# Logik und formale Methoden

## Vorlesungsskript

von
Burkhardt Renz

Wintersemester 2019/20

Burkhardt Renz
Technische Hochschule Mittelhessen
Rev 0.22 – 7. Oktober 2019

# Inhaltsverzeichnis

# 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: $\bot$

(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 Aussagenssymbolen vor, etwa $\mathcal{P} = \{P_0, P_1, P_2, \dots\}$ und sprechen dann von *der* Sprache der Aussagenlogik.[1]

  Für Anwendungen der Aussagenlogik in der Informatik und der Softwareentwicklung haben wir jedoch mit endlichen Mengen zu

[1] Die Menge der Aussagensymbole muss übrigens nicht unbedingt abzählbar sein, sondern kann eine beliebige Menge sein, siehe [7, 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.

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 $\bot$ 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 $\bot$ ist eine Formel.

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

(iii) Sind $\phi$ und $\psi$ 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 $\phi$ 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 $\phi$ und $\psi$ 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 $\bot$ 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:

[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.)

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

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

Zu einer Formel $\phi$ 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 $\phi$ beschriftet ist.

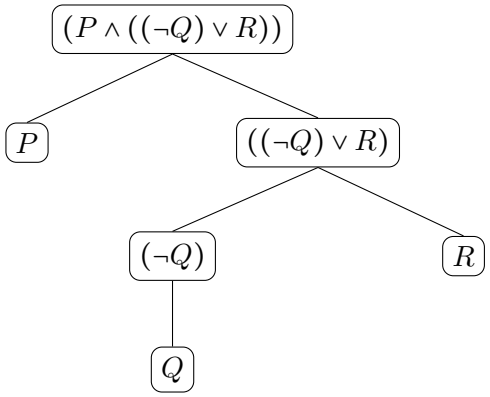
(iii) Ist ein Knoten mit einer Formel der Form $(\phi \wedge \psi)$, $(\phi \vee \psi)$ oder $(\phi \to \psi)$ beschriftet, dann hat er genau zwei Kinder und das linke ist mit $\phi$, das rechte mit $\psi$ beschriftet.

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

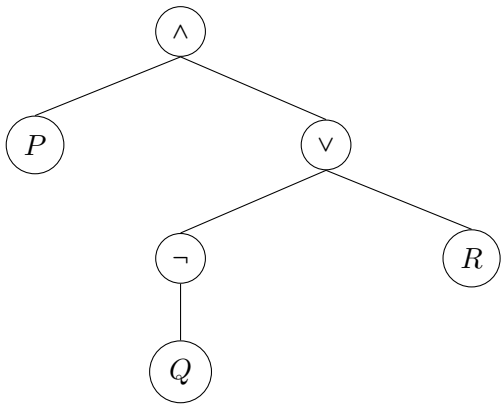
**Beispiel 3.1.** Gegeben sei die Formel

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

Der Syntaxbaum ist dann




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 Primnformeln 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 $\phi$, wenn gilt:

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

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

(iii) Gilt $\mathcal{E}$ für $\phi$ und $\psi$, 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 $\phi$ ist ein Formel, die als Beschriftung an einem Knoten des Syntaxbaums von $\phi$ vorkommt.

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

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

(ii) $\mathrm{rg}((\phi \square \psi)) = \max(\mathrm{rg}(\phi), \mathrm{rg}(\psi)) + 1$, wobei $\square$ für einen der Junktoren $\wedge, \vee, \rightarrow$ 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: $\neg, \wedge, \vee, \rightarrow$. Außerdem sind $\wedge$ und $\vee$ linksassoziativ; $\rightarrow$ 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:

$$\phi ::= P \mid \bot \mid (\neg\, \phi) \mid (\wedge\, \phi\, \ldots) \mid (\vee\, \phi\, \ldots) \mid (\rightarrow\, \phi\, \phi)$$

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. (`T` steht für `true` und `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 `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:

| | $\phi$ | $\neg\phi$ | |
|---|---|---|---|
| $\neg$ | `T` | `F` | $\neg\phi$ genau dann `true`, wenn $\phi$ `false`. |
| | `F` | `T` | |

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

$\wedge$

| $\phi$ | $\psi$ | $\phi \wedge \psi$ |
|---|---|---|
| T | T | T |
| T | F | F |
| F | T | F |
| F | F | F |

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

$\vee$

| $\phi$ | $\psi$ | $\phi \vee \psi$ |
|---|---|---|
| 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.

