# Gestión de revistas científicas:
# Justificación de las operaciones

9 de diciembre de 2013

## 1 La clase `Biblioteca`

### 1.1 La operación `baja_revista`

Esta operación da de baja, o lo que es lo mismo, elimina una revista de la biblioteca, a partir de su nombre, eliminando así toda la información sobre esta revista: su nombre, sus palabras clave, sus áreas temáticas de clasificación en la biblioteca y por último su índice de calidad.

#### 1.1.1 Implementación

```
void Biblioteca::baja_revista (const string &s){

//Pre: La revista, que tiene por nombre s, está en la biblioteca.

bool eliminada = false;
int i = 0;

//bucle 1;
//Inv : 0 <= i <= biblioteca.size(), el subvector biblioteca[0..i-1] no contiene la revista con nombre s, si  eliminada es true entonces se ha encontrado y eliminado la revista con nombre s de biblioteca[i].

while (i < biblioteca.size() and not eliminada){
    eliminar_revista(s, eliminada, i);
    if (not eliminada) ++i;
}
bool eliminada2 = false;
list<pair<string, string> >::iterator it2 = biblioteca[i].L2.begin();

//bucle 2;
//Inv: biblioteca[i].L2.end() es el tamaño de la lista, entonces la sublista inicializada en biblioteca[i].L2.begin() hasta la posición anterior al iterador no contiene la revista por nombre s y eliminada2 indica si se ha encontrado y eliminado la revista con nombre s de biblioteca[i].L2.

while (it2 != biblioteca[i].L2.end() and not eliminada2){
    if ((*it2).first == s){
        it2 = biblioteca[i].L2.erase(it2);
        eliminada2 = true;
    }
    else ++it2;
}

//Post La revista, que tiene por nombre s, deja de formar parte de la Biblioteca.

}
```

#### 1.1.2 Justificación

Justificación del bucle 1:

- *Inicializaciones*. De entrada, consideramos que no hemos visitado ninguna revista de la `biblioteca`, por tanto inicializamos la `i` a cero, la posición del primer elemento del vector, satisfaciendo la primera y la segunda parte del invariante, dado que no puede existir

la revista por nombre `s` en un subvector `biblioteca[0..i-1]`. Para satisfacer la tercera parte, solo hace falta poner el valor de `eliminada` a `false`, ya que cualquiera implicación que tenga como premisa `false` siempre se cumplirá (por definición).

- *Condición de salida*. Se puede salir del bucle por dos razones:

  - Si `i` llega a ser `biblioteca.size()` quiere decir, por el invariante, que hemos explorado todo el vector `biblioteca` y que `eliminada == false` (de otra manera, querría decir que en `biblioteca[i]` está la revista que estamos buscando, lo que es imposible porque `biblioteca[biblioteca.size()]` no existe), como se pretende en la postcondición.

  - En caso contrario, como para el invariante tenemos que `i <= biblioteca.size()`, se cumple que `i < biblioteca.size()` y para salir del bucle se tiene que cumplir que `eliminada` sea cierto.
    Pero si antes de llegar al final del vector `biblioteca` tenemos que `eliminada` pasa a ser `true`, querrá decir (de nuevo por el invariante) que hemos encontrado y eliminado la revista por nombre `s` dentro de `biblioteca[i]` y que `eliminada` ya nos dice que la revista por nombre `s` se ha encontrado y eliminado de `biblioteca` y también se cumple la postcondición.

- *Cuerpo del bucle*. Por el invariante sabemos que la revista por nombre `s` no se encuentra en `biblioteca[0..i-1]` y que por la condición de entrada sabemos que `i < biblioeca.size()` marca una posición válida del vector y que `eliminada == false`. Necesitamos comprobar entonces si en `biblioteca[i]` se encuentra la revista por nombre `s` o no y en caso afirmativo eliminarla. Para saberlo usaremos una función auxiliar `eliminar_revista` que nos dará esa información. Si no lo es entonces la revista por nombre `s` no se encuentra en `biblioteca[0..i]`, `eliminada` seguirá en `false` y solo hace falta incrementar `i` para que se vuelva a cumplir el invariante. Si la revista por nombre `s` se encuentre en `biblioteca[i]` entonces las dos primeras partes del invariante se continúan cumpliendo y solo hace falta asignar `eliminada = true` para poder salir del bucle satisfaciendo al mismo tiempo la tercera parte del invariante. Notad que no podemos incrementar `i` si hemos encontrado la revista por nombre `s`, porque entonces no se cumpliría la segunda parte del invariante.

