Ejercicio 1: Árboles binarios de búsqueda

```
fmod ARBIN is
   pr NAT .

   *** Tipos de datos y variables
   sort Arbin .
   vars A A' : Arbin .
   vars N N' C V C' V' : Nat .

   *** Constructores
   op vacio : -> Arbin [ctor] .
   op _[_,_]_ : Arbin Nat Nat Arbin -> Arbin [ctor] .


   *** Operaciones
      *** Suma dos valores dados
      op merge : Nat Nat -> Nat .
         eq merge(N, N') = N + N' .

      *** Inserta el nodo dado en el arbol binario según su clave
      op inserta : Nat Nat Arbin -> Arbin .
         eq inserta(C, V, vacio) = vacio [C, V] vacio .
         ceq inserta(C, V, A [C', V'] A') = A [C, merge(V', V)] A'
            if C == C' .
         ceq inserta(C, V, A [C', V'] A') = inserta(C, V, A) [C', V'] A'
            if C < C' .
         ceq inserta(C, V, A [C', V'] A') = A [C', V'] inserta(C, V, A')
            if C > C' .

      *** Devuelve el valor asociado a la clave indicada, dentro del árbol indicado
      op busca : Nat Arbin ~> Nat .
         ceq busca(C, A [C', V'] A') = V'
            if C == C' .
         ceq busca(C, A [C', V'] A') = busca(C, A)
            if C < C' .
         ceq busca(C, A [C', V'] A') = busca(C, A')
            if C > C' .
endfm
```

Ejercicio 2: Tienda de libros y películas

```
*******************************
*** EJERCICIO 1 - INICIO
***
fmod LISTA is
	pr STRING .

	sort Lista .
	subsort String < Lista .

	op nil : -> Lista [ctor] .
	op __ : Lista Lista -> Lista [ctor assoc id: nil] .
endfm
***
*** EJERCICIO 1 - FIN
*******************************


fmod ABB is
	pr LISTA .

	***Tipos de datos y constructores
	sort Par .
	op <_,_> : Nat Nat -> Par [ctor] .

	sort ABB .
	op vacio : -> ABB [ctor] .
	op _[_,_]_ : ABB String Par ABB -> ABB [ctor] .

	vars C C' V V' : Nat .
	vars P P' : Par .
	vars S S' : String .
	vars A A' HI HD : ABB .

	***Operadores y funciones
	op primero : Par -> Nat .
		eq primero(< C, V >) = C .

	op segundo : Par -> Nat .
		eq segundo(< C, V >) = V .

	*** La primera pareja es la que tenemos y la segunda la que queremos introducir.
	*** Sumamos la cantidad y actualizamos el precio.
	op insertarNuevo : Par Par -> Par .
		eq insertarNuevo(< C, V >, < C', V' >) = < C + C', V' > .

	op quitar : Par Nat ~> Par .
		ceq quitar(< C, V >, C') = < sd(C, C'), V >
			if C >= C' .

	op insertar : ABB String Par -> ABB .
		eq insertar(vacio, S, P) = vacio [S, P] vacio .
		ceq insertar(HI [S, P] HD, S', P') = HI [S, insertarNuevo(P, P')] HD
			if S == S' .
		ceq insertar(HI [S, P] HD, S', P') = HI [S, P] insertar(HD, S', P')
			if S < S' .
		ceq insertar(HI [S, P] HD, S', P') = insertar(HI, S', P') [S, P] HD
			if S > S' .

	op cantidad : ABB String -> Nat .
```

```
		eq cantidad(vacio, S) = 0 .
		ceq cantidad(HI [S, P] HD, S') = primero(P)
			if S == S' .
		ceq cantidad(HI [S, P] HD, S') = cantidad(HD, S')
			if S < S' .
		ceq cantidad(HI [S, P] HD, S') = cantidad(HI, S')
			if S > S' .

	op precio : ABB String ~> Nat .
		ceq precio(HI [S, P] HD, S') = segundo(P)
			if S == S' .
		ceq precio(HI [S, P] HD, S') = precio(HD, S')
			if S < S' .
		ceq precio(HI [S, P] HD, S') = precio(HI, S')
			if S > S' .

