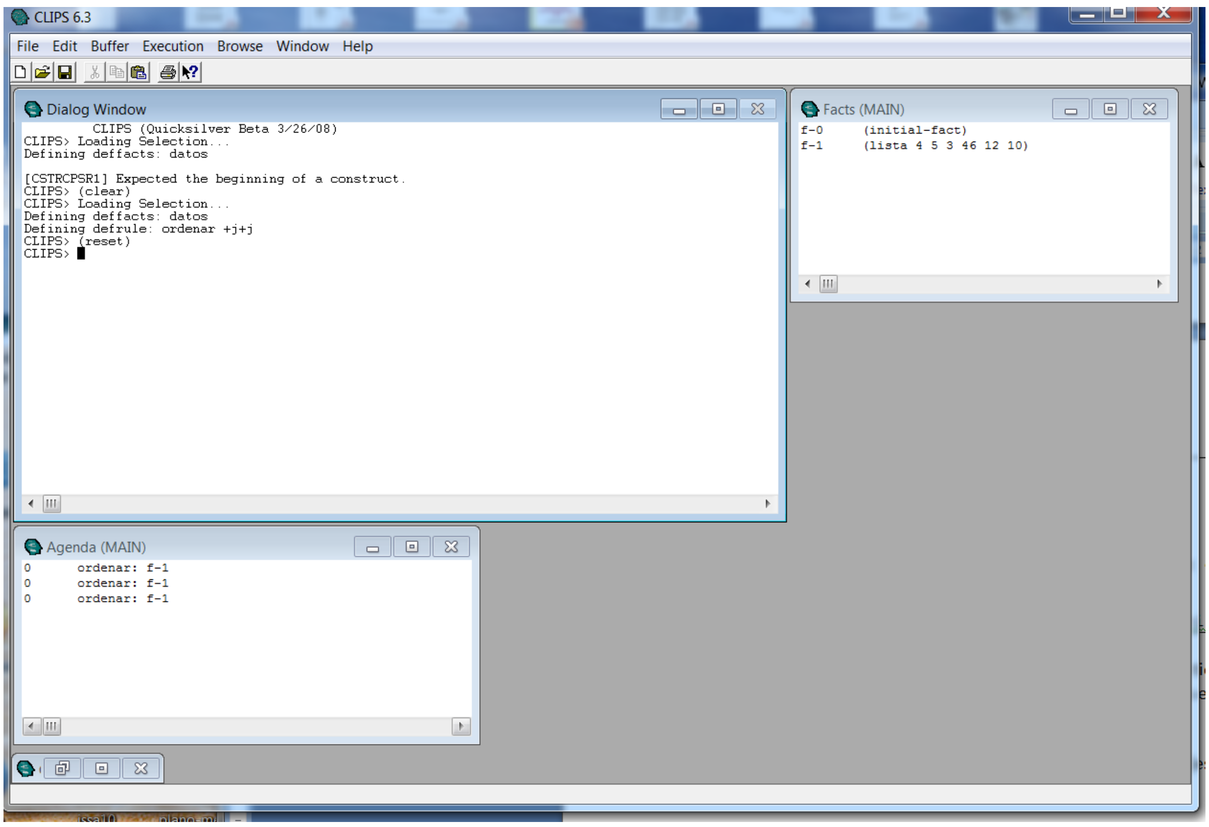
```
 (deffacts data
      (list 4 5 3 46 12 10))
```

```
(defrule sort
   ?f1 <- (list $?x ?y ?z $?w)
          (test (< ?z ?y))    ;; check if ?z is less than ?y
  =>
      (retract ?f1)
      (assert (list $?x ?z ?y $?w))  ;; exchange elements
```

In this example, when the inference engine halts is because there are no more instances of rules in the agenda to be triggered. This means that the list has been sorted.

Select **Breadth** as conflict resolution strategy.

