Logik

Vorlesung im Wintersemester 2014/2015

Stephan Kreutzer

Technische Universität Berlin

Version vom 17.10.2014

Inhaltsverzeichnis

1. Einführung

Dieses Buch ist im wesentlichen das Skript zu einer Vorlesung, die als Pflichtmodul im Bachelorstudiengang Informatik an der Technischen Universität Berlin gehalten wird. Thema dieser Vorlesung ist eine Einführung in die *Logik* wie sie im Kontext der Informatik benutzt wird.

Das Wort „Logik“ kommt vom altgriechischen Wort „logos“, die „Vernunft“. Die Logik hat ihren Ursprung in der (griechischen) Philosophie, in der Frage, was richtiges Argumentieren eigentlich bedeutet, d.h. was man aus einer Aussage folgerichtig ableiten kann. In diesem Sinne wird das Wort auch heute noch umgangssprachlich verwendet, etwa in dem Ausdruck „das ist ja logisch“.

In der Informatik hat Logik zwar auch diese Rolle als Basis logischen, d.h. korrekten Schließens, findet jedoch viel weitgehendere Anwendungen. Wir wollen dies anhand einiger Beispiele verdeutlichen.

Relationale Datenbanken Relationale Datenbanken sind auch heute noch das Standardmodell für Datenbanken in der industriellen Praxis. Eine relationale Datenbank besteht aus einer Menge von Tabellen bzw. Relationen, woraus sich der Name ableitet.

Zum Beispiel könnte eine Filmdatenbank wie z.B. IMDB, Tabellen

- `Schauspieler`(Vorname, Name, Geburtsjahr)
- `Film`(Titel, Regisseur, Entstehungsjahr)
- `Cast`(Filmtitel, Schauspieler)

haben, die Informationen über Schauspieler und Filme beinhalten sowie eine Tabelle, in der für jeden Film die Schauspieler aufgelistet sind, die in dem Film mitgespielt haben.

Ein Beispiel für eine Tabelle `Schauspieler` wäre etwa die folgende Tabelle, die den Namen, Vornamen und Geburtsdatum einiger Schauspieler enthält. Dabei entspricht jede Zeile einem Eintrag in die Tabelle und damit einem Schauspieler.

Schauspieler		
Kirstin	Dunst	1982
George	Clooney	1961
...	...	...

Ebenso könnten wir eine Tabelle

Film		
Melancholia	Lars von Trier	2011
Good Night, and Good Luck	George Clooney	2007
...	...	...

für Filme sowie eine Tabelle

Cast		
Melancholia	Kirsten Dunst	
Good Night, and Good Luck	George Clooney	
...	...	...

für die Schauspieler eines Film anlegen.

Das wichtigste Problem im Bereich der Datenbanken ist natürlich die Frage, wie wir möglichst effizient Informationen aus der Datenbank erhalten können. Datenbanksysteme stellen daher sogenannte *Anfragesprachen* zur Verfügung, mit denen Informationen aus Datenbanken abgefragt werden können. Die gängigsten davon ist die sogenannten *konjunktiven Anfragen* und die *Standard Query Language* (SQL).

Neben SQL gibt es aber auch noch verschiedene andere Anfragesprachen und natürlich stünde es uns frei, unsere eigene Sprache zu definieren.

Für jede Anfragesprache sind aber folgende Eigenschaften wichtig:

- Die Anfragesprache muss uns erlauben, die gewünschten Informationen eindeutig beschreiben zu können. D.h. es muss zum einen klar sein, welche Anfragen überhaupt gültige Anfragen sind (d.h. die *Syntax* muss präzise definiert sein), zum anderen muss eindeutig bestimmt sein, was Anfragen in der Sprache überhaupt bedeuten (d.h. die *Semantik* oder *Bedeutung* der Anfragen muss sauber definiert sein).

- Wir möchten möglichst effiziente Verfahren haben, Anfragen in Datenbanken auszuwerten. Man bezeichnet das als das *Auswertungsproblem* der Anfragesprache.

Für diese Aufgaben ist die natürliche Sprache, als z.B. deutsch, als Anfragesprache ungeeignet, da natürliche Sprache sehr viele Doppeldeutigkeiten enthält, die wir in der natürlichen Kommunikation mit Hilfe von Hintergrund- oder Kontextwissen richtig ausflösen. Dies ist aber für Datenbanken nicht praktikabel. Daher werden formale Sprachen, oder eben *Logiken*, wie eben SQL verwendet.

Ähnliche Anforderungen werden an Spezifikationssprachen in der formalen Verifikation gestellt. In diesem Anwendungsfeld sollen möglichst automatisch Hardware- oder Softwarekomponenten auf Korrektheit überprüft werden. Auch hier braucht man Sprachen, in denen die gewünschten Korrektheitseigentschaften formal sauber und uneindeutig formalisiert werden können.

Die gleichen Anforderungen finden sich auch im Bereich der Wissenrepräsentation und in vielen anderen Gebieten der Informatik.

Logik in der Informatik Die bisherigen Beispiele stammen aus sehr unterschiedlichen Bereichen der Informatik mit großer praktischer d.h. industrieller Relevanz. Gemeinsam ist allen Beispielen, dass wir jeweils formal saubere Methoden brauchen, um Informationen wie Datenbankanfragen, Expertenwissen oder Prozessspezifikationen zu formalisieren und mit den formalisierten Informationen algorithmisch zu arbeiten. Konkret brauchen wir in allen Beispielen formale Sprachen mit einer eindeutigen, formalen Syntax und Semantik, sowie Algorithmen um Ausdrücke in diesen Sprachen etwa in Datenbanken auszuwerten oder auf Erfüllbarkeit zu überprüfen.

Genau diesen Zweck erfüllt die Logik in der Informatik. Eine Logik ist letztlich nichts anderes als eine formale Sprache mit genau definierter Semantik. D.h. sie besteht aus einer Menge logischer Formeln sowie einer Methode, die jeder Formel ihre Bedeutung (Semantik) zuweist.

Z.B. ist die Formel $\exists x \forall y (x + x \leqslant y)$ eine korrekte Formel der Prädikatenlogik. Ihre Bedeutung hängt davon ab, in welchem Kontext wir die Formel betrachten. In den natürlichen Zahlen besagt sie z.B., dass es eine Zahl gibt, die zu sich selbst addiert die kleinste Zahl ist, was natürlich stimmt. In den reellen Zahlen wäre diese Aussage aber falsch.

Im weiteren Sinne könnte man auch die Anfragesprache SQL als Logik auffassen. Typischerweise aber versucht man in Logiken, die Syntax möglichst minimal zu halten, vermeidet also sogenannten *syntaktischen Zucker*, d.h. die Hinzunahme von Ausdrücken zur Sprache, die sich bereits anders auch definieren lassen. In Sprachen für den praktischen Grbrauch ist das natürlich gängige Praxis um die Benutzung der Sprache zu vereinfachen.

Es liegt nahe, dass man für die unterschiedlichen Anwendungen auch unterschiedliche Logiken braucht. So wird eine Logik, die sich gut als Anfragesprache für relationale Datenbanken eignet, sicherlich sehr andere Anforderungen erfüllen müssen, als eine Logik, die im Bereich der Verifikation verwendet wird. Entsprechend sind in der Informatik (im Unterschied z.B. der Mathematik) sehr viele Logiken definiert worden, von denen wir einige in dieser Vorlesung kennen lernen werden.

Für jede Logik werden wir jeweils folgende Aspekte betrachten.

- Die Syntax und Semantik der Logik. Wie sehen Formeln der Logik aus, was bedeuten sie.
- Möglichkeiten zur Formelmanipulation. Hier werden wir z.B. definieren, wann zwei Formeln derselben Logik äquivalent sind, d.h. das gleiche beschreiben.
- Auswerten von Formeln. Hier suchen wir nach effizienten Algorithmen, um Formeln auszuwerten.
- Erfüllbarkeit von Formeln. Hier suchen wir ebenfalls nach Möglichkeiten zu entscheiden, ob eine Formeln überhaupt erfüllbar ist, d.h. irgendwo gilt.

Während der Sinn der Formelauswertens bzw. des Erfüllbarkeitstests schon erwähnt wurde, ist der Punkt „Formelmanipulation“ vielleicht nicht auf Anhieb verständlich. Betrachten wir daher noch einmal das Beispiel der Datenbanken.

Wie schon gesagt ist eines der zentralen Probleme im Datenbankbereich die schnelle Auswertung von Anfragen. Dabei kann allerdings die gleiche Anfrage in verschiedenen Arten ausgewerten werden. Darüber hinaus können verschiedene Formeln, die die gleiche Anfrage definieren, sehr unterschiedlich schwierig auszuwerten sein. In der Praxis werden daher Anfragen umgeschrieben, d.h. in äquivalente Anfragen übersetzt, die besser auszuwerten sind. Wichtig ist dabei natürlich, dass durch das Umschreiben der Anfrage die Bedeutung nicht verändert wird, d.h. das man wirklich äquivalente Anfragen erhält. Aus diesem Grunde spielen Äquivalenzumformungen, d.h. Regeln, die es erlauben, Formeln in äquivalente Formeln zu überführen, eine wichtige Rolle.

Bezüglich der Komplexität ist in den allermeisten Fällen das Auswerten von Formeln erheblich einfacher als die Frage nach der Erfüllbarkeit von Formeln. Während sich die Auswertungsprobleme der hier behandelten Logiken in die Komplexiätsklassen wie NP und PSPACE einordnen lassen, werden wir später sehen, dass einige Erfüllbarkeitsprobleme sogar unentscheidbar sind, also durch keinen Algorithmus mit beliebiger Laufzeit gelöst werden können.

Wir werden uns im ersten Teil des Skripts mit der grundlegendsten aller Logiken beschäftigen, der *Aussagenlogik*. Auf Basis dieser Logik können wir dann ausdrucksstärkere Logiken definieren. Dies wird im zweiten und dritten Teil geschehen, in dem wir die Prädikatenlogik behandeln. Wir schließen das Skript mit einer kurzen Einführung in die Rekursions- oder Berechenbarkeitstheorie ab.

Danksagung. Ich möchte mich an dieser Stelle sehr herzlich bei meinen Kollegen Prof. Martin Grohe (RWTH Aachen) und Prof. Nicole Schweikardt (Humboldt-Universität Berlin) für viele hilfreiche Diskussionen über Inhalt und Form des Skriptes bedanken. Ebenso möchte ich mich bei meinen Mitarbeitern Christoph Dittmann, Roman Rabinovich und Sebastian Siebertz für die tatkräftige Unterstützung bei der Erstellung des Skripts bedanken. Schließlich gilt mein Dank den Studierenden meiner Vorlesung an der Technischen Universität Berlin, die mich auf Fehler im Skript hingewiesen und Verbesserungsvorschläge gemacht haben.

Teil I.

Aussagenlogik

2. Grundlagen der Aussagenlogik

2.1. Einleitung

Als Einführung in die Aussagenlogik betrachten wir folgendes Beispiel.

Beispiel 2.1 Betrachten wir folgende Aussage.

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

Also standen Taxis am Bahnhof.

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

Betrachten wir als Nächstes folgendes Beispiel.

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

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

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

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

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

Wir betrachten ein weiteres Beispiel.

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

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

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

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

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

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

In diesem Teil der Vorlesung werden wir Methoden kennen lernen um

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

2.2. Syntax und Semantik

In diesem Kapitel führen wir die *Syntax* und *Semantik* der Aussagenlogik ein. Generell sind Logiken formale Sprachen, d.h. Mengen oder Klassen von Wörtern über einem bestimmten Alphabet, zusammen mit einer Vorschrift, was die einzelnen Wörter (genannt Formeln) eigentlich bedeuten sollen. Unter der *Syntax* einer Logik versteht man dabei die Festlegung, welche Wörter korrekte Formeln der Logik sind. Die *Semantik* legt die Bedeutung der einzelnen Formeln fest.

2.2.1. Syntax der Aussagenlogik

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

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

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

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

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

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

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

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

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

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

- $\varphi_1 := (V_0 \vee V_1)$
- $\varphi_2 := (((V_0 \vee \bot) \leftrightarrow (\top \wedge V_1)) \wedge \neg(V_2 \wedge V_0))$

Die folgenden Ausdrücke sind hingegen keine korrekten Formeln.

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

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

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

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

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

- Ist φ atomar, dann ist $\text{sub}(\varphi) := \{\varphi\}$.

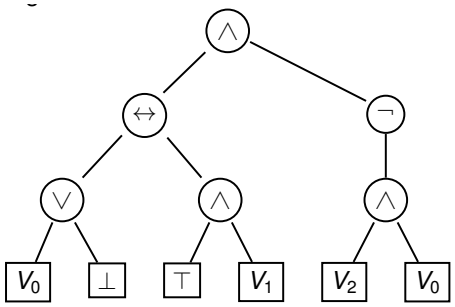


Abbildung 2.1.: Syntaxbaum der Formel $(((V_0 \vee \bot) \leftrightarrow (\top \wedge V_1)) \wedge \neg(V_2 \wedge V_0))$

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

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

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

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

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

⊣

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

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

Basisfälle. Wir definieren zunächst die Funktionswerte für atomare Formeln.

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

Induktionsschritt. Danach werden die Funktionswerte für zusammengesetzte Formeln definiert.

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

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

Basisfälle:

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

Induktionsschritt:

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

2.2.2. Semantik der Aussagenlogik

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

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

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

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

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

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

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

Basisfall.

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

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

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

Konjunktion

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

Disjunktion

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

Implikation

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

Biimplikation

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

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

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

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

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

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

Exklusives Oder

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

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

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

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

Um unnötige Klammern zu vermeiden,

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

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

2.3. Formalisierung natürlichsprachlicher Aussagen

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

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

Die darin enthaltenen elementaren, oder atomaren, Aussagen sind

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

Obige Aussage kann also wie folgt formalisiert werden

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

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

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

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

Die darin enthaltenen atomaren Aussagen sind

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

Insgesamt wird die Aussage also durch

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

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

Als letztes Beispiel betrachten wir noch einmal Beispiel 2.1.

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

Also standen Taxis am Bahnhof.

Als Aussagenvariablen wählen wir

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

Die Aussage wird dann durch

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

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

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

2.4. Erfüllbarkeit und Allgemeingültigkeit

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

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

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

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

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

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

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

Aus der Definition folgt direkt folgende Proposition.

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

Wie oben schon besprochen ist also die durch

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

formalisierte Aussage über Peter und seine Verabredung genau dann korrekt, wenn φ allgemeingültig ist. Wir brauchen also Methoden, um Allgemeingültigkeit und Erfüllbarkeit von Formeln zu entscheiden. In dieser Vorlesung werden wir drei Methoden kennen lernen:

- *Wahrheitstafeln*,
- *Resolution*,
- *Sequenzenkalkül*.

2.4.1. Wahrheitstafeln

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

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

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

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

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

X_H	X_M	X_S	$(X_H \to X_M)$	$(X_S \to X_H)$	$\begin{smallmatrix}(X_H\to X_M)\wedge\\(X_S\to X_H)\end{smallmatrix}$	$(X_S \to X_M)$	φ
0	0	0	1	1	1	1	1
0	0	1	1	0	0	0	1
0	1	0	1	1	1	1	1
0	1	1	1	0	0	1	1
1	0	0	0	1	0	1	1
1	0	1	0	1	0	0	1
1	1	0	1	1	1	1	1
1	1	1	1	1	1	1	1

Dies wird als die *erweiterte Wahrheitstafel* bezeichnet.

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

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

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

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

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

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

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

Betrachten wir nun noch einmal das Beispiel der Syllogismen (Beispiel 2.3 und 2.16), d.h. die Formel

$$\varphi := \big((X_M \to X_S) \wedge (X_K \to X_M)\big) \to (X_K \to X_S).$$

Die zugehörige Wahrheitstafel ist

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

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

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

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

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

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

Diese Beobachtung ist Kern des sogenannten *Wahrheitstafelverfahrens*.

Wahrheitstafelverfahren.

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

Ziel: Entscheide, ob φ erfüllbar ist.

