Korrektheit von Algorithmen

1. Einleitung

Ein Algorithmus ist eine endliche, vollständige, präzise und eindeutige Beschreibung eines endlichen Vorgangs (Definition).
Algorithmen müssen bestimmten Anforderungen entsprechen. Eine dieser Anforderungen ist, dass Algorithmen **korrekt** sein müssen. Diese Forderung scheint zwar selbstverständlich zu sein, aber auch eine systematische und gut geplante Algorithmenentwicklung hat nicht selbstverständlich zur Folge, dass die entwickelten Algorithmen auch korrekt sind.

2. Algorithmen und Korrektheit

Algorithmen müssen die folgenden Eigenschaften bzw. Kriterien erfüllen, um als korrekt bezeichnet werden zu können.

- **Partielle Korrektheit**

Wenn ein Algorithmus bei gültigen Eingabegrößen die richtigen Ausgabegrößen berechnet (wenn er also die funktionale Spezifikation erfüllt), dann nennt man ihn *partiell korrekt*.

- **Terminierung**

Die Ausführung eines Algorithmus muß in jedem Fall zu einem Ende kommen. (Keine Endlosschleifen!) Wenn der Algorithmus nach endlich vielen Schritten zum Ende kommt, sagt man, er „*terminiert*“. Wenn der Algorithmus in eine Endlosschleife gerät, dann sagt man, er „*hängt*“.

- **Totale Korrektheit**

Ein Algorithmus, der *partiell korrekt* ist und *terminiert* heißt *total korrekt*. Der Nachweis der totalen Korrektheit heißt *Verfikation* und ist häufig nicht trivial.

3. Partielle Korrektheit und funktionale Spezifikation

Um über einen Algorithmus eine Aussage treffen zu können, ob der Algorithmus korrekt ist oder nicht, brauchen wir einen Maßstab, an dem wird die Korrektheit messen können.
Die f*unktionale Spezifikation* eines Algorithmus ist – wie bereits angedeutet - ein solcher Maßstab. Wir werden uns deshalb noch einmal ins Gedächtnis rufen, was eine funktionale Spezifikation ist.
Eine funktionale Spezifikation ist die genaue Beschreibung der gültigen Eingabegrößen und der gültigen Ausgabegrößen eines gesuchten Algorithmus sowie eine Beschreibung des funktionalen Zusammenhangs zwischen Eingabe- und Ausgabegrößen.
Hat man diese Beschreibung eindeutig angegeben, so sagt man, man hat das Problem (bzw. den Algorithmus zur Lösung des Problems) spezifiziert.
An einem Beispiel möchten wir dies erläutern.

3.1.1. Beispiel

Betrachten Sie die folgende funktionale Spezifikation:

1. Eingabegrößen: zwei ganze Zahlen a und b,
2. Ausgabegrößen: „ja!“ oder „nein!“
3. Funktionaler Zusammenhang:

F(a,b) = „ja!“ , falls a=b,
„nein!“, sonst

Das folgende PASCAL-Programm erfüllt die funktionale Spezifikation:

```
program FunktionaleSpezifikation1;
uses wincrt;
var a,b : integer;
begin
readl(a);
readln(b);
if a=b then writeln(' ja!)
        else writeln(' nein!');
end.
```

Wir sagen dann, das Programm ist *partiell korrekt*.

Nachweis der partiellen Korrektheit.

Um die partielle Korrektheit des oben angegebenen Algorithmus' nachweisen zu können, müssen wir die verschiedenen Fälle möglicher Eingaben diskutieren und für jeden Fall angeben, ob der Algorithmus korrekt arbeitet oder nicht.

1. Fall: a=b : In diesem Fall wird in der IF-THEN-ELSE-Verzweigung (a=b)=true gefunden und entsprechend „ja" ausgegeben.
2. Fall: a<b : In diesem Fall wird in der IF-THEN-ELSE-Verzweigung (a=b)=false gefunden und entsprechend „nein" ausgegeben.
3. Fall: a>b : In diesem Fall wird in der IF-THEN-ELSE-Verzweigung (a=b)=false gefunden und entsprechend „nein" ausgegeben.

Es zeigt sich, dass in allen Eingabe-Fällen die richtige Ausgabe erfolgt. Also konnten wir nachweisen (bzw. beweisen), dass der Algorithmus partiell korrekt ist.

3.1.2. Aufgabe 1

Überprüfen Sie, ob auch das folgende PASCAL-Programm die oben angegebenen funktionale Spezifikation erfüllt? Begründen Sie ausführlich mit Hilfe von Fallunterscheidungen.

```
program FunktionaleSpezifikation1;
uses wincrt;
var a,b : integer;
begin
readl(a);
readln(b);
if a-b=0 then writeln(' ja!)
        else writeln(' nein!');
end.
```

3.1.3. Aufgabe 2

Schreiben Sie PASCAL-Programme, die die folgenden funktionalen Spezifikationen erfüllen.

a) Eingabegrößen: eine positive ganze Zahl a
Ausgabegrößen: „ja!" oder „nein"
Funktionaler Zusammenhang:

F(a) = { „ja!", falls a gerade
{ „nein!", falls a ungerade

b) Eingabegrößen: eine ganze Zahl a
Ausgabegrößen: „1" oder „0"
Funktionaler Zusammenhang:

F(a) = { „1", falls a<0
{ „0", sonst

3.1.4. Zusammenfassung:

Zusammenfassend gilt:

Ein Algorithmus ist partiell korrekt, wenn er die funktionale Spezifikation erfüllt.

