Logik

Vorlesung im Wintersemester 2014/2015

Stephan Kreutzer
Technische Universität Berlin

Vorläufige Version vom 9.10.2014

Inhaltsverzeichnis

Teil I.

Aussagenlogik

2. Grundlagen der Aussagenlogik

2.1. Einleitung

Als Einführung in die Aussagenlogik betrachten wir folgendes Beispiel.

Beispiel 2.1 Betrachten wir folgende Aussage.

- *Wenn der Zug zu spät ist und keine Taxis am Bahnhof stehen, kommt Peter zu spät zu seiner Verabredung.*
- *Peter kam nicht zu spät zu seiner Verabredung.*
- *Der Zug hatte Verspätung.*

Also standen Taxis am Bahnhof.

Ist das Argument *gültig*? Wie wir später sehen werden, ist das Argument in der Tat korrekt. Aber warum? Und wie können wir solche Argumente möglichst einfach formal beweisen? ⊣

Betrachten wir als Nächstes folgendes Beispiel.

Beispiel 2.2 *Sie kann nicht zu hause sein, da sie an Bord oder zu hause ist und ich gerade gehört habe, dass sie an Bord ist.*

Ist das Argument *gültig*? Das können wir nicht so genau feststellen, da in der Aussage Kontextwissen verwendet wird, das nicht klar bestimmt ist. Zum Beispiel ist nicht ganz klar, ob das zu Hause auch an Bord sein kann. ⊣

Das letzte Beispiel wirft die Frage auf, was wir überhaupt formal beweisen können. Grundsätzlich können wir nur klar formulierte Aussagen beweisen, die entweder richtig oder falsch sind. Dabei muss die Bedeutung aller verwendeten Ausdrücke bekannt sein. Ebenso das vorausgesetzte Hintergrundwissen.

Für solche Aufgaben ist die natürliche Sprache nicht gut geeignet, da sie viele Doppeldeutigkeiten enthält und üblicherweise viel Hintergrundwissen vorausgesetzt wird.

Wir werden daher formale Sprachen, oder Logiken, verwenden, in denen alle Ausdrücke formal und vollständig definiert sind. Das Ziel ist es, allgemeine Regeln für korrektes Schließen herleiten zu können und möglichst automatisch logische Folgerungen beweisen zu können.

Wir betrachten ein weiteres Beispiel.

Beispiel 2.3 Die Korrektheit folgender Aussagen folgt aus rein formalen Gründen.

Annahme 1:	**Alle** Menschen **sind** sterblich.
Annahme 2:	Sokrates **ist ein** Mensch.
Folgerung:	**Also ist** Sokrates sterblich.

Diese und verwandte Arten von Schlüssen werden *Syllogismen* genannt. Abstrakt hat der Schluss die folgende Form:

Annahme 1:	**Alle** A **sind** B.
Annahme 2:	C **ist ein** A.
Folgerung:	**Also ist** C B.

Wenn wir die Korrektheit dieser Schlussweise einmal nachgewiesen haben, können wir also sofort folgende Folgerung herleiten, ohne die darin verwendeten Begriffe kennen zu müssen.

Annahme 1:	**Alle** Ersetzungschiffren **sind** anfällig für brute-force Angriffe.
Annahme 2:	Der Caesar Chiffre **ist ein** Ersetzungschiffre.
Folgerung:	**Also ist** der Caesar Chiffre anfällig für brute-force Angriffe.

⊣

In diesem Teil der Vorlesung werden wir Methoden kennen lernen um

- Aussagen wie in den vorherigen Beispielen formal auszudrücken,
- formalisierte Aussagen zu manipulieren und
- logische Behauptungen zu beweisen.

2.2. Syntax und Semantik

In diesem Kapitel führen wir die *Syntax* und *Semantik* der Aussagenlogik ein. Generell sind Logiken formale Sprachen, d.h. Mengen oder Klassen von Wörtern über einem bestimmten Alphabet, zusammen mit einer Vorschrift, was

die einzelnen Wörter (genannt Formeln) eigentlich bedeuten sollen. Unter der *Syntax* einer Logik versteht man dabei die Festlegung, welche Wörter korrekte Formeln der Logik sind. Die *Semantik* legt die Bedeutung der einzelnen Formeln fest.

2.2.1. Syntax der Aussagenlogik

Die Aussagenlogik (engl. propositional logic) basiert auf dem Begriff der Propositionen. *Propositionen* sind Aussagen die entweder *wahr* oder *falsch* sein können. Beispiele für Propositionen sind A : „Die Sonne scheint“ oder B : „Die Vorlesung ist langweilig“.

Propositionen können durch *Verknüpfungen* kombiniert werden, z. B. durch *nicht*, *und*, *oder* oder *wenn ... dann ...*. Wir können zum Beispiel die verknüpfte Aussage A *und nicht* B bilden. Die *Aussagenlogik* studiert grundlegende Prinzipien korrekten Schließens mit Propositionen und deren Kombinationen.

Definition 2.4 (Aussagenvariablen) Eine *Aussagenvariable*, oder auch einfach *Variable*, hat die Form V_i für $i \in \mathbb{N}$. Die Menge aller Variablen bezeichnen wir als AVar. ⊣

Definition 2.5 (Syntax der Aussagenlogik) Das *Alphabet* der Aussagenlogik ist

$$\Sigma_{\mathrm{AL}} := \mathrm{AVar} \cup \{\top, \bot, \neg, \vee, \wedge, \rightarrow, \leftrightarrow, (,)\}.$$

Die Klasse AL der *aussagenlogischen Formeln* ist induktiv definiert durch:

- $\top, \bot$ sind aussagenlogische Formeln.
- Jede Variable $V_i \in \mathrm{AVar}$ ist eine aussagenlogische Formel.
- Wenn $\varphi \in \mathrm{AL}$ eine Formel ist, dann auch $\neg\varphi \in \mathrm{AL}$.
- Wenn $\varphi, \psi \in \mathrm{AL}$ Formeln sind, dann auch

$$(\varphi \vee \psi), \quad (\varphi \wedge \psi), \quad (\varphi \rightarrow \psi), \quad (\varphi \leftrightarrow \psi).$$

$\top$, $\bot$ und die Variablen werden *atomare Formeln* oder *Atome* genannt. $\neg$, $\vee$, $\wedge$, $\rightarrow$, $\leftrightarrow$ werden *aussagenlogische Verknüpfungen* genannt. ⊣

Beispiel 2.6 Die folgenden Ausdrücke sind Beispiele für korrekte aussagenlogische Formeln.

- $\varphi_1 := (V_0 \vee V_1)$

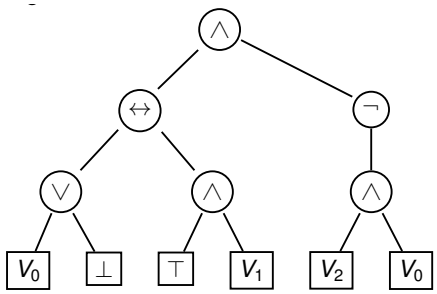


Abbildung 2.1.: Syntaxbaum der Formel $(((V_0 \vee \bot) \leftrightarrow (\top \wedge V_1)) \wedge \neg(V_2 \wedge V_0))$

- $\varphi_2 := (((V_0 \vee \bot) \leftrightarrow (\top \wedge V_1)) \wedge \neg(V_2 \wedge V_0))$

Die folgenden Ausdrücke sind hingegen keine korrekten Formeln.

- $\varphi_3 := V_0 \vee V_1$ (da die äußeren Klammern fehlen)
- $\varphi_4 := (V_1 \wedge V_2 \vee V_3)$ (wiederum fehlen Klammern, entweder um $(V_0 \vee V_1)$ oder $(V_1 \wedge V_2)$)
- $\varphi_5 := (V_0 \leftarrow V_1)$ (da das Symbol $\leftarrow$ nicht Teil des Alphabets ist.) ⊣

Die Struktur einer Formel kann elegant durch ihren *Syntax-* oder *Ableitungsbaum* dargestellt werden. Abbildung 2.2.1 zeigt den Syntaxbaum der Formel

$$\varphi := (((V_0 \vee \bot) \leftrightarrow (\top \wedge V_1)) \wedge \neg(V_2 \wedge V_0)).$$

Zur besseren Lesbarkeit von Formeln werden wir im weiteren auch $X, Y, Z, \ldots$ als Symbole für Variablen verwenden.

Definition 2.7 (Unterformeln) Die Menge $\mathrm{sub}(\varphi)$ der *Unterformeln* einer Formel φ ist induktiv wie folgt definiert:

- Ist φ atomar, dann ist $\mathrm{sub}(\varphi) := \{\varphi\}$.
- Ist $\varphi := \neg\psi$, dann ist $\mathrm{sub}(\varphi) := \{\varphi\} \cup \mathrm{sub}(\psi)$.
- Für alle $* \in \{\wedge, \vee, \rightarrow, \leftrightarrow\}$: Ist $\varphi := (\varphi_1 * \varphi_2)$, dann ist

$$\mathrm{sub}(\varphi) := \{\varphi\} \cup \mathrm{sub}(\varphi_1) \cup \mathrm{sub}(\varphi_2).$$

Wir fassen sub als Funktion $\mathrm{sub} : \mathrm{AL} \to \mathcal{P}(\mathrm{AL})$ auf, die jeder Formel φ die Menge $\mathrm{sub}(\varphi)$ ihrer Unterformeln zuweist. ⊣

Beispiel 2.8 Sei $\varphi := (((((T \wedge \neg C) \to L) \wedge \neg L) \wedge T) \to C)$.
Dann ist

$$\begin{aligned}\mathrm{sub}(\varphi) := \{\ & (((((T \wedge \neg C) \to L) \wedge \neg L) \wedge T) \to C),\\ & (((T \wedge \neg C) \to L) \wedge \neg L) \wedge T),\\ & C,\\ & ((T \wedge \neg C) \to L) \wedge \neg L),\\ & ((T \wedge \neg C) \to L),\\ & \neg L,\\ & L,\\ & (T \wedge \neg C),\\ & T,\\ & \neg C\ \}.\end{aligned}$$

⊣

Die Definition der Menge der Unterformeln ist ein Beispiel für eine sehr elegante Methode, Funktionen über Formeln zu definieren oder Aussagen über Formeln zu beweisen.

Induktive Definitionen. Eine Funktion $f : \mathrm{AL} \to M$, für eine beliebige Menge M, kann induktiv wie folgt definiert werden:

Basisfälle. Wir definieren zunächst die Funktionswerte für atomare Formeln.

- Definiere $f(\top)$ und $f(\bot)$.
- Definiere $f(X)$ für alle $X \in \mathrm{AVar}$.

Induktionsschritt. Danach werden die Funktionswerte für zusammengesetzte Formeln definiert.

- Definiere $f(\neg\varphi)$ aus $f(\varphi)$.
- Für $* \in \{\wedge, \vee, \to, \leftrightarrow\}$ definiere $f((\varphi\ *\ \psi))$ aus $f(\varphi)$ und $f(\psi)$.

Beispiel 2.9 Wir wollen eine Funktion $f : \mathrm{AL} \to \mathbb{N}$ definieren, so dass $f(\varphi)$ die „Größe“ der Formel φ angibt.

Basisfälle:

- Definiere $f(\top) = f(\bot) := 1$.
- Definiere $f(X) := 1$ für alle $X \in \mathrm{AVar}$.

Induktionsschritt:

- Definiere $f(\neg\varphi) := 1 + f(\varphi)$.
- Für $* \in \{\wedge, \vee, \rightarrow, \leftrightarrow\}$ definiere $f\big((\varphi * \psi)\big) := 1 + f(\varphi) + f(\psi)$. ⊣

2.2.2. Semantik der Aussagenlogik

Definition 2.10 Die Menge $\mathrm{var}(\varphi)$ der *Variablen einer Formel* φ ist die Menge

$$\mathrm{var}(\varphi) := \mathrm{AVAR} \cap \mathrm{sub}(\varphi).$$ ⊣

Definition 2.11 (1) Eine *Wahrheitsbelegung*, oder kurz *Belegung*, ist eine partielle Funktion

$$\beta : \mathrm{AVAR} \to \{0, 1\}.$$

(2) Eine Belegung β ist eine *Belegung für* eine Formel φ, oder ist *passend zu* φ, wenn $\mathrm{var}(\varphi) \subseteq \mathrm{def}(\beta)$. ⊣

Intuitiv steht in der vorherigen Definition der Wert 1 für *wahr* und 0 für *falsch*.

Definition 2.12 Per Induktion über die Struktur der Formeln in AL definieren wir eine Funktion $[\![\cdot]\!]^{\cdot}$, die jeder Formel $\varphi \in \mathrm{AL}$ und jeder zu φ passenden Belegung β einen *Wahrheitswert* $[\![\varphi]\!]^\beta \in \{0, 1\}$ zuordnet.

Basisfall.

- $[\![\bot]\!]^\beta := 0 \qquad [\![\top]\!]^\beta := 1$
- Für alle $X \in \mathrm{AVAR}$ gilt $[\![X]\!]^\beta := \beta(X)$

Induktionsschritt. Für zusammen gesetzte Formeln φ definieren wir $[\![\varphi]\!]^\beta$ durch die folgenden Wahrheitstafeln:

$[\![\varphi]\!]^\beta$	$[\![\neg\varphi]\!]^\beta$
0	1
1	0

Konjunktion

$[\![\varphi]\!]^\beta$	$[\![\psi]\!]^\beta$	$[\![(\varphi \wedge \psi)]\!]^\beta$
0	0	0
0	1	0
1	0	0
1	1	1

Disjunktion

$[\![\varphi]\!]^\beta$	$[\![\psi]\!]^\beta$	$[\![(\varphi \vee \psi)]\!]^\beta$
0	0	0
0	1	1
1	0	1
1	1	1

Implikation

$[\![\varphi]\!]^\beta$	$[\![\psi]\!]^\beta$	$[\![(\varphi \to \psi)]\!]^\beta$
0	0	1
0	1	1
1	0	0
1	1	1

Biimplikation

$[\![\varphi]\!]^\beta$	$[\![\psi]\!]^\beta$	$[\![(\varphi \leftrightarrow \psi)]\!]^\beta$
0	0	1
0	1	0
1	0	0
1	1	1

Das Symbol $\neg$ steht also für „nicht“ oder die „Negation“. Das Symbol $\wedge$ steht für “und” oder die Konjunktion, $\vee$ für „oder“ oder Disjunktion und $\to$ für die Implikation, „wenn ..., dann ...“. Schließlich steht $\leftrightarrow$ für die Biimplikation, „... genau dann, wenn ...“. ⊣

Bemerkung 2.13 Die logische Implikation $\to$ stimmt nicht immer mit der umgangssprachlichen Verwendung der Implikation überein. Zum Beispiel wird keine *Kausalität* impliziert.

$\varphi \to \psi$ heißt einfach, dass wann immer φ wahr ist, auch ψ wahr sein muss. Insbesondere ist $\varphi \to \psi$ als Aussage immer wahr, wenn φ falsch ist. ⊣

Unsere Wahl der Verknüpfungen $\neg$, $\wedge$, $\vee$, $\to$, $\leftrightarrow$ für die Aussagenlogik spiegelt unsere Verwendung in der natürlichen Sprache wieder, ist aber zu gewissem Grad willkürlich. Durch die Definition einer Wahrheitstafel können wir auch andere Verknüpfungen und somit auch andere „Aussagenlogiken“ definieren.

Wir könnten zum Beispiel das *Exklusive Oder* $\varphi \oplus \psi$ durch die Wahrheitstafel

Exklusives Oder

$[\![\varphi]\!]^\beta$	$[\![\psi]\!]^\beta$	$[\![\varphi \oplus \psi]\!]^\beta$
0	0	0
0	1	1
1	0	1
1	1	0

definieren und dann in unserer Aussagenlogik verwenden. Intuitiv bedeutet $\varphi \oplus \psi$ *„entweder φ oder ψ“*.

Notation 2.14 Zur besseren Lesbarkeit von Formeln vereinbaren wir folgende Notation. Wir bezeichnen

- Belegungen meistens mit $\beta, \gamma, \ldots$
- Formeln meistens mit $\varphi, \psi, \varphi', \ldots$
- Mengen von Formeln mit entsprechenden Großbuchstaben $\Phi, \Psi, \ldots$
- Variablen meistens mit $X, Y, \ldots, V_0, V_1, \ldots$

Um unnötige Klammern zu vermeiden,

- lassen wir die äußersten Klammern von Formeln weg,
- vereinbaren wir, dass $\neg$ stärker bindet als die anderen Verknüpfungen, und dass
- $\wedge$, $\vee$ stärker binden als $\rightarrow$, $\leftrightarrow$.

