ACTIVIDAD TEMA 1 (MAUDE)

JOSÉ ÁNGEL GARCÍA MARÍN - 49478872F

JUAN RONDÁN RAMOS – 48854653P

Subgrupo 3.3 – Octubre 2024

INDICE

DIFICULTADES

Trabajar con Maude para resolver ejercicios como los que hemos abordado, plantea una serie de dificultades propias de los lenguajes de especificación formal y la programación declarativa. A diferencia de los lenguajes de programación más comunes, donde uno controla el flujo de ejecución, Maude se basa en la lógica de reescritura.

Esta diferencia de paradigma puede ser un obstáculo importante al principio, ya que el programador no define un conjunto de instrucciones que se ejecutan secuencialmente, sino que establece reglas que se aplican de manera automática cuando son válidas. Este enfoque exige una mentalidad distinta, en la que hay que confiar en que las reglas establecidas conseguirán resolver el problema.

Otra de las dificultades más generales, se concentra en la correcta definición de los tipos y subtipos. En Maude, es crucial asegurarse de que las relaciones entre los distintos tipos estén bien definidas desde el principio. Un pequeño error en esta definición y el programa generará problemas para aplicar las reglas.

Por otro lado, las dificultades también surgen en la fase de implementación de las operaciones y su semántica. El hecho de definir cualquier tipo de función implica establecer ecuaciones que deben tratar todos los casos posibles y hacerlo de manera correcta. Además de los casos base, hay que definir otras expresiones más complejas y que, en su mayoría, son recursivas, además de que dependen de los casos base, por lo que hay que ser muy cauteloso al definirlas, ya que un error no es nada fácil de detectar.

Poniéndonos en casos más concretos en relación con los ejercicios realizados, sin duda alguna los que más dificultad plantean son los de árboles. Por ejemplo, en el 141, la función "esOrdenado" llevó mucho tiempo puesto que no contemplábamos una solución que no pasara por usar funciones auxiliares que se encargaran de comprobar si existía algún nodo que colgara de los subárboles que no cumplía con el orden. Tras mucho esfuerzo, nos encontramos con que la solución pasaba por, además de comprobar que los subárboles estaban ordenados, comprobar si todos los subnodos del subárbol derecho del subárbol izquierdo son menores o iguales que el nodo raiz y si todos los subnodos del subárbol izquierdo del subárbol derecho son mayores o iguales que el nodo raiz. En concreto esta última parte fue lo que más dificil nos resultó implementar sin utilizar las funciones auxiliares.

Situándonos ahora en el ejercicio 118, queremos recalcar la mala forma en la que Maude notifica de los errores. En este ejercicio concreto, justamente en la función "posición", Maude señalaba un error lo que a la ecuación se refería. Sin embargo, después de modificar dicha función en varias ocasiones, gastando una buena cantidad de tiempo reformulando la expresión, nos percatamos de que el error no era de la función en sí, sino que se trataba de un fallo tonto en el aparatado de sintaxis, y que nuestra ecuación era correcta desde un inicio, lo cual fue algo frustrante.

Uno de los ejercicios "básicos" (ya que es del primer apartado) que más nos ha costado, ha sido el 106, consiguiendo el aceptado de Maude un poco por sorpresa. Las funciones de la raíz cuadrada y la raíz enésima han supuesto un gran reto. Hay que resaltar que el ejercicio se ha resuelto utilizando operaciones auxiliares para ambas expresiones de la raíz, aunque en un inicio se trató de no hacerlo, ha sido la única forma en la que hemos conseguido que el programa funcione.

En cuanto a las funciones auxiliares, Raíz1 encontrará el mayor número natural m tal que el cuadrado de m sea menor o igual a n y Raíz2 tiene un 3º parámetro c, que se irá incrementando progresivamente hasta encontrar el mayor c tal que c^n sea menor o igual a m.

LISTADO DE EJERCICIOS

Los ejercicios que hemos conseguido resolver son los siguientes:

- 101. Naturales Básicos. Envío: 603.
- 102. Operaciones Multiplicativas. Envío: 637.
- 103. Sustracciones. Envío: 6602.
- 104. Comparaciones. Envío: 6751.
- 105. Divisiones. Envío: 5596.
- 106. Operaciones Avanzadas. Envío: 6684.
- 107. Naturales Completos y Factorización. Envío: 5635.
- 110. Vocales. Envío: 1445.
- 111. Pilas de Vocales. Envío: 1451.
- 112. Pilas Avanzadas. Envío: 1474.
- 113. Colas de Vocales. Envío: 1720.
- 114. Colas Avanzadas. Envío: 3016.
- 115. Listas de Vocales. Envío: 1823.
- 116. Acceso a las Listas. Envío: 2426.
- 117. Modificadores de Listas. Envío: 2449.
- 118. Consultas sobre Listas. Envío: 5670.
- 120. Conjuntos de Vocales. Envío: 3411.
- 121. Aritmética de Conjuntos. Envío: 3426.
- 122. Comparación de Conjuntos. Envío: 3469.
- 125. Bolsas de Vocales. Envío: 4701.
- 126. Bolsas Avanzadas. Envío: 4821.
- 140. Árboles Binarios de Vocales. Envío: 4258.
- 141. Árboles Binarios Avanzados. Envío: 6004.
- 142. Recorridos en árboles. Envío: 8417

