EJERCICIOS DE COMPRENSIÓN DE ESPECIFICACIONES ALGEBRAICAS

1) Dada la especificación algebraica de la Figura 1 y sabiendo que las generadoras son las operaciones *complejoNulo*, *setReal* y *setImag*:
   a) Crear tres términos diferentes que pertenezcan al tipo *Complejo*.
   b) Indicar cuál sería el patrón de los términos canónicos del tipo *Complejo*.
   c) Indicar si existen términos creados a partir de las generadoras que sean iguales (o más técnicamente, que pertenezcan a la misma clase de equivalencia).
   d) Si realizamos varias operaciones *setReal* sobre un mismo complejo, ¿qué valor devolvería si al complejo resultante le aplicamos la operación *getReal*?

```
espec Complejo

  usa Real;

  dominios Complejo como Comp;

  operaciones
    complejoNulo : -> Comp
    setReal : Comp Real -> Comp
    setImag : Comp Real -> Comp
    getReal : Comp -> Real
    getImag : Comp -> Real

  variables
    c : Comp; x, y : Real;

  ecuaciones
     -- Impurificadoras
     setReal(setReal(c,y),x) = set(c,x)

     -- Observadoras
     getReal(complejoNulo) = 0.0
     getReal(setReal(c,x)) = x
     getReal(setImag(c,x)) = getReal(c)

     getImag(complejoNulo) = 0.0
     getImag(setReal(c,x)) = getImag(c)
     getImag(setImag(c,x)) = x

fespec
```

Figura 1. Especificación Algebraica del TAD Complejo

2) Dada la especificación algebraica de la Figura 2 y sabiendo que las generadoras son las operaciones *estVacia*, *estUnitaria* y *concatena* indicar:
   a) Cuál sería el patrón que poseen los términos canónicos del TAD representado.
   b) Si dicha especificación algebraica podría representar una lista.
   c) Si dicha especificación algebraica podría representar un conjunto.

```
espec TipoRaroNatural
  usa Natural;

  generos TipoRaroNatural como TRNat;

  operaciones
    estVacia    : -> TRNat
    estUnitaria : Natural -> TRNat
    concatena   : TRNaT TRNaT -> TRNat

  variables
    l :TRNat;

  ecuaciones
    concatena(l,estVacia) = l
    concatena(estVacia,l) = l

fespec
```

Figura 2. Especificación Algebraica TipoRaroNatural

3) Para la especificación algebraica de la Figura 3, indicar:
   a) Si el elemento al cual se accede mediante la operación *peek* es el primero añadido o el último añadido a la estructura.
   b) Si el término *peek(empty)* representa un elemento de algún tipo de datos. Por ejemplo, *peek(push(x,empty))* es una forma más compleja de representar el término *x*.

```
espec TipoDesconocido(E)

usa Booleano

parametro
  generos Element como E
fparametro

generos
  TipoDesconocido(E) como TD(E)

operaciones
  empty   :  -> TD(E)
  push    : TD(E) E -> TD(E)
  parcial peek : TD(E) -> E
  isEmpty : TD(E) -> Booleano

vars
  s : TD(E); x : Element;

precondiciones
  [isEmpty(s) != TRUE] peek(s)

ecuaciones
   // Las generadoras son emptyStack y push(s,x) y son libres
   // Los patrones del tipo son:
   //   - empty
   //   - push(s,x)

   isEmpty(empty)     = TRUE
   isEmpty(push(s,x)) = FALSE

   peek(push(s,x)) = x

fespec
```

Figura 3. Especificación Algebraica del TAD *TipoDesconocido(E)*

4) Para la especificación algebraica de la Figura 4, y teniendo en cuenta cómo funciona la operación *getHead* (obtener cabeza), indicar:
   a) Si la cabeza de la lista es el primer o último elemento añadido a la estructura.
   b) Como habría que modificar la operación *getHead* para que devolviese lo contrario a la respuesta dada para el apartado a)

```
espec Lista(E)

usa Booleano;

parametro
  generos Element como E
fparametro

generos Lista(E)

operaciones
  emptyList : -> Lista(E)
  add       : Lista(E) E -> Lista(E)
  parcial getHead : Lista(E) -> E
  isEmpty   : Lista(E) -> Booleano

vars
  l : Lista(E); x,y : Element;

precondiciones
  [NOT isEmpty(l)] getHead(l)

ecuaciones
  // Las generadoras son emptyList y add. Son libres
  // Los patrones del tipo son:
  //   - emptyList
  //   - add(l,x)

  getHead(add(emptyList,x)) = x
  getHead(add(add(l,x),y))  = getHead(add(l,x))

  isEmpty(emptyList) = TRUE
  isEmpty(add(l,x))  = FALSE

fespec
```