Wir schreiben also $\neg X \wedge Y \rightarrow T$ für $((\neg X \wedge Y) \rightarrow T)$. Aber wir können nicht $X \wedge Y \vee Z$ schreiben. ⊣

2.3. Formalisierung natürlichsprachlicher Aussagen

Wir betrachten nun einige Beispiele von Aussagen in natürlicher Sprache und deren Formalisierung in der Aussagenlogik.

Beispiel 2.15 Betrachten wir die Aussage *„Wenn es kalt ist und regnet, gehe ich nicht spazieren“*.

Die darin enthaltenen elementaren, oder atomaren, Aussagen sind

- X_k: es ist kalt
- X_r: es regnet
- X_s: ich gehe spazieren

Obige Aussage kann also wie folgt formalisiert werden

$$X_k \wedge X_r \rightarrow \neg X_s$$ ⊣

Das folgende Beispiel ist nicht rein „aussagenlogisch“. Eine Formalisierung in der Aussagenlogik kann dennoch nützlich sein.

Beispiel 2.16 Wir betrachten noch einmal die Syllogismen aus Beispiel 2.3.

Annahme 1:	**Alle** Menschen **sind** sterblich.
Annahme 2:	Sokrates **ist ein** Mensch.
Folgerung:	**Also ist** Sokrates sterblich.

Die darin enthaltenen atomaren Aussagen sind

- X_M: x ist ein Mensch
- X_S: x ist sterblich
- X_K: x ist Sokrates

Insgesamt wird die Aussage also durch

$$\big((X_M \to X_S) \wedge (X_K \to X_M)\big) \to (X_K \to X_S)$$

formalisiert. Dies vernachlässigt aber, dass die Aussage für alle Menschen gelten soll. Wir werden daher im dritten Teil der Vorlesung die *Prädikatenlogik* kennen lernen, in der auch dieser Aspekt formalisiert werden kann. ⊣

Als letztes Beispiel betrachten wir noch einmal Beispiel 2.1.

- Wenn der Zug zu spät ist und keine Taxis am Bahnhof stehen, kommt Peter zu spät zu seiner Verabredung.
- Peter kam nicht zu spät zu seiner Verabredung.
- Der Zug hatte Verspätung.

Also standen Taxis am Bahnhof.

Als Aussagenvariablen wählen wir

T:	der Zug war zu spät
C:	es standen Taxis am Bahnhof
L:	Peter kam zu spät zu seiner Verabredung.

Die Aussage wird dann durch

$$\varphi := \big((T \wedge \neg C \to L) \wedge \neg L\big) \wedge T \to C$$

formalisiert. Jede zu φ passende Belegung entspricht nun einem möglichen „Sachverhalt“, z.B. die Belegung β, die T mit 1, C mit 0 und L mit 1 belegt entspricht dem Sachverhalt, dass der Zug zu spät war, es keine Taxis am Bahnhof gab und Peter zu spät zu seiner Verabredung kam.

Wenn wir also wissen wollen, ob die Schlussfolgerung oben korrekt ist, dann muss die Formel φ unter *allen* Sachverhalten, d.h. unter allen Belegungen, gelten. Dies führt zum Begriff der *Allgemeingültigkeit* bzw. der *Erfüllbarkeit* von Formeln, den wir im nächsten Abschnitt behandeln.

2.4. Erfüllbarkeit und Allgemeingültigkeit

Definition 2.17 Sei $\varphi \in \mathrm{AL}$ eine Formel.

(1) Eine zu φ passende Belegung β *erfüllt* φ, oder ist ein *Modell* von φ, wenn $[\![\varphi]\!]^\beta = 1$ (vgl. Def. 2.12). Wir schreiben $\beta \models \varphi$.

(2) φ ist *erfüllbar*, wenn es eine Belegung β gibt, die φ erfüllt. Anderenfalls ist φ *unerfüllbar*.

(3) φ ist *allgemeingültig*, oder eine *Tautologie*, wenn jede zu φ passende Belegung φ erfüllt. ⊣

Beispiel 2.18 (1) Die Formel $(X \to Y)$ ist erfüllbar aber nicht allgemeingültig.

(2) Die Formel $(X \wedge \neg X)$ ist unerfüllbar.

(3) Die Formel $(X \wedge \neg X \to Y)$ ist allgemeingültig. ⊣

Aus der Definition folgt direkt folgende Proposition.

Proposition 2.19 *Eine Formel $\varphi \in \mathrm{AL}$ is genau dann allgemeingültig, wenn $\neg\varphi$ unerfüllbar ist.*

Wie oben schon besprochen ist also die durch $\varphi := ((T \wedge \neg C \to L) \wedge \neg L) \wedge T \to C$ formalisierte Aussage über Peter und seine Verabredung genau dann korrekt, wenn φ allgemeingültig ist. Wir brauchen also Methoden, um Allgemeingültigkeit und Erfüllbarkeit von Formeln zu entscheiden. In dieser Vorlesung werden wir drei Methoden kennen lernen

- *Wahrheitstafeln*
- *Resolution*
- *Sequenzenkalkül*

2.4.1. Wahrheitstafeln

Wir können die Wahrheitswerte aussagenlogischer Formeln $\varphi \in \mathrm{AL}$ unter allen möglichen Belegungen in einer *Wahrheitstafel* darstellen.

Für jede mögliche Belegung $\beta : \mathrm{var}(\varphi) \to \{0,1\}$ hat die Tafel eine Zeile, die den Wahrheitswert $\beta(X)$ für alle $X \in \mathrm{var}(\varphi)$ sowie $[\![\varphi]\!]^\beta$ enthält.

Sei zum Beispiel $\varphi := \big((X_H \to X_M) \wedge (X_S \to X_H)\big) \to (X_S \to X_M)$. Die dazugehörige Wahrheitstafel ist

X_H	X_M	X_S	$\big((X_H \to X_M) \wedge (X_S \to X_H)\big) \to (X_S \to X_M)$
0	0	0	1
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	1

Es ist oft nützlich, die Wahrheitswerte der Unterformeln ebenfalls in der Wahrheitstafel aufzuführen, wie in der folgenden Tabelle.

X_H	X_M	X_S	$(X_H \to X_M)$	$(X_S \to X_H)$	$(X_H \to X_M) \wedge$ $(X_S \to X_H)$	$(X_S \to X_M)$	φ
0	0	0	1	1	1	1	1
0	0	1	1	0	0	0	1
0	1	0	1	1	1	1	1
0	1	1	1	0	0	1	1
1	0	0	0	1	0	1	1
1	0	1	0	1	0	0	1
1	1	0	1	1	1	1	1
1	1	1	1	1	1	1	1

Dies wird als die *erweiterte Wahrheitstafel* bezeichnet.

Im Spezialfall, dass die Formel φ keine Variable enthält, besteht die Wahrheitstafel aus einer einzigen Zeile. So ist zum Beispiel die erweiterte Wahrheitstafel der Formel $\varphi := (\top \to \bot) \to \bot$ gerade

$\bot$	$\top$	$\top \to \bot$	$(\top \to \bot) \to \bot$
$\bot$	$\top$	$\bot$	$\top$

Wenn wir mit Wahrheitstafeln arbeiten, nehmen wir implizit an, dass $[\![\varphi]\!]^\beta$ nur von den Belegungen der Variablen in $\mathrm{var}(\varphi)$ abhängt, aber nicht von der Belegung anderer Variablen. Eine Rechtfertigung dieser Annahme liefert das folgende Lemma.

Lemma 2.20 (Koinzidenzlemma) *Sei $\varphi \in \mathrm{AL}$ eine Formel und seien β, β' Belegungen so dass*

$$\beta(X) = \beta'(X) \qquad \textit{für alle } X \in \mathrm{var}(\varphi).$$

Dann gilt $[\![\varphi]\!]^\beta = [\![\varphi]\!]^{\beta'}$.

Intuitiv ist das Lemma klar, da die anderen Variablen $Y \notin \mathrm{var}(\varphi)$ nicht in der Definition von $[\![\varphi]\!]^\beta$ auftauchen. Formal kann das Lemma durch strukturelle Induktion bewiesen werden (Übung).

Betrachten wir nun noch einmal das Beispiel der Syllogismen (Beispiel 2.3 und 2.16), d.h. die Formel $\varphi := \big((X_M \to X_S) \wedge (X_K \to X_M)\big) \to (X_K \to X_S)$.

Die zugehörige Wahrheitstafel ist

X_H	X_M	X_S	$\big((X_H \to X_M) \wedge (X_S \to X_H)\big) \to (X_S \to X_M)$
0	0	0	1
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	1

In der letzten Spalte stehen ausschließlich einsen, d.h. die Formel ist unter allen passenden Belegungen wahr (wiederum verwenden wir implizit das Koinzidenzlemma). Diese Form der Syllogismen ist also korrekt.

Beobachtung 2.21 Sei $\varphi \in \mathrm{AL}$ eine Formel.

(1) φ ist genau dann *erfüllbar*, wenn die letzte Spalte der Wahrheitstafel mindestens eine 1 enthält.

(2) φ ist genau dann *unerfüllbar*, wenn alle Einträge der letzten Spalte 0 sind.

(3) φ ist genau dann *allgemeingültig*, wenn alle Einträge der letzten Spalte 1 sind. ⊣

Diese Beobachtung ist Kern des sogenannten *Wahrheitstafelverfahrens*.

Wahrheitstafelverfahren.

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

Ziel: Entscheide, ob φ erfüllbar ist.

Methode: (1) Berechne die Wahrheitstafel für φ.

(2) Überprüfe, ob die letzte Spalte eine 1 enthält.

Bemerkung. Für Allgemeingültigkeit entscheidet man, ob die letzte Spalte nur 1 enthält.

Effizienz des Wahrheitstafelverfahrens. Die Wahrheitstafel einer Formel mit n Variablen hat 2^n Zeilen, die alle ausgerechnet werden müssen. Das macht das Wahrheitstafelverfahren extrem ineffizient, außer für sehr kleine Formeln.

Variablen	Zeilen
10	$1{,}024 \approx 10^3$
20	$1{,}048{,}576 \approx 10^6$
30	$1{,}073{,}741{,}824 \approx 10^9$
40	$1{,}099{,}511{,}627{,}776 \approx 10^{12}$
50	$1{,}125{,}899{,}906{,}842{,}624 \approx 10^{15}$
60	$1{,}152{,}921{,}504{,}606{,}846{,}976 \approx 10^{18}$

Formeln, die in praktischen Anwendungen der Aussagenlogik auf Erfüllbarkeit getestet werden müssen, haben oft hunderte oder tausende, bisweilen sogar millionen Variablen. Das Wahrheitstafelverfahren findet daher in der Praxis keine nennenswerte Anwendung.

Bemerkung 2.22 Das Erfüllbarkeitsproblem der Aussagenlogik ist eines der am besten studierten Probleme der Informatik. Es ist algorithmisch „schwer“ zu lösen, genauer gesagt ist das Problem NP-vollständig (siehe Satz 3.20).

Allerdings existieren Verfahren, die das Problem für viele in der Praxis vorkommende Formeln sehr effizient lösen können. (Das Wahrheitstafelverfahren gehört nicht dazu). Diese haben wichtige Anwendungen in der Informatik, z.B. in der Verifikation. ⊣

2.5. Äquivalenz und Normalformen

In diesem Abschnitt werden wir Methoden behandeln, Formeln umzuformen, ohne den Wahrheitswert der Formeln zu ändern. Solche Umformungen sind wichtig für die Effizienz algorithmischer Probleme, etwa des Erfüllbarkeitstests. Dies werden wir insbesondere für sogenannte Normalformen benutzen.

Allgemein ist eine *Normalform* eine Klasse aussagenlogischer Formeln, so dass jede Formel in AL zu einer Formel in dieser Normalform äquivalent ist.

Dazu werden wir zunächst den Äquivalenzbegriff von Formeln formal definieren und einige nützliche Äquivalenzen zwischen Formeln einführen.

2.5.1. Äquivalenz von Formeln

Definition 2.23 Zwei Formeln $\varphi, \psi \in \mathrm{AL}$ sind *äquivalent*, geschrieben $\varphi \equiv \psi$, wenn für alle Belegungen β gilt:

$$\beta \models \varphi \iff \beta \models \psi$$

(vgl. Def. 2.17) ⊣

Proposition 2.24 (1) *Seien $\varphi, \psi \in \mathrm{AL}$. Dann gilt $\varphi \equiv \psi$ genau dann, wenn $\varphi \leftrightarrow \psi$ allgemeingültig ist.*

(2) *Eine Formel φ ist genau dann allgemeingültig, wenn $\varphi \equiv \top$.*

Beispiel 2.25 Für alle $X, Y \in \mathrm{AVAR}$:

$$X \to Y \equiv \neg X \vee Y$$
$$X \leftrightarrow Y \equiv (X \to Y) \wedge (Y \to X).$$

Wir beweisen die Äquivalenzen mit Hilfe einer Wahrheitstafel.

$[\![X]\!]^\beta$	$[\![Y]\!]^\beta$	$[\![X \to Y]\!]^\beta$	$[\![\neg X \vee Y]\!]^\beta$	$[\![(X \leftrightarrow Y)]\!]^\beta$	$[\![(X \to Y) \wedge (Y \to X)]\!]^\beta$
0	0	1	1	1	1
0	1	1	1	0	0
1	0	0	0	0	0
1	1	1	1	1	1

Wie man sieht, ergeben sich für die linken und rechten Seiten jeweils die selben Wahrheitswerte, d.h. die entsprechenden Formeln sind in der Tat äquivalent.⊣

Das vorherige Beispiel zeigt, dass wir in bestimmten Fällen Implikationen durch Negation und Disjunktion ausdrücken können. Wir würden nun gerne aus der Äquivalenz $(X \to Y) \equiv \neg X \vee Y$, für alle $X, Y \in \text{AVAR}$, schließen, dass für alle $\varphi, \psi \in \text{AL}$

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

gilt. Intuitiv ist natürlich völlig klar, dass die Äquivalenz für alle φ, ψ gilt, da wir ja in der Wahrheitstafel nirgends verwendet haben, dass X, Y Variablen sind. Das formale Hilfsmittel, solche Transferschlüsse machen zu dürfen, liefert das *Substitutionslemma*.

Substitution.

Informell kann man das Substitutionslemma wie folgt formulieren.
Substitutionslemma (informell). Wenn wir in einer Äquivalenz alle Vorkommen von Variablen $X_1, \ldots, X_n$ durch Formeln $\varphi_1, \ldots, \varphi_n$ ersetzen, bleibt die Äquivalenz erhalten.

Als Folgerung des Lemmas erhalten wir dann, dass für alle Formeln $\varphi, \psi \in \text{AL}$ gilt:

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

Wir formulieren das Lemma nun formal.

Definition 2.26 (Substitution) Eine *Substitution* ist eine partielle Abbildung

$$\mathcal{S} : \text{AVAR} \to \text{AL}$$

von Aussagenvariablen auf aussagenlogische Formeln mit endlichem Definitionsbereich, d.h. $\mathcal{S}$ ist nur für endlich viele Variablen definiert. ⊣

Für eine Formel $\varphi \in \text{AL}$ und eine Substitution $\mathcal{S}$ schreiben wir $\varphi\mathcal{S}$ für die Formel, die wir aus φ erhalten, wenn alle Vorkommen einer Variablen $X \in \text{Dom}(\mathcal{S})$ in φ durch die Formeln $\mathcal{S}(X)$ ersetzt werden.

Formal ist dies wie in der folgenden Definition formuliert.

Definition 2.27 Für jede Formel $\varphi \in \text{AL}$ und Substitution $\mathcal{S}$ definieren wir die Formel $\varphi\mathcal{S} \in \text{AL}$ induktiv wie folgt:

Induktionsbasis.

- $\bot\mathcal{S} := \bot, \qquad \top\mathcal{S} := \top$
- Für $X \in \text{AVAR}$ definieren wir $X\mathcal{S} := \begin{cases} \mathcal{S}(X) & \text{wenn } X \in \text{def}(\mathcal{S}) \\ X & \text{sonst.} \end{cases}$

Induktionsschritt.

