Aussagenlogik

3.0 VU Formale Modellierung

C.Fermüller, B.Gramlich, A.Leitsch, M.Oswald, G.Salzer
Institut für Computersprachen, Technische Universität Wien

Oktober 2011

Dieses Dokument geht auf das Skriptum der Lehrveranstaltung „Theoretische Informatik und Logik“ aus dem Sommersemester 2011 zurück. Die Notation wurde an jene der Lehrveranstaltung „Formale Modellierung“ angepasst. Der Inhalt deckt sich weitgehend, aber nicht ganz: In der Vorlesung bzw. auf den Folien wurden einige weitere Themen behandelt.

1 Einleitung

Logik ist eine Grundlagendisziplin, die sich mit den allgemeinen Prinzipien korrekten Schließens beschäftigt. Hier wird uns nur die *mathematische* (auch: *formale*, bzw. *symbolische*) Logik interessieren, die für diese Aufgabe formale *Kalküle*[1]bereitstellt und analysiert. Weiters sei gleich vorweg bemerkt, dass in dieser Lehrveranstaltung nur ein kleiner Ausschnitt der Themen, Methoden und Ergebnisse der mathematischen Logik, die für die Informatik relevant sind, präsentiert werden wird.

Mathematische Logik taucht in sehr vielen Gebieten der Informatik in unterschiedlicher Weise auf. Zu den prominentesten Anwendungsgebieten zählen: logik-orientierte Programmierung (z.B.: PROLOG), Programmverifikation, Semantik von Programmiersprachen, Spezifikationssprachen, wissensbasierte Systeme und automatisches Beweisen. Ganz allgemein sind die Analyse der Beziehung von Syntax und Semantik und das Augenmerk auf algorithmische Aspekte des Schließens derart grundlegend, dass oft folgende Analogie zum Verständnis der Rolle der Logik zitiert wird: Die Logik steht zur Informatik wie die Mathematik zur Physik; ähnlich wie die Mathematik oft als ‘Sprache der Physik’ bezeichnet wird, ist die Logik als das Fundament zentraler Aufgaben der Informatik zu betrachten.

[1]Das Wort *Kalkül* hat zwei unterschiedliche Bedeutungen. *Das* Kalkül ist gleichbedeutend mit *Berechnung* oder *Überlegung*; in diesem Sinn tritt das Wort etwa in der Redewendung „ins Kalkül ziehen“ auf. *Der* Kalkül hingegen bezeichnet laut Duden ein *System von Regeln zur schematischen Konstruktion von Figuren in der Mathematik*. Wir verwenden das Wort hier in der zweiten Bedeutung, also mit männlichem Artikel.

Aussagenlogik

Es gibt so viele unterschiedliche Logiken (im Sinn von: Kalkülen mit entsprechender Semantik) und verschiedene Themen in diesem Bereich, dass es unmöglich ist, die Disziplin umfassend in wenigen Worten zu definieren. In allen Fällen wird jedoch analysiert, welche Aussagen bzw. Schlussweisen sich allein auf Grund ihrer *Form* als wahr, gültig bzw. korrekt erkennen lassen.

1.1 Beispiel. Wenn ein Kind weiß, dass es entweder Pudding oder Torte als Nachspeise gibt und ihm außerdem gesagt wird, dass es keinen Pudding mehr gibt, so kann es daraus schließen, dass es Torte zur Nachspeise gibt. Vereinbaren wir A als Abkürzung für „Es gibt Pudding als Nachspeise“ und B als Abkürzung für „Es gibt Torte als Nachspeise“, so lässt sich dieser Schluss in üblicher Notation wie folgt darstellen:

$$\frac{A \vee B \qquad \neg A}{B}$$

Entscheidend ist, dass die Korrektheit (Zulässigkeit) dieser Argumentationsform offensichtlich nicht vom konkreten Inhalt der Aussagen A und B abhängt. So könnte A z.B. auch für „Das Programm terminiert innerhalb von 10 Sekunden“ und B für „Das Programm terminiert nie“ stehen. Weiters ist die Korrektheit des Schlusses unabhängig davon, ob die Aussagen A und B wahr oder falsch sind. □

Genau genommen haben wir uns in diesem sehr einfachen Beispiel bereits einer spezifischen Logik – nämlich der sogenannten *klassischen* Aussagenlogik – bedient. In der klassischen Aussagenlogik[2] geht man davon aus, dass jede Aussage entweder wahr oder falsch ist. Man sagt auch: Jeder Aussage wird einer der zwei *Wahrheitswerte* „wahr“ bzw. „falsch“ zugeordnet. Außerdem betrachtet man nur Aussagen, die entweder nicht mehr weiter zerlegt werden (sogenannte „atomare Aussagen“, für die einen nur interessiert, ob sie wahr oder falsch sind) oder die mittels der logischen *Junktoren* „nicht“ ($\neg$), „und“ ($\wedge$), „oder“ ($\vee$), bzw. „wenn ... dann“ ($\supset$) aus anderen Aussagen zusammengesetzt sind. Weiters setzt man das Prinzip der *Wahrheitsfunktionalität* (Kompositionalitätsprinzip) voraus: Die Wahrheit bzw. Falschheit einer zusammengesetzten Aussage hängt nur von der Wahrheit bzw. Falschheit ihrer unmittelbaren Teilaussagen ab.

Prädikatenlogik

Für die meisten Anwendungen ist die skizzierte logische Analyse von Aussagen noch zu grob. Man abstrahiert zwar weiterhin vom konkreten Inhalt der Aussagen, aber versteht atomare Aussagen als Behauptungen der Form „Die Eigenschaft P trifft auf das Objekt x zu“ bzw. allgemeiner: „Die Objekte $x_1, \ldots, x_n$ stehen in der (n-stelligen) Relation P zueinander“. Im ersten Fall nennt man P ein einstelliges (monadisches) Prädikat; im zweiten Fall ist P ein n-stelliges Prädikat. Neben den oben genannten Junktoren ist es nun auch zweckmäßig, die beiden *Quantoren* „für alle“ (Allquantor $\forall$) und „für (mindestens)

[2] Für „Aussage“ verwendet man in diesem Zusammenhang auch den Fachausdruck „Proposition“; daher ist *propositionale Logik* synonym zu Aussagenlogik.

ein“ (Existenzquantor $\exists$) zur Bildung zusammengesetzter Aussagen zu verwenden. Man spricht dann von klassischer *Prädikatenlogik* (bzw. Quantorenlogik) erster Stufe.[3]

1.2 Beispiel. Monadische Prädikate reichen aus, um die sogenannten Syllogismen darzustellen, die bereits in der Antike diskutiert wurden. Das bekannteste Beispiel eines Syllogismus ist wohl folgender Schluss:

$$\frac{\text{Sokrates ist ein Mensch.} \quad \text{Alle Menschen sind sterblich.}}{\text{Also ist auch Sokrates sterblich.}}$$

Die beiden Aussagen über dem Strich heißen *Prämissen*, die Aussage darunter *Konklusion* des Schlusses. Gemäß unserer Terminologie ist dabei „Sokrates“ ein Objekt und „ist sterblich“ bzw. „ist ein Mensch“ sind Beispiele für (monadische) Prädikate. In logischer Notation und in Verwendung naheliegender Abkürzungen lässt sich daher der Schluss als

$$\frac{M(s) \quad (\forall x)M(x) \supset S(x)}{S(s)}$$

darstellen. Entscheidend ist wiederum, dass es nur um die Form der Aussagen geht und dass die Zulässigkeit (Korrektheit, allgemeine Gültigkeit) des Schlusses keineswegs von der Wahrheit der Prämissen abhängt. Dies lässt sich einsehen, indem man beispielsweise für s die Zahl 2 setzt, für S das Prädikat „ist ungerade“ und für M das Prädikat „ist größer als 5“. Unter dieser Interpretation sind offensichtlich die beiden Prämissen und die Konklusion falsch. Der Schluss jedoch bleibt zulässig. Ein Schluss heißt nämlich (per Definition) zulässig, falls folgendes gilt: Immer dann, wenn alle Prämissen wahr sind, ist auch die Konklusion wahr. □

Der oben diskutierte Schluss lässt sich bei näherer Betrachtung in noch elementarere zerlegen, nämlich in den *Modus ponens* (mp) – aus A und der Aussage „wenn A dann B“ folgt B – und in die Spezialisierungsregel (sp) – wenn A für alle Objekte x gilt, dann gilt A auch für ein spezielles Objekt c:

$$\frac{A \quad A \supset B}{B}\ \text{mp} \qquad \frac{(\forall x)A[x]}{A[c]}\ \text{sp}$$

A und B sind dabei Platzhalter für beliebige Formeln, wobei die Schreibweise $A[x]$ anzeigt, dass A eine Variable x enthalten kann, die in $A[c]$ durch c ersetzt wurde. Die ursprüngliche Überlegung lässt sich mit Hilfe dieser beiden Regeln beweisen:

$$\frac{M(s) \qquad \dfrac{(\forall x)M(x) \supset S(x)}{M(s) \supset S(s)}\ \text{sp}}{S(s)}\ \text{mp}$$

Beim Anwenden der Spezialisierungsregel entspricht $M(x) \supset S(x)$ dem Platzhalter $A[x]$; beim Modus ponens übernimmt $M(s)$ die Rolle von A und $S(s)$ jene von B. Man spricht

[3] Höherstufige Logiken entstehen, wenn man auch Eigenschaften von Prädikaten, von Relationen zwischen Relationen, von Relationen zwischen Relationen zwischen Relationen usw. in Betracht zieht.

von einer *Instantiierung* der Regel: Z.B. fungiert $M(s)$ hier als *Instanz* der linken Prämisse des Modus ponens.

