Actividad nº7. Ejercicios de CLIPS. Proponer y resolver dos ejercicios distintos de los vistos en clase. El primero de ellos tratará de resolver cualquier operación sobre conjunto/s: complemento, diferencia,… El segundo cualquier operación sobre una lista/s de cualquier tipo (enteros, letras,): nº de elementos, nº primos…

Ejercicios de recapitulación

Inteligencia Artificial - Grado en Ingeniería Informática – 2019/2020

Estudiante: Borja López Pineda

Profesor: Luis Ignacio López Gómez

Índice

Resumen 2

1.- Diferencia 3

1.1.- Explicación 3

1.2.- Código 3

1.3.- Resultado 4

2.- Contar 5

2.1.- Explicación 5

2.2.- Código 5

2.3.- Resultado 6

Anexo 1. – Contar modificado 8

Resumen

Se plantean dos tipos de ejercicios, uno que manipule conjuntos y otro que manipule vectores.

Para el ejercicio de conjuntos he escogido realizar la operación diferencia en dos conjuntos para obtener todos los elementos que solo están presentes en uno de los dos conjuntos.

En el ejercicio de vectores he implementado un conjunto de reglas para contar el número de elementos que contiene un vector. Se incluyen dos versiones, una sencilla que requiere modificar el vector original y otra más costosa desde el punto de vista de la memoria, que deja intacto el vector original.

Diferencia

Explicación

Sean A y B dos conjuntos, A - B es la diferencia entre el conjunto A y el B. El conjunto diferencia entre A y B contiene los elementos de A que no están presentes en B. La diferencia no es conmutativa A - B ≠ B - A.

$$A - B = \{v \mid v \in A, v \notin B\}$$

En este ejemplo calcula $(A - B) \cup (B - A)$ o lo que es lo mismo $(A \cup B) - (A \cap B)$, es decir, los elementos no comunes entre A y B sin importar el orden.

Código

```
(defrule Diferencia
      (or
            (and
                  (cadena1 $? ?x $?)
                  (not (cadena2 $? ?x $?))
            )
            (and
                  (cadena2 $? ?x $?)
                  (not (cadena1 $? ?x $?))
            )
      )
      (not (diferencia $? ?x $?))
      ?f <- (diferencia $?y)
      =>
      (retract ?f)
      (assert (diferencia ?y ?x))
)

(deffacts vectores
      (cadena1 A B C D E)
      (cadena2 B C D F)
      (diferencia)
)
```

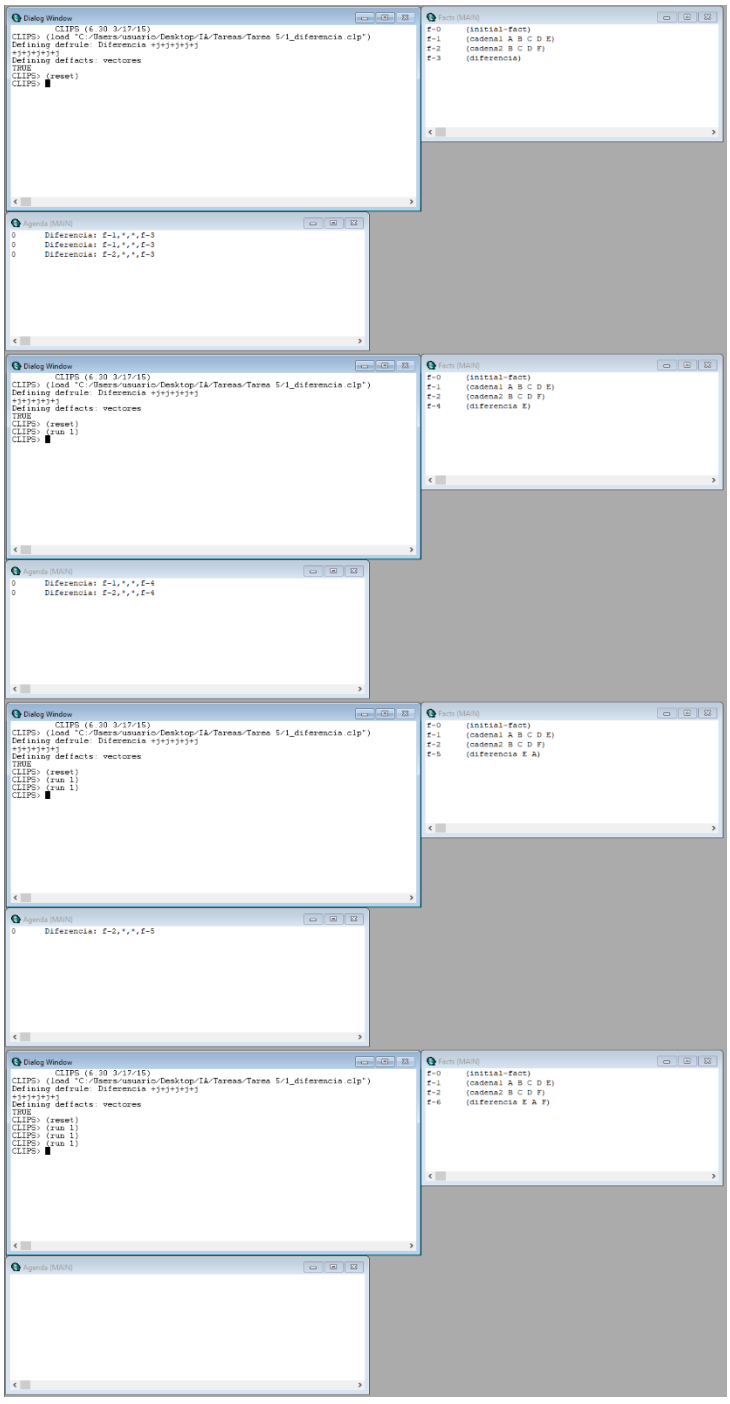
La regla busca un elemento en la cadena1 que no exista en la cadena2 o un elemento en la cadena2 que no exista en la cadena1. Estas dos condiciones son exclusivas, el mismo elemento no puede estar solo en el conjunto1 y, a la vez, estar solo en el conjunto2.

