Kapitel 1

Aussagenlogik

Kontext und Ziele

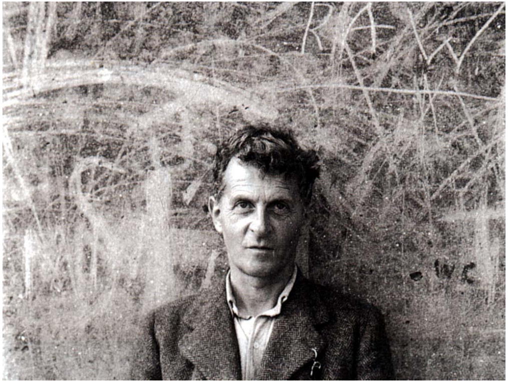
In diesem Kapitel führen wir die Aussagenlogik ein. Wir definieren formal ihre Syntax und Semantik, und wir leiten grundlegende Eigenschaften her. Schließlich diskutieren wir Algorithmen für das aussagenlogische Erfüllbarkeitsproblem.

Aussagen

Die Frage „Was ist eigentlich ein Wort?“ ist analog der „Was ist eine Schachfigur?“

Ludwig Wittgenstein, *Philosophische Untersuchungen*

- Aussagen (im Sinne der Aussagenlogik) sind sprachliche Gebilde, die entweder wahr oder falsch sind.
- Aussagen können mit Junktoren wie nicht, und, oder oder wenn ... dann zu komplexeren Aussagen verknüpft werden.
- Aussagenlogik beschäftigt sich mit allgemeinen Prinzipien des korrekten Argumentierens und Schließens mit Aussagen und Kombinationen von Aussagen.



Ludwig Wittgenstein (1889 – 1951)

Quelle: `http://en.wikipedia.org/wiki/Ludwig_Wittgenstein`

Syntax und Semantik

Logiken sind, wie auch Programmiersprachen, formale Sprachen, die durch eine Syntax und eine Semantik definiert sind.

Syntax

Die syntaktischen Objekte, oder Ausdrücke, einer Sprache sind Wörter über einem (endlichen oder unendlichen) Alphabet.

Die Syntax der Sprache legt fest, welche Wörter „korrekte“ Ausdrücke sind.

In einer Logik bezeichnen wir diese Ausdrücke in der Regel als Formeln, später kommen noch Terme hinzu. In einer Programmiersprache bezeichnen wir die Ausdrücke als (syntaktisch korrekte) Programme.

Semantik

Die Semantik einer Sprache legt die Bedeutung der Ausdrücke fest.

Im einfachsten Fall sind Formeln eine Logik wahr oder falsch; Programme einer Programmiersprache haben ein Ein- und Ausgabeverhalten.

Meistens sind die Semantiken aber komplizierter.

1.1 Syntax der Aussagenlogik

Symbolmenge und Alphabet der Aussagenlogik

Eine aussagenlogische Symbolmenge ist eine Menge σ, deren Elemente wir als Aussagensymbole bezeichnen.

Das Alphabet $\Sigma_{\mathsf{AL}(\sigma)}$ der Aussagenlogik über σ besteht aus folgenden Symbolen:

- den Aussagensymbolen in σ;
- den booleschen Konstanten $\bot, \top$;
- den Junktoren $\neg, \wedge, \vee, \rightarrow$;
- den Klammersymbolen $($, $)$.

Kurz:

$$\Sigma_{\mathsf{AL}(\sigma)} := \sigma \cup \big\{\bot, \top, \neg, \vee, \wedge, \rightarrow, (,)\big\}.$$

Vereinbarung

In dieser Vorlesung nehmen wir immer an, dass Symbolmengen abzählbar sind, also endlich oder abzählbar unendlich.

Praktisch alle Ergebnisse lassen sich auch auf überabzählbare Symbolmengen übertragen, das erfordert aber zum Teil mehr Aufwand und mengentheoretische Kenntnisse.

Zur Erinnerung

Eine Menge M heißt abzählbar unendlich, wenn sie unendlich ist und ihre Elemente sich in der Form $m_0, m_1, m_2, \dots$ aufzählen lassen. Formal heißt M genau dann abzählbar unendlich, wenn es eine bijektive Abbildung von der Menge $\mathbb{N} = \{0, 1, 2, \dots\}$ der natürlichen Zahlen auf die Menge M gibt.

Eine Menge M heißt abzählbar, wenn sie entweder endlich oder abzählbar unendlich ist. Eine Menge M heißt überabzählbar, wenn sie nicht abzählbar ist.

Beispiele

- Die Menge $\mathbb{N}$ ist abzählbar unendlich.
- Die Menge $\mathbb{R}$ aller reellen Zahlen ist überabzählbar.

- Ist Σ ein abzählbares Alphabet, so ist die Menge Σ^* aller endlichen Wörter über Σ abzählbar. Ist etwa $\Sigma = \{a_0, a_1, a_2, \dots\}$, so können wir eine Aufzählung von Σ^* wie folgt beginnen:

$$\varepsilon \quad \text{(das leere Wort)}$$

$$a_0,$$

$$a_1, a_0a_0, a_0a_1, a_1a_0, a_1a_1,$$

$$a_2, a_0a_2, a_2a_0, a_1a_2, a_2a_1, a_2a_2, a_0a_0a_0, a_0a_0a_1, a_0a_0a_2, \dots, a_2a_2a_2$$

$$a_3, a_0a_3, \dots, a_3a_3a_3a_3,$$

$$\dots$$

- Ist M eine unendliche Menge, so ist die Potenzmenge $2^M := \{N \mid N \subseteq M\}$ von M überabzählbar.

Definition 1.1

Die Menge $\mathrm{AL}(\sigma)$ der aussagenlogischen Formeln über der Symbolmenge σ ist die wie folgt rekursiv definierte Menge von Wörtern über dem Alphabet $\Sigma_{\mathrm{AL}(\sigma)}$.

(i) $P \in \mathrm{AL}(\sigma)$ für alle $P \in \sigma$.

(ii) $\bot, \top \in \mathrm{AL}(\sigma)$.

(iii) Wenn $\varphi, \psi \in \mathrm{AL}(\sigma)$, dann $\neg\varphi, (\varphi \wedge \psi), (\varphi \vee \psi), (\varphi \to \psi) \in \mathrm{AL}(\sigma)$.

Beispiel 1.2

Seien $P, Q, R \in \sigma$.

$$\begin{aligned} (\neg P \vee (P \to Q)) &\quad \in \mathrm{AL}(\sigma), \\ \neg\neg((P \wedge \bot) \to \neg R) &\quad \in \mathrm{AL}(\sigma), \\ (P \vee Q \wedge R) &\quad \notin \mathrm{AL}(\sigma), \\ (\neg R) &\quad \notin \mathrm{AL}(\sigma). \end{aligned}$$

Ergänzungen zu Seite 1.8

Wir nennen die Zeichen $\neg, \wedge, \vee, \rightarrow$ nicht, und, oder, impliziert. Das ist ein Vorgriff auf ihre (intuitive) semantische Bedeutung. Im Moment sind es einfach irgendwelche Zeichen, die wir auch „Häkchen“, „Hut“, „umgedrehten Hut“, und „Pfeil“ nennen könnten.

Bemerkungen

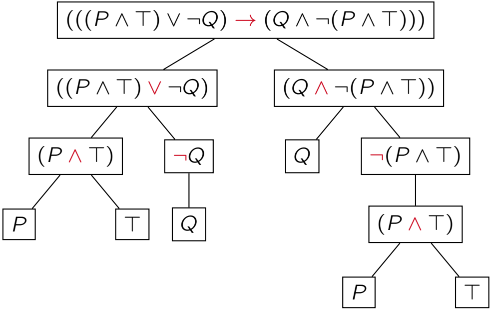
- Die rekursive Definition der Menge $\mathrm{AL}(\sigma)$ ist so zu verstehen, dass sich alle Elemente der Menge durch endlichmaliges Anwenden der Basisregeln (i), (ii) und der rekursiven Regeln (iii) erzeugen lassen.
- Wir können dann auch Beweise per Induktion über den Aufbau von $\mathrm{AL}(\sigma)$ führen.
- Hintergrund zu rekursiven Definitionen und Induktionsbeweisen findet sich in Anhang B (der das entsprechende Kapitel aus der Vorlesung Formale Systeme, Automaten, und Prozesse enthält).

Die Struktur einer Formel lässt sich bequem in einem Syntaxbaum (englisch: parse tree) darstellen.

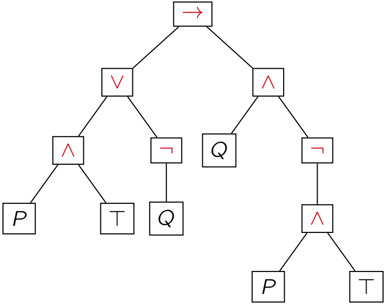
Beispiel 1.3

Syntaxbaum der Formel $(((P \wedge \top) \vee \neg Q) \to (Q \wedge \neg(P \wedge \top)))$:

Ausführlich:



Kurzform:



Eindeutige Lesbarkeit

Lemma 1.4

Jede aussagenlogische Formel hat genau einen Syntaxbaum.

Lemma 1.5

Keine aussagenlogische Formel ist ein Präfix einer anderen aussagenlogischen Formel.

Ergänzungen zu Seite 1.11

Zur Erinnerung

Ein Wort $v \in \Sigma^*$ ist Präfix eines Wortes $w \in \Sigma^*$, wenn es ein Wort $u \in \Sigma^*$ gibt, so dass $w = vu$.

Beide Lemmata lassen sich leicht per Induktion über den Aufbau der aussagenlogischen Formeln beweisen. Wir verzichten hier auf die Beweise.

Subformeln

Sei $\varphi \in \mathrm{AL}(\sigma)$.

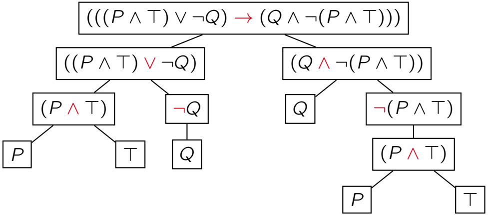
- Die Formeln ψ, die im ausführlichen Syntaxbaum einer Formel φ als Knotenbeschriftung vorkommen, nennen wir Subformeln von φ.
- Eine Subformel ψ von φ kann an mehreren Knoten des Syntaxbaums vorkommen. Wir sprechen dann von verschiedenen Vorkommen von ψ in φ.
- Die Menge aller Subformeln von φ bezeichnen wir mit $\mathrm{sub}(\varphi)$ und die Menge aller Symbole aus σ, die in φ vorkommen, mit $\mathrm{symb}(\varphi)$.

Beispiel 1.6

Für $\varphi := (((P \wedge \top) \vee \neg Q) \to (Q \wedge \neg(P \wedge \top)))$
ist

$$\mathrm{symb}(\varphi) = \{P, Q\},$$

$$\begin{aligned}\mathrm{sub}(\varphi) = \Big\{&(((P \wedge \top) \vee \neg Q) \to (Q \wedge \neg(P \wedge \top))),\\ &((P \wedge \top) \vee \neg Q), (Q \wedge \neg(P \wedge \top)),\\ &(P \wedge \top), \neg Q, Q, \neg(P \wedge \top), P, \top\Big\}.\end{aligned}$$



Konventionen zur Notation

- Aussagenlogische Formeln bezeichnen wir mit griechischen Kleinbuchstaben: φ, ψ, χ und auch Varianten wie φ', ψ_1.
- Aussagensymbole bezeichnen wir mit lateinischen Großbuchstaben P, Q, R und Varianten.
- Es ist bequem, für den Rest des Kapitels eine abzählbar unendliche Symbolmenge σ festzuhalten, zum Beispiel

  $$\sigma = \{P_0, P_1, P_2, \dots\}.$$

  Wir schreiben im Folgenden einfach AL statt AL(σ).

  Nur in Beispielen verwenden wir gelegentlich andere Symbolmengen.

Zur Verbesserung der Lesbarkeit vereinbaren wir noch einige vereinfachende Schreibweisen.

- Wir lassen überflüssige Klammern weg, insbesondere die äußeren Klammern einer Formel.
- Wir vereinbaren folgende Bindungsregeln:
  $\neg$ bindet stärker als $\vee, \wedge$, die wiederum stärker als $\to$ binden.
- Bei iterierten Disjunktionen oder Konjunktionen vereinbaren wir Linksklammerung.
- Außerdem schreiben wir statt $\varphi_1 \wedge \dots \wedge \varphi_k$ auch $\bigwedge_{i=1}^{k} \varphi_i$ und entsprechend für Disjunktionen $\bigvee$.

Lernen Sie die griechischen Buchstaben!
Siehe z.B. https://de.wikipedia.org/wiki/Griechisches_Alphabet.

Beispiele

(1) Zu den Bindungsregeln:
Wir schreiben $\varphi \vee \psi \to \chi \wedge \neg\psi$ statt $((\varphi \vee \psi) \to (\chi \wedge \neg\psi))$.

(2) Zur Linksklammerung:
Wir schreiben $\varphi_1 \wedge \varphi_2 \wedge \varphi_3$ statt $((\varphi_1 \wedge \varphi_2) \wedge \varphi_3)$.

(3) Man beachte aber, dass $\wedge$ und $\vee$ gleichstark binden, deswegen ist etwa $\varphi_1 \vee \varphi_2 \wedge \varphi_3$ keine erlaubte Schreibweise.

Computerlesbare Darstellung von Formeln

Die Übungen zu dieser Vorlesung enthalten auch Programmieraufgaben. Dazu verwenden wie eine Kodierung unser abstrakten Syntax über dem ASCII-Alphabet Σ_{ASCII}. Wir verwenden für Symbole aus Σ_{ASCII} einen Schreibmaschinenfont ($\texttt{A,B,...,a,b,...,0,1,...,(,),...}$).

- Aussagensymbole sind alle Wörter über Σ_{ASCII}, die aus Groß- und Kleinbuchstaben, Ziffern, sowie Unter- und Bindestrich bestehen und die mit einem Großbuchstaben oder einem Unterstrich beginnen.
- Für die booleschen Konstanten verwenden wir `0` statt $\bot$ und `1` statt $\top$.
- Für die Junktoren verwenden wir `~` statt $\neg$, `/\` statt $\wedge$, `\/` statt $\vee$, und `->` statt $\to$.

Beispiel 1.7

Die Formel $(\neg(P \wedge \neg Q) \to \top))$ sieht in dieser Syntax folgendermaßen aus:

```
( ~ ( P /\ ~ Q ) -> 1 )
```

Leerzeichen setzen wir beliebig, um die Lesbarkeit zu erhöhen.

