Practica 8

Agustin Stescovich Curi

1

(Funcion abstraccion) Necesito que todas las ventas registradas en `ventasPorProducto` aparezcan en `ventas`, que por cada producto la suma de los montos de todas sus ventas se equivalente al valor asociado a dicho producto en `totalPorProducto` y que la venta(monto) mas caro de cada producto en `ventasPorProducto` sea el valor asociado a dicho producto en `ultimoPrecio`.
(invariante de representacion) Para cada producto la suma de todas las ventas debe ser equivalente a `totalPorProducto`[producto] y la mayor de ellas debe ser equivalente a `ultimoPrecio`[producto].

$\texttt{tup} = tupla\langle producto, fecha, monto\rangle$

`pred` invRep (c:comercioImpl)$\{(\forall p : producto)(pertenece(p, c.ventas) \leftrightarrow (\exists s : seq\langle\texttt{tup}\rangle((\forall t : \texttt{tup})(t_1 = p \wedge t \in s \leftrightarrow t \in c.ventas) \wedge \sum_{i=0}^{|s|-1} s[i]_2 = c.totalPorProducto[p] \wedge (\exists m : \texttt{tup})(esMax(m, s) \wedge t_2 = ultimoPrecio[p]))))\}$

`pred` pertenece $(p : producto, v : seq\langle\texttt{tup}\rangle)\{(\exists t : \texttt{tup})(t \in v \wedge t_0 = p)\}$

`pred` esMax $(t : \texttt{tup}, v : seq\langle\texttt{tup}\rangle)\{t \in v \wedge (\forall t' : \texttt{tup})(t' \in v \rightarrow t'_2 \leq t_2)\}$

`pred` abs (c:comercioImpl,c':com)$\{(\forall p : producto)(p \in c'.vpp \wedge pertenece(p, c.ventas) \leftrightarrow (\forall t : \langle fecha, monto\rangle))(t \in c'.vpp[p] \leftrightarrow \langle p, t_0, t_1\rangle \in c.ventas \wedge (\exists t' : \langle fecha, monto\rangle))(t' \in c'.vpp[p] \wedge t'_0 \geq t_0 \wedge t'_1 = c.ultimoPrecio[p])) \wedge \sum_{i=0}^{|c'.vpp[p]|-1} c'.vpp[p][i][1] = c.totalPorProducto[p])\}$

2

2.1 Invrep(castellano)

Necesito que para cada alarma(key) en planta.alarmas, todos los sensores pertenecientes al conjunto asociado a la alarma sean claves en planta.sensores y que la alarma pertenezca al conjunto asociado a cada sensor.

2.2 Invariante de Representacion

`pred` invRep (p:plantaImpl){$(\forall a : alarma)(a \in p.alarmas \rightarrow (\forall s : sensor)(s \in p.alarmas[a] \rightarrow s \in p.sensores \wedge a \in p.sensores[s]))$}

2.3 Funcion Abstraccion

`pred` abs (p:plantaImpl, p':planta){$(\forall s : sensor, a : alarma)(a \in p'.alarmas \wedge < s, a > \in p'.sensores \leftrightarrow s \in p.alarmas[a] \wedge a \in p.alarmas[s])$}

3

3.1 Invrep(castellano)

Necesito: i)si un estudiante pertenece a `estudiantes` entonces pertence tambien a `faltas`, `notas` y `notasPorEstudiante` ii) La cantaidad de faltas por estudiante debe ser un numero mayor o igual a 0 iii) Si un estudiante pertenece al i-esimo conjunto en `notas` entonces la i-esima posicion en el *array* de la clave de dicho estudiante en `notasPorEstudiante` es mayor a 0. iv) `estudiantes`,`faltas` y `notasPorEstudiante` tienen el mismo tamaño siempre v) El tamaño de `notas` es 10 vi) el tamaño de todos los elementos de `notas` es el mismo que el de `estudiantes`

3.2 Invariante de representacion

`pred` invRep (s:secundarioImpl){$(\forall e : estudiante)(e \in s.estudiantes \rightarrow e \in s.faltas \wedge e \in s.notas \wedge e \in s.notasPorEstudiante \wedge s.faltas[e] \geq 0 \wedge (\forall i : \mathbb{Z})(0 \leq i \leq 10 \wedge e \in s.notas[i] \leftrightarrow s.notasPorEstudiante[e][i] > 0)) \wedge s.estudiantes.length = s.faltas.length = s.notasPorEstudiante.length \wedge s.notas.lenght = 10 \wedge (\forall j : \mathbb{Z})(0 \leq j \leq 10 \rightarrow s.notas[i].lenght = s.estudiantes.length)$}

