UNIVERSIDAD DE MÁLAGA

**Métodos formales para la Ingeniería del Software**
(4º curso del grado Ingeniería del Software)
Dpto. de Lenguajes y Ciencias de la Computación
E.T.S.I. INFORMÁTICA
**Segunda Práctica Evaluable**

# Enunciado

Un *sistema de ficheros* puede describirse como una colección de objetos (Ficheros y Directorios) y un conjunto de operaciones que pueden realizarse sobre estos objetos. Los objetos se estructuran formando un árbol, que tiene un único objeto raíz que no puede copiarse, ni borrarse, ni moverse en el árbol. Cada objeto del *sistema de ficheros* (excepto la raíz) tiene un único padre, que debe ser un directorio. El objetivo de la práctica es modelar en Alloy *sistemas de ficheros* sobre los que se podrán realizar dos tipos de operaciones, las que modifican la estructura del árbol, y las que trabajan sobre los ficheros. La práctica consta de una serie de fases en las que el modelo va completándose poco a poco.

# Primera Fase

En esta primera fase, se construye la estructura en árbol de los *sistema de ficheros* sin distinguir entre *directorios* y *ficheros*.

1. Define las signaturas `Objeto` y `SistemaFicheros` y las relaciones `raiz`, `hijos`, `cont` que, respectivamente, relacionan cada *sistema de ficheros* con distintos conjuntos/relaciones del modo siguiente:

   *a*) `raiz` asocia cada *sistema de ficheros* con su objeto raíz.

   *b*) `hijos` asocia cada *sistema de ficheros* con la relacion padre/hijos dentro de la estructura, es decir, si `S` es un `SistemaFicheros`, y `O1,O2,O3` son `Objetos`, entonces las ternas `(S,O1,O2),(S,O1,O3)` podrían estar en la relación `hijos` lo que significa que, dentro del *sistema de ficheros* `S`, el objeto `O1` es padre de los objetos `O2` y `O3`.

   *c*) `cont` relaciona cada *sistema de ficheros* con el conjunto de objetos que contiene.

   La figura 1 muestra varias posibles instancias de las signaturas de esta primera fase. Cada uno de los árboles se corresponde con un *sistema de ficheros* (las imágenes se han obtenido proyectado las instancias sobre la signatura `SistemaFicheros`). Por ejemplo, el *sistema de ficheros* que está a la izquierda de la figura (`S`) tiene como raíz al `Objeto1`, la relación `hijos` viene dada por el conjunto {`(S,Objeto1,Objeto4), (S,Objeto4,Objeto2), (S,Objeto4,Objeto3)` }, y la relación `cont` asocia el sistema de ficheros con el conjunto {`Objeto1,Objeto4,Objeto2,Objeto3` }.

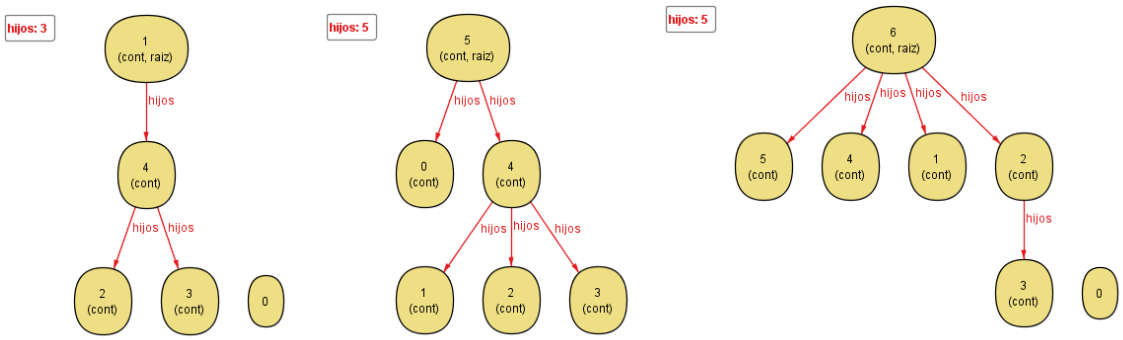


Figura 1: Ejemplos de *sistema de ficheros*

2. Añade las siguientes restricciones al modelo:

   *a*) Cada sistema de ficheros contiene un único objeto raíz que no tiene padre.

