Métodos Formales para Ingeniería de Software

Ma. Laura Cobo

Modelado Estático y Dinámico

Departamento de Ciencias e Ingeniería de la Computación
Universidad Nacional del Sur
Argentina

Alloy: modelado estático y dinámico

Modelado Estático

- Describe estados, no comportamientos
- Las propiedades son **invariantes**

Modelado Dinámico

- Describe transiciones entre estados
- Las propiedades son **operaciones**

Alloy: modelado estático y dinámico

Modelado Estático

- Describe estados, no comportamientos
- Las propiedades son **invariantes**

```
Una lista está ordenada
Cada libro tiene al menos un autor
```

Modelado Dinámico

- Describe transiciones entre estados
- Las propiedades son **operaciones**

```
Cómo funciona el algoritmo de ordenamiento
Cuál es el efecto de agregar un libro a la
biblioteca
```

Alloy: modelado estático

Modelos estáticos

- Define los valores permitidos como componentes de los estados
  - Valores para los conjuntos
  - Valores para las relaciones

- Una instancia del modelo es un conjunto que establece el estado de los valores de las componentes que:
  - Satisfacen las restricciones definidas por multiplicidad, hechos, condiciones, etc.

Alloy: modelado estático

Esta basado en la construcción del modelo:

- Clasificar los átomos:
  - Considerar qué es relevante
  - Determinar la estructura jerárquica
  - Determinar subconjuntos y clasificación ortogonal de los mismos
- Modelar relaciones
  - Agregar relaciones y restricciones sobre las relaciones existentes
  - Restricciones basadas en la naturaleza de las abstracciones (simetría, relaciones acíclicas, ordenadas, etc)

Alloy: modelado dinámico

Los modelos estáticos permiten describir estados legales para un sistema dinámico

Pero se espera que las transiciones legales entre estados puedan describirse. Por ejemplo

- Que una persona haya nacido antes de casarse
- Que exista el libro en la biblioteca antes de sacarlo en préstamo.

Alloy: modelado dinámico

Esto implica el modelado de la idea de **transición**

Así por ejemplo la relación prestado, no será igual todo el tiempo. Habrá un momento donde la relación está vacía, un momento donde la relación contendrá a un libro que ha sido prestado y así siguiendo.

Alloy: modelado dinámico

Alloy no cuenta con una noción de embebida de estado, ni de transición entre estados.

Sin embargo hay varias maneras de modelar los aspectos dinámicos de un sistema

¿Qué podemos esperar de las operaciones?

Dado que las signaturas son estáticas …

¿cómo hablamos de las operaciones? ¿de los efectos?

¿Qué podemos esperar de las operaciones?

Dado que los átomos son estáticos ….

¿cómo hablamos de las operaciones? ¿de los efectos?

Alloy no considera la noción de tiempo o de estado mutable.

Resulta necesario modelar estas nociones explícitamente

¿Qué podemos esperar de las operaciones?

Dado que los átomos son estáticos ….

¿Cómo expresamos una transición?

Puede modelarse a través de un predicado que establezca una relación entre dos estados

- El estado **anterior** a la transición y
- El estado **siguiente**

Requiere las restricciones necesarias: pre y post condiciones para cada transición, condiciones de marco “frame”

Solución: "nuevo" atómo

Una de la estrategias es tener un nuevo átomo que refleja el cambio realizado.

De esta manera capturamos la noción de **cambio**

Solución: "nuevo" atómo

Una de la estrategias es tener un nuevo átomo que refleja el cambio realizado.

De esta manera capturamos la noción de **cambio**

La desventaja de esta aproximación está en que ya no se trata del mismo "objeto" sino de otro que en general es igual salvo por el efecto esperado de la operación

Solución: "nuevo" átomo

Modelo

```
sig  Biblioteca { coleccion: set Libro}
sig  Libro { escritoPor: set Autor}
sig  Autor {}
```

Si agregamos un libro a nuestra biblioteca, el efecto es que hay una nueva biblioteca luego de cada mutación.

```
pred agregar[b,b1: Biblioteca, l:Libro] {
   b1.coleccion = b.coleccion + l
}
```

Solución: "nuevo" átomo

Modelo

```
sig  Biblioteca { coleccion: Libro -> Autor}
sig  Libro { escritoPor: set Autor}
sig  Autor {}
```

Si agregamos un libro a nuestra biblioteca, el efecto es que hay una nueva biblioteca luego de cada mutación.

```
pred agregar[b,b1:Biblioteca,l:Libro,a: Autor]
  {
     b1.coleccion = b.coleccion + l -> a
  }
```

¿Qué podemos esperar de las operaciones?

