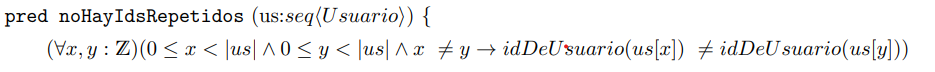
“Traducción” a lenguaje semiformal de la especificación

**Usuario válido :** la id del usuario tiene que ser un número positivo y la longitud de la lista que contiene el nombre del usuario tiene que ser mayor a 0.

Ejemplo u = (178, “pepin”), idDeUsuario (u) = 178 > 0 , por lo tanto verdadero, |nombreDeUsuario (u)| = 5 > 0, por lo tanto verdadero. Conclusión: u cumple predicado.



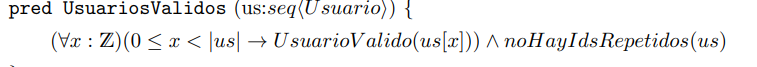
**NoHayIdsRepetidos:** para todos los elementos de la lista de Usuarios, ninguno se repite. Cada usuario tiene un id único.

****

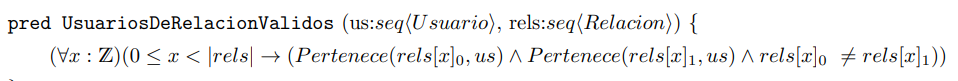
**UsuariosValidos:** tipo de dato de entrada**:** us = [ us1, us2, … , usn ], con usi = (id, nombre)

todos los elementos de la lista Usuarios cumplen

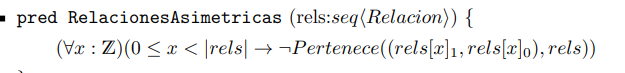
* UsuarioValido: la id del usuario tiene que ser un número positivo y la longitud de la lista que contiene el nombre del usuario tiene que ser mayor a 0.
* NoHayIdsRepetidos: para todos los elementos de la lista de Usuarios, ninguno se repite. Cada usuario tiene un id único.

****

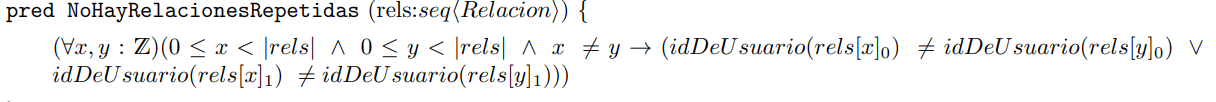
**UsuariosDeRelacionValidas:** cada elemento de la lista Relación (siendo un elemento de esa lista de la forma (usuario1, usuario2)) cumple que la primera posición (según el ejemplo sería usuario1) pertenece a la lista Usuario, lo mismo para el segundo elemento (usuario2). Además, un usuario no puede relacionarse consigo mismo, es decir usuario1 /= usuario2.



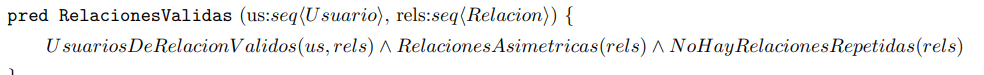
**RelacionesAsimetricas :** (rels[x]1, rels[x]0) no pertenece a la lista. Esto significa que el predicado se cumple, por ejemplo, si rels = [ (usario0, usuario1), (usuario2, usuario3), . . . , (usuario1234, usuario1235)], entonces (usuario1, usuario0) ni (usuario3, usuario2) pertenecen a la lista. O sea, la relación no es simétrica aRb (rels[x]1, rels[x]0) pertenece, pero al revés no.



**NoHayRelacionesRepetidas:** se cumple si no hay pares (usuarioK, usuarioM) y (usuarioK, usuarioM) en la lista de Relaciones, o sea, que una misma relación se repita en mi lista. Ejemplo más detallado en cuaderno.



**RelacionesValidas:** datos de entrada us = [ us1, us2, … , usn ], con usi = (id, nombre) y rels = [ (us1, us2), (us3, us4), … ,(us(n-1), usn) ]



