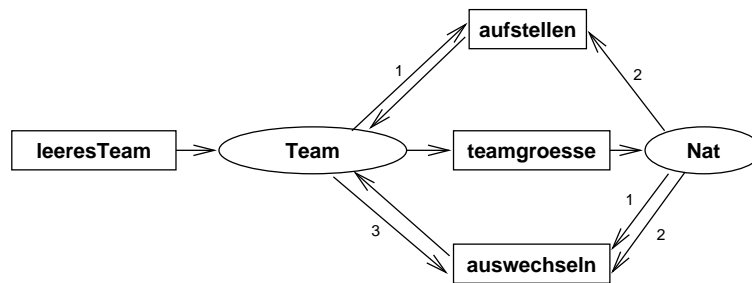


Übungen zu Einführung in die Informatik I

Aufgabe 1 Signaturen und Terme (Lösungsvorschlag)

a) Als Signaturgraph für die angegebene Signatur ergibt sich:



b) Syntaktische Korrektheit der Terme:

- (i) `aufstellen(auswechseln(1,2,leeresTeam()), leeresTeam())`
 ist **nicht syntaktisch korrekt**, da der zweite Parameter von `aufstellen` nicht vom richtigen Typ ist: Er ist vom Typ `Team` anstatt vom Typ `N`.
- (ii) `auswechseln(teamgroesse(leeresTeam()), 1, aufstellen(leeresTeam(),2))`
 ist **syntaktisch korrekt**.
- (iii) `teamgroesse(auswechseln(teamgroesse(leeresTeam()), leeresTeam(), 2))`
 ist **nicht syntaktisch korrekt**, da die Parameter der `auswechseln`-Operation vertauscht sind. Würde man sie umkehren, so wären sie vom richtigen Typ.

c) Ableitung für den Term `auswechseln(1,3,aufstellen(aufstellen(leeresTeam()),1),2))`

Variante 1:

$$\begin{aligned}
 &\text{auswechseln}(1,3,\text{aufstellen}(\text{aufstellen}(\text{leeresTeam}(),1),2)) \stackrel{G2}{=} \\
 &\text{aufstellen}(\text{auswechseln}(1,3,\text{aufstellen}(\text{leeresTeam}(),1)),2) \stackrel{G1}{=} \\
 &\text{aufstellen}(\text{aufstellen}(\text{leeresTeam}(),3),2)
 \end{aligned}$$

Variante 2:

$$\begin{aligned}
 &\text{auswechseln}(1,3,\text{aufstellen}(\text{aufstellen}(\text{leeresTeam}(),1),2)) \stackrel{G3}{=} \\
 &\text{auswechseln}(1,3,\text{aufstellen}(\text{aufstellen}(\text{leeresTeam}(),2),1)) \stackrel{G1}{=} \\
 &\text{aufstellen}(\text{aufstellen}(\text{leeresTeam}(),2),3)
 \end{aligned}$$

Gruppe B: identisch

Aufgabe 2 Markov-Algorithmus (Lösungsvorschlag)

Variante 1:

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

Schiffchen $S = \emptyset$

Regelmenge $T = \{$

$ab \rightarrow \epsilon$
 $ba \rightarrow ab$
 $ac \rightarrow \epsilon$
 $ca \rightarrow ac$
 $a \rightarrow a$
 $b \rightarrow b$
 $c \rightarrow c$
 $\epsilon \rightarrow \text{true}$

$\}$

Lösungsidee: Es werden jeweils Paare von 'a' und 'b' bzw. 'a' und 'c' gestrichen. Verbleiben am Ende keine Zeichen mehr, so hatte das Anfangswort gleichviele 'a' wie 'b' und 'c' zusammen, und es wird mit true terminiert. Bleiben noch Zeichen übrig, so wird für eine Endlosschleife gesorgt, indem eine Regel immer wieder anwendbar ist (linke Seite gleich rechter Seite).