Methode: (1) Berechne die Wahrheitstafel für φ.

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

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

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

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

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

Bemerkung 2.22 Das Erfüllbarkeitsproblem der Aussagenlogik ist eines der am besten studierten Probleme der Informatik. Es ist algorithmisch „schwer“ zu lösen, genauer gesagt ist das Problem NP-vollständig (siehe Satz 3.20). Allerdings existieren Verfahren, die das Problem für viele in der Praxis vorkommende Formeln sehr effizient lösen können. (Das Wahrheitstafelverfahren gehört nicht dazu). Diese haben wichtige Anwendungen in der Informatik, z.B. in der Verifikation. ⊣

2.5. Äquivalenz und Normalformen

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

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

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

2.5.1. Äquivalenz von Formeln

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

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

(vgl. Def. 2.17) ⊣

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

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

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

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

Wir beweisen die Äquivalenzen mit Hilfe einer Wahrheitstafel.

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

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

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

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

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

Substitution.

Informell kann man das Substitutionslemma wie folgt formulieren.

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

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

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

Wir formulieren das Lemma nun formal.

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

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

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

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

Formal ist dies wie in der folgenden Definition formuliert.

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

Induktionsbasis.

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

Induktionsschritt.

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

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

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

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

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

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

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

Dann gilt

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

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

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

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

Strukturelle Induktion.

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

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

Induktionsbasis. Alle atomaren Formeln φ haben die Eigenschaft A.

Induktionsschritt.

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

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

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

Sei $\mathcal{S}$ eine Substitution.

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

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

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

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

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

Induktionsbasis.

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

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

Induktionsschritt.

- *Negation.*

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

- *Konjunktion.*

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

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

⊣

Mit Hilfe der Behauptung können wir nun das Substitutionslemma beweisen.

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

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

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

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

Das schließt den Beweis des Substitutionslemmas ab. □

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

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

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

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

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

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

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

Mit Hilfe des Substitutionslemmas erhält man nun sofort folgende Konsequenz.

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

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

Wir würden nun gerne weiter folgern, dass

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

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

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

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

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

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

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

Mit Hilfe des Ersetzungslemmas können wir nun folgendes beweisen.

Korollar 2.35 *Für alle Formeln φ, ψ:*

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

Das folgende Theorem listet einige häufig benutzte Äquivalenzen auf.

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

In beiden Fällen gilt also die Äquivalenz.

□

Notation 2.37 Die folgende Notation ist oft nützlich:

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

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

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

Wie wir oben schon gesehen haben, gelten folgende Äquivalenzen.

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

Mit Hilfe des Ersetzungslemmas erhalten wir daher folgende Aussage.

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

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

2.5.2. Normalformen

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

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

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

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

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

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

Die Arbeitsweise des Algorithmus' wird im folgenden Beispiel demonstriert.

Algorithmus NNF(φ).

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

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

Algorithmus. Wiederhole die folgenden Schritte.

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

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

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

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

Setze $\varphi := \varphi'$.

Abbildung 2.2.: Ein Algorithmus für die NNF.

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

Der Beweis des Lemmas folgt sofort aus der Behautung. □

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

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

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

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

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

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

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

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

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

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

Einschub: Boolesche Funktionen

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

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

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

Beispiel 2.50 Definiere

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

⊣

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

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

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

Umgekehrt kann jede Boolesche Funktion durch eine Formel definiert werden.

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

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

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

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

Wir definieren nun die Funktion f durch die Formel

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

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

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

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

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

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

Das schließt den Beweis ab. □

Beispiel 2.53 Sei

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

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

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

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

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

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

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

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

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

Zurück zu Normalformen

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

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

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

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

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

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

□

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

2.6. Semantische Folgerung

Erinnern wir uns an das Beispiel vom Anfang der Vorlesung:

Voraussetzungen.

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

Folgerung. Also standen Taxis am Bahnhof.

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

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

verwendet. Die Argumentation kann wie folgt zusammengefasst werden.

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

folgern wir *Folgerung:* C

Frage. Wie können wir solche logischen Schlüsse formalisieren?

- Gegeben eine Menge von Voraussetzungen:

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

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

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

Damit zusammenhängend stellen sich algorithmische Fragen.

Frage. Wie können wir solche Folgerungen berechnen?

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

Das bedeutet: Wir wollen zeigen, dass

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

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

- *Aussagenlogische Resolution*
- *Der aussagenlogische Sequenzenkalkül*

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

3. Erfüllbarkeit und Entscheidbarkeit

3.1. Der Kompaktheitssatz der Aussagenlogik

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

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

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

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

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

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

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

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

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

Sei Φ also eine unendliche Menge aussagenlogischer Formeln. Ohne Beschränkung der Allgemeinheit nehmen wir an, dass es keine zwei verschiedenen Formeln $\psi, \psi' \in \Phi$ gibt, so dass $\psi \equiv \psi'$. Denn, angenommen, es gäbe solche Formeln. Dann ist Φ genau dann erfüllbar, wenn $\Phi \setminus \{\psi'\}$ erfüllbar ist. Es reicht also, den Beweis für Formelmengen zu zeigen, in denen alle Formeln paarweise nicht äquivalent sind.

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

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

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

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

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

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

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

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

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

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

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

Beweis. Sei $\varphi \in \Phi$. Da φ nur endlich viele Variablen enthält, ist $\varphi \in \Phi_n$ für ein n. Es gilt also $\beta_i \models \varphi$ für alle $i \geqslant n$, insbesondere also $\beta \models \varphi$ für alle $\beta \in I_n$.

Sei $\beta \in I_n$. So ein β existiert, da wegen $(*)$ $I_n \neq \varnothing$. Da nach $(*)$ für alle $i \leqslant n$ gilt $\alpha(X_i) = \beta(X_i)$ und $\beta \models \varphi$, folgt also $\alpha \models \varphi$. ⊣

Aus der Behauptung folgt sofort der erste Teil des Satzes.

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

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

Die Umkehrung ist trivial. □

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

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

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

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

3.2. Aussagenlogische Resolution

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

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

können mit Hilfe der Resolution auch semantische Folgerungen überprüft werden.

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

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

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

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

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

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

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

Für praktische Anwendungen der Resolutionsmethode ist folgender Satz nützlich.

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

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

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

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

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

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

in KNF schreiben wir als *Klauselmenge* wie folgt:

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

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

Die folgende Definition formalisiert diese Begriffe.

Definition 3.7

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

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

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

- Eine Klauselmenge $\mathcal{C}$ ist erfüllbar, wenn $\varphi(\mathcal{C})$ erfüllbar ist.
- Wenn $\mathcal{C}$ nur eine Klausel C enthält, schreiben wir einfach nur $\beta \models C$.

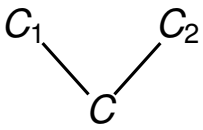
Offenbar erfüllt eine Belegung β eine Klauselmenge $\mathcal{C}$, wenn jede Klausel $C \in \mathcal{C}$ ein Literal L enthält, so dass $[\![L]\!]^\beta = 1$.

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

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

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

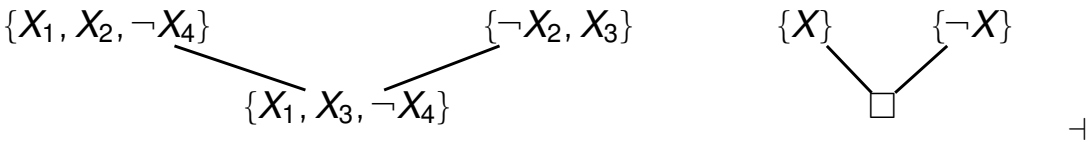
Wir werden Resolventen oft wie folgt graphisch darstellen



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

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

Die folgende Abbildung stellt die beiden Resolventen graphisch dar.



⊣

Bemerkung 3.10 Zwei Klauseln können mehr als eine Resolvente haben.

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

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

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

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

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

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

Insgesamt gilt also $\beta \models C$.

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

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

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

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

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

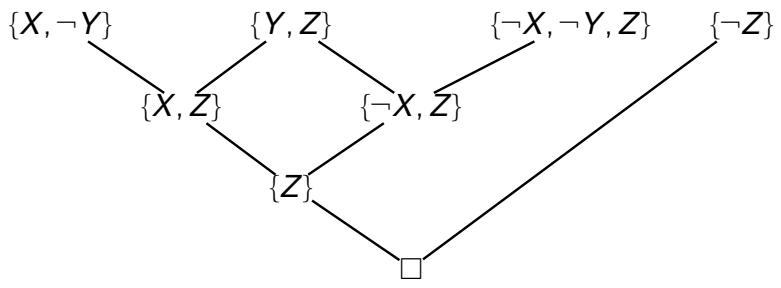
Wir sagen, dass C einen *Resolutionsbeweis* aus $\mathcal{C}$ hat und schreiben dies als $\mathcal{C} \vdash_R C$.

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

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

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

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



⊣

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

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

Zum Beweis des Satzes zeigen wir zunächst das folgende Lemma.

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

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

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

Wir zeigen als Nächstes die Umkehrung des Korollars.

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

Beweis. Wir zeigen zunächst die folgende Behauptung.

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

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

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

und

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

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

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

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

Falls $(C_1, \ldots, C_s)$ schon eine Ableitung von $\square$ aus $\mathcal{C}$ ist, sind wir fertig. Anderenfalls werden Klauseln C_i benutzt, die aus $\mathcal{C}$ durch Entfernen von $\neg V_n$ entstanden sind, d.h. $C_i \cup \{\neg V_n\} \in \mathcal{C}$.

Fügen wir zu diesen Klauseln und allen daraus folgenden Resolventen wieder $\neg V_n$ hinzu, so erhalten wir eine Ableitung $(C'_1, \ldots, C'_s)$ von $\{\neg V_n\}$ aus $\mathcal{C}$.

Analog ist entweder $(D_1, \ldots, D_t)$ bereits eine Ableitung von $\square$ aus $\mathcal{C}$ oder wir erhalten eine Ableitung $(D'_1, \ldots, D'_t)$ von $\{V_n\}$ aus $\mathcal{C}$. Ein weiterer Resolutionsschritt auf $\{\neg V_n\}$ und $\{V_n\}$ ergibt dann $\square$. In diesem Fall ist also $(C'_1, \ldots, C'_s, D'_1, \ldots, D'_t, \square)$ eine Resolutionswiderlegung von $\mathcal{C}$. ⊣

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

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

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

Zusammengenommen ergeben die beiden vorherigen Aussagen also folgenden Satz.

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

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

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

3.3. Das aussagenlogische Erfüllbarkeitsproblem

Definition 3.19 Das aussagenlogische Erfüllbarkeitsproblem (SAT) ist das Problem

SAT

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

Problem. Entscheide, ob φ erfüllbar ist.

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

3.3.1. Erinnerung: NP-Vollständigkeit

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

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

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

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

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

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

Eine Konfiguration (q, w, p) mit $w := w_1 \dots w_k$ und $p \leqslant k$ hat Nachfolger

$$(q', w_1 \dots w_{p-1} w' w_{p+1} \dots w_{k'}, p')$$

für alle $(q, w_p, q', w', m) \in \Delta$, wobei $p' := p + m$ und

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

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

Die durch eine NTM $\mathcal{M}$ akzeptierte Sprache $\mathcal{L}(\mathcal{M})$ ist definiert als

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

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

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

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

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

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

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

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

Im folgenden geben wir einige der wichtigsten Komplexitätsklassen an.

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

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

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

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

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

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

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

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

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

3.3.2. Der Satz von Cook und Levin

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

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

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

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

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

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

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

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

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

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

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

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

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

Dabei haben die Variablen folgende intendierte Bedeutung.

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

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

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

Als nächstes konstruieren wir folgende Formel

$$\begin{aligned} \textsc{Conf}(\overline{C}) := & \bigvee_{q \in Q} \Big(Q_q \wedge \bigwedge_{q' \neq q} \neg Q_{q'} \Big) \\ & \wedge \bigvee_{p \leqslant p(n)} \Big(P_p \wedge \bigwedge_{p' \neq p} \neg P_{p'} \Big) \\ & \wedge \bigwedge_{1 \leqslant i \leqslant p(n)} \bigvee_{a \in \Gamma} \big(S_{a,i} \wedge \bigwedge_{b \neq a \in \Gamma} \neg S_{b,i} \big). \end{aligned}$$

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

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

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

Aus der Konstruktion folgt sofort folgende Behauptung.

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

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

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

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

und

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

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

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

Wir betrachten als nächstes die Formel

$$\text{NEXT}(\overline{C}, \overline{C}') := \text{CONF}(\overline{C}) \wedge \text{CONF}(\overline{C}') \wedge \text{NOCHANGE}(\overline{C}, \overline{C}') \wedge \text{CHANGE}(\overline{C}, \overline{C}')$$

mit

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

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

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

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

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

$$\beta \text{ erfüllt } \text{NEXT}(\overline{C}, \overline{C}') \quad \Longleftrightarrow \quad \textit{conf}(\overline{C}, \beta) \vdash_{\mathcal{M}} \textit{conf}(\overline{C}', \beta)$$

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

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

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

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

Abschließend definieren wir noch

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

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

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

$Q_{q,t}$: Für alle $q \in Q$, $0 \leqslant t \leqslant p(n)$

Bedeutung: $\mathcal{M}$ ist zur Zeit t im Zustand $q \in Q$

$P_{i,t}$: Für alle $0 \leqslant i, t \leqslant p(n)$

Bedeutung: Zur Zeit t ist der Kopf an Position i

$S_{a,i,t}$: Für alle $a \in \Sigma \dot\cup \{\square\}$ und $0 \leqslant i, t \leqslant p(n)$

Bedeutung: *Zur Zeit t enthält Bandposition i das Symbol a*

In der folgenden Formel schreiben wir kurz

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

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

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

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

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

□

3.4. Der DPLL Algorithmus

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

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

Algorithmus DPLL(Φ)

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

Ausgabe. entscheide, ob Φ erfüllbar ist.

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

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

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

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

Abbildung 3.1.: Des DPLL Algorithmus

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

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

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

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

In praktischen Implementierungen des Algorithmus' werden verschiedene Optimierungen verwendet.

Auswahlregel: Welches Literal wird beim Verzweigen gewählt.

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

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

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

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

3.5. Eine Anwendung aus der künstlichen Intelligenz

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

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

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

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

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

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

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

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

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

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

- $((D_1, D_2), R_{\neq}), \quad ((M_1, M_2), R_{\neq}), \quad ((P_1, P_2), R_{\neq})$

  „Kein Lehrer hält zwei Klassen gleichzeitig“

- Für alle $i = 1, 2$: $((M_i, D_i), R_{\neq}), \quad ((M_i, P_i), R_{\neq}), \quad ((P_i, D_i), R_{\neq})$
  "Keine Klasse hat zwei Stunden gleichzeitig"

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

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

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

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

Wir formalisieren ein gegebenes Sudoku in der Aussagenlogik wie folgt.

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

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

- $\bigwedge_{1 \leqslant i,j \leqslant 9} \bigvee_{c=1}^{9} X^c_{i,j}$
  Bedeutung: „jede Position erhält eine Zahl"
- $\bigwedge_{1 \leqslant i,j \leqslant 9} \bigwedge_{1 \leqslant c < c' \leqslant 9} \neg(X^c_{i,j} \wedge X^{c'}_{i,j})$
  Bedeutung: „jede Position erhält höchstens eine Zahl"
- Für alle $1 \leqslant i, c \leqslant 9$ und $1 \leqslant j < j' \leqslant 9$: $\quad \neg(X^c_{i,j} \wedge X^c_{i,j'})$
  Bedeutung: „Keine Zeile enthält zwei gleiche Einträge"
- Für alle $1 \leqslant j, c \leqslant 9$ und $1 \leqslant i < i' \leqslant 9$: $\quad \neg(X^c_{i,j} \wedge X^c_{i',j})$
  Bedeutung: „Keine Spalte enthält zwei gleiche Einträge"

- Die entsprechende Aussage für jeden Block: $\neg(X^c_{i,j} \wedge X^c_{i',j'})$ für alle $1 \leqslant c \leqslant 9$ und i, j, i', j' mit
  - $(i, j) \neq (i', j')$ und
  - $i \div 3 = j \div 3 = i' \div 3 = j' \div 3$.

