Introducción a la Programación
Segundo Entregable de laboratorio

Ejercicio 1

Implementar la función `quienGana(j1: str, j2: str) ->str` , cuya especificación es la siguiente:

`problema quienGana` (in j1: $seq\langle Char\rangle$, in j2: $seq\langle Char\rangle$) : $seq\langle Char\rangle$ {
`requiere:` $\{juegaBien(j1) \land juegaBien(j2)\}$
`asegura:` $\{gana(j1, j2) \rightarrow res = "Jugador1"\}$
`asegura:` $\{gana(j2, j1) \rightarrow res = "Jugador2"\}$
`asegura:` $\{(\neg gana(j1, j2) \land \neg gana(j2, j1)) \rightarrow res = "Empate"\}$
}

`pred juegaBien` (j: $seq\langle Char\rangle$) {
$j = "Piedra" \lor j = "Papel" \lor j = "Tijera"$
}

`pred gana` (j1, j2: $seq\langle Char\rangle$) {
$piedraGanaAtijera(j1, j2) \lor tijeraGanaAPapel(j1, j2) \lor papelGanaAPiedra(j1, j2)$
}

`pred piedraGanaAtijera` (j1, j2: $seq\langle Char\rangle$) {
$j1 = "Piedra" \land j2 = "Tijera"$
}

`pred tijeraGanaAPapel` (j1, j2: $seq\langle Char\rangle$) {
$j1 = "Tijera" \land j2 = "Papel"$
}

`pred papelGanaAPiedra` (j1, j2: $seq\langle Char\rangle$) {
$j1 = "Papel" \land j2 = "Piedra"$
}

Aclaración: Respetar los nombres de “Piedra“, “Papel“ y “Tijera“ (empieza con mayúsculas)

Introducción a la Programación
Segundo Entregable de laboratorio

Ejercicio 2

Implementar la función `fibonacciNoRecursivo(n: int) ->int` , cuya especificación es la siguiente, **sin utilizar recursión**:

`problema fibonacciNoRecursivo` (in n: $\mathbb{Z}$) : $\mathbb{Z}$ {
    `requiere:` $\{n \geq 0\}$
    `asegura:` $\{(\exists l : seq\langle\mathbb{Z}\rangle)(|l| = n + 1 \land esSecuenciaFibonacci(l) \land l[|l| - 1] = result)\}$
}

`pred esSecuenciaFibonacci` (l: $seq\langle\mathbb{Z}\rangle$) {
    $(|l| > 0 \rightarrow l[0] = 0) \land$
    $(|l| > 1 \rightarrow l[1] = 1) \land$
    $(\forall i : \mathbb{Z})(2 \leq i < |l| \rightarrow l[i] = l[i-1] + l[i-2])$
}

Introducción a la Programación
Segundo Entregable de laboratorio

Ejercicio 3

Implementar la función `mesetaMasLarga(...) ->int` , cuya especificación es la siguiente:

`problema mesetaMasLarga` (in l: $seq\langle\mathbb{Z}\rangle$) : $\mathbb{Z}$ {
`requiere:` $\{True\}$
`asegura:` $\{(|l| = 0 \land result = 0) \lor (hayMesetaDeLong(l, result) \land \neg hayMesetaDeLong(l, result + 1))\}$
}

`pred hayMesetaDeLong` (l: $seq\langle\mathbb{Z}\rangle$, n: $\mathbb{Z}$) {
$(\exists i : \mathbb{Z})(\exists j : \mathbb{Z})(0 \leq i \land i \leq j < |l| \land j - i + 1 = n \land todosIguales(l, i, j))$
}

`pred todosIguales` (l: $seq\langle\mathbb{Z}\rangle$, i: $\mathbb{Z}$, j: $\mathbb{Z}$) {
$(\forall k : \mathbb{Z})(i \leq k \leq j \rightarrow l[k] = l[i])$
}

Introducción a la Programación
Segundo Entregable de laboratorio

Ejercicio 4

Implementar la función `def filasParecidas(...)->bool`, cuya especificación es la siguiente:

`problema filasParecidas` (in m: $seq\langle seq\langle \mathbb{Z}\rangle\rangle$) : Bool {
    `requiere:` $\{esMatriz(m)\}$
    `asegura:` $\{res = True \leftrightarrow (\exists n : \mathbb{Z})(filasParecidasAanterior(m, n))\}$
}

`pred esMatriz` (m: $seq\langle seq\langle \mathbb{Z}\rangle\rangle$) {
    $|m| > 0 \land |m[0]| > 0 \land (\forall i : \mathbb{Z})(0 \leq i < |m| \rightarrow |m[i]| = |m[0]|)$
}

