Validación y Verificación de Software

Sobre el curso...

Contenidos

En este curso estudiaremos:

un conjunto de técnicas de apoyo a la validación y verificación de software, mediante diferentes procesos de análisis, como la verificación de programas y la generación automática de tests.

Técnicas clásicas para la verificación de programas concurrentes.

Técnicas automáticas de análisis, como model checking y análisis basado en constraint solving.

El curso es relativamente autocontenido (no requiere conocimientos previos sustanciales)

Incluye conceptos básicos de V&V (testing, cobertura, etc.), herramientas “clásicas” de apoyo a la verificación, y técnicas automáticas de análisis.

Sí se requiere conocimientos de matemática discreta y lógica, programación (orientada a objetos) y estructuras de datos

Modalidad

El curso está pensado para ser *un curso práctico* (con práctica en laboratorio)

El curso será teórico-práctico

Veremos detalles sobre el funcionamiento de las técnicas de validación y verificación de software

Veremos demos de uso de herramientas de análisis

Tendremos ejercicios incluyendo prácticos de laboratorio, sobre los conceptos vistos

http://dc.exa.unrc.edu.ar/moodle/

Evaluación: trabajos prácticos obligatorios con ejercicios a resolver usando las técnicas/herramientas aprendidas con defensa individual. (tipo *take home*)

Qué me gustaría que pase...

Que aprendan un conjunto de técnicas novedosas, bastante sofisticadas, de apoyo a la verificación

a través de una variedad de mecanismos

Que conozcan varias herramientas de verificación/bug-finding/ testing.

y que las incorporen a su práctica de programación!

Que conozcan usos sumamente prácticos de conceptos provenientes de verificación formal de programas.

en particular, que aprecien el valor de las especificaciones formales (pre- y post-condiciones, invariantes, ...)

Análisis Automático: Vista Previa

Para aclarar qué entenderemos por análisis automático, el tema central de la materia, veamos un ejemplo vinculado al testing de una implementación del TAD Lista en Java.

Descripción del TAD Lista:

- elementos: listas finitas de elementos
    - una lista es una sucesión o secuencia finita de elementos, en la cual un extremo se reconoce como el comienzo de la lista
- operaciones:
    - esVacia: indica si la lista es vacía o no.
    - vaciar: elimina todos los elementos de la lista.
    - longitud: retorna la cantidad de elementos en la lista.

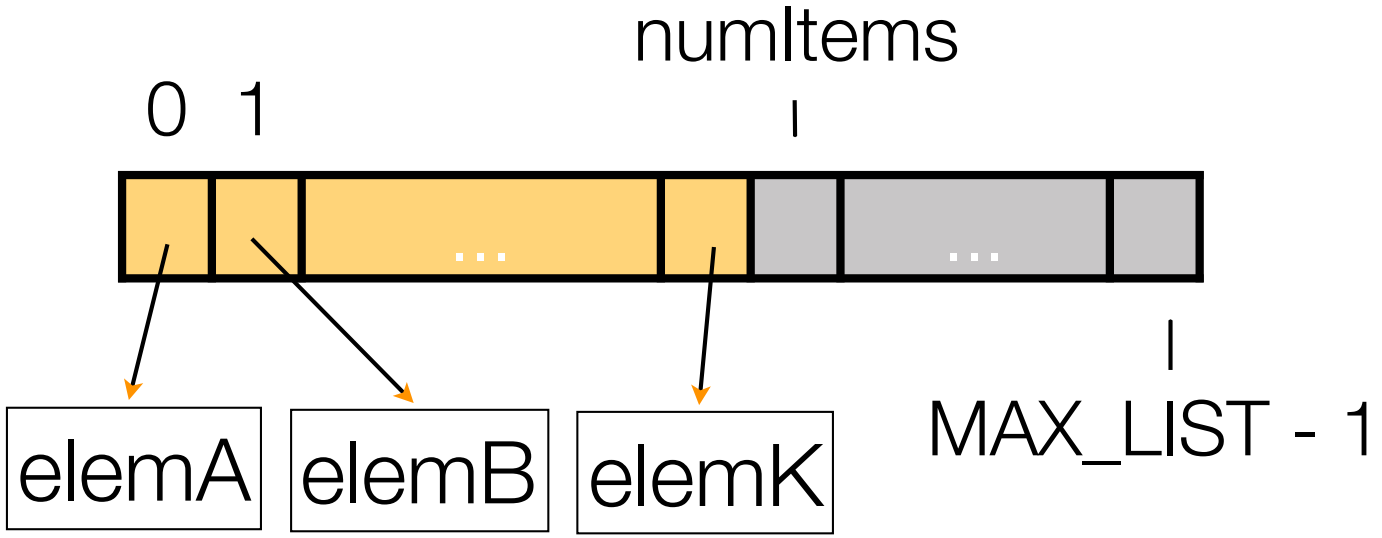
Análisis Automático: Vista Previa (cont)

- insertar: dado un elemento (no nulo) y una posición (entre 0 y la longitud de la lista), inserta el elemento en la posición dada, “moviendo” los elementos siguientes a esa posición.
- eliminar: dada una posición (entre 0 y la longitud de la lista menos 1), elimina el elemento en esa posición.
- obtener: dada una posición (entre 0 y la longitud de la lista menos 1), retorna el elemento en dicha posición sin modificar la lista.

Implementación del TAD Lista

Una de las implementaciones clásicas de listas es la basada en arreglos.

[elemA, elemB, ..., elemK]



Implementación del TAD Lista

```
/**
 * Implementacion del TAD Lista, usando un arreglo de objetos con tamaño
 * máximo MAX_LIST.
 * Esta clase implementa los métodos abstractos declarados en Lista.
 * @author Nazareno Aguirre
 * @version 0.2 16/9/2009
 */

public class ListaSobreArreglos implements Lista {

    private static final int MAX_LIST = 100; // numero máximo de items en la
                                             // lista.
    private Object items[];                  // arreglo usado para almacenar
                                             // los elementos de la lista.
    private int numItems;                    // entero que indica el número
                                             // de elementos en la lista, y
                                             // el ``fin'' de la lista en
                                             // items.

   /**
    * Constructor de la clase ListaSobreArreglos.
    * @pre. true.
    * @post. Se crea un arreglo de objetos de tamaño MAX_LIST, y se
    * inicializa numItems en 0.
    */
    public ListaSobreArreglos() {
        items = new Object[MAX_LIST];
    numItems = 0;
    }
```

Implementación del TAD Lista (cont.)

```
/**
 * inserta item en la posición index de la lista.
 * @param index es el índice en el cual se inserta el elemento.
 * @param item es el objeto a insertar en la lista.
 * @pre. 0<=index<=longitud()
 * @post. Si index es una posición válida, inserta item en esa posición.
 * Si index es una posición inválida, lanza una excepción de tipo
 * IndexOutOfBoundsException. Si la inserción falla por otro motivo, lanza
 * una excepción de tipo RunTimeException.
 */
 public void insertar(int index, Object item) throws RuntimeException, IndexOutOfBoundsException {
     if (numItems == MAX_LIST) {
      throw new RuntimeException("ListaSobreArreglos.insertar: Lista llena");
 }
 else {
      if ((index<0) || (index>=numItems)) {
          throw new IndexOutOfBoundsException("ListaSobreArreglos.insertar: índice inválido");
      }
      else {
          if (index == numItems) {
              // insertar item en la última posición
              items[index] = item;
              numItems = numItems + 1;
          }
          else {
              Object temp = items[index];
              items[index] = item;
              insertar(index+1, temp);
          }
      }
 }
 }
```

Análisis Automático: Vista Previa (cont)

- Correr Randoop con el ejemplo de listas sobre arreglos.
Si encuentra el bug en insertar (no se chequea non nullness- se rompe toString), agregar que lance IllegalArgumentException

```
if(item ==null){
        throw new IllegalArgumentException("ListaSobreArreglos.ins
```

- Quisiéramos poder garantizar que la implementación ListaSobreArreglos del TAD Lista es correcta
- Una forma de contribuir a garantizar la corrección es usar algún proceso de detección de bugs en la implementación
  - Veamos cómo una herramienta de generación de tests se puede usar para esto.

Mostrar cobertura
(1) marcar clase testeada como "use for coverage measurement" (2) mover a 'defaultpackage' los de (2) s tests 'positivos' (3) correr test0

DEMO Randoop