Die klassische Prädikatenlogik erster Stufe ist zweifellos der bedeutendste Formalismus der Logik. Die Aussagenlogik lässt sich als Spezialfall der Prädikatenlogik verstehen: Die atomaren Aussagen kann man als 0-stellige Prädikate betrachten; mit den Objekten verschwinden auch die Quantoren und es bleiben nur die Junktoren.[4]

Logische Kalküle

Hat man eine Formelsprache (Syntax) gewählt und die Interpretation der Formeln (ihre Semantik) festgelegt, möchte man die Gültigkeit bestimmter Formeln überprüfen können. Im einfachsten Fall reicht es, eine Formel bzgl. der Semantik auszuwerten. Etwa ist $s(0)+s(s(0)) = s(s(s(0)))$ durch Ausrechnen unschwer als wahr zu erkennen, falls s als die Funktion „addiere 1“, 0 als Null, $+$ als Addition und $=$ als Gleichheit interpretiert wird. Die Gültigkeit der Formel $(\forall n{\geq}0)(\exists k)(n{=}2 \cdot k \vee n{=}2 \cdot k + 1)$ hingegen ist schwieriger nachzuweisen, da die Semantik des Allquantors erfordert, für alle der unendlich vielen nicht-negativen Zahlen n ein k zu finden, sodass entweder $n{=}2 \cdot k$ oder $n{=}2 \cdot k + 1$ erfüllt ist. Daher versucht man Regelsysteme – sogenannte Kalküle – zu finden, mit deren Hilfe eine Formel durch eine *endliche* Anzahl von Regelanwendungen bewiesen werden kann. Dabei muss das Regelsystem zumindest *korrekt* sein: Jede mit den Regeln bewiesene Formel soll tatsächlich gültig sein. Manche Regelsysteme sind darüber hinaus auch *vollständig*: *Alle* gültigen Formeln lassen sich mit den Regeln beweisen.

Ein zentraler Begriff im Bereich der Kalküle ist der der *Ableitung*. Darunter versteht man allgemein eine Kette oder einen Baum von syntaktischen Objekten (z.B. von aussagenlogischen Formeln), die bzw. der durch systematische Regelanwendungen aus gegebenen Initialobjekten („Axiomen“ bzw. „Annahmen“) hervorgeht. Ableitungen entsprechen formalen Beweisen. Im Prinzip muss jeder exakte Nachweis der Richtigkeit einer Behauptung (sprich: jeder Beweis) als Ableitung in einem geeigneten Kalkül formalisierbar sein. Selbstverständlich ist es für die menschliche Kommunikation nicht zweckmäßig, jeden Beweis vollständig zu formalisieren. Anderseits sollte klar sein, dass immer dann, wenn Computersysteme Beweise finden oder überprüfen sollen, logische Kalküle unverzichtbar sind.

Logik in der Programmierung

Die zahlreichen Anwendungen der Logik in der Informatik sind oft sehr unterschiedlicher Natur. Wie bereits erwähnt, gibt es Programmiersprachen, die direkt auf bestimmten Kalkülen basieren. PROLOG, LISP und ML sind Beispiele dafür. Doch auch in allen üblichen prozeduralen Programmiersprachen tauchen Konzepte der mathematischen Logik mehrfach auf.

1.3 Beispiel. Der Wahrheitswert einer logischen Formel entscheidet darüber, ob in einer `if`-Anweisung der `then`- oder der `else`-Zweig gewählt bzw. wie oft eine `while`- oder

[4] Man spricht deshalb auch von Junktorenlogik.

`repeat`-Schleife durchlaufen wird. Will man in einer `if`-Anweisung die beiden Zweige ohne Änderung des Programmverhaltens vertauschen, helfen die algebraischen Regeln der Aussagenlogik bei der korrekten Umformung der Bedingung; etwa wird

$$\texttt{if } (x = 0) \wedge (y > 1) \texttt{ then } P_1 \texttt{ else } P_2$$

durch Anwendung der aussagenlogischen de-Morgan-Regel $\neg(A \wedge B) = \neg A \vee \neg B$ zum äquivalenten Programmstück

$$\texttt{if } (x \neq 0) \vee (y \leq 1) \texttt{ then } P_2 \texttt{ else } P_1 \ .$$

Ähnliches gilt für `while`-Schleifen, die mindestens einmal durchlaufen werden sollen und daher in `repeat`-Schleifen umgewandelt werden. Etwa transformiert sich die Schleife `while` $(x = 0) \wedge (y > 1)$ `do` P zu `repeat` P `until` $(x \neq 0) \vee (y \leq 1)$. □

1.4 Beispiel. Ausgangspunkt seien zwei Intervalle $[a, b]$ und $[c, d]$, etwa Aufnahmezeiten eines Videorecorders (erste Aufnahme von Zeit a bis Zeit b, zweite von c bis d). Wir nehmen an, dass $a < b$ und $c < d$ gilt.

Die Aufgabe besteht nun darin, eine Warnung auszugeben, wenn die Intervalle überlappen, wenn also eine Terminkollision vorliegt:

$$\texttt{if } \langle \textit{Bedingung} \rangle \texttt{ then write „Kollision!“}$$

Um eine geeignete $\langle \textit{Bedingung} \rangle$ zu finden, könnte man in einem ersten Ansatz alle Möglichkeiten auflisten, wie sich zwei Intervalle überlappen können:

1. Das erste Intervall enthält das zweite: $a \leq c$ und $d \leq b$.
2. Das zweite Intervall enthält das erste: $c \leq a$ und $b \leq d$.
3. Der Anfangspunkt des zweiten Intervalls liegt im ersten Intervall: $a \leq c$ und $c \leq b$.
4. Der Anfangspunkt des ersten Intervalls liegt im zweiten Intervall: $c \leq a$ und $a \leq d$.
5. Der Endpunkt des zweiten Intervalls liegt im ersten Intervall: $a \leq d$ und $d \leq b$.
6. Der Endpunkt des ersten Intervalls liegt im zweiten Intervall: $c \leq b$ und $b \leq d$.

Offensichtlich enthält die Disjunktion dieser sechs Konjunktionen von je zwei Vergleichen viel Redundanz. Insbesonders werden die ersten beiden Bedingungen bereits von den übrigen vier Bedingungen (unter der Voraussetzung $a < b$ und $c < d$) impliziert. Aber auch die direkte Umsetzung der Bedingungen 3 bis 6 ist keineswegs optimal.

Ein einfacherer Lösungansatz besteht darin, zuerst eine Bedingung für *Überlappungsfreiheit* aufzustellen und diese dann zu negieren. Eine notwendige und hinreichende Bedingung für Überlappungfreiheit ist:

Das erste Intervall endet vor dem zweiten $(b < c)$
oder das zweite vor dem ersten $(d < a)$.

Die gesuchte Bedingung für das Vorliegen einer Kollision lässt sich also schreiben als $\neg(b < c \vee d < a)$ bzw. durch Hineinziehen der Negation als $b \geq c \wedge d \geq a$.

Beim Aufstellen, Überprüfen und vor allem beim korrekten Vereinfachen und Umformen derartiger Bedingungen helfen die Grundgesetze der Logik. Im letzten Schritt wurde z.B. die Regel $\neg(A \vee B) = \neg A \wedge \neg B$ angewendet. Viele der Verfahren lassen sich auch leicht automatisieren. So könnte man mit einem geeigneten Werkzeug kontrollieren, ob sich die Formel aus dem ersten Lösungsansatz tatsächlich zu jener des zweiten Ansatzes vereinfach lässt. □

1.5 Beispiel. Zur Verifikation von Programmen werden *Zusicherungen* (assertions) sowie *Vor-* und *Nach-Bedingungen* (pre- und post-conditions) eingesetzt. Zusicherungen sind Bedingungen, die jedesmal ausgewertet werden, wenn sie im Zuge des Programmablaufs erreicht werden. Treffen sie zu, setzt das Programm mit der nächsten Anweisung fort; andernfalls wird eine Warnung ausgegeben, die signalisiert, dass das Programm vom vorhergesehenen Verhalten abweicht. Im Gegensatz zu den anderen Programmkonstrukten beeinflussen Zusicherungen nicht die Berechnungen. Im folgenden Programm kontrolliert die Zusicherung `assert`$(y \geq 0)$, dass der Wert von y wie vom Programmautor vorgesehen nie kleiner als Null wird.

