Logik und formale Methoden

Vorlesungsskript

von
Burkhardt Renz

Wintersemester 2019/20

Burkhardt Renz
Technische Hochschule Mittelhessen
Rev 0.40 – 29. Januar 2020

Inhaltsverzeichnis

Inhaltsverzeichnis ii

1 Einleitung 1
1.1 Klassische Logik 1
1.2 Mathematische Logik 1
1.3 Logik und Informatik 1
1.4 Programm der Veranstaltung 1

I Aussagenlogik 2

2 Aussagen und Formeln 3

3 Die formale Sprache der Aussagenlogik 6

4 Die Semantik der Aussagenlogik 12
4.1 Struktur/Modell 14
4.2 Wahrheitstafel 16
4.3 Semantische Äquivalenz und Substitution 17
4.4 Boolesche Operatoren und funktionale Vollständigkeit . 18

5 Das Beweissystem des natürlichen Schließens 20
5.1 Schlussregeln 22
5.2 Beispiele für das natürliche Schließen 24
5.3 Beweisstrategien 27
5.4 Eigenschaften der Herleitbarkeit $\vdash$ 29
5.5 Vollständigkeit des natürlichen Schließens 32

6 Normalformen 33
6.1 Negationsnormalform NNF 33
6.2 Konjunktive Normalform CNF 34
6.3 Disjunktive Normalform DNF 36
6.4 Normalformen und Entscheidungsprobleme 36

7 Die Komplexität des Erfüllbarkeitsproblems 38

7.1 Das Erfüllbarkeitsproblem 38
7.2 Komplexität von Algorithmen 39
7.3 Die Komplexität des Erfüllbarkeitsproblems 42

8 Hornlogik 44

9 Erfüllbarkeit und SAT-Solver 48
9.1 DIMACS-Format 48
9.2 Tseitin-Transformation 50
9.3 DPLL und CDLC 51

10 Anwendungen der Aussagenlogik in der Softwaretechnik 58
10.1 Anwendungen von SAT-Techniken 58
10.2 Codeanalyse 58
10.3 Featuremodelle für (Software-)Produktlinien 58

II Prädikatenlogik 59

11 Objekte und Prädikate 60
11.1 Elemente der Sprache der Prädikatenlogik 61
11.2 Prädikate und Relationen 62

12 Die formale Sprache der Prädikatenlogik 63
12.1 Signatur, Terme, Formeln 63
12.2 Freie und gebundene Variablen 65
12.3 Substitution 66

13 Semantik der Prädikatenlogik 68
13.1 Struktur/Modell 68
13.2 Semantische Folgerung und Äquivalenz 70
13.3 Fundamentale Äquivalenzen der Prädikatenlogik 71

14 Natürliches Schließen in der Prädikatenlogik 72
14.1 Schlussregeln 73
14.2 Beispiele 74
14.3 Vollständigkeit des natürlichen Schließens 75

15 Unentscheidbarkeit der Prädikatenlogik 76

16 Anwendungen der Prädikatenlogik in der Softwaretechnik 77
16.1 Analyse von Softwaremodellen mit Alloy 77

III Lineare Temporale Logik **78**

17 Dynamische Modelle **79**

18 Die formale Sprache der linearen temporalen Logik (LTL) **80**

19 Die Semantik der linearen temporalen Logik (LTL) **82**
19.1 Kripke-Struktur 82
19.2 Äquivalenzen von Formeln der LTL 87
19.3 Typische Aussagen in der LTL 88

20 Natürliches Schließen in der LTL **89**

21 Anwendungen der LTL in der Softwaretechnik **90**
21.1 Model Checking 90
21.2 Zielemodell in der Anforderungsanalyse 90

Literaturverzeichnis **91**

Kapitel 1

Einleitung

siehe Vorlesung

1.1 Klassische Logik

1.2 Mathematische Logik

1.3 Logik und Informatik

1.4 Programm der Veranstaltung

Teil I

Aussagenlogik

Kapitel 2

Aussagen und Formeln

In der Aussagenlogik werden Aussagen betrachtet, die wahr oder falsch sein können. Solche Aussagen nennt man *wahrheitsdefinite* Aussagen. Es gibt viele andere Formen von Aussagen in natürlichen Sprachen, wie z.B. ironische Äußerungen, Fragen oder Aufforderungen. Die Sprache der Aussagenlogik ist eine *formale* Sprache, in der nur wahrheitsdefinite Aussagen verwendet werden.

Beispiele

$P =$ „Göttingen ist nördlich von Frankfurt“
$Q =$ „6 ist eine Primzahl“
$R =$ „Jede gerade Zahl > 2 ist die Summe zweier Primzahlen“[1]

aber nicht:

„Könnten Sie bitte die Türe schließen“, eine Aufforderung
„Wie spät ist es“, eine Frage
„Guten Tag“, ein Gruß

Der *Inhalt* der Aussagen ist für die Aussagenlogik nicht wirklich von Belang. Wir studieren *nicht* den Wahrheitsgehalt von Aussagen, sondern die *Beziehung* der Aussagen.

Wir können atomare Aussagen miteinander verbinden und daraus zusammengesetzte Aussagen, *Formeln* bilden. Die Verbindung wird durch *Junktoren* hergestellt - siehe Tabelle 2.1

Mit Junktoren können wir Aussagen verbinden.

[1] Goldbach-Vermutung, benannt nach dem Mathematiker Christian Goldbach (1690 - 1764).

Tabelle 2.1: Junktoren und weitere Symbole

$\neg$	„nicht“, `not`	Negation
$\wedge$	„und“, `and`	Konjunktion
$\vee$	„oder“, `or`	Disjunktion
$\rightarrow$	„impliziert“, `implies`	Implikation
$\top$	„wahr“, `true`, verum	Wahrheit
$\bot$	„falsch“, `false`, absurdum	Widerspruch

Beispiele

$P \wedge Q$	„Göttingen ist nördlich von Frankfurt *und* 6 ist eine Primzahl“	①
$P \vee Q$	„Göttingen ist nördlich von Frankfurt *oder* 6 ist eine Primzahl“	②
$P \rightarrow Q$	„Göttingen ist nördlich von Frankfurt *impliziert* 6 ist eine Primzahl“	③
$Q \rightarrow P$	„6 ist eine Primzahl *impliziert* Göttingen ist nördlich von Frankfurt“	④

Bemerkungen

1. Die erste Aussage ist falsch. Allerdings muss man bei der Übertragung von Aussagen aus der natürlichen Sprache Vorsicht walten lassen:
   Die folgenden beiden Aussagen haben einen unterschiedlichen Sinn
   „Er ging zur Schule und ihm war langweilig.“
   „Ihm war langweilig und er ging zur Schule.“
   obwohl die Aussagen $P \wedge Q$ und $Q \wedge P$ in der formalen Sprache der Aussagenlogik äquivalent sind.
2. Die zweite Aussage ist wahr.
3. Die dritte Aussage ist falsch.
4. Die vierte Aussage ist wahr – obwohl offensichtlicher Unsinn! Implikationen in der formalen Logik darf man nicht mit *Kausalität* verwechseln. Die Aussage ist wahr, weil in der (klassischen) formalen Logik das Prinzip *ex falso quod libet* gilt: Aus einer falschen Voraussetzung darf man alles folgern. Warum diese Definition der Implikation sinnvoll ist, werden wir später sehen.
5. Es gibt Aussagen, die wahr sind, egal welche Wahrheitswerte die beteiligten atomaren Aussagen haben, wie z.B.
   $((P \rightarrow Q) \rightarrow (\neg P \vee Q)) \wedge ((\neg P \vee Q) \rightarrow (P \rightarrow Q))$
   Solche Aussagen nennt man allgemeingültig oder *Tautologie*.

Wir haben in dieser einführenden Diskussion zwei Konzepte verwendet, ohne sie genau zu unterscheiden: Die *Syntax* der Aussagenlogik, die festlegt, welche Formeln wir aus atomaren Aussagen und Junktoren bilden können, sowie die *Semantik* der Aussagenlogik, bei der wir von Wahrheitswerten der atomaren Aussagen reden und aus diesen den Wahrheitswert von Formeln ermitteln können.

In den folgenden drei Kapiteln werden wir diese beiden Konzepte und ihren Zusammenhang auf systematische Weise untersuchen.

Kapitel 3

Die formale Sprache der Aussagenlogik

In der Sprache der Aussagenlogik kombiniert man atomare Aussagen mit Junktoren. Dabei gehen wir von einer gegebenen Menge $\mathcal{P}$ von Aussagensymbolen sowie Junktoren aus. Genau genommen beziehen sich die folgenden Definitionen auf die Wahl dieser Menge und man sollte von *einer* Sprache der Aussagenlogik sprechen.

Definition 3.1 (Alphabet der Aussagenlogik)**.** Das *Alphabet* der Sprache der Aussagenlogik besteht aus

(i) einer Menge $\mathcal{P}$ von Aussagensymbolen,

(ii) den Junktoren: $\neg, \wedge, \vee, \rightarrow$

(iii) der Konstanten: $\perp$

(iv) den zusätzlichen Symbolen: $(,)$

Bemerkungen

- Für eine Sprache der Aussagenlogik nennt man die Wahl der Menge der Aussagensymbole sowie der Junktoren usw. auch die *logische Signatur*.
- Viele Autoren geben eine fixe (abzählbare) Menge von Aussagensymbolen vor, etwa $\mathcal{P} = \{P_0, P_1, P_2, \dots\}$ und sprechen dann von *der* Sprache der Aussagenlogik.[1]

[1] Die Menge der Aussagensymbole muss übrigens nicht unbedingt abzählbar sein, sondern kann eine beliebige Menge sein, siehe [Rau08, S. 4 und Abschnitt 1.5]. Das ist interessant, wenn man den Kompaktheitssatz der Aussagenlogik verwendet, um Aussagen über unendliche Mengen zu beweisen. In dieser Vorlesung werden wir uns damit nicht befassen, weil wir uns auf Anwendungen der formalen Logik in Informatik und Softwaretechnik konzentrieren.

Für Anwendungen der Aussagenlogik in der Informatik und der Softwareentwicklung haben wir jedoch mit endlichen Mengen zu tun und wir möchten den Aussagensymbole auch „sprechende“ Bezeichnungen geben können. Deshalb geht in unsere Definition des Alphabets die Wahl der Menge der Aussagensymbole ein.

- Die Bezeichnung von $\neg, \wedge$ etc. als *Junktoren* soll unterstreichen, dass wir eine formale Sprache definieren. Natürlich aber darf man schon daran denken, dass mit $\neg$ die *Negation* („nicht“), mit $\wedge$ die *Konjunktion* („und“), mit $\vee$ die *Disjunktion* („oder“) und mit $\rightarrow$ die *Implikation* („impliziert“) gemeint sind.
- Die Konstante $\perp$ steht für den *Widerspruch* („falsch“).
- Auch was die Junktoren angeht, treffen wir in der Definition eine Wahl. Wir könnten z.B. noch weitere Junktoren hinzunehmen, wie etwa $\leftrightarrow$ oder auch Junktoren weglassen. Wir werden später sehen, dass sich aus der Semantik der Aussagenlogik ergibt, dass die Menge von Operatoren, die unsere gewählten Junktoren definieren *funktional vollständig* ist, d.h. jede beliebige Boolesche Funktion darstellen kann.

Definition 3.2 (Formeln der Aussagenlogik)**.** Die *Formeln* der Aussagenlogik sind Zeichenketten, die nach folgenden Regeln gebildet werden:

(i) Jedes Aussagensymbol ist eine Formel und auch $\perp$ ist eine Formel.

(ii) Ist ϕ eine Formel, dann auch $(\neg\phi)$.

(iii) Sind ϕ und ψ Formeln, dann auch $(\phi \wedge \psi)$, $(\phi \vee \psi)$ und $(\phi \rightarrow \psi)$

Bemerkungen

- In der Definition der Formeln der Aussagenlogik verwenden wir auch die Implikation, wenn wir z.B. sagen: „Ist ϕ eine Formel, dann auch $(\neg\phi)$“. Man muss also unterscheiden, ob wir *über* die Logik sprechen — man sagt dann auch wir verwenden die *Metasprache* — oder ob wir eine logische Formel angeben — dann verwenden wir die *Objektsprache*, die wir eben definiert haben.
- Die Symbole ϕ und ψ sind also *Variablen der Metasprache*, sie stehen als Platzhalter für *Formeln der Objektsprache*.
- Unsere Definition der Menge der Formeln der Aussagenlogik ist eine sogenannte *induktive* Definition.
- Eine Formel, die nur aus einem Aussagensymbol oder $\perp$ besteht, nennt man auch *atomare* Formel oder *Primformel*.

Als *Grammatik* in Backus-Naur-Darstellung[2] können wir diese induktive Definition der Formeln der Aussagenlogik so ausdrücken:

$$\phi ::= P \,|\, \bot \,|\, (\neg\phi) \,|\, (\phi \wedge \phi) \,|\, (\phi \vee \phi) \,|\, (\phi \to \phi)$$

mit Aussagensymbolen $P \in \mathcal{P}$ und (bereits gebildeten) Formeln ϕ.

Zu einer Formel ϕ kann man ihren Syntaxbaum bilden:

Definition 3.3. Als *Syntaxbaum* bezeichnen wir einen endlichen Baum, dessen Knoten mit Formeln beschriftet sind und der folgende Eigenschaften hat:

(i) Die Blätter sind mit Primformeln beschriftet.

(ii) Ist ein Knoten mit einer Formel der Form $(\neg\phi)$ beschriftet, dann hat er genau ein Kind, das mit ϕ beschriftet ist.

(iii) Ist ein Knoten mit einer Formel der Form $(\phi \wedge \psi)$, $(\phi \vee \psi)$ oder $(\phi \to \psi)$ beschriftet, dann hat er genau zwei Kinder und das linke ist mit ϕ, das rechte mit ψ beschriftet.

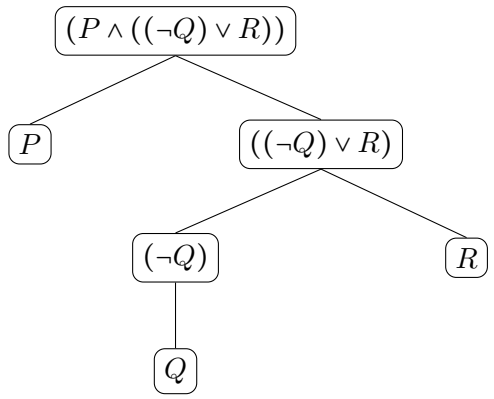
Ein Syntaxbaum repräsentiert die Formel, mit der die Wurzel des Baumes beschriftet ist.

Beispiel 3.1. Gegeben sei die Formel

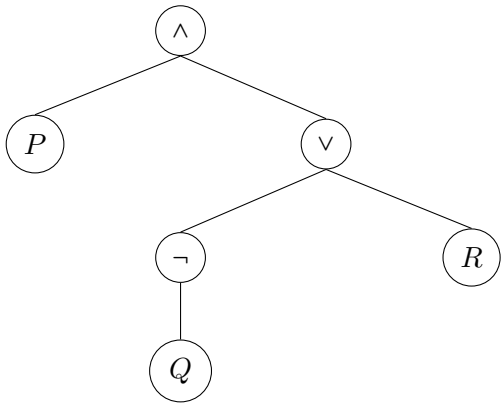
$$(P \wedge ((\neg Q) \vee R))$$

Der Syntaxbaum ist dann

[2] John W. Backus, amerikanischer Informatiker, 1942 - 2007; Peter Naur, dänischer Informatiker, 1928 - 2016 (Tatsächlich war Naur nicht erfreut über diese Bezeichnung, er hätte „Backus-Normalform“ vorgezogen.)



Wir verwenden oft die abgekürzte Darstellung des Syntaxbaums, in der die Knoten nicht mit den Subformeln bezeichnet werden, sondern mit dem Operator. Nur an den Blättern des Baums kommen dann die Primformeln vor. In unserem Beispiel also



Bemerkung Die Sprechweise „*der* Syntaxbaum einer Formel“ unterstellt, dass es nur *einen* solchen Baum zu einer gegebenen Formel gibt. Mit anderen Worten: die Grammatik aus der Definition der Formeln ist eindeutig.

Satz 3.1. *Jede Formel der Aussagenlogik hat genau einen Syntaxbaum.*

Zum Beweis von Aussagen über Formeln wendet man oft das Prinzip der *strukturellen Induktion* an.

Prinzip der strukturellen Induktion Sei $\mathcal{E}$ eine Eigenschaft. Dann gilt $\mathcal{E}(\phi)$ für alle Formeln ϕ, wenn gilt:

(i) $\mathcal{E}$ gilt für alle Primformeln und $\perp$.

(ii) Gilt $\mathcal{E}$ für ϕ, dann auch für $(\neg\phi)$.

(iii) Gilt $\mathcal{E}$ für ϕ und ψ, dann auch für $(\phi \wedge \psi)$, $(\phi \vee \psi)$ und $(\phi \to \psi)$.

Dieses Prinzip kann man anwenden, um obigen Satz zu beweisen.

Beweisidee Man beweist den Satz in folgenden Schritten:
Zunächst zeigt man: Jede Formel hat eine gerade Anzahl von Klammern und zwar ebensoviele öffende wie schließende Klammern.
Dann zeigt man: Jedes echte Anfangsstück einer Formel hat mehr öffnende als schließende Klammern.
Mit Hilfe dieser beiden Aussagen kann man dann den Satz beweisen.

Definition 3.4 (Subformel)**.** Eine *Subformel* einer Formel ϕ ist ein Formel, die als Beschriftung an einem Knoten des Syntaxbaums von ϕ vorkommt.

Definition 3.5 (Rang einer Formel)**.** Der *Rang* $\mathrm{rg}(\phi)$ einer Formel ist definiert durch

(i) $\mathrm{rg}(\phi) = 0$, wenn ϕ eine Primformel.

(ii) $\mathrm{rg}((\phi \ \square \ \psi)) = \max(\mathrm{rg}(\phi), \mathrm{rg}(\psi)) + 1$, wobei $\square$ für einen der Junktoren $\wedge, \vee, \to$ steht.

(iii) $\mathrm{rg}((\neg\phi)) = \mathrm{rg}(\phi) + 1$.

Der Rang einer Formel ist gerade die Tiefe des Syntaxbaums, d.h. die maximale Pfadlänge von der Wurzel zu einem beliebigen Blatt.

Die Eindeutigkeit des Syntaxbaums erlaubt es uns, Konventionen zu vereinbaren, mit denen man Klammern „sparen“ kann. Man vereinbart, dass die Junktoren in folgender Reihenfolge binden, die stärkste Bindung zuerst: $\neg, \wedge, \vee, \to$. Außerdem sind $\wedge$ und $\vee$ linksassoziativ; $\to$ ist rechtsassoziativ.

Bemerkung Die Definition der Syntax der Aussagenlogik verwendet die *Infix-Notation*, wie das in den Lehrbüchern über formale Logik üblich ist.

Man kann stattdessen die *Präfix-Notation* verwenden und hat dann folgende Grammatik:

$$\phi ::= P \,|\, \perp \,|\, (\neg\, \phi) \,|\, (\wedge\, \phi \ldots) \,|\, (\vee\, \phi \ldots) \,|\, (\to\, \phi\, \phi)$$

In der Präfix-Notation kann man die Junktoren $\wedge$ und $\vee$ als *n-äre* Junktoren definieren.

In der Logic Workbench (lwb), einer in Clojure geschriebenen Bibliothek von Funktionen für die Aussagen-, Prädikaten- und lineare temporale Logik wird Präfix-Notation mit folgenden Junktoren verwendet:

Tabelle 3.1: Junktoren für die Aussagenlogik in lwb

Junktor	**Beschreibung**	**Arität**
`not`	Negation	unär
`and`	Konjunktion	n-är
`or`	Disjunktion	n-är
`impl`	Implikation	binär
`equiv`	Äquivalenz	binär
`xor`	Exklusives Oder	binär
`ite`	If-then-else	ternär

Beispiel 3.2. Die Formel

$$(P \wedge ((\neg Q) \vee R))$$

in Infix-Notation wird in der Präfix-Notation der Logic Workbench so geschrieben:

```
(and P (or (not Q) R))
```

Diese Schreibweise entspricht genau dem Syntaxbaum der Formel.

Kapitel 4

Die Semantik der Aussagenlogik

In der klassischen Aussagenlogik wird jedem Aussagensymbol ein Wahrheitswert aus der Menge $\{\mathtt{T}, \mathtt{F}\}$ zugeordnet. ($\mathtt{T}$ steht für `true` und $\mathtt{F}$ für `false`, viele Autoren verwenden auch die Menge $\{1, 0\}$).

Die Bedeutung einer Formel der Aussagenlogik ergibt sich dann daraus, dass man diese Zuordnung von den Aussagensymbolen auf die Formel erweitert, in dem man definiert, welcher Wahrheitswert sich beim Zusammensetzen von Formeln durch die Junktoren ergibt. Auf diese Weise definiert man den Booleschen *Operator* zum jeweiligen Junktor.

Bemerkung Die Festlegung auf genau zwei Wahrheitswerte definiert eine *einfache* Logik, die man *klassische Logik* nennt. Man kann auch andere Festlegungen treffen wie:

- Łukasiewicź[1]-Logik mit den Wahrheitswerten $\{1, 1/2, 0\}$ oder $\{\mathtt{T}, \mathtt{U}, \mathtt{F}\}$. Łukasiewicź versteht den Wert $1/2$ als „nicht bewiesen, aber auch nicht widerlegt“, manchmal wird der dritte Wert $\mathtt{U}$ auch als „unbekannt“ interpretiert, wie zum Beispiel in SQL.
- Zadeh[2]-Logik mit Wahrheitswerten im Intervall $[0, 1]$; eine Logik mit dieser Semantik wird auch Fuzzy-Logik genannt.