	op vender : ABB String Nat -> ABB .
		eq vender(vacio, S, C) = vacio .
		ceq vender(HI [S, P] HD, S', C) = HI [S, quitar(P, C)] HD
			if S == S' .
		ceq vender(HI [S, P] HD, S', C) = HI [S, P] vender(HD, S', C)
			if S < S' .
		ceq vender(HI [S, P] HD, S', C) = vender(HI, S', C) [S, P] HD
			if S > S' .

	*******************************
	*** EJERCICIO 2 - INICIO
	***
	op inorden : ABB -> Lista .
		eq inorden(vacio) = nil .
		eq inorden(HI [S, P] HD) = inorden(HI) S inorden(HD) .
	***
	*** EJERCICIO 2 - FIN
	*******************************
endfm


*******************************
*** EJERCICIO 3 - INICIO
***
red inorden((vacio ["Alien", < 3, 10 >] vacio) ["Gladiator", < 5, 15 >] (vacio ["Harry Potter", < 4, 20 >] vacio)) .
***
*** EJERCICIO 3 - FIN
*******************************


fmod GENTE is
	pr STRING .

	sorts Gente Comprador Representante .
	subsort Comprador Representante < Gente .

	op nadie : -> Gente [ctor] .
	op __ : Gente Gente -> Gente [ctor assoc comm id: nadie] .

	op comprador : String Nat -> Comprador [ctor] .
	op representante : String Nat Nat -> Representante [ctor] .


	*******************************
	*** EJERCICIO 6A - INICIO
	***
	sort TipoCompra .		*** Basta un natural o un booleano. Hago esto por complicarlo un poco.
```

```
        ops libro peli : -> TipoCompra [ctor] .

        op comprador : String Nat TipoCompra -> Comprador [ctor] .
        op representante : String Nat Nat TipoCompra -> Representante [ctor] .
        ***
        *** EJERCICIO 6A - FIN
        ******************************
endfm


mod TIENDA is
        pr GENTE .
        pr ABB .

        *** Catalogo de peliculas, de libros y dinero en caja.
        sort Tienda .
        op [_|_,_,_] : Gente ABB ABB Nat -> Tienda [ctor] .

        var  G : Gente .
        var  S : String .
        vars N D D' V : Nat .
        vars P P' L L' : ABB .
        var  T : TipoCompra . *** Para ejercicio 6

        crl [venta-peli] : [ comprador(S, N) G | P, L, D ]
                => [ G | vender(P, S, N), L, D + (precio(P, S) * N) ]
                if cantidad(P, S) >= N .

        crl [venta-libro] : [ comprador(S, N) G | P, L, D ]
                => [ G | P, vender(L, S, N), D + (precio(L, S) * N) ]
                if cantidad(L, S) >= N .

        crl [nuevas-pelis] : [ representante(S, N, V) G | P, L, D ]
                => [ G | insertar(P, S, < N, V * 2 >), L, sd(D, N * V) ]
                if N * V <= D .

        crl [nuevos-libros] : [ representante(S, N, V) G | P, L, D ]
                => [ G | P, insertar(L, S, < N, V * 2 >), sd(D, N * V) ]
                if N * V <= D .


        ******************************
        *** EJERCICIO 4 - INICIO
        ***
        crl [venta-peli1] : [ comprador(S, s(N)) G | P, L, D ]
                => [ comprador(S, N) G | vender(P, S, 1), L, D + precio(P, S) ]
                if cantidad(P, S) > 0 .

        crl [venta-libro1] : [ comprador(S, s(N)) G | P, L, D ]
                => [ comprador(S, N) G | P, vender(L, S, 1), D + precio(L, S) ]
                if cantidad(L, S) > 0 .

        eq comprador(S, 0) = nadie . *** Podria ser una regla
        ***
        *** EJERCICIO 4 - FIN
        ******************************


        ******************************
        *** EJERCICIO 5 - INICIO
        ***
        crl [nuevas-pelis1] : [ representante(S, s(N), V) G | P, L, D ]
                => [ representante(S, N, V) G | insertar(P, S, < 1, V * 2 >), L, sd(D, V) ]
```

```
        if V <= D .