es verdadero si se cumple que

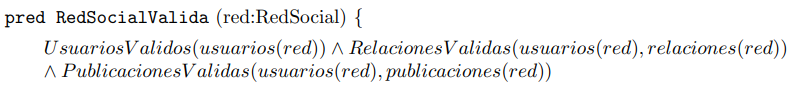
* **UsuariosDeRelacionValidas:** es verdadero si cada elemento de la lista Relación (de la forma (usuario1, usuario2)) cumple que la primera posición (según el ejemplo sería usuario1) pertenece a la lista Usuario, lo mismo para el segundo elemento (usuario2). Además, un usuario no puede relacionarse consigo mismo, es decir usuario1 /= usuario2.
* **RelacionesAsimetricas :** se cumple si (rels[x]1, rels[x]0) no pertenece a la lista. Esto significa que el predicado se cumple, por ejemplo, si rels = [ (usario0, usuario1), (usuario2, usuario3), . . . , (usuario1234, usuario1235) ], entonces (usuario1, usuario0) ni (usuario3, usuario2) pertenecen a la lista Relaciones y lo mismo para todos los elementos de dicha lista.
* **NoHayRelacionesRepetidas:** se cumple si no hay pares (usuarioK, usuarioM) y (usuarioK, usuarioM) en la lista de Relaciones, o sea, que una misma relación se repita en mi lista.
* **Resumen:** se cumple si todos los componentes de cada elem de lista Relación pertenecen a lista Usuario y ningún usuario se relaciona consigo mismo, si el inverso de una relación NO pertenece a la lista Relación y, si no se repiten las relaciones en la lista Relación.

**PublicacionesValidas:** tipo de dato de entrada us = [ us1, us2, … , usn ] y pubs = [ (id1, text1, name1), (id2, text2, name2), …. , (idn, textn, namen) ]

Cumple:

* **UsuariosDePublicacionSOnUsuariosDeRed (us:seq [usuario], pubs:seq[publicación]):** todos los usuarios que estén en un elemento de la lista Publicaciones, tienen que estar en la lista de Usuarios
* **UsuariosDelikeDePublicacionesSonUsuariosDeRed (us:seq[usuario], pubs:seq[publicación]):** todos los elementos de la lista de usuarios que le dieron like a una publicación, pertenecen a la lista Usuarios.
* **NoHayPublicacionesRepetidas (pubs:seq[publicación]):**  la id de un usuario de una publicación es distinta a la Id de otro usuario de otra publicación diferente. O el texto de una publicación es diferente al texto de otra publicación.

**RedSocialValida:** tipo de dato de entrada red = ( [Usuario], [Relacion], [Publicacion] )



Es verdadero si se cumple:

* **UsuariosValidos:** tipo de dato de entrada**:** us = [ us1, us2, … , usn ], con usi = (id, nombre) todos los elementos de la lista Usuarios cumplen
  + UsuarioValido: la id del usuario tiene que ser un número positivo y la longitud de la lista que contiene el nombre del usuario tiene que ser mayor a 0.
  + NoHayIdsRepetidos: para todos los elementos de la lista de Usuarios, ninguno se repite. Cada usuario tiene un id único.
* **RelacionesValidas:** datos de entrada us = [ us1, us2, … , usn ], con usi = (id, nombre) y rels = [ (us1, us2), (us3, us4), … ,(us(n-1), usn) ] es verdadero si se cumple que
  + **UsuariosDeRelacionValidas:** es verdadero si cada elemento de la lista Relación (de la forma (usuario1, usuario2)) cumple que la primera posición (según el ejemplo sería usuario1) pertenece a la lista Usuario, lo mismo para el segundo elemento (usuario2). Además, un usuario no puede relacionarse consigo mismo, es decir usuario1 /= usuario2.
  + **RelacionesAsimetricas :** se cumple si (rels[x]1, rels[x]0) no pertenece a la lista. Esto significa que el predicado se cumple, por ejemplo, si rels = [ (usario0, usuario1), (usuario2, usuario3), . . . , (usuario1234, usuario1235) ], entonces (usuario1, usuario0) ni (usuario3, usuario2) pertenecen a la lista Relaciones y lo mismo para todos los elementos de dicha lista.
  + **NoHayRelacionesRepetidas:** se cumple si no hay pares (usuarioK, usuarioM) y (usuarioK, usuarioM) en la lista de Relaciones, o sea, que una misma relación se repita en mi lista.
* **PublicacionesValidas:** tipo de dato de entrada us = [ us1, us2, … , usn ] y pubs = [ (id1, text1, name1), (id2, text2, name2), …. , (idn, textn, namen) ]

Verdadero si cumple los siguiente:

* + **UsuariosDePublicacionSOnUsuariosDeRed (us:seq [usuario], pubs:seq[publicación]):** todos los usuarios que estén en un elemento de la lista Publicaciones, tienen que estar en la lista de Usuarios
  + **UsuariosDelikeDePublicacionesSonUsuariosDeRed (us:seq[usuario], pubs:seq[publicación]):** todos los elementos de la lista de usuarios que le dieron like a una publicación, pertenecen a la lista Usuarios.
  + **NoHayPublicacionesRepetidas (pubs:seq[publicación]):**  la id de un usuario de una publicación es distinta a la Id de otro usuario de otra publicación diferente. O el texto de una publicación es diferente al texto de otra publicación.