También comprueba que ese elemento no se haya introducido ya en el conjunto resultado.

SI se cumplen las condiciones, elimina el vector resultado e introduce uno nuevo con el elemento que está solo en uno de los conjuntos.

El programa termina cuando todos los elementos que están solo en un conjunto pertenecen también al conjunto resultado. El resultado se encuentra en el vector diferencia. Los vectores originales no son modificados en el proceso.

Resultado



El conjunto cadena1 tiene 2 elementos que no están presentes en cadena2 y cadena2 tiene 1 elemento que no está presente en cadena1. De ahí las tres aplicaciones de la regla Diferencia que ha encontrado clips.

En su primera ejecución encuentra el elemento E porque está presente en cadena1, pero no en cadena2.

Lo mismo ocurre con A, existe en cadena1, pero no en cadena2.

Por último, agrega el elemento F al conjunto diferencia porque está en cadena2, pero no en cadena1.

Contar

Explicación

Las siguientes reglas logran contar el número de elementos que contiene un vector. Por elemento se entiende la definición de clips de valor, una cadena de texto separada por un espacio. Es por esto, que el siguiente código es aplicable a cualquier tipo de dato.

Código

```
(defrule Iniciar
      (not (resultado ?))
      =>
      (assert (resultado 0))
)
(defrule Contar
      ?fv <- (vector ? $?x)
      ?fr <- (resultado ?c)
      =>
      (retract ?fv)
      (retract ?fr)
      (assert (vector ?x))
      (assert (resultado (+ ?c 1)))
)
(defrule Final
      (vector)
      (resultado ?c)
      =>
      (printout t "Numero de elementos: " ?c crlf)
)


(deffacts vectores
      (vector 144 6 15 4 55)
)
```

