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
Práctica 2:Alloy

En los siguientes ejercicios deben modelarse en Alloy distintos sistemas, utilizando la estructura:

```
module ejercicio

//signaturas y relaciones
sig ...
sig ...
//facts: restricciones globales sobre el sistema
fact{
// restricción 1
// restricción 2
....

}
\\predicados
pred ...

pred show () {
//... completa con restricciones
}
run show
```

Ejercicio 1

Define una signatura `Color`, una signatura `Objeto`, y una relación `color` que asocie a cada objeto un color. Define la signatura `Caja`, y una relación `contiene` que asocie cada caja con el conjunto de objetos que contiene.

- Expresa con sintaxis Alloy, de la forma más sencilla posible, la restricción"cada objeto está en una y sólo una caja".
- Muestra que el modelo permite que:
  1. en alguna caja todos los objetos sean del mismo color
  2. alguna caja tenga más de dos objetos, y todos los objetos que contienen son de distinto color

Ejercicio 2

Supón que un hotel tiene distintas habitaciones. Cada habitación tiene asociada una serie de llaves, (imagina que es una llave electrónica, junto con varias codificaciones). Cada habitación puede estar ocupada por un cliente, y en ese caso, el cliente tiene la llave con la se abre la habitación.

Modela este sistema en Alloy con las distintas signaturas y relaciones:

1. Define la signaturas `Llave` y `Cliente`, y la relación `llave` que asocia a cada cliente la llave de la habitación que ocupa, si es que ocupa alguna.
2. Define las signaturas `Hotel` y `Habitacion` y las relaciones:
   a) `hab`: asocia cada hotel con el conjunto de sus habitaciones;
   b) `llaves`: asocia cada habitación con el conjunto de sus llaves;
   c) `llaveActual`: asocia cada habitación con la llave que se utiliza para abrirla;
   d) `ocupacion`: asocia cada habitación con el cliente que la ocupa, en caso de que no esté vacía.

Expresa con sintaxis Alloy, de la forma más sencilla posible, las siguientes restricciones sobre el modelo diseñado:

1. Todas las habitaciones pertenecen a exactamente un hotel

2. Las llaves de las habitaciones son disjuntas (ningún par de habitaciones comparten llaves)
3. Todas las habitaciones tienen al menos una llave asociada
4. La llave actual de cada habitación es una de sus llaves
5. Cada cliente ocupa a lo sumo una habitación
6. Si un cliente ocupa una habitación, entonces su llave es la llave actual de la habitación
7. Si un cliente no ocupa una habitación, entonces no tiene ninguna llave asociada

Ejercicio 3

El objetivo de este ejercicio es modelar una serie de edificios con plantas, y ascensores, junto con un conjunto de personas que puede estar en alguno de los edificios. Para ello:

Define las signaturas `Planta`, `Ascensor`, `Edificio` y `Persona`, y las relaciones:

1. `ascPlanta` y `ascPersonas` que relacionan cada ascensor con la planta en la que se encuentra, y con el conjunto de personas que están en el mismo, respectivamente.
2. `edifPlantas` y `edifAscensores` que relacionan cada edificio con sus plantas (al menos debe haber una planta en cada edificio), y el conjunto de ascensores que tiene, respectivemente.
3. `edificio` y `ascensor` que relacionan, respectivamente, cada persona con el edificio y el ascensor en el que se encuentra. Una persona podría no estar en ningún edificio ni en ningún ascensor.

- Expresa con sintaxis Alloy, de la forma más sencilla posible, las `facts` sobre el modelo diseñado:
    1. Cada planta está exactamente en un edificio
    2. Cada ascensor está exactamente en un edificio
    3. El número de ascensores de un edificio es estrictamente menor que el número de plantas del edificio
    4. Si un edificio tiene menos de 3 plantas entonces no tiene ascensores
    5. Si un edificio tiene más de 3 plantas entonces tiene algún ascensor
    6. Cada ascensor se encuentra en una planta del edificio al que pertence
    7. Si una persona está en un ascensor, entonces debe encontrarse también en el edificio en el que está dicho ascensor
    8. La relación `ascPersonas` contiene exactamente a las personas que están en el ascensor
- Añade restricciones que muestren que:
    1. Es posible que una persona esté en un edificio, pero no esté en ningún ascensor
    2. Es posible que un ascensor lleve a dos personas
    3. Es posible que todos los ascensores de un edificio estén en la misma planta
    4. Es posible que todos los ascensores de un edificio estén en plantas diferentes

