Métodos Formales en Ingeniería del Software

Ejercicios Maude

Juan José Méndez Torrero
i42metoj@uco.es

Universidad de Córdoba

10 de mayo de 2019

Índice

1. Naturales I

1.1. Ejercicio 1

Definir un módulo para sumar dos números naturales.

```
fmod SUMA is
    sort Nat .
    op 0 : -> Nat .
    op s_ : Nat -> Nat .
    op _+_ : Nat Nat -> Nat .

    vars M N : Nat .

    eq 0 + M = M .
    eq s_(M) + N = M + s_(N) .
endfm
```

1.2. Ejercicio 2

Definir un módulo para restar dos números naturales.

```
fmod RESTA is
    including SUMA .
    op _-_ : Nat Nat -> Nat .

    vars M N : Nat .

    eq M - 0 = M .
    eq 0 - M = 0 .
    eq s_(M) - s_(N) = M - N .
endfm
```

1.3. Ejercicio 3

Definir un módulo para calcular la diferencia entre dos números naturales.

```
fmod DIFERENCIA is
    sort Nat .
    op 0 : -> Nat .
    op s_ : Nat -> Nat .
    op _dif_ : Nat Nat -> Nat .

    vars M N : Nat .

    eq M dif 0 = M .
    eq 0 dif N = N .
    eq s_(M) dif s_(N) = M dif N .
endfm
```

1.4. Ejercicio 4

Definir el módulo MinusFive, que dado un número le resta 5.

```
fmod MINUSFIVE is
    sort Nat .
    op 0 : -> Nat .
    op s_ : Nat -> Nat .
    op minusfive_ : Nat -> Nat .

    vars M N : Nat .

    eq minusfive 0 = 0 .
    eq minusfive s_(0) = 0 .
    eq minusfive s_(s_(0)) = 0 .
    eq minusfive s_(s_(s_(0))) = 0 .
    eq minusfive s_(s_(s_(s_(0)))) = 0 .
    eq minusfive s_(s_(s_(s_(s_(M))))) = M .
endfm
```

1.5. Ejercicio 5

Definir el módulo que calcule el n-ésimo número par, siendo el primer par 2.

```
fmod NPAR is
    sort Nat .
    op 0 : -> Nat .
    op s_ : Nat -> Nat .
    op par_ : Nat -> Nat .
    op _+_ : Nat Nat -> Nat .

    vars N M : Nat .

    eq 0 + N = N .
    eq s N + M = N + s M .

    eq par 0 = 0 .
    eq par s 0 = s s 0 .
    eq par s N = s s 0 + par N .
endfm
```

1.6. Ejercicio 6

Calcular el n-ésimo número impar, siendo el primer impar 1.

```
fmod IMPAR is
    sort Nat .
    op 0 : -> Nat .
    op s_ : Nat -> Nat .
    op impar_ : Nat -> Nat .
    op _+_ : Nat Nat -> Nat .

    vars N M : Nat .

    eq 0 + N = N .
    eq s N + M = N + s M .
```

```

    eq impar 0 = 0 .
    eq impar s 0 = s 0 .
    eq impar s N = s s 0 + impar N .
endfm
```

1.7. Ejercicio 7

Calcular el n-ésimo valor de la sucesión de Fibonacci.

```
fmod FIBONACCI is
    sort Nat .
    op 0 : -> Nat .
    op s_ : Nat -> Nat .
    op fibo_ : Nat -> Nat .
    op _+_ : Nat Nat -> Nat .

    vars N M : Nat .

    eq 0 + N = N .
    eq s N + M = N + s M .

    eq fibo 0 = 0 .
    eq fibo s 0 = s 0 .
    eq fibo s s 0 = s 0 .
    eq fibo s s N = fibo N + fibo s N .
endfm
```

