Fachhochschule Wedel
Prof. Dr. Uwe Schmidt

13. April 1999

Grundlagen der Programmierung

Übungsaufgabe: Aussagenlogik: Boolesche Operationen und Wahrheitstabellen

---

Aufgabe 1:

Zeigen Sie mit Hilfe einer Wahrheitstabelle, daß das Distributivgesetz für $\vee$ und $\Leftrightarrow$ gilt

$p \vee (q \Leftrightarrow r) \Leftrightarrow (p \vee q \Leftrightarrow p \vee r)$

Aufgabe 2:

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?

Aufgabe 3:

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. $((p \Leftrightarrow q) \Leftrightarrow r) \Leftrightarrow ((p \Leftrightarrow q) \wedge (q \Leftrightarrow r))$

Aufgabe 4:

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

Aufgabe 5:

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 6:

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

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

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

FACHHOCHSCHULE WEDEL
Prof. Dr. Uwe Schmidt

13. April 1999

Grundlagen der Programmierung

Übungsaufgabe: Aussagenlogik: Transformation

Aufgabe 1:

Zeigen Sie durch Transformation, daß die folgenden Aussagen Sätze der Aussagenlogik sind

$(b \Rightarrow c) \Rightarrow (a \vee b \Rightarrow a \vee c)$

$(b \Rightarrow c) \Rightarrow (a \wedge b \Rightarrow a \wedge c)$

Aufgabe 2:

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 3:

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 4:

Zeigen Sie durch Transformation, daß die folgende Aussage ein Satz der Aussagenlogik ist.

$$(a \wedge b) \Leftrightarrow (a \vee b) \Leftrightarrow (a \Leftrightarrow b)$$

Hinweis:
Transformieren Sie $(a \wedge b) \Leftrightarrow (a \vee b)$ in $(a \Leftrightarrow b)$.

Aufgabe 5:

Vereinfachen Sie den folgenden Ausdruck:

$\neg((a \Leftrightarrow b) \wedge (\neg a \Rightarrow b) \wedge (\neg b \Rightarrow a))$

Fachhochschule Wedel
Prof. Dr. Uwe Schmidt

13. April 1999

Grundlagen der Programmierung

Übungsaufgabe: Aussagenlogik: Logeleien

Aufgabe 1:

"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 (???).

Aufgabe 2:

Neulich traf ich zwei Knuken: von ihnen hoffte ich zu erfahren, was die sechs Böwendolpen tun. Ich wußte, daß jede Böwendolpe genau eine dieser Tätigkeiten ausübt:

- *den Prunder blahmen*
- *am Sautraff sontern*
- *in der Hürn gauffen*
- *ihr Dückerneupchen jugeln*

Die Knuken gaben mir die folgenden Auskünfte

Amsknuke: "Wenn die Zirrböwendolpe nicht ihr Dückerneupchen jugelt, dann sontert sie am Sautraff."

Bonzknuke: "Wenn die Flontböwendolpe in der Hürn gaufft, blahmt die Noffböwendolpe den Prunder."

Amsknuke: "Wenn die Belkböwendolpe den Prunder blahmt, dann sontert die Noffböwendolpe am Sautraff."

Bonzknuke: Gaufft die Zirrböwendolpe in der Hürn, so jugelt die Ilkböwendolpe nicht ihr Dückerneupchen."

Amsknuke: "Wenn die Sorpböwendolpe nicht in der Hürn gaufft, tut dies die Ilkböwendolpe."

Bonzknuke: "Gaufft die Belkböwendolpe nicht in der Hürn, so blahmt die Noffböwendolpe nicht den Prunder."

Amsknuke: "Wenn die Ilkböwendolpe nicht am Sautraff sontert, dann jugelt die Noffböwendolpe ihr Dückerneupchen."

Bonzknuke: "Blahmt die Belkböwendolpe den Prunder, so sontert die Sorpböwendolpe am Sautraff."

