**Alloy ejemplos**

**sig A {}**

**some sig B {}**

**Escribiendo some adelante estoy exigiendo la existencia de algún átomo de la signatura B en el modelo.**

**abstract sig C {}**

**Todo átomo de una signatura abstracta va a tener que ser de signaturas que la extiendan, si no tiene signaturas que la extiendan esta restricción ya no vale.**

**abstract sig D {}**

**one sig E {}**

**Toda instancia del modelo tendrá 1 átomo de E**

**lone sig F {}**

**Toda instancia del modelo tendrá a lo sumo 1 átomo de F**

**sig G extends C {}**

**sig H extends C {}**

**2 signaturas que extienden a una misma signatura son disjuntas. En este caso ni G ni H tendrían átomos en común.**

**En cambio, cuando utilizamos la inclusión (in) no existe la restricción como en los extends.**

**sig l in D {}**

**sig J extends G {}**

**sig K extends G {}**

**sig L extends B {}**

**sig M in G {}**

**El comando run generara instancias.**

Podemos **generar** un **comando con restricciones** adicionales **agregando llaves:**

**run {}**

El **alcance/scope define** la **cantidad máxima de átomos** que **puede existir** en **cada asignatura** denuestro **modelo,** por **defecto** el **scope es** de **3** sinoespecificamosnada.

**Ejemplo:**

**Quiero generar una instancia sin restricciones adicionales con scope de 4**

**run { } for 4**

Paraejecutarloscomandos **Alloy toma el ultimo comando ejecutado o el primero de los comandos.**

Tenemos la visualización gráfica, la visualización en modo texto, la vista en tablas y la vista de árbol.

**La visualización en modo texto muestra todos los conjuntos y relaciones que tenemos definidos en el modelo.**

**Ejemplo:**

**this/A={}**

**this/B={B$0}**

**Aca tenemos la numeración de los átomos para la signatura B**

**this/L={}**

**this/C={J$0, J$1, K$0, K$1}**

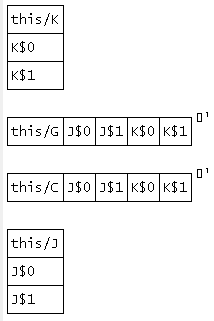
**this/G={J$0, J$1, K$0, K$1}**

**this/J={J$0, J$1}**

**this/K={K$0, K$1}**

**Por defecto al átomo lo nombra primero con la letra de la signatura luego el símbolo “$” y luego un número.**

**La visualización en tablas:**

****

**Comando run**

Todo son conjuntos y relaciones, las signaturas definen relaciones unarias, donde la relación es una tupla unaria que contiene al átomo.

Apretando el botón next vemos otras instancias generadas.

En el mismo modelo, probemos jugar con otras instancias:

**sig A {}**

**some sig B {}**

**abstract sig C {}**

**abstract sig D {}**

**one sig E {}**

**lone sig F {}**

**sig G extends C {}**

**sig H extends C {}**

**sig l in D {}**

**sig J extends G {}**

**sig K extends G {}**

**sig L extends B {}**

**sig M in G {}**

--------------

**run { }**

**run { } for 5**

**run noHayB {no B}**

Quiero generar una instancia del modelo en el que no haya átomos de la signatura B.

Si ejecutamos este comando el analizador va a decir que **no es posible encontrar tal instancia y que el predicado no es consistente con el modelo**, esto es **porque** con el **comando** run le **pedimos** que **genere** una **instancia donde no haya átomos de B y por otra parte en el modelo exigimos que, si o si exista algún átomo de B**, por lo tanto, no es posible generar la instancia.

A veces el scope no es suficiente para que se genere una instancia, asi que **cuando no anda una instancia podemos asegurarnos si es culpa del scope o no agregando** por ejemplo **for** 6 **a** **lo** **último** del comando.

**run noHayE {#E = 0}**

Quiero generar una instancia donde la cardinalidad del conjunto E sea 0 (es lo mismo que expresarlo como {no E}).

Si ejecutamos este comando el analizador va a decir que **no es posible encontrar tal instancia y que el predicado no es consistente con el modelo,** porque la definición del modelo dice que tiene que haber 1 átomo de E en cada instancia del modelo y el comando dice que no haya, por lo tanto, no es posible generar la instancia.