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

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

Teil II.

Strukturen

4. Strukturen

4.1. Einleitung

Die Aussagenlogik formalisiert das Schließen über Aussagen die entweder wahr oder falsch sein können. Die eigentliche Bedeutung der Aussagen ist dabei irrelevant. Um über Aussagen der folgenden Form zu sprechen, ist die Aussagenlogik also nicht geeignet:

Für jede reelle Zahl x gibt es eine natürliche Zahl n, die größer als x ist.

Eine Logik, die zur Formalisierung derartiger Aussagen geeignet ist, muss die Möglichkeit bieten über verschiedene Arten von Objekten zu sprechen. Außerdem muss sie die Möglichkeit bieten, einige Aussagen über alle Objekte zu machen, andere Aussagen nur über einige Objekte. Wir werden im nächsten Kapitel die Prädikatenlogik einführen, in der solche Aussagen gemacht werden können.

In diesem Kapitel entwickeln wir zunächst den Begriff der mathematischen *Strukturen* und schaffen damit den Kontext, in dem prädikatenlogische Formeln ausgewertet werden. Dieser Begriff wird hinreichend allgemein sein um fast alle in der Mathematik und Informatik auftretenden Strukturen zu erfassen, z.B. algebraische und geometrische Strukturen, Datenstrukturen, Datenbanken, Transitionssysteme und viele mehr. Außerdem führen wir Abbildungen zwischen Strukturen ein, die (in unterschiedlichem Maße) *strukturerhaltend* sind.

4.2. Relationen

Ein fundamentaler Grundbegriff ist der Begriff der Relation.

Definition 4.1 Sei $k \geqslant 0$ und A eine Menge.

(1) A^k ist die Menge aller k-Tupel von Elementen aus A.

(2) Eine *k-stellige Relation auf A* ist eine Teilmenge von A^k. ⊣

Bemerkung 4.2 Wir erlauben auch $k = 0$. Es gibt genau zwei *nullstellige* Relationen; $R \subseteq A^0$ ist entweder $\varnothing$ oder $\{()\}$. ⊣

Notation 4.3 Für einige spezielle Relationen wie z.B. $<, =$, benutzen wir *Infix*-Notation und schreiben $a = b$ oder $a < b$ anstatt $(a, b) \in =$ oder $(a, b) \in <$. ⊣

Als nächstes führen wir einige wichtige Eigenschaften von Relationen ein.

Definition 4.4 Eine binäre Relation $R \subseteq A^2$ einer Menge A ist

- *reflexiv*, wenn $(a, a) \in R$, für alle $a \in A$.
- *symmetrisch*, wenn aus $(a, b) \in R$ immer $(b, a) \in R$ folgt, für alle $a, b \in A$.
- *antisymmetrisch*, wenn $(a, b) \in R$ und $(b, a) \in R$ zusammen $a = b$ impliziert, für alle $a, b \in A$.
- *transitiv*, wenn aus $(a, b) \in R$ und $(b, c) \in R$ immer $(a, c) \in R$ folgt, für alle $a, b, c \in A$. ⊣

Definition 4.5 Eine *Äquivalenzrelation* ist eine binäre Relation, die *reflexiv*, *transitiv* und *symmetrisch* ist. ⊣

Beispiel 4.6 Wir betrachten einige Beispiele für Äquivalenzrelationen.

- *Gleichheit.* Für jede Menge A ist die Relation
$$\{(a, a) \in A^2 : a \in A\}$$
eine Äquivalenzrelation.
- *Gleichmächtigkeit.* Für jede Menge A ist die Relation
$$\{(A, B) \in \mathcal{P}(A)^2 : A, B \text{ haben die gleiche Kardinalität }\}$$
eine Äquivalenzrelation.
- *Logische Äquivalenz* ist die Relation
$$\{(\varphi, \psi) \in \mathrm{AL}^2 : \varphi \equiv \psi\}$$
eine Äquivalenzrelation. ⊣

Definition 4.7 Sei A eine Menge.

(1) Eine *(strikte) partielle Ordnung* $<$ über einer Menge A ist eine irreflexive und transitive binäre Relation über A.

(2) Eine *(strikte) lineare Ordung* $<$ über A ist eine partielle Ordung über A, so dass für alle $a, b \in A$:

$$a < b, \quad a = b \quad \text{oder} \quad b < a \tag{4.1}$$

Bemerkung 4.8

- Sei $<$ eine partielle Ordnung über einer Menge A. Dann impliziert Bedingung (1) aus Definition 4.7, dass $<$ antisymmetrisch ist.
- Bisweilen ist es nützlich, reflexive Ordungen zu betrachten. Wir definieren $\leqslant$ als $< \cup \{(a, a) \in A^2 : a \in A\}$. Das heißt, eine reflexive lineare Ordung ist eine reflexive, antisymmetrische, transitive und binäre Relation für die (4.1) aus Definition 4.7 gilt. ⊣

4.3. Strukturen

In diesem Abschnitt führen wir den allgemeinen Strukturenbegriff ein. Mathematische Strukturen bestehen aus einem Universum und aus ausgezeichneten Funktionen, Relationen und Konstanten auf diesem Universum. Die Namen für die in einer Struktur auftretenden Relationen und Funktionen bilden die Signatur der Struktur.

Definition 4.9 Eine *Signatur* ist eine Menge σ von *Relationssymbolen*, *Funktionssymbolen* und *Konstantensymbolen*. Jedes Relationssymbol $R \in \sigma$ und jedes Funktionssymbol $f \in \sigma$ hat eine *Stelligkeit* (englisch: arity)

$$ar(R) \in \mathbb{N} \text{ bzw. } ar(f) \in \mathbb{N}. \qquad \dashv$$

Notation 4.10

- Wir verwenden griechische Symbole σ, τ für Signaturen.
- Für Relationssymbole verwenden wir $R, P, Q, R', <, \leqslant, \ldots$
- Für Funktionssymbole verwenden wir $f, g, h, +, *, \ldots$
- Für Konstantensymbole verwenden wir $c, d, 0, 1, \ldots$ $\dashv$

Die Relations-, Funktions- und Konstantensymbole der Signatur können in beliebiger Weise durch konkrete Relationen, Funktionen und Konstanten interpretiert werden. Allgemein wird die Struktur festgelegt durch Angabe ihres Universums und der konkreten Interpretation der Relations-, Funktions- und Konstantensymbole über dem Universum.

Definition 4.11 Sei σ eine Signatur. Eine *σ-Struktur* $\mathcal{A}$ besteht aus

- einer nicht-leeren Menge A, dem *Universum* von $\mathcal{A}$,
- einer k-stelligen Relation $R^{\mathcal{A}} \subseteq A^k$ für jedes k-stellige Relationssymbol $R \in \sigma$,
- einer k-stelligen Funktion $f^{\mathcal{A}} : A^k \to A$ für jedes k-stelliges Funktionssymbol $f \in \sigma$ und
- einem Element $c^{\mathcal{A}} \in A$ für jedes Konstantensymbol $c \in \sigma$. $\dashv$

Bemerkung 4.12 Man beachte den Unterschied zwischen einem Symbol

$$R \in \sigma \qquad \text{oder} \qquad f \in \sigma$$

und seiner *Interpretation*

$$R^{\mathcal{A}} \qquad \text{bzw.} \qquad f^{\mathcal{A}}$$

in einer σ-Struktur $\mathcal{A}$. $\dashv$

Notation 4.13 Wir verwenden kalligraphische Buchstaben $\mathcal{A}, \mathcal{B}, ...$ für Strukturen und entsprechende lateinische Buchstaben $A, B \ldots$ für deren Universen.

Wir schreiben σ-Strukturen oft als Tupel

$$\mathcal{A} := \big(A, (R^{\mathcal{A}})_{R\in\sigma}, (f^{\mathcal{A}})_{f\in\sigma}, (c^{\mathcal{A}})_{c\in\sigma}\big)$$

oder, falls $\sigma := \{R_1, \ldots, R_k, f_1, \ldots, f_l, c_1, \ldots, c_m\}$ endlich ist, auch als

$$\mathcal{A} := \big(A, R_1^{\mathcal{A}}, \ldots, R_k^{\mathcal{A}}, f_1^{\mathcal{A}}, \ldots, f_l^{\mathcal{A}}, c_1^{\mathcal{A}}, \ldots, c_m^{\mathcal{A}}\big). \qquad \dashv$$

In der Literatur werden Strukturen oft mit deutschen Buchstaben $\mathfrak{A}$, $\mathfrak{B}$, $\mathfrak{C}$,... bezeichnet. Wir geben als nächstes einige Beispiele für häufig benutzte Strukturen.

Beispiel 4.14 Sei $\sigma_{ar} := \{+, *, 0, 1\}$ die Signatur der Arithmetik, wobei

- $+, *$ binäre Funktionsymbole und
- $0, 1$ Konstantensymbole sind.

Wir können σ_{ar}-Strukturen zur Modellierung von Körpern, etc. benutzen.

(1) Wir definieren eine σ_{ar}-Struktur $\mathcal{N} := (\mathbb{N}, +^{\mathcal{N}}, *^{\mathcal{N}}, 0^{\mathcal{N}}, 1^{\mathcal{N}})$ mit Universum $\mathbb{N}$, wobei
   - $+^{\mathcal{N}}$ und $*^{\mathcal{N}}$ die Addition und Multiplikation der natürlichen Zahlen sind und
   - $0^{\mathcal{N}} := 0$ und $1^{\mathcal{N}} := 1$.

   Die Struktur $\mathcal{N}$ entspricht also der Arithmetik.

(2) Eine andere σ_{ar}-Struktur ist $\mathcal{Z} := (\mathbb{Z}, +^{\mathcal{Z}}, *^{\mathcal{Z}}, 0^{\mathcal{Z}}, 1^{\mathcal{Z}})$ mit Universum $\mathbb{Z}$ und
   - $+^{\mathcal{Z}}$ und $*^{\mathcal{Z}}$ als Addition und Multiplikation der ganzen Zahlen und
   - $0^{\mathcal{Z}} := 0$ und $1^{\mathcal{Z}} := 1$. $\dashv$

Bemerkung 4.15 σ_{ar}-Strukturen müssen nicht „natürliche“ arithmetische Strukturen wie die rellen oder ganzen Zahlen sein. Wir können genauso eine σ_{ar}-Struktur

$$\mathcal{A} := (\mathbb{N}, +^{\mathcal{A}}, *^{\mathcal{A}}, 0^{\mathcal{A}}, 1^{\mathcal{A}})$$

mit Universum $\mathbb{N}$ definieren, wobei

- $+^{\mathcal{A}}(a, b) := a^2 + b^2$,
- $*^{\mathcal{A}}$ ist die übliche *Addition* der natürlichen Zahlen und
- $0^{\mathcal{A}} := 17$ sowie $1^{\mathcal{A}} := 0$. $\dashv$

Ein weiteres Beispiel, das wir in den folgenden Kapiteln oft benutzen werden, sind *Graphen*.

Definition 4.16 Ein *gerichteter Graph* $\mathcal{G}$ ist eine Struktur über der Signatur $\{E\}$, wobei E ein zweistelliges Relationssymbol ist welches durch eine binäre und irreflexive Relation über dem Universum interpretiert wird. ⊣

Wir werden Graphen meistens als ein Paar $\mathcal{G} := (V, E^{\mathcal{G}})$ notieren, wobei

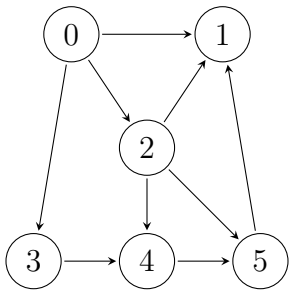
- das Universum V als Knotenmenge und
- die Relation $E^{\mathcal{G}} \subseteq V^2$ als Kantenmenge bezeichnet wird.

Für Graphen als Struktur $\mathcal{G}$ weichen wir also von der Konvention ab, das Universum mit G zu bezeichnen.

Beispiel 4.17 Sei $\mathcal{G} := (V, E^{\mathcal{G}})$ mit

$$V := \{0, 1, 2, 3, 4, 5\}$$
$$E^{\mathcal{G}} := \{(0,1), (0,2), (0,3), (2,1), (2,4), (2,5), (3,4), (4,5), (5,1)\}.$$

Wir werden Graphen wie üblich graphisch darstellen.

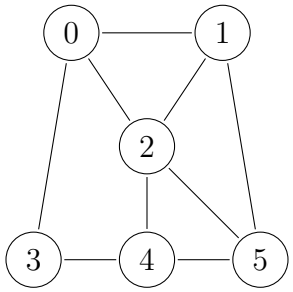


⊣

Definition 4.18 Ein *ungerichteter Graph*, oder einfach *Graph*, ist ein Paar $\mathcal{G} := (V, E^{\mathcal{G}})$ mit den folgenden Eigenschaften.

- $(V, E^{\mathcal{G}})$ ist ein gerichteter Graph.
- $E^{\mathcal{G}}$ ist symmetrisch. ⊣

Da die Kantenrelation in ungerichteten Graphen symmetrisch ist, wird die Richtung der Kanten in ungerichteten Graphen nicht gezeichnet:



Mit Hilfe von Graphen können viele Probleme formalisiert werden. Sie sind natürliche Modelle für Straßenkarten, Flugverbindungen oder Computernetzwerke. Auch elektrische Schaltkreise werden oft als Graphen modelliert, dort repräsentieren Knoten Komponenten wie z.B. Dioden, Transistoren oder Widerstände und Kanten repräsentieren die Drähte.

Definition 4.19 Sei $\mathcal{G} := (V, E^{\mathcal{G}})$ ein gerichteter Graph.

(1) Ein *Weg* in $\mathcal{G}$ ist ein Tupel $(v_0, \dots, v_l) \in V^{l+1}$, für ein $l \in \mathbb{N}$, so dass

$$(v_{i-1}, v_i) \in E^{\mathcal{G}} \qquad \text{für alle } 1 \leqslant i \leqslant l.$$

Wir nennen $(v_0, \dots, v_l) \in V^{l+1}$ einen *Weg von v_0 zu v_l* und l die *Länge* des Wegs.

Bemerkung. Das Tupel (v) ist ein Weg der Länge 0, für alle $v \in V$.

(2) Ein *Pfad* in $\mathcal{G}$ ist ein Weg $(v_0, \dots, v_l) \in V^{l+1}$ so dass $v_i \neq v_j$ für alle $0 \leqslant i < j \leqslant l$.

(3) Ein *geschlossener Weg* in $\mathcal{G}$ ist ein Weg $(v_0, \dots, v_l) \in V^{l+1}$ mit $v_0 = v_l$.

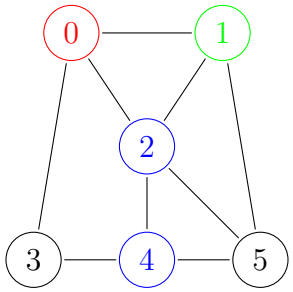
(4) Ein *Kreis* in $\mathcal{G}$ ist ein Pfad $(v_0, \dots, v_l) \in V^{l+1}$ mit $(v_l, v_0) \in E^{\mathcal{G}}$. ⊣

Wir betrachten oft auch *gefärbte Graphen*, wobei jeder Knoten mit einer Farbe aus einer festen Menge $\mathcal{C}$ von Farben gefärbt sein kann. Sei $\mathcal{C}$ eine endliche Menge von einstelligen Relationssymbolen und sei $\sigma := \{E\} \cup \mathcal{C}$, wobei wir annehmen, dass $E \notin \mathcal{C}$. Wir modellieren $\mathcal{C}$-gefärbte Graphen als Strukturen

$$\mathcal{G} := (V, E^{\mathcal{G}}, (C^{\mathcal{G}})_{C \in \mathcal{C}})$$

wobei $C^{\mathcal{G}}$ alle Knoten mit Farbe C enthält.