- *Finalización*. En cada vuelta, o bien decrece la distancia entre `biblioteca.size()` y el índice `i`, porque incrementamos `i`, o bien ponemos el booleano `eliminada` a `true` y salimos del bucle.

Justificación del bucle 2:

- *Inicializaciones*. De entrada, podemos aprovechar la `i` que nos viene del primer bucle (porqué es el índice de calidad de la revista) y declarar el iterador en `biblioteca[i].L2.begin()` de la lista de pairs de strings en esa posición del vector y ahorrarnos una búsqueda. Para satisfacer el invariante, solo hace falta poner el valor de `eliminada2` a `false`, ya que cualquiera implicación que tenga como premisa `false` siempre se cumplirá (por definición).

- *Condición de salida*. Se puede salir del bucle por dos razones:

- Si `it2` llega a ser `biblioteca[i].L2.end()` quiere decir que hemos llegado a explorar toda la lista `biblioteca[i].L2` y que `eliminada2 == false` (de otra manera querría decir que `(*it2)` es igual a la revista por nombre `s`, lo que es imposible por el tamaño de la lista).

- En caso contrario, se cumple que `it2 < biblioteca[i].L2.end()` y para salir del bucle se tiene que cumplir que `eliminada2` sea cierto.
  Pero si antes de llegar al final de la lista `biblioteca[i].L2` tenemos que `eliminada2` pasa a ser `true`, querrá decir, por el invariante, que hemos encontrado la revista por nombre `s` en la posición `(*it2)` y que `eliminada2` ya nos dice que la reviste por nombre `s` se ha encontrado y eliminado en `biblioteca[i].L2` y también se cumple la postcondición.

- *Cuerpo del bucle*. Por el invariante sabemos que la revista por nombre `s` no se encuentra en la sublista inicializada en `biblioteca[i].L2.begin()` hasta la posición anterior al iterador y que por la condición de entrada sabemos que `it2 < biblioteca[i].L2.end()` marca una posición válida del vector y que `eliminada2 == false`. Necesitamos comprobar entonces si `(*it2)` es igual a la revista por nombre `s` o no, Si no lo es, entonces la revista por nombre `s` no se encuentra en la sublista inicializada en `biblioteca[i].L2.begin()` hasta la posición anterior al iterador y solo hace falta incrementar `it2`. Si la revista por nombre `s` es igual a `(*it2)` entonces solo hace falta asignar `eliminada2` a `true` para poder salir del bucle satisfaciendo el invariante. Notad que no podemos incrementar `it2` si hemos encontrado la revista por nombre `s`.

- *Finalización*. En cada vuelta, o bien decrece la distancia entre `biblioteca[i].L2.end()` y el iterador `it2`, porque incrementamos `it2`, o bien ponemos el booleano `eliminada2` a `true` y salimos del bucle.

Todas las llamadas a operaciones son correctas. La postcondición de `eliminar_revista` hace que las operaciones posteriores a la llamada `eliminar_revista` permitan alcanzar la postcondición de `baja_revista`.

### 1.2 La operación auxiliar `eliminar_revista`

La operación `eliminar_revista` busca en una lista de revistas, de una determinada posición del vector, la revista que tiene por nombre `s` y la elimina.

#### 1.2.1 Implementación

```
void Biblioteca::eliminar_revista(const string &s, bool& eliminada, int i) {

//Pre: cierto.

list<Revista>::iterator it = biblioteca[i].L1.begin();

//Inv: biblioteca[i].L1.end() es el tamaño de la lista, entonces la sublista
inicializada en biblioteca[i].L1.begin() hasta la posición anterior al iterador
no contiene la revista por nombre s, si eliminada es true entonces (*it) es
igual a la revista por nombre s.
```

```
while (it != biblioteca[i].L1.end() and not eliminada){
      if ((*it).cons_nombre() == s){
            it = biblioteca[i].L1.erase(it);
            eliminada = true;
      }
      else ++it;
    }

//Post: Si eliminada es true la revista, que tiene por nombre s, deja de formar
//      parte de la biblioteca.

}
```

