



Universität Karlsruhe

Informatik 1 WS 00/01

Institut für Telematik, Forschungsgruppe C&M
Prof. Dr. S. Abeck, R. Scholderer

Musterlösungen zum Übungsblatt 7

Lösung

1. Relationale Algebra - Entwicklung von SQL-Anfragen

Teilaufgabe a)

1.

SELECT Name, Mat-Nr, Semester FROM Studierende WHERE Semester <= 2 OR Fach = "WiWi"

2.

SELECT Name, Vorname FROM Studierende WHERE Vorname <> "Peter" AND (Fach = "Info" OR Fach = "Bio")

Teilaufgabe b)

= {Name: CHARACTER(20), Vorname: CHARACTER(10), Mat-Nr: INTEGER, Fach: CHARACTER(4), Semester: INTEGER}

Lösung

2. Term der booleschen Algebra

a) $\Sigma^{(0)}$ ist als Teilmenge der Menge alle Operationen umfassenden Menge Σ . $\Sigma^{(0)}$ enthält sämtliche nullstelligen Operatoren aus Σ .

Der gegebene Term enthält die Elemente \top und \perp aus $\Sigma^{(0)}$.

b)

1. Regel für korrekte Terme:

Nach der ersten Regel für korrekten Terme sind die nullstelligen Unterterme \top , \perp , N, P, W korrekt, da \top und \perp aus $\Sigma^{(0)} \subseteq X$ und die Konstanten N, P, W aus X sind.

2. Regel für korrekte Terme:

- einstellige Unterterme:

einzigster einstelliger Operator ist \neg .

$\neg W$: Der Operator \neg ist einstellig und wird auf den korrekten Unterterm W angewendet. $\rightarrow \neg W$ ist korrekt.

$\neg(P \vee \neg W)$: Der Operator \neg ist einstellig und wird auf den (einen) Unterterm $(P \vee \neg W)$ angewendet. $\rightarrow \neg(P \vee \neg W)$ ist korrekt, falls $(P \vee \neg W)$ korrekt ist, was bei Betrachtung der zweistelligen Unterterme gezeigt werden muss.

- zweistellige Unterterme:

zweistellige Operatoren im gegebenen Term sind \vee und \wedge .

$(N \wedge \top)$: Der Operator \wedge ist zweistellig und wird auf die korrekten Unterterme N und \top angewendet.

\rightarrow Der Term $(N \wedge \top)$ ist korrekt.

$((N \wedge \top) \vee \perp)$: Der Operator \vee ist zweistellig und wird auf die korrekten Unterterme $(N \wedge \top)$ und \perp angewendet. \rightarrow Der Term $((N \wedge \top) \vee \perp)$ ist korrekt.

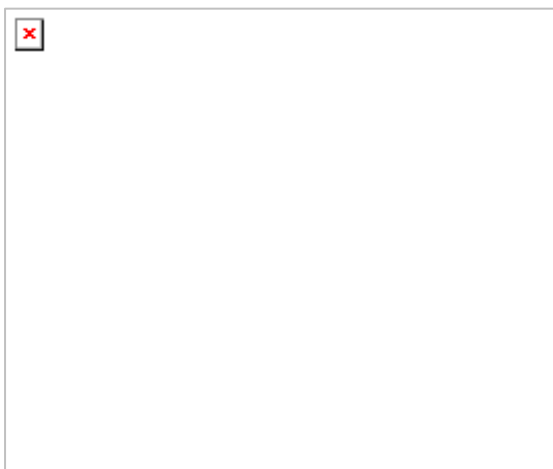
$(P \vee \neg W)$: Der zweistellige Operator \vee wird auf die korrekten Unterterme P und $\neg W$ angewendet.

$\rightarrow (P \vee \neg W)$ ist korrekt. (Damit folgt

auch die Korrektheit des einstelligen Unterterms $\neg(P \vee \neg W)$.)

$((N \wedge \top) \vee \perp) \wedge \neg(P \vee \neg W)$: Der zweistellige Operator \wedge wird hier auf die korrekten Unterterme $((N \wedge \top) \vee \perp)$ und $\neg(P \vee \neg W)$ angewendet, \rightarrow der gesamte Term ist korrekt.

c) Der Kantorovic-Baum zum gegebenen Term lässt sich wie folgt darstellen:



Lösung

3. Boolesche Algebra

a)

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

$\rightarrow (((a \wedge b) \vee (\neg a \wedge \neg b)) \wedge c) \vee (\neg(a \wedge b) \wedge \neg(\neg a \wedge \neg b) \wedge \neg c)$ (DeMorgan, Assoziativität.)

