**Enunciado**

Considere la siguiente situación:

Nippur de Lagash fue un personaje antológico de la antigua Sumeria, en el IV milenio a. C. y en la región histórica del Creciente Fértil, donde se considera que se originó la revolución neolítica en Occidente y que se corresponde con parte de los territorios del Antiguo Egipto, el Levante mediterráneo y la Mesopotamia. Nippur de Lagash debe su nombre a la antigua ciudad de Lagash, conocida como la Ciudad de las Blancas Murallas, en la que Nippur era general, pero que debió abandonar con motivo de la traicionera invasión del rey Luggal-Zaggizi de Uma.

Decidido a reconquistar Lagash y liberarla del tirano Luggal-Zaggiz, Nippur comienza a recorrer diferentes ciudades, con el fin de unir un ejército capaz de reconquistar la ciudad que tanto amó. Es por eso que logra contactar a tres guerreros: Sargón, Ur-El, y Hattusil. Cada uno de ellos pertenecía a una importante ciudad de aquella época: Elam, Akkad, y Hatti. Asimismo, cada uno de ellos era experto con un arma de guerra: hacha, espada, y lanza. Cabe destacar que Nippur tuvo éxito en su misión de reclutamiento, ya que entre los tres guerreros aportaron el 100% del ejército. Por último, Nippur sabe que:

* Ur-El no es de Hatti
* El guerrero que menos aportó es un experto con la lanza.
* Sargón aportó un porcentaje mayor del ejército que el guerrero de Elam.
* Hattusil, que no aportó el 40% del ejército, es un experto con el hacha.
* El guerrero de Akkad es un experto con la espada.
* Uno de los guerreros sólo aportó el 25% del total del ejército.

Determine, haciendo uso de la herramienta Alloy, para cada guerrero: a qué ciudad pertenece, con qué arma de guerra es un experto, y qué porcentaje del ejército aportó. Especifique todo comando necesario para obtener la respuesta.

IMPORTANTE: Deberán definirse las restricciones que modelen únicamente la información brindada en el enunciado. Es decir, no deberán modelarse restricciones que correspondan a inferencias obtenidas. a partir de dicha información.

**Resolución paso a paso**

**sig Guerrero {**

**nom: one Nombre,**

**ciu: one Ciudad,**

**experto: one Arma,**

**aporto: one Porcentaje**

**}**

**abstract sig Nombre {}**

/\*Si dejáramos nombre así: sig Nombre {} entonces al no ser abstracta podría pasar q existan átomos de nombres que no sean ninguno de los nombres de los guerreros, lo cual para la definición de nuestro modelo no tiene sentido.

Ocurre lo mismo para Ciudad, Arma y Porcentaje\*/

**sig Sargos, Urel, Hatussi extends Nombre {}**

/\*cada uno de ellos extiende a Nombre para q sean disjuntos entre sí, sino no tendría sentido\*/

**abstract sig Ciudad {}**

**sig Elam, Akkad, Hatti extends Ciudad {}**

**abstract sig Arma {}**

**sig Lanza, Espada, Hacha extends Arma {}**

**abstract sig Porcentaje {}**

/\*Porcentaje no necesitamos manejarlo con números enteros, el enunciado nos dice q entre los 3 personajes aportan el 100%

uno aporta 40%, otro el 25%, y el ultimo lo calculamos nosotros que seria 35%. Conociendo los porcentajes podemos utilizarlos

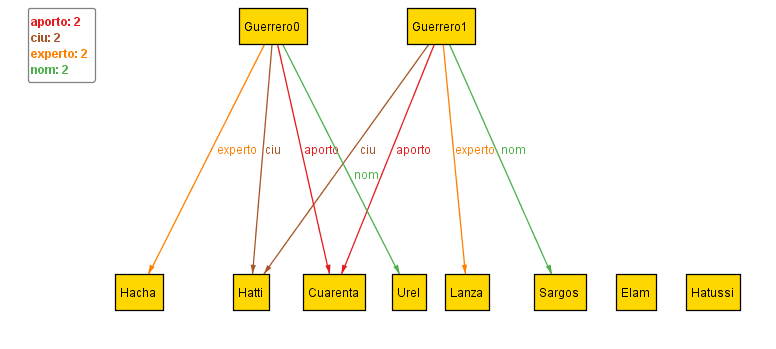
para extender a Porcentaje\*/

**sig Veinticinco, Treintaycinco, Cuarenta extends Porcentaje {}**

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

run {}

Si lo ejecutamos asi como esta, veriamos lo siguiente:

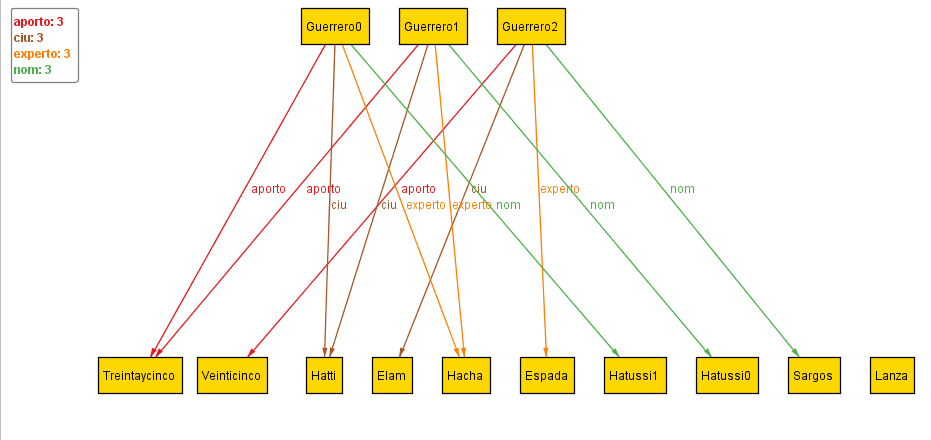


Nos estaría faltando un Guerrero (Hatussi) que se lo ve solori a la derecha. Por lo que tenemos que exigir que existan 3 Guerreros, esto lo podemos hacer mediante un hecho o con una restricción en el run {}.

Por lo que agregamos la siguiente línea:

**fact {#Guerrero = 3}** //Queremos q existan 3 guerreros

Ejecutamos y vemos lo siguiente:



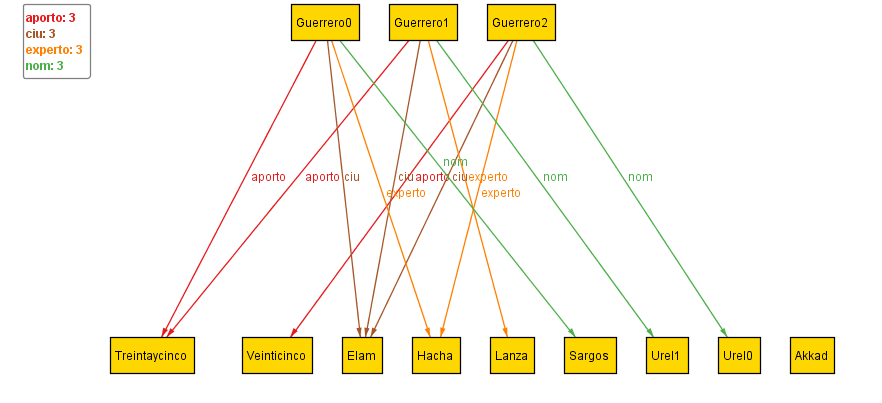
Si bien ahora tenemos 3 Guerreros, tenemos un nuevo problema, hay 2 guerreros que aportan el mismo porcentaje, hay 2 guerreros que vienen de Hatti y hay 2 guerreros que son expertos en hacha.

Hay que agregar más hechos, como los siguientes:

No hay 2 guerreros con el mismo nombre:

**fact{no disj g1,g2: Guerrero | g1.nom = g2.nom}**

/\*se lee no existen 2 átomos disjuntos de la signatura guerrero que los llamaremos g1 y g2, tales que comparten el mismo nombre\*/



Ahora podemos ver que los nombres (son las relaciones en verde) corresponden a un solo guerrero cada uno.

