UBA – Facultad de Ciencias Exactas y Naturales – Departamento de Computación

Algoritmos y Estructura de Datos I

Segundo cuatrimestre de 2016

17 de octubre de 2016

TPI - Demostración de correctitud - función JJOO::atletaProdigio()

Índice

1. Código

```
Atleta JJOO::atletaProdigio() const {
        vector <Competencia> compsOro = competenciasFinalizadasConOroEnPodio();
        Atleta res = compsOro[0].ranking()[0];
        int mayorAnio = res.anioNacimiento();

        unsigned int i = 0;

        while (i<compsOro.size()) {
            if (compsOro[i].ranking[0].anioNacimiento() > mayorAnio) {
                res = compsOro[i].ranking()[0];
                mayorAnio = compsOro[i].ranking()[0].anioNacimiento();
            }
            i++;
        }

        return res;
}
```

2. Transformación de estados del programa

Definimos

$P_C : i == 0 \land algunaVezSeCompitio \land res == ranking(comps_0)_0 \land mayorAnio == anioNacimiento(res)$

invariante $I : 0 \leq i \leq |compsOro| \land algunaVezSeCompitio \land esCampeon(res, j) \land mayorAnio == anioNacimiento(res) \land (\forall c \in compsOro[0..i))anioNacimiento(campeon(c)) \leq anioNacimiento(res)$

$Q_C : esCampeon(res, j) \land (\forall c \in compsOro)anioNacimiento(campeon(c)) \leq anioNacimiento(res)$

```
Atleta JJOO::atletaProdigio() const {
```

//estado E_0;
//vale $algunaVezSeCompitio : |competenciasConOroEnPodio(this)| > 0$;
por `requiere`

```
    vector <Competencia> compsOro = competenciasFinalizadasConOroEnPodio();
```

//estado E_1;
//vale $algunaVezSeCompitio \land compsOro == competenciasConOroEnPodio(this)$;

```
    Atleta res = compsOro[0].ranking()[0];
```

//estado E_2;
//vale $algunaVezSeCompitio \land compsOro == compsOro@E_1 \land res == compsOro[0].ranking()[0]$;

```
    int mayorAnio = res.anioNacimiento();
```

//estado E_3;
//vale $algunaVezSeCompitio \land compsOro == compsOro@E_2 \land res == res@E_2 \land mayorAnio == res.anioNacimiento()$;

```
    unsigned int i = 0;
```

//estado E_4;
//vale $algunaVezSeCompitio \land compsOro == compsOro@E_3 \land res == res@E_3 \land mayorAnio == mayorAnio@E_3 \land i == 0$;
//implica $algunaVezSeCompitio \land compsOro == compsOro@E_1 \land res == res@E_2 \land mayorAnio == res.anioNacimiento() \land i == 0$;
por estados anteriores

//implica $algunaVezSeCompitio \land compsOro == competenciasConOroEnPodio(this) \land res == res@E_2 \land mayorAnio == anioNacimiento(res) \land i == 0$;
por especificación de los problemas *competenciasFinalizadasConOroEnPodio* y *anioNacimiento*

//implica $P_C : i == 0 \land algunaVezSeCompitio \land res == ranking(comps_0)_0 \land mayorAnio == anioNacimiento(res)$
//implica $I : 0 \leq i \leq |compsOro| \land algunaVezSeCompitio \land esCampeon(res, j) \land mayorAnio == anioNacimiento(res) \land (\forall c \in compsOro[0..i))anioNacimiento(campeon(c)) \leq anioNacimiento(res)$

```
    while (i<compsOro.size()) {
        if (compsOro[i].ranking[0].anioNacimiento() > mayorAnio) {
            res = compsOro[i].ranking()[0];
            mayorAnio = compsOro[i].ranking()[0].anioNacimiento();
        }
        i++;
    }
```

//estado E_5;
//vale $Q_C : esCampeon(res, j) \land (\forall c \in compsOro)anioNacimiento(campeon(c)) \leq anioNacimiento(res)$;
//implica $esCampeon(res, j) \land (\forall c \in competenciasConOroEnPodio(j))anioNacimiento(campeon(c)) \leq anioNacimiento(res)$;

```
    return res;
}
```

Como el último estado es equivalente a la cláusula `asegura` de la especificación y `res` es el valor de retorno, el programa es correcto respecto de la especificación.