$\rightarrow (((a \wedge b) \vee (\neg a \wedge \neg b)) \wedge c) \vee ((\neg a \vee \neg b) \wedge (a \vee b) \wedge \neg c)$ (DeMorgan, Involution)

$$\rightarrow (a \wedge b \wedge c) \vee (\neg a \wedge \neg b \wedge c) \vee ((\neg a \vee \neg b) \wedge (a \vee b) \wedge \neg c) \quad (\text{Distributivität})$$

$$\rightarrow (a \wedge b \wedge c) \vee (\neg a \wedge \neg b \wedge c) \vee (((\neg a \wedge a) \vee (\neg a \wedge b) \vee (\neg b \wedge a) \vee (\neg b \wedge b)) \wedge \neg c)$$

(Distributivität, Assoziativität)

$$\rightarrow (a \wedge b \wedge c) \vee (\neg a \wedge \neg b \wedge c) \vee (\perp \vee (\neg a \wedge b) \vee (\neg b \wedge a) \vee \perp) \wedge \neg c \quad (\text{Neutrale Elemente})$$

$$\rightarrow (a \wedge b \wedge c) \vee (\neg a \wedge \neg b \wedge c) \vee (((\neg a \wedge b) \vee (a \wedge \neg b)) \wedge \neg c) \quad (\text{Idempotenz, Kommutativität})$$

$$\rightarrow (a \wedge b \wedge c) \vee (\neg a \wedge \neg b \wedge c) \vee (\neg a \wedge b \wedge \neg c) \vee (a \wedge \neg b \wedge \neg c) \quad (\text{Distributivität, Assoziativität})$$

b)

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

a	b	c	$a \wedge b$	$\neg a \wedge \neg b$	$((a \wedge b) \vee (\neg a \wedge \neg b))$	$(\neg ((a \wedge b) \vee (\neg a \wedge \neg b)) \wedge \neg c)$	$\neg ((a \wedge b) \vee (\neg a \wedge \neg b))$	$(\neg ((a \wedge b) \vee (\neg a \wedge \neg b)) \wedge \neg c)$	Erg.
0	0	0	0	1	1	0	0	0	0
0	0	1	0	1	1	1	0	0	1
0	1	0	0	0	0	0	1	1	1
0	1	1	0	0	0	0	1	0	0
1	0	0	0	0	0	0	1	1	1
1	0	1	0	0	0	0	1	0	0
1	1	0	1	0	1	0	0	0	0
1	1	1	1	0	1	1	0	0	1

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

a	b	c	$a \wedge b \wedge c$	$\neg a \wedge \neg b \wedge c$	$\neg a \wedge b \wedge \neg c$	$a \wedge \neg b \wedge \neg c$	Erg.
0	0	0	0	0	0	0	0
0	0	1	0	1	0	0	1
0	1	0	0	0	1	0	1

0	1	1	0	0	0	0	0
1	0	0	0	0	0	1	1
1	0	1	0	0	0	0	0
1	1	0	0	0	0	0	0
1	1	1	1	0	0	0	1

Lösung

4. Keller - TripelKeller

T ist die Menge, aus der alle Elemente im Keller sind.

Signatur $\Sigma_{\text{TripelKeller}}$

createStack: $\{\} \rightarrow \text{TripelKeller}(T)$
 push: $\text{TripelKeller}(T) \times T \times T \times T \rightarrow \text{TripelKeller}(T)$
 pop: $\text{TripelKeller}(T) \rightarrow \text{TripelKeller}(T)$
 top1: $\text{TripelKeller}(T) \rightarrow T$
 top2: $\text{TripelKeller}(T) \rightarrow T$
 top3: $\text{TripelKeller}(T) \rightarrow T$
 empty: $\text{TripelKeller}(T) \rightarrow B$

Axiome (mit $k \in \text{TripelKeller}(T)$ und $t_1, t_2, t_3 \in T$)

