**Algoritmos y Estructuras de Datos**
**Ingeniería en Informática, Curso 2º, Año 2003/2004**

# SEMINARIO DE
# ESPECIFICACIONES ALGEBRAICAS

Contenidos:

1. Descripción general de Maude
2. Comandos básicos
3. Formato de especificación
4. Ejemplos

Ejercicios

**OJO:** Antes de hacer esta práctica repasar las especificaciones algebraicas o axiomáticas (Tema 1 de la Parte 1 de la asignatura).

# 1. Descripción general de Maude

- **Maude** es una herramienta que permite escribir y ejecutar especificaciones formales axiomáticas. Automatiza el proceso de **reducción** de expresiones.

- Utiliza un lenguaje de especificación muy parecido al visto en clase. **Partes de la especificación**: nombre del módulo y del tipo definido, nombres de los conjuntos usados, sintaxis de las operaciones y semántica.

- Los TAD se llaman **sort** y los axiomas **equation**.

- ¡**Cuidado:** la sintaxis es muy estricta!

- **Página web de Maude**: http://maude.cs.uiuc.edu/

- Utilizaremos la **versión 1**: http://maude.cs.uiuc.edu/maude1/

- Descarga, instalación y ejecución (versión 1 para Linux):

```
>> wget http://maude.cs.uiuc.edu/maude1/current/system/maude-linux.tar.Z
>> gunzip -c maude-linux.tar.Z | tar -xvf -
>> cd maude-linux/bin
>> ./maude.linux
```

- Para salir: **`quit`**

# 2. Comandos básicos

| Sintaxis | Significado |
|---|---|
| **`in`** *`nombreFich`* **`.`** | Lee y procesa el archivo con nombre `nombreFich`. |
| **`red`** *`expresión`* **`.`** | Reduce una expresión, usando los axiomas definidos para el tipo. |
| **`quit`** | Salir del programa |

- ¡¡No olvidar terminar las expresiones con "  **`.`**" (espacio en blanco + punto)!!

- **Modo de uso**.
  - Escribir una especificación formal axiomática en un archivo, usando un editor de textos cualquiera.
  - Ejecutar **Maude**.
  - Cargar el fichero con el comando **`in`**.
  - Si hay errores, ejecutar **`quit`** y corregir la especificación.
  - Una vez que la especificación esté bien, probar expresiones usando el comando **`red`**.
  - Salir.
  - Las expresiones de ejemplo (para ejecutar con **`red`**) también se pueden incluir en otro fichero o en el mismo.

# 3. Formato de especificación

**NOMBRE**
Natural
**CONJUNTOS**
N Conjunto de naturales
B Conjunto de booleanos {true, false}

```
fmod NATURAL is
   protecting BOOL .
   sort N .
```

**fmod** NOMBRE **is** → Nombre del módulo que se está definiendo.

**protecting** NOMBRE . → Nombre de los módulos que se importan, es decir, de los tipos usados en la definición. El módulo **BOOL** está definido y contiene el tipo **Bool** de los booleanos (true, false, and, or, etc.). Puede importarse más de un módulo.

**sort** NOMBRE . → Nombre del conjunto del TAD que estamos definiendo en este módulo.

**SINTAXIS**
cero: → N
sucesor: N → N
esCero: N → B
igual: N x N → B
suma: N x N → N

```
op cero : -> N .
op sucesor : N -> N .
op esCero : N -> Bool .
op igual : N N -> Bool .
op suma : N N -> N .
```

Respetar la sintaxis:
- Espacios en blanco entre cada una de las partes de la descripción.
- No poner la x del producto vectorial.
- Acabar con: espacio en blanco + punto.

**SEMANTICA**
∀ m, n ∈ N

1. esCero (cero) = true
2. esCero (sucesor (n)) = false
3. igual (cero, n) = esCero (n)
4. igual (sucesor (n), cero) = false
5. igual (sucesor (n), sucesor (m)) = igual (n, m)
6. suma (cero, n) = n
7. suma (sucesor (m), n) = sucesor (suma (m, n))

```
vars n, m : N .

eq esCero (cero) = true .
eq esCero (sucesor (n)) = false .
eq igual (cero, n) = esCero (n) .
eq igual (sucesor (n), cero) = false .
eq igual (sucesor (n), sucesor (m)) = igual (n, m) .
eq suma (cero, n) = n .
eq suma (sucesor (m), n) = sucesor (suma (m, n)) .
```

**vars** n, m : N . → Nombre de las variables que se van a usar y su tipo.
**var** b : Bool .
**eq** exp1 = exp2 . → Axioma (**eq** → **equation**).