$\langle x = a \wedge y = b \wedge y \geq 0 \rangle$ ⇐ Vorbedingung
`while` $y > 0$ `do`
  $x \leftarrow x - 1$;
  $y \leftarrow y - 1$;
  `assert`$(y \geq 0)$; ⇐ Zusicherung
  `endwhile`;
$\langle x = a - b \wedge y = 0 \rangle$ ⇐ Nachbedingung

Vor- und Nachbedingungen sind Formeln, die die erlaubten Eingabedaten bzw. die gewünschten Ergebnisse charakterisieren. Sie bilden zusammen mit dem Programm eine Korrektheitsaussage, die wahr ist, falls das Programm für alle Eingaben, die die Vorbedingung erfüllen, Ausgabewerte berechnet, auf die die Nachbedingung zutrifft. Im Beispielprogramm behaupten die beiden Bedingungen, dass x am Programmende den Wert $a - b$ hat, falls zu Beginn x den Wert a und y den Wert b hat. Der Beweis von Korrektheitsaussagen ist prinzipiell schwieriger als die Überprüfung von Zusicherungen: Im ersten Fall müssen *alle* möglichen Variablenwerte untersucht werden, im zweiten Fall wird die Formel nur für die momentanen Werte berechnet. Zum Nachweis der Gültigkeit von Korrektheitsaussagen werden Kalküle wie der sogenannte *Hoare-Kalkül*[5] eingesetzt. □

Algorithmische Lösbarkeit

Die folgenden drei Beispiele beschreiben Resultate, die auf den ersten Blick abstrakt und theoretisch wirken. Bedenkt man aber, dass praktisch jede Programmiersprache die

[5] benannt nach dem britischen Informatiker Sir Charles Antony Richard Hoare, der unter anderem das Sortierverfahren *Quicksort* erfunden hat.

Datentypen der Reals und Integers kennt, dass die Spezifikation des Programmverhaltens Quantoren erfordert und dass die Überprüfung von Programmeigenschaften auf den Nachweis der Gültigkeit von Formeln hinausläuft, haben diese Resultate unmittelbare Auswirkungen auf Gebiete wie die Programmverifikation.

1.6 Beispiel. Die Frage nach der Lösbarkeit eines Systems von Gleichungen kann als existentiell quantifizierte Formel aufgefasst werden, nämlich als die Behauptung: „Es gibt Werte für die Variablen, sodass die Gleichungen erfüllt sind." Für lineare Gleichungen über den reellen Zahlen ist aus der Schule bekannt, dass es mit dem Gaußsche Eliminationsverfahren ein Regelsystem gibt, das korrekt und vollständig für das Lösbarkeitsproblem ist: Wenn das Verfahren die Lösbarkeit behauptet, ist das Gleichungssystem tatsächlich lösbar (Korrektheit), und wenn das Gleichungssystem lösbar ist, stellt das Eliminationsverfahren dies auch fest (Vollständigkeit). Darüberhinaus ist es sogar ein Entscheidungsverfahren: Auch wenn das Gleichungssystem nicht lösbar ist, liefert das Verfahren eine Antwort, nämlich „nicht lösbar".[6]

Im Fall der reellen Zahlen können diese Resultate verallgemeinert werden: Selbst wenn man beliebige (existentielle und universelle) Quantoren zulässt sowie nicht-lineare Gleichungen (Polynomgleichungen) erlaubt, gibt es ein korrektes und vollständiges Entscheidungsverfahren (Alfred Tarski, 1948). □

1.7 Beispiel. Diophantische Gleichungen sind Gleichungen mit ganzzahligen Koeffizienten, die über den ganzen bzw. den natürlichen Zahlen zu lösen sind. Für lineare diophantische Gleichungen gilt ähnliches wie für Gleichungen über den reellen Zahlen: Es gibt korrekte und vollständige Entscheidungsverfahren. Auch mit beliebigen Quantoren sind derartige Formeln, auch bekannt als „Presburger Arithmetik", immer noch lösbar (Mojzesz Presburger, 1929, D. C. Cooper 1972).

Anders sieht die Sache im Falle von diophantischen Polynomgleichungen aus (Hilberts 10. Problem). Falls diophantische Polynomgleichungen lösbar sind, lässt sich zwar eine Lösung durch systematisches Einsetzen aller möglichen Variablenwerte finden, es gibt aber prinzipiell kein Entscheidungsverfahren mehr: Die Unlösbarkeit eines Gleichungssystems lässt sich im Allgemeinen nicht feststellen, da es prinzipiell keinen korrekten, immer terminierenden Algorithmus für dieses Problem geben kann (Yuri Matiyasevich, 1970).

Noch schwieriger wird die Situation, wenn wir zusätzlich zu den Polynomen beliebige Quantoren zulassen (Arithmetik der natürlichen Zahlen mit Addition und Multiplikation): Die Gültigkeit derartiger Formeln ist nicht nur unentscheidbar, sondern es gibt nicht einmal mehr ein vollständiges Regelsystem (Kurt Gödel, 1930). In anderen Worten: aus prinzipiellen Gründen muss jedes korrekte Verfahren zur Überprüfung der Gültigkeit von arithmetischen Formeln sowohl bei gültigen als auch bei widerlegbaren Formeln gelegentlich passen, d.h., es liefert bei beiden Antwortmöglichkeiten nicht immer eine Antwort.
□

[6] Das Gaußsche Eliminationsverfahren leistet sogar noch mehr: Im Falle der Lösbarkeit liefert es eine endliche Darstellung aller Lösungen.

1.8 Beispiel. Die Gültigkeit aussagenlogischer Formeln in der klassischen Logik ist trivialerweise durch Ausprobieren immer feststellbar, da es nur endlich viele Belegungen der Variablen mit wahr/falsch gibt und alle Junktoren (etwa Und-, Oder-Verknüpfungen) immer berechenbar sind. Das Problem hier ist vielmehr, die Gültigkeit *effizient* zu überprüfen.

Anders sieht es aus, wenn wir Quantoren zulassen, also die Prädikatenlogik betrachten. Zur Überprüfung der Gültigkeit gibt es korrekte und vollständige Kalküle, das Problem ist aber letztlich unentscheidbar, da wiederum aus prinzipiellen Gründen jedes korrekte Verfahren für manche (genauer: unendlich viele) nicht-gültige Formeln keine Antwort liefern kann. □

Nicht-klassische Logiken

Die bisher genannten Beispiele könnten dazu verführen, die *klassische* Logik als einzig relevanten logischen Formalismus zu betrachten. Doch gerade in der Informatik werden auch Anwendungen verschiedenster nicht-klassischer Logiken immer wichtiger. Anzahl, Formen und Anwendungen nicht-klassischer Logiken sind kaum mehr überschaubar. Grundsätzlich lassen sich zwei Typen von „Nicht-Klassizität“ unterscheiden: Einerseits werden *Erweiterungen* der Syntax und Semantik der klassischen Logik betrachtet; anderseits sind *Alternativen* zur klassischen Semantik (die manchmal auch syntaktischen Niederschlag finden) die Basis vieler wichtiger Logiken.

Zur ersten Gruppe zählt vor allem die große Familie der *Modallogiken*, die entstehen, wenn man in der Sprache der Logik ausdrücken will, *in welcher Weise* (z.B. wann, wie, wo, in welchem Kontext) eine Formel (klassisch) gültig ist.

1.9 Beispiel. Bei der Spezifikation dynamischer Systeme ist es meist sinnvoll, Aussagen der Form „Zu allen zukünftigen Zeitpunkten gilt A“, „Zum nachfolgenden Zeitpunkt gilt A“, „Irgendwann in der Vergangenheit galt A“ zu betrachten. Dabei nennt man die jeweiligen zeitlichen Qualifikationen zur Aussage A *temporale Modaloperatoren*. Syntaktisch sind Modaloperatoren nichts anderes als einstellige Junktoren. In üblicher Notation schreibt man die genannten Beispiele modaler Aussagen als $[F]A$, $\bigcirc A$, bzw. $\langle P\rangle A$. Das F steht dabei für Future, das P für Past; $\bigcirc$ wird als „Next-Operator“ bezeichnet. Offensichtlich kann man aber die Semantik dieser Junktoren nicht mehr durch klassische Wahrheitstafeln wiedergeben. Vielmehr wird für die *Temporallogik* der klassische Begriff des Modells einer Formel auf eine Weise erweitert, die die Struktur des Zeitverlaufs repräsentiert. Dies geschieht so, dass (um in unserem Beispiel zu bleiben) Formeln der Form $[F]A \supset \bigcirc A$ als logisch gültig erkennbar werden. □

1.10 Beispiel. Wie bereits erwähnt, wird der sogenannte Hoare-Kalkül als Formalismus zum Nachweis der Korrektheit von Programmen eigesetzt. Ohne hier auf Details einzugehen sei darauf hingewiesen, dass sich dieser Kalkül als Spezialfall einer bestimmten Modallogik – der sogenannten *dynamischen Logik* – verstehen lässt.

Die Grundaussagen des Hoare-Kalküls sind von der Form $P\ \{\alpha\}\ Q$, wobei P eine Vorbedingung und Q eine Nachbedingung zum Programm α ist. Die Rolle von α ist

dabei die eines Modaloperators „Nach Termination des Programmstücks α gilt ...“. In üblicher modallogischer Notation würde man also $P\ \{\alpha\}\ Q$ als Formel $P \supset [\alpha]Q$ der dynamischen Logik schreiben. □

Zur zweiten Gruppe der nicht-klassischen Logiken – den Alternativen zur klassischen Logik – geben wir ebenfalls zwei informatik-relevante Beispiele.