Ahora tenemos que hacer lo mismo para las ciudades:

**fact{no disj g1,g2: Guerrero | g1.ciu = g2.ciu }**

Ídem para las armas, pero lo vamos a escribir distinto:

No hay 2 guerreros que sean expertos en la misma arma, es lo mismo que decir **que para todo par de guerreros que sean distintos su arma va a ser distinta**

**fact{all disj g1,g2: Guerrero | g1.experto != g2.experto }**

Ídem para los porcentajes, pero lo vamos a escribir distinto de nuevo:

**fact {all g1, g2: Guerrero | (g1 != g2) implies (g1.aporto != g2.aporto)}**

/\*

all g1,g2: Guerrero

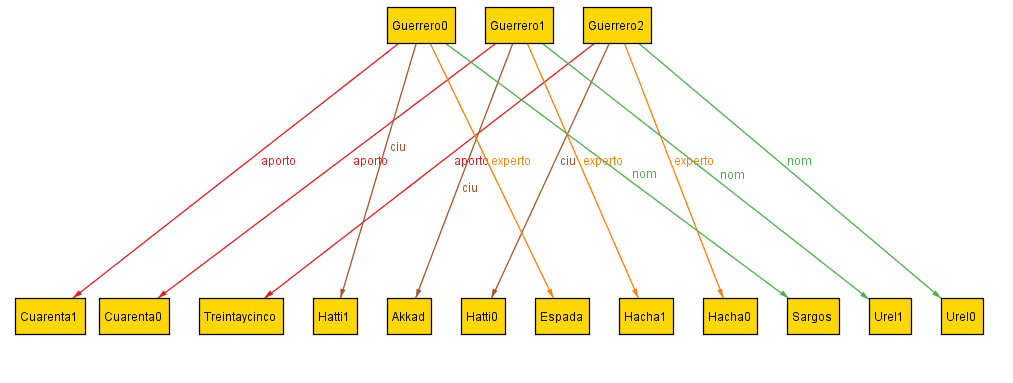
Se lee para todo Guerrero, g1 y g2 que no necesariamente son distintos

(g1 != g2) implies (g1.aporto != g2.aporto)

Si son distintos, o sea, si g1 es distinto de g2, entonces que no hayan aportado lo mismo, o sea que el aporte sea distinto

La implicación es implicación lógica, o sea, se hace verdadera en 3 casos y falsa en 1. \*/

Ahora si ejecutamos el run:

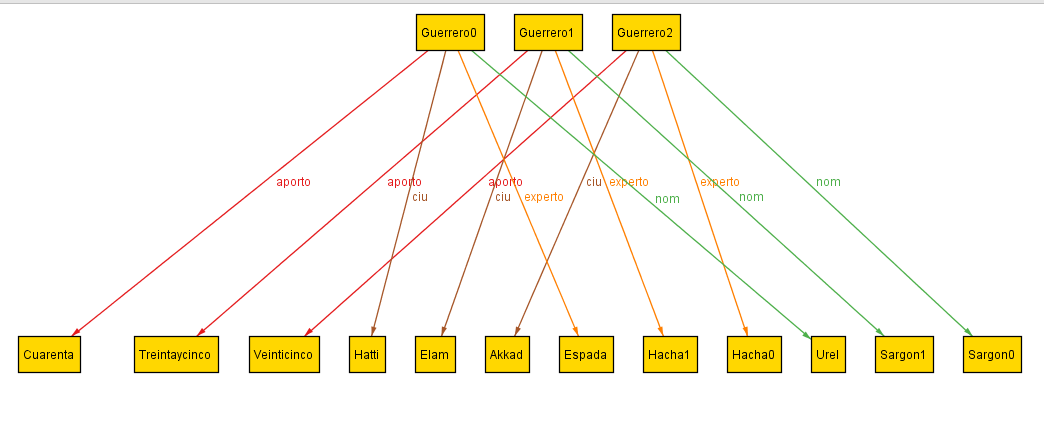


¡Ahora ya no hay 2 guerreros que posean la misma característica!

