4

Logik

„Wenn das Quadrat einer ganzen Zahl immer gerade ist, dann ist 7 ohne Rest[13] durch 3 teilbar.“ Diese zahlentheoretische Aussage ist zweifellos wahr, auch wenn sie keinen großen Sinn ergibt. Es handelt sich hierbei um die logische Implikation

$$(\forall x \in \mathbb{Z}) \ \left[x^2 \bmod 2 = 0\right] \ \Rightarrow \ 7 \bmod 3 = 0.$$

Aus der (universell quantifizierten) Aussage „$(\forall x \in \mathbb{Z}) \ \left[x^2 \bmod 2 = 0\right]$“ folgt die Aussage „$7 \bmod 3 = 0$“. Natürlich ist die zweite Aussage falsch, denn 7 geteilt durch 3 lässt den Rest 1. Aber die erste Aussage ist ebenfalls falsch, da zum Beispiel $3^2 = 9$ keine gerade Zahl ist. Somit ist die obige Implikation eine wahre Aussage, unabhängig davon, wie unsinnig die Aussage auch zu sein scheint.

4.1 Boolesche Ausdrücke

Eine *boolesche*[14] *Variable* ist eine Variable, der ein *Wahrheitswert wahr* oder *falsch*, *true* oder *false* bzw. 1 oder 0 zugeordnet werden kann. Wir werden in diesem Kapitel für Wahrheitswerte überwiegend die Bezeichnungen **true** und **false** verwenden; diese bezeichnet man auch als *boolesche Konstanten*. Boolesche Konstanten und Variablen können mit *booleschen Verknüpfungen* wie zum Beispiel dem logischen *Und* ($\wedge$), dem logischen *Oder* ($\vee$) und der *Negation* ($\neg$) zu booleschen Ausdrücken kombiniert werden. Ein boolescher Ausdruck ist also entweder eine boolesche Konstante oder Variable oder ein Ausdruck, der durch die Anwendung von booleschen Verknüpfungen auf einen oder mehrere boolesche Ausdrücke entsteht.

boolesche Variable
Wahrheitswert **true**
Wahrheitswert **false**
boolesche Konstante
boolesche Verknüpfung

[13] Für zwei ganze Zahlen $x, y \in \mathbb{Z}$, mit $y > 0$, sei x mod y der positive ganzzahlige Rest zwischen 0 und $y-1$ bei der Division von x durch y. Zum Beispiel ist $21 = 5 \cdot 4 + 1$ und somit $21 \bmod 4 = 1$, aber $-21 = -6 \cdot 4 + 3$ und somit $-21 \bmod 4 = 3$.

[14] George Boole (* 2. November 1815 in Lincoln, England; † 8. Dezember 1864 in Ballintemple, in der Grafschaft Cork, Irland) war ein englischer Mathematiker (Autodidakt), Logiker und Philosoph.

F. Gurski et al., *Exakte Algorithmen für schwere Graphenprobleme*, eXamen.press,
DOI 10.1007/978-3-642-04500-4_4,

boolescher Ausdruck **Definition 4.1 (boolescher Ausdruck).** *Ein* boolescher Ausdruck *ist entweder*

1. *eine boolesche Konstante,* **true** *oder* **false***,*
2. *eine boolesche Variable, wie zum Beispiel x oder y,*
3. *ein Ausdruck der Form* $(\neg\varphi)$*,*
4. *ein Ausdruck der Form* $(\varphi_1 \vee \varphi_2)$ *oder*
5. *ein Ausdruck der Form* $(\varphi_1 \wedge \varphi_2)$*,*

Negation $\neg$ Konjunktion $\wedge$ Disjunktion $\vee$ *wobei* φ*,* φ_1 *und* φ_2 *boolesche Ausdrücke sind. Die Verknüpfungen* $\neg$*,* $\wedge$ *und* $\vee$ *werden auch* Negation*,* Konjunktion *und* Disjunktion *genannt.*

Semantik Die Bedeutung (*Semantik*) eines booleschen Ausdrucks ist sehr einfach zu beschreiben. Ein Ausdruck ist entweder *wahr* oder *falsch*, je nachdem, welche Werte die beteiligten booleschen Variablen annehmen.

Definition 4.2 (Erfüllbarkeit boolescher Ausdrücke). *Sei* φ *ein boolescher Ausdruck. Die Menge* $X(\varphi)$ *der an* φ *beteiligten Variablen ist*

$$X(\varphi) = \begin{cases} \emptyset, & \text{falls } \varphi = \textbf{true} \text{ oder } \varphi = \textbf{false}, \\ \{x\}, & \text{falls } \varphi = x, \\ X(\varphi'), & \text{falls } \varphi = (\neg\varphi'), \\ X(\varphi_1) \cup X(\varphi_2), & \text{falls } \varphi = (\varphi_1 \vee \varphi_2) \text{ oder } \varphi = (\varphi_1 \wedge \varphi_2). \end{cases}$$

Belegung *Für eine Menge X von Variablen ist eine* Belegung der Variablen in *X eine Abbildung*

$$T : X \to \{\textbf{true}, \textbf{false}\},$$

die jeder Variablen $x \in X$ *einen Wahrheitswert* $T(x) \in \{\textbf{true}, \textbf{false}\}$ *zuordnet.*

Eine Belegung T erfüllt *einen booleschen Ausdruck* φ *mit der Variablenmenge* $\vDash$ $X(\varphi)$*, geschrieben* $T \vDash \varphi$*, falls*

1. $\varphi = \textbf{true}$*,*
2. $\varphi = x$ *und* $T(x) = \textbf{true}$*,*
3. $\nvDash$ $\varphi = (\neg\varphi')$ *und T erfüllt nicht den Ausdruck* φ'*, geschrieben* $T \nvDash \varphi'$*,*
4. $\varphi = (\varphi_1 \vee \varphi_2)$ *und es gilt* $T \vDash \varphi_1$ *oder* $T \vDash \varphi_2$*, oder*
5. $\varphi = (\varphi_1 \wedge \varphi_2)$ *und es gilt sowohl* $T \vDash \varphi_1$ *als auch* $T \vDash \varphi_2$*.*

Beispiel 4.3 (boolescher Ausdruck). Betrachte den booleschen Ausdruck

$$\varphi = (x_1 \wedge ((\neg x_2) \vee x_3))$$

mit der Variablenmenge $X(\varphi) = \{x_1, x_2, x_3\}$. Sei

$$T : X(\varphi) \to \{\textbf{true}, \textbf{false}\}$$

eine Belegung der Variablen von φ mit $T(x_1) = \textbf{true}$, $T(x_2) = \textbf{false}$ und $T(x_3) = \textbf{false}$.

Die Belegung T erfüllt den Ausdruck φ, weil sie die beiden Ausdrücke x_1 und $((\neg x_2) \vee x_3)$ der Konjunktion φ erfüllt. Der erste Ausdruck x_1 dieser Konjunktion

wird von T erfüllt, weil $T(x_1) = \mathbf{true}$ gilt. Der zweite Ausdruck $((\neg x_2) \vee x_3)$ dieser Konjunktion wird von T erfüllt, weil der erste Ausdruck $\neg x_2$ der Disjunktion $((\neg x_2) \vee x_3)$ in φ von T erfüllt wird, denn T erfüllt wegen $T(x_2) = \mathbf{false}$ nicht den Ausdruck x_2. Es ist nicht notwendig, dass die Belegung T auch den zweiten Ausdruck x_3 dieser Disjunktion erfüllt.

Im Folgenden lassen wir zur besseren Lesbarkeit überflüssige Klammerungen weg, wenn sich dadurch keine Mehrdeutigkeiten ergeben. Dabei geben wir der Negation $\neg$ eine höhere Priorität als den Verknüpfungen $\wedge$ und $\vee$. So kann beispielsweise der boolesche Ausdruck $((\neg x) \wedge (\neg y))$ einfacher als $\neg x \wedge \neg y$ geschrieben werden.

Boolesche Ausdrücke können durch die Verwendung weiterer boolescher Operationen oft vereinfacht werden. Die *Implikation* $\varphi_1 \Rightarrow \varphi_2$ entspricht zum Beispiel dem Ausdruck

Implikation

$$\neg\varphi_1 \vee \varphi_2$$

und die *Äquivalenz* $\varphi_1 \Leftrightarrow \varphi_2$ dem Ausdruck

Äquivalenz

$$(\varphi_1 \wedge \varphi_2) \vee (\neg\varphi_1 \wedge \neg\varphi_2).$$

Zwei boolesche Ausdrücke φ_1 und φ_2 sind *äquivalent*, geschrieben $\varphi_1 \equiv \varphi_2$, wenn für jede Belegung $T : X \to \{\mathbf{true}, \mathbf{false}\}$ mit $X = X(\varphi_1) \cup X(\varphi_2)$ die Belegung T genau dann φ_1 erfüllt, wenn sie φ_2 erfüllt. Oder, anders ausgedrückt, wenn jede dieser Belegungen

$\equiv$
äquivalente Ausdrücke

$$T : X(\varphi_1 \Leftrightarrow \varphi_2) \to \{\mathbf{true}, \mathbf{false}\}$$

den booleschen Ausdruck $\varphi_1 \Leftrightarrow \varphi_2$ erfüllt.

Tabelle 4.1. Häufig verwendete Äquivalenzen von booleschen Ausdrücken

$\varphi_1 \vee \varphi_2 \equiv \varphi_2 \vee \varphi_1$ $\varphi_1 \wedge \varphi_2 \equiv \varphi_2 \wedge \varphi_1$	Kommutativgesetze
$\neg\neg\varphi \equiv \varphi$	Doppelnegation
$(\varphi_1 \vee \varphi_2) \vee \varphi_3 \equiv \varphi_1 \vee (\varphi_2 \vee \varphi_3)$ $(\varphi_1 \wedge \varphi_2) \wedge \varphi_3 \equiv \varphi_1 \wedge (\varphi_2 \wedge \varphi_3)$	Assoziativgesetze
$(\varphi_1 \wedge \varphi_2) \vee \varphi_3 \equiv (\varphi_1 \vee \varphi_3) \wedge (\varphi_2 \vee \varphi_3)$ $(\varphi_1 \vee \varphi_2) \wedge \varphi_3 \equiv (\varphi_1 \wedge \varphi_3) \vee (\varphi_2 \wedge \varphi_3)$	Distributivgesetze
$\neg(\varphi_1 \vee \varphi_2) \equiv \neg\varphi_1 \wedge \neg\varphi_2$ $\neg(\varphi_1 \wedge \varphi_2) \equiv \neg\varphi_1 \vee \neg\varphi_2$	De Morgan'sche Regeln
$\varphi \vee \varphi \equiv \varphi$ $\varphi \wedge \varphi \equiv \varphi$	Idempotenzen boolescher Ausdrücke
$\varphi_1 \vee (\varphi_1 \wedge \varphi_2) \equiv \varphi_1$ $\varphi_1 \wedge (\varphi_1 \vee \varphi_2) \equiv \varphi_1$	Absorbtionsgesetze

Tabelle 4.1 enthält einige Äquivalenzen, die häufig bei der Umformung boolescher Ausdrücke verwendet werden. Die Richtigkeit dieser Äquivalenzen kann einfach mit Hilfe von *Wahrheitstabellen* überprüft werden. In einer Wahrheitstabelle für

Wahrheitstabellen

φ wird für jede Belegung $T : X(\varphi) \to \{\mathbf{true}, \mathbf{false}\}$ angegeben, ob sie φ erfüllt oder nicht.

Beispiel 4.4 (Nachweis eines Distributivgesetzes). Zum Nachweis des Distributivgesetzes

$$(\varphi_1 \vee \varphi_2) \wedge \varphi_3 \equiv (\varphi_1 \wedge \varphi_3) \vee (\varphi_2 \wedge \varphi_3) \tag{4.1}$$

aus Tabelle 4.1 genügt es, die Wahrheitstabellen für beide Seiten der Äquivalenz (4.1) aufzustellen und miteinander zu vergleichen. Tabelle 4.2 zeigt, dass sämtliche Wahrheitswerte für beide Seiten identisch, die entsprechenden booleschen Ausdrücke also äquivalent sind.

Tabelle 4.2. Nachweis eines Distributivgesetzes in Beispiel 4.4

φ_1	φ_2	φ_3	$(\varphi_1 \vee \varphi_2) \wedge \varphi_3$	$(\varphi_1 \wedge \varphi_3) \vee (\varphi_2 \wedge \varphi_3)$
false	**false**	**false**	**false**	**false**
false	**false**	**true**	**false**	**false**
false	**true**	**false**	**false**	**false**
false	**true**	**true**	**true**	**true**
true	**false**	**false**	**false**	**false**
true	**false**	**true**	**true**	**true**
true	**true**	**false**	**false**	**false**
true	**true**	**true**	**true**	**true**

Übung 4.5. Zeigen Sie die übrigen Äquivalenzen aus Tabelle 4.1 mittels Wahrheitstabellen nach dem Muster von Beispiel 4.4.

Das Assoziativgesetz erlaubt es uns, Klammerungen über Verknüpfungen gleicher Art ebenfalls einfach wegzulassen. Zum Beispiel können wir den Ausdruck $(\varphi_1 \vee \neg\varphi_2) \vee (\varphi_3 \vee \varphi_4)$ vereinfacht auch als $\varphi_1 \vee \neg\varphi_2 \vee \varphi_3 \vee \varphi_4$ schreiben, da alle denkbaren Klammerungen der vier booleschen Ausdrücke $\varphi_1, \neg\varphi_2, \varphi_3, \varphi_4$ mit der gleichen booleschen Verknüpfung zu äquivalenten Ausdrücken führen. Im Weiteren werden wir gelegentlich die Schreibweise

$\bigvee_{i=1}^{n} \varphi_i$

$$\bigvee_{i=1}^{n} \varphi_i \quad \text{für} \quad \varphi_1 \vee \varphi_2 \vee \cdots \vee \varphi_n \tag{4.2}$$

und

$\bigwedge_{i=1}^{n} \varphi_i$

$$\bigwedge_{i=1}^{n} \varphi_i \quad \text{für} \quad \varphi_1 \wedge \varphi_2 \wedge \cdots \wedge \varphi_n \tag{4.3}$$

verwenden.

Literal
positives Literal

Eine Variable x bzw. negierte Variable $\neg x$ nennen wir auch ein *Literal*. Nicht negierte Variablen nennen wir *positive Literale*. Eine Konjunktion $\varphi_1 \wedge \varphi_2 \wedge \cdots \wedge \varphi_n$

von Literalen $\varphi_1, \varphi_2, \ldots, \varphi_n$ heißt *Implikant*, eine Disjunktion $\varphi_1 \vee \varphi_2 \vee \cdots \vee \varphi_n$ von Literalen $\varphi_1, \varphi_2, \ldots, \varphi_n$ heißt *Klausel*. Eine Klausel mit höchstens einem positiven Literal heißt *Horn-Klausel*.[15] Ein boolescher Ausdruck φ ist in *konjunktiver Normalform* (kurz *KNF*), wenn er eine Konjunktion von Klauseln ist, wenn also φ die Form

Implikant
Klausel
Horn-Klausel
konjunktive Normalform
KNF

$$\varphi = \bigwedge_{i=1}^{n} C_i$$

hat und jedes C_i eine Klausel ist. Ein boolescher Ausdruck φ ist in *disjunktiver Normalform* (kurz *DNF*), wenn er eine Disjunktion von Implikanten ist, d. h., wenn φ die Form

disjunktive Normalform
DNF

$$\varphi = \bigvee_{i=1}^{n} D_i$$

hat und jedes D_i ein Implikant ist.

Satz 4.6. *Zu jedem booleschen Ausdruck gibt es einen äquivalenten booleschen Ausdruck in konjunktiver Normalform und einen äquivalenten booleschen Ausdruck in disjunktiver Normalform.*

Beweis. Dieser Satz lässt sich mit einer einfachen Induktion über den Aufbau der booleschen Ausdrücke zeigen.

Induktionsanfang: Ist φ eine boolesche Konstante oder Variable, so ist φ offensichtlich sowohl in konjunktiver als auch in disjunktiver Normalform.

Induktionsschritt:

1. Sei $\varphi = \neg\varphi'$.
   Der Ausdruck φ kann wie folgt in einen äquivalenten booleschen Ausdruck in konjunktiver (bzw. disjunktiver) Normalform transformiert werden. Transformiere zuerst φ' mit Hilfe der Induktionsvoraussetzung in einen äquivalenten Ausdruck in disjunktiver (bzw. konjunktiver) Normalform und wende anschließend die De Morgan'schen Regeln an, bis φ in konjunktiver (bzw. disjunktiver) Normalform ist.
2. Sei $\varphi = \varphi_1 \vee \varphi_2$.
   Der Ausdruck φ ist bereits in disjunktiver Normalform, wenn mit Hilfe der Induktionsvoraussetzung φ_1 und φ_2 in disjunktive Normalform transformiert wurden.
   Der Ausdruck φ kann wie folgt in einen äquivalenten Ausdruck in konjunktiver Normalform transformiert werden. Transformiere zuerst φ_1 und φ_2 mit Hilfe der Induktionsvoraussetzung in äquivalente Ausdrücke in konjunktiver Normalform und wende anschließend das Distributivgesetz an, bis φ in konjunktiver Normalform ist.
3. $\varphi = \varphi_1 \wedge \varphi_2$. Analog zu Fall 2.

[15] Alfred Horn (* 17. Februar 1918; † 16. April 2001) war ein amerikanischer Mathematiker.

Damit ist der Beweis abgeschlossen. ❑

Bei dem *Ausmultiplizieren* in Fall 2 und 3 mit Hilfe der Distributivgesetze können $n \cdot m$ Klauseln bzw. Implikanten für φ entstehen, wenn die disjunktiven bzw. konjunktiven Normalformen von φ_1 und φ_2 genau n bzw. m Terme enthalten. Die Größe eines Ausdrucks φ kann also bei der vollständigen Umwandlung in entweder eine konjunktive oder eine disjunktive Normalform exponentiell anwachsen. Dies gilt übrigens auch dann, wenn anschließend doppelte Literale in den Klauseln und Implikanten sowie doppelte Klauseln bzw. doppelte Implikanten aufgrund der Idempotenz boolescher Ausdrücke weggelassen werden.

Beispiel 4.7 (konjunktive und disjunktive Normalform). Ein äquivalenter Ausdruck in disjunktiver Normalform für einen booleschen Ausdruck φ kann direkt aus der vollständigen Wertetabelle für φ abgelesen werden. Die Implikanten lassen sich aus der Belegung $T : X \to \{\mathbf{true}, \mathbf{false}\}$ mit $T \vDash \varphi$ ableiten. Sei zum Beispiel $\varphi = ((\neg x_1 \wedge x_2) \vee \neg x_3) \wedge x_4$. Dann ist

$$\begin{aligned} \varphi_d \;=\; & (\neg x_1 \wedge \neg x_2 \wedge \neg x_3 \wedge x_4) \vee (\neg x_1 \wedge x_2 \wedge \neg x_3 \wedge x_4) \vee (\neg x_1 \wedge x_2 \wedge x_3 \wedge x_4) \vee \\ & (x_1 \wedge \neg x_2 \wedge \neg x_3 \wedge x_4) \quad \vee (x_1 \wedge x_2 \wedge \neg x_3 \wedge x_4) \end{aligned}$$

ein äquivalenter Ausdruck in disjunktiver Normalform, wie leicht aus Tabelle 4.3 abgelesen werden kann.