Amsknuke: "Sontert die Flontböwendolpe nicht am Sautraff, so blahmt die Zirrböwendolpe den Prunder."

Bonzknuke: "Wenn die Flontböwendolpe den Prunder blahmt, sontert die Noffböwendolpe am Sautraff."

Knuken sagen entweder stets die Wahrheit, oder sie lügen immer; mindestens eine der beiden hier erwähnten Knuken ist eine Lügnerin gewesen. Welche Böwendolpe tut was?

Setzen Sie diese Logelei in aussagenlogische Formeln um. Hierzu müssen als erstes die Booleschen Variablen bestimmt werden, dann die Aussagen der Knuken in Formeln umgesetzt werden, und es müssen die Randbedingungen, die Logikarten und die Einschränkungen, umgesetzt werden. Vorsicht beim Lösen.

Fachhochschule Wedel
Prof. Dr. Uwe Schmidt

13. April 1999

Grundlagen der Programmierung

Übungsaufgabe: Prädikatenlogik

Aufgabe 1:

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 2:

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 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

13. April 1999

Grundlagen der Programmierung

Übungsaufgabe: Prädikatenlogik und Spezifikationen

Aufgabe 1:

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)?

Aufgabe 2:

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 3:

Gegeben seien die folgenden Variablen

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

Spezifizieren Sie folgende Aufgaben mit Hilfe der Prädikatenlogik

1. Die Variable b soll anzeigen, ob alle Werte in dem Feld f auch im Feld g vorkommen.
2. Die Variable b soll anzeigen, ob in dem Feld f und im Feld g die gleichen Werte gespeichert sind.
3. Die Variable b soll anzeigen, ob im Feld f kein Wert 2 mal vorkommt.
4. Die Variable b soll anzeigen, ob im Feld f ein Wert 2 mal vorkommt.

FACHHOCHSCHULE WEDEL
Prof. Dr. Uwe Schmidt

13. April 1999

Grundlagen der Programmierung

Übungsaufgabe: Zuweisungen

Aufgabe 1:

Gegeben seinen folgende Variablen

var i : N_0
var x, y : R
var b : B
var f : array $[0..n-1]$ of R

mit $n > 0$.

Berechnen Sie zu den gegebenen Vor- und Nachbedingungen einen geeigneten Ausdruck E, so daß die Zuweisungen korrekt arbeiten.

1. $\{$ true $\}$ $i, y := 0, E$ $\{$ $y = \sum_{j=0}^{i-1} f[j]$ $\}$
2. $\{$ true $\}$ $i, y := 1, E$ $\{$ $y = \sum_{j=1}^{i} f[j-1]$ $\}$
3. $\{$ true $\}$ $i, y := 1, E$ $\{$ $y = \sum_{j=0}^{i-1} x^j * f[j]$ $\}$
4. $\{$ $y = \sum_{j=0}^{i-1} x^j * f[j]$ $\}$ $i, y := i+1, E$ $\{$ $y = \sum_{j=0}^{i-1} x^j * f[j]$ $\}$
5. $\{$ true $\}$ $i, b := 1, E$ $\{$ $b \Leftrightarrow \forall\, 0 < j < i \bullet f[j] \geq f[j-1]$ $\}$
6. $\{$ $b \Leftrightarrow \forall\, 0 \leq j < i-1 \bullet f[j+1] > f[j]$ $\}$ $i, b := i+1, E$
   $\{$ $b \Leftrightarrow \forall\, 0 \leq j < i-1 \bullet f[j+1] > f[j]$ $\}$

Aufgabe 2:

Berechnen Sie für die folgenden Programmstücke S und die gegebenen Nachbedingungen Q mit Hilfe der Beweisregeln für Zuweisung und Anweisungsfolge die zugehörige Vorbedingung P und vereinfachen Sie diese gegebenenfalls.