### 1.2.2 Justificación

Justificación del bucle:

- *Inicializaciones*. Inicialmente no hemos comprobado ningún elemento de `biblioteca[i].L1`, por tanto inicializamos el iterador en `biblioteca[i].L1.begin()`, la posición del primer elemento de la lista dado que no puede existir la revista por nombre `s` en una sublista inicializada en `biblioteca[i].L1.begin()` hasta la posición anterior al iterador. Para satisfacer el invariante, solo hace falta poner el valor de `eliminada` a `false`, ya que cualquiera implicación que tenga como premisa `false` siempre se cumplirá (por definición).

- *Condición de salida*. Se puede salir del bucle por dos razones:

  - Si `it` llega a ser `biblioteca[i].L1.end()` quiere decir que hemos llegado a explorar toda la lista `biblioteca[i].L1` y que `eliminada == false` (de otra manera querría decir que `(*it)` es igual a la revista por nombre `s`, lo que es imposible por el tamaño de la lista).

  - En caso contrario, se cumple que `it < biblioteca[i].L1.end()` y para salir del bucle se tiene que cumplir que `eliminada` sea cierto.
    Pero si antes de llegar al final de la lista `biblioteca[i].L1` tenemos que `eliminada` pasa a ser `true`, querrá decir por el invariante que hemos encontrado y eliminado la revista por nombre `s` en la posición `(*it)` y que `eliminada` ya nos dice que la reviste por nombre `s` se ha encontrado y eliminado de `biblioteca[i].L1` y también se cumple la postcondición.

- *Cuerpo del bucle*. Por el invariante sabemos que la revista por nombre `s` no se encuentra en la sublista inicializada en `biblioteca[i].L1.begin()` hasta la posición anterior al iterador y que por la condición de entrada sabemos que `it < biblioteca[i].L1.end()` marca una posición válida del vector y que `eliminada == false`. Necesitamos comprobar entonces si `(*it)` es igual a la revista por nombre `s` o no. Si no lo es, entonces la revista por nombre `s` no se encuentra en la sublista inicializada en `biblioteca[i].L1.begin()` hasta la posición anterior al iterador y solo hace falta incrementar `it` para que se vuelva a cumplir el invariante. Si la revista por nombre `s` es igual a `(*it)` entonces solo hace falta asignar `eliminada = true` para poder salir del bucle satisfaciendo el invariante. Notad que no podemos incrementar `it` si hemos encontrado la revista por nombre `s`.

- *Finalización*. En cada vuelta, o bien decrece la distancia entre `biblioteca[i].L1.end()` y el iterador `it`, porque incrementamos `it`, o bien ponemos el booleano `eliminada` a `true` y salimos del bucle.

## 2. La clase `Esquema`

### 2.1 La operación `calcular_c2`

La operación `calcular_c2` calcula y asigna el área temática en el cual está clasificada la revista `r` según el criterio de clasificación 2.

#### 2.1.1 Implementación

```
void Esquema::calcular_c2(Revista &r) {

//Pre: El esquema contiene todas las palabras clave de r.

    pair<bool, string> aux;
    Arbre<string> copia(a);
    aux = auxcalcular_c2(r, copia);
    r.mod_c2(aux.second);

//Post: Modifica y asigna el área temática en el cual está clasificada la
revista r según el criterio de clasificación 2.

}
```

#### 2.1.2 Justificación

Esta operación no contiene ningún bucle ni ninguna llamada recursiva a justificar. Básicamente, tenemos que preocuparnos de que las llamadas a otras operaciones sean correctas y que permitan obtener la postcondición. Por un lado, los resultados de `auxcalcular_c2` permiten obtener los de `calcular_c2` de manera inmediata.

### 2.2 La operación auxiliar `auxcalcular_c2`

Esta será la función más importante de `calcular_c2` y nos calculará el área temática.