1.2 Semantik der Aussagenlogik

Vorüberlegung zur Semantik

- Eine aussagenlogische Formel wird erst zur Aussage, wenn wir alle in ihr vorkommenden Aussagensymbole durch Aussagen ersetzen.
- Wir interessieren uns hier nicht so sehr für die tatsächlichen Aussagen, sondern nur für ihren Wahrheitswert, also dafür, ob sie wahr oder falsch sind.
- Um das festzustellen, reicht es, den Aussagensymbolen die Wahrheitswerte der durch sie repräsentierten Aussagen zuzuordnen.
- Die Bedeutung einer Formel besteht also aus ihren Wahrheitswerten unter allen möglichen Wahrheitswerten für die in der Formel vorkommenden Aussagensymbole.

Aussagenlogische Interpretationen

Wir erinnern uns, dass wir der Einfachheit halber eine feste Symbolmenge σ betrachten. Wir beziehen uns im Folgenden immer auf diese feste Symbolmenge.

Als Wahrheitswerte verwenden wir 0 (für „falsch") und 1 (für „wahr").

Definition 1.8

Eine aussagenlogische Interpretation ist eine Abbildung $\mathfrak{A} : \sigma \to \{0, 1\}$, die jedem Aussagensymbol einen Wahrheitswert zuweist.

Definition 1.9

Wir definieren für jede Formel $\varphi \in \mathrm{AL}$ und jede Interpretation $\mathfrak{A}$ einen Wahrheitswert $[\![\varphi]\!]^{\mathfrak{A}} \in \{0, 1\}$ rekursiv wie folgt:

(i) $[\![P]\!]^{\mathfrak{A}} := \mathfrak{A}(P)$ für alle $P \in \sigma$;

(ii) $[\![\bot]\!]^{\mathfrak{A}} := 0$ und $[\![\top]\!]^{\mathfrak{A}} := 1$;

(iii)

$$[\![\neg\varphi]\!]^{\mathfrak{A}} := \begin{cases} 1 & \text{if } [\![\varphi]\!]^{\mathfrak{A}} = 0, \\ 0 & \text{if } [\![\varphi]\!]^{\mathfrak{A}} = 1, \end{cases} \qquad [\![\varphi \wedge \psi]\!]^{\mathfrak{A}} := \begin{cases} 1 & \text{if } [\![\varphi]\!]^{\mathfrak{A}} = 1 \text{ und } [\![\psi]\!]^{\mathfrak{A}} = 1, \\ 0 & \text{if } [\![\varphi]\!]^{\mathfrak{A}} = 0 \text{ oder } [\![\psi]\!]^{\mathfrak{A}} = 0, \end{cases}$$

$$[\![\varphi \vee \psi]\!]^{\mathfrak{A}} := \begin{cases} 1 & \text{if } [\![\varphi]\!]^{\mathfrak{A}} = 1 \text{ oder } [\![\psi]\!]^{\mathfrak{A}} = 1, \\ 0 & \text{if } [\![\varphi]\!]^{\mathfrak{A}} = 0 \text{ und } [\![\psi]\!]^{\mathfrak{A}} = 0, \end{cases} \qquad [\![\varphi \to \psi]\!]^{\mathfrak{A}} := \begin{cases} 1 & \text{if } [\![\varphi]\!]^{\mathfrak{A}} = 0 \text{ oder } [\![\psi]\!]^{\mathfrak{A}} = 1, \\ 0 & \text{if } [\![\varphi]\!]^{\mathfrak{A}} = 1 \text{ und } [\![\psi]\!]^{\mathfrak{A}} = 0, \end{cases}$$

für alle $\varphi, \psi \in \mathrm{AL}$.

Intuitive Bedeutung der Semantik

Boolesche Konstanten: $\top$ und $\bot$ bedeuten einfach „wahr“ und „falsch“.

Aussagensymbole: Die Aussagensymbole stehen für irgendwelche Aussagen, von denen uns aber nur der Wahrheitswert interessiert. Dieser wird durch die Interpretation festgelegt.

Negation: $\neg\varphi$ bedeutet „nicht φ“.

Konjunktion: $(\varphi \wedge \psi)$ bedeutet „φ und ψ“.

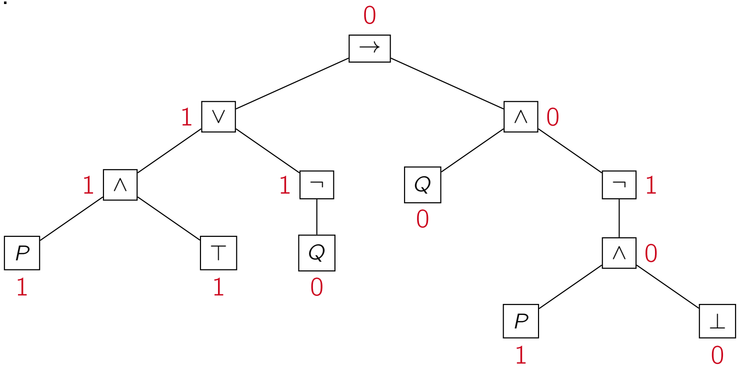
Disjunktion: $(\varphi \vee \psi)$ bedeutet „φ oder ψ“.

Implikation: $(\varphi \to \psi)$ bedeutet „φ impliziert ψ“ (oder „wenn φ dann ψ“).

Ein einfacher Weg, den Wahrheitswert einer Formel unter einer Interpretation auszurechnen, besteht darin, die Werte aller Subformeln induktiv in einem Syntaxbaum auszurechnen.

Beispiel 1.10

Sei $\varphi := (((P \wedge \top) \vee \neg Q) \to (Q \wedge \neg(P \wedge \bot)))$, und sei $\mathfrak{A}$ eine Interpretation mit $\mathfrak{A}(P) = 1$ und $\mathfrak{A}(Q) = 0$.



Wir können die Wahrheitswerte der Junktoren auch knapper beschreiben, wenn wir uns zu Nutze machen, dass unsere Wahrheitswerte auch natürliche Zahlen sind.

$$\begin{aligned}
[\![\neg\varphi]\!]^{\mathfrak{A}} &= 1 - [\![\varphi]\!]^{\mathfrak{A}}, \\
[\![\varphi \wedge \psi]\!]^{\mathfrak{A}} &= \min\left\{ [\![\varphi]\!]^{\mathfrak{A}}, [\![\psi]\!]^{\mathfrak{A}} \right\}, \\
[\![\varphi \vee \psi]\!]^{\mathfrak{A}} &= \max\left\{ [\![\varphi]\!]^{\mathfrak{A}}, [\![\psi]\!]^{\mathfrak{A}} \right\}, \\
[\![\varphi \to \psi]\!]^{\mathfrak{A}} &= \max\left\{ 1 - [\![\varphi]\!]^{\mathfrak{A}}, [\![\psi]\!]^{\mathfrak{A}} \right\}.
\end{aligned}$$

Damit lassen sich Wahrheitswerte von Formeln dann auch bequem mit einem „top-down“ Verfahren ausrechnen (im Gegensatz zum „bottom-up“ Verfahren mit dem Syntaxbaum).

Beispiel.

Sei $\varphi := \neg(P \vee Q) \to \top$, und sei $\mathfrak{A}$ eine Interpretation mit $\mathfrak{A}(P) = 1$ und $\mathfrak{A}(Q) = 0$.

Dann gilt

$$[\![\varphi]\!]^{\mathfrak{A}} = \max\left\{ 1 - [\![\neg(P \vee Q)]\!]^{\mathfrak{A}}, [\![\top]\!]^{\mathfrak{A}} \right\} = \max\left\{ 1 - [\![\neg(P \vee Q)]\!]^{\mathfrak{A}}, 1 \right\} = 1.$$

Definition 1.11

(1) Eine Interpretation $\mathfrak{A}$ erfüllt eine Formel $\varphi \in \mathrm{AL}$, wenn $[\![\varphi]\!]^{\mathfrak{A}} = 1$.

(2) Eine Interpretation $\mathfrak{A}$ erfüllt eine Formelmenge $\Phi \subseteq \mathrm{AL}$, wenn $[\![\varphi]\!]^{\mathfrak{A}} = 1$ für alle $\varphi \in \Phi$.

Statt $\mathfrak{A}$ erfüllt φ (bzw. Φ) sagen wir auch, $\mathfrak{A}$ ist ein Modell von φ (bzw. Φ) und schreiben

$$\mathfrak{A} \models \varphi \qquad \text{bzw.} \qquad \mathfrak{A} \models \Phi.$$

Erfüllbarkeit und Allgemeingültigkeit

Definition 1.12

(1) Eine Formel $\varphi \in \mathsf{AL}$ (bzw. eine Formelmenge $\Phi \subseteq \mathsf{AL}$) ist erfüllbar, wenn es eine Interpretation $\mathfrak{A}$ gibt, die φ (bzw. Φ) erfüllt.

(2) Eine Formel $\varphi \in \mathsf{AL}$ (bzw. eine Formelmenge $\Phi \subseteq \mathsf{AL}$) ist allgemeingültig, wenn alle Interpretationen $\mathfrak{A}$ die Formel φ (bzw. die Formelmenge Φ) erfüllen.

Eine allgemeingültige Formel bezeichnet man auch als Tautologie.

Beobachtung 1.13

Für alle $\varphi \in \mathsf{AL}$ gilt:

$$\varphi \textit{ ist unerfüllbar} \iff \neg\varphi \textit{ ist allgemeingültig.}$$

Ergänzungen zu Seite 1.22

Beispiel.

Die Formel $\varphi := \neg(P \vee Q) \to \top$ is allgemeingültig. Das liegt daran, dass eine Implikation immer wahr ist, wenn die rechte Seite wahr ist.

Das Koinzidenzlemma der Aussagenlogik

Lemma 1.14

Der Wert $[\![\varphi]\!]^{\mathfrak{A}}$ *einer Formel* φ *hängt nur von den Werten der Aussagensymbole, die in* φ *vorkommen, ab.*

Das heißt, für alle $\varphi \in \mathrm{AL}$ *und alle* $\mathfrak{A}, \mathfrak{B} : \sigma \to \{0, 1\}$, *wenn* $\mathfrak{A}(P) = \mathfrak{B}(P)$ *für alle* $P \in \mathrm{symb}(\varphi)$, *dann*

$$[\![\varphi]\!]^{\mathfrak{A}} = [\![\varphi]\!]^{\mathfrak{B}} .$$

Ergänzungen zu Seite 1.23

Beweis.
Das Lemma gilt „offensichtlich“, denn bei der Berechnung von $[\![\varphi]\!]^{\mathfrak{A}}$ werden ja nur die Werte $\mathfrak{A}(P)$ für $P \in \text{symb}(\varphi)$ verwendet.

Man kann das Lemma auch per Induktion über den Aufbau von φ beweisen. $\square$

Anmerkungen zum Koinzidenzlemma

Bemerkungen

- Um eine Formel φ unter einer Interpretation $\mathfrak{A}$ auszuwerten, genügt es also, die Restriktion von $\mathfrak{A}$ auf $\operatorname{symb}(\varphi)$ zu kennen, also die Abbildung
$$\mathfrak{A}{\restriction}_{\operatorname{symb}(\varphi)} : \operatorname{symb}(\varphi) \to \{0, 1\}$$
mit $\mathfrak{A}{\restriction}_{\operatorname{symb}(\varphi)}(P) = \mathfrak{A}(P)$ für alle $P \in \operatorname{symb}(\varphi)$.
- Weil $\operatorname{symb}(\varphi)$ endlich ist, gibt es auch nur endliche viele Abbildungen $\operatorname{symb}(\varphi) \to \{0, 1\}$.
Um die Semantik von φ vollständig zu beschreiben müssen wir also nur endlich viele (eingeschränkte) Interpretationen zu betrachten.

Notation

Für eine Formel $\varphi \in \mathrm{AL}$ und Aussagensymbole $P_1, \ldots, P_n \in \sigma$ schreiben wir oft $\varphi(P_1, \ldots, P_n)$, um auszudrücken, dass $\operatorname{symb}(P) \subseteq \{P_1, \ldots, P_n\}$.

Wahrheitstafeln

Wir können alle (eingeschränkten) Interpretationen für eine Formel $\varphi(P_1, \dots, P_n)$ in einer sogenannten Wahrheitstafel für φ auflisten, die eine Spalte für jedes Symbol P_i und eine Zeile für jede Kombination von Werten hat. In einer weiteren Spalte können wir dann den Wert von φ unter jeder dieser Interpretationen aufführen.

Beispiel 1.15

Betrachte die Formel
$\varphi(P_1, P_2, P_3) := (P_1 \vee \neg P_2) \to P_3$.

φ is erfüllbar, weil in der letzten Spalte der Wahrheitstafel eine 1 vorkommt, aber nicht allgemeingültig, weil auch eine 0 vorkommt.

P_1	P_2	P_3	φ
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1

1.3 Aussagenlogische Modellierung

Beispiel: Sudoku

	3							
			1	9	5			
	9	8					6	
8				6				
4					3			1
				2				
	6					2	8	
			4	1	9			5
							7	

5	3	4	6	7	8	9	1	2
6	7	2	1	9	5	3	4	8
1	9	8	3	4	2	5	6	7
8	5	9	7	6	1	4	2	3
4	2	6	8	5	3	7	9	1
7	1	3	9	2	4	8	5	6
9	6	1	5	3	7	2	8	4
2	8	7	4	1	9	6	3	5
3	4	5	2	8	6	1	7	9

Ziel

Zu jedem Sudoku Rätsel S konstruieren eine Formelmenge Φ_S, deren Modelle (erfüllende Interpretationen) gerade den Lösungen von S entsprechenden.

Sudoku (Forts.)

$(1,1)$	$(1,2)$	$(1,3)$	$(1,4)$	$(1,5)$	$(1,6)$	$(1,7)$	$(1,8)$	$(1,9)$
$(2,1)$	$(2,2)$	$(2,3)$	$(2,4)$	$(2,5)$	$(2,6)$	$(2,7)$	$(2,8)$	$(2,9)$
$(3,1)$	$(3,2)$	$(3,3)$	$(3,4)$	$(3,5)$	$(3,6)$	$(3,7)$	$(3,8)$	$(3,9)$
$(4,1)$	$(4,2)$	$(4,3)$	$(4,4)$	$(4,5)$	$(4,6)$	$(4,7)$	$(4,8)$	$(4,9)$
$(5,1)$	$(5,2)$	$(5,3)$	$(5,4)$	$(5,5)$	$(5,6)$	$(5,7)$	$(5,8)$	$(5,9)$
$(6,1)$	$(6,2)$	$(6,3)$	$(6,4)$	$(6,5)$	$(6,6)$	$(6,7)$	$(6,8)$	$(6,9)$
$(7,1)$	$(7,2)$	$(7,3)$	$(7,4)$	$(7,5)$	$(7,6)$	$(7,7)$	$(7,8)$	$(7,9)$
$(8,1)$	$(8,2)$	$(8,3)$	$(8,4)$	$(8,5)$	$(8,6)$	$(8,7)$	$(8,8)$	$(8,9)$
$(9,1)$	$(9,2)$	$(9,3)$	$(9,4)$	$(9,5)$	$(9,6)$	$(9,7)$	$(9,8)$	$(9,9)$

Koordinaten der Felder:

(i,j) bezeichnet das Feld in Zeile i und Spalte j.

