Trabajo Práctico

Fecha 16/09/2023 Algoritmos y Estructura de Datos

Grupo HSPJCEPICOWRBZPNWNAV

Integrante	LU	Correo electrónico
Moyano Tassara, Iván	237/21	ivanmoyano@hotmail.com
Viera, Joaquín Ezequiel	253/21	joaquin.2001.viera.chuko@gmail.com
Galli Casado Sastre, Lucas	739/21	lucasgalli01@gmail.com
Fernández, Ana Celeste	41/19	anacelestefernandez@gmail.com

Facultad de Ciencias Exactas y Naturales

Universidad de Buenos Aires

Ciudad Universitaria - (Pabellón I/Planta Baja)

Intendente Güiraldes 2610 - C1428EGA

Ciudad Autónoma de Buenos Aires - Rep. Argentina

Tel/Fax: (++54 +11) 4576-3300

http://www.exactas.uba.ar

1. Ejercicio 1: hayBallotage

1.1. Especificación

`proc` `hayBallotage` (in escrutinio :$seq\langle\mathbb{Z}\rangle$) : $Bool$

`requiere` $\{(esValido(escrutinio)\}$

`asegura` $\{res = False \iff (\exists i : \mathbb{Z})((0 \leq i < |escrutinio| - 1) \wedge$ escrutinio[i]/totalVotos(escrutinio) ≥ 0.45) $\vee$ (hayGranDiferencia(escrutinio,i) $\wedge$ escrutinio[i]/totalVotos(escrutinio) $\geq 0.4)\}$

1.2. Auxiliares

`aux` `totalVotos` (escrutinio :$seq\langle\mathbb{Z}\rangle$) : $\mathbb{Z}$ = $\sum_{i=0}^{|escrutinio|-1}(escrutinio[i])$;

`pred` `esValido` (in escrutinio : seq$\langle Z\rangle$) {

$(|escrutinio| \geq 3) \wedge (\forall i : \mathbb{Z})(0 \leq i < |escrutinio|) \rightarrow_L (escrutinio[i] \geq 0) \wedge ((\forall i, j : \mathbb{Z})(0 \leq i, j \leq |escrutinio| - 1) \wedge (i \neq j)) \rightarrow_L (escrutinio[i] \neq escrutinio[j])$ }

`pred` `hayGranDiferencia` (escrutinio :$seq\langle\mathbb{Z}\rangle$, i:$\mathbb{Z}$) {

$(\forall k : \mathbb{Z})(0 \leq k < |escrutinio| - 1 \longrightarrow_L escrutinio[i] - escrutinio[k] \geq 0{,}1 * totalVotos(escrutinio)$

}

1.3. Algoritmo

```
bool res := true;
int i := 0;
while(i <|escrutinio|- 2 && res){

   res =¬(porcentaje(escrutinio,i)≥ 0.45 || ganoPorDiferencia(escrutinio,i) ;


}
return res;
```

1.3.1. Auxiliares del Algoritmo:

```
porcentaje(escrutinio,i){

   sum := 0;
   j := 0;
   while(j<|escrutinio|){

      sum := sum + escrutinio[i]
   }
   int res := s[i]/sum;
   return res ;
}
ganoPorDiferencia(escrutinio, i){
   dif := porcentaje(escrutinio,i) - porcentaje(escrutinio,0);
   j := 1;
   while(j <|escrutinio| - 2){
```

```
    if(porcentaje(escrutinio,i) - porcentaje(escrutinio,j) >dif) {
       dif = porcentaje(escrutinio,i) - porcentaje(escrutinio,j);
       }
  }
  bool res := (dif >= 0.1 ) ∧ (porcentaje(escrutinio,i) >= 0.4);

  return res;
  }
```

