Métodos Formales para Ingeniería de Software

Introducción a Alloy

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

Alloy

¿ qué es Alloy ?

- Es un lenguaje formal de modelado
- Tiene sintaxis y semántica formal definida
- Las especificaciones se escriben en modo texto (existen representaciones visuales similares a los diagramas de clase UML o los diagramas entidad relación)
- Cuenta con una herramienta de verificación llamada ***Alloy Analyzer***. Este analizador puede utilizarse para analizar de manera automática propiedades sobre modelos Alloy.

Alloy

¿ Para qué fue creado ?

- **Liviano**: es pequeño y fácil de utilizar. Capaz de expresar propiedades comunes de manera natural
- **Preciso**: cuenta con una semántica matemática simple y uniforme
- **Tratable**: análisis semántico susceptible a ser eficiente y completamente automatizado (considerando limitaciones de alcance o *scope*)

Alloy: relacionado con …

UML

- tiene similitudes (notación grafica, restricciones lógicas) pero UML no es ni liviano, ni preciso.
- UML cuenta con muchas nociones de modelado que se encuentran omitidas en Alloy (diagramas de estado, caso de uso, etc.)
- Los diagramas y forma de navegación de Alloy están inspirados en UML.

Z

- Z es preciso pero intratable. Su tipografía estilizada hace que sea un lenguaje difícil para trabajar
- Z es más expresivo que Alloy pero más complicado
- La semántica basada en conjuntos de Alloy está inspirada en Z.

Alloy: Lenguaje

Objetivo

Especificación formal de modelos de datos orientados a objetos.

Observaciones:

- Puede utilizarse para modelados de datos en general:
  - ✓ Cualquier dominio de individuos y
  - ✓ Relaciones entre ellos
- Resulta particularmente útil para especificar clases, asociaciones entre ellas y restricciones sobre las asociaciones

Alloy: Analizador

Objetivo

Analiza las especificaciones Alloy

- Utiliza verificación limitada:
  - Limita el número de objetos de cada clase a un número fijo.
  - Realiza el chequeo para la cantidad de objetos establecida.

- Utiliza SAT-solvers para responder consultas de verificación:
  - Convierte las consultas en fórmulas de lógica booloeana e invoca al SAT-solver para responderlas

Alloy: lenguaje y analizador

EL lenguaje y analizador fueron desarrollados por Daniel Jackson y su grupo de trabajo en el MIT.

Referencias:

- "Alloy: A Lightweight Object Modeling Notation" Daniel Jackson, ACM Transactions on Software Engineering and Methodology (TOSEM), Volume 11, Issue 2 (April 2002), pp. 256-290.
- "Software Abstractions: Logic, Language and Analysis" by Daniel Jackson. MIT Press, 2012 (o la version original de 2006)

La documentación y versión actual de Alloy está disponible en:

- http://alloy.mit.edu/

Alloy: Semántica

Si se parte de una diagrama de clase:

- Cada clase corresponde a un conjunto de objetos (átomos)
- Cada objeto es una entidad abstracta, atómica e inmodificable
- El estado del modelo está determinado por:
  - ✓ Las relaciones entre átomos
  - ✓ La pertenencia de los átomos a conjuntos
  - ✓ El cambio los átomos y relaciones con el tiempo

El Lenguaje

La herramienta requiere de un lenguaje:

- para escribir el modelo y
- para expresar las propiedades

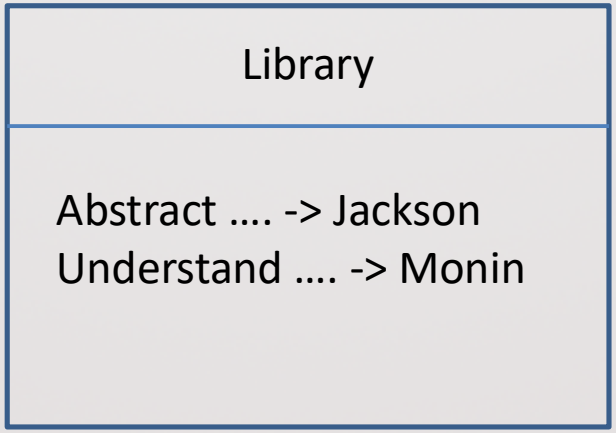
El Lenguaje

La herramienta requiere de un lenguaje:

- para escribir el modelo y
- para expresar las propiedades

Ejemplo: Biblioteca

Una biblioteca mantiene un mapeo de títulos de libros a autores



Alloy: Átomos y Relaciones

En Alloy todo se construye a partir de **átomos** y **relaciones**

- El **átomo** es la entidad primitiva. Es una entidad:
  - ✓ Abstracta y Atómica: no puede dividirse en partes más pequeñas
  - ✓ Inmodificable: sus propiedades no cambian con el tiempo
  - ✓ Ininterpretable: no tiene propiedades propias

- Una **relación** es una estructura que relaciona átomos.
  - ✓ Define un conjunto de tuplas
  - ✓ Cada tupla es una secuencia de átomos

Ejemplo: Biblioteca

Una biblioteca mantiene un mapeo de títulos de libros a autores

- **Relaciones unarias:** el conjunto de bibliotecas, el conjunto de títulos y el conjunto de autores
  Biblioteca = { (B0) }
  Libro = { (L0), (L1) ,(L2)}
  Autor = { (A0), (A1) }

- Una **relacion binaria** de Libro a Autor
  escritoPor = { (L0,A0), (L1,A1),(L2,A0) }

- Una **relacion ternaria** de Biblioteca a Libro a Autor
  coleccion = { (B0,L0,A0), (B0,L1,A1) }

Relaciones: conceptos

- **Tamaño de una relación:** el número de tuplas en la relación (equivalente a la cardinalidad de conjuntos, #)
- **Aridad de una relación:** el número de átomos en cada tupla de la relación

Ejemplos:

Biblioteca = { (B0) } es una relación de aridad 1 (unaria) y tamaño 1

coleccion = { (B0,L0,A0), (B0,L1,A1) } es una relación de aridad 3 (ternaria) y tamaño 2

Alloy: especificaciones

- **Signaturas** (signatures)
  - Describen las entidades con las que se quiere razonar
  - Definen conjuntos fijos
- **Campos** (fields)
  - Definen relaciones entre signaturas.
  - Es la única manera de incorporar relaciones en Alloy
- **Hechos** (facts)
- **Aserciones** (assertions)
- **Comandos y alcances** (commands – scope)
- **Predicados y funciones** (predicates – functions)

Resticciones o constraints simples (multiplicidad sobre signaturas y/o relaciones)

Signaturas

Una signatura define un conjunto de átomos (que representa un conjunto base)

La declaración

sig A { }

define un conjunto de nombre A.

- Un conjunto puede introducirse como una extensión de otro, de la siguiente manera:

sig A1 extends A { }

de esta manera el conjunto A1 es un subconjunto de A, aunque con ciertas restricciones

Ejemplo: Biblioteca

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

sig Novelista, Poeta, Periodista extends Autor { }
```

Observación: Los nombres de color naranja representan campos o fields, mientras que los de color azul representan signaturas

Ejemplo: Biblioteca

Modelo

```
sig  Biblioteca { coleccion: Libro -> Autor}
sig  Libro { escritoPor: set Autor}
sig  Autor {}
sig Novelista, Poeta, Periodista extends Autor {}
```

Algunas instancias del modelo:

Biblioteca = { ?? }
Libro = { ?? }
Autor = { ?? }
Novelista = { ?? }
Poeta = { ?? }
Periodista = { ?? }
coleccion = { ?? }
escritoPor = { ?? }

Ejemplo: Biblioteca

Modelo

```
sig  Biblioteca { coleccion: Libro -> Autor}
sig  Libro { escritoPor: set Autor}
sig  Autor {}
sig Novelista, Poeta, Periodista extends Autor {}
```

Algunas instancias del modelo:

Biblioteca = { (B0) }
Libro = { (L0),(L1)}
Autor = {(A0),(A1)}
Novelista = { }
Poeta = { }
Periodista = { }
coleccion = { }
escritoPor = { }

Ejemplo: Biblioteca

Modelo