- $(\neg\varphi)\mathcal{S} := \neg(\varphi\mathcal{S})$
- Für $* \in \{\wedge, \vee, \to, \leftrightarrow\}$ definieren wir $(\varphi * \psi)\mathcal{S} := (\varphi\mathcal{S} \ * \ \psi\mathcal{S})$. ⊣

Beispiel 2.28 Sei $\varphi := V_1 \wedge V_2 \to V_1 \vee V_3$ und sei

$$\mathcal{S} : \{V_1, V_2\} \to \text{AL}$$

definiert als $\mathcal{S}(V_1) := V_2$ und $\mathcal{S}(V_2) := (V_0 \vee V_1)$. Dann gilt

$$\begin{aligned} \varphi\mathcal{S} &= (V_1 \wedge V_2)\mathcal{S} \to (V_1 \vee V_3)\mathcal{S} \\ &= V_1\mathcal{S} \wedge V_2\mathcal{S} \to V_1\mathcal{S} \vee V_3\mathcal{S} \\ &= V_2 \wedge (V_0 \vee V_1) \to V_2 \vee V_3. \end{aligned}$$ ⊣

Beispiel 2.29 Sei $\varphi := ((V_1 \vee V_2) \wedge \neg(V_3 \wedge V_1))$ und sei $\mathcal{S} : \{V_1, V_2, V_3\} \to \text{AL}$ definiert als

$$\mathcal{S} : \begin{cases} V_1 & \mapsto \neg(V_2 \wedge V_3) \\ V_2 & \mapsto (V_4 \vee V_5) \\ V_3 & \mapsto (V_2 \wedge V_3) \end{cases}$$

Dann gilt

$$\begin{aligned} \varphi\mathcal{S} = ((V_1 \vee V_2) \wedge \neg(V_3 \wedge V_1))\mathcal{S} \\ ((V_1 \vee V_2)\mathcal{S} \wedge \neg(V_3 \wedge V_1)\mathcal{S}) \\ ((V_1 \vee V_2)\mathcal{S} \wedge (\neg(V_3 \wedge V_1))\mathcal{S}) \\ ((V_1\mathcal{S} \vee V_2\mathcal{S}) \wedge \neg((V_3 \wedge V_1)\mathcal{S})) \\ ((V_1\mathcal{S} \vee V_2\mathcal{S}) \wedge \neg(V_3\mathcal{S} \wedge V_1\mathcal{S})) \\ ((\neg(V_2 \wedge V_3) \vee (V_4 \vee V_5)) \wedge \neg((V_2 \wedge V_3) \wedge \neg(V_2 \wedge V_3))). \end{aligned}$$ ⊣

Lemma 2.30 (Substitutionslemma (formal)) *Sei $\mathcal{S}$ eine Substitution und seien $\varphi, \varphi' \in$ AL Formeln. Dann gilt*

$$\varphi \equiv \varphi' \quad \Rightarrow \quad \varphi\mathcal{S} \equiv \varphi'\mathcal{S}.$$

Der Beweis wird per Induktion über den Formelaufbau geführt. Wir führen *strukturelle Induktion* zunächst allgemein ein und beweisen dann das Lemma.

Strukturelle Induktion.

Beweise über strukturelle Induktion basieren auf dem induktiven Aufbau aussagenlogischer Formeln. Das Prinzip ist eine elegante und nützliche Methode, Eigenschaften aussagenlogischer Formeln zu beweisen.

Um zu beweisen, dass eine Eigenschaft A für alle Formeln der Aussagenlogik gilt, müssen wir folgende Aussagen zeigen:

Induktionsbasis. Alle atomare Formeln φ haben die Eigenschaft A.

Induktionsschritt.

- Wenn φ die Eigenschaft A hat, so auch $\neg\varphi$.
- Wenn ψ und φ die Eigenschaft A haben, dann gilt A auch für $(\varphi * \psi)$, mit $* \in \{\vee, \wedge, \rightarrow, \leftrightarrow\}$.

Dann gilt A für alle Formeln $\varphi \in \mathrm{AL}$.

Beweis von Lemma 2.30. Um das Substitutionslemma zu beweisen, brauchen wir zunächst noch etwas Notation.

Sei $\mathcal{S}$ eine Substitution.

- Eine Belegung β ist *passend für* $\mathcal{S}$, wenn sie passend für alle Formeln $\mathcal{S}(X)$ mit $X \in \mathrm{def}(\mathcal{S})$ ist.
- Ist β eine zu $\mathcal{S}$ passende Belegung, so definieren wir $\beta\mathcal{S}$ wie folgt

$$\beta\mathcal{S}(X) := \begin{cases} [\![\mathcal{S}(X)]\!]^\beta & \text{wenn } X \in \mathrm{def}(\mathcal{S}) \\ \beta(X) & \text{wenn } X \in \mathrm{def}(\beta) \setminus \mathrm{def}(\mathcal{S}). \end{cases}$$

Behauptung 1. Für alle Formeln φ und alle zu φ und $\mathcal{S}$ passenden Belegungen β gilt:

$$\beta \models \varphi\mathcal{S} \quad \Longleftrightarrow \quad \beta\mathcal{S} \models \varphi.$$

Proof. Wir beweisen die Behauptung per Induktion über den Formelaufbau.

Induktionsbasis.

- Für $\varphi \in \{\top, \bot\}$ gilt $\varphi\mathcal{S} = \varphi$.
- Für $\varphi := X$, wobei $X \in \mathrm{AVAR} \setminus \mathrm{def}(\mathcal{S})$, gilt $\beta\mathcal{S}(X) = \beta(X)$ und $\varphi = \varphi\mathcal{S}$ und daher $[\![\varphi\mathcal{S}]\!]^\beta = [\![\varphi]\!]^{\beta\mathcal{S}}$.

- Für $\varphi := X \in \mathrm{Dom}(\mathcal{S})$ gilt: $\varphi\mathcal{S} = \mathcal{S}(X)$ und $\beta\mathcal{S}(X) = [\![S(X)]\!]^\beta$.
  Somit, $\beta \models \varphi\mathcal{S} \iff \beta\mathcal{S} \models \varphi$.

Induktionsschritt.

- *Negation.*

$$\begin{aligned}
\beta \models (\neg\varphi)\mathcal{S} &\iff \beta \models \neg(\varphi\mathcal{S}) && \text{Def. der Substitution}\\
&\iff \beta \not\models \varphi\mathcal{S}\\
&\iff \beta\mathcal{S} \not\models \varphi && \text{Induktionsvoraussetzung}\\
&\iff \beta\mathcal{S} \models \neg\varphi.
\end{aligned}$$

- *Konjunktion.*

$$\begin{aligned}
\beta \models (\varphi \wedge \psi)\mathcal{S} &\iff \beta \models (\varphi\mathcal{S} \wedge \psi\mathcal{S}) && \text{Def. der Subst.}\\
&\iff \beta \models \varphi\mathcal{S} \text{ und } \beta \models \psi\mathcal{S}\\
&\iff \beta\mathcal{S} \models \varphi \text{ und } \beta\mathcal{S} \models \psi && \text{Ind.-Vor.}\\
&\iff \beta\mathcal{S} \models (\varphi \wedge \psi).
\end{aligned}$$

- Das Argument für $* \in \{\vee, \to, \leftrightarrow\}$ ist analog.

⊣

Mit Hilfe der Behauptung können wir nun das Substitutionslemma beweisen.

Seien φ, φ' äquivalente Formeln. Wir zeigen, dass $\varphi\mathcal{S} \equiv \varphi'\mathcal{S}$, d.h. das für alle passenden Belegungen β:

$$\beta \models \varphi\mathcal{S} \iff \beta \models \varphi'\mathcal{S}.$$

Sei β eine zu $\varphi\mathcal{S}$ und $\varphi'\mathcal{S}$ passende Belegung. Dann ist $\beta\mathcal{S}$ passend für φ.

$$\begin{aligned}
\beta \models \varphi\mathcal{S} &\iff \beta\mathcal{S} \models \varphi && \text{nach Behautung 1}\\
&\iff \beta\mathcal{S} \models \varphi' && \text{da } \varphi \equiv \varphi'\\
&\iff \beta \models \varphi'\mathcal{S} && \text{nach Behauptung 1.}
\end{aligned}$$

Das schließt den Beweis des Substitutionslemmas ab.

Notation 2.31 (1) Sei $\varphi \in \mathrm{AL}$ eine Formel. Wenn $X_1, \ldots, X_n$ Variablen in φ sind und $\psi_1, \ldots, \psi_n \in \mathrm{AL}$, schreiben wir

$$\varphi[X_1/\psi_1, \ldots, X_n/\psi_n]$$

für die Formel $\varphi\mathcal{S}$, wobei $\mathcal{S}$ die wie folgt definierte Substitution ist

$$\mathrm{def}(\mathcal{S}) := \{X_1, \ldots, X_n\} \text{ und } \mathcal{S}(X_i) := \psi_i.$$

(2) Wir schreiben $\varphi(X_1, \ldots, X_n) \in \mathrm{AL}$ um anzudeuten, dass $\varphi \in \mathrm{AL}$ und $\mathrm{var}(\varphi) \subseteq \{X_1, \ldots, X_n\}$. ⊣

Wie wir in Beispiel 2.25 schon gesehen haben, gelten folgende Äquivalenzen.

$$(X \to Y) \equiv \neg X \vee Y \qquad \text{und} \qquad (X \leftrightarrow Y) \equiv (X \to Y) \wedge (Y \to X)$$

Mit Hilfe des Substitutionslemmas erhält man nun sofort folgende Konsequenz.

Korollar 2.32 *Für alle Formeln $\varphi, \psi \in \mathrm{AL}$:*

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

Wir würden nun gerne weiter folgern, dass

$$\varphi \leftrightarrow \psi \equiv (\varphi \to \psi) \wedge (\psi \to \varphi) \equiv (\neg\varphi \vee \psi) \wedge (\neg\psi \vee \varphi)$$

Die Substitution erlaubt diese Folgerung aber nicht, da hier Formeln durch äquivalente Formeln ersetzt werden. Das folgende Lemma gibt aber die Möglichkeit, solche Schlüsse zu ziehen.

Lemma 2.33 (Ersetzungslemma) *Sei $\varphi \in \mathrm{AL}$ eine Formel und ψ eine Unterformel von φ. Sei φ' eine Formel, die man aus φ erhält, indem man ein Vorkommen der Unterformel ψ durch eine äquivalente Formel $\psi' \equiv \psi$ ersetzt. Dann gilt $\varphi \equiv \varphi'$.*

Proof. Wir beweisen das Ersetzungslemma durch strukturelle Induktion.

Induktionsanfang. Ist φ atomar, so gilt offenbar $\psi = \varphi$ und somit $\varphi' = \psi'$. Nach Voraussetzung gilt also $\varphi \equiv \varphi'$.

Induktionsschritt. Angenommen, $\varphi := \neg\vartheta$. Ist $\psi = \varphi$, so ist nichts zu zeigen. Sonst ist ψ eine Teilformel von ϑ und $\varphi' = \neg\vartheta'$, wobei ϑ' die Formel ist, die aus ϑ entsteht, indem ψ durch ψ' ersetzt wird. Nach IV gilt $\vartheta \equiv \vartheta'$. Nun gilt also für alle passenden Belegungen β: $[\![\varphi]\!]^\beta = 1 - [\![\vartheta]\!]^\beta = 1 - [\![\vartheta']\!]^\beta = [\![\varphi']\!]^\beta$ und somit $\varphi \equiv \varphi'$.

Schließlich bleibt noch der Fall $\varphi := (\varphi_1 \wedge \varphi_2)$. Die anderen Fälle sind analog. Angenommen, $\varphi := (\varphi_1 \wedge \varphi_2)$. Wiederum, falls $\psi = \varphi$, so ist nichts zu zeigen.

Sonst können wir o.B.d.A. annehmen, dass ψ eine Unterformel von φ_1 ist. Sei φ_1' die Formel, die aus φ_1 durch Ersetzen von ψ durch ψ' entsteht. Nach IV gilt also $\varphi_1 \equiv \varphi_1'$ und somit, mit der gleichen Argumentation wie vorher, $\varphi \equiv \varphi'$.

Bemerkung 2.34 Da Unterformeln mehrfach vorkommen können, ist φ' nicht eindeutig. ⊣

Mit Hilfe des Ersetzungslemmas können wir nun folgendes beweisen.

Korollar 2.35 *Für alle Formeln φ, ψ:*

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

Das folgende Theorem listet einige häufig benutzte Äquivalenzen auf.

Theorem 2.36 *Für alle $\psi, \varphi, \vartheta \in$ AL gilt:*

(1) $\neg\neg\varphi \equiv \varphi$ *(Elimination doppelter Negation)*

(2) $\varphi \to \psi \equiv \neg\varphi \vee \psi$ *(Elimination der Implikation)*

(3) $\varphi \leftrightarrow \psi \equiv (\varphi \to \psi) \wedge (\psi \to \varphi)$ *(Elimination der Biimplikation)*

(4) $\neg(\psi \wedge \varphi) \equiv \neg\psi \vee \neg\varphi$
$\neg(\psi \vee \varphi) \equiv \neg\psi \wedge \neg\varphi$ *(de Morgansche Regeln)*

(5) $\psi \wedge (\varphi \vee \vartheta) \equiv (\psi \wedge \varphi) \vee (\psi \wedge \vartheta)$
$\psi \vee (\varphi \wedge \vartheta) \equiv (\psi \vee \varphi) \wedge (\psi \vee \vartheta)$ *(Distributivität)*

(6) $\psi \wedge (\psi \vee \varphi) \equiv \psi \vee (\psi \wedge \varphi) \equiv \psi$ *(Absorbtionsgesetz)*

(7) $\psi \wedge \varphi \equiv \varphi \wedge \psi$
$\psi \vee \varphi \equiv \varphi \vee \psi$ *(Kommutativität von $\wedge$ und $\vee$)*

(8) $\psi \wedge (\varphi \wedge \vartheta) \equiv (\psi \wedge \varphi) \wedge \vartheta$
$\psi \vee (\varphi \vee \vartheta) \equiv (\psi \vee \varphi) \vee \vartheta$ *(Assoziativität von $\wedge$ und $\vee$)*

Proof. Wir beweisen hier exemplarisch einige der Äquivalenzen. Die restlichen sind zur Übung empfohlen.

De Morgansche Regeln. Wir zeigen hier die Regel $\neg(\psi \wedge \varphi) \equiv \neg\psi \vee \neg\varphi$. Sei β eine passende Belegung. Dann gilt

$$\begin{aligned} [\![\neg(\psi \wedge \varphi)]\!]^\beta = 1 \quad & \text{gdw.} \quad \text{wenn } [\![(\psi \wedge \varphi)]\!]^\beta = 0. \\ & \text{gdw.} \quad \text{mindestens eins von } [\![\psi]\!]^\beta, [\![\varphi]\!]^\beta \text{ gleich 0} \\ & \text{gdw.} \quad \text{mindestens eins von } [\![\neg\psi]\!]^\beta, [\![\neg\varphi]\!]^\beta \text{ gleich 1.} \\ & \text{gdw.} \quad [\![(\neg\psi \vee \neg\varphi)]\!]^\beta = 1. \end{aligned}$$

Distributivität. Wir zeigen hier den Fall $\psi \vee (\varphi \wedge \vartheta) \equiv (\psi \vee \varphi) \wedge (\psi \vee \vartheta)$. Sei β eine passende Belegung.

Angenommen, $[\![\psi \vee (\varphi \wedge \vartheta)]\!]^\beta = 0$. Also gilt $[\![\psi]\!]^\beta = 0$ und $[\![(\varphi \wedge \vartheta)]\!]^\beta = 0$. Es folgt also, dass $[\![\varphi]\!]^\beta = 0$ oder $[\![\vartheta]\!]^\beta = 0$.

O.B.d.A. sei $[\![\varphi]\!]^\beta = 0$. Dann gilt aber $[\![(\psi \vee \varphi)]\!]^\beta = 0$ und somit $[\![(\psi \vee \varphi) \wedge (\psi \vee \vartheta)]\!]^\beta = 0$.

Sei nun $[\![\psi \vee (\varphi \wedge \vartheta)]\!]^\beta = 1$. Es gilt also $[\![\psi]\!]^\beta = 1$ oder $[\![(\varphi \wedge \vartheta)]\!]^\beta = 1$.

Falls $[\![\psi]\!]^\beta = 1$ so folgt $[\![(\psi \vee \varphi)]\!]^\beta = 1$ und $[\![\psi \vee \vartheta]\!]^\beta = 1$ und somit $[\![(\psi \vee \varphi) \wedge (\psi \vee \vartheta)]\!]^\beta = 1$.

Anderenfalls gilt $[\![\varphi]\!]^\beta = [\![\vartheta]\!]^\beta = 1$. Dann ist aber auch $[\![(\psi \vee \varphi)]\!]^\beta = [\![\psi \vee \vartheta]\!]^\beta = 1$ und somit $[\![(\psi \vee \varphi) \wedge (\psi \vee \vartheta)]\!]^\beta = 1$.

In beiden Fällen gilt also die Äquivalenz.

