

Algoritmos y Estructuras de Datos I
Grado en Ingeniería Informática, Curso 2º

**TEMA 1. ESPECIFICACIONES
FORMALES ALGEBRAICAS
EN MAUDE**

Contenidos:

1. Descripción general de Maude
2. Comandos básicos
3. Formato de especificación
4. Ejemplos
5. Actividad de evaluación continua del tema 1

OJO: Antes de realizar esta actividad es conveniente repasar las especificaciones formales 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. Los tipos abstractos se definen dentro de **módulos** (**fmod ... endfm**).
- **Partes de la especificación:** nombre del módulo y del tipo definido; nombre de los conjuntos usados; sintaxis de las operaciones; y semántica de las mismas.
- 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
```

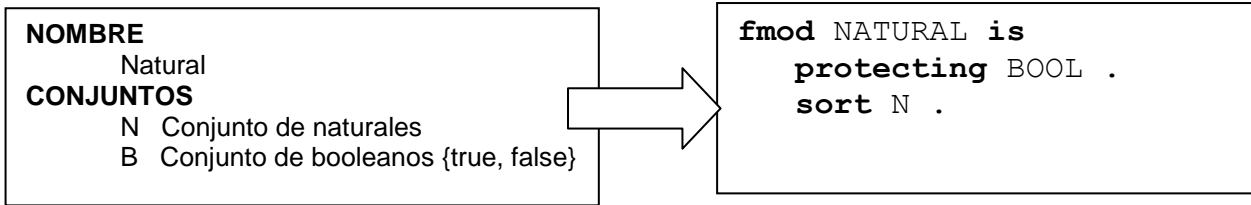
- Para salir: **quit**

2. Comandos básicos

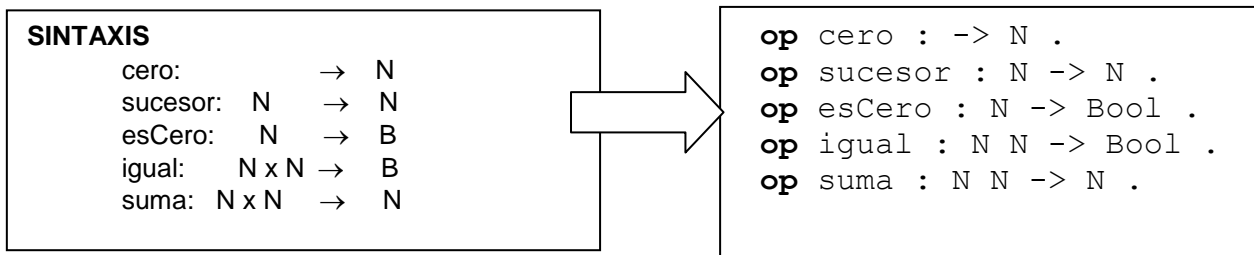
Sintaxis	Significado
in <i>nombreFich</i> .	Lee y procesa el archivo con nombre <i>nombreFich</i>
red <i>expresión</i> .	Reduce una expresión, usando los axiomas definidos para el tipo de esa expresión
quit	Salir del programa

- ¡¡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**. Comprobar que el resultado es el esperado.