```
sig Biblioteca { coleccion: Libro -> Autor}
sig Libro { escritoPor: set Autor}
sig Autor {}
sig Novelista, Poeta, Periodista extends Autor {}
```

Algunas instancias del modelo:

Biblioteca = { (B0) }
Libro = { (L0),(L1)}
Autor = {(A0),(A1)}
Novelista = { }
Poeta = { }
Periodista = { }
coleccion = { }
escritoPor = { }

Biblioteca = { (B0)}
Libro = {(L0),(L1), (L2)}
Autor = {(A0),(A1),(A2),(A3)}
Novelista = {(A0)}
Poeta = {(A1),(A2)}
Periodista = {(A3)}
coleccion = {(B0,L2,A3), (B0,L2,A0)}
escritoPor = {(L1,A2)}

Signaturas

Las signaturas declaradas de manera independiente son top-level signatures

IMPORTANTE:

- Las signaturas top-level son **disjuntas**
- Las diferentes extensiones de una signatura son **disjuntas**

Es posible definir signaturas **abstractas**

```
abstract sig A {}
```

Es posible definir signaturas relacionadas por el concepto de **subconjunto**

```
sig B1 in B {}
```

Signaturas

¿cómo se vería un dibujo de conjuntos con las siguientes declaraciones?

```
abstract sig A { }
sig B { }
sig A1 extends A { }
sig A2 extends A { }
sig A3 in A { }
```

Signaturas

Signaturas **abstractas**: no poseen elementos propios, ajenos a las signaturas que las extienden, a menos que no tengan signaturas que las extiendan

```
abstract sig A {}
```

```
abstract sig A { }
abstract sig B { }
sig A1 extends A { }
sig A2 extends A { }
sig A3 in A { }
sig B1 in B { }
```

B puede tener átomos propios (no pertenecientes a B1)

Ejemplo: Biblioteca

~~**sig** Biblioteca { coleccion: Libro -> Autor}~~
~~**sig** Libro { escritoPor: set Autor}~~
abstract sig Autor { }

sig Novelista, Poeta **extends** Autor { }
sig Periodista **in** Autor { }

¿representación de conjuntos? ¿representación gráfica?

Campos o fields

Las relaciones son declaradas como campos o fields de signaturas

Escribir

sig A { f: e}

introduce al modelo una relación f cuyo dominio es o está incluído en A y cuyo rango está dado por la expresión e.

Ejemplos:

sig A { f1 : A }

f1 es un subconjunto de A x A

sig B { f2: A -> A}

f2 es un subconjunto de ???

¿cuál es la aridad de f1? ¿y de f2?

Multiplicidades

El uso de la multiplicidad permite definir restricciones en cuanto al tamaño de los conjuntos (signaturas o relaciones).

Se escribe una keyword antes de la signatura, para restringir la cantidad de átomos que define esa signatura

m sig A {}

También pueden establecerse restricciones sobre los campos (fields)

sig A { f: **m** e}

sig A { f: e1 **m** -> n e}

Hay cuatro keywords de multiplicidad en Alloy:

- set : cualquier número
- some: uno o más
- lone : cero o uno
- one : exactamente uno

Multiplicidad: ejemplo

Un sistema de archivos donde cada directorio contiene cualquier número de objetos, y cada alias hace referencia a un único objeto.

abstract sig Object { }

sig Directory **extends** Object { }

sig File **extends** Object { }

sig Alias **in** File { }

Multiplicidad: ejemplo

Un sistema de archivos donde cada directorio contiene cualquier número de objetos, y cada alias hace referencia a un único objeto.

abstract sig Object { }

sig Directory **extends** Object { contents: **set** Objects }

sig File **extends** Object { }

sig Alias **in** File { to: **one** Object }

Multiplicidad: ejemplo

Un sistema de archivos donde cada directorio contiene cualquier número de objetos, y cada alias hace referencia a un único objeto.

