Fachhochschule Wedel
Prof. Dr. Uwe Schmidt

20. Januar 1999

Grundlagen der Programmierung

Übungsaufgabe: Aussagenlogik: Wahrheitstabellen

Aufgabe 1:

Beweisen Sie mit Hilfe einer Wahrheitstabelle, daß die folgende Formel ein Satz der Aussagenlogik ist.

$(x \Rightarrow y) \Leftrightarrow (\neg x \Rightarrow \neg y)$

Aufgabe 2:

Beweisen Sie mit Hilfe von Wahrheitstabellen, daß die folgende Formel (modus ponens) ein Satz der Aussagenlogik ist.

$((x \Rightarrow y) \wedge x) \Rightarrow y$

Aufgabe 3:

Beweisen Sie mit Hilfe von Wahrheitstabellen, daß die folgende Formel ein Satz der Aussagenlogik ist.

$((x \Rightarrow y) \wedge (y \Rightarrow z)) \Rightarrow (x \Rightarrow z)$

Welcher logische Sachverhalt wird durch diese Formel modelliert?

Zeigen Sie mit Hilfe von Wahrheitstabellen, daß die folgende Formel kein Satz der Aussagenlogik ist.

$((x \Rightarrow y) \wedge (y \Rightarrow z)) \Leftrightarrow (x \Rightarrow z)$

Aufgabe 4:

Prüfen Sie mit Hilfe von Wahrheitstabellen, ob die folgende Formel ein Satz der Aussagenlogik ist.

$(x \Rightarrow y) \Rightarrow (y \Rightarrow x)$

Aufgabe 5:

Beweisen Sie mit Hilfe einer Wahrheitstabelle, daß die Operation $\oplus$ assoziativ ist.

Aufgabe 6:

Überprüfen Sie mit Hilfe einer Wahrheitstabelle, ob die Operation $\Rightarrow$ assoziativ ist.

Aufgabe 7:

Beweisen Sie mit Hilfe von Wahrheitstabellen, daß die folgenden Formeln Sätze der Aussagenlogik sind.

$x \wedge (y \vee x) \Leftrightarrow x$

$x \vee (y \wedge x) \Leftrightarrow x$

$x \wedge (\neg x \vee y) \Leftrightarrow x \wedge y$

$x \vee (\neg x \wedge y) \Leftrightarrow x \vee y$

Wozu kann man diese Sätze verwenden?

FACHHOCHSCHULE WEDEL
Prof. Dr. Uwe Schmidt

20. Januar 1999

Grundlagen der Programmierung

Übungsaufgabe: Aussagenlogik: Transformation

Aufgabe 1:

Beweisen Sie durch Transformation, daß die folgende Formel (modus ponens) ein Satz der Aussagenlogik ist.

$((x \Rightarrow y) \wedge x) \Rightarrow y$

Aufgabe 2:

Beweisen Sie durch Transformation, daß die folgende Formel (modus tolens) ein Satz der Aussagenlogik ist.

$$((x \Rightarrow y) \wedge \neg y) \Rightarrow \neg x$$

Aufgabe 3:

Beweisen Sie durch Transformation, daß die folgende Formel ein Satz der Aussagenlogik ist. $(y \wedge (x \Rightarrow \neg y)) \Rightarrow \neg x$

Aufgabe 4:

Beweisen Sie durch Umformung und Zurückführen auf die Operationen $\vee$,$\wedge$ und $\neg$, daß $(x \oplus x) \Leftrightarrow \mathsf{false}$ ein Satz der Aussagenlogik ist.

Zeigen Sie ebenfalls durch Umformung, daß false ein neutrales Element bezüglich der Operation $\oplus$ ist.

Aufgabe 5:

Transformieren Sie die folgende Formel in disjunktive Form. Begründen Sie die Transformationsschritte durch Angabe des Namen der Transformation oder durch die zugehörige Formel. $((x \Rightarrow y) \Leftrightarrow (\neg x \Rightarrow \neg y)$

Aufgabe 6:

Prüfen Sie durch Transformation, ob die folgende Formel ein Satz der Aussagenlogik ist (ob sie immer wahr ist).

$$(x \Rightarrow y) \Rightarrow (x \vee \neg y)$$

Aufgabe 7:

Zeigen Sie durch Transformation der Formeln, daß die beiden Formeln

$$(x \Leftrightarrow y) \quad \text{und} \quad [(x \Rightarrow y) \wedge (y \Rightarrow x)]$$

äquivalent sind. Geben sie zusätzlich an, welche Gesetze der Aussagenlogik in den einzelnen Transformationsschritten verwendet werden.

Aufgabe 8:

Welche der folgenden Aussagen sind immer wahr? Überprüfen Sie dies mit Hilfe von Wahrheitstabellen.

1. $((p \Rightarrow q) \Rightarrow r) \Leftrightarrow (p \Rightarrow (q \Rightarrow r))$
2. $((p \Leftrightarrow q) \Leftrightarrow r) \Leftrightarrow (p \Leftrightarrow (q \Leftrightarrow r))$
3. $((p \Leftrightarrow q) \Leftrightarrow r) \Leftrightarrow ((p \Leftrightarrow q) \wedge (q \Leftrightarrow r))$

Fachhochschule Wedel
Prof. Dr. Uwe Schmidt

20. Januar 1999

Grundlagen der Programmierung

Übungsaufgabe: Aussagenlogik: Logeleien

Aufgabe 1:

Ritter und Knechte auf einer Insel (Ritter sagen immer die Wahrheit, Knechte lügen immer):

Drei Personen (b, c, und d) stehen zusammen im Garten, ein Besucher kommt vorbei und fragt b: „Sind sie ein Ritter oder ein Knecht?“ b antwortet, aber der Besucher versteht die Antwort nicht, da b etwas undeutlich spricht. Der Fremde fragt c: „Was hat b gesagt?“ c antwortet: „b sagte, er sei ein Knecht.“ Darauf mischt sich d ein: „Glauben Sie c nicht, er lügt!“

Was sind c und d?