Anfangsbeschriftung:

Ein spezifisches Sudoku S ist gegeben durch seine Anfangsbeschriftung. Formal können wir S auffassen als Abbildung

Notation: $[n] := \{1, \dots, n\}$

$$S : [9][9] \times [9] \to [9] \cup \{\square\},$$

wobei $S(i,j) = \square$ bedeutet, dass das Feld (i,j) am Anfang noch leer ist.

Lösungen

Ein Lösung für eine Sudoku S ist eine Abbildung $L : [9] \times [9] \to [9]$, so dass:

- jede Ziffer kommt in jeder Zeile, jeder Spalte, und jedem Block vor
- für alle (i,j) mit $S(i,j) \neq \square$ gilt $L(i,j) = S(i,j)$.

Die Menge aller Lösungen für S bezeichnen wir mit $\mathscr{L}(S)$.

Sudoku (Forts.)

Aussagensymbole:

Aussagensymbol $P_{i,j,k}$, für $i, j, k \in \{1, \dots, 9\}$, steht für die Aussage

„Das Feld (i, j) enthält die Zahl k."

Beschriftungen:

"Auf jedem Feld steht mindestens eine Zahl":

$$\varphi_{B1} := \bigwedge_{i,j=1}^{9} \bigvee_{k=1}^{9} P_{i,j,k}.$$

"Auf jedem Feld steht höchstens eine Zahl":

$$\varphi_{B2} := \bigwedge_{i,j=1}^{9} \bigwedge_{\substack{k,\ell=1\\ k\neq\ell}}^{9} \neg(P_{i,j,k} \wedge P_{i,j,\ell}).$$

Sudoku (Forts.)

Zeilen:

„Jede Zahl kommt in jeder Zeile vor“:

$$\varphi_{Ze} \;:=\; \bigwedge_{i=1}^{9} \bigwedge_{k=1}^{9} \bigvee_{j=1}^{9} P_{i,j,k}.$$

Spalten:

„Jede Zahl kommt in jeder Spalte vor“:

$$\varphi_{Sp} \;:=\; \bigwedge_{j=1}^{9} \bigwedge_{k=1}^{9} \bigvee_{i=1}^{9} P_{i,j.k}.$$

Blöcke:

„Jede Zahl kommt in jedem Block vor“:

$$\varphi_{Bl} \;:=\; \bigwedge_{i,j=0}^{2} \bigwedge_{k=1}^{9} \bigvee_{i',j'=1}^{3} P_{3i+i',3j+j',k}.$$

Anfangsbeschriftung:

Für eine spezifisches Sudoku

$$S : \{1, \dots, 9\} \times \{1, \dots, 9\} \to \{1, \dots, 9, \square\}$$

setzen wir

$$\Phi_{A(S)} \coloneqq \left\{ P_{i,j,k} \;\middle|\; i, j, k \in \{1, \dots, 9\} \text{ so dass } S(i,j) = k \right\}.$$

Gesamte aussagenlogische Spezifikation eines Sudokus

Für eine Sudoku S setzen wir

$$\Phi_S \coloneqq \{\varphi_{B1}, \varphi_{B2}, \varphi_{Ze}, \varphi_{Sp}, \varphi_{Bl}\} \cup \Phi_{A(S)}.$$

Beobachtungen

Sei S eine Sudoku und $\mathfrak{A} : \{P_{i,j,k} \mid i,j,k \in [9]\} \to \{0,1\}$ eine Interpretation.

(1) *Falls $\mathfrak{A} \models \{\varphi_{B1}, \varphi_{B2}\}$, dann gibt es (genau) eine Abbildung $L_{\mathfrak{A}} : [9] \times [9] \to [9]$, so dass für alle $i,j,k \in [9]$:*
$$\mathfrak{A}(P_{i,j,k}) = 1 \iff L_{\mathfrak{A}}(i,j) = k.$$

(2) *Falls $\mathfrak{A} \models \{\varphi_{B1}, \varphi_{B2}, \varphi_{Ze}, \varphi_{Sp}, \varphi_{Bl}\}$, dann gibt es für jedes $k \in [9]$ in jeder Zeile, in jeder Spalte, und in jedem Block eine Zelle (i,j), so dass $L_{\mathfrak{A}}(i,j) = k$.*

(3) *Falls $\mathfrak{A} \models \{\varphi_{B1}, \varphi_{B2}\} \cup \Phi_{A(S)}$, dann erweitert $L_{\mathfrak{A}}$ die Anfangsbeschriftung von S, d.h., für alle $i,j \in [9]$ mit $S(i,j) \neq \square$ gilt $L_{\mathfrak{A}}(i,j) = S(i,j)$.*

(4) *Falls $\mathfrak{A} \models \Phi_S$, dann $L_{\mathfrak{A}} \in \mathscr{L}(S)$, das heißt, $L_{\mathfrak{A}}$ is eine Lösung für S.*

(5) *Für alle Lösungen $L \in \mathscr{L}(S)$ sei $\mathfrak{A}_L : \{P_{i,j,k} \mid i,j,k \in [9]\} \to \{0,1\}$ die Interpretation mit $\mathfrak{A}_L(P_{i,j,k}) = 1 \iff L(i,j) = k$. Dann gilt $\mathfrak{A}_L \models \Phi_S$.*

Sudoku (Forts.)

Insgesamt

(1) *Für alle* $\mathfrak{A} \models \Phi_S$ *ist* $L_{\mathfrak{A}} \in \mathscr{L}(S)$.

(2) *Für alle* $L \in \mathscr{L}(S)$ *gilt* $\mathfrak{A}_L \models \Phi_S$.

Außerdem gilt $L = L_{\mathfrak{A}_L}$.

Die Modelle von Φ_S *entsprechen also gerade den Lösungen des Sudokus* S.

Beispiel: Stundenplanung

Ein Stundenplanungsproblem S sei gegeben durch

- eine Menge L von Lehrveranstaltungen,
- eine Menge D von Dozierenden,
- eine Abbildung $\mathscr{d} : L \to D$, die jeder Lehrveranstaltung $\ell \in L$ eine Dozierende $\mathscr{d}(\ell) \in D$ zuordnet,
- eine Menge R von Räumen,
- eine Menge Z von Zeitfenstern (von denen wir annehmen, dass sie sich nicht überlappen).

Ein Planung ist eine Abbildung $P : L \to R \times Z$, die Lehrveranstaltungen Räume und Zeitfenstern zuordnet.

Eine Planung ist korrekt, wenn sie folgende Bedingungen erfüllt.

- Zu jedem Zeitpunkt kann in jedem Raum nur eine Lehrveranstaltung stattfinden, das heißt, für $\ell_1, \ell_2 \in L, \ell_1 \neq \ell_2$ gilt $P(\ell_1) \neq P(\ell_2)$.
- Dozierende können zu jedem Zeitpunkt nur eine Veranstaltung abhalten, d.h., für alle $\ell_1, \ell_2 \in L$, $r_1, r_2 \in R$, $z_1, z_2 \in Z$, wenn $\mathscr{d}(\ell_1) = \mathscr{d}(\ell_2)$ und $P(\ell_1) = (r_1, z_1)$ und $P(\ell_2) = (r_2, z_2)$, dann $z_1 \neq z_2$.

Stundenplanung (Forts.)

Ziel

Zu jedem Stundenplanungsproblem $S = (L, D, \mathcal{d}, R, Z)$ konstruieren wir eine Formelmenge Φ_S, deren Modelle gerade den korrekten Planungen für S entsprechen.

Aussagensymbole

- Für alle $\ell \in L$, $r \in R$, $z \in Z$ ein Symbol $P_{\ell,r,z}$ mit der intuitiven Bedeutung:

  „Die Veranstaltung ℓ findet im Raum r zum Zeitpunkt z statt."
- Für alle $\ell \in L$ und $d \in D$ ein Symbol $Q_{\ell,d}$ mit der intuitiven Bedeutung:

  „Die Veranstaltung ℓ wird vom Dozierenden d gehalten."

Abbildungsbedingungen

„Jede Lehrveranstaltung hat mindestens einen Raum und einen Zeitfenster“:

$$\varphi_{A1} := \bigwedge_{\ell \in L} \bigvee_{r \in R, z \in Z} P_{\ell,r,z}.$$

„Jede Lehrveranstaltung hat höchstens einen Raum und einen Zeitfenster“:

$$\varphi_{A2} := \bigwedge_{\ell \in L} \bigwedge_{\substack{r,r' \in R, z,z' \in Z \\ (r,z) \neq (r',z')}} \neg(P_{\ell,r,z} \wedge P_{\ell,r',z'}).$$

Raumbedingungen

„Zu jedem Zeitpunkt kann in jedem Raum nur eine Lehrveranstaltung stattfinden“:

$$\varphi_R := \bigwedge_{r \in R, z \in Z} \bigwedge_{\substack{\ell,\ell' \in L \\ \ell \neq \ell'}} \neg(P_{\ell,r,z} \wedge P_{\ell',r,z}).$$

Stundenplanung (Forts.)

Dozierendenbedingungen

„Die Symbole $Q_{\ell,d}$ beschreiben die Abbildung $\mathscr{d}$“:

$$\varphi_{D1} \coloneqq \bigwedge_{\ell \in L} \Big(Q_{\ell,\mathscr{d}(\ell)} \wedge \bigwedge_{d \in D \setminus \{\mathscr{d}(\ell)\}} \neg Q_{\ell,d} \Big).$$

„Dozierende können zu jedem Zeitpunkt nur eine Veranstaltung abhalten.“

$$\varphi_{D2} \coloneqq \bigwedge_{d \in D} \bigwedge_{z \in Z} \bigwedge_{\substack{\ell,\ell' \in L \\ \ell \neq \ell'}} \bigwedge_{r,r' \in R} \neg (Q_{\ell,d} \wedge Q_{\ell',d} \wedge P_{\ell,r,z} \wedge P_{\ell',r',z}).$$

Gesamtspezifikation

$$\Phi_S \coloneqq \{\varphi_{A1}, \varphi_{A2}, \varphi_R, \varphi_{D1}, \varphi_{D2}\}.$$

Stundenplanung (Forts.)

Beobachtungen

(1) *Für alle Interpretationen* $\mathfrak{A} \models \Phi_S$ *definiert*

$$P_{\mathfrak{A}}(\ell) = (z, r) \;:\Longleftrightarrow\; \mathfrak{A}(P_{\ell,z,r}) = 1$$

eine korrekte Planung für S.

(2) *Für alle korrekten Planungen* P *für* S *erfüllt folgende Interpretation* $\mathfrak{A}_P$ *die Spezifikation* Φ_S:

$$\mathfrak{A}_P(P_{\ell,r,z}) := \begin{cases} 1 & \text{wenn } P(\ell) = (r, z), \\ 0 & \text{sonst,} \end{cases} \qquad \mathfrak{A}(Q_{\ell,d}) := \begin{cases} 1 & \text{wenn } \mathcal{d}(\ell) = d, \\ 0 & \text{sonst.} \end{cases}$$

für alle $\ell \in L, r \in R, z \in Z, d \in D$.

Die Modelle von Φ_S *entsprechen also genau den korrekten Plänen für* S.

Ergänzungen zu Seite 1.38

Es gibt in der Regel nicht eine „korrekte“ oder „beste“ aussagenlogische Spezifikation.
Stundenplanungsprobleme könnten wir beispielsweise auch ohne explizite Variablen $Q_{\ell,d}$ beschreiben.
(Wie? — Übung 😀)

Beispiel: Berechnungen von Turingmaschinen

Zentraler Schritt im Beweis des Satzes von Cook und Levin (NP-Vollständigkeit des aussagenlogischen Erfüllbarkeitsproblems) ist folgendes Lemma.

Lemma 1.16

Sei M eine nichtdeterministische Turingmaschine mit Eingabealphabet Σ, und sei $p(X)$ ein Polynom. Dann gibt es für alle Eingaben $x \in \Sigma^*$ eine aussagenlogische Formel $\varphi_{M,p,x}$, die genau dann erfüllbar ist, wenn M bei Eingabe x eine akzeptierende Berechnung der Länge höchstens $p(|x|)$ hat.
Weiterhin gibt es ein Polynom $q(X)$ (das von M und $p(X)$, aber nicht von x abhängt), so dass $|\varphi_{M,p,x}| \leq q(|x|)$.

Um das Lemma zu beweisen, konstruiert man eine Formelmenge, deren Modelle die akzeptierenden Berechnungen von M beschreiben.

Ergänzungen zu Seite 1.39

Der Satz von Cook und Levin wird in der Vorlesung Berechenbarkeit und Komplexität behandelt. Ich empfehle, den Beweis noch einmal anzusehen. Er enthält wichtige Ideen, um den Zusammenhang zwischen Logik und Berechenbarkeit zu erklären. Dieses Thema werden wir in einem späteren Kapitel wieder aufgreifen.

Falls Sie die Vorlesung Berechenbarkeit und Komplexität nicht gehört haben, finden Sie den Satz von Cook und Levin auch in praktisch jedem Lehrbuch zur theoretischen Informatik, beispielsweise (Sipser 2021; Wegener 2005).

Anmerkungen zur aussagenlogischen Modellierung

- Wir haben in verschiedenen Szenarien Problemstellungen durch eine aussagenlogische Formel oder Formelmenge Φ beschrieben, so dass die Modelle von Φ den „Lösungen“ der ursprünglichen Problemstellung entsprechen.
- Man mag sich fragen, was der Nutzen dieser aussagenlogischen Modellierung ist.
- Ein wichtiger Nutzen ist, dass wir jetzt SAT-Solver, das sind Algorithmen bzw. Softwarepakete zum Finden von erfüllenden Interpretationen für aussagenlogische Formeln, verwenden können, um unsere Probleme zu lösen.