1.8. Ejercicio 8

Calcular el n-ésimo término de la sucesión de triangulares.

```
fmod TRIANGULO is
    sort Nat .
    op 0 : -> Nat .
    op s_ : Nat -> Nat .
    op tri_ : Nat -> Nat .
    op _+_ : Nat Nat -> Nat .

    vars N M : Nat .

    eq 0 + N = N .
    eq s N + M = N + s M .

    eq tri 0 = 0 .
    eq tri s 0 = s 0 .
    eq tri s s 0 = s s 0 + tri s 0 .
    eq tri s N = s N + tri N .
endfm
```

1.9. Ejercicio 9

Dado un número natural, comprobar si es par o impar, devolviendo 0 en caso de que sea par y 1 en caso de que sea impar.

```
fmod PARIMPAR is
    sort Nat .
    op 0 : -> Nat .
    op s_ : Nat -> Nat .
    op parimpar_ : Nat -> Nat .

    vars N : Nat .

    eq parimpar 0 = 0 .
    eq parimpar s 0 = s 0 .

    eq parimpar s s N = parimpar N .
endfm
```

1.10. Ejercicio 10

Calcular la suma de los N primeros números pares.

```
fmod SUMAPAR is
    including NPAR .

    op sumapar_ : Nat -> Nat .

    vars N : Nat .

    eq sumapar s 0 = 0 .
    eq sumapar s s 0 = s s 0 .
    eq sumapar s N = par N + sumapar N .
endfm
```

1.11. Ejercicio 11

Calcular el cociente de la división.

```
fmod COCIENTE is
    including SUMA .
    including RESTA .

    op _div_ : Nat Nat -> Nat .
    op _resta_ : Nat Nat -> Nat .

    vars N M : Nat .

    eq 0 resta N = 0 .
    eq N resta 0 = s 0 .
    eq N resta N = s 0 .
    eq s N resta s M = N resta M .

    eq N div 0 = 0 .
    eq 0 div N = 0 .
    eq N div N = s 0 .
    eq N div M = (N resta M) + ((N - M) div M) .
endfm
```

1.12. Ejercicio 11

Calcular el resto de la división.

```
fmod RESTO is
    including RESTA .

    op __div_ : Nat Nat Nat -> Nat .
    op _restadiv_ : Nat Nat -> Nat .

    vars N M P : Nat .

    eq M 0 div P = M .
    eq M N div 0 = 0 .
    eq M N div P = N (N - P) div P .

    eq M restadiv M = 0 .
    eq M restadiv N = M M div N .
endfm
```

2. Naturales II

2.1. Ejercicio 1

Definir un módulo para realizar la multiplicación de los números naturales.

```
fmod MULTIPLICACION is
    including SUMA .
    op _*_ : Nat Nat -> Nat .

    vars N M : Nat .

    eq N * 0 = 0 .
    eq 0 * N = 0 .

    eq N * s 0 = N .
    eq s 0 * N = N .

    eq s N * M = M + (N * M) .
endfm
```

2.2. Ejercicio 2

Definir un módulo para calcular la potencia de un número natural.

```
fmod POTENCIA is
    including MULTIPLICACION .
    op _**_ : Nat Nat -> Nat .

    vars N M : Nat .

    eq 0 ** N = 0 .
    eq N ** 0 = s 0 .

    eq N ** s M = N * (N ** M) .
endfm
```

2.3. Ejercicio 3

Definir un módulo para calcular el factorial de un número natural.

```
fmod FACTORIAL is
    including MULTIPLICACION .
    op fac_ : Nat -> Nat .

    vars N  : Nat .

    eq fac 0 = 0 .
    eq fac s 0 = s 0 .

    eq fac s N = s N * fac N .
endfm
```

2.4. Ejercicio 4

Calcular el n-ésimo cúbico: $0^3, 1^3, 2^3, 3^3, 4^3, 5^3$.

```
fmod CUBICO is
    including POTENCIA .
    op cub_ : Nat -> Nat .

    vars N : Nat .

    eq cub 0 = 0 .
    eq cub s 0 = s 0 .
    eq cub N = N ** s s s 0 .
endfm
```