    crl [nuevos-libros1] : [ representante(S, s(N), V) G | P, L, D ]
        => [ representante(S, N, V) G | P, insertar(L, S, < 1, V * 2 >), sd(D, V) ]
        if V <= D .

    eq representante(S, 0, V) = nadie .
    ***
    *** EJERCICIO 5 - FIN
    ******************************


    ******************************
    *** EJERCICIO 6B - INICIO
    ***
    crl [venta-peli1] : [ comprador(S, s(N), peli) G | P, L, D ]
        => [ comprador(S, N, peli) G | vender(P, S, 1), L, D + precio(P, S) ]
        if cantidad(P, S) > 0 .

    crl [venta-libro1] : [ comprador(S, s(N), libro) G | P, L, D ]
        => [ comprador(S, N, libro) G | P, vender(L, S, 1), D + precio(L, S) ]
        if cantidad(L, S) > 0 .


    eq comprador(S, 0, T) = nadie . *** Podria ser una regla

    crl [nuevas-pelis1] : [ representante(S, s(N), V, peli) G | P, L, D ]
        => [ representante(S, N, V, peli) G | insertar(P, S, < 1, V * 2 >), L, sd(D, V) ]
        if V <= D .

    crl [nuevos-libros1] : [ representante(S, s(N), V, libro) G | P, L, D ]
        => [ representante(S, N, V, libro) G | P, insertar(L, S, < 1, V * 2 >), sd(D, V) ]
        if V <= D .

    eq representante(S, 0, V, T) = nadie .
    ***
    *** EJERCICIO 6B - FIN
    ******************************
endm


******************************
*** EJERCICIO 7 - INICIO
***
mod EJEMPLO is
    pr TIENDA .

    ops c1 c2 c3 : -> Comprador .
        eq c1 = comprador("a", 3) .
        eq c2 = comprador("b", 2) .
        eq c3 = comprador("c", 2) .
        eq c3 = comprador("a", 2, peli) .

    ops r1 r2 r3 : -> Representante .
        eq r1 = representante("a", 1, 10) .
        eq r2 = representante("b", 2, 20) .
        eq r3 = representante("c", 3, 30) .
        eq r3 = representante("d", 2, 20, libro) .

    op tienda : -> Tienda .
        eq tienda = [ c1 c2 c3 r1 r2 r3 | vacio, vacio, 100] .
endm
eof
```

```
rew tienda .
frew tienda .
search tienda =>* [ G:Gente | P:ABB, L:ABB, D:Nat] s.t. D:Nat > 100 .
show path 7 .
***
*** EJERCICIO 7 - FIN
*******************************
```

Ejercicio 1: Notación Peano

```
fmod Peano is

   sort PeanoNat .
   vars N M : PeanoNat .

   op 0 : -> PeanoNat [ctor] .            *** 0 es un dato del tipo PeanoNat
   op s : PeanoNat -> PeanoNat [ctor] .   *** s(PeanoNat) es un dato del tipo PeanoNat

   op _+_ : PeanoNat PeanoNat -> PeanoNat [assoc comm] .
      eq [s1] : 0 + N = N .
      eq [s2] : s(N) + M = s(N + M) .

   op _*_ : PeanoNat PeanoNat -> PeanoNat [assoc comm] .
      eq 0 * N = 0 .
      eq s(N) * M = M + (N * M) .

   op esPositivo : PeanoNat -> Bool .
      eq esPositivo(0) = false .
      eq esPositivo(s(N)) = true .
endfm



********************
*** EJEMPLOS
********************

*** 1 + 2 = 3 | s(0) + s(s(0)) = s(s(s(0)))
red s(0) + (s(s(0))) .

*** 1 + 2 + 6 = 9 | s(0) + s(s(0)) + s(s(s(s(s(s(0)))))) = s(s(s(s(s(s(s(s(s(0)))))))))
red s(0) + s(s(0)) + s(s(s(s(s(s(0)))))) .

*** 2 * 3 = 6
red s(s(0)) * s(s(s(0))) .

*** 2 * 0 = 0
red s(s(0)) * 0 .

*** false
red esPositivo(0) .

*** true
red esPositivo(s(s(0))) .