**run masDeUnF {#F > 1}**

Quiero generar una instancia donde haya más de un F.

Si ejecutamos este comando el analizador va a decir que **no es posible encontrar tal instancia y que el predicado no es consistente con el modelo,** porque la definición del modelo dice que a lo sumo tiene que haber un átomo de F y el comando dice que haya más de 1, por lo tanto, no es posible generar la instancia.

**run CnoGniH {some c:C | (c !in G) or (c !in H) } --OBS C es abstracta**

Quiero intentar generar una instancia del modelo en el cual haya al menos un átomo c de la signatura C talque no pertenezca a G o no pertenezca a H.

Esto es posible, en este caso el analizador muestra que hay un K que cumple con la condición, como tenemos un átomo de K que es de G (ya que K extiende a G) y este no pertenece a H, entonces es válido.

Esto se puede ver visualmente y también en el texto de la siguiente forma:

this/A={}

this/B={B$0, B$1, B$2}

this/L={}

this/C={K$0}

this/G={K$0}

this/J={}

this/K={K$0}

this/H={}

this/D={D$0}

this/E={E$0}

this/F={}

this/I={}

this/M={}

skolem $CnoGniH\_c={K$0}

Esto nos dice que hay un átomo de K que es K$0, hay un átomo de C que es K$0, hay un átomo de G que K$0 y que son el mismo átomo, y no está en H.

**run { some C - (G+H)}**

Queremos generar una instancia en la que haya algo, en el conjunto C – (G unión H).

Si ejecutamos este comando el analizador va a decir que **no es posible encontrar tal instancia y que el predicado no es consistente con el modelo,** porque C es abstracta y por definición todo átomo de C pertenece a quienes lo extienden y en este caso son G y H, por lo tanto, no es posible generar la instancia.

**run Dnol {some D-I }**

Queremos generar una instancia en la cual exista algún átomo de D que no sea átomo de I.

Es posible generar esta instancia ya que la relación de inclusión no tiene la limitación del extend, si lo vemos en el texto:

…

this/H={}

this/D={D$0}

this/I={}

**run BnoL {some B-L}**

Queremos generar una instancia en la que haya algún átomo de B que no pertenezca a L.

¡Si es posible porque B no es una signatura abstracta!

**run GyHsimultaneo {some (G&H)}**

Queremos generar una instancia en la cual haya un átomo que pertenezca a las signaturas de G y H en simultaneo (o sea en la intersección).

Como por definición son disjuntas ya que ambas extienden a C entonces no será posible generar esta instancia.

El analizador nos tirara un warning expresando que esta intersección es irrelevante ya que las 2 subexpresiones (G y H) son siempre disjuntas.

**run MyKsimultaneo {some (M&K)}**

Queremos generar una instancia en la cual haya un átomo que pertenezca a las signaturas de M Y K.

Es posible ya que K extiende a G y M está incluido en G así que no hay drama.

En el texto se vería así:

this/C={H$0, J$0, **K$0**}

this/G={J$0, **K$0**}

this/J={J$0}

this/K={**K$0**}

…

this/M={**K$0**}

**Convenciones**:

Los nombres de las signaturas comienzan en mayúsculas y los nombres de las relaciones en minúsculas.

**Agregando relaciones**

**sig A {r1: some B}**

**Todo átomo de A va a estar vinculado al menos con un átomo de B.**

r1 define una relación binaria entre átomos de A y de B

**some sig B {r2: H lone -> K}**

**Para las tuplas de los átomos de B, cada H va a estar vinculado con cualquier cantidad de K y a su vez que cada átomo de K venga asociado desde a lo sumo un H.  
La keyword lone afecta a los átomos que tenga más pegados en este caso a la H.**

r2 define una relación ternaria donde, la **1er** **componente** de la **tupla** será un **átomo** de **B**, la **2da** un **átomo** de **H** y **ultima** un **átomo** de **K**.

