Übungen zur Vorlesung
Safety-Critical Systems I
Sommersemester 2007

AG BS
Jan Peleska
Christof Efkemann

Blatt 4

Modelling Safety-Critical Controllers With Temporal Logic

Software Fault Tree Analysis (SFTA)

Aufgabe 1: Understanding SFTA

Read Nancy G. Leveson's text about SFTA in *Safeware, Section 18.2.2*. Please answer the following questions:

1. Why should fault trees in SFTA *never* be annotated with risk quantifications?
2. Use Fig. 18.5 (assignment statements, simple decision and associated software fault tree) to explain why/how SFTA *proceeds backward through the code*.
3. Invent an SFTA template for C switch-statements.

Aufgabe 2: Applying SFTA

Consider the following update functions of a Java list management class.

```
1   class Node {
2       Object  data;
3       Node next;
4   }
5
6   class List {
7       Node first;
8       Node last;
9       Node predecessorActualPtr;
10      int len;
11  }
12
13  public class MyList {
14
15      public static List create() {
16          List l = new List();
17          l.first = new Node();
18          l.last = new Node();
19          l.first.next = l.last;
20          l.last.next = l.first;
21          l.predecessorActualPtr = l.first;
22          l.len = 0;
23          return l;
24      }
25
```

```
26      public static void append(List l, Object data) {
27          Node newListElement = new Node();
28          newListElement.data = data;
29          newListElement.next = l.last;
30          l.last.next.next = newListElement;
31          l.last.next = newListElement;
32          l.len += 1;
33      }
34
35
36      public static void insert(List l, Object data) {
37          if ( l.predecessorActualPtr.next == l.last ) {
38              append(l,data);
39              l.predecessorActualPtr = l.last.next;
40          }
41          else {
42              Node newListElement = new Node();
43              newListElement.data = data;
44              newListElement.next = l.predecessorActualPtr.next;
45              l.predecessorActualPtr.next = newListElement;
46              l.predecessorActualPtr = newListElement;
47              l.len += 1;
48          }
49      }
50
51      public static boolean haveNext(List l) {
52          return l.predecessorActualPtr.next != l.last;
53      }
54
55      public static boolean delete(List l) {
56          if ( !haveNext(l) ) return false;
57          l.predecessorActualPtr.next
58              = l.predecessorActualPtr.next.next;
59          if ( l.predecessorActualPtr.next == l.last ) {
60              // das Element am Ende der Liste wird geloescht
61              l.last.next = l.predecessorActualPtr;
62          }
63          l.len -= 1;
64          return true;
65      }
66
67      public static int length(List l) {
68          return l.len;
69      }
70
71  }
```

Using SFTA, prove that the following hazards do not occur, as far as the functions above are concerned:

1. H1: Operation `length(l)` returns an incorrect list length.
2. H2: Operation `insert()` loses a previously existing list element.
3. H3: Operation `delete()` loses more than one list element.

Abgabe: Bis Dienstag, 10.07.2007, per Mail.