# Kapitel 4

# Aussagenlogik

Aussagenlogik war das erste logische System, das als mathematische Logik formuliert werden konnte (George Boole, Laws of Thought, 1854). Aussagenlogik ist die einfachste Logik und findet sich als Teillogik in den meisten anderen Logiken. Aussagenlogik hat wichtige Anwendungen in der Informatik, zum Beispiel bei der Beschreibung und Analyse von Schaltkreisen oder bei der Realisierung effizienter Datenstrukturen für endliche Mengen.

## 4.1 Boolesche Algebren

Das Konzept der Booleschen Algebra ist eine algebraische Abstraktion, das wichtige Aspekte der Aussagenlogik formuliert. Die Idee der algebraischen Abstraktion ist Ihnen bereits durch die Beispiele „Gruppe" und „Körper" vertraut.

Eine *Boolesche Algebra* ist ein Tupel $(B, 0, 1, \wedge, \vee, \neg)$ wie folgt:

1. $B$ ist eine Menge und $0, 1 \in B$.
2. $\wedge$ und $\vee$ sind Funktionen $B \times B \to B$.
3. Die Gleichungen in Abbildung 4.1 gelten für alle $x, y, z \in B$.

Die Gleichungen in Abbildung 4.1 bezeichnet man als die *Axiome für Boolesche Algebren*.[1] Die Funktionen $\wedge$, $\vee$ und $\neg$ bezeichnet man als *Konjunktion*, *Disjunktion* und *Negation*. Die Menge $B$ bezeichnet man als den *Träger* der Algebra. Den Träger einer Algebra $\mathcal{B}$ bezeichnet man auch mit $|\mathcal{B}|$. Schließlich bezeichnet man

[1] Die Klasse der Booleschen Algebren kann kompakter mit den Gleichungen für Kommutativität, Distributivität, Komplementierung und Identität beschrieben werden, da die anderen Gleichungen in Abbildung 4.1 aus diesen folgen. Die Beweise dafür findet man in: J. Eldon Whitesitt, *Boolean algebra and its applications*, Addison Wesley, 1961.

$$(x \wedge y) \wedge z = x \wedge (y \wedge z) \qquad \textit{(Assoziativität)}$$
$$(x \vee y) \vee z = x \vee (y \vee z)$$

$$x \wedge y = y \wedge x \qquad \textit{(Kommutativität)}$$
$$x \vee y = y \vee x$$

$$x \wedge x = x \qquad \textit{(Idempotenz)}$$
$$x \vee x = x$$

$$x \wedge (y \vee z) = (x \wedge y) \vee (x \wedge z) \qquad \textit{(Distributivität)}$$
$$x \vee (y \wedge z) = (x \vee y) \wedge (x \vee z)$$

$$x \wedge (x \vee y) = x \qquad \textit{(Absorption)}$$
$$x \vee (x \wedge y) = x$$

$$\neg(x \wedge y) = \neg x \vee \neg y \qquad \textit{(de Morgan)}$$
$$\neg(x \vee y) = \neg x \wedge \neg y$$

$$\neg\neg x = x \qquad \textit{(Doppelnegation)}$$

$$x \wedge \neg x = 0 \qquad \textit{(Komplemente)}$$
$$x \vee \neg x = 1$$

$$x \wedge 1 = x \qquad \textit{(Identitäten)}$$
$$x \vee 0 = x$$

Abbildung 4.1: Axiome für Boolesche Algebren

die Objekte 0 und 1 als die *Konstanten* der Algebra, und die Funktionen $\wedge$, $\vee$ und $\neg$ als die *Operationen* der Algebra.

Wir benutzen die Symbole 0, 1, $\wedge$, $\vee$ und $\neg$ für Boolesche Algebren, obwohl wir diese auch für andere Zwecke verwenden (zum Beispiel 0 für die Zahl 0). Die richtige Lesart eines Auftretens dieser Symbole muss also aus dem Kontext seiner Verwendung abgeleitet werden. Die Mehrfachverwendung von Symbolen bezeichnet man im Zusammenhang mit Programmiersprachen als Überladung.

Die Theorie der Booleschen Algebren bezeichnet man als *Boolesche Algebra*.

Beachten Sie, dass alle Axiome, die $\wedge$, $\vee$, 0 oder 1 enthalten, jeweils in einer zweiten Version vorkommen, die $\wedge$ mit $\vee$ und 0 mit 1 vertauscht. Man sagt, dass die eine Version *dual* zur anderen ist. Die perfekte Symmetrie zwischen $\wedge$ und $\vee$ einerseits und 1 und 0 andererseits bezeichnet man als das *Dualitätsprinzip* der Booleschen Algebra.

### Die zweiwertige Boolesche Algebra $\mathcal{T}$

Die zweielementigen Menge[2]

$$\mathbb{B} \stackrel{\text{def}}{=} \{0, 1\}$$

wird durch die folgenden Festlegungen zu einer Booleschen Algebra $(\mathbb{B}, 0, 1, \wedge, \vee, \neg)$:

$$\begin{aligned} 0 &= 0 \\ 1 &= 1 \\ x \wedge y &= \min\{x, y\} \\ x \vee y &= \max\{x, y\} \\ \neg x &= 1 - x \end{aligned}$$

Diese Algebra nennen wir *die zweiwertige Boolesche Algebra* und bezeichnen sie mit $\mathcal{T}$.

### Die Potenzmengenalgebren $\mathcal{P}_M$

Sei $M$ eine Menge. Dann wird die Potenzmenge $\mathcal{P}(M)$ durch die folgenden Festlegungen zu einer Booleschen Algebra $(\mathcal{P}(M), 0, 1, \wedge, \vee, \neg)$:

$$\begin{aligned} 0 &= \emptyset \\ 1 &= M \\ x \wedge y &= x \cap y \\ x \vee y &= x \cup y \\ \neg x &= M - x \end{aligned}$$

Die Boolesche Algebra $(\mathcal{P}(M), 0, 1, \wedge, \vee, \neg)$ bezeichnen wir mit $\mathcal{P}_M$.

Machen Sie sich klar, dass die zweiwertige Boolesche Algebra $\mathcal{T}$ genau dieselbe Struktur hat wie die Potenzmengenalgebra $\mathcal{P}_{\{\emptyset\}}$. Man sagt, dass $\mathcal{T}$ und $\mathcal{P}_{\{\emptyset\}}$ *isomorph* sind. Isomorphie von zwei Booleschen Algebren bedeutet, dass es eine Bijektion zwischen den Trägern der Algebren gibt, so dass sich die Konstanten

[2] Statt 0 und 1 kann man auch andere mathematische Objekte als Elemente für $\mathbb{B}$ wählen (zum Beispiel $\emptyset$ und $\{\emptyset\}$). Wichtig ist nur, dass $\mathbb{B}$ eine zweielementige Menge ist. Die Wahl der Zahlen 0 und 1 ist bequem, da wir damit auf die Notationen für Zahlen zurückgreifen können.

0 und 1 und die Operationen $\wedge$, $\vee$ und $\neg$ auf beiden Seiten genau entsprechen. M. Stone hat 1936 gezeigt, dass die Booleschen Algebren bis auf Isomorphie gerade die Potenzmengenalgebren sind.

### Boolesche Gleichungen und Tautologien

Ein *Boolescher Ausdruck* ist ein Ausdruck, der nur mit Variablen sowie 0, 1, $\wedge$, $\vee$, und $\neg$ gebildet ist. Eine *Boolesche Gleichung* ist eine Gleichung, die aus zwei Booleschen Ausdrücken gebildet ist. Eine Boolesche Gleichung ist in einer Booleschen Algebra $\mathcal{B}$ *gültig*, wenn beide Seiten der Gleichung für alle Variablenbelegungen in $\mathcal{B}$ den gleichen Wert liefern.

Eine Boolesche Gleichung ist *gültig*, wenn sie in jeder Booleschen Algebra gültig ist. Die Axiome für Boolesche Algebren und alle daraus abgeleiteten Booleschen Gleichungen sind gültig.

Eine *Tautologie* ist eine Boolesche Gleichung, die die in der zweiwertigen Booleschen Algebra gültig ist. Jede gültige Boolesche Gleichung ist eine Tautologie (da $\mathcal{T}$ eine Boolesche Algebra ist). Wenn eine Boolesche Gleichung keine Tautologie ist, dann ist sie nicht gültig.

Wir werden später zeigen, dass jede Tautologie eine gültigen Booleschen Gleichung ist. Das bedeutet, dass eine Boolesche Gleichung genau dann in allen Booleschen Algebren gültig ist, wenn sie in der zweiwertigen Booleschen Algebra gültig ist. Dieser sogenannte Tautologiesatz ist ein spektakuläres Resultat. Beispielsweise sind wir damit in der Lage, Gleichungen für Mengen mithilfe von Wahrheitstafeln zu verifizieren. Natürlich werden wir den Tautologiesatz erst benutzen, nachdem wir ihn bewiesen haben.

### Verifikation von Tautologien

Da eine Variable in der zweiwertigen Booleschen Algebra nur die Werte 0 und 1 annehmen kann, kann man für eine Boolesche Gleichung durch mechanisches Ausrechnen feststellen, ob es sich um eine Tautologie handelt. Dieses Nachrechnen lässt sich mithilfe sogenannter *Wahrheitstafeln* übersichtlich darstellen. Wir zeigen dies am Beispiel der Gleichung

$$(\neg x \vee y) \wedge (x \vee \neg y) = (x \wedge y) \vee (\neg x \wedge \neg y)$$

für die wir die folgende Wahrheitstafel erhalten:

| $x$ | $y$ | $(\neg x \vee y) \wedge (x \vee \neg y)$ | $(x \wedge y) \vee (\neg x \wedge \neg y)$ |
|---|---|---|---|
| 0 | 0 | 1 | 1 |
| 0 | 1 | 0 | 0 |
| 1 | 0 | 0 | 0 |
| 1 | 1 | 1 | 1 |

Jede der vier Zeilen im Rumpf der Tabelle entspricht einer Variablenbelegung, wobei nur die in der Gleichung vorkommenden Variablen betrachtet werden. Da die linke und die rechte Seite der Gleichung für alle Variablenbelegungen den gleichen Wert liefern, ist die Gleichung eine Tautologie.

### Notationen

Für Boolesche Algebren führen wir die folgenden Notationen ein:

$$x - y \stackrel{\text{def}}{=} x \wedge \neg y \qquad \textit{Differenz}$$

$$x \Rightarrow y \stackrel{\text{def}}{=} \neg x \vee y \qquad \textit{Implikation}$$

$$x \Leftrightarrow y \stackrel{\text{def}}{=} (x \Rightarrow y) \wedge (y \Rightarrow x) \qquad \textit{Äquivalenz}$$

Man rechnet leicht nach, dass in der zweiwertigen Booleschen Algebra für alle $x, y \in \mathbb{B}$ gilt:

$$\begin{aligned} x - y &= \text{if } y = 0 \text{ then } x \text{ else } 0 \\ x \Rightarrow y &= \text{if } x = 1 \text{ then } y \text{ else } 1 \\ x \Leftrightarrow y &= \text{if } x = y \text{ then } 1 \text{ else } 0 \end{aligned}$$

Damit wir nicht so viele Klammern schreiben müssen, legen wir eine Rangfolge für die Booleschen Infixoperatoren fest: $\wedge$, $\vee$, $-$, $\Rightarrow$, $\Leftrightarrow$ (aufsteigend). Damit können wir den Ausdruck

$$((\neg x \wedge y) \vee z) \Rightarrow (x \wedge y)$$

auch ohne Klammern beschreiben:

$$\neg x \wedge y \vee z \Rightarrow x \wedge y$$

Außerdem legen wir fest, dass die Booleschen Infixoperatoren rechtsassoziativ geklammert werden.

Oft ist es praktisch, die folgenden Notationen für Negation, Konjunktion und Disjunktion zu verwenden:

$$\begin{aligned} \overline{x} &\stackrel{\text{def}}{=} \neg x \\ x \cdot y &\stackrel{\text{def}}{=} x \wedge y \\ x + y &\stackrel{\text{def}}{=} x \vee y \end{aligned}$$

Wenn man $x \cdot y$ wie bei Zahlen noch zu $xy$ verkürzt, kann man Boolesche Ausdrücke kompakt und übersichtlich schreiben. Zum Beispiel:

$$\overline{x}(x+\overline{y}) + y(x+\overline{y}) = \neg x \wedge (x \vee \neg y) \vee y \wedge (x \vee \neg y)$$

### Einige gültige Boolesche Gleichungen

Wir demonstrieren an Beispielen, wie man mithilfe der Axiome zeigen kann, dass eine Gleichung in allen Booleschen Algebren gültig ist. Einige der hier gezeigten Gleichungen benötigen wir für spätere Beweise.

**Proposition 4.1.1** *In jeder Booleschen Algebra gelten folgende Gleichungen:*

$$x \wedge 0 = 0, \quad x \vee 1 = 1, \quad \neg 0 = 1, \quad \neg 1 = 0$$

**Beweis** Die erste Gleichung kann man wie folgt zeigen:

$$\begin{aligned} x \wedge 0 &= x \wedge (x \wedge \neg x) && \text{Komplemente} \\ &= (x \wedge x) \wedge \neg x && \text{Assoziativität} \\ &= x \wedge \neg x && \text{Idempotenz} \\ &= 0 && \text{Komplemente} \end{aligned}$$

Die zweite Gleichung folgt analog. Die dritte Gleichung erhält man wie folgt:

$$\begin{aligned} \neg 0 &= \neg(0 \wedge \neg 0) && \text{Komplemente} \\ &= \neg 0 \vee \neg\neg 0 && \text{de Morgan} \\ &= \neg 0 \vee 0 && \text{Doppelnegation} \\ &= 0 \vee \neg 0 && \text{Kommutativität} \\ &= 1 && \text{Komplemente} \end{aligned}$$

Die vierte Gleichung folgt analog. □

Da wir mit Assoziativität und Kommutativität von Operationen von den Zahlen her bestens vertraut sind, werden wir die Anwendung der entsprechenden Axiome nicht mehr erwähnen.

 

**Proposition 4.1.2** *In jeder Booleschen Algebra gilt:*

1. $\overline{x}y + xz = \overline{x}y + xz + yz$ *(Resolution)*
2. $\overline{x}y + xz = (\overline{x} + z)(x + y)$
3. $(x \Leftrightarrow y) = xy + \overline{x}\,\overline{y}$
4. $(\overline{x}y + xz)(\overline{x}u + xv) = \overline{x}yu + xzv$
5. $\overline{\overline{x}y + xz} = \overline{x}\,\overline{y} + x\overline{z}$

**Beweis** Gleichung (1) zeigen wir wie folgt:

$$\begin{aligned} \overline{x}y + xz &= \overline{x}(\overline{x} + z)y + x(x + y)z && \text{Absorption} \\ &= \overline{x}y + xz + \overline{x}yz + xyz \\ &= \overline{x}y + xz + (\overline{x} + x)yz \\ &= \overline{x}y + xz + yz \end{aligned}$$

Gleichung (2) folgt aus (1). Gleichung (3) folgt aus (2). Gleichungen (4) ist einfach. Gleichung (5) folgt mit (2). □

**Konditionale**

Wir führen die folgende Notation für Boolesche Algebren ein:

$$(x, y, z) \stackrel{\text{def}}{=} \overline{x}y + xz \qquad \textit{Konditional}$$

**Proposition 4.1.3** *In der zweiwertigen Booleschen Algebra $\mathcal{T}$ gilt für alle $x, y, z \in \mathbb{B}$:*

$$(x, y, z) = \text{if } x = 0 \text{ then } y \text{ else } z$$

Mit der Konditionalnotation können wir eine kanonische Form für Boolesche Ausdrücke formulieren, die die Grundlage für effiziente Lösungsalgorithmen für Booleschen Gleichungen liefert. Die Ideen dafür stammen von Claude Shannon (1938) und Randal Bryant (1986).

**Proposition 4.1.4** *In jeder Booleschen Algebra gilt:*

1. $(x, y, y) = y$
2. $\overline{(x, y, z)} = (x, \overline{y}, \overline{z})$
3. $(x, y, z) \cdot u = (x, y \cdot u, z \cdot u)$
4. $(x, y, z) \cdot (x, u, v) = (x, y \cdot u, z \cdot v)$
5. $(x, y, z) + u = (x, y + u, z + u)$
6. $(x, y, z) + (x, u, v) = (x, y + u, z + v)$
7. $(x, y, z) = (\overline{x} + z)(x + y)$
8. $\overline{(x, y, z)} = (\overline{x}, \overline{y}, \overline{z})$

**Beweis** Behauptung (1) ist einfach. Behauptung (2) folgt sofort aus Proposition 4.1.2 (5). Behauptung (3) ist einfach. Behauptung (4) folgt sofort aus Proposition 4.1.2 (4). Behauptungen (5) folgt mit $x + y = \overline{\overline{x}\,\overline{y}}$ aus Behauptung (3). Behauptungen (6) folgt ebenfalls mit $x + y = \overline{\overline{x}\,\overline{y}}$ aus Behauptung (4). Behauptung (7) folgt mit Proposition 4.1.2 (1). Behauptung (8) folgt aus (7). $\square$

Aus dem Beweis für die Behauptungen (5) und (6) sieht man, dass für jede binäre Operation $\circ$, die mit $\wedge$ und $\neg$ ausdrückbar ist (analog zu $x + y = \overline{\overline{x}\,\overline{y}}$), die folgenden Gleichungen gelten:

$$
\begin{aligned}
(x, y, z) \circ u &= (x, y \circ u, z \circ u) \\
(x, y, z) \circ (x, u, v) &= (x, y \circ u, z \circ v)
\end{aligned}
$$

### Reduktion auf 1, $\wedge$ und $\neg$

Aus der Gültigkeit der Gleichungen

$$
\begin{aligned}
0 &= \neg 1 \\
x \vee y &= \neg(\neg x \wedge \neg y)
\end{aligned}
$$

folgt, dass man zu jedem Booleschen Ausdruck $a$ einen Booleschen Ausdruck $a'$ bestimmen kann, sodass $a = a'$ gültig ist und $a'$ nur mit $1, \wedge, \neg$ und in $a$ vorkommenden Variablen gebildet ist.

Analog kann man auch auf $0, \vee$ und $\neg$ reduzieren.

### Bedingte Gleichungen

**Proposition 4.1.5** *In jeder Booleschen Algebra gilt:*

1. $x = y \iff \neg x = \neg y$
2. $x \wedge y = 1 \iff x = 1$ *und* $y = 1$
3. $x \vee y = 0 \iff x = 0$ *und* $y = 0$

**Beweis** Wir beginnen mit der Richtung „$\Leftarrow$" der ersten Behauptung. Sei $\neg x = \neg y$. Dann $\neg\neg x = \neg\neg y$. Also $x = y$ mit dem Doppelnegationsaxiom.

Als Nächstes zeigen wir die Richtung „$\Rightarrow$" der ersten Behauptung. Sei $x = y$. Das Doppelnegationsaxiom liefert $\neg\neg x = \neg\neg y$. Jetzt folgt $\neg x = \neg y$ mit der bereits bewiesenen Richtung „$\Leftarrow$" von Behauptung (1).

Die Richtung „$\Leftarrow$" der zweiten Behauptung folgt mit dem Identitätsaxiom für Konjunktion. Die Richtung „$\Rightarrow$" zeigen wir wie folgt. Sei $x \wedge y = 1$. Dann:

$$
\begin{aligned}
x &= x \wedge 1 && \text{Identität} \\
&= x \wedge (x \wedge y) && \text{Annahme} \\
&= x \wedge y && \text{Idempotenz} \\
&= 1 && \text{Annahme}
\end{aligned}
$$

Also $x = 1$. Die Gleichung $y = 1$ folgt analog.

Behauptung (3) lässt sich analog zu Behauptung (2) zeigen. □

### Normalisierung

**Proposition 4.1.6 (Normalisierung)** *Für jede Boolesche Algebra gilt:*

$$x = y \iff (x \Leftrightarrow y) = 1$$

**Beweis** Sei $x = y$. Dann $(\neg x + y)(x + \neg y) = 1$. Also $(x \Leftrightarrow y) = 1$ nach Definition von $\Leftrightarrow$.

Sei $(x \Leftrightarrow y) = 1$. Dann $(\overline{x} + y)(x + \overline{y}) = 1$ nach Definition von $\Leftrightarrow$. Also $\overline{x} + y = 1$ und $x + \overline{y} = 1$ nach Proposition 4.1.5. Folglich $x = x \cdot 1 = x(\overline{x} + y) = x \cdot \overline{x} + x \cdot y = x \cdot y$ und $y = y \cdot 1 = y(x + \overline{y}) = y \cdot x + y \cdot \overline{y} = x \cdot y$. Also $x = y$. □

**Proposition 4.1.7 (Normalisierung)** *Zu jedem endlichen System S von Booleschen Gleichungen kann man einen Booleschen Ausdruck a bilden, sodass die Gleichungen in S genau dann gelten, wenn die Gleichung $a = 1$ gilt (in jeder Booleschen Algebra).*

**Beweis** Zunächst bringen wir die Gleichungen von $S$ mithilfe von Proposition 4.1.6 in die Form $a_i = 1$. Jetzt folgt mit Proposition 4.1.5, dass die Gleichungen $a_1 = 1, \ldots, a_n = 1$ genau dann gültig sind, wenn die Gleichung $(a_1 \wedge \cdots \wedge a_n) = 1$ gültig ist. $\square$

## 4.2 Subsumtionsordnung

Die Elemente einer Potenzmengenalgebra $\mathcal{P}_M$ sind durch die Inklusionsordnung $X \subseteq Y$ für Mengen partiell geordnet. Interessanterweise kann man allgemein für Boolesche Algebren eine partielle Ordnung definieren, die für die Potemzmengenalgebren mit der Inklusionsordnung für Mengen übereinstimmt.