Dado que los átomos son estáticos ….

¿Cómo expresamos una transición?

Puede modelarse a través de un predicado que establezca una relación entre dos estados

- El estado **anterior** a la transición y
- El estado **siguiente**

Requiere las restricciones necesarias: pre y post condiciones para cada transición, condiciones de marco “frame”

Solución: "nuevo" patrón

Trata a las acciones y operaciones en un estado global, que define el comportamiento de una máquina abstracta

```
sig State { … }

pred  init [s: State] { … }
// describe el estado inicial
pred  inv [s: State] { … }
// describe los invariantes que todo
estado debe verificar
pred  op1 [s, s1: State] { … }
…
…
pred  opN [s, s1: State] { … }
```

Máquina abstracta

Puede chequearse que las operaciones preservan invariantes

```
assert initVerifies { all s: State |
   init[s] => inv[s] }

// para cada operación

assert  opPreserves {
   all s,s1: State |
        inv[s] && op[s,s1] => inv[s1]
}
```

Ejemplo

Modelo

```
sig  Biblioteca { coleccion: set Libro}
sig  Libro { escritoPor: set Autor}
sig  Autor {}
```

Agregando la signatura que modela el estado

Modelo

```
sig  Biblioteca { states: set State}
sig  Libro { escritoPor: set Autor}
sig  Autor {}

sig  State { coleccion: set Libro}
```

Ejemplo

Modelo

```
sig  Biblioteca { states: set State}
sig  Libro { escritoPor: set Autor}
sig  Autor {}

sig  State { coleccion: set Libro}
```

Estado inicial

```
pred init [s: State]
  #s.coleccion=0
```

Solución: "nuevo" patrón

Utilizar un patrón de trazas:

- Modela secuencias de ejecuciones sobre la máquina abstracta
- Crea un ordenamiento total sobre los estados
- Conecta estados sucesivos a través de operaciones
  - ✓ Todos los estados deben ser alcanzables

```
open util/ordering[State] as ord
…
fact traces {
   init [ord/first]
   all s:State - ord/last |
    let s1 = s.next |
        op1[s,s1] or … or opN[s,s1]
}
```

Ejemplo

Modelo

```
sig  Biblioteca { states: set State}
sig  Libro { escritoPor: set Autor}
sig  Autor {}

sig  State { coleccion: set Libro}
```

Ejemplo de operación sobre los estados

```
pred agregarAColeccion [s,s1: State, l:Libro]
  s1.coleccion = s.coleccion + l
```

Chequeando propiedades "safety"

Este tipo de propiedades pueden chequearse con una aserción, dado que todos los estados son alcanzables.

```
…
pred safe [s:State] { … }

assert  allReachableSafe {
    all s:State | safe[s]
}
```

Chequeando propiedades "safety"

Este tipo de propiedades pueden chequearse con una aserción, dado que todos los estados son alcanzables.

```
…
pred safe [s:State] { … }

assert  allReachableSafe {
    all s:State | safe[s]
}
```

Se controlan propiedades que capturan los lemas:

- ***"lo que quiero que suceda en algún momento sucederá",***
- ***"lo que no quiero que suceda no sucederá"***

Máquina Abstracta: detalles

La máquina abstracta es una parte del modelo que **no puede ser modularizada**.
Esto se debe a la dependencia de comportamiento de la misma con respecto a la parte del modelo afectada

Máquina Abstracta: detalles

La máquina abstracta es una parte del modelo que **no puede ser modularizada**.
Esto se debe a la dependencia de comportamiento de la misma con respecto a la parte del modelo afectada

Modelo estático

```
sig Color { }
sig Light { color: Color }
```

Máquina Abstracta: detalles

La máquina abstracta es una parte del modelo que **no puede ser modularizada**.
Esto se debe a la dependencia de comportamiento de la misma con respecto a la parte del modelo afectada

Modelo estático

```
sig Color { }
sig Light { color: Color }
```

Modelo dinámico

```
sig Color { }
sig Light { }

sig State {color: Light -> one Color }
```

Solución "Nuevo" Patrón: opciones

El estado puede pensarse de manera global (primera columna) o local (última columna)

Estado global: signatura State

```
sig Color { }
sig Light { }
sig State {color: Light -> one Color }
```

Estado local: signatura Time

```
sig Time { }
sig Color { }

sig Light {color: Color one -> Time }
```