2.5. Ejercicio 5

Calcular la suma de los N primeros cúbicos de la sucesión del ejercicio anterior.

```
fmod CUBICO2 is
    including CUBICO .
    op cub2_ : Nat -> Nat .

    vars N : Nat .

    eq cub2 0 = 0 .
    eq cub2 s 0 = s 0 .

    eq cub2 s N = cub s N + cub2 N .
endfm
```

3. Enteros

3.1. Ejercicio 1

Suma y resta de números enteros.
Lo primero que hacemos es declarar el módulo de los enteros:

```
fmod ENTEROS is
    including NATURALES .
    sort Int .
    subsort Nat < Int .
    op -_ : Int -> Int .

    vars N : Int .

    eq - 0 = 0 .
    eq - - N = N .
    eq s - s N = - N .
endfm
```

Seguidamente, crearemos la suma ya que la tendremos que utilizar a la hora de crear el módulo de la resta.

```
fmod SUMA is
    including ENTEROS .

    op _+_ : Int Int -> Int .
    op _-_ : Int Int -> Int .

    vars N M : Int .

    eq 0 + N = N .
    eq s M + N = M + s N .

    eq 0 + - N = - N .
    eq - N + 0 = - N .
    eq - N + s M = s - N + M .
    eq - N + - M = - (N + M) .
endfm
```

Finalemnte, con el módulo suma, crearemos el módulo de la resta:

```
fmod RESTA is
    including SUMA .
    op _-_ : Int Int -> Int .

    vars N M : Int .

    eq N - 0 = N .
    eq 0 - N = - N .
    eq s N - s M = N - M .
    eq s N - - M = N + s M .
    eq - N - M = - (N + M) .
endfm
```

3.2. Ejercicio 2

Multiplicación de números enteros.

```
fmod MULTIPLICACION is
    including SUMA .
    op _*_ : Int Int -> Int .

    vars N M : Int .

    eq N * 0 = 0 .
    eq 0 * N = 0 .

    eq N * s 0 = N .
    eq s 0 * N = N .

    eq - N * 0 = 0 .
    eq 0 * - N = 0 .

    eq - N * s 0 = - N .
    eq s 0 * - N = - N .

    eq s N * M = M + (N * M) .

    eq - s N * M = - (M + (N * M)) .
    eq N * - s M = - (N + (M * N)) .
endfm
```

3.3. Ejercicio 3

Potencia de números enteros.

```
fmod POTENCIA is
    including MULTIPLICACION .
    op _**_ : Int Int -> Int .

    vars N M : Int .

    eq 0 ** N = 0 .
    eq N ** 0 = s 0 .

    eq 0 ** N = 0 .
    eq N ** 0 = s 0 .
    eq 0 ** - N = 0 .
    eq - N ** 0 = s 0 .

    eq N ** s M = N * (N ** M) .

    eq - N ** s M = - N * (- N ** M) .
    eq N ** - M = s 0 .
endfm
```

4. Booleanos

4.1. Ejercicio 1

Definir un módulo para definir los booleanos.

```
fmod SIMPLE-BOOL is
    sort Boolean .
    including RESTA .

    op TRUE : -> Boolean .
    op FALSE : -> Boolean .

endfm
```