Die beiden Regeln zum Sortieren des Wortes (Regeln 2 und 4) können auch ersetzt werden durch Regeln zum Streichen analog zu Regeln 1 und 3 ($ba \rightarrow \epsilon$ und $ca \rightarrow \epsilon$).

Variante 2:

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

Schiffchen $S = \emptyset$

Regelmenge $T = \{$

$c \rightarrow b$
 $ab \rightarrow \epsilon$
 $ba \rightarrow \epsilon$
 $a \rightarrow a$
 $b \rightarrow b$
 $\epsilon \rightarrow \text{true}$

$\}$

Lösungsidee: Da der Unterschied zwischen 'b' und 'c' für den Algorithmus unwichtig ist, kann der Algorithmus durch anfängliches Umwandeln von 'c' in 'b' vereinfacht werden und es werden damit weniger Regeln benötigt. Wie in obiger Variante werden nun Paare von 'a' und 'b' gestrichen, falls die Anzahl gleich war mit true terminiert und ansonsten eine Endlosschleife erzeugt.

Gruppe B: Gleiche Überlegungen, nur 'a' wird durch 'c' ersetzt.

Aufgabe 3 Rekursive Programmierung (Lösungsvorschlag)

Idee: Schrittweise Abarbeitung der Sequenz durch Zerlegung mit first und rest bzw. last und lrest. Ist die Sequenz leer, so ist der Ergebniswert 0, ansonsten wird ein Element betrachtet und das Ergebnis um 1 erhöht, wenn es dem Suchelement entspricht, ansonsten wird lediglich verkürzt:

```
int anzahl (IntSequenz s, int el) {  
  
    return (isEmpty(s)) ? 0 :
```

```

        (first(s) == el) ? anzahl(rest(s), el) + 1 :
                          anzahl(rest(s), el) ;
    }

```

Lösungsvariante: Verwendung von last und lrest statt first und rest.

Gruppe B: identisch

Aufgabe 4 Bearbeitung von Zeichensequenzen (Lösungsvorschlag)

Wir verwenden das aus den Übungen bekannte Verfahren zur Umsetzung von repetitiver Rekursion in Iteration. Die beiden Parameter der Funktion uebertrageZahlen werden durch zwei Hilfsvariablen ersetzt:

```

IntSequenz entferneMehrfacheIterativ (IntSequenz s){

    IntSequenz a = create();
    IntSequenz b = s;

    while (! isEmpty(b)) {
        if (anzahl(a, first(b)) > 0) {b = rest(b);}
        else {a = append(a, first(b)); b = rest(b);}
    }

    return a;
}

```

Der Rumpf der Schleife liesse sich durch Negation der Bedingung noch ein wenig optimieren, die konsequente Umsetzung der Repetition wie oben ist allerdings ebenso richtig.

Gruppe B: append durch lappend, first durch last und rest durch lrest ersetzen.

Aufgabe 5 (Gruppe A) Strukturelle Induktion (Lösungsvorschlag)

Induktionsanfang: Für die leere Sequenz, d.h. für $s = \text{create}()$, liefert der Funktionsaufruf $\text{summe}(s) = 0$, da $\text{isEmpty}(\text{create}()) = \text{true}$. Andererseits folgt $f_{\text{summe}}(s) = 0$, da $\text{laenge}(s) = 0$ und $\sum_{i=1}^0 s_i = 0$. Für $s = \text{create}()$ ergibt sich somit das gewünschte Resultat.

Induktionsschluss: Induktionsannahme ist, die Operation $\text{int summe}(\text{IntSequenz } s)$ sei für eine bestimmte Sequenz s partiell korrekt; d.h. es gilt: $\text{summe}(s) = f_{\text{summe}}(s)$.

Sei nun $s' = \text{lappend}(s, \text{el})$, d.h. s' bezeichne alle aus s in einem Schritt erzeugbaren Sequenzen. Zu zeigen ist dann, dass aus der Induktionsannahme die Identität $\text{summe}(s') = f_{\text{summe}}(s')$ folgt.