#### 2.2.1 Implementación

```
pair<bool, string> Esquema::auxcalcular_c2(Revista &r, Arbre<string>& a){

//Pre: a no es vacío.

    string raiz = a.arrel();
    pair<bool, string> aux, aux1, aux2;
    aux.first = false;
    Arbre<string> a1, a2;
    a.fills(a1, a2);
    if (a1.es_buit() and a2.es_buit()){
      if (r.cons_palabrasclave(raiz)){
          aux.second = raiz;
          aux.first = true;
      }
      else aux.first = false;
    }

//HI: aux1.first = si esta en true ha encontrado una palabra clave de la revista
//    r en el hijo izquierdo, si está a false no.
//    aux1.second = área temática más alejada que contiene a todas las palabras
//    clave de una revista en el hijo izquierdo.
```

```
//HI: aux2.first = si esta en true ha encontrado una palabra clave de la revista
//    r en el hijo derecho, si está a false no.
//    aux2.second = área temática más alejada que contiene a todas las palabras
//    clave de una revista en el hijo derecho.

    else {
      if (not a1.es_buit() and not a2.es_buit()){
          aux1 = auxcalcular_c2(r, a1);
          aux2 = auxcalcular_c2(r, a2);
      }

//HI: aux1.first = si esta en true ha encontrado una palabra clave de la revista
//    r en el hijo izquierdo, si está a false no.
//    aux1.second = área temática más alejada que contiene a todas las palabras
//    clave de una revista en el hijo izquierdo.

      else if (not a1.es_buit() and a2.es_buit()){
          aux1 = auxcalcular_c2(r, a1);
          aux2.first = false;
      }

//HI: aux2.first = si esta en true ha encontrado una palabra clave de la revista
//    r en el hijo derecho, si está a false no.
//    aux2.second = área temática más alejada que contiene a todas las palabras
//    clave de una revista en el hijo derecho.

      else if (a1.es_buit() and not a2.es_buit()){
          aux2 = auxcalcular_c2(r, a2);
          aux1.first = false;
      }

      if (not aux1.first and not aux2.first) aux.first = false;
      else if (aux1.first and not aux2.first) aux = aux1;
      else if (not aux1.first and aux2.first) aux = aux2;
      else {
          aux.second = raiz;
          aux.first = true;
      }
    }
    return aux;

//Post: aux.first = si esta en true ha encontrado una palabra clave de la
//      revista r, si está a false no.
//      aux.second = área temática más alejada que contiene a todas las palabras
//      clave de una revista.

}
```

### 2.2.2 Justifiación

- *Caso directo*. Como precondición tenemos que el árbol no es vacío, entonces existen sus subárboles izquierdo y derecho y se pueden obtener sin errores con la operación `fills`. Los hijos los llamamos `a1` y `a2`, si `a1` y `a2` son vacíos, hemos de mirar si la raíz de este subárbol es palabra clave de la revista o no. Si no lo es, devolvemos el booleano a `false`, y nos da igual el área temática. Si es que si, devolvemos el booleano a `true`, y guardamos la raíz como posible área temática.

- *Caso recursivo*. Tenemos tres casos recursivos:

  - Si `a1` es vació pero `a2` no es vacío entonces debemos guardar el valor de realizar la

llamada recursiva en `a2`, que nos devuelve si hemos encontrado una palabra clave de la revista `r` y su posible área temática, que lo sabemos por `HI`, y de `a1` devolvemos el booleano a `false` porque es vació.

- Si `a2` es vació pero `a1` no es vacío entonces debemos guardar el valor de realizar la llamada recursiva en `a1`, que nos devuelve si hemos encontrado una palabra clave de la revista `r` y su posible área temática, que lo sabemos por `HI`, y de `a2` devolvemos el booleano a `false` porque es vacío.

- Si los dos no son vacíos, debemos guardar los dos valores de realizar las dos llamadas recursivas tanto para `a1` como para `a2`. La llamada recursiva a `a1` nos devuelve si hemos encontrado una palabra clave de la revista `r` y su posible área temática, que lo sabemos por `HI`. La llamada recursiva a `a2` nos devuelve si hemos encontrado una palabra clave de la revista `r` y su posible área temática, que también lo sabemos por `HI`.

Una vez hechas las llamadas recursivas, tenemos que comparar los booleanos que nos han llegado por cada hijo. Si uno es `true` y el otro es `false`, no quedaremos con toda la información del que tiene `true`. Si los dos son `true`, devolveremos el booleano de la raíz a `true` y guardaremos esa raíz como posible área temática. Si los dos son `false`, devolveremos el booleano a `false` y nos dará igual el área temática.