Sei $\mathbb{B} = \{\mathtt{T}, \mathtt{F}\}$. Die Junktoren $\neg, \wedge, \vee, \rightarrow$ können als *Operatoren* auf der Menge $\mathbb{B}$, also als Boolesche Operatoren aufgefasst werden, in dem man die Verknüpfungstafeln wie folgt definiert:

[1] Jan Łukasiewicź, polnischer Philosoph, Mathematiker und Logiker 1878 - 1956.
[2] Lotfi A. Zadeh, amerikanischer Informatiker, 1921 - 2017

$\neg$

ϕ	$\neg\phi$
T	F
F	T

$\neg\phi$ genau dann `true`, wenn ϕ `false`.

$\wedge$

ϕ	ψ	$\phi \wedge \psi$
T	T	T
T	F	F
F	T	F
F	F	F

$\phi \wedge \psi$ ist genau dann `true`, wenn sowohl ϕ als auch ψ `true` sind.

$\vee$

ϕ	ψ	$\phi \vee \psi$
T	T	T
T	F	T
F	T	T
F	F	F

$\phi \vee \psi$ ist genau dann `true`, wenn einer der Operanden `true` ist. $\vee$ ist also ein einschließendes oder.

$\rightarrow$

ϕ	ψ	$\phi \rightarrow \psi$
T	T	T
T	F	F
F	T	T
F	F	T

$\phi \rightarrow \psi$ ist genau dann `true`, wenn ϕ `false` oder ψ `true` ist.

Bemerkung
Der Operator $\rightarrow$ wird auch als *materielle Implikation*[3] bezeichnet. Er behauptet *keinen* kausalen Zusammenhang zwischen der linken Seite, dem *Antezedens* und der rechten Seite, der *Konsequenz* oder dem *Sukzedens*.

Es seien die Aussagen

P = „Die Erde umkreist die Sonne“,
P' = „Die Sonne umkreist die Erde“ und
Q = „6 ist Primzahl“

gegeben.

$P \rightarrow Q$ hat den Wahrheitswert `F` – wie man vielleicht erwarten würde, auch wenn kein inhaltlicher Zusammenhang zwischen den beiden Aussagen besteht.

Hingegen hat $P' \rightarrow Q$ den Wahrheitswert `T` – denn das Antezedens ist `F`. Das kann man vielleicht so interpretieren: „Wenn die Sonne um die Erde kreisen würde, dann wäre 6 eine Primzahl“ – und da ja die Sonne nicht um die Erde kreist, können wir über die 6 sagen, was wir wollen.

[3] nach Bertrand Russell und Alfred North Whitehead: *Principia Mathematica* (1910).

Dahinter steckt ein klassisches logisches Prinzip: *ex falso quodlibet*[4] – „aus Falschem folgt alles, was beliebt“.

Der für uns wichtigere Grund für die Verwendung der materiellen Implikation ist jedoch, dass man mit dieser Definition der Implikation eine *einfache* Logik bekommt – in der zum Beispiel folgender Sachverhalt einfach ausdrückbar ist:

$$\text{Die Aussage } \forall x \in \mathbb{N} : (x > 3) \to (x > 1) \text{ ist } \mathtt{T}.$$

Setze ein:

$x = 4$	$(4 > 3)$	$\to (4 > 1)$	Ergebnis
	T	T	T
$x = 2$	$(2 > 3)$	$\to (2 > 1)$	Ergebnis
	F	T	T
$x = 0$	$(0 > 3)$	$\to (0 > 1)$	Ergebnis
	F	F	T

In allen drei Fällen ist die Aussage `true`.

Die materielle Implikation führt zu sogenannten Paradoxien, z.B.

(1) $P \to (Q \to P)$ ist allgemeingültig
„Wenn P gilt, folgt P aus allem“.

(2) $\neg P \to (P \to Q)$ ist allgemeingültig
„Wenn P nicht gilt, dann folgt alles aus P“.

4.1 Struktur/Modell

Sei $\mathcal{P}$ die Menge der Aussagensymbole und $\mathbb{B}$ die Menge der Wahrheitswerte.

Definition 4.1. Eine Abbildung $\alpha : \mathcal{P} \to \mathbb{B}$ nennt man eine *Struktur* (auch *Modell*, *Bewertung* oder *Interpretation*) für die Sprache der Aussagenlogik.

Definition 4.2. Zu einer Struktur $\alpha : \mathcal{P} \to \mathbb{B}$ und einer Formel ϕ definiert man den Wahrheitswert $[\![\phi]\!]_\alpha \in \mathbb{B}$ der Formel induktiv durch

[4] eigentlich „ex falso sequitur quodlibet“.

$$
\begin{array}{llll}
(i) & [\![P]\!]_\alpha & := \alpha(P) \text{ für alle } P \in \mathcal{P} \\
(ii) & [\![\bot]\!]_\alpha & := \mathtt{F} \\
(iii) & [\![(\neg\phi)]\!]_\alpha & := \begin{cases} \mathtt{T} \text{ falls } [\![\phi]\!]_\alpha = \mathtt{F} \\ \mathtt{F} \text{ sonst} \end{cases} \\
(iv) & [\![(\phi \wedge \psi)]\!]_\alpha & := \begin{cases} \mathtt{T} \text{ falls } [\![\phi]\!]_\alpha = \mathtt{T} \text{ und } [\![\psi]\!]_\alpha = \mathtt{T} \\ \mathtt{F} \text{ sonst} \end{cases} \\
(v) & [\![(\phi \vee \psi)]\!]_\alpha & := \begin{cases} \mathtt{T} \text{ falls } [\![\phi]\!]_\alpha = \mathtt{T} \text{ oder } [\![\psi]\!]_\alpha = \mathtt{T} \\ \mathtt{F} \text{ sonst} \end{cases} \\
(vi) & [\![(\phi \rightarrow \psi)]\!]_\alpha & := \begin{cases} \mathtt{T} \text{ falls } [\![\phi]\!]_\alpha = \mathtt{F} \text{ oder } [\![\psi]\!]_\alpha = \mathtt{T} \\ \mathtt{F} \text{ sonst} \end{cases}
\end{array}
$$

Bemerkung
Sei ϕ eine Formel und α, β seien Strukturen mit $\alpha(P) = \beta(P)$ für alle Aussagensymbole P in ϕ. Dann gilt:

$$[\![\phi]\!]_\alpha = [\![\phi]\!]_\beta.$$

Definition 4.3 (Erfüllbarkeit)**.** Eine Formel ϕ heißt *erfüllbar*, wenn es eine Struktur α gibt mit $[\![\phi]\!]_\alpha = \mathtt{T}$.

Definition 4.4 (Falsifizierbarkeit)**.** Eine Formel ϕ heißt *falsifizierbar*, wenn es eine Struktur α gibt mit $[\![\phi]\!]_\alpha = \mathtt{F}$.

Definition 4.5 (Allgemeingültigkeit)**.** Eine Formel ϕ heißt *allgemeingültig*, wenn für alle Strukturen α gilt: $[\![\phi]\!]_\alpha = \mathtt{T}$.

Man schreibt dann $\vDash \phi$ und nennt ϕ eine *Tautologie*.

Definition 4.6 (Unerfüllbarkeit)**.** Eine Formel ϕ heißt *unerfüllbar*, wenn für alle Strukturen α gilt: $[\![\phi]\!]_\alpha = \mathtt{F}$.

Man schreibt dann $\nvDash \phi$ und nennt ϕ eine *Kontradiktion*.

Satz 4.1 (Dualitätsprinzip)**.** *Eine Formel ϕ ist genau dann allgemeingültig, wenn $\neg\phi$ unerfüllbar ist.*

Beweis:
Sei ϕ allgemeingültig, d.h. für alle Strukturen α gilt $[\![\phi]\!]_\alpha = \mathtt{T}$, d.h. aber auch dass für alle α gilt $[\![\neg\phi]\!]_\alpha = \mathtt{F}$, d.h. $\neg\phi$ ist unerfüllbar. Und da $\phi \equiv \neg\neg\phi$ gilt auch die Umkehrung.

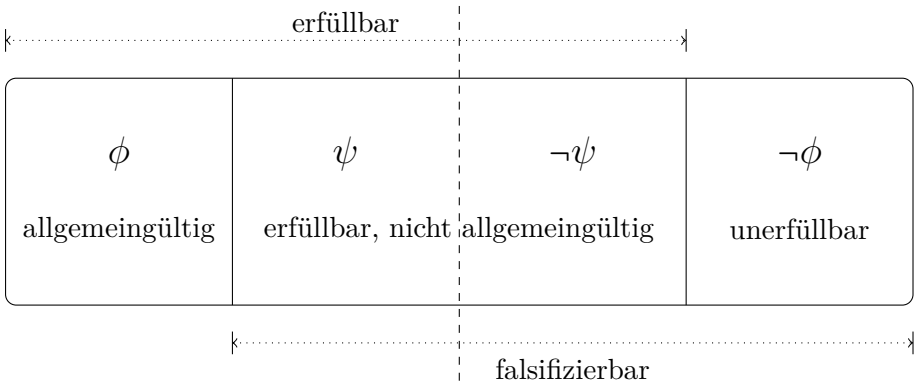


Abbildung 4.1: Dualitätsprinzip

4.2 Wahrheitstafel

Um zu prüfen, ob eine Formel allgemeingültig oder erfüllbar ist, kann man die Wahrheitstafel verwenden. Dabei geht man so vor:

1. Man ermittelt die Menge der Aussagensymbole der Formel. Für jede mögliche Belegung eines Symbols mit einem Wahrheitswert bildet man eine Zeile der Wahrheitstafel. Hat die Formel n verschiedene Aussagensymbole, hat die Wahrheitstafel also 2^n Zeilen. Jedes Symbol bekommt eine Spalte der Wahrheitstabelle.
2. Man bildet eine weitere Spalte der Wahrheitstafel, die mit der Formel beschriftet ist. In dieser Spalte überträgt man nun für jedes Symbol die Belegung der jeweiligen Zeile. Dann geht man entsprechend des zugehörigen Syntaxbaums von unten nach oben und ermittelt sukzessive die Wahrheitswerte der Subformeln, bis man beim Wahrheitswert der Formel angelangt ist.

Satz 4.2. *Die Methode des Aufstellens der Wahrheitstafel ist ein Entscheidungsverfahren für das* Erfüllbarkeitsproblem *und für das* Gültigkeitsproblem*.*

Eine Formel ϕ ist erfüllbar*, wenn es in der Wahrheitstafel* mindestens eine *Zeile mit dem Ergebnis* `T` *gibt.*

Eine Formel ϕ ist allgemeingültig*, wenn in der Wahrheitstafel* alle *Zeilen das Ergebnis* `T` *haben.*

4.3 Semantische Äquivalenz und Substitution

Definition 4.7 (Semantische Äquivalenz)**.** Zwei Formeln ϕ und ψ sind *semantisch äquivalent*, geschrieben $\phi \equiv \psi$, wenn für alle Interpretationen α gilt: $[\![\phi]\!]_\alpha = [\![\psi]\!]_\alpha$.

Definition 4.8 (Logische Konsequenz)**.** Sei Γ eine Menge von Formeln. Eine Formel ϕ heißt *logische Konsequenz* von Γ, geschrieben $\Gamma \vDash \phi$, wenn für alle Strukturen α gilt:

Ist $[\![\psi]\!]_\alpha = \mathtt{T}$ für alle $\psi \in \Gamma$, dann ist auch $[\![\phi]\!]_\alpha = \mathtt{T}$.

Definition 4.9 (Substitution)**.** Sei ϕ eine Subformel von ψ und ϕ' eine beliebige Formel. Dann ist $\psi[\phi'/\phi]$ (lese: ψ mit ϕ' an Stelle von ϕ) die Formel, die man erhält, wenn man jedes Vorkommen von ϕ in ψ durch ϕ' ersetzt (substituiert).

Satz 4.3 (Substitutionssatz)**.** *Sei ϕ eine Subformel von ψ und ϕ' eine semantisch äquivalente Formel, d.h. $\phi \equiv \phi'$, dann gilt:*

$\psi \equiv \psi[\phi'/\phi]$.

Wichtige semantische Äquivalenzen

Assoziativität
$(\phi \vee \psi) \vee \chi \equiv \phi \vee (\psi \vee \chi)$
$(\phi \wedge \psi) \wedge \chi \equiv \phi \wedge (\psi \wedge \chi)$

Kommutativität
$\phi \vee \psi \equiv \psi \vee \phi$
$\phi \wedge \psi \equiv \psi \wedge \phi$

Distributivität
$\phi \vee (\psi \wedge \chi) \equiv (\phi \vee \psi) \wedge (\phi \vee \chi)$
$\phi \wedge (\psi \vee \chi) \equiv (\phi \wedge \psi) \vee (\phi \wedge \chi)$

De Morgans Gesetze
$\neg(\phi \vee \psi) \equiv \neg\psi \wedge \neg\phi$
$\neg(\phi \wedge \psi) \equiv \neg\psi \vee \neg\phi$

Idempotenz
$\phi \vee \phi \equiv \phi$
$\phi \wedge \phi \equiv \phi$

Doppelte Negation
$\neg\neg\phi \equiv \phi$

Komplement
$\phi \wedge \neg\phi \equiv \bot$
$\phi \vee \neg\phi \equiv \top$

Identitätsgesetze
$\phi \wedge \top \equiv \phi$
$\phi \vee \bot \equiv \phi$

Absorption
$\phi \wedge (\phi \vee \psi) \equiv \phi$
$\phi \vee (\phi \wedge \psi) \equiv \phi$

Bemerkung Betrachtet man die Menge der Formeln der Aussagenlogik und bildet die Menge der Äquivalenzklassen bezüglich der semantischen Äquivalenz ($\equiv$), dann sagen obige Aussagen unter anderem, dass diese Menge eine *Boolesche Algebra* ist.

4.4 Boolesche Operatoren und funktionale Vollständigkeit

Die Anzahl der unären Operatoren auf $\mathbb{B}$ ist 4, die der binären Operatoren ist 16. Allgemein gilt:

Satz 4.4. *Für $n \in \mathbb{N}$ gibt es 2^{2^n} n-äre Boolesche Operatoren.*

Beweis:
Hat der Operator n Argumente, dann gibt es 2^n n-Tupel von möglichen verschiedenen Werten für die Argumente. Jede dieser Kombinationen kann als Ergebnis des Operators einen der beiden Wahrheitswerte `T` oder `F` haben, also gibt es 2^{2^n} Möglichkeiten.

Übung:
Erstellen Sie Tabellen für alle möglichen unären und binären Operatoren, finden Sie in der Literatur die gängigen Bezeichnungen und Symbole für die Operatoren.

Offenbar kann man Operatoren durch andere Operatoren ausdrücken, z.B.

$$\phi \leftrightarrow \psi \equiv (\phi \rightarrow \psi) \wedge (\psi \rightarrow \phi)$$

oder

$$\phi \rightarrow \psi \equiv \neg\phi \vee \psi$$

oder

$$\phi \wedge \psi \equiv \neg(\neg\phi \vee \neg\psi)$$

Definition 4.10. Eine Menge Boolescher Operatoren heißt *funktional vollständig*, wenn man jeden beliebigen Operator durch diese Operatoren logisch äquivalent ausdrücken kann.

Satz 4.5. *Für jeden n-ären Booleschen Operator gibt es eine äquivalente Formel, die nur die Operatoren $\neg$ und $\vee$ hat. D.h. $\{\neg, \vee\}$ ist* funktional vollständig.

Übung:
Beweisen Sie diesen Satz. Hinweis: Sehen Sie in das Skript von R. Stärk.

Kapitel 5

Das Beweissystem des natürlichen Schließens

Mit der Wahrheitstafel gibt es eine einfache Möglichkeit, festzustellen, ob eine Formel der Aussagenlogik erfüllbar ist. Man stellt die Wahrheitstafel auf und prüft, ob es eine Zeile mit dem Ergebnis `T` gibt.

Es gibt jedoch Beispiele, an denen man leicht sieht, dass diese Methode ihre Probleme hat. Nehmen wir etwa als Beispiel die folgende Formel:

$$(P_1 \wedge (P_1 \to P_2) \wedge \cdots \wedge (P_{n-1} \to P_n)) \to P_n$$

Diese Formel hat n Atome, die Wahrheitstafel also 2^n Zeilen.

Wir können jedoch auf eine andere Weise zu einer Lösung kommen. Wenn wir akzeptieren, dass die Schlussregel (man nennt sie *Modus ponens*)

$$\frac{\phi \quad \phi \to \psi}{\psi}$$

erlaubt ist, dann kann man so argumentieren:

P_1 und $P_1 \to P_2$ sind gegeben, also gilt P_2.

P_2 und $P_2 \to P_3$ sind gegeben, also gilt P_3.

...

P_{n-1} und $P_{n-1} \to P_n$ sind gegeben, also gilt P_n.

Und aus dieser Argumentation folgt, dass es sich bei der gegebenen Formel um eine Tautologie handelt.

In diesem Beispiel haben wir eine Schlussregel angewandt. Wichtig ist dabei zu sehen, dass sich damit die Perspektive gewandelt hat. In der Ermittlung der Wahrheitstafel wird die *Semantik* der Aussagenlogik verwendet: für jede mögliche Struktur (Belegung) wird der Wahrheitswert der Formel ermittelt. In der Argumentation mit der obigen Schlussregel wird kein Bezug auf die Semantik genommen, sondern die Schlussregel wird verwendet, um Transformation an den Symbolen vorzunehmen, die die Formel ausmachen.

Sei $\Gamma = \{\gamma_1, \gamma_2, \ldots, \gamma_n\}$ eine Menge von Formeln und ϕ eine Formel, dann gibt es zwei Sichten auf die Frage, ob ϕ aus Γ folgt:

- *Semantische Sicht*

  $$\Gamma \vDash \phi$$

  In allen Modellen (allen „möglichen Welten“), in denen $\gamma_1, \gamma_2, \ldots, \gamma_n$ gelten, ist auch ϕ wahr.

- *Syntaktische Sicht*

  $$\Gamma \vdash \phi$$

  Man kann aus den gegebenen Formeln $\gamma_1, \gamma_2, \ldots, \gamma_n$ die Formel ϕ herleiten, indem man ausschließlich die Regel des Beweissystems verwendet.

Es gibt für die Aussagenlogik (und für die Prädikatenlogik) verschiedene solcher Beweissysteme. Wir werden uns in der Veranstaltung mit dem *natürlichen Schließen* (auch *natürliche Deduktion*) befassen.

Das Kalkül des natürlichen Schließen wurde 1934 von Gerhard Gentzen[1] und unabhängig von ihm von Stanisław Jaśkowski[2] entwickelt.

Die Bezeichnung „Natürliches Schließen“ rührt daher, dass die Regeln des Kalküls das „natürliche“ Argumentieren von Mathematikern formalisieren.

[1] Gerhard Gentzen (1909–1945) deutscher Mathematiker und Logiker, [Gen35].
[2] Stanisław Jaśkowski (1906–1965), polnischer Logiker.

> „Mein erster Gesichtspunkt war folgender: Die Formalisierung des logischen Schließens, wie sie insbesondere durch Frege, Russell und Hilbert entwickelt worden ist, entfernt sich ziemlich weit von der Art des Schließens, wie sie in Wirklichkeit bei mathematischen Beweisen geübt wird. [...] Ich wollte nun zunächst einmal einen Formalismus aufstellen, der dem wirklichen Schließen möglichst nahekommt. So ergab sich ein ‚Kalkül des natürlichen Schließens'."[Gen35, S. 176].

5.1 Schlussregeln

Für die Herleitung von Formeln gibt es im natürlichen Schließen pro logischem Symbol zwei Regeln:

- eine, die das Symbol einführt (*Introduction*, abgekürzt durch i) und
- eine zweite, die das Symbol entfernt (*Elimination* auch: Auflösung), abgekürzt durch e).

Jede Regel gibt an, was *gegeben* sein muss (oberhalb des Strichs), damit die Umformung gemacht werden darf, also was sich aus dem Gegebenen *ergibt* (unterhalb des Strichs).

Konjunktion

Die Regeln für die Konjunktion sind:

	Einführung	*Elimination*	
$\wedge$	$\dfrac{\phi \quad \psi}{\phi \wedge \psi}$ $\wedge$i	$\dfrac{\phi \wedge \psi}{\phi}$ $\wedge e_1$	$\dfrac{\phi \wedge \psi}{\psi}$ $\wedge e_2$

Die Konjunktion kann man einführen, wenn man Herleitungen für die beiden Formeln der Konjunktion bereits hat.

Für die Elimination der Konjunktion gibt es zwei Subregeln: Eine Herleitung der Gesamtformel der Konjunktion kann man sowohl als Herleitung der linken Teilformel als auch der rechten Teilformel nehmen.

Disjunktion

Die Regeln für die Disjunktion sind:

	Einführung	*Elimination*
$\vee$	$\dfrac{\phi}{\phi \vee \psi}\ \vee\mathsf{i}_1$ $\qquad$ $\dfrac{\psi}{\phi \vee \psi}\ \vee\mathsf{i}_2$	$\dfrac{\phi \vee \psi \quad \boxed{\begin{array}{c}\phi\\ \vdots\\ \chi\end{array}} \quad \boxed{\begin{array}{c}\psi\\ \vdots\\ \chi\end{array}}}{\chi}\ \vee\mathsf{e}$

Wenn man eine Herleitung für ϕ hat, hat man auch eine Herleitung für $\phi \vee \psi$, ebenso darf man die Herleitung von ψ als Beweis für $\phi \vee \psi$ nehmen.