Hinweis: Nur c's und d's Aussagen sind relevant. d's Aussage, daß c lügt, ist gleichwertig damit, das c ein Knecht ist.

Aufgabe 2:

“Behauptungen, nichts als Behauptungen” wettert der Richter, “damit kann man doch keinen Prozeß führen!”

“Aber Herr Vorsitzender”, wirft die Referendarin ein, “auch aus Behauptungen lassen sich Schlüsse ziehen.”

Der Richter darauf: “Papperlapapp!”

Mehr fällt ihm offenbar nicht ein, die eifrige junge Referendarin aber nimmt sich das Protokoll vor.

So sieht es aus:

Behauptung 1: *"Sowohl Behauptung 3 als auch Behauptung 5 als auch Behauptung 6 sind falsch."*

Behauptung 2 *"Sowohl Behauptung 4 als auch Behauptung 6 als auch Behauptung 7 sind wahr."*

Behauptung 3: *"Entweder ist Behauptung 4 oder Behauptung 5 wahr."*

Behauptung 4: *"Behauptung 3 ist wahr, und von den Behauptungen 1 und 2 ist mindestens eine wahr."*

Behauptung 5: *"Entweder ist Behauptung 7 oder Behauptung 3 wahr."*

Behauptung 6: *"Falls Behauptung 1 wahr ist, sind weder Behauptung 4 noch Behauptung 8 wahr."*

Behauptung 7: *"Behauptung 2 ist falsch, aber Behauptung 5 ist wahr."*

Behauptung 8: *"Wenn Behauptung 7 falsch ist, dann ist von den Behauptungen 3 und 4 mindestens eine wahr."*

In dem sich immer noch hinziehenden Streit über die Verwertbarkeit der vorgebrachten Behauptungen ist gerade eine Pause eingetreten, in welche die Referendarin hineinplatzt:

"Ich hab's, Herr Vorsitzender!"

Und sie liest vor, was von den acht Behauptungen zu halten ist.

Welche der Behauptungen sind wahr, welche falsch?

Setzen Sie die acht Behauptungen in aussagenlogische Formeln mit den Variablen b_1 bis b_8.

Versuchen Sie, eine Lösung für die Belegung der Variablen zu finden (Vorsicht!!!). Eine Lösungsstrategie ist schrittweise Variablenelimination und Vereinfachung, eine Wahrheitstabelle bietet eine zweite Möglichkeit (???).

FACHHOCHSCHULE WEDEL
Prof. Dr. Uwe Schmidt

20. Januar 1999

Grundlagen der Programmierung

Übungsaufgabe: Prädikatenlogik

Aufgabe 1:

Gegeben seien die Variablen

var f : array $[0..n-1]$ of Z;
var g : array $[0..m-1]$ of Z;
var e : Z;
var b : Bool ;

Beschreiben Sie folgende Sachverhalte mit Hilfe der Prädikatenlogik

1. Die Variable b soll anzeigen, ob kein Wert aus dem Feld f auch im Feld g vorkommt.
2. Die Variable b soll anzeigen, ob der Wert e sowohl in dem Feld f und im Feld g gespeichert ist oder in beiden nicht vorkommt.
3. Die Variable b soll anzeigen, ob in dem Feld f und im Feld g der Wert e an der gleichen Stelle gespeichert ist.
4. Die Variable b soll anzeigen, ob im Feld f abwechselnd gerade und ungerade Zahlen gespeichert sind. Es soll offen bleiben, welche Eigenschaft das 1. Element hat.
5. Die Variable b soll anzeigen, ob im Feld f ein Wert mindestens 3 mal vorkommt.

Aufgabe 2:

Übersetzen Sie die folgenden englischen Sätze in Formeln der Prädikatenlogik.

1. Everybody loves somebody.
2. Somebody loves somebody.
3. Everybody loves everybody.
4. Nobody loves everybody.
5. Somebody loves nobody.

Aufgabe 3:

Analysieren Sie die folgenden Aussagen. Dabei ist die Grundmenge, über die Aussagen gemacht wird, die Menge aller Software, hier mit $Software$ bezeichnet, diese ist nicht leer.

Es werden folgende einstellige Elementaraussagen verwendet:

$vonWW(sw)$
: für ein Stück Software, das von der Firma *WinzigWeich* gebaut wurde

$preiswert(sw)$
: für ein Stück preiswerter Software, nicht preiswerte Software ist teuer

$fehleranfaellig(sw)$
: für fehleranfällige Software, nicht fehleranfällige Software ist zuverlässig