2. Ejercicio 2: HayFraude

2.1. Especificación

proc HayFraude (in escrutinio_presidencial: seq⟨$\mathbb{Z}$⟩, in escrutinio_senadores: seq⟨$\mathbb{Z}$⟩, in escrutinio_diputados: seq⟨$\mathbb{Z}$⟩) : Bool

requiere {$esValido(escrutinio\_presidencial)$∧ esValido(escrutinio_senadores)∧ esValido(escrutinio_diputados)∧|escrutinio_presidencial|=|escrutinio_senadores|∧ |escrutinio_presidencial|=|escrutinio_diputados|}

asegura {$res = False$↔ (totalVotos(escrutinio_presidencial) = totalVotos(escrutinio_senadores)) ∧ (totalVotos(escrutinio_senadores = totalVotos(escrutinio_diputados)))}

2.2. Auxiliares

Utilizamos la auxiliar totalVotos(ver item 1).

2.3. Algoritmo

```
hayFraude(escrutinio_p, escrutinio_s,escrutinio_d){
    int votos_p := 0;
    int votos_s := 0;
    int votos_d := 0;
    int i := 0;
    while(i<|escrutinio_p|){
        votos_p := votos_p + escrutinio_p[i];
        votos_s := votos_s + escrutinio_s[i];
        votos_d := votos_d + escrutinio_d[i];
        i:= i+1;
    }
    bool res:= not((votos_p == votos_s) && (votos_s == votos_d));

    return res;
}
```

2.4. Correctitud de Algoritmo:

Para demostrar la correctitud del ciclo definimos P_C y Q_C :

P_C = {i = 0 ∧ votos_p = 0 ∧ votos_s = 0 ∧ votos_d = 0 ∧ |escrutinio_p| = |escrutinio_s| ∧ |escrutinio_s|= |escrutinio_d|}

Q_C = { votos_p = $\sum_{i=0}^{|escrutinio\_p|-1} escrutinio\_p[i] \wedge$ votos_s = $\sum_{i=0}^{|escrutinio\_p|-1} escrutinio\_s[i] \wedge$ votos_d = $\sum_{i=0}^{|escrutinio\_p|-1} escrutinio\_d[i]\}$

Se puede obsvervar que para obtener wp(codigo previo al ciclo, P_C) basta con remplazar la variables i, votos_p, votos_s,votos_d por cero, generando (0=0) en cada uno de los lugares de las respectivas variables. Esto nos deja con la equivalencia, wp(codigo previo al ciclo, P_C) ≡ |escrutinio_p| = |escrutinio_s| ∧ |escrutinio_s| = |escrutinio_d|.
Como Pre ⇒ |escrutinio_p| = |escrutinio_s| ∧ |escrutinio_s| = |escrutinio_d| pues esta explicitamento dicho en su definicion, queda claro que Pre ⇒ wp(codigo previo al ciclo, P_c)

Demostraremos la correctidud parcial de $\{P_C\}$ S_C $\{Q_C\}$, donde S_C es el ciclo while presente en el algoritmo.

Teorema del Invariante:

Defnimos el invariante:

I ≡ (0 ≤ i ≤ |escrutinio_p|) ∧ (votos_p = $\sum_{j=0}^{i-1} escrutinio\_p[i]) \wedge$ (votos_s = $\sum_{j=0}^{i-1} escrutinio\_s[i]) \wedge$ (votos_d = $\sum_{j=0}^{i-1} escrutinio\_d[i]))$

1.) P_C ⇒ I:
P_C ⇒ i = 0 ∧ votos_p = 0 ∧ votos_s = 0 ∧ votos_d = 0
⇒ (i=0) ∧ (votos_p = $\sum_{j=0}^{0-1} escrutinio\_p[i]) \wedge$ (votos_s = $\sum_{j=0}^{0-1} escrutinio\_s[i]) \wedge$ (votos_d = $\sum_{j=0}^{0-1} escrutinio\_d[i])$
⇒ (0≤ i ≤ |escrutinio_p|) ∧ (votos_p = $\sum_{j=0}^{i-1} escrutinio\_p[i]) \wedge$ (votos_s = $\sum_{j=0}^{i-1} escrutinio\_s[i]) \wedge$ (votos_d = $\sum_{j=0}^{i-1} escrutinio\_d[i]) \equiv I$