Total: 24 ejercicios resueltos.

101. NATURALES BÁSICOS. ENVÍO: 603

********** NOMBRE **********

fmod NATURAL is

********** CONJUNTOS **********

protecting BOOL .

sort N .

sort NoN .

subsort NoN < N .

********** SINTAXIS **********

op cero : -> N .

op sucesor : N -> N .

op suma : N N -> N .

op esCero : N -> Bool .

op esIgual : N N -> Bool .

op esDistinto : N N -> Bool .

op NODEFINIDO : -> NoN .

op INFINITO : -> NoN .

op NEGATIVO : -> NoN .

********** SEMANTICA **********

var n m : N .

eq suma(cero, n) = n .

eq suma(sucesor(m), n) = sucesor(suma(m, n)) .

eq esCero(cero) = true .

eq esCero(sucesor(n)) = false .

eq esIgual(cero, n) = esCero(n) .

```
  eq esIgual(sucesor(n), cero) = false .
  eq esIgual(sucesor(n), sucesor(m)) = esIgual(n, m) .
  eq esDistinto(n, m) = not esIgual(n, m) .
endfm
```

102. OPERACIONES MULTIPLICATIVAS. ENVÍO: 637

Importado de 101. Cambios y aditivos (Tipo abstracto Natural):

********** SINTAXIS **********

```
op producto : N N -> N .
op potencia : N N -> N .
op cuadrado : N -> N .
op factorial : N -> N .
```

********** SEMANTICA **********

```
eq producto(cero, n) = cero .
eq producto(sucesor(n), m) = suma(m, producto(n, m)) .
eq potencia(cero, cero) = NODEFINIDO .
eq potencia(n, cero) = sucesor(cero) .
eq potencia(m, sucesor(n)) = producto(m, potencia(m, n)) .
eq cuadrado(n) = producto(n, n) .
eq factorial(cero) = sucesor(cero) .
eq factorial(sucesor(n)) = producto(sucesor(n), factorial(n)) .
```

103. SUSTRACCIONES. ENVÍO: 6602

Importado de 102. Cambios y aditivos (Tipo abstracto Natural):

********** SINTAXIS **********

```
op predecesor : N -> N .
op resta : N N -> N .
op diferencia : N N -> N .
op difUno : N N -> Bool .
```

********** SEMANTICA **********

```
eq predecesor(cero) = NEGATIVO .
eq predecesor(sucesor(n)) = n .
eq resta(cero, cero) = cero .
eq resta(cero, sucesor(m)) = NEGATIVO .
eq resta(sucesor(n), cero) = sucesor(n) .
eq resta(sucesor(n), sucesor(m)) = resta(n, m) .
eq diferencia(cero, cero) = cero .
eq diferencia(cero, sucesor(n)) = sucesor(n) .
eq diferencia(sucesor(n), cero) = sucesor(n) .
eq diferencia(sucesor(n), sucesor(m)) = diferencia(n, m) .
eq difUno(cero, cero) = true .
eq difUno(sucesor(n), cero) = esCero(n) .
eq difUno(cero, sucesor(n)) = esCero(n) .
eq difUno(sucesor(n), sucesor(m)) = difUno(n, m) .
```

104. COMPARACIONES. ENVÍO: 6751

Importado de 103. Cambios y aditivos (Tipo abstracto Natural):

********** SINTAXIS **********

```
  op esMenor : N N -> Bool .
  op esMenorIgual : N N -> Bool .
  op esMayor : N N -> Bool .
  op esMayorIgual : N N -> Bool .
  op maximo : N N -> N .
  op minimo : N N -> N .
```

********** SEMANTICA **********

```
  eq esMenor(cero, cero) = false .
  eq esMenor(cero, sucesor(n)) = true .
  eq esMenor(sucesor(n), cero) = false .
  eq esMenor(sucesor(n), sucesor(m)) = esMenor(n, m) .
  eq esMenorIgual(n, m) = esIgual(n, m) or esMenor(n, m) .
  eq esMayor(n, m) = esMenor(m, n) .
  eq esMayorIgual(n, m) = not esMenor(n, m) .
  eq maximo(n, m) = if esMenor(n, m) then m else n fi .
  eq minimo(n, m) = if esMenor(n, m) then n else m fi .
```