Für s' ist $\text{isEmpty}(s') = \text{false}$; gemäß der Implementierung ergibt sich wegen $\text{rest}(\text{lappend}(s, \text{el})) = s$ und $\text{first}(\text{lappend}(s, \text{el})) = \text{el}$:

$$\begin{aligned}
 \text{summe}(s') &= \text{summe}(\text{rest}(\text{lappend}(s, \text{el}))) + \text{first}(\text{lappend}(s, \text{el})) \\
 &= \text{summe}(s) + \text{el} \\
 \stackrel{\text{I.A.}}{=} f_{\text{summe}}(s) + \text{el} &= \sum_{i=1}^{\text{laenge}(s)} s_i + \text{el}.
 \end{aligned}$$

Wir müssen nun zeigen, dass $f_{\text{summe}}(s')$ denselben Wert hat. Dazu betrachten wir die Sequenzen s , s' genauer: Wir bezeichnen das eingefügte Element $e1$ mit s'_1 , da es durch das Voranhängen mit `lappend` das erste Element von s' ist. Zwischen den Elementen von s und s' besteht dann die Beziehung: $s_i = s'_{i+1}$ mit $1 \leq i \leq \text{laenge}(s)$, da alle Elemente um eine Position nach hinten geschoben wurden. Dann folgt:

$$\text{summe}(s') = \sum_{i=1}^{\text{laenge}(s)} s_i + e1 = s'_1 + \sum_{i=1}^{\text{laenge}(s)} s_i = s'_1 + \sum_{i=1}^{\text{laenge}(s)} s'_{i+1} = \sum_{i=1}^{\text{laenge}(s)+1} s'_i.$$

Da $\text{laenge}(s') = \text{laenge}(s) + 1$, folgt

$$\text{summe}(s') = \sum_{i=1}^{\text{laenge}(s')} s'_i = f_{\text{summe}}(s').$$

Die Operation `int summe(IntSequenz s)` ist somit partiell korrekt.

Aufgabe 5 (Gruppe B) Strukturelle Induktion (Lösungsvorschlag)

Induktionsanfang: Für die leere Sequenz, d.h. für $s = \text{create}()$, liefert der Funktionsaufruf $\text{summe}(s) = 0$, da $\text{isEmpty}(\text{create}()) = \text{true}$. Andererseits folgt $f_{\text{summe}}(s) = 0$, da $\text{laenge}(s) = 0$ und $\sum_{i=1}^0 s_i = 0$. Für $s = \text{create}()$ ergibt sich somit das gewünschte Resultat.

Induktionsschluß: Induktionsannahme ist, die Operation `int summe(IntSequenz s)` sei für eine bestimmte Sequenz s partiell korrekt, d.h. es gilt: $\text{summe}(s) = f_{\text{summe}}(s)$.

Sei nun $s' = \text{append}(s, e1)$, d.h. s' bezeichne alle aus s in einem Schritt erzeugbaren Sequenzen. Zu zeigen ist dann, dass aus der Induktionsannahme die Identität $\text{summe}(s') = f_{\text{summe}}(s')$ folgt.

Für s' ist $\text{isEmpty}(s') = \text{false}$; gemäß der Implementierung ergibt sich wegen $\text{lrest}(\text{append}(s, e1)) = s$ und $\text{last}(\text{append}(s, e1)) = e1$:

$$\begin{aligned} \text{summe}(s') &= \text{summe}(\text{lrest}(\text{append}(s, e1))) + \text{last}(\text{append}(s, e1)) \\ &= \text{summe}(s) + e1 \\ &\stackrel{I.A.}{=} f_{\text{summe}}(s) + e1 = \sum_{i=1}^{\text{laenge}(s)} s_i + e1. \end{aligned}$$

