UBA – Facultad de Ciencias Exactas y Naturales – Departamento de Computación– Algoritmos y Estructura de Datos I

Primer parcial – 25/04/2016

Nro. de orden: 9
LU: 526/15
Apellidos: FREUND
Nombres: TEODORO

1		2	3	4	5		TOTAL
a	b				a	b	
15		20	20	24	15		94

(A)

Aclaraciones: El parcial NO es a libro abierto. Cualquier decisión de interpretación que se tome debe ser aclarada y justificada. Para aprobar se requieren al menos 60 puntos. **Entregar cada ejercicio en hoja separada.**
Importante: Para la resolución del parcial **NO** es necesario ni está permitido el uso de **acum**. En caso de ser necesario, pueden asumir como definida la función *sum* ($\sum$) sobre listas. Las funciones *min* o *max* sobre listas, en caso de requerirlas, deben ser definidas.

```
tipo Nombre=String;
tipo Bloque=String;
tipo Provincia = Buenos Aires, Catamarca, Chaco,... ;
tipo Comision = Cultura, Educacion, Finanzas,... ;
tipo TipoMayoria = Simple, Absoluta ;

tipo Diputado {
        observador nombre (d: Diputado) : Nombre ;
        observador bloque (d: Diputado) : Bloque ;
        observador distrito (d: Diputado) : Provincia ;
        observador comisiones (d: Diputado) : [Comision] ;
        invariante comisionesOrdenadas : ordenada(comisiones(d)) ;
}

tipo Proyecto {
        observador numero (p: Proyecto) : Z ;
        observador descripcion (p: Proyecto) : String ;
        observador autores (p: Proyecto) : [Diputado] ;
        observador tratadoEn (p: Proyecto) : Comision ;
        observador aprobacion (p: Proyecto) : TipoMayoria ;
        invariante autoresOrdenados : ordenada(nombres(autores(p))) ;
        invariante autoresEnComision : ... ;
}
```

```
tipo Sesion {
        observador diputados (s: Sesion) : [Diputado] ;
        observador ordenDelDia (s: Sesion) : [Proyecto] ;
        observador presente (s: Sesion, n: Nombre) : Bool ;
            requiere n ∈ nombres(diputados(s)) ;
        observador voto (s: Sesion, nroP: Z, n: Nombre) : Bool ;
            requiere nroP ∈ numeros(ordenDelDia(s)) ∧
                n ∈ nombres(diputados(s)) ∧ presente(s, n) ;
        invariante diputadosDistintos :
            sinRepetidos(nombres(diputados(s))) ;
        invariante proyectosDistintos :
            sinRepetidos(numeros(ordenDelDia(s))) ;
        invariante camaraBienConstituida : ... ;
}

aux numeros (ps: [Proyecto]) : [Z] = [numero(p) | p ← ps] ;
aux nombres (ds: [Diputado]) : [Nombre] = [nombre(d) | d ← ds] ;
aux sinRepetidos (xs: [T]) : Bool = (∀i, j ← [0..|xs|), i ≠ j)xs_i ≠ xs_j ;
aux ordenada (xs: [T]) : Bool = (∀i ← [0..|xs| − 1))xs_i ≤ xs_{i+1} ;
```

Ejercicio 1. [15 puntos]

a) Completar el `invariante` autoresEnComision del tipo Proyecto, que indica que el proyecto tiene al menos un autor, y que todos los diputados que escribieron el proyecto forman parte de la comisión en la cual se trata dicho proyecto.

b) Completar el `invariante` camaraBienConstituida del tipo Sesion, que expresa que hay al menos un diputado de cada uno de los 24 distritos.

Ejercicio 2. [20 puntos] Especificar el `problema elMasTraidor` (s: Sesion) = `result` : Nombre
que devuelve el nombre de alguno de los diputados que más proyectos traicionó. Diremos que un diputado *traiciona* un proyecto cuando es uno de los autores del mismo pero lo vota de forma negativa.

Ejercicio 3. [20 puntos] Especificar el `problema losRechazados` (ss: [Sesion]) = `result` : [Z]
que devuelve una lista con los números de los proyectos no aprobados en alguna sesión de *ss*. La lista resultante debe estar ordenada de manera creciente y no contener repetidos. Considerar que en distintas sesiones no pueden tratarse proyectos diferentes que tengan el mismo número. La aprobación de un proyecto depende del tipo de mayoría:

- Si requiere mayoría simple, debe tener al menos la mitad de los votos de los diputados presentes.
- Si requiere mayoría absoluta, la cantidad de votos debe ser mayor a la mitad de la cantidad total de diputados.

Aclaración: Si un proyecto es tratado en más de una sesión de *ss*, y en algunas se aprueba y en otras no, el proyecto debe estar en la lista devuelta como resultado del problema.