105. DIVISIONES. ENVÍO:5596

Importado de 104. Cambios y aditivos (Tipo abstracto Natural):

********** SINTAXIS **********

```
  op division : N N -> N .
  op modulo : N N -> N .
  op mitad : N -> N .
  op esPar : N -> Bool .
```

********** SEMANTICA **********

```
  eq division(cero, cero) = NODEFINIDO .
  eq division(n, cero) = INFINITO .
  eq division(cero, n) = cero .
  eq division(n, n) = sucesor(cero) .
  eq division(n, m) = if esMayorIgual(n, m) then sucesor(division(resta(n, m), m)) else cero fi .
  eq modulo(cero, cero) = NODEFINIDO .
  eq modulo(n, cero) = NODEFINIDO .
  eq modulo(cero, n) = cero .
  eq modulo(n, n) = cero .
  eq modulo(n, m) = resta(n, producto(m, division(n, m))) .
  eq mitad(cero) = cero .
  eq mitad(n) = division(n, sucesor(sucesor(cero))) .
  eq esPar(cero) = true .
  eq esPar(sucesor(cero)) = false .
  eq esPar(sucesor(sucesor(n))) = esPar(n) .
```

106. OPERACIONES AVANZADAS. ENVÍO: 6684

Importado de 105. Cambios y aditivos (Tipo abstracto Natural):

********** SINTAXIS **********

op logaritmo : N -> N .

op raiz : N -> N .

op raiz : N N -> N .

op raiz1 : N N -> N .

op raiz2 : N N N -> N .

********** SEMANTICA **********

var n m c : N .

eq logaritmo(cero) = INFINITO .

eq logaritmo(sucesor(cero)) = cero .

eq logaritmo(n) = suma(sucesor(cero), logaritmo(division(n, sucesor(sucesor(cero))))) .

eq raiz1(n, m) = if esMenorIgual(cuadrado(m), n) then division(cuadrado(m), m) else raiz1(n, predecesor(m)) fi .

eq raiz(cero) = cero .

eq raiz(sucesor(cero)) = sucesor(cero) .

eq raiz(n) = if esMenorIgual(cuadrado(n), n) then sucesor(cero) else raiz1(n, n) fi .

eq raiz2(n, m, c) = if esMayor(potencia(c, n), m) then division(potencia(predecesor(c), n), potencia(predecesor(c), predecesor(n))) else raiz2(n, m, sucesor(c)) fi .

eq raiz(cero, n) = NODEFINIDO .

eq raiz(n, cero) = cero .

eq raiz(sucesor(cero), n) = n .

eq raiz(n, m) = raiz2(n, m, sucesor(cero)) .

107. NATURALES COMPLETOS Y FACTORIZACIÓN.
ENVÍO: 5635

Importado de 105. Cambios y aditivos (Tipo abstracto Natural):

********** SINTAXIS **********

```
  op mcd : N N -> N .

  op mcm : N N -> N .

  op coprimos : N N -> Bool .
```

********** SEMANTICA **********

```
  eq mcd(cero, m) = m .

  eq mcd(n, cero) = n .

  eq mcd(n, m) = if esMayor(n, m) then mcd(resta(n, m), m) else
mcd(resta(m, n), n) fi .

  eq mcm(cero, m) = cero .

  eq mcm(n, cero) = cero .

  eq mcm(n, m) = division(producto(n, m), mcd(n, m)) .

  eq coprimos(cero, m) = false .

  eq coprimos(n, cero) = false .

  eq coprimos(n, m) = esIgual(mcd(n, m), sucesor(cero)) .
```

110. VOCALES. ENVÍO: 1445

********** NOMBRE **********

fmod VOCAL is

********** CONJUNTOS **********

protecting BOOL .

sort V .

********** SINTAXIS **********

ops A E I O U : -> V .

op esIgual : V V -> Bool .

op esDistinta : V V -> Bool .

op esMenor : V V -> Bool .