  Über solche Algorithmen sprechen wir in Abschnitt 1.5.
- Ein weitere Nutzen nicht nur der aussagenlogischen Modellierung, sondern ganz allgemein der Modellierung in einer formalen Logik, besteht darin die Problemstellung präzisieren und eventuelle Unklarheiten entdecken und beheben zu können.

1.4 Normalformen und Funktionale Vollständigkeit

Äquivalenz

Definition 1.17

Zwei aussagenlogische Formeln φ, ψ sind äquivalent, wenn für alle Interpretationen $\mathfrak{A}$ gilt:

$$\mathfrak{A} \models \varphi \quad \Longleftrightarrow \quad \mathfrak{A} \models \psi.$$

Wir schreiben $\varphi \equiv \psi$.

Beispiel 1.18

$$\neg(P \wedge Q) \rightarrow \bot \quad \equiv \quad \neg(P \rightarrow \neg Q)$$

Ergänzungen zu Seite 1.42

Wir können die Äquivalenz nachweisen, indem wir Wahrheitstafeln konstruieren und vergleichen. Für die Formeln im Beispiel erhalten wir:

P	Q	$\neg(P \wedge Q) \to \bot$
0	0	0
0	1	0
1	0	0
1	1	1

P	Q	$\neg(P \to \neg Q)$
0	0	0
0	1	0
1	0	0
1	1	1

Beobachtung 1.19

Zwei aussagenlogische Formeln $\varphi(P_1, \dots, P_n)$ und $\psi(P_1, \dots, P_n)$ sind genau dann äquivalent, wenn in der letzten Spalte ihrer Wahrheitstafeln jeweils die gleichen Einträge stehen.

Wir führen folgende abkürzende Schreibweise ein: für Formeln $\varphi, \psi \in \mathrm{AL}$ schreiben wir

$$\varphi \leftrightarrow \psi \quad \text{statt} \quad (\varphi \to \psi) \wedge (\psi \to \varphi).$$

Beobachtung 1.20

Für alle Formeln $\varphi, \psi \in \mathrm{AL}$ gilt:

$$\varphi \equiv \psi \quad \iff \quad \varphi \leftrightarrow \psi \textit{ ist allgemeingültig.}$$

Ergänzungen zu Seite 1.43

Um äquivalent zu sein müssen Formeln nicht notwendigerweise dieselben Aussagensymbole enthalten. Beispielsweise gilt

$$P \vee \neg P \equiv \top. \qquad (\star)$$

Überprüft man die Äquivalenz von Formeln φ, ψ mit Hilfe von Wahrheitstafeln, ist es wichtig, dass bei beiden Formeln dieselben Aussagensymbole in der Wahrheitstafel verwendet werden, also die Symbole in $\text{symb}(\varphi) \cup \text{symb}(\psi)$.

Um die Äquivalenz $(\star)$ zu überprüfen konstruiert man also Wahrheitstafeln in P:

P	$P \vee \neg P$
0	1
1	1

P	$\top$
0	1
1	1

Ein formaler Beweis von Beobachtung 1.20 ist eine gute Übung.

Äquivalenzregeln

Die folgenden Äquivalenzen gelten für alle Formeln $\varphi, \psi, \chi \in \mathrm{AL}$.

Konjunktions- und Disjunktionsregeln

- Idempotenz: $\varphi \wedge \varphi \equiv \varphi, \qquad \varphi \vee \varphi \equiv \varphi.$
- Kommutativität: $\varphi \wedge \psi \equiv \psi \wedge \varphi, \qquad \varphi \vee \psi \equiv \psi \vee \varphi.$
- Assoziativität:
$$(\varphi \wedge \psi) \wedge \chi \equiv \varphi \wedge (\psi \wedge \chi),$$
$$(\varphi \vee \psi) \vee \chi \equiv \varphi \vee (\psi \vee \chi)$$
- Distributivität:
$$\varphi \wedge (\psi \vee \chi) \equiv (\varphi \wedge \psi) \vee (\varphi \wedge \chi),$$
$$\varphi \vee (\psi \wedge \chi) \equiv (\varphi \vee \psi) \wedge (\varphi \vee \chi).$$

Äquivalenzregeln (Forts.)

Negationsregeln

- Doppelte Negation:

$$\neg\neg\varphi \;\equiv\; \varphi$$

- De Morgansche Regeln:

$$\neg(\varphi \wedge \psi) \equiv (\neg\varphi \vee \neg\psi),$$
$$\neg(\varphi \vee \psi) \equiv (\neg\varphi \wedge \neg\psi).$$

Implikationsregeln

- Elimination der Implikation:

$$\varphi \to \psi \equiv \neg\varphi \vee \psi$$

Äquivalenzregeln (Forts.)

Regeln für boolesche Konstanten

- $$\neg\bot \equiv \top, \qquad \neg\top \equiv \bot.$$

- $$\begin{aligned} \varphi \wedge \top &\equiv \varphi, & \varphi \vee \bot &\equiv \varphi, \\ \varphi \wedge \bot &\equiv \bot, & \varphi \vee \top &\equiv \top, \end{aligned}$$
  $$\varphi \to \bot \equiv \neg\varphi.$$

- Tertium Non Datur:
  $$(\varphi \wedge \neg\varphi) \equiv \bot, \qquad (\varphi \vee \neg\varphi) \equiv \top.$$

Ergänzungen zu Seite 1.46

Alle Äquivalenzregeln können leicht mit Hilfe von Wahrheitstafeln bewiesen werden. Dazu beobachten wir zunächst, dass die Wahrheitswerte der Formeln auf beiden Seiten der Äquivalenzen nur von den Wahrheitswerten der Teilformeln φ, ψ, χ abhängt. Indem wir also die Werte der Formeln unter allen möglichen Werten der Teilformeln vergleichen, können wir die Äquivalenz überprüfen.

Zum Beispiel erhalten wir für die erste de Morgansche Regel

$$\neg(\varphi \wedge \psi) \equiv (\neg\varphi \vee \neg\psi).$$

folgende Wahrheitstafeln:

φ	ψ	$(\varphi \wedge \psi)$	$\neg(\varphi \wedge \psi)$
0	0	0	1
0	1	0	1
1	0	0	1
1	1	1	0

für die linke Seite und

φ	ψ	$\neg\varphi$	$\neg\psi$	$(\neg\varphi \vee \neg\psi)$
0	0	1	1	1
0	1	1	0	1
1	0	0	1	1
1	1	0	0	0

für die rechte Seite. Die letzten Spalten der beiden Wahrheitstafeln sind gleich, also sind die Formeln äquivalent.

Verwendung der Äquivalenzregeln

Durch schrittweises Anwenden der Äquivalenzregeln können wir eine gegebene Formel in eine zu ihr äquivalente Formel umformen und damit neue Äquivalenzen ableiten.

Beispiel 1.21

Betrachten wir die Äquivalenz $\neg(P \wedge Q) \to \bot \equiv \neg(P \to \neg Q)$, die wir bereits aus Beispiel 1.18 kennen. Es gilt

$$\begin{aligned}
\neg(P \wedge Q) \to \bot &\equiv \neg\neg(P \wedge Q) && \text{Konstantenregel} \\
&\equiv \neg(\neg P \vee \neg Q) && \text{De Morgan} \\
&\equiv \neg(\neg\neg P \to \neg Q) && \text{Implikation} \\
&\equiv \neg(P \to \neg Q) && \text{Doppelte Negation.}
\end{aligned}$$

Negationsnormalform

Definition 1.22

Eine Formel ist in Negationsnormalform (NNF), wenn sie keine Implikationen enthält und wenn Negationszeichen nur unmittelbar vor Aussagensymbolen auftreten.

Satz 1.23

Jede aussagenlogische Formel ist äquivalent zu einer Formel in NNF.

Beweis.

Per Induktion über den Aufbau definieren wir zu jeder Formel $\varphi \in \mathrm{AL}$ zwei Formeln φ' und φ'' in NNF, so dass gilt:

$$\varphi \equiv \varphi' \quad \text{und} \quad \neg\varphi \equiv \varphi''. \tag{$\star$}$$

Induktionsanfang:

$\varphi = P$ für ein Aussagensymbol P. Wir setzen $\varphi' := P$ und $\varphi'' := \neg P$.

$\varphi = \bot$. Wir setzen $\varphi' := \bot$ und $\varphi'' := \top$.

$\varphi = \top$. Wir setzen $\varphi' := \top$ und $\varphi'' := \bot$.

In allen drei Fällen gilt offensichtlich ($\star$).

Induktionsschritt:

$\varphi = \neg\psi$ für eine Formel ψ. Wir setzen $\varphi' := \psi''$ und $\varphi'' := \psi'$.

Dann folgt ($\star$) unmittelbar der Induktionsannahme.

$\varphi = \psi_1 \wedge \psi_2$ für Formeln ψ_1, ψ_2. Wir setzen

$$\varphi' := (\psi_1' \wedge \psi_2'),$$
$$\varphi'' := (\psi_1'' \vee \psi_2'').$$

Nach Induktionsannahme gilt $\psi_1 \equiv \psi_1'$ und $\psi_2 \equiv \psi_2'$, also auch $\varphi \equiv \varphi'$. Außerdem gilt

$$\begin{aligned} \neg\varphi &\equiv (\neg\psi_1 \vee \neg\psi_2) && \text{De Morgan} \\ &\equiv (\psi_1'' \vee \psi_2'') && \text{Induktionsannahme} \end{aligned}$$

Also gilt $(\star)$.

$\varphi = \psi_1 \vee \psi_2$ für Formeln ψ_1, ψ_2. Wir setzen

$$\varphi' := (\psi_1' \vee \psi_2'),$$
$$\varphi'' := (\psi_1'' \wedge \psi_2'').$$

Dann können wir $(\star)$ wieder mit Hilfe der Induktionsannahme und der De Morganschen Regel nachweisen.

$\varphi = \psi_1 \to \psi_2$ für Formeln ψ_1, ψ_2. Wir verwenden die Äquivalenz $\psi_1 \to \psi_2 \equiv \neg\psi_1 \vee \psi_2$ und kombinieren dann die Schritte für Disjunktion und Negation. Insgesamt erhalten wir

$$\varphi' := (\psi_1'' \vee \psi_2'),$$
$$\varphi'' := (\psi_1' \wedge \psi_2'').$$

Die Formeln φ' und φ'' sind in NNF, weil Negationszeichen nur im Induktionsanfang verwendet werden und dort unmittelbar vor einem Aussagensymbol stehen. $\square$

Umwandlung einer Formel in NNF

Der Beweis von Satz ist konstruktiv und gibt uns einen rekursiven Algorithmus, um eine gegebene Formel in eine Formel in NNF umzuwandeln.

Beispiel 1.24

Wir wandeln die Formel $\big(\neg P \wedge \neg((P \vee Q) \to P)\big) \to \bot$ in NNF um:

$$\begin{aligned}
\Big(\big(\neg P \wedge \neg((P \vee Q) \to P)\big) \to \bot\Big)' &\equiv \big(\neg P \wedge \neg((P \vee Q) \to P)\big)'' \vee (\bot)' \\
&\equiv \Big((\neg P)'' \vee \big(\neg((P \vee Q) \to P)\big)''\Big) \vee \bot \\
&\equiv \Big(P \vee ((P \vee Q) \to P)'\Big) \vee \bot \\
&\equiv \Big(P \vee ((P \vee Q)'' \vee (P)')\Big) \vee \bot \\
&\equiv \Big(P \vee (((P)'' \wedge (Q)'') \vee P)\Big) \vee \bot \\
&\equiv \Big(P \vee ((\neg P \wedge \neg Q) \vee P)\Big) \vee \bot.
\end{aligned}$$

Umwandlung in NNF (Forts.)

Etwas einfacher wird die Umwandlung, wenn man in der Eingabeformel zunächst alle Implikationen mittels der Äquivalenzregel eliminiert und dann die Negationen mit Hilfe der de Morganschen Regeln „nach innen schiebt“. Dabei entstehende doppelte Negationen kann man auf Grund der entsprechenden Äquivalenzregel weglassen.

Beispiel 1.24 (Forts.)

In unserem Beispiel sieht das wie folgt aus

$$\begin{aligned}
\Big(\neg P \wedge \neg\big((P \vee Q) \to P\big)\Big) \to \bot &\equiv \neg\Big(\neg P \wedge \neg\big(\neg(P \vee Q) \vee P\big)\Big) \vee \bot \\
&\equiv \Big(\neg\neg P \vee \neg\neg\big(\neg(P \vee Q) \vee P\big)\Big) \vee \bot \\
&\equiv \Big(P \vee \big(\neg(P \vee Q) \vee P\big)\Big) \vee \bot \\
&\equiv \Big(P \vee \big((\neg P \wedge \neg Q) \vee P\big)\Big) \vee \bot.
\end{aligned}$$

Iterierte Konjunktionen und Disjunktionen

Weil $\wedge$ assoziativ ist, können wir Formeln der Gestalt $\bigwedge_{i=1}^{n} \varphi_i$ etwas großzügiger interpretieren. Von nun an stehe $\bigwedge_{i=1}^{n} \varphi_i$ für $\varphi_1 \wedge \cdots \wedge \varphi_n$ mit irgendeiner Klammerung.

Entsprechend verfahren wir mit Disjunktionen.

Beispiel 1.25

Die Formel $\bigwedge_{i=1}^{4} \varphi_i$ kann für jede der folgenden Formeln stehen:

$$(((\varphi_1 \wedge \varphi_2) \wedge \varphi_3) \wedge \varphi_4), \quad ((\varphi_1 \wedge (\varphi_2 \wedge \varphi_3)) \wedge \varphi_4), \quad ((\varphi_1 \wedge \varphi_2) \wedge (\varphi_3 \wedge \varphi_4))$$
$$(\varphi_1 \wedge ((\varphi_2 \wedge \varphi_3) \wedge \varphi_4)), \quad (\varphi_1 \wedge (\varphi_2 \wedge (\varphi_3 \wedge \varphi_4))).$$

Wegen der Kommutativität kommt es bei Konjunktionen oder Disjunktionen nicht auf die Reihenfolge an, und wegen der Idempotenz können wir doppelte Formeln streichen. Das bedeutet, dass wir von Konjunktionen und Disjunktionen über endliche Mengen von Formeln sprechen können.

Als Spezialfall lassen wir sogar Konjunktionen und Disjunktionen über die leere Menge zu: die leere Konjunktion steht dabei einfach für $\top$, die leere Disjunktion für $\bot$.