3.3 Funcion abstraccion

`pred` abs (s:secundarioImpl,s':secundario){$(\forall e : estudiante)(e \in s'.estudiantes \leftrightarrow e \in s.estudiantes) \wedge (\forall e' : estudiantes, i : \mathbb{Z})(e' \in s'.faltas \wedge s'.faltas[e'] = i \leftrightarrow e' \in s.faltas \wedge s.faltas[e'] = i) \wedge (\forall e" : estudiantes, n : \mathbb{Z})(0 \leq n \leq 10 \wedge e" \in s'.notas \wedge n \in s'.notas[e"] \leftrightarrow e" \in s.notas[n] \wedge e" \in s.notasPorEstudiante \wedge s.notasPorEstudiante[e"][n] > 0)$}

4

HACER

5

5.1 Invariante de representacion

5.2 Funcion abstraccion

5.3 Modulo

```
modulo MIB implementa Matriz infinita de booleanos {
    var data : vector⟨vector⟨boolean⟩⟩
    var inv : vector⟨vector⟨boolean⟩⟩
    var vacio : boolean

    proc crear (): MIB {
        res := new MIB()
        res.data.vectorVacio()
        res.inv.vectorVacio()
        res.vacio := False
        return res
    }

    proc agregar (inout m : MIB, in f, c : ℤ, in b : boolean):  {
        act := m.data.longitud()
        if (act < f) then
            while (act ≤ f) do
                fil.vectorVacio()
                act++
                m.data.agregarAtras(fil)
                m.inv.agregarAtras(fil)
            endwhile
        endif
        act := m.data.obtener(f).longitud()
        if (act < c) then
            while (act ≤ c) do
                col=False :=
            act++
                m.data.agregarAtras(col)
                m.inv.agregarAtras(col)
            endwhile
        endif
        modificarPosicion(m.data.obtener(f),c,b)
        modificarPosicion(m.inv.obtener(f),c,!b)
    }

    proc ver (in m : MIB, in f, c : ℤ): boolean {
        if (m.data.longitud() > f) then
            return m.vacio
        elseIf (m.data.obtener(f).longitud() > c) then
            return m.vacio
```

```
        else
            return m.obtener(f).obtener(c)
        endif
    }

    proc complementar (inout m : MIB):  {
        copiaD := new vector⟨vector⟨boolean⟩⟩
        copiaI := new vector⟨vector⟨boolean⟩⟩
        copiaVacio := new boolean
        copiaD := m.data
        copiaI := m.inv
        copiaVacio := m.vacio
        m.data := copiaI
        m.inv := copiaD
        m.vacio := copiaVacio
    }

}
```

5.4 Complejidades

- $crear \in O(1)$
- $agregar \in O(max(f, c))$ [f y c son las entradas para la fila y columna]
- $ver \in O(1)$
- $complementar \in O(1)$

6 Hacer

7

Vagon es `String`
Tren es $listaEnlazada\langle Vagon\rangle$

```
modulo PDM implementa playaDeManiobras {
    var trenes : array⟨listaEnlazada⟨Vagon⟩⟩

    proc abrirPlaya (capacidad : ℤ): PDM {
        trenes := new array⟨listaEnlazada⟨Vagon⟩⟩(capacidad)
        res.trenes := trenes
        return res
    }

    proc recibirTren (p : PDM, t : Tren): ℤ {
        i := 0
        while (i¡p.trenes.length() & p.trenes[i]!=null) do
            i++
        endwhile
```

```
        p.trenes[i] := t
        return i
    }

    proc despacharTren (p : PDM, v : ℤ):  {
        p.trenes[v] := null
    }

    proc unirTrenes (p : PDM, v1 : ℤ, v2 : ℤ):  {
        via1 := p.trenes[v1]
        via2 := p.trenes[v2]
        via1.ultimo.siguiente := via2.primero
        via2.primero.anterior := via1.ultimo
        via1.ultimo := via2.ultimo
    }

    proc moverVagon (p : PDM, v : vagon, origen : ℤ, destino : ℤ):  {
        it := p.trenes[origen].iterador()
        actual := null
        while (it.haySiguiente() & act.val!=v) do
            actual := it.siguiente()
        endwhile
        if (p.trenes[origen].longitud=1) then
            p.trenes[origen].cabeza := null
            p.trenes[origen].ultimo := null
        elseIf (actual=p.trenes[origen].cabeza) then
            primero := p.trenes[origen].cabeza
            primero := actual.siguiente
            primero.anterior := null
        elseIf (actual=p.trenes[origen].ultimo) then
            ultimo := p.trenes[origen].ultimo
            ultimo := actual.anterior
            ultimo.siguiente := null
        else
            ant := actual.anterior
            sig := actual.siguiente
            ant.siguiente := sig
            sig.anterior := ant
        p.trenes[destino].agregarAtras(v)
    }

}
```