*b*) En cada sistema de ficheros, todos los objetos, salvo el raíz, tienen un único padre.

*c*) La relacion `cont` asocia cada sistema de ficheros con los nodos alcanzables (mediante la relación `hijos`) desde el nodo `raiz`

*d*) Ningún sistema de ficheros contiene ciclos.

*e*) Todos los objetos del sistema de ficheros son alcanzables desde el objeto `raiz`

3. Define las siguientes funciones:

   *a*) La función `padre` que devuelve el objeto que es el padre de `o` en el sistema de ficheros `s`

   ```
   fun padre(s:SistemaFicheros,o:Objeto):Objeto{}
   ```

   *b*) La función `descendientes` que devuelve el conjunto de objetos que son descendientes de `o` en el sistema de ficheros `s`

   ```
   fun descendientes(s:SistemaFicheros,o:Objeto):set Objeto{}
   ```

   *c*) La función `subarbol` que devuelve el subárbol del sistema de ficheros `s` que tiene como raíz el objeto `o`

   ```
   fun subarbol(s:SistemaFicheros,o:Objeto):Objeto->Objeto{}
   ```

## Segunda Fase

En esta fase, se distingue entre ficheros y directorios, se añade contenido a los ficheros, y se define una signatura que representa los buffers de la memoria del sistema.

1. Define la signatura `Objeto` como abstracta, y define dos nuevas signaturas `Directorio` y `Fichero` como los dos únicos tipos de objetos. Además, redefine las relaciones `raiz` y `hijos` de manera que la raíz y el padre de cualquier objeto del sistema deba ser un `Directorio`.

2. Define dos signaturas `Datos` y `Buffer` para representar, de forma muy abstracta, el contenido de los ficheros, y los buffers de la memoria.

3. Define la signatura abstracta `Estado`, que tiene sólo dos átomos, `Abierto` y `Cerrado`, que representan el estado de los ficheros.

4. Define las siguientes relaciones:

   *a*) `igual`, que asocia cada dato con el conjunto de datos que son iguales a él

   *b*) `bcontenido`, que asocia cada buffer con el dato (de `Datos`) que contiene, si no está vacío

   *c*) `fcontenido`, que asocia cada fichero con el dato (de `Datos`) que contiene, si no está vacío

   *d*) `estado`, que asocia cada fichero con su estado (de `Estado`)

   *e*) `fbuffers`, que asocia cada fichero con 0/1 buffer (de `Buffer`)

   *f*) Dentro de la signatura `SistemaFicheros` define la relación `buffers` que asocia a cada sistema de ficheros con el conjunto de sus buffers

5. Añade las siguientes restricciones (como `facts`) al sistema:

   *a*) La relación `igual` es simétrica, es decir, el dato `D0` es igual a `D1` si, y sólo si, `D1` es igual a `D0`.

   *b*) Un fichero tiene asociado un buffer si, y sólo si, está abierto

   *c*) No es posible que dos ficheros distintos tengan asociado el mismo buffer

   *d*) Si un fichero tiene asociado un buffer (mediante `fbuffer`), éste debe ser uno de los del sistema de ficheros en el que se encuentra

   *e*) El buffer asociado a un fichero, o está vacío o contiene un dato igual que el del fichero

   *f*) Un dato no puede ser el contenido de dos ficheros distintos

   *g*) Un dato no puede ser el contenido de dos buffers distintos

   *h*) Los buffers y los ficheros no comparten datos.

# Tercera Fase

En esta fase, se modelan los predicados que modifican la estructura de los sistemas de ficheros, y las que operan sobre los ficheros.