2.) {I ∧ B}S_C{I}:

wp(i:=i+1,I) ≡ def(I) ∧ $Q^I_{i=i+1}$ ≡ (0≤ i+1 ≤ |escrutinio_p|) ∧ (votos_p = $\sum_{j=0}^{i} escrutinio\_p[$ (votos_s = $\sum_{j=0}^{i} escrutinio\_s[i]) \wedge$ (votos_d = $\sum_{j=0}^{i} escrutinio\_d[i])$

(1) wp(votos_d := votos_d + escrutinio_d[i] wp(i:=i+1,I)) ≡ (0≤ i <|escrutinio_p|) ∧ (votos_p = $\sum_{j=0}^{i} escrutinio\_p[i]) \wedge$
(votos_s = $\sum_{j=0}^{i} escrutinio\_s[i]) \wedge$ (votos_d = $\sum_{j=0}^{i-1} escrutinio\_d[i])$

(2) wp(votos_s := votos_s + escrutinio_s[i], (1)) ≡ (0∧ i <|escrutinio_p|) ∧ (votos_p = $\sum_{j=0}^{i} escrutinio\_p[i]) \wedge$ (votos_s = $\sum_{j=0}^{i-1} escrutinio\_s[i]) \wedge$ (votos_d = $\sum_{j=0}^{i-1} escrutinio\_d[i])$

(3) wp(votos_p := votos_p + escrutinio_p[i], (2)) $\equiv$ (0$\wedge$ i <|escrutinio_p|) $\wedge$ (votos_p = $\sum_{j=0}^{i-1} escrutinio\_p[i]$)$\wedge$ (votos_s = $\sum_{j=0}^{i-1} escrutinio\_s[i]$)$\wedge$ (votos_d = $\sum_{j=0}^{i-1} escrutinio\_d[i]$)

wp(S,I)$\equiv$ $wp(votos_p := votos_p{+}escrutinio_p[i], votos_s := votos_s{+}escrutinio_s[i], votos_d := votos_d + escrutinio_d[i], i := i+1, I)$$\equiv$ (3) por Axioma 3

{I$\wedge$B} $\equiv$ (0 $\leq$ i <|escrutinio_p|) $\wedge$ (votos_p = $\sum_{j=0}^{i-1} escrutinio\_p[i]$)$\wedge$ (votos_s = $\sum_{j=0}^{i-1} escrutinio\_s[i]$)$\wedge$ (votos_d = $\sum_{j=0}^{i-1} escrutinio\_d[i]$)

Por lo tanto podemos ver que wp(S,I) $\Rightarrow$ {I$\wedge$B}

3.) I $\wedge$($\neg$ B) $\Rightarrow$ Q$_c$

I $\wedge\neg$ B $\equiv$ (0 $\leq$ i $\leq$ |escrutinio_p|) $\wedge$ (votos_p = $\sum_{j=0}^{i-1} escrutinio\_p[i]$)$\wedge$ (votos_s = $\sum_{j=0}^{i-1} escrutinio\_s[i]$)$\wedge$ (votos_d = $\sum_{j=0}^{i-1} escrutinio\_d[i]$)$\wedge$ (i $\geq$ |escrutinio_p|)
$\Rightarrow$ (votos_p = $\sum_{j=0}^{|escrutinio\_p|-1} escrutinio\_p[i]$)$\wedge$ (votos_s = $\sum_{j=0}^{|escrutinio\_p|-1} escrutinio\_s[i]$)$\wedge$ (votos_d = $\sum_{j=0}^{|escrutinio\_p|-1} escrutinio\_d[i]$)$\equiv$ Q$_C$

<u>Teorema de Terminacion:</u>
Definimos f_v := |escrutinio_p| - i, veamos que cumple las dos condiciones del teorema.

