Departamento de Cs. e Ingeniería de la Computación
Universidad Nacional del Sur

Métodos Formales para Ingeniería de Software

Trabajo Práctico N° 4

Especificación y Verificación - JML y KeY

Segundo Cuatrimestre de 2018

Ejercicios

1. Demuestre las siguientes fórmulas utilizando el Cálculo de Secuentes:

   a) $(\forall x\ p(x) \lor \forall x\ c(x)) \rightarrow \forall x\ (p(x) \lor c(x))$

   b) $\neg(\exists x\ p(x)) \rightarrow \forall x\ (\neg p(x))$

   c) $(\forall x\ p(x) \rightarrow \exists x\ q(x)) \rightarrow \exists x\ (p(x) \rightarrow q(x))$

   d) $((\forall x\ f(g(x)) = g(x)) \land (g(a) = b)) \rightarrow (f(b) = g(a))$

2. Considere la siguiente definición (parcial) de la clase `AyudaArreglo`:

```
public class AyudaArreglo {

    /*@ public normal_behavior
      @ ensures arreglo[pos] == nuevoElem;
      @
      @ also
      @
      @ public normal_behavior
      @ ensures true;
      @*/
    public static void reemplazarSiMayor(int nuevoElem, int pos, int[] arreglo) {
        if (nuevoElem > arreglo[pos]) {
            arreglo[pos] = nuevoElem;
        }
    }
}
```

   a) Utilice KeY para verificar si el método `reemplazarSiMayor` respeta sus contratos.

   b) Para cada caso, analice la prueba construida por la herramienta y explique los motivos por los cuales no cierra.

   c) Modifique los contratos del método `reemplazarSiMayor` (no su implementación) de manera tal que al utilizar nuevamente KeY para verificar su correctitud, las pruebas cierren.

3. Considere la siguiente definición (parcial) de la clase `A`:

```
public class A {

    private /*@ spec_public @*/ int value;

    /*@ public normal_behavior
      @ requires true;
      @ ensures (a != null && a.length >= 2) ==> a[0].value >= a[1].value;
      @*/
    private void m(/*@nullable @*/A[] a) {
        if (a == null) {
            return;
        }
        if (a[0].value < a[1].value) {
            int tmp = a[0].value;
            a[0].value = a[1].value;
            a[1].value = tmp;
        }
        /*
           Si el arreglo a posee al menos dos elementos, entonces
           intercambia su primer y segundo elemento (de ser necesario)
           para que el primero sea mayor o igual que el segundo.
        */
    }

    /*@ public normal_behavior
      @ requires true;
      @ ensures \result >= 0;
      @ assignable \nothing;
      @
      @ also
      @
      @ public normal_behavior
      @ requires true;
      @ ensures \result < 0;
      @ assignable \nothing;
      @*/
    public int n(int a1, int a2) {
        this.value = a1;
        return a1 - a2;
        /*
          Asigna a1 al atributo value y retorna el resultado de restar a1 y a2.
        */
    }
}
```

a) Utilice KeY para verificar si los métodos `m` y `n` respetan sus contratos.

b) Para cada caso, analice la prueba construida por la herramienta y explique los motivos por los cuales no cierra.

c) Modifique las especificaciones JML brindadas (y agregue nuevos contratos en caso de ser necesario) para capturar el comportamiento esperado de los métodos (sin modificar su implementación).

d) Utilice nuevamente KeY para verificar la correctitud de los contratos con respecto a la implementación de los métodos, logrando cerrar todas las pruebas.

4. Considere una versión reducida de un sistema de e-commerce en el que usuarios pueden hacer uso de un carrito de compras para efectuar órdenes con el objetivo de comprar distintos ítems. El sistema es tal que cada usuario posee un carrito, en el cual puede almacenar una única orden (su orden actual).