Dabei werden folgende Variablen verwendet:

var x, y, z : R;
    b : B;

$$\{\ P\ \}\ S\ \{\ Q\ \}$$

1. $\{\ P\ \}\ x, y, z := y, z, x\ \{\ x = w_1 \wedge y = w_2 \wedge z = w_3\ \}$
2. $\{\ P\ \}\ x := y;\ y := x\ \{\ x = w_1 \wedge y = w_2\ \}$
3. $\{\ P\ \}\ x := x - z; y := y + z\ \{\ x + y = c\ \}$
4. $\{\ P\ \}\ b := x > y\ \{\ b \oplus (x \leq y)\ \}$
5. $\{\ P\ \}\ b := x > y\ \{\ b \Leftrightarrow (x \geq y)\ \}$

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

13. April 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:

Berechnen Sie für die folgenden Programmstücke S und die gegebenen Nachbedingungen mit Hilfe der Beweisregeln für Zuweisung, Anweisungsfolge und Verzweigung die zugehörige Vorbedingung V und vereinfachen Sie diese gegebenenfalls.

Dabei werden folgende Variablen verwendet:

```
var x, y, z : R;
      i, j, k : Z;
      equal : B;
```

1. $\{\ P\ \}$ if i mod $2 = 1$ then $i := i + 1$ end if $\{\ i$ mod $2 = 0\ \}$
2. $\{\ V\ \}$ if $i = j$ then $equal :=$ true end if $\{\ equal \Leftrightarrow (i = j)\ \}$
3. $\{\ V\ \}$
   if $j = k + 1$
   then $i := i + 1;\ k := 0$
   else $k := k + 1$
   end if
   $\{\ m = i * j + k\ \}$

Fachhochschule Wedel
Prof. Dr. Uwe Schmidt

13. April 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

13. April 1999

Grundlagen der Programmierung

Übungsaufgabe: Schleifen

Aufgabe 1:

Gegeben seien zwei Felder

$$\mathsf{var}\ f_1 : \mathsf{array}\ [0..n_1 - 1]\ \mathsf{of}\ \mathsf{R};$$
$$\mathsf{var}\ f_2 : \mathsf{array}\ [0..n_2 - 1]\ \mathsf{of}\ \mathsf{R};$$

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 2:

Gegeben seien die folgenden Variablen:

$$\mathsf{var}\ f : \mathsf{array}\ [0..n - 1]\ \mathsf{of}\ \mathsf{Z};$$
$$\mathsf{var}\ b : \mathsf{B};$$
$$\mathsf{var}\ i : \mathsf{N}_0$$

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.

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:

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

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

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

var $i, n, x_0, x_1 : \mathsf{N}_0$;
$i, x_0, x_1 := 0, 0, 1$;
$\{ I \}$
while B do
$i, x_0, x_1 := i+1, E_0, E_1$
end while
$\{ x_0 = fib(n) \}$

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.

Fachhochschule Wedel
Prof. Dr. Uwe Schmidt

13. April 1999

Grundlagen der Programmierung

Übungsaufgabe: Rekursion und Schleifen

Aufgabe 1:

Die logischen Operatoren $\wedge$, $\vee$ und $\Leftrightarrow$ können durch gleichwertige bedingte Ausdrücke formuliert werden:

$$
\begin{aligned}
a \wedge b &\equiv \mathsf{if}\ a\ \mathsf{then}\ b\ \mathsf{else}\ \mathsf{false} \\
a \vee b &\equiv \mathsf{if}\ a\ \mathsf{then}\ \mathsf{true}\ \mathsf{else}\ b \\
a \Leftrightarrow b &\equiv \mathsf{if}\ a\ \mathsf{then}\ b\ \mathsf{else}\ \neg b
\end{aligned}
$$

Verwenden Sie diese Regeln, um in der folgenden Funktion die bedingten Ausdrücke durch logische Operatoren zu ersetzen:

$$
\begin{array}{l}
g(n : \mathsf{N}_0) : \mathsf{B} \\
\quad \mathsf{if}\ n = 0 \\
\quad \mathsf{then}\ \mathsf{true} \\
\quad \mathsf{else} \\
\qquad \mathsf{if}\ n\ \mathsf{mod}\ 2 = 0 \\
\qquad \mathsf{then}\ g(n\ \mathsf{div}\ 2) \\
\qquad \mathsf{else}\ \neg g(n\ \mathsf{div}\ 2)
\end{array}
$$

Wann ist in einer Programmiersprache diese Transformation erlaubt?