```
abstract sig Object { }

sig Directory extends Object { contents: set Objects }

sig File extends Object { }

sig Alias in File { to: one Object }
```

Observación: el keyword de multiplicidad por defecto (para relaciones binarias) es **one**, de manera que puede omitirse por resultar redundante.

sig A {f: e} es equivalente a sig A {f: **one** e}

Multiplicidades en relaciones binarias

Cada átomo s de S, es mapeado por f a lo sumo a **un valor** en T

sig S { f: **lone** T}

La relación f define una función.
La función es parcial

Cada átomo s de S, es mapeado por f a exactamente **un valor** en T

sig S { f: **one** T}

La relación f define una función.
En este caso es una función total

Multiplicidades en relaciones ternarias

Para cada átomo s de S, el campo f satisface: para cada átomo t de T, f mapea t a exactamente un átomo de U

sig S { f: T -> **one** U}

Para cada átomo s de S, el campo f satisface: para cada átomo u de U, a lo sumo un átomo de T es mapeado a u por f

sig S { f: T **lone** -> U}

Muchas estructuras relacionales pueden construirse utilizando multiplicidad.

Por ejemplo:

sig S { f: **some** T} (relación total)

sig S { f: **set** T} (relación parcial)

sig S { f: T **set** -> **set** U}

sig S { f: T **one** -> U}

……

Buscando instancias del modelo

El comando **run** se utiliza para indicarle al analizador de Alloy, que genere una instancia del modelo

El comando puede incluir condiciones:

- se lo puede guiar para elegir instancias con ciertas características (por ejemplo forzar a que ciertos conjuntos y relaciones sean no vacíos)
- las condiciones que se agregan al comando no son parte de la especificación “real”.

El analizador de Alloy ejecuta el último comando ejecutado o en su defecto sólo el primero que encuentre en el archivo

Debilidades de los modelos construidos

El modelo es irrestringido – sin restricciones

Hay conocimiento de nuestro dominio que no es considerado.

Es posible agregar **restricciones** o constraints para enriquecer el modelo

En las primeras etapas del proceso de desarrollo es común contar con modelos sin restricciones.

Alloy provee un feedback rápido sobre las debilidades del modelo

Se puede ir agregando restricciones hasta que estemos sastisfechos con él.

Hechos

Hechos (facts): son restricciones adicionales sobre signaturas y campos

El analizador de Alloy busca instancias del modelo que satisfagan, también, las restricciones definidas como hechos

Ejemplo:

"Los novelistas y poetas no escriben libros en común"

Hechos

Hechos (facts): son restricciones adicionales sobre signaturas y campos

El analizador de Alloy busca instancias del modelo que satisfagan, también, las restricciones definidas como hechos

Ejemplo:

"Los novelistas y poetas no escriben libros en común"

?????

Buscando instancias del modelo

El comando **run** se utiliza para indicarle al analizador de Alloy, que genere una instancia del modelo

El comando puede incluir condiciones:

- se lo puede guiar para elegir instancias con ciertas características (por ejemplo forzar a que ciertos conjuntos y relaciones sean no vacíos)
- las condiciones que se agregan al comando no son parte de la especificación “real”.

El analizador de Alloy ejecuta solo el primer run que encuentre en el archivo

Buscando instancias del modelo

Al comando **run** se le puede indicar el alcance o **scope** de búsqueda

El scope limita el tamaño de las instancias a considerar

Esta limitación hace tratable la búsqueda de instancias.

Se indica el máximo número de átomos para cada signatura
(el valor por defecto es **3**)

Ejemplo:

```
run { }                          // comando simple, scope 3
run { #Poeta > 1} for 4          // scope 4, instancias con al menos 2
                                    poetas

run { some Poeta}
```

Buscando instancias del modelo

El algoritmo que implementa el analizador prefiere instancias pequeñas

- Suele producir instancias vacías o triviales
- Igualmente es útil saber que estas instancias satisfacen las restricciones impuestas (ya sea porque se espera que así sea o no)

Es usual que las instancias mostradas no muestren todos los comportamientos interesantes que son posibles.

Aserciones

Suele ocurrir que el usuario cree que el modelo cumple ciertas restricciones que no son expresadas de manera directa.

Las aserciones expresan estas restricciones adicionales que el analizador chequea si se verifican

Si la aserción no se verifica entonces el analizador produce una instancia **contraejemplo** (que ilustra por qué la propiedad falla)

Si la propiedad expresada como aserción falla, puede ser exprese algo que debiera ser un invariante o bien es necesario refinar la especificación o aserción.

Aserciones

Las aserciones se chequean a través del comando check (el cual debe cubrir el alcance para todas las signaturas)

Ejemplo: Sea A una aserción

Modelo

```
abstract sig  Object { }
sig Directorio extends Object { }
sig Archivo extends Object { }
sig Alias extends Archivo { }
```

check A **for** 5 Object
check A **for** 3 Directorio, 2 Archivo
check A **for** 5 Object, 3 Directorio
check A **for** 5 Directorio, 2 Alias, 5 Archivos

check A **for** 3 Directorio, 3 Alias

Aserciones

Las aserciones se chequean a través del comando check (el cual debe curbir el alcance para todas las signaturas)

Ejemplo: Sea A una aserción

Modelo