**Ejercicio 1 – nombresDeUsuarios::redSocial->[String]**

Tipos de datos

Entrada: red (tipo ([Usuario],[Relacion],[Publicacion])

Salida: lista con nombres de usuarios (tipo [[Char]])

Instrucciones

Toma la LISTA USUARIOS de la red. Por cada elemento de la lista, devuelve el nombre (recordar que usuario es tipo (Int,String), así que devolvería el segundo elemento) y lo concatena a la lista resultado.

**Ejercicio 2 – amigosDe::redSocial->Usuario->[Usuario]**

Tipos de datos

Entrada: red, usuario (llamado us1)

Salida: lista de usuarios que son amigos del usuario (tipo [Usuario])

Instrucciones

Toma las relaciones de red; relaciones es una lista (tipo [Relacion]), por lo que la función va ir por cada elemento y fijarse si us1 está en la relación (relación es una tupla de dos usuarios). Si está, concatena al OTRO USUARIO (el que no es us1) a la lista que devuelve (al resultado).

**Ejercicio 3 – cantidadDeAmigos::redSocial->Usuario->Integer**

Tipos de datos

Entrada: red, usuario (us1)

Salida: Int

Instrucciones

Toma us1 y las relaciones en la red. La especificación pide usar amigosDe(red,us1) y sacar su longitud.

**Ejercicio 4 – usuarioConMasAmigos::redSocial->Usuario**

Tipos de datos

Entrada: red

Salida: usuario

Instrucciones

Toma la red (en específico, a sus usuarios) y se fija la cantidad de amigos de cada usuario. Devuelve el usuario con el máx.

**Ejercicio 5 – estaRobertoCarlos->redSocial->Bool**

Tipos de datos

Entrada: red

Salida: bool

Instrucciones

Toma la red (en específico, a sus usuarios) y se fija si un usuario tiene más de un millón de amigos.

**Ejercicio 6 – publicacionesDe::redSocial->Usuario->[Publicacion]**

Tipos de datos

Entrada: red, usuario (us1)

Salida: lista de publicaciones de us1 ([Publicacion])

Instrucciones

Toma la red, se fija en la Lista Publicaciones. Va por cada publicación y se fija si us1 es el publicador (por definición de Publicación, el usuario está en la primera coordenada de la tripla). Si cumple, concatena ESA PUBLICACION a la lista resultado.

**Recordatorio** Publicación = (UsuarioPublicador, ”texto de publicación”, [lista de usuarios que dieron like])

**Ejercicio 7 – publicacionesQueLeGustanA::redSocial->Usuario->[Publicacion]**

Tipos de datos

Entrada: red, usuario (us1)

Salida: lista de publicaciones que le gustan a us1 ([Publicacion])

Instrucciones

Toma la red, se fija en la Lista Publicaciones. Va por cada publicación y se fija si us1 pertenece a los usuarios que dieron like. Si cumple, se concatena ESA PUBLICACION a la lista resultado.

**Ejercicio 8 – lesGustanLasMismasPublicaciones::redSocial->Usuario->Usuario->Bool**

Tipos de datos

Entrada: red, usuario (us1), usuario (us2)

Salida: bool

Instrucciones

Toma la red y se fija en la Lista Publicaciones. Llama al ej7 (PublicacionesQueLeGustanA) y compara si la lista de us1 tiene los mismos elementos que la lista de us2.

**Recordatorio** Mismos elementos es otra función a ser definida

**Ejercicio 9 – tieneUnSeguidorFiel::redSocal->Usuario->Bool**

Tipos de datos

Entrada: red, usuario (us1)

Salida: bool

Instrucciones

Toma Lista Publiicaciones de la red. Busca las publicaciones de us1 (usando ej6). Va a la lista de likes de una publicación y se fija si hay algún usuario allí que esté en el resto de publicaciones (como likeador). Si no hay usuario que likeó TODAS las otras publicaciones, devuelve false.

**Ejercicio 10 – existeSecuenciaDeAmigos::redSocial->Usuario->Usuario->Bool**

Tipos de datos

Entrada: red, usuario (us1), usuario (us2)

Salida: bool

Instrucciones

Toma Lista Usuarios de la red. Arma la subsecuencia de us1 a us2 y devuelve si es una Cadena De Amigos.

**Observación, si us2 viene antes que us1 en la lista, debería testear con el REVERSO de la lista de usuarios**

**Recordatorio** es necesario definir las funciones subsecuencia, cadenaDeAmigos, reverso.

**DUDAS**

Preguntar si hay que considerar el caso en que us2 viene antes que us1 en el ej 10.

Preguntar qué funciones hay que describir: ¿las centrales o las auxiliares, o todas?