Die Aussagen über Software als prädikatenlogische Formeln

1. $\exists\, sw \in Software \bullet (vonWW(sw) \wedge fehleranfaellig(sw)) \Rightarrow preiswert(sw)$
2. $\exists\, sw \in Software \bullet vonWW(sw) \wedge (preiswert(sw) \wedge fehleranfaellig(sw))$
3. $\exists\, sw \in Software \bullet (\neg vonWW(sw) \wedge preiswert(sw)) \Rightarrow \neg fehleranfaellig(sw)$
4. $\exists\, sw \in Software \bullet preiswert(sw) \Rightarrow (vonWW(sw) \wedge fehleranfaellig(sw))$
5. $\exists\, sw \in Software \bullet preiswert(sw) \Rightarrow \neg(vonWW(sw) \wedge \neg preiswert(sw))$
6. $\forall\, sw \in Software \bullet vonWW(sw) \Rightarrow preiswert(sw) \wedge fehleranfaellig(sw)$
7. $\forall\, sw \in Software \bullet (\neg fehleranfaellig(sw) \vee \neg preiswert(sw)) \Rightarrow \neg vonWW(sw)$
8. $\forall\, sw \in Software \bullet \neg vonWW(sw) \vee (preiswert(sw) \wedge \neg fehleranfaellig(sw))$
9. $\forall\, sw \in Software \bullet fehleranfaellig(sw) \Rightarrow (vonWW(sw) \wedge \neg fehleranfaellig(sw))$
10. $\forall\, sw \in Software \bullet (\neg vonWW(sw) \vee \neg preiswert(sw)) \vee fehleranfaellig(sw)$
11. $\forall\, sw \in Software \bullet preiswert(sw) \Rightarrow (vonWW(sw) \wedge fehleranfaellig(sw))$
12. $\forall\, sw \in Software \bullet vonWW(sw) \Rightarrow (\neg preiswert(sw) \wedge fehleranfaellig(sw))$

Geben sie für die folgenden Aussagen die Nummer(n) von **gleichwertigen** Formeln an, Mehrfachnennungen sind möglich, gibt es keine Formel tragen Sie 0 an die vorgesehene Stelle ein.

1. Es gibt WinzigWeich–Software, die ist preiswert und zuverlässig.
2. Software von WinzigWeich ist immer preiswert und zuverlässig.
3. Software von WinzigWeich ist weder preiswert noch zuverlässig.
4. Jede preiswerte Software ist von WinzigWeich und fehleranfällig.
5. Es gibt teure Software oder fehleranfällige WinzigWeich–Software.
6. Software von WinzigWeich ist immer preiswert und fehleranfällig.
7. Jede preiswerte Winzigweich–Software ist fehleranfällig.
8. Es gibt preiswerte, aber fehleranfällige WinzigWeich-Software.
9. Falsch.
10. Wahr.

Fachhochschule Wedel
Prof. Dr. Uwe Schmidt

20. Januar 1999

Grundlagen der Programmierung

Übungsaufgabe: Spezifikationen

Aufgabe 1:

Gegeben seien die folgenden Variablen

var f : array $[0..n-1]$ of Z;
var w : Z;
var b : Bool ;
var i, j : N_0;

Spezifizieren Sie folgende Aufgaben mit Vor- und Nachbedingungen

1. Die Variable b soll anzeigen, ob das Feld f aufsteigend ohne Duplikate sortiert ist.
2. Die Variable b soll anzeigen, ob im Feld f ein Wert 2 mal vorkommt.
3. Wenn in dem Feld f ein Wert 2 mal gespeichert ist, so sollen die Variablen i und j auf diese Positionen zeigen, sonst sollen sie gleiche Werte enthalten.
4. Die Variable b soll anzeigen, ob 3 verschiedene Werte in dem Feld f gespeichert sind.
5. Die Variable w soll weder den kleinsten noch den größten Wert des Feldes f enthalten.
6. Die Variable w soll einen Wert zwischen dem kleinsten und dem größten Wert des Feldes enthalten.
7. Die Variable w soll einen Wert aus dem Feld f enthalten, dieser Wert soll zwischen dem kleinsten und dem größten Wert aus f liegen.

Aufgabe 2:

Gegeben sei das folgende Feld

var f : array $[0..n-1]$ of Z;

Spezifizieren Sie folgende Sachverhalte mit Hilfe der Prädikatenlogik

1. f enthält keine 0-en.
2. f enthält höchstens eine 0.
3. f enthält mindestens eine 0.
4. f enthält genau eine 0.
5. f enthält mindestens zwei 0-en.
6. f enthält zwei 0-en.
7. f enthält zwei benachbarte 0-en.
8. f enthält zwei 0-en, die nicht benachbart sind.
9. f enthält zwei 0-en, die durch eine Anzahl von Werten ungleich 0 getrennt sind.
10. f enthält zwei 0-en, die durch eine gerade Anzahl von Werten ungleich 0 getrennt sind.

Gibt es unter diesen Aussagen gleichwertige?

Welche Aussagen sind Konsequenz von darauf folgenden Aussagen (z.B. welche Aussagen impliziert Aussage 1)?

FACHHOCHSCHULE WEDEL
Prof. Dr. Uwe Schmidt

20. Januar 1999

Grundlagen der Programmierung

Übungsaufgabe: Zuweisungen

Aufgabe 1:

Gegeben seinen folgende Variablen

var $i, m, p, q, r, x, y : \mathsf{N}_0$
var $s : \mathsf{B}$
var $am : \mathsf{R}$
var f : array $[0..n-1]$ of N_0

Berechnen Sie zu den folgenden Vor- und Nachbedingungen einen geeigneten Ausdruck E, so daß die Zuweisung korrekt arbeitet. Verstärken sie die Vorbedingung auf geeignete Weise, wenn der Ausdruck E nicht immer auswertbar ist (illegale Indizes, Division durch 0, ...).