Ejercicio 4. [30 puntos] Especificar el `problema renunciarAComision` (s: Sesion, d: Diputado, c: Comision) , que modifica la sesión de forma tal que el diputado *d* ya no forme parte de la comisión *c*.

Ejercicio 5. [15 puntos] Dado el siguiente programa:

```
int restarONo(int a, int b){
  int result = 0;
  if(a > b){
    result = a - b;
  }
  return result;
}
```

a) Dar la trasformación de estados del programa presentado.

b) Decidir si el programa es correcto para las siguientes especificaciones. En caso afirmativo, demostrar. En caso contrario, justificar por qué no es correcto.

```
problema restarONo (a,b: Z) = res : Z {
    requiere a == b ;
    asegura res == 0 ;
}
```

```
problema restarONo (a,b: Z) = res : Z {
    asegura a > b ⇒ res == a − b ;
    asegura a ≤ b ⇒ res == 0 ;
}
```

```
problema restarONo (a,b: Z) = res : Z {
    asegura res ≥ 0 ;
}
```

Teodoro Freund 526/15

Orden 9

HOJA 1/6

FECHA

(1) a.

invariante autoresEnComision: |autores(p)| > 0 ∧
(∀ d ← autores(p)) tratado En(p) ∈ comisiones(d);

b.

invariante camaraBienConstituida:

(∀ i ← [0..24))(∃ d ← diputados(s)) ord(distrito(d)) == i;

NOTA

ORDEN

(2)

problema el MasTraidor (s: Sesion) = result: Nombre {

~~Requiere {diputados(s) ≠ ∅}~~

asegura result ∈ nombres(diputados(s));
asegura (∀ d ← diputados(s)) Traiciones(s, d) ≤
Traiciones(s, DipCNombre(s, result));

aux Traiciones(s: Sesion, d: Diputado): ℤ =
|[20 | p ← ordenDelDiz(s), d ∈ autores(p),
presente(s, nombre(d)), ¬voto(s, numero(p), nombre(d))]|;

aux DipCNombre(s: Sesion, n: Nombre): Diputado =
cab([d | d ← diputados(s), n == nombre(d)]);

}

(SE UTILIZA DipCNombre YA QUE EN EL TIPO PROYECTO PUEDE HABER ALGUN DIPUTADO QUE SE LLAME IGUAL QUE OTRO DIP. DISTINTO, QUE FORME PARTE DE LA SESION)

NOTA

(3)

```
Proble~a losRechazados (ss:[Sesion]):result:[ℤ] {
  requiere numerosUnicos: (∀s,s'←ss, p←ordenDelDia(s),
    p'←ordenDelDia(s')) numero(p)==numero(p') ⟹ p==p';
    ~~Igualdad Proyectos(p,p')~~;
  asegura ~~mismos~~(result, ProyRech(ss));
  asegura ordenada(result);
  aux BorrarRep(xs:[T]):[T]=[xs[i] | i←[0..|xs|), xs[i]∉xs[0..i)];
  aux ProyRech(ss:[Sesion]):[ℤ]=
      BorrarRep([numero(p) | s←ss, p←ordenDelDia(s),
                 Rechazado(s,p)]);
  aux Rechazado(s:Sesion, p:Proyecto):Bool=
      (aprobacion(p)==Simple ∧ DipPresentes(s)>~~Empates~~
                 2*Votos(p,s,DipPresentes(s)))
    ∨ (aprobacion(p)==Absoluta ∧ diputados(s)≥
                 2*Votos(p,s,DipPresentes(s)));
  aux DipPresentes(s:Sesion):[Diputado]=
      [d | d←diputados(s), presente(s,nombre(d))];
  aux Votos(p:Proyecto, s:Sesion, ds:[Diputado]):ℤ=
      |[2|d←ds, voto(s,numero(p),nombre(d))]|;
}
```

(4)

```
problema renunciarAComision (s: Sesion, d: Diputado, c: Comision) {
    requiere (∀p ← ordenDelDia(s), tratadoEn(p)==c, d ∈ autores(p))
                                                    |autores(p)|>1);
    modifica s;
    asegura |diputados(s)| == |diputados(pre(s))|;
    asegura |ordenDelDia(s)| == |ordenDelDia(pre(s))|;
    asegura (∀d' ← diputados(pre(s)), d' ≠ d)
                (∃d'' ← diputados(s)) d' == d'';
    asegura (∀d' ← diputados(pre(s)), d' == d) (∃d'' ← diputados(s))
                nombre(d') == nombre(d'') ∧
                bloque(d') == bloque(d'') ∧
                distrito(d') == distrito(d'') ∧
                mismos(comisiones(d'), c: comisiones(d''));
    asegura (∀p ← ordenDelDia(pre(s)), tratadoEn(p) ≠ c ∨
                                                    d ∉ autores(p))
                (∃p' ← ordenDelDia(s)) p == p';
```

(SIGUE ATRÁS) ¿Si el proyecto se trató en otra comisión pero p era autor?

