Especificación y Verificación de Software

Máster en Investigación en Informática (UCM)

Hoja 2 Curso 2009/2010

Ejercicio 1 Dada la especificación ecuacional

```
fmod NAT is
  sorts Nat .
  op 0 : -> Nat .
  op s : Nat -> Nat .
  op _+_ : Nat Nat -> Nat .

  vars X Y : Nat .

  eq X + 0 = X .
  eq X + s(Y) = s(X + Y) .
endfm
```

encuentra un álgebra que la satisfaga y en la que la interpretación del operador `_+_` no sea una operación conmutativa.

Ejercicio 2 Considera el siguiente módulo que calcula el mínimo de dos enteros:

```
fmod MIN1 is
  protecting INT .
  op minimum : Int Int -> Int [comm] .
  vars I J : Int .
  ceq minimum(I, J) = I if I <= J .
endfm
```

Explica por qué `red minimum(8,5)` devuelve `5` al ejecutar la especificación `MIN1` en Maude.

Ejercicio 3 Considera el siguiente módulo para computar el elemento más pequeño de un multiconjunto de números:

```
fmod NAT-MSET-MIN is
  sorts Nat NatMSet .
  subsort Nat < NatMSet .
  op 0 : -> Nat [ctor] .
  op s : Nat -> Nat [ctor] .
  op _ _ : NatMSet NatMSet -> NatMSet [assoc comm ctor] .
  op _<_ : Nat Nat -> Bool .
  op min : NatMSet -> Nat .
  vars N M : Nat .
  var S : NatMSet .
  eq 0 < s(N) = true .
  eq s(N) < 0 = false .
  eq s(N) < s(M) = N < M .
  eq min(N N S) = min(N S) .
  ceq min(N M S) = min(N S) if N < M .
  ceq min(N M) = N if N < M .
  eq min(N) = N .
endfm
```

Este módulo tiene dos funciones que no están completamente definidas, esto es, que al aplicarlas sobre algunos términos cerrados no se reducen a constructores.

- Identifica esas dos funciones mediante su evaluación sobre argumentos apropiados.
- Corrige la especificación.

Ejercicio 4 Define un tipo de datos `ListQid` para listas de identificadores con apóstrofo (`Qid`) y otro `Set-ListQid` para conjuntos de listas de dichos identificadores. Define sobre ellos las operaciones que sean necesarias para poder especificar una función

```
op perm : ListQid -> Set-ListQid .
```

que tome como argumento una lista de `Qid`s y devuelva el conjunto de *todas* las permutaciones de esta lista. *Indicación*: Una idea para generar todas las permutaciones de una lista como `'a : 'b : 'c` es generar "`'a` junto con las permutaciones de `'b : 'c`" y "`'b` junto con las permutaciones de `'a : 'c`" y "`'c` junto con las permutaciones de `'a : 'b`".

Solución: La idea es utilizar una función auxiliar `p(Q, L1, L2)` que genere las permutaciones de `Q : L1 : L2` que comienzan con `Q`, donde el siguiente símbolo a utilizar se coge de `L1` y los símbolos de `L2` ya han sido utilizados.

```
fmod PERM is
  pr QID .

  sorts ListQid Set-ListQid .
  subsort Qid < ListQid < Set-ListQid .

  op nil : -> ListQid .
  op _:_ : ListQid ListQid -> ListQid [assoc id: nil] .

  op empty : -> Set-ListQid .
  op __ : Set-ListQid Set-ListQid -> Set-ListQid [assoc comm id: empty] .

  op perm : ListQid -> Set-ListQid .
  op p : Qid ListQid ListQid -> Set-ListQid .
  op add : Qid Set-ListQid -> Set-ListQid .

  vars Q Q2 : Qid .
  vars L L1 L2 L3 : ListQid .
  var S : Set-ListQid .

  eq perm(nil) = empty .
  eq perm(Q : L) = p(Q, L, nil) .

  eq add(Q, empty) = empty .
  eq add(Q, L S) = (Q : L) add(Q, S) .

  eq p(Q, nil, nil) = Q .
  eq p(Q, nil, Q2 : L3) = add(Q, perm(Q2 : L3)) .
  eq p(Q, Q2 : L2, L3) = add(Q, perm(Q2 : L2 : L3)) p(Q2, L2, Q : L3) .
endfm
```

Ejercicio 5 Definimos un tipo de datos `Text` como sigue:

```
sorts Letter Text .
subsort Letter < Text .
ops a b c d e f g h i j k l m n o p q r s t u v w x y z : -> Letter [ctor] .
op __ : Text Text -> Text [ctor assoc]
```

1. Define una función

```
op isPalindrome : Text -> Bool .
```

que compruebe si un término cerrado de género `Text` es un palíndromo.

2. Define una función

   ```
   op _isPrefixOf_ : Text Text -> Bool .
   ```

   tal que t `isPrefixOf` t' sea `true` si y solo si t es un prefijo de t'.

3. Define una función

   ```
   op _isSubstringOf_ : Text Text -> Bool .
   ```

   tal que t `isSubstringOf` t' sea `true` si y solo si el texto t existe en alguna posición de t'. Por ejemplo, `a b c isSubstringOf a b d b e a b c x` es `true` mientras que `a b c isSubstringOf a b` es `false`.

Extendemos ahora el tipo `Text` con un género `Pattern` cuyos elementos son `Text`os que, además, pueden contener el carácter '?'. Este carácter corresponde al carácter comodín en UNIX y "encaja" con cualquier letra:

```
sort Pattern .
subsort Text < Pattern .
op ? : -> Patter [ctor] .
op __ : Pattern Pattern -> Pattern [ctor assoc] .
```

4. Extiende la función `_isPrefixOf_` a una función

   ```
   op _isPrefixOf_ : Pattern Text -> Bool .
   ```

   tal que p `isPrefixOf` t sea `true` si y solo si p "encaja" con el comienzo de t cuando cada ocurrencia de ? se puede reemplazar por cualquier letra.

5. Extiende la función `_isSubstringOf_` a una función sobre patrones

   ```
   op _isSubstringOf_ : Pattern Text -> Bool .
   ```