Aufgabe 2:

Gegeben sei das folgende Programmstück:

$$
\begin{array}{l}
\mathsf{var}\ z, q : \mathsf{N}_0 \\
\\
f(n : \mathsf{N}_0; m : \mathsf{N}_0) : \mathsf{N}_0 \\
\quad z := z + 1; \\
\quad \mathsf{if}\ n < m \\
\quad \mathsf{then}\ n \\
\quad \mathsf{else}\ f(n\ \mathsf{div}\ m + n\ \mathsf{mod}\ m, m)
\end{array}
$$

Geben Sie die Werte an, die in z und q gespeichert sind, nachdem die folgenden Anweisungen ausgeführt worden sind

1. $z := 0; q := f(777, 10)$
2. $z := 0; q := f(64, 8)$
3. $z := 0; q := f(1024, 2)$
4. $z := 0; q := f(100, 1)$

Aufgabe 3:

Transformieren Sie die folgende Funktion in eine gleichwertige Funktion, die mit einer Schleife arbeitet. Benutzen Sie hierzu die Techniken aus der Vorlesung.

$f(n : \mathsf{N}_0, x_0 : \mathsf{N}_0, x_1 : \mathsf{N}_0) : \mathsf{N}_0$
  if $n = 0$
  then x_0
  else $f(n - 1, x_1, x_0 + x_1)$

Aufgabe 4:

Transformieren Sie die folgende Funktion in eine gleichwertige Funktion, die mit einer Schleife arbeitet. Benutzen Sie hierzu die Techniken aus der Vorlesung.

$f(x : \mathsf{N}_0, y : \mathsf{N}_0) : \mathsf{N}_0$
  if $y = x$
  then x
  else
  if $y > x$
  then $f(x + 1, y - 2)$
  else $f(x - 1, y + 1)$

FACHHOCHSCHULE WEDEL
Prof. Dr. Uwe Schmidt

13. April 1999

Grundlagen der Programmierung

Übungsaufgabe: Rekursion und Schleifen

Aufgabe 1:

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 2:

Transformieren Sie die folgende Funktion in eine gleichwertige Funktion, die mit einer Schleife arbeitet. Benutzen Sie hierzu Techniken aus der Vorlesung.

$qs(n : \mathsf{N}_0, b : \mathsf{N}_0) : \mathsf{N}_0$
  if $n = 0$
  then 0
  else $(n \text{ mod } b) + qs(n \text{ div } b, b)$

Aufgabe 3:

Gegeben sei die folgende Testfunktion zum Überprüfen, ob eine natürliche Zahl n in Binärdarstellung eine gerade Anzahl von Bits besitzt.

$evenParity(n : \mathsf{N}_0) : \mathsf{B}$
  if $n = 0$
  then
    true
  else
    $(n \text{ mod } 2 = 0) \Leftrightarrow evenParity(n \text{ div } 2)$

Transformieren Sie diese Funktion in eine gleichwertige nicht rekursive Funktion, die nur noch mit einer Schleife arbeitet. Benutzen Sie hierzu Techniken aus der Vorlesung.

Aufgabe 4:

Gegeben seien die beiden Funktionen $primFaktorSumme$ und pfs.