$\rightarrow$

| $\phi$ | $\psi$ | $\phi \rightarrow \psi$ |
|---|---|---|
| T | T | T |
| T | F | F |
| F | T | T |
| F | F | T |

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

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:

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

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

[4] eigentlich „ex falso sequitur quodlibet“.

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 $\phi$ definiert man den Wahrheitswert $[\![\phi]\!]_\alpha \in \mathbb{B}$ der Formel induktiv durch

$$(i) \quad [\![P]\!]_\alpha := \alpha(P) \text{ für alle } P \in \mathcal{P}$$

$$(ii) \quad [\![\bot]\!]_\alpha := \mathtt{F}$$

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

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

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

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

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

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

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

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

**Definition 4.5** (Allgemeingültigkeit)**.** Eine Formel $\phi$ heißt *allgemeingültig*, wenn für alle Strukturen $\alpha$ gilt: $[\![\phi]\!]_\alpha = \mathtt{T}$.
Man schreibt dann $\vDash \phi$ und nennt $\phi$ eine *Tautologie*.

**Definition 4.6** (Unerfüllbarkeit)**.** Eine Formel $\phi$ heißt *unerfüllbar*, wenn für alle Strukturen $\alpha$ gilt: $[\![\phi]\!]_\alpha = \mathtt{F}$.
Man schreibt dann $\nvDash \phi$ und nennt $\phi$ eine *Kontradiktion*.

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

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

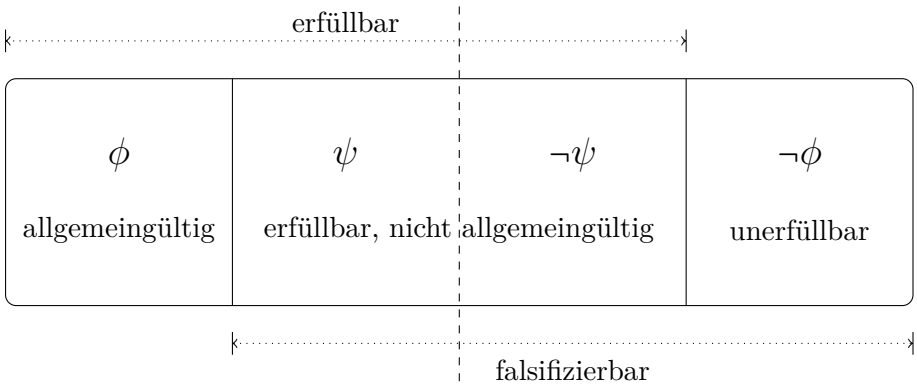



Abbildung 4.1: Dualitätsprinzip

## 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 $\phi$ ist* erfüllbar*, wenn es in der Wahrheitstafel* mindestens eine *Zeile mit dem Ergebnis* `T` *gibt.*

*Eine Formel $\phi$ 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 $\phi$ und $\psi$ sind *semantisch äquivalent*, geschrieben $\phi \equiv \psi$, wenn für alle Interpretationen $\alpha$ gilt: $[\![\phi]\!]_\alpha = [\![\psi]\!]_\alpha$.