Notemos que definimos la signatura Nombre y las que lo extienden para que solo Sargos, Urel y Hatussi sean los únicos nombres que puedan tener los guerreros. Ídem para las ciudades, las armas y los porcentajes.

Así como está definido podría ocurrir que exista mas de uno de los átomos de Sargón por ejemplo, esto se puede ver forzándolo con el comando run:

**Run {#Sargon = 2}**



Para evitar que haya guerreros con el mismo nombre definiremos las signaturas que extienden a Nombre como singleton:

**one sig Sargon, Urel, Hatussi extends Nombre {}**

Ídem para las signaturas que extienden a Ciudad, Arma y Porcentaje:

**one sig Elam, Akkad, Hatti extends Ciudad {}**

**one sig Lanza, Espada, Hacha extends Arma {}**

**one sig Veinticinco, Treintaycinco, Cuarenta extends Porcentaje {}**

Ahora ya no es posible que haya más de un Sargón y lo mismo para las otras signaturas modificadas.

¿Qué sucede ahora? No tenemos una única instancia, entonces las características de cada uno de los guerreros van variando, o sea, que de momento las restricciones que hemos ido agregando no alcanzan para dar una respuesta al modelo.

Modelemos la primera restricción:

Ur-El no es de Hatti, esto quiere decir: Existe un guerrero cuyo nombre es Ur-El que no viene de Hatti.

**fact {one g: Guerrero | (g.nom = Urel) and (g.ciu != Hatti) }**

**/\***El guerrero que menos aportó es un experto con la lanza. Se podria hacer asi:

**fact {one g : Guerrero | (g.aporto = Veinticinco) and (g.experto = Lanza)}**

pero Noni le da un giro:

**\*/**

**fact {one g : Guerrero | (g.experto = Lanza) and aportoElMenor[g]}**

aportoElMenor[g] viene de aporto el menor porcentaje, esto es el nombre que le vamos a dar a un predicado que modele una relación tal que se haga verdadera cuando el Guerrero que le pasamos como argumento sea el que aporto el menor porcentaje.

Para definir predicados utilizamos la palabra reservada **pred:**

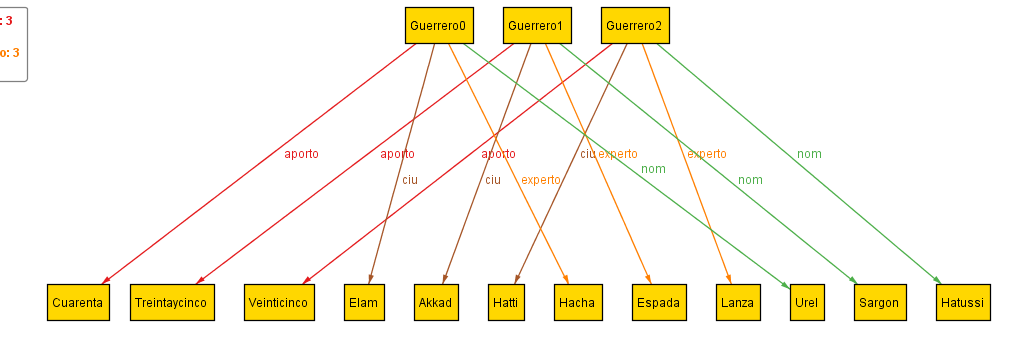
**Pred aportoElMenor[g : Guerrero]{**

**/\*Dentro del cuerpo del predicado escribimos la expresión booleana que determine que el Guerrero g es el que menos aporto\*/**

**(g.aporto = Veinticico)**

**}**

Si corremos el run, podemos ver que ahora el guerrero que menos aporto es el guerrero de la lanza.



Y esto ocurre en toda instancia de nuestro modelo.

