Fakultät IV

Klausur Informatik 1
02. März 2004

Jähnichen / Flik
Seifert / Neunast

Name: ....

Vorname: ....

Matr.-Nr.: ....

Bearbeitungszeit: 120 Minuten

- Als Hilfsmittel sind nur ein nicht programmierbarer Taschenrechner und ein doppelseitig und **in eigener Handschrift** beschriebenes DIN-A4-Blatt zugelassen.
- Benutzen Sie für die Lösung der Aufgaben nur das mit diesem Deckblatt ausgeteilte Papier. **Lösungen, die auf anderem Papier geschrieben werden, können nicht gewertet werden!**
- Schreiben Sie Ihre Lösungen auf das Aufgabenblatt der jeweiligen Aufgabe. Verwenden Sie auch die Rückseiten. **Schreiben Sie keine Lösungen einer Aufgabe auf ein Blatt, das nicht zu dieser Aufgabe gehört!** Wenn Sie zusätzliche, von uns ausgegebene Blätter verwenden, geben Sie unbedingt an, zu welcher Aufgabe die Lösung gehört!
- Schreiben Sie deutlich! Doppelte, unleserliche oder mehrdeutige Lösungen werden nicht gewertet! Streichen Sie gegebenenfalls eine Lösung durch!
- Schreiben Sie nur in **blau** oder **schwarz**. Lösungen, die mit Bleistift geschrieben sind, können nicht gewertet werden!
- Erscheint Ihnen eine Aufgabe mehrdeutig, wenden Sie sich an die Betreuer.
- Sollten Sie im Softwareteil eine Teilaufgabe nicht lösen können, so dürfen Sie die dort geforderte Funktion in anderen Teilaufgaben verwenden.
- Tragen Sie jetzt (vor Beginn der Bearbeitungszeit) auf *allen* Blättern Ihren Namen und Ihre Matrikelnummer ein.

Aufgabe	Punkte	erreicht
1	12	
2	10	
3	11	
4	2	
5	5	
6	3	
7	6	

Hinweise zur Bearbeitung der Opal-Aufgaben

Bei der Bearbeitung der Opal-Aufgaben kann auf `IMPORT`-, `SIGNATURE`- und `IMPLEMENTATION`-Deklarationen verzichtet werden. Es können alle Funktionen aus der Opal-Standardbibliothek verwendet werden. Eine nicht vollständige Liste der Strukturen und Funktionen aus der Bibliothek ist im folgenden angegeben.

```
SIGNATURE Bool
FUN true false :                bool
FUN not        : bool        -> bool
FUN and or     : bool ** bool -> bool
FUN =          : bool ** bool -> bool

SIGNATURE Denotation
FUN ++ : denotation ** denotation -> denotation
FUN #  : denotation -> nat

SIGNATURE Seq[data]
SORT data
TYPE seq == <>
            ::(ft : data, rt : seq)
FUN #       : seq -> nat
FUN ++      : seq ** seq -> seq
FUN exist?  : (data -> bool) ** seq -> bool
FUN forall? : (data -> bool) ** seq -> bool
FUN =       : (data ** data -> bool) -> seq ** seq -> bool

SIGNATURE SeqMap[from,to]
SORT from to
FUN map : (from -> to) -> seq[from] -> seq[to]

SIGNATURE SeqFilter[data]
SORT data
FUN filter : (data -> bool) -> seq[data] -> seq[data]

SIGNATURE SeqReduce[from,to]
SORT from to
FUN reduce : (from ** to -> to) ** to -> seq[from] -> to

SIGNATURE SeqZip[from1,from2,to]
SORT from1 from2 to
FUN zip : (from1 ** from2 -> to) -> seq[from1] ** seq[from2] -> seq[to]

SIGNATURE SeqOfSeq[data]
SORT data
FUN flat : seq[seq[data]]->seq[data]
```

1. Aufgabe (12 Punkte): Verwaltung von Lehrveranstaltungen

In der folgenden Aufgabe soll eine einfache Verwaltung von Lehrveranstaltungen und den dafür benötigten Räumen realisiert werden. In dieser Verwaltung ist es möglich, einen Raum für eine Veranstaltung zu reservieren. Dazu ist der folgende Datentyp `reservierung` gegeben.

```
TYPE reservierung == reservierung(raumname : denotation,
                                  zeit     : denotation,
                                  lvname   : denotation)
```

Eine Reservierung enthält somit einen Raumnamen, eine Zeit, und den Namen der Lehrveranstaltung, für die der Raum reserviert ist. Eine Reservierung für die "Informatik 1"-Vorlesung würde somit folgendermaßen aussehen: `reservierung("HE101", "FR 08:00-10:00", "INFORMATIK1")`.

Hinweis: Zeiten werden in dieser Aufgabe als Zeichenketten der Form "`FR 08:00-10:00`" dargestellt. Zwei Zeiten sind als gleich zu betrachten, wenn die entsprechenden Zeichenketten gleich sind.

Hinweis: Für die Lösung dieser Aufgabe können Sie die Funktion `istIn` benutzen, die feststellt, ob eine Zeichenkette in einer Liste von Zeichenketten vorkommt (`FUN istIn : denotation ** seq[denotation] -> bool`).

Beachten Sie: In den Unteraufgaben 1.1 und 1.3 dürfen **keine** Listenfunktionale benutzt werden.

1.1. Vorhandene Reservierungen (2 Punkte) Deklarieren und definieren Sie eine rekursive Funktion `buchungen`, die alle Zeiten liefert, zu denen ein bestimmter Raum reserviert ist. Eingaben der Funktion sind eine Liste mit Reservierungen und der Raumname.

Beachten Sie: Für die Lösung dieser Teilaufgabe dürfen Listenfunktionale **nicht** verwendet werden.

Lösung:

```
FUN buchungen : seq[reservierung] ** denotation -> seq[denotation]
DEF buchungen(<> , _) == <>
DEF buchungen(reservierung(r,z,_)::rest, raumName) ==
          IF raumName = r THEN z :: buchungen(rest, raumName)
          ELSE buchungen(rest, raumName) FI
```

1.2. Raumbelegung (2 Punkte) Deklarieren und definieren Sie eine Funktion `istFrei`, die überprüft, ob ein Raum zu einer bestimmten Zeit frei ist. Eingaben der Funktion sind eine Liste von Reservierungen, der Raumname und die Zeit.

Lösung:

```
FUN istFrei : seq[reservierung] ** denotation ** denotation -> bool
DEF istFrei(res, raumName, zeit) ==
  LET
    zeiten == buchungen(res,raumName)
  IN
    not(istIn(zeit, zeiten))
```

1.3. Freie Räume (3 Punkte) Deklarieren und definieren Sie eine rekursive Funktion `freieRaeume`, die die Raumnamen zurückliefert, die zu einer gegebenen Zeit frei sind. Eingaben der Funktion sind eine Liste von Reservierungen, die gewünschte Zeit und eine Liste mit allen Raumnamen.

Beachten Sie: Für die Lösung dieser Teilaufgabe dürfen Listenfunktionale **nicht** verwendet werden.

Lösung:

```
FUN freieRaeume : seq[reservierung] ** denotation ** seq[denotation] -> seq[denotation]
DEF freieRaeume(_ ,_ ,<>) == <>
DEF freieRaeume(res,zeit,raum::rest) ==
  IF istFrei(res,raum,zeit) THEN
    raum::freieRaeume(res,zeit,rest)
  ELSE
    freieRaeume(res,zeit,rest)
  FI
```

1.4. Raum buchen (1 Punkt) Deklarieren und definieren Sie eine Funktion `buchen`, die einen Raum bucht. Eingaben der Funktion sind eine Liste von Reservierungen und die gewünschte Reservierung. Falls die Reservierung nicht möglich ist, soll die Liste von Reservierungen nicht verändert werden.

Lösung:

```
FUN buchen :seq[reservierung] ** reservierung -> seq[reservierung]
DEF buchen(res ,reservierung(r,z,l)) ==
  IF istFrei(res,r,z) THEN
    reservierung(r,z,l)::res
  ELSE
    res
  FI
```

1.5. Überschneidungsfreiheit von Veranstaltungen (4 Punkte) Deklarieren und definieren Sie eine Funktion `ueberschneidungsfrei`, die feststellt, ob alle Lehrveranstaltungen in einer Liste zeitlich überschneidungsfrei sind. Eingaben der Funktion sind eine Liste von Reservierungen und eine Liste der Lehrveranstaltungsnamen.

Hinweis: Für die Lösung dieser Teilaufgabe können Sie die Funktion `lvZeiten` verwenden, die als Eingabe eine Liste von Reservierungen und einen Lehrveranstaltungsnamen erhält und als Resultat alle Zeiten liefert, zu denen für diese Veranstaltung ein Raum reserviert ist
(`FUN lvZeiten : seq[reservierung] ** denotation -> seq[denotation]`).

Lösung:

```
FUN ueberschneidungsfrei : seq[reservierung] ** seq[denotation] -> bool
DEF ueberschneidungsfrei(res, lvs) ==
                      not(mehrmals?(alleZeiten(res,lvs)))

FUN alleZeiten : seq[reservierung] ** seq[denotation] -> seq[denotation]
DEF alleZeiten(_, <>) == <>
DEF alleZeiten(res,lv::rest) == lvZeiten(res,lv) ++ alleZeiten(res,rest)


-- mehrmals ueberprueft ob ein die Sequenz ein Element enthaelt,
-- das mehrmals in der Sequenz vorkommt
FUN mehrmals? : seq[denotation] -> bool
DEF mehrmals?(<>) == false
DEF mehrmals?(a::R) == IF istIn(a,R) THEN true
                       ELSE mehrmals?(R) FI
```

oder

```
DEF ueberschneidungsfrei(res, lvs) ==
  LET
    alleZeiten == flat(map(\\l.lvZeiten(res,l))(lvs))
  IN
    not(mehrmals?(alleZeiten))
```

2. Aufgabe (10 Punkte): Stammbaum

Die verwandschaftliche Beziehung zwischen den Mitgliedern einer Familie kann mit Hilfe eines Stammbaums dargestellt werden. Dabei bezeichnet ein Knoten eine einzelne ledige Person (ein Single) oder ein verheiratetes Paar (bestehend aus Partner1 und Partner2). Zu diesen Familienmitgliedern werden alle direkten Nachkommen gespeichert, die wiederum ledige oder verheiratete Personen sein können.

Im Folgenden ist der Datentyp `stammbaum` gegeben, der es ermöglicht, einen solchen Stammbaum einer Familie zu speichern.

```
TYPE stammbaum == verheiratet (partner1   : person,
                               partner2   : person,
                               nachkommen : seq[stammbaum])
                  ledig       (single     : person,
                               nachkommen : seq[stammbaum])
```

2.1. Induzierte Signatur (3 Punkte) Geben Sie die induzierte Signatur des Datentyps `stammbaum` an. Benennen Sie dabei die einzelnen Komponenten und geben Sie an, um welche Art von Datentyp es sich bei `stammbaum` handelt. Welche der induzierten Funktionen sind nicht total? Begründen Sie Ihre Antwort.

Lösung:

```
-- Sorte
SORT stammbaum
-- Konstruktoren
FUN verheiratet  : person ** person ** seq[stammbaum] -> stammbaum
FUN leidig       : person ** seq[stammbaum] -> stammbaum
-- Diskriminatoren
FUN verheiratet? ledig?      : stammbaum -> bool
-- Selektoren
FUN single partner1 partner2 : stammbaum -> person
FUN nachkommen               : stammbaum -> seq[stammbaum]
```

Die Funktionen `single`, `partner1` und `partner2` sind nicht total, da es sich hierbei um Selektoren handelt, die nur für bestimmte Konstruktorvarianten definiert sind.

Der Datentyp `stammbaum` *ist ein Summentyp.*

2.2. Datentyp Deklaration (2 Punkte) In der Definition des Datentyps `stammbaum` wurde der Datentyp `person` benutzt. Deklarieren Sie zur Speicherung des Geschlechts einer Person einen Aufzählungstyp `geschlecht`. Deklarieren Sie weiterhin einen Datentyp `person`, der es ermöglicht zu einer Person Vorname, Nachname und das Geschlecht zu speichern.

Lösung:

```
TYPE geschlecht == maennlich weiblich

TYPE person == person(vorname    : denotation,
                      nachname   : denotation,
                      geschlecht : geschlecht)
```

2.3. Gleichheit von Personen (1 Punkt) Für die weitere Arbeit mit Stammbäumen wird eine Funktion benötigt, die feststellt, ob zwei Personen identisch sind, d.h. Name, Vorname und Geschlecht gleich sind.

Deklarieren und definieren Sie eine Funktion =, die überprüft, ob zwei Personen identisch sind. Gehen Sie davon aus, dass eine Funktion = mit der Funktionalität `FUN = : geschlecht ** geschlecht -> bool` definiert ist, die überprüft, ob zwei Geschlechter gleich sind.

Lösung:

```
FUN = : person ** person -> bool
DEF =(person(v1,n1,g1),person(v2,n2,g2)) == (v1=v2) and (n1=n2) and (g1=g2)
```

2.4. Geburt eines Kindes (4 Punkte) Deklarieren und definieren Sie eine Funktion `geburt`, die ein Kind als Nachkommen zu einem Stammbaum hinzufügt. Eingaben der Funktion sind zwei Personen und ein Stammbaum. Dabei stellt eine Person das neugeborene Kind und die andere Person einen Elternteil im Stammbaum dar. Als Ergebnis soll der neue Stammbaum zurückgeliefert werden.

Lösung:

```
FUN geburt : person ** person -> stammbaum -> stammbaum
DEF geburt(p1,p2)(verheiratet(P1,P2,nk)) ==
  IF (P1=p1) or (P2=p1) THEN
    verheiratet(P1,P2,ledig(p2,<>):: nk)
  ELSE
    verheiratet(P1,P2,map(geburt(p1,p2))(nk))
  FI
DEF geburt(p1,p2)(ledig(S,nk)) ==
  IF S=p1 THEN
    ledig(S,ledig(p2,<>):: nk)
  ELSE
    ledig(S,map(geburt(p1,p2))(nk))
  FI
```

3. Aufgabe (11 Punkte): Arithmetische Ausdrücke

In der folgenden Aufgabe werden einfache arithmetische Ausdrücke behandelt. Betrachtet werden dabei Variablen und Konstanten sowie die binären Operationen `plus` und `minus`, bei denen die Operanden wiederum Ausdrücke sind. Dazu ist der folgende Datentyp gegeben.

```
TYPE ausdruck == plus   (arg1 : ausdruck, arg2: ausdruck)
                 minus  (arg1 : ausdruck, arg2: ausdruck)
                 var    (name  : denotation)
                 konst  (wert  : real)
```

3.1. Substitution von Ausdrücken (3 Punkte) Deklarieren und definieren Sie eine Funktion `subst`, die in einem Ausdruck einen Teilausdruck durch einen neuen Ausdruck ersetzt. Eingaben der Funktion sind der gesamte Ausdruck, der zu ersetzende Teilausdruck und der neue Teilausdruck. Ein Aufruf von `subst(plus(var("n"), konst(3)), var("n"), konst(5))` liefert somit `plus(konst(5), konst(3))` als Ergebnis.

Hinweis: Es kann davon ausgegangen werden, dass ein Funktion = definiert ist, die `true` zurückliefert, falls zwei Ausdrücke gleich sind (`FUN =: ausdruck ** ausdruck -> bool`).

Lösung:

```
FUN subst: ausdruck ** ausdruck ** ausdruck -> ausdruck
DEF subst(plus(l, r), alt, neu) ==
  IF plus(l, r)=alt THEN neu
  ELSE
    plus(subst(l, alt, neu), subst(r, alt, neu))
  FI
DEF subst(minus(l, r), alt, neu) ==
  IF minus(l, r)=alt THEN neu
  ELSE
    minus(subst(l, alt, neu), subst(r, alt, neu))
  FI
DEF subst(e, alt, neu) == IF e=alt THEN neu ELSE e FI
```

3.2. Auswertung von Ausdrücken (3 Punkte) Deklarieren und definieren Sie eine Funktion `auswerten`, die einen Ausdruck auswertet. Eingaben der Funktion sind der auszuwertende Ausdruck und eine Belegungsfunktion, die als Eingabe einen Variablennamen bekommt und als Ergebnis deren Belegung als reellen Wert zurückliefert.

Hinweis: Es kann davon ausgegangen werden, dass die Belegungsfunktion für jede Variable eine gültige Belegung zurückliefert.

Lösung:

```
FUN auswerten: ausdruck ** (denotation -> real) -> real
DEF auswerten(plus(l, r), f) == auswerten(l, f) + auswerten(r, f)
DEF auswerten(minus(l, r), f) == auswerten(l, f) - auswerten(r, f)
DEF auswerten(var(n), f) == f(n)
DEF auswerten(konst(v), _) == v
```

Für die folgenden Teilaufgaben wird der Datentyp `ausdruck` so abgewandelt, dass anstelle der Funktionen `plus` und `minus` beliebige binäre reellwertige Funktionen benutzt werden können. Weiterhin ist jetzt ein Datentyp `belegung` zur Speicherung einer Variablenbelegung gegeben.

```
DATA ausdruckNeu == binOp(f    : real ** real -> real,
                          arg1 : ausdruckNeu,
                          arg2 : ausdruckNeu)
                    var  (name : denotation)
                    konst(wert : real)

DATA belegung == belegung(name: denotation, wert: real)
```

3.3. Auswertung von Ausdrücken (2 Punkte) Deklarieren und definieren Sie eine Funktion `auswertenNeu`, die einen Ausdruck und eine Liste von Variablenbelegungen übergeben bekommt und den Ausdruck auswertet.

Hinweis: Es kann davon ausgegangen werden, dass für jede Variable eine Belegung in der Liste existiert.

Lösung:

```
FUN auswertenNeu: ausdruckNeu ** seq[belegung] -> real
DEF auswertenNeu(binaer(f, l, r), S) == f(auswertenNeu(l, S), auswertenNeu(r, S))
DEF auswertenNeu(var(n), S) == wert(ft(filter(\\x. name(x) = n)(S)))
DEF auswertenNeu(konst(v), _) == v
```

3.4. Verknüpfung von Ausdrücken (3 Punkte) Deklarieren und definieren Sie eine Funktion `verknuepfen`, die eine Liste von Ausdrücken und eine kommutative Verknüpfungsfunktion bekommt und als Ergebnis einen Ausdruck liefert, der die Verknüpfung der Ausdrücke der Liste darstellt.

Hinweis: Gehen Sie bei der Lösung dieser Aufgabe davon aus, dass die übergebene Liste mindestens zwei Elemente enthält.

Lösung:

```
FUN verknuepfen: (real ** real -> real) ** seq[ausdruckNeu] -> ausdruckNeu
DEF verknuepfen(f, a1::a2::<>) == binaer(f(a1,a2))
DEF verknuepfen(f, e::E) == binaer(f, e, verknuepfen(f, E))
```

oder

```
DEF verknuepfen(f, e::E) == reduce(\\l,r. binaer(f,l,r),e)(E)
```

4. Aufgabe (2 Punkte): Codesicherung

4.1. (2 Punkte) Ermitteln Sie für den folgenden Datenstrom die CRC-Prüfbits (Generatorpolynom $x^5 + x^2 + 1$):

$$0 \quad 1 \quad 1 \quad 0 \quad 1 \quad 1 \quad 1 \quad 0 \quad 1 \quad 1 \quad 0 \quad 1$$

Geben Sie den zu versendenden Datenstrom an und kennzeichnen Sie die Prüfbits.

Lösung:

Lösung direkt auf der Bitfolge:

Das Generatorpolynom $G(x) = x^5 + x^2 + 1$ entspricht der Bitfolge 100101.

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

Die Prüfbits lauten: 00110. *Der zu versendende Datenstrom ist:* 01101110110100110

Lösung durch Polynomdivision:

x^{15}	$+x^{14}$	$+x^{12}$	$+x^{11}$	$+x^{10}$		$+x^8$	$+x^7$		$+x^5$				
$-(x^{15}$		$+x^{12}$		$+x^{10})$									
	x^{14}		$+x^{11}$			$+x^8$	$+x^7$		$+x^5$				
$-$	$(x^{14}$		$+x^{11}$		$+x^9)$								
					$+x^9$	$+x^8$	$+x^7$		$+x^5$				
				$-$	$(x^9$			$+x^6$		$+x^4)$			
						x^8	$+x^7$	$+x^6$	$+x^5$	$+x^4$			
					$-$	$(x^8$			$+x^5$		$+x^3)$		
							x^7	$+x^6$		$+x^4$	$+x^3$		
						$-$	$(x^7$			$+x^4$		$+x^2)$	
								x^6			$+x^3$	$+x^2$	
							$-$	$(x^6$			$+x^3$		$+x)$
												x^2	$+x$

5. Aufgabe (5 Punkte): Zahlendarstellungen

5.1. **(1 Punkt)** Wandeln Sie -251_{10} in eine 12-Bit-2-Komplementzahl um. Die Rechenschritte müssen erkennbar sein!

Lösung:

$$
\begin{aligned}
251/2 &= 125 \quad \textit{Rest } 1\\
125/2 &= 62 \quad \textit{Rest } 1\\
62/2 &= 31 \quad \textit{Rest } 0\\
31/2 &= 15 \quad \textit{Rest } 1\\
15/2 &= 7 \quad \textit{Rest } 1\\
7/2 &= 3 \quad \textit{Rest } 1\\
3/2 &= 1 \quad \textit{Rest } 1\\
1/2 &= 0 \quad \textit{Rest } 1\\
+251_{10} &= 000011111011_2\\
-251_{10} &= 111100000100_2 + 1 = 111100000101_2
\end{aligned}
$$

5.2. **(1 Punkt)** Subtrahieren Sie 1101.1001_2 von 0010.1011_2 (beides 8-Bit-2-Komplementzahlen). Kennzeichnen Sie deutlich, an welcher Stelle der Punkt im Ergebnis steht. Wie werden die 4 Bedingungsbits gesetzt?

Lösung:

	0	0	1	0.	1	0	1	1
-	1	1	0	1.	1	0	0	1
•	•		•					
	0	1	0	1.	0	0	1	0

$[z, n, c, v] = [0, 0, 1, 0]$

Hinweis: Für die nächsten beiden Teilaufgaben ist das aus der Vorlesung bekannte 16-Bit-Gleitpunktformat gegeben: 1-Bit-Vorzeichen, 4-Bit-Exponent, 11-Bit-Mantisse, Bias=7.

5.3. **(1 Punkt)** Stellen Sie 01001010.011000_2 in diesem 16-Bit-Gleitpunktformat dar.

Lösung:

$1001010.0110_2 = 1.001010011 \cdot 2^6$

Darstellung im Gleitpunktformat: $0'1101'00101001100$

5.4. **(2 Punkte)** Multiplizieren Sie die im o.g. 16-Bit-Gleitpunktformat dargestellten Zahlen 0100110011101100 und 0101100100000010 und geben Sie das Ergebnis wieder im 16-Bit-Gleitpunktformat an.

Lösung:

$0'1001'10011101100 = 1.100111011 \cdot 2^2$
$0'1011'00100000010 = 1.0010000001 \cdot 2^4$

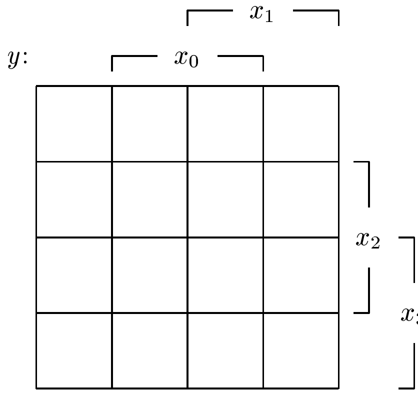
1.	1	0	0	1	1	1	0	1	1	·	1.	0	0	1	0	0	0	0	0	0	1	$\cdot 2^2 \cdot 2^4$
		1	1	0	0	1	1	1	0	1	1											
					1	1	0	0	1	1	1	0	1	1								
												1	1	0	0	1	1	1	0	1	1	
				•	•	•	•	•	•	•	•	•										
		1	1	1	0	1	0	0	0	1	1	0	0	1	0	1	1	1	0	1	1	$\cdot 2^6$

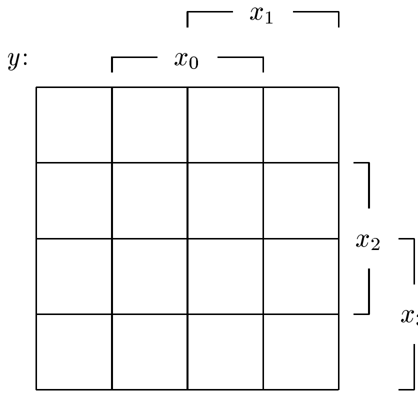
Darstellung im Gleitpunktformat: $0'1101'11010001100$

6. Aufgabe (3 Punkte): Minimierung boolescher Funktionen

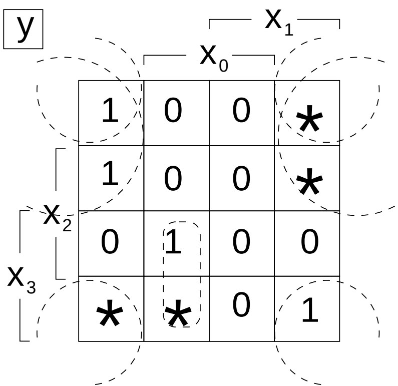
6.1. KV-Tafel (3 Punkte) Übertragen Sie die Werte der unvollständig definierten Funktion y in die nachstehenden KV-Tafeln. Ermitteln Sie daraus eine minimale DNF und eine minimale KNF für y.

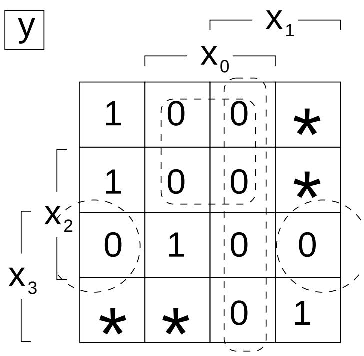
x_3	x_2	x_1	x_0	y
0	0	0	0	1
0	0	0	1	0
0	0	1	1	0
0	1	0	0	1
0	1	0	1	0
0	1	1	1	0
1	0	1	0	1
1	0	1	1	0
1	1	0	0	0
1	1	0	1	1
1	1	1	0	0
1	1	1	1	0





Lösung:



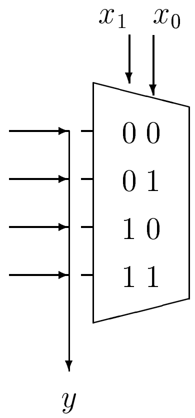


mDNF: $y = \overline{x_0} \cdot \overline{x_2} + \overline{x_0} \cdot \overline{x_3} + x_0 \cdot \overline{x_1} \cdot x_3$
mKNF: $\overline{y} = x_0 \cdot x_1 + x_0 \cdot \overline{x_3} + \overline{x_0} \cdot x_2 \cdot x_3 \Rightarrow y = (\overline{x_0} + \overline{x_1})(\overline{x_0} + x_3)(x_0 + \overline{x_2} + \overline{x_3})$

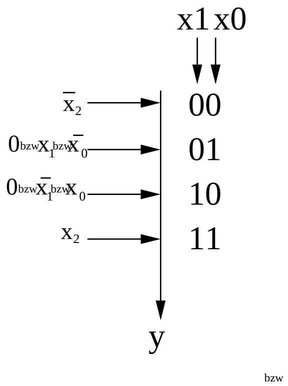
7. Aufgabe (6 Punkte): Schaltnetze

7.1. Multiplexer (2 Punkte) Realisieren Sie die mit der Wertetabelle vorgegebene boolesche Funktion y mittels der angegebenen Multiplexer-Schaltung. Geben Sie dazu die an den Eingängen des Multiplexers erforderlichen Ausdrücke an.

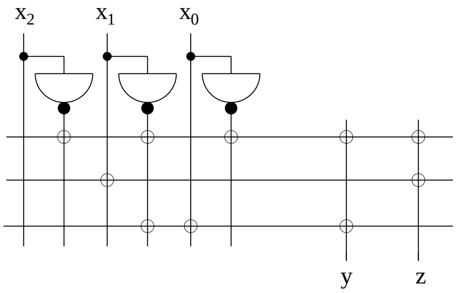
x_2	x_1	x_0	y
0	0	0	1
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	1



Lösung:



7.2. PLA (1 Punkt) Gegeben ist die folgende interne Beschaltung eines PLA. Tragen Sie die sich daraus ergebenden Werte für y und z in die vorgegebene Wertetabelle ein.

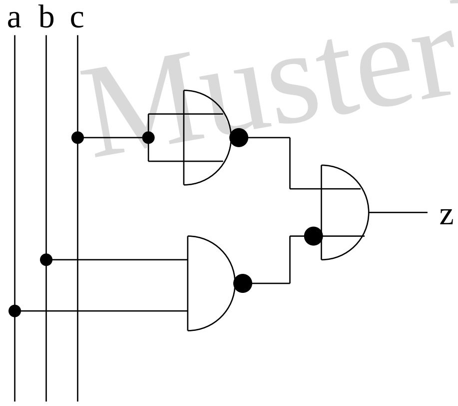


x_2	x_1	x_0	y	z
0	0	0		
0	0	1		
0	1	0		
0	1	1		
1	0	0		
1	0	1		
1	1	0		
1	1	1		

Lösung:

x_2	x_1	x_0	y	z
0	*0*	*0*	*1*	*1*
0	*0*	*1*	*1*	*0*
0	*1*	*0*	*0*	*1*
0	*1*	*1*	*0*	*1*
1	*0*	*0*	*0*	*0*
1	*0*	*1*	*1*	*0*
1	*1*	*0*	*0*	*1*
1	*1*	*1*	*0*	*1*

7.3. Gatter (2 Punkte) Gegeben ist das folgende Schaltnetz. Tragen Sie die Werte für z in die vorgegebene Wertetabelle ein und geben Sie eine boolesche Gleichung für z an.



a	b	c	z
0	0	0	
0	0	1	
0	1	0	
0	1	1	
1	0	0	
1	0	1	
1	1	0	
1	1	1	

Lösung:

a	b	c	z
0	0	0	1
0	0	1	0
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	1

$z = \overline{\overline{a \cdot b}} + \overline{c}$

7.4. ROM (1 Punkt) Eine unvollständig definierte boolesche Funktion y (d.h. mit "Don't-care"-Einträgen in der Wertetabelle) soll mit einem ROM-Baustein realisiert werden.
Welchen Wertebereich hat der Ausgang y des ROM-Bausteins für die nicht definierten Fälle?

Lösung:

$y \in \{0, 1\}$