Notation 2.37 Die folgende Notation ist oft nützlich:

- $\bigwedge_{i=1}^n \varphi_i$ als Abkürzung für $(\varphi_1 \wedge \varphi_2 \wedge \cdots \wedge \varphi_n)$.
- $\bigvee_{i=1}^n \psi_i$ als Abkürzung für $(\varphi_1 \vee \varphi_2 \vee \cdots \vee \varphi_n)$. ⊣

Beispiel 2.38 Wir können z.B. $\bigwedge_{i=1}^{999} X_i \to Y$ schreiben um zu formalisieren, dass wenn alle 999 X_i wahr sind, so muss auch Y wahr sein. ⊣

Bemerkung 2.39 In der Schreibweise $\bigwedge_{i=1}^{n} \varphi_i$ verwenden wir implizit die Assoziativitätsregeln, die garantieren, dass die genaue Klammerung der Konjunktion den Wahrheitswert nicht ändert. ⊣

Wie wir oben schon gesehen haben, gelten folgende Äquivalenzen.

$$\begin{aligned}
\varphi \to \psi &\equiv \neg\varphi \vee \psi \\
\varphi \leftrightarrow \psi &\equiv (\varphi \to \psi) \wedge (\psi \to \varphi) \equiv (\neg\varphi \vee \psi) \wedge (\neg\psi \vee \varphi) \\
\varphi \wedge \psi &\equiv \neg(\neg\varphi \vee \neg\psi).
\end{aligned}$$

Mit Hilfe des Ersetzungslemmas erhalten wir daher folgende Aussage.

Korollar 2.40 *Jede aussagenlogische Formel ist äquivalent zu einer Formel ohne $\wedge$, $\to$, $\leftrightarrow$, d.h. in der nur $\top$, $\bot$, Variablen und $\vee$ und $\neg$ vorkommen.*

Wir nennen solche Formeln *reduziert*. Reduzierte Formeln sind nützlich, um die Zahl der Fälle in strukturellen Induktionen zu begrenzen.

2.5.2. Normalformen

Definition 2.41 Eine Formel $\varphi \in \mathrm{AL}$ ist in *Negationsnormalform* (NNF), wenn die Symbole $\to$ und $\leftrightarrow$ nicht vorkommen und Negation nur vor Variablen auftritt. ⊣

Beispiel 2.42 Die Formel $\varphi := (\neg X \vee Y) \wedge \neg Z$ ist in Negationsnormalform.⊣

Theorem 2.43 *Jede Formel $\varphi \in \mathrm{AL}$ ist äquivalent zu einer Formel $\varphi^* \in \mathrm{AL}$ in Negationsnormalform.*

Wir geben einen Algorithmus an, der zu jeder Formel $\varphi \in \mathrm{AL}$ eine äquivalente Formel in NNF konstruiert. Damit beweisen wir gleichzeitig auch das Theorem.

Wir wissen bereits, dass φ zu einer Formel φ' ohne $\to$, $\leftrightarrow$ äquivalent ist. O.B.d.A. nehmen wir daher an, dass $\to$, $\leftrightarrow$ nicht in φ vorkommen.

Durch Anwendung der Äquivalenzen 1. und 4. in Theorem 2.36 können wir die Formel in Negationsnormalform umwandeln. Die ist der Kern des in Abbildung 2.2 angegebenen Algorithmus'.

Die Arbeitsweise des Algorithmus' wird im folgenden Beispiel demonstriert.

Algorithmus NNF(φ).

Eingabe. Eine Formel $\varphi \in$ AL ohne $\rightarrow, \leftrightarrow$.

Ausgabe. Eine Formel $\varphi^* \equiv \varphi$ in NNF

Algorithmus. Wiederhole die folgenden Schritte.

Wenn φ in NNF ist, gib $\varphi^* := \varphi$ aus.

Wenn φ eine Unterformel ψ der Form $\neg\neg\psi_1$ enthält, dann ersetze ψ durch ψ_1 um φ' zu erhalten.

Wenn φ eine Unterformel ψ der Form $\neg(\psi_1 \wedge \psi_2)$ enthält, dann ersetze ψ durch $(\neg\psi_1 \vee \neg\psi_2)$ um φ' zu erhalten.

Wenn φ eine Unterformel ψ der Form $\neg(\psi_1 \vee \psi_2)$ enthält dann ersetze ψ durch $(\neg\psi_1 \wedge \neg\psi_2)$ um φ' zu erhalten.

Setze $\varphi := \varphi'$.

Abbildung 2.2.: Ein Algorithmus für die NNF.

Beispiel 2.44 Sei $\varphi := \neg(\neg(X \vee Y) \wedge (\neg(X \wedge Y) \vee Z))$. Dann erzeugt der Algorithmus folgende Zwischenformeln:

$$\begin{aligned}
\neg(\neg(X \vee Y) \wedge (\neg(X \wedge Y) \vee Z)) &\equiv (\neg\neg(X \vee Y) \vee \neg(\neg(X \wedge Y) \vee Z)) \\
&\equiv ((X \vee Y) \vee \neg(\neg(X \wedge Y) \vee Z)) \\
&\equiv ((X \vee Y) \vee \neg((\neg X \vee \neg Y) \vee Z)) \\
&\equiv ((X \vee Y) \vee (\neg(\neg X \vee \neg Y) \wedge \neg Z)) \\
&\equiv ((X \vee Y) \vee ((\neg\neg X \wedge \neg\neg Y) \wedge \neg Z)) \\
&\equiv ((X \vee Y) \vee ((X \wedge \neg\neg Y) \wedge \neg Z)) \\
&\equiv ((X \vee Y) \vee ((X \wedge Y) \wedge \neg Z)) \qquad \dashv
\end{aligned}$$

Die Korrektheit und Vollständigkeit des Algorithmus' beweisen wir im folgenden Lemma. Daraus folgt dann auch direkt Theorem 2.43.

Lemma 2.45 *Der Algorithmus terminiert auf jeder gültigen Eingabe $\varphi \in$ AL und konstruiert eine Formel φ^* in NNF, so dass $\varphi \equiv \varphi^*$.*

Proof. Der Algorithmus ersetzt in jedem Schritt eine Unterformel ψ von φ durch eine äquivalente Formel. Nach dem Ersetzungslemma sind daher φ und φ' äquivalent. Daraus folgt sofort die Korrektheit des Algorithmus'. Wir müssen noch zeigen, dass der Algorithmus auch auf jeder Eingabe terminiert. Dazu definieren wir eine Funktion $h :$ AL $\rightarrow \mathbb{N}$, die die *Höhe* einer Formel angibt, wie folgt:

- Ist ψ atomar, so gilt $h(\psi) := 0$.
- Ist $\psi := \neg\psi'$, so gilt $h(\psi) := 1 + f(\psi')$.
- Ist $\psi := (\psi_1 \vee \psi_2)$ oder $\psi := (\psi_1 \wedge \psi_2)$, so gilt $h(\psi) := 1 + \max\{f(\psi_1), f(\psi_2)\}$.

Die Höhe einer Formel ist also die Höhe des Syntaxbaums der Formel. Offensichtlich ist eine Formel genau dann in NNF, wenn jede Unterformel der Form $\neg\psi'$ die Höhe 1 hat.

Sei $f : \text{AL} \to \mathbb{N}$ definiert durch

$$f(\varphi) := \sum\{3^{h(\psi)} : \psi = \neg\psi' \text{ kommt als Unterformel in } \varphi \text{ vor } \}.$$

Man beachte, dass wir in der Definition von f mehrfach vorkommende Unterformeln auch mehrfach zählen. Z.B. ist $f(\varphi) = 3^2 + 3^3 + 3^2$ für die Formel

$$\varphi := (\neg(X \vee Y) \wedge \neg\neg(X \vee Y)),$$

da $\neg(X \vee Y)$ zweimal gezählt wird.

Behauptung 2. Sei φ eine Formel und φ' die Formel, die aus φ in einem Schritt des Algorithmus' entsteht. Dann gilt $f(\varphi') < f(\varphi)$.

Proof. Angenommen, $\psi := \neg\neg\psi_1$. Wir ersetzen also ψ durch ψ_1. Dadurch erhöht sich die Höhe der restlichen Negationsformeln nicht. Die Zahl solcher Formeln reduziert sich um 2 und somit gilt $f(\varphi') < f(\varphi)$.

Angenommen, $\psi := \neg(\psi_1 \vee \psi_2)$ oder $\psi := \neg(\psi_1 \wedge \psi_2)$. Dann bleibt die Höhe aller Negationsformeln außer ψ gleich. In der Summierung wird also $3^{h(\psi)}$ durch $3^{h(\neg\psi_1)} + 3^{h(\neg\psi_2)} = 2 \cdot 3^{h(\psi)-1} < 3^{h(\psi)}$ ersetzt. Also gilt $f(\varphi') < f(\varphi)$. ⊣

Der Beweis des Lemmas folgt sofort aus der Behautung.

Wir beweisen als nächstes zwei weitere Normalformen, die unter anderem aus algorithmischer Sicht wichtig sind.

Definition 2.46 Ein *Literal* L ist eine Aussagenvariable $X \in \text{AVar}$ oder deren Negation $\neg X$. Wir schreiben $\overline{L}$ für das *Komplementliteral*, oder *duale Literal*, definiert als

$$\overline{L} := \begin{cases} \neg X & \text{wenn } L = X \\ X & \text{wenn } L = \neg X. \end{cases}$$

Definition 2.47 Eine Formel $\varphi \in \mathrm{AL}$ ist in *disjunktiver Normalform (DNF)*, wenn sie folgende Gestalt hat:

$$\bigvee_{i=1}^{n} \Big(\bigwedge_{j=1}^{n_i} L_{i,j} \Big).$$

φ ist in *konjunktiver Normalform (KNF)*, wenn sie folgende Gestalt hat:

$$\bigwedge_{i=1}^{n} \Big(\bigvee_{j=1}^{n_i} L_{i,j} \Big).$$

⊣

Theorem 2.48 (1) *Jede Formel $\varphi \in \mathrm{AL}$ ist äquivalent zu einer Formel in disjunktiver Normalform.*

(2) *Jede Formel $\varphi \in \mathrm{AL}$ ist äquivalent zu einer Formel in konjunktiver Normalform.*

Das Theorem kann ähnlich wie der Satz über die Negationsnormalform bewiesen werden (Übung). Wir verwenden hier allerdings einen anderen Ansatz über *Boolesche Funktionen*.

Einschub: Boolesche Funktionen

Definition 2.49 Eine (n-stellige) *Boolesche Funktion*, nach Georg Boole benannt, ist eine Funktion

$$f : \{0,1\}^n \to \{0,1\}.$$

Wir definieren $\mathbf{B}^n$ als Menge aller n-stelligen Booleschen Funktionen. ⊣

Beispiel 2.50 Definiere

$$f(X_1, \ldots, X_n) := \begin{cases} 1 & \text{wenn die Mehrheit der } X_i \text{ gleich 1 sind} \\ 0 & \text{sonst.} \end{cases}$$

⊣

Proposition 2.51 *Jede Formel $\varphi(X_1, \ldots, X_n) \in \mathrm{AL}$ definiert eine Boolesche Funktion $f_\varphi := f(\varphi)$ mit*

$$f_\varphi : \{0,1\}^n \to \{0,1\}$$
$$f_\varphi(v_1, \ldots, v_n) := [\![\varphi]\!]^\beta$$

wobei $\beta(X_i) := v_i$, für alle $1 \leqslant i \leqslant n$.

Umgekehrt kann jede Boolesche Funktion durch eine Formel definiert werden.

Theorem 2.52 *Zu jeder Booleschen Funktion $f : \{0,1\}^n \to \{0,1\}$ gibt es eine Formel $\varphi_f(X_1, \ldots, X_n)$, so dass $f_\varphi = f$.*

Proof. Für jede Sequenz $\overline{v} := (v_1, \ldots, v_n) \in \{0,1\}^n$ definieren wir

$$\varphi_{\overline{v}} := \Big(\bigwedge_{v_i=1} X_i\Big) \;\wedge\; \Big(\bigwedge_{v_i=0} \neg X_i\Big).$$

Offensichtlich gilt für jede Belegung β mit $\beta \models \varphi_{\overline{v}}$: $\beta(X_i) = 1 \iff v_i = 1$ für alle $1 \leqslant i \leqslant n$.

Wir definieren nun die Funktion f durch die Formel

$$\varphi_f(X_1, \ldots, X_n) := \bigvee_{\substack{\overline{v} \in \{0,1\}^n \\ f(\overline{v})=1}} \varphi_{\overline{v}}.$$

Es bleibt zu zeigen, dass für alle $\overline{v} := (v_1, \ldots, v_n)$ gilt:

$$f(\overline{v}) = 1 \iff [\![\varphi_f]\!]^\beta = 1$$

wobei $\beta(X_i) := v_i$, für alle $1 \leqslant i \leqslant n$.

(1) Wenn $f(\overline{v}) = 1$, dann $\beta \models \varphi_{\overline{v}}$ und $\varphi_{\overline{v}}$ ist Teil der Disjunktion in φ_f. Also $\beta \models \varphi_f$.

(2) Umgekehrt, wenn $\beta \models \varphi_f$, dann muss es ein Disjunktionsglied $\varphi_{\overline{w}}$ geben, so dass $\beta \models \varphi_{\overline{w}}$. Nach Konstruktion von φ_f gilt $f(\overline{w}) = 1$. Aber $\beta \models \varphi_{\overline{w}}$ genau dann, wenn $w_i = \beta(X_i) = v_i$ für alle $1 \leqslant i \leqslant n$. Also $f(\overline{v}) = 1$.

Das schließt den Beweis ab.

Beispiel 2.53 Sei

$$f(X_1, X_2, X_3) := \begin{cases} 1 & \text{falls die Mehrheit der } X_i \text{ gleich 1 ist} \\ 0 & \text{sonst.} \end{cases}$$

Dann ist $f(v_1, v_2, v_3) = 1$ wenn mindestens zwei der v_i gleich 1 sind. Wir erhalten also die folgende Formel

$$\begin{aligned} \varphi_f(X_1, X_2, X_3) := (&\neg X_1 \wedge X_2 \wedge X_3) && (=: \varphi_{0,1,1}) \\ &\vee (X_1 \wedge \neg X_2 \wedge X_3) && (=: \varphi_{1,0,1}) \\ &\vee (X_1 \wedge X_2 \wedge \neg X_3) && (=: \varphi_{1,1,0}) \\ &\vee (X_1 \wedge X_2 \wedge X_3) && (=: \varphi_{1,1,1}). \end{aligned}$$ ⊣

Wir haben bisher also folgenden Zusammenhang zwischen aussagenlogischen Formeln und Booleschen Funktionen gezeigt:

(1) Jede Formel $\varphi(X_1, \ldots, X_n) \in \mathrm{AL}$ definiert eine Boolesche Funktion f_φ.

(2) Zu jeder Booleschen Funktion $f : \{0,1\}^n \to \{0,1\}$ gibt es eine Formel $\varphi(X_1, \ldots, X_n)$, so dass $\varphi_f = f$.

D.h. es besteht ein Eins-Zu-Eins-Zusammenhang zwischen Formeln und Booleschen Funktionen. Eine nützliche Konsequenz dieses Zusammenhangs ist folgende Aussage.

Korollar 2.54 *Für alle $n \geqslant 0$ existieren genau 2^{2^n} paarweise nicht-äquivalente aussagenlogische Formeln in den Variablen $X_1, \ldots, X_n$.*

Proof. Es gibt 2^n verschiedene Belegungen der Variablen $X_1, \ldots, X_n$. Also existieren 2^{2^n} verschiedene n-stellige Boolesche Funktionen und somit ebensoviele paarweise nicht äquivalente aussagenlogische Formeln in den Variablen $X_1, \ldots, X_n$.

Zurück zu Normalformen

Mit Hilfe des Umwegs über Boolesche Funktionen können wir nun folgenden Satz beweisen.

Theorem 2.55 (1) *Jede Formel $\varphi \in \mathrm{AL}$ ist äquivalent zu einer Formel in disjunktiver Normalform.*

(2) *Jede Formel $\varphi \in \mathrm{AL}$ ist äquivalent zu einer Formeln in konjunktiver Normalform.*

Proof. Wir zeigen zunächst Teil 1. Sei $\varphi \in \mathrm{AL}$. Nach Proposition 2.51 ist φ äquivalent zu einer Booleschen Funktion f_φ. Nach Theorem 2.52 ist diese Funktion wiederum äquivalent zu einer aussagenlogischen Formel ψ. Die im Beweis des Theorems konstruierten Formeln sind in disjunktiver Normalform.