1. Define los siguientes predicados:

   *a)* `pred crear(s,s':SistemaFicheros,padre,o:Objeto){}`

   que construye el sistema de ficheros `s'`, añadiendo el objeto `o` al sistema de ficheros `s` como hijo de `padre`. Las precondiciones del predicado son:

   - El objeto `o` no está en el sistema de ficheros `s`
   - El objeto `padre` está en `s`

   La Figura 2 muestra el efecto que debe tener el predicado `crear` sobre el sistema de ficheros `s` (el árbol de la izquierda). En esta figura y las siguientes se han ocultado todos los átomos y relaciones que no tienen que ver con las operaciones que se definen.

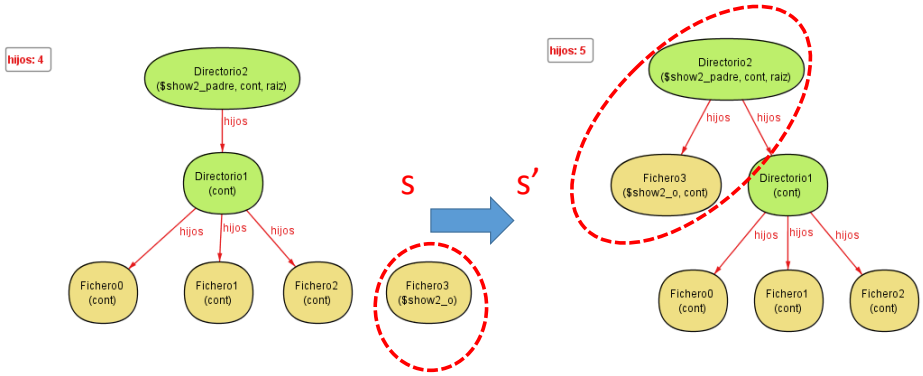


Figura 2: Ejemplo de la operación crear

   *b)* `pred mover(s,s':SistemaFicheros,o,padre:Objeto){}`

   que construye el sistema de ficheros `s'`, moviendo el subárbol que cuelga del objeto `o` (incluyéndolo) en el sistema de ficheros `s`, para pasar a ser hijo de `padre` en `s'`. Las precondiciones del predicado son:

   - El objeto `o` está en el sistema de ficheros `s`, pero no es su objeto `raiz`
   - El objeto `padre` está en `s`, y no es descendiente de `o` en `s`

   La Figura 3 muestra el efecto que debe tener el predicado `mover` sobre el sistema de ficheros `s` (el árbol de la izquierda)

   *c)* `pred borrar(s,s':SistemaFicheros,o:Objeto){}`

   que construye el sistema de ficheros `s'`, borrando el subárbol que cuelga del objeto `o` (incluyéndolo) en el sistema de ficheros `s`. La precondición del predicado es:

   - El objeto `o` está en el sistema de ficheros `s`, pero no es su objeto `raiz`

   La Figura 4 muestra el efecto que debe tener el predicado `borrar` sobre el sistema de ficheros `s` (el árbol de la izquierda)

   *d)* *Nota: la implementación del predicado copiar es difícil. Sólo debe intentarse cuando se haya terminado el resto de las fases de la práctica*

   `pred copiar(s,s':SistemaFicheros,o,padre:Objeto){}`

   que construye el sistema de ficheros `s'` a partir de `s`, copiando (duplicando) el subárbol que cuelga del objeto `o` (incluyéndolo) como hijo del objeto `padre`. La precondición del predicado es:

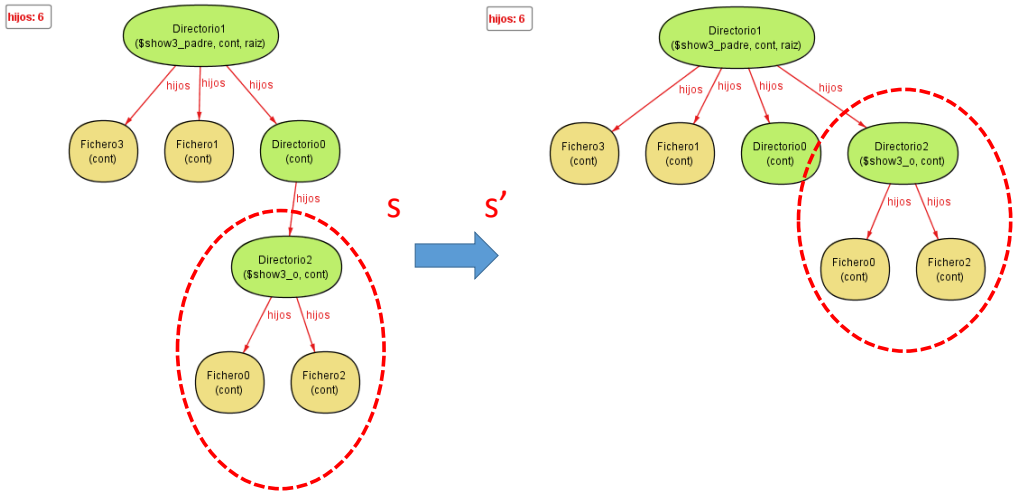


Figura 3: Ejemplo de la operación mover

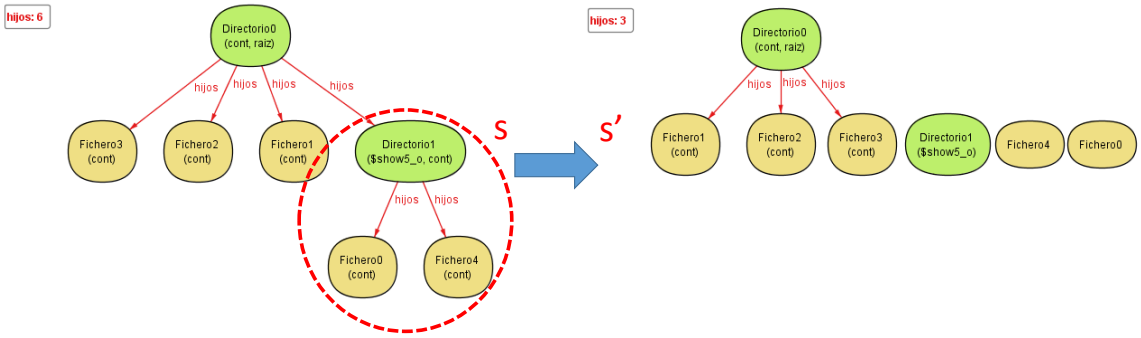


Figura 4: Ejemplo de la operación borrar

- El objeto `o` está en el sistema de ficheros `s`, pero no es su objeto `raiz`
- El objeto `padre` está en el sistema de ficheros `s`

La Figura 5 muestra el efecto que debe tener el predicado `borrar` sobre el sistema de ficheros `s` (el árbol de la izquierda)

# Cuarta fase

1. Define una signatura `Tiempo` que representa distintos instantes de tiempo
2. Modifica las relaciones `fcontenido`, `estado`, `fbuffer` y `bcontenido` para que puedan cambiar en distintos instantes de tiempo
3. Reescribe las restricciones del sistema teniendo en cuenta los cambios realizados sobre las relaciones en el punto anterior.
4. Define los siguientes predicados:

   *a*) `pred abrirFichero(f:Fichero,t,t':Tiempo){}`

   que hace que el fichero `f` esté `Abierto` en el instante `t'`. El fichero debe estar `Cerrado` en el instante `t`, y ninguna de las demás relaciones debe cambiar del instante `t` al `t'`.