Ejercicio 4

Supón que quieres modelar un sistema que asigna celdas a internos en una prisión. La asignación debe evitar situar dos internos en la misma celda si son miembros de diferentes bandas. Considera el siguiente modelo Alloy:

```
module prision
sig Banda {miembros: set Interno}
sig Interno {celda: Celda}
sig Celda {}
pred asigSegura () {
//... completa con restricciones
}
pred show () {
//... completa con restricciones
}
run show
```

1. En primer lugar, analiza el modelo tal y como aparece en el enunciado (con los cuerpos de los predicados vacíos) y observa que las instancias que se generan permiten, por ejemplo, que:

   a. Un preso pertenezca a más de una banda

   b. Una banda no tenga ningún miembro

   c. Hay presos que no pertenecen a ninguna banda

   Añade restricciones (como `facts`) que aseguren que ninguna instancia del modelo satisfaga las condiciones *a* y *b* anteriores.

2. Completa el predicado `asigSegura` con restricciones que garanticen una asignación segura de internos a celdas. Genera ejemplos tanto de asignaciones seguras como de no seguras, ejecutando el predicado `show`, con llamadas apropiadas al predicado `asigSegura`.

3. Escribe un nuevo predicado llamado `asigFeliz`, que represente que los miembros de una banda sólo comparten celda con miembros de la misma banda. Una asignación *segura* no implica necesariamente una asignación *feliz*. Escribe un aserto y verifícalo (con `check`). Encuentra un contraejemplo, y explica por qué ha ocurrido.

4. Añade una restricción como un hecho (`fact`) que asegure que la condición de *seguridad* también implica la condición de *felicidad*. Reescribe el predicado `show` para comprobar que no has añadido ninguna inconsistencia, y chequea el aserto para comprobar que ahora no se genera ningún contraejemplo.

Ejercicio 5

En una canción, Doris Day dice:
Everybody loves my baby, but my baby don't love nobody but me
(Todo el mundo quiere a mi niño, pero mi niño no quiere a nadie salvo a mi)
David Gries observó que, desde un punto de vista estrictamente formal, esta afirmación equivale a decir que: *yo soy mi niño*. Compruéba que esto es cierto, formalizando la canción con restricciones, y la deducción de David Gries como un aserto. A continuación, modifica las restricciones para que expresen lo que seguramente Doris Day quería decir, y prueba que ahora la afirmación tiene un contraejemplo.
Observación:
Si `Persona` representa el conjunto de personas, utiliza la siguiente declaración

```
sig Persona
one sig Baby in Persona{
}

one sig Doris in Persona{

}
```

para definir dos instancias concretas de persona `Baby` y `Doris`

Ejercicio 6

Este es el famoso problema del *apretón de manos* del matemático Paul Halmos.

Alice y Bob invitaron a otras 4 parejas a su casa a cenar. Como suele ocurrir, algunos de los asistentes eran más amables que otros, así que algunos de los asistentes se saludaron dándose un apretón de manos, mientras que otros no. Como resultado de este comportamiento hubo un número, en principio, impredecible de apretones de manos, teniendo en cuenta las restricciones lógicas: nadie se saluda a sí mismo, ni saluda a su propia pareja. En un momento dado, una vez que los saludos habían terminado, Alice alzó la voz y preguntó, con curiosidad, al resto de los nueve asistentes (las cuatro parejas invitadas y Bob), a cuantas personas habían saludado cada uno. Recibió nueve respuestas diferentes entre los valores 0 y 8. Es decir, un invitado le dijo que no había saludado a nadie, otro que había saludado sólo a una persona, y así sucesivamente. La pregunta es, con esta información, ¿a cuántos invitados saludó Bob?

Resuelve el problema modelando el sistema en Alloy y usando el analizador para encontrar una solución. Prueba con distintos números de parejas invitadas, es decir, con 1 pareja, 2 parejas, etc.

Observación:

Si `Persona` representa el conjunto de personas, utiliza la siguiente declaración

```
sig Persona
one sig Alice in Persona{
}

one sig Bob in Persona{

}
```

para definir dos instancias concretas de persona `Alice` y `Bob`

Ejercicio 7

Una *máquina de estados/sistema de transiciones* tiene uno o más estados iniciales y una relación de transición que conecta cada estado con sus sucesores. Construye en Alloy un modelo de máquina de estados, añadiendo restricciones y usando el analizador para resolverlas, para construir los siguientes tipos de máquinas:

a. Un máquina *determinista*, en la que cada estado tiene a lo sumo un sucesor;

b. Una máquina *no determinista*, en la que algunos estados tienen más de un sucesor;

c. Una máquina con estados *no alcanzables*

d. Una máquina sin estados *no alcanzables*

e. Una máquina *conectada* en la que cada estado es alcanzable desde cualquier otro estado;

f. Una máquina con *bloqueo*, es decir, con un estado alcanzable sin sucesores.

