Algoritmos y Estructuras de Datos
Ingeniería en Informática, Curso 2º, Año 2004/2005

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 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! Se diferencian mayúsculas/minúsculas.

- **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 de ejemplo 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. Un módulo puede contener varios TAD.

protecting NOMBRE . → Nombre de los módulos que se importan (los que contienen los tipos usados en la definición). El módulo **BOOL** está predefinido 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 cartesiano.
- 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))) .
```

- **Cuidado** con los paréntesis y los puntos. Si se ponen menos paréntesis de los necesarios, se queda esperando que se cierren, y parece que el programa se ha quedado colgado.

- 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 expresiones de ejemplo 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 T .

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

   vars x y : T .

   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
```

- **Ojo:** no hay ambigüedad en los axiomas. En caso de que se puedan aplicar varios axiomas para una expresión, se aplicará siempre el que aparezca primero.

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 < T .

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

   var s : S .
   var t : T .

   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
```

- **`subsorts Tipo1 < Tipo2 .`** → Las operaciones que devuelven un `Tipo2` pueden devolver también un `Tipo1`.
- Se pueden usar condicionales:
  `if` `condicion` **`then`** `valor1` **`else`** `valor2` **`fi .`**

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))) .

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

set trace off .

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 punto 4 (¡no activar la traza!). Decir cuántos axiomas es necesario aplicar en cada caso: prueba1.maude ←

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

2. (2 puntos) Añadir a la especificación formal del TAD **Natural** las operaciones: **predecesor**, **resta**, **producto**, **potencia**, **factorial**, **esMenor, esMayor**, **esImpar**, **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)	$3+2*2-1!$	¿ máximo$((4-2)^3, 2^3)$?	¿ $3! \leq 1-2-3$?	¿ Es par $(2^{1+1}-1^2)$?
1)	$2^2*(3-1)$	¿ $3^2+2 < 2*2^3$?	¿ máximo$(4!, 2^{2^2})$?	¿ Es par $(3! -3^2)$?
2)	$(2-1)^{(2+1)}$	¿ $(4+2)*2*1 < 4!$?	¿ mínimo$(3^{(2+1)}, 3!)$?	¿ Es par $(2+1 - 2*3)$?
3)	$(2*3)^{(3-2-1)}$	¿ mínimo$(2^{2!}, 2^2)$?	¿ $(2+1)! < (2-3)^{2+1}$?	¿ Es par $(4!-3!-2!)$?

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

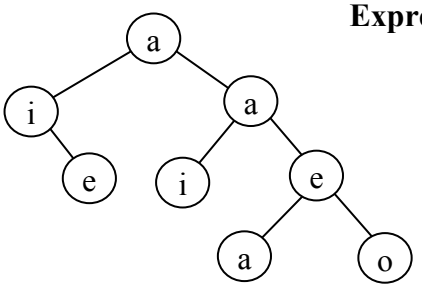
3. (3 puntos) Escribe una especificación formal para el TAD **bolsa de letras**. La especificación debe incluir las operaciones: **vacia** (devuelve una bolsa vacía), **esVacia** (comprueba si la bolsa está vacía), **poner** (mete una letra en una bolsa), **ponerVarias** (dada una letra, un natural *n* y una bolsa, mete la letra *n* veces en la bolsa), **quitar** (quita una letra dada de la bolsa), **quitarVarias** (dada una letra, un natural *n* y una bolsa, quita *n* apariciones de la letra en la bolsa), **quitarTodas** (quita todas las apariciones de una letra dada en la bolsa), **cuantos** (calcula el número de veces que aparece una letra dada en la bolsa), **cuantosTotal** (número total de elementos en la bolsa), **esIgual** (comparación entre bolsas), **union**, **interseccion** y **diferencia** (igual que con conjuntos, pero teniendo en cuenta el número de apariciones de cada letra), **comoConjunto** (dada una bolsa, devuelve otra bolsa eliminando las letras que aparezcan repetidas más de una vez, es decir, deja como máximo una aparición de cada letra). Probar la especificación con al menos 6 expresiones de ejemplo, en las que aparezcan todas las operaciones definidas. bolsa.maude ← prueba3.maude

4. (4 puntos) Escribe una 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), **altura** (calcula la altura del árbol), **anchuraNivel** (dado un árbol y un entero, calcula el número de nodos en ese nivel del árbol), **anchura** (dado un árbol, calcula el máximo número de nodos en cualquier nivel del árbol), **esAVL** (comprueba si cumple la condición de balanceo de los árboles AVL), **sacaHojas** (devuelve una bolsa con las letras de los nodos hoja), **sacaNivel** (dado un árbol y un natural *n*, devuelve una bolsa con los nodos del árbol que están a nivel *n*, siendo la raíz el nivel 0), **sacaTodos** (devuelve una bolsa con todos los elementos del árbol), **rsi** (operación de rotación simple a la izquierda), **rdi** (operación de rotación doble a la izquierda). arbol.maude ← prueba4.maude

Escribe las expresiones correspondientes a crear dos árboles binarios distintos 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. Probar los resultados obtenidos de aplicar las 8 últimas operaciones sobre esos árboles.

Ejemplo. Alumno: G**i**n**é**s G**a**rc**í**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) Para esta convocatoria especial de septiembre/diciembre, 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 módulos necesarios (**in** ...) y ejecutar las pruebas (**red** ...).
c) Todos estos ficheros deberán entregarse en papel y en formato electrónico (bien en disquete o por e-mail), como máximo hasta 2 días antes del examen correspondiente.
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) Los alumnos con nota mayor o igual a 5 (sobre 10) no deberán realizar en el examen parcial o final 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.
g) No existirá entrevista obligatoria de esta práctica, aunque se podrán hacer entrevistas individualizadas si el profesor considera necesario verificar la autoría y originalidad de la práctica.