Tabelle 4.3. Eine Wahrheitstabelle für Beispiel 4.7

$T(x_1)$	$T(x_2)$	$T(x_3)$	$T(x_4)$	$\vDash$	DNF φ_d	KNF φ_k
f	**f**	**f**	**f**	$T \nvDash \varphi$		$(x_1 \vee x_2 \vee x_3 \vee x_4)$
f	**f**	**f**	**t**	$T \vDash \varphi$	$(\neg x_1 \wedge \neg x_2 \wedge \neg x_3 \wedge x_4)$	
f	**f**	**t**	**f**	$T \nvDash \varphi$		$\wedge\ (x_1 \vee x_2 \vee \neg x_3 \vee x_4)$
f	**f**	**t**	**t**	$T \nvDash \varphi$		$\wedge\ (x_1 \vee x_2 \vee \neg x_3 \vee \neg x_4)$
f	**t**	**f**	**f**	$T \nvDash \varphi$		$\wedge\ (x_1 \vee \neg x_2 \vee x_3 \vee x_4)$
f	**t**	**f**	**t**	$T \vDash \varphi$	$\vee\ (\neg x_1 \wedge x_2 \wedge \neg x_3 \wedge x_4)$	
f	**t**	**t**	**f**	$T \nvDash \varphi$		$\wedge\ (x_1 \vee \neg x_2 \vee \neg x_3 \vee x_4)$
f	**t**	**t**	**t**	$T \vDash \varphi$	$\vee\ (\neg x_1 \wedge x_2 \wedge x_3 \wedge x_4)$	
t	**f**	**f**	**f**	$T \nvDash \varphi$		$\wedge\ (\neg x_1 \vee x_2 \vee x_3 \vee x_4)$
t	**f**	**f**	**t**	$T \vDash \varphi$	$\vee\ (x_1 \wedge \neg x_2 \wedge \neg x_3 \wedge x_4)$	
t	**f**	**t**	**f**	$T \nvDash \varphi$		$\wedge\ (\neg x_1 \vee x_2 \vee \neg x_3 \vee x_4)$
t	**f**	**t**	**t**	$T \nvDash \varphi$		$\wedge\ (\neg x_1 \vee x_2 \vee \neg x_3 \vee \neg x_4)$
t	**t**	**f**	**f**	$T \nvDash \varphi$		$\wedge\ (\neg x_1 \vee \neg x_2 \vee x_3 \vee x_4)$
t	**t**	**f**	**t**	$T \vDash \varphi$	$\vee\ (x_1 \wedge x_2 \wedge \neg x_3 \wedge x_4)$	
t	**t**	**t**	**f**	$T \nvDash \varphi$		$\wedge\ (\neg x_1 \vee \neg x_2 \vee \neg x_3 \vee x_4)$
t	**t**	**t**	**t**	$T \nvDash \varphi$		$\wedge\ (\neg x_1 \vee \neg x_2 \vee \neg x_3 \vee \neg x_4)$

In den Spalten mit den Überschriften $T(x_1), \ldots, T(x_4)$ sind zeilenweise die Belegungen der Variablen angegeben. Ein **t** bzw. **f** in Spalte $T(x_i)$ bedeutet $T(x_i) = \mathbf{true}$

bzw. $T(x_i) = \textbf{false}$. Die Spalte mit der Überschrift $\vDash$ kennzeichnet, ob die Belegung der Zeile den Ausdruck erfüllt oder nicht. Ein äquivalenter Ausdruck für φ in konjunktiver Normalform kann einfach durch Anwendung der De Morgan'schen Regeln auf $\neg\varphi'$ erzeugt werden, wenn $\varphi' = \neg\varphi$ in disjunktiver Normalform vorliegt. Für das obige Beispiel wäre dies der Ausdruck

$$\begin{aligned}\varphi_k = {} & (x_1 \vee x_2 \vee x_3 \vee x_4) && \wedge (x_1 \vee x_2 \vee \neg x_3 \vee x_4) && \wedge (x_1 \vee x_2 \vee \neg x_3 \vee \neg x_4) \wedge \\ & (x_1 \vee \neg x_2 \vee x_3 \vee x_4) && \wedge (x_1 \vee \neg x_2 \vee \neg x_3 \vee x_4) && \wedge (\neg x_1 \vee x_2 \vee x_3 \vee x_4) \quad \wedge \\ & (\neg x_1 \vee x_2 \vee \neg x_3 \vee x_4) && \wedge (\neg x_1 \vee x_2 \vee \neg x_3 \vee \neg x_4) && \wedge (\neg x_1 \vee \neg x_2 \vee x_3 \vee x_4) \wedge \\ & (\neg x_1 \vee \neg x_2 \vee \neg x_3 \vee x_4) && \wedge (\neg x_1 \vee \neg x_2 \vee \neg x_3 \vee \neg x_4). \end{aligned}$$

Die *Größe size*(φ) *eines booleschen Ausdrucks* φ ist die Anzahl aller Vorkommen von Literalen. Der Ausdruck φ aus Beispiel 4.7 hat somit eine Größe von 4, die disjunktive Normalform eine Größe von 20 und die konjunktive Normalform die Größe von 44. Es ist sehr wichtig, die Größe eines booleschen Ausdrucks φ genau zu definieren, da wir die Laufzeit von Algorithmen mit Eingabe φ immer in der Größe size(φ) der Eingabe φ messen müssen. size(φ)

Übung 4.8. Das „exklusive Oder" ist die boolesche Verknüpfung $\oplus$ definiert durch $x_1 \oplus x_2 = (\neg x_1 \wedge x_2) \vee (x_1 \wedge \neg x_2)$. Bestimmen Sie einen möglichst kleinen booleschen Ausdruck φ in konjunktiver Normalform äquivalent zu $x_1 \oplus x_2 \oplus x_3 \oplus x_4 \oplus x_5$.

4.2 SAT, 3-SAT, 2-SAT und Horn-SAT

Das Erfüllbarkeitsproblem für boolesche Ausdrücke in konjunktiver Normalform, auch SATISFIABILITY bzw. SAT genannt, ist eines der klassischen Beispiele für ein Entscheidungsproblem, von dem man bis heute nicht weiß, ob es effizient lösbar ist. Das heißt, es gibt bis heute keinen Algorithmus mit einer Laufzeit polynomiell in size(φ), der für einen gegebenen booleschen Ausdruck φ in konjunktiver Normalform entscheiden kann, ob es eine Belegung T der Variablen gibt, die φ erfüllt. Erfüllbarkeitsproblem SATISFIABILITY SAT

SATISFIABILITY (SAT)	
Gegeben:	Ein boolescher Ausdruck φ in konjunktiver Normalform.
Frage:	Gibt es eine Belegung T der Variablen in $X(\varphi)$, die φ erfüllt?

Einige interessante Varianten von SAT sind 3-SAT, 2-SAT und HORN-SAT. 3-SATISFIABILITY 3-SAT

3-SATISFIABILITY (3-SAT)	
Gegeben:	Ein boolescher Ausdruck φ in konjunktiver Normalform, in dem jede Klausel genau drei Literale enthält.
Frage:	Gibt es eine Belegung T der Variablen in $X(\varphi)$, die φ erfüllt?

2-SATISFIABILITY
2-SAT

2-SATISFIABILITY (2-SAT)	
Gegeben:	Ein boolescher Ausdruck φ in konjunktiver Normalform, in dem jede Klausel höchstens zwei Literale enthält.
Frage:	Gibt es eine Belegung T der Variablen in $X(\varphi)$, die φ erfüllt?

HORN-SATISFIABILITY
HORN-SAT

HORN-SATISFIABILITY (HORN-SAT)	
Gegeben:	Ein boolescher Ausdruck φ in konjunktiver Normalform, in dem jede Klausel höchstens ein positives Literal enthält.
Frage:	Gibt es eine Belegung T der Variablen in $X(\varphi)$, die φ erfüllt?

Für 3-SAT sind bisher ebenfalls keine Polynomialzeit-Algorithmen bekannt. Es ist offenbar genauso schwer zu entscheiden wie SAT, wie wir in Behauptung 5.22 in Abschnitt 5.1.3 sehen werden.

Übung 4.9. Zeigen Sie, dass sich jeder boolesche Ausdruck in einen äquivalenten booleschen Ausdruck in disjunktiver Normalform transformieren lässt, in dem jeder Implikant höchstens drei Literale enthält.

Die beiden Varianten HORN-SAT und 2-SAT sind im Gegensatz zu SAT und 3-SAT jedoch effizient entscheidbar, wie die beiden folgenden Sätze zeigen.

Satz 4.10. *Sei φ ein boolescher Ausdruck in konjunktiver Normalform, in dem entweder jede Klausel höchstens ein positives Literal enthält oder jede Klausel höchstens ein negiertes Literal enthält. Eine erfüllende Belegung T der Variablen in $X(\varphi)$ für φ kann in linearer Zeit konstruiert werden, falls eine solche Belegung existiert.*

Beweis. Der Algorithmus, der eine erfüllende Belegung in linearer Zeit findet, arbeitet wie folgt. Sei φ ein boolescher Ausdruck in konjunktiver Normalform, in der jede Klausel höchstens ein positives Literal enthält. Setze zu Beginn alle Variablen auf **false**. Wenn jede Klausel in φ mindestens ein negiertes Literal enthält, ist T bereits eine erfüllende Belegung für φ, und wir sind fertig. Sei also C_j eine Klausel mit ausschließlich positiven Literalen. Da die Klausel laut Voraussetzung höchstens ein positives Literal enthält, besteht C_j also aus genau einem positiven Literal x_i. Die Variable x_i muss also in jeder erfüllenden Belegung wahr sein. Wir setzen die Variable x_i deshalb erzwungenermaßen auf **true** und entfernen alle Klauseln aus φ, die das positive Literal x_i enthalten. Diese Klauseln sind ebenfalls erfüllt und bleiben es auch, da wir im weiteren Verlauf keine Variable zurück auf **false** setzen werden. Zusätzlich entfernen wir die negierten Literale $\neg x_i$ aus den übrigen Klauseln, da diese negierten Literale $\neg x_i$ nicht wahr sind und im weiteren Verlauf auch nicht wahr

werden. Entstehen dabei weitere Klauseln mit ausschließlich einem positiven Literal, so wird auch aus diesen Klauseln die entsprechende Variable erzwungenermaßen auf **true** gesetzt usw. Entsteht dabei irgendwann eine leere Klausel, also eine Klausel ohne Literale, so gibt es keine erfüllende Belegung T der Variablen in $X(\varphi)$ für φ, da das Setzen der Variablen auf **true** in allen Schritten zwingend war. Eine solche leere Klausel kann nur dann entstehen, wenn sie zu Beginn ausschließlich negierte Literale enthielt, die dann alle Schritt für Schritt entfernt wurden.

Das Belegen der Variablen kann in linearer Zeit erfolgen, indem an den Variablen x_i Verweise zu den Klauseln C_j gespeichert werden, die das Literal x_i oder $\neg x_i$ enthalten. Die Anzahl aller Verweise ist nicht größer als die Größe von φ. Jeder Verweis wird bei der Konstruktion der erfüllenden Belegung nur ein Mal verfolgt.

Der zweite Fall, in dem jede Klausel höchstens ein negiertes Literal enthält, kann analog behandelt werden. Dabei wird mit einer **true**-Belegung für alle Variablen begonnen, in der dann die Variablen schrittweise auf **false** gesetzt werden. ❑

Satz 4.11. *Sei φ ein boolescher Ausdruck in konjunktiver Normalform, in dem jede Klausel höchstens zwei Literale enthält. Eine erfüllende Belegung T der Variablen in $X(\varphi)$ für φ kann in linearer Zeit konstruiert werden, falls eine solche Belegung existiert.*

Beweis. Zu Beginn werden überflüssige Klauseln und Variablen aus φ entfernt. Klauseln der Form $(x_i \vee \neg x_i)$ werden ersatzlos gestrichen, da diese Klauseln immer wahr sind. Besteht eine Klausel aus genau einem positiven Literal x_i, so wird die Variable x_i auf **true** gesetzt. Anschließend werden alle Klauseln mit einem Literal x_i aus φ sowie alle Literale $\neg x_i$ aus den übrigen Klauseln entfernt. Analog wird mit den Klauseln verfahren, die aus genau einem negierten Literal $\neg x_i$ bestehen. Dazu wird die Variable x_i auf **false** gesetzt, und es werden alle Klauseln mit einem Literal $\neg x_i$ aus φ sowie alle Literale x_i aus den übrigen Klauseln entfernt. Dadurch entstehen möglicherweise weitere Klauseln mit genau einem Literal, die in gleicher Weise eliminiert werden. Entsteht dabei eine leere Klausel, d. h., wird das letzte Literal aus einer Klausel entfernt, so gibt es offensichtlich keine erfüllende Belegung, die φ erfüllt.

Nun ist φ ein boolescher Ausdruck in konjunktiver Normalform, sodass jede Klausel von φ genau zwei Literale für verschiedene Variablen enthält. Jede solche Klausel, $(\ell_i \vee \ell_j)$, ist einerseits äquivalent zu der Implikation $(\neg\ell_i \Rightarrow \ell_j)$ und andererseits äquivalent zu der Implikation $(\neg\ell_j \Rightarrow \ell_i)$. Zum Beispiel ist $(x_i \vee \neg x_j)$ äquivalent zu $(\neg x_i \Rightarrow \neg x_j)$ und zu $(x_j \Rightarrow x_i)$. Der boolesche Ausdruck φ kann daher auch als eine Konjunktion von Implikationen dargestellt werden.

Zur Konstruktion einer erfüllenden Belegung wird ein gerichteter Graph $G_\varphi = (V,E)$ definiert. Die Knotenmenge V von G_φ besteht aus der Menge aller Literale. Die Kantenmenge enthält genau dann die beiden Kanten $(\neg\ell_i, \ell_j)$ und $(\neg\ell_j, \ell_i)$, wenn φ die Klausel $(\ell_i \vee \ell_j)$ enthält. Zum Beispiel enthält E für eine Klausel $(x_i \vee \neg x_j)$ demnach die beiden Kanten $(\neg x_i, \neg x_j)$ und (x_j, x_i).

Ist in einer erfüllenden Belegung ein Literal ℓ_i **true**, dann müssen auch alle Literale, die in G_φ von ℓ_i über Kanten erreichbar sind, **true** sein. Dies folgt aus der Tran-

sitivität der Implikationen. Für die Konstruktion einer erfüllenden Belegung werden deshalb die Literale auf den Wegen in φ möglichst spät **true** gesetzt. Dazu werden zunächst die starken Zusammenhangskomponenten $S_1, \ldots, S_k \subseteq V$ von G_φ berechnet. Jedes S_i ist eine maximale Menge von Knoten mit der Eigenschaft, dass es für alle u und v in S_i in G_φ einen Weg von u nach v gibt.

starke Zusammenhangskomponente

Die Knotenmengen $S_1, S_2, \ldots, S_k$ bilden eine Partition von V, d. h., jedes Literal ist in genau einer Menge S_i. Die Literale einer starken Zusammenhangskomponente sind alle paarweise äquivalent. In jeder erfüllenden Belegung sind sie entweder alle **true** oder alle **false**. Enthält eine der starken Zusammenhangskomponenten beide Literale x_i und $\neg x_i$ einer Variablen x_i, dann existiert keine erfüllende Belegung T für φ.

Nun wird der *verdichtete Graph* $G'_\varphi = (V', E')$ mit der Knotenmenge $V' = \{S_1, S_2, \ldots, S_k\}$ aufgebaut (siehe Definition 3.35). G'_φ enthält genau dann eine gerichtete Kante (S_i, S_j), wenn es in G_φ eine Kante (u_i, u_j) mit $u_i \in S_i$ und $u_j \in S_j$ gibt. Der verdichtete Graph enthält keine Kreise und wird wie folgt für die Konstruktion einer erfüllenden Belegung verwendet.

Wir bearbeiten die Knoten aus G'_φ nun in topologischer Reihenfolge. Wähle einen beliebigen Knoten S_i mit

1. S_i hat keine einlaufenden Kanten in G'_φ und
2. S_i enthält ausschließlich Literale, deren Variablen bisher noch nicht gesetzt wurden.

Setze die Variablen der Literale aus S_i so, dass alle Literale in S_i den Wert **false** erhalten. Entferne anschließend den Knoten S_i und seine auslaufenden Kanten aus G'_φ und wiederhole die obige Anweisung, bis kein weiterer Knoten in G_φ die beiden Eigenschaften erfüllt. Alle bisher nicht zugewiesenen Variablen werden zum Schluss so gesetzt, dass die Literale in den übrigen starken Zusammenhangskomponenten den Wert **true** erhalten. Diese abschließende Belegung der verbleibenden Variablen ist zwingend und führt genau dann zu einer erfüllenden Belegung für φ, wenn G_φ keine Kreise über beide Literale x_i und $\neg x_i$ einer Variablen x_i besitzt.

Die einzelnen Phasen in der obigen Vorgehensweise können mit elementaren Standardtechniken so implementiert werden, dass die Konstruktion einer erfüllenden Belegung in linearer Zeit erfolgt. ❑

Beispiel 4.12 (Lösungsidee für 2-SAT). Für den booleschen Ausdruck

$$\varphi = (x_1 \vee x_2) \wedge (x_2 \vee \neg x_4) \wedge (\neg x_2 \vee \neg x_3) \wedge (x_3 \vee x_4)$$

besteht der Graph $G_\varphi = (V, E)$ aus acht Knoten und acht Kanten.

$$V = \{x_1, \neg x_1, x_2, \neg x_2, x_3, \neg x_3, x_4, \neg x_4\};$$
$$E = \left\{ \begin{array}{l} (\neg x_1, x_2), (\neg x_2, x_1), (\neg x_2, \neg x_4), (x_4, x_2), \\ (x_2, \neg x_3), (x_3, \neg x_2), (\neg x_3, x_4), (\neg x_4, x_3) \end{array} \right\}.$$

Es gibt in G_φ zwei Kreise, $(x_2, \neg x_3, x_4)$ und $(\neg x_2, x_3, \neg x_4)$, und vier starke Zusammenhangskomponenten, $S_1 = \{x_1\}$, $S_2 = \{x_2, \neg x_3, x_4\}$, $S_3 = \{\neg x_1\}$ und $S_4 =$

$\{\neg x_2, x_3, \neg x_4\}$. Der verdichtete Graph G'_φ besitzt somit vier Knoten, S_1, S_2, S_3 und S_4, und zwei Kanten, (S_3, S_2) und (S_4, S_1). Zu Beginn kann S_3 oder S_4 gewählt werden, weil beide Knoten keine einlaufenden Kanten besitzen. Wir entscheiden uns für S_3 und setzen $T(x_1) = \mathbf{true}$. Dadurch erhalten alle Literale in S_3 (hier nur das eine Literal $\neg x_1$) den Wert **false**. Gleichzeitig wird dadurch das Literal x_1 in S_1 auf **true** gesetzt. Wird S_3 aus dem verdichteten Graphen entfernt, kann S_2 oder S_4 gewählt werden. Beide Komponenten haben nun keine Vorgänger und enthalten noch keine gesetzten Literale. Wir entscheiden uns für S_4 und setzen $T(x_2) = \mathbf{true}$, $T(x_3) = \mathbf{false}$ und $T(x_4) = \mathbf{true}$. Dadurch erhalten alle Literale in S_4, also $\neg x_2$, x_3 und $\neg x_4$, den Wert **false**. Gleichzeitig erhalten dabei alle Literale in S_2 den Wert **true**. Wird S_4 aus dem verdichteten Graphen entfernt, bleiben die Komponenten S_1 und S_2 zurück. Nun werden alle noch nicht gesetzten Variablen so gesetzt, dass die Literale der übrig gebliebenen Komponenten S_1 und S_2 **true** sind, was in unserem Beispiel bereits der Fall ist.

