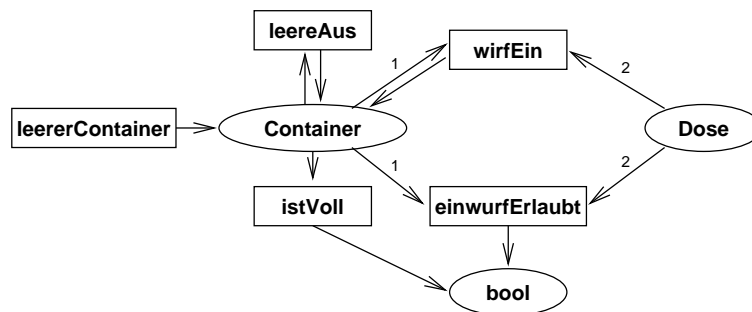


Übungen zu Einführung in die Informatik I

Aufgabe 1 Signaturen und Terme (Lösungsvorschlag)

a) Als Signaturgraph für die angegebene Signatur ergibt sich:



b) Syntaktische Korrektheit der Terme:

- (i) `wirfEin(leererContainer(), istVoll(leererContainer()))`
ist **nicht syntaktisch korrekt**, da der zweite Parameter von `wirfEin` nicht vom richtigen Typ ist: Er ist vom Typ `bool` anstatt vom Typ `Dose`.
- (ii) `einwurfErlaubt(wirfEin(leererContainer(), d1), d2, einwurfErlaubt(leererContainer(), d2))`
ist **nicht syntaktisch korrekt**, da die Anzahl der Parameter nicht stimmt. Statt zweier Parameter für `einwurfErlaubt` besitzt der Term drei Parameter.
- (iii) `istVoll(leereAus(wirfEin(leererContainer(), d1)))`
ist **syntaktisch korrekt**.
- (iv) `leereAus(wirfEin(wirfEin(d1, leererContainer()), d2))`
ist **nicht syntaktisch korrekt**, da die Parameter der inneren `wirfEin`-Operation vertauscht sind. Würde man sie umkehren, so wären sie vom richtigen Typ.

Aufgabe 2 Boolesche Funktionen (Lösungsvorschlag)

Gruppe A: Bei Gruppe A war folgende boolesche Aussage auf ihre Allgemeingültigkeit zu überprüfen:

$$\neg a \Rightarrow ((c \vee \neg d) \Rightarrow \neg(b \wedge a)).$$

Es empfiehlt sich, dieses Problem entweder durch Umformen oder durch die aus der Übung bekannte Widerspruchsmethode zu lösen; eine Lösung mit Hilfe einer Wertetabelle wäre natürlich auch denkbar, aber sehr umständlich und langwierig.

Überprüfen durch Umformen: Nach Einsetzen der Definition der Implikation ($x \Rightarrow y$ entspricht $\neg x \vee y$) in die angegebene boolesche Aussage ergibt sich:

$$\neg \neg a \vee (\neg(c \vee \neg d) \vee \neg(b \wedge a)).$$

Löst man die doppelte Negation auf und wendet die De Morganschen Gesetze an, so folgt:

$$a \vee (\neg c \wedge d) \vee \neg b \vee \neg a.$$

Nach Umstellen erhalten wir:

$$a \vee \neg a \vee (\neg c \wedge d) \vee \neg b.$$

Da $a \vee \neg a = \text{true}$ ist, folgt:

$$\text{true} \vee (\neg c \wedge d) \vee \neg b = \text{true}.$$

Der Ausdruck ist somit allgemeingültig.

Widerspruchsmethode: Wir nehmen an, es handle sich um keine allgemeingültige Aussage, d. h. die gegebene Aussage sei false. Dann muss gelten:

$$\begin{aligned} \neg a &= \text{true} \\ (c \vee \neg d) \Rightarrow \neg(b \wedge a) &= \text{false} \end{aligned}$$

Dies ist jedoch äquivalent mit:

$$\begin{aligned} a &= \text{false} \\ (c \vee \neg d) &= \text{true} \\ \neg b \vee \neg a &= \text{false} \end{aligned}$$

Da jedoch $a = \text{false}$ ist, folgt $\neg b \vee \neg a = \neg b \vee \text{true} = \text{true}$. Die gegebene Aussage kann somit nicht den Wert false annehmen und ist eine Tautologie.

Gruppe B: Die Gruppe B hatte folgende boolesche Aussage auf Allgemeingültigkeit zu überprüfen:

$$\neg f \Rightarrow ((e \vee \neg h) \Rightarrow \neg(g \wedge f)).$$

Diese Aussage geht durch die Substitution

$$\begin{aligned} f &\rightarrow a \\ g &\rightarrow b \\ e &\rightarrow c \\ h &\rightarrow d \end{aligned}$$

in die aus Gruppe A bekannte Aussage über. Sie ist also wie diese eine Tautologie; der Beweis erfolgt völlig analog.