After loading the file, and typing `(reset)`, we will obtain the following output:



```
         CLIPS (Quicksilver Beta 3/26/08)
CLIPS> Loading Selection...
Defining deffacts: datos

[CSTRCPSR1] Expected the beginning of a construct.
CLIPS> (clear)
CLIPS> Loading Selection...
Defining deffacts: datos
Defining defrule: ordenar +j+j
CLIPS> (reset)
CLIPS>
```

```
f-0      (initial-fact)
f-1      (lista 4 5 3 46 12 10)
```

```
0       ordenar: f-1
0       ordenar: f-1
0       ordenar: f-1
```

As observed in the Agenda, there are three rule instances `sort`. All of them match the only fact in the WM:

1. The first instance corresponds to the exchange between values 12 and 10.
2. The second instance corresponds to the exchange between values 46 and 12.
3. The third instance corresponds to the exchange between values 5 and 3.

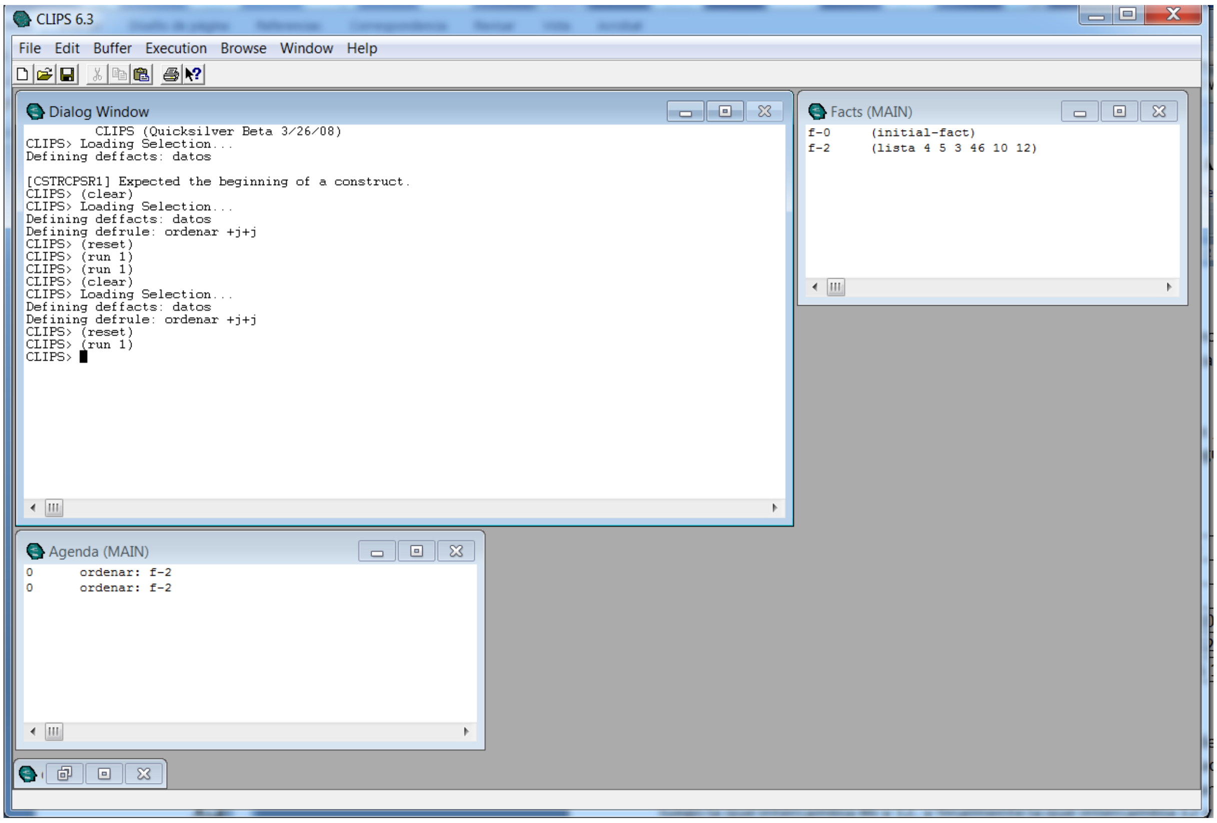
The order in which the Breadth strategy inserts the rules instances found in the matching phase of a cycle has to do with the order in which CLIPS perform the pattern-matching process. CLIPS starts by the last rule in the RBS, the first pattern on the LHS of the rule, or the first multi-value variable in the pattern of a rule.