Beispiel 4.20 Sei $\mathcal{C}$:= {Rot, Grün, Blau}. Im gezeichneten Graphen gilt $\text{Rot}^{\mathcal{G}} = \{0\}$, $\text{Grün}^{\mathcal{G}} = \{1\}$ und $\text{Blau}^{\mathcal{G}} = \{2, 4\}$.



⊣

4.4. Substrukturen

Definition 4.21 Sei τ eine Signatur und seien $\mathcal{A}, \mathcal{B}$ τ-Strukturen.

(1) $\mathcal{A}$ ist eine *Substruktur* von $\mathcal{B}$, geschrieben als $\mathcal{A} \subseteq \mathcal{B}$, wenn $A \subseteq B$ und

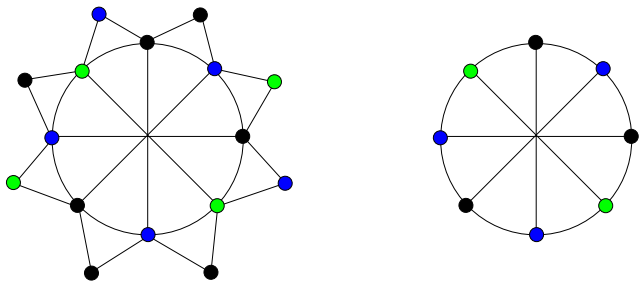
- für alle k-stelligen Relationssymbole $R \in \tau$ und alle $\overline{a} \in A^k$,

$$\overline{a} \in R^{\mathcal{A}} \quad \text{genau dann, wenn} \quad \overline{a} \in R^{\mathcal{B}},$$

- für alle k-stelligen Funktionssymbole $f \in \tau$ und alle $\overline{a} \in A^k$, $f^{\mathcal{A}}(\overline{a}) = f^{\mathcal{B}}(\overline{a})$ und
- für alle Konstantensymbole $c \in \tau$, $c^{\mathcal{A}} = c^{\mathcal{B}}$.

(2) Wenn $\mathcal{A}$ eine Substruktur von $\mathcal{B}$ ist, dann ist $\mathcal{B}$ eine *Erweiterung* von $\mathcal{A}$. ⊣

Beispiel 4.22 Die folgende Abbildung zeigt eine $\sigma := \{E, \text{Blau}, \text{Grün}\}$-Struktur (a) und eine Substruktur (b).



a) b)

⊣

Definition 4.23 Sei τ eine Signatur und $\mathcal{B}$ eine τ-Struktur mit Universum B. Eine Menge $A \subseteq B$ heißt *τ-abgeschlossen* in $\mathcal{B}$, wenn folgende Bedingungen gelten.

(1) Für alle Konstantensymbole $c \in \tau$ ist $c^{\mathcal{B}} \in A$.

(2) Wenn $f \in \tau$ ein k-stelliges Funktionssymbol und $\overline{a} \in A^k$ ein k-Tupel von Elementen ist, so ist $f(\overline{a}) \in A$. ⊣

Bemerkung 4.24 Wenn $\mathcal{A} \subseteq \mathcal{B}$, dann ist A *τ-abgeschlossen*. Umgekehrt gibt es für jede τ-abgeschossene Menge $A \subseteq B$ genau eine induzierte Substruktur $\mathcal{A} \subseteq \mathcal{B}$ mit Universum A. Diese wird die *durch A induzierte Substruktur* genannt. ⊣

Beispiel 4.25 Sei $\mathcal{Z} := (\mathbb{Z}, <^{\mathcal{Z}}, +^{\mathcal{Z}})$, wobei $<^{\mathcal{Z}}$ die natürliche Ordnung auf $\mathbb{Z}$ und $+^{\mathcal{Z}}$ die Addition auf den ganzen Zahlen ist. Dann ist $\mathcal{N} := (\mathbb{N}, <^{\mathcal{N}}, +^{\mathcal{N}})$, wobei $<^{\mathcal{N}}$ die natürliche Ordnung auf $\mathbb{N}$ und $+^{\mathcal{N}}$ die übliche Addition auf den natürlichen Zahlen ist, die durch $\mathbb{N}$ induzierte Substruktur von $\mathcal{Z}$. ⊣

Substrukturen sind *eine* Art, in der eine Struktur in einer anderen enthalten sein kann. Hier haben wir ein kleineres Universum, aber die gleiche Signatur. Eine andere Art, in der $\mathcal{A}$ in $\mathcal{B}$ enthalten sein kann, ist indem weniger Symbole zur Verfügung stehen, aber das Universum gleich bleibt.

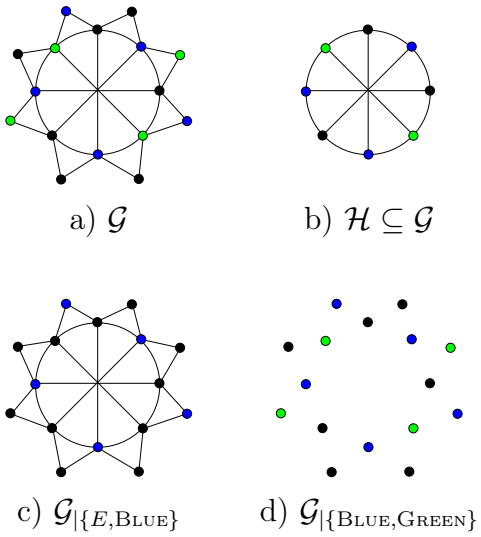
Definition 4.26 Sei $\sigma \subseteq \tau$ eine Signatur und sei $\mathcal{B}$ eine τ-Struktur.

Das σ-*Redukt* $\mathcal{B}_{|\sigma}$ von $\mathcal{B}$ ist definiert als die σ-Struktur $\mathcal{B}_{|\sigma}$ die man aus $\mathcal{B}$ enthält, indem die Symbole aus $\tau \setminus \sigma$ „entfernt“ werden, d.h. die Struktur mit

- Universum B und
- $S^{\mathcal{B}_{|\sigma}} = S^{\mathcal{B}}$ für jedes (Relations-, Funktions-, Konstanten-) Symbol $S \in \sigma$.

$\mathcal{B}$ heißt *Expansion* von $\mathcal{B}_{|\sigma}$. ⊣

Beispiel 4.27 (Fortsetzung von Beispiel 4.22.) Die folgende Abbildung zeigt eine $\sigma := \{E, \textsc{Blau}, \textsc{Grün}\}$-Struktur (a), eine Substruktur (b) sowie zwei verschiedene Redukte (c) und (d).



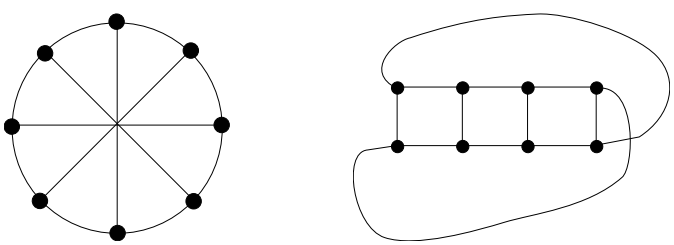
a) $\mathcal{G}$ b) $\mathcal{H} \subseteq \mathcal{G}$

c) $\mathcal{G}_{|\{E,\textsc{Blue}\}}$ d) $\mathcal{G}_{|\{\textsc{Blue},\textsc{Green}\}}$

⊣

4.5. Homo- und Isomorphismen

In diesem Abschnitt betrachten wir strukturerhaltende Abbildungen. Betrachten wir dazu zunächst einmal die folgenden Graphen?



Frage. *Sind die folgenden zwei Graphen verschieden?*

Mögliche Antworten.

Ja, wenn wir daran interessiert sind, wie sie gezeichnet sind.

Nein, wenn wir uns nur für ihre Knoten und Verbindungen dazwischen interessieren.

Als Strukturen $\mathcal{G}_1, \mathcal{G}_2$ über $\sigma_{Graph} := \{E\}$ sind sie gleich (im Sinne der Isomorphie, siehe weiter unten). Wenn wir uns für ihre Einbettung in den $\mathbb{R}^2$ (also eine Zeichnung wie oben) interessieren, brauchen wir eine andere Modellierung als Struktur, die diese Informationen enthält.

Wir betrachten im Folgenden zwei verschiedene Begriffe der Ähnlichkeit zwischen Strukturen. Der erste Begriff ist relativ schwach, wohingegen der zweite Begriff die "Gleichheit" zwischen Strukturen vollständig beschreibt.

Definition 4.28 Seien $\mathcal{A}, \mathcal{B}$ zwei σ-Strukturen. Ein *Homomorphismus* von $\mathcal{A}$ in $\mathcal{B}$ ist eine Funktion $h : A \to B$, so dass

- für alle k-stelligen Relationssymbole $R \in \sigma$ und alle $\overline{a} := a_1, \ldots, a_k \in A^k$:

$$\text{wenn} \quad \overline{a} \in R^{\mathcal{A}} \qquad \text{dann auch} \qquad \big(h(a_1), \ldots, h(a_k)\big) \in R^{\mathcal{B}},$$

- für alle k-stelligen Funktionssymbole $f \in \sigma$ und alle $\overline{a} := a_1, \ldots, a_k \in A^k$ gilt

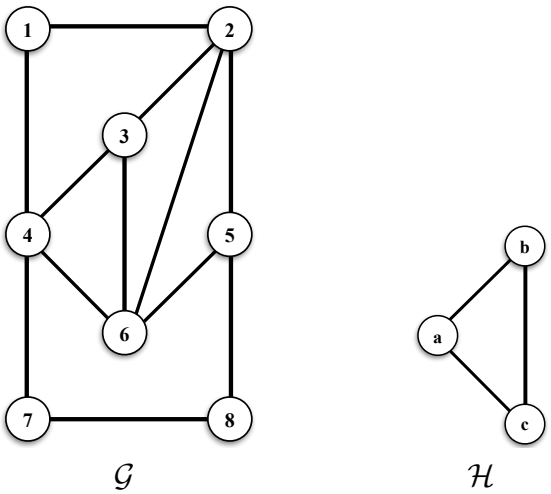
$$h(f^{\mathcal{A}}(\overline{a})) = f^{\mathcal{B}}(h(a_1), \ldots, h(a_k)\big) \quad \text{und}$$

- für alle Konstantensymbole $c \in \sigma$ gilt

$$h(c^{\mathcal{A}}) = c^{\mathcal{B}}.$$

Wir schreiben $h : \mathcal{A} \to_{hom} \mathcal{B}$ um zu sagen, dass h ein Homomorphismus von $\mathcal{A}$ nach $\mathcal{B}$ ist. ⊣

Beispiel 4.29 Wir betrachten die folgenden Graphen $\mathcal{G} := (G, E^{\mathcal{G}})$ und $\mathcal{H} := (H, E^{\mathcal{H}})$.



$\mathcal{G}$ $\qquad$ $\mathcal{H}$

Dann gilt $\mathcal{G} \to_{hom} \mathcal{H}$ und $\mathcal{H} \to_{hom} \mathcal{G}$. Zum Beispiel ist die Abbildung $h : G \to H$, die die Elemente $2, 4, 8$ auf das Element a, die Elemente $1, 3, 5, 7$ auf das Element b sowie das Element 6 auf c abbildet, ein Homomorphismus von $\mathcal{G}$ nach $\mathcal{H}$. Es gibt aber viele andere Homomorphismen von $\mathcal{G}$ nach $\mathcal{H}$. ⊣

Wie das vorherige Beispiel zeigt, können homomorphe Strukturen recht unterschiedlich "aussehen". Wir werden daher jetzt einen erheblich stärkeren Ähnlichkeitsbegriff zwischen Strukturen kennen lernen.

Definition 4.30 Seien $\mathcal{A}, \mathcal{B}$ zwei σ-Strukturen. Ein *Isomorphismus* von $\mathcal{A}$ in $\mathcal{B}$ ist eine Funktion $I : A \to B$, so dass

- I eine Bijektion zwischen A und B ist
- für alle k-stell. Relationssymb. $R \in \sigma$ und alle $\overline{a} := a_1, \ldots, a_k \in A^k$:

$$\overline{a} \in R^{\mathcal{A}} \qquad \text{genau dann, wenn} \qquad \big(I(a_1), \ldots, I(a_k)\big) \in R^{\mathcal{B}}.$$

- für alle k-stell. Funktionssymb. $f \in \sigma$ und alle $\overline{a} := a_1, \ldots, a_k \in A^k$ gilt

$$I(f^{\mathcal{A}}(\overline{a})) = f^{\mathcal{B}}(I(a_1), \ldots, I(a_k)\big).$$

- für alle Konstantensymbole $c \in \sigma$ gilt

$$I(c^{\mathcal{A}}) = c^{\mathcal{B}}.$$

Wir schreiben $I : \mathcal{A} \cong \mathcal{B}$ um zu sagen, dass I ein Isomorphismus von $\mathcal{A}$ nach $\mathcal{B}$ ist. ⊣

Definition 4.31 Sei σ eine Signatur.

(1) Zwei σ-Strukturen $\mathcal{A}, \mathcal{B}$ sind *isomorph*, geschrieben $\mathcal{A} \cong \mathcal{B}$, wenn es einen Isomorphismus zwischen $\mathcal{A}$ und $\mathcal{B}$ gibt.

(2) Zwei σ-Strukturen $\mathcal{A}, \mathcal{B}$ sind *homomorph*, geschrieben $\mathcal{A} \to_{hom} \mathcal{B}$, wenn es einen Homomorphismus von $\mathcal{A}$ nach $\mathcal{B}$ gibt. ⊣

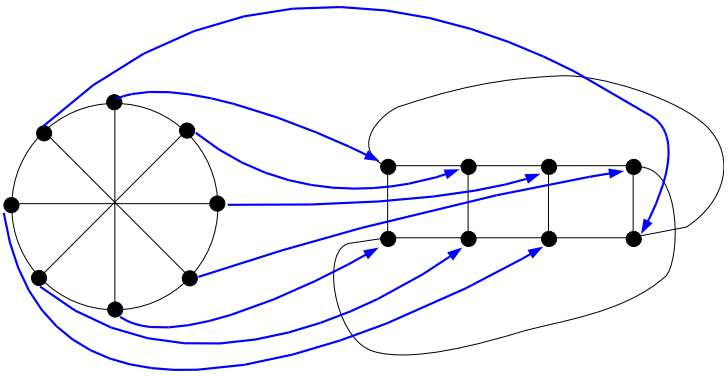
Man beachte, dass der Isomorphiebegriff symmetrisch ist, der Begriff eines Homomorphismus jedoch nicht.

Beispiel 4.32

- Wenn A, B endliche Mengen der gleichen Kardinalität sind, dann gilt für die $\varnothing$-Strukturen $(A, \varnothing) \cong (B, \varnothing)$.
- Wenn A, B endliche Mengen gleicher Kardinalität und $<^{\mathcal{A}}, <^{\mathcal{B}}$ lineare Ordnungen auf A, B sind, dann $(A, <^{\mathcal{A}}) \cong (B, <^{\mathcal{B}})$.

  Aber: $(\mathbb{Z}, <) \ncong (\mathbb{N}, <)$

- Als letztes Beispiel zeigen wir noch, dass die beiden Graphen zu Beginn des Abschnitts in der Tat isomorph sind. Ein möglicher Isomorphismus ist in folgender Abbildung dargestellt.



⊣

4.6. Constraint Satisfaction Probleme

Zum Abschluss dieses Kapitels betrachten wir noch einmal die Constraint Satisfaction Probleme aus Definition 3.29. Wir wiederholen die Definition zur Erinnerung:

Ein Constraint Satisfaction Problem $\mathcal{C} := (V, D, C)$ besteht aus

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

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

Constraint Satisfaction Probleme können äquivalent als Homomorphieproblem interpretiert. Zu einem CSP $\mathcal{C} := (V, D, C)$ wie zuvor definieren wir ein Homomorphieproblem über einer Signatur σ_C wie folgt.