NOTA

asegura (∀ p ← ordenDelDia(pre(s)), tratadoEn(p) == c ⇒ d ∈ autores(p))

(∃ p' ← ordenDelDia(s))
numero(p) == numero(p') ∧
descripcion(p) == descripcion(p') ∧
tratadoEn(p') == c ∧
aprobacion(p) == aprobacion(p') ∧
mismos(autores(p), d: autores(p'))j ✓

asegura (∀ n ← nombres(diputados(s)))
presente(s, n) == presente(pre(s), n)j ✓

~~asegura (∀ n ← nombres(s), [illegible])~~

asegura (∀ n ← nombres(diputados(s)); [illegible]
np ← numeros(ordenDelDia(s)), presente(s, n))
voto(s, np, n) == voto(pre(s), np, n)j ✓

}

NOTA

⑤ a.

```
int restarONo (int a, int b) {
    int result=0;
    // E1;
    // vale result==0;  ✓ (a y b no cumplen)  ✓
    // Pif: result==0;
    // implica Pif;
    if (a>b){
        // Eif0;
        // vale Pif ∧ B;  ✓
        // implica result==0 ∧ a>b;
        result = a-b;
        // Eif1;
        // vale result==a-b ∧ a>b;   implica
        // implica (result==a-b ∧ a>b) v (result==0 ∧ a≤b);  → P=>(PvQ)
        // implica Qif;  → SINTACTICAMENTE IGUALES  ✓
    }
    // E2;
    // vale Qif: (result==a-b ∧ a>b) v (result==0 ∧ a≤b);  ✓

    return result;
    // E3;
    // vale res==result@E2 ∧ Qif==Qif@E2;
    // implica (res==a-b ∧ a>b) v (res==0 ∧ a≤b);
}
```

(SIGUE ATRÁS)

NOTA

RAMA FALSE

// E e 0;
// vale P_if ∧ ¬B;
// implica result == 0 ∧ a ≤ b;
// implica (result == a-b ∧ a > b) ∨ (result == 0 ∧ a ≤ b)};
// implica Q_if; ~> SINTACTICAMENTE IGUALES

↳ q ⇒ p ∨ q
SI p NO SE INDEFINE
~~FALSE~~ (ESTE CASO)

↳ NOTAR, PARA TODAS LAS SUSTITUCIONES, QUE ESTE PROGRAMA SIEMPRE TERMINA

5. problema restar0N0 (a, b : ℤ) = res : ℤ {
    requiere a == b;
    asegura res == 0;
}

ES CORRECTO YA QUE SI a == b, como AL FINAL DEL PROGRAMA VALE QUE (res == a-b ∧ a > b) ∨ (res == 0 ∧ a ≤ b)
Y como a > b no vale, DEBE SER res == 0, CUMPLIENDO EL ASEGURA

problema restar0N0 (a, b : ℤ) = res : ℤ {
    asegura a > b ⇒ res == a-b;
    asegura a ≤ b ⇒ res == 0;
}

ES CORRECTO, PORQUE:
SEAN p = a > b, q = res == a-b, q' ~~a-b~~ res == 0
ENTONCES, POR DEFINICIONES DE LAS OPERACIONES LÓGICAS

NOTA

$(P \Rightarrow q) \wedge (\neg P \Rightarrow q') \iff (\neg P \vee q) \wedge (P \vee q') \iff$

$(\neg P \wedge P) \vee (\neg P \wedge q') \vee (q \wedge P) \vee (q \wedge q') \iff$

↳ como $(\neg P \wedge P)$ y $(q \wedge q')$ son siempre FALSAS ↳ no siempre; pues es cierto por tres de verdad si no es indefinido

$(\neg P \wedge q') \vee (q \wedge P) \iff$

$(a \leq b \wedge res == 0) \vee (res == a - b \wedge a > b) \iff$

↳ como ninguna se indefine pudo cambiar el orden ✓

$(res == a - b \wedge a > b) \vee (res == 0 \wedge a \leq b)$

↳ es decir que el problema asegura lo mismo que el programa cumple al finalizar ✓

```
problema restar0N0 (a, b : Z) = res : Z {
    asegura res >= 0 ;
}
```

notar que

$res == 0 \wedge a \leq b \Rightarrow res \geq 0$ y

$res == a - b \wedge a > b \Rightarrow res \geq 0$

~~[illegible]~~. entonces

$(res == 0 \wedge a \leq b) \vee (res == a - b \wedge a > b) \Rightarrow res \geq 0$ ✓

↳ por la regla de la eliminación del ✓

entonces, el programa siempre devuelve un $res \geq 0$, lo que cumple el asegura del problema ✓

NOTA