K1: $\text{empty}(\text{createStack}) = L$
 K2: $\text{empty}(\text{push}(k, t_1, t_2, t_3)) = O$
 K3: $\text{pop}(\text{push}(k, t_1, t_2, t_3)) = k$
 K4: $\text{top1}(\text{push}(k, t_1, t_2, t_3)) = t_1$
 K5: $\text{top2}(\text{push}(k, t_1, t_2, t_3)) = t_2$
 K6: $\text{top3}(\text{push}(k, t_1, t_2, t_3)) = t_3$

Der Ausdruck für die obige Graphik lautet:

$\text{top1}(\text{push}(\text{pop}(\text{push}(\text{push}(\text{createStack}, 3, 4, 1), 6, 5, 2)), 5, 6, 2)$

Lösung

5. Keller – Auswertung von Kellern

a) I: Integer, II: Bool, III: Character

b)

I: $\text{empty}(\text{pop}(\text{push}(\text{push}(\text{pop}(\text{push}(\text{CreateStack}, 2)), 99), 1))) =^{(K3)}$
 $\text{empty}(\text{pop}(\text{push}(\text{push}(\text{CreateStack}, 99), 1))) =^{(K3)}$

$\text{empty}(\text{push}(\text{CreateStack}, 99)) =^{(K2)} 0$

II: $\text{top}(\text{push}(\text{push}(\text{push}(\text{push}(\text{CreateStack}, 1), 7), 6), 2)) =^{(K4)} 2$

III: $\text{push}(\text{push}(\text{push}(\text{CreateStack}, 2),$
 $\text{top}(\text{pop}(\text{push}(\text{push}(\text{CreateStack}, 1), 3))))), 1) =^{(K3)}$
 $\text{push}(\text{push}(\text{push}(\text{CreateStack}, 2), \text{top}(\text{push}(\text{CreateStack}, 1))), 1) =^{(K4)}$
 $\text{push}(\text{push}(\text{push}(\text{CreateStack}, 2), 1), 1)$

Der Keller hat also die Form



c) Ein zweites Auswertungsverfahren wäre die Auswertung von außen nach innen. Das heißt, dass die Axiome zunächst möglichst weit außen und dann weiter innen angewendet werden:

$\text{empty}(\text{pop}(\text{push}(\text{push}(\text{pop}(\text{push}(\text{CreateStack}, 2)), 99), 1))) =^{(K3)}$
 $\text{empty}(\text{push}(\text{pop}(\text{push}(\text{CreateStack}, 2)), 99)) =^{(K2)} 0$

d) Wie man am Beispiel von Aufgabe c) sieht, ist die Auswertung von außen nach innen schneller. Hier lässt sich mittels den Axiomen K1 und K2 bzw. K3 das Ergebnis des Kellerausdrucks sofort angeben, sobald push (oder createStack) die zweit innerste Operation ist. Operationen die weiter innen auftreten interessieren nicht mehr.

Pseudocode:

Vorbedingung des Algorithmus ist die Eingabe eines gültigen Kellerausdrucks. Andernfalls terminiert er nicht, da dann $\text{auswertbar}(0)$ immer falsch ist.

```

Klammertiefe=0;
  Solange (!auswertbar(0))
    Klammertiefe = Klammertiefe+1;
    für alle Operationen der Klammertiefe
      falls (auswertbar(klammertiefe))
        auswerten(klammertiefe);
auswerten(0);

```

Lösung

6. Sortierverfahren und Aufwandsabschätzung

a. Pseudocode

Seien $x[1..n]$ und $y[1..n]$ Stapel der Höhe n .

```

yn=1;
solange(yn <= n) //Für alle Elemente aus Stapel y
  xn=2;          // Vergleich zwischen 1. und 1. Element unnötig

```

```

min=1;
solange(xn <= n-yn+1) //Für alle verbleibenden Elemente aus x
    falls(x[xn] < x[min]) //Suche minimales Element
        min=xn;
    xn=xn+1;

y[yn]=x[min]; //Kopiere minimales Element in Stapel y
x.del(min); // und lösche es aus Stapel x
yn++;

```

b. Stapel x Stapel y

```

6 2 9 5 leer
6 9 5 2
6 9 2 5
9 2 5 6
leer 2 5 6 9

```

c. Das Verfahren auf dem Übungsblatt erfordert 19 Vergleiche und 23 Zuweisungen. In der Tabelle werden die Vergleiche und Zuweisungen aufgelistet. Der Stapel x wird wie folgt durchnummeriert: $x_1=6$, $x_2=2$, $x_3=9$ und $x_4=5$ (Analog zu Stapel y).