7.1 Complejidades

- $abrirPlaya \in O(1)$
- $recibirTren \in O(v)$

- $despacharTren \in O(1)$
- $unirTrenes \in O(1)$
- $moverVagon \in O(t)$

7.2 Invariante de Representacion

`pred` invRep (p:PDM){$(\forall t : tren)(t \in p.trenes \leftrightarrow (\exists i : \mathbb{Z})(0 \leq i \leq p.trenes.length() \wedge_L p.trenes[i] = t))$}

7.3 Funcion Abstraccion

`pred` abs (p:PDM, p':playaDeManiobras){$p.trenes.length() = p'.trenes.longitud() \wedge (\forall v : Vagon)(v \in p.trenes \wedge v \in p'.trenes \leftrightarrow (\exists via : \mathbb{Z})(0 \leq via \leq p'.trenes.longitud() \wedge_L (\exists vag : \mathbb{Z})(0 \leq vag \leq p'.trenes[vag].longitud() \wedge_L p.trenes[via][vag] = p'.trenes[via][vag] = v)))$}

8

8.1 Modulo

```
modulo IB implementa ingresosAlBanco {
    var data : vector⟨ℤ⟩
    var totalHasta : vector⟨ℤ⟩

    proc nuevoIngresos (): IB {
        var res : IB
        res.data := vectorVacio()
        res.totalHasta := vectorVacio()
        return res
    }

    proc registrarNuevoDia (i : IB, cant : ℤ):  {
        agregarAtras(i.data,cant)
        if (longitud(i.totalHasta)=0) then
            agregarAtras(i.totalHasta,cant)
        else
            ult := ultimo(i.totalHasta)
            agregarAtras(i.totalHasta,ult+cant)
    }

    proc cantDias (i : IB): ℤ {
        return longitud(i.data)
    }

    proc cantPersonas (i : IB, desde : ℤ, hasta : ℤ): ℤ {
        return res.totalHasta[hasta]-res.totalHasta[desde]+res.data[desde]
    }
```

requiere que la longitud sea mayor a 1

```
proc mediana (i : IB): ℤ {
    d := 0
    while (0 ≤ d < long(i.data) − 1 & cantPersonas(i, 0, d) ≤ cantPersonas(i, d + 1, long(i.data)))
do
        i++
    endwhile
    return d-1
}
```

8.2

El sistema crece en funcion de la cantidad de dias que pasaron, de manera que el tamaño crecera a n cuando estemos en el n-dia

8.3 Invariante de Representacion

pred invRep (i:IB){$i.data.longitud() = i.totalHasta.longitud() \wedge (\forall d : \mathbb{Z})(0 \leq d \leq i.data.longitud() \rightarrow_L tot(i, d) = i.totalHasta[d])$}
aux tot $(i : IB, d : dia) = \sum_{j=0}^{hasta} i.data[j]$

8.4 Funcion Abstraccion

pred abs (i:IB, i':ingresosAlBanco){$i.data.longitud() = i'.totales.longitud() \wedge (\forall dia : \mathbb{Z})(0 \leq dia < i.data.longitud() \rightarrow_L i.data[dia] = i'.totales[dia])$}
}

9

9.1 Modulo

Cliente es $\mathbb{Z}$

```
modulo madereraImpl implementa Maderera {
    var deposito : colaPrioridadLog⟨ℤ⟩
    var ventas : DiccionarioDigital⟨Cliente, vector⟨fecha, tamano⟩⟩

    proc comprarUnListon (m : madereraImpl, tam : ℤ):  {
        encolar(m.deposito,t,t)
    }

    proc venderUnListon (m : madereraImpl, tam : ℤ, c : Cliente, f : fecha):  {
        liston := desencolarMax(m.deposito)
        prior := liston-t
        encolar(m.deposito,prior,prior)
        ven := obtener(m.ventas,c)
        agregarAtras(ven, ⟨f, t⟩)
    }
```

```
    proc ventasACliente (m : madereraImpl, c : Cliente): vector⟨fecha, tamano⟩ {
        return obtener(m.ventas,c)
    }
}
```