1.) {I $\wedge$ B $\wedge$ v$_o = f_v$}S{f$_v$ <v_0 }
Basta ver que wp(S,f_v <v_0) $\Rightarrow$ {I $\wedge$ B $\wedge$ $v_o = f_v$ }.
wp(S,f_v <v_0) $\equiv$ wp(votos_p := votos_p + escrutinio_p[i], votos_s := votos_s + escrutinio_s[i],votos_p := votos_p + escrutinio_p[i],i:=i+1,f_v <v_0).

wp(i:= i+1, f_v <v_0) $\equiv$ |escrutinio$_p| - i - 1 <$v$_0$

wp(votos_d := votos_d + escrutinio_d[i],|escrutinio_p| -i-1) $\equiv$
(0 $\leq$ i <|escrutinio_p|) $\wedge$ (|escrutinio_p| -i-1)
Como en resto de los wp no aparece no aparece nada que depende de la varible que se esta modificando es evidente que resutla:

wp(S,f_v <v_0) $\equiv$ (0 leq i <|escrutinio_p|) $\wedge$ (|escrutinio_p| -i-1)

{ I $\wedge$ B $\wedge$ ($v_0 = f_v$)} $\equiv$ (0 $\leq$ i <|escrutinio_p|) $\wedge$ (votos_p = $\sum_{j=0}^{i-1} escrutinio\_p[i]$)$\wedge$ (votos_s = $\sum_{j=0}^{i-1} escrutinio\_s[i]$)$\wedge$ (votos_d = $\sum_{j=0}^{i-1} escrutinio\_d[i]$)$\wedge$(v$_0$ = |escrutinio_p| - i)

como v_0 = |escrutinio_p| -i y son numeros enteros vale que v_0 >|escrutinio_p| - i -1, luego vale que { I $\wedge$ B $\wedge$ ($v_0 = f_v$)} $\Rightarrow$ wp(S,f_v <v_0).

2.) I $\wedge$ f_v $\leq$ 0 $\Rightarrow$ $\neg$ B
I $\wedge$ f_v $\leq$ 0 $\equiv$ (0 $\leq$ i $\leq$ |escrutinio_p|) $\wedge$ ... $\wedge$ (|escrutinio_p| - i).

Como |escrutinio_p| $\leq$ i $\wedge$ e $\geq$ i $\Rightarrow$ i = |escrutinio_p| $\Rightarrow$ $\neg$(i<|escrutinio_p|) $\equiv$ $\neg$ B. Esto demuestra que se cumple lo pedido

Para finalizar vemos que $Q_C \Rightarrow$ wp(codigo posterior al ciclo, Post).

Para facilitar esta ultima parte voy a reescribir la Post es base a las variables usadas en el algoritmo ya que dada la correctitud parcial del ciclo su remplazo es razonable.
Post = {res = False $\iff$ votos_p = votos_s $\wedge$ votos_s = votos_d }
Luego:

wp(res := not((votos_p == votos_s) $\wedge$ (votos_s == votos_d)), Post) $\equiv$ not((votos_p == votos_s) $\wedge$ (votos_s == votos_d)) = False $\iff$ votos_p == votos_s $\wedge$ votos_s == votos_d $\equiv True.Luegoloquequeriamosprobarvaletrivialmente$

3. Ejercicio 3: obtenerSenadoresEnProvincia

3.1. Especificación

proc obtenerSenadoresEnProvincia (in escrutinio $: seq\langle\mathbb{Z}\rangle$, out res $:\mathbb{Z}$x$\mathbb{Z}$)
requiere $\{esValido(escrutinio)\}$
asegura $\{salioPrimero(res[0]) \wedge salioSegundo(res[1])\}$