********** SEMANTICA **********

var v w : V .

eq esIgual(v, v) = true .

eq esIgual(v, w) = false .

eq esDistinta(v, w) = not esIgual(v, w) .

ceq esMenor(A, w) = true if w =/= A .

ceq esMenor(E, w) = true if w =/= A and w =/= E .

ceq esMenor(I, w) = true if w =/= A and w =/= E and w =/= I .

ceq esMenor(O, w) = true if w =/= A and w =/= E and w =/= I and w =/= O .

eq esMenor(v, w) = false .

endfm

111. PILAS DE VOCALES. ENVÍO: 1451

Importado de 110 (Tipo Abstracto Vocal). Cambios y aditivos (Tipo abstracto Pila):

********** NOMBRE **********

fmod PILA is

********** CONJUNTOS **********

protecting BOOL .

protecting VOCAL .

sort MensajePilas .

sort P .

subsorts MensajePilas < V .

********** SINTAXIS **********

op pilaVacia : -> P .

op esVacia : P -> Bool .

op push : V P -> P .

op pop : P -> P .

op tope : P -> V .

op ERRORPILAVACIA : -> MensajePilas .

********** SEMANTICA **********

var p : P .

var v : V .

eq esVacia(pilaVacia) = true .

eq esVacia(push(v, p)) = false .

eq pop(pilaVacia) = pilaVacia .

eq pop(push(v, p)) = p .

eq tope(pilaVacia) = ERRORPILAVACIA .

```
  eq tope(push(v, p)) = v .


endfm
```

112. PILAS AVANZADAS. ENVÍO: 1474

Importado de 111 y 101 (Tipo abstracto Vocal y Tipo abstracto Natural). Cambios y aditivos (Tipo abstracto Pila):

********** CONJUNTOS **********

```
  protecting NATURAL .
```

********** SINTAXIS **********

```
  op eslgual : P P -> Bool .
  op primero : P -> V .
  op tamano  : P -> N .
  op cuentaVocal : V P -> N .
```

********** SEMANTICA **********

```
  var p q : P .
  var v w : V .
  eq eslgual(pilaVacia, pilaVacia) = true .
  eq eslgual(push(v, p), push(v, q)) = eslgual(p, q) .
  eq eslgual(p, q) = false .
  eq primero(pilaVacia) = ERRORPILAVACIA .
  eq primero(push(v, pilaVacia)) = v .
  eq primero(push(v, p)) = primero(p) .
  eq tamano(pilaVacia) = cero .
  eq tamano(push(v, p)) = sucesor(tamano(p)) .
  eq cuentaVocal(v, pilaVacia) = cero .
  eq cuentaVocal(v, push(v, p)) = sucesor(cuentaVocal(v, p)) .
  eq cuentaVocal(v, push(w, p)) = cuentaVocal(v, p) .
```

113. COLAS DE VOCALES. ENVÍO: 1720

Importado de 110 (Tipo abstracto vocal). Cambios y aditivos (Tipo abstracto Cola):

********** NOMBRE **********

fmod COLA is

********** CONJUNTOS **********

protecting BOOL .

protecting VOCAL .

sort MensajeColas .

sort C .

subsorts MensajeColas < V .

********** SINTAXIS **********

op colaVacia : -> C .

op esVacia : C -> Bool .

op meter : V C -> C .

op sacar : C -> C .

op cabecera : C -> V .

op ERRORCOLAVACIA : -> MensajeColas .

********** SEMANTICA **********

var c : C .

var v w : V .

eq esVacia(colaVacia) = true .

eq esVacia(meter(v, c)) = false .

eq sacar(colaVacia) = colaVacia .

eq sacar(meter(v, colaVacia)) = colaVacia .

eq sacar(meter(v, c)) = meter(v, sacar(c)) .

```
  eq cabecera(colaVacia) = ERRORCOLAVACIA .
  eq cabecera(meter(v, colaVacia)) = v .
  eq cabecera(meter(v, c)) = cabecera(c) .

endfm
```

114. COLAS AVANZADAS. ENVÍO: 3016

Importado de 110, 101 y 113 (Tipo abstracto Vocal, Tipo abstracto Natural y Tipo abstracto Cola). Cambios y aditivos (Tipo abstracto Cola):

********** SINTAXIS **********

```
op meterVarias : V N C -> C .
op sacarVarias : N C -> C .
op eslgual : C C -> Bool .
op tamano : C -> N .
```

********** SEMANTICA **********

```
var c d : C .
var v w : V .
var n : N .
eq meterVarias(v, cero, c) = c .
eq meterVarias(v, sucesor(n), c) = meterVarias(v, n, meter(v,c)) .
eq sacarVarias(cero, c) = c .
eq sacarVarias(sucesor(n), c) = sacarVarias(n, sacar(c)) .
eq eslgual(colaVacia, colaVacia) = true .
eq eslgual(meter(v, c), meter(v, d)) = eslgual(c, d) .
eq eslgual(c, d) = false .
eq tamano(colaVacia) = cero .
eq tamano(meter(v, c)) = sucesor(tamano(c)) .
```

115. LISTAS DE VOCALES. ENVÍO 1823

Importado de 110 y 101 (Tipo abstracto Vocal y Tipo abstracto Natural). Cambios y aditivos (Tipo abstracto Lista):

********** NOMBRE **********

fmod LISTA is

********** CONJUNTOS **********

protecting BOOL .

protecting VOCAL .

protecting NATURAL .

sort MensajeListas .

sort L .

subsorts MensajeListas < V .