Aufgabe 3 Markov-Algorithmus (Lösungsvorschlag)

Variante 1:

Zeichenvorrat $V = \{ a, b \}$

Schiffchen $S = \{ GG, UG, GU, UU \}$

Regelmenge $T = \{$

GGa	→	UG
GGb	→	GU
GG	→.	true
UGa	→	GG
UGb	→	UU
UG	→	UG
GUa	→	UU
GUb	→	GG
GU	→	GU
UUa	→	GU
UUb	→	UG
UU	→	UU
ϵ	→	GG

$\}$

Lösungsidee: 4 Shuttle-Zeichen repräsentieren die Kombinationen gerader und ungerader bereits verarbeiteter a und b. Je zwei Produktionen zum Durchlaufen der Zeichenkette und eine zum Terminieren bzw. zur Erzeugung einer Endlosschleife.

Variante 2:

Zeichenvorrat $V = \{ a, b \}$

Schiffchen $S = \emptyset$

Regelmenge $T = \{$

ba	→	ab
aa	→	ϵ
bb	→	ϵ
a	→	a
b	→	b
ϵ	→.	true

$\}$

Lösungsidee: Zuerst Vertauschen, so dass alle a vor allen b sind. Dann paarweises Streichen, bis entweder nichts mehr übrig (terminieren mit true) oder noch a und/oder b übrig, was zur Endlosschleife führt.

Gruppe B: a und b wird durch c und d ersetzt.

Aufgabe 4 Rekursive Programmierung (Lösungsvorschlag)

Eine rekursive Methode, die das Maximum ans Ende schiebt, lautet:

```
public IntSequenz verschiebeMax (IntSequenz a){
    return (isEmpty(a) || isEmpty(rest(a))) ? a
        : first(a) > first(rest(a))
            ? lappend(verschiebeMax(lappend(rest(rest(a)), first(a))), first(rest(a)))
            : lappend(verschiebeMax(rest(a)), first(a));
}
```

Aufgabe 5 Sortieren von Sequenzen von Zahlen (Lösungsvorschlag)

- a) In der Funktion `sort` wird bei jedem rekursiven Aufruf das letzte Element der Liste `a` entfernt und in der so verkürzten und unsortierten Liste `a` das Maximum ans Listenende geschoben. Dieses letzte Element wird am Anfang der Ergebnisliste eingefügt.

Unter Vernachlässigung der Details der Funktion `verschiebeMax`, die in Aufgabe 4 zu formulieren war, wird somit bei jedem rekursiven Aufruf von `sort` das Maximum aus der unsortierten Liste entnommen und am Beginn der Ergebnisliste eingefügt. Ohne Berücksichtigung des Algorithmus aus Aufgabe 4, ergäbe sich das Sortierprinzip: Sortieren durch Auswählen des Maximums.

Berücksichtigt man, dass in der Funktion `verschiebeMax` das Maximum durch Vertauschen benachbarter Listenelemente ans Ende geschoben wird, so erkennt man, dass es sich um eine Implementierung handelt, die das Sortierprinzip Bubblesort umsetzt.

Da eine richtige Implementierung des Algorithmus aus Aufgabe 4 in dieser Aufgabe nicht vorausgesetzt werden kann, wird sowohl für Bubblesort als auch für Selectionsort bzw. Maximum-Einfügen die volle Punktzahl vergeben.

- b) Wir wählen die Abstiegsfunktion:

$$h(a, b) = \text{laenge}(a).$$

Hat die zu sortierende Liste die Länge Null, so erfolgt kein rekursiver Aufruf und die Funktion terminiert.

Da im rekursiven Aufruf die Funktion `sort` mit den Argumenten $a' = \text{verschiebeMax}(\text{lrest}(a))$ und $b' = \text{lappend}(b, \text{last}(a))$ aufgerufen wird, betrachten wir für $\text{laenge}(a) > 0$ die Abstiegsfunktion h mit den Argumenten a' und b' :

$$\begin{aligned} h(\text{verschiebeMax}(\text{lrest}(a)), \text{lappend}(b, \text{last}(a))) \\ = \text{laenge}(\text{verschiebeMax}(\text{lrest}(a))) \end{aligned}$$

Die Funktion `verschiebeMax` verändert nicht die Länge der Liste; deshalb gilt:

$$\begin{aligned} \text{laenge}(\text{verschiebeMax}(\text{lrest}(a))) &= \text{laenge}(\text{lrest}(a)) \\ &= \text{laenge}(a) - 1 \\ &= h(a, b) - 1. \end{aligned}$$

Die Abstiegsfunktion nimmt somit mit jedem rekursiven Aufruf um 1 ab und wird nach einer *endlichen* Anzahl von Schritten Null; aufgrund der Fundiertheit von \mathbb{N} terminiert damit die Funktion `sort`.