3.2. Auxiliares

pred salioPrimero (escrutinio $: seq\langle\mathbb{Z}\rangle$, i $:\mathbb{Z}$) {
$\#mayoresAi(i) = 0$
}
pred salioSegundo (escrutinio $: seq\langle\mathbb{Z}\rangle$, i $:\mathbb{Z}$) {
$\#mayoresAi(i) = 1$
}
aux #mayoresAi (escrutinio $: seq\langle\mathbb{Z}\rangle$, i $:\mathbb{Z}$) : $\mathbb{Z}$ =
$\sum_{j=0}^{|escrutinio|-2}$(if $escrutinio[j] > escrutinio[i]$ then 1 else 0 fi);

3.3. Algoritmo

```
mayoresAi(escrutinio,i){
    int res := 0;
    int j := 0;
    while(j<|escrutinio|{
        if(escrutinio[i] > escrutinio[j]{
        res = res + 1;
        }
    }
    return res;
}

salioPrimero(escrutinio,i){
    bool res := (mayoresAi==0)
    return res;
```

```
}

salioSegundo(escrutinio,i){
    bool res := (mayoresAi==1)
    return res;
}

obtenerSenadoresEnProvincia(escrutinio){
    [int] res:= [-1,-1];
    int k := 0 ;
    while(k<|escrutinio| - 1){
        if(salioPrimero(escrutinio,k){
            res [0] := k;
        }

        if(salioSegundo(escrutinio,k){
            res [1] := k;
        }
        k := k + 1;
    }
    return res;
}
```

3.4. Correctitud de Algoritmo

3.4.1. Corrección de obtenerSenadoresEnProvincia

Para probar la correctitud de obtenerSenadoresEnProvincia, como tenemos un ciclo de la forma:
while B do
S
endwhile.
Por lo tanto, será cómodo usar el teorema del invariante y de terminación de un ciclo.
Por claridad, notemos que:
B: {k <|escrutinio| - 1}
Pc: {res=[-1,-1] $\wedge$ k=0}
Qc: {salioPrimero(res[0]) $\wedge$ salioSegundo(res[1])}
Usemos el teorema del invariante para ver que si el ciclo termina, lo hace bien.
Proponemos el siguiente invariante:
I: {(((res[0]=-1) $\wedge_L$
(($\exists$ m:$\mathbb{Z}$)(k $\leq$ m $\leq$ |escrutinio|-1)$\wedge_L$
$(salioPrimero(res[m]) = true)$))$\vee$(salioPrimero(res[0])=true))$\wedge_L$((($res[1] = -1$)$\wedge_L$
(($\exists$ n:$\mathbb{Z}$)(k $\leq$ n $\leq$ |escrutinio|-1)$\wedge_L$
$(salioSegundo(res[n]) = true)$))$\vee$(salioSegundo(res[1])=true))
$\wedge$ (0 $\leq$ k $\leq$ |escrutinio|)}

Para ver que se cumple el teorema del invariante hace falta ver tres cosas:
1) Pc $\rightarrow$ I
Estoy es fácil de ver, ya que si k = 0, o bien res[0] = res[1] = -1 y naturalmente en escrutinio hay alguna posición que salió primera y segunda, o bien alguno de

los dos, res[0] ó res[1] tomó el valor 0, en caso tal, este valor salió primero o segundo según corresponda (no pasa que res[0] ó res[1] tome el valor 0, ademas podes cambiar el naturalmente por que estas evaluando el existe en todo el rango de escrutinio).