1. $\{$ true $\}$ $i, m := 0, E$ $\{$ $m = i!$ $\}$
2. $\{$ true $\}$ $i, p := 0, E$ $\{$ $p = i * y$ $\}$
3. $\{$ true $\}$ $i, s := 1, E$ $\{$ $s \Leftrightarrow (\forall\, 0 < j < i \bullet f[j-1] < f[j]$ $\}$
4. $\{$ true $\}$ $q, r := 0, E$ $\{$ $x = q * y + r$ $\}$
5. $\{$ true $\}$ $i, m := 0, E$ $\{$ $\forall\, 0 \leq j < i \bullet f[j] \leq m$ $\}$
6. $\{$ true $\}$ $i, m := 0, E$ $\{$ $\forall\, 0 \leq j \leq i \bullet f[j] \leq m$ $\}$
7. $\{$ true $\}$ $i, m := 1, E$ $\{$ $\forall\, 0 \leq j < i \bullet f[j] \leq m$ $\}$
8. $\{$ true $\}$ $i, x := 0, E$ $\{$ $x = \sum_{j=0}^{i-1} f[j]$ $\}$
9. $\{$ true $\}$ $i, x := 0, E$ $\{$ $x = \sum_{j=0}^{i} f[j]$ $\}$
10. $\{$ $x = \sum_{j=0}^{i-1} f[j]$ $\}$ $i, x := i + 1, E$ $\{$ $x = \sum_{j=0}^{i-1} f[j]$ $\}$
11. $\{$ $am = \frac{1}{i} \sum_{j=0}^{i-1} f[j]$ $\}$ $i, am := i + 1, E$ $\{$ $am = \frac{1}{i} \sum_{j=0}^{i-1} f[j]$ $\}$
12. $\{$ $true$ $\}$ $i, am := 1, E$ $\{$ $am = \frac{1}{i} \sum_{j=0}^{i-1} f[j]$ $\}$
13. $\{$ $true$ $\}$ $i, am := 0, E$ $\{$ $am = \frac{1}{i} \sum_{j=0}^{i-1} f[j]$ $\}$

Aufgabe 2:

Gegeben seinen folgende Variablen

var $i, j : \mathsf{N}_0$

und die Nachbedingung $P: \ j = \sum_{k=0}^{i} k$

Berechnen Sie zu den folgenden Programmstücken die zugehörige Vorbedingung V aus:

1. $\{\ V\ \}\ i, j := i + 1, j + i\ \{\ P\ \}$
2. $\{\ V\ \}\ j, i := j + i, i + 1\ \{\ P\ \}$
3. $\{\ V\ \}\ j := j + i; i := i + 1\ \{\ P\ \}$
4. $\{\ V\ \}\ i := i + 1; j := j + i\ \{\ P\ \}$

Aufgabe 3:

Gegeben seinen folgende Variablen

var $i, j, k : \mathsf{N}_0$

Berechnen Sie zu den folgenden Programmstücken und Nachbedingungen die zugehörigen Vorbedingungen V:

1. $\{\ V\ \}\ i, j, k := j, k, i\ \{\ i = c_1 \wedge j = c_2 \wedge k = c_3\ \}$
2. $\{\ V\ \}\ i := j;\ j := k;\ k := i\ \{\ i = c_1 \wedge j = c_2 \wedge k = c_3\ \}$
3. $\{\ V\ \}\ i := j - i;\ j := j - i;\ i := i + j\ \{\ i = c_1 \wedge j = c_2\ \}$
4. $\{\ V\ \}\ i, j := i - k, j + k\ \{\ i + j = c_1\ \}$

Aufgabe 4:

Gegeben seinen folgende Variablen

var $i, j, k, t : \mathsf{Z}$

Geben Sie für die folgenden Mehrfachzuweisungen gleichwertige Programmstücke an, die mit Folgen von Einfachzuweisungen arbeiten. Überprüfen Sie Ihre Lösung, indem Sie zu der Nachbedingung P:

$$i = c_1 \wedge j = c_2 \wedge k = c_3$$

die zugehörigen Vorbedingungen berechnen.

1. $i, j, k := j, k, i$
2. $i, j, k := j, i, k$
3. $i, j, k := i + j, i - j, j - i$
4. $i, j, k := i + 1, j + i, k + j$

FACHHOCHSCHULE WEDEL
Prof. Dr. Uwe Schmidt

20. Januar 1999

Grundlagen der Programmierung

Übungsaufgabe: Verzweigungen

Aufgabe 1:

Gegeben seien die folgenden Variablen

var $x, y, r : integer$;
$overflow$: B

integer sei dabei ein Teilbereich (ein Intervall) aus den ganzen Zahlen, $maxint$ sei die größte Zahl, $minint$ die kleinste Zahl dieses Bereichs (z.B für Pascal: $maxint = 2^{31} - 1$ und $minint = -2^{31}$).

Das folgende Programmstück berechnet die Summe der Variablen x und y in der Variablen r, es berücksichtigt aber Überläufe, d.h. wenn ein Überlauf bei der Addition auftritt wird $maxint$ oder $minint$ als Resultat geliefert. Außerdem ist sichergestellt, daß bei den Operationen in den Tests nie der *integer* Zahlenbereich verlassen wird.

```
if (x ≥ 0)⇔(y < 0)
        then
        r := x + y
else
if x ≥ 0
        then
        if x ≤ maxint − y
                then
                r := x + y
        else
                r := maxint
        end if
else
        if minint − x ≤ y
                then
                r := x + y
        else
                r := minint
        end if
end if
end if
```

Begründen Sie mit Hilfe von Zusicherungen in dem Programmtext, daß das oben beschriebene Verhalten sichergestellt ist.