Will man die Disjunktion entfernen und dabei χ herleiten, muss man für jede Teilformel der Disjunktion eine Herleitung von χ finden. Diese Regel entspricht also der Beweistechnik der Fallunterscheidung.

Implikation

Die Regeln für die Implikation sind:

	Einführung	*Elimination*
$\rightarrow$	$\dfrac{\boxed{\begin{array}{c}\phi\\ \vdots\\ \psi\end{array}}}{\phi \rightarrow \psi}\ \rightarrow\mathsf{i}$	$\dfrac{\phi \quad \phi \rightarrow \psi}{\psi}\ \rightarrow\mathsf{e}, \mathsf{MP}$

Die Implikation leitet man her, indem man die Hypothese als gegeben annimmt und dann daraus die Folgerung herleitet. In der Regel wird in der Box oberhalb des Strichs angegeben, dass ϕ nur *innerhalb* der Box als gegeben angenommen werden darf. Die senkrechten Punkte $\vdots$ markieren die Beweisverpflichtung, nämlich dass sie durch einen Beweis ersetzt werden müssen, der ψ aus ϕ herleitet.

Die Implikation kann man entfernen, wenn man die Hypothese ϕ bewiesen hat und ebenso, dass $\phi \rightarrow \psi$ gilt. Dann hat man ψ bewiesen. Diese Schlussfigur ist schon seit der Antike geläufig und wird als *Modus ponens* bezeichnet, deshalb auch die Abkürzung MP.

Negation

	Einführung	*Elimination*
$\neg$	$\dfrac{\boxed{\begin{array}{c}\phi\\ \vdots\\ \bot\end{array}}}{\neg\phi}\ \neg\mathsf{i}$	$\dfrac{\phi \quad \neg\phi}{\psi}\ \neg\mathsf{e}, \mathsf{EFQ}$

Will man beweisen, dass $\neg\phi$ gilt — also die Negation einführen —, nimmt man an, dass ϕ bewiesen ist und führt diesen Beweis dann fort, bis man den Widerspruch $\bot$ hergeleitet hat. Daraus ergibt sich, dass $\neg\phi$ bewiesen ist.

Widerspruchsbeweis, EFQ

	Einführung	*Elimination*
$\mathsf{RAA}, \bot$	$\dfrac{\boxed{\begin{array}{c}\neg\phi\\ \vdots\\ \bot\end{array}}}{\phi}\ \mathsf{RAA}$	$\dfrac{\bot}{\phi}\ \bot\mathsf{e}, \mathsf{EFQ}$

Man kann eine Formel ϕ in der klassischen Logik auch durch einen Widerspruchsbeweis herleiten[3]. Man nimmt das Gegenteil von ϕ an und und führt diese Annahme zum Widerspruch. Die Regel sagt dann, dass nach diesem Widerspruchsbeweis ϕ bewiesen ist.

Dass aus dem Widerspruch jede beliebige Aussage folgt, wird auch als *Ex falso quodlibet* oder genauer *Ex falso sequitur quodlibet* bezeichnet.

5.2 Beispiele für das natürliche Schließen

Gentzens Beispiel

Als erstes Beispiel nehmen wir ein Beispiel aus der Arbeit von Gerhard Gentzen, mit der er das natürliche Schließen motiviert [Gen35].

Bewiesen werden soll die Formel

$$(X \vee (Y \wedge Z)) \to ((X \vee Y) \wedge (X \vee Z))$$

[3] In der *intuitionistischen* Logik ist diese Schlussregel nicht erlaubt.

Im folgenden Beweis geben die Angaben rechts die jeweils verwendete Regel an und die Zeile (oder Zeilen), auf die sie angewandt wurden.

1.	$(X \vee (Y \wedge Z))$	angenommen
2.	X	angenommen
3.	$(X \vee Y)$	$\vee\mathsf{i}_1$ 2
4.	$(X \vee Z)$	$\vee\mathsf{i}_1$ 2
5.	$((X \vee Y) \wedge (X \vee Z))$	$\wedge\mathsf{i}$ 3, 4
6.	$(Y \wedge Z)$	angenommen
7.	Y	$\wedge\mathsf{e}_1$ 6
8.	$(X \vee Y)$	$\vee\mathsf{i}_2$ 7
9.	Z	$\wedge\mathsf{e}_2$ [6]
10.	$(X \vee Z)$	$\vee\mathsf{i}_2$ 9
11.	$((X \vee Y) \wedge (X \vee Z))$	$\wedge\mathsf{i}$ 8, 10
12.	$((X \vee Y) \wedge (X \vee Z))$	$\vee\mathsf{e}$ 1, 2-5, 6- 11
13.	$(X \vee (Y \wedge Z)) \rightarrow ((X \vee Y) \wedge (X \vee Z))$	$\rightarrow\mathsf{i}$ 1-12

Abgeleitete Regeln der Aussagenlogik

Hat man eine Herleitung $P \vdash Q$, dann kann man sie in anderen Herleitungen wie eine Regel verwenden. Zwar führen wir den Beweis im natürlichen Schließen mit bestimmten Symbolen aus, da jedoch der Beweis selbst ganz unabhängig von der Wahl der speziellen Symbole wie etwa P ist, gilt er für *jedes* solche Symbol und kann deshalb selbst wie eine Regel verwendet werden.

Auf dem Merkblatt für die Regeln des natürlichen Schließens https://esb-dev.github.io/mat/rules.pdf sind vier solcher abgeleiteter Regeln aufgeführt, die im Folgenden bewiesen werden.

$$\frac{\phi}{\neg\neg\phi}\ \neg\neg\mathsf{i} \qquad\qquad \frac{\neg\neg\phi}{\phi}\ \neg\neg\mathsf{e}$$

$$\frac{\phi \rightarrow \psi \quad \neg\psi}{\neg\phi}\ \mathsf{MT} \qquad\qquad \frac{}{\phi \vee \neg\phi}\ \mathsf{TND}$$

Beweis für die Einführung der doppelten Negation:

1.	P	gegeben
2.	$\neg P$	angenommen
3.	$\bot$	$\neg$e 2, 1
4.	$\neg\neg P$	$\neg$i 2-3

Beweis für die Elimination der doppelten Negation:

1.	$\neg\neg P$	gegeben
2.	$\neg P$	angenommen
3.	$\bot$	$\neg$e 1, 2
4.	P	RAA 2-3

Beweis für Modus Tollens

1.	$P \to Q$	gegeben
2.	$\neg Q$	gegeben
3.	P	angenommen
4.	Q	$\to$ e 1, 3
5.	$\bot$	$\neg$e 2, 4
6.	$\neg P$	$\neg$i 3-5

Beweis für *Tertium Non Datur*

1.	$\neg(P \vee \neg P)$	angenommen
2.	P	angenommen
3.	$P \vee \neg P$	$\vee\mathsf{i}_1$ 2
4.	$\bot$	$\neg$e 1, 3
5.	$\neg P$	$\neg$i 2-4
6.	$P \vee \neg P$	$\vee\mathsf{i}_2$ 5
7.	$\bot$	$\neg$e 1, 6
8.	$P \vee \neg P$	RAA 1-7

Die Beweise für diese abgeleiteten Regel haben wir mit *konkreten* Formeln, in diesem Fall Primformeln wie P und Q, durchgeführt. Sie als Regel zu verwenden bedeutet jedoch, dass sie für *jede* beliebige Formel gelten, weshalb ja auch in der Formulierung der Regel die Symbole der Metasprache ϕ und ψ verwendet wurden. Da aber in den Beweisen selbst keinerlei spezifische Eingenschaften von P oder Q verwendet wurden, können wir die Beweise für beliebige Formeln abstrahieren — und somit wie Regeln verwenden.

Dies ist in der Logic WorkBench generell der Fall: ein im natürlichen Schließen geführter Beweis kann gespeichert werden (mit der Funktion `export` und dann in anderen Beweisen verwendet werden (siehe Wiki zum Natürlichen Schließen in lwb).

5.3 Beweisstrategien

In diesem Abschnitt wird an einem Beispiel erläutert, welche Strategien man beim Entwickeln einer Herleitung im natürlichen Schließen verwenden kann.

Als Beispiel nehmen wir:

$$P \to (Q \vee R) \vdash Q \vee (\neg P \vee R)$$

Der erste Schritt besteht immer darin, dass man die Voraussetzungen als gegeben hinschreibt und dann eine Lücke lässt, der das zu zeigende Ziel folgt. Die Lücke stellt die Beweisverpflichtung dar.

In unserem Beispiel führt dieser erste Schritt zu:

1. $P \to (Q \vee R)$ angenommen
2. $\vdots$
3. $Q \vee (\neg P \vee R)$

Im weiteren Vorgehen untersucht man, welche Regeln man anwenden kann. Dazu gibt es zwei Möglichkeiten:

- Bei den Zeilen oberhalb der Beweisverpflichtung gibt es eine Formel, für deren Haupt-Junktor eine Regel zur *Elimination* anwendbar ist. Dann kann man diese Regel *vorwärts* verwenden und erhält eine neue Zeile oberhalb der Beweisverpflichtung — oder man schließt den Beweis ab.
- Bei den Zeilen unterhalb der Beweisverpflichtung gibt es eine Formel, für deren Haupt-Junktor eine Regel zur *Einführung* angewandt werden kann. Dann kann man diese Regel *rückwärts* anwenden und erhält eine neue Zeile (oder einen Block).

In unserem Beispiel ist der Haupt-Junktor der Voraussetzung die Implikation. Um sie auflösen zu können, haben wir als Regel den Modus Ponens, der allerdings nur angewandt werden kann, wenn nicht nur $P \to (Q \vee R)$ gegeben ist, sondern auch P.

Betrachten wir also das Ziel. Dort ist der Haupt-Junktor die Disjunktion. Diese können wir einführen, wenn eine der Seiten gegeben ist. Das ist aber auch nicht der Fall.

In dieser Situation muss man zu anderen Waffen greifen. Wir können immer die Regel TND vorwärts und die Regel RAA rückwärts anwenden.

In unserem Beispiel scheint TND eine gute Wahl zu sein.

Dann haben wir folgende Situation:

1.	$P \to (Q \vee R)$	angenommen
2.	$P \vee \neg P$	TND
3.	$\vdots$	
4.	$Q \vee (\neg P \vee R)$	

In den folgenden Schritten hilft das Anwenden der Regeln für die Elimination von Junktoren bzw. der Einführung von Junktoren.

Wir verwenden die Regel zur Elimination der Disjunktion:

1.	$P \to (Q \vee R)$	angenommen
2.	$P \vee \neg P$	TND
3.	P	angenommen
4.	$\vdots$	
5.	$Q \vee (\neg P \vee R)$	
6.	$\neg P$	angenommen
7.	$\vdots$	
8.	$Q \vee (\neg P \vee R)$	
9.	$Q \vee (\neg P \vee R)$	$\vee$ e

Es ist jetzt nicht mehr schwierig, die beiden Beweisverpflichtungen zu erfüllen:

1.	$P \to (Q \vee R)$	angenommen
2.	$P \vee \neg P$	TND
3.	P	angenommen
4.	$Q \vee R$	MP 1, 3
5.	Q	angenommen
6.	$Q \vee (\neg P \vee R)$	$\vee$ i_1 5
7.	R	angenommen
8.	$\neg P \vee R$	$\vee$ i_2 7
9.	$Q \vee (\neg P \vee R)$	$\vee$ i_2 8
10.	$Q \vee (\neg P \vee R)$	$\vee$ e 4, 5-6, 7-9
11.	$\neg P$	angenommen
12.	$\neg P \vee R$	$\vee$ i_1 11
13.	$Q \vee (\neg P \vee R)$	$\vee$ i_2 12
14.	$Q \vee (\neg P \vee R)$	$\vee$ e 2, 3-10, 11-13

Man mag versucht sein, die genannten Strategie *schematisch* anzuwenden und einfach auszuprobieren, welche Regel anwendbar ist. Dies kann in einfachen Fällen zu einer Herleitung führen, trägt jedoch wenig zum *Verständnis* des bewiesenen Sachverhalts bei. Deshalb zum Schluß die

> ***Goldene Regel***
>
> Man muss sich die Aussage klarmachen, die man beweisen möchte und einen Beweis formulieren, ohne direkt an die Regeln zu denken, sondern eher wie die *Idee* des Beweises aussehen kann. Erst dann setzt man diese Idee in die Anwendung der Regeln um.

Werkzeuge wie die Logic Workbench eignen sich dann dafür zu überprüfen, ob die Umsetzung der Idee in einzelne Schritte korrekt durchgeführt wurde. Man vermeidet damit Flüchtigkeitsfehler.

5.4 Eigenschaften der Herleitbarkeit $\vdash$

In diesem Abschnitt betrachten wir Eigenschaften, die die Herleitbarkeit $\vdash$ mittels des natürlichen Schließens hat.

Im Folgenden seien stets Γ, Γ', Δ Mengen von Formeln der Aussagenlogik sowie ϕ und ψ einzelne Formeln der Aussagenlogik.

Lemma 5.1. *(Monotonie)*

$$\Gamma \vdash \phi \Rightarrow \Gamma \cup \Delta \vdash \phi$$

Beweis. Monotonie ist eine offensichtliche Eigenschaft von Herleitungen: Zu einer gegebenen Herleitung von ϕ aus Γ kann man weitere Voraussetzungen nach Belieben hinzufügen, sie werden ja für die Herleitung gar nicht notwendigerweise benötigt. □

Lemma 5.2.

(a) $\phi \in \Gamma \Rightarrow \Gamma \vdash \phi$

(b) $\Gamma \vdash \phi$ *und* $\Gamma' \vdash \psi \Rightarrow \Gamma \cup \Gamma' \vdash \phi \wedge \psi$

(c) $\Gamma \vdash \phi \wedge \psi \Rightarrow \Gamma \vdash \phi$ *und* $\gamma \vdash \psi$

(d) $\Gamma \cup \{\phi\} \vdash \psi \Rightarrow \Gamma \vdash \phi \to \psi$

(e) $\Gamma \vdash \phi$ *und* $\Gamma' \vdash \phi \to \psi \Rightarrow \Gamma \cup \Gamma' \vdash \psi$

(f) $\Gamma \vdash \bot \Rightarrow \Gamma \vdash \phi$

(g) $\Gamma \cup \{\neg\phi\} \vdash \bot \Rightarrow \Gamma \vdash \phi$

Beweis.

(a) Wenn $\phi \in \Gamma$ gilt, besteht die Herleitung von ϕ trivialerweise aus der Wiederholung von ϕ.

(b) Wegen Monotonie gilt $\Gamma \vdash \phi \Rightarrow \Gamma \cup \Gamma' \vdash \phi$ sowie $\Gamma' \vdash \psi \Rightarrow \Gamma \cup \Gamma' \vdash \psi$, also wegen der Regel $\wedge\mathsf{i}$ auch $\Gamma \cup \Gamma' \vdash \phi \wedge \psi$.

(c) Wenn man aus der Herleitung von $\phi \wedge \psi$ mit der Regel $\wedge\mathsf{e}_1$ bzw. der Regel $\wedge\mathsf{e}_2$ die Konjunktion eliminiert, erhält man $\Gamma \vdash \phi$ bzw. $\Gamma \vdash \psi$.

(d) Die Aussage gilt wegen der Regel $\to \mathsf{i}$.

(e) Wegen Monotonie gilt unter den gegebenen Voraussetzung $\Gamma \cup \Gamma' \vdash \phi$ sowie $\Gamma \cup \Gamma' \vdash \phi \to \psi$, also $\Gamma \cup \Gamma' \vdash \psi$ wegen der Regel $\to \mathsf{e}$, dem Modus Ponens.

(f) Die Aussage gilt wegen der Regel EFQ.

(g) Eine Herleitung für $\Gamma \vdash \phi$ ergibt sich durch die Regel RAA, nach deren Anwendung man die gegebene Herleitung einsetzt und den Widerspruch erhält.

□

Satz 5.1. *(Endlichkeitssatz)*

Gilt $\Gamma \vdash \phi$, dann gibt es eine endliche Teilmenge $\Gamma' \in \Gamma$, für die $\Gamma' \vdash \phi$ gilt.

Beweis. Wenn $\phi \in \Gamma$ ist, dann reicht $\Gamma' = \{\phi\}$ für die Herleitung aus.
Für die Induktion über die Länge von Herleitungen setzen wir voraus, dass die Aussage nun für alle Herleitungen der Länge $n-1$ gelte.
Für eine Herleitung der Länge n, betrachten wir die letzte angewandte Regel. Sie braucht nur endlich viele vorausgesetzte Aussagen, also folgt die Behauptung durch natürliche Induktion. □

Definition 5.1. (Konsistenz) Eine Menge Γ von Formeln der Aussagenlogik ist *konsistent*, wenn $\Gamma \nvdash \bot$, *inkonsistent* andernfalls.

Lemma 5.3. *Folgende Aussagen sind äquivalent:*

(i) Γ *ist konsistent*

(ii) *Es gibt kein* ϕ *mit* $\Gamma \vdash \phi$ *und* $\Gamma \vdash \neg\phi$

(iii) *Es gibt mindestens eine Formel* ϕ *mit* $\Gamma \vdash \neg\phi$

Das Lemma lässt sich auch so formulieren:

Folgende Aussagen sind äquivalent:

(i') Γ *ist inkonsistent*

(ii') *Es gibt ein* ϕ *mit* $\Gamma \vdash \phi$ *und* $\Gamma \vdash \neg\phi$

(iii') *Für alle* ϕ *gilt* $\Gamma \vdash \phi$

Beweis.

(i') $\Rightarrow$ (iii') Sei ϕ eine beliebige Formel, da Γ inkonsistent ist, gilt $\Gamma \vdash \phi$ nach Lemma 5.2 (f).

(iii') $\Rightarrow$ (ii') Da *alle* Formeln herleitbar sind, gibt es ein ϕ mit $\Gamma \vdash \phi$ sowie $\Gamma \vdash \neg\phi$.

(ii') $\Rightarrow$ (i') Es gibt nach Voraussetzung ein ϕ mit $\Gamma \vdash \phi$ und $\Gamma \vdash \neg\phi$, d.h. $\Gamma \vdash \phi \wedge \neg\phi$, also wegen der Regel $\neg\,\mathsf{e}$ auch $\Gamma \vdash \bot$.

□

Lemma 5.4.

(i) $\Gamma \vdash \phi \Leftrightarrow \Gamma \cup \{\neg\phi\} \vdash \bot$

(ii) $\Gamma \vdash \neg\phi \Leftrightarrow \Gamma \cup \{\phi\} \vdash \bot$

Beweis.

(i) $\Rightarrow$) Wenn $\Gamma \vdash \phi$ gilt und zusätzlich die Voraussetzung $\neg\phi$, kann man mit der Regel $\neg\,\mathsf{e}$ den Widerspruch herleiten.
$\Leftarrow$) siehe Lemma 5.2 (g).

(ii) $\Rightarrow$) Wenn $\Gamma \vdash \neg\phi$ gilt und zusätzlich die Voraussetzung ϕ, kann man mit der Regel $\neg\,\mathsf{e}$ den Widerspruch herleiten.
$\Leftarrow$) Eine Herleitung für $\Gamma \vdash \neg\phi$ ergibt sich durch die Regel $\neg\,\mathsf{i}$, nach deren Anwendung man die gegebene Herleitung einsetzt und den Widerspruch erhält.

□

Definition 5.2. (Maximal konsistente Formelmenge) Eine Formelmenge Γ ist *maximal konsistent*, wenn sie konsistent ist, aber jede echte Obermenge $\Gamma' \supset \Gamma$ inkonsistent ist.

Lemma 5.5. *Eine maximal konsistente Menge* Γ *ist* abgeschlossen *bezüglich Herleitbarkeit, d.h.*

$$\Gamma \vdash \phi \Rightarrow \phi \in \Gamma \ \textit{ für alle } \ \phi.$$

Beweis. Es gelte für eine maximal konsistente Menge Γ und eine Formel ϕ: $\Gamma \vdash \phi$. Angenommen $\phi \notin \Gamma$, dann ist $\Gamma \cup \{\phi\}$ inkonsistent, weil Γ maxiaml konsistent ist. Also gilt nach Lemma 5.4 $\Gamma \vdash \neg\phi$. D.h. es gibt ein ϕ mit $\Gamma \vdash \phi$ und $\Gamma \vdash \neg\phi$, d.h. Γ ist inkonsistent. □

Lemma 5.6. *Eine maximal konsistente Menge* Γ *ist* negationstreu*, d.h. für eine beliebige Formel* ϕ *gilt*

$$\textit{entweder } \ \Gamma \vdash \phi \ \textit{ oder } \ \Gamma \vdash \neg\phi.$$

.

Beweis. Gilt $\Gamma \vdash \phi$, dann kann wegen der Konsistenz von Γ nicht $\Gamma \vdash \neg\phi$ gelten.
Gilt $\Gamma \nvdash \phi$, dann ist $\Gamma \cup \{\neg\phi\}$ konsistent. Da Γ maximal konsistent ist, folgt dann $\neg\phi \in \Gamma$, und somit $\Gamma \vdash \neg\phi$. □

5.5 Vollständigkeit des natürlichen Schließens

siehe Vorlesung

Kapitel 6

Normalformen

6.1 Negationsnormalform NNF

Wir beobachten zunächst, dass man jede Formel äquivalent so umformen kann, dass sie keine Implikationen mehr enthält. Dazu verwendet man die Äquivalenz $\phi \to \psi \equiv \neg\phi \vee \psi$.

Definition 6.1 (Literal)**.** Ein *Literal* ist eine Primaussage P oder ihre Negation $\neg P$.

Wir formulieren einen Algorithmus für die Elimination der Implikation aus aussagenlogischen Formeln:

```
function IMPL_FREE(φ) {
// pre: beliebige Formel φ
// post: äquivalente Umformung von φ, die kein → mehr enthält
    case {
        φ ist Literal:
            return φ;
        φ hat die Form ¬φ1:
            return ¬IMPL_FREE(φ1);
        φ hat die Form φ1 ∧ φ2:
            return IMPL_FREE(φ1) ∧ IMPL_FREE(φ2);
        φ hat die Form φ1 ∨ φ2:
            return IMPL_FREE(φ1) ∨ IMPL_FREE(φ2);
        φ hat die Form φ1 → φ2:
            return ¬IMPL_FREE(φ1) ∨ IMPL_FREE(φ2);
    }
}
```

Definition 6.2 (Negationsnormalform NNF)**.** Eine Formel ϕ ohne Implikation ist in der *Negationsnormalform NNF*, wenn jede Negation direkt vor einer Primaussage steht.

Beispiele

$\neg\neg\neg P$	nicht NNF	$\neg P$	NNF
$\neg(P \wedge Q)$	nicht NNF	$\neg P \vee \neg Q$	NNF
$\neg(P \vee Q)$	nicht NNF	$\neg P \wedge \neg Q$	NNF

Wir formulieren einen Algorithmus, der eine Formel in die Negationsnormalform bringt:

function $\mathsf{NNF}(\phi)$ {
// pre: ϕ hat keine Implikationen
// post: äquivalente Umformung von ϕ in NNF
    case {
        ϕ ist Literal:
            return ϕ;
        ϕ hat die Form $\neg\neg\phi_1$:
            return $\mathsf{NNF}(\phi_1)$;
        ϕ hat die Form $\phi_1 \wedge \phi_2$:
            return $\mathsf{NNF}(\phi_1) \wedge \mathsf{NNF}(\phi_2)$;
        ϕ hat die Form $\phi_1 \vee \phi_2$:
            return $\mathsf{NNF}(\phi_1) \vee \mathsf{NNF}(\phi_2)$;
        ϕ hat die Form $\neg(\phi_1 \wedge \phi_2)$:
            return $\mathsf{NNF}(\neg\phi_1 \vee \neg\phi_2)$;
        ϕ hat die Form $\neg(\phi_1 \vee \phi_2)$:
            return $\mathsf{NNF}(\neg\phi_1 \wedge \neg\phi_2)$;
    }
}

6.2 Konjunktive Normalform CNF

Definition 6.3 (Klausel)**.** Eine Formel der Form $\phi = \hat{P}_1 \vee \hat{P}_2 \vee \cdots \vee \hat{P}_n$ mit Literalen $\hat{P}_i$ nennt man eine *Klausel*.

Beispiele:
$P \vee \neg Q \vee \neg R$ ist eine Klausel
$\neg P_1 \vee \neg P_2 \vee \cdots \vee \neg P_n \vee Q$ ist eine Klausel,
sie ist äquivalent zu
$P_1 \wedge P_2 \wedge \cdots \wedge P_n \rightarrow Q$.

Bemerkung: Oft stellt man Klauseln als Mengen von Literalen dar, wobei mit $\bar{P}$ die Negation $\neg P$ einer Primaussage bezeichnet wird. Der Grund dafür besteht darin, dass logischen Operationen mit Formeln in konjunktiver Normalform und Klauseln mengentheoretischen Operationen auf Klauselmengen und Klauseln entsprechen.

Beispiele: Korrespondierend zu obigen Beispielen
$\{P, \bar{Q}, \bar{R}\}$
$\{\bar{P}_1, \bar{P}_2, \ldots, \bar{P}_n, Q\}$

Definition 6.4 (Konjunktive Normalform CNF)**.** Eine Formel ϕ ist in der *konjunktiven Normalform CNF*, wenn sie die Form $\phi_1 \wedge \phi_2 \wedge \cdots \wedge \phi_n$ hat mit lauter Klauseln ϕ_i.

Wir formulieren einen Algorithmus, der eine Formel in die konjunktive Normalform bringt:

```
function DISTR(φ1, φ2) {
// pre: φ1 und φ2 sind in CNF
// post: berechnet CNF für φ1 ∨ φ2
    case {
        φ1 hat die Form φ11 ∧ φ12:
            return DISTR(φ11, φ2)∧ DISTR(φ12, φ2);
        φ2 hat die Form φ21 ∧ φ22:
            return DISTR(φ1, φ21)∧ DISTR(φ1, φ22);
        default:
            return φ1 ∨ φ2;
    }
}
```

Mit Hilfe der Hilfsfunktion DISTR ist es nun einfach, einen Algorithmus für das Erzeugen der CNF zu formulieren:

```
function CNF(φ) {
// pre: φ ist in NNF
// post: eine zu φ äquivalente Formel in CNF
    case {
        φ ist Literal:
            return φ;
        φ hat die Form φ1 ∧ φ2:
            return CNF(φ1) ∧ CNF(φ2);
        φ hat die Form φ1 ∨ φ2:
            return DISTR(CNF(φ1), CNF(φ2));
```

```
    }
}
```

6.3 Disjunktive Normalform DNF

Definition 6.5 (Monom)**.** Eine Formel der Form $\psi = \hat{P}_1 \wedge \hat{P}_2 \wedge \cdots \wedge \hat{P}_n$ mit Literalen $\hat{P}_i$ bezeichnet man als *Monom*.

Definition 6.6 (Disjunktive Normalform DNF)**.** Eine Formel ψ ist in der *disjunktiven Normalform DNF*, wenn sie die Form $\psi_1 \vee \psi_2 \vee \cdots \vee \psi_n$ hat mit lauter Monomen ψ_i.

Beobachtung: („Dualität“ von CNF und DNF)
Ist die Formel ϕ in CNF, dann ist $\neg\phi$ modulo simpler Transformationen in DNF.

Ist ϕ in CNF, dann hat ϕ die Form $\phi_1 \wedge \phi_2 \wedge \cdots \wedge \phi_n$ mit Klauseln ϕ_i.
Betrachte nun $\neg\phi$, also $\neg(\phi_1 \wedge \phi_2 \wedge \cdots \wedge \phi_n)$.
Nach De Morgan ist dies äquivalent zu $\neg\phi_1 \vee \neg\phi_2 \vee \cdots \vee \neg\phi_n$.
Wendet man nun De Morgan auch auf die Klauseln ϕ_i an und eliminiert doppelte Negation, dann erhält man Monome und die Formel $\neg\phi$ ist in DNF.

6.4 Normalformen und Entscheidungsprobleme

Ein *komplementäres* Paar von Literalen ist eine Primaussage samt seiner Negation, also z.B. P und $\neg P$.

CNF und Gültigkeit

Satz 6.1 (CNF und Gültigkeit)**.** *Sei ϕ in CNF. Dann gilt:*
ϕ ist allgemeingültig $\Leftrightarrow$ Jede Klausel enthält ein komplementäres Paar von Literalen

DNF und Erfüllbarkeit

Satz 6.2 (DNF und Erfüllbarkeit)**.** *Sei ψ in DNF. Dann gilt:*
ψ ist unerfüllbar $\Leftrightarrow$ Jedes Monom enthält ein komplementäres Paar von Literalen

Übung Vorausgesetzt ϕ ist eine Tautologie in konjunktiver Normalform CNF, wie kann man dann daraus eine Herleitung von ϕ nach den Regeln des natürlichen Schließens *konstruieren*? Diese Übung ergibt einen anderen Beweis der Vollständigkeit des natürlichen Schließens (siehe 5.5).

Kapitel 7

Die Komplexität des Erfüllbarkeitsproblems

7.1 Das Erfüllbarkeitsproblem

Entscheidungsfragen der Aussagenlogik

In der Aussagenlogik kann man sich zu einer gegebenen Formel ϕ folgende Fragen stellen:

- Ist ϕ *allgemeingültig*?
  Diese Frage nennt man auch das Gültigkeitsproblem. Die Formel ϕ ist allgemeingültig, wenn sie für jede beliebige Interpretation der Aussagensymbole wahr ist. Ist eine Formel ϕ allgemeingültig, nennt man ϕ auch eine *Tautologie*.
- Ist ϕ *unerfüllbar*?
  Diese Frage nennt man auch das Unerfüllbarkeitsproblem. Die Formel ϕ ist unerfüllbar, wenn es keine Interpretation der Aussagensymbole gibt, die sie wahr macht. Ist eine Formel ϕ unerfüllbar, sagt man auch ϕ ist der *Widerspruch* oder eine *Kontradiktion*.
- Ist ϕ *erfüllbar*?
  Diese Frage nennt man auch das Erfüllbarkeitsproblem. Die Formel ϕ ist erfüllbar, wenn es eine Interpretation gibt, unter der die Formel wahr ist. In der Regel ist man dann natürlich auch interessiert daran, ein solches Modell zu finden.
- Ist ϕ *falsifizierbar*?
  Diese Frage nennt man auch das Widerlegungsproblem. Die Formel ϕ ist falsifizierbar oder widerlegbar, wenn es eine Interpretation gibt, unter der die Formel falsch ist. Auch dann möchte man üblicherweise wissen, für welches Modell die Formel falsch ist.

Diese Fragen hängen miteinander zusammen. Denn aus den Definitionen ergibt sich unmittelbar:

- Eine Formel ϕ ist genau dann allgemeingültig, wenn $\neg\phi$ unerfüllbar ist.
- Eine Formel ϕ ist genau dann falsifizierbar, wenn $\neg\phi$ erfüllbar ist.

Also genügt es, die Frage zu betrachten, ob eine Formel *erfüllbar* ist. Wenn ja, unter welcher Belegung?

Das Erfüllbarkeitsproblem

Die Frage, ob eine Formel erfüllbar ist, wird als das *Erfüllbarkeitsproblem*, auch *SAT-Problem* (SAT = *satisfiability*) oder ganz kurz *SAT*, bezeichnet.

Es ist nicht schwierig, das Erfüllbarkeitsproblem zu lösen. Hat man eine Formel ϕ, dann stellt man die Wahrheitstafel für die Formel auf und sieht nach, ob es eine Zeile der Wahrheitstafel gibt, in der die Formel wahr ist.

Dieses Vorgehen ist jedoch nur für Formeln mit wenig Atomen geeignet. Hat ϕ n Atome, dann hat die Wahrheitstafel 2^n Zeilen. Hat also zum Beispiel eine Formel 100 Atome und die Untersuchung einer Zeile der Wahrheitstafel dauert 10^{-10} Sekunden, dann dauert die Untersuchung der gesamten Wahrheitstafel $2^{100} \times 10^{-10}$ Sekunden, also etwa 4×10^{12} Jahre. „Das ist länger als die Zeit, die seit der Entstehung des Universums vergangen ist.“[1]

7.2 Komplexität von Algorithmen

Ein *Algorithmus* ist ein Verfahren, das in endlich vielen Schritten zur Lösung eines Problems führt. Mit dieser informellen Definition eines Algorithmus wollen wir im Folgenden arbeiten.[2]

Die *Komplexität* eines Algorithmus wird gemessen durch die Menge an Ressourcen (Zeit oder Speicher), die für die Durchführung des Algorithmus im Prinzip benötigt wird. Wir werden die Laufzeit eines Algorithmus im Folgenden als seine Komplexität betrachten.

[1] Die beispielhafte Berechnung der Ineffizienz der Entscheidung des Erfüllbarkeitsproblems durch die Wahrheitstafel ist aus: Uwe Schöning: *Das SAT-Problem* in: Informatik Spektrum Band 33 Heft 5 Oktober 2010

[2] Präziseres findet man in: Michael Sipser: *Introduction to the Theory of Computation*, Kap. 3.3 The Definition of Algorithm

Arten von Algorithmen

Definition 7.1. Ein Algorithmus ist *deterministisch*, wenn die Berechnung und somit das Ergebnis vollständig durch die Eingabe bestimmt wird. Ein deterministischer Algorithmus ist genau dann *korrekt*, wenn das Ergebnis zur gegebenen Eingabe korrekt ist.

Beispiel 7.1. Der Algorithmus mittels der Wahrheitstafel die Erfüllbarkeit einer Formel zu berechnen ist ein deterministischer Algorithmus für das Erfüllbarkeitsproblem.

Definition 7.2. Ein Algorithmus ist *nichtdeterministisch*, wenn seine Schritte mehrere mögliche Folgeschritte haben, deren Wahl nicht durch die Eingabe und bisherige Berechnung bestimmt wird. Zu einer Eingabe sind also mehrere verschiedene Berechnungsergebnisse möglich. Ein nichtdeterministischer Algorithmus heißt *korrekt*, wenn unter den möglichen Ergebnissen wenigstens eines korrekt ist.

Beispiel 7.2. Ein nichtdeterministischer Algorithmus für das Erfüllbarkeitsproblem ist leicht zu definieren. Der erste Schritt besteht darin, eine Interpretation der Aussagensymbole zu raten. Im zweiten Schritt wird die Formel berechnet.
Dieser Algorithmus ist sogar ein korrekter nichtdeterministischer Algorithmus. Wenn die Formel erfüllbar ist, dann gibt es eine erfüllende Interpretation. Rät der Algorithmus mal richtig, dann entscheidet er das Erfüllbarkeitsproblem korrekt.

Laufzeit von Algorithmen

Nun definieren wir Maße für die Komplexität von Algorithmen:

Definition 7.3. Ein Algorithmus heißt *polynomiell*, wenn seine Laufzeit nach oben durch ein Polynom in n (n ist die Größe der Eingabe) beschränkt ist. Algorithmen, die in polynomieller Zeit ein korrektes Ergebnis liefern, werden oft auch als *effiziente* Algorithmen bezeichnet.

Beispiel 7.3. Unser nichtdeterministischer Algorithmus durch Erraten das Erfüllbarkeitsproblem zu lösen ist polynomiell: Das Erraten einer Belegung ist linear bezüglich der Größe der Eingabe, d.h. der Anzahl n der Atome und das Überprüfen, ob das Erratene zutrifft ist offensichtlich effizient durchführbar.

Definition 7.4. Ein Algorithmus läuft in *exponentieller* Zeit, wenn die Laufzeit nach unten durch eine Funktion 2^{cn} für die Größe der Eingabe n und ein $c > 0$ beschränkt ist.

Beispiel 7.4. Die Methode mit der Wahrheitstafel die Erfüllbarkeit einer Formel zu entscheiden ist exponentiell: Die Größe der Eingabe ist die Zahl n der Atome der Formel. Im schlechtesten Fall muss man zur Entscheidung alle 2^n Zeilen der Wahrheitstafel konstruieren.

Klassen von Problemen und $\mathcal{NP}$-Vollständigkeit

Definition 7.5. Die Klasse $\mathcal{P}$ bezeichnet die Probleme, die in polynomieller Zeit durch einen deterministischen Algorithmus gelöst werden können.

Beispiel 7.5. Es ist nicht bekannt, ob das Erfüllbarkeitsproblem zur Klasse $\mathcal{P}$ gehört. Es wird vermutet, dass dies nicht der Fall ist, wie wir gleich genauer diskutieren werden.

Definition 7.6. Die Klasse $\mathcal{NP}$ bezeichnet die Probleme, die ein nicht-deterministischer Algorithmus in polynomieller Zeit lösen kann.

Beispiel 7.6. Das Erfüllbarkeitsproblem gehört zur Klasse $\mathcal{NP}$. Man sagt auch: SAT ist $\mathcal{NP}$.

Man kann auch so ausdrücken, dass ein Problem in $\mathcal{NP}$ ist: eine *Lösung* des Problems ist in polynomieller Zeit *überprüfbar*, auch wenn sie möglicherweise nicht in polynomieller Zeit *gefunden* werden kann.

Vermutung: Ist $\mathcal{P} = \mathcal{NP}$? Dies ist eine grundlegende Frage der Informatik, die ungelöst ist.[3] Allgemein wird vermutet, dass gilt:

$$\mathcal{P} \neq \mathcal{NP}.^{4}$$

Zur genaueren Bestimmung der Komplexität des Erfüllbarkeitsproblems benötigen wir noch weitere Definitionen:

Definition 7.7. Ein Problem P ist $\mathcal{NP}$-schwierig, wenn sich jedes Problem Q in $\mathcal{NP}$ deterministisch in polynomieller Zeit auf P reduzieren lässt.

[3] Das Clay Mathematics Institute http://www.claymath.org hat dieses Problem als eines der 7 Millenniums-Probleme ausgewählt – neben u.a. der Riemann-Vermutung oder der (mittlerweile von Grigori Perelman gelösten) Poincaré-Vermutung.

[4] Donald E. Knuth sieht das anders: „. . . almost everybody who has studied the subject thinks that satisfiability cannot be decided in polynomial time. The author of this book, however, suspects that $N^{O(1)}$-step algorithms do exist, yet that they're unknowable. Almost all polynomial time algorithms are so complicated that they lie beyond human comprehension, and could never be programmed for an actual computer in the real world. Existence is different from embodiment.“ [Knu15, S. 1]

Definition 7.8. Ein Problem P ist $\mathcal{NP}$-vollständig, wenn es in $\mathcal{NP}$ und $\mathcal{NP}$-schwierig ist.

7.3 Die Komplexität des Erfüllbarkeitsproblems

Nun stehen alle Definitionen zur Verfügung, um den Satz zu formulieren, der die Komplexität des Erfüllbarkeitsproblems bestimmt:

Satz 7.1 (Cook 1971, Levin 1973)**.** *Das Erfüllbarkeitsproblem ist $\mathcal{NP}$-vollständig.*[5]

Wie sieht es mit dem komplementären Problem der Nichterfüllbarkeit bzw. der Allgemeingültigkeit aus? Dieses Problem ist insofern schwieriger, als man ja zeigen muss, dass es kein Modell gibt, bzw. dass die Aussage in allen Interpretationen wahr ist. Hier hilft „Raten“ nicht mehr. Genauer sagt man:

Definition 7.9. Ein Problem ist in der Klasse $Co\text{-}\mathcal{NP}$, wenn das komplementäre Problem in $\mathcal{NP}$ ist.

Beispiel: Das Nichterfüllbarkeitsproblem ist in $Co\text{-}\mathcal{NP}$.

Satz 7.2. *$Co\text{-}\mathcal{NP} = \mathcal{NP}$ genau dann, wenn Nichterfüllbarkeit in $\mathcal{NP}$ ist.*

Vermutung: Es ist nicht bekannt, ob es einen nichtdeterministischen polynomiellen Algorithmus für das Nichterfüllbarkeitsproblem gibt. Man nimmt vielmehr an, dass gilt:

$$Co\text{-}\mathcal{NP} \neq \mathcal{NP}$$

Die Ergebnisse über die Komplexität des Erfüllbarkeitsproblems können zu der Annahme verleiten, dass es keinen „effizienten“ Algorithmus für SAT geben kann und deshalb Probleme, die man als aussagenlogische Formeln formulieren kann, nur gelöst werden können, wenn man wenige aussagenlogischen Atome in der Formel hat.

[5] Stephen A. Cook, amerikanischer Informatiker, heute Professor für Informatik in Toronto. Er erhielt 1982 für diesen Satz den Turing-Award.
Leonid Levin, ukrainischer Informatiker, hat die Theorie der $\mathcal{NP}$-Vollständigkeit und den Satz über das Erfüllbarkeitsproblem 1973 entwickelt. Seine Ergebnisse waren im Westen zunächst nicht bekannt. 1978 emigrierte er in die USA.

Dies ist jedoch keineswegs der Fall: „Zum Glück gibt es und gab es Algorithmenentwickler, Theoretiker und Praktiker, die sich von diesem Negativergebnis [Satz von Cook und Levin] nicht abschrecken ließen. Dies hat in den vergangenen Jahren dazu geführt, dass die ‚SAT-Solver'-Technologie immer weiter vorangeschritten ist [und] dass diese das SAT-Problem lösenden Verfahren heutzutage mit Formeln, die Tausende von Variablen enthalten, in Sekundenbruchteilen fertig werden."[6]

[6] So schreibt Uwe Schöning im bereits zitierten Artikel. Ein solcher SAT-Solver (eigentlich eine Familie von SAT-Solvern) ist `Sat4j` – siehe http://www.sat4j.org/.

Kapitel 8

Hornlogik

Betrachtet man spezielle Klassen von Formeln, dann findet man oft effiziente Algorithmen für die Entscheidung des Erfüllbarkeitsproblems. Eine wichtige solche Klasse sind die Hornformeln[1].

Definition 8.1 (Hornklausel)**.** Eine Klausel $\hat{P}_1 \vee \hat{P}_2 \vee \cdots \vee \hat{P}_n$ heißt *Hornklausel*, wenn höchstens eines der Literale $\hat{P}_i$ positiv ist.

Definition 8.2 (Hornformel)**.** Eine Formel ϕ in CNF heißt *Hornformel*, wenn jede Klausel eine Hornklausel ist.

Definition 8.3 (Arten von Hornklauseln)**.**

(a) Besteht eine Hornklausel nur aus einem positiven Literal, dann heißt sie *Tatsachenklausel*, kurz *Tatsache*.

(b) Hat eine Hornklausel ein positives Literal und mindestens ein negatives Literal, dann heißt sie *Prozedurklausel* oder *Regel*.

(c) Hat die Hornklausel kein positives Literal, dann heißt sie *Zielklausel* oder *Frageklausel*, kurz *Ziel*.

Beispiel 8.1 (Hornklauseln und Hornformel)**.**

- $\{P\}$ ist eine Tatsachenklausel,
- $\{\neg P, Q\}$ und $\{\neg P, \neg Q, R\}$ sind Regeln und
- $\{\neg P, \neg R\}$ ist eine Zielklausel.

Die Hornformel in diesem Beispiel ist also:

$$P \wedge (\neg P \vee Q) \wedge (\neg P \vee \neg Q \vee R) \wedge (\neg P \vee \neg R)$$

[1] Alfred Horn, amerikanischer Mathematiker, 1918 - 2001

Wir können Hornformel auch (alternativ) definieren, indem wir eine Grammatik angeben:

Definition 8.4 (Hornformel)**.** Eine Hornformel ist eine Formel ϕ der Aussagenlogik, die folgender Grammatik genügt:

$T ::= \top \to P$ für Aussagensymbole P (Tatsache)
$P ::= P \mid P \wedge P$ für Aussagensymbole P
$Z ::= P \mid P \to \bot$ (Ziel)
$R ::= P \to Q$ für Aussagensymbole Q (Regel)
$H ::= \phi \wedge T \mid \phi \wedge Z \mid \phi \wedge R$

Begründung dieser Definition

1. Jede Hornklausel ist entweder eine Tatsache, eine Regel oder ein Ziel.
2. Eine Tatsache muss wahr sein, d.h. $P \equiv \top \to P$.
3. Eine Prozedurklausel (Regel) $\neg P_1 \vee \cdots \vee \neg P_n \vee Q$ ist äquivalent zu $P_1 \wedge \cdots \wedge P_n \to Q$.
4. Eine Zielklausel $\neg P_1 \vee \cdots \vee \neg P_n$ ist äquivalent zu $\neg(P_1 \wedge \cdots \wedge P_n)$, also $P_1 \wedge \cdots \wedge P_n \to \bot$.

Beispiel 8.2 (Hornformel)**.**

Die Hornformel aus Beispiel 8.1 ausgedrückt mit der Grammatik 8.4:

$$(\top \to P) \wedge (P \to Q) \wedge (P \wedge Q \to R) \wedge (P \wedge R \to \bot)$$

In der Hornlogik wird typischerweise ein Widerlegungsverfahren verwendet. In der Hornformel wird die angestrebte Aussage als *nicht* wahr angenommen (deshalb ist die Form der Zielklausel so wie oben definiert). Kann man nun die Nichterfüllbarkeit der Hornformel zeigen, sieht man, dass die Zielklausel zutrifft.

Wir formulieren einen Algorithmus für die Entscheidung der Erfüllbarkeit von Hornformeln (den sogenannten Markierungsalgorithmus):

function $\mathsf{HORN}(\phi)$ {
// pre: ϕ ist eine Hornformel
// post: entscheidet die Frage, ob ϕ erfüllbar ist

```
    markiere alle ⊤ in φ
    while ( es gibt P_1 ∧ P_2 ∧ ··· ∧ P_n → Q mit
            P_1, P_2, ..., P_n markiert, aber Q nicht) {
        markiere Q
    }
    if ( ⊥ ist markiert ){
        return „unerfüllbar“
    } else {
        return „erfüllbar“
    }
}
```

Eigenschaften dieses Algorithmus

1. Er entscheidet, ob die Hornformel ϕ erfüllbar ist.
2. Ist ϕ erfüllbar, dann können wir eine erfüllende Interpretation ablesen, nämlich

$$\alpha(P_i) = \begin{cases} \texttt{T} \text{ falls } P_i \text{ markiert} & \text{ist} \\ \texttt{F} \text{ sonst} \end{cases}$$

3. Der Algorithmus endet nach spätestens $n+2$ Markierungsschritten, wenn n die Zahl der Atome in ϕ ist.

Beispiel 8.3 (Markierungsalgorithmus)**.**

Gegeben sei die Aussage P, d.h. $\top \to P$, eine *Tatsache*.
Ferner sei bekannt, dass folgende *Regeln* gelten: $P \to Q$ und $P \wedge Q \to R$.
Man möchte nun in dieser Situation wissen, ob $P \wedge R$ gilt.

In der Hornlogik klärt man die Frage durch ein *Widerlegungsverfahren*: Man nimmt an, dass $P \wedge R$ *nicht* gilt. Durch diese Annahme entsteht folgende Hornformel:

$$(\top \to P) \wedge (P \to Q) \wedge (P \wedge Q \to R) \wedge (P \wedge R \to \bot)$$

Wenn gezeigt werden kann, dass diese Formel *unerfüllbar* ist, dann hat man gezeigt, dass in der gegebenen Situation $P \wedge R$ *gilt*. Dies zeigt man nun mit Hilfe des Markierungsalgorithmus:

$(\top \to P) \wedge$		$(P \to Q) \wedge$		$(P \wedge Q \to R) \wedge$			$(P \wedge R \to \bot)$		
*									
*	*	*		*			*		
*	*	*	*	*	*		*		
*	*	*	*	*	*	*	*	*	
*	*	*	*	*	*	*	*	*	*

Der Algorithmus endet in diesem Beispiel, wenn $\bot$ markiert ist. Das bedeutet, dass die Formel unerfüllbar ist, d.h. dass in der gegebenen Situation $P \wedge R$ gilt.

Kapitel 9

Erfüllbarkeit und SAT-Solver

Die Entscheidungsfragen der Aussagenlogik lassen sich alle auf das Erfüllbarkeitsproblem zurückspielen.

Ein Programm, das für eine Formel der Aussagenlogik das Erfüllbarkeitsproblem löst (und eine erfüllende Belegung ermittelt), heißt *SAT-Solver*.

Die Kunst in der Entwicklung von SAT-Solvern besteht darin, Algorithmen zu finden, die für große Klassen von Formeln das Problem effizient entscheiden, obgleich es $\mathcal{NP}$-vollständig ist.[1]

Typischerweise setzen SAT-Solver voraus, dass eine Formel in CNF vorliegt. Das Eingabeformat ist üblicherweise DIMACS (vorgeschlagen vom Center for Discrete Mathematics and Theoretical Computer Science `http://dimacs.rutgers.edu/`).

9.1 DIMACS-Format

SAT-Solver verwenden eine einfache Variante des DIMACS-Formats. Eine Formel in CNF wird in einer ASCII-Datei gespeichert, die in drei Sektionen aufgebaut ist:

Zuerst kommen optionale *Kommentarzeilen*, die durch ein kleines `c` an der ersten Position gekennzeichnet sind.

[1] Donald E. Knuth: „The story of satisfiability is the tale of a triumph of software engineering, blended with rich doses of beautiful mathematics. Thanks to elegant new data structures and other techniques, modern SAT solvers are able to deal routinely with practical problems that involve many thousands of variables, although such problems were regarded as hopeless just a few years ago.“[Knu15, S. iv]

Darauf folgt die *Präambel*, die aus einer Zeile der Form

`p cnf v c`

besteht, wobei `v` für die Zahl der Atome der Formel[2] und `c` für die Zahl der Klauseln steht.

Danach folgen die *Klauseln*. Die Literale werden durch Integers codiert. Dabei steht die positive Zahl für ein Atom, die negative Zahl für seine Negation. Jede Klausel nimmt eine Zeile der Datei ein, sie enthält die Literale getrennt durch Leerzeichen und wird abgeschlossen durch eine `0`.

Beispiel ϕ sei die folgende Formel in CNF:

$$\begin{aligned}
(P_1 \vee P_2 \vee P_3)\wedge \\
(P_1 \vee \neg P_2 \vee \neg P_3)\wedge \\
(P_1 \vee \neg P_5)\wedge \\
(\neg P_2 \vee \neg P_3 \vee \neg P_5)\wedge \\
(\neg P_1 \vee \neg P_2 \vee P_3)\wedge \\
(P_4 \vee P_6)\wedge \\
(P_4 \vee \neg P_6)\wedge \\
(P_2 \vee \neg P_4)\wedge \\
(\neg P_3 \vee \neg P_4)
\end{aligned}$$

Sie wird im DIMACS-Format so geschrieben:

```
c Beispiel DIMACS Vorlesung LfM
p cnf 6 9
1 2 3 0
1 -2 -3 0
1 -5 0
-2 -3 -5 0
-1 -2 3 0
4 6 0
4 -6 0
2 -4 0
-3 -4 0
```

Doch stop! Wir haben gesehen, dass die Umformung einer beliebigen Formel in eine äquivalente Formel in CNF im schlechtesten Fall eine exponentielle Laufzeit (in Bezug auf die Zahl der Atome der Formel) haben kann. Führt das nicht schon von vorneherein dazu, dass die Entscheidung der Erfüllbarkeit *nicht* effizient sein kann, noch ehe der SAT-

[2] Sieht man die aussagenlogische Formel als eine Boolesche Funktion spricht man auch von *Variablen*, deshalb der Buchstabe `v`.

Solver überhaupt startet, weil CNF als Eingabeform erwartet wird? Dies ist nicht der Fall:

9.2 Tseitin-Transformation

Definition 9.1 (Erfüllbarkeitsäquivalenz)**.** Zwei Formeln der Aussagenlogik ϕ und ψ heißen *erfüllbarkeitsäquivalent*, wenn gilt:

$$\phi \text{ erfüllbar} \Leftrightarrow \psi \text{ erfüllbar.}$$

Die *Tseitin*[3]*-Transformation* besteht nun darin, eine beliebige Formel ϕ in eine *erfüllbarkeitsäquivalente* Formel in CNF umzuformen. Die Tseitin-Transformation ist linear in der Zahl der Atome der Formel.

function $\mathsf{TSEITIN}(\phi)$ {
// post: Eine erfüllbarkeitsäquivalente Formel ϕ' in CNF

1. Führe für jede Subformel ψ von ϕ, die kein Atom ist, ein neues Atom T_ψ hinzu.
2. Setze $\phi' = T_\phi$.
3. Durchlaufe den Syntaxbaum von ϕ und füge ϕ' je nach Form der Subformel an einem inneren Knoten die Formel in CNF nach Tabelle 9.1 verbunden mit $\wedge$ hinzu. (Atome werden analog zu den anderen Subformeln behandelt.)

}

Tabelle 9.1: Regeln für die Tseitin-Transformation

$\phi = \neg\phi_1$	$(\neg T_\phi \vee \neg T_{\phi_1}) \wedge (T_\phi \vee T_{\phi_1})$
$\phi = \phi_1 \wedge \phi_2$	$(\neg T_\phi \vee T_{\phi_1}) \wedge (\neg T_\phi \vee T_{\phi_2}) \wedge (T_\phi \vee \neg T_{\phi_1} \vee \neg T_{\phi_2})$
$\phi = \phi_1 \vee \phi_2$	$(T_\phi \vee \neg T_{\phi_1}) \wedge (T_\phi \vee \neg T_{\phi_2}) \wedge (\neg T_\phi \vee T_{\phi_1} \vee T_{\phi_2})$
$\phi = \phi_1 \rightarrow \phi_2$	$(T_\phi \vee T_{\phi_1}) \wedge (T_\phi \vee \neg T_{\phi_2}) \wedge (\neg T_\phi \vee \neg T_{\phi_1} \vee T_{\phi_2})$

Die Tseitin-Transformation ϕ' einer Formel ϕ hat folgende Eigenschaften:

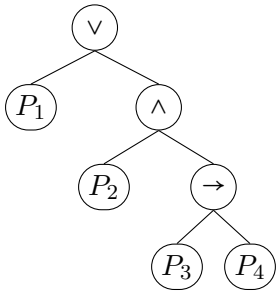
1. Die Menge der Atome von ϕ ist eine Teilmenge der Atome von ϕ'.
2. Wenn α eine erfüllende Interpretation von ϕ ist, dann gibt es eine Erweiterung von α auf die Atome von ϕ', so dass diese Interpretation ϕ' erfüllt.

[3] nach Grigori S. Zeitin, russischer Mathematiker, geb. 1936 [Tse83]

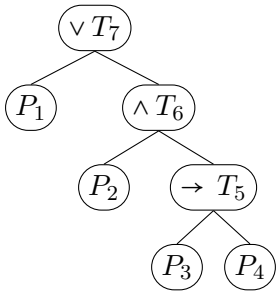
3. Ist α' eine erfüllende Interpretation von ϕ', dann erfüllt sie auch ϕ.

Beispiel

Betrachten wir die Formel $\phi = P_1 \vee (P_2 \wedge (P_3 \rightarrow P_4))$.



Im Schritt 1 legen wir pro Knoten, der nicht Blatt ist, ein neues Atom fest.



Im Schritt 2 erstellen wir ϕ':

$$
\begin{aligned}
T_7 \wedge \\
(T_7 \vee \neg P_1) \wedge (T_7 \vee \neg T_6) \wedge (\neg T_7 \vee P_1 \vee T_6) \wedge \\
(\neg T_6 \vee P_2) \wedge (\neg T_6 \vee T_5) \wedge (T_6 \vee \neg P_2 \vee \neg T_5) \wedge \\
(T_5 \vee P_3) \wedge (T_5 \vee \neg P_4) \wedge (\neg T_5 \vee \neg P_3 \vee P_4)
\end{aligned}
$$

9.3 DPLL und CDLC

Definition 9.2 (Wert-Propagation)**.** Sei ϕ eine Formel in CNF und $\hat{P}$ ein Literal. Dann bezeichnet $\phi[\top/\hat{P}]$ die Formel, die dadurch entsteht, dass $\hat{P}$ durch den Wert $\top$ ersetzt wird.

Durch die Wert-Propagation wird die gegebene Formel folgendermaßen vereinfacht:

- Jede Klausel der Formel, die $\hat{P}$ enthält, fällt weg. Denn sie wird durch die Wertzuweisung von $\top$ an $\hat{P}$ wahr.
- In jeder Klausel der Formel, in der $\neg\hat{P}$ vorkommt, entfällt dieser Ausdruck, da er $\bot$ ist und deshalb die Klausel nicht wahr machen kann.

Der Basis-Algorithmus von SAT-Solvern basiert wesentlich auf der Wert-Propagation.

Definition 9.3 (Einzelklausel)**.** Eine Klausel einer Formel in CNF heißt *Einzelklausel*, wenn sie aus genau einem Literal besteht.

Definition 9.4 (Reines Literal)**.** Ein Literal in einer Formel in CNF heißt *reines Literal*, wenn es in allen Klauseln nur als P oder $\neg P$ vorkommt.

Viele SAT-Solver verwenden (stark weiterentwickelte) Varianten des folgenden Algorithmus von Martin Davis[4], Hilary Putnam[5], George Logemann[6] und Donald W. Loveland[7] ([DP60], [DLL62]).

boolean function $\mathsf{DPLL}(\phi, \mathcal{M})$ {
// pre: ϕ eine Formel in CNF, $\mathcal{M}$ eine partielle Interpretation
// return: `true` falls ϕ erfüllbar ist
// post: Eine Interpretation $\mathcal{M} = \{\hat{P}_1, \hat{P}_2, \dots\}$ der Aussagensymbole
// von ϕ, falls ϕ erfüllbar ist
// modifies: $\mathcal{M}$

case {
$\phi = \top$:
return `true`;
$\phi = \bot$:
return `false`;
ϕ hat eine Einzelklausel $(\hat{P})$:
return $\mathsf{DPLL}(\phi[\top/\hat{P}], \mathcal{M} \cup \hat{P})$;
ϕ hat ein reines Literal $\hat{P}$:
return $\mathsf{DPLL}(\phi[\top/\hat{P}], \mathcal{M} \cup \hat{P})$;
otherwise:

[4] Martin Davis, amerikanischer Logiker und Informatiker, geb. 1928
[5] Hilary Putnam, amerikanischer Philosoph, 1926 - 2016
[6] George Logemann, amerikanischer Mathematiker, 1938 - 2012
[7] Donald W. Loveland, amerikanischer Informatiker, geb. 1934

wähle zu einem Atom Q in der Formel mit $\hat{Q} \notin \mathcal{M}$
return $\mathsf{DPLL}(\phi[\top/Q], \mathcal{M} \cup Q) \vee \mathsf{DPLL}(\phi[\bot/Q], \mathcal{M} \cup \neg Q)$;
}
}

Idee des Algorithmus

1. Propagation von Einzelklauseln
   Besteht eine Klausel nur aus einem Literal, also P oder $\neg P$, dann muss P = `true` bzw. $\neg P$ = `true` sein, damit die Formel in CNF erfüllbar wird.
   Folge: Man kann alle Klauseln weglassen, in denen $\hat{P}$ vorkommt, und außerdem $\neg\hat{P}$ in allen Klauseln, in denen es vorkommt.
2. Elimination von reinen Literalen
   Wenn in der Formel nur $\hat{P}$, niemals aber $\neg\hat{P}$ vorkommt, dann kann man $\hat{P}$ = `true` setzen. Denn dadurch fallen alle Klauseln weg, die $\hat{P}$ enthalten, d.h. die Wahl kann niemals zu einem Widerspruch führen, weil ja weder P noch $\neg P$ noch vorkommt.
   Folge: Man kann alle Klauseln weglassen, in denen $\hat{P}$ vorkommt, denn sie sind dann erfüllt.
3. Rekursion (*Backtracking*)
   Wenn keine der genannten Möglichkeiten bestehen, muss man ein Aussagensymbol Q der Formel ϕ wählen. Es gilt dann natürlich:
   ϕ erfüllbar $\Leftrightarrow$ $\phi \wedge Q$ erfüllbar oder $\phi \wedge \neg Q$ erfüllbar.

Beispiele

Wir betrachten zunächst das einfache Beispiel der Formel, an der wir den Markierungsalgorithmus für Horn-Formeln demonstriert haben:

$$\begin{aligned} P\wedge \\ (\neg P \vee Q)\wedge \\ (\neg P \vee \neg Q \vee R)\wedge \\ (\neg P \vee \neg R) \end{aligned}$$

Im ersten Schritt setzt man P `true`, da die erste Klausel eine Einzelklausel ist. Die Propagation dieses Werts führt dazu, dass diese Klausel selbst wegfällt und in allen anderen Klauseln $\neg P$ gestrichen werden kann. Man erhält also:

$$
\begin{aligned}
Q\wedge \\
(\neg Q \vee R)\wedge \\
\neg R
\end{aligned}
$$

Nun ist Q auf `true` zu setzen und dieser Wert zu propagieren, also folgt:

$$
\begin{aligned}
R\wedge \\
\neg R
\end{aligned}
$$

Und diese beiden Einzelnklauseln bilden den Widerspruch, d.h. die Formel ist *unerfüllbar*.

Nach diesem einfachen Beispiel, das gezeigt hat, dass der Markierungsalgorithmus im Grunde mit der Idee der Propagation von Einzelklauseln arbeitet, folgt das Beispiel von 9.1:

ϕ sei wieder die folgende Formel in CNF:

$$
\begin{aligned}
(P_1 \vee P_2 \vee P_3)\wedge \\
(P_1 \vee \neg P_2 \vee \neg P_3)\wedge \\
(P_1 \vee \neg P_5)\wedge \\
(\neg P_2 \vee \neg P_3 \vee \neg P_5)\wedge \\
(\neg P_1 \vee \neg P_2 \vee P_3)\wedge \\
(P_4 \vee P_6)\wedge \\
(P_4 \vee \neg P_6)\wedge \\
(P_2 \vee \neg P_4)\wedge \\
(\neg P_3 \vee \neg P_4)
\end{aligned}
$$

Schritt 1 Es gibt keine Einzelklausel, aber $\neg P_5$ ist ein reines Literal, d.h. wir brauchen für P_5 nur den Fall $\neg P_5 = \texttt{true}$ zu betrachten.
D.h. $\mathcal{M} = \{\neg P_5\}$ und $\phi[\top/\neg P_5]$ enthält die Klauseln mit $\neg p_5$ nicht mehr, also

$$
\begin{aligned}
(P_1 \vee P_2 \vee P_3)\wedge \\
(P_1 \vee \neg P_2 \vee \neg P_3)\wedge \\
(\neg P_1 \vee \neg P_2 \vee P_3)\wedge \\
(P_4 \vee P_6)\wedge \\
(P_4 \vee \neg P_6)\wedge \\
(P_2 \vee \neg P_4)\wedge \\
(\neg P_3 \vee \neg P_4)
\end{aligned}
$$

Schritt 2 Probiere $P_1 = \texttt{true}$, also $\mathcal{M} = \{\neg P_5, P_1\}$. Die Klauseln mit P_1 sind `true` und in den Klauseln mit $\neg P_1$ kann man $\neg P_1$ weglassen. Dann bleibt

$$
\begin{aligned}
(\neg P_2 \vee P_3)\wedge \\
(P_4 \vee P_6)\wedge \\
(P_4 \vee \neg P_6)\wedge \\
(P_2 \vee \neg P_4)\wedge \\
(\neg P_3 \vee \neg P_4)
\end{aligned}
$$

Schritt 3 Probiere $P_2 = \texttt{true}$, also $\mathcal{M} = \{\neg P_5, P_1, P_2\}$. Die Klauseln mit P_2 sind `true` und in den Klauseln mit $\neg P_2$ kann man $\neg P_2$ weglassen. Dann bleibt

$$
\begin{aligned}
(P_3)\wedge \\
(P_4 \vee P_6)\wedge \\
(P_4 \vee \neg P_6)\wedge \\
(\neg P_3 \vee \neg P_4)
\end{aligned}
$$

(P_3) ist nun eine Einzelklausel, d.h. P_3 muss `true` sein, also $\mathcal{M} = \{\neg P_5, P_1, P_2, P_3\}$. Es bleibt:

$$
\begin{aligned}
(P_4 \vee P_6)\wedge \\
(P_4 \vee \neg P_6)\wedge \\
(\neg P_4)
\end{aligned}
$$

$(\neg P_4)$ ist nun auch Einzelklausel, und es bleibt:

$$
\begin{aligned}
(P_6)\wedge \\
(\neg P_6)
\end{aligned}
$$

Dies ist aber der Widerspruch.