**FIN** Natural

```
endfm
```

- Ejecutar expresiones de ejemplo:

```
Maude> red suma (sucesor(sucesor (cero)), sucesor (sucesor (cero))) .

Maude> red esCero(suma(sucesor(sucesor(cero)), sucesor(cero))) .
```

- Para mostrar los axiomas aplicados en cada paso:

```
Maude> set trace on .

Maude> red esCero(sucesor(sucesor(cero)) .
```

- Para desactivar la traza:

```
Maude> set trace off .
```

- Para guardar los resultados en disco:
  - Escribir la especificación y las reducciones en un fichero. Por ejemplo, `ejemplo.maude`
  - Ejecutar desde la línea de comandos, redirigiendo la salida a un fichero:
    `>> ./maude.linux ejemplo.maude > salida.txt`
  - Analizar los resultados en el fichero de salida.

# 4. Ejemplos

## 4.1. Fichero: `natural.maude`

http://dis.um.es/~ginesgm/files/doc/natural.maude

```
fmod NATURAL is
   protecting BOOL .
   sort N .

   op cero : -> N .
   op sucesor : N -> N .
   op esCero : N -> Bool .
   op igual : N N -> Bool .
   op suma : N N -> N .

   vars n, m : N .

   eq esCero (cero) = true .
   eq esCero (sucesor (n)) = false .
   eq igual (cero, n) = esCero (n) .
   eq igual (sucesor (n), cero) = false .
   eq igual (sucesor (n), sucesor (m)) = igual (n, m) .
   eq suma (cero, n) = n .
   eq suma (sucesor (m), n) = sucesor (suma (m, n)) .
endfm
```

## 4.2. Fichero: `letra.maude`

http://dis.um.es/~ginesgm/files/doc/letra.maude

```
fmod LETRA is
   protecting BOOL .
   sort L .

   op a : -> L .
   op e : -> L .
   op i : -> L .
   op o : -> L .
   op u : -> L .
   op igual : L L -> Bool .

   vars x, y : L .

   eq igual (a, a) = true .
   eq igual (e, e) = true .
   eq igual (i, i) = true .
   eq igual (o, o) = true .
   eq igual (u, u) = true .
   eq igual (x, y) = false .
endfm
```

### 4.3. Fichero: `pila.maude`

http://dis.um.es/~ginesgm/files/doc/pila.maude

```
in letra .

fmod PILA is
   protecting BOOL .
   protecting LETRA .
   sort Mensaje .
   sort S .
   subsorts Mensaje < L .

   op error : -> Mensaje .
   op crearPila : -> S .
   op esVacia : S -> Bool .
   op pop : S -> S .
   op tope : S -> L .
   op push : L S -> S .

   var s : S .
   var t : L .

   eq esVacia (crearPila) = true .
   eq esVacia (push (t, s)) = false .
   eq pop (crearPila) = crearPila .
   eq pop (push (t, s)) = s .
   eq tope (crearPila) = error .
   eq tope (push (t, s)) = t .
endfm
```

### 4.4. Fichero: `ejemplo.maude`

http://dis.um.es/~ginesgm/files/doc/ejemplo.maude

```
in natural .

set trace on .

red suma (sucesor(sucesor (cero)), sucesor (sucesor (cero))) .

set trace off .

red igual(suma(sucesor(cero), cero), sucesor(cero)) .

in pila .

red pop(push(a, push(e, pop (push(i, pop(crearPila)))))) .

red tope(pop(push(a, crearPila))) .

red push (tope(crearPila), crearPila) .

quit
```

# Ejercicios

1. (1 punto) Comprobar el resultado de las siguientes expresiones usando las especificaciones formales definidas en el apartado 4. Decir cuántos axiomas es necesario aplicar en cada caso: prueba1.maude ←

a) push(tope(push(a, crearPila)), pop(push(e, push(i, crearPila))))
b) igual(sucesor(sucesor(cero)), suma(sucesor(cero), sucesor(cero)))
c) tope(pop(pop(push(i, push(o, push(u, crearPila))))))
d) igual(e, tope(pop(push(e, pop(crearPila)))))

2. (2 puntos) Añadir a la especificación formal del TAD **Natural** las operaciones: **predecesor**, **resta**, **producto**, **potencia**, **factorial**, **esMenor, esMenorIgual**, **esPar**, **mínimo** y **máximo**. Convertir las siguientes expresiones a la notación definida y comprobar el resultado que se obtiene, indicando el número de axiomas aplicados. natural.maude ← prueba2.maude

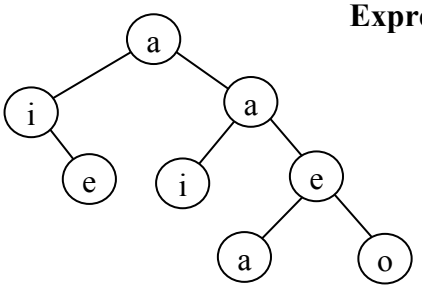
| | | | |
|---|---|---|---|
| 0) | $2^2*(3-1)$ | ¿ $3+3^2 < 3*2^3$ ? | ¿ mínimo{4!, $2^{2^2}$} ? ¿ Es par (3! -$3^2$) ? |
| 1) | $(3-2)^{(2+1)}$ | ¿ (4+2)*2*1 < 4! ? | ¿ máximo{$2^{(1+2)}$, 2!}? ¿ Es par (2+2 - 2*3) ? |
| 2) | 3+2*2-1! | ¿ mínimo{$(2-2)^3$, $2^{3-3}$}? | ¿ 3! ≤ 1-2-3 ? ¿ Es par ($2^{1+1}$-$1^2$) ? |
| 3) | $(2*3)^{(3-1-1)}$ | ¿ máximo{$2^{2!}$, $2^2$} ? | ¿ (2+1)! < $(2-3)^{2+1}$ ? ¿ Es par (4! - 3!) ? |