Ein Algorithmus ist nur dann korrekt, wenn er in jedem Fall zu einem Ende kommt, d.h. der Algorithmus muß bei jeder erdenklichen Eingabe anhalten und falls gewünscht, ein Ergebnis liefern
Wir merken uns:

Ein Algorithmus terminiert, wenn er in jedem Fall zu einem Ende kommt.

Der Nachweis der Terminierung ist in vielen Fällen nicht trivial. Wie beim Nachweis der partiellen Korrektheit empfiehlt es sich, Fallunterscheidungen über alle möglichen Eingaben zu diskutieren. Es ist dann zu zeigen, dass der Algorithmus in jedem dieser Fälle anhält.

3.2. BEISPIEL

Betrachten wir das folgende PASCAL-Programm.

```
program Korrekt_oder_nicht_korrekt;
 uses wincrt;

{Funktionale Spezifikation:
 Eingabe: eine positive ganze Zahl n
 Ausgabe: eine ganze Zahl zaehler
 Funktionaler Zusammenhang: zaehler := a div 2
 D.h. Das Programm dividiert also eine positive ganze
 Zahlen n durch 2}

 var n,zaehler: integer;

 begin
 readln(n);
 zaehler := 0;
 if n>0
 then
   repeat
     n := n - 2 ;
     zaehler := zaehler + 1 ;
 until n=0;
 writeln(zaehler);
 end.
```

Die genauerer Untersuchung der möglichen Eingabefälle zeigt, dass dieses Programm nur für gerade positive ganze Zahlen korrekt arbeitet. Wird eine ungerade positive ganze Zahl eingegeben, so gerät das Programm in eine Endlosschleife und „hängt sich auf“. Der Algorithmus terminiert nicht.

4. Nachweis der totalen Korrektheit

Um für einen Algorithmus nachzuweisen, ob er total korrekt ist, müssen wir nachweisen, dass er sowohl partiell korrekt ist als auch terminiert. Ist bereits ein Kriterium nicht erfüllt, so gilt der Algorithmus nicht mehr als total korrekt.
Wir wollen uns nun anhand eines Beispielalgorithmus‘ mit dem Nachweis der totalen Korrektheit beschäftigen.

4.1. BEISPIEL

Betrachten wir das folgende PASCAL-Programm.

```
program Korrekt_oder_nicht_korrekt;
uses wincrt;
 {Funktionale Spezifikation:
 Eingabe: zwei ganze Zahlen a und b
 Ausgabe: eine ganze Zahl c
 Funktionaler Zusammenhang: c:= a+b
 D.h. Das Programm berechnet die Summe zweier ganzer Zahlen a und b}
 var a,b,c: integer;
 begin
 readln(a);
 readln(b);
c:= a;
 repeat
   c := c + 1 ;
   b := b - 1 ;
 until b<=0;
 writeln(c);
 end.
```

Auf Grundlage der funktionalen Spezifikation wollen wir die totale Korrektheit des Programms diskutieren. Dazu betrachten wir Fallunterscheidungen über die möglichen Eingabegrößen a und b

1. Fall: a=0 und b>0:
   Seien a=0 und b>0 dann wird vor der Schleife c:=a=0 gesetzt.
   In der repeat-until-Schleife wird der Wert der Variablen c so lange erhöht und der Wert der Variablen b solange erniedrigt, bis b=0 ist. Nach der Schleife gilt dann zunächst c:=a+b=0+b gilt.
   Damit arbeitet der Algorithmus in diesem Fall korrekt.
2. Fall: a=0 und b=0:
   Seien a=0 und b=0 dann wird vor der Schleife zunächst c:=a=0 gesetzt
   In der repeat-until-Schleife wird der Wert der Variablen c um 1 erhöht und der Wert der Variablen b um 1 erniedrigt, so dass b=-1 gilt, wodurch die Schleife abbricht. Nach der Schleife gilt dann c=1, womit c=1<>a+b=0 gilt.
   Damit arbeitet der Algorithmus in diesem Fall nicht korrekt.
3. Eine Betrachtung der weiteren Fälle
   a=0 und b<0
   a<0 und b=0
   a<0 und b<0
   a<0 und b>0
   a>0 und b=0
   a>0 und b<0
   a>0 und b>0
   erübrigt sich, da bereits gezeigt wurde, dass der Algorithmus im Fall a=0 und b=0 nicht korrekt arbeitet.

Bemerkung:
Der angegebene Algorithmus arbeitet nur für Eingaben der Form a>=0 und b>= 0 korrekt. Eine entsprechende funktionale Spezifkation würde lauten:

Eingabe: zwei positive ganze Zahlen a und b
Ausgabe: eine positive ganze Zahl c
Funktionaler Zusammenhang: c:= a+b

5. Abschließende Bemerkungen

Die Korrektheit eines Algorithmus ist ein wichtiges Kriterium im Bereich der Softwareentwicklung. Wie die Praxis zeigt, sind viele der heute zu Standardanwendungen gewordenen Softwareprodukte (z.B. Betriebssysteme WINDOWS etc.) Fällen noch fehlerhaft. In solchen Fällen kann man davon ausgehen, dass die zugrundeliegenden Algorithmen nicht ausreichend geprüft wurden.
Ein formaler Beweis der Korrektheit von Programmen oder Programmteilen ist in vielen Fällen noch völlig unüblich und vor allem darin begründet, dass eine solche Beweisführung nicht trivial ist, sondern im Gegenteil, wegen der Komplexität heutiger Programme, häufig kaum zu bewältigen ist.
Wir können jedoch davon ausgehen, dass zentrale Algorithmen, wie z.B. der Kern eines Betriebsystems formal auf ihre Korrektheit hin untersucht worden sind, zumindest aber ausreichend getestet wurden.