Schritt 4 Probiere $P_2 = \texttt{false}$, also $\mathcal{M} = \{\neg P_5, P_1, \neg P_2\}$. Dann bleibt

$$\begin{aligned}(P_4 \vee P_6)\wedge\\(P_4 \vee \neg P_6)\wedge\\(\neg P_4)\wedge\\(\neg P_3 \vee \neg P_4)\end{aligned}$$

$(\neg P_4)$ ist jetzt Einzelklausel, d.h. P_4 muss `false` sein, d.h.

$$\begin{aligned}(P_6)\wedge\\(\neg P_6)\wedge\\(\neg P_3)\end{aligned}$$

Erneut der Widerspruch.

Schritt 5 Probiere $P_1 = \texttt{false}$, also $\mathcal{M} = \{\neg P_5, \neg P_1\}$. Dann bleibt:

$$\begin{aligned}(P_2 \vee P_3)\wedge\\(\neg P_2 \vee \neg P_3)\wedge\\(P_4 \vee P_6)\wedge\\(P_4 \vee \neg P_6)\wedge\\(P_2 \vee \neg P_4)\wedge\\(\neg P_3 \vee \neg P_4)\end{aligned}$$

Schritt 5 Probiere $P_2 = \texttt{true}$, also $\mathcal{M} = \{\neg P_5, \neg P_1, P_2\}$. Dann bleibt:

$$\begin{aligned}(\neg P_3)\wedge\\(P_4 \vee P_6)\wedge\\(P_4 \vee \neg P_6)\wedge\\(\neg P_3 \vee \neg P_4)\end{aligned}$$

$(\neg P_3)$ ist jetzt Einzelklausel , d.h. P_3 muss `false` sein, also $\mathcal{M} = \{\neg P_5, \neg P_1, P_2, \neg P_3\}$. Dann bleibt:

$$\begin{aligned}(P_4 \vee P_6)\wedge\\(P_4 \vee \neg P_6)\end{aligned}$$

P_4 ist jetzt reines Literal, d.h P_4 muss `true` sein, dann bleibt nichts mehr übrig, alle Klauseln sind erfüllt. D.h. die Wahl der Belegung für P_6 ist beliebig.

Ergebnis $\mathcal{M} = \{\neg P_5, \neg P_1, P_2, \neg P_3, P_4, P_6\}$ ist eine erfüllende Interpretation.

Von DPLL zu CDLC

Der vorgestellte DPLL-Algorithmus gibt die Idee wieder, heutige SAT-Solver verwenden verbesserte Algorithmen mit folgenden Eigenschaften:

1. Oft wird keine Analyse bezüglich reiner Literale gemacht, weil diese Analyse aufwändig ist.
2. Stattdessen wird im Fall eines *Konflikts* genau analysiert, wodurch er entstanden ist und eine neue Klausel hinzugefügt, die dazu führt, dass der Algorithmus einmal gefundene Abhängigkeiten zwischen Atomen nicht „vergisst“ und deshalb erneut auswerten muss.

   Die Idee kann man an unserem Beispiel oben sehen. Wir haben in Schritt 1 P_1 als `true` und in Schritt 2 P_2 als `true` gewählt. Diese beiden Entscheidungen haben zum Widerspruch geführt. Also muss gelten: $\neg p_1 \vee \neg p_2$. Diese Klausel können wir nun unserer ursprünglichen Formel hinzufügen. Der Algorithmus hat aus dem Konflikt „gelernt“.

   Klausel-lernende SAT-Solver werden auch CDLC-Solver (*Conflict Driven Clause Learning*) genannt.
3. Diese Analyse kann noch weitergehen: Man merkt sich in welchem Level der Analyse man eine Entscheidung getroffen hat und macht bei einem Konflikt nicht einfach *Backtracking*, sondern *Backjumping*.
4. Außerdem werden Heuristiken eingesetzt, welche Variable beim Backjumping „ausprobiert“ wird. Ein Beispiel wäre etwa: DLCS (*Dynamic Largest Combined Sum*) – man wählt die Variable, die positiv oder negativ am häufigsten in der Formel vorkommt.

   Würde man diese Strategie in unserem Beispiel verwenden, müsste man in Schritt 2 mit P_2 = `true` starten, danach P_3 mit `true` probieren, was zum Konflikt führt. Doch P_3 = `false` führt zu einer erfüllenden Belegung. Wir wären also etwas schneller.

Wie aktuelle SAT-Solver arbeiten wird in [KS06] und [Knu15] beschrieben.

Kapitel 10

Anwendungen der Aussagenlogik in der Softwaretechnik

siehe Vorlesung

10.1 Anwendungen von SAT-Techniken

- 2003 – Validierung von Produktkonfigurationen in der Automobilindustrie, Carsten Sinz et al, siehe [SKK03].
- 2009 – Formale Verifizierung des Intel Core 7 Prozessors, Kaivola et al, siehe [KGN+09]
- 2010 – Verifikation von Windows 7 Gerätetreibern mit einem SMT-Solver, De Moura, Bjørner, siehe [dMB10]. SMT (SAT modulo Theory) baut auf dem Konzept von SAT-Solvern auf. Microsoft Research hat einen SMT-Solver namens Z3 entwickelt, siehe `https://github.com/Z3Prover/z3/wiki`.
- 2016/18 – Management von Abhängigkeiten innerhalb des Ökosystems der Eclipse Plattform, Le Berre, Rapicault, siehe [BR18].
- ...

10.2 Codeanalyse

10.3 Featuremodelle für (Software-)Produktlinien

Teil II

Prädikatenlogik

Kapitel 11

Objekte und Prädikate

Die Ausdruckskraft der Aussagenlogik ist beschränkt. Wir werden folgenden Schluss sicherlich für richtig halten:

> Alle Quadratzahlen $\neq 0$ sind positiv (P)
> 16 ist eine Quadratzahl (Q)
> Also folgt: 16 ist positiv (R)

Übertragen wir diesen Schluss etwas naiv in die Aussagenlogik, so ergibt sich $P \wedge Q \rightarrow R$ — und es gibt keinerlei Grund anzunehmen, dass diese Aussage zutrifft.

Dies liegt daran, dass wir in der Aussagenlogik den Zusammenhang zwischen der ersten und der zweiten Aussage nicht erfassen können, nämlich, dass die 16 ein Exemplar von der Sorte *Quadratzahl* ist.

Also: Wir müssen unterscheiden können zwischen

Objekten, den Dingen eines Universums, einer „Welt“, wie z.B. Zahlen, Strings, Werten, Objekten usw. und

Prädikaten, Aussagen über die Objekte, die wahr oder falsch sein können.

Beispiel Ist etwa unser Universum die Welt der ganzen Zahlen $\mathbb{Z}$, dann könnten wir folgende Prädikate haben

> $Sq(x)$ bedeutet „x ist eine Quadratzahl $\neq 0$“
> $Pos(x)$ bedeutet „x ist positiv“

Dann können wir das einleitende Beispiel so ausdrücken

$$\forall x(Sq(x) \to Pos(x)) \wedge Sq(16) \to Pos(16)$$

11.1 Elemente der Sprache der Prädikatenlogik

Gegeben sei stets eine Menge $\mathbb{U}$, das Universum, auch genannt „Miniwelt“. In der Regel wird in der Literatur vorausgesetzt, dass das Universum nicht leer ist, d.h. $\mathbb{U} \neq \varnothing$[1]

Wir verwenden zusätzlich zu den Junktoren der Aussagenlogik

Variablen
$x, y, z \ldots$
Variablen sind Platzhalter für beliebige Objekte des Universums, z.B. steht in $Sq(x)$ die Variable x für ein Element des Universums, also in unserem Beispiel für eine Zahl in $\mathbb{Z}$.

Konstanten
$c, d, e \ldots$
Konstanten sind bestimmte, benannte Elemente des Universums, z.B. 16, die Zahl $16 \in \mathbb{Z}$.

Funktionen
$f, g, h \ldots$
Funktionen operieren auf den Elementen des Universums und ergeben wieder Elemente des Universums, z.B. $plus(16, 9)$ ergibt das Element $25 \in \mathbb{Z}$ mit der naheliegenden Definition von $plus$.

Prädikate
$P, Q, R \ldots$
Prädikate sind boolesche Funktionen, die Aussagen über Elemente der Welt machen. Die Wertemenge eines Prädikats ist also in $\mathbb{B} = \{\mathtt{T}, \mathtt{F}\}$. In unserem Beispiel ist Sq ein Prädikat der Arität 1, $Sq(x)$ ist genau dann wahr, wenn $x \in \mathbb{Z}$ eine Quadratzahl ist.

Gleichheit
$=$
Gleichheit ist ein spezielles Prädikat, das wir in die Sprache von vorneherein aufnehmen.[2]

[1] Ist das Universum leer, dann ist jede Aussage der Form $\exists x \ldots$ immer falsch und eine der Form $\forall x \ldots$ immer wahr – diese Fälle möchte man gerne vermeiden.

[2] Man kann auch Prädikatenlogik ohne Gleichheit machen, für Anwendungen ist es jedoch nützlich, die Gleichheit als *logisches* Symbol aufzufassen.

Quantoren
$\forall x\phi, \exists y\phi$
Quantoren machen Aussagen über *alle* Elemente des Universums oder darüber, ob ein Element des Universums mit einer bestimmten Eigenschaft *existiert*.

11.2 Prädikate und Relationen

Es besteht eine wichtige grundlegende Beziehung zwischen *Prädikaten und Relationen*:

Sei R eine n-wertige Relation über $\mathbb{U}$, d.h. $R \subset \mathbb{U}^n$. Dann kann man diese Relation repräsentieren durch die boolesche Funktion $r : \mathbb{U}^n \to \mathbb{B}$ definiert durch

$$r(a_1, \ldots, a_n) = \mathtt{T} \text{ genau dann, wenn } (a_1, \ldots, a_n) \in R$$

Ist etwa $Sq \subset \mathbb{Z} = \{x / \exists y \text{ mit } x = y^2 \text{ und } x > 0\}$, also

$$Sq = \{1, 4, 9, 16, 25, \ldots\}$$

dann entspricht diese einstellige Relation gerade dem oben verwendeten Prädikat Sq.

Offenbar sind also Prädikate und Relationen austauschbare Konzepte, deshalb spricht man bei der Prädikatenlogik erster Ordnung auch manchmal von *relationaler Logik*.

Bemerkung Man könnte in unserem Beispiel auch eine binäre Relation $Sq \subset \mathbb{N}^2$ auf folgende Weise definieren:

Sei $square : \mathbb{N} \to \mathbb{N}$ die Funktion, die einem $x \in \mathbb{N}$ sein Quadrat als Funktionswert zuordnet, also $x \mapsto x^2$. Diese Funktion kann man als binäre Relation auffassen, nämlich $Sq = \{(x, x^2) / x \in \mathbb{N}\}$. Ein Tupel (x, y) ist also genau dann in Sq, wenn y das Quadrat von x ist. Auf diese Weise entspricht jeder n-stelligen Funktion eine $n+1$-stellige Relation.

Kapitel 12

Die formale Sprache der Prädikatenlogik

12.1 Signatur, Terme, Formeln

In der Sprache der Prädikatenlogik verwendet man Funktions- und Konstantensymbole sowie Prädikatssymbole. Genau genommen bezieht man sich auf eine bestimmte Wahl dieser Symbole und spricht dann von *einer* Sprache $\mathcal{L}$ der Prädikatenlogik.

Definition 12.1 (Signatur)**.** Die *Signatur* einer Sprache $\mathcal{L}$ besteht aus einer Menge von Prädikatssymbolen $\{P_1, \ldots, P_n\}$, von Funktionssymbolen $\{f_1, \ldots, f_m\}$ und von Konstantensymbolen $\{c_1, \ldots, c_k\}$.

Man schreibt die Signatur so:

$$\{P_1^{r_1}, \ldots, P_n^{r_n}; f_1^{a_1}, \ldots, f_m^{a_m}; c_1, \ldots, c_k\}$$

wobei die r und die a die Arität der Prädikatssymbole bzw. Funktionssymbole bezeichnet.

Bemerkungen

- Die Signatur besteht aus den „nicht-logischen“ Symbolen der Sprache $\mathcal{L}$.
- Manche Autoren definieren die Signatur dadurch, dass sie nur die Arität von Prädikatssymbolen, Funktionssymbolen und die Zahl der Konstanten angeben. Denn „Name ist Schall und Rauch“[1]. In dieser Sicht wird die Signatur so angegeben:

$$< r_1, \ldots, r_n; a_1, \ldots, a_m; k >$$

[1] J.W. von Goethe, Faust I, Marthens Garten

wobei r_i die Arität eines Prädikatensymbols, a_i die Arität eines Funktionssymbols und k die Zahl der Konstanten ist.

- Man kann Konstantensymbole auch als Funktionssymbole der Arität 0 auffassen.
- Prädikatssymbole der Arität 0 kann man als Aussagensymbole auffassen. Oder anders: Wenn in der Signatur nur Prädikatssymbole der Arität 0 vorkommen und keine anderen Symbole und wir uns auf die Junktoren $\neg, \wedge, \vee, \rightarrow$ beschränken, dann erhalten wir gerade die Definition eine Sprache der Aussagenlogik, wie in Teil I.

Im Folgenden sei eine Signatur gegeben, sowie eine Menge von Variablen $\{x, y, \dots\}$.

Definition 12.2 (Terme)**.** Die *Terme* sind Zeichenketten, die nach folgenden Regeln gebildet werden:

(i) Jede Variable ist ein Term.

(ii) Jedes Konstantensymbol ist ein Term.

(iii) Sind $t_1, \dots, t_n$ Terme und f ein Funktionssymbol mit der Arität n, dann ist auch $f(t_1, \dots, t_n)$ ein Term.

Als *Grammatik* in Backus-Naur-Darstellung können wir diese Regeln so ausdrücken:

$$t ::= x \mid c \mid f \mid f(t, \dots, t)$$

mit Variablen x, Konstantensymbolen c und Funktionssymbolen f.

Definition 12.3 (Formeln)**.** Die *Formeln* sind die Zeichenketten, die durch folgende Regeln generiert werden können:

(i) $\perp$ ist eine Formel (genannt: der Widerspruch) und $\top$ ist eine Formel (genannt: die Wahrheit).

(ii) Ist P ein Prädikatssymbol der Arität 0, dann ist P eine Formel.
Ist P ein Prädikatssymbol der Arität $n \geq 1$ und sind $t_1, \dots, t_n$ Terme, dann ist $P(t_1, \dots, t_n)$ eine Formel.
Sind t_1 und t_2 Terme, dann ist $(t_1 = t_2)$ eine Formel.

(iii) Ist ϕ eine Formel, dann auch $(\neg\phi)$.
Sind ϕ und ψ Formeln, dann auch $(\phi \wedge \psi)$, $(\phi \vee \psi)$ und $(\phi \rightarrow \psi)$.

(iv) Ist ϕ eine Formel und x eine Variable, dann sind $\forall x\ \phi$ und $\exists x\ \phi$ Formeln.

In Backus-Naur-Darstellung:

$$\begin{aligned}\phi ::= \quad & \bot \,|\, \top \,|\, P \,|\, P(t,\ldots,t) \,|\, (t = t) \,| \\ & (\neg\phi), \,|\, (\phi \wedge \phi) \,|\, (\phi \vee \phi) \,|\, (\phi \to \phi) \,| \\ & \forall x\ \phi \,|\, \exists x\ \phi\end{aligned}$$

mit Termen t, Variablen x und Prädikatssymbolen P.

Bemerkungen

- Die Argumente von Prädikatssymbolen dürfen *nur* Terme sein, keine anderen Prädikate – wir definieren hier nämlich die Prädikatenlogik *erster* Stufe.
- In unserer Definition der Sprachen der Prädikatenlogik, haben wir die Gleichheit = als *logisches* Symbol mit aufgenommen. In vielen Büchern wird unterschieden zwischen der Prädikatenlogik und der Prädikatenlogik mit Gleichheit.

  Man könnte denken, dass man die Gleichheit einfach dadurch einführen kann, dass man ein Prädikatsymbol für die Gleichheit definiert und dabei verlangt, dass in jedem Modell der Sprache dieses gerade die Identitäsrelation über dem Universum des Modells ist. (Vorgriff – siehe Abschnitt 13.1) [Hof11, S.114f]

Bindungsregeln
$\neg, \forall x, \exists x$ binden am stärksten, dann
$\wedge$ und $\vee$ (linksassoziativ) und schließlich
$\to$ (rechtsassoziativ.)

Diese Bindungsregeln erlauben uns, sparsamer (und dadurch besser lesbar) mit Klammern umzugehen, ohne die Grammatik mehrdeutig zu machen.

12.2 Freie und gebundene Variablen

Sei ϕ eine Formel der Prädikatenlogik. Ein Vorkommen der Variablen x in ϕ heißt *frei*, wenn x im Syntaxbaum nicht Abkömmling eines Quantorknotens $\forall x$ oder $\exists x$ ist. Andernfalls heißt das Vorkommen *gebunden*.

Die Namen von gebundenen Variablen spielen keine Rolle, deshalb werden Formeln, die sich nur durch den Namen gebundener Variablen unterscheiden, identifiziert. $\forall x P(x, y)$ ist also dieselbe Formel wie $\forall z P(z, y)$, nicht jedoch $\forall x P(x, z)$, da die Bedeutung der freien Variablen vom Kontext abhängt.

Definition 12.4 (Satz)**.** Eine Formel ϕ, die keine freien Variablen hat, heißt *Satz* oder *geschlossene* Formel.

Definition 12.5 (Grundterm)**.** Ein Term t, der keine Variablen hat, heißt *geschlossener* Term oder *Grundterm.*

Bemerkung
Hat eine Formel freie Variablen, dann macht man sie zu einem Satz, in dem man sich alle freien Variablen durch den Allquantor gebunden denkt.

12.3 Substitution

Variablen sind Platzhalter für Terme. Substitution besteht darin, solche Platzhalter durch Terme zu ersetzen.

Ein Ersetzen von Variablen durch Terme ist nur möglich für Variablen, die *nicht* durch einen Quantor gebunden sind. Gebundene Variablen stehen für ein bestimmtes Element des Universums oder für alle Elemente des Universums, können somit nicht durch den Term ersetzt werden.

Definition 12.6 (Substitution)**.** Gegeben eine Variable x und ein Term t. Die Formel $\phi[t/x]$ ist die Formel, die aus ϕ entsteht, in dem jedes *freie* Vorkommen von x durch t ersetzt wird.

Dabei darf der eingesetzte Term t keine Variablen enthalten, die durch die Substitution in den Geltungsbereich eines Quantors kommen würden.

Die einschränkende Bemerkung zur Substitution sei an einem Beispiel erläutert:

Haben wir die Formel $\forall x P(x, y)$ und wollen y durch den Term $f(x)$ ersetzen, dann ergäbe sich durch einfaches Einsetzen $\forall x P(x, f(x))$ und plötzlich ist die Variable x im Term $f(x)$ in den Bereich des Allquantors gekommen. Dies ist nicht erlaubt.

Richtig wäre es, zunächst die gebundene Variable umzubenennen, also etwa $\forall z P(z, y)$, wodurch sich die Formeln nicht ändert. Jetzt ist die Substitution möglich:

$(\forall x P(x, y))[f(x)/y]$ ergibt $\forall z P(z, f(x))$, nicht aber $\forall x P(x, f(x))$.

Man sagt, dass eine Term t frei ist für eine Variable x in einer Formel ϕ, wenn t keine Variable enthält, die beim Einsetzen für x in den Geltungsbereich eines Quantors käme.

Bei einer Substitution muss man also erst prüfen, ob der zu substituierende Term frei für die Variable in der Formel ist. Ist dies nicht der Fall, muss man gebundene Variablen in der Formel so umtaufen, dass keine „Kollision“ auftritt.

Bemerkung In der Logic Workbench (lwb) hat man als Junktoren die Junktoren aus der Aussagenlogik, siehe Tabelle 3.1, darüber hinaus:

- das ausgezeichnete Symbol = für die Gleichheit,
- den Allquantor, z.B. `(forall [x y] (and (P x) (Q y)))` mit unären Prädikaten P und Q sowie
- den Existenzquantor, z.B. `(exists [x] (= (f x) (g x)))` mit den unären Funktion f und g.

Kapitel 13

Semantik der Prädikatenlogik

13.1 Struktur/Modell

Definition 13.1 (Struktur/Modell)**.** Eine *Struktur* (auch Modell) für die Sprache $\mathcal{L}$ ist ein Paar $\mathcal{M} =< \mathbb{U}, I >$ mit einer nicht-leeren Menge $\mathbb{U}$ und einer Funktion I, die jedem Symbol in $\mathcal{L}$ eine Interpretation zuordnet nach folgenden Regeln:

(i) Ist P ein 0-äres Prädikatensymbol, dann ist $I(P)$ ein Wahrheitswert.

(ii) Ist P ein n-äres Prädikatensymbol für $n > 0$, dann ist $I(P) \subseteq \mathbb{U}^n$ eine n-äre Relation über $\mathbb{U}$.

(iii) Ist c ein Konstantensymbol, dann ist $I(c) \in \mathbb{U}$, ein Element von $\mathbb{U}$.