Figura 4. Especificación Algebraica del TAD *Lista(E)*

5) Para la especificación algebraica de la Figura 5, indicar si la operación unión se corresponde con la operación *unión* para conjuntos.

```
espec Conjunto

  usa Booleano

  parametro
    generos Element as E
    operaciones
      igual : Element Element -> Boolean
    variables
      x, y, z : Element;
    ecuaciones
      igual(x,x) = TRUE
      igual(x,y) = igual(y,x)
      [igual(x,y) and igual(y,z)] igual(x,z) = TRUE;
  fparametro

  generos Conjunto(E) como Cjt(E)

  operaciones
    emptySet : -> Cjt(E)
    add      : Cjt(E) E -> Cjt(E)
    contains : Cjt(E) E -> Booleano
    union    : Cjt(E) Cjt(E) -> Cjt(E)
  var
    s, s2 : Cjt(E); x, y: E
  ecuaciones
    // Las generadoras son emptySet y add y no son libres
    add(add(s,x),x) = add(s,x)
    add(add(s,x),y) = add(add(s,y),x)

    contains(emptySet,x) = FALSE
    [igual(x,y) == TRUE] contains(add(s,x),y) = TRUE
    [igual(x,y) != TRUE] contains(add(s,x),y) = contains(s,y)

    union(emptySet,s2) = s2
    union(add(s,x),s2) = add(union(s,s2),x)
fespec
```

Figura 5. Especificación algebraica del TAD Conjunto

6) Para la especificación algebraica de la Figura 6, indicar que hacen las operaciones *op1*, *op2*, *op3* y *op4*.

```
espec Lista(E)
usa Booleano
parametro
  generos Element como E
  operaciones
    igual : Element Element -> Boolean
    menor : Element Element -> Element
  variables
    x, y, z : Element;
  ecuaciones
    igual(x,x) = TRUE; igual(x,y) = igual(y,x)
    [igual(x,y) and igual(y,z)] igual(x,z) = TRUE;
    menor(x,x) = x;
    menor (x,y) = menor (y,x)
    [(menor(x,y) == x) and (menor(y,z) == z)] menor(x,z) = x
    [NOT(igual(x,y)) and (menor(x,y) != y)] menor(x,y) = x
  fparametro
  generos Lista(E)
  operaciones
    emptyList : -> Lista(E)
    add       : Lista(E) E -> Lista(E)
    isEmpty   : Lista(E) -> Boolean
    concatena : Lista(E) Lista(E) -> Lista(E)
    parcial op1 : Lista(E) -> E
    op2, op3  : Lista(E) E -> Lista(E)
    op4       : Lista(E)   -> Lista(E)
  vars
    l, l2 : Lista(E); x ,y : Element;
  precondiciones [NOT isEmpty(l)] op1(l)
  ecuaciones
    // Las generadoras son emptyList y add. Son libres
    // Los patrones del tipo son emptyList, add(l,x)
    isEmpty(emptyList) = TRUE; isEmpty(add(l,x)) = FALSE

    op1(add(emptyList,x)) = x
    op1(add(add(l,x),y))  = menor(op1(add(l,x)),y)

    op2(emptyList,x) = emptyList
    [igual(x,y) == TRUE]  op2(add(l,x),y)  = l
    [igual(x,y) == FALSE] op2(add(l,x),y)  = add(op2(l,y),x)

    op3(emptyList,x) = emptyList
    [igual(x,y) == TRUE]  op3(add(l,x),y)  = op3(l,y)
    [igual(x,y) == FALSE] op3(add(l,x),y)  = add(op3(l,y),x)

    concatena(emptyList,l) = l
    concatena(add(l2,x),l) = add(concatena(l2,l),x)
    op4(emptyList) = emptyList
    op4(add(l,x))  = concatena(op4(l),add(emptyList,x))
fespec
```

Pablo Sánchez Barreiro.