- Für jedes constraint $c := ((v_1, \ldots, v_n), R)$ enthält die Signatur σ_C ein n-stelliges Relationssymbol R_c.
- Wir definieren weiterhin eine σ-Struktur $\mathcal{A}$ mit Universum $A := V$ und $R_c^{\mathcal{A}} := \{(v_1, \ldots, v_n)\}$.
- Schließlich definieren wir eine σ-Struktur $\mathcal{B}$ mit Universum D und $R_c^{\mathcal{B}} := R$, für $c = ((v_1, \ldots, v_n), R)$.

Nun gilt: $\mathcal{A} \to_{hom} \mathcal{B}$ genau dann, wenn das CSP $\mathcal{C}$ lösbar ist.

Umgekehrt lässt sich das Homomorphieproblem zwischen zwei Strukturen sehr leicht als CSP schreiben. CSPs und Homomorphieprobleme sind also nur unterschiedliche Sichtweisen auf das selbe Problem.

CSPs sind ein in der künstlichen Intelligenz viel untersuchtes Problem. Das CSP-Problem, also das Problem zu entscheiden, ob ein gegebenes CSP eine Lösung hat, ist NP-vollständig. Wie wir gesehen haben, kann das CSP-Problem, äquivalent auch als Homomorphieproblem angesehen werden, d.h. als Problem für zwei gegebene Strukturen $\mathcal{A}, \mathcal{B}$ zu entscheiden, ob $\mathcal{A} \to_{hom} \mathcal{B}$.

In der aktuellen Forschung wird zur Zeit die Komplexität des Homomorphieproblems für feste Strukturen $\mathcal{B}$ intensiv untersucht. Zum Beispiel können wir sehr leicht das 3-Färbbarkeitsproblem als Homomorphieproblem für die Struktur $\mathcal{B}$ schreiben, die nur aus einem Dreieck besteht. Für dieses $\mathcal{B}$ ist das Homomorphieproblem also NP-vollständig. Für andere Strukturen $\mathcal{B}$ ist das Problem jedoch in PTime.

Vermutung 4.33 (Feder, Vardi) *Für jede feste Struktur $\mathcal{B}$ ist das Homomorphieproblem entweder NP-vollständig oder in* PTime.

Teil III.

Prädikatenlogik

5. Grundlagen der Prädikatenlogik

In diesem Kapitel führen wir die *Prädikatenlogik*, abgekürzt FO für „first-order logic“, ein. Die Prädikatenlogik macht Aussagen über Strukturen und deren Elemente. Insbesondere also sind Formeln nicht einfach nur wahr oder falsch. Wir können in der Prädikatenlogik viele der üblichen Aussagen der Mathematik formalisieren, aber ebenso gut gängige Anfragen an Datenbanken und ähnliche Dinge der Informatik beschreiben.

5.1. Syntax der Prädikatenlogik

Wir führen zunächst die Syntax der Prädikatenlogik ein. Dazu brauchen wir zunächst den Begriff der Variablen sowie der Terme.

Definition 5.1 (Variablen und Terme) Wir fixieren die Menge $\textsc{Var} := \{v_i : i \in \mathbb{N}\}$ von *Variablen erster Stufe*, kurz *Variablen*. wir mit VAR.

Sei σ eine Signatur. Die Menge $\mathcal{T}_\sigma$ der σ-Terme ist induktiv wie folgt definiert.

Basisfall

- $v_i \in \mathcal{T}_\sigma$ für alle $v_i \in \textsc{Var}$.
- $c \in \mathcal{T}_\sigma$ für alle Konstantensymbole $c \in \sigma$.

Induktionsschritt

- Ist $f \in \sigma$ ein k-stelliges Funktionssymbol und $t_1, \ldots, t_k \in \mathcal{T}_\sigma$ dann ist

$$f(t_1, \ldots, t_k) \in \mathcal{T}_\sigma\,.$$

Ein Term, in dem keine Variablen vorkommen, heißt *Grundterm*. ⊣

Definition 5.2 (Syntax der Prädikatenlogik) Sei σ eine Signatur. Die Menge $\textsc{fo}[\sigma]$ der *prädikatenlogischen Formeln über* σ ist induktiv wie folgt definiert.

Basisfall

- $t = t' \in \textsc{fo}[\sigma]$ für alle Terme $t, t' \in \mathcal{T}_\sigma$.
- $R(t_1, \ldots, t_k) \in \textsc{fo}[\sigma]$, für alle k-stelligen Relationssymbole $R \in \sigma$ und alle $t_1, \ldots, t_k \in \mathcal{T}_\sigma$.

Formeln der Form $t = t'$ und $R(t_1, \ldots, t_k)$ heißen *atomar*.

Induktionsschritt

- Wenn $\varphi \in \text{FO}[\sigma]$, dann $\neg\varphi \in \text{FO}[\sigma]$.
- Wenn $\varphi, \psi \in \text{FO}[\sigma]$, dann ist $(\varphi \vee \psi) \in \text{FO}[\sigma]$, $(\varphi \wedge \psi) \in \text{FO}[\sigma]$, $(\varphi \to \psi) \in \text{FO}[\sigma]$ und $(\varphi \leftrightarrow \psi) \in \text{FO}[\sigma]$.
- Wenn $\varphi \in \text{FO}[\sigma]$ und $x \in \text{VAR}$, dann ist $\exists x\varphi \in \text{FO}[\sigma]$ und $\forall x\varphi \in \text{FO}[\sigma]$.

$\text{FO}[\sigma]$ heißt die *Prädikatenlogik über* σ oder die *Sprache/Logik erster Stufe über* σ.⊣

Bemerkung 5.3 Man beachte, dass die "Gleichheit" immer in der Prädikatenlogik verwendet werden darf. Wir können also immer Formeln der Art $v_0 = v_1$ schreiben. Aus diesem Grund vereinbaren wir, dass das Symbol $=$ nicht in der Signatur vorkommt. ⊣

Wir betrachten einige Beispiele.

Beispiel 5.4 Sei $\sigma_{Graph} := \{E\}$, wobei E eine binäre Relation ist. Die folgenden Ausdrücke sind Formeln in $\text{FO}[\sigma_{Graph}]$, der Sprache der Graphen.

(1) $E(v_0, v_1)$

(2) $\exists v_0 \forall v_1 E(v_0, v_1)$

(3) $\exists v_1 \exists v_2 \big((E(v_0, v_1) \wedge E(v_1, v_2)) \wedge E(v_2, v_3)\big)$

(4) $\exists v_1 \big(E(v_0, v_1) \wedge \exists v_0 (E(v_1, v_0) \wedge E(v_0, v_3))\big)$

(5) $\exists v_0 E(v_0, v_0) \wedge \exists v_1 E(v_1, v_0)$

(6) $\exists v_0 E(v_1, v_0) \wedge \exists v_0 E(v_2, v_0)$

Das letzte Beispiel zeigt, dass Variablen auch mehrfach quantifiziert werden dürfen.⊣

Notation 5.5 Um die Lesbarkeit von Formeln zu erhöhen, vereinbaren wir folgende Notation.

(1) Wie schon in der Aussagenlogik werden wir auch Variablen $x, y, z, ...$ verwenden, auch wenn die streng genommen nicht in der Variablenmenge VAR vorkommen.

(2) Wir vereinbaren die gleichen Klammerregeln wie in der Aussagenlogik, schreiben also $(\varphi_1 \wedge \varphi_2 \wedge \varphi_3)$ statt $((\varphi_1 \wedge \varphi_2) \wedge \varphi_3)$. Hierbei ist aber Vorsicht geboten. Die Formeln $\exists x E(x, x) \vee \exists z E(x, z)$ und $\exists x (E(x, x) \vee \exists z E(x, z))$ haben eine komplett andere Bedeutung. Wir können also in der zweiten Formel nicht einfach die Klammern um die Disjunktion weglassen.

(3) Relationssymbole $<, \leqslant, >, \geqslant, ...$ und Funktionssymbole $+, *, ...$ werden wir oft in Infixnotation schreiben. D.h. wir verwenden Formeln der Form $x < y + z$ statt korrekt zu schreiben: $<(x, +(y, z))$. ⊣

Beispiel 5.6 Sei $\sigma := \{<, +, *\}$, wobei $<$ eine binäre Relation und $+$ und $*$ binäre Funktionen sind. Die folgenden Ausdrücke sind Formeln in $\text{FO}[\sigma]$, der Sprache der Arithmetik mit Ordnung. Zur leichteren Lesbarkeit verwenden wir $<$ und $+$ in Infixnotation, auch wenn das streng genommen keine prädikatenlogischen Formeln sind.

- $x < x + x$
- $\forall x \exists y\ x < y$
- $\exists x \neg \exists y\ y < x$ ⊣

Definition 5.7 Sei σ eine Signatur. Wir schreiben $\text{var}(t)$ für die in einem σ-Term t vorkommenden Variablen. Formal wird $\text{var}(t)$ wie folgt induktiv definiert:

- Wenn $t = v_i \in \text{VAR}$ dann $\text{var}(t) := \{v_i\}$.
- Wenn $t = c$ für ein Konstantensymbol $c \in \sigma$ dann $\text{var}(t) := \varnothing$.
- Wenn $t = f(t_1, \ldots, t_k)$ für ein k-stelliges Funktionssymbol $f \in \sigma$ und Terme $t_1, \ldots, t_k \in \mathcal{T}_\sigma$ dann $\text{var}(t) := \text{var}(t_1) \cup \cdots \cup \text{var}(t_k)$. ⊣

Definition 5.8 (Freie und gebundene Variablen) Sei σ eine Signatur. Die Menge $\mathit{frei}(\varphi)$ der *freien Variablen* einer Formel $\varphi \in \text{FO}[\sigma]$ ist induktiv definiert durch:

- Für $t_1, t_2 \in \mathcal{T}_\sigma$ ist $\mathit{frei}(t_1 = t_2) := \text{var}(t_1) \cup \text{var}(t_2)$.
- Wenn $\varphi = R(t_1, \ldots, t_k)$ für $R \in \sigma$ und $t_1, \ldots, t_k \in \mathcal{T}_\sigma$, dann $\mathit{frei}(\varphi) := \bigcup_{i=1}^{k} \text{var}(t_i)$.
- $\mathit{frei}(\neg\varphi) := \mathit{frei}(\varphi)$ für alle $\varphi \in \text{FO}[\sigma]$.
- $\mathit{frei}\big((\varphi * \psi)\big) := \mathit{frei}(\varphi) \cup \mathit{frei}(\psi)$ für alle $* \in \{\vee, \wedge, \to, \leftrightarrow\}$ und $\varphi, \psi \in \text{FO}[\sigma]$.
- Wenn $\varphi = \exists x\psi$ oder $\varphi = \forall x\psi$, für $x \in \text{VAR}$ and $\psi \in \text{FO}[\sigma]$, dann $\mathit{frei}(\varphi) := \mathit{frei}(\psi) \setminus \{x\}$.

Eine Formel φ mit $\mathit{frei}(\varphi) = \varnothing$ heißt ein *Satz*. Eine Variable, die in φ vorkommt, aber nicht frei ist, heißt *gebunden*. Wir schreiben $\varphi(v_1, \ldots, v_k)$, um zu sagen, dass $\mathit{frei}(\varphi) \subseteq \{v_1, \ldots, v_k\}$. ⊣

Beispiel 5.9 Betrachten wir noch einmal die Formeln aus den Beispielen 5.4 und 5.6.

- Für $\varphi = E(x, y)$ gilt $\mathit{frei}(\varphi) = \{x, y\}$.
- Für $\varphi = \exists x \forall y (x = y \vee E(x, y))$ gilt $\mathit{frei}(\varphi) = \varnothing$.
- Für $\varphi = \exists x_1 \exists x_2 \big((E(x, x_1) \wedge E(x_1, x_2)) \wedge E(x_2, y)\big)$ gilt $\mathit{frei}(\varphi) = \{x, y\}$.
- Für $\varphi = x < x + x$ gilt $\mathit{frei}(\varphi) = \{x\}$.
- Für $\varphi = \forall x \exists y\ x < y$ gilt $\mathit{frei}(\varphi) = \varnothing$.
- Für $\varphi = \exists x \neg \exists y\ y < x$ gilt $\mathit{frei}(\varphi) = \varnothing$. ⊣

5.2. Semantik der Prädikatenlogik

In der Aussagenlogik wurde die Semantik durch eine Belegung der Variablen mit Wahrheitswerten definiert. In der Prädikatenlogik werden wir ebenfalls Belegungen als Basis der Semantik verwenden, allerdings wird hier den Variablen ein Wert aus dem Universum der Struktur zugewiesen.

Definition 5.10 (Belegungen und Interpretationen) Sei σ eine Signatur und $\mathcal{A}$ eine σ-Struktur.

(1) Eine *Belegung* in $\mathcal{A}$ ist eine Funktion $\beta : \text{def}(\beta) \to A$ mit $\text{def}(\beta) \subseteq \text{VAR}$. β ist *passend* für $\varphi \in \text{FO}[\sigma]$, wenn $\mathit{frei}(\varphi) \subseteq \text{def}(\beta)$.

(2) Eine *σ-Interpretation* ist ein Paar $(\mathcal{A}, \beta)$, bestehend aus einer σ-Struktur $\mathcal{A}$ und einer Belegung β in $\mathcal{A}$. $\mathcal{I} := (\mathcal{A}, \beta)$ ist passend für $\varphi \in \text{FO}[\sigma]$, wenn β zu φ passt. ⊣

Wir vereinbaren als nächstes folgende Notation.

Notation 5.11 Sei $\mathcal{A}$ eine σ-Struktur.

(1) Ist β eine Belegung, $x \in \text{VAR}$ und $a \in A$ ein Element, dann definieren wir eine neue Belegung $\beta[x/a]$ mit $\text{def}(\beta[x/a]) := \text{def}(\beta) \cup \{x\}$ durch

$$\beta[x/a](y) := \begin{cases} a & \text{wenn } x = y \\ \beta(y) & \text{sonst.} \end{cases}$$

Die Belegung $\beta[x/a]$ ist also genauso wie β definiert, mit dem Unterschied, dass x nun durch a interpretiert wird.

(2) Ist $\mathcal{I} := (\mathcal{A}, \beta)$ eine σ-Interpretation, $x \in \text{VAR}$ und $a \in A$ ein Element, dann definieren wir $\mathcal{I}[x/a]$ als $(\mathcal{A}, \beta[x/a])$. ⊣

Definition 5.12 Sei σ eine Signatur. Induktiv über den Formelaufbau definieren wir eine Funktion $[\![\cdot]\!]^{\cdot}$, die jedem Term $t \in \mathcal{T}_\sigma$ und jeder σ-Interpretation $\mathcal{I} := (\mathcal{A}, \beta)$ für t einen Wert $[\![t]\!]^{\mathcal{I}} \in A$ zuweist.

Basisfall.

- $[\![v_i]\!]^{\mathcal{I}} := \beta(v_i)$ für alle $v_i \in \text{VAR}$
- $[\![c]\!]^{\mathcal{I}} := c^{\mathcal{A}}$ für alle Konstantensymbole $c \in \sigma$.

Induktionsschritt.

- Ist $f \in \sigma$ ein k-stelliges Funktionssymbol und $t_1, \ldots, t_k \in \mathcal{T}_\sigma$ dann gilt

$$[\![f(t_1, \ldots, t_k)]\!]^{\mathcal{I}} := f^{\mathcal{A}}\big([\![t_1]\!]^{\mathcal{I}}, \ldots, [\![t_k]\!]^{\mathcal{I}}\big). \qquad \dashv$$

Wir können nun die Semantik prädikatenlogischer Formeln definieren.

Definition 5.13 (Semantik der Prädikatenlogik) Sei σ eine Signatur. Induktiv über den Formelaufbau definieren wir eine Funktion $[\![\cdot]\!]^{\cdot}$, die jeder Formel $\varphi \in \text{FO}[\sigma]$ und jeder σ-Interpretation $\mathcal{I} := (\mathcal{A}, \beta)$ für φ einen Wahrheitswert $[\![\varphi]\!]^{\mathcal{I}} \in \{0, 1\}$ zuordnet.