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

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

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

**Satz 4.3** (Substitutionssatz)**.** *Sei $\phi$ eine Subformel von $\psi$ und $\phi'$ 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

# Formale Beweise durch natürliches Schließen

siehe Vorlesung

## 5.1 Schlussregeln

## 5.2 Beweisstrategien

## 5.3 Vollständigkeit des natürlichen Schließens

# 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 eine 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 $\phi$ 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 NNF(φ) {
// pre: φ hat keine Implikationen
// post: äquivalente Umformung von φ in NNF
      case {
            φ ist Literal:
                  return φ;
            φ hat die Form ¬¬φ1:
                  return NNF(φ1);
            φ hat die Form φ1 ∧ φ2:
                  return NNF(φ1) ∧ NNF(φ2);
            φ hat die Form φ1 ∨ φ2:
                  return NNF(φ1) ∨ NNF(φ2);
            φ hat die Form ¬(φ1 ∧ φ2):
                  return NNF(¬φ1 ∨ ¬φ2);
            φ hat die Form ¬(φ1 ∨ φ2):
                  return NNF(¬φ1 ∧ ¬φ2);
      }
}
```

## 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 $\phi$ ist in der *konjunktiven Normalform CNF*, wenn sie die Form $\phi_1 \wedge \phi_2 \wedge \cdots \wedge \phi_n$ hat mit lauter Klauseln $\phi_i$.

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 $\psi$ ist in der *disjunktiven Normalform DNF*, wenn sie die Form $\psi_1 \vee \psi_2 \vee \cdots \vee \psi_n$ hat mit lauter Monomen $\psi_i$.

**Beobachtung:** („Dualität“ von CNF und DNF)
Ist die Formel $\phi$ in CNF, dann ist $\neg\phi$ modulo simpler Transformationen in DNF.
Ist $\phi$ in CNF, dann hat $\phi$ die Form $\phi_1 \wedge \phi_2 \wedge \cdots \wedge \phi_n$ mit Klauseln $\phi_i$.
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 $\phi_i$ 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 $\phi$ in CNF. Dann gilt:*
*$\phi$ ist allgemeingültig $\Leftrightarrow$ Jede Klausel enthält ein komplementäres Paar von Literalen*

### DNF und Erfüllbarkeit

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

# Kapitel 7

# Die Komplexität des Erfüllbarkeitsproblems

## 7.1 Das Erfüllbarkeitsproblem

### Entscheidungsfragen der Aussagenlogik

In der Aussagenlogik kann man sich zu einer gegebenen Formel $\phi$ folgende Fragen stellen:

- Ist $\phi$ *allgemeingültig*?
  Diese Frage nennt man auch das Gültigkeitsproblem. Die Formel $\phi$ ist allgemeingültig, wenn sie für jede beliebige Interpretation der Aussagensymbole wahr ist. Ist eine Formel $\phi$ allgemeingültig, nennt man $\phi$ auch eine *Tautologie*.
- Ist $\phi$ *unerfüllbar*?
  Diese Frage nennt man auch das Unerfüllbarkeitsproblem. Die Formel $\phi$ ist unerfüllbar, wenn es keine Interpretation der Aussagensymbole gibt, die sie wahr macht. Ist eine Formel $\phi$ unerfüllbar, sagt man auch $\phi$ ist der *Widerspruch* oder eine *Kontradiktion*.
- Ist $\phi$ *erfüllbar*?
  Diese Frage nennt man auch das Erfüllbarkeitsproblem. Die Formel $\phi$ 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 $\phi$ *falsifizierbar*?
  Diese Frage nennt man auch das Widerlegungsproblem. Die Formel $\phi$ 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 $\phi$ ist genau dann allgemeingültig, wenn $\neg\phi$ unerfüllbar ist.
- Eine Formel $\phi$ 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 $\phi$, 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 $\phi$ $n$ 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 $4 \times 10^{12}$ 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.

### 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.

[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

Beispiel: 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: 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: 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: 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: 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 nichtdeterministischer Algorithmus in polynomieller Zeit lösen kann.

Beispiel: 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.

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

[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." [5, S. 1]

## 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.

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]

[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.

[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 $\phi$ 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*.

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

**Definition 8.4** (Hornformel)**.** Eine Hornformel ist eine Formel $\phi$ 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)
$H ::= \phi \wedge T \mid \phi \wedge Z \mid \phi \wedge R$

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

### Begründung dieser Definition

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

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 HORN(φ) {
// 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 $\phi$ erfüllbar ist.
2. Ist $\phi$ 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 $\phi$ ist.

# 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.

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