$pfs(n : \mathsf{N}_0, t : \mathsf{N}_0) : \mathsf{N}_0$
  if $n < t$
  then 0
  else
  if n mod $t = 0$
  then $t + pfs(n \text{ div } t, t)$
  else $pfs(n, t + 1)$

$primFaktorSumme(n : \mathsf{N}_0) : \mathsf{N}_0$
  $pfs(n, 2)$

Welche Resultate liefern die Aufrufe von

1. $primFaktorSumme(2)$
2. $primFaktorSumme(8)$
3. $primFaktorSumme(10)$

Transformieren Sie die Funktion pfs in eine gleichwertige Funktion, die mit einer Schleife arbeitet. Benutzen Sie hierzu die Techniken aus der Vorlesung.

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

13. April 1999

Grundlagen der Programmierung

Übungsaufgabe: Rekursion

Aufgabe 1:

Gegeben sei das folgende Feld

$$\mathsf{var}\ f : \mathsf{array}\ [0..n-1]\ \mathsf{of}\ \mathsf{N}_1;$$

und ein Algorithmus

$$\begin{array}{l} maxi(i : \mathsf{N}_0; j : \mathsf{N}_0) : \mathsf{N}_0 \\ \quad \mathsf{if}\ i+1 = j \\ \quad \mathsf{then}\ f[i] \\ \quad \mathsf{else}\ \mathsf{max}\ (maxi(i, j\ \mathsf{div}\ 2), maxi(j\ \mathsf{div}\ 2+1, j)) \end{array}$$

Dieser Algorithmus soll das Maximum der Werte in dem Teilstück zwischen i einschließlich und j ausschließlich berechnen. (max ist dabei eine vordefinierte 2-stellige Maximumsfunktion). Dieser Algorithmus isf für Parallelrechner geeignet.

Wieviele Funktionsaufrufe von max werden bei einem Aufruf von $maxi(0, n)$ ausgeführt?

Wieviele Aufrufe von $maxi$ sind dann maximal gleichzeitig aktiv, d.h. wie groß ist die maximale Rekursionstiefe von $maxi$

Läßt sich dieser Algorithmus einfach in eine Schleife transformieren?

Aufgabe 2:

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 3:

Das Potenzieren i^j von Zahlen $i, j \in \mathsf{N}_0$ 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:

Gegeben sei folgendes Programmstück:

```
var r, z : N0;

a(x : N0; y : N0) : N0
        z := z + 1;
        if x = 0
        then
                y + 1
        else
        if y = 0
        then
                a(x − 1, 1)
        else
                a(x − 1, a(x, y − 1))
```

Welche Werte enthalten die Variablen r und z nach der Ausführung der folgenden Anweisungen:

1. $z := 0; r := a(0, 1)$
2. $z := 0; r := a(1, 0)$
3. $z := 0; r := a(1, 1)$
4. $z := 0; r := a(1, 2)$

Fachhochschule Wedel
Prof. Dr. Uwe Schmidt

13. April 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.

Aufgabe 5:

Gegeben sei die folgende kontextfreie Grammatik

$G = (N, T, P, S)$

mit

$T = \{i, +, -, *\}$
$N = \{S\}$
P :

$S ::= i$
$S ::= S + S$
$S ::= S * S$
$S ::= -S$

Geben Sie die Menge der Wörter an, die in der von G definierten Sprache $L(G)$ enthalten sind bis einschließlich der Wortlänge 5.

Konstruieren Sie für die Zeichenreihe

$-i * i + i$

einen Ableitungsbaum.

Konstruieren Sie für die gleiche Zeichenreihe einen 2. strukturell nicht identischen Ableitungsbaum.

Konstruieren Sie für die gleiche Zeichenreihe einen 3. strukturell nicht zum 1. und zum 2. identischen Ableitungsbaum.

Warum sind mehrdeutige kontextfreie Grammatiken ungeeignet für die Definition von Programmiersprachen?

Warum sind rechtslineare kontextfreie Grammatiken ungeeignet für die Definition von Programmiersprachen?