1.11 Beispiel. In vielen Anwendungen wissensbasierter Systeme hat man es mit *vager* oder *unbestimmter* Information zu tun. Insbesonders die Semantik natürlichsprachlicher Aussagen lässt sich oft durch die Zuordnung von einem der beiden Wahrheitswerte „wahr“ bzw. „falsch“ nicht adäquat formalisieren. Für Aussagen wie

„Diese Vorlesung ist wichtig.“
„Die Übungsaufgabe ist leicht.“
„der Student ist engagiert.“

ist es sinnvoll, ein Spektrum von Wahrheitswerten *zwischen* „absolut falsch“ und „absolut wahr“ vorzusehen. Als eine zweckmäßige Formalisierung der Semantik von einstelligen Prädikaten wie „ist wichtig“ oder „ist leicht“ verwendet man in der sogenannten *Fuzzy-Logik* entsprechend Funktionen, die jedem Objekt eine Zahl aus dem reell-wertigen Einheitsintervall $[0, 1]$ zuordnen. Ähnlich ordnet man für ein zweistelliges Prädikat wie „ärgert“ jedem geordneten Paar von Objekten (Individuen) eine relle Zahl zwischen 0 und 1 zu; 0 spielt dabei die Rolle von „absolut falsch“ und 1 die von „absolut wahr“. Die Semantik der üblichen aussagenlogischen Junktoren wird durch Funktionen über $[0, 1]$ festgelegt. Beispielsweise ist die Funktion $(x, y) \mapsto \min\{x, y\}$ für $x, y \in [0, 1]$ ein wichtiges Beispiel für die Semantik der Konjunktion ($\wedge$) in der Fuzzy-Logik. Diese Festlegung bedeutet: der Wahrheitswert einer Aussage der Form $A \wedge B$ ist gleich dem Minimum der Wahrheitswerte der Aussagen A bzw. B. Ähnlich bietet sich als Semantik der Negation ($\neg$) beispielsweise die Funktion $x \mapsto 1 - x$ an. Man beachte, dass hier die klassische Logik als Spezialfall der Fuzzy-Logik in Erscheinung tritt: Wenn statt dem Einheitsintervall $[0, 1]$ nur die Teilmenge $\{0, 1\}$ als Wahrheitswertmenge genommen wird, entsprechen die genannten Funktionen den vertrauten klassischen Wahrheitstafeln. □

1.12 Beispiel. Eine schwierige Aufgabe der Informatik ist das Design von Entwicklungsumgebungen, die es erlauben, beweisbar korrekte Software automatisch (oder zumindest semi-automatisch) aus der formalen Spezifikation der Aufgabenstellung zu gewinnen. Für dieses Paradigma der „Extraktion von Programmen aus Beweisen“ ist die sogenannte *intuitionistische Logik* (auch: *konstruktive Logik*) besser geeignet als die klassische Logik.

Die intuitionistische Logik ergibt sich, wenn man die Gültigkeit einer Formel nicht einfach mit der konstanten Zuordnung des Wahrheitswertes „wahr“ identifiziert, sondern mit der garantierten *Beweisbarkeit* der Formel. Solange man nur endliche Gegenstandsbereiche betrachtet, entsteht so noch kein Unterschied zur klassischen Logik. Aber bereits in der elementaren Arithmetik kann man nicht garantieren, dass zu jeder Aussage A

entweder A selbst oder deren Negation $\neg A$ beweisbar ist.[7] Unter der intuitionistischen Interpretation kann daher die aussagenlogische Formel $A \vee \neg A$ – im Gegensatz zur klassischen Logik – *nicht* logisch gültig sein.

Die intendierte Semantik der intuitionistischen Logik definiert die Gültigkeit zusammengesetzter Aussagen in Bezug auf (mögliche) Beweise für die jeweiligen Teilaussagen. Z.B. gilt $A \wedge B$ genau dann, wenn sowohl ein Beweis von A als auch einer von B vorliegt. Die Beweisbarkeit von $A \vee B$ muss ebenfalls von zwei Elementen bezeugt werden: Einem Indikator, der sagt, ob die linke oder die rechte Teilaussage beweisbar ist und einem Beweis der entsprechenden Teilaussage. Am interessantesten ist die Interpretation der Gültigkeit von $A \supset B$: Sie besagt, dass ein Algorithmus (Programm) vorliegt, der jeden gegeben Beweis von A in einen Beweis von B transformiert. Vereinbart man nun noch $\neg A$ als Abkürzung für $A \supset \bot$, wobei $\bot$ irgendeine fixe unbeweisbare Aussage ist, so ist damit die Semantik der intuitionistischen Aussagenlogik im Prinzip vollständig festgelegt. Sie lässt sich auch auf naheliegende Weise auf die Prädikatenlogik erweitern. □

2 Aussagenlogische Funktionen

Die klassische Aussagenlogik (Propositionallogik) formalisiert die Verknüpfung einfacher Ja/Nein-Aussagen durch Junktoren[8] wie Und, Oder, Nicht oder Wenn-dann. Formal werden Junktoren durch ein- bzw. zweistellige Funktionen über der Wahrheitswertmenge $\mathbb{B} = \{1, 0\}$ repräsentiert. Die folgende Tabelle gibt eine Übersicht über *Wahrheitstafeln*, die diese aussagenlogischen Funktionen definieren.

true	false
1	0
$\top$	$\bot$

x	not
1	0
0	1
	$\neg$

x	y	and	nand	or	nor	iff	xor	implies		if	
1	1	1	0	1	0	1	0	1	0	1	0
1	0	0	1	1	0	0	1	0	1	1	0
0	1	0	1	1	0	0	1	1	0	0	1
0	0	0	1	0	1	1	0	1	0	1	0
		$\wedge$	$\uparrow$	$\vee$	$\downarrow$	$\equiv$	$\not\equiv$	$\supset$	$\not\supset$	$\subset$	$\not\subset$

Die letzte Zeile der Tabelle gibt die Symbole an, mit denen wir diese Funktionen repräsentieren werden.

Die aussagenlogischen Funktionen können aus einer kleinen Zahl von Basisfunktionen zusammengesetzt werden.

2.1 Definition. Eine Menge von Funktionen heißt *funktional vollständig*, wenn alle anderen Funktionen durch sie ausgedrückt werden können.

[7] Aus den Sätzen von Gödel (1931) und Turing (1936) folgt auch, dass es für jeden hinreichend ausdrucksstarken, korrekten Kalkül Aussagen über die Termination oder Korrektheit von konkreten Programmen gibt, die weder beweisbar noch widerlegbar sind.

[8] Junktoren werden auch (aussgenlogische) „Operatoren“ oder „Konnektive“ genannt.

2.2 Beispiel. Jede aussagenlogische Funktion ist eindeutig durch ihre Wahrheitstafel beschrieben. Wie wir bei der Konstruktion von konjunktiven und disjunktiven Normalformen weiter unten (Abschnitt 6) sehen werden, kann jede Wahrheitstafel durch eine Formel beschrieben werden, die nur die Funktionen not, and und or benützt. Daher ist die Menge {not, and, or} funktional vollständig. Tatsächlich ist nur entweder and oder or notwendig, da sich die eine Funktion durch die andere ausdrücken lässt:

$$x \text{ and } y = \text{not}(\text{not } x \text{ or not } y) \qquad x \text{ or } y = \text{not}(\text{not } x \text{ and not } y)$$

Die Gültigkeit dieser Gleichungen kann mittels der Wahrheitstafelmethode, also durch „Ausprobieren“ aller möglichen Wertebelegungen für die Variablen, überprüft werden. Für die erste Gleichung erhalten wir:

x	y	not A	not B	not A or not B	not(not A or not B)	A and B
1	1	0	0	0	1	1
1	0	0	1	1	0	0
0	1	1	0	1	0	0
0	0	1	1	1	0	0

Um zu zeigen, dass eine Menge von Funktionen funktional vollständig ist, reicht es zu zeigen, dass alle Funktionen einer beliebigen anderen funktional vollständigen Menge ausgedrückt werden können, also etwa jene der Menge {not, and}. □

2.3 Beispiel. Der Funktion nand (negiertes Und) ist funktional vollständig[9], da not $x = x$ nand x und x and $y = (x \text{ nand } y)$ nand $(x \text{ nand } y)$. □

3 Syntax und Semantik der klassischen Aussagenlogik

3.1 Definition (Syntax aussagenlogischer Formeln). Die Menge $\mathcal{A}$ der aussagenlogischen Formeln ist die kleinste Menge, für die gilt:

(A1) $\mathcal{V} = \{A, B, C, \ldots, A_0, A_1, \ldots\} \subseteq \mathcal{A}$

(A2) $\{\top, \bot\} \subseteq \mathcal{A}$

(A3) $\neg F \in \mathcal{A}$, wenn $F \in \mathcal{A}$.