In this way, CLIPS first tries to match from left to right all possible elements with the multi-value variable `$?x`; next, it matches all previously matched elements but one to the multi-value variable `$?x`, then all but two elements and so on. More precisely, the order in which CLIPS performs the pattern-matching process is:

#Match	**$?x**	**?y**	**?z**	**$?w**	**Test**
1	(4 5 3 46)	12	10	()	**TRUE**
2	(4 5 3)	46	12	(10)	**TRUE**
3	(4 5)	3	46	(12 10)	FALSE
4	(4)	5	3	(46 12 10)	**TRUE**
5	()	4	5	(3 46 12 10)	FALSE

The Breadth strategy inserts the activations in the same order as CLIPS performs the patter-matching process, since it adds each new activation at the end of the Agenda. On the other hand, the Depth strategy inserts activations in the reverse order to which is shown in the table above, since it adds each new activation at the beginning of the Agenda; that is, first, it would insert the rule instance that exchanges 5 and 3, then that that exchanges 46 and 12, and finally that that exchanges 12 and 10.

Following the Breadth strategy, we execute the first cycle `(step)` and the output is:



```
          CLIPS (Quicksilver Beta 3/26/08)
CLIPS> Loading Selection...
Defining deffacts: datos

[CSTRCPSR1] Expected the beginning of a construct.
CLIPS> (clear)
CLIPS> Loading Selection...
Defining deffacts: datos
Defining defrule: ordenar +j+j
CLIPS> (reset)
CLIPS> (run 1)
CLIPS> (run 1)
CLIPS> (clear)
CLIPS> Loading Selection...
Defining deffacts: datos
Defining defrule: ordenar +j+j
CLIPS> (reset)
CLIPS> (run 1)
CLIPS> 
```

```
f-0      (initial-fact)
f-2      (lista 4 5 3 46 10 12)
```

```
0      ordenar: f-2
0      ordenar: f-2
```

- The fact `f-1` is deleted; as a result, the two rule instances that depended on `f-1` are also deleted.
- A new fact `f-2` is generated where the elements 12 and 10 have been exchanged.

- Automatically, the next matching phase is performed and two instance rules appear with the fact `f-2` (the first of them corresponds to elements 46 and 10, and the second one to elements 5 and 3).

Continue the execution of the RBS step by step until the Agenda is empty and check the final state of the WM.

3.2 DNA mutations

Given two DNA sequences, write a RBS (a single rule, if possible) that counts the number of mutations (a global variable can be used to store the number of matching symbols, alternatively a fact can be used to store this value). For example, for the two DNA sequences:

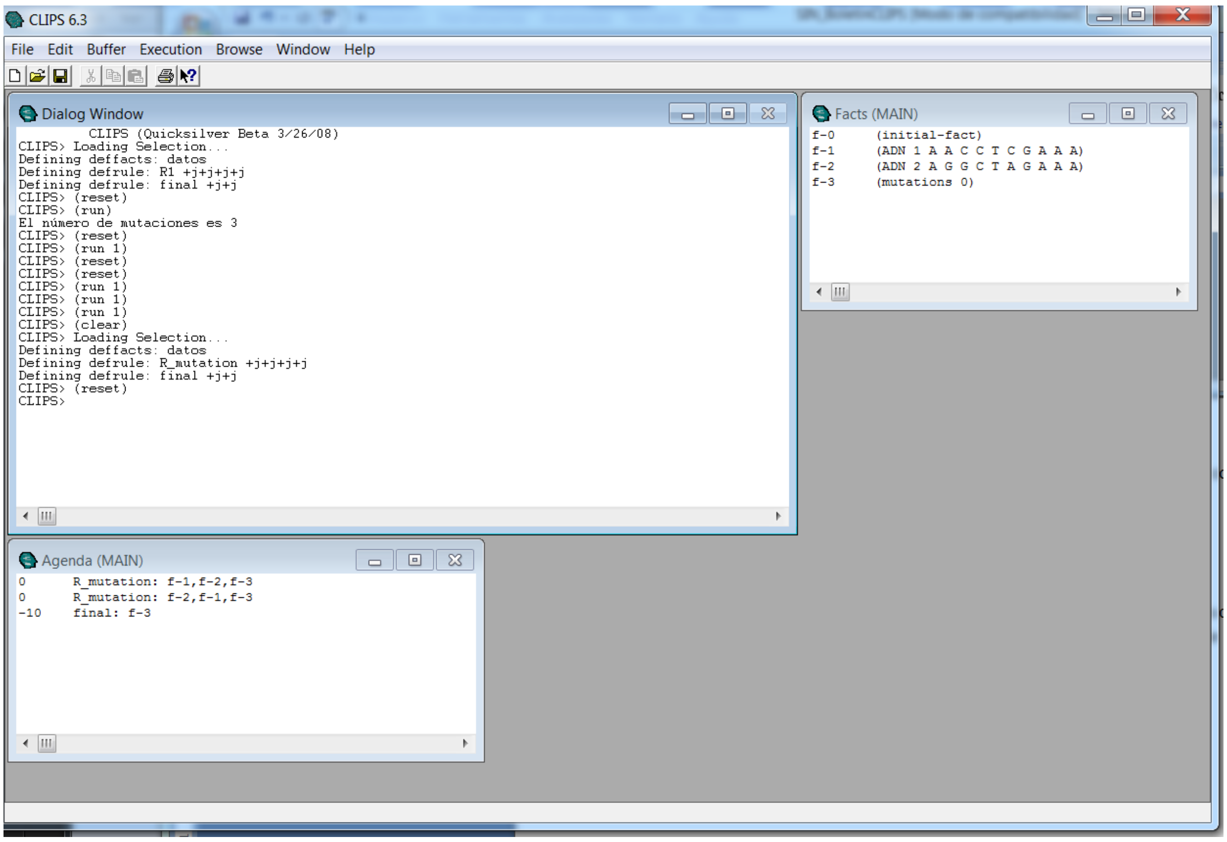
(A A C C T C G A A A) and **(A G G C T A G A A A)** there are three mutations.

The CLIPS file would be the following (it can be downloaded from poliformaT DNAMutations.`clp`).

```
 (deffacts data
       (ADN 1 A A C C T C G A A A)
       (ADN 2 A G G C T A G A A A)
       (mutations 0))

  (defrule R_mutation
    ?f1 <-   (ADN ?n1 $?x ?i1 $?y1)
    ?f2 <-   (ADN ?n2 $?x ?i2 $?y2)
    ?f3 <-   (mutations ?m)
        (test (and (neq ?n1 ?n2)(neq ?i1 ?i2)))
    =>
        (retract ?f1 ?f2 ?f3)
        (assert (ADN ?n1 $?y1))
        (assert (ADN ?n2 $?y2))
        (assert (mutations (+ ?m 1))))

  (defrule final
    (declare (salience -10))
     (mutations ?m)
   =>
     (printout t "The number of mutations is " ?m crlf))
```

In this case, we have defined a `final` rule that shows on the screen the result. We assign to this rule a lower priority (salience) so that it is only triggered when there are no instances of the rule `R_mutation`.

Select **Depth** as conflict resolution strategy. After loading the file, and typing `(reset)`, we obtain the following output:



```
         CLIPS (Quicksilver Beta 3/26/08)
CLIPS> Loading Selection...
Defining deffacts: datos
Defining defrule: R1 +j+j+j+j
Defining defrule: final +j+j
CLIPS> (reset)
CLIPS> (run)
El número de mutaciones es 3
CLIPS> (reset)
CLIPS> (run 1)
CLIPS> (reset)
CLIPS> (reset)
CLIPS> (run 1)
CLIPS> (run 1)
CLIPS> (run 1)
CLIPS> (clear)
CLIPS> Loading Selection...
Defining deffacts: datos
Defining defrule: R_mutation +j+j+j+j
Defining defrule: final +j+j
CLIPS> (reset)
CLIPS>
```

```
f-0      (initial-fact)
f-1      (ADN 1 A A C C T C G A A A)
f-2      (ADN 2 A G G C T A G A A A)
f-3      (mutations 0)
```

```
0      R_mutation: f-1,f-2,f-3
0      R_mutation: f-2,f-1,f-3
-10    final: f-3
```