Teil 2 folgt nun leicht aus dem ersten Teil: Sei $\varphi \in \mathrm{AL}$. Nach Teil 1 ist $\neg\varphi$ äquivalent zu einer Formel $\psi := \bigvee_{i=1}^{n} \bigwedge_{j=1}^{n_i} L_{i,j}$ in DNF. Mit Hilfe der de Morganschen Regeln erhält man

$$\varphi \equiv \neg \bigvee_{i=1}^{n} \Big(\bigwedge_{j=1}^{n_i} L_{i,j} \Big) \quad \equiv \quad \bigwedge_{i=1}^{n} \Big(\bigvee_{j=1}^{n_i} \overline{L_{i,j}} \Big)$$

Bemerkung 2.56 Formeln in *disjunktiver Normalform* können sehr effizient auf Erfüllbarkeit getestet werden. Für alle $n \in \mathbb{N}$ gibt es aber Formeln $\varphi_n \in \mathrm{AL}$, so dass die Länge einer kürzesten, zu φ_n äquivalenten, DNF Formel exponentiell länger als φ_n ist. Es gibt also im Allgemeinen keinen effizienten Weg, um aussagenlogische Formeln in disjunktive oder konjunktive Normalform umzuwandeln. Jedoch kann zu jeder Formel $\varphi \in \mathrm{AL}$ in Polynomialzeit eine Formel $\psi \in \mathrm{AL}$ in *konjunktiver Normalform* konstruiert werden, so dass φ genau dann erfüllbar ist, wenn ψ erfüllbar ist. Siehe Theorem 3.5. Dies wird in praktischen Anwendungen benutzt, da die meisten aktuellen SAT-Löser Formeln in KNF als Eingabe erwarten. ⊣

2.6. Semantische Folgerung

Erinnern wir uns an das Beispiel vom Anfang der Vorlesung:

Voraussetzungen.

- Wenn der Zug zu spät ist und keine Taxis am Bahnhof stehen, kommt Peter zu spät zu seiner Verabredung.
- Peter kam nicht zu spät zu seiner Verabredung.
- Der Zug hatte Verspätung.

Folgerung. Also standen Taxis am Bahnhof.

Wir haben das Beispiel wie folgt in der Aussagenlogik formalisiert. Als Aussagenvariablen wurden die Variablen

T: Zug zu spät $\quad C$: Taxis am Bahnhof $\quad L$: Peter kam zu spät

verwendet. Die Argumentation kann wie folgt zusammengefasst werden.

$$\begin{aligned} \text{Aus den} \quad & \textit{Voraussetzungen:} \ \{(T \wedge \neg C) \to L, \quad \neg L, \quad T\} \\ \text{folgern wir} \quad & \textit{Folgerung:} \ C \end{aligned}$$

Frage. Wie können wir solche logischen Schlüsse formalisieren?

- Gegeben eine Menge von Voraussetzungen:

$$\left\{ \begin{array}{c} \text{„Zug zu spät und keine Taxis impliziert Peter zu spät“,} \\ \text{„Peter nicht zu spät, “Zug zu spät“} \end{array} \right\}.$$

  Können wir daraus „es gab Taxis am Bahnhof“ schließen?

- Gibt es allgemeine Methoden, um solche Folgerungen aus den Voraussetzungen zu ziehen? Methoden, mit denen
  - nur logisch korrekte Folgerungen abgeleitet werden können, die aber
  - allgemein genug sind, damit alle logisch korrekten Folgerungen abgeleitet werden können?

Damit zusammenhängend stellen sich algorithmische Fragen.

Frage. Wie können wir solche Folgerungen berechnen?

Die Wahrheitstafelmethode kann zwar für endliche Voraussetzungsmengen benutzt werden. Allerdings ist die Methode sehr ineffizient, da die Wahrheitstafeln sehr groß werden können.

Wir werden daher hier effizientere Verfahren kennen lernen, mit denen solche Folgerungen berechnet werden können.

Zunächst aber führen wir den *Folgerungsbegriff* zwischen Formelmengen und Formeln ein, einen der wichtigsten Begriffe der Logik überhaupt, nicht nur für die Aussagenlogik.

Definition 2.57 Sei $\Phi \subseteq \mathrm{AL}$ eine Formelmenge und $\psi \in \mathrm{AL}$ eine Formel.

ψ *folgt aus* Φ, wenn jede zu $\Phi \cup \{\psi\}$ passende Belegung β, die Φ erfüllt, auch ψ erfüllt. Wir schreiben $\Phi \models \psi$.

Falls $\Phi := \{\varphi\}$ nur eine Formel enthält, schreiben wir nur $\varphi \models \psi$. ⊣

Bemerkung 2.58 Wir verwenden das Symbol $\models$ sowohl für die Modellbeziehung $\beta \models \psi$ als auch für die semantische Folgerung $\Phi \models \psi$. Da auf der linken Seite des Symbols aber grundverschiedene Objekte stehen, besteht keine Verwechslungsgefahr. ⊣

Das folgende Lemma listet einige einfache Eigenschaften der Folgerungsbeziehung auf, die sofort aus der Definition folgen.

Lemma 2.59 *Sei* $\Phi \subseteq \mathrm{AL}$ *und* $\psi, \psi' \in \mathrm{AL}$.

(1) $\psi \equiv \psi'$ *genau dann, wenn* $\psi \models \psi'$ *und* $\psi' \models \psi$.

(2) $\Phi \models \psi$ *genau dann, wenn* $\Phi \cup \{\neg\psi\}$ *unerfüllbar ist (siehe Def. 3.1).*

(3) Φ *ist unerfüllbar genau dann, wenn für alle* $\varphi \in \mathrm{AL}$ *gilt* $\Phi \models \varphi$.

(4) *Sei* $\Phi_0 \subseteq \Phi$. *Wenn* $\Phi_0 \models \psi$, *dann auch* $\Phi \models \psi$.

Mit Hilfe des Begriffs der logischen Folgerung können wir die Argumentation im obigen Beispiel wie folgt zusammen fassen:

Aus den *Voraussetzungen:* $\{(T \wedge \neg C) \to L, \quad \neg L, \quad T\}$
folgern wir *Folgerung:* C

Das bedeutet: Wir wollen zeigen, dass

$$\{(T \wedge \neg C) \to L, \quad \neg L, \quad T\} \quad \models \quad C$$

Wir werden in der Vorlesung zwei Methoden kennen lernen, mit denen semantische Folgerungen automatisch und elegant überprüft werden können.

- *Aussagenlogische Resolution*
- *Der aussagenlogische Sequenzenkalkül*

Zunächst jedoch werden wir einen Satz beweisen, der uns in bestimmten Situationen auch die Behandlung unendlicher Formelmengen erlaubt bzw. vereinfacht.

3. Erfüllbarkeit und Entscheidbarkeit

3.1. Der Kompaktheitssatz der Aussagenlogik

In bestimmten Anwendungen der Aussagenlogik treten unendliche Formelmengen auf, die auf Erfüllbarkeit untersucht werden sollen. Wir werden als nächstes einen Satz beweisen, der uns diese Aufgabe wesentlich vereinfachen wird, da er es erlaubt, Erfüllbarkeit unendlicher Formelmengen auf Erfüllbarkeit endlicher Formelmengen zu reduzieren.

Definition 3.1 Eine Menge Φ aussagenlogischer Formeln ist *erfüllbar*, wenn es eine Belegung β gibt, die zu allen $\varphi \in \Phi$ passt und alle $\varphi \in \Phi$ erfüllt. Wir schreiben wiederum $\beta \models \Phi$. $\dashv$

Theorem 3.2 (Kompaktheits- oder Endlichkeitssatz) *Sei $\Phi \subseteq \mathrm{AL}$ eine Formelmenge und $\psi \in \mathrm{AL}$ eine Formel.*

(1) *Φ ist genau dann erfüllbar, wenn jede endliche Teilmenge $\Phi' \subseteq \Phi$ erfüllbar ist.*

(2) *$\Phi \models \psi$ genau dann, wenn eine endliche Teilmenge $\Phi_0 \subseteq \Phi$ existiert, so dass $\Phi_0 \models \psi$.*

Der Satz kann auch für überabzählbare Variablenmengen und somit überabzählbare Formelmengen bewiesen werden. Wir werden uns hier aber auf den abzählbaren Fall beschränken.

Proof. Wir werden zunächst den ersten Teil des Satzes beweisen, d.h.

> Eine Menge $\Phi \subseteq \mathrm{AL}$ ist genau dann erfüllbar, wenn jede endliche Teilmenge $\Phi' \subseteq \Phi$ erfüllbar ist.

Offenbar ist die Aussage trivial, wenn Φ bereits endlich ist.

Sei Φ also eine unendliche Menge aussagenlogischer Formeln. Ohne Beschränkung der Allgemeinheit nehmen wir an, dass es keine zwei verschiedenen

Formeln $\psi, \psi' \in \Phi$ gibt, so dass $\psi \equiv \psi'$. Denn, angenommen, es gäbe solche Formeln. Dann ist Φ genau dann erfüllbar, wenn $\Phi \setminus \{\psi'\}$ erfüllbar ist. Es reicht also, den Beweis für Formelmengen zu zeigen, in denen alle Formeln paarweise nicht äquivalent sind.

Zum Beweis der Hinrichtung sei Φ erfüllbar. Also existiert eine Belegung β, die jede Formel in Φ erfüllt. Also ist auch jede endliche Teilmenge von Φ erfüllbar, z. B. durch die Belegung β.

Zum Beweis der Rückrichtung nehmen wir an, dass alle endlichen Teilmengen $\Phi' \subseteq \Phi$ erfüllbar sind. Seien $X_1, X_2, \ldots$ die in Φ vorkommenden Aussagenvariablen.

Für $n \geqslant 0$ sei $\Phi_n \subseteq \Phi$ die Menge aller Formeln aus Φ, in denen nur die Variablen $X_1, \ldots, X_n$ vorkommen (es müssen aber nicht alle vorkommen). Es gilt also $\Phi_0 \subseteq \Phi_1 \subseteq ... \subseteq \Phi$.

Nach Korollar 2.54 gibt es höchstens 2^{2^n} paarweise nicht-äquivalente Formeln in n Variablen. Da Φ keine paarweise äquivalenten Formeln enthält, umfasst jedes Φ_n höchstens 2^{2^n} Formeln und ist damit endlich. Nach Voraussetzung gibt es also für jedes $n \geqslant 0$ eine Belegung β_n, so dass $\beta_n \models \Phi_n$ und somit auch $\beta_n \models \Phi_i$ für alle $i \leqslant n$. Sei $I_0 := \{\beta_n : n \geqslant 0\}$.

Wir konstruieren induktiv eine Belegung $\alpha : \{X_1, ...\} \to \{0, 1\}$ und Mengen $I_n \subseteq I_0$ wie folgt.

Dabei bewahren wir für alle n stets folgende Eigenschaft $(*)$:

- I_n ist unendlich, $\beta_1, \ldots, \beta_{n-1} \notin I_n$ und
- für alle $\beta, \beta' \in I_n$ und $j \leqslant n$ gilt $\beta(X_j) = \beta'(X_j) = \alpha(X_j)$.

Induktionsbasis $n = 1$. Da Φ unendlich ist, existiert ein $t \in \{0, 1\}$ so dass $\beta_n(X_1) = t$ für unendlich viele $\beta_n \in I_0$. Setze $\alpha(X_1) := t$ und $I_1 := \{\beta \in I_0 : \beta(X_1) = t \text{ und } \beta \neq \beta_1\}$. Offenbar ist $(*)$ erfüllt.

Induktionsvoraussetzung. Seien $I_{n-1}, \alpha(X_i)$ für alle $i < n$ schon konstruiert so dass $(*)$ gilt.

Induktionsschritt. Da I_{n-1} wegen $(*)$ unendlich ist, gibt es ein $t \in \{0, 1\}$, so das $\beta(X_n) = t$ für unendlich viele $\beta \in I_{n-1}$. Setze $\alpha(X_n) := t$ und $I_n := \{\beta \in I_{n-1} : \beta(X_n) = t \text{ und } \beta \neq \beta_{n-1}\}$. Offenbar ist $(*)$ erfüllt.

Behauptung 3. $\alpha \models \Phi$.

Proof. Sei $\varphi \in \Phi$. Da φ nur endlich viele Variablen enthält, ist $\varphi \in \Phi_n$ für ein n. Es gilt also $\beta_i \models \varphi$ für alle $i \geqslant n$, insbesondere also $\beta \models \varphi$ für alle $\beta \in I_n$. Sei $\beta \in I_n$. So ein β existiert, da wegen $(*)$ $I_n \neq \varnothing$. Da nach $(*)$ für alle $i \leqslant n$ gilt $\alpha(X_i) = \beta(X_i)$ und $\beta \models \varphi$, folgt also $\alpha \models \varphi$. ⊣

Aus der Behauptung folgt sofort der erste Teil des Satzes.

Wir zeigen nun den zweiten Teil des Satzes. Nach Lemma 2.59 gilt $\Phi \models \psi$ genau dann, wenn $\Phi \cup \{\neg\psi\}$ unerfüllbar ist.

Dies ist aber nach Teil 1. genau dann der Fall, wenn bereits eine endliche Teilmenge Φ_0 unerfüllbar ist. Ist $\neg\psi \in \Phi_0$, so gilt also $\Phi_0 \setminus \{\neg\psi\} \models \psi$. Anderenfalls ist $\Phi_0 \subseteq \Phi$, und da Φ_0 unerfüllbar ist, folgt $\Phi_0 \models \psi$ aus Lemma 2.59.

Die Umkehrung ist trivial.

Beispiel 3.3 Als Anwendung des Kompaktheitsatzes erhalten wir folgende Aussage aus der Graphentheorie.

Ein Graph $G := (V, E)$ besteht aus einer Knotenmenge V und einer Kantenmenge $E \subseteq \{\{u, v\} : u \neq v, u, v \in V\}$.

G ist *3-färbbar*, wenn es eine Funktion $c : V \to \{C_1, C_2, C_3\}$ gibt, so dass $c(u) \neq c(v)$ für alle Kanten $\{u, v\} \in E$.

Mit Hilfe des Kompaktheitssatzes kann man nun leicht folgern, dass ein Graph genau dann 3-färbbar ist, wenn bereits jeder endliche Untergraph 3-färbbar ist. ⊣

3.2. Aussagenlogische Resolution

Wir werden in diesem Abschnitt eine Methode kennen lernen, um die Unerfüllbarkeit aussagenlogischer Formeln nachzuweisen. Da für eine Formelmenge $\Phi \subseteq \mathrm{AL}$ und eine Formel ψ gilt:

$$\Phi \models \psi \text{ gdw. } \Phi \cup \neg\psi \text{ unerfüllbar ist}$$

können mit Hilfe der Resolution auch semantische Folgerungen überprüft werden.

Beispiel 3.4 Um die Idee der Resolutionsmethode zu demonstrieren, betrachten wir zunächst einmal folgendes Beispiel. Wir wollen zeigen, dass die folgende Formel φ unerfüllbar ist.

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

Angenommen, φ wäre erfüllbar. Sei β eine Belegung, so dass $\beta \models \varphi$.

- Offensichtlich gilt, $\beta \models \neg V$

- Aus $\beta \models \neg V$ und $\beta \models Y \vee Z \vee V$ folgt $\beta \models Y \vee Z$
- Aus $\beta \models Y \vee Z$ und $\beta \models \neg Y \vee W$ folgt $\beta \models Z \vee W$
- Aus $\beta \models Z \vee W$ und $\beta \models \neg W \vee Z$ folgt $\beta \models Z$
- Aus $\beta \models \neg X \vee \neg Z$ und $\beta \models X \vee \neg Z$ folgt aber auch $\beta \models \neg Z$
- Offensichtlich ist das ein Widerspruch zu $\beta \models Z$.

In diesem Beispiel haben wir wiederholt folgende Argumentation verwendet. Wenn eine Belegung β sowohl eine Formel $(Y \vee X)$ als auch $(\neg Y \vee Z)$ erfüllt, dann muss sie auch $(X \vee Z)$ erfüllen. Anders formuliert, erfüllt β die Formel $(Y \vee X) \wedge (\neg Y \vee Z)$, dann erfüllt sie auch $(X \vee Z)$. Man beachte, dass die Formel $(Y \vee X) \wedge (\neg Y \vee Z)$ in konjunktiver Normalform ist. Man nennt die Formel $(X \vee Z)$ die *Resolvente* aus $(Y \vee X)$ und $(\neg Y \vee Z)$.

Natürlich kann man das gleiche Prinzip auch anwenden, wenn statt X und Z kompliziertere Formeln stehen. Wichtig ist nur, dass Y einmal positiv und einmal negativ vorkommt. Das ist im wesentlichen schon das Grundprinzip der Resolution. Wir werden später sehen, dass diese einfache Art der Schlussfolgerung schon reicht, um alle aussagenlogischen Formeln auf Erfüllbarkeit zu testen. ⊣

Die aussagenlogische Resolution ist eine Methode um zu zeigen, dass eine Formel in *konjunktiver Normalform* nicht erfüllbar ist. Wir haben bereits gezeigt, dass jede Formel $\varphi \in \mathrm{AL}$ zu einer Formel ψ in KNF äquivalent ist. Also kann die Resolutionsmethode für alle Formeln verwendet werden, indem sie zunächst in KNF umgewandelt werden.

Für praktische Anwendungen der Resolutionsmethode ist folgender Satz nützlich.