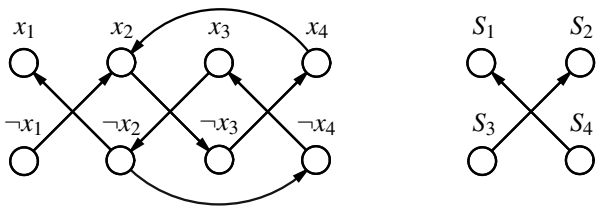

Abb. 4.1. Graph G_φ mit vier starken Zusammenhangskomponenten (links) und der verdichtete Graph G'_φ (rechts)

4.3 Boolesche Funktionen und Schaltkreise

Wertet man einen booleschen Ausdruck bezüglich einer Belegung seiner Variablen mit Wahrheitswerten aus, so erhält man einen Wahrheitswert. Boolesche Ausdrücke stellen also Funktionen dar, so genannte boolesche Funktionen, die einem Tupel von Wahrheitswerten einen Wahrheitswert zuordnen.

Definition 4.13 (boolesche Funktion). *Eine* n-stellige boolesche Funktion *ist eine Abbildung*

$$f : \{\mathbf{true}, \mathbf{false}\}^n \to \{\mathbf{true}, \mathbf{false}\}.$$

n-stellige boolesche Funktion

Einstellige (bzw. zweistellige) boolesche Funktionen nennt man auch unäre *(bzw.* binäre*)* boolesche Funktionen.

unäre boolesche Funktion
binäre boolesche Funktion

Jeder boolesche Ausdruck φ kann als eine $|X(\varphi)|$-stellige boolesche Funktion

$$f_\varphi : \{\mathbf{true}, \mathbf{false}\}^{|X(\varphi)|} \to \{\mathbf{true}, \mathbf{false}\}$$

mit

$$f_\varphi(\alpha_1,\ldots,\alpha_n) = \begin{cases} \textbf{true}, & \text{falls } T \vDash \varphi \text{ mit } T(x_i) = \alpha_i \text{ für } 1 \le i \le n, \\ \textbf{false}, & \text{falls } T \nvDash \varphi \text{ mit } T(x_i) = \alpha_i \text{ für } 1 \le i \le n \end{cases}$$

angesehen werden. Die Operation $\neg$ ist eine der insgesamt vier unären booleschen Funktionen. Die anderen drei unären booleschen Funktionen sind die beiden konstanten Funktionen $f(x) = \textbf{true}$ und $f(x) = \textbf{false}$ sowie die Identität $f(x) = x$. Die booleschen Operationen $\wedge$, $\vee$, $\Rightarrow$ und $\Leftrightarrow$ sind vier der insgesamt 16 binären booleschen Funktionen.

Umgekehrt gibt es für jede n-stellige boolesche Funktion f auch einen booleschen Ausdruck φ_f mit $|X(\varphi)| = n$, sodass für jede Belegung $T : X(\varphi) \to \{\textbf{true}, \textbf{false}\}$ genau dann $f(T(x_1),\ldots,T(x_n)) = \textbf{true}$ gilt, wenn $T \vDash \varphi$ gilt.

Boolesche Funktionen können nicht nur durch boolesche Ausdrücke, sondern auch durch boolesche Schaltkreise dargestellt werden.

boolescher Schaltkreis

Definition 4.14 (boolescher Schaltkreis). *Ein* boolescher Schaltkreis *ist ein gerichteter kreisfreier Graph $G = (V,E)$, dessen Knoten mit einem konstanten Wert (***true** *oder* **false***), einer der Eingabevariablen $x_1, x_2, \ldots, x_n$ oder einer der booleschen Operationen $\wedge$, $\vee$ und $\neg$ markiert sind. Die Knoten werden je nach Markierung* **true***-,* **false***-, x_i-, $\wedge$-, $\vee$- oder $\neg$-*Gatter *genannt. Jedes x_i-Gatter heißt auch* Eingabegatter.

true-Gatter
false-Gatter
x_i-Gatter
$\wedge$-Gatter
$\vee$-Gatter
$\neg$-Gatter
Eingabegatter
Ausgabegatter

true*-,* **false***- und Eingabegatter haben keine, $\neg$-Gatter haben genau eine und $\wedge$- und $\vee$-Gatter haben je zwei einlaufende Kanten. Jedes Gatter kann eine beliebige Anzahl von auslaufenden Kanten haben. Gatter ohne auslaufende Kanten werden* Ausgabegatter *genannt.*

Jede Belegung T der Variablen $x_1, x_2, \ldots, x_n$ definiert für jedes Gatter g einen Wahrheitswert $f(g) \in \{\textbf{true}, \textbf{false}\}$, der wie folgt induktiv definiert ist:

- *$f(g)$ ist* **true***,* **false** *bzw. $T(x_i)$, falls g ein* **true***-Gatter,* **false***-Gatter bzw. x_i-Gatter ist.*
- *Ist g ein $\neg$-Gatter und ist g' der Vorgänger von g, dann ist $f(g) = \textbf{true}$, falls $f(g') = \textbf{false}$ ist, und es ist $f(g) = \textbf{false}$, falls $f(g') = \textbf{true}$ ist.*
- *Ist g ein $\wedge$-Gatter (bzw. ein $\vee$-Gatter) und sind g_1 und g_2 die beiden Vorgänger von g, dann ist $f(g) = \textbf{true}$, falls $f(g') = \textbf{true}$ und $f(g') = \textbf{true}$ (bzw. falls $f(g') = \textbf{true}$ oder $f(g') = \textbf{true}$) ist. Ansonsten ist $f(g) = \textbf{false}$.*

Boolesche Schaltkreise können mehrere Ausgabegatter haben; sie können also mehrere boolesche Funktionen gleichzeitig berechnen.

Natürlich kann man auch Schaltkreise mit anderen Gattertypen definieren, beispielsweise Schaltkreise, in denen zusätzlich $\Leftrightarrow$-Gatter erlaubt sind. Da sich diese jedoch durch $\wedge$-, $\vee$- und $\neg$-Gatter ausdrücken lassen, kann man darauf auch verzichten. Ebenso könnte man in Definition 4.14 beispielsweise auf $\vee$-Gatter verzichten, ohne dass die resultierenden Schaltkreise an Ausdrucksstärke verlieren würden, da sich $\vee$-Gatter durch $\wedge$- und $\neg$-Gatter simulieren lassen. Auf die *Größe eines Schaltkreises* (also die Anzahl seiner Gatter) wirkt es sich natürlich aus, welche Arten von Gattern zur Verfügung stehen. Je nachdem, welche boolesche Funktion zu berechnen ist, kann die Größe eines booleschen Schaltkreises stark mit den erlaubten Gattertypen variieren. Die in Definition 4.14 verwendeten Gattertypen sind bei der Definition von Schaltkreisen am gebräuchlichsten.

Größe eines Schaltkreises

Übung 4.15. (a) Zeigen Sie, wie sich ein $\Leftrightarrow$-Gatter durch $\wedge$-, $\vee$- und $\neg$-Gatter simulieren lässt.
(b) Zeigen Sie, wie sich ein $\vee$-Gatter durch $\wedge$- und $\neg$-Gatter simulieren lässt.
(c) Die boolesche Funktion $\oplus$ (das „exklusive Oder“) wurde in Übung 4.8 definiert. Durch welche Mengen von Gattertypen kann man $\oplus$-Gatter ausdrücken? Stellen Sie eine Vermutung an, durch welche Mengen von Gattertypen man $\oplus$-Gatter *nicht* ausdrücken kann, und versuchen Sie, Ihre Vermutung zu beweisen.

Beispiel 4.16 (boolescher Schaltkreis). Abbildung 4.2 zeigt einen booleschen Schaltkreis mit drei Eingabegattern, x_1, x_2 und x_2, und zwei Ausgabegattern. Für jede Belegung T der drei Variablen x_1, x_2 und x_3, die den booleschen Ausdruck

$$(\neg x_1 \wedge x_2) \vee (x_1 \wedge \neg x_2) \tag{4.4}$$

erfüllt, ist der Wahrheitswert des oberen Ausgabegatters **true**. Für jede Belegung T der drei Variablen x_1, x_2 und x_3, die den booleschen Ausdruck

$$(x_1 \wedge \neg x_2) \vee (x_3 \wedge (\mathbf{false} \vee \mathbf{true})) \tag{4.5}$$

erfüllt, ist der Wahrheitswert des unteren Ausgabegatters **true**. Für alle anderen Belegungen sind die Wahrheitswerte beider Ausgabegatter **false**. Der boolesche Ausdruck in (4.5) ist übrigens äquivalent zu $(x_1 \wedge \neg x_2) \vee x_3$.

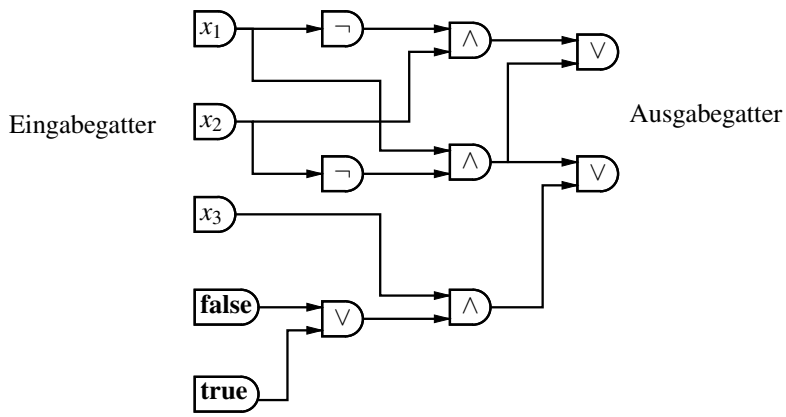

Abb. 4.2. Ein boolescher Schaltkreis mit drei Eingabegattern und zwei Ausgabegattern

Boolesche Funktionen lassen sich mit Hilfe von booleschen Schaltkreisen oft kompakter als mit Hilfe von booleschen Ausdrücken repräsentieren. Dies liegt daran, dass gleiche Teilausdrücke in booleschen Schaltkreisen nicht dupliziert werden müssen, da die Gatter beliebig viele auslaufende Kanten haben dürfen. Dennoch besitzen nicht alle booleschen Funktionen auch eine kompakte Darstellung durch einen booleschen Schaltkreis. Der folgende Satz sagt, dass nicht jede n-stellige boolesche

Funktion mit einem booleschen Schaltkreis der Größe höchstens $2^n/2n$ berechnet werden kann. Diese Aussage folgt aus der Tatsache, dass es mehr n-stellige boolesche Funktionen als boolesche Schaltkreise gibt, die mit $2^n/2n$ Gattern realisiert werden können. Insbesondere folgt aus Satz 4.17, dass für hinreichend großes n nicht jede n-stellige boolesche Funktion mit einem booleschen Schaltkreis berechnet werden kann, der höchstens $p(n)$ Gatter enthält, wobei p ein beliebiges Polynom ist, denn asymptotisch wächst $2^n/2n$ schneller als jedes Polynom.

Satz 4.17. *Für jedes $n \geq 1$ gibt es eine n-stellige boolesche Funktion, die nicht durch einen booleschen Schaltkreis mit höchstens $2^n/2n$ Gattern berechnet werden kann.*

Beweis. Für den Fall $n = 1$ gilt die Aussage des Satzes offenbar, denn $2^1/2 \cdot 1 = 1$ und mit nur einem Gatter kann man beispielsweise nicht die unäre boolesche Funktion $\neg$ realisieren; dafür bräuchte man ja ein Eingabe- und ein $\neg$-Gatter.

Sei im Folgenden also $n > 1$. Jede boolesche Funktion ist eindeutig durch die Menge der Belegungen der n Variablen bestimmt, die als Ergebnis den Wahrheitswert **true** haben. Bei n Variablen gibt es 2^n Belegungen und 2^{2^n} Auswahlmöglichkeiten der 2^n Belegungen. Somit gibt es genau 2^{2^n} verschiedene boolesche Funktionen mit n Variablen.

Die Frage, wie viele boolesche Schaltkreise mit höchstens $m = 2^n/2n$ Gattern es gibt, ist da schon etwas schwieriger. Gäbe es für ein $n \geq 1$ weniger als 2^{2^n} boolesche Schaltkreise mit höchstens m Gattern, dann wäre mindestens eine der 2^{2^n} booleschen Funktionen nicht durch einen Schaltkreis mit höchstens m Gattern realisierbar.

Wir können die Anzahl der booleschen Schaltkreise mit höchstens m Gattern relativ einfach grob abschätzen, indem wir für jeden Schaltkreis die Gatter nummerieren, typisieren (also für jedes Gatter festlegen, welchen Typ es hat) und ihre jeweils höchstens zwei Vorgänger festlegen. Da wir bei n Variablen $x_1, x_2, \ldots, x_n$ und den fünf Symbolen **true**, **false**, $\neg$, $\wedge$ und $\vee$ insgesamt $n+5$ verschiedene Gattertypen haben und jedes Gatter höchstens zwei von m Vorgängern hat, gibt es höchstens $(n+5) \cdot m^2$ verschiedene Arten von Gattern, auch wenn einige Gatter keine zwei Vorgänger haben. Um einen Schaltkreis mit höchstens m Gattern festzulegen, wird jedem seiner Gatter eine solche Art zugewiesen. Insgesamt gibt es also höchstens

$$\left((n+5) \cdot m^2\right)^m$$

boolesche Schaltkreise. Natürlich ist nicht alles, was hier gezählt wurde, auch tatsächlich ein (syntaktisch korrekter) boolescher Schaltkreis; beispielweise wurde auch der „Schaltkreis“ mit m $\neg$-Gattern, aber ohne ein einziges Eingabegatter gezählt. Aber selbst diese grobe Überschätzung der Anzahl der booleschen Schaltkreise mit m Gattern ist noch nicht ausreichend, jede der 2^{2^n} booleschen Funktionen durch einen solchen Schaltkreis zu berechnen, da für $m = 2^n/2n$ gilt:

$$\left((n+5) \cdot \frac{2^n}{2n} \cdot \frac{2^n}{2n}\right)^{\frac{2^n}{2n}} < 2^{2^n}. \tag{4.6}$$

Die Ungleichung (4.6) kann leicht wie folgt Schritt für Schritt aufgelöst werden:

$$\begin{aligned}
\left((n+5)\cdot\tfrac{2^n}{2n}\cdot\tfrac{2^n}{2n}\right)^{\frac{2^n}{2n}} &< 2^{2^n}\\
\log_2\left(\left((n+5)\cdot\tfrac{2^n}{2n}\cdot\tfrac{2^n}{2n}\right)^{\frac{2^n}{2n}}\right) &< 2^n\\
\tfrac{2^n}{2n}\cdot\log_2\left((n+5)\cdot\tfrac{2^n}{2n}\cdot\tfrac{2^n}{2n}\right) &< 2^n\\
\tfrac{2^n}{2n}\cdot(\log_2(n+5)+\log_2(2^n)-\log_2(2n)+\log_2(2^n)-\log_2(2n)) &< 2^n\\
2^n\cdot\left(1-\tfrac{2\cdot\log_2(2n)-\log_2(n+5)}{2n}\right) &< 2^n\\
2^n\cdot\left(1-\tfrac{\log_2(\frac{4n^2}{n+5})}{2n}\right) &< 2^n\\
0 &< \tfrac{\log_2(\frac{4n^2}{n+5})}{2n}.
\end{aligned}$$

Da für $n > 1$ die letzte Ungleichung stimmt, wie man leicht nachrechnen kann, gilt die ursprüngliche Ungleichung (4.6). ❑

4.4 Relationale Strukturen und Logik höherer Ordnung

Die Logik höherer Ordnung umfasst weitaus mehr als das, was wir mit booleschen Ausdrücken, booleschen Funktionen oder booleschen Schaltkreisen beschreiben können. Die bisher betrachtete *Aussagenlogik* wird in der Logik höherer Ordnung durch Quantifizierungen, Konstanten und Relationen zur *Prädikatenlogik* erweitert. Dazu definieren wir zuerst das Vokabular der Logik. Anschließend erklären wir den Aufbau der Formeln und werden dann auf die Bedeutung der aufgebauten Formeln eingehen.

Aussagenlogik
Prädikatenlogik

In den nachfolgenden Definitionen werden Formeln als Zeichenfolgen aufgefasst. Diese Zeichenfolgen enthalten Symbole für Variablen, Konstanten und Relationen. Weiterhin werden spezielle Zeichen wie zum Beispiel das Gleichheitszeichen, runde Klammern, das Komma und Symbole für die booleschen Verknüpfungen $\wedge$, $\vee$ und $\neg$ und die Quantoren $\exists$ und $\forall$ verwendet. Diese Symbole werden aber auch in der mathematischen Sprache benutzt, mit der wir die Logik definieren. Dies ist anfangs etwas verwirrend, aber dennoch kein Widerspruch. Denn schließlich wollen wir die Semantik der hier eingeführten Logik genau so definieren, wie wir sie intuitiv bereits kennen.

Definition 4.18 (relationales Vokabular, relationale Signatur). *Sei $\mathscr{S}$ eine endliche Menge von Symbolen. Die Symbole aus $\mathscr{S}$ werden für die Namen der Konstanten und Relationen verwendet. Jedes Symbol aus $\mathscr{S}$ besitzt eine nicht negative*

ganzzahlige Stelligkeit. Symbole c mit der Stelligkeit 0, $ar(c) = 0$, werden für Konstanten verwendet. Symbole R mit einer positiven Stelligkeit, $ar(R) > 0$, werden für $ar(R)$-stellige Relationen verwendet. Die Symbolmenge $\mathscr{S}$ wird auch (relationales) Vokabular *oder* (relationale) Signatur *genannt.*

Vokabular
Signatur

relationale Struktur

Definition 4.19 (endliche relationale Struktur). *Sei $\mathscr{S}$ eine Signatur. Eine* relationale Struktur über $\mathscr{S}$ *(eine $\mathscr{S}$-*Struktur*) ist ein Paar $S = (D, \mathscr{R}^S)$, wobei D eine endliche, nicht leere* Definitionsmenge *und $\mathscr{R}^S$ eine endliche Menge von Konstanten aus D und Relationen über D ist.*

Definitionsmenge

1. *Jeder Konstanten $c^{\mathscr{S}} \in \mathscr{R}^S$ ist ein Symbol $c \in \mathscr{S}$ mit der Stelligkeit 0 zugeordnet.*
2. *Jeder Relation $R^{\mathscr{S}} \in \mathscr{R}^S$ ist ein Symbol $R \in \mathscr{S}$ mit einer positiven Stelligkeit zugeordnet.*
3. *Alle Konstanten und Relationen haben paarweise verschiedene Symbole aus der Signatur $\mathscr{S}$.*

Wir beschränken uns auf relationale Strukturen mit einer endlichen, nicht leeren[16] Definitionsmenge D. Zur Vereinfachung verwenden wir für die Konstanten und Relationen aus $\mathscr{R}^S$ und deren Symbole aus $\mathscr{S}$ ähnliche Bezeichner. Die Variablen für Konstanten und Relationen bezeichnen wir mit ihrem Symbol und dem hochgestellten Strukturbezeichner. Zum Beispiel ist R^S eine Relation mit Symbol R in der Struktur S.

relationale Nachfolgerstruktur

Beispiel 4.20 (Nachfolgerstruktur). Sei n eine positive ganze Zahl und $\mathscr{S} = \{\text{first}, \text{suc}\}$ eine Signatur mit den beiden Symbolen „first“ und „suc“. Sei $S_n = (D_n, \mathscr{R}^{S_n})$ die relationale Struktur über Signatur $\mathscr{S}$ mit $D_n = \{1, \ldots, n\}$ und $\mathscr{R}^{S_n} = \{\text{first}^{S_n}, \text{suc}^{S_n}\}$ mit der Konstanten

$$\text{first}^{S_n} = 1 \in D_n$$

und der binären Relation

$$\text{suc}^{S_n} = \{(1,2), (2,3), \cdots, (n-1, n)\} \subseteq D_n^2.$$

Die Relation suc^{S_n} ist hier eine binäre *Nachfolgerrelation* auf der Definitionsmenge D_n mit first^{S_n} als erster Komponente des ersten Elements.