********** SINTAXIS **********

op listaVacia : -> L .

op esVacia : L -> Bool .

op insertar : V L -> L .

op insertarFinal : V L -> L .

op tamano : L -> N .

op ERRORLISTAVACIA : -> MensajeListas .

********** SEMANTICA **********

var l : L .

var v w : V .

eq esVacia(listaVacia) = true .

eq esVacia(insertar(v, l)) = false .

eq insertarFinal(v, listaVacia) = insertar(v, listaVacia) .

eq insertarFinal(v, insertar(w, l)) = insertar(w, insertarFinal(v, l)) .

```
  eq tamano(listaVacia) = cero .
  eq tamano(insertar(v, l)) = sucesor(tamano(l)) .

endfm
```

116. ACCESO A LAS LISTAS. ENVÍO: 2426

Importado de 115 (Tipo abstracto Lista). Cambios y aditivos (Tipo abstracto Lista):

********** SINTAXIS **********

op primero : L -> V .

op ultimo : L -> V .

op cabeza : L -> L .

op cola : L -> L .

********** SEMANTICA **********

eq primero(listaVacia) = ErrorListaVacia .

eq primero(insertar(v, listaVacia)) = v .

eq primero(insertar(v, l)) = v .

eq ultimo(listaVacia) = ErrorListaVacia .

eq ultimo(insertar(v, listaVacia)) = v .

eq ultimo(insertar(v, l)) = ultimo(l) .

eq cabeza(listaVacia) = listaVacia .

eq cabeza(insertar(v, listaVacia)) = listaVacia .

eq cabeza(insertar(v, insertar(w, l))) = insertar(v, cabeza(insertar(w, l))) .

eq cola(listaVacia) = listaVacia .

eq cola(insertar(v, listaVacia)) = listaVacia .

eq cola(insertar(v, l)) = l .

117. MODIFICADORES DE LISTAS. ENVÍO2449

Importado de 116 (Tipo abstracto Lista). Cambios y aditivos (Tipo abstracto Lista):

********** SINTAXIS **********

```
op invertir : L -> L .
op concatenar : L L -> L .
op insertarVarias : V N L -> L .
op lista : V -> L .
op lista : V V -> L .
op lista : V V V -> L .
```

********** SEMANTICA **********

```
var m n : N .
var l k : L .
var v w u : V .
eq invertir(listaVacia) = listaVacia .
eq invertir(insertar(v, listaVacia)) = insertar(v, listaVacia) .
eq invertir(insertar(v, l)) = insertarFinal(v, invertir(l)) .
eq concatenar(listaVacia, k) = k .
eq concatenar(insertar(v, l), k) = insertar(v, concatenar(l, k)) .
eq insertarVarias(v, cero, l) = l .
eq insertarVarias(v, sucesor(m), l) = insertar(v, insertarVarias(v, m, l)) .
eq lista(v) = insertar(v, listaVacia) .
eq lista(v, w) = insertar(v, insertar(w, listaVacia)) .
eq lista(u, v, w) = insertar(u, insertar(v, insertar(w, listaVacia))) .
```

118. CONSULTAS SOBRE LISTAS. ENVÍO: 5670

Importado de 101, 110 y 115. (Tipo abstracto natural, Tipo abstracto Vocal y Tipo abstracto Lista). Cambios y aditivos (Tipo abstracto Lista):

********** SINTAXIS **********

```
  op esIgual : L L -> Bool .

  op esMenor : L L -> Bool .

  op buscar : V L -> N .

  op posicion : N L -> V .

  op ERRORFUERADELISTA : -> MensajeListas .
```

********** SEMANTICA **********

```
  var m n : N .

  var l k : L .

  var v w u : V .

  eq esIgual(listaVacia, listaVacia) = true .

  eq esIgual(insertar(v, l), listaVacia) = false .

  eq esIgual(listaVacia, insertar(w, k)) = false .

  eq esIgual(insertar(v, l), insertar(w, k)) = if esIgual(v, w) then esIgual(l, k) else false fi .

  eq esMenor(listaVacia, listaVacia) = false .

  eq esMenor(listaVacia, insertar(w, k)) = true .

  eq esMenor(insertar(v, l), listaVacia) = false .

  eq esMenor(insertar(v, l), insertar(w, k)) = if esIgual(v, w) then esMenor(l, k) else esMenor(v, w) fi .

  eq buscar(v, listaVacia) = sucesor(cero) .

  eq buscar(v, insertar(v, l)) = cero .

  eq buscar(v, insertar(w, l)) = sucesor(buscar(v, l)) .
```

```
eq posicion(n, listaVacia) = ERRORFUERADELISTA .

eq posicion(cero, insertar(v, l)) = v .

eq posicion(sucesor(n), insertar(v, l)) = posicion(n, l) .
```