**Ojo**: no hacer todas las expresiones, sólo las de la fila correspondiente al valor **D** calculado con la fórmula: **D = (DNI del alumno) módulo 4**.

3. (3 puntos) Escribe una especificación formal para el TAD **lista de letras**. La especificación debe incluir las operaciones: **vacia** (devuelve una lista vacía), **insertar** (dada una lista y un elemento, inserta el elemento al principio de la lista), **concatenar** (unir dos listas), **longitud** (devuelve la longitud de una lista), **primero**, **ultimo**, **cabecera** (devuelve todos los elementos menos el último), **cola** (devuelve todos los elementos menos el primero), **invertir** (invierte el orden de los elementos de una lista), **elemento** (dada una lista y un entero *n*, devuelve el elemento *n*-ésimo de la lista, siendo el primero el 1, después el 2, etc.) y **buscar** (dada una lista y una letra, devuelve un entero que indica la primera aparición de la letra en la lista). Probar la especificación con al menos 5 expresiones de ejemplo, en las que aparezcan todas las operaciones definidas. lista.maude ← prueba3.maude

4. (4 puntos) Escribe la especificación formal del TAD **árbol binario de letras**. La especificación debe incluir las operaciones: **crear** (crea un árbol vacío), **construir** (crea un árbol, dada la raíz y dos subárboles), **hijoIzq**, **hijoDer**, **raiz** (operaciones de consulta), **altura** (calcula la altura del árbol), **numNodos** (calcula el número de nodos del árbol), **esAVL** (comprueba si cumple la condición de balanceo de los árboles AVL), **esPerfBal** (comprueba si cumple la condición de los árboles perfectamente balanceados), **preOrden**, **inOrden**, **postOrden** (recorre un árbol y almacena el resultado en una lista de letras), **contarLetra** (dado un árbol y una letra, devuelve un entero indicando el número de veces que aparece esa letra en el árbol) y **nodosNivel** (dado un árbol y un entero, devuelve una lista de letras con los nodos que están a ese nivel; se supone que la raíz tiene nivel 0, sus hijos 1, sus nietos 2, y así sucesivamente).
Escribe dos expresiones correspondientes a crear sendos árboles binarios que contengan las vocales del nombre y apellidos del alumno. Se requiere que uno de ellos sea un AVL y otro no. Dibujar a mano la estructura de los árboles creados y entregarla con la memoria de la práctica. Probar los resultados obtenidos de aplicar las 9 últimas operaciones anteriores sobre esos árboles. arbol.maude ← prueba4.maude

**Ejemplo**. Alumno: G**i**n**e**s G**a**r**c**i**a** M**a**t**e**o**s**




**Expresión:** construir(a, construir(i, crear, construir(e, crear, crear)), construir(a, construir(i, crear, crear), construir(e, construir(a, crear, crear), construir(o, crear, crear))))

# Evaluación

a) La práctica se deberá realizar individualmente.
b) Para cada ejercicio se deberán crear los ficheros con los nombres indicados al margen en la hoja anterior. Los ficheros **pruebaX.maude** deberán cargar los tipos necesarios (**in** ...) y ejecutar las pruebas (**red** ...).
c) Todos estos ficheros deberán entregarse en papel, hasta el 2 de septiembre. Si el alumno tiene cuenta de prácticas en la asignatura, los ficheros deberán estar accesibles dentro de la cuenta en un directorio: practica0 (indicar la cuenta y el password). En otro caso deberán entregarse en un disquete.
d) ¡¡No activar la traza para ejecutar las expresiones de ejemplo (**set trace off**)!!
e) Para aprobar la práctica es **imprescindible** que existan todos los ficheros .maude indicados en la lista de ejercicios, y que todos ellos puedan ser cargados sin error en el intérprete de Maude. Cualquier práctica que no funcione en Maude será puntuada con un 0.
f) No existirá entrevista obligatoria de esta práctica, aunque se podrán hacer entrevistas individualizadas si el profesor lo considera conveniente.
g) Los alumnos con nota mayor o igual a 4 (sobre 10) no deberán realizar en el examen los ejercicios correspondientes al tema 1, parte 1. La nota obtenida en esta práctica se guardará como la nota de los ejercicios del examen correspondiente.