

**Beispiel 4.9 (Streichholzspiel).** Ein Beispiel für eine Instanz des Folgerungsproblems ist die Frage nach einer Gewinnstrategie für folgendes Streichholzspiel. Gegeben sind  $n$  Streichhölzer, wobei  $n$  eine ganze Zahl  $\geq 3$  ist. Spielerin A und Spieler B ziehen abwechselnd, beginnend mit Spielerin A. Jeder Spielzug besteht darin, entweder ein oder zwei Streichhölzer zu entfernen. Verlierer ist, wer das letzte Streichholz entfernt. Gefragt ist, ob Spielerin A eine Gewinnstrategie hat, d.h., ob es Zugmöglichkeiten für die initiale Konfiguration (mit  $n$  Streichhölzern) und alle folgenden Konfigurationen, in denen Spielerin A ziehen muss, gibt, so dass Spielerin A gewinnt, unabhängig davon, wie sich ihr Gegenspieler B verhält. Wir verwenden Aussagensymbole  $x_1, \dots, x_n$ , wobei  $x_i$  die intuitive Bedeutung hat, dass Spielerin A eine Gewinnstrategie für die Konfiguration mit genau  $i$  Streichhölzern hat. Sei  $\alpha_n$  folgende Formel:

$$\alpha_n \stackrel{\text{def}}{=} \neg x_1 \wedge x_2 \wedge \bigwedge_{3 \leq i \leq n} (x_i \leftrightarrow \neg x_{i-1} \vee \neg x_{i-2}).$$

Intuitiv steht das negative Literal  $\neg x_1$  für die Tatsache, dass Spielerin A verliert, wenn nur noch ein Streichholz vorhanden ist und sie den nächsten Zug machen muss. Für zwei Streichhölzer hat sie jedoch eine Gewinnstrategie: sie entfernt ein Streichholz und versetzt Spieler B damit in die Lage, das letzte Streichholz entfernen zu müssen. Für  $i \geq 3$  Streichhölzer liegt für Spielerin A genau dann eine Gewinnstrategie vor, wenn durch die Wegnahme von einem oder zwei Streichhölzern eine Konfiguration mit  $i-1$  bzw.  $i-2$  Streichhölzern erreicht werden kann, für welche es keine Gewinnstrategie gibt. Dies erklärt die Teilformel  $x_i \leftrightarrow \neg x_{i-1} \vee \neg x_{i-2}$ . Es gilt nun:

$$\left. \begin{array}{l} \text{Spielerin A hat eine Gewinnstrategie} \\ \text{für die Konfiguration mit } n \text{ Streichhölzern} \end{array} \right\} \text{gdw } \alpha_n \Vdash x_n$$

Durch Induktion nach  $n$  kann nun gezeigt werden, dass  $\alpha_n \Vdash x_n$  genau dann, wenn  $n \bmod 3 \neq 1$ , also wenn

$$n \in \{2, 3, 5, 6, 8, 9, 11, 12, \dots\} = \{3k+2, 3k+3 : k \in \mathbb{N}\}.$$

Die Gewinnstrategie sieht wie folgt aus. Liegen  $3k+3$  Streichhölzer auf dem Tisch, so entferne zwei Streichhölzer. Damit ergibt sich die Konfiguration mit  $3k+1$  Streichhölzern, in der Spieler B keine Gewinnstrategie hat. Liegen  $3k+2$  Streichhölzer auf dem Tisch, so entferne ein Streichholz, was ebenfalls zur Konfiguration mit  $3k+1$  Streichhölzern führt. ■

Ist  $\mathfrak{F}$  eine Formelmenge und  $\alpha$  eine Formel, so gilt:

$$\begin{array}{l} \mathfrak{F} \Vdash \alpha \\ \text{gdw } \text{jedes Modell für } \mathfrak{F} \text{ ist ein Modell für } \alpha \\ \text{gdw } \text{für jede Belegung } I \text{ gilt: ist } \beta^I = 1 \text{ für alle } \beta \in \mathfrak{F}, \text{ so ist } (\neg \alpha)^I = 0 \\ \text{gdw } \mathfrak{F} \cup \{\neg \alpha\} \text{ hat kein Modell} \\ \text{gdw } \mathfrak{F} \cup \{\neg \alpha\} \text{ ist unerfüllbar} \end{array}$$