Sei $\mathcal{B}$ eine Boolesche Algebra. Dann definiert

$$x \leq y \quad \overset{\text{def}}{\iff} \quad x \wedge y = x$$

eine binäre Relation $\leq$ auf $|\mathcal{B}|$, die wir als die *Subsumtionsordnung von* $\mathcal{B}$ bezeichnen. Da wir keine weiteren Ordnungen auf Booleschen Algebren betrachten werden, sprechen wird auch kurz von der *Ordnung von* $\mathcal{B}$.

Sei $M$ eine Menge. Die Ordnung der Potenzmengenalgebra $\mathcal{P}_M$ ist offensichtlich wie angekündigt gerade die Inklusionsordnung der Potenzmenge $\mathcal{P}(M)$.

Die Ordnung der zweiwertigen Booleschen Algebra $\mathcal{T}$ ist die Menge $\{(0,0),\ (0,1),\ (1,1)\}$. Es gilt also $0 \leq 1$.

**Proposition 4.2.1** *Sei $\mathcal{B}$ eine Boolesche Algebra. Dann ist die Ordnung von $\mathcal{B}$ eine partielle Ordnung für $|\mathcal{B}|$, die für alle $x, y \in |\mathcal{B}|$ die folgenden Eigenschaften erfüllt:*

1. $0$ *ist das kleinste Element von* $|\mathcal{B}|$.
2. $1$ *ist das größte Element von* $|\mathcal{B}|$.
3. $x \wedge y$ *ist das Infimum von* $\{x, y\}$.
4. $x \vee y$ *ist das Supremum von* $\{x, y\}$.

**Beweis** Zunächst zeigen wir, dass die Ordnung von $\mathcal{B}$ eine partielle Ordnung ist. Reflexivität und Antisymmetrie sind leicht zu zeigen. Wir zeigen die Transitivität. Sei $x \leq y$ und $y \leq z$. Dann $x \wedge y = x$ und $y \wedge z = y$. Es genügt zu zeigen, dass $x \wedge z = x$. Das gelingt wie folgt:

$$\begin{aligned} x \wedge z &= x \wedge y \wedge z && \text{da } x \wedge y = x \\ &= x \wedge y && \text{da } y \wedge z = y \\ &= x && \text{da } x \wedge y = x \end{aligned}$$

Eigenschaft (1) folgt mit Proposition 4.1.1. Eigenschaft (2) folgt direkt aus dem Identitätsaxiom für 1.

Wir zeigen, dass $x \wedge y$ das Infimum von $\{x, y\}$ ist. Die Beziehungen $x \wedge y \leq x$ und $x \wedge y \leq y$ folgen direkt aus der Idempotenz der Konjunktion. Sei $z$ eine untere Schranke für $\{x, y\}$. Dann $z \wedge x = z$ und $z \wedge y = z$. Wir müssen $z \leq x \wedge y$ zeigen. Dafür genügt die Gleichung $z \wedge (x \wedge y) = z$, die direkt aus den Annahmen folgt.

Behauptung (4) folgt analog. □

**Proposition 4.2.2** *Für jede Boolesche Algebra $\mathcal{B}$ und für alle $x, y \in |\mathcal{B}|$ sind die folgenden Aussagen paarweise äquivalent:*

$$\begin{array}{cccc} x \leq y & x \cdot y = x & x \cdot \overline{y} = 0 & x - y = 0 \\ \overline{y} \leq \overline{x} & x + y = y & \overline{x} + y = 1 & x \Rightarrow y = 1 \end{array}$$

**Beweis** Wir nummerieren die Aussagen von links nach rechts, erste Zeile vor zweiter Zeile, durch.

„(1) $\iff$ (2)“. Gilt nach Definition von $\leq$.

„(3) $\iff$ (4)“. Gilt nach Definition von $-$.

„(7) $\iff$ (8)“. Gilt nach Definition von $\Rightarrow$.

„(2) $\Rightarrow$ (3)“. Sei $x \cdot y = x$. Dann $x \cdot \overline{y} = x \cdot y \cdot \overline{y} = x \cdot 0 = 0$.

„(3) $\Rightarrow$ (5)“. Sei $x \cdot \overline{y} = 0$. Dann $\overline{y} \cdot \overline{x} = \overline{y} \cdot \overline{x} + 0 = \overline{y} \cdot \overline{x} + x \cdot \overline{y} = (\overline{x} + x)\overline{y} = 1 \cdot \overline{y} = \overline{y}$. Also $\overline{y} \leq \overline{x}$.

„(5) $\Rightarrow$ (6)“. Sei $\overline{y} \leq \overline{x}$. Dann $\overline{y} \cdot \overline{x} = \overline{y}$. Also $x + y = \overline{\overline{x + y}} = \overline{\overline{x} \cdot \overline{y}} = \overline{\overline{y}} = y$.

„(6) $\Rightarrow$ (7)“. Sei $x + y = y$. Dann $\overline{x} + y = \overline{x} + x + y = 1 + y = 1$.

„(7) $\Rightarrow$ (2)“. Sei $\overline{x} + y = 1$. Dann $x \cdot y = 0 + x \cdot y = x \cdot \overline{x} + x \cdot y = x(\overline{x} + y) = x \cdot 1 = x$. □

Die nächste Proposition stellt fest, dass man in jeder Booleschen Algebra Ungleichungen des Typs $a \leq a'$ und Gleichungen des Typs $a = a'$ auf Gleichungen des Typs $a = 1$ zurückführen kann.

$$
\begin{array}{lcll}
X, Y & \in & \mathit{Var} & \mathit{Variable} \\
A, B & \in & \mathit{For} \ = & \mathit{Formel} \\
 & & \quad 0 & \quad \mathit{Null} \\
 & & \mid \ 1 & \quad \mathit{Eins} \\
 & & \mid \ X & \quad \mathit{Atom} \\
 & & \mid \ \neg A & \quad \mathit{Negation} \\
 & & \mid \ A_1 \wedge A_2 & \quad \mathit{Konjunktion} \\
 & & \mid \ A_1 \vee A_2 & \quad \mathit{Disjunktion} \\
 & & \mid \ A_1 - A_2 & \quad \mathit{Differenz} \\
 & & \mid \ A_1 \Rightarrow A_2 & \quad \mathit{Implikation} \\
 & & \mid \ A_1 \Leftrightarrow A_2 & \quad \mathit{\ddot{A}quivalenz} \\
 & & \mid \ (A_1, A_2, A_3) & \quad \mathit{Konditional}
\end{array}
$$

Abbildung 4.2: Boolesche Formeln

## 4.3 Boolesche Formeln

Unser Ziel ist die Entwicklung von Prozeduren, die zu einer vorgelegten Boolschen Gleichung entscheiden, ob sie gültig ist. Um eine tragfähige Grundlage für die Entwicklung und Analyes solcher Prozeduren zu haben, formalisieren wir Boolesche Ausdrücke als mathematische Objekte. Wir gehen dabei so vor, wie wir es in Kapitel 3 am Beispiel von arithmetischen Ausdrücken gelernt haben.

Sei also eine Menge *Var* gegeben, deren Elemente als Variablen dienen sollen. Darauf aufbauend definieren wir wie in Abbildung 4.2 gezeigt eine Menge *For* von *Formeln*. Die Formeln entsprechen Booleschen Ausdrücken, wobei wir auch für die abgeleiteten Notationen für Differenz, Implikation, Äquivalenz und Expansion explizite Darstellungen vorgesehen haben.

Die Funktionen

$$\mathit{VV} \in \mathit{For} \to \mathcal{P}_{\mathit{fin}}(\mathit{Var}) \qquad \mathit{vorkommende\ Variablen}$$

$$\_[\_/\_] \in \mathit{For} \times \mathit{For} \times \mathit{Var} \to \mathit{For} \qquad \mathit{Substitution}$$

werden so wie man es erwartet durch strukturelle Rekursion definiert. Da diese Definitionen völlig offensichtlich sind, verzichten wir auf ihre Wiedergabe.

Für jede Boolesche Algebra $\mathcal{B} = (|\mathcal{B}|, 0^{\mathcal{B}}, 1^{\mathcal{B}}, \wedge^{\mathcal{B}}, \vee^{\mathcal{B}}, \neg^{\mathcal{B}})$ definieren wir eine *Denotationsfunktion*

$$\mathcal{B} \in \mathit{For} \to (\mathit{Var} \to |\mathcal{B}|) \to |\mathcal{B}|$$

$$\mathcal{B} \in For \to (Var \to |\mathcal{B}|) \to |\mathcal{B}|$$

$$\begin{aligned}
\mathcal{B}(0)\sigma &= 0^{\mathcal{B}} \\
\mathcal{B}(1)\sigma &= 1^{\mathcal{B}} \\
\mathcal{B}(X)\sigma &= \sigma X \\
\mathcal{B}(\neg A)\sigma &= \neg^{\mathcal{B}}(\mathcal{B}(A)\sigma) \\
\mathcal{B}(A_1 \vee A_2)\sigma &= \mathcal{B}(A_1)\sigma \vee^{\mathcal{B}} \mathcal{B}(A_2)\sigma \\
\mathcal{B}(A_1 - A_2)\sigma &= \mathcal{B}(A_1)\sigma -^{\mathcal{B}} \mathcal{B}(A_2)\sigma \\
\mathcal{B}(A_1 \Rightarrow A_2)\sigma &= \mathcal{B}(A_1)\sigma \Rightarrow^{\mathcal{B}} \mathcal{B}(A_2)\sigma \\
\mathcal{B}(A_1 \Leftrightarrow A_2)\sigma &= \mathcal{B}(A_1)\sigma \Leftrightarrow^{\mathcal{B}} \mathcal{B}(A_2)\sigma \\
\mathcal{B}(A_1, A_2, A_3)\sigma &= (\mathcal{B}(A_1)\sigma, \mathcal{B}(A_2)\sigma, \mathcal{B}(A_3)\sigma)^{\mathcal{B}}
\end{aligned}$$

Abbildung 4.3: Denotationsfunktion einer Boolesche Algebra $\mathcal{B}$

durch strukturelle Rekursion wie in Abbildung 4.3 gezeigt. Dabei benutzen wir für die Denotationsfunktion dasselbe Symbol wie für die Algebra (also $\mathcal{B}$). Die Denotationsfunktion $\mathcal{B}$ formalisiert die Denotation von Booleschen Ausdrücken in der Algebra $\mathcal{B}$. Wieder folgen wir der Methode, die wir in Kapitel 3 am Beispiel von arithmetischen Ausdrücken kennengelernt haben. Die Funktionen $Var \to |\mathcal{B}|$ bezeichnen wir wieder als *Belegungen*.

Der Methode aus Kapitel 3 folgend könnten wir jetzt für jede Boolesche Algebra eine Äquivalenzrelation für Formeln definieren. Da wir jedoch in erster Linie an der Gültigkeit von Gleichungen in *allen* Booleschen Algebren interessiert sind, definieren wir stattdessen eine stärkere binäre Relation $\models\!\!\!\dashv$ wie folgt:

$$A \models\!\!\!\dashv B \quad \overset{\text{def}}{\iff} \quad \forall \text{ Boolesche Algebra } \mathcal{B}\colon\ \mathcal{B}(A) = \mathcal{B}(B)$$

Die Aussage $A \models\!\!\!\dashv B$ gilt also genau dann, wenn die Gleichung „$A = B$“ in jeder Boolesche Algebra gültig ist. Offensichtlich ist $\models\!\!\!\dashv$ eine Äquivalenzrelation auf *For*. Wir nennen zwei Formeln $A$ und $B$ *äquivalent*, wenn $A \models\!\!\!\dashv B$ gilt.

Auch die signifikanten Variablen einer Formel definieren wir in Bezug auf alle Booleschen Algebren. Für jede Formel $A \in For$ definieren wir die Menge der *signifikanten Variablen von A* wie folgt:

$$SV(A) \overset{\text{def}}{=} \{\, X \in Var \mid \forall B \in For\colon\ A \models\!\!\!\dashv B \Rightarrow X \in VV(B) \,\}$$

Die signifikanten Variablen von $A$ sind also genau die Variablen, die in allen zu $A$ äquivalenten Formeln vorkommen. Offensichtlich gilt $SV(A) \subseteq VV(A)$.

Damit ist die Formalisierung der Syntax und Semantik von Booleschen Ausdrücken abgeschlossen. Da wir der Methode aus Kapitel 3 weitgehend gefolgt sind, gelten wieder entsprechende Propositionen zu Substitution, Kongruenz und Variablenunabhängigkeit. Die Beweise sind analog zu denen in Kapitel 3.

**Lemma 4.3.1 (Substitution)** *Seien $A, B \in For$ und $X \in Var$. Sei $\mathcal{B}$ eine Boolesche Algebra und $\sigma \in Var \to |\mathcal{B}|$. Dann:*

$$\mathcal{B}(A[B/x])\sigma = \mathcal{B}(A)(\sigma[(\mathcal{B}(B)\sigma)/X])$$

**Proposition 4.3.2 (Kongruenz)** *Seien $A_1, A_2, B_1, B_2 \in For$ und $X \in Var$. Dann:*

$$A_1 \models\!\!\mid A_2 \ \wedge\ B_1 \models\!\!\mid B_2 \quad\Rightarrow\quad A_1[B_1/X] \models\!\!\mid A_2[B_2/X]$$

**Proposition 4.3.3 (Variablenunabhängigkeit)** *Sei $A \in For$ eine Formel und $\mathcal{B}$ eine Boolesche Algebra. Seien $\sigma, \sigma' \in Var \to |\mathcal{B}|$ Belegungen, die auf $VV(A)$ übereinstimmen. Dann $\mathcal{B}(A)\sigma = \mathcal{B}(A)\sigma'$.*

**Proposition 4.3.4 (Variablenunabhängigkeit)** *Sei $Var' \subseteq Var$. Dann:*

1. $For_{Var'} \subseteq For_{Var}$.
2. *Für jede Boolesche Algebra $\mathcal{B}$ gilt:*

$$\forall A, B \in For_{Var'}: \quad \mathcal{B}_{Var'}(A) = \mathcal{B}_{Var'}(B) \quad\Longleftrightarrow\quad \mathcal{B}_{Var}(A) = \mathcal{B}_{Var}(B)$$

3. $\forall A, B \in For_{Var'}: \quad A \models\!\!\mid_{Var'} B \quad\Longleftrightarrow\quad A \models\!\!\mid_{Var} B$.

### Einige Begriffe

Abschließend definieren wir noch eine Reihe von Begriffen, die im Zusammenhang mit Aussagenlogik gebräuchlich sind. Wir werden jedoch auf den Gebrauch dieser Begriffe verzichten, da sie nicht wirklich essentiell sind.

Seien $A$ und $B$ Formeln. Wir definieren:

- *A gültig*, wenn $A \models\!\!\mid 1$.
- *A unerfüllbar*, wenn $A \models\!\!\mid 0$.
- *A erfüllbar*, wenn nicht $A \models\!\!\mid 0$.
- *A impliziert B*, wenn $A \wedge B \models\!\!\mid A$.

## 4.4 Boolesche Funktionen und Boolesche Mengen

Für die praktischen Anwendungen der Aussagenlogik genügt die zweiwertige Boolesche Algebra $\mathcal{T}$. Es ist ohne weiteres möglich, auf das abstrakte Konzept der Booleschen Algebra zu verzichten und die erforderlichen computationalen Methoden nur in Hinblick auf die zweiwertige Boolesche Algebra zu entwickeln. Damit verschenkt man allerdings wichtige theoretische Einsichten. Da man zudem keine Arbeit spart, werden wir weiter der abstrakten Entwicklungslinie folgen. Bevor wir damit fortfahren, wollen wir jedoch die für praktische Anwendungen relevanten Sichtweisen formulieren. Dabei stellen wir uns absichtlich naiv und versuchen zu vergessen, was wir bereits über Boolesche Algebren wissen.

Wir nehmen an, dass eine Menge *Var* gegeben ist. Die Elemente von *Var* nennen wir *Variablen* und bezeichnen sie mit $X$. Was die hier zu entwickelnde konkrete Sichtweise angeht, könnte man Variablen treffender als *Mermale* bezeichnen.

Eine *Boolesche Belegung* ist eine Funktion $\mathit{Var} \to \mathbb{B}$, die jeder Variablen einen Wert in $\mathbb{B}$ zuordnet. Die Menge aller Booleschen Belegungen bezeichnen wir mit $\Sigma$, und einzelne Boolesche Belegungen mit $\sigma$:

$$\sigma \in \Sigma \stackrel{\text{def}}{=} \mathit{Var} \to \mathbb{B} \qquad \textit{Boolesche Belegungen}$$

Wir können uns Belegungen als eine Variante von Tupeln vorstellen, bei der die Konstituenten durch Variablen identifiziert werden. Betrachten Sie dazu das Tupel

$$(1, 0, 0)$$

Dieses Tupel können wir auch als eine Funktion

$$\{1 \mapsto 1,\ 2 \mapsto 0,\ 3 \mapsto 0\}$$

des Typs $\{1, 2, 3\} \to \mathbb{B}$ darstellen. Dabei agieren die Zahlen 1, 2, 3 als Namen für die Konstituenten des Tupels. Wenn wir $\mathit{Var} = \{1, 2, 3\}$ setzen, entsprechen die Tupel in $\mathbb{B}^3$ genau den Belegungen in $\Sigma$. Dabei agieren die Variablen als Namen für die Konstituenten der Tupel.

**Proposition 4.4.1** *Sei Var eine $n$-elementige Menge ($n \geq 0$). Dann $\Sigma \cong \mathbb{B}^n$.*

**Beweis** Sei $X_1 < \cdots < X_n$ eine totale Ordnung für *Var*. Dann definiert $f(\sigma) = (\sigma X_1, \ldots, \sigma X_n)$ eine Bijektion $\Sigma \to \mathbb{B}^n$. $\square$

Belegungen sind flexibler als Tupel, da man die Namen für die Konstituenten durch die Vorgabe der Variablen frei wählen kann.[3] Zudem erhalten wir durch die

[3] Diese Idee findet man auch in Programmiersprachen. Zum Beispiel in der Form von Records in Pascal und Standard ML, und in der Form von Structures in C.

Vorgabe von unendlich vielen Variablen Belegungen mit unendlich viele Konstituenten.

Man kann Variablen als Merkmale und Belegungen als Zustände bezeichnen. Ein Zustand legt für jedes Merkmal einen Wert fest.

Wir nehmen jetzt an, dass die Menge *For* der Booleschen Formeln und die Denotationsfunktion

$$\mathcal{T} \in For \to \Sigma \to \mathbb{B}$$

für die zweiwertige Boolesche Algebra wie im letzten Abschnitt definiert sind. Booleschen Formeln fassen wir als Beschreibungen für Funktionen des Typs $\Sigma \to \mathbb{B}$ auf. Die Denotationsfunktion $\mathcal{T}$ liefert zu jeder Formel die durch sie beschriebene Funktion. Es stellt sich die Frage, ob wir jede Funktion des Typs $\Sigma \to \mathbb{B}$ mit einer Booleschen Formel beschreiben können. Wir werden zeigen, dass dies genau dann der Fall ist, wenn die Menge der Variablen (*Var*) endlich ist (was bei praktischen Anwendungen immer der Fall ist). Die Funktionen, die wir mit Booleschen Formeln beschreiben können, bezeichnen wir als *Boolesche Funktionen*.

Für die Hardware von Computern verwendet man Schaltkreise, die Boolesche Funktionen berechnen. Solche Schaltkreise werden heute mithilfe von CAD-Systemen entwickelt. Dabei verwendet man Boolesche Formeln sowohl zur Beschreibung der zu berechnenden Funktion (Spezifikation) als auch zur Beschreibung des berechnenden Schaltkreises (Implementierung).[4] Um zu verifizieren, dass ein Schaltkreis die spezifizierte Funktion berechnet, benötigt man ein Verfahren, das entscheidet, ob zwei Boolesche Formeln dieselbe Funktion beschreiben. Wir werden solche Verfahren entwickeln.

Wir sagen, dass eine Boolesche Belegung $\sigma$ eine Boolesche Formel $A$ *erfüllt*, wenn $\mathcal{T}(A)\sigma = 1$ gilt.

### Boolesche Mengen

Sei $M$ eine Menge. Da $\mathbb{B}$ genau zwei Elemente hat, läßt sich jede Funktion $f \in M \to \mathbb{B}$ durch die Menge aller $x \in M$ darstellen, für die $f(x) = 1$ gilt. Umgekehrt lässt sich jede Teilmenge $N$ von $M$ durch die Funktion $M \to \mathbb{B}$ darstellen, die genau für die Elemente von $N$ das Ergebnis 1 liefert (die sogenannte *charakteristische Funktion von N*).

**Proposition 4.4.2** *Sei Var eine n-elementige Menge* ($n \geq 0$). *Dann:*

$$\mathcal{P}(\mathbb{B}^n) \;\cong\; \mathcal{P}(\Sigma) \;\cong\; \Sigma \to \mathbb{B}$$

[4] Für die Beschreibung von Schaltkreisen benötigt man Boolesche Formeln mit let $X$=$A$ in $B$.

**Beweis** Die erste Isomorphie folgt aus Proposition 4.4.1, die zweite haben wir gerade argumentiert. □

Wir können Boolesche Formeln also auch als Beschreibungen von Teilmengen von $\mathbb{B}^n$ auffassen. Diese Sicht eröffnet viele interessante Anwendungen, da jede endliche Menge nach Binärcodierung als eine Teilmenge von $\mathbb{B}^n$ aufgefasst werden kann (für hinreichend großes $n$). Für große endliche Menge erweist sich die aussagenlogische Darstellung als außerordentlich effizient. Die dafür benötigten Techniken werden wir später kennenlernen (sogenannte OBDD-Darstellung).