Ergänzungen zu Seite 1.51

Die Konventionen für leere Konjunktionen und Disjunktionen sind semantisch sinnvoll:

- Eine Konjunktion über eine Menge von Formeln ist erfüllt, wenn alle Formeln in der Menge erfüllt sind. Bei einer leeren Menge ist das automatisch gegeben, deswegen ist die leere Konjunktion immer wahr.
- Eine Disjunktion über eine Menge von Formeln ist erfüllt, wenn mindestens eine Formel in der Menge erfüllt ist. Bei einer leeren Menge ist das nicht möglich, deswegen ist die leere Disjunktion immer falsch.

Literale und Klauseln

Definition 1.26

(1) Ein Literal ist eine Formel der Gestalt P oder $\neg P$ für ein Aussagensymbol P.
Im ersten Fall sprechen wir von einem positiven, im zweiten Fall von einem negativen Literal.
Für eine Literal λ bezeichnen wir mit $\overline{\lambda}$ das zu λ komplementäre Literal, also $\overline{\lambda} := \neg P$ falls $\lambda = P$ und $\overline{\lambda} := P$ falls $\lambda = \neg P$.

(2) Eine disjunktive Klausel ist eine Disjunktion von Literalen, also eine Formel der Gestalt
$$\bigvee_{i=1}^{k} \lambda_i, \text{ für Literale } \lambda_1, \ldots, \lambda_k.$$
Als Spezialfall lassen wir auch den Fall $k = 0$ zu, also die leere Disjunktion $\bot$.

(3) Eine konjunktive Klausel ist eine Konjunktion von Literalen, also eine Formel der Gestalt
$$\bigwedge_{i=1}^{k} \lambda_i, \text{ für Literale } \lambda_1, \ldots, \lambda_k.$$
Als Spezialfall lassen wir auch den Fall $k = 0$ zu, also die leere Konjunktion $\top$.

Ergänzungen zu Seite 1.52

In der Literatur spricht man häufig einfach von „Klauseln“, gemeint sind dann disjunktive Klauseln. Konjunktive Klauseln werden in diesem Zusammenhang manchmal als Terme bezeichnet, die Bezeichnung verwenden wir aber nicht, weil Terme bei uns später eine völlig andere Bedeutung haben.

Konjunktive und disjunktive Normalform

Definition 1.27

(1) Eine aussagenlogische Formel ist in konjunktiver Normalform (KNF), wenn sie eine Konjunktion von disjunktiven Klauseln ist, also die Gestalt

$$\bigwedge_{i=1}^{\ell} \bigvee_{j=1}^{k_i} \lambda_{i,j}$$

hat, für $\ell, k_1, \ldots, k_m \in \mathbb{N}$ und Literale $\lambda_{i,j}$.

(2) Eine aussagenlogische Formel ist in disjunktiver Normalform (DNF), wenn sie eine Disjunktion von konjunktiven Klauseln ist.

Beispiele 1.28

(1) $(P_1 \wedge \neg P_2 \wedge P_3) \vee (\neg P_2 \wedge \neg P_3) \vee (P_2 \wedge P_1)$ ist in DNF, aber nicht in KNF.

(2) $(P_1 \vee \neg P_2 \vee P_3) \wedge (\neg P_2 \vee \neg P_3) \wedge (P_2 \vee P_1)$ ist in KNF, aber nicht in DNF.

(3) $P_1 \wedge (P_2 \vee (P_3 \wedge P_4))$ ist weder in KNF noch in DNF.

(4) $P_1 \wedge (P_2 \wedge (P_3 \vee P_4))$ ist in KNF, aber nicht in DNF.

(5) $P_1 \vee \neg P_1$ ist in KNF und in DNF.

(6) $\perp$ ist in KNF und in DNF.

Satz über DNF und KNF

Satz 1.29

Jede aussagenlogische Formel ist äquivalent zu einer Formel in DNF und zu einer Formel in KNF.

Beweis.

Wir führen zunächst den Beweis für DNF.

Am einfachsten ist es, den Beweis per Induktion über den Aufbau der Formeln zu führen. Wir führen einen etwas anderen Beweis, der dem entspricht, was man bei der praktischen Umwandlung einer Formel in eine äquivalente DNF Formel tut.

Sei $\varphi \in \mathrm{AL}$ eine beliebige Formel.

Ziel:

Konstruktion einer zu φ äquivalenten Formel in DNF.

Schritt 1:

Konstruktion einer zu φ äquivalenten Formel φ_1 in NNF (ist möglich nach Satz 1.23).

Schritt 2:

Wiederhole folgende Umformungen, solange das noch möglich ist:

- Wenn φ_1 noch eine Subformel ψ der Gestalt $\psi_1 \wedge (\chi_1 \vee \chi_2)$ enthält, ersetze ψ durch $(\psi_1 \wedge \chi_1) \vee (\psi_1 \wedge \chi_2)$.

- Wenn φ_1 noch eine Subformel ψ der Gestalt $(\chi_1 \vee \chi_2) \wedge \psi_1$ enthält, ersetze ψ durch $(\chi_1 \wedge \psi_1) \vee (\chi_2 \wedge \psi_1)$.

Sei φ_2 die Formel, die durch diese Umformungen entsteht.

Um zu sehen, dass das Verfahren nach endlich vielen Schritten terminiert, betrachtet man am besten den Syntaxbaum der Formel. Man zählt für jedes Auftreten eines Disjunktionssymbols, also jeden $\vee$-Knoten v im Syntaxbaum, wieviele $\wedge$-Knoten auf dem Weg von v zur Wurzel vorkommen. Diese Zahl verringert sich für mindestens ein Auftreten eines Disjunktionssymbols, und und sie vergrößert sich für kein Auftreten. Außerdem bleibt die Höhe des Baums gleich. Deswegen macht man bei jeder Iteration eine echten Fortschritt, und das Verfahren terminiert. (Eine weitere Formalisierung dieses Arguments verwendet eine Potentialfunktion, die sich in jedem Schritt echt verringert. Wir verzichten hier darauf.)

Wegen der Distributivität und Kommutativität ist nach jedem Ersetzungsschritt ist die neue Formel äquivalent zur alten. Also ist φ_2 äquivalent to φ_1 und damit zu ψ.

φ_2 ist eine Disjunktion von Konjunktion von Literalen und booleschen Konstanten, das heißt,

$$\psi_2 = \bigvee_{i=1}^{\ell} \chi_i$$

wobei χ_i eine Konjunktion von Literalen und booleschen Konstanten ist.

- Wenn χ_i die boolesche Konstante $\bot$ enthält, dann ist die Konjunktion unerfüllbar, und wir streichen χ_i aus der Disjunktion.
- Wenn χ_i die boolesche Konstante $\top$ enthält, dann streichen wir diese Konstante aus der Konjunktion.

Die entstehende Formel φ_3 ist äquivalent zu φ_2 und damit zu φ, und sie ist in DNF.

Den Beweis für KNF könnten wir jetzt völlig analog führen. Stattdessen können wir auch wie folgt argumentieren. Um eine Formel φ in KNF zu bringen, bringen wir $\neg\varphi$ in DNF. Sei

$$\psi = \bigvee_{i=1}^{\ell} \bigwedge_{j=1}^{k_i} \lambda_{i,j}$$

die resultierende DNF Formel. Dann ist φ äquivalent to $\neg\psi$. Wir müssen also jetzt $\neg\psi$ in KNF bringen. Das ist einfach, unter Verwendung der de Morganschen Regeln erhalten wir

$$\neg\psi \equiv \bigwedge_{i=1}^{\ell} \bigvee_{j=1}^{k_i} \overline{\lambda}_{i,j}.$$

□

Bemerkung 1.30

Wir werden später noch einen zweiten Beweis des Satzes sehen. In gewisser Weise ist der einfacher, aber dafür algorithmisch weniger effizient.

Umwandlung in KNF und DNF

Der Beweis von Satz 1.29 ist konstruktiv und gibt uns eine Verfahren, eine gegebene Formel in eine KNF oder in eine DNF Formel umzuwandeln.

Beispiel 1.31

Wir betrachten wieder die Formel $\varphi := \Big(\neg P \wedge \neg\big((P \vee Q) \to P\big)\Big) \to \bot$ aus Beispiel 1.24 und wandeln sie KNF um.

Schritt 1. Umwandlung in NNF (vgl. Beispiel 1.24): $\varphi \equiv \Big(P \vee \big((\neg P \wedge \neg Q) \vee P\big)\Big) \vee \bot$.

Schritt 2. Anwendung der de Morganschen Regeln:

$$\begin{aligned}
\Big(P \vee \big((\neg P \wedge \neg Q) \vee P\big)\Big) \vee \bot &\equiv \Big(P \vee \big((\neg P \vee P) \wedge (\neg Q \vee P)\big)\Big) \vee \bot \\
&\equiv \Big((P \vee \neg P \vee P) \wedge (P \vee \neg Q \vee P)\Big) \vee \bot \\
&\equiv (P \vee \neg P \vee P \vee \bot) \wedge (P \vee \neg Q \vee P \vee \bot).
\end{aligned}$$

Schritt 3. Elimination der Booleschen Konstanten:

$$(P \vee \neg P \vee P \vee \bot) \wedge (P \vee \neg Q \vee P \vee \bot) \equiv (P \vee \neg P \vee P) \wedge (P \vee \neg Q \vee P).$$

Größe der Normalformen

- Die Umwandlung in NNF vergrößert eine Formel nicht wesentlich:
  Zu jeder Formel $\varphi \in \mathrm{AL}$ *gibt es eine äquivalent Formel* φ' *in NNF, so dass* $|\varphi'| = O(|\varphi|)$.
- Anders ist es bei DNF und KNF:
  Es gibt eine Familie $(\varphi_n)_{n \in \mathbb{N}_{>0}}$ *von Formeln* $\varphi_n \in \mathrm{AL}$ *der Länge* $|\varphi_n| = O(n)$, *so dass für alle* $n \in \mathbb{N}_{>0}$ *sowohl die kleinste zu* φ_n *äquivalente DNF-Formel als auch die kleinste zu* φ_n *äquivalente KNF-Formel Länge mindestens* 2^n *haben.*

Ergänzungen zu Seite 1.57

Wir beweisen diese Aussagen nicht. Hier nur einige kurze Bemerkungen zur Idee der Beweise:

- Die obere Schranke für NNF folgt leicht aus dem Beweis von Satz 1.23.
- Für die untere Schranke für DNF und KNF konstruieren wir für alle n eine Formel $\varphi_n(P_1, \dots, P_{n+1})$ der Größe $O(n)$, die genau dann von einer Interpretation $\mathfrak{A}$ erfüllt wird, wenn $\mathfrak{A}(P_i) = 1$ für eine gerade Anzahl von $i \in [n+1]$. Es lässt sich dann zeigen, dass es zu dieser Formel keine äquivalenten DNF oder KNF Formeln mit weniger als 2^n Klauseln gibt.

Eine Abbildung $F : \{0,1\}^n \to \{0,1\}$ bezeichnen wir als n-stellige Boolesche Funktion.

Beispiel 1.32

Die boolesche Funktion $EVEN_n : \{0,1\}^n \to \{0,1\}$ sei definiert durch

$$EVEN_n(x_1, \dots, x_n) := \begin{cases} 1 & \text{falls } \sum_{i=1}^n x_i \equiv 0 \bmod 2, \\ 0 & \text{sonst.} \end{cases}$$

Also $EVEN_n(x_1, \dots, x_n) = 1$ genau dann, wenn $x_i = 1$ für eine gerade Anzahl von $i \in [n]$.

Beobachtung 1.33

Jede Formel $\varphi(P_1, \dots, P_n) \in \mathrm{AL}$ definiert eine n-stellige boolesche Funktion $F_{\varphi(P_1,\dots,P_n)} : \{0,1\}^n \to \{0,1\}$ definiert durch

$$F_{\varphi(P_1,\dots,P_n)}(x_1, \dots, x_n) := [\![\varphi]\!]^{\mathfrak{A}}$$

für die Interpretation $\mathfrak{A} : \{P_1, \dots, P_n\} \to \{0,1\}$ mit $\mathfrak{A}(P_i) = x_i$.

Ergänzungen zu Seite 1.58

Wir nehmen hier an, dass $P_i \in \sigma$ für alle $i \in \mathbb{N}_{>0}$.

Zur Erinnerung

Die Schreibweise $\varphi(P_1, \dots, P_n)$ zeigt an, dass $\mathrm{symb}(\varphi) \subseteq \{P_1, \dots, P_n\}$.

Es ist hier wichtig, dass wir die Symbole P_i explizit auflisten. Die Funktion $F_{\varphi(P_1,\dots,P_n)}$ hängt im allgemeinen von der Reihenfolge ab, in der wir die Symbole auflisten.

Es ist außerdem nützlich, dass wir auch $\mathrm{symb}(\varphi) \subset \{P_1, \dots, P_n\}$ erlauben. Dadurch können wir mit einer Formel mit k Aussagensymbolen auch n-stellige boolesche Funktion für $n > k$ definieren.

Beispiel.

Die Formel $\top$ mit $\mathrm{symb}(\top) = \emptyset$ definiert für alle $n \in \mathbb{N}$ die konstante boolesche Funktion $F_{\top(P_1,\dots,P_n)} : \{0,1\}^n \to \{0,1\}$ mit $F_{\top(P_1,\dots,P_n)}(x_1, \dots, x_n) = 1$ für alle $x_1, \dots, x_n \in \{0,1\}$.

Satz 1.34

Zu jeder booleschen Funktion $F : \{0,1\}^n \to \{0,1\}$ *gibt es eine Formel* $\varphi_F(P_1, \dots, P_n) \in \mathrm{AL}$, *so dass*

$$F = F_{\varphi_F(P_1,\dots,P_n)}.$$

Beweis.

Für $i \in [n]$ und $x \in \{0,1\}$ sei $\lambda_{i,x} := \begin{cases} P_i & \text{falls } x = 1, \\ \neg P_i & \text{falls } x = 0. \end{cases}$

Weiterhin sei für $(x_1, \dots, x_n) \in \{0,1\}^n$

$$\chi_{(x_1,\dots,x_n)}(P_1, \dots, P_n) := \lambda_{1,x_1} \wedge \dots \wedge \lambda_{n,x_n}.$$

Dann gilt für alle Interpretation $\mathfrak{A}$:

$$\mathfrak{A} \models \chi_{(x_1,\dots,x_n)} \iff \mathfrak{A}(P_i) = x_i \text{ für alle } i \in [n].$$

Jetzt setzen wir

$$\varphi_F(P_1, \dots, P_n) := \bigvee_{\substack{(x_1,\dots,x_n) \in \{0,1\}^n \\ \text{so dass } F(x_1,\dots,x_n)=1}} \chi_{(x_1,\dots,x_n)}.$$

Dann gilt für alle Interpretationen $\mathfrak{A}$:

$$\mathfrak{A} \models \varphi_F \iff \begin{array}{l} \text{es existiert ein Tupel } (x_1, \dots, x_n) \in \{0,1\}^n, \\ \text{so dass } F(x_1, \dots, x_n) = 1 \text{ und } \mathfrak{A} \models \chi_{(x_1,\dots,x_n)} \end{array}$$

$$\iff \text{es existiert ein Tupel } (x_1, \dots, x_n) \in \{0,1\}^n, \text{ so dass } F(x_1, \dots, x_n) = 1 \text{ und } \mathfrak{A}(P_i) = x_i \text{ für alle } i \in [n]$$
$$\iff F\big(\mathfrak{A}(P_1), \dots, \mathfrak{A}(P_n)\big) = 1.$$

Also $F_{\varphi_F(P_1,\dots,P_n)} = F$. □

Alternativer Beweis von Satz 1.29.