```

Ejercicio 2: Pila de enteros

```
fmod PILA is
        pr NAT .

        ***Tipo de datos
        sort Pila .
        var N : Nat .
        var P : Pila .

        ***Constructores
        op pila-vacia : -> Pila [ctor] .
        op apila : Nat Pila -> Pila [ctor] .

        *** Operadores y funciones
        op desapila : Pila -> Pila .
                eq desapila(pila-vacia) = pila-vacia .
                eq desapila(apila(N, P)) = P .

        op cima : Pila ~> Nat .
                eq cima(apila(N, P)) = N .
endfm



**********
*** EJEMPLOS
**********

*** La pila con solo 7 se escribe apila(7, pila-vacia).
red apila(7, pila-vacia) .

*** La pila con 3 en la cima y 7 debajo se escribe apila(3, apila(7, pila-vacia))
red apila(3, apila(7, pila-vacia)) .

*** Si desapilamos nos quedamos con la anterior:
red desapila(apila(3, apila(7, pila-vacia))) .

*** La cima de la pila desapilada es 7
red cima(desapila(apila(3, apila(7, pila-vacia)))) .
```

Ejercicio 3: Lista de enteros

```
**************************
*** MODULO PERSONA
**************************
fmod PERSONA is
        pr STRING .

        ***Tipos de datos
        sort Persona .

        ***Variables
        vars S S' : String .
        vars N N' : Nat .

        ***Constructores
        op <_,_> : String Nat -> Persona [ctor] .

        ***Operadores
        ops max min : Persona Persona -> Persona [comm] .
                ceq [max] : max(< S, N >, < S', N' >) = < S, N > if N >= N' .
                ceq [min] : min(< S, N >, < S', N' >) = < S, N > if N <= N' .

        ops _<=_ _>_ : Persona Persona -> Bool .
                eq < S, N > <= < S', N' > = N <= N' .
                eq < S, N > > < S', N' > = N > N' .

        ops a b c d : -> Persona .
                eq a = < "a", 100 > .
                eq b = < "b", 80 > .
                eq c = < "c", 150 > .
                eq d = < "d", 10 > .
endfm


**************************
*** MODULO LISTA
**************************
fmod LISTA is
        pr PERSONA .

        ***Tipos de datos
        sorts Lista ListaOrd .
        subsort Persona < ListaOrd < Lista .

        ***Variables
        vars P P' : Persona .
        vars L L' : Lista .
        var  LO : ListaOrd .

        ***Constructores
        op lv : -> ListaOrd [ctor] .
        op __ : Lista Lista -> Lista [ctor assoc id: lv] .


        cmb P P' L : ListaOrd
                if P <= P' /\
                        P' L : ListaOrd .

        op head : Lista ~> Persona .
```

```
        eq head(P L) = P .

    op tail : Lista ~> Lista .
        eq tail(P L) = L .

    op tam : Lista -> Nat .
        eq tam(lv) = 0 .
        eq tam(P L) = s(tam(L)) .

    op esta? : Lista Persona -> Bool .
        eq esta?(L P L', P) = true .
        eq esta?(L, P) = false [owise] .

    op ordena : Lista -> Lista .
        ceq ordena(L P P' L') = ordena(L P' P L')
            if P > P' .
        eq ordena(L) = L [owise] .

    op inserta-ord : ListaOrd Persona -> ListaOrd .
        eq inserta-ord(lv, P) = P .
        eq inserta-ord(P L, P') = if P <= P'
            then P inserta-ord(L, P')
            else P' P L
            fi .

    op ordena-por-insercion : ListaOrd -> ListaOrd .
        eq ordena-por-insercion(lv) = lv .
        ceq ordena-por-insercion(P L) = inserta-ord(L', P)
            if L' := ordena-por-insercion(L) .
endfm


**********
*** EJEMPLOS
**********
    ***Texteo modulo PERSONA
    red a .
    red a <= b .
    red a > b .
    red max(a, b) .