Die Unterscheidung zwischen den Symbolen „first” und „suc” aus der Signatur $\mathscr{S}$ und den Bezeichnern „first^{S_n}” und „suc^{S_n}”, die wir für die Relationen der relationalen Struktur verwenden, ist wirklich notwendig. Denn die Symbole der Signatur dienen zur Definition einer logischen Formel unabhängig von einer speziellen gegebenen relationalen Struktur.

[16] Es ist durchaus üblich, die leere Menge als Definitionsmenge auszuschließen. Ob man die leere Menge ausschließt oder zulässt, macht einen Unterschied bei Beweissystemen. Die Aussage $\exists x\ (x = x)$ ist zum Beispiel für nicht leere Definitionsmengen wahr, aber für leere Definitionsmengen falsch.

Die Gleichheit von zwei relationalen Strukturen ist (wie zum Beispiel auch bei Graphen) über die Existenz eines Isomorphismus zwischen den beiden Strukturen definiert. Alle in diesem Buch betrachteten algorithmischen Fragestellungen auf relationalen Strukturen unterscheiden nicht zwischen isomorphen Strukturen. Wir sind also immer nur an der dargestellten Struktureigenschaft interessiert und nicht an der konkreten Repräsentation der Struktureigenschaft (siehe auch Übung 4.23).

isomorphe Strukturen

Definition 4.21 (isomorphe Strukturen). *Es seien $S_1 = (D_1, \mathscr{R}_1^{S_1})$ eine relationale Struktur über der Signatur $\mathscr{S}_1$ und $S_2 = (D_2, \mathscr{R}_2^{S_1})$ eine relationale Struktur über der Signatur $\mathscr{S}_2$. S_1 und S_2 sind* isomorph*, wenn es eine bijektive Abbildung $h : D_1 \to D_2$ mit den folgenden Eigenschaften gibt:*

1. *Für jede Konstante $c_1^{S_1} \in \mathscr{R}^{S_1}$ gibt es eine Konstante $c_2^{S_2} \in \mathscr{R}^{S_2}$,[17] sodass*

$$h(c_1^{S_1}) = c_2^{S_2}. \tag{4.7}$$

2. *Für jede Relation $R_1^{S_1} \in \mathscr{R}_1^{S_1}$ gibt es eine Relation $R_2^{S_2} \in \mathscr{R}_2^{S_2}$ mit gleicher Stelligkeit $ar(R_1) = ar(R_2) = n$ für ein $n > 0$,[18] sodass*

$$\{(h(a_1), \ldots, h(a_n)) \mid (a_1, \ldots, a_n) \in R_1^{S_1}\} = R_2^{S_2}. \tag{4.8}$$

relationale Vorgängerstruktur

Beispiel 4.22 (Vorgängerstruktur). Sei n eine positive ganze Zahl und $\mathscr{S}' = \{\text{last}, \text{pre}\}$ eine Signatur mit den beiden Symbolen „last“ und „pre“. Sei $S'_n = (D_n, \mathscr{R}^{S'_n})$ die relationale Struktur über der Signatur $\mathscr{S}'$ mit $D_n = \{1, \ldots, n\}$ und $\mathscr{R}^{S'_n} = \{\text{last}^{S'_n}, \text{suc}^{S'_n}\}$ mit der Konstanten

$$\text{last}^{S'_n} = n \in D_n$$

und der binären Relation

$$\text{pre}^{S'_n} = \{(n, n-1), (n-1, n-2), \cdots, (2,1)\} \subseteq D_n^2.$$

Die relationale Nachfolgerstruktur S_n aus Beispiel 4.20 und die obige relationale Struktur S'_n sind isomorph. Dies belegt die bijektive Abbildung $h : D_n \to D_n$ mit $h(i) = n + 1 - i$, da

$$h(\text{first}^{S_n}) = h(1) = n = \text{last}^{S'_n}$$

und

$$\{(h(a_1), h(a_2)) \mid (a_1, a_n) \in \text{suc}^{S_n}\} = \text{pre}^{S'_n}.$$

Übung 4.23. Definiere eine relationale Struktur $S = (D, \mathscr{R}^S)$ und eine Eigenschaft für S, die zwischen isomorphen Strukturen unterscheidet.

[17] Da h eine Bijektion ist, gibt es umgekehrt auch für jede Konstante $c_2^{S_2} \in \mathscr{R}^{S_2}$ eine Konstante $c_1^{S_1} \in \mathscr{R}^{S_1}$ mit (4.7).

[18] Da h eine Bijektion ist, gibt es umgekehrt auch für jede Relation $R_2^{S_2} \in \mathscr{R}_2^{S_2}$ eine Relation $R_1^{S_1} \in \mathscr{R}_1^{S_1}$ gleicher Stelligkeit mit (4.8).

4.5 Logik erster Ordnung

atomare Formel erster Ordnung
Formel erster Ordnung
Quantoren $\exists$ und $\forall$

Definition 4.24 (Formel erster Ordnung). *Sei $\mathscr{X}$ eine Menge von Symbolen für Variablen erster Ordnung und sei $\mathscr{S}$ eine Signatur. Eine* atomare Formel erster Ordnung über der Signatur $\mathscr{S}$ *ist von der Form $(x_1 = x_2)$ oder $R(x_1,\ldots,x_n)$, wobei $x_1,\ldots,x_n \in \mathscr{X} \cup \mathscr{S}$ Symbole für Variablen erster Ordnung oder Konstanten sind und R ein Relationssymbol ist. Die* Formeln erster Ordnung *werden wie folgt aus den atomaren Formeln erster Ordnung, den Symbolen für die booleschen Verknüpfungen $\wedge$, $\vee$ und $\neg$ sowie den Symbolen für die Quantoren $\exists$ und $\forall$ gebildet.*

1. *Jede atomare Formel erster Ordnung ist eine Formel erster Ordnung.*
2. *Sind φ_1 und φ_2, φ' Formeln erster Ordnung, dann sind auch $(\varphi_1 \wedge \varphi_2)$, $(\varphi_1 \vee \varphi_2)$, $(\neg\, \varphi')$, $(\exists x\ \varphi')$ und $(\forall x\ \varphi')$ Formeln erster Ordnung, wobei $x \in \mathscr{X}$ ein Symbol für eine* Variable erster Ordnung *ist.*

FO

Die Klasse aller Formeln erster Ordnung bezeichnen wir mit FO *(englisch:* first order*).*

Kommt in einer quantifizierten Formel φ erster Ordnung eine Variable x im Wirkungsbereich eines Quantors $\exists$ oder $\forall$ vor, so nennen wir dieses Vorkommen von x *gebunden*. Kommt eine Variable y in φ vor, die nicht durch einen Quantor gebunden ist, so sagen wir, y kommt *frei* vor. Eine Variable nennen wir *gebunden* bzw. *frei*, wenn sie wenigstens einmal gebunden bzw. wenigstens einmal frei in φ vorkommt. Ein und dieselbe Variable kann in einer Formel also sowohl frei als auch gebunden vorkommen. Um die in einer Formel φ frei vorkommenden Variablen $x_1,\ldots,x_n$ kenntlich zu machen, schreiben wir für φ auch $\varphi(x_1,\ldots,x_n)$. Kommen in einer quantifizierten Formel φ freie Variablen vor, so heißt φ *offen*. Sind jedoch alle Variablenvorkommen in φ gebunden (enthält φ also keine freien Variablen), so heißt φ *geschlossen*. Jede geschlossene quantifizierte Formel φ lässt sich zu entweder **true** oder **false** auswerten. Eine offene quantifizierte Formel $\varphi(x_1,\ldots,x_n)$ ist hingegen eine boolesche Funktion ihrer $n \geq 1$ freien Variablen, die von $\{\mathbf{true},\mathbf{false}\}^n$ in $\{\mathbf{true},\mathbf{false}\}$ abbildet (siehe Abschnitt 4.3).

gebundene Variable
freie Variable
offene Formel
geschlossene Formel

Eine Formel erster Ordnung über einer Signatur $\mathscr{S}$ kann von einer relationalen Struktur über der Signatur $\mathscr{S}$ erfüllt werden. Wann dies genau der Fall ist, regelt die *Semantik*. Viele Dinge lassen sich vereinfachen, wenn die Semantik einer Formel φ für eine Struktur S mit Hilfe der Lösungsmenge $L^S(\varphi)$ definiert wird.

Semantik

Definition 4.25 (Semantik einer Formel erster Ordnung). *Seien φ eine Formel erster Ordnung und $S = (D, \mathscr{R}^S)$ eine relationale Struktur, beide definiert über einer gemeinsamen Signatur. Die Semantik von φ für die Struktur S definieren wir mit Hilfe der* Menge aller Lösungen $L^S(\varphi)$ *von φ für S. Diese Lösungsmenge ist rekursiv über*

Lösungsmenge

den Aufbau der Formeln definiert.[19] *Wir setzen voraus, dass für jedes verwendete Symbol in φ auch tatsächlich eine Konstante bzw. Relation in $\mathscr{R}^S$ existiert.*

Ist φ eine Formel mit n freien Variablen $x_1,\ldots,x_n$, dann ist $L^S(\varphi)$ eine n-stellige Relation über D.

1. *Ist φ eine atomare Formel erster Ordnung der Form $(c_1 = c_2)$ mit zwei Symbolen c_1 und c_2 für Konstanten, dann ist*

$$L^S(\varphi()) \;=\; \begin{cases} \{()\}, & \text{falls } c_1^S = c_2^S, \\ \emptyset & \text{sonst.} \end{cases}$$

2. *Ist $\varphi(x)$ eine atomare Formel erster Ordnung der Form $(x = c)$ oder $(c = x)$ mit einem Symbol c für eine Konstante und einem Symbol x für eine Variable erster Ordnung, dann ist*

$$L^S(\varphi(x)) \;=\; \{(c^S)\}.$$

3. *Ist $\varphi(x_1,x_2)$ eine atomare Formel der Form $(x_1 = x_2)$ mit zwei Symbolen x_1 und x_2 für Variablen erster Ordnung, dann ist*

$$L^S(\varphi(x_1,x_2)) \;=\; \{(a,a) \mid a \in D\}.$$

4. *Ist $\varphi(x_1,\ldots,x_n)$ eine atomare Formel erster Ordnung der Form $R(x_{i_1},\ldots,x_{i_k})$ mit $i_1,\ldots,i_k \in [n] = \{1,\ldots,n\}$ und den Symbolen $x_1,\ldots,x_n$ für Variablen erster Ordnung, dann ist* [n]

$$L^S(\varphi(x_1,\ldots,x_n)) \;=\; \left\{ (a_1,\ldots,a_n) \;\middle|\; \begin{array}{l} a_1,\ldots,a_n \in D, \\ (a_{i_1},\ldots,a_{i_k}) \in R^S \end{array} \right\}.$$

5. *Ist $\varphi(x_1,\ldots,x_n)$ eine Formel erster Ordnung der Form $\neg\varphi'(x_1,\ldots,x_n)$ mit den Symbolen $x_1,\ldots,x_n$ für Variablen erster Ordnung, dann ist*

$$L^S(\varphi(x_1,\ldots,x_n)) \;=\; \left\{ (a_1,\ldots,a_n) \;\middle|\; \begin{array}{l} a_1,\ldots,a_n \in D, \\ (a_1,\ldots,a_n) \notin L^S(\varphi') \end{array} \right\}.$$

6. *Ist $\varphi(x_1,\ldots,x_n)$ eine Formel erster Ordnung der Form*

$$(\varphi_1(x_{i_1},\ldots,x_{i_k}) \wedge \varphi_2(x_{j_1},\ldots,x_{j_\ell}))$$

mit $i_1,\ldots,i_k,\ j_1,\ldots,j_\ell \in [n]$ und den Symbolen $x_1,\ldots,x_n$ für Variablen erster Ordnung, dann ist

$$L^S(\varphi(x_1,\ldots,x_n)) \;=\; \left\{ (a_1,\ldots,a_n) \;\middle|\; \begin{array}{l} a_1,\ldots,a_n \in D, \\ (a_{i_1},\ldots,a_{i_k}) \in L^S(\varphi_1) \wedge \\ (a_{j_1},\ldots,a_{j_\ell}) \in L^S(\varphi_2) \end{array} \right\}.$$

Für $\varphi(x_1,\ldots,x_n) = (\varphi_1(x_{i_1},\ldots,x_{i_k}) \vee \varphi_2(x_{j_1},\ldots,x_{j_\ell}))$ ist $L^S(\varphi(x_1,\ldots,x_n))$ analog definiert.

[19] In den Definitionen verwenden wir die gewohnte mathematische Ausdrucksweise zur Beschreibung mengentheoretischer Zusammenhänge. Diese mathematischen Beschreibungen dürfen nicht mit den definierten Formeln erster Ordnung verwechselt werden, auch wenn sie zum Teil die gleichen Zeichen (wie zum Beispiel die Zeichen „$\exists$", „$\forall$", „$\wedge$", „$\vee$", „$\neg$", „$=$", „(" und „)") verwenden.

7. Ist $\varphi(x_1,\ldots,x_n)$ eine Formel erster Ordnung der Form

$$\varphi(x_1,\ldots,x_n) = \exists x_{n+1}\ \varphi'(x_{i_1},\ldots,x_{i_k})$$

mit $i_1,\ldots,i_k \in [n+1]$ und den Symbolen $x_1,\ldots,x_{n+1}$ für Variablen erster Ordnung, dann ist

$$L^S(\varphi(x_1,\ldots,x_n)) \;=\; \left\{(a_1,\ldots,a_n) \;\middle|\; \begin{array}{l} a_1,\ldots,a_n \in D, \\ \exists a_{n+1} \in D : (a_{i_1},\ldots,a_{i_k}) \in L^S(\varphi') \end{array} \right\}.$$

Für

$$\varphi(x_1,\ldots,x_n) = \forall x_{n+1}\ \varphi'(x_{i_1},\ldots,x_{i_k})$$

ist $L^S(\varphi(x_1,\ldots,x_n))$ analog definiert.

Sind $S = (D, \mathscr{R}^S)$ eine relationale Struktur und φ eine Formel erster Ordnung mit n freien Variablen, beide definiert über einer gemeinsamen Signatur, dann ist jedes $(a_1,\ldots,a_n)$ mit $a_1,\ldots,a_n \in D$ eine *Instanz für φ*. Die Instanzen in der Lösungsmenge $L^S(\varphi)$ sind die *Lösungen für φ* . Wir sagen auch für $(a_1,\ldots,a_n) \in L^S(\varphi)$, die Instanz $(a_1,\ldots,a_n)$ der relationalen Struktur S erfüllt die Formel $\varphi(x_1,\ldots,x_n)$.

Instanz für φ

Lösung für φ

Eine Struktur S erfüllt eine Formel φ, bezeichnet mit

$$S \vDash \varphi(x_1,\ldots,x_n),$$

falls $L^S(\varphi)$ nicht leer ist. Enthält φ keine freien Variablen, dann ist $L^S(\varphi)$ entweder leer (also $L^S(\varphi) = \emptyset$) oder die Menge mit dem leeren Tupel $()$ (also $L^S(\varphi) = \{()\}$).

Um die Lesbarkeit der Formeln zu erleichtern, erlauben wir uns wie gehabt, Klammerungen wegzulassen, wenn dadurch keine Mehrdeutigkeiten entstehen. Abgekürzte Schreibweisen mit den Symbolen $\Rightarrow$ und $\Leftrightarrow$ sind ebenfalls erlaubt, d. h., wir schreiben $(\varphi_1 \Rightarrow \varphi_2)$ für $(\neg\varphi_1 \vee \varphi_2)$ und $(\varphi_1 \Leftrightarrow \varphi_2)$ für $((\varphi_1 \wedge \varphi_2) \vee (\neg\varphi_1 \wedge \neg\varphi_2))$. Gelegentlich schreiben wir aus Gründen der Übersichtlichkeit statt $\varphi = (\exists x\ \varphi')$ bzw. $\varphi = (\forall x\ \varphi')$ auch $\varphi = (\exists x)\ [\varphi']$ bzw. $\varphi = (\forall x)\ [\varphi']$.

Beispiel 4.26. Die quantifizierte boolesche Formel

$$\begin{array}{l} \varphi(x,y,z) = \\ \quad ((\exists x)(\forall w)\,[(w \geq x \Rightarrow w \geq y) \wedge (x \geq w \Rightarrow y \geq w)] \wedge (\exists z)\,[\neg(y = z)]) \Rightarrow \neg(x = z) \end{array}$$

ist eine Formel erster Ordnung. Für das Relationssymbol $\geq$ wird hier die übliche Infixnotation, also $x \geq y$, statt der Präfixnotation $\geq (x,y)$ verwendet. Alle Vorkommen der Variablen w in der Formel φ sind durch den Allquantor $\forall$ gebunden, die Variable y kommt in φ nur frei vor und die Variablen x und z kommen in φ sowohl gebunden als auch frei vor.

Die Semantik einer logischen Formel ist immer im Zusammenhang mit einer relationalen Struktur zu interpretieren. Bevor wir unsere Formeln erweitern, möchten wir uns einige der wichtigsten relationalen Strukturen etwas detaillierter anschauen.