 - Salir.
 - Las expresiones de ejemplo (para ejecutar con **red**) también se pueden incluir en otro fichero y ejecutar **in** con ese fichero para probarlas.

3. Formato de especificación

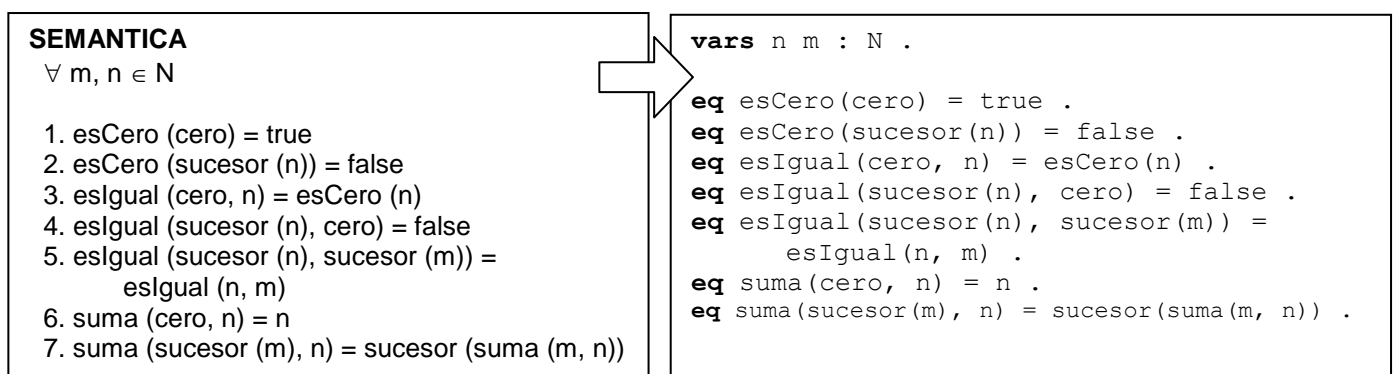


- fmod** NOMBRE **is** → Nombre del módulo que se está definiendo. Un módulo puede contener varios TAD.
- protecting** NOMBRE . → Nombre de los módulos que se importan (los que contienen los tipos usados en la definición). El módulo **BOOL** está predefinido y contiene el tipo **Bool** de los booleanos (true, false, and, or, not, etc.). Puede importarse más de un módulo.
- sort** Nombre . → Nombre del conjunto del TAD que estamos definiendo en este módulo.

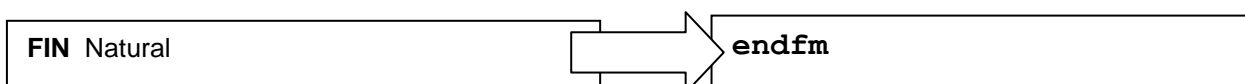


Respetar la sintaxis:

- 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.



- vars** n m : N . → Nombre de las variables que se van a usar y su tipo.
- var** b : Bool . → Usar var para una variable y vars cuando sean varias.
- eq** exp1 = exp2 . → Axioma (eq → equation).



- 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. **Consejo:** usar un editor de texto que empareje los paréntesis.

- Para mostrar los axiomas aplicados en cada paso:

```
Maude> set trace on .
```

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

- Para desactivar la traza (por defecto está desactivada):

```
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.
- **Consejo 1:** en caso de obtener un “wrong answer” o un “runtime error” en el juez on-line al hacer un envío, pinchar sobre el enlace que aparece para ver la causa del fallo. El juez mostrará una expresión donde tu especificación produce un resultado incorrecto.
- **Consejo 2:** recuerda comprobar los ejemplos de reducciones que aparecen en los enunciados de los problemas, y otras que puedas considerar convenientes. ¡Pero recuerda no incluir esas reducciones de ejemplo (**red**) en el fichero `.maude` al hacer los envíos al juez!
- **Consejo 3:** si sigues sin encontrar la causa del error, busca en el Mooshak las preguntas que otros usuarios han hecho sobre el mismo problema. Puede que alguna te sea de utilidad. Si no encuentras nada, escribe una nueva pregunta (pero, por favor, solo después de haber seguido los consejos anteriores y de haberte calentado un poco la cabeza). Los jueces humanos te contestarán en cuanto puedan. Recuerda que las preguntas son públicas para todos los usuarios de Mooshak. Si la pregunta es particular, pregunta a tu profesor en persona.

4. Ejemplos

4.1. Fichero: natural.maude

```
***** NOMBRE *****
fmod NATURAL is

***** CONJUNTOS *****
  protecting BOOL .
  sort N .
  sort NoN .
  subsort NoN < N .

***** SINTAXIS *****
  op cero : -> N .
  op sucesor : N -> N .
  op suma : N N -> N .
  op esCero : N -> Bool .
  op esIgual : N N -> Bool .
  op esDistinto : N N -> Bool .

  op NODEFINIDO : -> NoN .
  op INFINITO : -> NoN .
  op NEGATIVO : -> NoN .

***** SEMANTICA *****
  var n m : N .

  eq suma(cero, n) = n .
  eq suma(sucesor(m), n) = sucesor(suma(m, n)) .
  eq esCero(cero) = true .
  eq esCero(sucesor(n)) = false .
  eq esIgual(cero, n) = esCero(n) .
  eq esIgual(sucesor(n), cero) = false .
  eq esIgual(sucesor(n), sucesor(m)) = esIgual(n, m) .
  eq esDistinto(n, m) = not esIgual(n, m) .
endfm
```

- Para poner un comentario en Maude, se deben poner tres o más asteriscos seguidos (***) . El comentario se extiende hasta el final de la línea.

4.2. Fichero: vocal.maude

```
***** NOMBRE *****
fmod VOCAL is

***** CONJUNTOS *****
  protecting BOOL .
  sort V .

***** SINTAXIS *****
  ops A E I O U : -> V .
  op esIgual : V V -> Bool .
  op esDistinta : V V -> Bool .

***** SEMANTICA *****
  var v w : V .

  eq esIgual(v, v) = true .
  eq esIgual(v, w) = false .
  eq esDistinta(v, w) = not esIgual(v, w) .
endfm
```

- Con “ops” se pueden juntar varias operaciones que tengan la misma sintaxis.

- **Ojo:** en Maude no hay ambigüedad en los axiomas anteriores. En caso de que se puedan aplicar varios axiomas distintos para una misma expresión, se aplicará siempre el que aparezca en primer lugar.

4.3. Fichero: pila.maude

```
***** NOMBRE *****
fmod VOCAL is
***** CONJUNTOS *****
  protecting BOOL .
  sort V .
***** SINTAXIS *****
  ops A E I O U : -> V .
  op esIgual : V V -> Bool .
  op esDistinta : V V -> Bool .
***** SEMANTICA *****
  var v w : V .
  eq esIgual(v, v) = true .
  eq esIgual(v, w) = false .
  eq esDistinta(v, w) = not esIgual(v, w) .
endfm
```

```
***** NOMBRE *****
fmod PILA is
***** CONJUNTOS *****
  protecting BOOL .
  protecting VOCAL .
  sort MensajePilas .
  sort P .
  subsorts MensajePilas < V .
***** SINTAXIS *****
  op pilaVacía : -> P .
  op esVacía : P -> Bool .
  op push : V P -> P .
  op pop : P -> P .
  op tope : P -> V .
  op ERRORPILAVACIA : -> MensajePilas .
***** SEMANTICA *****
  var p : P .
  var v : V .
  eq esVacía(pilaVacía) = true .
  eq esVacía(push(v, p)) = false .
  eq pop(pilaVacía) = pilaVacía .
  eq pop(push(v, p)) = p .
  eq tope(pilaVacía) = ERRORPILAVACIA .
  eq tope(push(v, p)) = v .
endfm
```

- Para definir mensajes de error en una especificación:
 1. Definir un tipo asociado a los mensajes de error (**sort** MensajePilas).
 2. Declararlo como **subsort** del tipo adecuado:
subsorts Tipo1 < Tipo2 . → Las operaciones que devuelven un Tipo2 pueden devolver también un Tipo1.
 3. Crear una, o varias, operaciones constantes (op ERRORPILAVACIA : -> MensajePilas .)
 4. Usar dicha operación constante donde corresponda.
- También se pueden usar condicionales en los axiomas, de la forma:
if condicionBooleana **then** valor1 **else** valor2 **fi**