There are two rule instances appearing in the Agenda corresponding to the same mutation; this is because:

- `f-1,f-2,f-3` match the rule R_mutation, and
- `f-2,f-1,f-3` also matches the rule R_mutation

However, when a rule instance is triggered and its RHS is executed, it will delete all facts that have been instantiated by the patterns on the LHS of the second rule instance, causing that the second rule instance depending on these same facts is automatically deleted.

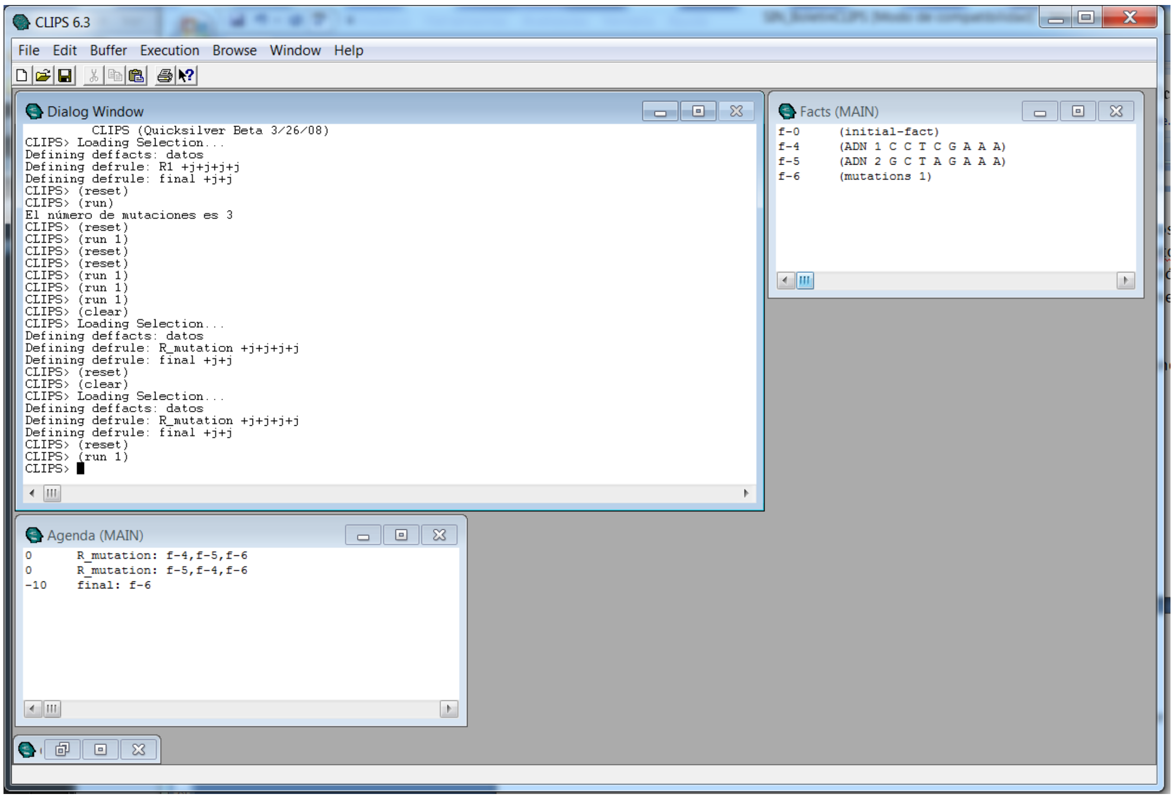
As previously explained, the order in which the two rule instances are inserted in the Agenda depends on the order in which CLIPS performs the pattern-matching process and the conflict resolution strategy selected:

1. CLIPS starts with the third pattern matching with `f-3`; then it matches the second pattern with the first fact that finds `f-1`; consequently, the first pattern matches the fact `f-2` so that the test is evaluated to TRUE.
2. Next it tries to find other combination of facts that matches the LHS of the rule; the third pattern with `f-3`, the second pattern with the next fact that it finds, `f-2`, and, consequently, the first pattern with the fact `f-1` so that the test is evaluated to TRUE.