120. CONJUNTOS DE VOCALES. ENVÍO: 3411

Importado de 101 y 110 (Tipo abstracto Natural y Tipo abstracto Vocal). Cambios y aditivos (Tipo Abstracto Conjunto Vocales):

********** NOMBRE **********

fmod CONJUNTOVOCALES is

********** CONJUNTOS **********

protecting BOOL .

protecting NATURAL .

protecting VOCAL .

sort C .

********** SINTAXIS **********

ops cjtoVacio : -> C .

op esVacio : C -> Bool .

op insertar : V C -> C .

op esMiembro : V C -> Bool .

op tamano : C -> N .

********** SEMANTICA **********

var c : C .

var v w : V .

eq esVacio(cjtoVacio) = true .

eq esVacio(insertar(v, c)) = false .

eq esMiembro(v, cjtoVacio) = false .

eq esMiembro(v, insertar(w, c)) = esIgual(v, w) or esMiembro(v, c) .

eq tamano(cjtoVacio) = cero .

eq tamano(insertar(v, c)) = suma(tamano(c),if esMiembro(v, c) then cero else sucesor(cero) fi) .

121. ARITMÉTICA DE CONJUNTOS. ENVÍO: 3426

Importado de 120 (Tipo abstracto Conjunto Vocales). Cambios y aditivos (Tipo Abstracto Conjunto Vocales):

********** SINTAXIS **********

```
  op union : C C -> C .

  op interseccion : C C -> C .

  op diferencia : C C -> C .

  op eliminar : V C -> C .
```

********** SEMANTICA **********

```
  var c d : C .

  var v w : V .

  eq union(cjtoVacio, cjtoVacio) = cjtoVacio .

  eq union (c, cjtoVacio) = c .

  eq union(cjtoVacio, c) = c .

  eq union(c, insertar(v,d)) = if esMiembro(v, c) then union(c, d) else
insertar(v,union(c, d)) fi .

  eq interseccion(cjtoVacio, cjtoVacio) = cjtoVacio .

  eq interseccion (c, cjtoVacio) = cjtoVacio .

  eq interseccion(cjtoVacio, c) = cjtoVacio .

  eq interseccion(c, insertar(v,d)) = if esMiembro(v, c) then insertar(v,
interseccion(c, d)) else interseccion(c, d) fi .

  eq diferencia(cjtoVacio, cjtoVacio) = cjtoVacio .

  eq diferencia(cjtoVacio, c) = cjtoVacio .

  eq diferencia(c, cjtoVacio) = c .

  eq diferencia(insertar(v,c), d) = if not esMiembro(v, d) then
insertar(v,diferencia(c, d)) else diferencia(c, d) fi .
```

eq eliminar(v,cjtoVacio) = cjtoVacio .

eq eliminar(v,c) = if esMiembro(v, c) then diferencia(c, insertar(v,cjtoVacio)) else c fi .

122. COMPARACIÓN DE CONJUNTOS. ENVÍO: 3469

Importado de 121 (Tipo abstracto Conjunto Vocales). Cambios y aditivos (Tipo Abstracto Conjunto Vocales):

********** SINTAXIS **********

op eslgual : C C -> Bool .

op esMenor : C C -> Bool .

op esMenorIgual : C C -> Bool .

op disjuntos : C C -> Bool .

********** SEMANTICA **********

var c d : C .

var v w : V .

eq eslgual(cjtoVacio, cjtoVacio) = true .

eq eslgual(c, cjtoVacio) = false .

eq eslgual(cjtoVacio, c) = false .

eq eslgual(c, d) = if eslgual(tamano(union(c, d)), tamano(c)) then if eslgual(tamano(union(c, d)), tamano(d)) then true else false fi else false fi .

eq esMenor(cjtoVacio, cjtoVacio) = false .

eq esMenor(c, cjtoVacio) = false .

eq esMenor(cjtoVacio, c) = true .

eq esMenor(c, d) = if not eslgual(c, d) then if eslgual(interseccion(c,d), c) then true else false fi else false fi .

eq esMenorIgual(cjtoVacio, cjtoVacio) = true .

eq esMenorIgual(c, cjtoVacio) = false .

eq esMenorIgual(cjtoVacio, c) = true .

eq esMenorIgual(c, d) = if eslgual(interseccion(c, d), c) then true else false fi .

eq disjuntos(c, d) = if esIgual(tamano(interseccion(c, d)), cero) then true else false fi .