2) {I∧B}S{I} es una tripla de Hoare válida.
Para esto, calculemos la weakest precondition de I dado S y veamos que I∧B la implica.
Escribiendo a S = S1;S2;S3, donde S1 es el primer if, S2 el segundo y S3 la asignación tenemos que:
Wp(S,I) = Wp(S1;S2,I^{k}_{k+1}) = Wp(S1,Wp(S2,I^{k}_{k+1}))
Usando que S2 es un condicional de la forma
if (salioSegundo(escrutinio,k)) then res[1] = k else skip endif, tenemos que:
Wp(S,I) =
= Wp(S1;def(salioSegundo(escrutinio,k))$\wedge_L((salioSegundo(escrutinio, res[1])\wedge I^{res[1]}_{k})$
∨(¬salioSegundo(escrutinio,res[1])∧ I)).
Por un lado def(salioSegundo(escrutinio,res[1]) = true, por otro lado
salioSegundo(escrutinio,res[1])∧$I^{res[1]}_{k}$ = ((res[0]=-1) $\wedge_L$
((∃ m:$\mathbb{Z}$)(k ≤ m ≤ |escrutinio|-1)$\wedge_L$
$(salioPrimero(escrutinio, res[m]) = true)))$∨(salioPrimero(escrutinio,res[0])=true))∧
(0 ≤ k ≤ |escrutinio|). Esto pues el segundo ∧ del invariante I se vuelve true debido a que salioSegundo(escrutinio,k) = true.
Por último, ¬salióSegundo(escrutinio,k)∧I =
(((res[0]=-1) $\wedge_L$
((∃ m:$\mathbb{Z}$)(k ≤ m ≤ |escrutinio|-1)$\wedge_L$
$(salioPrimero(res[m]) = true)))$∨(salioPrimero(res[0])=true))$\wedge_L(((res[1] = -1)\wedge_L$
((∃ n:$\mathbb{Z}$)(k ≤ n ≤ |escrutinio|-1)$\wedge_L$
$(salioSegundo(res[n]) = true)))$
∧ (0 ≤ k ≤ |escrutinio|).
De modo que Wp(S2,$I^{res[1]}_{k}$) = (((res[0]=-1) $\wedge_L$
((∃ m:$\mathbb{Z}$)(k ≤ m ≤ |escrutinio|-1)$\wedge_L$
$(salioPrimero(res[m]) = true)))$∨(salioPrimero(res[0])=true))$\wedge_L(((res[1] = -1)\wedge_L$
((∃ n:$\mathbb{Z}$)(k ≤ n ≤ |escrutinio|-1)$\wedge_L$
$(salioSegundo(res[n]) = true)))$
∧ (0 ≤ k ≤ |escrutinio|).
Llamemos a toda esta nueva proposición J. Así, pasando en limpio, tenemos que Wp(S,I) = Wp(S1,J).
Usando que S1 es un condicional if salioPrimero(escrutinio,k) = true then res[0] = k else skip endif, podemos decir que:
Wp(S,I) = def(J)$\wedge_L(salioPrimero(escrutinio, k)\wedge J^{res[0]}_{k})$
∨(¬salioPrimero(escrutinio,k)∧J).
Hagamos nuevamente observaciones. def(J) = true.(aca no seria el definido de la guarda?)
salioPrimero(escrutinio,k)∧$J^{res[0]}_{k}$ = true. Pues $J^{r}_{k}es[0]$ es un ∨ y una de las proposiciones que tiene se hace true precisamente porque salioPrimero(escrutinio,k) = true.
Como todo Wp(S1;J) es un ∨, del mismo modo por ya tener un true, se hace true. De modo que
Wp(S,I) = true.

3) Veamos que I∧ ¬B → Qc.

Esto es directo ya que si no se cumple B, entonces en el I nunca se va a cumplir que existe un m entre k y escrutinio - 1, por lo que esa parte va a ser falsa. Haciendo que deba ser verdadero tanto salioPrimero(escrutinio,res[0]) como salioSegundo(escrutinio,res[1]). Con esto se demostró que si el algoritmo termina, lo hace bien.(que pasa si en res[0] o res[1] hay un -1 entonces no hay primero o segundo, deberiamos pedir de precondicion que haya mas de dos partidos)

Demostremos que termina con el teorema de terminación de un ciclo. Propongamos la función fv = |escrutinio| - k.
Veamos que se cumplen:
1) La tripla de Hoare $\{I \wedge B \wedge fv=v0\}S\{fv < v0\}$
Para esto, calculamos la weakest precondition. Wp(S,fv <v0) =
= Wp(S1;S2,fv - 1 <v0) = Wp(S1,Wp(S2,fv - 1 <v0)).
Notemos que Wp(S2,fv - 1 <v0) = def(salioSegundo(escrutinio,k)$\wedge_L$
$((salioSegundo(escrutinio, k)$$\wedge$fv - 1 <v0)
$\vee$ ($\neg$salioSegundo(escrutinio,k)$\wedge$fv - 1 <v0)) = fv - 1 <v0.
De modo que Wp(S,fv <v0) = Wp(S1, fv - 1 <v0).
Por motivos completamente análogos a los anteriores, este último weakest precondition se escribe como
Wp(S,fv <v0) = Wp(S1, fv - 1 <v0) = fv - 1 <v0.
Falta ver que I$\wedge$B$\wedge$ fv = v0 $\rightarrow$ fv - 1 <v0.
Esto es directo ya que fv = v0 $\rightarrow$ fv - 1 <v0.

Ahora veamos que 2) I$\wedge$fv $\leq$ 0 $\rightarrow$ $\neg$B.
Pues claro, si fv = |escrutinio| - k $\leq$ 0, es |escrutinio| $\leq$ k.
De modo que $\neg$B. Probando así que el algoritmo termina.
Con esto y la buena terminación se probó la correctitud del algoritmo obtenerSenadoresEn

4. Ejercicio 4: calcularDHondtEnProvincia

4.1. Especificación

proc calcularDHondtEnProvincia (in escrutinio :$seq\langle\mathbb{Z}\rangle$, in cant_bancas: $\mathbb{Z}$) : $seq\langle seq\langle\mathbb{R}\rangle\rangle$

requiere $\{esValido(escrutinio)\}$
asegura $\{res[0] = sacarUltimo(escrutinio)\}$
asegura $\{esDHondt(res)\}$
asegura $\{|res| = cant\_bancas\}$
asegura $\{(\forall i : \mathbb{Z})(0 \leq i \leq cant\_bancas - 1) \rightarrow_L (|res[i]| = |escrutinio| - 1\}$

4.2. Auxiliares

pred esDHondt (m: $seq\langle seq\langle\mathbb{R}\rangle\rangle$) {
$(\forall i, j : \mathbb{Z})(((0 \leq i \leq |escrutinio| - 1) \wedge (0 \leq j \leq |cant\_bancas| - 1)) \longrightarrow_L res[i][j] = res[0][j]/j)$
}

proc sacarUltimo (in s $seq\langle T\rangle$) : $seq\langle T\rangle$
requiere $\{|s| > 0\}$

asegura $\{(|res| = |s| - 1) \wedge$ $((\forall$ i : $\mathbb{Z})$(0 $\leq$ i $\leq$ $|res|)$ $\longrightarrow_L res[i] = s[i])\}$

5. Ejercicio 5: obtenerDiputadosEnProvincia

5.1. Especificación

proc obtenerDiputadosEnProvincia (in cant_bancas : $\mathbb{Z}$, in escrutinio: $seq\langle\mathbb{Z}\rangle$, in dHondt : seq$\langle seq\langle\mathbb{Z}\rangle\rangle$) : $seq\langle\mathbb{Z}\rangle$

requiere $\{esDHondtDiputados(dHondt)\}$

asegura $\{res[i] = \sum_{j=0}^{|dHondt[0]|-1}$ if((mayoresAij(dHondt,i,j) <cant_bancas)) then 1 else 0 fi $\}$