(A4) $(F * G) \in \mathcal{A}$, wenn $F, G \in \mathcal{A}$ und $* \in \{\wedge, \uparrow, \vee, \downarrow, \equiv, \not\equiv, \supset, \subset\}$.

Die Elemente von $\mathcal{V}$ heißen *aussagenlogische Variablen*. Die Formeln $\top$ und $\bot$ heißen *aussagenlogische Konstanten*. Aussagenlogische Variablen und Konstanten heißen auch *atomare aussagenlogische Formeln* oder kurz *Atome*. Die Zeichen $\neg$, $\wedge$, $\vee$, $\supset$, etc. nennt man *Junktoren*, *Konnektive* oder auch *logische Operatoren*.

[9] Genau genommen müsste man sagen: Die Menge, deren einziges Element die Funktion nand ist, ist funktional vollständig.

Zur Formalisierung der Semantik beziehen wir uns auf die Menge $\mathcal{I}$ der *aussagenlogischen Variablenbelegungen*; das sind alle Funktionen vom Typ $\mathcal{V} \mapsto \mathbb{B}$, wobei $\mathbb{B} = \{1, 0\}$ die Menge der Wahrheitswerte symbolisiert. Man nennt diese Funktionen auch *(aussagenlogische) Interpretationen.*

3.2 Definition (Semantik aussagenlogischer Formeln). Der Wert einer Formel in einer Interpretation I wird festgelegt durch die Funktion $\mathsf{val}{:}\mathcal{I} \times \mathcal{A} \mapsto \mathbb{B}$, die folgendermaßen definiert ist:

(v1) $\mathsf{val}_I(A) = I(A)$ für $A \in \mathcal{V}$;

(v2) $\mathsf{val}_I(\top) = 1$ und $\mathsf{val}_I(\bot) = 0$;

(v3) $\mathsf{val}_I(\neg F) = \mathsf{not}\,\mathsf{val}_I(F)$;

(v4) $\mathsf{val}_I(\,(F * G)\,) = \mathsf{val}_I(F) \circledast \mathsf{val}_I(G)$, wobei $\circledast$ die logische Funktion zum Operator $*$ ist.

3.3 Beispiel. Sei F die aussagenlogische Formel $((A \wedge \neg B) \supset (\neg\top \vee C))$. Den Wahrheitswert von F in einer Interpretation I mit $I(A) = 1$, $I(B) = 0$ und $I(C) = 0$ erhält man wie folgt:

$$\begin{aligned}
\mathsf{val}_I(F) &= \mathsf{val}_I((A \wedge \neg B))\ \mathsf{implies}\ \mathsf{val}_I((\neg\top \vee C)) \\
&= (\,\mathsf{val}_I(A)\ \mathsf{and}\ \mathsf{val}_I(\neg B))\ \mathsf{implies}\ (\,\mathsf{val}_I(\neg\top)\ \mathsf{or}\ \mathsf{val}_I(C)) \\
&= (I(A)\ \mathsf{and}\ \mathsf{not}\,\mathsf{val}_I(B))\ \mathsf{implies}\ (\,\mathsf{not}\,\mathsf{val}_I(\top)\ \mathsf{or}\ I(C)) \\
&= (1\ \mathsf{and}\ \mathsf{not}\,I(B))\ \mathsf{implies}\ (\mathsf{not}\,1\ \mathsf{or}\ 0) \;=\; (1\ \mathsf{and}\ \mathsf{not}\,0)\ \mathsf{implies}\ (\mathsf{not}\,1\ \mathsf{or}\ 0) \\
&= (1\ \mathsf{and}\ 1)\ \mathsf{implies}\ (0\ \mathsf{or}\ 0) \;=\; 1\ \mathsf{implies}\ 0 \;=\; 0
\end{aligned}$$

□

Wenn der Kontext eindeutig ist, lassen wir im Folgenden das Adjektiv 'aussagenlogisch' meist weg. Außerdem werden wir zur Erhöhung der Lesbarkeit weitgehend auf Unterstreichungen verzichten und auch überflüssige Klammern weglassen. Insbesondere steht

$$F_1 \wedge \ldots \wedge F_n \qquad \text{für} \qquad (F_1 \wedge (\ldots \wedge F_n))$$

und

$$F_1 \vee \ldots \vee F_n \qquad \text{für} \qquad (F_1 \vee (\ldots \vee F_n)).$$

Diese Klammereinsparungsregeln sind durch die Semantik von Konjunktion und Disjunktion gerechtfertigt: Die Funktionen and und or sind nämlich assoziativ.

Da jede Formel nur endlich viele Variablen enthält, kann ihre Semantik vollständig durch eine Wahrheitstafel beschrieben werden, die für jede Belegung der Variablen das Ergebnis von $\mathsf{val}_I(F)$ angibt.

3.4 Beispiel. Als Wahrheitstafel für die Formel $(A \wedge \neg B) \supset (\neg\top \vee C)$ erhalten wir:

A	B	C	$\neg B$	$A \wedge \neg B$	$\neg\top$	$\neg\top \vee C$	$(A \wedge \neg B) \supset (\neg\top \vee C)$
1	1	1	0	0	0	1	1
1	1	0	0	0	0	0	1
1	0	1	1	1	0	1	1
1	0	0	1	1	0	0	0
0	1	1	0	0	0	1	1
0	1	0	0	0	0	0	1
0	0	1	1	0	0	1	1
0	0	0	1	0	0	0	1

Genauer müsste die Überschrift der Spalten statt A, B und C eigentlich $I(A)$, $I(B)$ und $I(C)$ lauten; ebenso müsste statt jeder Formel F der Ausdruck $\mathsf{val}_I(F)$ stehen, in der letzten Spalte etwa $\mathsf{val}_I((A \wedge \neg B) \supset (\neg\top \vee C))$.

Die Wahrheitstafel lässt sich kompakter schreiben, wenn die Teilformeln keine eigenen Spalten erhalten, sondern ihre Ergebnisse unter den Operator der entsprechenden Teilformel geschrieben werden:

A	B	C	$(A \wedge \neg B) \supset (\neg\top \vee C)$
1	1	1	1 1 0 1 1 0 1 1 1
1	1	0	1 1 0 1 1 0 1 0 0
1	0	1	1 1 1 0 1 0 1 1 1
1	0	0	1 1 1 0 0 0 1 0 0
0	1	1	0 0 0 1 1 0 1 1 1
0	1	0	0 0 0 1 1 0 1 0 0
0	0	1	0 0 1 0 1 0 1 1 1
0	0	0	0 0 1 0 1 0 1 0 0

□

3.5 Definition. Eine Formel $F \in \mathcal{A}$ heißt

- *gültig* oder *Tautologie*, wenn $\mathsf{val}_I(F) = 1$ für alle $I \in \mathcal{I}$.
- *erfüllbar,* wenn $\mathsf{val}_I(F) = 1$ für ein $I \in \mathcal{I}$. (I heißt *Modell* von F.)
- *widerlegbar*[10], wenn $\mathsf{val}_I(F) = 0$ für ein $I \in \mathcal{I}$. (I heißt *Gegenbeispiel* für F.)
- *unerfüllbar,* wenn $\mathsf{val}_I(F) = 0$ für alle $I \in \mathcal{I}$.

[10] Man beachte, dass der Begriff „widerlegbar“ in der Literatur manchmal in ganz anderer Bedeutung – nämlich dual zum syntaktischen Begriff „beweisbar“ – verwendet wird. Unsere Verwendungsweise bezeichnet hingegen einen semantischen Sachverhalt.

3.6 Beispiel. $A \supset (B \supset A)$ ist gültig und daher auch erfüllbar:

A	B	$(B \supset A)$	$A \supset (B \supset A)$
1	1	1	1
1	0	1	1
0	1	0	1
0	0	1	1

$(A \wedge \neg B) \vee B$ ist erfüllbar und widerlegbar, also weder gültig noch unerfüllbar:

A	B	$\neg B$	$(A \wedge \neg B)$	$(A \wedge \neg B) \vee B$
1	1	0	0	1
1	0	1	1	1
0	1	0	0	1
0	0	1	0	0

□

3.7 Definition. Zwei Formeln $F, G \in \mathcal{A}$ heißen *äquivalent* (Notation: $F = G$), wenn $\mathsf{val}_I(F) = \mathsf{val}_I(G)$ für alle $I \in \mathcal{I}$ gilt.

Der folgende Satz zeigt, dass sich der semantische Äquivalenzbegriff im syntaktischen Operator $\equiv$ widerspiegelt.