The two rule instances appearing in the Agenda, and corresponding to the same mutation are:

#Match	**?n1**	**?n2**	**$?x**	**?i1**	**?i2**
1	1	2	(A)	A	G
2	2	1	(A)	G	A

Following the Depth strategy, we execute the first cycle `(step)` and the output is:



```
         CLIPS (Quicksilver Beta 3/26/08)
CLIPS> Loading Selection...
Defining deffacts: datos
Defining defrule: R1 +j+j+j+j
Defining defrule: final +j+j
CLIPS> (reset)
CLIPS> (run)
El número de mutaciones es 3
CLIPS> (reset)
CLIPS> (run 1)
CLIPS> (reset)
CLIPS> (reset)
CLIPS> (run 1)
CLIPS> (run 1)
CLIPS> (run 1)
CLIPS> (clear)
CLIPS> Loading Selection...
Defining deffacts: datos
Defining defrule: R_mutation +j+j+j+j
Defining defrule: final +j+j
CLIPS> (reset)
CLIPS> (clear)
CLIPS> Loading Selection...
Defining deffacts: datos
Defining defrule: R_mutation +j+j+j+j
Defining defrule: final +j+j
CLIPS> (reset)
CLIPS> (run 1)
CLIPS>
```

```
f-0     (initial-fact)
f-4     (ADN 1 C C T C G A A A)
f-5     (ADN 2 G C T A G A A A)
f-6     (mutations 1)
```

```
0       R_mutation: f-4,f-5,f-6
0       R_mutation: f-5,f-4,f-6
-10     final: f-6
```

- Facts `f-1`, `f-2` and `f-3` are removed; and as a consequence, the other rule instance in the Agenda is deleted
- Three new facts `f-4`, `f-5` and `f-6` are generated
- Automatically, the next matching phase is carried out and two rule instances appear corresponding to the mutation of the first elements in each DNA sequence (C in sequence 1 y G in sequence 2).

Continue the execution of the RBS step by step until the final rule is triggered and the total number of mutations are shown on the screen. Try this RBS with two other DNA sequences.

ANNEX: Additional features of the CLIPS language

The main elements of the CLIPS language have already been presented. In this section we present additional features of CLIPS that can be useful in the design and development of a RBS.

Command `printout`

This command allows to output to the device represented by `<logical-name>`.

```
   (printout <logical-name> <expression>*)
```

The output device can be a file or the screen. The symbol **t** is used to identify the standard output. Examples:

```
CLIPS> (printout t "Hello there!" crlf)
Hello There!
CLIPS> (open "data.txt" mydata "w")
TRUE
CLIPS> (printout mydata "red green")
CLIPS> (close)
TRUE
CLIPS>
```

`crlf` means carriage return line feed

Another example of how the command `printout` is used on the RHS of a rule to show outputs on the screen.

```
(defrule find-data-1
      (data ?x $?y ?z)
=>
      (printout t "?x = " ?x crlf
                  "?y = " ?y crlf
                  "?z = " ?z crlf
                  "------" crlf))
```

Command `bind`

This command is used in CLIPS to assign a value to a variable, single-value or multi-value: `(bind <variable> <expression>*)`

Examples:

```
(defrule roll-the-dice
      (roll-the-dice)
=>
      (bind ?roll1 (random 1 6))
      (bind ?roll2 (random 1 6))
      (printout t "Your roll is: " ?roll1 " " ?roll2 crlf))
```

```
(defrule drop-all-water-from-Y-to-X
    ?f1 <-  (jug name ?namex capacity ?capx contents ?contx)
    ?f2 <-  (jug name ?namey capacity ?capy contents ?conty)
      (test  (and (> ?conty 0) (< ?contx ?capx) (neq ?namex ?namey)))
         (test (< (+ ?contx ?conty) ?capx))
=>
      (retract ?f1 ?f2)
      (bind ?newcont (+ ?contx ?conty))
      (assert (jug name ?namex capacity ?capx contents ?newcont))
      (assert (jug name ?namey capacity ?capy contents 0)))
```

Global variables

Global variables must be defined at the beginning of the program using the command **`defglobal`**. The name of a global variables goes between asterisks (*).