Wir bezeichnen nun das eingefügte Element $e1$ mit $s'_{\text{laenge}(s)+1}$, da es das neue letzte Element ist. Zwischen den anderen Elementen von s' und s besteht die Beziehung: $s_i = s'_i$ mit $1 \leq i \leq \text{laenge}(s)$, da die vorderen Elemente in der Reihenfolge und Position übereinstimmen. Dann folgt:

$$\text{summe}(s') = \sum_{i=1}^{\text{laenge}(s)} s_i + s'_{\text{laenge}(s)+1} = \sum_{i=1}^{\text{laenge}(s)+1} s'_i.$$

Da $\text{laenge}(s') = \text{laenge}(s) + 1$, folgt

$$\text{summe}(s') = \sum_{i=1}^{\text{laenge}(s')} s'_i = f_{\text{summe}}(s').$$

Die Operation `int summe(IntSequenz s)` ist somit partiell korrekt.

Aufgabe 6 (Gruppe B) Terminierung (Lösungsvorschlag)

Als Abstiegsfunktion wählen wir die Funktion:

$$h(a,b) := \begin{cases} 0; & (\text{isEmpty}(a) \vee \text{isEmpty}(b)) = \text{true} \\ \text{laenge}(a) + \text{laenge}(b); & \text{sonst} \end{cases}$$

Erfolgt kein rekursiver Aufruf, d.h. ist die Bedingung $(\text{isEmpty}(a) \vee \text{isEmpty}(b))$ erfüllt, so ist $h(a,b) = 0$ und die Funktion $\text{schnittmenge}(a, b)$ terminiert.

Im Falle eines rekursiven Aufrufs ist die Bedingung $(\text{isEmpty}(a) \vee \text{isEmpty}(b))$ nicht erfüllt, die Abstiegsfunktion ist verschieden von Null und es sind drei Fälle zu betrachten:

Fall I: $\text{first}(a) == \text{first}(b)$ Der rekursive Aufruf erfolgt hier in der Form $\text{schnittmenge}(\text{rest}(a), \text{rest}(b))$. Wir betrachten

$$h(\text{rest}(a), \text{rest}(b)) = \begin{cases} 0; & (\text{isEmpty}(\text{rest}(a)) \vee \text{isEmpty}(\text{rest}(b))) = \text{true}; \\ \text{laenge}(a) + \text{laenge}(b) - 2; & \text{sonst} \end{cases}$$

$$h(\text{rest}(a), \text{rest}(b)) < h(a,b)$$

Fall II: $\text{first}(a) < \text{first}(b)$ Der rekursive Aufruf erfolgt hier in der Form $\text{schnittmenge}(\text{rest}(a), b)$. Wir betrachten

$$h(\text{rest}(a), b) = \begin{cases} 0; & (\text{isEmpty}(\text{rest}(a)) \vee \text{isEmpty}(b)) = \text{true} \\ \text{laenge}(a) + \text{laenge}(b) - 1; & \text{sonst} \end{cases}$$
$$h(\text{rest}(a), b) < h(a,b)$$

Fall III: $\text{first}(a) > \text{first}(b)$ Der rekursive Aufruf erfolgt hier in der Form $\text{schnittmenge}(a, \text{rest}(b))$. Wir betrachten

$$h(a, \text{rest}(b)) = \begin{cases} 0; & (\text{isEmpty}(a) \vee \text{isEmpty}(\text{rest}(b))) = \text{true} \\ \text{laenge}(a) - 1 + \text{laenge}(b); & \text{sonst} \end{cases}$$
$$h(a, \text{rest}(b)) < h(a,b)$$

Im Fall eines rekursiven Aufrufes tritt also entweder sofort der Terminierungsfall ein, wenn $h(\text{rest}(a), \text{rest}(b)) = 0$, $h(a, \text{rest}(b)) = 0$ oder $h(\text{rest}(a), b) = 0$. Oder aber die Abstiegsfunktion h , die in diesem Fall nur Werte größer als 1 annehmen kann, wird um 2 bzw. 1 kleiner. Jede absteigende Folge in \mathbb{N} ist aufgrund der Fundiertheit von \mathbb{N} endlich und damit terminiert die Funktion schnittmenge nach endlich vielen Schritten.