- *Finalización*. Cada llamada recursiva hace más pequeño el árbol.

Notad que hacemos una llamada a una función auxiliar `cons_palabrasclave` que nos ayudará a alcanzar la postcondición.

### 2.3 La operación auxiliar `cons_palabrasclave`

Está función nos indicará si la palabra que estamos evaluando es palabra clave de la revista o no.

#### 2.3.1 Implementación

```
bool Revista::cons_palabrasclave(const string &s) const {

//Pre: cierto.

bool trobat = false;
list<string>::const_iterator it = palabrasclave.begin();

//Inv: palabrasclave.end() es el tamaño de la lista, entonces la sublista
inicializada en palabrasclave.begin() hasta la posición anterior al iterador no
contiene la palabra s, si trobat es true entonces (*it) = s.

while (it != palabrasclave.end() and not trobat){
    if (*it == s) trobat = true;
    else ++it;
}
return trobat;

//Post: Indica si s es palabra clave del parámetro implícito o no.

}
```

#### 2.3.2 Justificación

Justificación del bucle:

- *Inicializaciones*. Inicialmente no hemos comprobado ningún elemento de `palabrasclave`, por tanto inicializamos el iterador en `palabrasclave.begin()`, la posición del primer elemento de la lista dado que no puede existir la palabra `s` en una sublista inicializada en `palabrasclave.begin()` hasta la posición anterior al iterador. Para satisfacer el invariante, solo hace falta poner el valor de `trobat` a `false`, ya que cualquiera implicación que tenga como premisa `false` siempre se cumplirá (por definición).

- *Condición de salida*. Se puede salir del bucle por dos razones:

  - Si `it` llega a ser `palabrasclave.end()` quiere decir, que hemos llegado a explorar toda la lista `palabrasclave` y que `trobat == false` (de otra manera querría decir que `(*it)` es igual a la palabra `s`, lo que es imposible por el tamaño de la lista).

  - En caso contrario, se cumple que `it < palabrasclave.end()` y para salir del bucle se tiene que cumplir que `trobat` sea cierto.
    Pero si antes de llegar al final de la lista `palabrasclave` tenemos que `trobat` pasa a ser `true`, querrá decir por el invariante que hemos encontrado la palabra `s` en la posicion `(*it)` y que `trobat` ya nos dice que la palabra `s` se encuentra en `palabrasclave` y también se cumple la postcondición.

- *Cuerpo del bucle*. Por el invariante sabemos que la palabra `s` no se encuentra en la sublista inicializada en `palabrasclave.begin()` hasta la posición anterior al iterador, y que por la condición de entrada sabemos que `it < palabrasclave.end()` marca una posición válida de la lista y que `trobat == false`. Necesitamos comprobar entonces si `(*it)` es igual a la palabra `s` o no, Si no lo es, entonces la palabra `s` no se encuentra en la sublita inicializada en `palabrasclave.begin()` hasta la posición anterior al iterador, y solo hace falta incrementar `it`. Si la palabra `s` es igual a `(*it)` entonces solo hace falta asignar `trobat` a `true` para poder salir del bucle satisfaciendo el invariante. Notad que no podemos incrementar `it` si hemos encontrado la palabra `s`.

- *Finalización*. En cada vuelta, o bien decrece la distancia entre `palabrasclave.end()` y el iterador `it`, porque incrementamos `it`, o bien ponemos el booleano `trobat` a `true` y salimos del bucle.

### 2.4 La operación auxiliar `mod_c2`

Está función asigna a la revista el área temática según el criterio de clasificación 2.

#### 2.4.1 Implementación

```
void Revista::mod_c2(const string &s){

//Pre: cierto.

    c2 = s;

//Post: La revista de nombre s pasa a tener una nueva área temática asignada
según el criterio de clasificación 2.
```

```
}
```

### 2.4.2 Justificación

Como el área temática asignada a una revista según el criterio de clasificación 2 es un campo privado, está función modificara o añadirá el correspondiente área temática a la revista. La nueva área temática será el string `s`.