   El conjunto de clases contenidas en el directorio **ClasesTP4-Ej4** del archivo **ClasesTP4.zip** intenta modelar el comportamiento del sistema arriba descripto:

   - La clase **User** corresponde a un usuario del sistema, proveyendo un método para que el usuario finalice su compra.
   - La clase **ShoppingCart** modela el comportamiento del carrito de compras, proveyendo diversas consultas para el mismo, así como también operaciones para añadirle una orden y proceder a despacharlo.
   - La clase **PaymentMethod** modela los distintos medios de pago que puede utilizar un usuario, siendo los únicos medios habilitados: Efectivo (*Cash*), Tarjeta de Débito (*Debit Card*) y Tarjeta de Crédito (*Credit Card*). En particular, cada medio de pago posee un valor límite asociado, el cual puede ser obtenido mediante el método `methodLimit`. Dicho límite podría ser utilizado para verificar si es posible utilizar un medio de pago para abonar una compra de un usuario (ver método `finishPurchase` de clase *User*). Nótese que el método `methodLimit` lanza una excepción en el caso en que el medio de pago sea inválido.
   - La clase **Orden** modela una orden que un usuario podría efectuar. En particular, el método `close` efectúa el cierre de una orden. Por otra parte, el método `orderValue` permite obtener el valor total de una orden, el cual se determina como la suma de los valores de los ítems que posee.
   - La clase **Item** modela las características de un ítem, incluyendo su identificador y su valor. Asimismo, el método `getValue` permite obtener el valor asociado al ítem.
   - Por último, las excepciones lanzadas por el método `methodLimit` de la clase *PaymentMethod* serán instancias de la clase **InvalidPaymentMethodExc**.

   a) Defina invariantes para modelar las siguientes restricciones:
      - El valor de un ítem debe ser un número positivo.
      - Una orden se encuentra abierta o cerrada (siendo estos dos estados mutuamente excluyentes).
      - Una orden no puede albergar más de dos ítems.

   b) Utilice KeY para verificar si los siguientes métodos respetan todos sus contratos:
      - Contratos del método `methodLimit` de la clase **PaymentMethod**.
      - Contrato del método `close` de la clase **Order**.
      - Contratos del método `orderValue` de la clase **Order**.

   Para el inciso *b*), si alguna de las pruebas no cierra, trate de encontrar el motivo por el cual no cierra analizando las metas abiertas (*open goals*) de la prueba y contrastándolas con la implementación de los métodos y sus correspondientes especificaciones JML. Solucione el problema (*i.e.*, logre cerrar las pruebas de todos los contratos, de acuerdo al comportamiento esperado de cada método) modificando las especificaciones JML provistas en las clases y/o el código Java, según corresponda y esté permitido.

Consideraciones adicionales para la resolución de este ejercicio:

- Se permite añadir invariantes adicionales a los especificados en el enunciado, en caso que lo considere necesario.
- Deberá proveer una justificación para toda modificación realizada en las especificaciones JML de las clases o en sus respectivos códigos JAVA.
- Sólo se permite modificar el código JAVA de aquellos métodos para los cuales no se prohíbe explícitamente su modificación.
- Las modificaciones realizadas sobre el código JAVA deberán ser tales que respeten el comportamiento esperado del método (en función de la descripción brindada).
- Se recomienda almacenar los archivos con extensión *.proof* generados por KEY para cada una de las pruebas realizadas (pruebas de todos los contratos).
- Se recomienda almacenar la versión final de todo archivo *.java* suministrado en el directorio **ClasesTP4-Ej4**. Es decir, la versión de los archivos *.java* que incluya las modificaciones realizadas para la resolución de este ejercicio.

5. Considere una pequeña porción de una aplicación para e-books. Cada libro es representado por una instancia de la clase **Libro** con las siguientes características:

   - Posee un número de *ISBN*, el cual es un número natural que provee un identificador unívoco para los libros.
   - Posee un *conjunto de páginas* (instancias de la clase **Pagina**), cada una de las cuales se halla identificada por un número. Asimismo, la cantidad de elementos en dicho conjunto determina la *cantidad de páginas* del del libro.
   - Posee tres *marcadores*, cada uno de los cuales puede ser utilizado para identificar (marcar) un número de página del libro . Los marcadores son inicializados en 0 y permanecen así hasta ser reemplazados por un número de página del libro (en cuyo caso dejan de estar libres).
   - Posee un indicador de *página actual*, que referencia al número de página que se encuentra actualmente leyendo.
   - El lector del libro puede intentar pasar una página del libro (método `pasarPagina`), moviéndose de la página actual a la siguiente, siempre y cuando no se encuentre en la última página del libro. De no ser posible pasar de página, el método `pasarPagina` lanza una excepción.
   - El lector del libro puede marcar la página actual (método `marcarPaginaActual`). El comportamiento esperado del marcado es el siguiente: si hay un marcador "libre", entonces se utiliza el slot correspondiente al menor número de marcador; en caso contrario, se reemplaza el marcador que referencia al número de página más chico.
   - El lector del libro puede saltar a la página indicada por un marcador (método `saltarAMarcador`). Si se utiliza un número de marcador válido que referencia a una página del libro, entonces se modifica la página actual para referenciar a aquella indicada por el marcador.

   Resuelva los siguientes incisos utilizando el conjunto de clases provisto en el directorio **ClasesTP4-Ej5** del archivo **ClasesTP4.zip**:

a) Defina invariantes para modelar las siguientes restricciones:

- Todo libro tiene al menos una página.
- Todo libro tiene exactamente tres slots para almacenar marcadores.
- Distintos marcadores (no libres) de un libro no pueden referenciar a la misma página del libro.
- La página actual de un libro debe referenciar a un número de página válido.

b) Brinde especificaciones JML correspondientes a contratos para el método `eliminarPagina` que cubran todos los casos posibles para el comportamiento esperado del método (en función de la descripción brindada), tanto cuando se trata de un comportamiento normal como de un comportamiento excepcional.

c) Brinde especificaciones JML correspondientes a contratos para el método `pasarPagina` que cubran todos los casos posibles para el comportamiento esperado del método (en función de la descripción brindada), tanto cuando se trata de un comportamiento normal como de un comportamiento excepcional.

d) Brinde especificaciones JML correspondientes a contratos para el método `marcarPaginaActual` que cubran todos los casos posibles para el comportamiento esperado del método (en función de las descripciones brindadas), tanto para cuando se produce la marca como cuando no.

OBSERVACIÓN: Considerar bajo qué condiciones no se produce la marca, cuándo se produce la marca en marcador 1, el marcador 2, o en el marcador 3.

e) Utilice KeY para verificar la correctitud de los contratos definidos en el inciso anterior, con respecto al comportamiento esperado del método `marcarPaginaActual`, utilizando la implementación provista para el método.

f) Utilice KeY para verificar si el método `saltarAMarcador` respeta sus contratos.

Para los incisos *e*) y *f*), si alguna de las pruebas no cierra, trate de encontrar el motivo por el cual no cierra analizando las metas abiertas (*open goals*) de la prueba y contrastándolas con la implementación de los métodos y sus correspondientes especificaciones JML. Solucione el problema (*i.e.*, logre cerrar las pruebas de todos los contratos, de acuerdo al comportamiento esperado de cada método) modificando las especificaciones JML provistas en las clases y/o el código Java, según corresponda y esté permitido.

Consideraciones adicionales para la resolución de este ejercicio:

- Se permite añadir invariantes adicionales a los especificados en el enunciado, en caso que lo considere necesario.
- Deberá brindar una justificación para toda modificación realizada en las especificaciones JML de la clase **Libro** o en su respectivo código Java.
- Sólo se permite modificar el código Java de aquellos métodos para los cuales no se prohíbe explícitamente su modificación.
- Las modificaciones realizadas sobre el código Java deberán ser tales que respeten el comportamiento esperado del método (en función de la descripción brindada).
- Se recomienda almacenar los archivos con extensión *.proof* generados por KeY para cada una de las pruebas realizadas (pruebas de todos los contratos).
- Se recomienda almacenar la versión final del archivo *.java* correspondiente a la clase **Libro**.

6. Considere la siguiente porción de código Java:

```
public class C {

    private /*@ spec_public @*/ int x;

    public int m() {
        int  y = x;
        int  z = 0;
        while (y > 0) {
            z = z + x;
            y = y - 1;
        }
        return z;
    }
}
```

   a) ¿Qué hace el método `m`, suponiendo que `x` es un número entero positivo?

   b) Brinde especificaciones JML para el método `m`, considerando todos los casos posibles para su comportamiento.

   c) Demuestre la correctitud parcial del método `m` mediante la definición de un invariante de ciclo.

   d) ¿Es posible demostrar la correctitud total de `m`? Justifique su respuesta.