4.2. Ejercicio 2

Definir un módulo para usar operadores sobre booleanos: not, or, and, xor, nor, nand.

```
fmod OPERADORES is
    including TOBOOL .
    op not_ : Boolean -> Boolean .
    op _or_ : Boolean Boolean -> Boolean .
    op _and_ : Boolean Boolean -> Boolean .
    op _xor_ : Boolean Boolean -> Boolean .
    op _nor_ : Boolean Boolean -> Boolean .
    op _nand_ : Boolean Boolean -> Boolean .

    eq not TRUE = FALSE .
    eq not FALSE = TRUE .

    eq TRUE or TRUE = TRUE .
    eq TRUE or FALSE = TRUE .
    eq FALSE or TRUE = TRUE .
    eq FALSE or FALSE = FALSE .

    eq TRUE and TRUE = TRUE .
    eq TRUE and FALSE = FALSE .
    eq FALSE and TRUE = FALSE .
    eq FALSE and FALSE = FALSE .

    eq TRUE xor TRUE = TRUE .
    eq TRUE xor FALSE = FALSE .
    eq FALSE xor TRUE = FALSE .
    eq FALSE xor FALSE = TRUE .

    eq TRUE nor TRUE = FALSE .
    eq TRUE nor FALSE = FALSE .
    eq FALSE nor TRUE = FALSE .
    eq FALSE nor FALSE = TRUE .

    eq TRUE nand TRUE = FALSE .
    eq TRUE nand FALSE = TRUE .
    eq FALSE nand TRUE = TRUE .
    eq FALSE nand FALSE = TRUE .
endfm
```

4.3. Ejercicio 3

Definir un módulo para usar operadores sobre booleanos que incluya naturales: not, or, and, xor, nor, nand. Un 0 representa un false, mientras que cualquier otro numero representa un verdadero.

Para la creación de este módulo, vamos a crear un módulo auxiliar que se encargará de pasar un número natural a una variable de tipo booleano. Para ello, hemos hecho lo siguiente:

```
fmod TOBOOL is
    including SIMPLE-BOOL .
    op tobool_ : Nat -> Boolean .

    vars N : Nat .

    eq tobool 0 = FALSE .
    eq tobool N = TRUE .
endfm
```

4.4. Ejercicios 4 y 5

Definir un módulo para comparaciones entre naturales: <, <=, >, >=, ==, !=, máximo entre dos números, mínimo entre dos números, máximo entre tres números, mínimo entre tres números.

Definir un módulo para el esquema condicional IF-THEN-ELSE

```
fmod COMPARACIONES is
    including OPERADORES .
    op _<_ : Nat Nat -> Boolean .
    op _<=_ : Nat Nat -> Boolean .
    op _>_ : Nat Nat -> Boolean .
    op _>=_ : Nat Nat -> Boolean .
    op _===_ : Nat Nat -> Boolean .
    op _!=_ : Nat Nat -> Boolean .
    op max__ : Nat Nat -> Nat .
    op min__ : Nat Nat -> Nat .
    op max3___ : Nat Nat Nat -> Nat .
    op min3___ : Nat Nat Nat -> Nat .
    op if_then_else_ : Boolean Nat Nat -> Nat .
    op if_then_else_ : Boolean Boolean Boolean -> Boolean .


    vars N M O : Nat .
    vars P Q : Boolean .

    eq 0 < s N = TRUE .
    eq N < 0 = FALSE .
    eq s N < s M = N < M .

    eq 0 <= 0 = TRUE .
    eq 0 <= s N = TRUE .
    eq s N <= 0 = FALSE .
    eq s N <= s M = N <= M .

    eq 0 > s N = FALSE .
```

```
    eq N > 0 = TRUE .
    eq s N > s M = N > M .

    eq 0 >= 0 = TRUE .
    eq s N >= 0 = TRUE .
    eq 0 >= s N = FALSE .
    eq s N >= s M = N >= M .

    eq s N === 0 = FALSE .
    eq 0 === s N = FALSE .
    eq 0 === 0 = TRUE .
    eq s N === s M = N === M .

    eq s N != 0 = TRUE .
    eq 0 != s N = TRUE .
    eq 0 != 0 = FALSE .
    eq s N != s M = N != M .

    eq max N M = (N - M) + M .

    eq min N M = (N + M) - max N M .

    eq max3 N M O = max max N M max M O .

    eq min3 N M O = min min N M min M O .

    eq if TRUE then N else M = N .
    eq if FALSE then N else M = M .

    eq if TRUE then P else Q = P .
    eq if FALSE then P else Q = Q .
endfm
```