```
p cnf v c
```

[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.“[5, S. iv]

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** $\phi$ 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-Solver überhaupt startet, weil CNF als Eingabeform erwartet wird? Dies ist nicht der Fall:

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

## 9.2 Tseitin-Transformation

**Definition 9.1** (Erfüllbarkeitsäquivalenz)**.** Zwei Formeln der Aussagenlogik $\phi$ und $\psi$ heißen *erfüllbarkeitsäquivalent*, wenn gilt:

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

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

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

1. Führe für jede Subformel $\psi$ von $\phi$, die kein Atom ist, ein neues Atom $T_\psi$ hinzu.
2. Setze $\phi' = T_\phi$.
3. Durchlaufe den Syntaxbaum von $\phi$ und füge $\phi'$ 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

| | |
|---|---|
| $\phi = \neg\phi_1$ | $(\neg T_\phi \vee \neg T_{\phi_1}) \wedge (T_\phi \vee T_{\phi_1})$ |
| $\phi = \phi_1 \wedge \phi_2$ | $(\neg T_\phi \vee T_{\phi_1}) \wedge (\neg T_\phi \vee T_{\phi_2}) \wedge (T_\phi \vee \neg T_{\phi_1} \vee \neg T_{\phi_2})$ |
| $\phi = \phi_1 \vee \phi_2$ | $(T_\phi \vee \neg T_{\phi_1}) \wedge (T_\phi \vee \neg T_{\phi_2}) \wedge (\neg T_\phi \vee T_{\phi_1} \vee T_{\phi_2})$ |
| $\phi = \phi_1 \to \phi_2$ | $(T_\phi \vee T_{\phi_1}) \wedge (T_\phi \vee \neg T_{\phi_2}) \wedge (\neg T_\phi \vee \neg T_{\phi_1} \vee T_{\phi_2})$ |

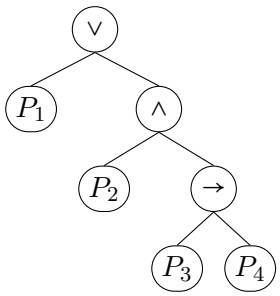
Die Tseitin-Transformation $\phi'$ einer Formel $\phi$ hat folgende Eigenschaften:

1. Die Menge der Atome von $\phi$ ist eine Teilmenge der Atome von $\phi'$.
2. Wenn $\alpha$ eine erfüllende Interpretation von $\phi$ ist, dann gibt es eine Erweiterung von $\alpha$ auf die Atome von $\phi'$, so dass diese Interpretation $\phi'$ erfüllt.
3. Ist $\alpha'$ eine erfüllende Interpretation von $\phi'$, dann erfüllt sie auch $\phi$.

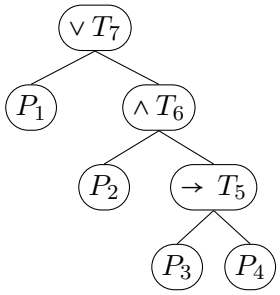
### Beispiel

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

[3] nach Grigori S. Tseitin, russischer Mathematiker, geb. 1936 [8]




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




Im Schritt 2 erstellen wir $\phi'$:

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

## 9.3 DPLL

**Definition 9.2** (Einzelklausel)**.** Eine Klausel einer Formel in CNF heißt *Einzelklausel*, wenn sie aus genau einem Literal besteht.

**Definition 9.3** (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 Varianten des folgenden Algorithmus von Martin Davis[4], Hilary Putnam[5], George Logemann[6] und Donald W. Loveland[7] ([2], [1]).

[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

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

```
    case {
        φ = ⊤:
            return true;
        φ = ⊥:
            return false;
        φ hat eine Einzelklausel (P̂):
            return DPLL(φ[⊤/P̂], M ∪ P̂);
        φ hat ein reines Literal P̂:
            return DPLL(φ[⊤/P̂], M ∪ P̂);
        otherwise:
            wähle zu einem Atom Q in der Formel mit Q̂ ∉ M
            return DPLL(φ[⊤/Q], M ∪ Q) ∨ DPLL(φ[⊥/Q], M ∪ ¬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 $\phi$ wählen. Es gilt dann natürlich: $\phi$ erfüllbar $\Leftrightarrow \phi \wedge Q$ erfüllbar oder $\phi \wedge \neg Q$ erfüllbar.