Aufgabe 6 (Gruppe B) Terminierung (Lösungsvorschlag)

Als Abstiegsfunktion wählen wir die Funktion:

$$h(a,b) := \begin{cases} 0; & (\text{isEmpty}(a) \vee \text{isEmpty}(b)) = \text{true} \\ \text{laenge}(a) + \text{laenge}(b); & \text{sonst} \end{cases}$$

Erfolgt kein rekursiver Aufruf, d.h. ist die Bedingung $(\text{isEmpty}(a) \vee \text{isEmpty}(b))$ erfüllt, so ist $h(a,b) = 0$ und die Funktion $\text{schnittmenge}(a, b)$ terminiert.

Im Falle eines rekursiven Aufrufs ist die Bedingung $(\text{isEmpty}(a) \vee \text{isEmpty}(b))$ nicht erfüllt, die Abstiegsfunktion ist verschieden von Null und es sind drei Fälle zu betrachten:

Fall I: $\text{last}(a) == \text{last}(b)$ Der rekursive Aufruf erfolgt hier in der Form $\text{schnittmenge}(\text{lrest}(a), \text{lrest}(b))$. Wir betrachten

$$h(\text{lrest}(a), \text{lrest}(b)) = \begin{cases} 0; & (\text{isEmpty}(\text{lrest}(a)) \vee \text{isEmpty}(\text{lrest}(b))) = \text{true}; \\ \text{laenge}(a) + \text{laenge}(b) - 2; & \text{sonst} \end{cases}$$

$$h(\text{lrest}(a), \text{lrest}(b)) < h(a, b)$$

Fall II: $\text{last}(a) < \text{last}(b)$ Der rekursive Aufruf erfolgt hier in der Form $\text{schnittmenge}(a, \text{lrest}(b))$. Wir betrachten

$$h(a, \text{lrest}(b)) = \begin{cases} 0; & (\text{isEmpty}(a) \vee \text{isEmpty}(\text{lrest}(b))) = \text{true} \\ \text{laenge}(a) + \text{laenge}(b) - 1; & \text{sonst} \end{cases}$$

$$h(a, \text{lrest}(b)) < h(a, b)$$

Fall III: $\text{last}(a) > \text{last}(b)$ Der rekursive Aufruf erfolgt hier in der Form $\text{schnittmenge}(\text{lrest}(a), b)$. Wir betrachten

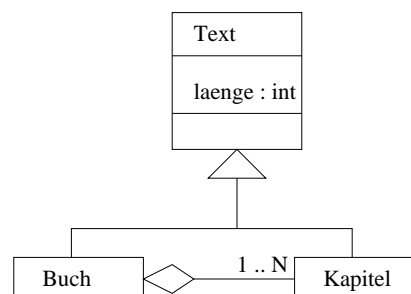
$$h(\text{lrest}(a), b) = \begin{cases} 0; & (\text{isEmpty}(\text{lrest}(a)) \vee \text{isEmpty}(b)) = \text{true} \\ \text{laenge}(a) - 1 + \text{laenge}(b); & \text{sonst} \end{cases}$$

$$h(\text{lrest}(a), b) < h(a, b)$$

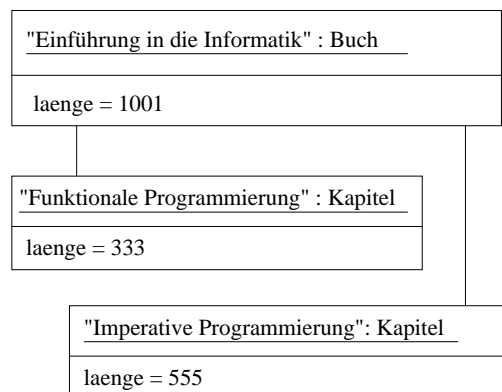
Im Fall eines rekursiven Aufrufes tritt also entweder sofort der Terminierungsfall ein, wenn $h(\text{lrest}(a), \text{lrest}(b)) = 0$, $h(a, \text{lrest}(b)) = 0$ oder $h(\text{lrest}(a), b) = 0$. Oder aber die Abstiegsfunktion h , die in diesem Fall nur Werte größer als 1 annehmen kann, wird um 2 bzw. 1 kleiner. Jede absteigende Folge in \mathbb{N} ist aufgrund der Fundiertheit von \mathbb{N} endlich und damit terminiert die Funktion schnittmenge nach endlich vielen Schritten.