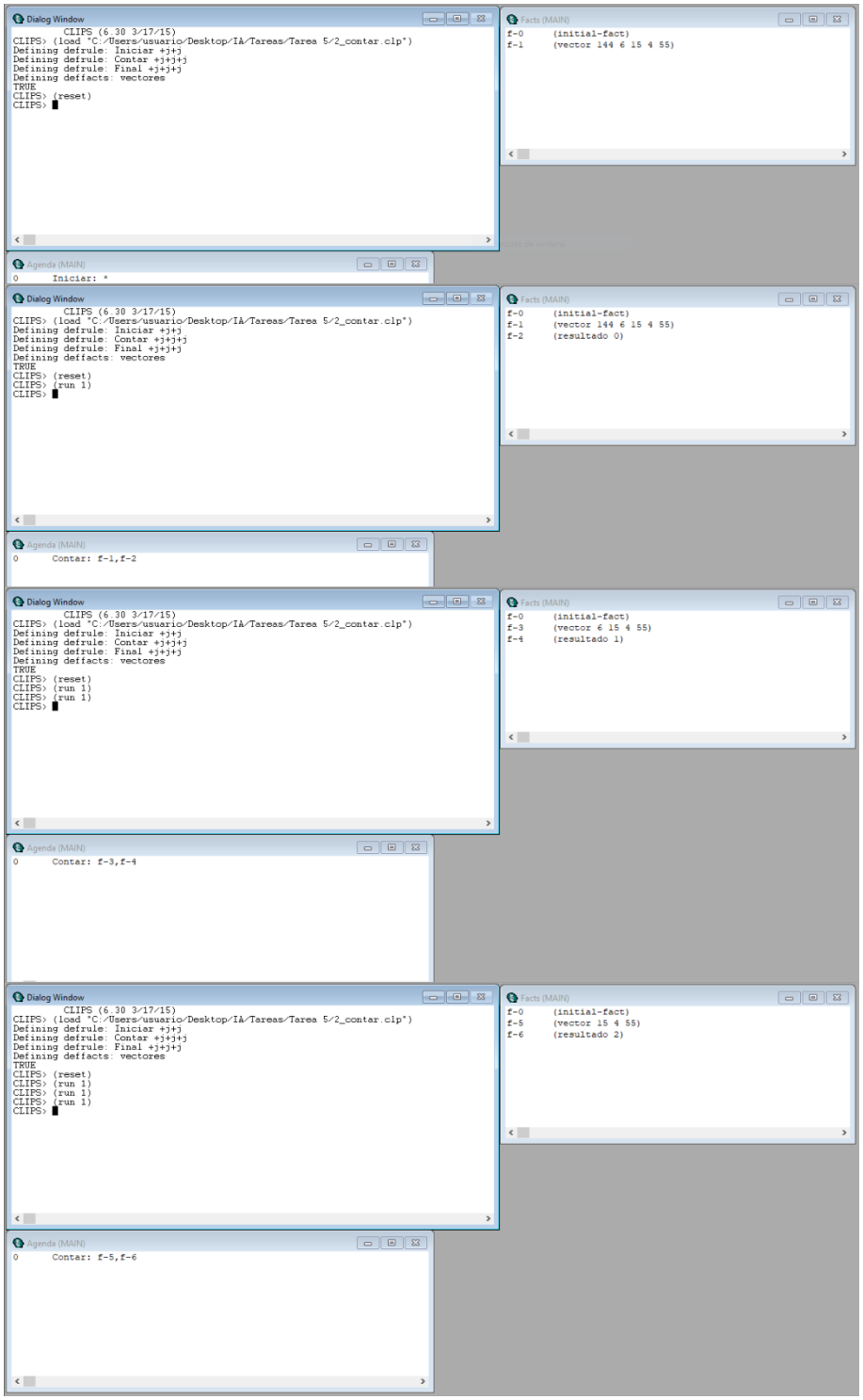
La regla Iniciar detecta que no existe el vector resultado y lo crea con el valor inicial 0.

La regla Contar busca un vector llamado vector que tenga al menos un elemento, se guarda el resto de elementos que pueda tener ese vector. Busca un vector llamado resultado con un solo valor y almacena también ese valor. Si las condiciones se satisfacen, elimina los dos vectores e inserta un nuevo vector vector sin el elemento encontrado y un nuevo vector resultado con el contador incrementado en una unidad.

La regla Final solo se ejecuta cuando el vector vector está vacío y existe un vector resultado con un elemento. Muestra el valor del resultado.

En este proceso el vector llamado vector resultado modificado, su valor se pierde. Cada vez que un elemento es contado, se elimina para evitar que cuente el mismo elemento infinitas veces. Si fuese necesario dejar intacto el vector de origen, sería posible crear un vector auxiliar donde fuéramos agregando los elementos ya contados. Evitaría un bucle infinito y dejaría intactos los datos originales.

Resultado

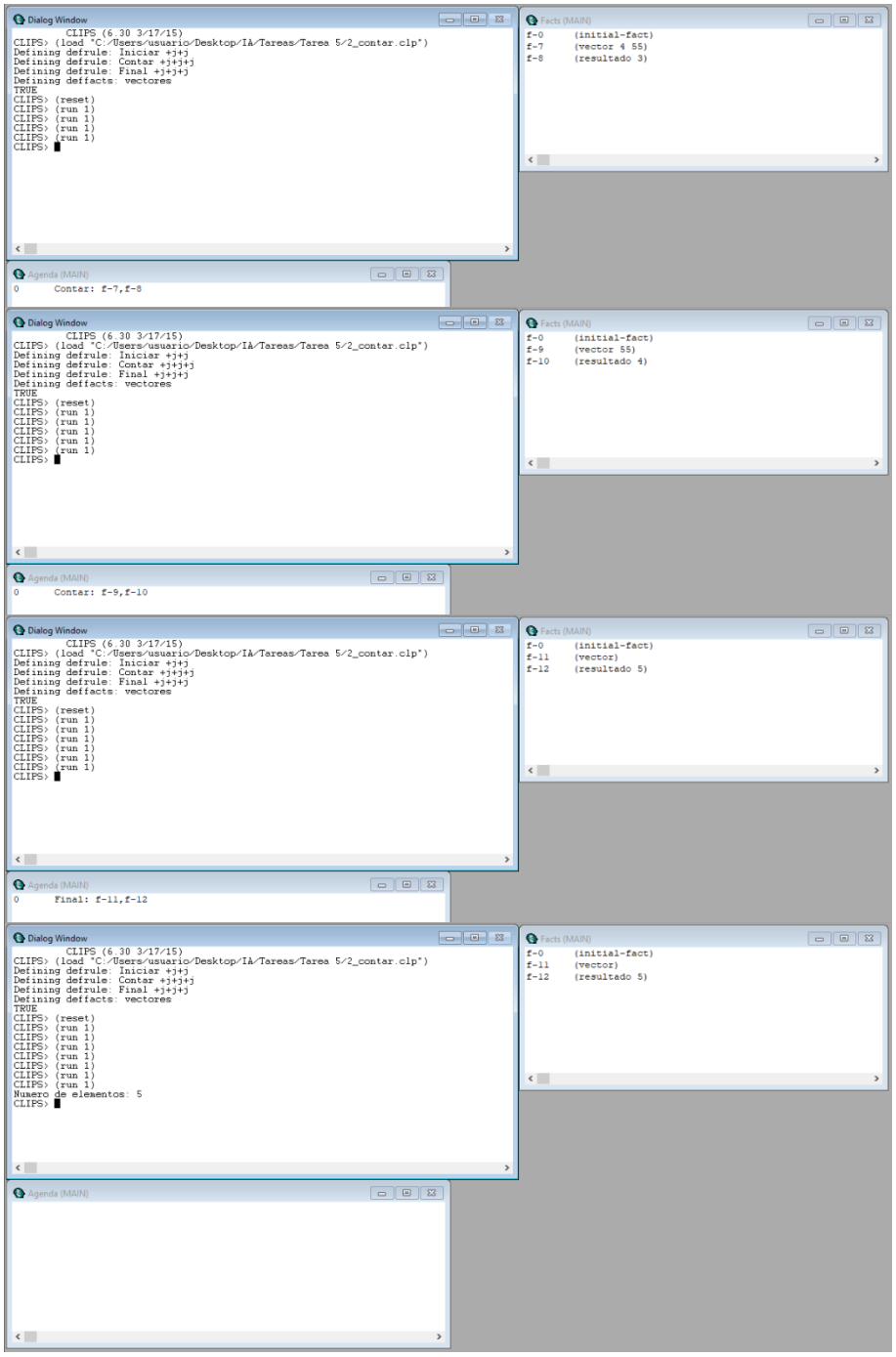


Al principio no existe el vector resultado hasta que es creado por la regla Iniciar.

El vector resultado se inicializa a 0.

En su primera aplicación la regla Contar encuentra el elemento 144 en el vector y lo elimina, el contador aumenta de 0 a 1.

El proceso se repite con el segundo elemento del vector, pasando el a valer 2 el contador.



Cuando el último elemento es contado, el vector queda vacío y se aplica la regla Final que muestra por pantalla el valor almacenado en resultado

Anexo 1. – Contar modificado

Esta revisión del código contiene las modificaciones necesarios para controlar la no repetición de elementos utilizando un vector auxiliar en lugar de modificar el vector original.

```
(defrule Iniciar
      (not (resultado ?))
      =>
      (assert (resultado 0))
      (assert (aux))
)
(defrule Contar
      (vector $? ?e $?)
      (not (aux $? ?e $?))
      ?fa <- (aux $?x)
      ?fr <- (resultado ?c)
      =>
      (retract ?fa)
      (retract ?fr)
      (assert (aux ?x ?e))
      (assert (resultado (+ ?c 1)))
)
(defrule Final
      (not
            (and
                  (vector $? ?e $?)
                  (not (aux $? ?e $?))
            )
      )
      (resultado ?c)
      =>
      (printout t "Numero de elementos: " ?c crlf)

)


(deffacts vectores
      (vector 144 6 15 4 55)
)
```