125. BOLSAS DE VOCALES. ENVÍO: 4701

Importado de 105 y 110 (Tipo abstracto Natural y Tipo abstracto Vocal).
Cambios y aditivos (Tipo Abstracto Bolsa de Vocales):

********** NOMBRE **********

fmod BOLSA is

********** CONJUNTOS **********

protecting BOOL .

protecting NATURAL .

protecting VOCAL .

sort B .

********** SINTAXIS **********

op bolsaVacia : -> B .

op esVacia : B -> Bool .

op insertar : V B -> B .

op contar : V B -> N .

op eliminar : V B -> B .

********** SEMANTICA **********

var v w : V .

var b : B .

eq esVacia(bolsaVacia) = true .

eq esVacia(insertar(v, b)) = false .

eq contar(v, bolsaVacia) = cero .

eq contar(v, insertar(v, b)) = sucesor(contar(v, b)) .

eq contar(v, insertar(w, b)) = contar(v, b) .

eq eliminar(v, bolsaVacia) = bolsaVacia .

eq eliminar(v, insertar(v, b)) = b .

```
 eq eliminar(v, insertar(w, b)) = insertar(w, eliminar(v, b)) .


endfm
```

126. BOLSAS AVANZADAS. ENVÍO: 4821

Importado de 125 (Tipo Abstracto Bolsa de Vocales). Cambios y aditivos (Tipo Abstracto Bolsa de Vocales):

********** SINTAXIS **********

op union : B B -> B .

op interseccion : B B -> B .

op diferencia : B B -> B .

op esIgual : B B -> Bool .

********** SEMANTICA **********

var v w : V .

var b d : B .

eq union(b, bolsaVacia) = b .

eq union(b, insertar(v, d)) = union(insertar(v, b), d) .

eq interseccion(bolsaVacia, d) = bolsaVacia .

eq interseccion(insertar(v, b), d) = if contar(v, d) == cero then interseccion(b, d) else insertar(v, interseccion(b, eliminar(v, d))) fi .

eq diferencia(bolsaVacia, d) = bolsaVacia .

eq diferencia(insertar(v, b), d) = if contar(v, d) == cero then insertar(v, diferencia(b, d)) else diferencia(b, eliminar(v, d)) fi .

eq esIgual(bolsaVacia, bolsaVacia) = true .

eq esIgual(insertar(v, b), d) = if contar(v, d) == cero then false else esIgual(b, eliminar(v, d)) fi .

eq esIgual(bolsaVacia, d) = false .

140. ÁRBOLES BINARIOS DE VOCALES. ENVÍO: 4258

Importado de 105 y 110 (Tipo abstracto Natural y Tipo abstracto Vocal).
Cambios y aditivos (Tipo abstracto Árbol Binario):

********** NOMBRE **********

fmod ARBOLBINARIO is

********** CONJUNTOS **********

protecting BOOL .

protecting VOCAL .

protecting NATURAL .

sort AB .

********** SINTAXIS **********

op arbolVacio : -> AB .

op esVacio : AB -> Bool .

op construir : V AB AB -> AB .

op construirRaiz : V -> AB .

op altura : AB -> N .

op numNodos : AB -> N .

********** SEMANTICA **********

var a b : AB .

var v : V .

eq esVacio(arbolVacio) = true .

eq esVacio(construir(v, a, b)) = false .

eq construirRaiz(v) = construir(v, arbolVacio, arbolVacio) .

eq altura(arbolVacio) = cero .

eq altura(construir(v, a, b)) = sucesor(maximo(altura(a), altura(b))) .

eq numNodos(arbolVacio) = cero .

```
  eq numNodos(construir(v, a, b)) = sucesor(suma(numNodos(a),
numNodos(b))) .


endfm
```

141. ÁRBOLES BINARIOS AVANZADOS ENVÍO: 6004

Importado de 105, 110 Y 140 (Tipo abstracto Natural, Tipo abstracto Vocal y Tipo abstracto Árbol Binario). Cambios y aditivos (Tipo abstracto Árbol Binario):

********** SINTAXIS **********

```
  op esIgual : AB AB -> Bool .

  op esOrdenado : AB -> Bool .

  op cuentaVocal : V AB -> N .

  op cuentaHojas : AB -> N .

  op espejo : AB -> AB .
```

