

Métodos Formales para Ingeniería de Software

Ma. Laura Cobo

Módulos en Alloy

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

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

Módulos en Alloy

- Alloy cuenta con un sistema de módulos que permiten la modularización y reuso de modelos.
- Un módulo define un modelo que puede ser incorporado como **submodelo** de otro
- Para facilitar el reuso, los módulos pueden ser **paramétricos** para una o más **signaturas**

Módulos en Alloy

Un ejemplo de módulo podría ser el siguiente:

```
module util/relation

-- r es una relación acíclica sobre el
conjunto S

pred acyclic[r: univ-> univ, S: set univ]
{all x:S | x !in x.^r}
```

Módulos en Alloy

```
module util/relation
```

```
-- r es una relación acíclica sobre el conjunto S
```

```
pred acyclic[r: univ-> univ, S: set univ] {all x:S |  
    x !in x.^r}
```

O para asegurarse que entre las subcarpetas no este la carpeta que las contiene

```
module fileSystem
```

```
open util/relation as rel
```

```
sig Object {}
```

```
sig Folder extends Object{ subFolders: set  
Folder}
```

```
fact {acyclic[subFolders, Folder]}
```

Declaración de módulos

```
module moduleName
```

Es el encabezado o header del módulo

Declaración de módulos

```
module moduleName
```

Es el encabezado o header del módulo

Un módulo puede **importar** otro módulo a través de la sentencia open

```
open moduleName
```

La sentencia open puede interpretarse como inclusión textual

Declaración de módulos

```
module moduleName
```

Es el encabezado o header del módulo

Un módulo puede **importar** otro módulo a través de la sentencia `open`

```
open moduleName
```

Un módulo puede importar a otro que a su vez importa un tercer módulo, y así siguiendo

No se permiten ciclos en la estructura de importación

Definición

Todo módulo tiene un path que debe hacer match con el archivo correspondiente en el file system del sistema operativo

El path puede:

- Ser sólo el nombre de un archivo (sin la extensión als)
- Ser el path completo desde la raíz

La raíz del path en el encabezado del módulo a importar es la raíz para cada importación

Definición

```
module C/F/mod  
  open D/lib1  
  open C/E/H/lib2  
  open C/E/G/lib3
```

El *path name* del módulo en el encabezamiento especifica el directorio raíz desde el que se importa cada archivo

¿ de acuerdo a lo especificado cómo podría ser la estructura de directorios que contiene los módulos importados?

Definición

Ejemplo

```
module biblioteca  
  open lib/libro
```

Si el path de *biblioteca.als* es *<p>* en el file system, entonces el analizador de Alloy buscará *libro.als* en *<p>/lib/*

Definición

Ejemplo

```
module biblioteca  
  open lib/libro
```

Si el path de *biblioteca.als* es *<p>* en el file system, entonces el analizador de Alloy buscará *libro.als* en *<p>/lib/*

Alloy cuenta con una librería de módulos predefinidos ¿qué acciones toma el analizador en caso de querer importar estos módulos?

Definición

Ejemplo

```
module biblioteca  
  open lib/libro
```

Si el path de *biblioteca.als* es *<p>* en el file system, entonces el analizador de Alloy buscará *libro.als* en *<p>/lib/*

Alloy cuenta con una librería de módulos predefinidos ¿qué acciones toma el analizador en caso de querer importar estos módulos?

*Cualquier módulo importado será **buscado primero** entre los predefinidos.*

Si esta búsqueda falla se aplica la búsqueda descripta previamente

Renombrado

El módulo, **debe** contar con un nombre corto cuando:

1. el path de importación incluye / (es decir: es un path, no sólo un nombre) y
2. se importa más de un módulo con algún predicado/función de igual parte pública

Esto se logra con la keyword **as**

```
open util/relation as rel
```


Colisión de nombres

Los módulos definen sus propios **espacios de nombres**

La colisión de nombres se evita utilizando **nombres calificados**.

```
module fileSystem
open util/relation as rel
sig Object {}
sig Folder extends Object{ subFolders: set
Folder}

fact {rel/acyclic[subFolders,Folder]}
```

Módulos Parametrizados

- Un modelo m puede **parametrizarse** mediante uno o más parámetros signatura, $[x_1, \dots, x_n]$
- Cada importación debe instanciar cada parámetro con un nombre de signatura.
- El efecto de abrir $m[S_1, \dots, S_n]$ es que se obtiene una copia de m con cada parámetro signatura x_i reemplazado por la signatura S_i

Módulos Parametrizados: Ejemplo

```
module graph [Node] // 1 signature param
  open  util/relation as rel
  pred dag [r: Node -> Node ] {
    acyclic [r, Node ]
  }
```

```
module fileSystem
  open util/graph [Object] as g
  sig Object { }
  sig Folder extends Object {
    subFolders: set Folder
  }
  fact {g/dag [subFolders]}
```

El módulo predefinido “Ordering”

- Crea un ordenamiento lineal simple sobre los átomos en la signatura **S**

```
module util/ordering[S]
```

- También restringe a todos los átomos, permitidos por el scope, a existir.
 - Por ejemplo si el scope de la signatura **S** es 5 abrir `ordering[S]` forzará a **S** a tener 5 elementos, creando un orden lineal sobre esos 5 elementos

El módulo predefinido “Ordering”

```
module util/ordering[S]
  private one sig Ord {
    First, Last: S,
    Next, Prev: S -> lone S
  }
  fact {
    // todos los elementos de S están
    // totalmente ordenados
    S in Ord.First.*(Ord.Next)
  }
  ...
}
```

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

El módulo predefinido “Ordering”

```
// las restricciones definen un orden total
Ord.Prev = ~(Ord.Next)
one Ord.First
one Ord.Last
no Ord.First.Prev
no Ord.Last.Next
```

Cuenta con todas las restricciones necesarias, como las funciones (primer elemento, ultimo elemento, siguiente, anterior, siguientes, anteriores) y predicados (está antes, está después ...) necesarios