Theorem 3.5 *Zu jeder Formel φ gibt es eine Formel ψ in KNF, so dass*

(1) *φ ist genau dann erfüllbar, wenn ψ erfüllbar ist.*

(2) *$|\psi| \leqslant c \cdot |\varphi|$ für eine Konstante $c \in \mathbb{N}$ unabhängig von φ.*

(3) *ψ kann aus φ effizient (in Linearzeit) berechnet werden.*

Notation 3.6 Um das Schreiben von Resolutionsableitungen zu vereinfachen, verwenden wir folgende Notation. Eine Formel

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

in KNF schreiben wir als *Klauselmenge* wie folgt:

$$\{\neg X, \neg Z\}, \quad \{X, \neg Z\}, \quad \{\neg Y, W\}, \quad \{Y, Z, V\}, \quad \{\neg V\}, \quad \{\neg W, Z\}$$

D.h., aus jeder Disjunktion $(Y \vee Z \vee V)$ wird eine als *Klausel* bezeichnete Menge $\{Y, Z, V\}$. ⊣

Die folgende Definition formalisiert diese Begriffe.

Definition 3.7

- Eine *Klausel* ist eine endliche Menge von Literalen. Die *leere Klausel* wird mit $\square$ bezeichnet.
- Mit jeder Formel $\varphi := \bigwedge_{i=1}^{n} \left(\bigvee_{j=1}^{m_i} L_{i,j} \right)$ in KNF assoziieren wir eine endliche Menge $\mathcal{C}(\varphi)$ von Klauseln wie folgt:
  - zu jeder Disjunktion $\bigvee_{j=1}^{m_i} L_{i,j}$ definieren wir die Klausel $C_i := \{L_{i,j} : 1 \leqslant j \leqslant m_i\}$
  - wir definieren $\mathcal{C}(\varphi) := \{C_1, \ldots, C_n\}$.
- Umgekehrt entspricht jeder Menge $\mathcal{C} := \{C_1, \ldots, C_n\}$ von Klauseln $C_i := \{L_{i,j} : 1 \leqslant j \leqslant m_i\}$ die Formel $\varphi(\mathcal{C}) := \bigwedge_{i=1}^{n} \left(\bigvee_{j=1}^{m_i} L_{i,j} \right)$. Falls $\mathcal{C} = \varnothing$, definieren wir $\varphi(\mathcal{C}) := \top$. Der leeren Klausel $\square$ wird die Formel $\bot$ zugeordnet. ⊣

Offensichtlich entsprechen sich Formeln in KNF und Klauselmengen eins zu eins. Wir weiten daher unsere Notation für Formeln auf Klauselmengen aus.

- Für eine Belegung β und Klauselmenge $\mathcal{C}$ schreiben wir $\beta \models \mathcal{C}$ für $\beta \models \varphi(\mathcal{C})$.
- Wir schreiben $\mathcal{C} \models C$ um zu sagen, dass jede $\mathcal{C}$ erfüllende Belegung auch die Klausel C erfüllt.
- Eine Klauselmenge $\mathcal{C}$ ist erfüllbar, wenn $\varphi(\mathcal{C})$ erfüllbar ist.
- Wenn $\mathcal{C}$ nur eine Klausel C enthält, schreiben wir einfach nur $\beta \models C$.

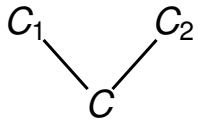
Offenbar erfüllt eine Belegung β eine Klauselmenge $\mathcal{C}$, wenn jede Klausel $C \in \mathcal{C}$ ein Literal L enthält, so dass $[\![L]\!]^\beta = 1$.

Insbesondere sind also Klauselmengen, die die leere Klausel enthalten, unerfüllbar. Wir erinnern an dieser Stelle an die Definition des dualen Literals aus Definition 2.46. Für ein Literal L ist $\bar{L}$ das duale Literal, d.h. $\bar{X} = \neg X$ und $\overline{\neg X} = X$.

Definition 3.8 Seien C, C_1, C_2 Klauseln. C ist eine *Resolvente* von C_1, C_2, wenn es ein Literal L gibt mit $L \in C_1$ und $\bar{L} \in C_2$ und $C = \big(C_1 \setminus \{L\}\big) \cup \big(C_2 \setminus \{\bar{L}\}\big)$.

Wir sagen, dass C_1 und C_2 *resolviert* werden und schreiben $Res(C_1, C_2)$ für die Menge der Resolventen von C_1 und C_2. ⊣

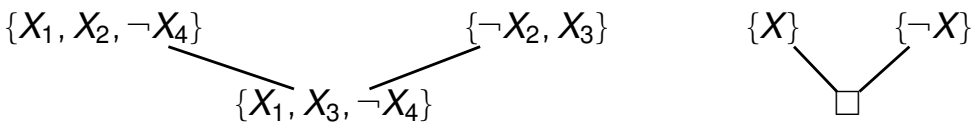
Wir werden Resolventen oft wie folgt graphisch darstellen



Beispiel 3.9 (1) Seien $C_1 := \{X_1, X_2, \neg X_4\}$ und $C_2 := \{\neg X_2, X_3\}$. Dann ist die Resolvente von C_1 und C_2 eindeutig und ergibt $\{X_1, X_3, \neg X_4\}$.

(2) Die Resolvente aus $\{X\}$ und $\{\neg X\}$ ist $\square$.

Die folgende Abbildung stellt die beiden Resolventen graphisch dar.



⊣

Bemerkung 3.10 Zwei Klauseln können mehr als eine Resolvente haben.

Zum Beispiel haben die Klauseln $\{X, Y, Z\}$ und $\{\neg X, \neg Y, W\}$ als Resolventen $\{Y, \neg Y, Z, W\}$ und $\{X, \neg X, Z, W\}$. Dies ist aber ein degenerierter Fall, der im weiteren keine Rolle spielen wird. Insbesondere ist die Resolvente in diesem Fall immer allgemeingültig, da sie ein Literal und sein duales Literal enthält.⊣

Lemma 3.11 *Sei $\mathcal{C}$ eine Klauselmenge. Seien $C_1, C_2 \in \mathcal{C}$ und $C \in Res(C_1, C_2)$. Dann gilt $\{C_1, C_2\} \models C$ und $\mathcal{C}$ und $\mathcal{C} \cup \{C\}$ sind äquivalent.*

Proof. Wir zeigen zunächst, dass $\{C_1, C_2\} \models C$. Wir müssen also zeigen, dass jede Belegung β mit $\beta \models \{C_1, C_2\}$ auch C erfüllt.

Sei also β eine Belegung mit $\beta \models \{C_1, C_2\}$. Sei L das resolvierte Literal, d.h. $C := \big(C_1 \setminus \{L\}\big) \cup \big(C_2 \setminus \{\bar{L}\}\big)$.

(1) Angenommen, $[\![L]\!]^\beta = 1$. Da $\beta \models \{C_2\}$ folgt, dass es ein $L' \in C_2 \setminus \{\bar{L}\}$ gibt mit $[\![L']\!]^\beta = 1$. Also $\beta \models \{C\}$, da nach Definition der Resolvente $L' \in C$.

(2) Sei nun $[\![L]\!]^\beta = 0$. Da $\beta \models \{C_1\}$, gibt es ein Literal $L' \in C_1 \setminus \{L\}$ mit $[\![L']\!]^\beta = 1$. Also $\beta \models \{C\}$, da nach Definition der Resolvente $L' \in C$.

Insgesamt gilt also $\beta \models C$.

Wir zeigen als Nächstes, dass $\mathcal{C}$ und $\mathcal{C} \cup \{C\}$ äquivalent sind. Dazu müssen wir zeigen, dass für alle Belegungen β gilt:

$$\beta \models \mathcal{C} \text{ gdw. } \beta \models \mathcal{C} \cup \{C\}.$$

Die Rückrichtung ist offensichtlich trivial. Für die andere Richtung sei β eine Belegung mit $\beta \models \mathcal{C}$. Da $C_1, C_2 \in \mathcal{C}$ gilt insbes. $\beta \models \{C_1, C_2\}$. Wie wir bereits gesehen haben gilt $\{C_1, C_2\} \models C$. Somit folgt auch $\beta \models C$, also erfüllt β alle Klauseln in $\mathcal{C} \cup \{C\}$.

Definition 3.12 (1) Eine *Resolutionsableitung* einer Klausel C aus eine Klauselmenge $\mathcal{C}$ ist eine Sequenz $(C_1, \ldots, C_n)$, so dass $C_n = C$ und für alle $1 \leqslant i \leqslant n$

- $C_i \in \mathcal{C}$ oder
- es gibt $j, k < i$ mit $C_i \in \mathit{Res}(C_j, C_k)$.

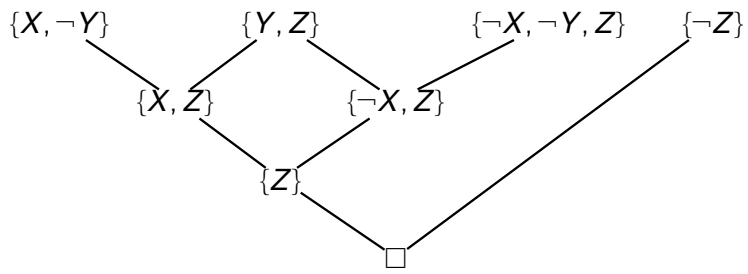
Wir sagen, dass C einen *Resolutionsbeweis* aus $\mathcal{C}$ hat und schreiben dies als $\mathcal{C} \vdash_R C$.

(2) Eine *Resolutionswiderlegung* einer Klauselmenge $\mathcal{C}$ ist eine Resolutionsableitung der leeren Klausel $\square$. ⊣

Beispiel 3.13 Sei $\mathcal{C} := \big\{\{X, \neg Y\}, \quad \{Y, Z\}, \quad \{\neg X, \neg Y, Z\}, \quad \{\neg Z\}\big\}$.
Dann ist

$$(\{X, \neg Y\}, \{Y, Z\}, \{X, Z\}, \{\neg X, \neg Y, Z\}, \{\neg X, Z\}, \{Z\}, \{\neg Z\}, \square)$$

eine Resolutionswiderlegung von $\mathcal{C}$. Es ist oft hilfreich, Resolutionsableitungen graphisch darzustellen.



⊣

Wir werden als nächstes zeigen, dass die Resolutionsmethode eine korrekte und auch vollständige Methode ist, um Unerfüllbarkeit von Klauselmengen, und damit auch aussagenlogischen Formeln, nachzuweisen.

Theorem 3.14 *Eine Menge $\mathcal{C}$ von Klauseln hat genau dann eine Resolutionswiderlegung, wenn $\mathcal{C}$ unerfüllbar ist.*

Zum Beweis des Satzes zeigen wir zunächst das folgende Lemma.

Lemma 3.15 *Sei $\mathcal{C}$ eine Klauselmenge und C eine Klausel. Wenn $\mathcal{C} \vdash_R C$ dann $\mathcal{C} \models C$.*

Proof. Sei $\big(C_1, \ldots, C_n\big)$ eine Resolutionsableitung von C aus $\mathcal{C}$. Per Induktion über i zeigen wir, dass $\mathcal{C} \models C_i$ für alle $1 \leqslant i \leqslant n$. Für $i = n$ folgt somit $\mathcal{C} \models C_n = C$.
Induktionsbasis: $i = 1$. Es gilt $C_1 \in \mathcal{C}$ und somit $\mathcal{C} \models C_1$.
Induktionsschritt. Angenommen, die Behauptung gilt für $1, ..., i$. Falls $C_{i+1} \in \mathcal{C}$, so gilt $\mathcal{C} \models C_{i+1}$. Anderenfalls gibt es $j, k < i + 1$ mit $C_{i+1} \in \mathit{Res}(C_j, C_k)$. Aus der Induktionsannahme folgt $\mathcal{C} \models C_j$ und $\mathcal{C} \models C_k$. Nach Lemma 3.11 gilt $\{C_j, C_k\} \models C_{i+1}$ und somit $\mathcal{C} \models C$.

Korollar 3.16 (Korrektheit des Resolutionskalküls) *Wenn eine Menge $\mathcal{C}$ von Klauseln eine Resolutionswiderlegung hat, dann ist $\mathcal{C}$ unerfüllbar.*

Wir zeigen als Nächstes die Umkehrung des Korollars.

Lemma 3.17 (Vollständigkeit des Resolutionskalküls) *Jede unerfüllbare Klauselmenge $\mathcal{C}$ hat eine Resolutionswiderlegung.*

Proof. Wir zeigen zunächst die folgende Behauptung.

Behauptung 4. Sei $n \in \mathbb{N}$ und sei $\mathcal{C}$ eine unerfüllbare Klauselmenge in der nur die Variablen $\{V_1, \ldots, V_{n-1}\}$ vorkommen. Dann hat $\mathcal{C}$ eine Resolutionswiderlegung.

Proof. Der Beweis der Behauptung wird per Induktion über n geführt.
Induktionsbasis $n = 1$. In diesem Fall ist $\mathcal{C}$ unerfüllbar und enthält keine Variablen. Also $\mathcal{C} := \{\square\}$ und somit existiert eine Resolutionswiderlegung.
Induktionsschritt $n \to n + 1$. Sei $\mathcal{C}$ eine unerfüllbare Klauselmenge in den Variablen $\{V_1, \ldots, V_n\}$.Wir definieren

$$\mathcal{C}^+ := \{C \setminus \{\neg V_n\} : C \in \mathcal{C} \text{ und } V_n \notin C\}$$

und

$$\mathcal{C}^- := \{C \setminus \{V_n\} : C \in \mathcal{C} \text{ und } \neg V_n \notin C\}.$$

D.h., man erhält $\mathcal{C}^+$ indem alle Klauseln, die V_n enthalten entfernt werden und $\neg V_n$ aus den anderen entfernt wird. $\mathcal{C}^-$ ist analog definiert. Intuitiv entspricht $\mathcal{C}^+$ der Klauselmenge, die man erhält, wenn man V_n mit dem Wahrheitswert 1 belegt, $\mathcal{C}^-$ der Klauselmenge, wenn V_n mit 0 belegt wird. Denn wird z.B. V_n mit 1 belegt, dann werden alle Klauseln erfüllt, die V_n enthalten (diese werden nicht in $\mathcal{C}^+$ aufgenommen). Die Klauseln, die $\neg V_n$ enthalten (aber nicht V_n), können nicht über das Literal $\neg V_n$ erfüllt werden, daher kann es aus den Klauseln gelöscht werden.

$\mathcal{C}^+$ und $\mathcal{C}^-$ sind beide unerfüllbar. Denn wäre z.B. $\mathcal{C}^+$ erfüllbar durch $\beta \models \mathcal{C}^+$, dann würde $\beta' := \beta \cup \{V_n \mapsto 1\}$ die Menge $\mathcal{C}$ erfüllen.

Nach Induktionsvoraussetzung gibt es Resolutionsableitungen $(C_1, \ldots, C_s)$ und $(D_1, \ldots, D_t)$ der leeren Klausel $C_s = D_t = \square$ aus $\mathcal{C}^+$ bzw. $\mathcal{C}^-$.

Falls $(C_1, \ldots, C_s)$ schon eine Ableitung von $\square$ aus $\mathcal{C}$ ist, sind wir fertig. Anderenfalls werden Klauseln C_i benutzt, die aus $\mathcal{C}$ durch Entfernen von $\neg V_n$ entstanden sind, d.h. $C_i \cup \{\neg V_n\} \in \mathcal{C}$. Fügen wir zu diesen Klauseln und allen daraus folgenden Resolventen wieder $\neg V_n$ hinzu, so erhalten wir eine Ableitung $(C'_1, \ldots, C'_s)$ von $\{\neg V_n\}$ aus $\mathcal{C}$. Analog ist entweder $(D_1, \ldots, D_t)$ bereits eine Ableitung von $\square$ aus $\mathcal{C}$ oder wir erhalten eine Ableitung $(D'_1, \ldots, D'_t)$ von $\{V_n\}$ aus $\mathcal{C}$. Ein weiterer Resolutionsschritt auf $\{\neg V_n\}$ und $\{V_n\}$ ergibt dann $\square$. In diesem Fall ist also $(C'_1, \ldots, C'_s, D'_1, \ldots, D'_t, \square)$ eine Resolutionswiderlegung von $\mathcal{C}$. ⊣

Mit Hilfe der Behauptung folgt nun leicht die Vollständigkeit der Resolution. Sei dazu $\mathcal{C}$ eine unerfüllbare Klauselmenge.

Ist $\mathcal{C}$ endlich, dann enthält sie nur endlich viele Variablen und der Beweis folgt sofort aus der Behauptung.