Wir nehmen jetzt die Sicht ein, dass Boolesche Formeln Mengen von Booleschen Belegungen beschreiben. Wie wir gleich sehen werden, ist die damit verbundene Interpretation von Booleschen Formeln besonders leicht zu verstehen.

Wir definieren die Funktion

$$\begin{aligned} &\mathcal{M} \in \mathit{For} \to \mathcal{P}(\Sigma) \\ &\mathcal{M}(A) = \{\, \sigma \in \Sigma \mid \mathcal{T}(A)\sigma = 1 \,\} \end{aligned}$$

Diese liefert zu jeder Booleschen Formel die Menge aller Booleschen Belegungen, die die Formel erfüllen.

Eine *Boolesche Menge* ist eine Menge $D$, für die eine Boolesche Formel $A$ mit $D = \mathcal{M}(A)$ existiert. Die Menge aller Booleschen Mengen bezeichnen wir mit

$$BS \stackrel{\text{def}}{=} \{\, \mathcal{M}(A) \mid A \in \mathit{For} \,\}$$

Machen Sie sich klar, dass die Booleschen Funktionen gerade die charakteristischen Funktionen der Booleschen Mengen sind.

Für $X \in \mathit{Var}$ und $b \in \mathbb{B}$ definieren wir die *Literalmenge* $D_{X,b}$ wie folgt:

$$D_{X,b} \stackrel{\text{def}}{=} \{\, \sigma \in \Sigma \mid \sigma X = b \,\}$$

Außerdem verwenden wir für Teilmengen $M \subseteq \Sigma$ die *Komplementnotation*:

$$-M \stackrel{\text{def}}{=} \Sigma - M$$

Offensichtlich gilt für alle $X \in \mathit{Var}$: $D_{X,1} = -D_{X,0}$ und $D_{X,0} = -D_{X,1}$. Außerdem gilt für alle $\sigma \in \Sigma$: $\{\sigma\} = \bigcap_{X \in \mathit{Var}} D_{X,\sigma X}$.

**Proposition 4.4.3** *Für alle $A, B \in For$ und $X \in Var$:*

$$
\begin{aligned}
\mathcal{M}(0) &= \emptyset \\
\mathcal{M}(1) &= \Sigma \\
\mathcal{M}(X) &= D_{X,1} \\
\mathcal{M}(\neg A) &= -\mathcal{M}(A) \\
\mathcal{M}(A \wedge B) &= \mathcal{M}(A) \cap \mathcal{M}(B) \\
\mathcal{M}(A \vee B) &= \mathcal{M}(A) \cup \mathcal{M}(B) \\
\mathcal{M}(A - B) &= \mathcal{M}(A) - \mathcal{M}(B) \\
\mathcal{M}(A \Rightarrow B) &= -\mathcal{M}(A) \cup \mathcal{M}(B)
\end{aligned}
$$

Wir machen also die interessante Entdeckung, dass $\mathcal{M}$ die Booleschen Operationen durch die entsprechenden Mengenoperationen interpretiert. Die Variablen spielen jetzt die Rolle von Konstanten, die die entsprechenden Literalmengen denotieren.

Die Gleichungen aus Proposition 4.4.3 zeigen, dass wir $\mathcal{M}$ auch unabhängig von $\mathcal{T}$ definieren können.

Obwohl mit der Funktionsinterpretation $\mathcal{T}$ und der Mengeninterpretation $\mathcal{M}$ von Booleschen Formeln recht unterschiedliche Intuitionen verbunden sind, sind sie technisch gesehen völlig isomorph. Insbesondere haben wir:

**Proposition 4.4.4** $\forall A, B \in For\colon\ \mathcal{M}(A) = \mathcal{M}(B) \iff \mathcal{T}(A) = \mathcal{T}(B)$.

**Proposition 4.4.5** *Die Menge BS der Booleschen Mengen enthält $\emptyset$, $\Sigma$ und die Literalmengen. Außerdem ist BS unter Komplementen, Durchschnitten, Vereinigungen und Differenzen abgeschlossen.*

**Beweis** Folgt aus Proposition 4.4.3. □

Für praktische Anwendungen kommt man immer mit endlich vielen Variablen aus. In diesem Fall kann jede Teilmenge von $\Sigma$ durch eine Boolesche Formel beschrieben werden.

**Proposition 4.4.6** *Wenn Var endlich ist, dann $\{\, \mathcal{M}(A) \mid A \in For \,\} = \mathcal{P}(\Sigma)$.*

**Beweis** Sei $M \subseteq \Sigma$. Dann

$$M \;=\; \bigcup_{\sigma \in M} \{\sigma\} \;=\; \bigcup_{\sigma \in M} \left( \bigcap_{X \in Var} D_{X,\sigma X} \right)$$

Sei *Var* endlich. Dann ist $\Sigma = Var \to \mathbb{B}$ endlich. Also ist $M$ endlich. Die obige Gleichung stellt $M$ als die Vereinigung von endlich vielen Durchschnitten von

endlich vielen Literalmengen dar. Also folgt mit Proposition 4.4.5, dass $M \in BS$.
□

**Proposition 4.4.7** *Wenn Var endlich ist, dann* $\{\, \mathcal{T}(A) \mid A \in For \,\} = \Sigma \to \mathbb{B}$.

**Beweis** Folgt aus Proposition 4.4.6. □

## 4.5 Modellierung mit Booleschen Mengen

George Boole (Laws of Thought, 1854) suchte nach einem mathematischen System, das die Kombination von logischen Aussagen (im philosophischen Sinne) beschreibt. Er betrachtete Boolesche Ausdrücke und interpretierte sie im obigen Sinne als Beschreibungen von Mengen. Allerdings gab es die kleine Schwierigkeit, dass damals der Begriff der Menge als mathematisches Objekt noch gar nicht erfunden war. Deswegen war Boole gezwungen, Boolesche Ausdrücke als abstrakte Ausdrücke zu betrachten und die für sie geltenden Gleichungen durch Axiome zu beschreiben. Das war revolutionär, da Mathematiker bis dahin nur mit arithmetischen Ausdrücken gerarbeitet hatten. Aus heutiger Sicht hat Boole also mit dem abstrakten Begriff der Booleschen Algebra gearbeitet, und dafür die heute allgemein verwendete axiomatische algebraische Methode erfunden.[5]

Am Beispiel einer sogenannten Denksportaufgabe zeigen wir, wie man logische Aussagen mit Booleschen Ausdrücken beschreiben kann.

> „Worin besteht das Geheimnis Ihres langen Lebens?“ wird ein Hundertjähriger gefragt. „Ich halte mich streng an drei Diätregeln: Wenn ich kein Bier zu einer Mahlzeit trinke, dann esse ich immer Fisch. Immer wenn ich sowohl Fisch und Bier zur selben Mahlzeit esse, verzichte ich auf Eiscreme. Wenn ich Eiscreme esse oder kein Bier trinke, dann esse ich keinen Fisch.“ Können Sie diese Regeln vereinfachen?

Zunächst gehen wir davon aus, dass Mahlzeiten hinreichend genau durch Belegungen beschrieben werden können. Dafür verwenden wir drei Variablen $B$, $E$ und $F$ mit den folgenden Bedeutungen:

- $B = 1$ genau dann, wenn zu der Mahlzeit Bier getrunken wird.
- $E = 1$ genau dann, wenn zu der Mahlzeit Eiscreme gegessen wird.

[5] Sie sollten unbedingt einen Blick in George Booles faszinierendes Buch „An Investigation of the Laws of Thought“ (1854) werfen (in Bibliotheken finden Sie Nachdrucke). Danach werden Sie Mathematik mit anderen Augen sehen.

- $F = 1$ genau dann, wenn zu der Mahlzeit Fisch gegessen wird.

Die Beschreibung von Mahlzeiten durch Belegungen stellt eine radikale Abstraktion dar. Wenn wir uns auf die Variablen $B$, $E$ und $F$ beschränken, werden nur die Aspekte einer Mahlzeit dargestellt, die für die Regeln relevant sind. Insgesamt können damit nur $2^3 = 8$ verschiedene Typen von Mahlzeiten dargestellt werden.

Jede Diätregel fassen wir als die Beschreibung der Menge von Mahlzeiten auf, die die Regel erfüllen. Technisch bedeutet das, dass wir jede Diätregel als einen Booleschen Ausdruck auffassen, der eine Menge von Belegungen beschreibt (gemäß der Funktion $\mathcal{M}$). Damit entsprechen die Diätregeln den folgenden Booleschen Ausdrücken:

$$
\begin{aligned}
R_1 &\stackrel{\text{def}}{=} \neg B \Rightarrow F \\
R_2 &\stackrel{\text{def}}{=} F \wedge B \Rightarrow \neg E \\
R_3 &\stackrel{\text{def}}{=} E \vee \neg B \Rightarrow \neg F
\end{aligned}
$$

Bei den Symbolen $R_1$, $R_2$ und $R_3$ handelt es sich nicht um Variablen, sondern um Namen für Ausdrücke. Machen Sie sich klar, dass die durch $\mathcal{M}$ formulierte Interpretation von implikativen Booleschen Ausdrücken

$$\mathcal{M}(A_1 \Rightarrow A_2) \;=\; -\mathcal{M}(A_1) \cup \mathcal{M}(A_2)$$

tatsächlich dem natürlichsprachlichem „wenn $A_1$, dann $A_2$“ entspricht.

Der Ausdruck

$$D \stackrel{\text{def}}{=} R_1 \wedge R_2 \wedge R_3$$

beschreibt die Menge aller Belegungen, die Mahlzeiten beschreiben, die alle drei Regeln erfüllen.

Man überzeugt sich leicht davon, dass die Belegung

$$\{B \mapsto 1,\; F \mapsto 0,\; E \mapsto 1\}$$

den Ausdruck $D$ erfüllt. Das bedeutet, das eine Mahlzeit, bei der Bier getrunken, kein Fisch gegessen, und Eiscreme genossen wird, die Diätregeln erfüllt.

Die Denksportaufgabe verlangt nach einer vereinfachten Darstellung der Diätregeln. Auf unser Modell übertragen bedeutet dies, einen möglichst einfachen Booleschen Ausdruck zu bestimmen, der dieselbe Belegungsmenge wie $D$ beschreibt. Man kann zeigen, dass in allen Booleschen Algebren

$$D = B \wedge (\neg F \vee \neg E)$$

gilt. Wegen Proposition 4.4.4 bedeutet dies, dass der Ausdruck $B \wedge (\neg F \vee \neg E)$ genau dieselbe Belegungsmenge wie $D$ beschreibt. Also kann man die Diätregeln des Hundertjährigen auch wie folgt formulieren: „Trinke zu jeder Mahlzeit Bier und esse zu keiner Mahlzeit sowohl Fisch und Eiscreme“.

Wir werden Algorithmen entwickeln, die zu $D$ den oben angegebenen vereinfachten Ausdruck berechnen können. In diesem Zusammenhang werden wir auch klären, was unter einer „einfachsten Darstellung“ eines Ausdrucks zu verstehen ist.

## 4.6 Unendlich viele Variablen

Einige theoretische Anwendungen der Aussagenlogik benötigen unendlich viele Variablen.[6] In diesem Fall gibt es Teilmengen von $\Sigma = \mathit{Var} \to \mathbb{B}$, die nicht durch Boolesche Formeln beschrieben werden können. Betrachten Sie dazu die Menge $M_0$, die alle Belegungen enthält, die für unendlich viele Variablen 0 liefern. Da jede Formel immer nur eine Aussage über die endlich vielen in ihr enthaltenen Variablen machen kann (Proposition 4.3.3), ist klar, dass $M_0$ durch keine Formel beschrieben werden kann.

Seien $V \subseteq \mathit{Var}$, $\sigma, \sigma' \in \Sigma$, und $M \subseteq \Sigma$. Wir definieren:

$$\begin{aligned}
\sigma =_V \sigma' \quad &\overset{\text{def}}{\iff} \quad \forall X \in V\colon\ \sigma(X) = \sigma'(X) \\
V \textit{ entscheidet } M \quad &\overset{\text{def}}{\iff} \quad \forall \sigma \in \Sigma\colon\ (\sigma \in M \iff \exists \sigma' \in M\colon\ \sigma =_V \sigma') \\
M \textit{ endlich entscheidbar} \quad &\overset{\text{def}}{\iff} \quad \text{Es gibt eine endliche Menge } V \subseteq \mathit{Var}, \\
& \qquad\ \ \text{die } M \text{ entscheidet}
\end{aligned}$$

Wegen Proposition 4.3.3 ist jede Boolesche Menge endlich entscheidbar. Wir werden zeigen, dass auch umgekehrt jede endlich entscheidbare Menge eine Boolesche Menge ist. Das bedeutet, dass das Konzept der Booleschen Menge auch ohne das Konzept der Booleschen Formel charakterisiert werden kann.

**Proposition 4.6.1** *Sei $V \subseteq \mathit{Var}$ und $M \subseteq \Sigma$. Dann gilt:*

$$V \textit{ entscheidet } M \quad \iff \quad \forall \sigma \in M\ \forall \sigma' \in \Sigma\colon\ \sigma =_V \sigma' \ \Rightarrow\ \sigma' \in M$$

**Proposition 4.6.2** *Seien $M, M' \subseteq \Sigma$ und $V \subseteq \mathit{Var}$. Dann:*

[6] Beispielsweise benötigt man für den Beweis des Herbrandschen Satzes der Prädikatenlogik die im Folgenden bewiesene Kompaktheitseigenschaft für aussagenlogische Formeln mit unendlich vielen Variablen.

1. *$V$ entscheidet $M$* $\iff$ *$V$ entscheidet $-M$.*
2. *$V$ entscheidet $M$ und $M'$* $\Rightarrow$ *$V$ entscheidet $M \cap M'$ und $M \cup M'$.*

**Beweis** Wir beweisen die erste Behauptung. Da $-(-M) = M$, genügt es, die Richtung von links nach rechts zu zeigen. Dazu nehmen wir an, dass $V$ die Menge $M$ entscheidet. Weiter wählen wir $\sigma \in -M$ und $\sigma' \in \Sigma$ mit $\sigma =_V \sigma'$. Wir zeigen durch Widerspruch, dass $\sigma' \in -M$.

Sei $\sigma' \notin -M$. Dann $\sigma \in M$. Da $V$ die Menge $M$ entscheidet und $\sigma =_V \sigma'$, folgt $\sigma' \in M$. Also $\sigma' \notin -M$. Widerspruch.

Die zweite Behauptung folgt mit ähnlichen Argumenten. □

**Proposition 4.6.3 (Abschlusseigenschaften)** *Die Mengen $\emptyset$ und $\Sigma$ sind endlich entscheidbar. Auch die Literalmengen $D_{X,b}$ sind endlich entscheidbar. Wenn $D$ und $D'$ endlich entscheidbar sind, dann sind $-D$, $D \cap D'$ und $D \cup D'$ endlich entscheidbar.*

**Beweis** Die erste und zweite Behauptung sind offensichtlich. Die restlichen Behauptungen folgen mit Proposition 4.6.2. □

Sei $M \subseteq \Sigma$ und $V \subseteq Var$ eine nichtleere Variablenmenge.[7] Wir definieren die *$V$-Projektion von $M$* wie folgt:

$$Pro(M, V) \stackrel{\text{def}}{=} \{ s \in V \to \mathbb{B} \mid \exists \sigma \in M\colon\ s \subseteq \sigma \}$$

Wenn $V$ eine endliche Menge ist, dann ist auch $Pro(M, V)$ eine endliche Menge.

**Proposition 4.6.4** *Sei $M \subseteq \Sigma$ und $\emptyset \neq V \subseteq Var$. Dann:*

$$V \text{ entscheidet } M \iff M = \{ \sigma \in \Sigma \mid \exists s \in Pro(M, V)\colon\ s \subseteq \sigma \}$$

**Satz 4.6.5 (Boolesche Mengen)** *Eine Menge $M \subseteq \Sigma$ ist genau dann eine Boolesche Menge, wenn sie endlich entscheidbar ist.*

**Beweis** Wegen Proposition 4.3.3 ist jede Boolesche Menge endlich entscheidbar.

Sei $M \subseteq \Sigma$ eine Menge, die durch eine endliche Menge $V \subseteq Var$ entschieden wird. Dann folgt mit Proposition 4.6.4:

$$M = \bigcup_{s \in Pro(M,V)} \{ \sigma \in \Sigma \mid s \subseteq \sigma \} = \bigcup_{s \in Pro(M,V)} \left( \bigcap_{X \in V} D_{X,s(X)} \right)$$

[7] Projektionen auf die leere Variablenmenge funktionieren nicht, da $Pro(M, \emptyset) = \emptyset$ für alle $M \subseteq \Sigma$ gelten würde.

Diese Gleichung stellt $M$ als die Vereinigung von endlich vielen Durchschnitten von endlich vielen Literalmengen dar. Also folgt mit Proposition 4.6.3, dass $M$ endlich entscheidbar ist. $\square$

**Proposition 4.6.6** *Sei $V \subseteq Var$ eine $n$-elementige Menge ($n \geq 0$). Dann gibt es genau $2^{2^n}$ verschiedene Teilmengen von $\Sigma$, die von $V$ entschieden werden.*

**Beweis** Wenn $n = 0$ ist, dann $V = \emptyset$. Man sieht leicht, dass $\emptyset$ und $\Sigma$ die zwei einzigen Mengen sind, die von $\emptyset$ entschieden werden. Also gibt es genau $2^{2^0} = 2$ Mengen, die von $V$ entschieden werden.

Wenn $n > 1$ ist, dann ist die Funktion

$$\begin{aligned} &f \in \{\, M \subseteq \Sigma \mid V \text{ entscheidet } M \,\} \to \mathcal{P}(V \to \mathbb{B}) \\ &f(M) = Pro(M, V) \end{aligned}$$

eine Bijektion. Also werden genau soviele Mengen von $V$ entschieden wie $\mathcal{P}(V \to \mathbb{B})$ Elemente hat. Also gibt es genau $2^{2^n}$ Mengen, die von $V$ entschieden werden. $\square$

Eine Menge $F \subseteq \mathcal{P}(\Sigma)$ von endlich entscheidbaren Mengen heißt *erfüllbar*, wenn ein $\sigma \in \Sigma$ existiert, sodass $\forall M \in F\colon\ \sigma \in M$. Der nächste Satz formuliert eine als Kompaktheit bezeichnete Eigenschaft von endlich entscheidbaren Mengen, die wir später für den Beweis eines wichtigen prädikatenlogischen Satzes (Herbrandscher Satz) benötigen.

**Satz 4.6.7 (Kompaktheit)** *Sei Var eine abzählbare Menge und sei $F$ eine Menge von endlich entscheidbaren Mengen. Dann ist $F$ erfüllbar genau dann, wenn jede endliche Teilmenge von $F$ erfüllbar ist.*

**Beweis** Die Richtung von links nach rechts ist trivial.

Sei jede endliche Teilmenge von $F$ erfüllbar. Wir werden eine Belegung $\sigma$ konstruieren, die in jeder Menge von $F$ enthalten ist.

Sei $X_0, X_1, X_2, \cdots$ eine Aufzählung der Variablen von *Var*. Sei $F_n \subseteq F$ die Menge aller Mengen in $F$, die von der Variablenmenge $\{X_0, \ldots, X_n\}$ entschieden werden. Da alle Mengen in $F$ endlich entscheidbar sind, gilt:

$$F_0 \subseteq F_1 \subseteq F_2 \subseteq \cdots \subseteq F = \bigcup_{n \in \mathbb{N}} F_n$$

Sei $n \in \mathbb{N}$. Da alle Mengen in $F_n$ von $\{X_0, \ldots, X_n\}$ entschieden werden, folgt mit Proposition 4.6.6, dass $F_n$ endlich ist. Also existiert nach Annahme für alle $n \in \mathbb{N}$ eine Belegung $\sigma_n$, die $F_n$ erfüllt.

Wir werden eine Belegung $\sigma$ konstruieren, für die gilt: Für alle $n \in \mathbb{N}$ gibt es unendlich viele $k \in \mathbb{N}$, so dass $\sigma$ mit $\sigma_k$ auf $\{X_0, \ldots, X_n\}$ übereinstimmt.

Bevor wir $\sigma$ konstruieren, zeigen wir, dass $\sigma$ die Menge $F$ erfüllt. Sei $M \in F$. Dann existiert ein $n \in \mathbb{N}$ mit $M \in F_n$. Also existiert ein $k \geq n$, so dass $\sigma$ mit $\sigma_k$ auf $X_0, \ldots, X_n$ übereinstimmt. Da $M \in F_k$ und $\sigma_k$ die Menge $F_k$ erfüllt, gilt $\sigma_k \in M$. Da $M$ von $\{X_0, \ldots, X_n\}$ entschieden wird, gilt $\sigma \in M$. Also erfüllt $\sigma$ die Menge $F$.

Wir konstruieren jetzt $\sigma$. Dazu konstruieren wir rekursiv eine Folge

$$s_0 \subseteq s_1 \subseteq s_2 \subseteq \cdots$$

von Funktionen $\mathit{Var} \overset{\mathit{fin}}{\rightharpoonup} \mathbb{B}$ wie folgt:

1. $\mathit{Dom}\ s_n = \{X_0, \ldots, X_n\}$.
2. Für alle $n \in \mathbb{N}$ existieren unendlich viele $k \in \mathbb{N}$ mit $s_n \subseteq \sigma_k$.

Jetzt ist $\sigma = \bigcup_{n \in \mathbb{N}} s_n$ eine Belegung mit der angekündigten Eigenschaft. □

## 4.7 Dualitätssatz

Wir setzen jetzt unsere Entwicklung der Theorie der Booleschen Algebren fort.

Wir haben bereits erwähnt, dass Boolesche Algebren eine als Dualität bezeichnete Symmetrie zwischen $\wedge$ und $\vee$ einerseits und 1 und 0 andererseits aufweisen. Der sogenannten Dualitätssatz formuliert einen wichtigen Aspekt dieser Symmetrie.