Ejercicio 8

Define un modelo Alloy de *Asignaturas*. Cada asignatura tiene un grupo de estudiantes matriculados (como máximo 10), de los cuales algunos pueden estar ya aprobados. Una vez definidas las signaturas *Asignatura* y *Estudiante*, modela los siguientes predicados:

1. `inicio(a:Asignatura)`, que define el estado inicial de la asignatura `a`, sin estudiantes matriculados

2. `matricular(a,a':Asignatura,e:Estudiante)`, que modela el efecto de matricular un nuevo estudiante `e` en la asignatura `a`, donde `a'` es la asignatura resultante. Para matricular un estudiante es necesario que en `a` haya menos de 10 estudiantes matriculados y que el estudiante no esté ya matriculado.

3. `aprobar(a,a':Asignatura,e:Estudiante)`, que modela el cambio de estado de la asignatura `a` en la asignatura `a'`, cuando un estudiante `e` ha aprobado la asignatura.

4. `certAprobado(a,a':Asignatura,e:Estudiante)`, que modela el éxito del estudiante que, a final de curso, cuando ha aprobado la asignatura, se le concede un certificado. Una vez que el estudiante ha recibido el certificado, se le elimina del conjunto de matriculados y aprobados.

5. `nocertAprobado(a,a':Asignatura,e:Estudiante)`, que modela el fracaso del estudiante que, a final de curso, no ha aprobado la asignatura. Una vez que el curso ha terminado, se elimina al estudiante del conjunto de matriculados.

Finalmente, comprueba si las siguientes asertos son ciertos. En caso contrario, corrige el modelo para que lo sean:

1. Ningún estudiante puede aprobar dos veces la misma asignatura.

2. Un estudiante que no ha aprobado no puede obtener un certificado.

Ejercicio 9

El objetivo de este ejercicio es modelar el *problema de los filósofos* en Alloy para demostrar que el bloqueo es imposible si se imponen algunas restricciones. El problema de los filósofos es un problema de sincronización. Consiste en n filósofos que se sientan alrededor de una mesa redonda, con un tenedor entre cada par de filósofos. Todos los filósofos quieren comer, pero necesitan los dos tenedores adyacentes para hacerlo. El bloqueo ocurre cuando cada filósofo tiene un tenedor, y espera para coger el segundo.

```
module filosofos
sig Tenedor{}
sig Filosofo{
   izda:Tenedor,
   dcha:Tenedor
}
```

En este ejercicio, primero se modelan en Alloy las restricciones características del modelo. A continuación, se describen las condiciones para el bloqueo y una estrategia de asignación de tenedores que impide el bloqueo. Finalmente, se prueba que con esta estrategia el bloqueo no es posible.

1. Añade `facts` o predicados que especifiquen que hay sólo una mesa en la que están sentados los filósofos, con un tenedor entre cada par de filósofos contiguos. Genera algunas instancias con diferentes tamaños para comprobar la validez de tu código.

2. Supón que modelamos el estado del sistema como sigue:

   ```
   sig Estado{
      asignacion:Filosofo -> set Tenedores
   }
   ```

   Expresa los siguientes predicados:

   ```
   // Es cierto si el filósofo f puede comer en estado e,
   // es decir, si tiene los dos tenedores
   pred puedeComer(f:Filosofo,e:Estado){
   ...
   }

   // Es cierto, si en el estado e, el filósofo f puede coger un tenedor
   // es decir, si alguno de los dos tenedores adyacentes está libre
   pred puedeCogerTenedor(f:Filosofo,e:Estado){
   ...
   }
   ```

3. Expresa lo que significa que un estado sea *válido*, es decir, un estado en el que ningún tenedor es utilizado simultáneamente por más de un filósofo, y en que cada filósofo puede solo tener el tenedor de su derecha y/o el de su izquierda

```
pred esValido(e:Estado){
...
}
```

4. Expresa en Alloy lo que es un bloqueo

```
pred deadlock(e:Estado){
...
}
```

5. Encuentra una instancia del modelo que represente un estado bloqueado

6. Un criterio para evitar el bloqueo es tener un monitor que revise el estado de la mesa y compruebe que no todos los tenedores están siendo utilizados simultáneamente. Escribe el predicado correspondiente:

```
pred monitorOk(e:Estado){
...
}
```

7. Escribe un aserto que verifique si la solución *monitor* efectivamente impide el bloqueo

```
assert monitorValido(e:Estado){
...
}
check monitorValido for 10 but 1 Estado
```