Beispiel 4.27 (relationale boolesche Struktur). Sei $S = (D, \mathscr{R}^S)$ eine relationale Struktur über der Signatur $\mathscr{S} = \{T, F\}$ mit $D = \{0,1\}$, $\mathscr{R}^S = \{T^S, F^S\}$, $\text{ar}(T) = \text{ar}(F) = 0$, $T^S = 1$ und $F^S = 0$. Betrachte die folgende Formel erster Ordnung:

relationale boolesche Struktur

$$\begin{aligned}\varphi(x_1,x_2) \;=\; \exists x_3\, \forall x_4\, (\; &((x_1 = T^S) \vee (x_3 = T^S) \vee (x_4 = T^S)) \wedge \\ &((x_2 = T^S) \vee (x_3 = F^S) \vee (x_4 = T^S)) \wedge \\ &((x_1 = F^S) \vee (x_2 = F^S) \vee (x_4 = T^S))\;).\end{aligned}$$

Die Formel $\varphi(x_1,x_2)$ besitzt zwei Lösungen, $(0,1)$ und $(1,0)$, d. h., die Lösungsmenge $L^S(\varphi(x_1,x_2))$ ist $\{(0,1),(1,0)\}$. Sie kann wie folgt schrittweise bestimmt werden:

$$\begin{array}{lll} L^S(x_1 = T^S) = \{(1)\} & L^S(x_3 = T^S) = \{(1)\} & L^S(x_4 = T^S) = \{(1)\} \\ L^S(x_2 = T^S) = \{(1)\} & L^S(x_3 = F^S) = \{(0)\} & \\ L^S(x_1 = F^S) = \{(0)\} & L^S(x_2 = F^S) = \{(0)\} & \end{array}$$

$$\begin{aligned}\varphi_1(x_1,x_3,x_4) &= (x_1 = T^S) \vee (x_3 = T^S) \vee (x_4 = T^S) \\ L^S(\varphi_1(x_1,x_3,x_4)) &= \{(0,0,1),(0,1,0),(0,1,1),(1,0,0),\\ &\qquad (1,0,1),(1,1,0),(1,1,1)\}\end{aligned}$$

$$\begin{aligned}\varphi_2(x_2,x_3,x_4) &= (x_2 = T^S) \vee (x_3 = F^S) \vee (x_4 = T^S) \\ L^S(\varphi_1(x_2,x_3,x_4)) &= \{(0,0,0),(0,0,1),(0,1,1),(1,0,0),\\ &\qquad (1,0,1),(1,1,0),(1,1,1)\}\end{aligned}$$

$$\begin{aligned}\varphi_3(x_1,x_2,x_4) &= (x_1 = F^S) \vee (x_2 = F^S) \vee (x_4 = T^S) \\ L^S(\varphi_1(x_1,x_2,x_4)) &= \{(0,0,0),(0,0,1),(0,1,0),(0,1,1),\\ &\qquad (1,0,0),(1,0,1),(1,1,1)\}\end{aligned}$$

$$\begin{aligned}\varphi'(x_1,x_2,x_3,x_4) &= \varphi_1(x_1,x_3,x_4) \wedge \varphi_2(x_2,x_3,x_4) \wedge \varphi_3(x_1,x_2,x_4) \\ L^S(\varphi'(x_1,x_2,x_3,x_4)) &= \{(0,0,0,1),(0,0,1,1),(0,1,0,1),(0,1,1,0),(0,1,1,1),\\ &\qquad (1,0,0,0),(1,0,0,1),(1,0,1,1),(1,1,0,1),(1,1,1,1)\}\end{aligned}$$

$$L^S(\forall x_4\ \varphi'(x_1,x_2,x_3,x_4)) = \{(0,1,1),(1,0,0)\}$$

$$L^S(\exists x_3 \forall x_4\ \varphi'(x_1,x_2,x_3,x_4)) = \{(0,1),(1,0)\}.$$

Das Paar $(0,1)$ ist eine Lösung für φ, da für $x_1 = 0$ und $x_2 = 1$ ein $x_3 = 1$ existiert, sodass für $x_4 = 0$ und $x_4 = 1$ alle drei Disjunktionen erfüllt sind. Das Paar $(1,0)$ ist eine Lösung, da für $x_1 = 1$ und $x_2 = 0$ ein $x_3 = 0$ existiert, sodass für $x_4 = 0$ und $x_4 = 1$ alle drei Disjunktionen erfüllt sind. Die anderen Paare, $(0,0)$ und $(1,1)$, sind keine Lösungen, da in diesem Fall für $x_4 = 0$ mindestens eine der drei Disjunktionen nicht erfüllt ist.

Formeln erster Ordnung über der obigen relationalen Struktur S sind nichts anderes als boolesche Ausdrücke mit Quantifizierungen. Schreiben wir zum Beispiel x_i für $x_i = T^S$ und $\neg x_i$ für $x_i = F^S$, dann sieht die obige Formel wie folgt aus:

$$\varphi(x_1,x_2) \;=\; \exists x_3\, \forall x_4\, ((x_1 \vee x_3 \vee x_4) \wedge (x_2 \vee \neg x_3 \vee x_4) \wedge (\neg x_1 \vee \neg x_2 \vee x_4)).$$

Relation
Funktion

Beispiel 4.28 (Funktion). Sei $S = (D, \mathscr{R}^S)$ eine relationale Struktur über der Signatur $\mathscr{S} = \{f\}$ mit $\mathscr{R}^S = \{f^S\}$ und $\mathrm{ar}(f) = 2$. Dann ist die Relation f^S genau dann eine Funktion, wenn die Struktur S die folgende Formel erfüllt:

$$\varphi \;=\; \forall x\, \forall y\, \forall z\, ((f(x,y) \wedge f(x,z)) \;\Rightarrow\; (y = z)),$$

d. h., f ist genau dann eine Funktion, wenn die Lösungsmenge $L^S(\varphi)$ nicht leer ist. Da φ keine freien Variablen hat, kann $L^S(\varphi)$ höchstens das leere Tupel $()$ enthalten.

Seien zum Beispiel $D = \{0,1\}$ und $f = \{(0,1),(1,0)\}$. Dann erhalten wir die folgenden Lösungsmengen:

$$\begin{aligned}
&L^S(f(x,y)) && = \{(0,1),(1,0)\}\\
&L^S(f(x,z)) && = \{(0,1),(1,0)\}\\
&L^S(y = z) && = \{(0,0),(1,1)\}\\
&L^S(f(x,y) \wedge f(x,z)) && = \{(0,1,1),(1,0,0)\}\\
&L^S(((f(x,y) \wedge f(x,z)) \Rightarrow (y = z)) && = \{(0,0,0),(0,0,1),(0,1,0),\\
& && \quad (0,1,1),(1,0,0),(1,0,1),\\
& && \quad (1,1,0),(1,1,1)\}\\
&L^S(\forall z\, (f(x,y) \wedge f(x,z)) \Rightarrow (y = z)) && = \{(0,0),(0,1),(1,0),(1,1)\}\\
&L^S(\forall y\, \forall z\, (f(x,y) \wedge f(x,z)) \Rightarrow (y = z)) && = \{(0),(1)\}\\
&L^S(\forall x\, \forall y\, \forall z\, (f(x,y) \wedge f(x,z)) \Rightarrow (y = z)) && = \{()\}.
\end{aligned}$$

Für $f = \{(0,0),(0,1)\}$ hingegen entsteht eine leere Lösungsmenge für φ:

$$\begin{aligned}
&L^S(f(x,y)) && = \{(0,0),(0,1)\}\\
&L^S(f(x,z)) && = \{(0,0),(0,1)\}\\
&L^S(y = z) && = \{(0,0),(1,1)\}\\
&L^S(f(x,y) \wedge f(x,z)) && = \{(0,0,0),(0,0,1),\\
& && \quad (0,1,0),(0,1,1)\}\\
&L^S(((f(x,y) \wedge f(x,z)) \Rightarrow (y = z)) && = \{(0,0,0),(0,1,1),(1,0,0),\\
& && \quad (1,0,1),(1,1,0),(1,1,1)\}\\
&L^S(\forall z\, (f(x,y) \wedge f(x,z)) \Rightarrow (y = z)) && = \{(1,0),(1,1)\}\\
&L^S(\forall y\, \forall z\, (f(x,y) \wedge f(x,z)) \Rightarrow (y = z)) && = \{(1)\}\\
&L^S(\forall x\, \forall y\, \forall z\, (f(x,y) \wedge f(x,z)) \Rightarrow (y = z)) && = \{\}.
\end{aligned}$$

Die Funktion f ist surjektiv und injektiv und damit bijektiv, wenn S zusätzlich die Formeln

$$\forall x\, \exists y\, f(x,y)$$

und

$$\forall x\, \forall y\, \forall z\, ((f(x,z) \wedge f(y,z)) \;\Rightarrow\; (x = y))$$

erfüllt.

relationale Wortstruktur

Beispiel 4.29 (relationale Wortstruktur). Eine endliche Folge w von n Buchstaben aus einem endlichen Alphabet Σ, geschrieben $w \in \Sigma^*$, kann als eine Abbildung $f_w : \{1,\ldots,n\} \to \Sigma$ aufgefasst werden. Der i-te Buchstabe im Wort w ist genau dann der

Buchstabe a, wenn $f_w(i) = a$ gilt. Jedes Wort w über $\Sigma = \{a_1, \ldots, a_k\}$ kann als eine relationale Struktur $\lfloor w \rfloor$

$$\lfloor w \rfloor \;=\; (D, \{\mathrm{suc}^{\lfloor w \rfloor}, \mathrm{lab}_{a_1}^{\lfloor w \rfloor}, \ldots, \mathrm{lab}_{a_k}^{\lfloor w \rfloor}\})$$

über der Signatur

$$\mathscr{S} = \{\mathrm{suc}, \mathrm{lab}_{a_1}, \ldots, \mathrm{lab}_{a_k}\}$$

repräsentiert werden mit $\mathrm{ar}(\mathrm{suc}) = 2$ und $\mathrm{ar}(\mathrm{lab}_{a_i}) = 1$ für $a_i \in \Sigma$. Für ein Wort w mit n Zeichen ist

1. $D \;=\; \{1, \ldots, n\}$,
2. $\mathrm{suc}^{\lfloor w \rfloor} \;=\; \{(1,2), (2,3), \ldots, (n-1, n)\}$ und
3. $\mathrm{lab}_{a_i}^{\lfloor w \rfloor} \;=\; \{i \in \mathbb{N} \mid 1 \leq i \leq n,$ der i-te Buchstabe in w ist $a_i\}$ für $a_i \in \Sigma$.

In einem Wort $w \in \Sigma^*$ mit $\Sigma = \{a, b\}$ stehen an einer Stelle genau dann drei gleiche Buchstaben nebeneinander, wenn die relationale Struktur $\lfloor w \rfloor$ die folgende Formel erfüllt:

$$\varphi = \exists x\, \exists y\, \exists z \left(\begin{array}{l} \mathrm{suc}(x,y) \wedge \mathrm{suc}(y,z) \wedge \\ \left(\begin{array}{l} (\mathrm{lab}_a(x) \wedge \mathrm{lab}_a(y) \wedge \mathrm{lab}_a(z)) \vee \\ (\mathrm{lab}_b(x) \wedge \mathrm{lab}_b(y) \wedge \mathrm{lab}_b(z)) \end{array} \right) \end{array} \right).$$

Sei zum Beispiel $w = abbbaa$. Dann ist

1. $D \;=\; \{1, \ldots, 6\}$,
2. $\mathrm{suc}^{\lfloor abbbaa \rfloor} \;=\; \{(1,2), (2,3), (3,4), (4,5), (5,6)\}$,
3. $\mathrm{lab}_a^{\lfloor abbbaa \rfloor} \;=\; \{1, 5, 6\}$,
4. $\mathrm{lab}_b^{\lfloor abbbaa \rfloor} \;=\; \{2, 3, 4\}$,
5. $L^{\lfloor abbbaa \rfloor}(\mathrm{suc}(x,y) \wedge \mathrm{suc}(y,z)) = \{(1,2,3), (2,3,4), (3,4,5), (4,5,6)\}$,
6. $L^{\lfloor abbbaa \rfloor} \left(\begin{array}{l} (\mathrm{lab}_a(x) \wedge \mathrm{lab}_a(y) \wedge \mathrm{lab}_a(z)) \vee \\ (\mathrm{lab}_b(x) \wedge \mathrm{lab}_b(y) \wedge \mathrm{lab}_b(z)) \end{array} \right) =$
$$\left\{ \begin{array}{l} (1,1,1), (1,1,5), (1,1,6), (1,5,5), (1,5,6), (1,6,6), (2,2,2), \\ (2,2,3), (2,2,4), (2,3,3), (2,3,4), (2,4,4), (3,3,3), (3,3,4), \\ (3,4,4), (4,4,4), (5,5,5), (5,5,6), (5,6,6), (6,6,6) \end{array} \right\},$$
7. $L^{\lfloor abbbaa \rfloor} \left(\begin{array}{l} \mathrm{suc}(x,y) \wedge \mathrm{suc}(y,z) \wedge \\ \left(\begin{array}{l} (\mathrm{lab}_a(x) \wedge \mathrm{lab}_a(y) \wedge \mathrm{lab}_a(z)) \vee \\ (\mathrm{lab}_b(x) \wedge \mathrm{lab}_b(y) \wedge \mathrm{lab}_b(z)) \end{array} \right) \end{array} \right) = \{(2,3,4)\}$ und
8. $L^{\lfloor abbbaa \rfloor}(\varphi) = \{()\}$.

Beispiel 4.30 (relationale Graphenstruktur). Ein gerichteter Graph $G = (V_G, E_G)$ kann wie folgt als eine relationale Struktur $\lfloor G \rfloor \;=\; (D, \{E^{\lfloor G \rfloor}\})$ über der Signatur $\{E\}$ mit der Definitionsmenge $D = V_G$ und der binären Relation $E^{\lfloor G \rfloor} = E_G$ mit $\mathrm{ar}(E) = 2$ aufgefasst werden. Für ungerichtete Graphen fordern wir zusätzlich $(u,v) \in E^{\lfloor G \rfloor} \Rightarrow (v,u) \in E^{\lfloor G \rfloor}$. In diesem Fall ist die Relation $E^{\lfloor G \rfloor}$ symmetrisch. relationale Graphenstruktur $\lfloor G \rfloor$

Sei zum Beispiel G ein gerichteter Graph und

$$\text{outdegree}_{\geq 3}(x) \;=\; \exists x_1\, \exists x_2\, \exists x_3 \left(\begin{array}{l} \neg(x_1 = x_2) \wedge \neg(x_1 = x_3) \wedge \neg(x_2 = x_3) \\ \wedge E(x,x_1) \wedge E(x,x_2) \wedge E(x,x_3) \end{array} \right).$$

Die Lösungsmenge $L^{\lfloor G \rfloor}(\text{outdegree}_{\geq 3}(x))$ enthält genau dann ein Tupel (u), wenn u ein Knoten mit mindestens drei auslaufenden Kanten ist.

In ähnlicher Weise kann für jedes beliebige $k > 0$ eine Formel $\text{outdegree}_{\geq k}(x)$ für Knoten mit mindestens k auslaufenden Kanten aufgestellt werden. Die genaue Anzahl k von auslaufenden Kanten kann mit der folgenden Formel erfasst werden:

$$\text{outdegree}_{=k}(x) \;=\; (\text{outdegree}_{\geq k}(x) \wedge \neg \text{outdegree}_{\geq k+1}(x)).$$

Entsprechende Formeln für einlaufende Kanten lassen sich analog definieren.

Ein ungerichteter Graph G ist genau dann ein vollständiger Graph, wenn $\lfloor G \rfloor$ die folgende Formel erfüllt:

$$\text{complete-graph}() \;=\; \forall x_1\, \forall x_2\, (\neg(x_1 = x_2) \Rightarrow E(x_1,x_2)).$$

Ein ungerichteter Graph G hat genau dann keinen induzierten Weg der Länge 3, wenn $\lfloor G \rfloor$ folgende Formel[20] erfüllt:

$$\text{cograph} \;=\; \forall x_1\, \forall x_2\, \forall x_3\, \forall x_4 \left(\begin{array}{l} (E(x_1,x_2) \wedge E(x_2,x_3) \wedge E(x_3,x_4)) \\ \Rightarrow (E(x_1,x_3) \vee E(x_1,x_4) \vee E(x_2,x_4)) \end{array} \right).$$

4.6 Logik zweiter Ordnung

Viele Eigenschaften einer relationalen Struktur $S = (D, \mathscr{R}^S)$ lassen sich mit Hilfe der Logik nur dann beschreiben, wenn die verwendeten Variablen nicht nur elementare Objekte der Grundmenge D, sondern auch mehrstellige Relationen über D oder zumindest Teilmengen von D repräsentieren können. Diese Erweiterung der Logik führt zu den *Formeln zweiter Ordnung*.

Relationsvariable
atomare Formel zweiter Ordnung
Formel zweiter Ordnung

Definition 4.31 (Formel zweiter Ordnung). *Sei $\mathscr{X}$ eine Menge von Symbolen für Variablen erster Ordnung und Relationsvariablen und sei $\mathscr{S}$ eine Signatur. Jede Relationsvariable X besitzt wie jedes Relationssymbol eine* Stelligkeit *$ar(X) \geq 1$. Eine atomare Formel zweiter Ordnung über der Signatur $\mathscr{S}$ ist entweder eine atomare Formel erster Ordnung über der Signatur $\mathscr{S}$ oder von der Form $X(x_1,\ldots,x_n)$ mit Variablen $x_1,\ldots,x_n$ erster Ordnung und einer Relationsvariablen X mit $ar(X) = n$. Die* Formeln zweiter Ordnung *werden wie folgt aus den atomaren Formeln zweiter Ordnung, den Symbolen für die booleschen Verknüpfungen $\wedge$, $\vee$ und $\neg$ sowie den Symbolen für die Quantoren $\exists$ und $\forall$ gebildet.*

1. Jede atomare Formel zweiter Ordnung ist eine Formel zweiter Ordnung.

[20] Ein ungerichteter Graph, der keinen induzierten Weg mit vier Knoten enthält, wird auch *Co-Graph* genannt, siehe Kapitel 3 und 9. Was hier formal in einer geeigneten logischen Struktur ausgedrückt wird, ist genau das, was anders bereits in Kapitel 3 definiert wurde.

2. *Sind* φ_1, φ_2 *und* φ' *Formeln zweiter Ordnung, dann sind auch* $(\varphi_1 \wedge \varphi_2)$, $(\varphi_1 \vee \varphi_2)$, $(\neg\ \varphi')$, $(\exists x\ \varphi')$, $(\forall x\ \varphi')$, $(\exists X\ \varphi)$ *und* $(\forall X\ \varphi)$ *Formeln zweiter Ordnung, wobei x ein Symbol für eine* Variable erster Ordnung *und X ein Symbol für eine* Relationsvariable *sind.*

Für Variablen erster Ordnung verwenden wir kleine Buchstaben, für Relationsvariablen große Buchstaben. Die Klasse aller Formeln zweiter Ordnung bezeichnen wir mit SO *(englisch:* second order*).* SO

Die Semantik einer Formel φ zweiter Ordnung für eine relationale Struktur $S = (D, \mathscr{R}^S)$ ist analog zur Semantik einer Formel erster Ordnung definiert. Der Unterschied besteht lediglich darin, dass der Wert einer Relationsvariablen X eine Teilmenge von $D^{\mathrm{ar}(X)}$ und nicht ein Element der Menge D ist.

Definition 4.32 (Semantik einer Formel zweiter Ordnung). *Sei* φ *eine Formel zweiter Ordnung über einer Signatur* $\mathscr{S}$ *und sei* $S = (D, \mathscr{R}^S)$ *eine relationale Struktur über* $\mathscr{S}$*. Die Definition der Menge aller Lösungen* $L^S(\varphi)$ *für Formeln erster Ordnung wird wie folgt auf Formeln* φ *zweiter Ordnung erweitert.*

1. *Ist* $\varphi(x_1, \ldots, x_n, X)$ *eine atomare Formel zweiter Ordnung der Form*

$$X(x_{i_1}, \ldots, x_{i_k})$$

mit $i_1, \ldots, i_k \in [n]$ *und der Relationsvariablen X mit* $ar(X) = k$*, dann ist*

$$L^S(\varphi) \;=\; \{(a_1, \ldots, a_n, A) \mid a_1, \ldots, a_n \in D, A \subseteq D^{ar(X)}, (a_{i_1}, \ldots, a_{i_k}) \in A\}.$$

2. *Ist* $\varphi(x_1, \ldots, x_n, X_1, \ldots, X_{n'})$ *eine Formel zweiter Ordnung der Form*

$$(\neg \varphi'(x_{i_1}, \ldots, x_{i_k}, X_{i'_1}, \ldots, X_{i'_{k'}}))$$

mit $i_1, \ldots, i_k \in [n]$ *und* $i'_1, \ldots, i'_{k'} \in [n']$*, dann ist*

$$\begin{aligned} L^S(\varphi) \;=\; \{(a_1, \ldots, a_n, A_1, \ldots, A_{n'}) \mid\ & a_1, \ldots, a_n \in D, \\ & A_1 \subseteq D^{ar(X_1)}, \ldots, A_{n'} \subseteq D^{ar(X_{n'})}, \\ & (a_{i_1}, \ldots, a_{i_k}, A_{i'_1}, \ldots, A_{i'_{k'}}) \notin L^S(\varphi')\}. \end{aligned}$$

3. *Ist* $\varphi(x_1, \ldots, x_n, X_1, \ldots, X_{n'})$ *eine Formel zweiter Ordnung der Form*

$$(\varphi_1(x_{i_1}, \ldots, x_{i_k}, X_{i'_1}, \ldots, X_{i'_{k'}}) \wedge \varphi_2(x_{j_1}, \ldots, x_{j_\ell}, X_{j'_1}, \ldots, X_{j'_{\ell'}}))$$

mit $i_1, \ldots, i_k,\ j_1, \ldots, j_\ell \in [n]$ *und* $i'_1, \ldots, i'_{k'},\ j'_1, \ldots, j'_{\ell'} \in [n']$*, dann ist*

$$\begin{aligned} L^S(\varphi) \;=\; \{(a_1, \ldots, a_n, A_1, \ldots, A_{n'}) \mid\ & a_1, \ldots, a_n \in D, \\ & A_1 \subseteq D^{ar(X_1)}, \ldots, A_{n'} \subseteq D^{ar(X_{n'})}, \\ & (a_{i_1}, \ldots, a_{i_k}, A_{i'_1}, \ldots, A_{i'_{k'}}) \in L^S(\varphi_1) \wedge \\ & (a_{j_1}, \ldots, a_{j_\ell}, A_{j'_1}, \ldots, A_{j'_{\ell'}}) \in L^S(\varphi_2)\}. \end{aligned}$$

Für die boolesche Verknüpfung $\vee$ *ist* $L^S(\varphi)$ *analog definiert.*

4. Ist $\varphi(x_1,\dots,x_n,X_1,\dots,X_{n'})$ *eine Formel zweiter Ordnung der Form*

$$\exists x_{n+1}\ \varphi'(x_{i_1},\dots,x_{i_k},X_{i'_1},\dots,X_{i'_{k'}})$$

mit $i_1,\dots,i_k \in [n+1]$ *und* $i'_1,\dots,i'_{k'} \in [n']$*, dann ist*

$$L^S(\varphi(x_1,\dots,x_n,X_1,\dots,X_{n'})) \;=$$

$$\left\{ (a_1,\dots,a_n,A_1,\dots,A_{n'}) \;\middle|\; \begin{array}{l} a_1,\dots,a_n \in D, \\ A_1 \subseteq D^{ar(X_1)},\dots,A_{n'} \subseteq D^{ar(X_{n'})}, \\ \exists a_{n+1} \in D : (a_{i_1},\dots,a_{i_k},A_{i'_1},\dots,A_{i'_{k'}}) \in L^S(\varphi') \end{array} \right\}.$$

Für den Allquantor $\forall$ *ist* $L^S(\varphi)$ *analog definiert.*
Bei Quantifizierungen mit einer Relationsvariablen $X_{n'+1}$ *gilt für die Indizierungen* $i_1,\dots,i_k \in [n]$ *und* $i'_1,\dots,i'_{k'} \in [n'+1]$*, und für die möglichen Werte der hinzugefügten gebundenen Relationsvariablen gilt* $A_{n'+1} \subseteq D^{ar(X_{n'+1})}$.

Übung 4.33. Sei $\varphi(x_1,\dots,x_n,X_1,\dots,X_{n'})$ eine Formel zweiter Ordnung der Form

$$\exists X_{n'+1}\ \varphi'(x_{i_1},\dots,x_{i_k},X_{i'_1},\dots,X_{i'_{k'}})$$

mit $i_1,\dots,i_k \in [n]$ und $i'_1,\dots,i'_{k'} \in [n'+1]$. Definieren Sie formal die Lösungsmenge

$$L^S(\varphi(x_1,\dots,x_n,X_1,\dots,X_{n'})).$$

In (4.2) und (4.3) wurde bereits die Schreibweise $\bigvee_{i=1}^n \varphi_i$ für $\varphi_1 \vee \varphi_2 \vee \dots \vee \varphi_n$ und die Schreibweise $\bigwedge_{i=1}^n \varphi_i$ für $\varphi_1 \wedge \varphi_2 \wedge \dots \wedge \varphi_n$ eingeführt. Zur weiteren Vereinfachung schreiben wir:

$\neq$
$\forall x_1,\dots,x_n$
$\exists x_1,\dots,x_n$
$\bigwedge_{i=1}^n \varphi_i$
$\bigvee_{i=1}^n \varphi_i$

$$(x \neq y) \qquad \text{für } \neg(x = y),$$

$$\forall x_1,\dots,x_n \quad \text{für } \forall x_1\, \forall x_2\, \cdots \forall x_n \text{ und}$$

$$\exists x_1,\dots,x_n \quad \text{für } \exists x_1\, \exists x_2\, \cdots \exists x_n.$$

Nützlich sind auch die folgenden Abkürzungen:

$$X = \emptyset \qquad \text{für } \forall x\, \neg X(x),$$

$$X \subseteq Y \qquad \text{für } \forall x\, (X(x) \Rightarrow Y(x)),$$

$$X = Y \qquad \text{für } \forall x\, (X(x) \Leftrightarrow Y(x)),$$

$$Z = X \cup Y \quad \text{für } \forall x\, (Z(x) \Leftrightarrow (X(x) \vee Y(x))),$$

$$Z = X \cap Y \quad \text{für } \forall x\, (Z(x) \Leftrightarrow (X(x) \wedge Y(x))) \text{ und}$$

$$Z = X - Y \quad \text{für } \forall x\, (Z(x) \Leftrightarrow (X(x) \wedge \neg Y(x))).$$

In den folgenden Beispielen werden einige nützliche Eigenschaften mit Formeln zweiter Ordnung definiert.

Beispiel 4.34 (gerade Anzahl von Elementen). Betrachte die folgenden Formeln $\varphi_{\geq 1}(x,X)$, $\varphi_{\geq 2}(x,X)$ und φ über der leeren Signatur $\mathscr{S} = \emptyset$, wobei X eine binäre Relationsvariable ist:

$$\begin{aligned}
\varphi_{\geq 1}(x,X) &= \exists y\,(X(x,y) \vee X(y,x)),\\
\varphi_{\geq 2}(x,X) &= X(x,x) \vee \exists y_1, y_2 \begin{pmatrix} (X(x,y_1) \wedge X(y_2,x)) \vee \\ ((y_1 \neq y_2) \wedge X(x,y_1) \wedge X(x,y_2)) \vee \\ ((y_1 \neq y_2) \wedge X(y_1,x) \wedge X(y_2,x)) \end{pmatrix},\\
\varphi &= \exists X\ \forall x\,(\varphi_{\geq 1}(x,X) \wedge \neg\varphi_{\geq 2}(x,X)).
\end{aligned}$$

Sei $S = (D,\emptyset)$ eine relationale Struktur über der leeren Signatur $\mathscr{S} = \emptyset$. Dann enthält die Lösungsmenge $L^S(\varphi_{\geq 1}(x,X))$ bzw. $L^S(\varphi_{\geq 2}(x,X))$ alle Paare (a,A), sodass a mindestens einmal bzw. mindestens zweimal in den Paaren der binären Relation A vorkommt. Die Definitionsmenge D besitzt genau dann eine gerade Anzahl von Elementen, wenn S die obige Formel φ erfüllt. Analog lassen sich auch Formeln $\varphi(X)$ mit $\mathrm{ar}(X) = 1$ definieren, sodass $L^S(\varphi(X))$ alle Teilmengen D' von D enthält, deren Größe ein Vielfaches einer ganzen positiven Zahl $k \in \mathbb{N}$ ist.

Beispiel 4.35 (Färbbarkeit von Knoten in Graphen). Sei $G = (V,E)$ ein ungerichteter Graph. Wie wir aus Kapitel 3 wissen, ist eine Teilmenge der Knoten von G eine *unabhängige Menge*, wenn sie keine zwei adjazente Knoten enthält; und G ist *dreifärbbar*, wenn seine Knotenmenge vollständig in höchstens drei unabhängige Mengen aufgeteilt werden kann. Diese Eigenschaften können mit den folgenden Formeln zweiter Ordnung für relationale Strukturen $\lfloor G \rfloor$ über der Signatur $\mathscr{S} = \{E\}$ definiert werden:

$$\begin{aligned}
\text{independent-set}(X) &= \forall x,y\,(X(x) \wedge X(y) \Rightarrow \neg E(x,y)),\\
\text{3-colorable}() &= \exists X_1, X_2, X_3 \begin{pmatrix} (\forall x\,(X_1(x) \vee X_2(x) \vee X_3(x))) \\ \wedge\ \text{independent-set}(X_1) \\ \wedge\ \text{independent-set}(X_2) \\ \wedge\ \text{independent-set}(X_3) \end{pmatrix}.
\end{aligned}$$

Hier ist es nicht notwendig zu überprüfen, ob die drei Mengen auch disjunkt sind, da die Knoten, die sich in mehreren Mengen befinden, beliebig zugeordnet werden können.

Beispiel 4.36 (Weg). Sei $G = (V,E)$ ein gerichteter Graph. Betrachte die folgende Formel zweiter Ordnung über der Signatur $\mathscr{S} = \{E\}$:

$$\mathrm{path}(x,y) = \forall X\,(X(x) \wedge \neg X(y)) \Rightarrow (\exists x', y'\,(X(x') \wedge \neg X(y') \wedge E(x',y'))).$$

Dann enthält die Lösungsmenge $L^{\lfloor G \rfloor}(\mathrm{path}(x,y))$ alle Knotenpaare, zwischen denen es einen Weg in G gibt.

G ist genau dann *stark zusammenhängend*, wenn $\lfloor G \rfloor$ die folgende Formel erfüllt:

$$\text{strong-connected}() = \forall x,y\ \mathrm{path}(x,y).$$

Beispiel 4.37 (Hamilton-Weg). Sei $S = (D, \emptyset)$ eine relationale Struktur über der leeren Signatur $\mathscr{S} = \emptyset$. Eine binäre Relation $R \subseteq D^2$ ist eine totale Ordnung auf D, wenn für alle Elemente $a, b \in D$, $a \neq b$, entweder $(a,b) \in R$ oder $(b,a) \in R$ gilt und R transitiv, aber nicht reflexiv ist. Die Lösungsmenge $L^S(\text{total-order}(X))$ für die folgenden Formel total-order(X) über der leeren Signatur $\mathscr{S} = \emptyset$ enthält alle totalen Ordnungen auf D:

$$\begin{aligned}\text{total-order}(X) \;=\; & (\forall x, y\, ((X(x,y) \vee X(y,x) \vee (x = y))) \wedge \\ & (\forall x, y, z\, (((X(x,y) \wedge X(y,z)) \Rightarrow X(x,z))) \wedge \\ & (\forall x\, \neg X(x,x)).\end{aligned}$$

Zwei Elemente, a und b, sind in einer totalen Ordnung R benachbart, wenn es kein Element c mit $(a,c) \in R$ und $(c,b) \in R$ gibt:

$$\text{neighbour}(x, y, X) \;=\; \neg(\exists z\, (X(x,z) \wedge X(z,y))).$$

Ein gerichteter Graph $G = (V, E)$ besitzt einen *Hamilton-Weg*, wenn es eine totale Ordnung auf der Knotenmenge V gibt, sodass benachbarte Knoten mit einer Kante verbunden sind (siehe auch Abschnitt 3.4). Er besitzt also genau dann einen Hamilton-Weg, wenn $\lfloor G \rfloor$ die folgende Formel erfüllt:

$$\text{Hamilton-path}() \;=\; \exists X\, (\text{total-order}(X) \wedge \forall x, y\, (\text{neighbour}(x, y, X) \Rightarrow E(x,y))).$$

Übung 4.38. In Kapitel 3 wurde der Begriff des Hamilton-Kreises für Graphen definiert. Beschreiben Sie diese Eigenschaft durch eine Formel zweiter Ordnung.

4.7 Monadische Logik zweiter Ordnung

Wenn in den Formeln zweiter Ordnung nur einstellige Relationsvariablen verwendet werden, dann sprechen wir von der monadischen Logik zweiter Ordnung.

monadische Logik zweiter Ordnung

MSO

Definition 4.39 (monadische Logik zweiter Ordnung). *Eine Formel φ zweiter Ordnung ist eine Formel in* monadischer Logik zweiter Ordnung*, wenn jede freie und jede gebundene Variable in φ entweder eine Variable erster Ordnung oder eine Relationsvariable X mit $ar(X) = 1$ ist. Die Klasse aller Formeln in monadischer Logik zweiter Ordnung bezeichnen wir mit* MSO *(englisch:* monadic second order*).*

Mengenvariable

Die Beschreibungskraft der monadischen Logik zweiter Ordnung liegt zwischen der Beschreibungskraft der Logik erster Ordnung und der der Logik zweiter Ordnung. Einstellige Relationen sind Mengen. Daher nennen wir die Relationsvariablen in der monadischen Logik auch *Mengenvariablen*. Die Symbole der Signatur $\mathscr{S}$ können jedoch eine beliebige Stelligkeit besitzen. Die Formel 3-colorable aus Beispiel 4.35 ist offensichtlich eine Formel in monadischer Logik zweiter Ordnung. Die Formel Hamilton-path aus Beispiel 4.37 ist dagegen nicht in monadischer Logik, da die Relationsvariable X zweistellig ist.

Nicht alle Eigenschaften, die in der Logik zweiter Ordnung beschreibbar sind, lassen sind auch in monadischer Logik zweiter Ordnung definieren. Büchi, Elgot und Trachtenbrot haben Anfang der 1960er Jahre den folgenden fundamentalen Satz bewiesen.

Satz 4.40 (Büchi [Büc60]). *Sei $\Sigma = \{a_1, \ldots, a_k\}$ ein endliches Alphabet. Eine Sprache $L \subseteq \Sigma^*$ ist genau dann* regulär,[21] *wenn es eine Formel φ in monadischer Logik zweiter Ordnung über der Signatur*

$$\mathscr{S} = \{suc, lab_{a_1}, \ldots, lab_{a_k}\}$$

gibt mit

$$w \in L \quad \Leftrightarrow \quad \lfloor w \rfloor \vDash \varphi.$$

ohne Beweis

Die relationale Struktur $\lfloor w \rfloor$ für ein Wort $w \in \Sigma^*$ ist in Beispiel 4.29 definiert. Mit Hilfe dieses Satzes lässt sich nun leicht zeigen, dass viele weitere interessante Eigenschaften nicht in monadischer Logik zweiter Ordnung definiert werden können.

Behauptung 4.41. *Die folgenden Eigenschaften φ sind nicht in monadischer Logik zweiter Ordnung definierbar:*

1. *Sei $S = (D, \emptyset)$ eine relationale Struktur, und es gelte $S \vDash \varphi(X_1, X_2)$ genau dann, wenn die beiden Mengen X_1 und X_2 gleich groß sind.*
2. *Sei $G = (V, E)$ ein ungerichteter Graph, und es gelte $\lfloor G \rfloor \vDash \varphi$ genau dann, wenn G der vollständig bipartite Graph $K_{n,n}$ ist, für ein $n \geq 1$.*
3. *Sei $G = (V, E)$ ein ungerichteter Graph, und es gelte $\lfloor G \rfloor \vDash \varphi$ genau dann, wenn G einen Hamilton-Kreis besitzt.*
4. *Sei $G = (V, E)$ ein ungerichteter Graph, und es gelte $\lfloor G \rfloor \vDash \varphi$ genau dann, wenn G ein perfektes Matching besitzt.*
5. *Sei $G = (V, E)$ ein ungerichteter Graph, und es gelte $\lfloor G \rfloor \vDash \varphi(u, v, X)$ genau dann, wenn X die Menge aller Knoten eines einfachen Wegs von u nach v in G ist.*

Beweis.

1. Eine einfache nicht reguläre Sprache ist zum Beispiel die Menge aller Wörter über dem binären Alphabet $\{a, b\}$, in denen die Anzahl der a's genauso groß ist wie die Anzahl der b's. Wir bezeichnen diese Sprache im Folgenden mit $L^{a=b}$. Sei equal-size(X, Y) eine Formel in monadischer Logik zweiter Ordnung über der leeren Signatur $\mathscr{S} = \emptyset$, die genau dann von einer relationalen Struktur $S = (D, \emptyset)$ erfüllt wird, wenn die beiden Mengenvariablen X und Y gleich groß sind. Dann ist

[21] Eine Sprache $L \subseteq \Sigma^*$ ist *regulär*, wenn es die Sprache eines endlichen Automaten ist oder wenn sie von einer regulären Grammatik erzeugt werden kann, siehe zum Beispiel [HMU02].

$$\varphi = \exists X, Y \; \forall x \; ((X(x) \Leftrightarrow \text{lab}_a(x)) \wedge (Y(x) \Leftrightarrow \text{lab}_b(x)) \wedge \text{equal-size}(X, Y))$$

eine Formel in monadischer Logik zweiter Ordnung über der Signatur

$$\mathscr{S} = \{\text{suc}, \text{lab}_a, \text{lab}_a\},$$

die genau dann von einer relationalen Struktur $\lfloor w \rfloor$ mit $w \in \{a,b\}^*$ erfüllt wird, wenn w ein Wort in der oben definierten nicht regulären Sprache $L^{a=b}$ ist. Dies steht jedoch im Widerspruch zum Satz von Büchi.

2. Für ein Wort $w = w_1 w_2 \cdots w_{2n} \in \{a,b\}^*$ der Länge $n > 0$ sei $G_w = (V,E)$ der Graph mit Knotenmenge $V = \{1, 2, \ldots, 2n\}$ und Kantenmenge

$$E = \{(i,j) \,|\, ((w_i = a) \wedge (w_j = b)) \vee ((w_i = b) \wedge (w_j = a))\}.$$

G_w ist also der vollständig bipartite Graph $K_{i,j}$, wobei i die Anzahl der a's und j die Anzahl der b's im Wort w ist. Offenbar ist G_w genau dann ein $K_{n,n}$, wenn in w die Anzahl der a's genauso groß ist wie die Anzahl der b's, nämlich gleich n. Da diese Sprache nicht regulär ist, gibt es nach dem Satz von Büchi keine Formel φ in monadischer Logik zweiter Ordnung, mit deren Hilfe entschieden werden kann, ob G ein vollständig bipartiter Graph $K_{n,n}$ für ein $n \geq 1$ ist.