3.8 Satz. *F und G sind äquivalent genau dann, wenn $F \equiv G$ gültig ist.*

Beweis. $F = G$ wenn $\mathsf{val}_I(F) = \mathsf{val}_I(G)$ für alle $I \in \mathcal{I}$. Das bedeutet, dass $\mathsf{val}_I(F) = 1$ genau dann, wenn $\mathsf{val}_I(G) = 1$, und dass $\mathsf{val}_I(F) = 0$ genau dann, wenn $\mathsf{val}_I(G) = 0$. Laut Definition von $\equiv$ ist das aber gleichbedeutend mit $\mathsf{val}_I(F \equiv G) = 1$ für alle I, also mit der Gültigkeit von $F \equiv G$. □

4 Formeln und Formelschemata

Eine Substitution (Ersetzung) ist eine Abbildung von Variablen A_i auf Formeln F_i. Eine Substitution auf eine Formel anzuwenden bedeutet, in der Formel gleichzeitig alle Vorkommnisse der Variablen A_i durch die entsprechenden Formeln F_i zu ersetzen. Da wir nur Substitutionen benötigen, die eine endliche Zahl von Variablen ersetzen und die anderen unverändert lassen, können wir sie durch Ausdrücke der Form $[A_1 \mapsto F_1, \cdots, A_n \mapsto F_n]$ beschreiben. Das Ergebnis der Anwendung einer Substitution $\sigma = [A_1 \mapsto F_1, \cdots, A_n \mapsto F_n]$ auf eine Formel F schreiben wir als $F\sigma = F[A_1 \mapsto F_1, \cdots, A_n \mapsto F_n]$.

Jede Formel kann als Formelschema aufgefasst werden, indem man die Variablen als Platzhalter für beliebige Formeln interpretiert. Handelt es sich beim Schema um eine Tautologie, sind auch die daraus abgeleiteten Formeln Tautologien.

4.1 Satz. *Ist σ eine Substitution und F eine gültige (bzw. unerfüllbare) Formel, dann ist auch $F\sigma$ eine gültige (bzw. unerfüllbare) Formel.*

BEWEIS. Sei $\sigma = [A_1 \mapsto F_1, \cdots, A_n \mapsto F_n]$. Die Auswertung von $F\sigma$ in einer Interpretation I, d.h. $\mathsf{val}_I(F\sigma)$, ist gleichbedeutend mit der Auswertung von F in einer Interpretation I', wobei I' folgendermaßen definiert ist:

$$I'(A) = \begin{cases} \mathsf{val}_I(F_i) & \text{wenn } A = A_i \text{ für ein } i = 1, \ldots, n \\ I(A) & \text{sonst} \end{cases}$$

Da F gültig ist, evaluiert F in allen Interpretationen zu 1, insbesondere in I'. Wir erhalten also für alle I:

$$\mathsf{val}_I(F\sigma) = \mathsf{val}(I', F) = 1$$

d.h., $F\sigma$ ist gültig. Der Beweis für unerfüllbare Formeln erfolgt analog. □

4.2 FOLGERUNG. *Gilt $F = G$, dann auch $F\sigma = G\sigma$.*

4.3 BEISPIEL. $F = (A \wedge (A \supset B)) \supset B$ ist gültig, daher auch

$$F[A \mapsto X \vee \neg Y,\, B \mapsto \neg X \wedge Y] = \Big((X \vee \neg Y) \wedge ((X \vee \neg Y) \supset (\neg X \wedge Y))\Big) \supset (\neg X \wedge Y) \ .$$

□

Der folgende Satz drückt aus, dass man aus einer Formel äquivalente Formeln erhält, wenn beliebige Teilformeln durch äquivalente ersetzt werden. Dies ist in Zusammenhang mit dem letzten Satz nützlich beim Umformen von Formeln mit Hilfe gültiger Gesetze wie der De-Morgan-Regel.

4.4 PROPOSITION. *Angenommen es gilt $F_1 = G_1$ und $F_2 = G_2$ für Formeln F_1, G_1, F_2, G_2. Dann gilt $\neg F_1 = \neg G_1$ sowie $(F_1 * F_2) = (G_1 * G_2)$ für $* \in \{\wedge, \vee, \supset\}$.*

BEWEIS. Wir zeigen die Fälle $\neg$ und $\wedge$; der Nachweis für $\vee$ und $\supset$ verläuft analog.

Nach Definition von val gilt für beliebiges I:

$$(\star)\ \ \mathsf{val}_I(\neg F_1) = \neg\, \mathsf{val}_I(F_1),\ \mathsf{val}_I(\neg G_1) = \neg\, \mathsf{val}_I(G_1).$$

Wegen $\mathsf{val}_I(F_1) = \mathsf{val}_I(G_1)$ (dies folgt aus $F_1 = G_1$) gilt auch $\neg(\mathsf{val}_I(F_1)) = \neg(\mathsf{val}_I(G_1))$ (da $\neg$ eine Boolsche Funktion ist). Wegen $(\star)$ erhalten wir dann

$$\mathsf{val}_I(\neg F_1) = \mathsf{val}_I(\neg G_1).$$

Da dies für beliebiges I gilt folgt $\neg F_1 = \neg G_1$.

Wir zeigen nun den Fall für $\wedge$: Sei I eine beliebige Interpretation. Dann gilt per Definition von val:

$$\begin{aligned} &(i) \quad \mathsf{val}_I((F_1 \wedge F_2)) = \mathsf{val}_I(F_1) \wedge \mathsf{val}_I(F_2), \\ &(ii) \quad \mathsf{val}_I((G_1 \wedge G_2)) = \mathsf{val}_I(G_1) \wedge \mathsf{val}_I(G_2). \end{aligned}$$

Nach Voraussetzung gilt $\mathsf{val}_I(F_1) = \mathsf{val}_I(G_1)$ und $\mathsf{val}_I(F_2) = \mathsf{val}_I(G_2)$. Da $\wedge$ eine Boolsche Funktion ist folgt dann auch

$$(iii) \quad \mathsf{val}_I(F_1) \wedge \mathsf{val}_I(F_2) = \mathsf{val}_I(G_1) \wedge \mathsf{val}_I(G_2).$$

Aus $(i), (ii)$ und (iii) folgt dann

$$\mathsf{val}_I((F_1 \wedge F_2)) = \mathsf{val}_I((G_1 \wedge G_2)).$$

Da dies für beliebige I gilt folgt $(F_1 \wedge F_2) = (G_1 \wedge G_2)$. □

4.5 Satz. *Seien σ_1, σ_2 Substitutionen, sodass $A\sigma_1 = A\sigma_2$ für alle Variablen A gilt. Dann gilt $F\sigma_1 = F\sigma_2$ für alle Formeln F.*

Beweis. Mit Induktion nach $o(F)$, der Anzahl des Vorkommens logischer Operatoren in F.

$o(F) = 0$: Dann ist F eine Variable und $F\sigma_1 = F\sigma_2$ gilt nach Voraussetzung.

Induktionshypothese (IH): Angenommen $F\sigma_1 = F\sigma_2$ für alle Formeln F mit $o(F) \leq n$.

Wir betrachten den Fall $n+1$, d.h. sei F eine Formel mit $o(F) = n+1$. Wir unterscheiden die folgenden Fälle:

(a) $F = \neg F'$,

(b) $F = (F_1 \wedge F_2)$,

(c) $F = (F_1 \vee F_2)$,

(d) $F = (F_1 \supset F_2)$.

Wir betrachten den Fall (a):

$$F\sigma_1 = (\neg F')\sigma_1 = \neg(F'\sigma_1), \ (a1)$$
$$F\sigma_2 = (\neg F')\sigma_2 = \neg(F'\sigma_2). \ (a2)$$

Da $o(F') \leq n$ wenden wir (IH) an und erhalten $F'\sigma_1 = F'\sigma_2$. Unter Verwendung von Proposition 4.4 erhalten wir

$$\neg(F'\sigma_1) = \neg(F'\sigma_2).$$

Mittels $(a1)$ und $(a2)$ ergibt sich dann $F\sigma_1 = F\sigma_2$.

$$(F_1 \wedge F_2)\sigma_1 = F_1\sigma_1 \wedge F_2\sigma_1, \ (b1)$$
$$(F_1 \wedge F_2)\sigma_2 = F_1\sigma_2 \wedge F_2\sigma_2. \ (b2)$$

Da $o(F_1) \leq n$ und $o(F_2) \leq n$ wenden wir (IH) an und erhalten

$$F_1\sigma_1 = F_1\sigma_2, \ F_2\sigma_1 = F_2\sigma_2 \ (b3).$$

Wir verwenden Proposition 4.4 und erhalten

$$F_1\sigma_1 \wedge F_2\sigma_1 = F_1\sigma_2 \wedge F_2\sigma_2.$$

Aus $(b1)$ und $(b2)$ ergibt sich dann $F\sigma_1 = F\sigma_2$.

Die Fälle (c),(d) sind analog zu (b). □

4.6 Beispiel. Sei $F = (A \wedge B) \supset C$, $\sigma_1 = [B \mapsto A \supset B,\, C \mapsto B]$ und $\sigma_2 = [B \mapsto \neg A \vee B,\, C \mapsto B]$. Da

$$B\sigma_1 = (A \supset B) \equiv (\neg A \vee B) = B\sigma_2 \quad \text{und} \quad C\sigma_1 = B \equiv B = C\sigma_2$$

gilt, erhalten wir

$$F\sigma_1 = (A \wedge (A \supset B)) \supset B \equiv (A \wedge (\neg A \vee B)) \supset B = F\sigma_2 \ .$$

Diese Formel lässt sich als Anwendung der Regel $(A \supset B) = (\neg A \vee B)$ auf eine Teilformel links oder rechts des Äquivalenzzeichens interpretieren. Das Resulat ist die äquivalente Formel auf der anderen Seite von $\equiv$. □

5 Logische Konsequenz und Implikation

5.1 Definition. Wir definieren $F_1, \ldots, F_n \models_I G$ wenn gilt:

$$\text{Ist } \mathsf{val}_I(F_i) = 1 \text{ für } 1 \leq i \leq n, \text{ dann gilt auch } \mathsf{val}_I(G) = 1.$$

Eine Formel G ist eine *logische Konsequenz* der Formeln $F_1, \ldots, F_n$, geschrieben $F_1, \ldots, F_n \models G$, wenn für *alle* Interpretationen I $F_1, \ldots, F_n \models_I G$ gilt.