3. Correctitud del ciclo

3.1. $P_c \rightarrow I$

$P_C : i == 0 \land algunaVezSeCompitio \land res == ranking(comps_0)_0 \land mayorAnio == anioNacimiento(res)$

$I : 0 \leq i \leq |compsOro| \land esCampeon(res, j) \land mayorAnio == anioNacimiento(res) \land$
$(\forall c \in compsOro[0..i))anioNacimiento(campeon(c)) \leq anioNacimiento(res)$

1. $mayorAnio == anioNacimiento(res)$: trivial
2. $i == 0 \rightarrow 0 \leq i$
3. $algunaVezSeCompitio \rightarrow |compsOro| > 0 \rightarrow 0 == i < |compsOro|$
   (como $competenciasFinalizadasConOroEnPodio$ es correcta respecto de su especificación, $compsOro == competenciasConOroEnPodio(j)$)
4. $res == ranking(comps_0)_0 \rightarrow (\exists c \in competenciasConOroEnPodio(c))res == ranking(c)_0 \rightarrow esCampeon(res, j)$
5. $i == 0 \rightarrow compsOro[0..i)$ es una lista vacía y cualquier predicado sobre todos sus elementos es verdadero.

3.2. $(I \land \neg B) \rightarrow Q_C$

$(I\ and \neg B) : 0 \leq i \leq |compsOro| \land esCampeon(res, j) \land mayorAnio == anioNacimiento(res) \land$
$(\forall c \in compsOro[0..i))anioNacimiento(campeon(c)) \leq anioNacimiento(res) \land i \geq |compsOro|$

$Q_C : esCampeon(res, j) \land (\forall c \in compsOro)anioNacimiento(campeon(c)) \leq anioNacimiento(res)$

1. $esCampeon(res, j)$: trivial
2. $(i \leq |compsOro| \land i \geq |compsOro|) \rightarrow i == |compsOro|$
3. $i == |compsOro| \rightarrow compsOro[0..i) == compsOro$
4. $(((\forall c \in compsOro[0..i))anioNacimiento(campeon(c)) \leq anioNacimiento(res)) \land compsOro[0..i) == compsOro) \rightarrow$
   $(\forall c \in compsOro)anioNacimiento(campeon(c)) \leq anioNacimiento(res)$

3.3. Preservación de I

Sea $Pif = I \land B : 0 \leq i < |compsOro| \land esCampeon(res, j) \land mayorAnio == anioNacimiento(res) \land$
$(\forall c \in compsOro[0..i))anioNacimiento(campeon(c)) \leq anioNacimiento(res)$

```
        while (i<compsOro.size()) {
//estado C_0;
//vale I ∧ B;
//implica P_if;
                if (compsOro[i].ranking[0].anioNacimiento() > mayorAnio) {
                        res = compsOro[i].ranking()[0];
                        mayorAnio = compsOro[i].ranking()[0].anioNacimiento();
                }
//estado C_1;
//vale Q_if : 0 ≤ i < |compsOro| ∧ esCampeon(res, j) ∧ mayorAnio == anioNacimiento(res) ∧
//(∀c ∈ compsOro[0..i])anioNacimiento(campeon(c)) ≤ anioNacimiento(res);
//vale i == i@C_0;
                i++;
//estado C_2;
//vale i == i@C_1 + 1 ∧ 0 ≤ i@C_1 < |compsOro| ∧ esCampeon(res, j) ∧ mayorAnio == anioNacimiento(res) ∧
//(∀c ∈ compsOro[0..i@C_1])anioNacimiento(campeon(c)) ≤ anioNacimiento(res);
//implica i == i@C_0 + 1 ∧ 0 ≤ i < |compsOro| + 1 ∧ esCampeon(res, j) ∧ mayorAnio == anioNacimiento(res) ∧
//(∀c ∈ compsOro[0..i@C_1 + 1))anioNacimiento(campeon(c)) ≤ anioNacimiento(res);
//implica I : 0 ≤ i ≤ |compsOro| ∧ esCampeon(res, j) ∧ mayorAnio == anioNacimiento(res) ∧
//(∀c ∈ compsOro[0..i))anioNacimiento(campeon(c)) ≤ anioNacimiento(res);
        }
```

3.4. Demostración del condicional

Tenemos:

$P_{if} : 0 \leq i < |compsOro| \land esCampeon(res, j) \land mayorAnio == anioNacimiento(res) \land$
$(\forall c \in compsOro[0..i))anioNacimiento(res) \leq anioNacimiento(campeon(c))$

$Q_{if} : 0 \leq i < |compsOro| \land esCampeon(res, j) \land mayorAnio == anioNacimiento(res) \land$
$(\forall c \in compsOro[0..i])anioNacimiento(res) \leq anioNacimiento(campeon(c))$

$A : anioNacimiento(campeon(compsOro_i)) > mayorAnio$

Queremos ver que, dado P_{if}, Q_{if} vale para tanto si A es verdadero como si es falso.