Die Spezifikation für dieses Programmstück hat folgendes Aussehen:

$\{\ maxint \geq x \wedge x \geq minint$
$\quad \wedge\ maxint \geq y \wedge y \geq minint\ \}$

„obiges Programmstück zur Addition“

$\{\ (x + y < minint \Rightarrow r = minint)$
$\wedge\ (minint \leq x + y \wedge x + y \leq maxint \Rightarrow r = x + y)$
$\wedge\ (maxint < x + y \Rightarrow r = maxint)\ \}$

Erweitern Sie das Programmstück so, daß in der Booleschen Variablen $overflow$ ein Über– oder Unterlauf angezeigt wird.

Aufgabe 2:

Gegeben seien die ganzzahligen Variablen a, b, c und d und eine Boolesche Variable r.

Transformieren Sie das folgende Programmstück in ein gleichwertiges, das nur noch aus einer Zuweisung besteht und höchstens drei arithmetische Vergleiche enthält.

```
if ((a > 0) ∧ (b ≤ 0))
        ∨ ((a ≤ 0) ∧ (b > 0))
then
        if c ≥ d
        then r := false
        else r := true
        end if
else
        if c < d
        then r := false
        else r := true
        end if
end if
```

Fachhochschule Wedel
Prof. Dr. Uwe Schmidt

20. Januar 1999

Grundlagen der Programmierung

Übungsaufgabe: Schleifen

Aufgabe 1:

Lineare Suche: Entwickeln Sie eine Funktion *suche*, die für eine natürliche Zahl $n > 1$ den kleinsten Teiler von n berechnet, der größer als 1 ist.

Das Prädikat für die Suche ist also

$P(i) : n \text{ mod } i = 0$

Warum ist hier die Terminierung gesichert?

Aufgabe 2:

Lineare Suche: Entwickeln Sie ein Programm, daß für eine ganze Zahl $n > 1$ den größten Teiler von n, der kleiner als n ist, berechnet (lineare Suche mit dekrementieren der Laufvariablen).

Warum ist hier die Terminierung gesichert?

Wie kann man die Aufgabenstellung lösen unter Verwendung der 1. Aufgabe.

Aufgabe 3:

Programmverfeinerung: Das folgende Programm berechnet für die Zahlenfolge i^3 den n-ten Folgenwert in der Variablen y.

```
var i, n, y : N0;
{ true }
i, y := 0, 0;
while i ≠ n do
      i, y := i + 1, (i + 1)^3
end while
{ y = n^3 }
```

Verfeinern Sie dieses Programm so, daß es keine Potenzierung und keine Multiplikation mehr enthält.

Aufgabe 4:

Programmverfeinerung: Das folgende Programm berechnet das Maximum der Elemente eines Feldes.

```
var f : array [0..n − 1] of R;
var i : N0;
var maxi, mini, maxi2 : R;

{ n > 0 }
i, maxi := 1, f[0];
while i ≠ n do
      maxi := max (maxi, f[i]);
      i := i + 1
end while
{ maxi = MAX_{j=0}^{n−1} f[j] }
```

Hier wird angenommen, daß eine Operation max als elementare Operation zur Verfügung steht.

1. verfeinern Sie das Programm so, daß in einer Variablen $mini$ gleichzeitig das Minimum berechnet wird (mit Hilfe einer elementaren Operation min). Wie müssen Nachbedingung und Invariante erweitert werden?
2. verfeinern Sie das Programm so, daß in einer Variablen $maxi_2$ das 2.-größte Element im Feld berechnet wird. Wenn es kein 2.-größtes Element gibt, so soll $maxi = maxi_2$ gelten. Wie müssen Nachbedingung und Invariante erweitert werden?

FACHHOCHSCHULE WEDEL
Prof. Dr. Uwe Schmidt

20. Januar 1999

Grundlagen der Programmierung

Übungsaufgabe: Schleifen

Aufgabe 1:

Die Fibonacci Zahlenfolge ist auf den natürlichen Zahlen ab 0 wie folgt definiert

$$
\begin{aligned}
&fib(0) = 0 \\
&fib(1) = 1 \\
&fib(i) = fib(i-1) + fib(i-2) \quad \text{für } i > 1
\end{aligned}
$$

Vervollständigen Sie das folgende Programmstück so, daß es diese Zahlenfolge berechnet.

```
var i, n, x0, x1 : N0;
i, x0, x1 := 0, 0, 1;
        { I }
while B do
        i, x0, x1 := i + 1, E0, E1
end while
        { x0 = fib(n) }
```

Geben Sie hierzu eine geeignete Invariante an, und leiten Sie aus dieser die Ausdrücke E_0 und E_1 ab. Die Invariante muß also etwas über x_0 und über x_1 aussagen.

Aufgabe 2:

Gegeben seien zwei Felder

```
var f1 : array [0..n1 − 1] of R;
var f2 : array [0..n2 − 1] of R;
```

1. Spezifizieren Sie ein Programm, das testet ob alle Elemente aus f_1 auch in f_2 vorkommen.
2. Entwickeln Sie aus der Spezifikation auf systematische Weise ein Programm (durch schrittweise Verfeinerung), das diese Eigenschaft testet.

Aufgabe 3:

Gegeben seien 2 natürliche Zahlen p und q. Entwerfen Sie ein Programm, das den größten gemeinsamen Teiler ($ggt(p,q)$) von p und q in einer Variablen x berechnet. Nutzen Sie dabei die Eigenschaft aus, daß der ggt von 2 Zahlen x und y auch die Differenz $x-y$ teilt.

var $p, q, x, y : Nat$;
$\{ V \}$
$x, y := p, q$;
$\{ I \}$
while B do
S
end while
$\{ ggt(x,y) = ggt(p,q) \wedge x = y \}$

Wählen Sie I, B und S geeignet, und sichern Sie die Terminierung mit einer geeigneten Varianten t und einer davon abhängigen Vorbedingung V.