Wir betrachten zunächst nur Formeln, die mit Variablen, 0, 1, $\neg$, $\vee$ und $\wedge$ gebildet sind. Die Menge aller dieser Formeln bezeichnen wir mit $\mathit{For}_1$. Jede Formel lässt sich nach $\mathit{For}_1$ übersetzen. Wir definieren eine Funktion, die zu einer Formel in $\mathit{For}_1$ *die duale Formel* liefert:

$$\begin{aligned}
\widehat{\ } &\in \mathit{For}_1 \to \mathit{For}_1 \\
\widehat{X} &= X \\
\widehat{\neg A} &= \neg \widehat{A} \\
\widehat{0} &= 1 \\
\widehat{1} &= 0 \\
\widehat{A \vee B} &= \widehat{A} \wedge \widehat{B} \\
\widehat{A \wedge B} &= \widehat{A} \vee \widehat{B}
\end{aligned}$$

Offensichtlich ist die Dualisierungsfunktion *selbstinvertierend*, das heißt, $\widehat{\widehat{A}} = A$ für jede Formel $A \in For_1$. Der Dualitätssatz stellt fest, dass eine Boolesche Gleichung genau dann gültig ist, wenn die dazu duale Gleichung gültig ist.

**Satz 4.7.1 (Dualität)** $\forall A, B \in For_1 \colon\; A \models\!\!\mid B \iff \widehat{A} \models\!\!\mid \widehat{B}$.

**Beweis** Da die Dualisierungsfunktion selbstinvertierend ist, genügt es, eine Richtung der Äquivalenz zu beweisen:

$$(*) \quad \forall A, B \in For_1 \colon\; A \models\!\!\mid B \Rightarrow \widehat{A} \models\!\!\mid \widehat{B}$$

Sei $\mathcal{B}$ eine Boolesche Algebra. Wir werden eine Funktion

$$\delta \in ((Var \to |\mathcal{B}|) \to |\mathcal{B}|) \to ((Var \to |\mathcal{B}|) \to |\mathcal{B}|)$$

definieren, die die Dualisierung von Formeln auf der Ebene der Denotationen simuliert:

$$(**) \quad \forall A \in For_1 \colon \mathcal{B}(\widehat{A}) = \delta(\mathcal{B}(A))$$

Offensichtlich folgt aus der Existenz einer solchen Funktion $\delta$

$$\forall A, B \in For_1 \colon\; \mathcal{B}(A) = \mathcal{B}(B) \Rightarrow \mathcal{B}(\widehat{A}) = \mathcal{B}(\widehat{B})$$

und damit $(*)$.

Wir kommen jetzt zur Definition der semantischen Dualisierungsfunktion $\delta$. Da in jeder Booleschen Algebra die Gleichungen

$$\overline{0} = 1, \quad \overline{1} = 0, \quad \overline{x \vee y} = \overline{x} \wedge \overline{y}, \quad \overline{x \wedge y} = \overline{x} \vee \overline{y}$$

gelten, hat die zu einer Formel $A$ duale Formel $\widehat{A}$ dieselbe Denotation wie $\neg A'$, wobei wir $A'$ aus $A$ bilden, indem wir jede Variable $X$ durch $\neg X$ ersetzen. Also definieren wir

$$\delta f \sigma \overset{\text{def}}{=} \neg^{\mathcal{B}}(f(\neg\sigma)) \quad \text{mit} \;\; \neg\sigma \overset{\text{def}}{=} \lambda X \in Var \,\boldsymbol{.}\, \neg^{\mathcal{B}}(\sigma X)$$

und zeigen $(**)$ durch Induktion über $A$.

Wir betrachten den Fall $A = A_1 \wedge A_2$. Sei $\sigma \in Var \to |\mathcal{B}|$. Dann gilt:

$$\begin{aligned}
\mathcal{B}(\widehat{A})\sigma &= \mathcal{B}(\widehat{A_1 \wedge A_2})\sigma \\
&= \mathcal{B}(\widehat{A_1} \vee \widehat{A_2})\sigma && \text{Definition } \widehat{A} \\
&= \mathcal{B}(\widehat{A_1})\sigma \vee^{\mathcal{B}} \mathcal{B}(\widehat{A_2})\sigma && \text{Definition } \mathcal{B} \\
&= \delta(\mathcal{B}(A_1))\sigma \vee^{\mathcal{B}} \delta(\mathcal{B}(A_2))\sigma && \text{Induktionsannahme} \\
&= \neg^{\mathcal{B}}(\mathcal{B}(A_1)(\neg\sigma)) \vee^{\mathcal{B}} \neg^{\mathcal{B}}(\mathcal{B}(A_2)(\neg\sigma)) && \text{Definition } \delta \\
&= \neg^{\mathcal{B}}(\mathcal{B}(A_1)(\neg\sigma) \wedge^{\mathcal{B}} \mathcal{B}(A_2)(\neg\sigma)) && \text{De Morgan} \\
&= \neg^{\mathcal{B}}(\mathcal{B}(A_1 \wedge A_2)(\neg\sigma)) && \text{Definition } \mathcal{B} \\
&= \delta(\mathcal{B}(A_1 \wedge A_2))\sigma && \text{Definition } \delta \\
&= \delta(\mathcal{B}(A))\sigma
\end{aligned}$$

Die anderen Fälle folgen analog. □

Wir versuchen jetzt, Dualisierung auf Formeln mit $-$, $\Rightarrow$, $\Leftrightarrow$ und Konditionalen zu erweitern. Für Differenz und Implikation gelten die Booleschen Gleichungen

$$\overline{x - y} = \overline{y} \Rightarrow \overline{x}, \qquad \overline{x \Rightarrow y} = \overline{y} - \overline{x}$$

Also sind die Operationen $x - y$ und $y \Rightarrow x$ zueinander dual (beachten Sie das Vertauschen der Argumente). Also erweitern wir die Dualisierungsfunktion für Formeln wie folgt:

$$\begin{aligned}
\widehat{A - B} &= \widehat{B} \Rightarrow \widehat{A} \\
\widehat{A \Rightarrow B} &= \widehat{B} - \widehat{A}
\end{aligned}$$

Der Beweis des Dualitätssatzes lässt sich problemlos auf die neuen Formelvarianten erweitern (wegen der obigen Gleichungen).

Für Konditionale gilt die Boolesche Gleichung

$$\overline{(x, y, z)} = (\overline{x}, \overline{z}, \overline{y})$$

Also haben Konditionale Konditionale als duale Form. Also erweitern wir die Dualisierungsfunktion für Formeln wie folgt:

$$(\widehat{A, B, C}) = (\widehat{A}, \widehat{C}, \widehat{B})$$

Damit lässt sich der Beweis des Dualitätssatzes problemlos auf Konditionale erweitern.

 

Für Äquivalenzen existiert keine duale Formelvariante. Trotzdem haben wir die Möglichkeit, die Dualisierungsfunktion und den Dualitätssatz auf Äquivalenzen auszudehnen, indem wir auf die gültige Boolesche Gleichung

$$\overline{x \Leftrightarrow y} = (\overline{x} - \overline{y}) + (\overline{y} - \overline{x})$$

zurückgreifen. Wenn wir die Dualisierungsfunktion entsprechend mit

$$\widehat{A \Leftrightarrow B} = (\widehat{A} - \widehat{B}) + (\widehat{B} - \widehat{A})$$

erweitern, können wir den Beweis des Dualitätssatzes problemlos auf Äquivalenzen erweitern. Allerdings ist die so erweiterte Dualisierungsfunktion nicht mehr selbstinvertierend.

## 4.8 Übersetzung und Vereinfachung

Eine *Übersetzungsfunktion* ist eine Funktion $f \in For \rightharpoonup For$ mit $f \subseteq \models\mid$.

Seien $M, N \subseteq For$ Formelmengen. Wir sagen, dass $M$ *in* $N$ *übersetzbar ist*, wenn es eine Übersetzungsfunktion $M \to N$ gibt.

Eine Formelmenge $K \subseteq For$ heißt *kanonisch*, wenn

$$\forall A, B \in K\colon\ A \neq B \Rightarrow \mathcal{T}(A) \neq \mathcal{T}(B)$$

Dabei bezeichnet $\mathcal{T}(A)$ die Denotation der Formel $A$ in der zweiwertigen Booleschen Algebra $\mathcal{T}$. Offensichtlich ist die Formelmenge $\{0, 1\}$ kanonisch.

**Proposition 4.8.1** *Seien $M, K \subseteq For$ mit $K$ kanonisch. Dann gibt es höchstens eine Übersetzungsfunktion $M \to K$.*

**Beweis** Seien $f, g \in M \to K$ Übersetzungsfunktionen und $A \in For$. Wir müssen zeigen, dass $f(A) = g(A)$ gilt. Da $f(A), g(A) \in K$ und $K$ kanonisch, genügt es zu zeigen, dass $\mathcal{T}(f(A)) = \mathcal{T}(g(A))$. Dies ist der Fall, da $f(A) \models\mid A \models\mid g(A)$ gilt ($f$ und $g$ sind Übersetzungsfunktionen). □

**Proposition 4.8.2 (Kanonische Übersetzungsfunktion)** *Sei $f$ eine Übersetzungsfunktion mit Ran $f$ kanonisch. Dann gilt für alle $A, B \in$ Dom $f$:*

$$A \models\mid B \iff f(A) = f(B) \iff \mathcal{T}(A) = \mathcal{T}(B)$$

**Beweis** Seien $A, B \in Dom\ f$. Da $f$ eine Übersetzungsfunktion ist, gilt $A \models\!\mid f(A)$ und $B \models\!\mid f(B)$. Also gilt

$$f(A) = f(B) \Rightarrow A \models\!\mid B$$

Die Implikation

$$A \models\!\mid B \Rightarrow \mathcal{T}(A) = \mathcal{T}(B)$$

folgt sofort aus der Definition von $\models\!\mid$. Da $Ran\ f$ kanonisch ist, gilt

$$\mathcal{T}(A) = \mathcal{T}(B) \Rightarrow f(A) = f(B)$$

□

Wir werden im nächsten Abschnitt eine kanonische Formelmenge $K$ und eine Übersetzungsfunktion $f \in For \to K$ definieren. Proposition 4.8.2 sagt uns einerseits, dass wir mit der Übersetzungsfunktion $f$ die Äquivalenz von beliebigen Formeln entscheiden können. Andererseits stellt sie fest, dass aus der Existenz von $K$ und $f$ folgt, dass eine Boolesche Gleichung genau dann gültig ist, wenn sie in der zweiwertigen Booleschen Algebra $\mathcal{T}$ gültig ist.

Sei $For_0$ die Menge der variablenfreien Formeln. Da $\{0, 1\}$ kanonisch ist, gibt es höchstens eine Übersetzungsfunktion $For_0 \to \{0, 1\}$ (Proposition 4.8.1). Wir werden zeigen, dass eine solche Übersetzungsfunktion existiert, indem wir sie mithilfe sogenannter *Vereinfachungsregeln* beschreiben. Diese Beschreibung liefert gleichzeitig ein Berechnungsverfahren für die Übersetzungsfunktion.

Abbildung 4.4 zeigt einige Vereinfachungsregeln für Formeln.[8] Mit diesen Regeln können wir Formeln schrittweise vereinfachen. Zum Beispiel:

$$\begin{array}{rlll} ((1, X, 0) \vee Y) \wedge (Z \Rightarrow Z) & \to & (0 \vee Y) \wedge (Z \Rightarrow Z) & \text{da } (1, X, 0) \to 0 \\ & \to & (0 \vee Y) \wedge 1 & \text{da } (Z \Rightarrow Z) \to 1 \\ & \to & 0 \vee Y & \\ & \to & Y & \end{array}$$

Vereinfachung bedeutet also, dass wir einen Teilformel der zu vereinfachenden Formel gemäß einer der Regeln ersetzen. Die zwei ersten Schritte des Beispiels ersetzen jeweils eine echte Teilformel.

**Proposition 4.8.3** *Seien $A$ und $B$ Formeln, sodass $B$ aus $A$ durch Anwendung einer der Regeln in Abbildung 4.4 erhalten werden kann. Dann $A \models\!\mid B$, $VV(B) \subseteq VV(A)$, und $B$ ist echt kleiner als $A$.*

[8] Die gezeigte Auswahl ist etwas willkürlich. Unter anderem haben wir auf Regeln wie $A \wedge \neg A \to 0$ verzichtet.

| | | | | |
|---|---|---|---|---|
| $\neg 0 \to 1$ | $\neg 1 \to 0$ | $\neg\neg A \to A$ | | |
| $0 \wedge A \to 0$ | $1 \wedge A \to A$ | $A \wedge 0 \to 0$ | $A \wedge 1 \to A$ | $A \wedge A \to A$ |
| $0 \vee A \to A$ | $1 \vee A \to 1$ | $A \vee 0 \to A$ | $A \vee 1 \to 1$ | $A \vee A \to A$ |
| $0 - A \to 0$ | $1 - A \to \neg A$ | $A - 0 \to A$ | $A - 1 \to 0$ | $A - A \to 0$ |
| $0 \Rightarrow A \to 1$ | $1 \Rightarrow A \to A$ | $A \Rightarrow 0 \to \neg A$ | $A \Rightarrow 1 \to 1$ | $A \Rightarrow A \to 1$ |
| $0 \Leftrightarrow A \to \neg A$ | $1 \Leftrightarrow A \to A$ | $A \Leftrightarrow 0 \to \neg A$ | $A \Leftrightarrow 1 \to A$ | $A \Leftrightarrow A \to 1$ |
| $(0, A, B) \to A$ | $(1, A, B) \to B$ | $(A, 0, 1) \to A$ | $(A, 1, 0) \to \neg A$ | $(A, B, B) \to B$ |

Abbildung 4.4: Vereinfachungsregeln für Formeln

**Beweis** Beide Behauptungen können getrennt für jede Regel verifiziert werden. Wegen der Kongruenzeigenschaft (Proposition 4.3.2) genügt es, für jede Instanz jeder Regel zu zeigen, dass die linke Seite äquivalent zu rechten ist. Das ist nicht schwer. □

**Proposition 4.8.4** *Jede variablenfreie Formel $A$ lässt sich durch Anwenden der Regeln in Abbildung 4.4 in eine äquivalente Formel $B$ mit $B \in \{0, 1\}$ vereinfachen.*

**Beweis** Auf eine variablenfreie Formel ist nur dann keine der Regeln anwendbar, wenn es sich um 0 oder 1 handelt. Da jede Regel die Formel echt kleiner macht (Proposition 4.8.3), sind ausgehend von einer Formel nur endlich viele Regelanwendungen möglich. Da jede Regel eine Formel durch eine äquivalente Formel ersetzt (Proposition 4.8.3), ist die vereinfachte Formel äquivalent zur Ausgangsformel. □

Wir wissen jetzt, dass es genau eine Übersetzungsfunktion $f \in For_0 \to \{0, 1\}$ gibt, und dass wir zu einer Formel $A$ die Übersetzung $f(A)$ berechnen können, indem wir die Vereinfachungsregeln solange anwenden, bis wir entweder 0 oder 1 erhalten. Dabei spielt es keine Rolle wie wir die Vereinfachungsregeln im Einzelnen anwenden (welche Regel auf welche Teilformel).

Wir wissen jetzt auch, dass zwei variablenfreie Formeln $A$, $B$ genau dann äquivalent sind, wenn $\mathcal{T}(A) = \mathcal{T}(B)$ gilt. Das folgt mit Proposition 4.8.2.

Schließlich haben wir jetzt ein Verfahren, mit dem wir entscheiden können, ob zwei variablenfreie Formeln äquivalent sind. Dabei vereinfachen wir beide Formeln mit den Regeln in Abbildung 4.4. Wenn wir für beide Formeln dasselbe Endergebnis erhalten, sind die Formeln äqivalent. Wenn wir dagegen verschiedene Endergebnisse erhalten, sind die Formeln nicht äquivalent.

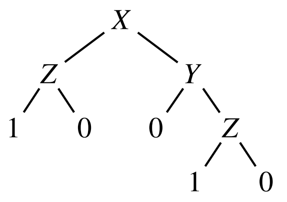

Abbildung 4.5: Der Entscheidungsbaum $(X, (Z, 1, 0), (Y, 0, (Z, 1, 0)))$

## 4.9 Entscheidungsbäume

Wir definieren jetzt eine kanonische Formelmenge, in die jede Formel übersetzt werden kann.

Zunächst definieren wir eine Formelmenge $DT \subseteq For$ mithilfe von struktureller Rekursion:

1. $0 \in DT$ und $1 \in DT$.
2. Wenn $X \in Var$ und $A, B \in DT$, dann $(X, A, B) \in DT$.

Die Elemente von $DT$ bezeichnen wir als *Entscheidungsbäume*. Dieser Name ist aus zwei Gründen sinnvoll. Einerseits können wir die Elemente von $DT$ wie in Abbildung 4.5 gezeigt als geordnete Binärbäume auffasen, deren Blätter mit 0 oder 1 und deren andere Knoten mit Variablen markiert sind.[9] Andererseits gilt die folgende Proposition, die jeden mit einer Variablen markierten Knoten mit einer Entscheidung assoziiert:

**Proposition 4.9.1** *Für jeden Entscheidungsbaum $(X, A, B)$ und jede Funktion $\sigma \in Var \to \mathbb{B}$ gilt:*

$$\mathcal{T}(X, A, B)\sigma \;=\; \text{if } \sigma X = 0 \text{ then } \mathcal{T}(A)\sigma \text{ else } \mathcal{T}(B)\sigma$$

**Beweis** Folgt mit Proposition 4.1.3. □

Ein Entscheidungsbaum $A \in DT$ heißt *reduziert*, wenn für jeden Teilbaum $(X, A_0, A_1)$ von $A$ gilt: die Unterbäume $A_0$ und $A_1$ sind verschieden.

Im Folgenden nehmen wir an, dass eine totale Ordnung $<$ für die Menge $Var$ der Variablen gegeben ist.

[9] Entscheidungsbäume sind geordnete Binärbäumen, bei denen man zwischen linken und rechten Unterbäumen unterscheidet.

Ein Entscheidungsbaum $A \in DT$ heißt *geordnet*, wenn für jeden Teilbaum $(X, A_0, A_1)$ von $A$ gilt: $X$ ist echt kleiner als jede Variable, die in einem der Unterbäumen $A_0$ oder $A_1$ vorkommt. Anschaulich gesprochen heißt das, dass die Variablen auf einem Pfad von der Wurzel zu einem Blatt aufsteigend geordnet sind.

Ein Entscheidungsbaum heißt *Primbaum* genau dann, wenn er reduziert und geordnet ist.

Der Entscheidungsbaum in Abbildung 4.5 ist reduziert. Wenn wir annehmen, dass $X < Y < Z$ gilt, ist er auch geordnet und folglich ein Primbaum.

**Proposition 4.9.2** *Für alle Entscheidungsbäume $A_0$, $A_1$ und für jede Variable $X$ sind die folgenden Bedingungen äquivalent:*

1. *$(X, A_0, A_1)$ ist ein Primbaum.*
2. *$A_0$ und $A_1$ sind verschiedene Primbäume und $X$ ist echt kleiner als jede der in $A_0$ oder $A_1$ vorkommenden Variablen.*

**Lemma 4.9.3** *Sei $(X, A_0, A_1)$ eine Formel, sodass die Variable $X$ in $A_0$ und $A_1$ nicht vorkommt, und sodass $\mathcal{T}(A_0) \neq \mathcal{T}(A_1)$. Dann existiert $\sigma \in Var \to \mathbb{B}$ mit $\mathcal{T}(A)\sigma \neq \mathcal{T}(A)(\sigma[1/X])$.*

**Beweis** Proposition 4.9.1 und 4.3.3. □

**Lemma 4.9.4 (Kanonizität)** *Seien $A$ und $B$ verschiedene Primbäume. Dann $\mathcal{T}(A) \neq \mathcal{T}(B)$.*

**Beweis** Durch Induktion über das Maximum $n$ der Größen von $A$ und $B$.

$n = 1$. Dann $A, B \in \{0, 1\}$. Also $\mathcal{T}(A) \neq \mathcal{T}(B)$, da $A \neq B$.

$n > 1$. Wir unterscheiden 4 Fälle:

$A = (X, A_0, A_1)$ *und* $X$ *kommt in* $B$ *nicht vor.* Nach Induktionsannahme gilt $\mathcal{T}(A_0) \neq \mathcal{T}(A_1)$. Also existiert nach Lemma 4.9.3 ein $\sigma \in Var \to \mathbb{B}$ mit $\mathcal{T}(A)\sigma \neq \mathcal{T}(A)(\sigma[1/X])$. Da $X$ in $B$ nicht vorkommt, gilt $\mathcal{T}(B)\sigma = \mathcal{T}(B)(\sigma[1/X])$. Also $\mathcal{T}(A) \neq \mathcal{T}(B)$.

$B = (X, B_0, B_1)$ *und* $X$ *kommt in* $A$ *nicht vor.* Analog.

$A = (X, A_0, A_1)$ *und* $B = (X, B_0, B_1)$ *und* $A_0 \neq B_0$. Nach Induktionsannahme gilt $\mathcal{T}(A_0) \neq \mathcal{T}(B_0)$. Also existiert ein $\sigma \in Var \to \mathbb{B}$ mit $\mathcal{T}(A_0)\sigma \neq \mathcal{T}(B_0)\sigma$. Da $X$ weder in $A_0$ noch $A_1$ vorkommt,folgt $\mathcal{T}(A)(\sigma[0/X]) = \mathcal{T}(B)(\sigma[0/X])$. Also $\mathcal{T}(A) \neq \mathcal{T}(B)$.