3. Der vollständig bipartite Graph $K_{n,m}$, $n,m \geq 2$, hat genau dann einen Hamilton-Kreis, wenn $n = m$ gilt. Somit folgt die Behauptung aus der zweiten Aussage des Satzes.
4. Der vollständig bipartite Graph $K_{n,m}$, $n,m \geq 1$, hat genau dann ein perfektes Matching, wenn $n = m$ gilt. Somit folgt die Behauptung wieder aus der zweiten Aussage des Satzes.
5. Seien $G = (V,E)$ ein ungerichteter Graph und $\varphi(u,v,X)$ eine Formel in monadischer Logik zweiter Ordnung über der Signatur $\mathscr{S} = \{E\}$, sodass die Lösungsmenge $L^{\lfloor G \rfloor}(\varphi(u,v,X))$ alle Tripel (u,v,X) mit der Eigenschaft enthält, dass X die Menge aller Knoten eines einfachen Wegs vom Knoten u zum Knoten v ist. Dann ist

$$\varphi = \exists u, v, X \; ((\forall x \, X(x)) \wedge E(v,u) \wedge \varphi(u,v,X))$$

ebenfalls eine Formel in monadischer Logik zweiter Ordnung. G hat genau dann einen Hamilton-Kreis, wenn $\lfloor G \rfloor \vDash \varphi$ gilt. Die Existenz einer solchen Formel in monadischer Logik zweiter Ordnung widerspricht jedoch der dritten Aussage.

Damit ist der Beweis abgeschlossen. ❑

In unserer bisherigen Repräsentation eines Graphen $G = (V,E)$ als eine relationale Struktur $\lfloor G \rfloor$ über der Signatur $\mathscr{S} = \{E\}$ wird die Kantenmenge von G als eine binäre Relation betrachtet. Diese binäre Relation ist nicht Teil der Definitionsmenge. In der monadischen Logik zweiter Ordnung, in der neben den Variablen erster Ordnung nur Mengenvariablen erlaubt sind, kann somit nicht über Kanten quantifiziert werden. Dieses Problem kann jedoch durch eine etwas andere Repräsentation von Graphen umgangen werden, bei der die Kanten mit in die Definitionsmenge aufgenommen werden.

Beispiel 4.42 (relationale Graphenstruktur vom Typ II). Ein gerichteter Graph $G = (V,E)$ kann wie folgt als relationale Struktur $S = (D, \mathscr{R}^G)$ über der Signatur $\mathscr{S} = \{\overleftarrow{E}, \overrightarrow{E}\}$ repräsentiert werden: relationale Graphenstruktur vom Typ II

$$\begin{aligned} D &= V \cup E \\ \overleftarrow{E}^G &= \{(v,e) \mid v \in V, e \in E, e = (v,u) \text{ für ein } u \in V\} \\ \overrightarrow{E}^G &= \{(e,v) \mid v \in V, e \in E, e = (u,v) \text{ für ein } u \in V\}. \end{aligned}$$

Diese Repräsentation des Graphen $G = (V,E)$ als relationale Struktur

$$S = (D, \{\overleftarrow{E}^G, \overrightarrow{E}^G\})$$

bezeichnen wir mit $\lceil G \rceil$. Eine Variable x erster Ordnung repräsentiert genau dann eine Kante, wenn $\lceil G \rceil \models \exists y\, (\overleftarrow{E}(y,x) \vee \overrightarrow{E}(x,y))$ gilt. Die atomare Formel $E(x,y)$ erster Ordnung über der Signatur $\{E\}$ entspricht daher der Formel $\exists e\ \overleftarrow{E}(x,e) \wedge \overrightarrow{E}(e,y)$ erster Ordnung über der Signatur $\{\overleftarrow{E}, \overrightarrow{E}\}$. $\lceil G \rceil$

Wir verwenden die folgenden Formeln, um in der relationalen Struktur $\lceil G \rceil$ Knoten, Kanten und ungerichtete Kanten zu erkennen:

$\text{d-edge}_1(e) = \exists u\, \overleftarrow{E}(u,e) \vee \overrightarrow{E}(e,u)$		e ist gerichtete Kante
$\text{d-edge}_2(u,v) = \exists e\, \overleftarrow{E}(u,e) \wedge \overrightarrow{E}(e,v)$		gerichtete Kante (u,v)
$\text{d-edge-set}(E') = \forall e\, (E'(e) \Rightarrow \text{d-edge}_1(e))$		E' ist gerichtete Kantenmenge
$\text{u-edge}_1(e) = \exists u\, \overleftarrow{E}(u,e) \wedge \overrightarrow{E}(e,u)$		e ist ungerichtete Kante
$\text{u-edge}_2(u,v) = \text{d-edge}_2(u,v) \wedge \text{d-edge}_2(v,u)$		ungerichtete Kante $\{u,v\}$
$\text{u-edge-set}(E') = \forall e\, (E'(e) \Rightarrow \text{u-edge}_1(e))$		E' ist ungerichtete Kantenmenge
$\text{vertex}(u) = \neg(\text{d-edge}_1(u))$		u ist Knoten
$\text{vertex-set}(U) = \forall u\, (U(u) \Rightarrow \text{vertex}(u))$		U ist Knotenmenge

Sei FO_1, SO_1 bzw. MSO_1 die Klasse aller Formeln, die für Strukturen $\lfloor G \rfloor$ über der Signatur $\{E\}$ in der Logik erster Ordnung, der Logik zweiter Ordnung bzw. in monadischer Logik zweiter Ordnung definiert sind. Sei weiter FO_2, SO_2 bzw. MSO_2 die Klasse aller Formeln, die für Strukturen $\lceil G \rceil$ über der Signatur $\{\overleftarrow{E}, \overrightarrow{E}\}$ in der Logik erster Ordnung, der Logik zweiter Ordnung bzw. in monadischer Logik zweiter Ordnung definiert sind. Da die atomare Formel $E(x,y)$ für Strukturen $\lfloor G \rfloor$ gleichbedeutend mit der Formel $\text{d-edge}_2(x,y)$ für Strukturen $\lceil G \rceil$ ist, gilt offenbar $\text{FO}_1, \text{SO}_1, \text{MSO}_1$ $\text{FO}_2, \text{SO}_2, \text{MSO}_2$

$$\text{FO}_1 \subseteq \text{FO}_2,\ \text{MSO}_1 \subseteq \text{MSO}_2 \ \text{ und } \text{SO}_1 \subseteq \text{SO}_2.$$

Eine Quantifizierung der Art $\exists x$ oder $\forall x$ für eine Kantenvariable x in einer Formel erster Ordnung für $\lceil G \rceil$ kann durch eine Quantifizierung der Art $\exists x,y\, E(x,y)$ bzw. $\forall x,y\, E(x,y)$ für Knotenvariablen x und y in einer Formel erster Ordnung für $\lfloor G \rfloor$ ausgedrückt werden. Relationsvariablen mit einer Stelligkeit von n lassen sich so durch Relationsvariablen mit der Stelligkeit $2n$ darstellen. Daraus folgt

$$\mathrm{FO}_2 \subseteq \mathrm{FO}_1 \text{ und } \mathrm{SO}_2 \subseteq \mathrm{SO}_1$$

und somit

$$\mathrm{FO}_1 = \mathrm{FO}_2 \text{ und } \mathrm{SO}_1 = \mathrm{SO}_2.$$

Die Klasse MSO_2 enthält jedoch Eigenschaften, die nicht in MSO_1 sind. Dies folgt beispielsweise aus Behauptung 4.41 zusammen mit Behauptung 4.43 unten. Daraus ergibt sich der in Abb. 4.3 dargestellte Gesamtzusammenhang.

$$\begin{array}{ccccc} \mathrm{FO}_2 & \subseteq & \mathrm{MSO}_2 & \subseteq & \mathrm{SO}_2 \\ \| & & \cup & & \| \\ \mathrm{FO}_1 & \subseteq & \mathrm{MSO}_1 & \subseteq & \mathrm{SO}_1 \end{array}$$

Abb. 4.3. Inklusionen zwischen FO_1, FO_2, MSO_1, MSO_2, SO_1 und SO_2

Behauptung 4.43. *Sei $G = (V,E)$ ein gerichteter Graph. Die Eigenschaft, ob G einen Hamilton-Kreis hat, kann für die relationale Struktur $\lceil G \rceil$ in monadischer Logik zweiter Ordnung definiert werden.*

Beweis. Offensichtlich ist ein gerichteter Graph $G = (V,E)$ genau dann ein Kreis, wenn erstens jeder Knoten genau eine auslaufende Kante hat und zweitens für jede echte nicht leere Teilmenge $U \subseteq V$ der Knoten eine Kante von einem Knoten aus U zu einem Knoten außerhalb von U existiert. Ein gerichteter Graph besitzt genau dann einen Hamilton-Kreis, wenn es eine Teilmenge $E' \subseteq E$ der Kanten gibt, sodass der Graph (V,E') ein Kreis ist. Dies kann mit den folgenden Formeln in monadischer Logik zweiter Ordnung über der Signatur $\mathscr{S} = \{\overleftarrow{E}, \overrightarrow{E}\}$ definiert werden:

$$\begin{aligned}
\mathrm{outdeg}_{\geq 1}(u,E') &= \exists e\, (E'(e) \wedge \overleftarrow{E}(u,e)) \\
\mathrm{outdeg}_{\geq 2}(u,E') &= \exists e_1\, e_2 \begin{pmatrix} (e_1 \neq e_2) \wedge E'(e_1) \wedge E'(e_2) \wedge \\ \overleftarrow{E}(u,e_1) \wedge \overleftarrow{E}(u,e_2) \end{pmatrix} \\
\mathrm{outdeg}_{=1}(u,E') &= \mathrm{outdeg}_{\geq 1}(u,E') \wedge \neg \mathrm{outdeg}_{\geq 2}(u,E') \\
\mathrm{connected}(E') &= \forall U \left(\begin{array}{c} \begin{pmatrix} (\exists u\, (\mathrm{vertex}(u) \wedge U(u))) \wedge \\ (\exists u\, (\mathrm{vertex}(u) \wedge \neg U(u))) \end{pmatrix} \\ \Rightarrow \begin{pmatrix} \exists u\, v\, e\, U(u) \wedge \neg U(v) \wedge \\ E'(e) \wedge \overleftarrow{E}(u,e) \wedge \overrightarrow{E}(e,v) \end{pmatrix} \end{array} \right) \\
\text{Hamilton-cycle}() &= (\exists E'\, (\mathrm{connected}(E')) \wedge (\forall u\, \mathrm{outdeg}_{=1}(u,E')).
\end{aligned}$$

Damit ist der Beweis abgeschlossen. ❑

Übung 4.44. Definieren Sie eine Formel φ in monadischer Logik zweiter Ordnung über der Signatur $\mathscr{S} = \{\overleftarrow{E}, \overrightarrow{E}\}$, sodass ein Graph G genau dann ein perfektes Matching hat, wenn $\lceil G \rceil \models \varphi$ gilt.

Die Möglichkeit, über Kantenmengen zu quantifizieren, ist gleichwertig mit der Betrachtung des Inzidenzgraphen. Der *Inzidenzgraph eines Graphen* $G = (V,E)$ ist der Graph mit der Knotenmenge $V \cup E$ und der Kantenmenge

Inzidenzgraph

$$\begin{aligned} &\{(u,e) \mid u \in V, e \in E, e = (u,v) \text{ für ein } v \in V\} \\ \cup\ &\{(e,v) \mid v \in V, e \in E, e = (u,v) \text{ für ein } u \in V\}. \end{aligned}$$

Um mit der monadischen Logik zweiter Ordnung zählen zu können, wird sie oft um das atomare Mengenprädikat $\mathrm{Card}_{p,q}(X)$ mit $p,q \in \mathbb{N}$, $p \geq 2$ und $0 \leq q < p$ erweitert, wobei X eine Mengenvariable ist. Für eine relationale Struktur $S = (D, \mathscr{R}^S)$ enthält die Lösungsmenge $L^S(\mathrm{Card}_{p,q}(X))$ alle Teilmengen $D' \subseteq D$ mit $|D'| \bmod p = q$. Wir bezeichnen mit CMSO (bzw. C_rMSO) die Klasse aller Formeln in monadischer Logik zweiter Ordnung mit den zusätzlichen atomaren Mengenprädikaten $\mathrm{Card}_{p,q}(X)$ für $p,q \in \mathbb{N}$ und $p \geq 2$ (bzw. für $p,q \in \mathbb{N}$, $p \geq 2$ und $p \leq r$). Die Abkürzung CMSO bezieht sich auf die englische Bezeichnung „*counting monadic second order*“.

CMSO
C_rMSO

Offensichtlich kann das Mengenprädikat $\mathrm{Card}_{p,q}(X)$ für jedes $q > 0$ ausschließlich mit dem Mengenprädikat $\mathrm{Card}_{p,0}(X)$ ausgedrückt werden, wie die folgende Formel zeigt:

$\mathrm{Card}_{p,q}(X)$

$$\mathrm{Card}_{p,q}(X) \Leftrightarrow \exists x_1, \ldots, x_q, Y \left(\begin{array}{l} x_1 \neq x_2 \wedge x_1 \neq x_3 \wedge \cdots \wedge x_1 \neq x_q \wedge \\ x_2 \neq x_3 \wedge x_2 \neq x_4 \wedge \cdots \wedge x_2 \neq x_q \wedge \\ \cdots \\ x_{q-2} \neq x_{q-1} \wedge x_{q-2} \neq x_q \\ x_{q-1} \neq x_q \wedge \\ \neg Y(x_1) \wedge \cdots \wedge \neg Y(x_q) \wedge \mathrm{Card}_{p,0}(Y) \wedge \\ \forall z\ (X(z) \Rightarrow Y(z) \vee z = x_1 \vee \cdots \vee z = x_q) \end{array} \right).$$

Die Formel $\mathrm{Card}_{p,0}(X)$ kann jedoch relativ einfach in der allgemeinen Logik zweiter Ordnung ausgedrückt werden. Die Erweiterung der Logik um das Mengenprädikat $\mathrm{Card}_{p,0}(X)$ ist also nur für die monadische Logik sinnvoll.

$\mathrm{Card}_{p,0}(X)$

Übung 4.45. Definieren Sie die Formel $\mathrm{Card}_{p,0}(X)$ für $p \in \mathbb{N}$ und eine Mengenvariable X in der Logik zweiter Ordnung. Für eine relationale Struktur $S = (D, \mathscr{R}^S)$ enthält die Lösungsmenge $L^S(\mathrm{Card}_{p,0}(X))$ alle Teilmengen $D' \subseteq D$ mit $|D'| \bmod p = 0$.

4.8 Die Komplexität der Logik

Im Folgenden werden wir einige interessante Aspekte der Logik im Zusammenhang mit der Komplexität von Problemen diskutieren. In der klassischen Komplexitätstheorie, die in Kapitel 5 genauer vorgestellt wird, versucht man, den notwendigen

Zeit- und Platzbedarf zur Lösung eines Problems möglichst genau abzuschätzen. Dazu werden Berechnungsmodelle verwendet wie zum Beispiel Turingmaschinen. Diese entscheiden Sprachen $L \subseteq \Sigma^*$ über einem endlichen Alphabet Σ. Die Sprache L enthält die Ja-Instanzen des Problems, also die Instanzen $w \in \Sigma^*$, die eine bestimmte Eigenschaft erfüllen. Probleme für Strukturen, wie zum Beispiel Graphenprobleme, werden für die Analyse in der klassischen Komplexitätstheorie in Sprachen transformiert. Eine relationale Struktur, wie zum Beispiel ein Graph, wird mit einer solchen Transformation auf ein Wort $w \in \Sigma^*$ abgebildet.

In der *darstellenden Komplexitätstheorie* (englisch: *descriptive complexity*) dagegen geht es darum, die notwendigen Mittel zur Beschreibung einer Eigenschaft zu bestimmen. Die Instanzen sind relationale Strukturen, möglicherweise zusammen mit einer logischen Formel. Wir wissen bereits, dass jedes Wort w über einem endlichen Alphabet $\Sigma = \{a_1, \ldots, a_n\}$ auch als eine relationale Struktur $\lfloor w \rfloor$ über der Signatur $\mathscr{S} = \{\mathrm{suc}, \mathrm{lab}_{a_1}, \ldots, \mathrm{lab}_{a_n}\}$ angesehen werden kann. Daher sind die Eingaben in der darstellenden Komplexitätstheorie in gewissem Sinne allgemeiner.

In der klassischen wie auch in der darstellenden Komplexitätstheorie werden bei der Komplexitätsanalyse die Ressourcen immer in Abhängigkeit von der Größe der Eingabe gemessen. Somit müssen wir auch in der darstellenden Komplexitätstheorie zuerst die Größe der Eingabe, also die Größe einer relationalen Struktur, und die Größe einer logischen Formel definieren.

Definition 4.46 (Größe einer relationalen Struktur). *Sei $S = (D, \mathscr{R}^S)$ eine relationale Struktur über der Signatur $\mathscr{S}$. Für ein Element $a \in D$ sei $size(a) = 1$, für eine Relation $A \subseteq D^k$, $k \geq 1$, sei $size(A) = k \cdot |D|$.*

Größe $\mathrm{size}(S)$ einer relationalen Struktur S

Seien $\mathscr{R}_0^S$ die Menge der Konstanten und $\mathscr{R}_+^S$ die Menge der Relationen $R^S \in \mathscr{R}^S$ mit $ar(R) > 0$. Die Größe $\mathrm{size}(S)$ einer relationalen Struktur *S ist definiert als*

$$size(S) = |D| + |\mathscr{R}_0^S| + \sum_{R^S \in \mathscr{R}_+^S} size(R^S).$$

Für eine Lösung $(A_1, \ldots, A_n) \in L^S(\varphi)$ sei $size((A_1, \ldots, A_n)) = \sum_{i=1}^n size(A_i)$.

Größe $\mathrm{size}(\varphi)$ einer Formel φ

Die Größe $\mathrm{size}(\varphi)$ einer Formel *φ ist die Anzahl der Zeichen und Symbole in φ.*

Größe $\mathrm{size}(L^S(\varphi))$ der Lösungsmenge $L^S(\varphi)$

Die Größe $\mathrm{size}(L^S(\varphi))$ der Lösungsmenge *$L^S(\varphi)$ für eine Formel φ mit n freien Variablen ist*

$$size(L^S(\varphi)) = \begin{cases} |L^S(\varphi)| \cdot n, & \text{falls } n > 0, \\ 1 & \text{sonst.} \end{cases}$$

Wir unterscheiden zwischen den folgenden fünf Varianten einer Problemstellung. Sei dabei φ eine gegebene Formel über einer gegebenen Signatur $\mathscr{S}$.

Entscheidungsproblem $\textsc{Dec}_\varphi$

Das *Entscheidungsproblem* ist die Frage, ob die Lösungsmenge eine Lösung enthält oder aber leer ist (siehe dazu auch Abschnitt 5.1).