`pred filasParecidasAanterior` (m: $seq\langle seq\langle \mathbb{Z}\rangle\rangle$, n: $\mathbb{Z}$) {
    $(\forall i : \mathbb{Z})(1 \leq i < |m| \rightarrow filaAnteriorMasN(m, i, n))$
}

`pred filaAnteriorMasN` (m: $seq\langle seq\langle \mathbb{Z}\rangle\rangle$, i, n: $\mathbb{Z}$) {
    $(\forall j : \mathbb{Z})(0 \leq j < |m[0]| \rightarrow m[i][j] = m[i-1][j] + n)$
}

Introducción a la Programación
Segundo Entregable de laboratorio

Ejercicio 5

Contamos con un listado de vuelos, donde cada vuelo está representado por dos datos: la ciudad de la cual parte el vuelo (origen), y la ciudad a la cual llega (destino). Implementar la función `sePuedeLlegar`, que indica si hay una forma de llegar desde una ciudad a otra utilizando las conexiones disponibles en el listado de vuelos, y en caso afirmativo devuelve la cantidad de vuelos que deben tomarse.

La especificación detallada de la función que debe implementarse es la siguiente:

`problema sePuedeLlegar` (in origen: String, in destino: String, in vuelos: $seq\langle String \times String\rangle$) : $\mathbb{Z}$ {
  `requiere:` $\{origen \neq destino\}$
  `requiere:` $\{soloParteUnVueloDeCadaCiudad(vuelos) \land soloLlegaUnVueloAcadaCiudad(vuelos) \land sinRepetidos(vuelos)\}$
  `asegura:` $\{hayRuta(vuelos, origen, destino) \rightarrow largoDeRuta(vuelos, origen, destino, res)\}$
  `asegura:` $\{\neg hayRuta(vuelos, origen, destino) \rightarrow res = -1\}$
}

`pred soloParteUnVueloDeCadaCiudad` (vuelos: $seq\langle String \times String\rangle$) {
  $(\forall i, j : \mathbb{Z})((0 \leq i < |vuelos| \land 0 \leq j < |vuelos| \land i \neq j) \rightarrow vuelos[i]_0 \neq vuelos[j]_0)$
}

`pred soloLlegaUnVueloAcadaCiudad` (vuelos: $seq\langle String \times String\rangle$) {
  $(\forall i, j : \mathbb{Z})((0 \leq i < |vuelos| \land 0 \leq j < |vuelos| \land i \neq j) \rightarrow vuelos[i]_1 \neq vuelos[j]_1)$
}

`pred hayRuta` (vuelos: $seq\langle String \times String\rangle$, origen, destino: String) {
  $(\exists ruta : seq\langle String \times String\rangle)(vuelosValidos(ruta, vuelos) \land |ruta| \geq 1 \land ruta[0]_0 = origen \land ruta[|ruta| - 1]_1 = destino \land caminoDeVuelos(ruta))$
}

`pred vuelosValidos` (ruta, vuelos: $seq\langle String \times String\rangle$) {
  $sinRepetidos(ruta) \land (\forall v : String \times String)(v \in ruta \rightarrow v \in vuelos)$
}

`pred caminoDeVuelos` (vuelos: $seq\langle String \times String\rangle$) {
  $(\forall i : \mathbb{Z})(1 \leq i < |ruta| \rightarrow ruta[i]_0 = ruta[i-1]_1)$
}

`pred largoDeRuta` (ruta: $seq\langle String \times String\rangle$, origen, destino: String, longCamino: $\mathbb{Z}$) {
  $(\exists ruta : seq\langle String \times String\rangle)(vuelosValidos(ruta, vuelos) \land |ruta| \geq 1 \land ruta[0]_0 = origen \land ruta[|ruta| - 1]_1 = destino \land caminoDeVuelos(ruta) \land |ruta| = longCamino)$
}

Nota: Si desean realizar pruebas, se espera que el input tenga el siguiente formato:

```
origen
destino
ciudadOrigen [COMA] ciudadDestino [ESPACIO] ciudadOrigen [COMA] ciudadDestino...
```

Por ejemplo:

```
rosario
jujuy
misiones,jujuy salta,chubut rosario,misiones
```

Notar que cada vuelo se separa del siguiente mediante un espacio, y las ciudades origen y destino de un vuelo se separan por una coma

En este caso, el origen sería *Rosario*, el destino *Jujuy*, y existen 3 vuelos (*Misiones*→*Jujuy*,*Salta*→*Chubut* y *Rosario*→*Misiones*)