Man sagt auch: G *folgt logisch* aus $F_1, \ldots, F_n$.

Für $n = 0$ ist $\models G$ gleichbedeutend mit „G ist gültig."

Der folgende Satz zeigt, dass der semantische Begriff der logischen Konsequenz im Implikationsoperator sein syntaktisches Pendant besitzt.

5.2 Satz (Deduktionstheorem).
$F_1, \ldots, F_n \models G$ gilt genau dann, wenn $F_1, \ldots, F_{n-1} \models (F_n \supset G)$ zutrifft.

Beweis. Es genügt zu zeigen, dass für alle I gilt:

$$F_1, \ldots, F_n \models_I G \text{ genau dann wenn } F_1, \ldots, F_{n-1} \models_I (F_n \supset G).$$

(a) Angenommen $F_1, \ldots, F_n \models_I G$. Wir unterscheiden zwei Fälle

(a1) $\mathsf{val}_I(F_i) = 0$ für ein $i \leq n$. Ist $i \leq n-1$ dann gilt sicher $F_1, \ldots, F_{n-1} \models_I F_n \supset G$ – per Definition von $\models_I$. Ist $i = n$ so ist $\mathsf{val}_I(F_n) = 0$ und daher $\mathsf{val}_I(F_n \supset G) = 1$. Auch in diesem Falle gilt $F_1, \ldots, F_{n-1} \models_I F_n \supset G$.

(a2) $\mathsf{val}_I(F_i) = 1$ für alle $i \leq n$. Nach Definition von $\models_I$ muss $\mathsf{val}_I(G) = 1$ gelten. Damit gilt aber $\mathsf{val}_I(F_n \supset G) = 1$ und somit auch $F_1, \ldots, F_{n-1} \models_I F_n \supset G$.

(b) Angenommen, $F_1, \ldots, F_{n-1} \models_I F_n \supset G$.

(b1) Ist $\mathsf{val}_I(F_i) = 0$ für ein $i \leq n-1$ dann gilt trivialerweise $F_1, \ldots, F_n \models_I G$.

(b2) Angenommen, $\mathsf{val}_I(F_i) = 1$ für $i \leq n-1$. Nach Annahme und nach Definition von $\models_I$ gilt dann $\mathsf{val}_I(F_n \supset G) = 1$.

Ist $\mathsf{val}_I(F_n) = 0$ so gilt sicher $F_1, \ldots, F_n \models_I G$.

Ist nun $\mathsf{val}_I(F_n) = 1$, so folgt wegen $\mathsf{val}_I(F_n \supset G) = 1$ auch $\mathsf{val}_I(G) = 1$; daraus folgt aber $F_1, \ldots, F_n \models_I G$. □

5.3 Folgerung. *$F_1, \ldots, F_n \models G$ gilt genau dann, wenn $F_1 \supset (F_2 \supset \cdots (F_n \supset G) \cdots)$ bzw. $(F_1 \wedge \cdots \wedge F_n) \supset G$ eine gültige Formel ist.*

5.4 Beispiel. B folgt logisch aus A und $A \supset B$, d.h., es gilt $A, A \supset B \models B$. Daher sind die Formeln $A \supset ((A \supset B) \supset B)$ und $(A \wedge (A \supset B)) \supset B$ Tautologien.

Umgekehrt ergibt sich aus der Tatsache, dass $(A \wedge B) \supset (A \vee B)$ eine Tautologie ist, dass $A \vee B$ eine logische Konsequenz von $A \wedge B$ ist. □

6 Normalformen

Unter der Normalform einer Formel versteht man eine standardisierte vereinfachte Variante, die äquivalent zur ursprünglichen Formel ist.

6.1 Definition. Ein *Literal* ist eine aussagenlogische Variable $A \in \mathcal{V}$ oder ihr Negat, $\neg A$. Seien im Folgenden $L_{i,j}$ Literale für $i = 1, \ldots, m$, $j = 1, \ldots, n_i$.

Eine Formel ist in *disjunktiver Normalform (DNF)*, wenn sie die Form

$$(L_{1,1} \wedge \cdots \wedge L_{1,n_1}) \vee \cdots \vee (L_{m,1} \wedge \cdots \wedge L_{m,n_m})$$

besitzt, d.h., wenn sie eine Disjunktion von Konjunktionen von Literalen ist.

Eine Formel ist in *konjunktiver Normalform (KNF, CNF)*, wenn sie die Form

$$(L_{1,1} \vee \cdots \vee L_{1,n_1}) \wedge \cdots \wedge (L_{m,1} \vee \cdots \vee L_{m,n_m})$$

besitzt, d.h., wenn sie eine Konjunktion von Disjunktionen von Literalen ist. Die einzelnen Disjunktionen einer KNF werden auch *Klauseln* genannt, die KNF entsprechend auch *Klauselnormalform*.[11]

Als Spezialfall rechnet man auch die Formeln $\top$ und $\bot$ zu den konjunktiven und disjunktiven Normalformen. Dabei entspricht $\top$ einer leeren Konjunktion und $\bot$ einer leeren Disjunktion.

Da Konjunktion und Disjunktion assoziativ, kommutativ und idempotent sind, wird für KNFs und DNFs oft die Mengenschreibweise verwendet. Demnach kann die Menge von Literalmengen

$$\{\{L_{1,1}, \ldots, L_{1,n_1}\}, \ldots, \{L_{m,1}, \ldots, L_{m,n_m}\}\}$$

[11] Man beachte, dass Formeln wie $L_1 \wedge \cdots \wedge L_n$ und $L_1 \vee \cdots \vee L_n$, d.h. reine Konjunktionen und Disjunktionen von Literalen, sowohl in konjunktiver als auch in disjunktiver Normalform sind.

sowohl als DNF als auch als KNF interpretiert werden. In diesem Skriptum wird die Mengennotation ausschließlich für KNFs verwendet. Dem entsprechend repräsentiert eine *Klauselmenge*, d.h. eine endliche Menge von endlichen Mengen von Literalen, hier immer eine KNF.

Gemäß der oben getroffenen Vereinbarung steht die leere Menge $\{\}$ auf der Ebene der Klauseln für $\bot$, auf der Ebene einer vollständigen Kauselnormalform jedoch für $\top$.

6.2 Satz. *Zu jeder aussagenlogischen Formel gibt es eine äquivalente Formel in disjunktiver bzw. konjunktiver Normalform.*

Als Beweis dieses Satzes beschreiben wir zwei Methoden, um zu einer gegebenen Formel eine äquivalente in Normalform zu konstruieren.

Syntaktische (algebraische) Methode

Schritt 1: Ersetze alle Operatoren durch $\wedge$, $\vee$ und $\neg$.

$$\begin{array}{rlrl} A \supset B &= \neg A \vee B & A \not\supset B &= A \wedge \neg B \\ A \subset B &= A \vee \neg B & A \not\subset B &= \neg A \wedge B \\ A \uparrow B &= \neg A \vee \neg B & A \downarrow B &= \neg A \wedge \neg B \\ (A \equiv B) &= (A \wedge B) \vee (\neg A \wedge \neg B) & (A \not\equiv B) &= (\neg A \wedge B) \vee (A \wedge \neg B) \end{array}$$

Schritt 2: Schiebe alle Negationen direkt vor die Variablen und Konstanten.

$$\neg(A \wedge B) = \neg A \vee \neg B \qquad \neg(A \vee B) = \neg A \wedge \neg B \qquad \neg\neg A = A$$

Schritt 3: Reduziere die Formel mittels der Konstanten $\top$ und $\bot$.

$$A \wedge \top = A \qquad A \wedge \bot = \bot \qquad A \vee \top = \top \qquad A \vee \bot = A \qquad \neg\top = \bot \qquad \neg\bot = \top$$

Schritt 4: Wende die Distributivgesetze an. Die linke Äquivalenz führt zu einer DNF, die rechte zu einer KNF.

$$A \wedge (B \vee C) = (A \wedge B) \vee (A \wedge C) \qquad A \vee (B \wedge C) = (A \vee B) \wedge (A \vee C)$$

In den letzten beiden Schritten berücksichtigen wir implizit die Kommutativität von Konjunktion und Disjunktion. Mit anderen Worten: $A \wedge \top$ steht auch für $\top \wedge A$ und $A \wedge (B \vee C)$ auch für $(B \vee C) \wedge A$, etc.