- c) Eine iterative Implementierung der Funktion `sortiere` lautet:

```
IntSequenz sortIterativ (IntSequenz a){
  IntSequenz a2 = verschiebeMax(a);
  IntSequenz b = create();
  while (!isEmpty(a2)) {
    b = lappend(b, last(a2));
    a2 = verschiebeMax(lrest(a2));
  }
  return b;
}
```

- d) Es bezeichne n die Länge der zu sortierenden Liste. Die Ausführung der Funktion `sortiere` hat eine n -malige Ausführung der Funktion `verschiebeMax`, Komplexität $O(n)$, zur Folge. Die Funktionen `isEmpty`, `lrest`, `lappend` und `last` haben die Komplexität $O(1)$ und

somit keinen Einfluß auf die Komplexität von `sortiere`. Der n-malige Aufruf einer Funktion der Komplexität $O(n)$ führt somit insgesamt zu der Komplexität $O(n^2)$ für die Funktion `sortiere`.

Aufgabe 6 **Reihungen (Lösungsvorschlag)**

Die Darstellung von Graphen durch eine boolesche Matrix der angegebenen Art nennt man **Adjazenzmatrix**. Die Bestimmung von Wegen einer Länge größer eins erfolgt durch Und-Verknüpfung von Matrixeinträgen. Die Matrix, in der festgehalten ist, ob ein Knoten von einem anderen aus durch einen Weg beliebiger Länge erreichbar ist, heißt transitive Hülle. Die in der Aufgabe geforderte Errechnung der Weglänge zwei ist eine Vorüberlegung zur Berechnung der transitiven Hülle:

```
boolean[][] weglaenge2 (boolean[][] g) { //Vorbedingung: g quadratisch!

    boolean[][] ergebnis;
    int knotenzahl = g.length;
    ergebnis = new boolean[knotenzahl][knotenzahl];

    for (int i = 0; i < knotenzahl; i++) {
        for (int j = 0; j < knotenzahl; j++) {
            ergebnis[i][j] = false; //Vorbesetzung mit false
            for (int k = 0; k < knotenzahl; k++) {
                if (g[i][k] && g[k][j]) ergebnis[i][j] = true; //Weg von i nach j ex.
            } // Ende k-Schleife
        } // Ende j-Schleife
    } // Ende i-Schleife

    return ergebnis;
}
```

Aufgabe 7 **Binärbäume (Lösungsvorschlag)**

Der entscheidende Unterschied zur Übungsaufgabe 46 besteht darin, dass im allgemeinen Binärbaum keine Sortierungseigenschaft gewährleistet ist. Damit ist immer sowohl der linke und rechte Unterbaum zu überprüfen und mit dem Wert im betrachteten Knoten (lokale Wurzel) zu vergleichen:

```
int gibMaximum () { //Vorbedingung: Baum ist nicht leer

    int maximum = inhalt; //Vorbesetzung
    if (! linkesKind.leer) { // linker Unterbaum nicht leer
        int linkesMaximum = linkesKind.gibMaximum();
        if (linkesMaximum > maximum) maximum = linkesMaximum;
    } //Ende if
    if (! rechtesKind.leer) { //rechter Unterbaum nicht leer
        int rechtesMaximum = rechtesKind.gibMaximum();
        if (rechtesMaximum > maximum) maximum = rechtesMaximum;
    } // Ende if

    return maximum;
}
```

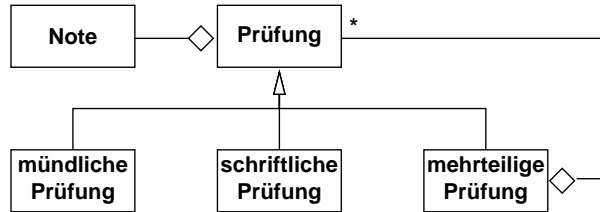
Eine andere Variante könnte z.B. alle 4 Fälle (Knoten ist Blatt, hat nur linkes Kind, hat nur rechtes Kind oder hat zwei Kinder) abfragen.

Gruppe B: analog dazu für das Minimum des Baums.

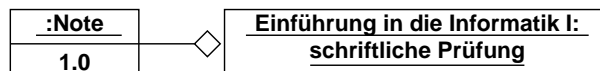
Aufgabe 8 Modellierung (Lösungsvorschlag)

Die Aufgabenstellung lässt sich mit Hilfe des Kompositions-Musters (composite pattern) ausdrücken:

a) Klassendiagramm:



b) Instanzdiagramm:



Variante(n): Modellierung der Note als ganze Zahl `int` und/oder Modellierung der Veranstaltungstitels (z.B. als `String`), statt die Instanz selbst so zu nennen.

Aufgabe 9 Referenzgeflecht (Lösungsvorschlag)

Das Referenzgeflecht, das bei der Ausführung des angegebenen Programmausschnittes entsteht, hat folgende Gestalt:

