Technische Universität München
Informatik / Lehrstuhl 7
Esparza/Schwoon

WS 2007/08
Übungsblatt 5
16. November 2007

Übung zu Logik

Bearbeiten Sie diese Aufgaben bis zur nächsten Übung am Freitag, 23. November, um 11:50 Uhr. Die Lösungen werden in der Übung besprochen.

Aufgabe 1 DPLL-Verfahren

(a) Führen Sie das DPLL-Verfahren mit folgenden Formeln durch. Welche von ihnen sind erfüllbar?

$$F_1 = \neg A \wedge (A \vee C) \wedge (B \vee \neg C)$$
$$F_2 = (\neg A \vee B) \wedge (\neg B \vee C) \wedge (\neg C \vee A)$$
$$F_3 = (A \vee \neg B) \wedge (C \vee \neg A) \wedge (A \vee B) \wedge (\neg A \vee \neg C)$$
$$F_4 = (A \vee \neg E) \wedge (\neg C \vee \neg D) \wedge (\neg A \vee B \vee D) \wedge (C \vee D \vee \neg E) \wedge (\neg A \vee \neg E) \wedge (\neg B \vee \neg E)$$

(b) Konstruieren Sie eine Formel F mit genau einer erfüllenden Belegung, so dass jede maximale Herleitung des DPLL-Verfahrens für F eine Anwendung der Splitting-Regel enthält.

Aufgabe 1 (Lösungsvorschlag) *DPLL-Verfahren*

(a) Wir gehen wie folgt vor:

- F_1 lässt sich unter dreifacher Anwendung der One-Literal-Regel wie folgt transformieren:

$$\Big\{F_1\Big\} \overset{\neg A}{\Longrightarrow} \Big\{\{\{C\},\{B,\neg C\}\}\Big\} \overset{C}{\Longrightarrow} \Big\{\{\{B\}\}\Big\} \overset{B}{\Longrightarrow} \Big\{\emptyset\Big\}$$

  Da die leere Formel hergeleitet wurde, ist F_1 erfüllbar. Die Variablen, auf die wir die One-Literal-Regel angewandt haben (auf den Pfeilen angedeutet), geben die erfüllende Belegung wieder.

  Alternativ kann man dreimal die Pure-Literal-Regel anwenden, auch hier sind die verwendeten Variablen angedeutet:

$$\Big\{F_1\Big\} \overset{B}{\Longrightarrow} \Big\{\{\{\neg A\},\{A,C\}\}\Big\} \overset{C}{\Longrightarrow} \Big\{\{\{\neg A\}\}\Big\} \overset{\neg A}{\Longrightarrow} \Big\{\emptyset\Big\}$$

- Hier muss zunächst die Splitting-Regel angewendet werden, zum Beispiel mit der Variablen A. Man erhält einen Block mit zwei Formeln:

  $$\Big\{F_2 \cup \big\{\{A\}\big\},\ F_2 \cup \big\{\{\neg A\}\big\}\Big\}$$

  Um die Erfüllbarkeit zu zeigen, reicht es ja aus, eine einzige Formel im Block auf die leere Formel zu reduzieren. Versucht man dies mit der linken Formel, so erhält man diese durch dreimalige Anwendung der One-Literal-Regel, wobei man alle Variablen positiv belegt. Alternativ kann man es mit der rechten Formel probieren, hier werden alle Variablen negativ. Die Formel ist also erfüllbar, man kann sogar zwei erfüllende Belegungen finden.

- Auch hier ist zunächst Splitting angesagt, sagen wir mit A:

  $$\Big\{F_3 \cup \big\{\{A\}\big\},\ F_3 \cup \big\{\{\neg A\}\big\}\Big\}$$

  Wendet man jetzt auf der linken Formel die One-Literal-Regel mit A an, so erhält man:

  $$\Big\{\big\{\{C\},\{\neg C\}\big\},\ F_3 \cup \big\{\{\neg A\}\big\}\Big\}$$

  Offensichtlich ist die linke Formel nicht erfüllbar, durch eine weitere Anwendung der One-Literal-Regel erhält man die leere Klausel. Versucht man sein Glück mit der rechten Formel, kommt man schnell zu $B \vee \neg B$ und weiter zur leeren Klausel. Die maximale Herleitung endet also mit:

  $$\Big\{\{\emptyset\},\ \{\emptyset\}\Big\}$$

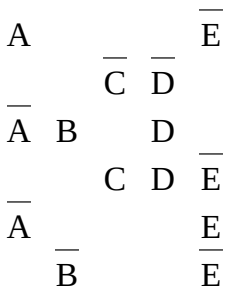
  Die Formel ist unerfüllbar, da in der maximale Herleitung jede Formel die leere Klausel enthält.

- Die in der Aufgabenstellung angegebene Formel F_4 ist erfüllbar, wie man durch dreimaliges Anwenden der Pure-Literal-Regel herausbekommt (zunächst mit $\neg E$, dann z.B. mit $\neg A$ und $\neg C$).

  Die Aufgabenstellung enthielt leider einen Tippfehler in der vorletzten Klausel, dessen Abwesenheit die Lösung interessanter gemacht hätte. Nachstehend die Lösung für die eigentlich beabsichtigte Formel

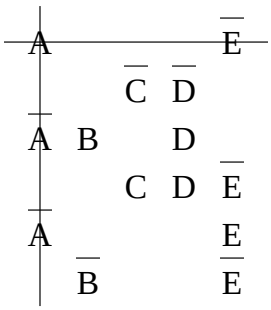
  $$F_4' = (A \vee \neg E) \wedge (\neg C \vee \neg D) \wedge (\neg A \vee B \vee D) \wedge (C \vee D \vee \neg E) \wedge (\neg A \vee E) \wedge (\neg B \vee \neg E)$$

  An diesem Beispiel soll eine andere Notation demonstriert werden, mit der man sich ggfs. Schreibarbeit sparen kann. Dazu ordnet man die Formel als Matrix an, wobei jede Zeile einer Klausel und jede Spalte einer Variablen entspricht. (Zuvor sollte man die Bereinigungsregel angewandt haben, falls notwendig.) In der Matrix schreiben wir $\overline{A}$ für $\neg A$ (nur der Kompaktheit halber). Die Matrix sieht für F_4' wie folgt aus:

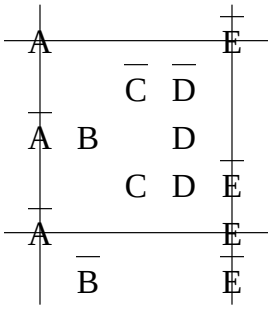


Die Anwendungen der Regeln lassen sich jetzt als das Streichen von Zeilen bzw. Spalten der Matrix interpretieren: Wendet man z.B. die One-Literal-Regel mit einer Klausel {A} an, so streicht man sämtliche Zeilen, die A enthalten (aber nicht $\neg A$) und dann die ganze Spalte, in der es evtl. noch Zeilen mit $\neg A$ gibt. Alles, was nicht gestrichen wurde, ist der Rest der Formel. Gelingt es, alle Zeilen zu streichen, so ist die Formel erfüllbar. Entsteht eine nicht gestrichene Zeile, in der alle Literale gestrichen sind, so ist sie unerfüllbar (leere Klausel). Beim Splitting kopiert man die Matrix, so dass man zwei Kopien hat, und wendet dann die One-Literal-Regel auf das positive bzw. negative Literal an.

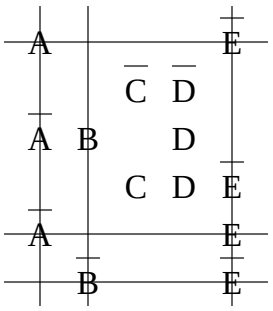
Im vorliegenden Fall könnten wir die Splitting-Regel für die Variable A anwenden und dann auf eine der Kopien die One-Literal-Regel für A. Das Ergebnis wäre wie folgt:



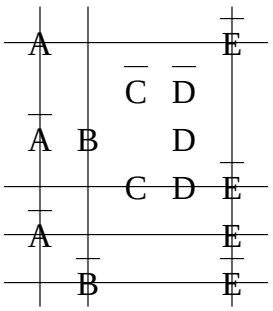
Wegen der zweiten Zeile von unten ist jetzt die One-Literal-Regel für E angesagt:



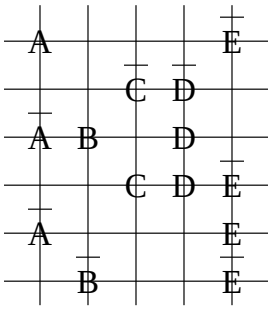
Anschließend die One-Literal-Regel für $\neg B$:



Jetzt kann die Subsumption-Regel auf die Zeile mit C und D angewendet werden, indem diese gestrichen wird:



Im weiteren Verlauf wendet man One-Literal-Regeln für D und $\neg C$ an. Im Endergebnis sind alle Zeilen gestrichen, die Formel ist also erfüllbar.



(Wäre man hier nicht auf Erfüllbarkeit gekommen, hätte man die Prozedur noch mit $\neg A$ wiederholen müssen.)

(b) Eine einfache Möglichkeit ist wie folgt:

$$F = (A \vee B) \wedge (\neg A \vee B) \wedge (A \vee \neg B)$$

Wie man sich leicht überzeugt, kann hier nur die Splitting-Regel angewendet werden. Die einzige erfüllende Belegung ist, beide Variablen auf 1 zu setzen. (Jede der drei Klauseln schließt eine der anderen Belegungen aus.)

Aufgabe 2 BDDs allgemein

Sei $F := (x \vee y) \rightarrow z$ und $G := x \leftrightarrow (y \wedge z)$, wobei $x < y < z$ gilt.

(a) Zeichnen Sie BDDs für F und G.

(b) In der Vorlesung wurde ein Algorithmus vorgestellt, der den Oder-Operator auf BDDs implementiert. Gewinnen Sie daraus einen BDD für $F \vee G$.

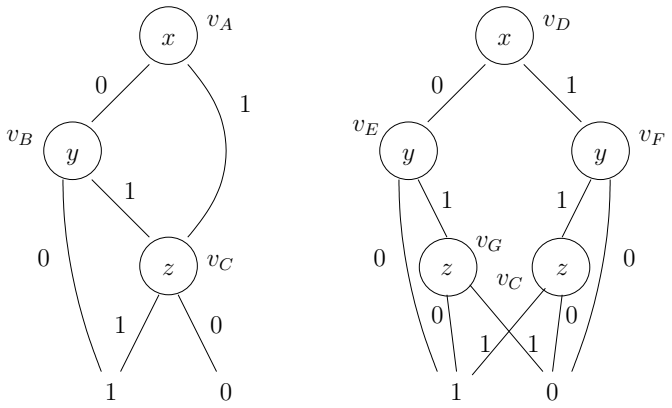
(c) Wie müsste man den Algorithmus aus der Vorlesung abwandeln, um einen BDD für $F \wedge G$ zu erhalten?

(d) Sei $H := ((z \leftrightarrow (x \oplus y)) \wedge (w \leftrightarrow (y \vee z))) \vee x \vee (u \leftrightarrow (x \vee w))$. Zeichnen Sie einen BDD für H. Wie viele erfüllende Belegungen hat H? Überlegen Sie, wie Sie diese Antwort direkt aus dem BDD ablesen können.
Hinweis: Sie können den BDD entweder selbst berechnen oder eines der BDD-Tols von der Vorlesungs-Webseite dafür benutzen. Bestimmen Sie die Anzahl der erfüllenden Belegungen, die es von jedem einzelnen Knoten aus gibt, und beginnen Sie bei den "untersten" Knoten.

(e) Überlegen Sie, wie man BDDs einsetzen könnte, um die Spiele Sudoku und Minesweeper zu lösen. Denken Sie an die Diskussionen aus den vorangegangenen Übungen. Wie löst man die dabei besprochenen Probleme?

Aufgabe 2 (Lösungsvorschlag) *BDDs allgemein*

(a) Die BDDs sehen wie folgt aus (links F, rechts G):



Systematisch konstruieren kann man sie, indem man die Äquivalenz

$$F \equiv ite(x, F[x/1], F[x/0])$$

anwendet (die auch hinter der Multi-BDD-Prozedur aus der Vorlesung steckt). Z.B. ist $F_1 := F[x/1] = z$ und $F_0 := F[x/0] = y \rightarrow z$, also ist die Wurzel ein Knoten mit x und Kindern, die den BDDs für F_1 und F_0 entsprechen. Diese Vorgehensweise wendet man rekursiv auf F_1 und F_0 an.

Für Aufgabe (b) sind die Knoten der BDDs bereits mit Namen versehen worden (v_A bis v_G). Man beachte, dass v_C in beiden BDDs vorkommt. (Würde man die BDDs für F und G in einem MultiBDD darstellen, so würde v_C nur einmal auftreten.)

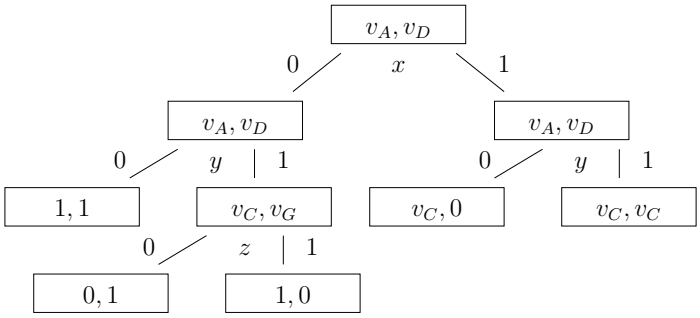
(b) Anmerkung: Zeilen 1 und 2 des Oder-Operators sind wie folgt:

if $v_F = K_1$ **or** $v_G = K_1$ **then return** K_1
else if $v_F = v_G = K_0$ **then return** K_0

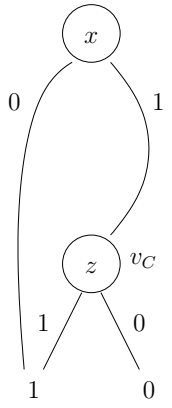
Zur Steigerung der Effizienz kann diese Zeilen ersetzen und zusätzliche Sonderfälle einführen:

if $v_F = K_1$ **or** $v_G = K_1$ **then return** K_1
else if $v_F = K_0$ **then return** v_G
else if $v_G = K_0$ **or** $v_F = v_G$ **then return** v_F

Die untenstehende Abbildung veranschaulicht den Ablauf des Algorithmus auf den beiden BDDs aus (a). Ein Kästchen bedeutet einen Prozeduraufruf mit den beiden Knoten, die im Kästchen genannt werden; trifft einer der oben genannten Fälle auf, so kann der Algorithmus direkt ein Resultat liefern, sonst kommt es zu zwei rekursiven Aufrufen für die Null- und die Eins-Kinder der kleinsten noch vorhandenen Variablen, die unter dem Kästchen genannt wird.



Alle Aufrufe auf der linken Hälfte liefern die 1 zurück, alle Aufrufe auf der rechten Hälfte den Knoten v_C. Damit ergibt sich folgendes Ergebnis:

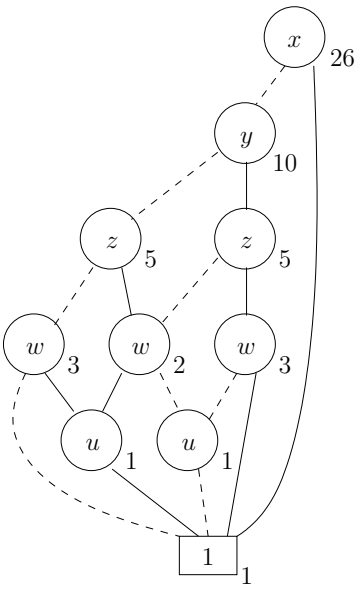


(c) Analog zum Oder-Operator ergeben sich folgende Sonderfälle:

if $v_F = K_0$ **or** $v_G = K_0$ **then return** K_0
else if $v_F = K_1$ **then return** v_G
else if $v_G = K_1$ **or** $v_F = v_G$ **then return** v_F

Die übrigen Zeilen bleiben unverändert.

(d) Untenstehende Zeichnung zeigt den BDD (mit DDcal berechnet). Der Übersicht halber wurden der 0-Knoten und die zu ihm führenden Kanten weggelassen sowie die Kantenbeschriftungen weggelassen (0-Kanten sind gestrichelt, 1-Kanten durchgezogen).



Die Anzahl der erfüllenden Belegungen berechnet man wie folgt, indem man jedem Knoten und jeder Kante ein Gewicht zuweist, das aussagt, wie viele erfüllende Belegungen der Knoten bzw. die Kante "beiträgt". Das geht wie folgt:

- Dem 1-Knoten wird das Gewicht 1 zugewiesen (eine einzige Belegung, nämlich die "leere Belegung"), dem 0-Knoten das Gewicht 0.
- Jedem anderen Knoten wird die Summe seiner beiden ausgehenden Kanten zugewiesen.
- Jeder Kante wird das Gewicht ihres Zielknotens zugewiesen, multipliziert mit 2 für jede "übersprungene" Variable.

Beispiel: Die 0-Kante des linken w-Knotens trägt das Gewicht 2, da sie zum 1-Knoten geht (mit Gewicht 1) und dabei eine Variable überspringt. Die 1-Kante dieses Knotens hat Gewicht 1. Insgesamt hat der Knoten damit das Gewicht 3.

Die Berechnung der Gewichte erfolgt von unten nach oben. Neben jedem Knoten ist sein Gewicht vermerkt (als Summe der beiden Kantengewichte, die nicht extra aufgeführt sind). Insgesamt gibt es also 26 erfüllende Belegungen.

(e) Grundsätzlich lassen sich BDDs einsetzen, um beide Spiele zu lösen.

Bei Sudoku übersetzt man die auf Blatt 2 besprochene Formel in einen BDD. Besteht dieser BDD nur als dem 0-Knoten, so gibt es keine Lösung. Andernfalls liest man irgendeine erfüllende Belegung ab und trägt diese ein. Dies ist allerdings weniger effizient als der Einsatz eines SAT-Solvers, aus zwei Gründen:

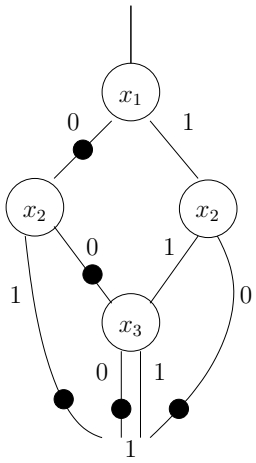
- Wir sind nur an *irgendeiner* Lösung interessiert. Ein SAT-Solver findet die erstbeste Lösung, gibt sie aus und terminiert. Bei einem BDD werden zunächst *sämtliche* Lösungen berechnet, dann lediglich eine abgelesen.
- Es ist schwierig, eine gute Variablenordnung für BDDs zu finden, da die Abhängigkeiten kreuz und quer über das gesamte Sudoku-Feld verteilt sind.

Bei Minesweeper sieht die Sache etwas freundlicher für BDDs aus. Abhängigkeiten zwischen Feldern sind nur lokal, so dass sich meistens eine gute Variablenordnung finden lässt. Außerdem sind wir explizit an Mehrdeutigkeiten interessiert. Übersetzen wir also ein Minesweeper-Feld, wie in Blatt 2 besprochen, in einen BDD B. Wenn er nur aus dem 0-Knoten besteht, gibt es keine Lösung. Andernfalls gehen den BDD Knoten für Knoten durch. Wenn es einen mit $v_{x,y}$ beschrifteten Knoten gibt, dessen 0-Kante nicht direkt zum 0-Knoten führt, so gibt es eine Lösung, in der das entsprechende Feld *keine* Mine enthält. Gibt es einen mit $v_{x,y}$ beschrifteten Knoten, dessen 1-Kante nicht direkt zum 0-Knoten führt, so gibt es eine Lösung, in der das Feld eine Mine enthält. Kommen beide Fälle vor, so kann das Feld noch nicht aufgedeckt werden. Wir decken alle Felder auf, die eindeutig sind. Anschließend kann man mit geeigneten BDD-Operationen (Konjunktion, Elimination der eindeutigen Variablen) B so modifizieren, dass die neue Spielsituation dargestellt wird.

Anmerkung: Die Vergleiche von BDDs und SAT-Solvern für die beiden Spiele sind rein theoretischer Natur, es wäre interessant, hierzu Experimente durchzuführen.

Aufgabe 3 Komplementkanten

Eine Variante von BDDs, mit der man oft kleinere Graphen erhält, sind *BDDs mit Komplementkanten* (KBDDs). Ein Beispiel für einen KBDD sehen Sie unten abgebildet. In einem KBDD können Kanten positiv oder negativ auftreten (in der Zeichnung unten sind negative Kanten mit einem schwarzen Kreis markiert). Außerdem entfällt das Blatt mit der 0, und die Wurzel des KBDD wird mit einer eingehenden Kante versehen.
Die Semantik einer negativen Kante ist, dass der Untergraph, der beim Zielknoten beginnt, als das Komplement der Formel gewertet wird, die er normalerweise darstellen würde. Eine negative Kante zum 1-Knoten wird z.B. als Kante zum 0-Knoten gewertet. Der mit x_3 beschriftete Knoten stellt daher die Formel x_3 dar, der linke der beiden x_2-Knoten aber die Formel $\neg x_2 \wedge \neg x_3$. Die eingehende Kante der Wurzel zeigt an, ob die dem KBDD entsprechende Formel oder ihr Komplement gemeint ist. Im Beispiel unten ist die eingehende Kante der Wurzel positiv.



(a) Stellen Sie die Wahrheitstafel für die Formel auf, die dem oben abgebildeten KBDD mit Komplementkanten entspricht.

(b) KBDDs sind i.A. nicht mehr eindeutig, d.h. es kann mehrere KBDDs geben, die dieselbe Formel darstellen. Finden Sie zwei Paare von KBDDs, die dieselbe Formel darstellen.

(c) KBDDs lassen sich allerdings eindeutig machen, wenn man darauf verzichtet, negative 0-Kanten zu benutzen. Zeigen Sie dazu, dass folgende Äquivalenz gilt:

$$ite(x, F, \neg G) \equiv \neg ite(x, \neg F, G)$$

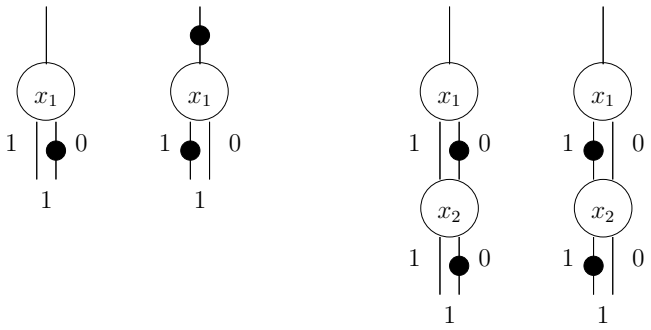
Benutzen Sie diese Äquivalenz, um den oben abgebildeten KBDD in den eindeutigen KBDD zu transformieren, der keine negativen 0-Kanten enthält.

Aufgabe 3 **(Lösungsvorschlag)** *Komplementkanten*

(a) Bemerkung: Eine Belegung liefert genau einem KBDD genau dann eine 1, wenn der zugehörige Pfad, der mit der eingehenden Kante der Wurzel beginnt und zum Blatt mit der 1 führt, eine *gerade Anzahl negativer Kanten* enthält. Daraus können wir folgende Wahrheitstafel ableiten:

x_1	x_2	x_3	
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	1

(b) Zwei Beispiele sind unten angegeben. Die beiden KBDDs auf der linken Seite stehen für die Formel x_1, die beiden auf der rechten Seite für die Formel x_2.



(c) Wir nutzen die Äquivalenz

$$ite(x, F, G) \equiv (x \rightarrow F) \wedge (\neg x \rightarrow G) \equiv (x \wedge F) \vee (\neg x \wedge G)$$

und zeigen:

$$\begin{aligned} ite(x, F, \neg G) &\equiv (x \rightarrow F) \wedge (\neg x \rightarrow \neg G) \\ &\equiv (\neg x \vee F) \wedge (x \vee \neg G) \\ &\equiv \neg(x \wedge \neg F) \wedge \neg(\neg x \wedge G) \\ &\equiv \neg\big((x \wedge \neg F) \vee (\neg x \wedge G)\big) \\ &\equiv \neg ite(x, \neg F, G) \end{aligned}$$

Einen Knoten n, von dem eine negative 0-Kante ausgeht, wandelt man also dergestalt um, dass man den Zustand *aller* ein- und ausgehenden Kanten von n umkehrt (d.h. aus negativen Kanten positive macht und umgekehrt). Dabei beginnt man sinnvollerweise mit den Knoten, die in der Variablenordnung ganz unten stehen und arbeitet sich Schritt für Schritt nach oben vor. Im vorliegenden Fall erhält man den unten angegebenen KBDD; man überzeuge sich, dass seine Wahrheitstafel mit der des ursprünglichen KBDDs übereinstimmt.