### Beispiel

$\phi$ 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 $P_2 =$ `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 $P_1 =$ `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 $P_2$ = `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.

### Bemerkungen

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“.

   Klauselernende 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 den DPLL-Algorithmus einsetzen, wird beschrieben in [6] und [5].

# Kapitel 10

# Anwendungen der Aussagenlogik in der Softwaretechnik

siehe Vorlesung

## 10.1 Codeanalyse

## 10.2 Featuremodelle für Softwareproduktlinien

# 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 $P \wedge Q \to R$ — 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 Univer-

sum 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]

**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 $r : \mathbb{U}^n \to \mathbb{B}$

[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.

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:

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

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

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

- 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:

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

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) $\bot$ 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 $\phi$ eine Formel, dann auch $(\neg\phi)$.
Sind $\phi$ und $\psi$ Formeln, dann auch $(\phi \wedge \psi)$, $(\phi \vee \psi)$ und $(\phi \rightarrow \psi)$.

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

In Backus-Naur-Darstellung:

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

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) [3, S.114f]

**Bindungsregeln**
$\neg, \forall x, \exists x$ binden am stärksten, dann
$\wedge$ und $\vee$ (linksassoziativ) und schließlich
$\rightarrow$ (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 $\phi$ eine Formel der Prädikatenlogik. Ein Vorkommen der Variablen $x$ in $\phi$ 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 $\phi$, 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 $\phi$ 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 $\phi$, 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.

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

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

(i) 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 Konstantensysmbolen 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 $\phi$, geschrieben

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

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

$$(i) \quad [\![P(t_1, \ldots, t_n)]\!]_l^{\mathcal{M}} := \begin{cases} \mathtt{T} \text{ falls } (I(t_1), \ldots I(t_n)) \in P^{\mathcal{M}} \\ \mathtt{F} \text{ sonst} \\ \text{bzw. } I(P) \text{ falls } P0\text{-är} \end{cases}$$

$$(ii) \quad [\![s = t]\!]_l^{\mathcal{M}} := \begin{cases} \mathtt{T} \text{ falls } I(s) = I(t) \\ \mathtt{F} \text{ sonst} \end{cases}$$

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

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

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

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

$$(vii) \quad [\![\forall x\ \phi]\!]_l^{\mathcal{M}} := \begin{cases} \mathtt{T} \text{ falls für alle } a \in \mathbb{U} \text{ gilt: } [\![\phi]\!]_{l[x \mapsto a]}^{\mathcal{M}} = \mathtt{T} \\ \mathtt{F} \text{ sonst} \end{cases}$$

$$(viii) \quad [\![\exists x\ \phi]\!]_l^{\mathcal{M}} := \begin{cases} \mathtt{T} \text{ falls es existiert ein } a \in \mathbb{U} \text{ mit: } [\![\phi]\!]_{l[x \mapsto a]}^{\mathcal{M}} = \mathtt{T} \\ \mathtt{F} \text{ sonst} \end{cases}$$

$$(ix) \quad [\![\top]\!]_l^{\mathcal{M}} := \mathtt{T}$$

$$(x) \quad [\![\bot]\!]_l^{\mathcal{M}} := \mathtt{F}$$

## 13.2 Semantische Folgerung und Äquivalenz

**Definition 13.5** (Semantische Folgerung)**.** Sei $\Gamma$ eine Menge prädikatenlogischer Formeln und $\phi$ eine prädikatenlogische Formel. Man sagt:

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

Besteht $\Gamma$ nur aus einer Formel, sage $\psi$, dann schreibt man auch $\psi \models \phi$.