Verfahren auf Übungsblatt

Stapel x	Stapel y	Vergleiche	# Ver-gleiche	Zuweisung	# Zu-weisung
6 2 9 5	leer	1* ($yn \leq n$) 4* ($xn \leq n-yn+1$) 2<6, 2<9, 2<5	8	yn=1, xn=2, min=1, 3* ($xn=xn+1$) y[1]=x[2], x.del[2], yn=yn+1	9
6 9 5	2	1* ($yn \leq n$) 3* ($xn \leq n-yn+1$) 9<6, 6<5	6	xn=2, min=1, 2*($xn=xn+1$) min=3, y[2]=x[3], x.del[3], yn=yn+1	8
6 9	2 5	1* ($yn \leq n$) 2* ($xn \leq n-yn+1$) 9<6	5	xn=2, min=1, xn=xn+1; y[3]=x[1], x.del[1], yn=yn+1	6

9	2 5 6	1* (yn<=n) 1* (xn <= n-yn+1)	2	xn=2, min=1, y[3]=x[1], x.del[1], yn=yn+1	5
leer	2 5 6 9	1* (yn<=n)	1		
			Gesamt 22		Gesamt 28

Lösung

7. Java: Sortieren durch Vertauschen

a. Pseudo-Code

```

vertauschung = true;
solange (vertauschung == true)
  i = #Index des ersten Elements des Stapels#
  vertauschung = falsch
  solange (x[i] nicht das letzte Elem. im Stapel)
    falls (x[i] > x[i+1])
      vertausche Werte von x[i] und x[i+1];
      vertauschung = true;
  nimm nächstes Stapелеlement

```

b.

```

import info1.*;
public class Vertausche{

    public static void main(String[] args ) {
        // Initialisierung
        int n=4;
        int[] a = new int[n];
        int b;
        boolean vertauschung;
        int zuweisung = 0;
        int vergleiche = 0;

        //Eingabe
        for(int j = 0; j < n; j++){
            System.out.print((j+1)+".tes Element:");
            a[j] = Console.in.readInt();
        }

        //Sortierung

        vertauschung = true;
        vergleiche++; //letzter Vergleich bei while-Schleife
        while(vertauschung == true){
            int i = 0;
            vertauschung = false;
            vergleiche++; //letzter Vergleich bei while-Schleife
            zuweisung++;
        }
    }
}

```

```

        while (i > n-1) {
            if (a[i] > a[i+1]) {
                b = a[i];
                a[i] = a[i+1];
                a[i+1] = b;
                vertauschung=true;
            }
            i++;
        }
    }

    //Ausgabe
    System.out.println("Sortierte Liste: ");
    for (int j = 0; j < n; j++)
        System.out.print(a[j]);
    System.out.println(" ");
    System.out.println("Anzahl der Zuweisungen: "+zuweisung);
    System.out.println("Anzahl der Vergleiche: "+vergleiche);
}
}
}

```

Lösung

8. Türme von Hanoi

Vorbemerkung: Ist bekannt, wie sich $n-1$ Scheiben von Platz a nach Platz b verlegen lassen, so ist auch bekannt, wie $n-1$ von Platz c nach a verlegt werden können. Dies ist offensichtlich, wenn man die Plätze beispielsweise einfach umbenennt.

Für $n=1$ ist die Lösung klar: Verlege eine Scheibe von Platz a nach Platz b.

Hat man bereits eine Zugfolge für $n-1$ Scheiben, dann ergibt sich daraus unmittelbar eine Zugfolge für n Scheiben:

Verlege n Scheiben von a nach b mittels Platz c

wie folgt:

Verlege $n-1$ Scheiben von a nach c mittels Platz b (bekannt n. Vor.)

Setze eine Scheibe vom Platz a auf Platz b (Platz a ist nun leer.)

Verlege $n-1$ Scheiben von c nach b mittels Platz a

Man beachte, daß beim Umlegen der $n-1$ oberen Scheiben die unterste nicht stört, weil sie größer als alle anderen ist.