Man beachte, dass die Formel $\varphi_F(P_1, \dots, P_n)$, die wir im Beweis des Satzes 1.34 konstruieren, in DNF ist.

Um also zu einer gegebenen Formel ψ, oBdA mit $\text{symb}(\psi) \subseteq \{P_1, \dots, P_n\}$, eine äquivalente DNF-Formel zu finden, betrachten wir die Funktion $F := F_{\psi(P_1,\dots,P_n)}$ und konstruieren dazu wie im Beweis des Satzes eine DNF-Formel $\varphi(P_1, \dots, P_n)$ mit $F_{\varphi(P_1,\dots,P_n)} = F$. Dann gilt $\varphi \equiv \psi$. □

Die Konstruktion aus dem Beweis von Satz 1.34 lässt sich am besten anhand einer Funktionstafel (oder Wahrheitstafel) für F erklären.

Beispiel 1.35

Wir betrachten die Funktion $EVEN_3 : \{0,1\}^3 \to \{0,1\}$ (vgl. Beispiel 1.32).

x_1	x_2	x_3	$EVEN_3(x_1, x_2, x_3)$
0	0	0	1
0	0	1	0
0	1	0	0
1	0	0	0
0	1	1	1
1	0	1	1
1	1	0	1
1	1	1	0

$$\begin{aligned}\varphi_{EVEN_3}(P_1, P_2, P_3) :=\; & (\neg P_1 \wedge \neg P_2 \wedge \neg P_3)\\ & \vee (\neg P_1 \wedge P_2 \wedge P_3)\\ & \vee (P_1 \wedge \neg P_2 \wedge P_3)\\ & \vee (P_1 \wedge P_2 \wedge \neg P_3).\end{aligned}$$

Funktional vollständige Junktorenmengen

Definition 1.36

Eine Menge $\mathscr{J} \subseteq \{\wedge, \vee, \rightarrow, \neg, \bot, \top\}$ ist funktional vollständig, wenn es zu jeder Formel $\varphi \in \mathrm{AL}$ eine äquivalente Formel $\varphi' \in \mathrm{AL}$ existiert, die keines der Symbole in $\{\wedge, \vee, \rightarrow, \neg, \bot, \top\} \setminus \mathscr{J}$ verwendet.

Satz 1.37

Die Mengen $\{\vee, \neg\}$, $\{\wedge, \neg\}$, $\{\rightarrow, \neg\}$ *und* $\{\rightarrow, \bot\}$ *sind funktional vollständig.*

Beweis.
$\{\vee, \neg\}$ ist funktional vollständig, weil wir $\{\wedge, \rightarrow, \bot, \top\}$ mit Hilfe von $\vee$ und $\neg$ ausdrücken können:

$$\begin{aligned} \varphi \wedge \psi &\equiv \neg(\neg\varphi \vee \neg\psi), \\ \varphi \rightarrow \psi &\equiv \neg\varphi \vee \psi, \\ \top &\equiv P \vee \neg P && \text{(für beliebige } P \in \sigma) \\ \bot &\equiv \neg\top \equiv \neg(P \vee \neg P). \end{aligned}$$

Um jetzt zu zeigen, dass eine weiter Menge $\mathscr{J}$ funktional vollständig ist, müssen wir noch zeigen, dass sich die Junktoren in $\{\vee, \neg\} \setminus \mathscr{J}$ mit Hilfe der Symbole in $\mathscr{J}$ ausdrücken lassen.
Für $\mathscr{J} = \{\wedge, \neg\}$:

$$\varphi \vee \psi \equiv \neg(\neg\varphi \wedge \neg\psi).$$

Für $\mathscr{J} = \{\rightarrow, \neg\}$:

$$\varphi \vee \psi \equiv \neg\varphi \rightarrow \psi$$

Für $\mathscr{J} = \{\to, \bot\}$:

$$\neg\varphi \equiv \varphi \to \bot,$$
$$\varphi \vee \psi \equiv \neg\varphi \to \psi \equiv (\varphi \to \bot) \to \psi.$$

Weitere Junktoren

Wir können die Aussagenlogik auch um weitere Junktoren erweitern. Dabei ist ein *n-stelliger Junktor* einfach eine n-stellige boolesche Funktion $J : \{0,1\}^n \to \{0,1\}$, für ein $n \in \mathbb{N}$.

In der *Erweiterung der Aussagenlogik um eine Menge $\mathscr{J}$* von Junktoren ist dann für alle n-stelligen $J \in \mathscr{J}$ und alle Formeln $\varphi_1, \dots, \varphi_n$ auch $J(\varphi_1, \dots, \varphi_n)$ eine Formel mit der Semantik

$$[\![J(\varphi_1, \dots, \varphi_n)]\!]^{\mathfrak{A}} := J\Big([\![\varphi_1]\!]^{\mathfrak{A}}, \dots, [\![\varphi_n]\!]^{\mathfrak{A}} \Big).$$

Beispiele 1.38

(1) „Exklusives Oder“: $\oplus : \{0,1\}^2 \to \{0,1\}$ mit

$$\oplus(x_1, x_2) := \begin{cases} 1 & \text{falls } x_1 = 1, x_2 = 0 \text{ oder } x_1 = 0, x_2 = 1, \\ 0 & \text{sonst.} \end{cases}$$

(2) „NAND“ („not and“): $\overline{\wedge} : \{0,1\}^2 \to \{0,1\}$ mit

$$\overline{\wedge}(x_1, x_2) := \begin{cases} 0 & \text{falls } x_1 = 1, x_2 = 1, \\ 1 & \text{sonst.} \end{cases}$$

Ergänzungen zu Seite 1.62

Für zweistellige Junktoren verwenden wir oft Symbole wie $\oplus, \overline{\wedge}$ und verwenden Infix-Notation, d.h., statt $\overline{\wedge}(\varphi, \psi)$ schreiben wir $(\varphi \mathbin{\overline{\wedge}} \psi)$.

Ein funktional vollständiger Junktor

Der Begriff der funktionalen Vollständigkeit lässt sich direkt auf erweiterte Junktorenmengen übertragen.

Satz 1.39

$\{\,\overline{\wedge}\,\}$ *ist funktional vollständig.*

($\overline{\wedge}$ ist der NAND Junktor aus Beispiel 1.38.)

Beweis.
Weil $\{\wedge, \neg\}$ funktional vollständig ist reicht es, $\neg$ und mit Hilfe von $\overline{\wedge}$ und dann $\wedge$ mit Hilfe von $\neg$ und $\overline{\wedge}$ auszudrücken:

$$\neg\varphi \equiv (\varphi \mathbin{\overline{\wedge}} \varphi),$$
$$\varphi \wedge \psi \equiv \neg(\varphi \mathbin{\overline{\wedge}} \psi) \equiv \big((\varphi \mathbin{\overline{\wedge}} \psi) \mathbin{\overline{\wedge}} (\varphi \mathbin{\overline{\wedge}} \psi)\big).$$

$\square$

1.5 Erfüllbarkeitsalgorithmen

Das aussagenlogische Erfüllbarkeitsproblem

In diesem Abschnitt betrachten im Algorithmen für das folgende algorithmische Problem.

Sat

Eingabe: Formel $\varphi \in \mathrm{AL}$

Problem: Berechne eine Interpretation $\mathfrak{A} : \mathrm{symb}(\varphi) \to \{0, 1\}$ mit $\mathfrak{A} \models \varphi$, falls φ erfüllbar, oder gib „unerfüllbar“ aus, falls φ nicht erfüllbar

SAT ist sowohl in der theoretischen Informatik als auch in der Praxis ein extrem wichtiges Problem. Die praktische Relevanz ist darauf zurückzuführen, dass sich zahlreiche kombinatorische Probleme auf einfache Weise als aussagenlogische Erfüllbarkeitsprobleme modellieren lassen. Beispiele haben wir im Abschnitt 1.3 gesehen.

Man beachte, dass wir im erfüllbaren Fall von SAT als Ausgabe eine erfüllende Interpretation und nicht nur die Antwort „erfüllbar" erwarten. Das ist das, was man in der Praxis in der Regel braucht.

In der Vorlesung Berechenbarkeit und Komplexität haben wird hauptsächlich die Entscheidungsvariante des Problems betrachtet, bei der nur entschieden werden muss, ob die Eingabeformel φ erfüllbar ist. Um sie von der Entscheidungsvariante zu unterscheiden, bezeichnen wir unsere Variante manchmal als die Suchvariante.

Wir haben in der Vorlesung Berechenbarkeit und Komplexität gezeigt, dass sich aus einem effizienten Algorithmus für die Entscheidungsvariante auch ein effizienter Algorithmus für die Suchvariante konstruieren lässt.

Wichtige Einschränkungen des Erfüllbarkeitsproblems

KNF-SAT

Eingabe: Formel $\varphi \in$ AL in KNF

Problem: Berechne eine Interpretation $\mathfrak{A} : \text{symb}(\varphi) \to \{0, 1\}$ mit $\mathfrak{A} \models \varphi$, falls φ erfüllbar, oder gib „unerfüllbar“ aus, falls φ nicht erfüllbar

Sei $k \in \mathbb{N}_{>0}$. Eine Formel φ ist in k-KNF, wenn φ in KNF ist und wenn jede disjunktive Klausel von φ genau k Literale enthält.

k-SAT

Eingabe: Formel $\varphi \in$ AL in k-KNF

Problem: Berechne eine Interpretation $\mathfrak{A} : \text{symb}(\varphi) \to \{0, 1\}$ mit $\mathfrak{A} \models \varphi$, falls φ erfüllbar, oder gib „unerfüllbar“ aus, falls φ nicht erfüllbar

Ergänzungen zu Seite 1.66

Achtung

k-KNF ist keine Normalform, d.h., für jedes $k \in \mathbb{N}_{>0}$ gibt es Formeln $\varphi \in \mathrm{AL}$, die zu keiner Formel in k-KNF äquivalent sind.

Satz 1.40 (Satz von Cook und Levin)

Die Entscheidungsvariante von SAT *(und sogar* 3-SAT*) ist NP-vollständig.*

Bemerkung 1.41

Damit ist natürlich auch die Suchvariante NP-schwer.

Ergänzungen zu Seite 1.67

Der Satz wird in der Vorlesung Berechenbarkeit und Komplexität bewiesen.

Stephen Cook (*1939)

Quelle: https://en.wikipedia.org/wiki/Stephen_Cook

Leonid Levin (*1948)

Quelle: https://en.wikipedia.org/wiki/Leonid_Levin

Der Wahrheitstafelalgorithmus

Wahrheitstafelalgorithmus

Eingabe: Formel $\varphi \in \mathrm{AL}$

(1) Berechne die Wahrheitstafel für φ.

(2) Falls in der letzten Spalte mindestens eine 1 auftaucht, gib die Interpretation aus, die der entsprechenden Zeile entspricht, sonst gib „unerfüllbar“ aus.

Die Laufzeit dieses Algorithmus ist exponentiell.

Ergänzungen zu Seite 1.69

Wir berechnen die Wahrheitstafel zeilenweise und brechen bei der ersten 1 in der letzten Spalte ab. Dann können wir bei einer erfüllbaren Formel Glück haben und schnell eine 1 finden. Bei einer unerfüllbaren Formel ist die Laufzeit aber auf jeden Fall mindestens $n2^n$, wobei n die Zahl der Aussagensymbole ist.

Der DNF Algorithmus

DNF-SAT

Eingabe: Formel $\varphi \in \mathrm{AL}$ in DNF

Problem: Berechne eine Interpretation $\mathfrak{A} : \mathrm{symb}(\varphi) \to \{0, 1\}$ mit $\mathfrak{A} \models \varphi$, falls φ erfüllbar, oder gib „unerfüllbar“ aus, falls φ nicht erfüllbar

Lemma 1.42

DNF-SAT *ist in Linearzeit lösbar.*

DNF-Algorithmus

Eingabe: Formel $\varphi \in \mathrm{AL}$

(1) Berechne eine zu φ äquivalente DNF-Formel φ'.

(2) Löse DNF-SAT mit Eingabe φ'.

Im worst-case ist die Laufzeit dieses Algorithmus auch exponentiell, weil es Formeln gibt, deren kleinste äquivalente DNF exponentiell groß ist.

Ergänzungen zu Seite 1.70

Beweis des Lemmas.
Eine konjunktive Klausel ist genau dann erfüllbar, wenn sie für kein Aussagensymbol P sowohl das Literal P als auch das Literal $\neg P$ enthält. Das lässt sich leicht in Linearzeit überprüfen. (Wie? Welche Datenstruktur würden Sie verwenden?) Im erfüllbaren Fall können wir dann auch direkt ein Modell ablesen, nämlich die Interpretation, die alle Literale in der Konjunktion auf 1 setzt.

Eine DNF Formel ist erfüllbar, wenn mindestens eine Ihrer konjunktiven Klauseln erfüllbar ist; ein Modell für diese Klausel ist dann auch ein Modell für die DNF-Formel. Wir gehen einfach der Reihe nach alle Klauseln durch. □

Reduktion auf KNF-SAT

Zwei Formeln φ, ψ sind erfüllbarkeitsäquivalent, wenn

$$\varphi \text{ erfüllbar} \iff \psi \text{ erfüllbar}$$

Beobachtung 1.43

Erfüllbarkeitsäquivalenz ist eine Äquivalenzrelation mit genau zwei Äquivalenzklassen: die Klasse der erfüllbaren Formeln und die Klasse der unerfüllbaren Formeln.

Insbesondere ist jede Formel erfüllbarkeitsäquivalent zu $\perp$ oder zu $\top$.