    ***Texteo modulo PERSONA
    red a b c d .
    red tail(a b c d) .
    red head(a b c d) .
    red tam(a b c d) .
    red esta?(a b c, d) .
    red esta?(a b c d, d) .
    red a d .
    red d a .
    red ordena(a b c d) .
    red inserta-ord(a, b) .
    red inserta-ord(inserta-ord(inserta-ord(inserta-ord(lv, a), b), c), d) .

    ***Texteo modulo CONJUNTO
    red a, b, c, d .

    ***Texteo modulo UNIVERSIDAD
```

Ejercicio 4: Juego de vasijas

```
mod DIE-HARD is
  protecting NAT .

        *** Tipos de datos y variables
  sorts Vasija ConjVasija .
  subsort Vasija < ConjVasija .
  vars M1 N1 M2 N2 : Nat .

        *** Constructores y reglas de reescritura
  op vasija : Nat Nat -> Vasija [ctor] .   *** Capacidad / Contenido actual
        rl [vacia] : vasija(M1, N1) => vasija(M1, 0) .
        rl [llena] : vasija(M1, N1) => vasija(M1, M1) .
  op __ : ConjVasija ConjVasija -> ConjVasija [ctor assoc comm] .


        ***Operadores
        op initial : -> ConjVasija .
                eq initial = vasija(3, 0) vasija(5, 0) vasija(8,0) .

        crl [transfer1] : vasija(M1, N1) vasija(M2, N2) => vasija(M1, 0) vasija(M2, N1 + N2)
                if N1 + N2 <= M2 .

        crl [transfer2] : vasija(M1, N1) vasija(M2, N2) => vasija(M1, sd(N1 + N2, M2)) vasija(M2, M2)
                if N1 + N2 > M2 .

Endm

***(
Maude> search [1] initial =>* vasija(N:Nat, 4) B:ConjVasija .
        search in DIE-HARD : initial =>* B:ConjVasija vasija(N:Nat, 4) .

        Solution 1 (state 75)
        states: 76  rewrites: 2134 in 0ms cpu (8ms real) (~ rewrites/second)
        B:ConjVasija --> vasija(3, 3) vasija(8, 3)
        N:Nat --> 5


Maude> show path 75 .
        state 0, ConjVasija: vasija(3, 0) vasija(5, 0) vasija(8, 0)
        ===[ rl vasija(M1, N1) => vasija(M1, M1) [label fill] . ]===>
        state 2, ConjVasija: vasija(3, 0) vasija(5, 5) vasija(8, 0)
        ===[ crl vasija(M1, N1) vasija(M2, N2) => vasija(M1, sd(M2, N1 + N2))
        vasija(M2, M2) if N1 + N2 > M2 = true [label transfer2] . ]===>
        state 9, ConjVasija: vasija(3, 3) vasija(5, 2) vasija(8, 0)
        ===[ crl vasija(M1, N1) vasija(M2, N2) => vasija(M1, 0) vasija(M2, N1 + N2)
        if N1 + N2 <= M2 = true [label transfer1] . ]===>
        state 20, ConjVasija: vasija(3, 0) vasija(5, 2) vasija(8, 3)
        ===[ crl vasija(M1, N1) vasija(M2, N2) => vasija(M1, 0) vasija(M2, N1 + N2)
        if N1 + N2 <= M2 = true [label transfer1] . ]===>
        state 37, ConjVasija: vasija(3, 2) vasija(5, 0) vasija(8, 3)
        ===[ rl vasija(M1, N1) => vasija(M1, M1) [label fill] . ]===>
        state 55, ConjVasija: vasija(3, 2) vasija(5, 5) vasija(8, 3)
        ===[ crl vasija(M1, N1) vasija(M2, N2) => vasija(M1, sd(M2, N1 + N2))
        vasija(M2,  M2) if N1 + N2 > M2 = true [label transfer2] . ]===>
        state 75, ConjVasija: vasija(3, 3) vasija(5, 4) vasija(8, 3)
)
```

Ejercicio 5: Elección de líder en un anillo

```
load model-checker.maude

*** MODULO LIDER
*****************************
mod LIDER is
        pr QID .

        ***Tipos de datos, subdatos y variables
        sorts Mode Node Configuration Attribute AttributeSet Msg .
        subsort Attribute < AttributeSet .
        subsort Msg Node < Configuration .
        var  AtS : AttributeSet .
        vars N N' : Nat .
        vars O O' : Qid .