```
abstract sig  Object { }
sig Directorio extends Object { }
sig Archivo extends Object { }
sig Alias extends Archivo { }
```

check A **for** 5 Object
check A **for** 3 Directorio, 2 Archivo
check A **for** 5 Object, 3 Directorio
check A **for** 5 Directorio, 2 Alias, 5 Archivos

✖ **check** A **for** 3 Directorio, 3 Alias

Como definir restricciones

Analizamos que escribir en Alloy, pero no la sintaxis y semántica del lenguaje.

Veamos ahora que podemos usar para escribir las restricciones

Agregando restricciones

Con los elementos que contamos hasta ahora no podemos definir restricciones tales como:

- determinar si una relación es simétrica
- asegurar condiciones con respecto a la naturaleza de la relación.

Para poder comenzar a definir restricciones más interesantes, se agregan tres constantes (que definen tres conjuntos predefinidos):

- none : conjunto vacío
- univ : conjunto universal (formado por todos los átomos de una instancia del modelo)
- iden : conjunto identidad (relación de todos los átomos consigo mismos para una dada instancia del modelo)

Cuantificadores

Paralelamente se incluye una rica colección de cuantificadores

- all x: S | F — F se verifica para cada x en S
- some x:S | F — F se verifica para algún x en S
- no x:S | F — F falla para cada x en S
- lone x:S | F — F se verifica para a los sumo un x en S
- one x:S | F — F se verifica para exactamente un x en S

Los cuantificadores **también** pueden aplicarse a expresiones que denotan relaciones

- some e — e tiene tuplas, es no vacío
- no e — e no tiene tuplas, es es vacío
- lone e — e tiene a los sumo una tupla
- one e — e tiene exactamente una tupla

Todo es un conjunto en Alloy

No hay escalares

No se puede hablar directamente de elementos o tuplas de relaciones.

Resulta necesario usar relaciones “singleton” como

let nombre = **one** Libro

La cuantificación

all x:S | …. x ….

determina que x = {t} para cada átomo t de la signatura S

let nombre = **one** Libro

La sintaxis la veremos un poquito mas adelante en la clase

Operadores para conjuntos

Operaciones:

- **+** : unión
- **&** : intersección
- **-** : diferencia
- **in** : subconjunto
- **=** : igualdad

Modelo

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

sig Novelista, Poeta extends Autor { }
sig Periodista in Autor { }
```

Ejemplo:
Todos los autores periodistas del libro identificado como book

```
{ let book = one Libro {
        book.escritoPor & Periodista } }
```

Operadores relacionales

Operaciones:

- **->** : producto
- **~** : traspuesta
- **.** : join o dot join
- **[ ]** : box join
- **^** : clausura transitiva
- ***** : clausura reflexivo-transitiva
- **<:** : restricción de dominio
- **:>** : restricción de rango
- **++** : override

Producto

- **p -> q**
  - p y q son relaciones
  - p -> q es una nueva relación, esta relación está formada por las tuplas que se forman concatenando cada elemento de p con cada elemento de q

Ejemplo:

Biblioteca = { (B0)}
Libro = { (L0), (L1)}
Autor = { (A0), (A1) }

Libro -> Autor = { (L0,A0), (L0,A1), (L1,A0), (L1,A1) }
Biblioteca -> Libro -> Autor =
{ (B0,L0,A0), (B0,L0,A1), (B0,L1,A0), (B0,L1,A1) }

Traspuesta

- **~ p**
  - Construye la imagen espejada a la definida por la relación p, es decir contiene tuplas con los átomos invertidos.