********** SEMANTICA **********

```
  var a b a1 a2 : AB .

  var v w x : V .

  eq esIgual(arbolVacio, arbolVacio) = true .

  eq esIgual(construir(v, a1, a2), construir(v, a, b)) =  esIgual(a1, a) and esIgual(a2, b) .

  eq esIgual(a, b) = false .

  eq esOrdenado(arbolVacio) = true .

  eq esOrdenado(construir(v, arbolVacio, arbolVacio)) = true .

  eq esOrdenado(construir(v, construir(w, a1, a2), arbolVacio)) = esOrdenado(construir(w, a1, a2)) and not esMenor(v, w) and esOrdenado(construir(v, a2, arbolVacio)) .

  eq esOrdenado(construir(v, arbolVacio, construir(w, a1, a2))) = esOrdenado(construir(w, a1, a2)) and not esMenor(w, v) and esOrdenado(construir(v, arbolVacio, a1)) .

  eq esOrdenado(construir(v, construir(w, a1, a2), construir(x, a, b))) = esOrdenado(construir(w, a1, a2)) and esOrdenado(construir(x, a, b)) and not esMenor(v, w) and not esMenor(x, v) and esOrdenado(construir(v, a2, a)) .
```

```
  eq cuentaVocal(v, arbolVacio) = cero .

  eq cuentaVocal(v, construir(w, a, b)) = suma((if esIgual(v, w) then
sucesor(cero) else cero fi), suma(cuentaVocal(v, a), cuentaVocal(v, b)))
.

  eq cuentaHojas(arbolVacio) = cero .

  eq cuentaHojas(construir(v, arbolVacio, arbolVacio)) = sucesor(cero) .

  eq cuentaHojas(construir(v, a, b)) = suma(cuentaHojas(a),
cuentaHojas(b)) .

  eq espejo(arbolVacio) = arbolVacio .

  eq espejo(construir(v, a, b)) = construir(v, espejo(b), espejo(a)) .
```

142. RECORRIDOS EN ÁRBOLES ENVÍO: 8417

Importado de 105, 110 Y 141 (Tipo abstracto Natural, Tipo abstracto Vocal y Tipo abstracto Árbol Binario). Cambios y aditivos (Tipo abstracto Árbol Binario):

********** SINTAXIS **********

op preOrden : AB -> L .

op inOrden : AB -> L .

op postOrden : AB -> L

********** SEMANTICA **********

eq preOrden(arbolVacio) = listaVacia .

eq preOrden(construir(v, a, b)) = concatenar(insertar(v, preOrden(a)), preOrden(b)).

eq inOrden(arbolVacio) = listaVacia .

eq inOrden(construir(v, a, b)) = concatenar(inOrden(a), insertar(v, inOrden(b))) .

eq postOrden(arbolVacio) = listaVacia .

eq postOrden(construir(v, a, b)) = concatenar(postOrden(a), insertarFinal(v, postOrden(b))) .

ESTIMACIÓN DE TIEMPO Y CONCLUSIONES

Para esta práctica estimamos que hemos invertido entre 18 y 20 horas de trabajo individual cada miembro, además del trabajo conjunto realizado en clase, en torno a las 3-4 horas.

Podemos afirmar que la mayor parte de este tiempo se ha empleado en los problemas más complejos, como puede ser la función esOrdenado del ejercicio 141, o en cualquier ejercicio de Árboles Binarios que han sido resueltos.

De igual modo, los ejercicios que menos tiempo nos ha llevado, han sido del 101 al 107 (excluyendo el 106), ya que, al tratarse de operaciones matemáticas relativamente sencillas, como multiplicaciones, divisiones, restas… nos ha resultado más amigable.

En cuanto a los problemas de listas y colas, creemos que son de una dificultad media, pues en estos temas tenemos algunos conocimientos de otras asignaturas, pero tampoco los suficientes como para poder decir que es "fácil".

No obstante, ambos integrantes del grupo coincidimos en que son estos ejercicios más difíciles de resolver son los que hacen mejorar nuestra capacidad de búsqueda de soluciones y resolución de problemas.

En definitiva, Maude es algo diferente a lo que estamos acostumbrados, pues como ya se mencionó anteriormente, los alumnos nos esperamos diseñar algoritmos para buscar soluciones e ir mejorándolos a base de errores. Con Maude, es distinto, solo se deben crear unas reglas a partir de las cuales el problema se resolverá de forma automática. Esto, puede sonar parecido, pero después de esta experiencia, podemos asegurar que no es así. Sin embargo, todo es cuestión de práctica y de acostumbrase, y Maude es una curiosa forma de aprender cosas nuevas.