4.5. Ejercicio 6

Determinar si un número es par.

```
fmod PAR is
    including COMPARACIONES .
    op par_ : Nat -> Boolean .

    vars N : Nat .

    eq par 0 = TRUE .
    eq par s 0 = FALSE .
    eq par s s N = par N .
endfm
```

4.6. Ejercicio 7

Determinar si un número es par.

```
fmod PAR is
    including COMPARACIONES .
    op par_ : Nat -> Boolean .

    vars N : Nat .
```

```

    eq par 0 = TRUE .
    eq par s 0 = FALSE .
    eq par s s N = par N .
endfm
```

4.7. Ejercicio 8

Determinar si un número es impar.

```
fmod IMPAR-BOOL is
    including COMPARACIONES .
    op impar_ : Nat -> Boolean .

    vars N : Nat .

    eq impar 0 = FALSE .
    eq impar s 0 = TRUE .
    eq impar s s N = impar N .
endfm
```

4.8. Ejercicio 9

Determinar si un número está incluido en la sucesión de Fibonacci.

```
fmod ISFIB is
    including FIBONACCI .
    op aux___ : Nat Nat Nat -> Boolean .
    op isFibo_ : Nat -> Boolean .

    var N M P : Nat .

    eq aux N s(s(N)) P = FALSE .
    eq aux N M P = if (N === P) then TRUE else (aux N (s(M)) (fibo (s(M)))) .
    eq isFibo N = aux N 0 (fibo 0) .
endfm
```

4.9. Ejercicio 10

Determinar si un número es divisible por otro.

```
fmod DIVISIBLE is
    including RESTO .

    op _divisible_ : Nat Nat -> Boolean .

    vars N M P : Nat .

    eq N divisible M = if (N div M) * M === N  then TRUE else FALSE .
endfm
```

4.10. Ejercicio 11

Determinar si un número es primo.

```
fmod ISPRIMO is
    including DIVISIBLE .
    op isprim_ : Nat -> Boolean .
    op aux__ : Nat Nat -> Boolean .
    op _====_ : Boolean Boolean -> Boolean .

    var N M : Nat .

    eq TRUE ==== FALSE = FALSE .
    eq FALSE ==== TRUE = FALSE .
    eq TRUE ==== TRUE = TRUE .
    eq FALSE ==== FALSE = TRUE .

    eq aux N s 0 = TRUE .
    eq aux N N = TRUE .
    eq aux s s 0 s s 0 = TRUE .

    eq aux N M = if ((N div M) * M === N) ==== TRUE then FALSE else aux N s M .

    eq isprim N = aux N s(s(0)) .
endfm
```

5. Listas

Para estos ejercicios sólo se ha creado un módulo en el que hemos incluido los cinco ejercicios de Listas.

```
fmod LISTA is
    including NATURALES .
    sort List .
    op nill : -> List .
    op __ : List Nat -> List .
    op append : List Nat -> List .
    op length : List -> Nat .
    op pop : List -> Nat .
    op popback : List -> Nat .
    op auxreverse : List -> List .
    op reverse : List -> List .

    vars N M : Nat .
    vars L P : List .

    eq length(nill) = 0 .
    eq length(L N) = s length(L) .

    eq append(nill,0) = nill 0 .
    eq append(L,N) = L N .

    eq pop(nill) = 0 .
    eq pop(nill N) = N .
    eq pop(L N M) = pop(L N) .

    eq popback(nill) = 0 .
    eq popback(nill N) = N .
    eq popback(L N M) = popback(L M) .

    eq auxreverse(nill) = nill .
    eq auxreverse(nill N) = nill .
    eq auxreverse(L N) = auxreverse(L) N .

    eq reverse(nill) = nill .
    eq reverse(L) = reverse(auxreverse(L)) pop(L) .
endfm
```