Basisfall.

- Für alle Terme $t, t' \in \mathcal{T}_\sigma$ definieren wir

$$[\![t = t']\!]^{\mathcal{I}} := \begin{cases} 1 & \text{wenn } [\![t]\!]^{\mathcal{I}} = [\![t']\!]^{\mathcal{I}} \\ 0 & \text{sonst.} \end{cases}$$

- Für alle k-stelligen Relationssymbole $R \in \sigma$ und alle Terme $t_1, \ldots, t_k \in \mathcal{T}_\sigma$ definieren wir

$$[\![R(t_1, \ldots, t_k)]\!]^{\mathcal{I}} := \begin{cases} 1 & \text{wenn } ([\![t_1]\!]^{\mathcal{I}}, \ldots, [\![t_k]\!]^{\mathcal{I}}) \in R^{\mathcal{A}} \\ 0 & \text{sonst.} \end{cases}$$

Induktionsschritt.

- Die Semantik der Verknüpfungen $\neg, \vee, \wedge, \rightarrow, \leftrightarrow$ ist wie in der Aussagenlogik definiert, z.B. wenn $\varphi := \neg\psi \in \text{FO}[\sigma]$ dann definieren wir

$$[\![\varphi]\!]^{\mathcal{I}} := 1 - [\![\psi]\!]^{\mathcal{I}}.$$

- Wenn $\varphi := \exists x\psi \in \text{FO}[\sigma]$, dann definieren wir

$$[\![\varphi]\!]^{\mathcal{I}} := \begin{cases} 1 & \text{es gibt } a \in A, \text{ so dass } [\![\psi]\!]^{\mathcal{I}[x/a]} = 1 \\ 0 & \text{sonst.} \end{cases}$$

- Wenn $\varphi := \forall x\psi \in \text{FO}[\sigma]$, dann definieren wir

$$[\![\varphi]\!]^{\mathcal{I}} := \begin{cases} 1 & [\![\psi]\!]^{\mathcal{I}[x/a]} = 1 \text{ für alle } a \in A \\ 0 & \text{sonst.} \end{cases}$$

⊣

Wir betrachten als nächstes einige Beispiele von Formeln und deren Bedeutung.

Beispiel 5.14 Sei $\sigma := \{+, *, 0, 1\}$ die Signatur der Arithmetik und sei $\mathcal{A} := (\mathbb{N}, +^{\mathcal{A}}, *^{\mathcal{A}}, 0^{\mathcal{A}}, 1^{\mathcal{A}})$ die Struktur über den natürlichen Zahlen mit der üblichen Interpretation von $+^{\mathcal{A}}, *^{\mathcal{A}}, 0^{\mathcal{A}}, 1^{\mathcal{A}}$. Sei $\beta : x \mapsto 2, y \mapsto 3$ und $\mathcal{I} := (\mathcal{A}, \beta)$. Dann gilt

- $[\![x * x]\!]^{\mathcal{I}} := \beta(x) *^{\mathcal{A}} \beta(x) = 4$ und
- $[\![x * x = y + 1]\!]^{\mathcal{I}} = 1$, da $\beta(x) *^{\mathcal{A}} \beta(x) = \beta(y) +^{\mathcal{A}} 1^{\mathcal{A}} = 4$.

Sei $\varphi(x,y) := \exists z(x * x = y + z)$. Um zu zeigen, dass $[\![\varphi]\!]^{\mathcal{I}} = 1$, müssen wir ein Element $a \in \mathbb{N}$ finden mit $[\![x * x = y + z]\!]^{\mathcal{I}\cup\{z\mapsto a\}} = 1$. Sei $\beta' := \beta \cup \{z \mapsto 1\}$ und $\mathcal{I}' := (\mathcal{A}, \beta')$. Dann gilt $[\![x * x = y + z]\!]^{\mathcal{I}'} = 1$ wie zuvor und daher $[\![\exists z(x * x = y + z)]\!]^{\mathcal{I}} = 1$. ⊣

Beispiel 5.15 Sei $\sigma_{Graph} := \{E\}$.

- $\varphi := E(x,y)$. Dann ist $\mathit{frei}(\varphi) = \{x,y\}$. Sei $\mathcal{I} = (\mathcal{G}, \beta)$ mit $\mathcal{G} = (V, E^{\mathcal{G}})$, $\beta(x) := u \in V$ und $\beta(y) := v \in V$. Dann gilt $[\![\varphi]\!]^{\mathcal{I}} = 1$ genau dann, wenn es eine Kante zwischen u und v in $\mathcal{G}$ gibt.
- Sei $\varphi := \exists x \forall y\big(x = y \vee E(x,y)\big)$. Dann ist $\mathit{frei}(\varphi) := \varnothing$ und φ gilt in einem Graphen, wenn es einen Knoten mit Kanten zu allen anderen gibt.
- Sei $\varphi := \exists x_1 \exists x_2\big((E(x,x_1) \wedge E(x_1,x_2)) \wedge E(x_2,y)\big)$. Dann ist $\mathit{frei}(\varphi) := \{x,y\}$. Dann gilt φ in einem Graphen mit Belegung β, wenn es einen Weg der Länge 1 oder 3 von $\beta(x)$ zu $\beta(y)$ gibt. Man beachte, dass x, x_1, x_2 und y nicht unbedingt paarweise verschieden sein müssen.
- Sei $\varphi := \forall x \forall y\big(E(x,y) \to E(y,x)\big)$. Dann ist $\mathit{frei}(\varphi) = \varnothing$ und φ gilt in einem Graphen, wenn er „ungerichtet“ist, d.h. es zu jeder Kante von u nach v auch die Kante von v nach u gibt. ⊣

Beispiel 5.16 Sei $\sigma := \{<\}$ die Signatur der Ordnungen, d.h. $<$ ist eine binäre Relation. Wir beschreiben im folgenden einige Eigenschaften von Relationen $<$ durch logische Formeln.

(1) „$<$ ist irreflexiv“: $\forall x\, \neg x < x$.

(2) „$<$ ist transitiv“: $\forall x \forall y \forall z\big((x < y \wedge y < z) \to x < z\big)$.

(3) Das heißt, wir können wie folgt sagen, dass $<$ eine partielle Ordnung ist:
$\varphi_{\mathit{par\text{-}ord}} := \Big(\forall x\, \neg x < x\Big) \wedge \Big(\forall x \forall y \forall z\big((x < y \wedge y < z) \to x < z\big)\Big)$.

(4) „$<$ ist total“: $\varphi_t := \forall x \forall y (x < y \vee x = y \vee y < x)$.

(5) Eine lineare Ordung $<$ wird formalisiert durch $\varphi_{\mathit{ord}} := \varphi_{\mathit{par\text{-}ord}} \wedge \varphi_t$. ⊣

Beispiel 5.17 Ein *vertex cover* eines ungerichteten Graphen $\mathcal{G} := (V, E^{\mathcal{G}})$ ist eine Menge $X \subseteq V$, so dass $u \in X$ oder $v \in X$ für alle Kanten $e := \{u,v\} \in E^{\mathcal{G}}$. Das vertex cover Problem ist das folgende Problem: Gegeben ein Graph $\mathcal{G}$ und $k \in \mathbb{N}$, enthält $\mathcal{G}$ ein vertex cover der Größe $\leqslant k$?

In dieser Formalisierung wird eine Aussage über eine Menge X und über alle Kanten gemacht (nicht über Knoten). Dies können wir in der Prädikatenlogik nicht direkt ausdrücken. Wir formulieren das Problem daher um.

$\mathcal{G}$ enthält ein vertex cover der Größe $\leqslant k$, wenn

- es k Knoten $x_1, \ldots, x_k$ gibt, so dass
- für alle u, v: Wenn es eine Kante zwischen u und v gibt, dann ist u eins der x_i oder v eins der x_i.

Das können wir nun eins-zu-eins in die Prädikatenlogik übersetzen.

$$\exists x_1 \ldots \exists x_k \; \forall u \forall v \Big(E(u,v) \to \big(\bigvee_{i=1}^{k} u = x_i \vee \bigvee_{i=1}^{k} v = x_i \big) \Big) \qquad \dashv$$

Wie in der Aussagenlogik hängt eine Formel nur von den Teilen einer Interpretation ab, die auch von der Formel benutzt werden.

Theorem 5.18 (Koinzidenzlemma) *Seien σ, τ, τ' Signaturen, so dass $\sigma \subseteq \tau \cap \tau'$. Sei $\mathcal{I} := (\mathcal{A}, \beta)$ eine τ-Interpretation und $\mathcal{J} := (\mathcal{B}, \gamma)$ eine τ'-Interpretation, so dass*

- *$A = B$,*
- *$S^{\mathcal{A}} = S^{\mathcal{B}}$ für alle Symbole S, die in σ vorkommen.*

(1) *Ist $t \in \mathcal{T}_\sigma$ ein σ-Term und $\beta(x) = \gamma(x)$ für alle $x \in \mathrm{var}(t)$, dann*

$$[\![t]\!]^{\mathcal{I}} = [\![t]\!]^{\mathcal{J}}.$$

(2) *Ist $\varphi \in \mathrm{FO}[\sigma]$ eine Formel und $\beta(x) = \gamma(x)$ für alle $x \in \mathit{frei}(\varphi)$, dann ist*

$$[\![\varphi]\!]^{\mathcal{I}} \quad = \quad [\![\varphi]\!]^{\mathcal{J}}.$$

Wir führen als nächstes noch einige wichtige Schreibweisen und Konzepte ein.

Definition 5.19 Sei $\varphi \in \mathrm{FO}[\sigma]$ eine Formel und sei $\Phi \subseteq \mathrm{FO}[\sigma]$ eine Formelmenge.

(1) Eine σ-Interpretation $\mathcal{I}$ *erfüllt* φ, wenn $\mathcal{I}$ zu φ passt und $[\![\varphi]\!]^{\mathcal{I}} = 1$. Wir sagen auch: $\mathcal{I}$ ist ein *Modell* von φ und schreiben $\mathcal{I} \models \varphi$.

(2) Eine σ-Interpretation $\mathcal{I}$ *erfüllt* Φ, wenn $\mathcal{I}$ zu allen $\psi \in \Phi$ passt und $[\![\psi]\!]^{\mathcal{I}} = 1$ für alle $\psi \in \Phi$. Wir sagen auch: $\mathcal{I}$ ist ein *Modell* von Φ und schreiben $\mathcal{I} \models \Phi$.$\dashv$

Notation 5.20 (1) Sei $\varphi \in \mathrm{FO}[\sigma])$ eine Formel mit $\mathit{frei}(\varphi) \subseteq \{x_1, \ldots, x_k\}$.

Sei $\mathcal{A}$ eine σ-Struktur und β eine Belegung, so dass $\beta(x_i) := a_i$, für alle $1 \leqslant i \leqslant k$. Wir schreiben $\mathcal{A} \models \varphi[x_1/a_1, \ldots, x_k/a_k]$ statt $\mathcal{I} \models \varphi$. Dies ist möglich, da nach dem Koinzidenzlemma die Belegung der Variablen außer $x_1, \ldots, x_k$ nicht relevant ist.

(2) Wir schreiben $\varphi(x_1, \ldots, x_k)$ um anzudeuten, dass $\mathit{frei}(\varphi) \subseteq \{x_1, \ldots, x_k\}$. In diesem Fall vereinfachen wir obige Notation weiter und schreiben $\mathcal{A} \models \varphi[a_1, \ldots, a_k]$. Man beachte, dass dies von der Sequenz $x_1, \ldots, x_k$ abhängt.

(3) Wenn φ ein Satz ist, schreiben wir $\mathcal{A} \models \varphi$. ⊣

Viele unserer bisherigen Beispiele waren Sätze. Z.B. haben wir Sätze der Prädikatenlogik gesehen, die besagen, dass das Relationssymbol $<$ durch eine partielle Ordnung interpretiert werden muss. In verschiedenen Anwendungen hat man aber oft Formeln $\varphi(x_1, \ldots, x_k)$ mit freien Variablen und möchte in einer gegebenen Struktur $\mathcal{A}$ z.B. die Menge aller Belegungen β der Variablen $x_1, \ldots, x_k$ ausrechnen, die die Formel in $\mathcal{A}$ erfüllen, d.h. so dass gilt $(\mathcal{A}, \beta) \models \varphi$. Wir führen daher folgende Definition ein.

Definition 5.21 Sei σ eine Signatur und $\mathcal{A}$ eine σ-Struktur mit Universum A. Für eine Formel $\varphi(x_1, \ldots, x_k) \in \text{FO}[\sigma]$ definieren wir

$$\varphi(\mathcal{A}) := \{(a_1, \ldots, a_k) \in A^k : \mathcal{A} \models \varphi[a_1, \ldots, a_k]\}.$$ ⊣

Bemerkung 5.22 Die Relation $\varphi(\mathcal{A})$ hängt nicht nur von $\mathcal{A}$ sondern auch von der Sequenz $(x_1, \ldots, x_k) \in \text{VAR}^k$ ab. Wir müssen daher diese Sequenz jeweils angeben, bevor wir die Notation benutzen können. Diese Anforderung ist sinnvoll und findet sich in der Methodendefinition aller gängigen Programmiersprachen. Sei z.B.

`Boolean phi(int` x_1`, ..., int` x_k`)`

eine Java-Methode. Dann wird mit $x_1, \ldots, x_k$ eine Ordnung der Parameter festgelegt. Wir können dann *`Boolean b = phi(3, 5, ..., 17);`* benutzen, was ohne vorherige Angabe der Methodendefinition nicht möglich wäre. ⊣

5.3. Relationale Datenbanken

Wir werden in diesem Abschnitt eine wichtige Anwendung der Logik in der Informatik kurz besprechen, die *relationalen Datenbanken*. Eine *relationale Datenbank* ist eine endliche Menge von *„Tabellen“*. Zum Beispiel könnte eine Filmdatenbank wie imdb.org folgende Tabellen enthalten.

Actors

Schauspieler	*ID*	*Geburtsdatum*
George Clooney	1	6. Mai 1961
Scarlett Johansson	2	22. November 1984
Jeff Daniels	3	19. Februar 1955
...	...	...

Movies

Titel	*Regie*	*Schauspieler*
Good night ... and good luck	Georg Clooney	1
Good night ... and good luck	Georg Clooney	3
Lost in translation	Sofia Coppola	2
...	...	...

Die Menge τ von *Tabellennamen* heißt *Datenbankschema*.

- Jede *Spalte* einer Tabelle in der Datenbank enthält Einträge vom selben Typ, z.B. Wörter oder Zahlen. In Datenbankterminologie werden Spaltennamen *Attribute* genannt. Jedes Attribut i hat einen Typ D_i, genannt *domain*.

- Jede *Zeile* der Tabelle enthält ein Tupel $(x_1, \ldots, x_n) \in D_1 \times D_2 \times \cdots \times D_n$.

Eine Datenbanktabelle kann daher als n-stellige Relation über der Menge $D := D_1 \cup \cdots \cup D_n$ aufgefasst werden. Eine relationale Datenbank mit Schema τ kann also als τ-Struktur $\mathcal{D}$ wie folgt geschrieben werden:

- Das Universum $A := D_1 \cup D_2 \cup \cdots \cup D_n$ ist die Vereinigung aller Domains.

- Für jede Tabelle $R \in \tau$ enthält die Struktur eine Relation $R^{\mathcal{D}}$, die alle Tupel der Tabelle enthält.

Relationale Datenbanken können also leicht als mathematische Strukturen modelliert werden. Eine Abfrage an eine Datenbank entspricht dann dem Auswerten logischer Formeln in Strukturen. Es existiert daher ein enger Zusammenhang zwischen mathematischer Logik, besonders dem Teilgebiet der *endlichen Modelltheorie*, und der Theorie relationaler Datenbanken. Historisch wurden relationale Datenbanken nach logischen Strukturen modelliert. Die Theorie relationaler Strukturen, insbesondere die Ausdrucksstärke von Logiken als Anfragesprache war bereits sehr gut untersucht und hat die Entwicklung relationaler Datenbanken stark beeinflusst.