        *** Constructores para los nodos
        ops idle activo lider : -> Mode [ctor] .

        op mt : -> AttributeSet [ctor] .
        op _,_ : AttributeSet AttributeSet -> AttributeSet [ctor assoc comm id: mt] .

        op modo`:_ : Mode -> Attribute [ctor] .
        op prioridad`:_ : Nat -> Attribute [ctor] .
        op siguiente`:_ : Qid -> Attribute [ctor] .

        op none : -> Configuration [ctor] .
        op __ : Configuration Configuration -> Configuration [ctor assoc comm id: none] .

        ***Constructores para los mensajes
        op <_|_> : Qid AttributeSet -> Node [ctor] .
        op to_:_ : Qid Nat -> Msg [ctor] .


        rl [activar] : < O  | modo : idle, siguiente : O', prioridad : N >
                => < O  | modo : activo, siguiente : O', prioridad : N >
                (to O' : N) .

        crl [borrar] :
                (to O : N)
                < O | modo : activo, prioridad : N', AtS >
        => < O | modo : activo, prioridad : N', AtS >
        if N < N' .

        crl [reenviar] :
                (to O : N)
                < O | modo : activo, prioridad : N', siguiente : O' >
        => < O | modo : activo, prioridad : N', siguiente : O' >
                (to O' : N)
        if N > N' .

        rl [lider] :
                (to O : N)
                < O | modo : activo, prioridad : N, AtS >
        => < O | modo : lider, prioridad : N, AtS > .

        op init : -> Configuration .
        eq init = < 'n1 | modo : idle, prioridad : 1, siguiente : 'n2 >
                < 'n2 | modo : idle, prioridad : 2, siguiente : 'n3 >
                < 'n3 | modo : idle, prioridad : 3, siguiente : 'n4 >
```

```
                < 'n4 | modo : idle, prioridad : 4, siguiente : 'n5 >
                < 'n5 | modo : idle, prioridad : 5, siguiente : 'n1 > .
endm


*** MODULO LIDER
*****************************
mod INVARIANTE is
        pr LIDER .

        var AtS : AttributeSet .
        var C : Configuration .
        var O : Qid .

        op numLideres : Configuration -> Nat .
                eq numLideres(< O | modo : lider, AtS > C) = 1 + numLideres(C) .
                eq numLideres(C) = 0 [owise] .

        op 1lider : Configuration -> Bool .
                eq 1lider(C) = numLideres(C) <= 1 .
endm

rew init .
search init =>* C:Configuration s.t. not 1lider(C:Configuration) .


*** MODULO PROPS
*****************************
mod PROPS is
        pr SATISFACTION .
        pr LIDER .

        subsort Configuration < State .
        var AtS : AttributeSet .
        var C : Configuration .
        var O : Qid .

        op hayLider : -> Prop [ctor] .
        ops idle activo esLider : Qid -> Prop [ctor] .

        eq < O | modo : lider, AtS > C |= hayLider = true .
        eq < O | modo : lider, AtS > C |= esLider(O) = true .
        eq < O | modo : idle, AtS > C |= idle(O) = true .
        eq < O | modo : idle, AtS > C |= activo(O) = true .
endm


*** MODULO CHECK
*****************************
mod CHECK is
        pr PROPS .
        pr MODEL-CHECKER .
        pr LTL-SIMPLIFIER .

***( Disponible en LIDER
  op init : -> Configuration .
  eq init = < 'n1 | modo : idle, prioridad : 1, siguiente : 'n2 >
        < 'n2 | modo : idle, prioridad : 2, siguiente : 'n3 >
        < 'n3 | modo : idle, prioridad : 3, siguiente : 'n4 >
        < 'n4 | modo : idle, prioridad : 4, siguiente : 'n5 >
        < 'n5 | modo : idle, prioridad : 5, siguiente : 'n1 > .
)
endm
```

```
red modelCheck(init, <> hayLider -> <> esLider('n5)) .
red modelCheck(init, <> esLider('n5)) .
red modelCheck(init, idle('n1) -> <> idle('n1)) .
```