Lo malo es que **como aun podemos generar muchas instancias entonces todavía no podemos generar una solución para nuestro problema esto es porque puede haber variantes.**

Modelemos la restricción:

Sargón aportó un porcentaje mayor del ejército que el guerrero de Elam.

**Lo siguiente no se hace**: Uno podría pensar que Sargon no es el guerrero de Elam y querer modelar esa restricción. En el **enunciado** **dice** que **no** se **modelan** **inferencias** obtenidas de las restricciones.

**fact {one g1, g2 : Guerrero | (g1.nom = Sargon) and (g2.ciu = Elam) and aportoMayor[g1,g2] }**

**…**

**pred aportoMayor[g1,g2 : Guerrero]{**

**(g1.aporto = Cuarenta) or /\*Si g1 aporto el 40 entonces es el que más aporto\*/**

**( (g1.aporto = Treintaycinco) and (g2.aporto = Veinticinco) )**

**}**

Modelamos otra restricción:

/\*Hattusil, que no aportó el 40% del ejército, es un experto con el hacha. \*/

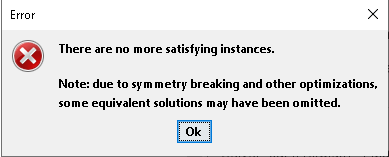
**fact {one g : Guerrero | (g.nom = Hatussi) and (g.experto = Hacha) and noAporto[g] }**

**pred noAporto[g: Guerrero]{**

**(g.aporto != Cuarenta)**

**}**

**Y como se fueron achicando las instancias que satisfacen al problema, al pedirle mas soluciones llega a un punto en el que se queda sin nuevas instancias y sale este cartelito :D**

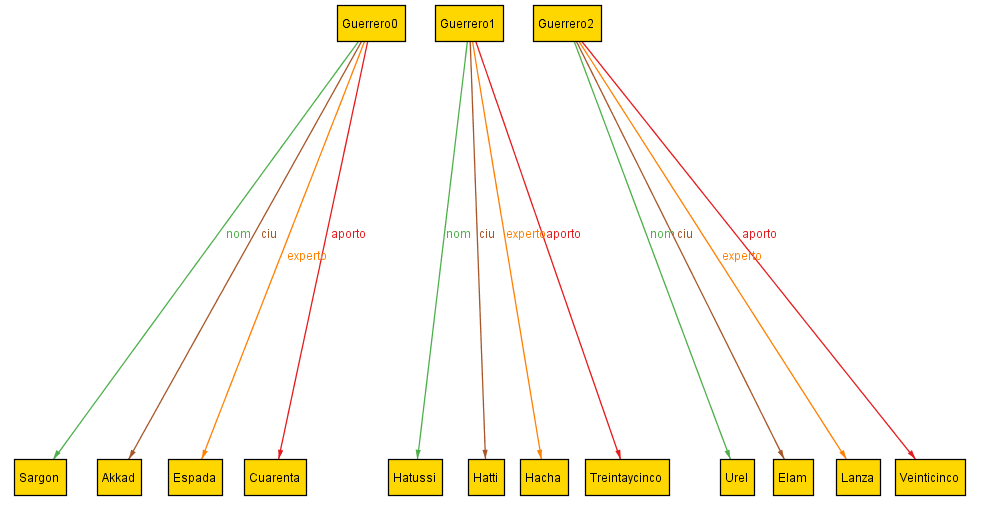


Agregamos la restricción:

/\*El guerrero de Akkad es un experto con la espada. \*/

fact {one g : Guerrero | (g.ciu = Akkad) and (g.experto = Espada) }

Con esta restricción obtenemos la instancia final, ya que solo podemos generar una instancia.



¡Por último se puede chequear que las restricciones que se comentan en el enunciado se cumplan en la instancia final!

La respuesta al ejercicio se puede dar sacando captura de la instancia final obtenida o escribiendo al final del programa el resultado final:

/\* Guerrero0: Sargon, Akkad, Espada, Cuarenta

Guerrero1: Hattusil, Hatti, Hacha, Treintaycinco

Guerrero2: UrEl, Elam, Lanza, Veinticinco

\*/