$A = (X, A_0, A_1)$ *und* $B = (X, B_0, B_1)$ *und* $A_1 \neq B_1$. Analog. □

## 4.10 Bestimmung des äquivalenten Primbaums

Wir zeigen jetzt, dass man zu jeder Formel einen äquivalenten Primbaum berechnen kann. Wegen der gerade gezeigten Kanonizität von Primbäumen erhalten wir damit eine Entscheidungsprozedur für die Äquivalenz von Formeln. Außerdem folgt, dass eine Boolesche Gleichung genau dann in jeder Booleschen Algebra gültig ist, wenn sie in der zweiwertigen Algebra gültig ist.

**Proposition 4.10.1** *Jeder geordnete Entscheidungsbaum kann mit der Regel*

$$(X, A, A) \quad \to \quad A$$

*zu einem äquivalenten Primbaum vereinfacht werden.*

**Beweis** Die Regel führt geordnetet Bäume in geordnete Bäume über. Sie ist genau dann auf einen Baum anwendbar, wenn er nicht reduziert ist. Da sie die Größe eines Baums echt verkleinert, kann sie auf einen Baum nur endlich oft angewendet werden. Da sie einen Teilbaum durch eine äquivalenten Teilbaum ersetzt (für jede Formel $(X, A, A)$ gilt $(X, A, A) \models\!\!\!\dashv A$), überführt sie jeden Baum in einen äquivalenten Baum (Kongruenzeigenschaft). $\Box$

**Proposition 4.10.2 (Shannon Expansion)** *Für jede Formel A und jede Variable X gilt:*

1. $X \wedge A \models\!\!\!\dashv X \wedge A[1/X]$.
2. $\neg X \wedge A \models\!\!\!\dashv X \wedge A[0/X]$.
3. $A \models\!\!\!\dashv (X, A[0/X], A[1/X])$.

**Beweis**

1. Durch strukturelle Induktion über $A$.

   $X \notin VV(A)$. Dann $A[1/X] = A$. Also $X \wedge A \models\!\!\!\dashv X \wedge A[1/X]$.

   $A = X$. Dann $X \wedge A \models\!\!\!\dashv X \wedge X \models\!\!\!\dashv X \wedge 1 \models\!\!\!\dashv \wedge A[1/X]$.

   $A = \neg A'$. Zuerst stellen wir fest, dass

   $$(*) \quad x \wedge \neg y = x \wedge \neg(x \wedge y)$$

   eine gültige Boolesche Gleichung ist. Damit haben wir:

   $$\begin{aligned} X \wedge A &\models\!\!\!\dashv X \wedge \neg(X \wedge A') && (*) \\ &\models\!\!\!\dashv X \wedge \neg(X \wedge A'[1/X]) && \text{Induktionsannahme und Kongruenz} \\ &\models\!\!\!\dashv X \wedge \neg A'[1/X] && (*) \\ &\models\!\!\!\dashv X \wedge A[1/X]) \end{aligned}$$

$A = A_1 \wedge A_2$. Zuerst stellen wir fest, dass

$$(**) \quad x \wedge (y \wedge z) = x \wedge ((x \wedge y) \wedge (x \wedge z))$$

eine gültige Boolesche Gleichung ist. Damit haben wir:

$$\begin{aligned} X \wedge A &\models\!\mid X \wedge ((X \wedge A_1) \wedge (X \wedge A_2)) && (**) \\ &\models\!\mid X \wedge ((X \wedge A_1[1/X]) \wedge (X \wedge A_2[1/X])) && \text{Induktionsannahme} \\ &\models\!\mid X \wedge (A_1[1/X] \wedge A_2[1/X]) && (**) \\ &\models\!\mid X \wedge A[1/X]) \end{aligned}$$

Die restlichen Fälle folgen analog.

2. Analog zu (1).
3. Mit (2) und (1) folgt:

$$\begin{aligned} A &\models\!\mid \neg X \wedge A \vee X \wedge A \\ &\models\!\mid \neg X \wedge A[0/X] \vee X \wedge A[1/X] \\ &\models\!\mid (X, A[0/X], A[1/X]) \end{aligned}$$

□

**Lemma 4.10.3** *Zu jeder Formel $A$ kann man einen äquivalenten Primbaum $B$ mit $VV(B) \subseteq VV(A)$ berechnen.*

**Beweis** Durch Induktion über die Anzahl $n$ der in $A$ vorkommenden Variablen.

$n = 0$. Dann bekommt man den Primbaum $B$ mit Proposition 4.8.4.

$n > 1$. Sei $X$ die kleinste in $A$ vorkommende Variable. Die Induktionsannahme liefert uns Primbäume $B_0$ und $B_1$ wie folgt:

$$\begin{aligned} B_0 &\models\!\mid A[0/X] \;\wedge\; VV(B_0) \subseteq VV(A) - \{X\} \\ B_1 &\models\!\mid A[1/X] \;\wedge\; VV(B_1) \subseteq VV(A) - \{X\} \end{aligned}$$

Proposition 4.10.2 liefert $A \models\!\mid (X, B_0, B_1)$. Wenn $B_0 \neq B_1$, dann ist $(X, B_0, B_1)$ ein Primbaum mit den behaupteten Eigenschaften (Proposition 4.9.2). Falls $B_0 = B_1$, gilt $(X, B_0, B_1) \models\!\mid B_1$. Also ist $B_1$ ein Primbaum mit den behaupteten Eigenschaften (Proposition 4.10.1). □

**Satz 4.10.4 (Primbäume)** *Die Menge der Primbäume ist kanonisch und zu jeder Formel existiert genau ein äquivalenter Primbaum. Der zu einer Formel äquivalente Primbaum enthält nur Variablen, die in der Formel vorkommen.*

**Beweis** Lemmata 4.9.4 und 4.10.3. □

Den zu einer Formel $A$ äquivalenten Primbaum bezeichnen wir mit $\pi A$.

**Korollar 4.10.5 (Signifikante Variablen)** *Die signifikanten Variablen einer Formel $A$ sind genau die Variablen, die in $\pi A$ vorkommen.*

**Satz 4.10.6 (Tautologie)** $\forall A, B \in For\colon\ A \models\!\!\!\!\models B \iff \mathcal{T}(A) = \mathcal{T}(B)$.

**Beweis** Folgt mit Proposition 4.8.2 aus Satz 4.10.4. □

**Korollar 4.10.7** *Eine Boolesche Gleichung ist genau dann in allen Booleschen Algebren gültig, wenn sie in der zweiwertigen Booleschen Algebra gültig ist.*

**Beweis** Folgt als Satz 4.10.6. □

Eine Variable $X$ heißt *signifikant* für eine Funktion $f \in (Var \to \mathbb{B}) \to \mathbb{B}$, wenn eine Belegung $\sigma \in Var \to \mathbb{B}$ und ein $b \in \mathbb{B}$ existieren mit $f(\sigma) \neq f(\sigma[b/X])$.

**Korollar 4.10.8 (Signifikante Variablen)** *Sei $A$ eine Formel und $X$ eine Variable. Dann ist $X$ genau dann signifikant für $A$, wenn $X$ signifikant für $\mathcal{T}(A)$ ist.*

**Beweis** Wir zeigen, dass $X$ genau dann nicht signifikant für $A$ ist, wenn $X$ nicht signifikant für $\mathcal{T}(A)$ ist.

Sei $X$ für $A$ nicht signifikant. Dann gibt es eine zu $A$ äquivalente Formel $B$ mit $X \notin VV(B)$. Mit Proposition 4.3.3 folgt, dass $X$ für $\mathcal{T}(B)$ nicht signifikant ist. Da $\mathcal{T}(A) = \mathcal{T}(B)$, folgt, dass $X$ nicht signifikant für $\mathcal{T}(A)$ ist.

Sei $X$ für $\mathcal{T}(A)$ nicht signifikant. Dann folgt mit dem Substitutionslemma 4.3.1, dass $\mathcal{T}(A) = \mathcal{T}(A[0/X])$. Mit dem Tautologiesatz 4.10.6 folgt $A \models\!\!\!\!\models A[0/X]$. Da $X \notin VV(A[0/X])$, folgt, dass $X$ nicht signifikant für $A$ ist. □

Die folgende Proposition fasst wichtige Eigenschaften von Primbäumen zusammen:

**Proposition 4.10.9** *Für alle Formeln $A$, $B$ gilt:*

1. $\pi A \models\!\!\!\!\models A$
2. $VV(\pi A) = SV(A) \subseteq VV(A)$
3. $\pi A = A \iff A$ *Primbaum*
4. $A \models\!\!\!\!\models B \iff \pi A = \pi B$

### Methode zur Bestimmung des äquivalenten Primbaums

Gegeben eine Formel $A$, kann $\pi A$ wie folgt bestimmt werden:

1. Wähle eine zu $A$ äquivalente Formel $B$ wie folgt:
   a) $VV(B) \subseteq VV(A)$.
   b) $VV(B) = \emptyset \;\Rightarrow\; B = 0 \vee B = 1$.
2. Wenn $VV(B) = \emptyset$, dann $\pi A = B$.
3. Wenn $VV(B) \neq \emptyset$, bestimme die kleinste in $B$ vorkommende Variable $X$.
4. Bestimme durch Rekursion: $A_0 = \pi(B[0/X])$ und $A_1 = \pi(B[1/X])$.
5. Wenn $A_0 = A_1$, dann $\pi A = A_0$.
6. Wenn $A_0 \neq A_1$, dann $\pi A = (X, A_0, A_1)$.

Die Methode terminiert, da die rekursiven Aufrufe Formeln mit weniger vorkommenden Variablen bearbeiten. Die für Schritt 1 erforderliche Vereinfachung kann durch die Vereinfachungsregeln in Abbildung 4.4 erledigt werden. Die Korrektheit der Methode folgt mit den Argumenten aus dem Beweis von Lemma 4.10.3.

Wir demonstrieren die Methode an der Formel

$$A \;=\; (X+Z)(\overline{X}+\overline{Y}+\overline{Z})(X\overline{Y}+\overline{Z})$$

und der Variablenordnung $X < Y < Z$. Wir expandieren $A$ nach $X$ und vereinfachen:

$$A[0/X] = (0+Z)(\overline{0}+\overline{Y}+\overline{Z})(0\overline{Y}+\overline{Z}) \mathrel{\models\!\!\!\dashv} Z \cdot 1 \cdot \overline{Z} \mathrel{\models\!\!\!\dashv} 0$$
$$A[1/X] = (1+Z)(\overline{1}+\overline{Y}+\overline{Z})(1\overline{Y}+\overline{Z}) \mathrel{\models\!\!\!\dashv} 1(\overline{Y}+\overline{Z})(\overline{Y}+\overline{Z}) \mathrel{\models\!\!\!\dashv} \overline{Y}+\overline{Z}$$

Links sind wir fertig. Rechts expandieren wir nach $Y$ und vereinfachen:

$$A[1/X][0/Y] = \overline{0}+\overline{Z} = 1$$
$$A[1/X][1/Y] = \overline{1}+\overline{Z} = \overline{Z}$$

Links sind wir fertig. Rechts expandieren wir nach $Z$:

$$A[1/X][1/Y][0/Z] = 1$$
$$A[1/X][1/Y][0/Z] = 0$$

Damit ist $\pi A$ der folgende Baum:

```
    X
   / \
  0   Y
     / \
    1   Z
       / \
      1   0
```

**Methoden zur Verifikation der Gültigkeit von Booleschen Gleichungen**

Wir haben bisher drei Methoden, um zu verifizieren, ob eine Boolesche Gleichung gültig ist:

1. Wir können versuchen, die Gleichung mithilfe der Axiome für Boolesche Algebren beweisen. Falls die Gleichung ungültig ist, liefert dieses Vorgehen kein brauchbares Ergebnis.

2. Wir können die Gleichung für alle Belegungen $Var \to \mathbb{B}$ nachrechnen. Dabei können wir uns auf die endlich vielen Variablen beschränken, die in der Gleichung vorkommen. Falls die Gleichung ungültig ist, bekommen wir eine Belegung, die dies zeigt.

3. Wir können zu jeder Seite der Gleichung den äquivalenten Primbaum bestimmen. Die Gleichung ist gültig genau dann, wenn die beiden Primbäume gleich sind.

Die zweite und dritte Methode können automatisiert werden. Da die Anzahl der zu prüfenden Belegungen exponentiell in der Anzahl der vorkommenden Variablen wächst, wird Methode 2 für größere Probleme sehr schnell unbrauchbar. Dagegen kann Methode 3 so verfeinert werden, dass sie auch bei einigen hundert Variablen häufig zu brauchbaren Ergebnissen führt.

Eine in allen Fällen effiziente Verifikationsmethode ist nicht zu erwarten, da das Falsifizieren von Äquivalenzen wegen

$$A \not\models\!\!\!\dashv 0 \iff \exists \sigma \in Var \to \mathbb{B}\colon \mathcal{T}(A)\sigma = 1$$

äquivalent zum NP-vollständigen Erfüllbarkeitsproblem ist.

## 4.11 Primbaumalgorithmen

Formeln und Primbäume sind zwei unterschiedliche Darstellungssysteme für Boolesche Mengen, deren Eigenschaften sich vorteilhaft ergänzen. Formeln eignen sich gut für die Modellierung von Problemen (siehe Diätproblem). Primbäume liefern eine effiziente Datenstruktur für Boolesche Mengen, wie wir in diesem Abschnitt sehen werden. Wir können also sagen, dass es sich bei Formeln um eine *modellierungsorientierte* und bei Primbäumen um eine *verarbeitungsorientierte Darstellung* für Boolesche Mengen handelt. Mit einem Übersetzer, der Formeln in äquivalente Primbäume übersetzt, können die Vorteile beider Darstellungen kombiniert werden.

$$neg \in DT \to DT$$

$$\begin{aligned} neg(0) &= 1 \\ neg(1) &= 0 \\ neg(X, A_0, A_1) &= (X, neg(A_0), neg(A_1)) \end{aligned}$$

Abbildung 4.6: Negationsalgorithmus für Entscheidungsbäume

Wir geben jetzt effiziente Algorithmen an, die die Booleschen Operationen für Primbäume realisieren. Beispielsweise werden wir eine Prozedur *neg* sehen, der zu einem Primbaum $A$ den Primbaum für $\neg A$ liefert. Weiter werden wir eine Prozedur *and* sehen, der zu zwei Primbäumen $A$ und $B$ den zu $A \wedge B$ äquivalenten Primbaum liefert. Schließlich werden wir im nächsten Abschnitt eine Datenstruktur für Primbäume sehen, mit der die Gleichheit von Primbäumen in konstanter Zeit getestet werden kann.

**Negation**

Abbildung 4.6 definiert eine Funktion und eine Prozedur $neg \in DT \to DT$. Die Definition erfolgt mithilfe von struktureller Rekursion. Man sieht sofort, das die Prozedur *neg* lineare Laufzeit hat. Die Funktion *neg* liefert zu einem Entscheidungsbaum den Baum, den man erhält, wenn man an allen Blättern 0 mit 1 und 1 mit 0 vertauscht.

**Proposition 4.11.1** *Für jeden Entscheidungsbaum A gilt:*

1. $neg(A) \models\mid \neg A$.
2. *Wenn $A$ ein Primbaum ist, dann ist $neg(A)$ ein Primbaum.*

**Beweis** Behauptung (1) folgt mit struktureller Induktion über $A$. Die Fälle $A = 0$ und $A = 1$ sind offensichtlich. Der rekursive Fall $A = (X, A_0, A_1)$ folgt mit Proposition 4.1.4 (2).

Behauptung (2) folgt ebenfalls mit struktureller Induktion über $A$. Die Fälle $A = 0$ und $A = 1$ sind offensichtlich. Den rekursiven Fall $A = (X, A_0, A_1)$ zeigen wir wie folgt. Sei $A = (X, A_0, A_1)$ ein Primbaum. Nach Induktionsannahme sind $neg(A_0)$ und $neg(A_1)$ Primbäume. Daraus folgt mit Behauptung (1): $neg(A_0) = \pi(\neg A_0)$ und $neg(A_1) = \pi(\neg A_1)$. Die Definition von *neg* liefert jetzt:

$$neg(A) \;=\; (X, neg(A_0), neg(A_1)) \;=\; (X, \pi(\neg A_0), \pi(\neg A_0))$$

Da $A = (X, A_0, A_1)$ ein Primbaum ist und $\pi(\neg A_0)$ und $\pi(\neg A_1)$ nur Variablen enthalten, die in $A_0$ beziehungsweise $A_1$ vorkommen (Primbaumsatz), ist

$$
\begin{aligned}
&\mathit{and} \in DT \times DT \to DT\\
&\qquad\qquad\qquad \mathit{and}(0, B) = 0\\
&\qquad\qquad\qquad \mathit{and}(1, B) = B\\
&\qquad\qquad\qquad \mathit{and}(A, 0) = 0\\
&\qquad\qquad\qquad \mathit{and}(A, 1) = A\\
&\qquad\qquad\qquad \mathit{and}(A, A) = A\\
&\mathit{and}((X, A_0, A_1),\ (Y, B_0, B_1)) =\\
&\quad \text{case } X = Y \text{ then } \mathit{red}\,(X,\ \mathit{and}(A_0, B_0),\ \mathit{and}(A_1, B_1))\\
&\qquad\quad X < Y \text{ then } \mathit{red}\,(X,\ \mathit{and}(A_0, (Y, B_0, B_1)),\ \mathit{and}(A_1, (Y, B_0, B_1)))\\
&\qquad\quad X > Y \text{ then } \mathit{red}\,(Y,\ \mathit{and}((X, A_0, A_1), B_0),\ \mathit{and}((X, A_0, A_1), B_1))
\end{aligned}
$$

$$
\begin{aligned}
&\mathit{red} \in \mathit{Var} \times DT \times DT \to DT\\
&\mathit{red}(X, A_0, A_1) = \text{ if } A_0 = A_1 \text{ then } A_0 \text{ else } (X, A_0, A_1)
\end{aligned}
$$

Abbildung 4.7: Konjunktionsalgorithmus für Entscheidungsbäume

$\mathit{neg}(A) = (X, \pi(\neg A_0), \pi(\neg A_0))$ ein Primbaum, wenn $\pi(\neg A_0) \neq \pi(\neg A_1)$. Wegen des Primbaumsatzes ist dies der Fall, wenn $\neg A_0 \not\models\!\!\!\dashv \neg A_1$ gilt. Da $A_0$ und $A_1$ verschiedene Primbäume sind, folgt mit dem Primbaumsatz $A_0 \not\models\!\!\!\dashv A_1$. Also $\neg A_0 \not\models\!\!\!\dashv \neg A_1$ mit Proposition 4.1.5. □

**Konjunktion**

Abbildung 4.7 definiert eine Funktion und eine Prozedur

$$\mathit{and} \in DT \times DT \to DT$$

Die Definition erfolgt mithilfe von struktureller Rekursion und einer Hilfsfunktion *red*. Im nächsten Abschnitt werden wir eine Datenstruktur angeben, mit der die Gleichheit von Entscheidungsbäumen in konstanter Zeit getestet werden kann. Damit hat $\mathit{and}(A, B)$ die Laufzeit $O(|A| \cdot |B|)$, da für jedes Paar von Teilbaumauftreten von $A$ und $B$ höchstens ein Rekursionsschritt benötigt wird.[10]

**Proposition 4.11.2** *Für jeden Entscheidungsbaum* $(X, A_0, A_1)$ *gilt:*

1. $\mathit{red}(X, A_0, A_1) \models\!\!\!\dashv (X, A_0, A_1)$.

[10] $O(|A| \cdot |B|)$ ist eine scharfe obere Schranke, da es für jedes $n$ Primbäume $A$, $B$ der Größe $n$ gibt, sodass der Primbaum für $A \wedge B$ die Größe $\Theta(n^2)$ hat. Siehe [Bryant 1986].

2. *Wenn $A_0$ und $A_1$ Primbäume sind, in denen nur Variablen vorkommen, die echt größer als $X$ sind, dann ist $red(X, A_0, A_1)$ ein Primbaum.*

**Beweis** Folgt mit Proposition 4.10.1 und 4.9.2. □

**Proposition 4.11.3** *Für alle Entscheidungsbäume $A$, $B$ gilt:*

1. $and(A, B) \models\mathrel{\!\!\mid} A \wedge B$.
2. *Wenn $A$ und $B$ Primbäume sind, dann ist $and(A, B)$ ein Primbaum.*

**Beweis** Behauptung (1) folgt mit struktureller Induktion über $(A, B)$. Die Fälle $A = 0$, $A = 1$, $B = 0$, $B = 1$ und $A = B$ sind offensichtlich. Die verbleibenden Fälle folgen mit Proposition 4.11.2 (1) und den Gleichungen (3) und (4) von Proposition 4.1.4.

Behauptung (2) folgt ebenfalls mit struktureller Induktion über $(A, B)$. Die Fälle $A = 0$, $A = 1$, $B = 0$, $B = 1$ und $A = B$ sind offensichtlich. Wir zeigen einen der verbleibenden Fälle. Die anderen folgen analog.

Seien $A = (X, A_0, A_1)$ und $B = (X, B_0, B_1)$ Primbäume. Dann sind $and(A_0, B_0)$ und $and(A_1, B_1)$ nach Induktionsannahme ebenfalls Primbäume. Mit Behauptung (1) und dem Primbaumsatz folgt $and(A_0, B_0) = \pi(A_0 \wedge B_0)$ und $and(A_1, B_1) = \pi(A_0 \wedge B_0)$. Die Definition von *and* liefert jetzt:

$$and(A, B) = red\,(X,\ \pi(A_0 \wedge B_0),\ \pi(A_1 \wedge B_1))$$

Da $(X, A_0, A_1)$ und $(X, B_0, B_1)$ Primbäume sind, enthalten $\pi(A_0 \wedge B_0)$ und $\pi(A_1 \wedge B_1)$ nur Variablen, die echt größer als $X$ sind (Primbaumsatz). Also folgt mit Proposition 4.11.2, dass $and(A, B)$ ein Primbaum ist. □

### Weitere binäre Operationen

Die Booleschen Operation $\vee$, $-$, $\Rightarrow$ und $\Leftrightarrow$ lassen sich alle nach demselben Schema realisieren wie Konjunktion. Letztlich liegt das daran, dass sie sich alle mit Konjunktion und Negation ausdrücken lassen (z.B. $x \Rightarrow y = \neg(x \wedge y)$). Auf die Gültigkeit der entsprechenden Booleschen Gleichungen haben wir bereits im Kontext von Proposition 4.1.4 hingewiesen.