Damit erhalten wir folgendes Lemma, welches zeigt, dass logische Folgerbarkeit auf einen Unerfüllbarkeitstest zurückgeführt werden kann. Von dieser Beobachtung machen die Interpreten von Logikprogrammiersprachen Gebrauch. In diesem Zusammenhang steht  $\mathfrak{F}$  für das Logikprogramm und  $\alpha$  für die Frage. Statt eines expliziten Tests, ob  $\alpha$  eine Konsequenz von  $\mathfrak{F}$  ist, wird die Unerfüllbarkeit der Formelmenge geprüft, die aus dem Logikprogramm und der Negation der Frage besteht.

**Lemma 4.10 (Folgerungsrelation und Unerfüllbarkeit).** Sei  $\mathfrak{F}$  eine Formelmenge und  $\alpha$  eine Formel. Dann gilt  $\mathfrak{F} \Vdash \alpha$  genau dann, wenn die Formelmenge  $\mathfrak{F} \cup \{\neg\alpha\}$  unerfüllbar ist.

Ist nun  $\mathfrak{F}$  eine unerfüllbare Formelmenge, dann ist auch  $\mathfrak{F} \cup \{\neg\alpha\}$  für jede Formel  $\alpha$  unerfüllbar. Aus Lemma 4.10 folgt daher:

Ist  $\mathfrak{F}$  unerfüllbar, so gilt  $\mathfrak{F} \Vdash \alpha$  für jede Formel  $\alpha$ .

So gilt z.B.  $\{x \vee \neg y, \neg x, y \wedge z\} \Vdash \neg z$ . Für einelementige Formelmengen, etwa  $\mathfrak{F} = \{\beta\}$ , kann die Aussage von Lemma 4.10 wie folgt umformuliert werden:

$$\beta \Vdash \alpha \quad \text{gdw} \quad \beta \wedge \neg\alpha \text{ ist unerfüllbar}$$

**Zusammenhänge zwischen Erfüllbarkeit, Folgerbarkeit, Äquivalenz und Gültigkeit.** Es besteht ein enger Zusammenhang zwischen semantischer Äquivalenz, Gültigkeit, logischer Folgerbarkeit als binäre Relation über Formeln sowie Erfüllbarkeit (bzw. Unerfüllbarkeit). Jeder der vier Begriffe lässt sich auf die anderen drei Begriffe zurückführen. Z.B. gilt für die Konsequenzrelation  $\Vdash$  als binäre Relation über Formeln:

$$\begin{aligned} \alpha \Vdash \beta & \quad \text{gdw} \quad \alpha \rightarrow \beta \text{ ist gültig} \\ & \quad \text{gdw} \quad \alpha \wedge \neg\beta \text{ ist unerfüllbar} \\ & \quad \text{gdw} \quad \alpha \rightarrow \beta \equiv \text{true} \end{aligned}$$

Für endliche Formelmengen gilt:

$$\begin{aligned} \{\beta_1, \dots, \beta_n\} \Vdash \alpha & \quad \text{gdw} \quad \beta_1 \wedge \dots \wedge \beta_n \Vdash \alpha \\ & \quad \text{gdw} \quad \beta_1 \wedge \dots \wedge \beta_n \rightarrow \alpha \text{ ist gültig} \\ & \quad \text{gdw} \quad \beta_1 \wedge \dots \wedge \beta_n \wedge \neg\alpha \text{ ist unerfüllbar} \end{aligned}$$

Die Darstellung der semantischen Äquivalenz durch die Konzepte Folgerbarkeit, Gültigkeit oder Unerfüllbarkeit ist wie folgt:

$$\begin{aligned} \alpha \equiv \beta & \quad \text{gdw} \quad \alpha \Vdash \beta \text{ und } \beta \Vdash \alpha \\ & \quad \text{gdw} \quad \alpha \leftrightarrow \beta \text{ ist gültig} \\ & \quad \text{gdw} \quad \alpha \oplus \beta \text{ ist unerfüllbar} \end{aligned}$$