Ejemplo:

ejemplo = { (a0,a1,a2,a3), (b0,b1,b2,b3) }
~ejemplo = { (a3,a2,a1,a0), (b3,b2,b1,b0) }

Join: **p.q**

- Sea p una relación de aridad m y q una relación de aridad n, el join es una relación de aridad m+n-2, tal que las tuplas que contiene surgen de combinar los primeros m-1 elementos de cada tupla de p y los últimos n-1 elementos de cada tupla de q, siempre y cuando el join entre las tuplas exista.
- Para cada par de tuplas $(s_1,....,s_m)$ en p y $(t_1,....,t_n)$ en q
  - Si s_m != t_1 entonces no se agrega nada al p.q
  - Si $s_m = t_1$ entonces $(s_1,...,s_{m-1},t_2,...,t_n)$ esta en la relación p.q
- El join **no está definido** en caso que ambas relaciones sean de aridad uno

Ejemplo:

{ (a,b) } . { (a,c) } = { }
{ (a,b) } . { (b,c) } = { (a,c) }
{ (x,b), (y,b), (z,a), (z,c) } . { (b,d), (c,e) } = { (x,d), (y,d), (z,e) }

Box Join

- **p[q]**
  - Semánticamente es idéntico al join anterior, pero toma los argumentos en orden diferente

p [q] ≡ q . p

Ejemplo:

```
let mateo = one Autor {   // los libros que el autor identificado
 mateo[escritoPor]  }     // como mateo escribió

let mateo = one Autor {
    escritoPor.mateo }    // alternativa equivalente
```

Box Join

- **p[q]**
  - Semánticamente es idéntico al join anterior, pero toma los argumentos en orden diferente

p [q] ≡ q . p ≡ ~p . ~q ≡ ~q[~p]

Ejemplo:

```
let mateo = one Autor {   // los libros que el autor identificado
mateo[escritoPor]  }      // como mateo escribió

let mateo = one Autor {
    escritoPor.mateo }    // alternativa equivalente

let mateo = one Autor {
    ~mateo.~escritoPor }  // otra alternativa

let mateo = one Autor {
    ~escritoPor[ ~mateo ] }   // otra alternativa
```

Clausura Transitiva

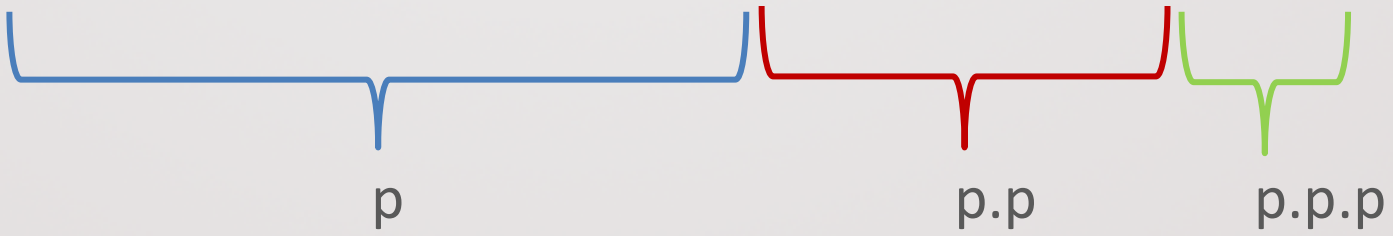
- **^ r**
  - Intuitivamente es la aplicación recursiva del operador join entre la relación y sí misma (hasta que no se agreguen tuplas)

^ r = r + r.r + r.r.r + …

Ejemplo:

p = { (s0,s1), (s1,s2), (s2,s3),(s4,s7) } p es una relación de S x S

^ p = { (s0,s1), (s1,s2), (s2,s3),(s4,s7), (s0,s2), (s1,s3), (s0,s3) }



Clausura Reflexivo-Transitiva

- *** r**
  - Intuitivamente es la clausura transitiva más el conjunto identidad

*** r ≡ ^ r + iden**

Ejemplo:

p = { (s0,s1), (s1,s2), (s2,s3),(s4,s7) } p es una relación de S x S

^ p = { (s0,s1), (s1,s2), (s2,s3),(s4,s7), (s0,s2), (s1,s3), (s0,s3) }

iden = { (s0,s0), (s1,s1), (s2,s2), (s3,s3), (s4,s4), (s7,s7) }

* p = { (s0,s1), (s1,s2), (s2,s3),(s4,s7), (s0,s2), (s1,s3), (s0,s3),
(s0,s0), (s1,s1), (s2,s2), (s3,s3), (s4,s4), (s7,s7) }

Restricciones de Dominio y Rango

- Estos operadores se utilizan para filtrar relaciones a un dominio o rango dados

- Si **s** es un conjunto y **r** una relación entonces
  - **s <: r** contiene las tuplas de **r** que **comienzan** con elementos que están en el conjunto **s**
  - **r :> s** contiene las tuplas de **r** que **terminan** con elementos que están en el conjunto **s**