Lemma 1.44

Es gibt einen Linearzeitalgorithmus, der zu einer gegebenen Formel $\varphi \in \mathsf{AL}$ eine erfüllbarkeitsäquivalente Formel φ' in 3-KNF berechnet.

Als Konsequenz aus diesem Lemma kann man sich darauf konzentrieren, möglichst gute Algorithmen für KNF-SAT zu entwickeln. Das geschieht in der Praxis.

Ergänzungen zu Seite 1.71

Das Lemma wird in der Vorlesung Berechenbarkeit und Komplexität bewiesen.

Der DPLL Algorithmus

Der DPLL-Algorithmus ist ein Erfüllbarkeitsalgorithmus für Formeln in KNF.

Algorithm $\mathrm{DPLL}(\varphi)$

Eingabe: $\varphi \in$ AL in KNF

$(\varphi', \mathfrak{A}') \leftarrow \textsc{Simplify}(\varphi)$ ▷ $\varphi' \in$ AL in KNF mit $\mathrm{symb}(\varphi') \subseteq \mathrm{symb}(\varphi)$, $\mathfrak{A}' : \mathrm{symb}(\varphi) \setminus \mathrm{symb}(\varphi') \to \{0,1\}$, Details siehe nächste Folie

if $\varphi' = \top$ then ▷ leere KNF

  return $\mathfrak{A}'$

else if $\perp$ Klausel von φ' then ▷ leere Klausel

  return „unerfüllbar“

else

  wähle ein Literal $\lambda \in \mathrm{symb}(\varphi')$ ▷ mit geeigneter Heuristik

  $\mathfrak{A} \leftarrow \mathrm{DPLL}(\varphi' \wedge (\lambda))$ ▷ entweder $\mathfrak{A} \models \varphi' \wedge (\lambda)$ oder $\mathfrak{A} =$ „unerfüllbar“

  if $\mathfrak{A} =$ „unerfüllbar“ then

    $\mathfrak{A} \leftarrow \mathrm{DPLL}(\varphi' \wedge (\overline{\lambda}))$ ▷ entweder $\mathfrak{A} \models \varphi' \wedge (\overline{\lambda})$ oder $\mathfrak{A} =$ „unerfüllbar“

    if $\mathfrak{A} =$ „unerfüllbar“ then

      return „unerfüllbar“

  return $\mathfrak{A} \cup \mathfrak{A}'$ ▷ jetzt ist $\mathfrak{A} : \mathrm{symb}(\varphi') \to \{0,1\}$ mit $\mathfrak{A} \models \varphi'$

Ergänzungen zu Seite 1.72

Der DPLL-Algorithmus wurde 1961 von Martin Davis, George Logemann und Donald W. Loveland basierend auf einem früheren Algorithmus von Davis und Hilary Putnam eingeführt.

Die meisten modernen SAT-Solver (Softwaretools zum Lösen des aussagenlogischen Erfüllbarkeitsproblems) basieren auf diesem Algorithmus.

Notation

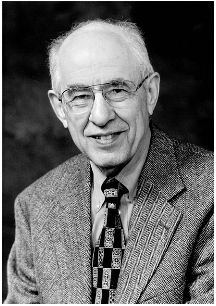
- Für eine Literal λ schreiben wir im Folgenden (λ), um die disjunktive Klausel zu bezeichnen, die nur aus dem einen Literal λ besteht.
  Auch wenn λ und (λ) formal genau dieselbe Formel sind, spielen sie verschiedenen Rollen, deswegen ist die notationelle Unterscheidung sinnvoll.

- Für Funktionen $f : A \to B$ und $g : C \to D$ mit $A \cap C = \emptyset$ ist $f \cup g : A \cup C \to B \cup D$ die Funktion mit $(f \cup g){\restriction}_A = f$ und $(f \cup g){\restriction}_C = g$.

  Also $\mathfrak{A} \cup \mathfrak{A}' : \mathrm{symb}(\varphi) \to \{0,1\}$ mit

$$\mathfrak{A} \cup \mathfrak{A}'(P) = \begin{cases} \mathfrak{A}(P) & \text{falls } P \in \mathrm{symb}(\varphi'), \\ \mathfrak{A}'(P) & \text{falls } P \in \mathrm{symb}(\varphi) \setminus \mathrm{symb}(\varphi'). \end{cases}$$

Martin Davis (1928–2023)
Quelle: https://en.wikipedia.org/wiki/Martin_Davis_(mathematician)



Hilary Putnam (1926–2016)
Quelle: https://en.wikipedia.org/wiki/Hilary_Putnam

DPLL Algorithmus (cont'd)

Wir müssen noch die Vereinfachungssubroutine aus Zeile 1 angeben. Zunächst die Spezifikation des Ein- und Ausgabeverhaltens:

Algorithm $\textsc{Simplify}(\varphi)$

Eingabe: $\varphi \in$ AL in KNF

Ausgabe: $\varphi' \in$ AL in KNF und Interpretation $\mathfrak{A}'$, so dass:

(i) $\text{symb}(\varphi') \subseteq \text{symb}(\varphi)$;

(ii) $\text{def}(\mathfrak{A}') = \text{symb}(\varphi) \setminus \text{symb}(\varphi')$;

(iii) für alle $\mathfrak{A} : \text{symb}(\varphi') \to \{0, 1\}$ gilt:

$$\mathfrak{A} \models \varphi' \quad \iff \quad \mathfrak{A} \cup \mathfrak{A}' \models \varphi;$$

(iv) wenn φ eine Einerklausel enthält, d.h., eine Klausel, die nur aus einem einzelnen Literal λ besteht, so gilt $\text{symb}(\varphi') \subseteq \text{symb}(\varphi) \setminus \text{symb}(\lambda)$.

Ergänzungen zu Seite 1.74

Erfüllt SIMPLIFY diese Spezifikation, so ist leicht zu sehen, dass DPLL korrekt ist. Dabei sorgen (i)–(iii) für die korrekte Antwort.

Bedingung (iv) garantiert, dass der Algorithmus terminiert, denn bei jedem rekursiven Aufruf enthält die Formel eine Einerklausel, und damit reduziert SIMPLIFY die Anzahl der Aussagesymbole.

Wir verzichten auf einen formalen Beweis der Korrektheit.

Simplify

SIMPLIFY(φ) wendet solange wie möglich immer wieder folgende Vereinfachungsregeln an.

Menge der Literale von φ

Pure Literal Rule (PLR)

Wenn es ein Literal $\lambda \in \text{lit}(\varphi)$ $\text{lit}(\varphi)$ gibt, so dass $\overline{\lambda} \notin \text{lit}(\varphi)$, dann entstehe φ' aus φ durch Streichen aller Klauseln, in denen λ vorkommt.

$\mathfrak{A}'$ ist definiert durch $\mathfrak{A}'(\lambda) := 1$ und $\mathfrak{A}'(P) := 0$ (beliebig) für alle $P \in \text{symb}(\varphi) \setminus \big(\text{symb}(\varphi') \cup \text{symb}(\lambda)\big)$.

Unit Propagation Rule (UPR)

Wenn φ eine Einerklausel (λ) enthält, dann entstehe φ' aus φ durch

- Streichen aller Klauseln, in denen λ vorkommt,
- Streichen von $\overline{\lambda}$ aus allen Klauseln, in denen es vorkommt.

$\mathfrak{A}'$ ist definiert durch $\mathfrak{A}'(\lambda) := 1$ und $\mathfrak{A}'(P) = 0$ (beliebig) für alle $P \in \text{symb}(\varphi) \setminus \big(\text{symb}(\varphi') \cup \text{symb}(\lambda)\big)$.

Notation

- $\mathrm{lit}(\varphi)$ bezeichnet die Menge aller Literale, die in φ vorkommen.
- Für eine Literal λ und eine Interpretation $\mathfrak{A}$ bedeutet $\mathfrak{A}(\lambda) = 1$ entweder $\mathfrak{A}(P) = 1$, falls $\lambda = P$, oder $\mathfrak{A}(P) = 0$, falls $\lambda = \neg P$.

Beweis der Korrektheit von Simplify.

Um zu zeigen, dass $\textsc{Simplify}(\varphi)$ korrekt ist, müssen wir zeigen, dass bei Eingabe φ für die Ausgabe $(\varphi', \mathfrak{A}')$ folgendes gilt:

(i) φ' is eine KNF-Formel mit $\mathrm{symb}(\varphi') \subseteq \mathrm{symb}(\varphi)$;

(ii) $\mathfrak{A}' : \mathrm{symb}(\varphi) \setminus \mathrm{symb}(\varphi') \to \{0, 1\}$;

(iii) für alle $\mathfrak{A} : \mathrm{symb}(\varphi') \to \{0, 1\}$

$$\mathfrak{A} \models \varphi' \quad \iff \quad \mathfrak{A} \cup \mathfrak{A}' \models \varphi.$$

(iv) wenn φ eine Einerklausel enthält, so gilt $\operatorname{symb}(\varphi') \subseteq \operatorname{symb}(\varphi) \setminus \operatorname{symb}(\lambda)$.

Enthält φ eine Einerklausel, so ist (UPR) mindestens einmal anwendbar. Das garantiert (iv).

Weil Simplify einfach nur wiederholt die beiden Regeln PLR und UPR anwendet, reicht es zu zeigen, dass eine einzelnen Anwendung einer der Regeln die Bedingungen (i)–(iii) erfüllt.

PLR

Sei λ das Literal, auf das wir die Regel anwenden. Es gilt $\operatorname{symb}(\varphi') \subseteq \operatorname{symb}(\varphi) \setminus \operatorname{symb}(\lambda)$ und

$$\operatorname{def}(\mathfrak{A}') = \operatorname{symb}(\lambda) \cup \Big(\operatorname{symb}(\varphi) \setminus \big(\operatorname{symb}(\varphi') \cup \operatorname{symb}(\lambda)\big)\Big) = \operatorname{symb}(\varphi) \setminus \operatorname{symb}(\varphi'),$$

Bedingungen (i) und (ii) sind also erfüllt.

Um zu sehen, dass Bedingung (iii) erfüllt ist, nehmen wir an, dass

$$\varphi = \bigwedge_{i=1}^{\ell} \psi_i \wedge \bigwedge_{j=1}^{m} \chi_j$$

für disjunktive Klauseln ψ_i, χ_j, so dass alle Klauseln ψ_i das Literal λ nicht enthalten und alle Klauseln χ_j das Literal λ enthalten. Man beachte, dass keine der Klauseln $\overline{\lambda}$ enthält. Dann gilt

$$\varphi' = \bigwedge_{i=1}^{\ell} \psi_i. \tag{$\star$}$$

und

$$\mathfrak{A}' \models \underbrace{\bigwedge_{j=1}^{m} \chi_j}_{=:\varphi''}, \tag{$\star\star$}$$

Sei nun $\mathfrak{A} : \mathrm{symb}(\varphi') \to \{0, 1\}$.

„$\Longrightarrow$“ Gelte $\mathfrak{A} \models \varphi'$. Dann gilt $\mathfrak{A} \cup \mathfrak{A}' \models \varphi'$ und $\mathfrak{A} \cup \mathfrak{A}' \models \varphi''$ wegen $(\star\star)$. Also $\mathfrak{A} \cup \mathfrak{A}' \models \varphi' \wedge \varphi'' = \varphi$.

„$\Longleftarrow$“ Gelte $\mathfrak{A} \cup \mathfrak{A}' \models \varphi = \varphi' \wedge \varphi''$. Dann gilt $\mathfrak{A} \models \varphi'$, weil $\mathrm{symb}(\mathfrak{A}') \cap \mathrm{symb}(\varphi') = \emptyset$.

UPR

Sei λ das Literal, auf das wir die Regel anwenden. Wie bei PLR ist leicht zu sehen, dass (i) und (ii) erfüllt sind.

Um zu zeigen, dass (iii) erfüllt ist, nehmen wir an, dass

$$\varphi = \bigwedge_{i=1}^{\ell} \psi_i \wedge \bigwedge_{j=1}^{m} \chi_j \wedge \bigwedge_{k=1}^{n} \xi_k$$

für disjunktive Klauseln ψ_i, χ_j, ξ_k, so dass alle Klauseln ψ_i weder λ noch $\overline{\lambda}$ enthalten, alle Klauseln χ_j das Literal λ enthalten und alle Klauseln ξ_k das Literal $\overline{\lambda}$, aber nicht λ enthalten. Für alle k sei ξ'_k die disjunktive Klausel, die aus ξ_k entsteht, wenn wir $\overline{\lambda}$ löschen. Dann gilt

$$\varphi' = \bigwedge_{i=1}^{\ell} \psi_i \wedge \bigwedge_{k=1}^{n} \xi'_k \tag{†}$$

und

$$\mathfrak{A}' \models \underbrace{\bigwedge_{j=1}^{m} \chi_j}_{=:\varphi''} . \tag{††}$$

„$\Longrightarrow$" Gelte $\mathfrak{A} \models \varphi'$. Dann gilt

$$\mathfrak{A} \cup \mathfrak{A}' \models \bigwedge_{i=1}^{\ell} \psi_i \wedge \bigwedge_{k=1}^{n} \xi_k ,$$

denn $\mathfrak{A} \models \xi'_k$ und damit $\mathfrak{A} \cup \mathfrak{A}' \models \xi_k \equiv \xi'_k \vee \overline{\lambda}$. Außerdem gilt $\mathfrak{A} \cup \mathfrak{A}' \models \varphi''$ wegen (††). Also $\mathfrak{A} \cup \mathfrak{A}' \models \varphi$.

„$\Longleftarrow$“ Gelte $\mathfrak{A} \cup \mathfrak{A}' \models \varphi$. Weil $\mathfrak{A}' \not\models \overline{\lambda}$ erfüllt $\mathfrak{A}'$ kein Literal in ξ_k, für alle k, also muss $\mathfrak{A} \models \xi'_k$ gelten. Weil $\mathrm{symb}(\mathfrak{A}') \cap \mathrm{symb}(\psi_i) = \emptyset$ für alle j muss $\mathfrak{A} \models \psi_j$ gelten. Also gilt $\mathfrak{A} \models \varphi'$ wegen (†). $\square$

Beispiel 1.45

Wir betrachten folgende KNF-Formel:

$$\begin{aligned}\varphi \coloneqq\, & (P_1 \vee \neg P_5 \vee \neg P_6 \vee P_7) \wedge (\neg P_1 \vee P_2 \vee \neg P_5) \\ & \wedge (\neg P_1 \vee \neg P_2 \vee \neg P_3 \vee \neg P_5 \vee \neg P_6) \wedge (P_1 \vee P_2 \vee \neg P_4 \vee P_7) \\ & \wedge (\neg P_4 \vee \neg P_6 \vee \neg P_7) \wedge (P_3 \vee \neg P_5 \vee P_7) \\ & \wedge (P_3 \vee \neg P_4 \vee \neg P_5) \wedge (P_5 \vee \neg P_6) \\ & \wedge (P_5 \vee P_4 \vee \neg P_8) \wedge (P_1 \vee P_3 \vee P_5 \vee P_6 \vee P_7) \\ & \wedge (\neg P_7 \vee P_8) \wedge (\neg P_6 \vee \neg P_7 \vee \neg P_8).\end{aligned}$$

Wir betrachten einen Lauf von $\mathrm{DPLL}(\varphi)$.