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

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

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

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

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

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

Auch in der Prädikatenlogik gilt das Dualitätsprinzip:

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

## 13.3 Fundamentale Äquivalenzen der Prädikatenlogik

**Satz 13.2.** *$\phi$, $\psi$ und $\chi$ seien Formeln der Aussagenlogik, wobei in $\chi$ 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

siehe Vorlesung

## 14.1 Schlussregeln

## 14.2 Beweisstrategien

## 14.3 Vollständigkeit des natürlichen Schließens

# 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: $\bot$

(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 defininerten 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 $\bot$ ist eine Formel.

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

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

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

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

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

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

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

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:

$$\Diamond\,\phi \stackrel{\mathsf{def}}{=} \neg\bot\,\mathcal{U}\,\phi$$
$$\Box\,\phi \stackrel{\mathsf{def}}{=} \neg\Diamond\,\neg\phi$$
$$\phi\,\mathcal{W}\,\psi \stackrel{\mathsf{def}}{=} (\phi\,\mathcal{U}\,\psi) \vee \Box\,\phi$$
$$\phi\,\mathcal{R}\,\psi \stackrel{\mathsf{def}}{=} \neg(\neg\phi\,\mathcal{U}\,\neg\psi)$$

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 $(S, s_0, \to, L)$ bestehend aus

- einer Menge von Zuständen $S$,
- einem ausgezeichneten Startzustand $s_0 \in S$,
- einer Übergangsrelation $\to \subseteq S \times S$, die jedem Zustand $s$ einen Folgezustand $s'$ zuordnet (d.h. $\forall s\, \exists s'$ mit $s \to s'$) und
- einer Beschriftungsfunktion $L : S \to \mathbb{P}(\mathcal{P})$ 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 $\alpha_s : \mathcal{P} \to \mathbb{B}$ 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 [4, 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.

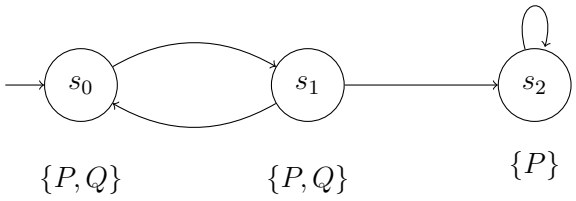



Abbildung 19.1: Beispiel einer Kripke-Struktur

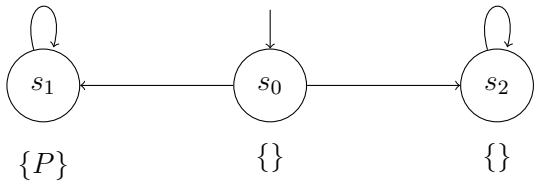



Abbildung 19.2: Beispiel für die Semantik der Negation

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

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

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

Hat man einen Pfad $\pi = s_1 \to s_2 \to s_3 \ldots$ gegeben, dann bezeichnet man mit $\pi^i$ den Pfad, der im $i$-ten Zustand von $\pi$ 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 die Semantik 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 $\phi$ 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 $\phi$

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

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

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

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

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

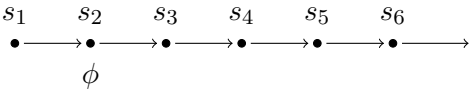
$$(vi) \quad [\![\phi \to \psi]\!]^{\mathcal{K}}_{\pi} := \begin{cases} \mathtt{T} \text{ falls } [\![\phi]\!]^{\mathcal{K}}_{\pi} = \mathtt{F} \text{ oder } [\![\psi]\!]^{\mathcal{K}}_{\pi} = \mathtt{T} \\ \mathtt{F} \text{ sonst} \end{cases}$$