### Übersetzung von Formeln in Primbäume

Mithilfe der Prozeduren *neg* und *and* kann man Formeln in Primbäume übersetzen. Sei $For_1$ die Menge aller Formeln, die mit Variablen, 0, 1, $\neg$, $\vee$ und $\wedge$ gebildet werden können, und $PT$ die Menge aller Primbäume. Abbildung 4.8 definiert eine Funktion und eine Prozedur $com \in For_1 \to PT$, die Formeln in Primbäume übersetzt.

$$com \in For_1 \to PT$$

$$\begin{aligned}
com(0) &= 0\\
com(1) &= 1\\
com(X) &= (X, 0, 1)\\
com(\neg A) &= neg(com(A))\\
com(A_1 \wedge A_2) &= and(com(A_1), com(A_2))\\
com(A_1 \vee A_2) &= neg(and(neg(com(A_1)), neg(com(A_2))))
\end{aligned}$$

Abbildung 4.8: Übersetzung von Formeln in Primbäume

**Proposition 4.11.4** *Für jede Formel $A \in For_1$ gilt: $com(A) \mathrel{|\!\!=\!\!|} A$.*

**Beweis** Durch strukturelle Induktion über $A$. Der Fall $\neg A$ folgt mit Proposition 4.11.1, und $A_1 \wedge A_2$ mit Proposition 4.11.3. Der Fall $A_1 \vee A_2$ folgt mit $A_1 \vee A_2 \mathrel{|\!\!=\!\!|} \neg(\neg A_1 \wedge \neg A_2)$ und den Propositionen 4.11.1 und 4.11.3. □

## 4.12 Minimale Graphdarstellung

Wir zeigen jetzt, wie Entscheidungsbäume so dargestellt werden können, dass die Gleichheit von zwei Bäumen in konstanter Zeit getestet werden kann. Eine solche Darstellung liefert die Grundlage für eine effiziente Realisierung der Primbaumalgorithmen. Die vorgestellte Darstellungstechnik kann auf beliebige Bäume übertragen werden.

Die grundlegende Idee unserer Darstellung beruht auf der Beobachtung, dass eine endliche Menge von Bäumen platzsparend durch einen zyklenfreien Graphen dargestellt werden kann. Abbildung 4.9 zeigt links einen Graphen, der alle Teilbäume des rechts gezeigten Entscheidungsbaums darstellt. Jeder Knoten des Graphens stellt einen Teilbaum des Entscheidungsbaums dar. Dabei entsprechen die gestrichelten Kanten des Graphens den zu den linken Unterbäumen führenden Kanten des Baums, und die durchgezogenen Kanten den zu den rechten Unterbäumen führenden Kanten. Der mit $X$ markierte Knoten des Graphens stellt den kompletten Entscheidungsbaum dar. Da der Graph mehrfach auftretende Teilbäume des Baums nur einmal darstellt, ist er kleiner als der Baum (5 statt 9 Knoten).

Eine Graphdarstellung einer Menge von Entscheidungsbäumen heißt *minimal*, wenn zwei verschiedene Knoten des Graphens immer zwei verschiedene Entscheidungsbäume darstellen. Die in Abbildung 4.9 gezeigte Graphdarstellung ist

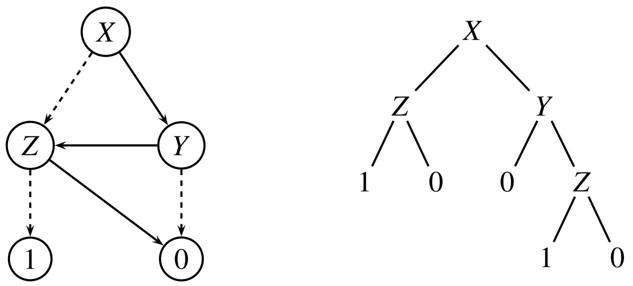


Abbildung 4.9: Graphdarstellung der Teilbäume eines Entscheidungsbaums

minimal. Da bei einer minimalen Graphdarstellung einer Menge $T$ von Entscheidungsbäumen die Knoten des Graphens bijektiv den Entscheidungsbäumen in $T$ entsprechen, kann die Gleichheit von zwei dargestellten Bäumen in konstanter Zeit getestet werden.

Wir formalisieren unser Konzept der Graphdarstellung wie folgt. Ein *Entscheidungsgraph* ist eine Funktion $\gamma$, für die ein $N \in \mathbb{N}^+$ existiert, sodass:

1. $\gamma \in \{2, \ldots, N\} \to \mathit{Var} \times \{0, \ldots, N\} \times \{0, \ldots, N\}$.
2. $\forall\, (n, (X, n_0, n_1)) \in \gamma :\; n > n_0 \wedge n > n_1$.

Unter den *Knoten von $\gamma$* verstehen wir die Zahlen $0, \ldots, N$. Die Knoten 0 und 1 stellen die Entscheidungsbäume 0 und 1 dar. Alle anderen Knoten stellen zusammengesetzte Entscheidungsbäume dar. Die zweite Bedingung der Definition sorgt dafür, dass Entscheidungsgraphen azyklisch sind. Die Graphdarstellung in Abbildung 4.9 entspricht dem Entscheidungsgraphen

$$\{\, 2 \mapsto (Z, 1, 0),\quad 3 \mapsto (Y, 0, 2),\quad 4 \mapsto (X, 2, 3) \,\}$$

Die Menge aller Entscheidungsgraphen bezeichnen wir mit *DG*. Die Funktion

$$\begin{aligned}
&\tau \in DG \to \mathbb{N} \overset{\mathit{fin}}{\rightharpoonup} DT \\
&\quad \tau\gamma 0 = 0 \\
&\quad \tau\gamma 1 = 1 \\
&\quad \tau\gamma n = (X, \tau\gamma n_0, \tau\gamma n_1) \quad \text{falls } \gamma n = (X, n_0, n_1)
\end{aligned}$$

liefert zu jedem Entscheidungsgraphen $\gamma$ eine Funktion

$$\tau\gamma \;\in\; \{0, 1\} \cup \mathit{Dom}\,\gamma \to DT$$

die jedem Knoten von $\gamma$ einen Entscheidungsbaum zuordnet. Die rekursive Definition von $\gamma$ ist wohlfundiert, da Entscheidungsgraphen azyklisch sind.

Ein Entscheidungsgraph $\gamma$ heißt *minimal*, wenn $\tau\gamma$ injektiv ist. Die Größe eines minimalen Entscheidungsgraphens hängt linear von der Anzahl der dargestellten Entscheidungsbäume ab.

**Satz 4.12.1** *Ein Entscheidungsgraph ist minimal genau dann, wenn er injektiv ist.*

**Beweis** Sei $\gamma$ ein Entscheidungsgraph.

Sei $\gamma$ minimal und seien $n, n' \in \mathit{Dom}\ \gamma$ mit $\gamma n = \gamma n'$. Wir müssen $n = n'$ zeigen. Da $\gamma n = \gamma n'$, existieren $X$, $n_0$ und $n_1$ mit $\gamma n = \gamma n' = (X, n_0, n_1)$. Da $n, n' > 1$, folgt aus der Definition von $\tau$, dass

$$\tau\gamma n \;=\; (X, \tau\gamma n_0, \tau\gamma n_1) \;=\; \tau\gamma n'$$

Da $\tau\gamma$ injektiv ist, folgt $n = n'$.

Sei $\gamma$ injektiv. Wir müssen

$$\forall n, n' \in \{0, 1\} \cup \mathit{Dom}\ \gamma : \;\; \tau\gamma n = \tau\gamma n' \Rightarrow n = n'$$

zeigen. Wir tun das durch Induktion über $k = \max\{n, n'\}$. Für $k = 0$ und $k = 1$ kann die Behauptung mithilfe der Definition von $\tau$ leicht verifiziert werden. Sei $k > 1$ und $\tau\gamma n = \tau\gamma n'$. Wir müssen $n = n'$ zeigen. Da $k > 1$, folgt $n > 1$ oder $n' > 1$. Da $\tau\gamma n = \tau\gamma n'$, folgt aus der Definition von $\tau$, dass $n, n' > 1$ und

$$\gamma n = (X, n_0, n_1) \qquad \gamma n' = (X', n_0', n_1')$$

für passende Variablen $X$, $X'$ und Zahlen $n_0, n_1, n_0', n_1' < k$. Da

$$(X, \tau\gamma n_0, \tau\gamma n_1) \;=\; \tau\gamma n \;=\; \tau\gamma n' \;=\; (X', \tau\gamma n_0', \tau\gamma n_1')$$

folgt $X = X'$, $\tau\gamma n_0 = \tau\gamma n_0'$ und $\tau\gamma n_1 = \tau\gamma n_1'$. Mit der Induktionsannahme bekommen wir $n_0 = n_0'$ und $n_1 = n_1'$. Also $\gamma n = \gamma n'$. Da $\gamma$ injektiv ist, folgt $n = n'$. □

**Korollar 4.12.2** *Ein Entscheidungsgraph $\gamma$ ist minimal genau dann, wenn zu jeder Variablen $X$ und zu je zwei Knoten $n_0$, $n_1$ von $\gamma$ höchstens ein Knoten $n$ von $\gamma$ mit $\gamma n = (X, n_0, n_1)$ existiert.*

Mit dem Korollar sieht man sofort, dass der Entscheidungsgraph in Abbildung 4.9 minimal ist.

Die Primbaumalgorithmen des letzten Abschnitts allozieren die benötigten Entscheidungsbäume schrittweise: Begonnen wird mit den primitiven Bäumen 0 und 1, und zu einer Variable $X$ und zwei bereits allozierten Bäumen $A$, $B$ wird gegebenfalls der Baum $(X, A, B)$ alloziert. Also können wir die folgende Schnittstelle für die Darstellung von Entscheidungsbäumen annehmen:

```
type var = int
type dt
datatype info = F | T | D of var * dt * dt
val put : info -> dt
val get : dt -> info
val eq  : dt * dt -> bool
```

Dabei soll `put F` die Darstellung des Baums 0 und `put T` die Darstellung des Baums 1 liefern. Wenn $n_0$ und $n_1$ bereits Darstellungen von Bäumen $A_0$ und $A_1$ sind, soll `put(D(`$X$`,`$n_0$`,`$n_1$`))` die Darstellung des Baums $(X, A_0, A_1)$ liefern. Die Prozedur `get` ist invers zu `put` und liefert zu einer Darstellung Information über die oberste Ebene des dargestellten Baums. Die Prozedur `eq` testet, ob zwei Darstellungen denselben Baum darstellen.

Diese Schnittstelle können wir mit einer minimalen Graphdarstellung so realisieren, dass die Operationen `put`, `get` und `eq` jeweils nur konstante Zeit benötigen. Dafür verwenden wir eine imperative Datenstruktur, deren Zustände minimale Entscheidungsgraphen darstellen. Der initiale Zustand der Datenstruktur entspricht dem Entscheidungsgraphen $\emptyset$, der die Bäume 0 und 1 darstellt. Am interessantesten ist die Realisierung der Operation `put`. Wenn der aktuelle Zustand der Datenstruktur dem Entscheidungsgraphen $\gamma$ entspricht, und `put` zu $(X, n_0, n_1)$ die Darstellung für den Baum $(X, \tau\gamma n_0, \tau\gamma n_1)$ liefern soll, gibt es zwei Möglichkeiten:

1. Wenn bereits ein Knoten $n$ mit $\gamma n = (X, n_0, n_1)$ existiert, ist dieser die eindeutige Darstellung für den Baum $(X, \tau\gamma n_0, \tau\gamma n_1)$.
2. Ansonsten muss der Entscheidungsgraph $\gamma$ um einen Knoten erweitert werden, da er den Baum $(X, \tau\gamma n_0, \tau\gamma n_1)$ noch nicht darstellt. Der neue Knoten ist $n = \max(\{2\} \cup \mathit{Dom}\ \gamma)$ und der erweiterte Entscheidungsgraph ist $\gamma' = \gamma[(X, n_0, n_1)/n]$. Mit Korollar 4.12.2 folgt, dass $\gamma'$ wieder ein minimaler Entscheidungsgraph ist.

Damit entwickeln sich die Zustände der Datenstruktur wie folgt:

$$\emptyset = \gamma_0 \subseteq \gamma_1 \subseteq \ldots \subseteq \gamma_k$$

Die dadurch ausgedrückte Monotonie erlaubt wichtige Optimierungen. Beispielsweise sorgt sie dafür, dass `get` und `put` bei wiederholter Anwendung auf dasselbe Argument immer dasselbe Ergebnis wie vorher liefern.

Mit diesen Ideen können wir die Datenstruktur für Entscheidungsbäume wie folgt realisieren. Den jeweiligen minimalen Entscheidungsgraphen legen wir in einer Reihung `graph` ab und den nächsten zu allozierenden Knoten merken wir uns mit einer Referenz `next`:

```
type dt = int
val graph : (var*dt*dt) array
val next  : dt ref
```

Die Realisierung von `get` und `eq` mit konstantem Zeitaufwand ist offensichtlich. Damit wir `put` mit konstantem Zeitaufwand realisieren können, benötigen wir für den jeweils dargestellten minimalen Entscheidungsgraphen $\gamma$ eine effiziente Darstellung der inversen Funktion $\gamma^{-1}$. Diese realisieren wir mithilfe einer Hashtabelle und bauen sie schrittweise zusammen mit $\gamma$ auf. Damit lässt sich `put` mit konstantem Zeitaufwand realisieren.

Wir erwähnen noch eine Optimierung für die Negationsprozedur *neg*. Wenn die Prozedur *neg* die Darstellung $n'$ der Negation eines durch $n$ dargestellten Baums berechnet, kann sie bei $n$ und $n'$ jeweils die Darstellung des negierten Baums vermerken ($n'$ beziehungsweise $n$). Danach können $n$ und $n'$ mit konstantem Zeitaufwand negiert werden.

Die Konjunktionsprozedur *and* können wir mit einem Arbeitsspeicher optimieren, in dem *and* zu je zwei Knoten bereits berechnete Ergebnisse vermerkt. Mithilfe von Hashingtechniken können die Schreib- und Leseoperationen des Arbeitspeicher mit konstantem Zeitaufwand realisiert werden. Damit bekommt wir für die Prozedur *and* die Laufzeit $O(|n_1| \cdot |n_2|)$, wobei $|n_i|$ die Anzahl der von $n_i$ aus erreichbaren Knoten des Entscheidungsgraphens bezeichnet.

Die in den letzten zwei Abschnitten vorgestellten Techniken entsprechen der von Randal Bryant 1986 entwickelten OBDD-Darstellung für Boolesche Funktionen.[11] Die OBDD-Darstellung hat in der Praxis zu dramatischen Verbesserungen bei der Darstellungen von Booleschen Funktionen und Mengen geführt und ist in Hinblick auf Effizienz bis heute konkurrenzlos.

Neben ihren Vorteilen hat die OBDD-Darstellung den Nachteil, dass die Größe der minimalen Graphdarstellung ganz erheblich von der Variablenordnung abhängen kann. Je nach Wahl der Variablenordnung kann sich die Größe der minimalen Graphdarstellung exponentiell ändern.

Seien $X_1, \dots, X_{2n}$ paarweise verschiedene Variablen. Dann beschreibt die Formel

$$(b_1 \wedge X_{n+1}) \;\vee\; \cdots \;\vee\; (b_n \wedge X_{2n})$$

offensichtlich für alle $(b_1, \dots, b_n) \in \mathbb{B}^n$ eine andere Boolesche Funktion. Wenn

[11] OBDD steht für Ordered Binaray Decision Diagram. Werfen Sie einen Blick in das Originalpapier: Randal E. Bryant, Graph-based Algorithms for Boolean Function Manipulation. IEEE Transactions on Computers, C-35-8, pp 677-691, 1986.

wir den Primbaum für die Formel

$$A_n \stackrel{\text{def}}{=} (X_1 \wedge X_{n+1}) \vee \cdots \vee (X_n \wedge X_{2n})$$

gemäß der Variablenordnung $X_1 < X_2 < \cdots < X_{2n}$ bestimmen, wird er für jedes Tupel $(b_1, \ldots, b_n) \in \mathbb{B}^n$ einen Teilbaum enthalten, der äquivalent zu der Formel

$$(b_1 \wedge X_{n+1}) \vee \cdots \vee (b_n \wedge X_{2n})$$

ist (Shannon Expansion). Damit hat der Primbaum für $A_n$ mindestens $2^n$ verschiedene Teilbäume. Wenn wir den Primbaum für $A_n$ dagegen nach der Ordnung $X_1 < X_{n+1} < X_2 < X_{n+2} < \cdots < X_n < X_{2n}$ entwickeln, wird diese exponentielle Explosion vermieden und der Primbaum für $A_n$ hat nur $2n + 2$ Knoten.

## 4.13 Klauselformen

Mit Entscheidungsbäumen haben wir eine wichtige kanonische Form für Boolesche Formeln kennengelernt. Jetzt entwickeln wir eine zweite kanonische Form, die sogenannte Klauselformen als Darstellungen verwendet.

Betrachten Sie die Formel

$$((X \vee Z) \wedge (\neg X \vee (\neg Y \vee \neg Z))) \wedge ((X \vee \neg Y) \vee \neg Z)$$

Die Denotation dieser Formel kann durch die *konjunktive Klauselform*

$$\mathbf{K}\,\{\{X, Z\},\ \{\neg X, \neg Y, \neg Z\},\ \{X, \neg Y, \neg Z\}\}$$

dargestellt werden. Dabei handelt es sich um eine Menge von Mengen von Formeln, wobei die äußere Menge konjunktiv und die inneren Mengen disjunktiv interpretiert werden. Beim Übergang von einer Formel mit binären Konjunktionen und Disjunktionen zu einer konjunktiven Klauselform geht Anordnungsinformation verloren. Diese ist semantisch gesehen irrelevant, da Konjunktion und Disjunktion in Booleschen Algebren assoziativ, kommutativ und idempotent sind.

Es gibt auch *disjunktive Klauselformen*. Diese interpretieren die äußere Menge disjunktiv und die inneren Mengen konjunktiv. Beispielsweise kann die Denotation der Formel

$$(X \wedge Z) \vee ((\neg X \wedge (\neg Y \wedge \neg Z)) \vee ((X \wedge \neg Y) \wedge \neg Z))$$

durch die folgende disjunktive Klauselform dargestellt werden:

$$\mathbf{D}\,\{\{X, Z\},\ \{\neg X, \neg Y, \neg Z\},\ \{X, \neg Y, \neg Z\}\}$$

$$
\begin{array}{rcll}
C, D \in \mathit{Cla} & = & \mathcal{P}_{\mathit{fin}}(\mathit{For}) & \textit{Klausel} \\
S \in \mathit{CF} & = & \mathcal{P}_{\mathit{fin}}(\mathit{Cla}) & \textit{Klauselform} \\
\mathit{For} & = & \dots & \text{Formel} \\
& | & \mathbf{K}S & \textit{konjunktive Klauselform} \\
& | & \mathbf{D}S & \textit{disjunktive Klauselform} \\
\mathcal{B}(\mathbf{K}S)\sigma & = & \bigwedge\limits_{C \in \mathbf{K}S}^{\mathcal{B}} \bigvee\limits_{A \in C}^{\mathcal{B}} \mathcal{B}(A)\sigma & \text{Denotation} \\
\mathcal{B}(\mathbf{D}S)\sigma & = & \bigvee\limits_{C \in \mathbf{K}S}^{\mathcal{B}} \bigwedge\limits_{A \in C}^{\mathcal{B}} \mathcal{B}(A)\sigma & \\
\widehat{\mathbf{K}S} & = & \mathbf{D}\{\, \{\, \widehat{A} \mid A \in C \,\} \mid C \in S \,\} & \text{Dualisierung} \\
\widehat{\mathbf{D}S} & = & \mathbf{K}\{\, \{\, \widehat{A} \mid A \in C \,\} \mid C \in S \,\} &
\end{array}
$$

Abbildung 4.10: Klauselformen

Abbildung 4.10 fasst die grundlegenden Definitionen für Klauselformen zusammen. Eine *Klausel* ist eine endliche Menge von Formeln. Eine *Klauselform* ist eine endliche Menge von Klauseln. Die Menge der Formeln erweitern wir um zwei neue Varianten $\mathbf{K}S$ und $\mathbf{D}S$, die wir als *konjunktive* und *disjunktive Klauselform* bezeichnen. Für jede Boolesche Algebra $\mathcal{B}$ erweitern wir die Denotationsfunktion wie besprochen auf die neuen Formelvarianten. Dabei wählen wir $0^{\mathcal{B}}$ als Denotation für die leere Disjunktion und $1^{\mathcal{B}}$ als Denotation für die leere Konjunktion. Damit gelten die Äqivalenzen

$$\mathbf{K}\emptyset \mathrel{\models\!\!\!\dashv} 1, \quad \mathbf{K}\{\emptyset\} \mathrel{\models\!\!\!\dashv} 0, \quad \mathbf{D}\emptyset \mathrel{\models\!\!\!\dashv} 0, \quad \mathbf{D}\{\emptyset\} \mathrel{\models\!\!\!\dashv} 1.$$

Schließlich erweitern wir die Dualisierungsfunktion $\widehat{\ } \in \mathit{For} \to \mathit{For}$ auf die neuen Formelvarianten.