Ist $\mathcal{C}$ unendlich, dann folgt aus dem Kompaktheitssatz, dass bereits eine endliche Teilmenge $\mathcal{C}' \subseteq \mathcal{C}$ unerfüllbar ist. Also hat $\mathcal{C}'$ eine Resolutionswiderlegung. Diese ist aber auch eine Resolutionswiderlegung von $\mathcal{C}$.

Zusammengenommen ergeben die beiden vorherigen Aussagen also folgenden Satz.

Theorem 3.18 *Eine Menge $\mathcal{C}$ von Klauseln hat genau dann eine Resolutionswiderlegung, wenn $\mathcal{C}$ unerfüllbar ist.*

Der Resolutionskalkül ist also eine vollständige Methode, um Unerfüllbarkeit (und damit auch Erfüllbarkeit) nachzuweisen.

Trotz seiner abstrakten Formulierung hat das Erfüllbarkeitsproblem der Aussagenlogik wichtige algorithmische Anwendungen in der Informatik. Ein effizientes Verfahren zur Lösung des Problems hätte daher weitreichende Auswirkungen. Wir werden aber als Nächstes zeigen, dass ein solches Verfahren vermutlich nicht existieren kann, da das Problem NP-vollständig ist.

3.3. Das aussagenlogische Erfüllbarkeitsproblem

Definition 3.19 Das aussagenlogische Erfüllbarkeitsproblem (SAT) ist das Problem

SAT

Eingabe.	Eine aussagenlogische Formel $\varphi \in \mathrm{AL}$
Problem.	Entscheide, ob φ erfüllbar ist.

⊣

Theorem 3.20 (Cook 1970, Levin 1973) SAT *ist* NP*-vollständig.*

3.3.1. Erinnerung: NP-Vollständigkeit

Wir erinnern hier kurz an den Begriff der NP-Vollständigkeit eines Berechnungsproblems P bzw. einer Sprache $L \subseteq \Sigma^*$ über einem Alphabet Σ. Wir führen hier die Begriffe nur soweit ein, wie sie zum Beweis des Satzes von Cook und Levin nötig sind. Für eine ausführliche Behandlung des Stoffes verweisen wir auf die Vorlesung TheGI 2.

Definition 3.21 Ein *nicht-deterministischer Turing-Akzeptor* (NTM) ist ein Tupel $M := (Q, \Sigma, \Gamma, \delta, q_0, F_a, F_r)$, wobei

- Q eine endliche Menge ist, deren Elemente *Zustände* heißen,
- Σ eine endliche Menge ist, das *Eingabealphabet*,
- $\Gamma \supseteq \Sigma \cup \{\square\}$ eine endliche Menge ist, das *Arbeitsalphabet*,
- $\delta \subseteq (Q \setminus F) \times \Gamma \times Q \times \Gamma \times \{-1, 0, 1\}$ die *Transitionsrelation*,
- $q_0 \in Q$ der *Anfangszustand* ist und
- $F = F_a \dot\cup F_r \subseteq Q$ die Menge der *Endzustände* bezeichnet. Hierbei ist $F_a \cap F_r = \varnothing$. ⊣

Wir nehmen im weiteren Verlauf immer an, dass $\Sigma := \{0, 1\}$ und $\Gamma := \Sigma \dot\cup \{\square\}$. Der Begriff einer Konfiguration (q, w, p) überträgt sich direkt von deterministischen Turing-Maschinen.

Die Berechnung einer nicht-deterministischen Turing-Maschine $(Q, \Sigma, \Gamma, \Delta, q_0, F_a, F_r)$ auf Eingabe $w \in \Sigma^*$ ist ein *Berechnungsbaum*:

Die *Startkonfiguration* ist $(q_0, w, 0)$. Eine Konfiguration (q, w, p) mit $q \in F_a \cup F_r$ hat keinen Nachfolger und heißt *Stopkonfiguration*.

Eine Konfiguration (q, w, p) mit $w := w_1 \dots w_k$ und $p \leqslant k$ hat Nachfolger $(q', w_1 \dots w_{p-1} w' w_{p+1} \dots w_{k'}, p')$ für alle $(q, w_p, q', w', m) \in \Delta$ wobei $p' := p+m$. und

$$k' := \begin{cases} k+m & \text{wenn } p = k \text{ und } m = 1 \\ k & \text{sonst.} \end{cases}$$

Ein *Berechnungspfad* oder *Lauf* einer NTM $\mathcal{M}$ auf Eingabe w: ist ein Pfad im Berechnungsbaum von der Anfangskonfiguration $(q_0, w, 0)$ zu einer Stopkonfiguration (q_f, w', p). Wir nennen den Lauf *akzeptierend*, wenn $q \in F_a$ und *verwerfend* sonst. Die durch eine NTM $\mathcal{M}$ akzeptierte Sprache $\mathcal{L}(\mathcal{M})$ ist definiert als

$$\mathcal{L}(\mathcal{M}) := \{w \in \Sigma^* : \text{ es } \textit{existiert} \text{ ein akzeptierender Lauf von } \mathcal{M} \text{ auf } w\}$$

Bemerkung 3.22 Man beachte, dass auch hier Turing-Maschinen ein beidseitig unbeschränktes Arbeitsband verwenden. Wir nehmen aber für den Rest dieses Abschnitts immer an, dass die Maschine nie nach links über die 0-Position hinausläuft. Dies ist keine Einschränkung der Allgemeinheit, da wir jede NTM leicht so umschrieben können, dass sie nur im nicht-negativen Teil des Bandes bleibt. Dies erleichtert uns später die Formeln im Beweis des Satzes von Cook und Levin. ⊣

Definition 3.23 Sei $\mathcal{M}$ ein nicht-deterministischer Turing-Akzeptor und seien $S, T : \mathbb{N} \to \mathbb{N}$ Funktionen.

(1) $\mathcal{M}$ ist *T-Zeitbeschränkt*, wenn sie auf jeder Eingabe $w \in \Sigma^*$ nach $\leqslant T(|w|)$ Schritten hält. Genauer: Auf Eingabe w ist die Länge jedes Berechnungspfades von $\mathcal{M}$ höchstens $T(|w|)$.

(2) $\mathcal{M}$ ist *S-Platzbeschränkt* wenn, auf Eingabe $w \in \Sigma^*$, jeder Berechnungspfad $\leqslant S(|w|)$ Zellen benutzt. Wir nehmen dabei an, dass die NTM ein separates Eingabeband hat, dessen Größe wir nicht mitzählen. (Siehe Vorlesung TheGI 2.) ⊣

Definition 3.24 Seien $T, S : \mathbb{N} \to \mathbb{N}$ monoton wachsende Funktionen.

(1) $\textsc{Ntime}(T)$ ist die Klasse aller Sprachen $\mathcal{L}$ für die es eine T-zeitbeschränkte NTM gibt, die $\mathcal{L}$ entscheidet.

(2) $\textsc{Nspace}(S)$ ist die Klasse aller Sprachen $\mathcal{L}$, für die es eine S-platzbeschränkte NTM gibt, die $\mathcal{L}$ entscheidet. ⊣

Im folgenden geben wir einige der wichtigsten Komplexitätsklassen an.

- Zeitkomplexitätsklassen:
  - $\text{NP} := \bigcup_{d\in\mathbb{N}} \textsc{Ntime}(n^d)$
  - $\text{NExptime} := \bigcup_{d\in\mathbb{N}} \textsc{Ntime}(2^{n^d})$

- Platzkomplexitätsklassen:
  - $\text{NLogspace} := \bigcup_{d\in\mathbb{N}} \textsc{Nspace}(d \log n)$
  - $\text{NPspace} := \bigcup_{d\in\mathbb{N}} \textsc{Nspace}(n^d)$
  - $\text{NExpspace} := \bigcup_{d\in\mathbb{N}} \textsc{Nspace}(2^{n^d})$

Zur Definition der NP-Vollständigkeit brauchen wir noch den Begriff der Polynomialzeit-Many-One-Reduktion.

Definition 3.25 Eine Sprache $\mathcal{L}_1 \subseteq \Sigma^*$ ist *polynomiell reduzierbar* auf $\mathcal{L}_2 \subseteq \Sigma^*$, geschrieben $\mathcal{L}_1 \leqslant_p \mathcal{L}_2$, wenn es eine polynomialzeit berechenbare Funktion $f : \Sigma^* \to \Sigma^*$ gibt, so dass für alle $w \in \Sigma^*$

$$w \in \mathcal{L}_1 \qquad \Longleftrightarrow \qquad f(w) \in \mathcal{L}_2. \qquad \dashv$$

Lemma 3.26 *Wenn $\mathcal{L}_1 \leqslant_p \mathcal{L}_2$ und $\mathcal{L}_2 \in$ PTIME, dann auch $\mathcal{L}_1 \in$ PTIME.*

Wir brauchen meistens die Kontraposition: Wenn $\mathcal{L}_1 \leqslant_p \mathcal{L}_2$ und $\mathcal{L}_1 \notin$ PTIME, dann auch $\mathcal{L}_2 \notin$ PTIME.

Definition 3.27 Sei Σ ein Alphabet. Ein Problem $\mathcal{L} \subseteq \Sigma^*$ ist NP-vollständig, wenn es in NP liegt und für jedes Problem $\mathcal{L}' \in$ NP gilt $\mathcal{L}' \leqslant_p \mathcal{L}$. Man kann auch sagen $\mathcal{L}$ ist maximal (bzgl. $\leqslant_p$) in NP. ⊣

In TheGI 2 haben Sie schon gesehen, wie NP-Vollständigkeit eines Problems $\mathcal{L} \in$ NP dadurch gezeigt werden kann, dass man ein anderes NP-vollständiges Problem auf $\mathcal{L}$ reduziert. Die Herausforderung besteht darin, ein erstes Problem zu finden, von dem man NP-Vollständigkeit ohne Reduktionen nachweisen kann. Ende der 60er Jahre bewies Stephen Cook dies für das SAT Problem. Für seine bahnbrechenden Arbeiten zur NP-Vollständigkeit erhielt er 1982 den Turing-Award.

3.3.2. Der Satz von Cook und Levin

Theorem 3.28 (Cook 1970, Levin 1973) SAT *ist* NP*-vollständig.*

Proof. Wir zeigen zunächst, dass SAT $\in$ NP. D.h. wir müssen eine polynomialzeit beschränkte NTM angeben, die SAT entscheidet. Eine solche NTM kann bei Eingabe $\varphi \in$ AL einfach eine Belegung der Variablen var(φ) raten und dann in Polynomialzeit überprüfen, ob die geratene Belegung die Formel erfüllt.

Um die NP-Vollständigkeit des SAT Problems zu zeigen, müssen wir also nun noch folgendes beweisen: Für jede Sprache $\mathcal{L} \in$ NP gilt

$$\mathcal{L} \leqslant_p \text{SAT}.$$

Sei also $\mathcal{L} \in$ NP mit $\mathcal{L} \subseteq \Sigma^*$. Für jede Eingabe $w \in \Sigma^*$ konstruieren wir eine aussagenlogische Formel $\varphi_{\mathcal{L},w}$, so dass

$$w \in \mathcal{L} \iff \varphi_{\mathcal{L},w} \text{ erfüllbar.}$$

Da $\mathcal{L} \in$ NP, existiert eine 1-Band NTM $\mathcal{M} := (Q, \Sigma, \Gamma, q_0, \Delta, F_a, F_r)$, die $\mathcal{L}$ in Zeit $p(n)$, für ein Polynom p, entscheidet.

Man beachte: auf Eingaben der Länge n ist die Länge jedes Berechnungspfads und auch die Länge jeder erreichbaren Konfiguration von $\mathcal{M}$ durch $p(n)$ beschränkt.

Die Idee der Konstruktion von $\varphi_{\mathcal{L},w}$ besteht darin, den Lauf von $\mathcal{M}$ auf Eingabe w durch eine Formel zu beschreiben.

Beschreiben einer Konfiguration. Als ersten Schritt repräsentieren wir die Menge potentieller Konfigurationen $Q \times \{0, \ldots, p(n)\} \times \Gamma^{p(n)}$ durch die Variablenmenge

$$\overline{C} := \{Q_q : q \in Q\} \cup \{P_i : 0 \leqslant i \leqslant p(n)\} \cup \{S_{a,i} : a \in \Gamma, 0 \leqslant i \leqslant p(n)\}$$

Eine Belegung β soll nun folgendermaßen einer konkreten Konfiguration $(q, p, a_0 \ldots a_{p(n)})$ entsprechen:

$$\beta(Q_s) := \begin{cases} 1 & s = q \\ 0 & s \neq q \end{cases}, \qquad \beta(P_s) := \begin{cases} 1 & s = p \\ 0 & s \neq p \end{cases} \qquad \beta(S_{a,i}) := \begin{cases} 1 & a = a_i \\ 0 & a \neq a_i \end{cases}$$

Dabei haben die Variablen folgende intendierte Bedeutung.

Q_q: Für alle $q \in Q$
Bedeutung: $\mathcal{M}$ ist im Zustand $q \in Q$

P_i: Für alle $0 \leqslant i \leqslant p(n)$
Bedeutung: der Kopf steht an Position i

$S_{a,i}$: Für alle $a \in \Gamma$ und $0 \leqslant i \leqslant p(n)$
Bedeutung: Bandposition i enthält Symbol a

Als nächstes konstruieren wir folgende Formel

$$\textsc{Conf}(\overline{C}) := \bigvee_{q \in Q} \Big(Q_q \wedge \bigwedge_{q' \neq q} \neg Q_{q'} \Big) \qquad \wedge \qquad \bigvee_{p \leqslant p(n)} \Big(P_p \wedge \bigwedge_{p' \neq p} \neg P_{p'} \Big) \wedge$$

$$\bigwedge_{1 \leqslant i \leqslant p(n)} \bigvee_{a \in \Gamma} \big(S_{a,i} \wedge \bigwedge_{b \neq a \in \Gamma} \neg S_{b,i} \big)$$

Die Formel beschreibt, dass genau eine der Variablen Q_q, für $q \in Q$, mit 1 belegt werden muss, genau eine Variable P_p für ein $p \leqslant p(n)$ sowie für jede Position $0 \leqslant i \leqslant p(n)$ genau eine der Variablen $S_{a,i}$ für ein $a \in \Gamma$. D.h. eine erfüllende Belegung der Formel wählt genau einen Zustand, genau eine Bandposition des Kopfes und für jede Position des Bandes genau eine Beschriftung aus.

Für jede Belegung β von $\overline{C}$ definieren wir *conf*$(\overline{C}, \beta)$ als

$$\big\{(q, p, w_1 \dots w_{p(n)}) : \beta(Q_q) = 1, \beta(P_p) = 1, \beta(S_{w_i,i}) = 1, 0 \leqslant i \leqslant p(n)\big\}$$

Aus der Konstruktion folgt sofort folgende Behauptung.

Behauptung 5. Wenn $\beta \models \textsc{Conf}(\overline{C})$, dann gilt $|\mathrm{conf}(\overline{C}, \beta)| = 1$.

Man beachte, dass β auch für weitere Variablen neben $\overline{C}$ definiert sein kann.

Wenn β eine Variablenbelegung ist, die die Formel $\textsc{Conf}(\overline{C})$ erfüllt, dann schreiben wir im Folgenden kurz *(q, p, w) := conf*$(\overline{C}, \beta)$. Nach Behauptung 5 ist die Konfiguration (q, p, w) eindeutig bestimmt. Im folgenden werden wir die Formel Conf auch für Variablenmengen

$$\overline{C}' := \big\{Q'_q,\, P'_i,\, S'_{a,i} : \quad q \in Q, \quad a \in \Gamma, \quad 0 \leqslant i < p(n)\big\}$$

und

$$\overline{C}_t := \big\{Q_{q,t},\, P'_{i,t},\, S'_{a,i,t} : \quad q \in Q, \quad a \in \Gamma, \quad 0 \leqslant i < p(n)\big\}$$

für ein $t \leqslant p(n)$ verwenden. In der Formel Conf werden dann die Variablen aus $\overline{C}$ durch die entsprechenden Variablen aus $\overline{C}'$ bzw. $\overline{C}_t$ ersetzt.

conf$(\overline{C}, \beta)$ ist eine *potentielle* Konfiguration von $\mathcal{M}$, aber eventuell ist sie nicht von der Startkonfiguration von $\mathcal{M}$ auf Eingabe w erreichbar. Umgekehrt induziert jede Konfiguration $(q, p, w_1 \dots w_{p(n)})$ eine Belegung β für $\overline{C}$, die $\textsc{Conf}(\overline{C})$ erfüllt.