- Las expresiones condicionales pueden aparecer dentro de otra expresión, como por ejemplo:
`suma(cero, if condicion then sucesor(cero) else cero fi)`
- También pueden ser anidados. Cada `if` debe cerrarse con un `fi`:

```
if cond1 then if cond2 then A else B fi else C fi *** equivale a
if cond1 then ( if cond2 then A else B fi ) else C fi
```

4.4. Fichero: ejemplo.maude

```
in natural .
red suma (sucesor(sucesor (cero)), sucesor (sucesor (cero))) .
red esCero(suma(sucesor(sucesor(cero)), sucesor(sucesor(cero)))) .

set trace on .
red suma (sucesor(sucesor (cero)), sucesor (sucesor (cero))) .
red esCero(suma(sucesor(sucesor(cero)), sucesor(sucesor(cero)))) .

set trace off .
in pila .
red pop(push(a, push(e, pop (push(i, pop(pilaVacía)))))) .
red tope(pop(push(a, pilaVacía)) .
red push (tope(pilaVacía), pilaVacía) .

quit
```

5. Actividad de evaluación continua del tema 1

1. **Juez on-line:** <http://mooshak.inf.um.es>
2. **Concurso:** AED1, 15/16. Actividad T1
3. **Grupos de dos:** esta actividad se hará en grupos de 2 alumnos. Se debe usar la cuenta de uno (muy importante: hacer envíos solo con una de las dos cuentas). Los alumnos que hicieron esta actividad en cursos pasados no pueden formar grupo con un alumno que no la haya hecho nunca.
4. **Ejercicios a resolver (incluidos los ya resueltos):**
 - a. 101-107: hacer un mínimo de 4
 - b. 110-118: hacer un mínimo de 5
 - c. 120-126: hacer un mínimo de 3
 - d. 140-145: hacer un mínimo de 2
5. Durante la primera sesión de laboratorio dedicada a esta actividad (semana del 28 de septiembre al 2 de octubre), será **imprescindible la asistencia y resolver presencialmente al menos uno** de los problemas propuestos y que no estén resueltos previamente. En esta primera sesión se repartirán las cuentas a los alumnos que no las tengan.
6. **Documentación:** una vez acabada la actividad, habrá que preparar una **breve memoria** de la misma. La memoria contendrá:
 - a. Una breve descripción de cómo se han resuelto las operaciones más complejas (en total, dos hojas como máximo de descripciones).
 - b. Listado de los envíos realizados al juez (decir qué cuenta se ha usado).
 - c. El código de las especificaciones realizadas. Se puede omitir el código de las especificaciones que están incluidas dentro de otras.
 - d. Una estimación del tiempo total usado para resolver esta actividad (en horas de trabajo).
7. Los alumnos que hubieran hecho esta actividad en cursos anteriores deben repetirla. Además, deberán indicar en la memoria el año, el grupo, el compañero y la cuenta utilizada en ese curso anterior donde se hizo la actividad.
8. **Plazo:** el juez on-line estará abierto entre el 27/9/2015 y el 13/10/2015 a las 10:00. La entrega de la documentación será a lo largo de este día 13/10/2015.
9. **Forma de entrega de la memoria:**
 - a. **Profesor Norberto:** entregar **en papel** en su casillero (en la conserjería).
 - b. **Profesor Ginés:** entregar el archivo **PDF** en la Tarea correspondiente del Aula Virtual.
10. **Nota:** convalidación del tema 1 (si asistencia a clase y entrega de resúmenes). 5 puntos por hacer los ejercicios mínimos. 10 puntos por hacer 23 ejercicios. **En todo caso, los problemas deben estar aceptados en el juez on-line, pero la aceptación no es el único requisito para tener la actividad superada.**

AVISO IMPORTANTE

Existe un mecanismo de **comprobación automática** de todos los envíos, de todos los grupos, de todas las titulaciones y con los envíos de los cursos anteriores. La copia de esta práctica (fuera de la coincidencia casual) supondrá el suspenso fulminante de toda la asignatura en la convocatoria que corresponda, para los grupos implicados en la copia, con la consiguiente anulación del resto de actividades de evaluación continua.