Example of the declaration of global variable:

```
      (defglobal ?*lista_a* = (create$))
      (defglobal ?*valor* = 0)
```

As observed. it is possible to assign a value to a global variable when it is declared with the command defglobal.

Another example:

```
CLIPS> (defglobal ?*x* = 3)
CLIPS> ?*x*
3
CLIPS> red
red
CLIPS>(bind ?a 5)
5
CLIPS> (+ ?a 3)
8
CLIPS> (reset)
CLIPS> ?a
[EVALUATN1] Variable a is unbound
FALSE
CLIPS>
```

Command `deffunction`

As in other programming languages, CLIPS allows the definition of functions by using the comand **`deffunction`**. Functions defined with `deffunction` have a global scope and are invoked as any other function. In addition, these functions can be used as arguments of other functions.

```
(deffunction <function-name> [optional-comment]
     (?arg1 ?arg2 … ?argM [$?argN])
     (<action1>
     <action2>
     ……
     <action (k-1)>
     <action k>)
```

1) ?arg$_i$: parameters
2) The function only returns the value of the last action <action k>. This action can be a function, a variable or a constant.

```
(deffunction hypotenuse
      (?a ?b)
      (sqrt (+ (* ?a ?a)(* ?b ?b))))


(defrule compute-hypotenuse
   (dimensions ?base ?height)
 =>
   (printout t "Hypotenuse =" (hypotenuse ?base ?height) crlf))
```

Procedural programming

It is possible to insert procedural code on the RHS. These structures are:

```
- while
- if then else
- break
- return
```

Example:

```
(deffunction start ()
   (reset)
   (printout t "Maximum depth:= " )
   (bind ?*depth* (read))
   (printout t "Search strategy " crlf "
              1.- Breadth" crlf "    2.- Depth" crlf )
   (bind ?a (read))
   (if (= ?a 1)
       then    (set-strategy breadth)
       else   (set-strategy depth))
  (printout t " Execute run to start the program " crlf))
```

Predefined functions

Predefined functions that can be used in tests on the LHS of rules.

(**evenp** <arg>): TRUE if the number is even

(**floatp** <arg>): TRUE if the number is float
(**integerp** <arg>): TRUE if the number is interger
(**lexemep** <arg>): TRUE if the argument is a symbol or a string
(**numberp** <arg>): TRUE if the argument is a number
(**oddp** <arg>): TRUE if the number is odd
(**pointerp** <arg>): TRUE if the argument is an external address
(**sequencep** <arg>): TRUE if it is a mult-field value
(**stringp** <arg>): TRUE if the argument is a string
(**symbolp** <arg>): TRUE if the argument is a symbol

Commands to manage multi-value variables

To find an element in a list use the command `member$` :
`(member$ <single-field-expression> <multifield expression>)`

Examples:

```
   CLIPS> (member$ blue (create$ red 3 "text" 8.7 blue))
   5
   CLIPS> (member$ 4 (create$ red 3 "text" 8.7 blue))
   FALSE
   CLIPS>
```

To obtain the length of a list use the command `length$`:
`(length$ <expression>*)`

Example:

```
(defrule paint-jug-in-red
 ?f <-  (jug name ?name $?z colours $?colours)
        (test (not (member$ red $?colours)))   ;; the jug is not painted in red
        (test (< (length$ $?colours) 2))     ;; the jug is painted in
                                            ;; one colour as a maximum
=>
      (retract ?f)
      (assert (jug name ?name $?z colours $?colours red)))
```