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

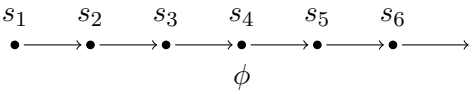
$$(viii) \quad [\![\phi \,\mathcal{U}\, \psi]\!]^{\mathcal{K}}_{\pi} := \begin{cases} \mathtt{T} \text{ falls } \exists_{\, i \geq 1} \text{ mit } [\![\psi]\!]^{\mathcal{K}}_{\pi^i} = \mathtt{T} \\ \quad \text{und } \forall_{\, j=1,\ldots i-1} \, [\![\phi]\!]^{\mathcal{K}}_{\pi^j} = \mathtt{T} \\ \mathtt{F}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

**Veranschaulichung der Semantik der temporalen Operatoren der LTL**

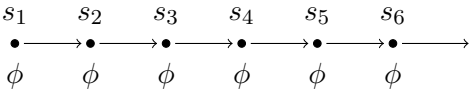
- $\bigcirc\,\phi$ bedeutet, dass $\phi$ im nächsten Zustand gilt:



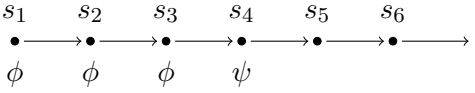

- $\Diamond\,\phi$ bedeutet, dass $\phi$ irgendwann auf dem Pfad gilt:



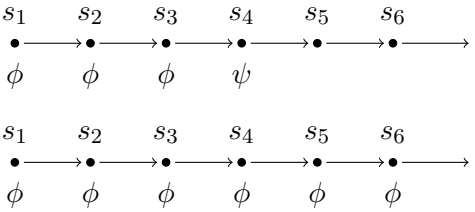

- $\Box\,\phi$ bedeutet, dass $\phi$ immer auf dem Pfad gilt:



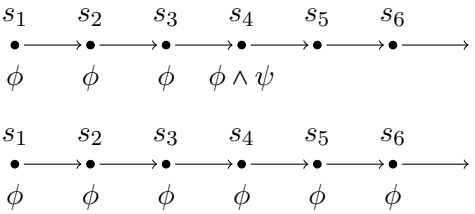

- $\phi\,\mathcal{U}\,\psi$ bedeutet, dass $\psi$ irgendwann auf dem Pfad gilt, und dass bis dahin auf jeden Fall $\phi$ wahr ist:




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




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




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

- Sei $\pi$ ein *Pfad* über $\mathcal{K}$. Dann sagt man, dass $\pi$ eine Formel $\phi$ erfüllt, geschrieben $\pi \vDash \phi$, wenn gilt $[\![\phi]\!]^{\mathcal{K}}_{\pi} = \mathtt{T}$.
- Sei $s$ ein *Zustand* von $\mathcal{K}$. Dann sagt man, dass $s$ eine Formel $\phi$ erfüllt, geschrieben $s \vDash \phi$, wenn für alle Pfade $\pi$, die mit $s$ beginnen, gilt $\pi \vDash \phi$.
- Man sagt, dass die *Kripke-Struktur* $\mathcal{K}$ eine Formel $\phi$ 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 $s_2$.
- $\mathcal{K} \vDash \Box (\neg q \rightarrow \Box (P \wedge \neg Q))$
  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 $s_0 \rightarrow s_2 \rightarrow s_2 \rightarrow \ldots$ ist $P$ niemals true.
- $\mathcal{K} \nvDash \neg\Diamond P$
  denn auf dem Pfad $s_0 \rightarrow s_1 \rightarrow s_1 \rightarrow \ldots$ ist $P$ schließlich true.

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

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

**Definition 19.6** (Semantische Äquivalenz)**.** Zwei Formeln $\phi$ und $\psi$ sind *semantisch äquivalent*, geschrieben $\phi \equiv \psi$, wenn für alle Pfade $\pi$ in beliebigen Kripke-Strukturen gilt: $\pi \vDash \phi \Leftrightarrow \pi \vDash \psi$.

## 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

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

- 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


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

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

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

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

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

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

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

[8] 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.