Aus den Definitionen ist offensichtlich, dass jede Formel mit Klauselformen in eine Formel ohne Klauselformen übersetzt werden kann. Also gilt der Tautologiesatz 4.10.6 auch für die neuen Formelvarianten. Auch der Dualitätssatz 4.7.1 gilt für die neuen Formelvarianten. (Übung: Erweitern Sie den Beweis des Dualitätssatzes um die neuen Formelvarianten.)

*Äquivalenz von Klauselformen* definieren wir wie folgt:

$$S \mathrel{\models\!\!\!\dashv} S' \;\overset{\text{def}}{\Longleftrightarrow}\; \mathbf{K}S \mathrel{\models\!\!\!\dashv} \mathbf{K}S' \,\wedge\, \mathbf{D}S \mathrel{\models\!\!\!\dashv} \mathbf{D}S'$$

**Proposition 4.13.1** *Seien $S$ und $S'$ äquivalente Klauselformen und $C \in S$. Dann $S' \models\!\!\!\!\dashv S' \cup \{C\}$.*

**Beweis** Folgt sofort aus der Tatsache, dass in jeder Booleschen Algebra für alle $x, y, z$ gilt: wenn $xy = z$, dann $z = zy$; und wenn $x + y = z$ dann $z = z + y$. □

**Bereinigte Klauselformen**

Eine Klausel $C$ heißt *trivial*, wenn es eine Formel $A$ gibt, so dass $A \in C$ und $\neg A \in C$. Eine Klausel $C$ *subsumiert* eine Klausel $D$, wenn $C \subseteq D$. Das Löschen von trivialen und subsumierten Klauseln lässt die Denotation einer Klauselform unverändert:

**Proposition 4.13.2 (Löschen von tivialen und subsumierten Klauseln)** *Sei $S$ eine Klauselform und $C$ eine Klausel. Wenn $C$ trivial ist oder von einer Klausel in $S$ subsumiert wird, gilt: $S \cup \{C\} \models\!\!\!\!\dashv S$.*

**Beweis** Man sieht leicht, dass die folgenden Booleschen Gleichungen gelten (die zweite Gleichung folgt mit einem der Absorptionsaxiome):

$$x\overline{x}y + z = z$$
$$x + xy + z = x + z$$

Für die disjunktive Interpretation rechtfertigt die erste Gleichung das Löschen von trivialen Klauseln und die zweite Gleichung das Löschen von subsumierten Klauseln. Für die konjunktive Interpretation rechtfertigen die dualen Gleichungen

$$(x + \overline{x} + y)z = z$$
$$x(x + y)z = xz$$

das Löschen von trivialen und subsumierten Klauseln. □

**Beispiel** Mit der Proposition folgt die Äquivalenz

$$\{\{X, \neg X\},\ \{Y, Z\},\ \{Z\}\} \models\!\!\!\!\dashv \{\{Z\}\}$$

da die erste Klausel tivial ist und die zweite Klausel von der dritten Klausel subsumiert wird. □

Eine Klauselform heißt *bereinigt*, wenn sie keine trivialen und keine subsumierten Klauseln enthält.

## 4.14 Konjunktive und disjunktive Normalformen

Ein *Literal* ist eine Formel $A \in \mathit{For}$, so dass eine Variable $X \in \mathit{Var}$ existiert mit $A = X$ oder $A = \neg X$. Ein Literal kann man sich als eine mit einem Vorzeichen versehene Variable vorstellen. Zwei Literale $L_1$ und $L_2$ heißen *komplementär*, wenn $L_1 = \neg L_2$ oder $L_2 = \neg L_1$ gilt. Mit $-L$ bezeichnen wir das zu $L$ komplementäre Literal. Für jedes Literal $L$ gilt $-(-L) = L$.

Eine Klausel heißt *literal*, wenn sie nur Literale enthält. Eine Klauselform heißt *literal*, wenn sie nur literale Klauseln enthält.

**Proposition 4.14.1** *Sei $S$ eine literale Klauselform. Dann $\widehat{\mathbf{K}S} = \mathbf{D}S$ und $\widehat{\mathbf{D}S} = \mathbf{K}S$.*

**Beweis** Gilt, da $\widehat{L} = L$ für jedes Literal $L$. □

**Proposition 4.14.2 (Äquivalenz von literalen Klauselformen)** *Seien $S$ und $S'$ literale Klauselformen. Dann sind die folgenden Aussagen paarweise äquivalent:*

$$S \models\!\!\!\!\models S' \qquad \mathbf{K}S \models\!\!\!\!\models \mathbf{K}S' \qquad \mathbf{D}S \models\!\!\!\!\models \mathbf{D}S'$$

$$\mathcal{T}(\mathbf{K}S) = \mathcal{T}(\mathbf{K}S') \qquad \mathcal{T}(\mathbf{D}S) = \mathcal{T}(\mathbf{D}S')$$

**Beweis** Die Äquivalenz zwischen $\mathbf{K}S \models\!\!\!\!\models \mathbf{K}S'$ und $\mathcal{T}(\mathbf{K}S) = \mathcal{T}(\mathbf{K}S')$ folgt aus dem Tautologiesatz 4.10.6. Damit folgt auch die Äquivalenz zwischen $\mathbf{D}S \models\!\!\!\!\models \mathbf{D}S'$ und $\mathcal{T}(\mathbf{D}S) = \mathcal{T}(\mathbf{D}S')$. Die Äquivalenz zwischen $\mathbf{K}S \models\!\!\!\!\models \mathbf{K}S'$ und $\mathbf{D}S \models\!\!\!\!\models \mathbf{D}S'$ folgt aus Proposition 4.14.1 und dem Dualitätssatz 4.7.1. Aus dieser Äquivalenz folgt die Äquivalenz zwischen $S \models\!\!\!\!\models S'$ und $\mathbf{K}S \models\!\!\!\!\models \mathbf{K}S'$. □

**Proposition 4.14.3 (Triviale literale Klauselformen)** *Für jede literale Klauselform $S$ sind die folgenden Aussagen äquivalent:*

1. *Jede Klausel in $S$ ist trivial.*
2. $\mathbf{K}S \models\!\!\!\!\models 1$.
3. $\mathbf{D}S \models\!\!\!\!\models 0$.

**Beweis** Die Äquivalenz von (2) und (3) folgt mit dem Dualitätssatz und Proposition 4.14.1.

Wenn $S$ nur triviale Klauseln enthält, gilt $S \models\!\!\!\!\models \emptyset$ (Proposition 4.13.2). Also $\mathbf{K}S \models\!\!\!\!\models \mathbf{K}\emptyset \models\!\!\!\!\models 1$.

Sei $\mathbf{D}S \models\!\!\!\!\models 0$. Wir zeigen durch Widerspruch, dass $S$ nur triviale Klauseln enthält. Sei $C \in S$ eine nichttriviale Klausel. Dann existiert eine Belegung $\sigma \in Var \to \mathbb{B}$ mit $\mathcal{T}(\mathbf{D}\{C\})\sigma = 1$. Also $\mathcal{T}(\mathbf{D}S)\sigma = 1$. Das ist ein Widerspruch zu $\mathbf{D}S \models\!\!\!\!\models 0$. □

Eine Klausel heißt *normal*, wenn sie literal und nichttrivial ist. Eine Klauselform heißt *normal*, wenn sie nur normale Klauseln enthält.

Eine normale Klauselform $S$ heißt *konjunktive Normalform (KNF)* für eine Formel $A$, wenn $A \models\!\mid \mathbf{K}S$. Eine normale Klauselform $S$ heißt *disjunktive Normalform (DNF)* für eine Formel $A$, wenn $A \models\!\mid \mathbf{D}S$.

**Proposition 4.14.4** *Für jede Klauselform S und jede Formel A gilt:*

1. *S ist DNF für A genau dann, wenn S KNF für $\widehat{A}$ ist.*
2. *S ist KNF für A genau dann, wenn S DNF für $\widehat{A}$ ist.*

**Beweis** Dualitätssatz und Proposition 4.14.1. □

Wir definieren eine Funktion, die zu einem Entscheidungsbaum eine Klauselform bestimmt:

$$\begin{aligned}
&\mathit{dnf} \in DT \to CF\\
&\mathit{dnf}(0) = \emptyset\\
&\mathit{dnf}(1) = \{\emptyset\}\\
&\mathit{dnf}(X, A_0, A_1) = \{\, C \cup \{\neg X\} \mid C \in \mathit{dnf}(A_0)\,\} \cup \{\, C \cup \{X\} \mid C \in \mathit{dnf}(A_1)\,\}
\end{aligned}$$

**Proposition 4.14.5 (Berechnung einer DNF)** *Sei A ein geordneter Entscheidungsbaum. Dann ist $\mathit{dnf}(A)$ eine bereinigte DNF für A, die nur Variablen enthält, die in A vorkommen.*

**Beweis** Durch strukturelle Induktion über $A$.

$A = 0$ *oder* $A = 1$. Dann sind die Behauptungen offensichtlich.

$A = (X, A_0, A_1)$. Dann

$$(*) \quad \mathit{dnf}(A) = \{\, C \cup \{\neg X\} \mid C \in \mathit{dnf}(A_0)\,\} \cup \{\, C \cup \{X\} \mid C \in \mathit{dnf}(A_1)\,\}$$

Die Induktionsannahme liefert uns, dass $\mathit{dnf}(A_0)$ und $\mathit{dnf}(A_1)$ literale und bereinigte Klauselformen sind, in denen nur Variablen aus $A_0$ beziehungsweise $A_1$ vorkommen. Da $A$ eine geodneter Entscheidungsbaum ist, kommt $X$ weder in $\mathit{dnf}(A_0)$ noch $\mathit{dnf}(A_1)$ vor. Folglich kann das Hinzufügen von $\neg X$ beziehungsweise $X$ zu den Klauseln von $\mathit{dnf}(A_0)$ und $\mathit{dnf}(A_1)$ keine triviale Klauseln erzeugen. Also ist $\mathit{dnf}(A)$ eine normale Klauselform, in der nur Variablen aus $A$ vorkommen.

Wir überzeugen uns jetzt davon, dass $\mathit{dnf}(A)$ keine subsumierten Klauseln enthält. Seien $D_1, D_2 \in \mathit{dnf}(A)$ mit $D_1 \subseteq D_2$. Da $X$ nicht in $\mathit{dnf}(A_0)$ und $\mathit{dnf}(A_1)$ vorkommt, gilt entweder $D_1, D_2 \in \{\, C \cup \{\neg X\} \mid C \in \mathit{dnf}(A_0)\,\}$ oder $D_1, D_2 \in \{\, C \cup \{X\} \mid C \in \mathit{dnf}(A_1)\,\}$. Wir betrachten nur den zweiten Fall, der erste folgt analog. Wegen $D_1 \subseteq D_2$ muss im zweiten Fall $D_1 - \{X\} \subseteq D_2 - \{X\}$

gelten. Da $D_1 - \{X\}$ und $D_2 - \{X\}$ beide in $dnf(A_1)$ sind und $dnf(A_1)$ bereinigt ist, folgt $D_1 - \{X\} = D_2 - \{X\}$. Also $D_1 = D_2$.

Schließlich zeigen wir, dass $A \mathrel{|\!\!=\!\!|} \mathbf{D}(dnf(A))$ gilt:

$$
\begin{array}{llr}
A & \mathrel{|\!\!=\!\!|} (X, A_0, A_1) & \\
 & \mathrel{|\!\!=\!\!|} (X, \mathbf{D}(dnf(A_0)), \mathbf{D}(dnf(A_1))) & \text{Induktionsannahme} \\
 & \mathrel{|\!\!=\!\!|} \neg X \wedge \mathbf{D}(dnf(A_0)) \;\vee\; X \wedge \mathbf{D}(dnf(A_1)) & \\
 & \mathrel{|\!\!=\!\!|} \mathbf{D}\{\, C \cup \{\neg X\} \mid C \in dnf(A_0)\,\} \;\vee\; \mathbf{D}\{\, C \cup \{X\} \mid C \in dnf(A_1)\,\} & \\
 & \mathrel{|\!\!=\!\!|} \mathbf{D}(\{\, C \cup \{\neg X\} \mid C \in dnf(A_0)\,\} \cup \{\, C \cup \{X\} \mid C \in dnf(A_1)\,\}) & \\
 & \mathrel{|\!\!=\!\!|} \mathbf{D}(dnf(X, A_0, A_1)) & (*) \\
 & \mathrel{|\!\!=\!\!|} \mathbf{D}(dnf(A)) &
\end{array}
$$

□

**Beispiel** Sei $A$ der Primbaum

```
      X
     / \
    0   Y
       / \
      1   Z
         / \
        1   0
```

Dann $dnf(A) = \{\{X, \neg Y\},\ \{X, Y, \neg Z\}\}$. □

**Proposition 4.14.6 (Berechnung einer KNF)** *Sei $A$ ein geordneter Entscheidungsbaum. Dann ist $dnf(\widehat{A})$ eine bereinigte KNF für $A$, die nur Variablen enthält, die in $A$ vorkommen.*

**Beweis** Zunächst stellen wir fest, dass $\widehat{A}$ wieder ein geordneter Entscheidungsbaum ist. Also folgt mit Proposition 4.14.5, dass $dnf(\widehat{A})$ eine bereinigte DNF für $\widehat{A}$ ist, die nur Variablen enthält, die in $A$ vorkommen. Jetzt folgt die Behauptung mit Proposition 4.14.4. □

## 4.15 Resolution und Primformen

Eine Klausel $D$ heißt *Resolvente von zwei Klauseln $C_1$ und $C_2$*, wenn eine Formel $A$ existiert mit:

1. $A \in C_1$ und $\neg A \in C_2$.
2. $D = (C_1 - \{A\}) \cup (C_2 - \{\neg A\})$.

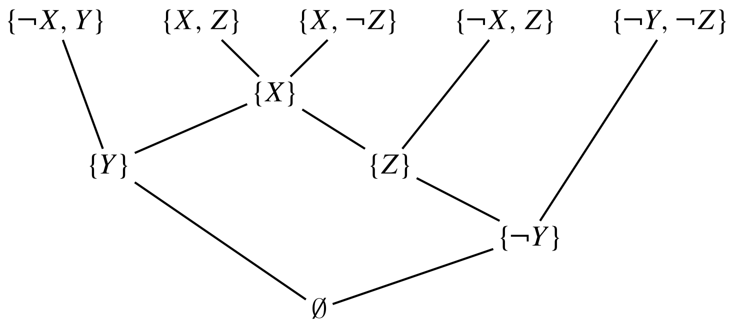


Abbildung 4.11: Ein Resolutionsgraph

Eine Klausel $D$ heißt *Resolvente einer Klauselform $S$*, wenn $C$ Resolvente von zwei Klauseln in $S$ ist. Das Hinzufügen von Resolventen lässt die Denotationen einer Klauselform unverändert:

**Proposition 4.15.1 (Hinzufügen von Resolventen)** *Sei $S$ eine Klauselform und $C$ eine Resolvente von $S$. Dann $S \mathrel{\models\!\!\!\dashv} S \cup \{C\}$.*

**Beweis** Mit Proposition 4.1.2 (1) folgt, dass

$$xy + \overline{x}z = xy + \overline{x}z + yz$$

eine gültige Boolesche Gleichung ist. Damit ist das Hinzufügen von Resolventen für die disjunktive Interpretation gerechtfertigt. Für die konjunktive Interpretation ist das Hinzufügen von Resolventen durch die duale Gleichung gerechtfertigt:

$$(x + y)(\overline{x} + z) = (x + y)(\overline{x} + z)(y + z)$$ □

**Beispiel** Abbildung 4.11 zeigt mithilfe eines sogenannten *Resolutionsgraphen*, wie eine Klauselmenge

$$S \;=\; \{\{\neg X, Y\},\; \{X, Z\},\; \{X, \neg Z\},\; \{\neg X, Z\},\; \{\neg Y, \neg Z\}\}$$

schrittweise um insgesamt fünf Resolventen erweitert wird. Da schließlich die leere Klausel hinzugefügt wird, die alle Klausel subsumiert, haben wir mit den Propositionen 4.15.1 und 4.13.2: $S \mathrel{\models\!\!\!\dashv} \{\emptyset\}$. □

Eine Klausel $C$ heißt *echte Resolvente einer Klauselform $S$*, wenn gilt: (1) $C$ ist Resolvente von $S$; (2) $C$ ist nicht trivial; (3) $C$ wird von keiner Klausel in $S$ subsumiert. Machen Sie sich klar, dass der Resolutionsgraph in Abbildung 4.11 nur echte Resolventen hinzufügt.

Eine Klauselmenge heißt *saturiert* genau dann, wenn sie keine echte Resolvente hat.

**Proposition 4.15.2 (Löschen von tivialen und subsumierten Klauseln)**
*Seien $S$ und $S'$ Klauselformen und sei $S'$ aus $S$ durch das Löschen einer tivialen oder subsumierten Klausel erhalten. Dann ist $S$ genau dann saturiert, wenn $S'$ saturiert ist.*

Eine Klauselform heißt *Primform*, wenn sie literal, bereinigt und saturiert ist. Eine Primform $S$ heißt *konjunktive Primform (KPF)* für eine Formel $A$, wenn $A \models\!\!\!\!\models \mathbf{K}S$. Eine Primform $S$ heißt *disjunktive Primform (DPF)* für eine Formel $A$, wenn $A \models\!\!\!\!\models \mathbf{D}S$.

Wir werden zeigen, dass zu jeder Formel genau eine konjunktive und genau eine disjunktive Primform existiert.

Primformen haben Anwendungen beim Schaltkreisentwurf (minimale disjunktive Normalformen) und in der Künstlichen Intelligenz (Truth Maintenance Systems).[12]

**Beispiel** Betrachten Sie die Klauselform $S = \{\{X, \neg Y\},\ \{X, Y, \neg Z\}\}$. Da $\{X, \neg Z\}$ eine echte Resolvente von $S$ ist, ist $S$ nicht saturiert. Hinzufügen der echten Resolvente und Löschen der subsumierten Klausel liefert die äquivalente Primform $\{\{X, \neg Y\},\ \{X, \neg Z\}\}$. □

**Beispiel** Betrachten Sie die Formel[13]

$$A = (\neg X \Rightarrow Z) \ \wedge\ (Z \wedge X \Rightarrow \neg Y) \ \wedge\ (Y \vee \neg X \Rightarrow \neg Z)$$

Wir wollen die konjunktive und die disjunktive Primform für $A$ bestimmen.

Zunächst bestimmen wir eine KNF für $A$. Dazu bestimmen wir für jede der drei Teilformeln von $A$ mithilfe von Äquivalenzumformungen eine KNF:

$$\begin{aligned}
\neg X \Rightarrow Z &\models\!\!\!\!\models \neg\neg X \vee Z \models\!\!\!\!\models X \vee Z \\
Z \wedge X \Rightarrow \neg Y &\models\!\!\!\!\models \neg(Z \wedge X) \vee \neg Y \models\!\!\!\!\models \neg X \vee \neg Y \vee \neg Z \\
Y \vee \neg X \Rightarrow \neg Z &\models\!\!\!\!\models \neg(Y \vee \neg X) \vee \neg Z \models\!\!\!\!\models (\neg Y \wedge X) \vee \neg Z \\
&\models\!\!\!\!\models (\neg Y \vee \neg Z) \wedge (X \vee \neg Z)
\end{aligned}$$

Damit haben wir eine KNF für $A$:

$$A \models\!\!\!\!\models \mathbf{K}\,\{\{X, Z\},\ \{\neg X, \neg Y, \neg Z\},\ \{\neg Y, \neg Z\},\ \{X, \neg Z\}\}$$

[12] Es gibt kein Lehrbuch, das Primformen systematisch behandelt. Eine interessante Arbeit über die Berechnung von Primformen ist: Alex Kean und George Tsiknis, An Incremental Method for Generating Prime Implicants/Implicates. Journal of Symbolic Computation (1990) 9, 185–206.

[13] Die Formel entspricht der Konjunktion der Diätregeln in Abschnitt 4.5, wobei wir statt $B$, $E$ und $F$ die Buchstaben $X$, $Y$ und $Z$ für die Variablen benutzen.

Wir löschen die zweite Klausel, da sie von der dritten Klausel subsumiert wird, und fügen die Resolvente $\{X\}$ der ersten und letzten Klausel hinzu:

$$A \;\mathrel{\models\!\!\!\dashv}\; \mathbf{K}\,\{\{X, Z\},\ \{\neg Y, \neg Z\},\ \{X, \neg Z\},\ \{X\}\}$$

Jetzt löschen wir die von $\{X\}$ subsumierten Klauseln und bekommen:

$$A \;\mathrel{\models\!\!\!\dashv}\; \mathbf{K}\,\{\{X\},\ \{\neg Y, \neg Z\}\}$$

Da diese Klauselform literal, bereinigt und saturiert ist, handelt es sich um die konjunktive Primform für $A$.

Die gerade vorgeführte Bestimmung der KPF aus der anfänglichen KNF kann durch den Resolutionsgraphen

$$\{X, Z\}\,✓ \quad \{\neg X, \neg Y, \neg Z\}\,✓ \quad \{\neg Y, \neg Z\} \quad \{X, \neg Z\}\,✓$$

$$\{X\}$$

übersichtlich dargestellt werden. Subsumierte Klauseln werden dabei durch das Symbol ✓ markiert.