Beispiel 5.23 Betrachten wir noch einmal das Filmbeispiel. Der domain aller Einträge sind Zeichenketten. Sei Σ^* die Menge aller Zeichenketten über dem Alphabet $\{a, \ldots, z, A, \ldots, Z, 0, \ldots, 9\}$. Die Filmdatenbank entspricht folgender Struktur $\mathcal{D}$ über der Signatur $\sigma := \{\textit{Actors, Movies}\}$:

- Das Universum ist $D := \Sigma^*$.

- Die Relationen
  - $\textit{Actors}^{\mathcal{D}} := \{$ *(George Clooney, 1, 6 May 1961), (Scarlett Johansson, 2, 22 November 1984), (Jeff Daniels, 3, 19 February 1955)* $\}$ und
  - $\textit{Movies}^{\mathcal{D}} :=$ { *(Good night ... and good luck, George Clooney, 1), (Good night ... and good luck, George Clooney, 3), (Lost in translation, Sofia Coppola, 2)* }. ⊣

Im obigen Beispiel ist das Universum der Struktur $\mathcal{D}$ unendlich. Dies ist eine unangenehme Eigenschaft, da die Komplemente von endlichen Relationen dann unendlich sind. Datenbanken werden daher meistens als *endliche Strukturen* modelliert. Das Universum enthält nur den *active domain* der Datenbank, d.h. die Menge aller Elemente, die in der Datenbank vorkommen.

Beispiel 5.24 Im Beispiel ist der active domain die Menge

D := { George Clooney, Scarlett Johansson, Jeff Daniels, 1, 2, 3, 6 May 1961, 22 November 1984, 19 February 1955, Good night ... and good luck, Lost in translation, Sofia Coppola }.

Die Relationen sind genauso wie zuvor definiert. Die Datenbank entspricht also der Struktur $\mathcal{D}$ über der Signatur $\sigma = \{Actors, Movies\}$ mit Universum D und obigen Relationen. ⊣

Beispiel 5.25 Mögliche Anfragen über der Signatur des obigen Beispiels sind:

(1) $\exists x \exists d \exists n_1 \exists n_2 \big(\text{Movies}(x, d, n_1) \wedge \text{Movies}(x, d, n_2) \wedge n_1 \neq n_2\big)$. Die Formel formuliert die Aussage, dass es einen Film mit mehr als einem Schauspieler gibt.

(2) $\exists d \exists f \exists n \exists b$ $\text{Movies}(f, d, n) \wedge \text{Actors}(d, n, b)$. Die Formel besagt, dass es einen Regisseur gibt, der im eigenen Film mitspielt. ⊣

Die Anfragen beschreiben Eigenschaften der Datenbank, die wahr oder falsch sein können. Derartige Anfragen werden als *Boolesche Anfragen* bezeichnet. Meistens wollen wir nicht-Boolesche Informationen aus einer Datenbank gewinnen, z.B. *"Gib alle Filme von George Clooney aus"*.

Dies können wir durch eine Anfrage mit freien Variablen erreichen:

$$\varphi(f) := \exists n \text{ Movies}(f, \text{„G-Clooney“}, n).$$

Man beachte, dass wir in prädikatenlogischen Formeln nicht einfach Elemente des Universums einer Struktur verwenden dürfen, wir können also nicht einfach „George Clooney“ in unseren Formeln verwenden. Formal gesehen müssen wir daher ein Konstantensymbol *G-Clooney* einführen und dieses in der Struktur durch das Object „George Clooney“ interpretieren. Formal arbeiten wir in diesem Beispiel also mit Strukturen über der Signatur *{Actors, Movies, G-Clooney}*.

Die Menge $\varphi(\mathcal{D})$ (siehe Definition 5.21) enthält für eine Datenbank $\mathcal{D}$ also genau die Filme, in denen George Clooney mitgespielt hat.

In praktischen Anwendungen von Datenbanken verwendet man meistens nicht die Prädikatenlogik as Anfragesprache. Die wichtigste Anfragesprache in relationalen Datenbanksystemen ist die *standard query language* (SQL). SQL ist äquivalent zur Prädikatenlogik in dem Sinn, dass sich genau die gleichen Eigenschaften und Anfragen an Datenbanken definieren lassen[1]. Allerdings enthält SQL noch weitere Funktionen, wie z.B. Aggregationsfunktionen (Mittelwerte, Summenbildung etc).

In SQL erhält man alle Filme von Regisseur George Clooney mit folgender Anfrage.

```
SELECT Title
FROM Movies
WHERE Director=''George Clooney''
```

[1] Aktuelle SQL-Standards enthalten auch Konzepte zur Fixpunktbildung, die über die Prädikatenlogik hinausgehen. Allerdings werden diese selten verwendet, daher ist der logische Kern von SQL immernoch die Prädikatenlogik.

Als Antwort erhält man die Tabelle

Answer
Good night ... and good luck

In der Sprache der relationalen Strukturen entspricht dies der unären Relation

$$\varphi((\mathcal{D}, \text{G-Clooney})) = \{\text{Good night} \ldots \text{and good luck}\}.$$

Es ist eins der eleganten Eigenschaften des relationalen Datenbankmodells, dass Antworten auf Anfragen selbst wieder Tabellen sind und somit in Datenbanken gespeichert werden können.

5.4. Logische Folgerung, Erfüllbarkeit und Allgemeingültigkeit

Wie in der Aussagenlogik können wir nun Begriffe wie Erfüllbarkeit, logische Folgerung, etc. definieren. Der Begriff der logischen Folgerung ist vermutlich der wichtigste Begriff der Logik überhaupt und wird uns im weiteren Verlauf noch mehrfach beschäftigen.

Definition 5.26 (Erfüllbarkeit und Allgemeingültigkeit) Sei σ eine Signatur. Sei $\varphi \in \text{FO}[\sigma]$ eine Formel und $\Phi \subseteq \text{FO}[\sigma]$ eine Formelmenge.

(1) Eine Interpretation $\mathcal{I}$ passt zu Φ, wenn $\mathcal{I}$ zu allen $\psi \in \Phi$ passt.

(2) Die Formel φ ist *erfüllbar*, wenn sie ein Modell hat, d.h. eine Interpretation $\mathcal{I}$ existiert, so dass $\mathcal{I} \models \varphi$. Anderenfalls ist es *unerfüllbar*. Analog ist Φ erfüllbar, wenn es ein Modell von Φ gibt, und ansonsten unerfüllbar.

(3) φ ist *allgemeingültig*, oder eine *Tautologie*, wenn alle passenden Interpretationen φ erfüllen. Ebenso ist Φ allgemeingültig, wenn alle zu Φ passenden Belegungen auch Φ erfüllen. ⊣

Als nächstes führen wir den Begriff der logischen Folgerung für die Prädikatenlogik ein.

Definition 5.27 (Logische Folgerung) Sei σ eine Signatur. Sei $\Phi \subseteq \text{FO}[\sigma]$ und $\psi \in \text{FO}[\sigma]$. ψ ist eine *Folgerung* von Φ, geschrieben $\Phi \models \psi$, wenn für jede zu Φ und ψ passende σ-Interpretation $\mathcal{I}$ gilt:

$$\mathcal{I} \models \Phi \quad \Longrightarrow \quad \mathcal{I} \models \psi.$$

Statt $\varnothing \models \psi$ schreiben wir $\models \psi$. ⊣

Das folgende Lemma, dessen Beweis zur Übung empfohlen ist, gibt einige einfache Eigenschaften der Folgerungsbeziehung an.

Lemma 5.28 (1) *Für alle* $\Phi \subseteq \text{FO}[\sigma]$ *und* $\psi \in \text{FO}[\sigma]$*:*

$$\Phi \models \psi \quad \iff \quad \Big(\Phi \cup \{\neg\psi\} \quad \textit{ist unerfüllbar}\Big)$$

(2) *Für alle* $\psi \in \text{FO}[\sigma]$*:*

$$\models \psi \quad \iff \quad \Big(\psi \textit{ ist eine Tautologie}\Big).$$

Im Beispiel 5.16 haben wir bereits gesehen, dass es einen Satz $\varphi_{ord} \in \text{FO}[\{<\}]$ gibt, der genau dann in einer $\{<\}$-Struktur $\mathcal{A} := (A, <^{\mathcal{A}})$ gilt, wenn $<^{\mathcal{A}}$ eine lineare Ordnung auf A ist. D.h. die Klasse der linearen Ordnungen ist in diesem Sinne in der Prädikatenlogik *definierbar*. Ebenso ist die Klasse der ungerichteten, schleifenfreien Graphen durch den Satz $\varphi := \forall x \neg E(x,x) \wedge \forall x \forall y \big(E(x,y) \to E(y,x)\big)$ definiert. Für einige natürliche Klassen von Strukturen gibt es aber keinen Satz der Prädikatenlogik, der die Klasse definiert. Manchmal gibt es aber eine unendliche Menge Φ von Formeln, die die Klasse beschreibt. Man sagt dann, dass Φ die Klasse *axiomatisiert*. Die folgende Definition definiert diese Begriffe genauer.

Definition 5.29 Sei σ eine Signatur und $\Phi \subseteq \text{FO}[\sigma]$ eine Menge von σ-Sätzen.

(1) Die *Modellklasse* von Φ, geschrieben $\text{Mod}(\Phi)$, ist die Klasse aller σ-Strukturen $\mathcal{A}$ mit $\mathcal{A} \models \Phi$. Falls $\Phi := \{\varphi\}$ nur einen Satz enthält, schreiben wir kurz $\text{Mod}(\varphi)$.

(2) Eine Klasse $\mathcal{C}$ von σ-Strukturen ist *axiomatisiert* durch Φ, wenn $\mathcal{C} = \text{Mod}(\Phi)$. Wir nennen Φ ein *Axiomensystem* für $\mathcal{C}$.

(3) Eine Klasse $\mathcal{C}$ von σ-Strukturen ist *axiomatisierbar* (in der Prädikatenlogik), oder FO-*axiomatisierbar*, wenn $\mathcal{C} = \text{Mod}(\Phi)$ für eine Menge $\Phi \subseteq \text{FO}[\sigma]$. Wenn es eine endliche Menge Φ mit $\mathcal{C} = \text{Mod}(\Phi)$ gibt, so heißt $\mathcal{C}$ *endlich axiomatisierbar* (in der Prädikatenlogik) oder *definierbar* (in der Prädikatenlogik). ⊣

Bemerkung 5.30 Es gilt offensichtlich, dass eine Klasse von Strukturen endlich axiomatisierbar ist, genau dann, wenn sie bereits durch einen einzigen Satz der Prädikatenlogik definiert wird. ⊣

Wie wir oben schon gesehen haben, die die Klasse aller linearen Ordnungen FO-definierbar. Die Klasse $\mathcal{O}$ aller unendlichen linearen Ordnungen ist jedoch nicht FO-definierbar (siehe Kapitel 6.3). Sie ist aber axiomatisierbar in der Prädikatenlogik, z.B. durch folgende Menge von Formeln:

$$\Phi_{ord} := \{\varphi_{ord}\} \cup \{\varphi_i : i \geqslant 1\},$$

wobei $\varphi_i := \exists x_1 \ldots \exists x_i \bigwedge_{1 \leqslant j < j' \leqslant i} x_j < x_{j'}$ besagt, dass es (mindestens) i Elemente $x_1, \ldots, x_i$ gibt, die eine strikt aufsteigende Kette ergeben, d.h. es gilt $x_j < x_{j'}$ für alle $j < j'$.

Als weiteres Beispiel zeigen wir, dass die Klasse aller Gruppen definierbar ist. Da wir im weiteren nur Definierbarkeit in der Prädikatenlogik betrachten werden, lassen wir im Folgenden den Zusatz FO-definierbar meistens weg und sprechen kurz von definierbar und Definierbarkeit.

Beispiel 5.31 Eine *Gruppe* ist ein Tupel $(G, \circ, e, ^{-1})$, wobei $\circ$ eine zweistellige Funktion ist, e eine Konstante, und $^{-1}$ eine einstellige Funktion, so dass folgende Bedingungen erfüllt sind.

- $\circ$ ist assoziativ, d.h. $(a \circ (b \circ c)) = ((a \circ b) \circ c)$.
- e ist das neutrale Element, d.h. $a \circ e = a$.
- $^{-1}$ ist die inverse Funktion, d.h. $a \circ a^{-1} = e$.

Die Klasse aller Gruppen ist endlich axiomatisierbar durch

$$\Phi_{Gruppe} := \left\{ \begin{array}{c} \forall x \forall y \forall z \big(x \circ (y \circ z) = (x \circ y) \circ z \big) \\ \forall x (x \circ e = x) \\ \forall x (x \circ x^{-1} = e) \end{array} \right\}.$$

Gilt nun $\Phi_{Gruppe} \models \psi$, für eine Formel ψ, so gilt ψ in allen Gruppen. Zum Beispiel gilt $\Phi_{Gruppe} \models \forall x (x^{-1} \circ x = e)$ (in jeder Gruppe ist das Rechtsinverse auch das Linksinverse). Andererseits gilt $\Phi_{Gruppe} \not\models \forall x \forall y (x \circ y = y \circ x)$, da nicht alle Gruppen kommutativ sind. ⊣

Das vorherige Beispiel demonstriert eine fundamentale Anwendung der Definierbarkeit oder Axiomatisierbarkeit. Denn angenommen eine Klasse von Strukturen, wie z.B. die Klasse der Gruppen, ist FO-definierbar durch einen Satz φ. Wenn wir nun eine Aussage über Gruppen in der Prädikatenlogik durch einen Satz ψ formalisieren können, dann gilt diese Aussage in allen Gruppen genau dann, wenn $\varphi \models \psi$. Ist die Klasse nur axiomaitiserbar, z.B. durch eine Menge Φ, dann gitl analog die Aussage ψ in allen Strukturen der Klasse genau dann, wenn $\Phi \models \psi$.

Im Beispiel oben können wir also Sätze über Gruppen dadurch beweisen, dass wir die Aussage in der Prädikatenlogik durch einen Satz ψ formulieren und dann zeigen, dass ψ aus Φ_{Gruppe} folgt. Wir werden später mit dem Sequenzenkalkül ein Verfahren kennen lernen, in denen solche Folgerungen formal bewiesen werden können. Siehe Kapitel 6.4. Mit Hilfe solcher Methoden lassen sich daher Aussagen über Gruppen und anderen Strukturen sehr allgemein beweisen, was z.B. in sogenannten *Theorembeweisern* ausgenutzt wird. Allerdings ist es im allgemeinen unentscheidbar, ob eine Formel φ aus einer Formelmenge Φ folgt. Daher sind automatischen Theorembeweisern Grenzen gesetzt.

Die Klasse aller (prädikatenlogisch formalisierbaren) Theoreme, die in allen Gruppen gelten, ist also gerade die Klasse aller FO-Formeln, die aus den Gruppenaxiomen Φ_{Gruppe} logisch folgen. Wir nennen das die *Theorie* der Gruppen.

Definition 5.32 Sei σ eine Signatur.

(1) Eine *σ-Theorie* ist eine erfüllbare Menge $T \subseteq \text{FO}[\sigma]$ von Sätzen, die unter $\models$ abgeschlossen ist, d.h. wenn $T \models \psi$ für einen σ-Satz ψ gilt, dann ist $\psi \in T$.

(2) Eine Theorie T ist *vollständig*, wenn für jeden Satz $\psi \in \text{FO}[\sigma]$ gilt:

$$\psi \in T \text{ oder } \neg\psi \in T.$$ ⊣

Beispiel 5.33 Wir haben bereits gesehen, dass die Klasse aller Gruppen endlich axiomatisiert ist durch eine Formelmenge Φ_{Gruppe}. Die Gruppentheorie ist also

$$T := \{\psi \in \text{FO}[\sigma] : \psi \text{ ist ein Satz und } \Phi_{Gruppe} \models \psi\}.$$

Die Theorie ist nicht vollständig, da nicht alle Gruppen kommutativ sind und somit für den Satz $\varphi := \forall x \forall (x \circ y = y \circ x)$, der besagt, dass die Operation $\circ$ kommutativ ist, weder $\varphi \in T$ noch $\neg\varphi \in T$. ⊣

Man beachte, dass ψ auch freie Variablen enthalten kann. Zum Beispiel gilt $\Phi_{Gruppe} \models x^{-1} \circ x = e$ (in jeder Gruppe ist das Rechtsinverse auch das Linksinverse).