Aufgabe 4:

Gegeben seien die folgenden Variablen:

var f : array $[0..n-1]$ of Z;
var b : B;
var i : N_0

Konstruieren Sie ein Programmstück, das folgendes Prädikat in der Variablen b berechnet.

$$\forall\, 0 < j < n \bullet f[j] \geq 0 \Leftrightarrow f[j-1] < 0$$

Das Programmstück soll systematisch mit den Techniken aus der Vorlesung aus dem $\forall$ Quantor abgeleitet werden, es soll so arbeiten, daß keine überflüssigen Berechnungen gemacht werden, nachdem das Resultat feststeht.

Fachhochschule Wedel
Prof. Dr. Uwe Schmidt

20. Januar 1999

Grundlagen der Programmierung

Übungsaufgabe: Prozeduren, Rekursion und Schleifen

Aufgabe 1:

Entwerfen Sie einen Algorithmus (eine Funktion) mit Namen ps, der für eine reelle Zahl x und eine natürliche Zahl n die Summe der Potenzen x^i berechnet für $0 \leq i < n$, also $1 + x + x^2 + \ldots + x^{n-1}$.

Nutzen Sie hierbei die Eigenschaft aus, daß
$1 + x + x^2 + \ldots + x^{n-1} \;=\; 1 + x * (1 + x + \ldots + x^{n-2})$
ist.

Die Funktion ps soll ohne Zuweisungen arbeiten, also als Rumpf nur einen Ausdruck enthalten.

$ps(x : \mathsf{R}, n : \mathsf{N}_0) : \mathsf{R}$

Ist dieser Algorithmus zur Parallelverarbeitung geeignet?

Ist dieser Algorithmus einfach in eine Schleife zu transformieren, da er mit einer Endrekursion arbeitet?

Aufgabe 2:

Das Multiplizieren $i * j$ von Zahlen $i, j \in \mathsf{N}_0$ kann durch wiederholte Addition implementiert werden. Dieser Algorithmus benötigt aber $j - 1$ Additionen. Einen sehr viel schnelleren Algorithmus erhält man, wenn man die folgenden beiden Gleichungen ausnutzt (logarithmisches Multiplizieren)

$i * j = (i + i) * (j/2)$ für gerades j

$i * j = i + i * (j - 1)$ für ungerades j

Als Basisfall (direkt berechenbaren Fall) verwenden Sie bitte den Fall $j = 0$.

1. Schreiben Sie für das Multiplizieren einen Algorithmus $mult_1(i : \mathsf{N}_0; j : \mathsf{N}_0) : \mathsf{N}_0$ mit 2 Parametern i und j, der mit einer Rekursion arbeitet und die beiden Gleichungen ausnutzt, also nur Addition, Division durch 2, Test auf gerade oder ungerade Zahlen (modulo 2) und Subtraktion verwendet.
2. Schreiben Sie für das Multiplizieren einen gleichwertigen Algorithmus
   $mult_2(i : \mathsf{N}_0; j : \mathsf{N}_0) : \mathsf{N}_0$,
   der nicht mit Rekursion arbeitet, sondern mit einer Schleife. $mult_2$ soll dabei systematisch durch Programmtransformation aus $mult_1$ entwickelt werden.

Aufgabe 3:

Konstruieren Sie eine Funktion *halbeSumme*, die für alle natürlichen Zahlen $n \geq 0$ die Summe aller 2-ten Zahlen berechnen soll, d.h. für gerade Zahlen n ist die Summe $0 + 2 + 4 + \cdots + (n-2) + n$ zu berechnen, für ungerade n die Summe $1 + 3 + \cdots + (n-2) + n$

Schreiben Sie eine linear rekursive Funktion $halbeSumme(n : \mathsf{N}_0) : \mathsf{N}_0$ zur Berechnung dieser Summe. In diser Funktion ist keine Fallunterscheidung in gerade und ungerade Zahlen erforderlich.

Transformieren Sie diese Funktion in eine gleichwertige, die aber mit einer Schleife arbeitet. Machen Sie dabei (in Kladde) in einem Zwischenschritt eine Transformation in eine Hilfsfunktion, die mit einer Endrekursion arbeitet, und setzen Sie daraus die Lösung zusammen.

Vergleichen Sie die folgende Funktion mit der Funktion *halbeSumme*.

$f(n : \mathsf{N}_0) : \mathsf{N}_0$
  if n mod $2 = 0$
  then $n * (n + 2)$ div 4
  else $(n + 1) * (n + 1)$ div 4

Welche Unterschiede gibt es bezüglich der berechneten Resultate?

Welche Unterschiede gibt es bezüglich der Anzahl der ausgeführten Operationen?

Fachhochschule Wedel
Prof. Dr. Uwe Schmidt

20. Januar 1999

Grundlagen der Programmierung

Übungsaufgabe: Algorithmen, Rekursion und Schleifen

Aufgabe 1:

Es sei der folgende Algorithmus zur Berechnung des Minimums zweier Zahlen gegeben

$min(x : \mathsf{N}_0; y : \mathsf{N}_0) : \mathsf{N}_0$
    if $x > y$
    then y
    else x

Erklären Sie, wie dieser Algorithmus bei einem Aufruf von

$min(min(5,3), min(4,7))$

abgearbeitet wird, indem Sie den Algorithmus–Rumpf gemäß der Semantik von Algorithmenanwendungen an den Aufrufstellen einsetzen.