(iv) Ist f ein Funktionssymbol mit Arität n, dann ist $I(f) : \mathbb{U}^n \to \mathbb{U}$ eine Funktion.

Die Menge $\mathbb{U}$ nennt man auch das Universum. Die Interpretation schreibt man auch oft so:

- $P^{\mathcal{M}}$ für die Prädikate über $\mathbb{U}$
- $c^{\mathcal{M}}$ für die Elemente zu den Konstantensymbolen und
- $f^{\mathcal{M}}$ für die Funktionen zu den Funktionssymbolen

Definition 13.2 (Variablenbelegung)**.** Sei $\mathcal{M} =< \mathbb{U}, I >$ eine Struktur für $\mathcal{L}$. Eine *Variablenbelegung* in $\mathcal{M}$ ist eine Funktion l, die jeder Variablen x einen Wert $l(x) \in \mathbb{U}$ zuordnet.

Man schreibt $l[x \mapsto a]$ für die Variablenbelegung, die x auf a abbildet und alle anderen Variablen auf $l(y)$, d.h.

$$l[x \mapsto a](y) = \begin{cases} a & \text{falls } y = x. \\ l(y) & \text{falls } y \neq x. \end{cases}$$

Definition 13.3 (Interpretation der Terme)**.** Sei $\mathcal{M} =< \mathbb{U}, I >$ eine Struktur für $\mathcal{L}$ und l eine Variablenbelegung. Für einen Term t von $\mathcal{L}$ definiert man die Interpretation $I(t)$ bezüglich $\mathcal{M}$ und der Variablenbelegung l induktiv über die Länge des Terms durch

(i) $I(x) := l(x)$ für die Variablen x,

(ii) $I(c) := I(c)$ für die Konstanten c, und

(iii) $I(f(t_1, \ldots, t_n) := I(f)(I(t_1), \ldots, I(t_n))$ für die Funktionen f.

Definition 13.4 (Interpretation der Formeln)**.** Sei $\mathcal{M} =< \mathbb{U}, I >$ eine Struktur für $\mathcal{L}$ und l eine Variablenbelegung, dann ist $\mathcal{M}$ ein Modell für eine Formel ϕ, geschrieben

$$\mathcal{M} \models_l \phi \quad \text{für die Formel } \phi,$$

falls $[\![\phi]\!]_l^{\mathcal{M}} = \mathtt{T}$. Dabei wird $[\![\phi]\!]_l^{\mathcal{M}}$ induktiv definiert über den strukturellen Aufbau von ϕ:

$$
\begin{array}{lll}
(i) & [\![P(t_1,\ldots,t_n)]\!]_l^{\mathcal{M}} & := \begin{cases} \mathtt{T} \text{ falls } (I(t_1),\ldots I(t_n)) \in P^{\mathcal{M}} \\ \mathtt{F} \text{ sonst} \\ \text{bzw. } I(P) \text{ falls } P0\text{-är} \end{cases} \\
(ii) & [\![s = t]\!]_l^{\mathcal{M}} & := \begin{cases} \mathtt{T} \text{ falls } I(s) = I(t) \\ \mathtt{F} \text{ sonst} \end{cases} \\
(iii) & [\![\neg\phi]\!]_l^{\mathcal{M}} & := \begin{cases} \mathtt{T} \text{ falls } [\![\phi]\!]_l^{\mathcal{M}} = \mathtt{F} \\ \mathtt{F} \text{ sonst} \end{cases} \\
(iv) & [\![\phi \wedge \psi]\!]_l^{\mathcal{M}} & := \begin{cases} \mathtt{T} \text{ falls } [\![\phi]\!]_l^{\mathcal{M}} = \mathtt{T} \text{ und } [\![\psi]\!]_l^{\mathcal{M}} = \mathtt{T} \\ \mathtt{F} \text{ sonst} \end{cases} \\
(v) & [\![\phi \vee \psi]\!]_l^{\mathcal{M}} & := \begin{cases} \mathtt{T} \text{ falls } [\![\phi]\!]_l^{\mathcal{M}} = \mathtt{T} \text{ oder } [\![\psi]\!]_l^{\mathcal{M}} = \mathtt{T} \\ \mathtt{F} \text{ sonst} \end{cases} \\
(vi) & [\![\phi \rightarrow \psi]\!]_l^{\mathcal{M}} & := \begin{cases} \mathtt{T} \text{ falls } [\![\phi]\!]_l^{\mathcal{M}} = \mathtt{F} \text{ oder } [\![\psi]\!]_l^{\mathcal{M}} = \mathtt{T} \\ \mathtt{F} \text{ sonst} \end{cases} \\
(vii) & [\![\forall x\ \phi]\!]_l^{\mathcal{M}} & := \begin{cases} \mathtt{T} \text{ falls für alle } a \in \mathbb{U} \text{ gilt: } [\![\phi]\!]_{l[x\mapsto a]}^{\mathcal{M}} = \mathtt{T} \\ \mathtt{F} \text{ sonst} \end{cases} \\
(viii) & [\![\exists x\ \phi]\!]_l^{\mathcal{M}} & := \begin{cases} \mathtt{T} \text{ falls es existiert ein } a \in \mathbb{U} \text{ mit: } [\![\phi]\!]_{l[x\mapsto a]}^{\mathcal{M}} = \mathtt{T} \\ \mathtt{F} \text{ sonst} \end{cases} \\
(ix) & [\![\top]\!]_l^{\mathcal{M}} & := \mathtt{T} \\
(x) & [\![\bot]\!]_l^{\mathcal{M}} & := \mathtt{F}
\end{array}
$$

13.2 Semantische Folgerung und Äquivalenz

Definition 13.5 (Semantische Folgerung)**.** Sei Γ eine Menge prädikatenlogischer Formeln und ϕ eine prädikatenlogische Formel. Man sagt:

$\Gamma \vDash \phi$ d.h. ϕ *folgt semantisch* aus Γ, genau dann wenn jedes Modell für Γ auch ein Modell für ϕ ist.

Besteht Γ nur aus einer Formel, sage ψ, dann schreibt man auch $\psi \vDash \phi$.

Definition 13.6 (Semantische Äquivalenz)**.** Zwei Formeln ϕ und ψ sind *semantisch äquivalent*, geschrieben $\phi \equiv \psi$, genau dann, wenn $\phi \vDash \psi$ und $\psi \vDash \phi$ gilt.

In der Prädikatenlogik können wir nun dieselben Definitionen wie in der Aussagenlogik machen:

Definition 13.7 (Erfüllbarkeit)**.** Eine prädikatenlogische Formel ϕ heißt *erfüllbar*, wenn sie ein Modell hat.

Definition 13.8 (Falsifizierbarkeit)**.** Eine prädikatenlogische Formel ϕ heißt *falsifizierbar*, wenn es ein Modell $\mathcal{M}$ gibt mit $\mathcal{M} \not\models \phi$.

Definition 13.9 (Allgemeingültigkeit)**.** Eine prädikatenlogische Formel ϕ heißt *allgemeingültig*, wenn sie in jedem Modell wahr ist.
Man schreibt dann $\models \phi$ und nennt ϕ eine *Tautologie*.

Definition 13.10 (Unerfüllbarkeit)**.** Eine prädikatenlogische Formel ϕ heißt *unerfüllbar*, wenn es kein Modell für sie gibt.
Man schreibt dann $\not\models \phi$ und nennt ϕ eine *Kontradiktion*.

Auch in der Prädikatenlogik gilt das Dualitätsprinzip:

Satz 13.1 (Dualitätsprinzip)**.** *Eine prädikatenlogische Formel ϕ ist genau dann allgemeingültig, wenn $\neg\phi$ unerfüllbar ist.*

13.3 Fundamentale Äquivalenzen der Prädikatenlogik

Satz 13.2. *ϕ, ψ und χ seien Formeln der Aussagenlogik, wobei in χ die Variable x nicht vorkommt.*

Es gelten folgende (semantische) Äquivalenzen:

$$\neg(\forall x\ \phi) \equiv \exists x\ (\neg\phi) \tag{13.1}$$
$$\neg(\exists x\ \phi) \equiv \forall x\ (\neg\phi) \tag{13.2}$$
$$\forall x\ \phi \wedge \forall x\ \psi \equiv \forall x\ (\phi \wedge \psi) \tag{13.3}$$
$$\exists x\ \phi \vee \exists x\ \psi \equiv \exists x\ (\phi \vee \psi) \tag{13.4}$$
$$\forall x\ (\forall y\ \phi) \equiv \forall y\ (\forall x\ \phi) \tag{13.5}$$
$$\exists x\ (\exists y\ \phi) \equiv \exists y\ (\exists x\ \phi) \tag{13.6}$$
$$(\forall x\ \phi) \wedge \chi \equiv \forall x\ (\phi \wedge \chi) \tag{13.7}$$
$$(\forall x\ \phi) \vee \chi \equiv \forall x\ (\phi \vee \chi) \tag{13.8}$$
$$(\exists x\ \phi) \wedge \chi \equiv \exists x\ (\phi \wedge \chi) \tag{13.9}$$
$$(\exists x\ \phi) \vee \chi \equiv \exists x\ (\phi \vee \chi) \tag{13.10}$$

Kapitel 14

Natürliches Schließen in der Prädikatenlogik

In Kapitel 5 wurde das Beweissystems des natürlichen Schließens nach Gerhard Gentzen für die Aussagenlogik eingeführt. Es beruht auf Regeln, wie Formeln (syntaktisch) umgeformt werden dürfen, um aus gegebenen Aussagen Schlussfolgerungen herzuleiten.

Ein wichtiges Ergebnis war dabei die Vollständigkeit des natürlichen Schließens als Beweissystem für die Aussagenlogik, d.h. dass für eine Menge Γ von gegebenen Formeln und eine Formel ϕ der Aussagenlogik gilt:

$$\Gamma \vdash \phi \Leftrightarrow \Gamma \vDash \phi$$

Die „syntaktische Sicht“ und die „semantische Sicht“ sind also äquivalent und wir können zwischen ihnen wechseln, je nachdem welche für eine konkrete Fragestellung besser geeignet ist.

Das Beweissystem des natürlichen Schließens hat Gerhard Gentzen für die Prädikatenlogik (mit Gleichheit) eingeführt. In diesem Kapitel werden die Regeln für die Quantoren und die Gleichheit vorgestellt.

Der Satz über die Vollständigkeit des natürlichen Schließens gilt auch für die Prädikatenlogik. Dies wurde zuerst von Kurt Gödel[1] für das Hilbert[2]-Kalkül gezeigt. Gerhard Gentzen hat ihn für sein Beweissystem des Sequenzenkalküls bewiesen. Man muss den Vollständigkeitssatz so begreifen, dass er nicht für ein spezielles Beweissystem gilt, sondern

[1] Kurt Gödel (1906–1978), österreichisch-amerikanischer Logiker.
[2] David Hilbert (1862–1943), deutscher Mathematiker.

eine Eigenschaft der Logik selbst ist, vorausgesetzt natürlich, dass das Beweissystem „vernünftig“ definiert ist.[3]

14.1 Schlussregeln

Allquantor

	Einführung	*Elimination*
$\forall$	$\dfrac{\boxed{\begin{array}{l} x_0 \\ \qquad\vdots \\ \qquad \phi[x_0/x] \end{array}}}{\forall x\phi}$ $\forall$ x i	$\dfrac{\forall x\phi}{\phi[t/x]}$ $\forall$ x e

Um den Allquantor einzuführen, hat man folgende Beweisverpflichtung: Gegeben sei ein beliebiges Objekt x_0 des Universums. Man muss dann zeigen, dass die Formel ϕ mit x_0 an Stelle der Variablen x gilt (dies schreibt man kurz als $\phi[x_0/x]$). Dabei darf in dieser Herleitung keinerlei *spezielle* Eigenschaft von x_0 vorkommen, denn x_0 steht ja für ein *beliebiges* Objekt des Universums. Man sagt auch, dass x_0 ein *frisches* beliebiges Objekt ist, sein Name darf somit nicht außerhalb der Box vorkommen.

Die Entfernung des Allquantors ist ein naheliegender Schritt: Wenn ϕ für alle x gilt, dann kann man ein beliebiges konkretes t des Universums an Stelle von x in die Formel ϕ einsetzen.

Existenzquantor

	Einführung	*Elimination*
$\exists$	$\dfrac{\phi[t/x]}{\exists x\phi}$ $\exists$ x i	$\dfrac{\exists x\phi \quad \boxed{\begin{array}{ll} x_0 & \phi[x_0/x] \\ & \vdots \\ & \chi \end{array}}}{\chi}$ $\exists$ x e

Den Existenzquantor kann man einführen, indem man einen *Zeugen* vorweist: Gilt ϕ mit t an Stelle von x, dann gibt es offenbar ein x für das ϕ gilt, nämlich eben t.

[3] „... completeness should be understood not as a statement about these specific rules, but as a statement about the logic itself. Completeness asserts the existence of a list of rules that allows us to deduce every consequence from any set of formulas of the logic.“ [Hed04, S. 44]

Will man den Existenzquantor entfernen, muss man ein Objekt x_0 nehmen, das ϕ an Stelle von x erfüllt. Solch ein Objekt existiert, weil ja $\exists x\phi$ gilt. Nun hat man die Beweisverpflichtung zu zeigen, dass daraus χ herleitbar ist. In dieser Herleitung darf man keine spezielle Aussage über x_0 verwenden, außer $\phi[x_0/x]$.

Gleichheit

	Einführung	*Elimination*
$=$	$\dfrac{}{t = t}$ $= \mathsf{i}, \mathsf{ID}$	$\dfrac{t_1 = t_2 \quad \phi[t_1/x]}{\phi[t_2/x]}$ $= \mathsf{e}, \mathsf{SUB}$

Die Regel ID besagt, dass ein Symbol, das für ein Objekt steht, dieses eindeutig bestimmt. Dies ist gewissermaßen die Charakteristik der Gleichheit.

Die Entfernung der Gleichheit besteht darin, dass wenn t_1 und t_2 gleich sind, man in einer Formel ϕ t_1 durch t_2 ersetzen kann. Dies klingt wie selbstverständlich, muss aber mit Vorsicht gehandhabt werden. Es sind nur gültige Substitutionen erlaubt: In allen Substitutionen $\phi[t/x]$ muss t frei für x in der Formel ϕ sein, d.h. keine freie Variable y in t gelangt durch das Einsetzen von x in ϕ in den Bereich eines Quantors $\forall y$ oder $\exists y$.

14.2 Beispiele

Gentzen zeigt in [Gen35, S. 183] an drei Beispielen, wie das natürliche Schließen geht. Das erste Beispiel für eine Formel der Aussagenlogik wurde in Abschnitt 5.2 verwendet.

Für die Prädikatenlogik verwendet Gentzen die beiden folgenden Beispiele:

Beispiel: Vertauschen von Quantoren

Herleitung für

$$\exists x \forall y F(x, y) \to \forall y \exists x F(x, y)$$

1.	$\exists x \forall y F(x,y)$	angenommen
2.	a	angenommen
3.	$\forall y F(a,y)$	angenommen
4.	b	beliebig
5.	$F(a,b)$	$\forall$e 3, 4
6.	$\exists x F(x,b)$	$\exists$i 2, 5
7.	$\forall y \exists x F(x,y)$	$\forall$i 4-6
8.	$\forall y \exists x F(x,y)$	$\exists$e 1, 2-7
9.	$\exists x \forall y F(x,y) \to \forall y \exists x F(x,y)$	$\to$ i 1-8

Negation und Quantoren

Als weiteres Beispiel folgt ein Beweis für

$$\neg \exists x G(x) \to \forall y \neg G(y)$$

Der Beweis folgt wieder der Argumentation von Gentzen in [Gen35, S. 183]:

1.	$\neg \exists x G(x)$	angenommen
2.	a	beliebig
3.	$G(a)$	angenommen
4.	$\exists x G(x)$	$\exists$i 2, 3
5.	$\bot$	$\neg$e 1, 4
6.	$\neg G(a)$	$\neg$i 3-5
7.	$\forall y \neg G(y)$	$\forall$i 2-6
8.	$\neg \exists x G(x) \to \forall y \neg G(y)$	$\to$ i 1-7

14.3 Vollständigkeit des natürlichen Schließens

siehe Vorlesung

Kapitel 15

Unentscheidbarkeit der Prädikatenlogik

siehe Vorlesung

Kapitel 16

Anwendungen der Prädikatenlogik in der Softwaretechnik

siehe Vorlesung

16.1 Analyse von Softwaremodellen mit Alloy

Teil III

Lineare Temporale Logik

Kapitel 17

Dynamische Modelle

siehe Vorlesung

Kapitel 18

Die formale Sprache der linearen temporalen Logik (LTL)

In der Sprache der linearen temporalen Logik (LTL) erweitert man die formale Sprache der Aussagenlogik durch weitere Junktoren, mit denen temporale Eigenschaften formuliert werden können.

Definition 18.1 (Alphabet der LTL)**.** Das *Alphabet* der Sprache der linearen temporalen Logik (LTL) besteht aus

(i) einer Menge $\mathcal{P}$ von Aussagensymbolen,

(ii) den (aussagenlogischen) Junktoren: $\neg, \wedge, \vee, \rightarrow$

(iii) den (temporalen) Junktoren: $\bigcirc$, $\mathcal{U}$

(iv) der Konstanten: $\perp$

(v) den zusätzlichen Symbolen: $($, $)$

Bemerkungen

- Wie im Falle der Aussagenlogik, definieren wir *eine* Sprache der linearen temporalen Logik durch die Vorgabe der Menge $\mathcal{P}$ und der eben definierten Junktoren.
- Die formale Sprache der LTL ist eine Erweiterung der Aussagenlogik, in der zwei neue Junktoren vorkommen:
  $\bigcirc$ steht für „zum nächsten Zeitpunkt“ (*next*)
  $\mathcal{U}$ steht für „bis“ (*until*)

Definition 18.2 (Formeln der LTL)**.** Die *Formeln* der linearen temporalen Logik sind Zeichenketten, die nach folgenden Regeln gebildet werden:

(i) Jedes Symbol $P \in \mathcal{P}$ ist eine Formel und auch $\perp$ ist eine Formel.

(ii) Ist ϕ eine Formel, dann auch $\neg\phi$.

(iii) Sind ϕ und ψ Formeln, dann auch $(\phi \wedge \psi)$, $(\phi \vee \psi)$ und $(\phi \rightarrow \psi)$

(iv) Ist ϕ eine Formel, dann auch $\bigcirc\,\phi$.

(v) Sind ϕ und ψ Formeln, dann auch $(\phi\,\mathcal{U}\,\psi)$

Als *Grammatik* in Backus-Naur-Darstellung können wir diese induktive Definition der Formeln der LTL so ausdrücken:

$$\phi ::= P \,|\, \perp \,|\, \neg\phi \,|\, (\phi \wedge \phi) \,|\, (\phi \vee \phi) \,|\, (\phi \rightarrow \phi) \,|\, \bigcirc\,\phi \,|\, (\phi\,\mathcal{U}\,\phi)$$

mit Variablen $P \in \mathcal{P}$ und (bereits gebildeten) Formeln ϕ.

Die Präzedenz der Junktoren wird folgendermaßen definiert: Die unären Junktoren $\neg$ und $\bigcirc$ binden stärker als die binären, sie selbst binden gleich stark. Binäre Junktoren binden in folgender Reihenfolge, die stärkste Bindung zuerst: $\mathcal{U}, \wedge, \vee, \rightarrow$. Außerdem sind $\wedge$ und $\vee$ linksassoziativ, $\mathcal{U}$ und $\rightarrow$ sind rechtsassoziativ.

Bemerkung
In Formeln der LTL werden wir oft vier weitere Junktoren verwenden, die folgendermaßen definiert werden:

$$\begin{aligned}
\Diamond\,\phi &\stackrel{\text{def}}{=} \neg\perp\,\mathcal{U}\,\phi \\
\Box\,\phi &\stackrel{\text{def}}{=} \neg\Diamond\,\neg\phi \\
\phi\,\mathcal{W}\,\psi &\stackrel{\text{def}}{=} (\phi\,\mathcal{U}\,\psi) \vee \Box\,\phi \\
\phi\,\mathcal{R}\,\psi &\stackrel{\text{def}}{=} \neg(\neg\phi\,\mathcal{U}\,\neg\psi)
\end{aligned}$$

Wir lesen sie so:

$\Diamond$ steht für „irgendwann“ (*eventually*),
$\Box$ steht für „immer“ (*always*),
$\mathcal{W}$ steht für „sofern nicht“ (*unless, weak until*) und
$\mathcal{R}$ steht für „löst ab“ (*release*).

$\Diamond$ und $\Box$ haben dieselbe Bindungspräzedenz wie $\bigcirc$ und $\mathcal{W}$ und $\mathcal{R}$ die von $\mathcal{U}$.

Kapitel 19

Die Semantik der linearen temporalen Logik (LTL)

19.1 Kripke-Struktur

Modelle in der temporalen Logik enthalten ein implizites Konzept einer diskreten Zeit: Man denkt sich die „Welt“ des Modells als bestehend aus Zuständen, in denen gewissen Aussagen wahr sind und einer Übergangsrelation der Zustände. Jeder Zustandsübergang entspricht dann gerade einem Zeitschritt. Präziser definiert man die Kripke-Struktur[1]:

Definition 19.1 (Kripke-Struktur)**.** Eine *Kripke-Struktur* $\mathcal{K}$ ist ein Tupel $(S, s_0, \rightarrow, L)$ bestehend aus

- einer Menge von Zuständen S,
- einem ausgezeichneten Startzustand $s_0 \in S$,
- einer Übergangsrelation $\rightarrow \subseteq S \times S$, die jedem Zustand s einen Folgezustand s' zuordnet (d.h. $\forall s\, \exists s'$ mit $s \rightarrow s'$) und
- einer Beschriftungsfunktion $L : S \rightarrow \mathbb{P}(\mathcal{P})$ von S in die Potenzmenge von $\mathcal{P}$, die jedem Zustand eine Menge von (wahren) Aussagenatomen zuordnet.

