10.12.15

Software Technik 06

Valentin Steiner 357980 Coanda, Radu 344664 Gregor Kobsik 359198
Jean Tekam Daniel Pujiula Buhl

Aufgabe 7.1:

(a)

Ohne die Programmcode anzusehen wäre das Black-Box Verfahren eine mögliche Vorgehensweise. Dabei benutzen wir die Spezifikation der Betragsfunktion um unsere Eingabe und Ausgabedaten zu generieren.

$$|x| = \begin{cases} x \text{ wenn } x \geq 0 \\ -x \text{ wenn } x < 0 \end{cases}$$

Von dieser Spezifikation aus, können wir jetzt die Kriterien für Äquivalenzklassen generieren.

1. x <0 ;
2. x >0 ;
3. x = 0; wobei diese Eingabe besonders behandelt sein soll.

(b)

1. Mit der Eingabe [-1;1] werden die wichtigsten Verhalten geprüft.
2. Zudem sind Randfälle zu betrachten: z.B. MAX_INT und MIN_INT, die den maximalen und minimalen Integer-Bereich abdecken. Durch die Speicherung der Zahlen im Zweierkomplement, hat man eine negative Zahl mehr als positive. Das kann zu einem Überlauf führen.

formale Partitionierung war gewünscht

3. Zudem ist die Eingabe von nicht erlaubten Werten sinnvoll. In diesem Fall sollte die Methode auch keinen Wert zurückliefern, sondern einen Fehler ausgeben. Mögliche Werte dafür sind Floats und Strings, sowie die Referenz auf ein „null“-Wert z.B. durch eine nicht initialisierte Variable. Des Weiteren ist die Eingabe von dem Verhalten mit einem unsigned Integer zu prüfen.
4. Die Folgenden Werte wären repräsentativ: 0,1,-1, 123456, –2.147.483.648, 2.147.483.647, int r, new Integer i = -2, „Hallo“, 3f

1,5/2

Aufgabe 7.2:

```
public static int binary2decimal ( String    string ) {
    int decimalNumber = 0;                                          //1
    int exponent = 0 ;
    char currentChar ;
    CharacterIterator stringIterator =
        new   StringCharacterIterator   (   string   );
    currentChar = stringIterator . last () ;
    while (( currentChar != CharacterIterator .DONE)                //2a
        && (decimalNumber < Integer .MAXVALUE)) {                   //2b
        if ( currentChar != '0 ' && currentChar !=    '1 ' )        //3
            return -1;                                              //4
                                                                    //5
        if ( currentChar == '1 ' )                                  //6
            decimalNumber += Math.pow(2 , exponent );               //7
                                                                    //8
        exponent++;                                                 //9
        currentChar = stringIterator . previous ();                 //10
    }                                                               //11
    return decimalNumber ;                                          //12
}
```

Für eine komplette Anweisungs- und Zweigüberdeckung reichen die Eingaben:

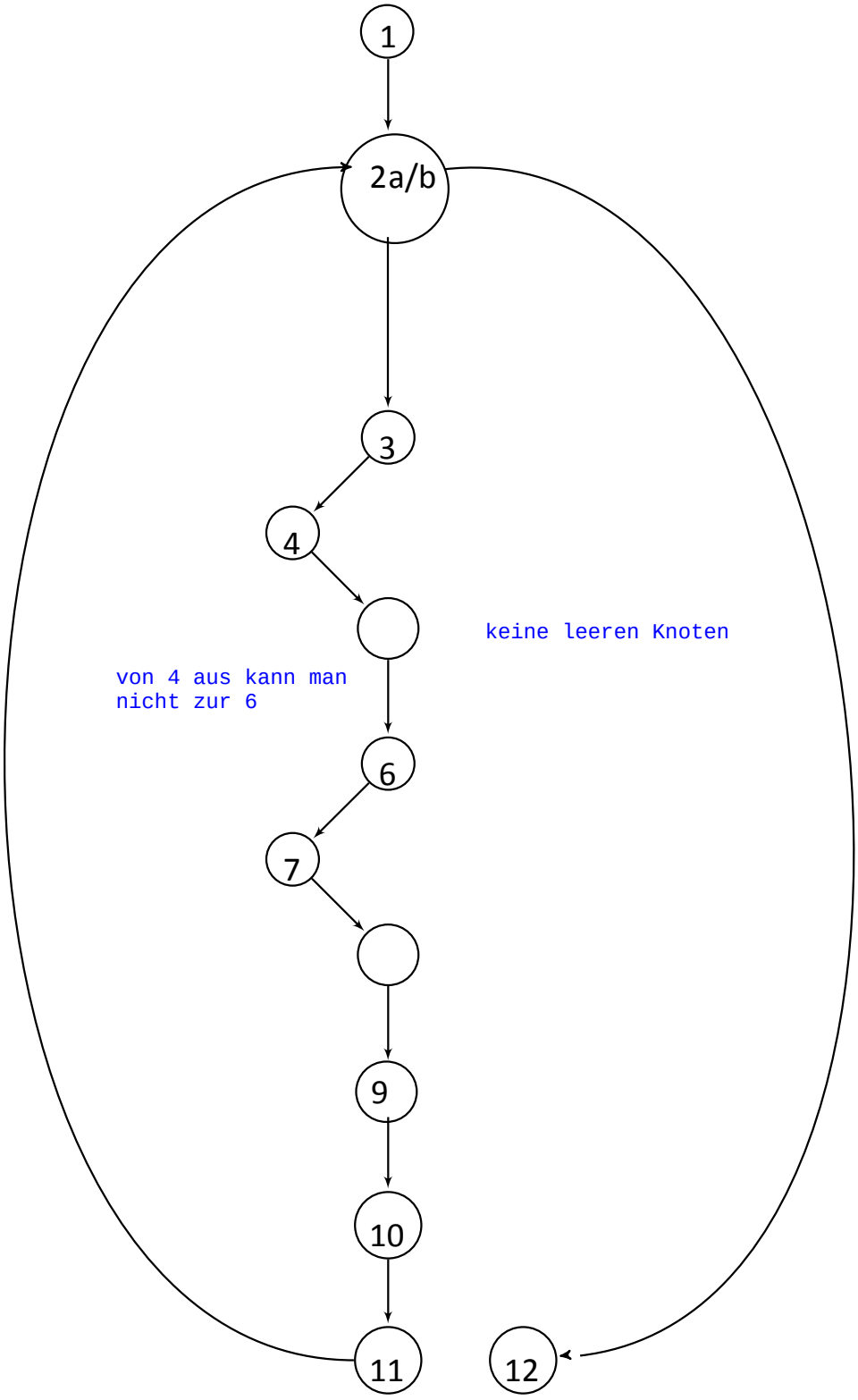
für Zweigüberdeckung sind 3 Fälle nötig

1. "2"
2. "101"

Für die Pfadüberdeckung haben wir Fälle:

1. Schleife nie durchlaufen wird mit der Eingabe "" (leere Eingabe) erreicht.
2. Bei einem Durchlauf der Schleife braucht man als Eingabe Strings die "0" oder "1" an der ersten Stelle(von rechts angefangen) gefolgt von einem beliebigen Zeichen außer "0" oder "1". Eingaben: e1, 20 etc.
3. Bei mehreren Schleifendurchläufe benötigt man mindestens 2 Zeichen der Menge "0", "1" gefolgt von einem beliebigen Zeichen außer "0" oder "1". Eingaben: 200, e10 etc.

10 Fälle notwendig


1
2a/b
3
4
keine leeren Knoten
von 4 aus kann man
nicht zur 6
6
7
9
10
11
12

0,5/10

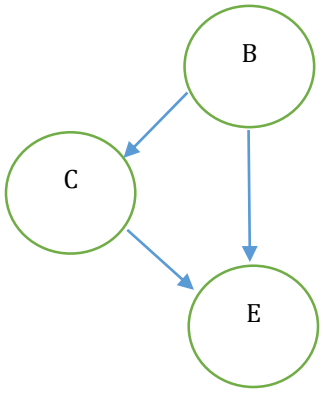
Aufgabe 7.3

a) Der Zweigüberdeckungstest ist stärker, da nicht nur alle Anweisungen ausgeführt werden, sondern auch alle möglichen Entscheidungen (false & true) durchlaufen werden. Hier kann sich herausstellen, dass manche Verzweigungen nie benutzt werden und somit eine Unnötige Abfrage darstellen.