Dabei ist  $\oplus$  der sogenannte Paritätsoperator, auch “exklusives oder” oder XOR genannt. Dieser ist durch

$$\alpha \oplus \beta \stackrel{\text{def}}{=} (\neg\alpha \wedge \beta) \vee (\alpha \wedge \neg\beta)$$

definiert. Offenbar ist  $\alpha \oplus \beta$  äquivalent zu  $\neg(\alpha \leftrightarrow \beta)$ .

Diese Beobachtung ist für algorithmische Zwecke interessant, da z.B. jeder Algorithmus für das Erfüllbarkeitsproblem zugleich als Gültigkeitstest, Folgerbarkeitstest oder Äquivalenztest eingesetzt werden kann.

**Das Model Checking Problem der Aussagenlogik.** Wir werden im Verlauf der Vorlesung sehen, dass das Erfüllbarkeitsproblem, das Gültigkeitsproblem, das Äquivalenzproblem und das Folgerbarkeitsproblem der Aussagenlogik in exponentieller Zeit lösbar sind. Sie zählen zu “algorithmisch schwierigen” Problemen, für die es vermutlich keine effizienten Algorithmen gibt. Das Model Checking Problem ist jedoch gänzlich anders geartet, da dieses eine Formel  $\alpha$  und eine Belegung  $I$  als gegeben voraussetzt und lediglich die Berechnung des Wahrheitswerts  $\alpha^I \in \{0, 1\}$  erfordert, statt eine Aussage über alle Belegungen zu machen. Im Falle der Aussagenlogik ist das Model Checking Problem sehr einfach (bei geeigneter Implementierung in linearer Zeit) zu lösen. Beispielsweise kann die Formel in Postfixnotation gebracht werden, die dann “von links nach rechts” ausgewertet werden kann. Dies entspricht im Wesentlichen der Erstellung des *Syntaxbaums* der Formel, dessen Knoten im Postorder-Prinzip generiert und in Bottom-Up-Manier bewertet werden. In Abbildung 31 ist der Syntaxbaum für die Formel

$$\alpha = (\neg x \wedge \neg(y \wedge z)) \wedge \neg(\neg y \wedge x)$$

skizziert. Die Beschriftungen in dieser Abbildung deuten an, wie die Wahrheitswerte der Teilformeln – dargestellt durch die Knoten des Syntaxbaums – von unten nach oben propagiert werden. Ausgangspunkt ist dabei die Belegung  $I$  mit  $x^I = 0$  und  $y^I = z^I = 1$ , welche Wahrheitswerte für die Blätter (die mit je einem der Atome  $x, y, z$  beschriftet sind) vorgibt.

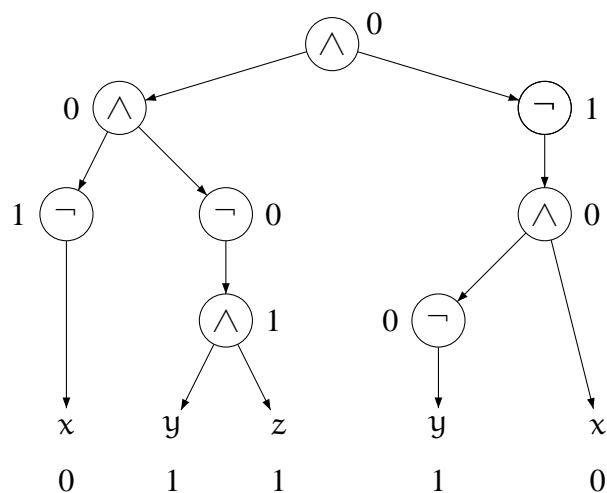


Abbildung 31: Bottom-Up-Bewertung des Syntaxbaums für aussagenlogische Formel