Aufgabe 7 Modellierung (Lösungsvorschlag)

- a) Ein Klassendiagramm zur Beschreibung der vorgegebenen Struktur des Buches hat folgende Gestalt:

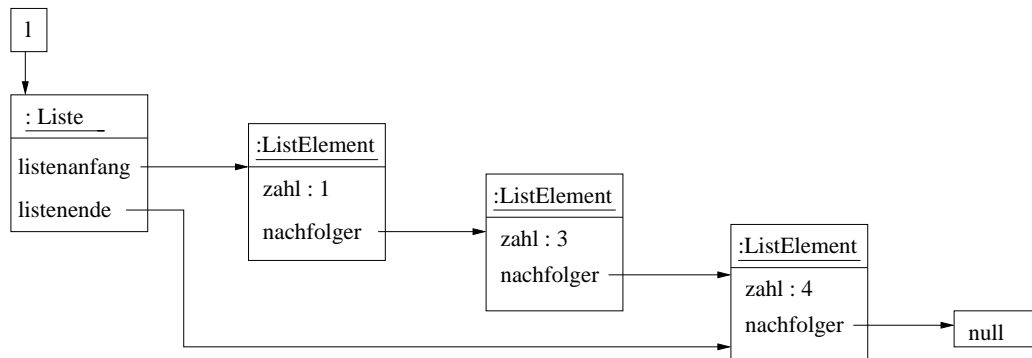


- b) Für das zugehörige Instanzdiagramm ergibt sich:



Aufgabe 8 Referenzgeflecht (Lösungsvorschlag)

- a) Bei der Ausführung des angegebenen Programmausschnitts entsteht folgendes Referenzgeflecht:



- b) Mit folgendem Programmausschnitt wird zwischen dem Element mit der Zahl 1 und dem Element mit der Zahl 3 ein Element mit der Zahl 2 eingefügt:

```
el = new ListElement(2);
el.nachfolger = l.listenanfang.nachfolger;
l.listenanfang.nachfolger = el;
```

Aufgabe 9 Suchen in Reihungen (Lösungsvorschlag)

Variante 1: iterativ mit einer while-Schleife:

```
int suchen (int[] r, int suchelement) {

    int aktuellerIndex = 1;
    while ((aktuellerIndex < r.length) && (r[aktuellerIndex] != suchelement)) {
        if (suchelement < r[aktuellerIndex]) {aktuellerIndex = 2 * aktuellerIndex;}
        else {aktuellerIndex = 2 * aktuellerIndex + 1;}
    }

    if (aktuellerIndex < r.length) {return aktuellerIndex;}
    else return 0;
}
```

Hinweis: Das Verfahren hat Ähnlichkeit mit der binären Suche in Reihungen. Schleifenbedingung und die Fortschaltung der Indexposition wurden allerdings anders gewählt.

Variante 2: rekursiv mit Einbettung:

```
int suchen (int[] r, int suchelement) {

    return suchenBett (r, suchelement, 1);
}
```

```
int suchenBett (int[] r, int s, int i) {  
  
    return (i >= r.length) ? 0 :  
           (s == r[i]) ? i :  
           (s < r[i]) ? suchenBett(r, s, 2*i) :  
                       suchenBett(r, s, 2*i + 1);  
}
```

Wesentlich ist in beiden Varianten, dass die Prüfung, ob der Index noch innerhalb der Reihung liegt, stattfindet, bevor auf ein Element $r[i]$ der Reihung zugegriffen wird.

Gruppe B: identisch