0,5/0,5

b)

```
a. static int f(int x) {
b. int res = 0;
c. if (x<0) {res = x;}
d. else {} //Fehler, Code fehlt. (res = -x);
e. return res;
f. }
```



Die Funktion soll immer eine negative Zahl zurückliefern.

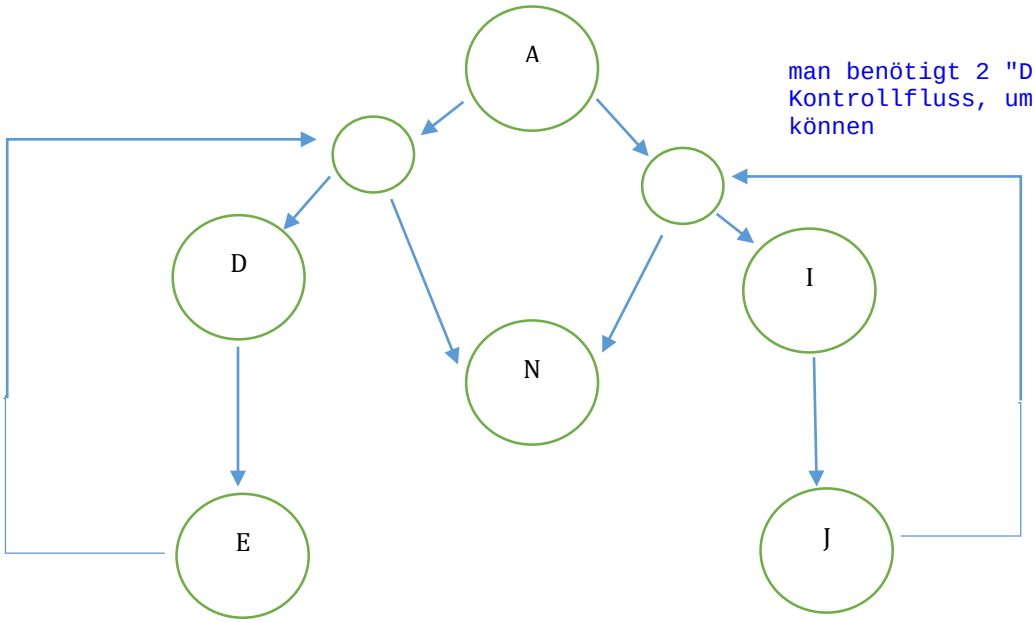
Bei der Eingabe von -2 erreicht man die volle Anweisungsüberdeckung bei einem Anweisungsüberdeckungstest, jedoch entdeckt man nicht, dass die Funktionalität in Zeile D fehlt, da dort keine Anweisung steht.
Erst mit dem Zweigüberdeckungstest, indem man alle Zweige durchgeht, z.B. mit den Eingaben 2 und -2 erkennt man, dass der erste Wert ein falsches Ergebnis liefert.

2,5/2,5

c)

```
int f(int x) {

a.              int res;//Fehler, (int res = 0);
b.              if (x > 0) {
c.                    while (x>0) {
d.                          res += x;
e.                          x--;
f.                    }
g.              } else if (x<0) {
h.                    while (x <0) {
i.                          res += x;
j.                          x++;
k.                    }
l.              }
m.
n.              return res;
          }
```



man benötigt 2 "Dreiecke" im Kontrollfluss, um den Beweis lifern zu können

Die Funktion summiert alle Zahlen von der Angegebenen Zahl bis zu 0, jeweils von unten oder von oben.

Bei den Testeingaben, wie -2 oder 2 liefert die Funktion einen richtigen Wert, nur bei der Eingabe 0 liefert die Funktion keinen Wert, da die Variable res nicht initialisiert ist.
Hier kann man den Fehler bei einem Anweisungsüberdeckungstest nicht ausfindig machen, da alle Anweisungen bei der Eingabe z.B. von 2 und -2 durchgeführt werden und man eine 100% Überdeckung hat.
Erst wenn man versucht den Pfadüberdeckugstest durchzuführen erkennt man den Fehler, da bei einer Eingabe in der die While-Schleifen nicht durchlaufen werden bekommt die Variable res ihren Wert.

4,5/6

insgesamt 6,5/10