Aufgabe 2:

Gegeben sei die folgende rekursive Funktion zur Berechnung des größten gemeinsamen Teilers zweier natürlicher Zahlen ≥ 1.

$ggt(x : \mathsf{N}_1; y : \mathsf{N}_1) : \mathsf{N}_1$
    if $x = y$
    then x
    else
    if $x > y$
    then $ggt(x - y, y)$
    else $ggt(x, y - x)$

Transformieren Sie diese Funktion in einen gleichwertigen Algorithmus, in dem die Rekursion durch eine Schleife ersetzt ist.

Welche Eigenschaft der Funktion ermöglicht diese Transformation?

Warum terminiert dieser Algorithmus für alle Argumente ≥ 1?

Aufgabe 3:

Gegeben sei der folgende Algorithmus zur Berechnung der Summe der ersten n Quadratzahlen. (n ≥ 0).

$qsum(n : \mathsf{N}_0) : \mathsf{N}_0$
    if $n = 0$
    then 0
    else $n^2 + qsum(n-1)$

1. Transformieren Sie diesen Algorithmus in einen gleichwertigen Algorithmus, der aber nur noch eine Endrekursion enthält.
2. Transformieren Sie diesen neuen Algorithmus in einen gleichwertigen Algorithmus, der mit einer Schleife arbeitet.

Aufgabe 4:

Die Zahlenfolge G sei auf den natürlichen Zahlen ab 0 wie folgt definiert:

$G(0) = 0$

$G(n) = n - G(G(n-1)) \qquad n > 0$

1. Schreiben Sie eine rekursive Funktion zur Berechnung von G
2. Berechnen Sie den Wert $G(2)$, indem Sie die Funktion schrittweise einsetzen und auswerten
3. Läßt sich die rekursive Formulierung einfach in einen Algorithmus umformen, der mit einer Schleife arbeitet und keine Funktionsaufrufe mehr enthält?
   Wenn ja, welche Art von Transformationen sind notwendig, wenn nein, warum nicht?

Aufgabe 5:

Entwickeln Sie eine Funktion $teilerSummeGleich(n : \mathsf{N}_1) : \mathsf{B}$ zum Testen, ob eine Zahl $n \in \mathsf{N}_1$ gleich der Summe ihrer echten Teiler ist, z.B. ist die Teilersumme von 12 gleich $2 + 3 + 4 + 6$ gleich 15

Ein echter Teiler t von n hat die Eigenschaft $1 < t \wedge t < n$. Zerlegen Sie die Aufgabe in Teilaufgaben: eine Funktion zur Berechnung der Teilersumme: $teilerSumme(n : \mathsf{N}_1) : \mathsf{N}_0$, eine zur Berechnung des kleinsten Teilers: $kleinsterTeiler(n : \mathsf{N}_1) : \mathsf{N}_1$ und eine zum Suchen von Teilern: $findeTeiler(n : \mathsf{N}_1, i : \mathsf{N}_1) : \mathsf{N}_1$ und, wenn notwendig, weiterer Funktionen.

Entwickeln Sie eine 2. Funktion $primFaktorenSumme(n : \mathsf{N}_1) : \mathsf{N}_0$, die die Summer aller Primfaktoren berechnet, für 12 ergibt sich der Wert $2 + 2 + 3$ gleich 7.

Randbedingung: Alle Funktionen sollen ohne Schleifen also rekursiv arbeiten.

FACHHOCHSCHULE WEDEL
Prof. Dr. Uwe Schmidt

20. Januar 1999

Grundlagen der Programmierung

Übungsaufgabe: Rekursion und Schleifen

Aufgabe 1:

Die Zahlenfolge Q sei auf den natürlichen Zahlen ab 1 wie folgt definiert:

$Q(1) = 1$

$Q(2) = 1$

$Q(n) = Q(n - Q(n-1)) + Q(n - Q(n-2)) \qquad n > 2$

1. Schreiben Sie einen rekursiven Algorithmus zur Berechnung von Q. Dieser Algorithmus sollte nur eine if Verzweigung enthalten.
2. Berechnen Sie den Wert $Q(3)$, indem Sie den Algorithmus schrittweise einsetzen und auswerten
3. Läßt sich die rekursive Formulierung einfach in einen Algorithmus mit Schleifen umformen?
   Wenn ja, welche Art von Transformationen sind notwendig, wenn nein, warum nicht?
4. Warum terminiert der Algorithmus für alle natürlichen Zahlen?

Aufgabe 2:

Die *Collatz*–Funktion ist eine auf den natürlichen Zahlen ab 1 definierte Zahlenfoge. Sie wird durch folgende Gleichungen festgelegt:

$$\begin{array}{lcll} C(1) & = & 0 & \\ C(n) & = & 1 + C(n \text{ div } 2) & n \textit{ gerade} \\ C(n) & = & 1 + C(3 * n + 1) & n \textit{ ungerade},\ n > 1 \end{array}$$

1. Schreiben Sie einen rekursiven Algorithmus, der der mathematischen Definition entspricht.
2. Transformieren Sie den Algorithmus in einen zweiten Algorithmus, der mit Endrekursion arbeitet.
3. Transformieren Sie diesen Algorithmus in einen äquivalenten Algorithmus, der mit einer Schleife arbeitet.
4. Terminiert dieser Algorithmus immer? Warum ist diese Frage nicht einfach zu beantworten?