Wir betrachten als nächstes die folgende Formel

$$\textsc{Next}(\overline{C}, \overline{C}') := \textsc{Conf}(\overline{C}) \wedge \textsc{Conf}(\overline{C}') \wedge \textsc{Nochange}(\overline{C}, \overline{C}') \wedge \textsc{Change}(\overline{C}, \overline{C}').$$

$$\textsc{Nochange} := \bigvee_{0 \leqslant p \leqslant p(n)} \Big(P_p \wedge \bigwedge_{\substack{i \neq p \\ a \in \Gamma}} \big(S_{a,i} \to S'_{a,i}\big)\Big)$$

$$\textsc{Change} := \bigvee_{0 \leqslant p \leqslant p(n)} \Big(P_p \wedge \bigvee_{\substack{q \in Q \\ a \in \Gamma}} \big(Q_q \wedge S_{a,p} \wedge \bigvee_{(q,a,q',b,m) \in \Delta} (Q'_{q'} \wedge S'_{b,p} \wedge P'_{p+m})\big)\Big)$$

Sie erzwingt in erfüllenden Belegungen β zunächst, dass die Belegung von $\overline{C}$ und $\overline{C}'$ Konfigurationen von $\mathcal{M}$ entspricht. Danach sagt die Formel Nochange,

dass die Beschriftung des Bandes sich nur dort ändern darf, wo sich der Schreib/Lesekopf befindet. Schließlich stellt die Formel CHANGE sicher, dass die Veränderung zwischen den Q_q und Q'_q, zwischen $S_{a,p}$ und $S'_{a,p}$ sowie zwischen P_p und P'_p einer Transition von $\mathcal{M}$ entspricht. Aus der Konstruktion der Formel folgt somit die nächste Behauptung.

Behauptung 6. Für jede Belegung β, die auf $\overline{C}, \overline{C}'$ definiert ist:

$$\beta \text{ erfüllt } \textsc{Next}(\overline{C}, \overline{C}') \quad \Longleftrightarrow \quad \mathit{conf}(\overline{C}, \beta) \vdash_{\mathcal{M}} \mathit{conf}(\overline{C}', \beta)$$

Wir haben also bisher eine Formel $\textsc{Conf}(\overline{C})$ definiert, die besagt, dass $\overline{C}$ eine potentielle Konfiguration beschreibt, sowie ein Formel $\textsc{Next}(\overline{C}, \overline{C}')$, die einen Schritt $\mathit{conf}(\overline{C}, \beta) \vdash_{\mathcal{M}} \mathit{conf}(\overline{C}', \beta)$ der NTM $\mathcal{M}$ beschreibt.

Wir müssen nun noch die Startkonfiguration beschreiben. Sei dazu $w := w_0 \dots w_{n-1} \in \Sigma^*$ eine Eingabe von $\mathcal{M}$. Wir definieren

$$\textsc{Start}_{\mathcal{M},w}(\overline{C}) := \textsc{Conf}(\overline{C}) \wedge Q_{q_0} \ \wedge P_0 \ \wedge \bigwedge_{i=0}^{n-1} S_{w_i,i} \wedge \bigwedge_{i=n}^{p(n)} S_{\square,i}$$

Eine Belegung β erfüllt $\textsc{Start}_{\mathcal{M},w}(\overline{C})$ genau dann, wenn $\overline{C}$ die Startkonfiguration von $\mathcal{M}$ auf Eingabe w repräsentiert.

Abschließend definieren wir noch

$$\textsc{Acc-Conf}(\overline{C}) := \textsc{Conf}(\overline{C}) \wedge \bigvee_{q \in F_a} Q_q$$

Offenbar erfüllt eine Belegung β die Formel $\textsc{Acc-Conf}(\overline{C})$ genau dann, wenn $\overline{C}$ eine akzeptierende Stopkonfiguration von $\mathcal{M}$ repräsentiert.

Wir können nun die verschiedenen Formeln zu einer Gesamtformel zusammenfügen. Dazu verwenden wir folgende Variablen:

$Q_{q,t}$: Für alle $q \in Q$, $0 \leqslant t \leqslant p(n)$
Bedeutung: $\mathcal{M}$ ist zur Zeit t im Zustand $q \in Q$

$P_{i,t}$: Für alle $0 \leqslant i, t \leqslant p(n)$
Bedeutung: Zur Zeit t ist der Kopf an Position i

$S_{a,i,t}$: Für alle $a \in \Sigma \dot\cup \{\square\}$ und $0 \leqslant i, t \leqslant p(n)$
Bedeutung: *Zur Zeit t enthält Bandposition i das Symbol a*

In der folgenden Formel schreiben wir kurz

$$\overline{C}_t := \{Q_{q,t},\ P_{i,t},\ S_{a,i,t} : \quad q \in Q, 0 \leqslant i \leqslant p(n), \quad a \in \Gamma\}$$

Zu einer gegebenen 1-Band NTM $\mathcal{M} := (Q, \Sigma, \Gamma, q_0, \Delta, F_a, F_r)$, die $\mathcal{L}$ in Zeit $p(n)$ akzeptiert und einer Eingabe $w := w_0 \dots w_{n-1}$ konstruieren wir nun die Formel $\varphi_{\mathcal{M},w}$ als

$$\begin{array}{ll} \textsc{Start}_{\mathcal{M},w}(\overline{C}_0) \ \wedge & C_0 \text{ kodiert Startkonfiguration} \\ \bigvee_{0 \leqslant t \leqslant p(n)} \left\{ \begin{array}{l} \textsc{Acc-Conf}(\overline{C}_t) \ \wedge \\ \bigwedge_{0 \leqslant i < t} \textsc{Next}(\overline{C}_i, \overline{C}_{i+1}) \end{array} \right. & \begin{array}{l} \mathcal{M} \text{ akzeptiert nach } t \text{ Schritten} \\ \overline{C}_0, ..., \overline{C}_t \text{ kodieren einen Berechnungspfad} \end{array} \end{array}$$

Man beachte hierbei, dass eine akzeptierende oder verwerfende Stopkonfiguration keinen Nachfolger hat. Aus der Konstruktion folgt somit unmittelbar folgende Behauptung, die den Beweis abschließt.

Behauptung 7. $w \in \mathcal{L} \iff \varphi_{\mathcal{M},w}$ erfüllbar.

3.4. Der DPLL Algorithmus

Wir werden als nächstes einen auf der Resolution basierenden Algorithmus kennen lernen, um Formeln auf Unerfüllbarkeit zu testen. Der DPLL-Algorithmus ist benannt nach seinen Erfindern, *Davis, Putnam, Logemann, Loveland*. Der Algorithmus kombiniert *backtracking* mit *Einheitsresolution*, bei der nur Resolventen gebildet werden können, wenn mindestens eine der Klauseln nur ein Literal enthält. Varianten des DPLL-Algorithmus bilden die Basis der meisten aktuellen SAT-Löser, wie z.B. BerkMin, zChaff, etc.

Der Algorithmus arbeitet auf Formeln $\varphi := \bigwedge_{i=1}^{n} \left(\bigvee_{j=1}^{n_i} L_{i,j} \right)$ in konjunktiver Normalform. Wiederum repräsentieren wir solche Formeln φ als Klauselmenge $\Phi := \big\{\{L_{1,1}, \dots, L_{1,n_1}\}, \dots, \{L_{n,1}, \dots, L_{n,n_n}\}\big\}$.

Der Algorithmus ist in Abbildung 3.1 angegeben. Die ersten beiden Zeilen enthalten die Abbruchbedingungen. Wenn Φ leer ist, ist die Klauselmenge offensichtlich erfüllbar. Enthält Φ hingegen die leere Klausel, ist sie unerfüllbar.

Der nächste Schritt ist das sogenannte Boolean Constraint Propagation, oder auch Einheitsresolution. Wenn Φ eine Klausel enthält, die nur aus einem Literal L besteht, dann muss offenbar jede Φ erfüllende Belegung auch L mit 1 belegen. Wir können also alle Klauseln entfernen, die L enthalten, da diese schon erfüllt sind. Aus allen anderen Klauseln entfernen wir $\bar{L}$, da diese nicht von $\bar{L}$ erfüllt werden können.

Algorithmus DPLL(Φ)

Eingabe. Klauselmenge $\Phi := \big\{\{L_{1,1}, \ldots, L_{1,n_1}\}, \ldots, \{L_{n,1}, \ldots, L_{n,n_n}\}\big\}$

Ausgabe. entscheide, ob Φ erfüllbar ist.

Algorithmus. Wenn Φ leer ist, gib *erfüllbar* zurück.
Wenn Φ die leere Klausel $\square$ enthält, gib *unerfüllbar* zurück.

Unit Clause/Boolean Constraint Propagation (bcp):
Solange Φ eine *Einheitsklausel* $\{L\}$ enthält „setze $L := 1$“
d.h. entferne alle Klauseln, die L enthalten und
entferne $\overline{L}$ aus allen anderen Klauseln

Anderenfalls, wähle ein Literal L, das in Φ vorkommt.

Ergibt $DPLL(\Phi \cup \{\{L\}\})$ oder $DPLL(\Phi \cup \{\{\overline{L}\}\})$ *erfüllbar*, gib *erfüllbar* zurück, sonst *unerfüllbar*.

Abbildung 3.1.: Des DPLL Algorithmus

Falls keine Einheitsresolution mehr ausgeführt werden kann, wählt der Algorithmus als nächstes ein beliebiges Literal L. In den rekursiven Aufrufen belegt er einmal das Literal L mit 1 und das andere Mal mit 0. Dies wird in Abbildung 3.1 dadurch gelöst, dass einmal die Klausel $\{L\}$ und einmal die Klausel $\{\bar{L}\}$ zur Klauselmenge hinzugenommen wird. Diese werden dann sofort im rekursiven Aufruf durch Einheitsresolution entfernt.

Es besteht ein enger Zusammenhang zwischen Resolutionswiderlegungen und Widerlegungen einer Formel durch den DPLL-Algorithmus. Dazu stellt man einen Lauf des DPLL-Algorithmus als Entscheidungsbaum dar. Hat der Baum die Höhe h, so gibt es auch eine (baumartige) Resolutionswiderlegung gleicher Höhe. Der DPLL-Algorithmus ist also nicht „schneller“ als die Resolution.

In praktischen Implementierungen des Algorithmus' werden verschiedene Optimierungen verwendet.

Auswahlregel: Welches Literal wird beim Verzweigen gewählt.

Conflict Analysis: Bei einem Backtracking Schritt wird der Grund der Unerfüllbarkeit (Conflict) analysiert und intelligenter zurück gesprungen.

Clause Learning: Aus der Konfliktanalyse werden neue Klauseln generiert, die zur Formel hinzugenommen werden.

Dies soll verhindern, dass in den gleichen Konflikt mehrfach hineingelaufen wird.

Random restarts: Bisweilen wird ein DPLL-Lauf abgebrochen und neu angefangen. Die gelernten Klauseln bleiben erhalten.

3.5. Eine Anwendung aus der künstlichen Intelligenz

Eine mögliche Anwendung der Aussagenlogik sind sogenannte *constraint satisfaction Probleme* (CSPs).

Definition 3.29 (Constraint Satisfaction Problem) Ein *Constraint Satisfaction Problem* besteht aus

- einer Menge V von Variablen
- einem Wertebereich D
- und einer Menge C von *constraints*, d.h. Tupeln $((v_1, \ldots, v_n), R)$ mit $v_1, \ldots, v_n \in V$ und $R \subseteq D^n$

Eine *Lösung* eines CSPs ist eine Abbildung $f : V \to D$, so dass für alle constraints $((v_1, \ldots, v_n), R) \in C$ gilt: $(f(v_1), \ldots, f(v_n)) \in R$. ⊣

CSPs sind eine in der künstlichen Intelligenz viel untersuchte Problemklasse, da viele natürliche Probleme als CSPs modelliert werden können. Ein einfaches Beispiel ist die Stundenplangenerierung.

Beispiel 3.30 Eine (kleine) Schule mit nur einem Klassenzimmer hat drei Lehrer die jeweils ein Fach Deutsch (D), Mathematik (M) und Physik (P) unterrichten und zwei Klassen, die in jedem der drei Fächer unterrichtet werden sollen.

Als *Variablenmenge* können wir die Menge $V := \{D_1, D_2, M_1, M_2, P_1, P_2\}$ wählen. Als *Wertebereich* D wählen wir die Menge der möglichen Unterrichtszeiten $8, 9, 10, ..., 13$.

Die Bedeutung ist, dass wenn wir z.B. eine Variable D_1 auf eine Zeit 10 abbilden, dann soll der Deutschlehrer die Klasse 1 um 10 Uhr unterrichten.

In der Constraint-Menge C müssen wir nun alle Einschränkungen auflisten, die gewährleisten, dass eine Lösung, d.h. eine Abbildung von V nach D, auch eine realisierbare Lösung ist.

Constraint-Menge C. Sei dazu $R_{\neq} := \{(t, t') \in D \times D : t \neq t'\}$. Wir nehmen folgende Constraints in die Menge C auf.

- $((D_1, D_2), R_{\neq}), \quad ((M_1, M_2), R_{\neq}), \quad ((P_1, P_2), R_{\neq})$
  „Kein Lehrer hält zwei Klassen gleichzeitig“
- Für alle $i = 1, 2$: $((M_i, D_i), R_{\neq}), \quad ((M_i, P_i), R_{\neq}), \quad ((P_i, D_i), R_{\neq})$
  “Keine Klasse hat zwei Stunden gleichzeitig”

Eine Lösung des CSPs ist nun eine Abbildung $f : V \to D$ die alle Constraints erfüllt. D.h. die Abbildung weist keinem Lehrer zwei Klassen gleichzeitig und keiner Klasse zwei Unterrichtsstunden zur selben Zeit zu. Es ist also eine mögliche Lösung des Stundenplanproblems. ⊣

Beispiel 3.31 Ein weiteres Beispiel ist das Spiel Sudoku. Ein Sudoku besteht aus einer 9×9 Felder großen Tabelle. In jedem Feld kann eine Zahl zwischen 1 und 9 stehen. Zu Beginn sind überlicherweise nur wenige Felder gefüllt.

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

Ziel ist es, die fehlenden Positionen so mit den Zahlen $1, ..., 9$ zu füllen, dass in jeder Zeile und jeder Spalte und in jedem Block jede der Zahlen $1, ..., 9$ genau einmal vorkommt.

Wir formalisieren ein gegebenes Sudoku in der Aussagenlogik wie folgt.

Variablen. Als Variablen verwenden wir $X^c_{i,j}$ für alle $1 \leqslant i, j, c \leqslant 9$. Die Idee ist, dass $X^c_{i,j}$ mit 1 belegt werden soll, wenn in Position (i, j) die Zahl c steht.

Generische Formeln. Als nächstes geben wir allgemeine Formeln an, die erzwingen werden, dass jede erfüllende Belegung β einer gültigen Beschriftung des Sudokus entsprechen.

- $\bigwedge_{1 \leqslant i,j \leqslant 9} \bigvee_{c=1}^{9} X_{i,j}^{c}$
  Bedeutung: „jede Position erhält eine Zahl“
- $\bigwedge_{1 \leqslant i,j \leqslant 9} \bigwedge_{1 \leqslant c < c' \leqslant 9} \neg(X_{i,j}^{c} \wedge X_{i,j}^{c'})$
  Bedeutung: „jede Position erhält höchstens eine Zahl“
- Für alle $1 \leqslant i, c \leqslant 9$ und $1 \leqslant j < j' \leqslant 9$: $\neg(X_{i,j}^{c} \wedge X_{i,j'}^{c})$
  Bedeutung: „Keine Zeile enthält zwei gleiche Einträge“
- Für alle $1 \leqslant j, c \leqslant 9$ und $1 \leqslant i < i' \leqslant 9$: $\neg(X_{i,j}^{c} \wedge X_{i',j}^{c})$
  Bedeutung: „Keine Spalte enthält zwei gleiche Einträge“
- Die entsprechende Aussage für jeden Block: $\neg(X_{i,j}^{c} \wedge X_{i',j'}^{c})$ für alle $1 \leqslant c \leqslant 9$ und i, j, i', j' mit
  - $(i, j) \neq (i', j')$ und
  - $i \div 3 = j \div 3 = i' \div 3 = j' \div 3$.

Sei Φ die Menge alle bisher definierten Formeln. Die Menge Φ beschreibt ganz allgemein Lösungen für Sudokus. D.h., eine erfüllende Belegung entspricht einer gültigen Beschriftung eines Sudokus. Die Menge aller erfüllenden Belegungen der Formelmenge entspricht also der Menge aller korrekt ausgefüllten Sudokus.

Möchte man nun die Lösung für ein konkret gegebenes, teilausgefülltes Sudoku finden, muss man noch explizit die vorbesetzten Felder mit in die Formelmenge aufnehmen. Steht also, wie in dem Beispiel vorher, auf Position $(2, 1)$ eine 1, so nimmt man noch die Formel $X_{2,1}^{1}$ zur Formelmenge hinzu. Dies erzwingt, dass jede erfüllende Belegung die Position $(2, 1)$ korrekt beschriftet.⊣