Bemerkungen

1. Man könnte in die Definition der Kripke-Struktur auch die Wahl von $\mathcal{P}$ explizit aufnehmen. In den Beispielen, die wir betrachten, ergibt sich die Menge der Atome aus der Beschriftungsfunktion.
2. Die Beschriftungsfunktion L ordnet jedem Zustand die in diesem Zustand wahren Aussagen aus $\mathcal{P}$ zu. Dies kann man auch so sehen: L ordnet jedem Zustand s eine Interpretation $\alpha_s : \mathcal{P} \rightarrow \mathbb{B}$ zu.

[1] Saul A. Kripke (* 1940), amerikanischer Logiker.

3. Eine Kripke-Struktur kann man als gerichteten Graphen sehen, in dem die Zustände S die Knoten sind und die Übergangsrelation gerade die gerichteten Kanten. Zudem wird jeder Zustand mit den in ihm gültigen Aussagen gemäß der Beschriftungsfunktion L markiert. Der Startzustand wird durch einen eingehenden Pfeil ohne Startknoten markiert.
4. Manchmal zeichnet man in einer Kripke-Struktur keinen Startzustand aus, man bezeichnet sie dann als Übergangssystem (siehe [HR04, Abschnitt 3.2]).
5. Manche Autoren lassen in Kripke-Strukturen auch mehrere Startzustände zu.
6. Wenn in einem konkreten System die Übergangsrelation nicht die Eigenschaft hat, dass es zu jedem Zustand einen Folgezustand gibt, kann man den Graph um einen Zustand erweitern, der einen Übergang auf sich selbst hat und hat damit die Definition einer Kripke-Struktur erfüllt.

Beispiele In Abb.19.1 und 19.2 werden die Graphen zu zwei Beispielen dargestellt.

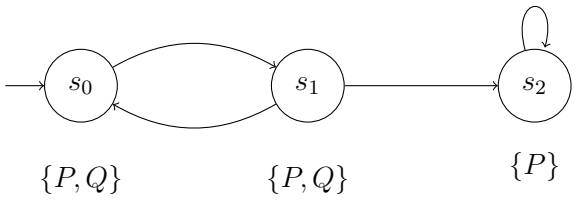


Abbildung 19.1: Beispiel einer Kripke-Struktur

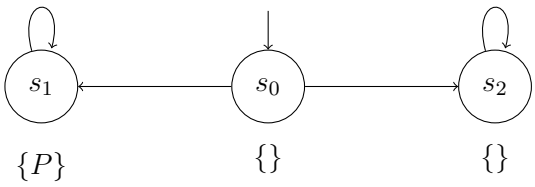


Abbildung 19.2: Beispiel für die Semantik der Negation

Definition 19.2 (Pfad und Berechnung)**.** Sei $\mathcal{K} = (S, s_0, \rightarrow, L)$ eine Kripke-Struktur.

Ein *Pfad* π ist eine unendliche Folge $s_1, s_2, s_3, \ldots$ von Zuständen $s_i \in S$ mit $s_i \rightarrow s_{i+1}$ für alle $i \geq 1$. Man schreibt einen Pfad gerne so:

$$\pi = s_1 \rightarrow s_2 \rightarrow s_3 \rightarrow \ldots$$

Hat man einen Pfad $\pi = s_1 \to s_2 \to s_3 \ldots$ gegeben, dann bezeichnet man mit π^i den Pfad, der im i-ten Zustand von π beginnt, also z.B. $\pi^2 = s_2 \to s_3 \to s_4 \ldots$.

Eine *Berechnung* ist ein Pfad, der mit mit dem Startzustand $s_0 \in S$ beginnt.

Nun haben wir alle Notation, um Semantiken der LTL definieren zu können:

Definition 19.3 (Semantik der LTL für einen Pfad)**.** Sei $\mathcal{K}$ eine Kripke-Struktur und $\pi = s_1 \to s_2 \to s_3 \to \ldots$ ein Pfad. Für eine Formel ϕ der linearen temporalen Logik definiert man

$$\pi \vDash \phi,$$

falls $[\![\phi]\!]^{\mathcal{K}}_{\pi} = \mathtt{T}$. Dabei wird $[\![\phi]\!]^{\mathcal{K}}_{\pi}$ induktiv definiert über den strukturellen Aufbau von ϕ

$$(i) \quad [\![\bot]\!]^{\mathcal{K}}_{\pi} := \mathtt{F}$$

$$(ii) \quad [\![P]\!]^{\mathcal{K}}_{\pi} := \begin{cases} \mathtt{T} \text{ falls } P \in L(s_1) \\ \mathtt{F} \text{ sonst} \end{cases}$$

$$(iii) \quad [\![\neg\phi]\!]^{\mathcal{K}}_{\pi} := \begin{cases} \mathtt{T} \text{ falls } [\![\phi]\!]^{\mathcal{K}}_{\pi} = \mathtt{F} \\ \mathtt{F} \text{ sonst} \end{cases}$$

$$(iv) \quad [\![\phi \wedge \psi]\!]^{\mathcal{K}}_{\pi} := \begin{cases} \mathtt{T} \text{ falls } [\![\phi]\!]^{\mathcal{K}}_{\pi} = \mathtt{T} \text{ und } [\![\psi]\!]^{\mathcal{K}}_{\pi} = \mathtt{T} \\ \mathtt{F} \text{ sonst} \end{cases}$$

$$(v) \quad [\![\phi \vee \psi]\!]^{\mathcal{K}}_{\pi} := \begin{cases} \mathtt{T} \text{ falls } [\![\phi]\!]^{\mathcal{K}}_{\pi} = \mathtt{T} \text{ oder } [\![\psi]\!]_{\pi} = \mathtt{T} \\ \mathtt{F} \text{ sonst} \end{cases}$$

$$(vi) \quad [\![\phi \to \psi]\!]^{\mathcal{K}}_{\pi} := \begin{cases} \mathtt{T} \text{ falls } [\![\phi]\!]^{\mathcal{K}}_{\pi} = \mathtt{F} \text{ oder } [\![\psi]\!]^{\mathcal{K}}_{\pi} = \mathtt{T} \\ \mathtt{F} \text{ sonst} \end{cases}$$

$$(vii) \quad [\![\bigcirc\, \phi]\!]^{\mathcal{K}}_{\pi} := \begin{cases} \mathtt{T} \text{ falls } [\![\phi]\!]^{\mathcal{K}}_{\pi^2} = \mathtt{T} \\ \mathtt{F} \text{ sonst} \end{cases}$$

$$(viii) \quad [\![\phi\, \mathcal{U}\, \psi]\!]^{\mathcal{K}}_{\pi} := \begin{cases} \mathtt{T} \text{ falls } \exists_{\,i \geq 1} \text{ mit } [\![\psi]\!]^{\mathcal{K}}_{\pi^i} = \mathtt{T} \\ \quad \text{und } \forall_{\,j=1,\ldots i-1}\ [\![\phi]\!]^{\mathcal{K}}_{\pi^j} = \mathtt{T} \\ \mathtt{F} \text{ sonst} \end{cases}$$

Veranschaulichung der Semantik der temporalen Operatoren der LTL

- $\bigcirc\,\phi$ bedeutet, dass ϕ im nächsten Zustand gilt:

$$\begin{array}{cccccc} s_1 & s_2 & s_3 & s_4 & s_5 & s_6 \\ \bullet \longrightarrow & \bullet \longrightarrow & \bullet \longrightarrow & \bullet \longrightarrow & \bullet \longrightarrow & \bullet \longrightarrow \\ & \phi & & & & \end{array}$$

- $\Diamond\,\phi$ bedeutet, dass ϕ irgendwann auf dem Pfad gilt:

$$\begin{array}{cccccc} s_1 & s_2 & s_3 & s_4 & s_5 & s_6 \\ \bullet \longrightarrow & \bullet \longrightarrow & \bullet \longrightarrow & \bullet \longrightarrow & \bullet \longrightarrow & \bullet \longrightarrow \\ & & & \phi & & \end{array}$$

- $\Box\,\phi$ bedeutet, dass ϕ immer auf dem Pfad gilt:

$$\begin{array}{cccccc} s_1 & s_2 & s_3 & s_4 & s_5 & s_6 \\ \bullet \longrightarrow & \bullet \longrightarrow & \bullet \longrightarrow & \bullet \longrightarrow & \bullet \longrightarrow & \bullet \longrightarrow \\ \phi & \phi & \phi & \phi & \phi & \phi \end{array}$$

- $\phi\,\mathcal{U}\,\psi$ bedeutet, dass ψ irgendwann auf dem Pfad gilt, und dass bis dahin auf jeden Fall ϕ wahr ist:

$$\begin{array}{cccccc} s_1 & s_2 & s_3 & s_4 & s_5 & s_6 \\ \bullet \longrightarrow & \bullet \longrightarrow & \bullet \longrightarrow & \bullet \longrightarrow & \bullet \longrightarrow & \bullet \longrightarrow \\ \phi & \phi & \phi & \psi & & \end{array}$$

- $\phi\,\mathcal{W}\,\psi$ bedeutet, dass ϕ gilt bis ψ gilt oder für immer, wenn ein solcher Zustand nicht existiert:

$$\begin{array}{cccccc} s_1 & s_2 & s_3 & s_4 & s_5 & s_6 \\ \bullet \longrightarrow & \bullet \longrightarrow & \bullet \longrightarrow & \bullet \longrightarrow & \bullet \longrightarrow & \bullet \longrightarrow \\ \phi & \phi & \phi & \psi & & \end{array}$$

$$\begin{array}{cccccc} s_1 & s_2 & s_3 & s_4 & s_5 & s_6 \\ \bullet \longrightarrow & \bullet \longrightarrow & \bullet \longrightarrow & \bullet \longrightarrow & \bullet \longrightarrow & \bullet \longrightarrow \\ \phi & \phi & \phi & \phi & \phi & \phi \end{array}$$

- $\phi\,\mathcal{R}\,\psi$ bedeutet, dass ϕ gilt einschließlich dem ersten Zustand, in dem ψ gilt oder immer, wenn ein solcher Zustand nicht existiert:

$$\begin{array}{cccccc} s_1 & s_2 & s_3 & s_4 & s_5 & s_6 \\ \bullet \longrightarrow & \bullet \longrightarrow & \bullet \longrightarrow & \bullet \longrightarrow & \bullet \longrightarrow & \bullet \longrightarrow \\ \phi & \phi & \phi & \phi \wedge \psi & & \end{array}$$

$$\begin{array}{cccccc} s_1 & s_2 & s_3 & s_4 & s_5 & s_6 \\ \bullet \longrightarrow & \bullet \longrightarrow & \bullet \longrightarrow & \bullet \longrightarrow & \bullet \longrightarrow & \bullet \longrightarrow \\ \phi & \phi & \phi & \phi & \phi & \phi \end{array}$$

Definition 19.4 (Semantik für Pfade, Zustände und Strukturen)**.** Sei $\mathcal{K}$ eine Kripke-Struktur. Es gilt dann

- Sei π ein *Pfad* über $\mathcal{K}$. Dann sagt man, dass π eine Formel ϕ erfüllt, geschrieben $\pi \vDash \phi$, wenn gilt $[\![\phi]\!]_\pi^{\mathcal{K}} = \mathtt{T}$.
- Sei s ein *Zustand* von $\mathcal{K}$. Dann sagt man, dass s eine Formel ϕ erfüllt, geschrieben $s \vDash \phi$, wenn für alle Pfade π, die mit s beginnen, gilt $\pi \vDash \phi$.
- Man sagt, dass die *Kripke-Struktur* $\mathcal{K}$ eine Formel ϕ erfüllt, geschrieben $\mathcal{K} \vDash \phi$, wenn für den Startzustand s_0 gilt: $s_0 \vDash \phi$.

Beispiele Wenn wir zunächst das obige Beispiel 19.1 betrachten, ist leicht zu sehen, dass gilt:

- $\mathcal{K} \vDash \Box P$
  denn in allen Zuständen ist P true.
- $s_0 \vDash \bigcirc (P \vee Q)$
  denn in s_1 gilt $P \vee Q$.
- $s_0 \vDash \bigcirc (P \wedge Q)$
  denn in s_1 gilt $P \wedge Q$.
- $s_1 \nvDash \bigcirc (P \wedge Q)$
  denn zwar gilt in s_0 gilt $P \wedge Q$, nicht aber in s_2.
- $\mathcal{K} \vDash \Box (\neg Q \to \Box (P \wedge \neg Q))$
  denn der einzige Zustand mit $\neg Q$ ist s_2, ab dann gilt aber immer $P \wedge \neg Q$).

An Beispiel 19.2 kann man sehen, dass obgleich für Pfade gilt $\pi \vDash \phi \Leftrightarrow \pi \nvDash \neg\phi$ gilt, dies für Kripke-Strukturen nicht der Fall ist:

- $\mathcal{K} \nvDash \Diamond P$
  denn auf dem Pfad $s_0 \to s_2 \to s_2 \to \ldots$ ist P niemals true.
- $\mathcal{K} \nvDash \neg\Diamond P$
  denn auf dem Pfad $s_0 \to s_1 \to s_1 \to \ldots$ ist P schließlich true.

Bemerkung Die Semantik der linearen temporalen Logik unterscheidet sich grundlegend je nach Definition. Betrachtet man die Semantik der LTL für Pfade, dann gilt der Satz vom ausgeschlossenen Dritten, d.h. $\vDash_\pi \phi \vee \neg\phi$ für eine beliebige Formel ϕ der LTL. Für die Semantik der LTL für Kripke-Strukturen ist dies nicht der Fall. Obiges Beispiel zeigt, dass es Kripke-Strukturen geben kann, in denen weder ϕ noch $\neg\phi$ gilt. Das liegt daran, dass es in einer Kripke-Struktur Pfade π_1 und π_2 geben kann, für die $\pi_1 \vDash \phi$ und $\pi_2 \vDash \neg\phi$ gilt.[2]

[2] Dass der Satz vom ausgeschlossenen Dritten (*Tertium non datur*) *nicht* richtig ist, gilt auch für die Kripke-Semantik der intuitionistischen Logik. Diese Semantik

Für die Definition von Erfüllbarkeit, Allgemeingültigkeit und semantischer Äquivalenz wird die Pfad-Semantik zugrundegelegt.

Definition 19.5 (Erfüllbarkeit, Allgemeingültigkeit)**.**

- Eine Formel ϕ heißt *erfüllbar*, wenn es eine Kripke-Struktur gibt mit einem Pfad π so dass gilt $\pi \models \phi$.
- Eine Formel ϕ heißt *allgemeingültig*, wenn für alle Pfade in allen Kripke-Strukturen gilt: $\pi \models \phi$.

Definition 19.6 (Semantische Äquivalenz)**.** Zwei Formeln ϕ und ψ sind *semantisch äquivalent*, geschrieben $\phi \equiv \psi$, wenn für alle Pfade π gilt: $\pi \models \phi \Leftrightarrow \pi \models \psi$.

19.2 Äquivalenzen von Formeln der LTL

- Dualität

$$\neg \bigcirc \phi \equiv \bigcirc \neg \phi$$
$$\neg \Diamond \phi \equiv \Box \neg \phi$$
$$\neg \Box \phi \equiv \Diamond \neg \phi$$

- Idempotenz

$$\Diamond \Diamond \phi \equiv \Diamond \phi$$
$$\Box \Box \phi \equiv \Box \phi$$
$$\phi \mathcal{U} (\phi \mathcal{U} \psi) \equiv \phi \mathcal{U} \psi$$
$$(\phi \mathcal{U} \psi) \mathcal{U} \psi \equiv \phi \mathcal{U} \psi$$

- Absorption

$$\Diamond \Box \Diamond \phi \equiv \Box \Diamond \phi$$
$$\Box \Diamond \Box \phi \equiv \Diamond \Box \phi$$

---

wird über Kripke-Strukturen definiert, die zusätzlich zu unserer Definition die Eigenschaft der *Monotonie* haben. Diese Eigenschaft bedeutet, dass in jedem Folgezustand s' eines beliebigen Zustands s gilt: $L(s) \subseteq L(s')$. D.h. also, dass mit jedem Zustandsübergang mehr atomare Aussagen wahr werden.
Die Semantik der Junktoren der intuitionistischen Aussagenlogik werden dann in den Begriffen der LTL so definiert:

- $P \wedge Q \stackrel{\text{def}}{=} \mathcal{K} \models P \wedge Q$
- $P \vee Q \stackrel{\text{def}}{=} \mathcal{K} \models P \vee Q$
- $\neg P \stackrel{\text{def}}{=} \mathcal{K} \models \neg \Diamond P$
- $P \rightarrow Q \stackrel{\text{def}}{=} \mathcal{K} \models \Box (P \rightarrow Q)$

Siehe [Bor05, Chap. 9] sowie [vD13, Section 6.3].

- Expansion

$$\phi \mathcal{U} \psi \equiv \psi \vee (\phi \wedge \bigcirc (\phi \mathcal{U} \psi))$$
$$\Diamond \phi \equiv \phi \vee \bigcirc \Diamond \phi$$
$$\Box \phi \equiv \phi \wedge \bigcirc \Box \phi$$

- Distributiv-Gesetze

$$\Diamond (\phi \vee \psi) \equiv \Diamond \phi \vee \Diamond \psi$$
$$\Box (\phi \wedge \psi) \equiv \Box \phi \wedge \Box \psi$$
$$\bigcirc (\phi \mathcal{U} \psi) \equiv \bigcirc \phi \mathcal{U} \bigcirc \psi$$

19.3 Typische Aussagen in der LTL

siehe Vorlesung

Kapitel 20

Natürliches Schließen in der LTL

siehe Vorlesung

Kapitel 21

Anwendungen der LTL in der Softwaretechnik

siehe Vorlesung

21.1 Model Checking

Konzept des Model Checkings

Der Model Checker SPIN

Beispiele für Model Checking

21.2 Zielemodell in der Anforderungsanalyse

Literaturverzeichnis

[Bor05] Richard Bornat. *Proof and Disproof in Formal Logic: An introduction for programmers.* Oxford, 2005.

[BR18] Daniel Le Berre and Pascal Rapicault. Boolean-based dependency management for the eclipse ecosystem. *International Journal on Artificial Intelligence Tools*, 27(1):1–23, 2018.

[DLL62] Martin Davis, George Logemann, and Donald Loveland. A machine program for theorem-proving. *Commun. ACM*, 5(7):394–397, 1962.

[dMB10] Leonardo Mendonça de Moura and Nikolaj Bjørner. Applications and challenges in satisfiability modulo theories. In *Second International Workshop on Invariant Generation, WING 2009, York, UK, March 29, 2009 and Third International Workshop on Invariant Generation, WING 2010, Edinburgh, UK, July 21, 2010*, volume 1 of *EPiC Series in Computing*, pages 1–11. EasyChair, 2010.

[DP60] Martin Davis and Hilary Putnam. A computing procedure for quantification theory. *J. ACM*, 7(3):201–215, 1960.

[Gen35] Gerhard Gentzen. Untersuchungen über das logische schließen. i. *Mathematische Zeitschrift*, 39:176–210, 1935.

[Hed04] Shawn Hedman. *A First Course in Logic: An Introduction to Model Theory, Proof Theory, Computability, and Complexity.* Oxford, 2004.

[Hof11] Dirk W. Hoffmann. *Grenzen der Mathematik: Eine Reise durch die Kerngebiete der mathematischen Logik.* Heidelberg, 2011.

[HR04] Michael Huth and Mark Ryan. *Logic in Computer Science: Modelling and Reasoning about Systems.* Cambridge, UK, 2nd edition edition, 2004.

[KGN+09] Roope Kaivola, Rajnish Ghughal, Naren Narasimhan, Amber Telfer, Jesse Whittemore, Sudhindra Pandav, Anna Slobodová, Christopher Taylor, Vladimir A. Frolov, Erik Reeber,

and Armaghan Naik. Replacing testing with formal verification in intel core i7 processor execution engine validation. In *Computer Aided Verification, 21st International Conference, CAV 2009, Grenoble, France, June 26 - July 2, 2009. Proceedings*, volume 5643 of *Lecture Notes in Computer Science*, pages 414–429, 2009.

[Knu15] Donald E. Knuth. *The Art of Computer Programming, Volume 4, Fascicle 6: Satisfiability.* Boston, MA, 2015.

[KS06] Daniel Kroening and Ofer Strichman. *Decision Procedures: An Algorithmic Point of View.* 2006.

[Rau08] Wolfgang Rautenberg. *Einführung in die Mathematische Logik, 3., überarbeitete Auflage.* 2008.

[SKK03] Carsten Sinz, Andreas Kaiser, and Wolfgang Küchlin. Formal methods for the validation of automotive product configuration data. *AI EDAM*, 17(1):75–97, 2003.

[Tse83] G. S. Tseitin. On the complexity of derivation in propositional calculus. In J. Siekmann and G. Wrightson, editors, *Automation of Reasoning 2: Classical Papers on Computational Logic 1967-1970*, pages 466–483. Berlin, Heidelberg, 1983.

[vD13] Dirk van Dalen. *Logic and Structure.* Berlin, 5th edition, 2013.