Override

- **p ++ q**
  - p y q son dos relaciones de aridad dos o más
  - El resultado es similar a la unión de p y q, a excepción de que las tuplas de q pueden reemplazar a las tuplas de p. Cada tupla en p que hace match con una tupla de q (comienza con el mismo elemento) es descartada

$$p \mathbin{++} q \equiv p - (domain(q) <: p) + q$$

Ejemplo:

viejaDireccion = { (N0,DO), (N1,D1), (N1,D2) }
nuevaDireccion = { (N1,D4), (N3,D3) }
viejaDireccion + nuevaDireccion = {(N0,DO), (N1,D1), (N1,D2), (N1,D4), (N3,D3) }

Override

- **p ++ q**
  - p y q son dos relaciones de aridad dos o más
  - El resultado es similar a la unión de p y q, a excepción de que las tuplas de q pueden reemplazar a las tuplas de p. Cada tupla en p que hace match con una tupla de q (comienza con el mismo elemento) es descartada

$$p \mathbin{++} q \equiv p - (domain(q) <: p) + q$$

Ejemplo:

viejaDireccion = { (N0,DO), (N1,D1), (N1,D2) }
nuevaDireccion = { (N1,D4), (N3,D3) }
viejaDireccion ++ nuevaDireccion = {(N0,DO), ~~(N1,D1), (N1,D2)~~, (N1,D4), (N3,D3) }

Override

- **p ++ q**
  - p y q son dos relaciones de aridad dos o más
  - El resultado es similar a la unión de p y q, a excepción de que las tuplas de q pueden reemplazar a las tuplas de p. Cada tupla en p que hace match con una tupla de q (comienza con el mismo elemento) es descartada

$$p \mathbin{++} q \equiv p - (domain(q) <: p) + q$$

Ejemplo:

viejaDireccion = { (N0,DO), (N1,D1), (N1,D2) }
nuevaDireccion = { (N1,D4), (N3,D3) }

viejaDireccion ++ nuevaDireccion = { (N0,D0), (N1,D4), (N3,D3) }

Operadores lógicos

Los operadores usuales están disponibles

• **not**	**!**	negación
• **and**	**&&**	conjunción
• **or**	**||**	disyunción
• **implies**	**=>**	implicación
• **else**		alternativa
•	**< = >**	si solo si

Observar las reglas de precedencia de todos los operadores

Predicados - Funciones

Pueden utilizarse como “macros”

- Pueden nombrarse y utilizarse en diferentes contextos (hechos, aserciones, condiciones del comando run)
- Pueden ser parametrizadas
- Se utilizan para factorizar comportamiento común

Predicados

- Capturan restricciones que el diseñador no desea guardar como hechos
- **Restricciones** que se reusan en varios contextos

Funciones

- **Expresiones** que se reusan en diferentes contextos

Funciones

Es una **expresión** nombrada, con cero o más argumentos y provee un resultado

Ejemplo:

"Los libros que escribió un autor"

```
fun escribio [a: Autor]: set Libro {
                        {l: Libro | l in escritoPor.a }  }
```

Funciones

Es una **expresión** nombrada, con cero o más argumentos y provee un resultado

Ejemplo:

"Los libros que escribió un autor"

```
fun escribio [a: Autor]: set Libro {
                    {l: Libro | l in escritoPor.a }  }
```

"Los autores novelistas de un libro"

```
fun autoresNovelistas [l: Libro]: set Novelista {
                    l.escritoPor & Novelista }
```

Predicados

Es una **restricción** nombrada, con cero o más argumentos

Los predicados no se incluyen en el análisis a menos que sean referenciados en los esquemas que se analizan (run, aserción, hecho)

Ejemplo:

"¿ Es autor de un libro ?"

pred esAutorDe [a: Autor, l:Libro] { a **in** l.escritoPor }

Para profundizar el tema …

Material de lectura

- capítulos 3 y 4 de “Software Abstractions: Logic, Language, and Analysis” de Daniel Jackson, revised edition, MIT Press, 2012

Para profundizar el tema ….

Material de lectura

- capítulo 3 de “Software Abstractions: Logic, Language, and Analysis” de Daniel Jackson, revised edition, MIT Press, 2012