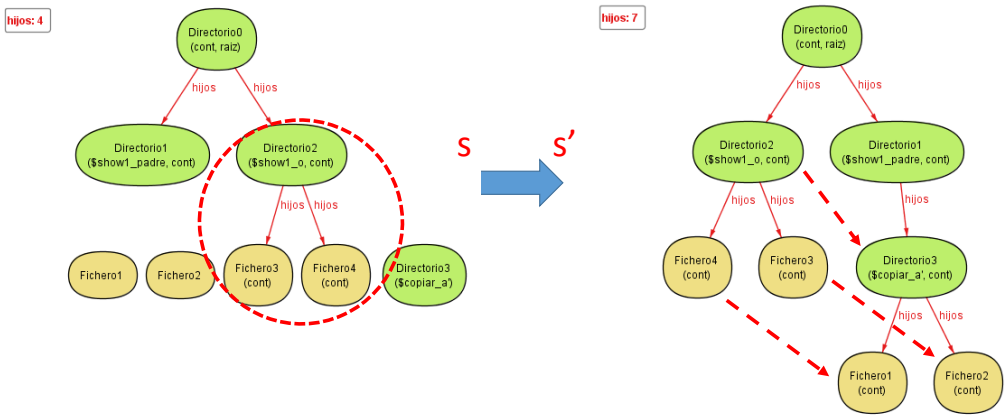


Figura 5: Ejemplo de la operación copiar

*b*) `pred cerrarFichero(f:Fichero,t,t':Tiempo){}`

que hace que el fichero `f` esté `Cerrado` en el instante `t'`. El fichero debe estar `Abierto` en el instante `t`, y ninguna de las demás relaciones debe cambiar del instante `t` al `t'`.

*c*) `pred leerFichero(f:Fichero,t,t':Tiempo){}`

que copia el contenido de `f` en su buffer del instante `t` al instante `t'`. El fichero `f` debe estar abierto en el instante `t`, y el contenido del buffer associado a `f` en `t'` debe ser `igual` al contenido de `f` en `t`. El resto de relaciones no debe cambiar de `t` a `t'`.

*d*) `pred escribirFichero(f:Fichero,b:Buffer,t,t':Tiempo){}`

que escribe el contenido del buffer `b` sobre el fichero `f`. El fichero `f` debe estar abierto y el buffer `b` debe ser del sistema de ficheros en el que se encuentra `f`. Despues de ejecutar el predicado, el contenido de `f` en `t'` debe ser `igual` al contenido de `b` en `t`. El buffer asociado `f` debe quedar vacio en el instante `t'`. El resto de relaciones no debe cambiar de `t` a `t'`.

5. Utilizando la sentencia `open util / ordering [ Tiempo ]` que ordena los instantes de tiempo, define el predicado

   ```
   pred traza(){}
   ```

   que itera sobre los instantes de tiempo y aplica de manera indeterminista los predicados `abrirFichero`, `cerrarFichero`, `leerFichero` y `escribirFichero`.

# Documentos a entregar

**D1.** Cuatro ficheros `.als` que se correspondan con cada una de las fases en las que se ha dividido la práctica. En cada fichero, debe estar claro qué hechos (`fact`), predicados, o funciones pertenecen a cada una de las fases. Cada restricción, predicado y función debe estar acompañada de un comentario que explique su funcionalidad.

**D2.** Ejemplos representativos de instancias para cada una de las cuatro fases de la práctica.

Todas las signaturas/relaciones definidas deben corresponder a algún concepto explicado en el enunciado. Si algún estudiante lo ve estrictamente necesario puede añadir más signaturas/relaciones, pero explicando claramente su necesidad y uso.

**Recuerda que esta es una práctica que debéis hacer en parejas. Los dos componentes deben participar IGUALMENTE en el desarrollo de la práctica, y además la documentación entregada DEBE ser el fruto del trabajo EXCLUSIVO de cada pareja. Cualquier anomalía que se detecte en este sentido se tendrá en cuenta en la evaluación de la práctica.**