Die Korrektheit der Methode beruht auf drei Tatsachen. Erstens sind alle angegeben Gleichungen gültige Äquivalenzen, wodurch alle durch Umformung entstehenden Formeln äquivalent zur ursprünglichen sind, insbesondere die im letzten Schritt konstruierte DNF bzw. KNF. Zweitens kann man sich davon überzeugen, dass auf jede Formel, die noch nicht in disjunktiver bzw. konjunktiver Normalform vorliegt, eine der Äquivalenzen anwendbar ist. Und drittens gibt es keine unendlich langen Umformungsketten, wenn die Äquivalenzen von links nach rechts angewendet werden. Somit terminiert das Verfahren, und das Ergebnis ist eine DNF/KNF, die äquivalent zur ursprünglichen Formel ist.

6.3 Beispiel. Wir konstruieren Normalformen zur Formel $(A \uparrow (B \vee C)) \supset (\neg D \wedge \neg\neg\top)$.

Schritt 1:	$\neg(\neg A \vee \neg(B \vee C)) \vee (\neg D \wedge \neg\neg\top)$
Schritt 2:	$(A \wedge (B \vee C)) \vee (\neg D \wedge \top)$
Schritt 3:	$(A \wedge (B \vee C)) \vee \neg D$
Schritt 4:	Ausdistribuieren liefert:
DNF:	$(A \wedge B) \vee (A \wedge C) \vee \neg D$
KNF:	$(A \vee \neg D) \wedge (B \vee C \vee \neg D)$

In der Mengennotation entspricht die KNF der Klauselmenge $\{\{A, \neg D\},\ \{B, C, \neg D\}\}$. □

Semantische Methode

Sei $F \in \mathcal{A}$ eine aussagenlogische Formel. Die semantische Methode konstruiert die DNF/KNF an Hand der Wahrheitstafel, wobei jede Interpretation I (jede Zeile der Wahrheitstafel) entweder eine Konjunktion zur DNF oder eine Disjunktion zur KNF beisteuert.

DNF: Zu jedem $I \in \mathcal{I}$ definieren wir eine charakteristische Konjunktion K_I:

$$K_I = \bigwedge_{A \text{ in } F} L_A \qquad \text{wobei} \quad L_A = \begin{cases} A & \text{falls } I(A) = 1 \\ \neg A & \text{falls } I(A) = 0 \end{cases}$$

K_I ist wahr ausschließlich in I, und falsch in allen anderen Interpretationen: $\mathsf{val}_I(K_I) = 1$ und $\mathsf{val}(I', K_I) = 0$ für $I' \neq I$. Die gesuchte DNF ist die Disjunktion der Konjunktionen K_I für jene I, in denen F den Wert 1 besitzt:

$$\mathsf{DNF}_F = \bigvee_{I \in \mathcal{I}, \mathsf{val}_I(F) = 1} K_I$$

Die Formel DNF_F evaluiert in einer Interpretation I genau dann zu 1, wenn sie K_I enthält, was wiederum per Konstruktion nur dann der Fall ist, wenn F in I zu 1 evaluiert. In Summe erhalten wir: $\mathsf{val}_I(\mathsf{DNF}_F) = 1$ genau dann, wenn $\mathsf{val}_I(F) = 1$.

KNF: Zu jedem $I \in \mathcal{I}$ definieren wir eine charakteristische Disjunktion D_I:

$$D_I = \bigvee_{A \text{ in } F} L_A \qquad \text{wobei} \quad L_A = \begin{cases} A & \text{falls } I(A) = 0 \\ \neg A & \text{falls } I(A) = 1 \end{cases}$$

D_I ist falsch ausschließlich in I, und wahr in allen anderen Interpretationen: $\mathsf{val}_I(D_I) = 0$ und $\mathsf{val}(I', D_I) = 1$ für $I' \neq I$. Die gesuchte KNF ist die Konjunktion der Disjunktionen D_I für jene I, in denen F den Wert $\bot$ besitzt:

$$\mathsf{KNF}_F = \bigwedge_{I \in \mathcal{I}, \mathsf{val}_I(F) = 0} D_I$$

Die Formel KNF_F evaluiert in einer Interpretation I genau dann zu 0, wenn sie D_I enthält, was wiederum per Konstruktion nur dann der Fall ist, wenn F in I zu 0 evaluiert. In Summe erhalten wir: $\mathsf{val}_I(\mathsf{KNF}_F) = 0$ genau dann, wenn $\mathsf{val}_I(F) = 0$.

A	B	C	D	$B \vee C$	$A \uparrow (B \vee C)$	$\neg D$	F	K_I bzw. D_I
0	0	0	0	0	1	1	1	...
1	0	0	0	0	1	1	1	...
0	1	0	0	1	1	1	1	$\neg A \wedge B \wedge \neg C \wedge \neg D \quad (K_I)$
1	1	0	0	1	0	1	1	...
0	0	1	0	1	1	1	1	...
1	0	1	0	1	0	1	1	...
0	1	1	0	1	1	1	1	...
1	1	1	0	1	0	1	1	...
0	0	0	1	0	1	0	0	...
1	0	0	1	0	1	0	0	...
0	1	0	1	1	1	0	0	...
1	1	0	1	1	0	0	1	...
0	0	1	1	1	1	0	0	...
1	0	1	1	1	0	0	1	...
0	1	1	1	1	1	0	0	$A \vee \neg B \vee \neg C \vee \neg D \quad (D_I)$
1	1	1	1	1	0	0	1	...

Abbildung 1: Wahrheitstafel für die Formel $(A \uparrow (B \vee C)) \supset \neg D$ (Beispiel 6.4)

DNF_F bzw. KNF_F ist die eindeutig bestimmte *vollständige* DNF bzw. KNF von F: Jede Konjunktion K_I bzw. jede Disjunktion D_I enthält alle in F vorkommenden Variablen.

6.4 Beispiel. Wir konstruieren die Normalformen der Formel $F = (A \uparrow (B \vee C)) \supset \neg D$. Als Ausgangspunkt dient die Wahrheitstafel in Abbildung 1. Exemplarisch greifen wir je eine Interpretation mit $\mathsf{val}_I(F) = 1$ bzw. $\mathsf{val}_I(F) = 0$ heraus. Die letzte Spalte der dritten Zeile gibt eine charakteristische Konjunktion für die Interpretation mit $I(B) = 1$ und $I(A) = I(C) = I(D) = 0$ an. Offenbar evaluiert die Konjunktion ausschließlich in dieser Interpretation zu 1. Da die Tafel elf Zeilen besitzt, in denen F zu 1 evaluiert, erhalten wir eine DNF mit insgesamt elf Konjunktionen:

$$\begin{aligned}\mathsf{DNF}_F \;=\; & (\neg A \wedge \neg B \wedge \neg C \wedge \neg D) \vee (A \wedge \neg B \wedge \neg C \wedge \neg D) \vee (\neg A \wedge B \wedge \neg C \wedge \neg D) \\ & \vee (A \wedge B \wedge \neg C \wedge \neg D) \vee (\neg A \wedge \neg B \wedge C \wedge \neg D) \vee (A \wedge \neg B \wedge C \wedge \neg D) \\ & \vee (\neg A \wedge B \wedge C \wedge \neg D) \vee (A \wedge B \wedge C \wedge \neg D) \vee (A \wedge B \wedge \neg C \wedge D) \\ & \vee (A \wedge \neg B \wedge C \wedge D) \vee (A \wedge B \wedge C \wedge D)\end{aligned}$$

Die vorletzte Zeile der Wahrheitstafel repräsentiert eine Interpretation, in der F zu 0 evaluiert. Die letzte Spalte gibt die charakteristische Disjunktion zu dieser Interpretation an. Offenbar evaluiert die Disjunktion ausschließlich in dieser Interpretation zu 0. Da die Tafel fünf Zeilen besitzt, in denen F zu 0 evaluiert, erhalten wir eine KNF mit insgesamt fünf Disjunktionen:

$$\begin{aligned}\mathsf{KNF}_F \;=\; & (A \vee B \vee C \vee \neg D) \wedge (\neg A \vee B \vee C \vee \neg D) \wedge (A \vee \neg B \vee C \vee \neg D) \\ & \wedge (A \vee B \vee \neg C \vee \neg D) \wedge (A \vee \neg B \vee \neg C \vee \neg D)\end{aligned}$$

bzw. in Mengennotation geschrieben:

$$\begin{aligned}\mathsf{KNF}_F \quad = \quad & \{\,\{A, B, C, \neg D\},\ \{\neg A, B, C, \neg D\},\ \{A, \neg B, C, \neg D\},\\ & \{A, B, \neg C, \neg D\},\ \{A, \neg B, \neg C, \neg D\}\,\}.\end{aligned}$$

□

Die Länge einer disjunktiven bzw. konjunktiven Normalform kann exponentiell größer sein als die ursprüngliche Formel. Verursacht wird dieses Wachstum bei der algebraischen Methode durch die Elimination von $\equiv$ und $\not\equiv$ sowie durch das Anwenden der Distributivität, bei der semantischen Methode durch die Anzahl der möglichen Interpretationen (Zeilen der Wahrheitstafel). Während die semantische Methode aber immer lange Normalformen liefert, erhält man durch die algebraische Methode meist deutlich kürzere. Der schlechteste Fall tritt ein, wenn man eine DNF in eine KNF (oder umgekehrt) transformieren muss.