5.2. Auxiliares

aux mayoresAij (M: $seq\langle seq\langle\mathbb{Z}\rangle, i : \mathbb{Z}, j : \mathbb{Z}\rangle$) : $\mathbb{Z}$ = $\sum_{k=0}^{|M|-1} \sum_{l=0}^{|M[0]|-1}$ if (M[k,l] >M[i,j]) then 1 else 0 fi;

pred esDHondtDiputados (m: $seq\langle seq\langle\mathbb{R}\rangle\rangle$) {

$(\forall i, j : \mathbb{Z})(((0 \leq i \leq |umbralElectoral(escrutinio)|-1) \wedge (0 \leq j \leq |cant\_bancas|-1)) \longrightarrow_L res[i][j] = res[0][j]/j)$

}

proc umbralElectoral (in escrutinio: $seq\langle\mathbb{Z}\rangle$) : $seq\langle\mathbb{Z}\rangle$

requiere $\{True\}$

asegura $\{|res| \leq |escrutinio|\}$

asegura $\{(\forall i : \mathbb{Z})(0 \leq i \leq |escrutinio| - 1_L escrutinio[i] \in res \iff escrutinio[i] > 0{,}3 * totalVotos(escrutinio)\}$

6. Ejercicio 6: validarListasDiputadosEnProvincia

6.1. Especificación

proc validarListasDiputadosEnProvincia (in cant_bancas : $\mathbb{Z}$, in listas : seq$\langle seq\langle dni : \mathbb{Z} \times género : \mathbb{Z}\rangle\rangle$) : $Bool$

requiere $\{cant\_bancas \geq 0\}$

asegura $\{(\forall i : \mathbb{Z})(0 \leq i \leq |listas| - 1) \rightarrow_L ((|listas[i]| = cant\_bancas) \wedge (alternanciaDeGénero(listas[i])) \wedge (distintosDNI(listas[i])\}$

6.2. Auxiliares

pred alternanciaDeGénero (in listita : seq$\langle dni : \mathbb{Z} \times género : \mathbb{Z}\rangle$){

$((\forall i : \mathbb{Z})(1 \leq i \leq |listita|-2) \rightarrow_L (listita[i][1] = 1 \iff (listita[i+1][1] = 2) \wedge (listita[i-1][1] = 2))) \wedge (\forall i : \mathbb{Z})(1 \leq i \leq |listita| - 2) \rightarrow_L (listita[i][1] = 2 \iff (listita[i+1][1] = 1) \wedge (listita[i-1][1] = 1))$

}

pred distintosDNI (in listita : seq$\langle dni : \mathbb{Z} \times género : \mathbb{Z}\rangle$){

$(\forall i, j : \mathbb{Z})((0 \leq i, j \leq |listita| - 1) \wedge (i \neq j)) \rightarrow_L (listita[i][0] \neq listita[j][0])$

```
}
```

6.3. Algoritmo

```
alternanciaDeGenero(listita){
bool res:= true;
if(listita[i][1] == 1){
    for(int i=1; int i<listita.length - 1, i++){
       if(listita[i][1] mod 2==1){
           res := res && (listita[i][1] ==2);
       }else {
           res := res && (listita[i][1] ==1);
       }
    }
 }else if(listita[i][1] ==2){
        for(int i=1; int i<listita.length - 1, i++){
           if(listita[i][1] mod 2==1){
               res := res && (listita[i][1] == 1);
           }else {
               res := res && (listita[i][1] == 2);
           }
       }

 }else{
    res:= !res;
 }
return res;
}


distintosDNI(listita){
bool res:= true;
for(int i=0; i<listita.length; i++){
   for(int j=i+1; j<listita.length; j++){
     res:= res &&(listita[i][0] != listita[j][0]);
   }
 }
return res;
}

validarListasDiputadosEnProvincia(cant_bancas,listas){
bool res:= true;
for(int i= 0; i<listas.length;i++){
   res:= distintosDNI(listas[i]) && alternanciaDeGenero(listas[i]) &&
           ((listas[i].length)==cant_bancas);

 }
}
```