Rama `true`

```
if (compsOro[i].ranking()[0].anioNacimiento() > mayorAnio) {
```

//estado ifT_0;
//vale $P_{if} : 0 \leq i < |compsOro| \land esCampeon(res, j) \land mayorAnio == anioNacimiento(res) \land$
$(\forall c \in compsOro[0..i))anioNacimiento(res) \leq anioNacimiento(campeon(c))$;
//vale $A : anioNacimiento(campeon(compsOro_i)) > mayorAnio$;

```
res = compsOro[i].ranking()[0];
```

//estado ifT_1;
//vale $((\forall c \in compsOro[0..i))anioNacimiento(res@ifT_0) \leq anioNacimiento(campeon(c))) \land$
$anioNacimiento(campeon(compsOro_i)) > mayorAnio@ifT_0 \land res == compsOro[i].ranking()[0] \land i == i@ifT_0$;

```
mayorAnio = compsOro[i].ranking()[0].anioNacimiento();
```

//estado ifT_2;
//vale $((\forall c \in compsOro[0..i))anioNacimiento(res@ifT_0) \leq anioNacimiento(campeon(c))) \land$
$anioNacimiento(campeon(compsOro_i)) > mayorAnio@ifT_0 \land res == res@ifT_1$
$\land mayorAnio == compsOro[i].ranking()[0].anioNacimiento() \land i == i@ifT_1$;
//implica $((\forall c \in compsOro[0..i))anioNacimiento(res@ifT_0) \leq anioNacimiento(campeon(c))) \land$
$anioNacimiento(campeon(compsOro_i)) > mayorAnio@ifT_0 \land res == res@compsOro[i].ranking()[0] \land mayorAnio ==$
$compsOro[i].ranking()[0].anioNacimiento() \land 0 \leq i < |comps|$;
por estados anteriores

//implica $((\forall c \in compsOro[0..i))anioNacimiento(res@ifT_0) \leq anioNacimiento(campeon(c))) \land$
$anioNacimiento(campeon(compsOro_i)) > mayorAnio@ifT_0 \land res == ranking(compsOro_i)_0 \land$
$mayorAnio == anioNacimiento(ranking(compsOro_i)_0) \land 0 \leq i < |comps|$;
porque *ranking* y *anioNacimiento* son correctas respecto de su especificación

//implica $((\forall c \in compsOro[0..i])anioNacimiento(res) \leq anioNacimiento(campeon(c))) \land res == ranking(compsOro_i)_0 \land$
$mayorAnio == anioNacimiento(ranking(compsOro_i)_0) \land 0 \leq i < |comps|$;
porque $compsOro[0..i] == compsOro[0..i) + +compsOro_i$ y ya chequeamos que el año de nacimiento fuera mayor

//implica Q_{if}; por definición de *campeon*

}

Rama `false`

//estado ifF_0;
//vale $Pif : 0 \leq i < |compsOro| \land esCampeon(res, j) \land mayorAnio == anioNacimiento(res) \land$
$(\forall c \in compsOro[0..i))anioNacimiento(res) \leq anioNacimiento(campeon(c))$
//vale $\neg A : anioNacimiento(campeon(compsOro_i)) \leq mayorAnio$;
//implica $(\forall c \in compsOro[0..i])anioNacimiento(res) \leq anioNacimiento(campeon(c))$
porque $compsOro[0..i] == compsOro[0..i) + +compsOro_i \land anioNacimiento(campeon(compsOro_i)) \leq mayorAnio \land$
$mayorAnio == anioNacimiento(res)$

4. Terminación del ciclo

4.1. v decrece

Sea la función variante $v : |compsOro| - i$.
Quiero ver que $v@C_2 < v@C_1$.

```
        while (i<compsOro.size()) {
```

//estado C_0;
//variante $v : |compsOro| - i$;

```
                if (compsOro[i].ranking()[0].anioNacimiento() > mayorAnio) {
                        res = compsOro[i].ranking()[0];
                        mayorAnio = compsOro[i].ranking()[0].anioNacimiento();
                }
```

//estado C_1;
//vale $i == i@C_0$;

```
                i++;
```

//estado C_2;
//vale $i == i@C_1 + 1 \land v == v - i$;
//implica $i == i@C_0 + 1$;
//implica $v == v - (i@C_0 + 1)$;
//implica $v == (v - i@C_0) - 1$;
//implica $v == v@C_0 - 1$;
//implica $v@C_2 < v@C_0$;

```
        }
```

4.2. $v \leq 0 \rightarrow \neg B$

Quiero ver que $v \leq 0 \rightarrow \neg B$.

$v \leq 0 \leftrightarrow |compsOro| - i \leq 0 \leftrightarrow |compsOro| \leq i \leftrightarrow \neg B$

5. Conclusión

Como pudimos probar que el ciclo termina y es correcto, y que la cláusula `asegura` del problema `atletaProdigio` vale en el estado final del programa, hemos demostrado que la función `JJ00::atletaProdigio()` es correcta respecto de la especificación propuesta.