Die Bestimmung einer DPF für $A$ ist besonders einfach, wenn wir von der KPF für $A$ ausgehen. Mit dem Distributivitätsgesetz bekommen wir:

$$A \;\mathrel{\models\!\!\!\dashv}\; \mathbf{K}\,\{\{X\},\ \{\neg Y, \neg Z\}\} \;\mathrel{\models\!\!\!\dashv}\; \mathbf{D}\,\{\{X, \neg Y\},\ \{X, \neg Z\}\}$$

Da die rechts stehende Klauselform literal, bereinigt und saturiert ist, handelt es sich um die DPF für $A$. □

Zu einer normalen Klauselform können wir eine äquivalente Primform bestimmen, indem wir solange subsumierte Klauseln löschen und echte Resolventen hinzufügen, bis keiner dieser beiden Schritte mehr möglich ist. Wir zeigen jetzt, dass dieser Bereinigungs- und Saturationsprozess immer terminiert. Dazu benutzen wir die folgenden Argumente:

1. Das Hinzufügen einer echten Resolvente vergrößert die Menge der von den Klauseln der Klauselform subsumierten Klauseln (mindestens um die hinzugefügte Resolvente).

2. Das Löschen von subsumierten Klauseln lässt die Menge der subsumierten Klauseln unverändert.

3. Zu einer Klauselform können auch durch wiederholtes Bilden von Resolventen nur endlich viele Resolventen gebildet werden, da alle Resolventen nur die endlich vielen Formeln enthalten können, die in der Klauselform enthalten sind.

Seien $S$ und $S'$ zwei Klauselformen. Wir schreiben $S \xrightarrow{r} S'$ genau dann, wenn $S'$ aus $S$ durch das Löschen einer subsumierten Klausel oder durch das Hinzufügen einer echten Resolvente erhalten werden kann.

**Lemma 4.15.3 (Saturation)** *Sei $S$ eine normale Klauselform. Dann:*

1. *Wenn $S \xrightarrow{r} S'$, dann $S \models\!\!\mid S'$, $VV(S') \subseteq VV(S)$ und $S'$ ist normal.*
2. *$S$ ist eine Primform genau dann, wenn es kein $S'$ mit $S \xrightarrow{r} S'$ gibt.*
3. *Es gibt keine unendliche Kette $S \xrightarrow{r} S_1 \xrightarrow{r} S_2 \xrightarrow{r} \cdots$.*

**Beweis** Die erste Behauptung folgt mit den Propositionen 4.13.2 und 4.15.1. Die zweite Behauptung ist offensichtlich. Die dritte Behauptung zeigen wir durch Widerspruch.

Sei $S$ eine Klauselform, für die eine unendliche Kette

$$S \xrightarrow{r} S_1 \xrightarrow{r} S_2 \xrightarrow{r} S_3 \xrightarrow{r} \cdots$$

existiert. Wir definieren

$$Cla_S \stackrel{\text{def}}{=} \mathcal{P}(\bigcup_{C \in S} C)$$

$Cla_S$ ist die Menge aller Klauseln, die man mit den in den Klauseln von $S$ enthalten Formeln bilden kann. Aus der Definition von $\xrightarrow{r}$ folgt:

$$\forall i \in \mathbb{N}^+ : \; S_i \subseteq Cla_S$$

Für jede Klauselform $S_i$ der Kette definieren wir

$$Sub_i \stackrel{\text{def}}{=} \{\, C \in Cla_S \mid \exists D \in S_i : \; D \subseteq C \,\}$$

Wir haben:

- $Sub_i = Sub_{i+1}$, wenn $S_{i+1}$ aus $S_i$ durch Löschen einer subsumierten Klausel erhalten wird.
- $Sub_i \subsetneq Sub_{i+1}$, wenn $S_{i+1}$ aus $S_i$ durch Hinzufügen einer echten Resolvente erhalten wird.
- $Sub_1 \subseteq Sub_2 \subseteq Sub_3 \subseteq \; \cdots \; \subseteq Cla_S$.

Da $Cla_S$ endlich ist, kann die Kette nur endlich viele Resolventen hinzufügen. Also werden ab einem bestimmten $S_k$ nur noch subsumierte Klauseln gelöscht. Da $S_k$ endlich ist, können aber nur endlich viele Klauseln gelöscht werden. Damit haben wir einen Widerspruch zu der Annahme, dass die Kette unendlich ist. □

**Lemma 4.15.4 (Berechnung von Primformen)** *Zu jeder normalen Klauselform $S$ kann eine äquivalente Primform berechnet werden, die nur Variablen enthält, die in $S$ vorkommen.*

**Beweis** Folgt sofort aus dem Saturationslemma 4.15.3. □

**Lemma 4.15.5 (Berechnung von Primformen)** *Zu jeder Formel $A$ kann eine konjunktive Primform [disjunktive Primform] berechnet werden, die nur Variablen enthält, die in $A$ vorkommen.*

**Beweis** Sei $A$ eine Formel. Dann bestimmen wir zuerst den zu $A$ äquivalenten Primbaum $\pi A$ (Primbaumsatz 4.10.4). Aus $\pi A$ berechnen wir eine DNF und eine KNF für $A$ (Propositionen 4.14.5 und 4.14.6). Zur DNF und KNF bestimmen wir jeweils eine äquivalente Primform (Lemma 4.15.4). □

## 4.16 Eigenschaften von Primformen

Wir wissen bereits, dass für jede Formel eine disjunktive und eine konjunktive Primform existieren. Wir zeigen jetzt, dass diese Primformen eindeutig sind. Die Eindeutigkeit folgt aus dem sogenannten Resolutionssatz, der eine weitere interessante Eigenschaft von Primformen formuliert.

In diesem Abschnitt verwenden wir die folgende Hilfsnotation:

$$\mathcal{K}(S) \stackrel{\text{def}}{=} \{\sigma \in \mathit{Var} \to \mathbb{B} \mid \mathcal{T}(\mathbf{K}S)\sigma = 1\}$$

**Lemma 4.16.1** *Sei $S$ eine normale Klauselform, $C$ eine normale Klausel, und*

$$S' = \{ D - C \mid D \in S \text{ und } \forall L \in C\colon\ -L \notin D \}$$

*Dann:*

1. *$S'$ und $C$ haben keine Variablen gemeinsam.*
2. *$\mathcal{K}(S') \neq \emptyset \;\Rightarrow\; \mathcal{K}(S) \neq \emptyset$.*
3. *$S'$ nicht saturiert $\;\Rightarrow\;$ $S$ nicht saturiert.*

**Beweis** Die erste Behauptung ist offensichtlich.

Für die zweite Behauptung nehmen wir an, dass $\sigma' \in \mathcal{K}(S')$ gilt. Da $S'$ und $C$ keine Variablen gemeinsam haben, gibt es eine Belegung $\sigma$, die für die Variablen in $S'$ mit $\sigma'$ übereinstimmt und $\forall L \in C\colon\ \mathcal{T}(-L)\sigma = 1$ erfüllt. Also gilt $\sigma \in \mathcal{K}(S)$.

Für die dritte Behauptung nehmen wir an, dass $S'$ eine echte Resolvente hat. Wir gehen in zwei Schritten von $S'$ zu $S$ zurück und zeigen jeweils, dass die Existenz einer echten Resolvente erhalten bleibt.

Zuerst fügen wir die beim Übergang von $S$ zu $S'$ gelöschten Literale aus $C$ wieder zu den Klauseln von $S'$ hinzu. Nach dem Hinzufügen der Literale kann die Resolvente immer noch gebildet werden. Sie kann jetzt nicht trivial sein, da die neuen Literale nur bisher nicht vorkommende Variablen enthalten und diese nur mit einem Vorzeichen einführen. Die Wiedereinführung der Literale kann auch zu keiner Subsumtion der Resolvente führen (da Löschen von Literalen in allen Klauseln Subsumtion erhält).

Im zweiten Schritt fügen wir wieder die Klauseln hinzu, die Komplemente von Literalen von $C$ enthalten. Die neuen Klauseln können die Resolvente nicht subsumieren, da sie jeweils mindestens ein Literal enthalten, dass bisher nicht vorkam. Also ist $S$ nicht saturiert. □

**Lemma 4.16.2** *Sei $S$ eine saturierte und normale Klauselform mit $\mathcal{K}(S) = \emptyset$. Dann $\emptyset \in S$.*

**Beweis** Durch Induktion über die Anzahl $n$ der in $S$ vorkommenden Variablen.

*Sei $n = 0$.* Die leere Klausel ist die einzige Klausel, die keine Variablen enthält. Da $\mathcal{K}(\emptyset) = (Var \to \mathbb{B}) \neq \emptyset$, folgt $\emptyset \in S$.

*Sei $n > 0$.* Sei $X$ eine Variable, die in $S$ vorkommt. Wir definieren $S_1$ und $S_2$ wie folgt:

$$S_1 = \{\, C - \{X\} \mid C \in S,\ \neg X \notin C \,\}$$
$$S_2 = \{\, C - \{\neg X\} \mid C \in S,\ X \notin C \,\}$$

Mit $\mathcal{K}(S) = \emptyset$ und Lemma 4.16.1 folgt, dass $\mathcal{K}(S_1) = \mathcal{K}(S_2) = \emptyset$, und dass $S_1$ und $S_2$ beide saturiert sind. Also gilt nach Induktionsannahme $\emptyset \in S_1$ und $\emptyset \in S_2$. Folglich enthält $S$ die Klauseln $\{X\}$ und $\{\neg X\}$. Da $S$ saturiert ist, muss $S$ die Resolvente $\emptyset$ enthalten. □

**Lemma 4.16.3** *Sei $S$ eine saturierte und normale Klauselform und $C$ eine normale Klausel mit $\mathcal{K}(S) \subseteq \mathcal{K}(\{C\})$. Dann wird $C$ von einer Klausel in $S$ subsumiert.*

**Beweis** Sei $S' = \{\, D - C \mid D \in S \text{ und } \forall L \in C\colon\ -L \notin D \,\}$. Mit Lemma 4.16.1 folgt, dass $S'$ saturiert ist. Sei $S'' = S \cup \{\, \{-L\} \mid L \in C \,\}$. Da $\mathcal{K}(S) \subseteq \mathcal{K}(\{C\})$, folgt $\mathcal{K}(S'') = \emptyset$. Da $S' = \{\, D - C \mid D \in S'' \text{ und } \forall L \in C\colon\ -L \notin D \,\}$, folgt mit Lemma 4.16.1, dass $\mathcal{K}(S') = \emptyset$. Also $\emptyset \in S'$ mit Lemma 4.16.2. Also existiert eine Klausel $D \in S$ mit $D \subseteq C$. □

**Satz 4.16.4 (Resolution)** *Seien $S$ und $S'$ zwei äquivalente normale Klauselformen. Wenn $S$ eine Primform ist, dann wird jede Klausel in $S'$ von einer Klausel in $S$ subsumiert.*

**Beweis** Sei $C \in S'$. Dann $\mathcal{K}(S) = \mathcal{K}(S') \subseteq \mathcal{K}(\{C\})$. Also folgt mit Lemma 4.16.3, dass $C$ von einer Klausel in $S$ subsumiert wird. □

**Satz 4.16.5 (Primformen)** *Zu jeder normalen Klauselform existiert genau eine äquivalente Primform. Diese Primform enthält nur Variablen, die in der Formel vorkommen.*

**Beweis** Die Existenz von Primformen haben wir bereits gezeigt (Lemma 4.15.4). Wir zeigen jetzt die Eindeutigkeit.

Seien $S$ und $S'$ zwei äquivalente Primformen. Wir zeigen $S \subseteq S'$. Die Aussage $S' \subseteq S$ folgt analog.

Sei $C \in S$. Der Resolutionssatz 4.16.4 liefert eine Klausel $C' \in S'$ mit $C' \subseteq C$. Nochmaliges Anwenden des Resolutionssatzes liefert eine Klausel $C'' \in S$ mit $C'' \subseteq C'$. Da $C'' \subseteq C' \subseteq C$ und $C''$ und $C$ beide in $S$ sind, folgt $C'' = C$, da $S$ bereinigt ist. Also $C = C' \in S'$. □

**Satz 4.16.6 (Primformen)** *Für jede Formel existiert genau eine konjunktive und genau eine disjunktive Primform. Diese Primformen enthalten nur Variablen, die in der Formel vorkommen.*

**Beweis** Die Existenz von Primformen haben wir bereits gezeigt (Lemma 4.15.5). Die Eindeutigkeit folgt mit Proposition 4.14.2 und Satz 4.16.5. □

**Korollar 4.16.7 (Signifikante Variablen)** *Sei $S$ die DPF oder KPF für eine Formel $A$. Dann $SV(A) = VV(S)$.*

### Minimalformen

Eine normale Klauselform $S$ heißt *Minimalform*, wenn

1. $\forall C \in S\colon\ S - \{C\} \not\models\!\!\mid S$.
2. $\forall C \in S\ \ \forall D \subsetneq C\colon\ (S - \{C\}) \cup \{D\} \not\models\!\!\mid S$.

**Proposition 4.16.8** *Sei $S$ eine Minimalform. Dann ist $S$ eine Teilmenge der zu $S$ äquivalenten Primform.*

**Beweis** Sei $S'$ die zu $S$ äquivalente Primform und sei $C \in S$. Der Resolutionssatz liefert eine Klausel $D \in S'$ mit $D \subseteq C$. Wir zeigen durch Widerspruch, dass

$D = C$. Sei also $D \neq C$. Mit Proposition 4.13.1 folgt $S \models\!\mid S \cup \{D\}$. Da $D$ die Klausel $C \in S$ subsumiert, gilt $S \models\!\mid (S - \{C\}) \cup \{D\}$ (Proposition 4.13.2). Das ist ein Widerspruch zur Annahme, dass $S$ eine Minimalform ist. □

Gegeben eine normale Klauselform $S$, können wir eine zu $S$ äquivalente Minimalform wie folgt bestimmen. Zuerst bestimmen wir die Primform zu $S$. Aus der Primform löschen wir dann schrittweise solche Klauseln, die durch Resolution wieder hinzugefügt werden können. Proposition 4.16.8 und der Primformsatz 4.16.5 garantieren, dass wir auf diese Weise alle zu $S$ äquivalenten Minimalformen finden.

**Beispiel** Wir betrachten die Primform:

$$S = \{\{\neg X, Z\},\ \{\neg Y, Z\},\ \{\neg X, Y\},\ \{Y, \neg Z\},\ \{X, \neg Z\},\ \{X, \neg Y\}\}$$

Proposition 4.16.8 schränkt den Suchraum für die zu $S$ äquivalente Minimalformen auf die Teilmengen von $S$ ein ($2^6$ viele). Man überzeugt sich leicht davon, dass

$$S_1 = \{\{\neg Y, Z\},\ \{\neg X, Y\},\ \{X, \neg Z\}\}$$
$$S_2 = \{\{\neg X, Z\},\ \{Y, \neg Z\},\ \{X, \neg Y\}\}$$

zwei zu $S$ äquivalente Minimalformen sind (Hinzufügen von Resolventen führt zur Primform zurück, Löschen von Klauseln eliminiert diese Möglichkeit). Es gibt weitere zu $S$ äquivalente Minimalformen, diese enthalten allerdings 4 Klauseln. Hier ist eine Beispiel:

$$S_3 = \{\{\neg Y, Z\},\ \{\neg X, Y\},\ \{Y, \neg Z\},\ \{X, \neg Y\}\}$$ □

## 4.17 Kreuzmultiplikation

Mithilfe von Kreuzmultiplikation kann man zu einer KNF eine DNF und zu einer DNF eine KNF bestimmen. Zudem hat Kreuzmultiplikation die nützliche Eigenschaft, dass sie eine saturierte Klauselform liefert, wenn die Ausgangsform keine trivialen Klauseln enthält.

Sei $S = \{C_1, \ldots, C_n\}$ eine Klauselform. Wir definieren das *Kreuzprodukt* von $S$ wie folgt:

$$\vec{S} \overset{\text{def}}{=} \{\, \{A_1, \ldots, A_n\} \mid A_1 \in C_1 \wedge \cdots \wedge A_n \in C_n \,\}$$

Damit bekommen wir $\vec{\emptyset} = \{\emptyset\}$ und $\overrightarrow{S \cup \{\emptyset\}} = \emptyset$. Die Bildung des Kreuzprodukts $\vec{S}$ wird besonders anschaulich, wenn man sich die Klauseln von $S$ als die

 

Spalten einer Matrix $C_1 \dots C_n$ vorstellt. Die Klauseln des Kreuzprodukts $\vec{S}$ entsprechen dann den Pfaden, die man von links nach rechts durch die Matrix ziehen kann, wenn man in jeder Spalte genau eine Formel besuchen muss.[14]

**Beispiel** Betrachten Sie die Klauselform

$$S = \{\{X, \neg Y\},\ \{X, Y, \neg Z\}\}$$

Durch kreuzmultiplizieren bekommen wir

$$\vec{S} = \{\{X\},\ \{X, Y\},\ \{X, \neg Z\},\ \{\neg Y, X\},\ \{\neg Y, Y\},\ \{\neg Y, \neg Z\}\}$$

Wir stellen fest, das $\vec{S}$ saturiert ist, obwohl $S$ nicht saturiert ist. Wenn wir $\vec{S}$ bereinigen (das heißt triviale und subsumierte Klauseln löschen), bekommen wir eine zu $\vec{S}$ äquivalente Klauselform

$$S' = \{\{X\},\ \{\neg Y, \neg Z\}\}$$

Kreuzmultiplizieren von $S'$ liefert die zu $S$ äquivalente Primform

$$\overrightarrow{S'} = \{\{X, \neg Y\},\ \{X, \neg Z\}\} \qquad \square$$

**Proposition 4.17.1** *Sei $C \in S$. Dann $\vec{S} = \{\, D \cup \{A\} \mid D \in (\overrightarrow{S - \{C\}}) \ \wedge\ A \in C \,\}$.*

**Beweis** Sei $S = \{C_1, \dots, C_n\} \cup \{C\}$ mit $n \geq 0$. Dann:

$$\begin{aligned}
\vec{S} &= \{\, \{A_1, \dots, A_n, A\} \mid A_1 \in C_1 \wedge \cdots \wedge A_n \in C_n \wedge A \in C \,\} \\
&= \{\, D \cup \{A\} \mid D \in \{\, \{A_1, \dots, A_n\} \mid A_1 \in C_1 \wedge \cdots \wedge A_n \in C_n \,\} \wedge A \in C \,\} \\
&= \{\, D \cup \{A\} \mid D \in (\overrightarrow{S - \{C\}}) \ \wedge\ A \in C \,\}
\end{aligned} \qquad \square$$

**Proposition 4.17.2** *Für jede Klauselform $S$ gilt: $\mathbf{D}\vec{S} \models\!\!\!\!\;\dashv \mathbf{K}S$ und $\mathbf{K}\vec{S} \models\!\!\!\!\;\dashv \mathbf{D}S$.*

**Beweis** Wir zeigen $\mathbf{D}\vec{S} \models\!\!\!\!\;\dashv \mathbf{K}S$ durch Induktion über die Anzahl der Klauseln in $S$. Sei $n = 0$. Dann $S = \emptyset$ und $\vec{S} = \{\emptyset\}$. Also $\mathbf{D}\vec{S} = \mathbf{D}\{\emptyset\} \models\!\!\!\!\;\dashv 1 \models\!\!\!\!\;\dashv \mathbf{K}\emptyset \models\!\!\!\!\;\dashv \mathbf{K}S$.

Sei $n > 0$. Dann existiert eine Klausel $C \in S$. Also folgt mit Proposition 4.17.1:

$$\begin{aligned}
\mathbf{D}\vec{S} &= \mathbf{D}\{\, D \cup \{A\} \mid D \in (\overrightarrow{S - \{C\}}) \ \wedge\ A \in C \,\} \\
&\models\!\!\!\!\;\dashv \bigvee_{A \in C} (\mathbf{D}(\overrightarrow{S - \{C\}}) \wedge A) \\
&\models\!\!\!\!\;\dashv \bigvee_{A \in C} (\mathbf{K}(S - \{C\}) \wedge A) && \text{Induktionsannahme} \\
&\models\!\!\!\!\;\dashv \mathbf{K}(S - \{C\}) \wedge \bigvee_{A \in C} A \\
&\models\!\!\!\!\;\dashv \mathbf{K}S
\end{aligned}$$

Die Behauptung $\mathbf{K}\vec{S} \models\!\!\!\!\;\dashv \mathbf{D}S$ folgt analog. $\square$

[14] Diese Metapher liegt Bibels Matrixkalkül zugrunde.

**Proposition 4.17.3** *Wenn $S$ eine Klauselform ohne triviale Klauseln ist, dann ist $\vec{S}$ saturiert.*

**Beweis** Sei $S$ eine Klauselform ohne triviale Klauseln. Seien $D_1, D_2 \in \vec{S}$, $A \in D_1$ und $\neg A \in D_2$, sodass die Resolvente $R = (D_1 - \{A\}) \cup (D_2 - \{\neg A\})$ nicht trivial ist. Wir müssen zeigen, dass $R$ von einer Klausel in $\vec{S}$ subsumiert wird.

Wenn $D_1$ eine triviale Klausel ist, ist $\neg A \in D_1$, da $R$ nicht trivial ist. Also wird $R$ von $D_2$ subsumiert.

Wenn $D_2$ eine triviale Klausel ist, folgt analog, dass $R$ von $D_1$ subsumiert wird.

Seien jetzt $D_1$ und $D_2$ beide nicht trivial. Wir betrachten die Klauseln von $S$, die die Formel $A$ zu $D_1$ beitragen. Da $D_2$ die Formel $A$ nicht enthält ($D_2$ wäre sonst trivial), müssen diese Klauseln jeweils eine von $A$ und $\neg A$ verschiedene Formel zu $D_2$ beitragen (da $S$ keine trivialen Klauseln enthält). Jetzt bilden wir eine Klausel $D \in \vec{S}$, indem wir wie bei $D_1$ vorgehen, aber statt $A$ jeweils die Formel nehmen, dass für $D_2$ verwendet wurde. Da $D$ nur Formeln enthält, die entweder in $D_1$ oder $D_2$ vorkommen und von $A$ und $\neg A$ verschieden sind, gilt $D \subseteq R$. (Die Idee für den letzten Beweisschritt kam Guido Tack am 12. Mai 2001 in der Sonne.) $\square$

 