Aufgabe 3:

Das Potenzieren i^j von Zahlen $i, j \in \mathbb{N}_0$ kann durch wiederholte Multiplikation implementiert werden. Dieser Algorithmus benötigt aber mindestens $j-1$ Multiplikationen. Einen sehr viel schnelleren Algorithmus erhält man, wenn man die folgenden beiden Gleichungen ausnutzt

$i^j = (i * i)^{j/2}$ für gerades j

$i^j = i * i^{j-1}$ für ungerades j

Schreiben Sie für das Potenzieren einen Algorithmus *power* mit 2 Parametern i und j, der mit einer Rekursion (ohne Schleifen und Zuweisungen) arbeitet und die beiden Gleichungen ausnutzt. Als Basisfall (direkt berechenbaren Fall) verwenden Sie bitte den Fall für $j = 0$: $i^0 = 1$ für alle i.

Geben Sie an, wie ein Aufruf von $power(3, 5)$ schrittweise durch rekursive Aufrufe von *power* mit anderen Argumenten berechnet wird.

Aufgabe 4:

Die Quersumme einer natürlichen Zahl $n \geq 0$ ist eine Zahl zwischen 0 und 9. Sie berechnet sich aus der Summe der Ziffern von n. Ist diese Summe > 9 so wird der Prozeß der Quersummenbildung wiederholt.

Entwickeln Sie hierfür durch schrittweise Verfeinerung eine Funktion $quersumme$ mit einem Parameter n. Verwenden sie für die Konstruktion eine Funktion $ziffernsumme$ mit ebenfalls einem Parameter zur Summenbildung der Ziffern.

Beide Funktionen sollen ohne Schleifen und Zuweisungen arbeiten. Für $quersumme$ ergibt sich dann eine endrekursive Lösung, für $ziffernsumme$ eine linear rekursive Form.

Erklären Sie, warum die Funktionen immer terminieren.

Transformieren Sie die endrekursive Form der Funktion $quersumme$ in eine Schleife.

Transformieren Sie die linear rekursive Form der Funktion $ziffernsumme$ in eine Schleife. Hinweis: machen Sie dabei die Transformation in Kladde in zwei Schritten und setzen Sie diese beiden Teile zu einem Funktionsrumpf zusammen.

Verallgemeinern Sie die rekursiven Funktionen so, daß nicht nur die Quersumme im 10-er Zahlensystem berechnet werden kann, sondern zu einer beliebigen Basis $b \geq 2$

Fachhochschule Wedel
Prof. Dr. Uwe Schmidt

20. Januar 1999

Grundlagen der Programmierung

Übungsaufgabe: Automaten und Formale Sprachen

Aufgabe 1:

Konstruieren Sie einen endlichen Automaten $A = (I, Q, \delta, q_0, F)$ mit dem Eingabealphabet $I = \{1, 2, 3\}$. Der Automat soll alle Zahlen erkennen, bei denen nie eine kleinere auf eine größere Ziffer folgt. Die akzeptierte Sprache enthält also z.B. die Wörter 1, 2, 3, 123, 111, 112233, 2223, aber nicht 21 und 332. Das leere Wort soll ebenfalls nicht akzeptiert werden. Hinweis: 4 Zustände reichen.

Die Zustandsmenge Q:

Der Startzustand q_0:

Die Endzustandsmenge F:

Die Übergangstabelle δ als Grafik (Zustands–Übergangs-Diagramm):

Die Übergangstabelle δ als Tabelle:

Konstruieren Sie eine rechtslineare Grammatik für die oben beschriebene Sprache.

Aufgabe 2:

Konstruieren Sie eine rechtslineare Grammatik $G = (T, N, P, S)$ für das Erkennen von rationalen Zahlen in Dezimaldarstellung. Das Alphabet der Terminatsymbole sei dabei $T = \{0, \ldots, 9, .\}$. Die rationalen Zahlen sollen dabei mindestens eine Ziffer vor dem Dezimalpunkt und eine Ziffer nach dem Dezimalpunkt besitzen.

Die Menge der Nichtterminalsymbole N:

Das Startsymbol S:

Die Menge der Produktionen P:

Aufgabe 3:

Konstruieren Sie eine kontextfreie Grammatik $G = (T, N, P, S)$ für das Erkennen von rationalen Zahlen in Dezimaldarstellung. Verwenden Sie dabei die BNF-Notation. Das Alphabet der Terminatsymbole sei dabei $T = \{0, \ldots, 9, .\}$. Die rationalen Zahlen sollen dabei mindestens eine Ziffer vor dem Dezimalpunkt und eine Ziffer nach dem Dezimalpunkt besitzen.

Die Menge der Nichtterminalsymbole N:

Das Startsymbol S:

Die Menge der Produktionen P:

Aufgabe 4:

Konstruieren Sie eine kontextfreie Grammatik $G = (T, N, P, S)$ für aussagenlogische Ausdrücke mit beliebigen Variablennamen und den Konstanten true und false. Verwenden Sie dabei die BNF-Notation. Das Alphabet der Terminatsymbole sei dabei $T = \{a, \ldots, z, 0, \ldots, 9, \mathsf{true}, \mathsf{false}, \wedge, \vee, \oplus, \Rightarrow, \Leftrightarrow, \oplus, (,)\}$. Es sollen dabei die Prioritäten gemäß der Konventionen in der Vorlesung berücksichtigt werden. Beliebig tiefe Klammerung soll möglich sein, Variablennamen sollen mit einem Buchstaben beginnen, sie dürfen beliebig lang sein und Buchstaben und Ziffern enthalten.