(1) Keine Vereinfachung ist möglich, es gilt $\varphi \neq \top$, und $\bot$ ist keine Klausel von φ.
Wähle (in Zeile 7) das Literal $\lambda := P_6$ und rufe $\mathrm{DPLL}(\varphi \wedge (P_6))$ auf.

(2) UPR mit $\lambda := P_6$ liefert die Formel

$$
\begin{aligned}
\varphi_1 := {} & (P_1 \vee \neg P_5 \vee P_7) \wedge (\neg P_1 \vee P_2 \vee \neg P_5) \\
& \wedge (\neg P_1 \vee \neg P_2 \vee \neg P_3 \vee \neg P_5) \wedge (P_1 \vee P_2 \vee \neg P_4 \vee P_7) \\
& \wedge (\neg P_4 \vee \neg P_7) \wedge (P_3 \vee \neg P_5 \vee P_7) \\
& \wedge (P_3 \vee \neg P_4 \vee \neg P_5) \wedge (P_5) \\
& \wedge (P_5 \vee P_4 \vee \neg P_8) \\
& \wedge (\neg P_7 \vee P_8) \wedge (\neg P_7 \vee \neg P_8).
\end{aligned}
$$

(3) UPR mit $\lambda := P_5$ liefert dann

$$\begin{aligned}\varphi_2 := (P_1 \vee P_7) \wedge (\neg P_1 \vee P_2) \\ \wedge (\neg P_1 \vee \neg P_2 \vee \neg P_3) \wedge (P_1 \vee P_2 \vee \neg P_4 \vee P_7) \\ \wedge (\neg P_4 \vee \neg P_7) \wedge (P_3 \vee P_7) \\ \wedge (P_3 \vee \neg P_4) \\ \wedge (\neg P_7 \vee P_8) \wedge (\neg P_7 \vee \neg P_8).\end{aligned}$$

(4) PLR mit $\lambda := \neg P_4$ liefert

$$\begin{aligned}\varphi_3 := (P_1 \vee P_7) \wedge (\neg P_1 \vee P_2) \\ \wedge (\neg P_1 \vee \neg P_2 \vee \neg P_3) \\ \wedge (P_3 \vee P_7) \\ \wedge (\neg P_7 \vee P_8) \wedge (\neg P_7 \vee \neg P_8).\end{aligned}$$

(5) Jetzt ist keine weitere Vereinfachung möglich, es gilt $\varphi_3 \neq \top$, und $\bot$ ist keine Klausel von φ_3. Wähle das Literal $\lambda := P_7$ (in Zeile 7 von $\mathrm{DPLL}(\varphi \wedge (P_6)))$ und rufe $\mathrm{DPLL}(\varphi_3 \wedge (P_7))$ auf (in Zeile 8).

(6) UPR mit $\lambda := P_7$ liefert

$$\varphi_4 := (\neg P_1 \vee P_2) \wedge (\neg P_1 \vee \neg P_2 \vee \neg P_3) \wedge (P_8) \wedge (\neg P_8).$$

(7) UPR mit $\lambda := P_8$ liefert

$$\varphi_5 := (\neg P_1 \vee P_2) \wedge (\neg P_1 \vee \neg P_2 \vee \neg P_3) \wedge \bot.$$

Also gibt $\mathrm{DPLL}(\varphi_3 \wedge P_7)$ (in Zeile 5) den Wert „unerfüllbar“ zurück.
Daher:

(8) Backtracking, zurück zu Schritt (5):
Rufe $\mathrm{DPLL}(\varphi_3 \wedge (\neg P_7))$ auf (in Zeile 10 von $\mathrm{DPLL}(\varphi \wedge (P_6)))$.

(9) UPR mit $\lambda := \neg P_7$ liefert

$$\varphi_6 := (P_1) \wedge (\neg P_1 \vee P_2) \wedge (\neg P_1 \vee \neg P_2 \vee \neg P_3) \wedge (P_3).$$

(10) UPR mit $\lambda := P_1$ liefert

$$\varphi_7 := (P_2) \wedge (\neg P_2 \vee \neg P_3) \wedge (P_3).$$

(11) UPR mit $\lambda := P_2$ liefert

$$\varphi_8 := (\neg P_3) \wedge (P_3).$$

(12) UPR mit $\lambda := P_3$ liefert

$$\varphi_9 := \bot.$$

Also gibt $\mathrm{DPLL}(\varphi_3 \wedge (\neg P_7))$ (in Zeile 5) den Wert „unerfüllbar“ zurück.
Daher:

(13) Backtracking, zurück zu Schritt (5):
$\mathrm{DPLL}(\varphi \wedge P_6)$ gibt (in Zeile 12) „unerfüllbar“ zurück.
Daher:

(14) Backtracking, zurück zu Schritt (1):
Rufe $\mathrm{DPLL}(\varphi \wedge (\neg P_6))$ auf (in Zeile 10 von $\mathrm{DPLL}(\varphi)$).

(15) UPR mit $\lambda := \neg P_6$ liefert

$$
\begin{aligned}
\varphi_{10} := & (\neg P_1 \vee P_2 \vee \neg P_5) \\
& \wedge (P_1 \vee P_2 \vee \neg P_4 \vee P_7) \\
& \wedge (P_3 \vee \neg P_5 \vee P_7) \\
& \wedge (P_3 \vee \neg P_4 \vee \neg P_5) \\
& \wedge (P_5 \vee P_4 \vee \neg P_8) \wedge (P_1 \vee P_3 \vee P_5 \vee P_7) \\
& \wedge (\neg P_7 \vee P_8).
\end{aligned}
$$

(16) PLR mit $\lambda := P_2$ liefert

$$\begin{aligned}\varphi_{11} := (&P_3 \lor \neg P_5 \lor P_7) \\ &\land (P_3 \lor \neg P_4 \lor \neg P_5) \\ &\land (P_5 \lor P_4 \lor \neg P_8) \land (P_1 \lor P_3 \lor P_5 \lor P_7) \\ &\land (\neg P_7 \lor P_8).\end{aligned}$$

(17) PLR mit $\lambda := P_3$ liefert

$$\varphi_{12} := (P_5 \lor P_4 \lor \neg P_8) \land (\neg P_7 \lor P_8).$$

(18) PLR mit $\lambda := P_4$ liefert

$$\varphi_{13} := (\neg P_7 \lor P_8).$$

(19) PLR mit $\lambda := \neg P_7$ liefert

$$\varphi_{13} := \top.$$

Verfolgen wir die Schritte zurück, so sehen wir, dass der Aufruf $\textsc{Simplify}(\varphi \wedge (\neg P_6))$ das Paar $(\varphi', \mathfrak{A}')$ mit $\varphi' = \varphi_{13} = \top$ und folgendem $\mathfrak{A}'$ zurückgibt:

P	$\mathfrak{A}'(P)$	
P_6	0	Schritt (15)
P_2	1	Schritt (16)
P_1	0	
P_3	1	Schritt (17)
P_4	1	Schritt (18)
P_5	0	
P_7	0	Schritt (19)
P_8	0	

Weil $\varphi' = \top$ gibt $\mathrm{DPLL}(\varphi)$ in Zeile 3 die Interpretation $\mathfrak{A}'$ zurück.

Effizienz

- SIMPLIFY ist sehr effizient, bei cleverer Implementierung läuft es in Linearzeit. Das ist wichtig, denn bei einem Lauf von DPLL wird SIMPLIFY typischerweise sehr häufig aufgerufen.
- Im worst-case benötigt DPLL eine exponentielle Laufzeit. Oft läuft er aber deutlich schneller.
- Praktische SAT-Solver basieren auf DPLL, modifizieren und erweitern den Algorithmus aber noch auf vielfältige Weise.

Hornklauseln und Hornformeln

Hornformeln sind spezielle aussagenlogische Formeln, die die Basis der logischen Programmierung bilden, und für die das Erfüllbarkeitsproblem effizient gelöst werden kann.

Definition 1.46

(1) Eine Hornklausel ist eine disjunktive Klausel, in der höchstens ein positives Literal vorkommt.

(2) Eine Hornformel ist eine Konjunktion endlich vieler Hornklauseln.

Beispiele 1.47

- $(\neg P \vee \neg Q \vee \neg R)$ ist eine Hornklausel.
- $(\neg P \vee \neg Q \vee R)$ ist eine Hornklausel.
- $(\neg P \vee Q \vee R)$ ist *keine* Hornklausel.
- (P) ist eine Hornklausel.
- $\perp$ (die leere disjunktive Klausel) ist eine Hornklausel.
- $(P \vee \neg Q) \wedge (\neg R \vee \neg P \vee \neg Q) \wedge (Q)$ ist eine Hornformel.

Hornklauseln als Implikationen

Beobachtung 1.48

Eine disjunktive Klausel

$$\neg P_1 \vee \ldots \vee \neg P_k \vee Q_1 \vee \ldots \vee Q_\ell$$

ist äquivalent zur Implikation

$$P_1 \wedge \ldots \wedge P_k \to Q_1 \vee \ldots \vee Q_\ell$$

(möglicherweise gilt hier $k = 0$ oder $\ell = 0$).

Beobachtung 1.49

Jede Hornklausel ist äquivalent zu einer Implikation folgender Gestalt:

$$\begin{aligned} P_1 \wedge \ldots \wedge P_k &\to Q && \text{oder} \\ P_1 \wedge \ldots \wedge P_k &\to \bot && \text{oder} \\ \top &\to Q && \text{oder} \\ \top &\to \bot. \end{aligned}$$

Der Streichungsalgorithmus

Algorithm $\textsc{HornSat}(\varphi)$

Eingabe: Hornformel $\varphi \in$ AL

$\mathfrak{A}(P) := 0$ for all $P \in \text{symb}(\varphi)$ ▷ Initialisiere Interpretation $\mathfrak{A}$

Repeat

  if $\perp$ Klausel von φ then ▷ leere Klausel

    return „unerfüllbar“

  else if jede Klausel von φ hat ein negatives Literal then

    return $\mathfrak{A}$

  else ▷ φ hat ein Klausel, die nur aus eine positiven Literal besteht

    wähle Klausel (P)

    $\mathfrak{A}(P) \leftarrow 1$

    Streiche alle Klauseln, die das Literal P enthalten

    Streiche das Literal $\neg P$ aus allen Klauseln, die es enthalten

▷ Jetzt kommt P nicht mehr in φ vor

Satz 1.50

Für jede Hornformel φ mit n Aussagensymbolen terminiert $\textsc{HornSat}(\varphi)$ nach höchstens n Iterationen der Schleife und gibt ein Modell von φ aus, falls φ erfüllbar ist, oder „unerfüllbar“, falls φ unerfüllbar ist.

Bemerkungen 1.51

- Weil sich ein einzelner Durchlauf der Schleife von $\textsc{HornSat}$ offensichtlich in Polynomialzeit durchführen lässt, läuft der Algorithmus auf jeden Fall in Polynomialzeit.

  Bei geschickter Implementierung läuft er sogar in Linearzeit.
- Der Streichungsalgorithmus ist im Kern der gleiche Algorithmus wie der Markierungsalgorithmus für das Leerheitsproblem kontextfreier Sprachen (siehe Vorlesung Formale Sprachen, Automaten, und Prozesse).

Beweis des Satzes.

HornSat(φ) terminiert nach höchstens n Iterationen, weil bei jeder Iteration mindestens ein Aussagensymbol komplett aus der Formel gestrichen wird.

Für die Korrektheit beachte man zunächst, dass der Algorithmus korrekt terminiert:

- Falls $\perp$ Klausel von φ, so ist φ unerfüllbar.
- Falls jede Klausel von φ ein negatives Literal enthält, so erfüllt die Interpretation, die alle Symbole auf 0 setzt, alle Klauseln.

Falls HornSat noch nicht terminiert, wendet der Algorithmus einmal die Unit Propagation Rule von Simplify an. Die Korrektheit folgt dann aus der Korrektheit dieser Regel. $\square$

Beispiel 1.52

Wir wenden HornSat auf folgende Hornformel an

$$\varphi = (\neg P_2 \vee \neg P_1 \vee P_4) \wedge (\neg P_5 \vee \neg P_4 \vee \neg P_2 \vee P_3) \wedge (\neg P_5 \vee \neg P_4 \vee P_2) \wedge (P_5) \wedge (\neg P_3) \wedge (P_4).$$

Wir betrachten einen Lauf von HornSat(φ).

(1) Initialisierung auf

P	P_1	P_2	P_3	P_4	P_5
$\mathfrak{A}(P)$	0	0	0	0	0

(2) Wähle (in Zeile 9) $P = P_4$. Nach Ausführung der Zeilen 10-12 ergibt sich

P	P_1	P_2	P_3	P_4	P_5
$\mathfrak{A}(P)$	0	0	0	1	0

und

$$\varphi = (\neg P_5 \vee \neg P_2 \vee P_3) \wedge (\neg P_5 \vee P_2) \wedge (P_5) \wedge (\neg P_3).$$

(3) Wähle (in Zeile 9) $P = P_5$. Nach Ausführung der Zeilen 10-12 ergibt sich

P	P_1	P_2	P_3	P_4	P_5
$\mathfrak{A}(P)$	0	0	0	1	1

und

$$\varphi = (\neg P_2 \vee P_3) \wedge (P_2) \wedge (\neg P_3).$$

(4) Wähle (in Zeile 9) $P = P_2$. Nach Ausführung der Zeilen 10-12 ergibt sich

P	P_1	P_2	P_3	P_4	P_5
$\mathfrak{A}(P)$	0	1	0	1	1

und

$$\varphi = (P_3) \wedge (\neg P_3).$$

(5) Wähle (in Zeile 9) $P = P_3$. Nach Ausführung der Zeilen 10-12 ergibt sich

P	P_1	P_2	P_3	P_4	P_5
$\mathfrak{A}(P)$	0	1	1	1	1

und

$$\varphi = \perp.$$

(6) beim nächsten Durchlauf der Schleife gibt der Algorithmus (in Zeile 4) „unerfüllbar“ zurück.