To create a multi-value list use the command `create$`:
`(create$ <expression>*)`

Examples:

```
   CLIPS> (create$ hammer drill saw screw pliers wrench)
   (hammer drill saw screw pliers wrench)
   CLIPS> (create$ (+ 3 4) (* 2 3) (/ 8 4))
   (7 6 2)
   CLIPS> (create$)
   ()
   CLIPS> (bind ?lista (create$ 1 2 3 4))
   (1 2 3 4)
```

Another example:

```
(defrule paint-jug-in-red
    ?f <-  (jug name ?name $?z colours $?colours)
      (test (not (member red $?colours)))
=>
     (retract ?f)
     (assert (jug name ?name $?z colours (create$ $?colours red))))
```

To return an element in a list at a given position use the command `nth$`: `(nth$ <integer-expression> <multifield-expression>)`

Examples:

```
   CLIPS> (nth$ 3 (create$ a b c d e f g))
   c
   CLIPS>
```

To delete an element in a list use the the command `delete-member$`:
`(delete-member$ <multifield-expression> <expression>+)`

Examples:

```
   CLIPS> (delete-member$ (create$ 3 6 78 4 5 6 12 32 5 8 11 5 6)
   (create$ 5 6))
   (3 6 78 4 12 32 5 8 11)


   CLIPS> (delete-member$ (create$ 3 6 78 4 5 6 12 32 5 8 11 5 6) 5 6)
   (3 78 4 12 32 8 11)
   CLIPS>
```

To replace an element in a list use the command `replace-member$`

```
(replace-member$  <multifield-expression>
                  <substitute-expression>
                  <search-expression>+)
```

Examples:

```
CLIPS> (replace-member$ (create$ 3 6 78 4 5 6 12 32 5 8 11 5 6)
HELLO (create$ 5 6))
(3 6 78 4 HELLO 12 32 5 8 11 HELLO)
CLIPS> (replace-member$ (create$ 3 6 78 4 5 6 12 32 5 8 11 5 6)
HELLO 5 6)
(3 HELLO 78 4 HELLO HELLO 12 32 HELLO 8 11 HELLO HELLO)
CLIPS>
```

To insert an element in a list use the command `insert$`:

```
    (insert$ <multifield-expression>
             <integer-expression>
             <single-or-multi-field-expression>+)
```

Examples:

```
   CLIPS> (insert$ (create$ a b c d) 1 x)
   (x a b c d)
   CLIPS> (insert$ (create$ a b c d) 4 y z)
   (a b c y z d)
   CLIPS> (insert$ (create$ a b c d) 5 (create$ q r))
   (a b c d q r)
   CLIPS>
```

To find the first element in a list use the command `first$`:

```
(first$ <multifield-expression>)
```

Examples:

```
   CLIPS> (first$ (create$ a b c))
   (a)
   CLIPS> (first$ (create$))
   ()
   CLIPS>
```

To return the rest of elements in a list use the command `rest$`:

```
(rest$ <multifield-expression>)
```

Examples:

```
   CLIPS> (rest$ (create$ a b c))
   (b c)
   CLIPS> (rest$ (create$))
   ()
   CLIPS>
```

Command `read`

To read data from a file: `(read [<logical-name>])`

Examples:

```
CLIPS> (open "data.txt" mydata "w")
TRUE
```

```
CLIPS> (printout mydata "red green")
CLIPS> (close)
TRUE
CLIPS> (open "data.txt" mydata)
TRUE
CLIPS> (read mydata)
red
CLIPS> (read mydata)
green
CLIPS> (read mydata)
EOF
CLIPS> (close)
TRUE
CLIPS>
```

Command `readline`

To read a line from a file: `(readline [<logical-name>])`

Examples:

```
CLIPS> (open "data.txt" mydata "w")
TRUE
CLIPS> (printout mydata "red green")
CLIPS> (close)
TRUE
CLIPS> (open "data.txt" mydata)
TRUE
CLIPS> (readline mydata)
"red green"
CLIPS> (readline mydata)
EOF
CLIPS> (close)
TRUE
CLIPS>
```