$\textsc{Dec}_\varphi$	
Gegeben:	Eine relationale Struktur $S = (D, \mathscr{R}^S)$ über $\mathscr{S}$.
Frage:	Gilt $L^S(\varphi(X)) \neq \emptyset$?

Manchmal gehört ein Parameter k zur Eingabe, und man fragt, ob die Lösungsmenge eine Lösung der Größe mindestens oder höchstens k enthält. Ein solches Problem, das wie ein Entscheidungsproblem eine Ja/Nein-Antwort verlangt, nennen wir zur Unterscheidung auch ein *parametrisiertes Problem* und bezeichnen es mit PAR_φ. (In Abschnitt 5.2 werden wir parametrisierte Probleme jedoch in einer etwas anderen Weise darstellen, wobei wir den Parameter besonders hervorheben.)

parametrisiertes Problem PAR_φ

PAR_φ	
Gegeben:	Eine relationale Struktur $S = (D, \mathscr{R}^S)$ über $\mathscr{S}$ und eine Zahl $k \in \mathbb{Z}$.
Frage:	Gibt es eine Lösung $A \in L^S(\varphi(X))$ mit $\text{size}(A) \geq k$ (bzw. $\text{size}(A) \leq k$)?

Beim *Optimierungsproblem* wird nach einer maximalen bzw. minimalen Lösung gesucht.

Optimierungsproblem OPT_φ

OPT_φ	
Gegeben:	Eine relationale Struktur $S = (D, \mathscr{R}^S)$ über $\mathscr{S}$.
Ausgabe:	Eine Lösung $A \in L^S(\varphi(X))$, für die $\text{size}(A)$ maximal (bzw. minimal) ist.

Beim *Suchproblem* wird eine Lösung gefunden bzw. explizit erzeugt.

Suchproblem SEARCH_φ

SEARCH_φ	
Gegeben:	Eine relationale Struktur $S = (D, \mathscr{R}^S)$ über $\mathscr{S}$.
Ausgabe:	Eine Lösung $A \in L^S(\varphi(X))$.

Beim *Konstruktionsproblem* werden sogar alle Lösungen gefunden bzw. explizit erzeugt.

Konstruktionsproblem CON_φ

CON_φ	
Gegeben:	Eine relationale Struktur $S = (D, \mathscr{R}^S)$ über $\mathscr{S}$.
Ausgabe:	Sämtliche Lösungen aus $L^S(\varphi(X))$.

In den Varianten $\text{DEC}(\varphi)$, $\text{PAR}(\varphi)$, $\text{OPT}(\varphi)$, $\text{SEARCH}(\varphi)$ bzw. $\text{CON}(\varphi)$ enthält die Eingabe zusätzlich die Signatur $\mathscr{S}$ und die Formel φ über $\mathscr{S}$.

$\text{DEC}(\varphi)$, $\text{PAR}(\varphi)$, $\text{OPT}(\varphi)$, $\text{CON}(\varphi)$, $\text{SEARCH}(\varphi)$

Parametrisierte Probleme und Optimierungsprobleme werden oft für *gewichtete* Strukturen definiert, bei denen die Elemente der Grundmenge mit einer Bewertungsfunktion der Art $f : D \to \mathbb{Z}$ bewertet sind. Solche Bewertungsfunktionen erlauben es, die Elemente der Lösungsmenge unterschiedlich zu gewichten. Bei Optimierungsproblemen wird dann nach einer gewichteten Lösung gesucht, die maximal oder minimal ist. Das folgende Beispiel soll eine Problembeschreibung erläutern, in der nach einer minimalen gewichteten Lösung gesucht wird.

Bewertungsfunktion

Beispiel 4.47 (Steinerbaum). Seien $G = (V,E)$ ein ungerichteter kantenbewerteter Graph mit $f : E \to \mathbb{Z}$ und $U \subseteq V$ eine Teilmenge der Knoten von G. Die Kosten einer Kantenmenge $E' \subseteq E$ sind definiert durch

$$f(E') \;=\; \sum_{e \in E'} f(e).$$

Steinerbaum
Steinerknoten

Ein *Steinerbaum* ist ein minimaler zusammenhängender Teilgraph $G' = (V',E')$ von G, der alle Knoten aus U enthält, d. h., $U \subseteq V'$. Die Knoten der Menge U werden *Steinerknoten* genannt.

Die Eingabe (G,U) kann wie folgt als eine relationale Struktur betrachtet werden. Dazu erweitern wir die relationale Struktur $\lceil G \rceil$ über der Signatur $\mathscr{S} = \{\overleftarrow{E}, \overrightarrow{E}\}$ zu $\lceil G,U \rceil = (V, R^{\lceil G,U \rceil})$ über der Signatur $\mathscr{S}' = \{\overleftarrow{E}, \overrightarrow{E}, \text{Steiner}\}$ mit $\text{ar}(\text{Steiner}) = 1$ und $\text{Steiner}^{\lceil G,U \rceil} = U$. Das Steinerbaumproblem kann nun als ein Optimierungsproblem mit der Formel Steiner-tree(E') zweiter Ordnung über $\mathscr{S}' = \{\overleftarrow{E}, \overrightarrow{E}, \text{Steiner}\}$ formuliert werden:

$$\begin{aligned} \text{Steiner-tree}(E') \;&=\; \text{u-edge-set}(E') \wedge \\ &\quad\; \exists V' \begin{pmatrix} \text{vertex-set}(V') \wedge \\ \forall u\, (\text{Steiner}(u) \Rightarrow V'(u)) \wedge \\ \text{con-u-subgr}(V',E') \end{pmatrix} \end{aligned}$$

mit

$$\begin{aligned} \text{con-u-subgr}(V',E') \;&=\; \text{vertex-set}(V') \wedge \text{u-edge-set}(E') \wedge \\ &\quad\; \forall U \left(\begin{array}{c} (U \subseteq V' \wedge U \neq \emptyset \wedge U \neq V') \\ \Rightarrow \begin{pmatrix} \exists u\, v\, e\, U(u) \wedge \neg U(v) \wedge V'(v) \wedge \\ E'(e) \wedge \overleftarrow{E}(u,e) \wedge \overrightarrow{E}(e,v) \end{pmatrix} \end{array} \right). \end{aligned}$$

Für einen gegebenen Graphen G mit einer Knotenmenge U und einer Kantenbewertung $f : E \to \mathbb{Z}$ wird nun nach einer Lösung $A \in L^{\lceil G,U \rceil}(\text{Steiner-tree}(E'))$ gesucht, für die $f(A)$ minimal ist.

Σ_t^{FO}, Π_t^{FO}
Σ_t^{SO}, Π_t^{SO}

Definition 4.48 (Σ_t^{FO}, Π_t^{FO}, Σ_t^{SO} und Π_t^{SO}). *Mit Σ_0^{FO} und Π_0^{FO} bezeichnen wir die Klasse aller Formeln erster Ordnung und mit Σ_0^{SO} und Π_0^{SO} bezeichnen wir die Klasse aller Formeln zweiter Ordnung ohne Quantifizierungen.*

Für $t \geq 0$ sei

1. *Σ_{t+1}^{FO} die Klasse aller Formeln erster Ordnung der Form $\exists x_1, \ldots, x_n\ \varphi$ mit $\varphi \in \Pi_t^{FO}$,*
2. *Π_{t+1}^{FO} die Klasse aller Formeln erster Ordnung der Form $\forall x_1, \ldots, x_n\ \varphi$ mit $\varphi \in \Sigma_t^{FO}$,*
3. *Σ_{t+1}^{SO} die Klasse aller Formeln zweiter Ordnung der Form $\exists x_1, \ldots, x_n, X_1, \ldots, X_{n'}\ \varphi$ mit $\varphi \in \Pi_t^{SO}$ und*
4. *Π_{t+1}^{SO} die Klasse aller Formeln zweiter Ordnung der Form $\forall x_1, \ldots, x_n, X_1, \ldots, X_{n'}\ \varphi$ mit $\varphi \in \Sigma_t^{SO}$.*

Die Formel

$$\exists x_1, x_2\ \forall y_1, y_2\ \exists z\ (R(x_1, y_1, z) \vee R(x_2, y_2, z))$$

ist zum Beispiel aus Σ_3^{FO} und die Formel

$$\forall X\ \exists x_1, x_2\ \forall y_1, y_2\ \exists z\ (X(x_1, y_1, z) \vee X(x_2, y_2, z))$$

aus Π_4^{SO}. Die Formeln in Σ_0^{SO} und Π_0^{SO} werden auch Formeln erster Ordnung mit freien Relationsvariablen genannt. Der wesentliche Unterschied zwischen der Logik erster und zweiter Ordnung ist übrigens nicht die Verwendung von Relationsvariablen, sondern das Quantifizieren über Relationsvariablen.

Weite einer Formel erster Ordnung

Ein äußerst wichtiger Parameter für die Bestimmung der Komplexität von $\textsc{Dec}(\varphi)$, $\textsc{Par}(\varphi)$, $\textsc{Opt}(\varphi)$ und $\textsc{Con}(\varphi)$ ist die Weite einer Formel φ. Die *Weite w einer Formel φ erster Ordnung* ist die maximale Anzahl von freien Variablen in allen Teilformeln von φ. Die totale Anzahl aller Variablen ist offensichtlich eine obere Schranke für die Weite w von φ und somit gilt offensichtlich $w \leq \mathrm{size}(\varphi)$. Andererseits kann jede Formel mit einer Weite von w in eine äquivalente Formel umgewandelt werden, in der es höchstens w Variablen gibt. Dazu werden für die Variablen in den Teilformeln von φ immer wieder die gleichen Bezeichner verwendet. Zum Beispiel hat die Formel

$$\varphi(x_1, x_2) \quad = \quad \underbrace{\underbrace{(\exists x_3\ \underbrace{E(x_1, x_3)}_{\text{Weite 2}})}_{\text{Weite 1}} \wedge \underbrace{(\exists x_4\ \underbrace{E(x_4, x_2)}_{\text{Weite 2}})}_{\text{Weite 1}}}_{\text{Weite 2}}$$

eine Weite von 2, verwendet aber vier Variablen, $x_1, \ldots, x_4$. Dagegen hat die äquivalente Formulierung

$$\varphi(x_1, x_2) \quad = \quad (\exists x_2\ E(x_1, x_2)) \wedge (\exists x_1\ E(x_1, x_2))$$

ebenfalls eine Weite von 2, verwendet aber nur zwei Variablen. Solche Optimierungen erschweren jedoch gelegentlich die Lesbarkeit einer Formel.

Satz 4.49. *Für eine gegebene relationale Struktur $S = (D, \mathscr{R}^S)$ über der Signatur $\mathscr{S}$ und eine gegebene Formel φ erster Ordnung über $\mathscr{S}$ kann die Lösungsmenge $L^S(\varphi)$ in der Zeit*

$$\mathscr{O}(size(\varphi) \cdot size(S)^w \cdot w)$$

berechnet werden, wobei w die Weite der Formel φ ist.

Beweis. Die rekursive Definition der Lösungsmenge legt eine rekursive Berechnungsstrategie nahe. In Tabelle 4.4 sind die Laufzeiten für die Berechnung der Lösungsmengen rekursiv aufgebauter Formeln aufgeführt, die wir im Folgenden erläutern möchten.

Für die Fälle 1 bis 3 ist der angegebene Zeitaufwand offensichtlich. Für die übrigen Fälle wird eine beliebige Ordnung (Nummerierung) der Elemente in D festgelegt, damit sie sich effizient mit einer Fachverteilung sortieren lassen. Die lexikographische Sortierung einer Relation $R^S \subseteq D^k$ ist somit in der Zeit $\mathscr{O}(\mathrm{size}(R^S) \cdot k)$ möglich.

Tabelle 4.4. Zeitaufwand zur Berechnung der Lösungsmengen von Formeln erster Ordnung

Fall	Formel φ	Zeitaufwand zur Berechnung von $L^S(\varphi)$
1	$\varphi = (c_1 = c_2)$	$\mathcal{O}(1)$
2	$\varphi = (x = c)$	$\mathcal{O}(1)$
3	$\varphi(x_1,x_2) = (x_1 = x_2)$	$\mathcal{O}(\|D\|)$
4	$\varphi(x_1,\ldots,x_n) = R(x_{i_1},\ldots,x_{i_k})$	$\mathcal{O}(\text{size}(R)\cdot k)$
5	$\varphi(x_1,\ldots,x_n) = \neg\varphi'(x_{i_1},\ldots,x_{i_k})$	$\mathcal{O}(\|D\|^n + \text{size}(L^S(\varphi'))\cdot k)$
6	$\varphi(x_1,\ldots,x_n) = \begin{pmatrix}\varphi_1(x_{i_1},\ldots,x_{i_k})\wedge\\ \varphi_2(x_{j_1},\ldots,x_{j_\ell})\end{pmatrix}$	$\mathcal{O}(\text{size}(L^S(\varphi_1))\cdot k + \text{size}(L^S(\varphi_2))\cdot l)$
7	$\varphi(x_1,\ldots,x_n) = \begin{pmatrix}\varphi_1(x_{i_1},\ldots,x_{i_k})\vee\\ \varphi_2(x_{j_1},\ldots,x_{j_\ell})\end{pmatrix}$	$\mathcal{O}(\text{size}(L^S(\varphi_1))\cdot k + \text{size}(L^S(\varphi_2))\cdot l)$
8	$\varphi(x_1,\ldots,x_n) = \exists x_{n+1}\ \varphi'(x_{i_1},\ldots,x_{i_k})$	$\mathcal{O}(\text{size}(L^S(\varphi'))\cdot k)$
9	$\varphi(x_1,\ldots,x_n) = \forall x_{n+1}\ \varphi'(x_{i_1},\ldots,x_{i_k})$	$\mathcal{O}(\text{size}(L^S(\varphi'))\cdot k)$

Für Fall 4 werden die k-Tupel $(a_1,\ldots,a_k)$ aus R^S in lexikographischer Reihenfolge inspiziert. Sei $h : [k] \to [n]$ die Zuordnung der Positionen in R zu den freien Variablen $x_1,\ldots,x_n$. Für das Beispiel $\varphi(x_1,x_2,x_3) = R(x_1,x_2,x_1,x_1,x_3,x_2)$ wäre $h(1) = 1, h(2) = 2, h(3) = 1, h(4) = 1, h(5) = 3, h(6) = 2$. Ein k-Tupel $(a_1,\ldots,a_k)$ aus R^S wird in die Lösungsmenge aufgenommen, wenn für jede freie Variable x_i alle Einträge a_j mit $h(j) = i$ gleich sind und $(a_1,\ldots,a_k)$ noch nicht aufgenommen wurde. Der Auswahlprozess kann nach der lexikographischen Sortierung in der Zeit $\mathcal{O}(\text{size}(R^S))$ durchgeführt werden.

Für Fall 5 wird analog wie im Fall 4 zuerst die Menge

$$\{(a_1,\ldots,a_n) \in D^n \mid (a_{i_1},\ldots,a_{i_k}) \in \varphi'(x_{i_1},\ldots,x_{i_k})\}$$

berechnet und mit einer Fachverteilung lexikographisch aufsteigend sortiert. Anschließend werden alle n-Tupel aus D^n in lexikographisch aufsteigender Reihenfolge inspiziert und in die Lösungsmenge von $\neg\varphi(x_1,\ldots,x_n)$ aufgenommen, wenn sie nicht in der obigen Menge enthalten sind.

Für die Fälle 6 und 7 werden die Lösungsmengen

$$\{(a_1,\ldots,a_n) \in D^n \mid (a_{i_1},\ldots,a_{i_k}) \in \varphi_1(x_{i_1},\ldots,x_{i_k})\}$$

und

$$\{(a_1,\ldots,a_n) \in D^n \mid (a_{j_1},\ldots,a_{j_\ell}) \in \varphi_2(x_{j_1},\ldots,x_{j_\ell})\}$$

aufgebaut, lexikographisch sortiert und zur Ergebnismenge gemischt bzw. disjunkt vereinigt.

Zur Berechnung der Lösungsmenge einer Quantifizierung in den Fällen 8 und 9 wird ebenfalls zuerst die lexikographisch aufsteigend sortierte Menge

$$\{(a_1,\ldots,a_{n+1}) \in D^{n+1} \mid (a_{i_1},\ldots,a_{i_k}) \in \varphi'(x_{i_1},\ldots,x_{i_k})\}$$

aufgebaut. Im Fall 8 werden diese $(n+1)$-Tupel lediglich auf die ersten n Positionen reduziert. Im Fall 9 wird ein n-Tupel genau dann in die Lösungsmenge aufgenommen, wenn alle $|D|^n$ unter den $(n+1)$-Tupeln mit gleichem x_{n+1} existieren.

Die Berechnung der Lösungsmenge $L^S(\varphi)$ aus den unmittelbaren Teilformeln ist also immer in der Zeit $\mathcal{O}(\text{size}(S)^w \cdot w)$ möglich. Da die Formel höchstens $\text{size}(\varphi)$ Teilformeln enthalten kann, folgt daraus die Aussage des Satzes. ❑

Satz 4.49 impliziert die beiden folgenden Behauptungen.

Korollar 4.50. *Für eine gegebene relationale Struktur $S = (D, \mathscr{R}^S)$ über der Signatur $\mathscr{S}$ und eine gegebene Formel φ erster Ordnung über $\mathscr{S}$ mit einer konstanten Anzahl von Variablen kann die Lösungsmenge $L^S(\varphi)$ in polynomieller Zeit berechnet werden.*

Korollar 4.51. *Für eine Formel φ erster Ordnung sind die Probleme* $\textsc{Dec}_\varphi$, $\textsc{Par}_\varphi$, $\textsc{Opt}_\varphi$ *und* $\textsc{Con}_\varphi$ *in polynomieller Zeit entscheidbar bzw. berechenbar.*

Übung 4.52. In welcher Zeit ist das Problem $\textsc{Search}_\varphi$ berechenbar?

Weitere Resultate zur Komplexität der Logik (u. a. bezüglich der oben definierten Variante $\textsc{Dec}(\varphi)$ von $\textsc{Dec}_\varphi$) werden in Abschnitt 5.1.4 angegeben (siehe die Sätze 5.39 und 5.40). Diese Resultate erfordern jedoch einige grundlegende Begriffe der Komplexitätstheorie, der wir uns im nächsten Kapitel zuwenden.

4.9 Literaturhinweise

Die mathematische Logik ist ein Teilgebiet der Mathematik und spielt in der Informatik eine bedeutende Rolle. Oft wird sie in die Teilgebiete Modelltheorie, Beweistheorie, Mengenlehre und Rekursionstheorie aufgeteilt. Es gibt eine Vielzahl von Abhandlungen über Themengebiete der mathematischen Logik. Die drei folgenden Bücher verdienen jedoch für die in diesem Buch betrachteten Fragestellungen eine besondere Aufmerksamkeit. Aspekte der Logik im Zusammenhang mit der parametrisierten Komplexität sind im Lehrbuch von Flum und Grohe [FG06] dargestellt. Die monadische Logik und ihre Bedeutung für die effiziente Lösbarkeit von Graphenproblemen für Graphen mit beschränkter Baumweite (Abschnitt 10.6) und Graphen mit beschränkter Cliquenweite (Abschnitt 11.6) werden von Courcelle [Cou] ausführlich beschrieben. Eine der klassischen Einführungen in die darstellende Komplexitätstheorie ist das Lehrbuch von Immerman [Imm98].