**Esto visualmente se ve como B conectado a través de H a K.**

Notarque **cuando queremos definir relaciones que son mas que binarias necesitamos** hacerusodel **producto (->)**

**abstract sig C {r3: J}**

**Todo átomo C estará relacionado con un J, esto es porque por defecto si no se escribe nada se toma como si hubiese un one delante de la J.**

**r3 define una relación binaria entre átomos de C y de J.**

**abstract sig D {r4: A some -> one G}**

**Para todo D, cada A esta vinculado con exactamente un G y cada G está vinculado con al menos un A**

**r2 define una relación ternaria donde, la 1er componente de la tupla será un átomo de D, la 2da un átomo de A y ultima un átomo de G.**

**one sig E {}**

**lone sig F {}**

**sig G extends C {}**

**sig H extends C {}**

**sig l in D {}**

**sig J extends G {}**

**sig K extends G {}**

**sig L extends B {}**

**sig M in G {}**

**Importante**

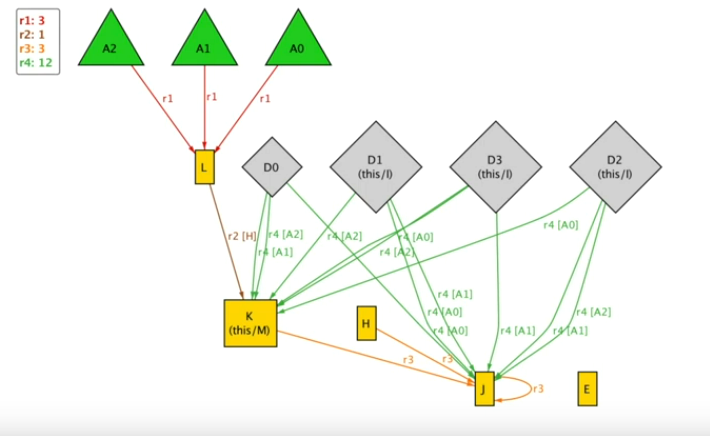
Cuando definimos **relaciones** que son **mas** **que** **binarias** por **defecto** tendremos la palabra reservada “**set**”.

Cuando definimos **relaciones** **binarias** por **defecto** tendremos la palabra reservada “**one**”.

**Evaluador**

Dentro de la ventana de la instancia, se puede encontrar la opción de “Evaluador”.  
El Evaluador nos permite ejecutar condiciones que queremos ver si se verifican para la instancia de tengamos generado.

Veremos ejemplos para la siguiente instancia visualizada gráficamente:



En el Evaluador escribimos expresiones booleanas u otro tipo de expresiones,

**Ejemplo 1**:

**A**

Acá **le estaríamos pidiendo al analizador que muestre los átomos del conjunto A**.

Resultado:



**Ejemplo 2:**

A + J //aca estaríamos consultado la unión de A y J, o sea los atomos de ambos conjuntos



**Ejemplo 3:**

r1 //aca le pedimos que nos muestre la relación r1



Lo **interesante** es que en el Evaluador **podemos hacer referencias explicitas a los átomos individuales** de esa instancia cosa que en el modelo no podemos porque la definición del modelo se aplica a todos los átomos de un conjunto.

Para esto **si** **queremos** **escoger** un **átomo**, **vamos** a la **visualización** en formato de **texto**, **agarramos** un **átomo** **de** un **conjunto** y lo escribimos en el Evaluador, ejemplo:

Dado el siguiente conjunto:

This/A<:r1={A$0-> L$0, A$1->L$0, A$2->L$0}

Podemos escribir en el Evaluador si el átomo A$0 pertenece a la signatura B:

A$0 **in** B

False //En este caso nos retorna falso porque el átomo no pertenece a B

**Join** se utiliza para quedarnos con cierta parte de algunas relaciones, ejemplo:

Supongamos que del ejemplo 3 nos queremos quedar con la 2da componente de las tuplas que correspondan a tuplas vinculadas con A0. **Ejemplo 4:**

A$0.r1

Escribimos esto porque r1 tiene tuplas cuyo primer componente es un átomo de A, al hacer el join con A$0 le decimos que nos queremos quedar con la segunda componente que tenían a A$0 como primer componente.

**Multiplicidad, en toda instancia del modelo**:

1. **Lone**: Habrá 0 o 1 átomo
2. **One**: Habrá 1 átomo
3. **Set**: Cualquier número (¿0 o más?)
4. **Some**: Habrá 1 o más átomos (o al menos 1 átomo)

Ejemplos:

**some** **sig** B {} //mínimo que haya un átomo de B en el modelo

**one** **sig** E {} //Toda instancia del modelo tendrá un átomo de E

**lone** **sig** F {} //Toda instancia del modelo tendrá a lo sumo un átomo de F