5.5. Äquivalenz und Normalformen

Ähnlich wie für die Aussagenlogik können auch in der Prädikatenlogik Formeln durch Umformungen in äquivalente Formeln übersetzt werden. Oft wird gewünscht, dass die neue Formel eine Standardgestalt hat, d.h. in einer Normalform ist.

Definition 5.34 Sei σ eine Signatur. Zwei σ-Formeln $\varphi, \psi \in \text{FO}[\sigma]$ sind *äquivalent*, geschrieben $\varphi \equiv \psi$, wenn für alle σ-Interpretationen $\mathcal{I}$ passend zu φ und ψ gilt

$$\mathcal{I} \models \varphi \quad \Longleftrightarrow \quad \mathcal{I} \models \psi.$$ ⊣

Bemerkung 5.35 Nach Definition gilt für alle Formeln $\varphi, \psi \in \text{FO}[\sigma]$

$$\varphi \equiv \psi \quad \Longleftrightarrow \quad \Big(\varphi \leftrightarrow \psi \text{ ist allgemeingültig }\Big)$$ ⊣

5.5.1. Substitution

Genauso wie in der Aussagenlogik wollen wir einen Begriff der *Substitution* einführen. Unser Ziel dabei ist es, Variablen durch Terme *sinnvoll* zu ersetzen. Wenn wir z.B. in der σ_{ar}-Formel $\exists y\ y * y = x + x$ die Variable x durch $(1+1)$ ersetzen, erhalten wir $\exists y\ y * y = (1 + 1)\ +\ (1 + 1)$.

Man muss aber aufpassen, welche Variablen man ersetzt und was man für sie einsetzt. Das Umbenennen einer Variablen in einer Formel, d.h. Substitution durch eine andere Variable, soll den Sinn der Formel nicht ändern. Das folgende Beispiel zeigt potentielle Probleme.

Sei $\varphi := \exists y\, y * y = x + x$.

(1) Wenn wir in φ die gebundene Variable y durch x ersetzen, erhalten wir die Formel

$$\exists x\, x * x = x + x$$

mit vollständig anderer Bedeutung. *Wir sollten daher nur freie Variablen substituieren.*

(2) Wenn wir in φ die freie Variable x durch y ersetzen, erhalten wir

$$\exists y\, y * y = y + y$$

was ebenfalls eine andere Bedeutung hat. *Wir müssen also auf Konflikte mit gebundenen Variablen achten.*

Wir werden daher Substitutionen so definieren, dass diese Probleme vermieden werden.

Definition 5.36 Sei σ eine Signatur.

(1) Eine σ-*Substitution* ist eine Abbildung $\mathcal{S} : \mathrm{def}(\mathcal{S}) \to \mathcal{T}_\sigma$ mit endlichem Wertebereich $\mathrm{def}(\mathcal{S}) \subseteq \textsc{Var}$.

(2) Für eine Substitution $\mathcal{S}$ definieren wir

$$\mathrm{var}(\mathcal{S}) := \bigcup_{x \in \mathrm{def}(\mathcal{S})} \mathrm{var}(\mathcal{S}(x))$$

als die Menge der Variablen, die in einem Term im Bild der Substitution vorkommen. ⊣

Anders als in der Aussagenlogik werden hier also Variablen nicht durch Formeln sondern durch Terme ersetzt.

Beispiel 5.37 Sei $\mathcal{S}$ definiert als

$$\mathcal{S} : \begin{array}{l} x \mapsto (y + z) \\ y \mapsto z + v. \end{array}$$

Dann $\mathrm{var}(\mathcal{S}) := \{y, z, v\}$. ⊣

Wie schon zuvor definieren wir die Anwendung einer Substitution getrennt für Terme und Formeln.

Definition 5.38 Sei $\mathcal{S}$ eine σ-Substitution. Induktiv über die Struktur von Termen definieren wir für jeden Term $t \in \mathcal{T}_\sigma$ den Term $t\mathcal{S}$, der durch *Anwendung* von $\mathcal{S}$ auf t entsteht, als:

- Wenn $t := x$, wobei $x \in \textsc{Var}$, dann $t\mathcal{S} := \begin{cases} \mathcal{S}(x) & \text{wenn } x \in \mathrm{def}(\mathcal{S}), \\ x & \text{sonst.} \end{cases}$
- Wenn $t := c$, für ein Konstantensymbol $c \in \sigma$, dann $t\mathcal{S} := c$.
- Wenn $t := f(t_1, \ldots, t_k)$, für ein k-stelliges Funktionssymbol $f \in \sigma$ und σ-Terme $t_1, \ldots, t_k \in \mathcal{T}_\sigma$, dann $t\mathcal{S} := f(t_1\mathcal{S}, \ldots, t_k\mathcal{S})$. ⊣

Beispiel 5.39 (Fort. von Beispiel 5.37) Betrachten wir wieder die Substitution $\mathcal{S}$ mit $\mathcal{S}(x) := (y + z)$ und $\mathcal{S}(y) := z + v$. Für $t := (x + y)$ gilt

$$t\mathcal{S} := \big((y + z) + (z + v)\big). \qquad \dashv$$

Definition 5.40 Sei $\mathcal{S}$ eine σ-Substitution. Induktiv über den Formelaufbau definieren wir für jede Formel $\varphi \in \text{FO}[\sigma]$ die Formel $\psi\mathcal{S}$, die durch Anwenden von $\mathcal{S}$ auf φ entsteht, als:

- Wenn $\varphi := R(t_1, \dots, t_k)$, wobei $R \in \sigma$ ein k-stelliges Relationssymbol ist und $t_1, \dots, t_k \in \mathcal{T}_\sigma$, dann $\varphi\mathcal{S} := R(t_1\mathcal{S}, \dots, t_k\mathcal{S})$.
- Wenn $\varphi := t_1 = t_2$ für σ-Terme t_1, t_2, dann $\varphi\mathcal{S} := t_1\mathcal{S} = t_2\mathcal{S}$.
- Wenn $\varphi := \neg\psi$ für $\psi \in \text{FO}[\sigma]$, dann $\varphi\mathcal{S} := \neg\, \psi\mathcal{S}$.
- Wenn $\varphi := (\psi_1 * \psi_2)$ für $\psi_1, \psi_2 \in \text{FO}[\sigma]$ und $* \in \{\vee, \wedge, \to, \leftrightarrow\}$, dann $\varphi\mathcal{S} := (\psi_1\mathcal{S} * \psi_2\mathcal{S})$.
- Sei $\varphi := \exists x\psi$ für eine Variable x und $\psi \in \text{FO}[\sigma]$.
  - Wenn $x \notin \text{var}(\mathcal{S})$, dann $\varphi\mathcal{S} := \exists x\psi\mathcal{S}'$, wobei $\mathcal{S}' := \mathcal{S}_{|\text{def}(\mathcal{S})\setminus\{x\}}$.
  - Wenn $x \in \text{var}(\mathcal{S})$, dann sei y die erste Variable in $\text{VAR} := \{v_0, v_1, v_2, \dots\}$, die nicht in $\mathit{frei}(\varphi) \cup \text{var}(\mathcal{S})$ vorkommt, und definiere $\mathcal{S}' := \mathcal{S}_{\text{def}(\mathcal{S})\setminus\{x\}} \cup \{x \mapsto y\}$. Wir definieren $\varphi\mathcal{S} := \exists y\psi\mathcal{S}'$.
- Der Fall $\varphi := \forall x\psi$ ist analog. $\dashv$

Wenn also bei der Substitution einer Variablen x, die Variable x nicht in den Variablen von $\mathcal{S}$ vorkommt, ist die Substitution sicher und wir substituieren in ψ. Anderenfalls können wir nicht einfach substituieren. Wir müssen zunächst x in y umbenennen, wobei y beliebig gewählt werden kann, solange $y \notin \mathit{frei}(\varphi) \cup \text{var}(\mathcal{S})$. Hier verwenden wir die erste unbenutzte Variable.

Beispiel 5.41 (Fort. von Beispiel 5.37) Wir betrachten wieder die Substitution $\mathcal{S}$ aus den vorherigen Beispielen. Für $\varphi := \exists a \forall z(x + y + z = a + x)$ gilt also

$$\begin{aligned}
\varphi\mathcal{S} &:= \exists a(\forall z(x + y + z = a + x)\mathcal{S} \\
&= \exists a\ \forall v_0(x + y + v_0 = a + x)\mathcal{S}' \\
&= \exists a\ \forall v_0\ (x + y + v_0)\mathcal{S}' = (a + x)\mathcal{S}' \\
&= \exists a\ \forall v_0\ \big((x + z) + (z + v) + v_0 = (a + (y + z))\big).
\end{aligned} \qquad \dashv$$

Notation 5.42

- Analog zur aussagenlogischen Substitution schreiben wir für eine Substitution $\mathcal{S}$ mit $\text{def}(\mathcal{S}) := \{x_1, \dots, x_n\}$ und $\mathcal{S}(x_i) := t_i$, $1 \leqslant i \leqslant n$,

$$[x_1/t_1, \dots, x_n/t_n].$$

Insbesondere schreiben wir $\varphi[x_1/t_1, \dots, x_n/t_n]$ statt $\varphi\mathcal{S}$.

- Wenn $\varphi(x_1, \ldots, x_k)$ eine Formel und $t_1, \ldots, t_k \in \mathcal{T}_\sigma$ Terme sind, schreiben wir
$$\varphi(t_1, \ldots, t_k)$$
statt $\varphi[x_1/t_1, \ldots, x_k/t_k]$. ⊣

Völlig analog zum aussagenlogischen Fall kann man nun das Substitutionslemma der Prädikatenlogik beweisen.

Lemma 5.43 (Substitutionslemma) *Sei $\mathcal{S}$ eine σ-Substitution. Für alle σ-Formeln φ, ψ gilt*
$$\varphi \equiv \psi \quad \Longrightarrow \quad \varphi\mathcal{S} \equiv \psi\mathcal{S}.$$

Ebenso gibt es analog zur Aussagenlogik auch ein entsprechendes Ersetzungslemma.

Lemma 5.44 (Ersetzungslemma) *Sei τ eine Signatur und seien $\varphi, \psi, \vartheta \in \mathrm{FO}[\tau]$. Sei ϑ eine Teilformel von ψ und $\vartheta \equiv \varphi$. Ferner, sei ψ' die Formel, die aus ψ entsteht, indem ϑ durch φ ersetzt wird. Dann gilt $\psi \equiv \psi'$.*

5.5.2. Nützliche Äquivalenzen von einigen Formeln

Wir geben in diesem Abschnitt einige Äquivalenzen zwischen Formeln an, die oft gebraucht werden. Die meistens sehr einfachen Beweise sind zur Übung empfohlen.

Lemma 5.45 *Sei $\varphi, \psi \in \mathrm{FO}[\sigma]$ und $x \in \mathrm{VAR}$.*

(1) $\neg\exists x\, \varphi \equiv \forall x\, \neg\varphi, \qquad \neg\forall x\, \varphi \equiv \exists x\, \neg\varphi.$

(2) *Wenn $x \notin \mathit{frei}(\varphi)$, dann*
$$\varphi \vee \exists x\, \psi \equiv \exists x(\varphi \vee \psi), \qquad \varphi \wedge \forall x\, \psi \equiv \forall x(\varphi \wedge \psi),$$
$$\varphi \wedge \exists x\ \psi \equiv \exists x(\varphi \wedge \psi), \qquad \varphi \vee \forall x\, \psi \equiv \forall x(\varphi \vee \psi).$$

Die Äquivalenzen in Teil (2) folgen aus dem Koinzidenzlemma.

Lemma 5.46 *Sei $\varphi \in \mathrm{FO}[\sigma]$ eine Formel und $x, y \in \mathrm{VAR}$ Variablen, so dass y nicht in φ vorkommt. Dann gilt*
$$\exists x\varphi \equiv \exists y\varphi[x/y], \qquad \forall x\varphi \equiv \forall y\varphi[x/y].$$

Beispiel 5.47 Mit obigen Äquivalenzen und den schon aus der Aussagenlogik bekannten Äquivalenzen, können wir zum Beispiel Folgendes beweisen.
$$\begin{aligned}\neg\exists x\big(x = y \vee \forall y E(x,y)\big) &\equiv \forall x \neg\big(x = y \vee \forall y E(x,y)\big)\\ &\equiv \forall x\big(\neg x = y \wedge \neg\forall y E(x,y)\big)\\ &\equiv \forall x\big(\neg x = y \wedge \exists y \neg E(x,y)\big).\end{aligned}$$

Ein weiteres Beispiel ist folgende Äquivalenzumformung.
$$\begin{aligned}\big(\exists x E(x,y)\big) \vee \big(\forall z E(y,z)\big) &\equiv \forall z\big(\exists x E(x,y)\big) \vee \big(E(y,z)\big)\\ &\equiv \forall z \exists x\big(E(x,y) \vee E(y,z)\big).\end{aligned}$$ ⊣

5.5.3. Normalformen

Ähnlich wie bei der Aussagenlogik werden wir als Nächstes einige syntaktische Normalformen für die Prädikatenlogik einführen, die uns das Arbeiten mit Formeln in bestimmten Situationen erleichtern. Genauer werden wir reduzierte Formeln, die Negationsnormalform und die Pränexnormalform vorstellen.

Wir haben bereits gesehen, dass Folgendes gilt.

(1) $\psi \wedge \varphi \quad \equiv \quad \neg(\neg\varphi \vee \neg\psi)$,

(2) $\psi \leftrightarrow \varphi \quad \equiv \quad (\varphi \rightarrow \psi) \wedge (\psi \rightarrow \varphi)$,

(3) $\psi \rightarrow \varphi \quad \equiv \quad \neg\psi \vee \varphi$,

(4) $\forall x\psi \quad \equiv \neg\exists\neg\psi$.

Mit Hilfe dieser Äquivalenzen können wir also jede Formel der Prädikatenlogik in eine äquivalente Formel umwandeln, in denen die Symbole $\leftrightarrow, \rightarrow, \wedge$ nicht vorkommen. Wir nennen solche Formeln *reduziert*.

Definition 5.48 Eine prädikatenlogische Formel ist in *Negationsnormalform* (NNF), wenn sie die Verknüpfungen $\rightarrow, \leftrightarrow$ nicht enthält und Negation nur direkt vor atomaren Formeln vorkommt. ⊣

Theorem 5.49 *Jede Formel der Prädikatenlogik ist logisch äquivalent zu einer Formel in Negationsnormalform.*

Beweis. Wir wissen bereits, dass die Verknüpfungen $\rightarrow, \leftrightarrow$ eliminiert werden können. Durch wiederholte Anwendung der De Morganschen Regeln $\neg(\psi \wedge \varphi) \equiv (\neg\psi \vee \neg\varphi)$ und $\neg(\psi \vee \varphi) \equiv (\neg\psi \wedge \neg\varphi)$ sowie der Äquivalenzen $\neg\exists x\psi \equiv \forall x\neg\psi$ und $\neg\forall x\psi \equiv \exists x\neg\psi$ und $\neg\neg\psi \equiv \psi$ kann jede Formel in eine äquivalente Formel in NNF umgewandelt werden. □

Als letzte Normalform stellen wir die Pränexnormalform vor. Ziel dieser Normalform ist es, die Formel so umzuschreiben, dass alle Quantoren vorne stehen, zum Beispiel $\exists u \forall x \exists y \exists z \big(R(u,u) \wedge R(x,y) \wedge \neg R(y,y) \wedge R(y,z)\big)$.

Definition 5.50 Eine Formel φ ist *bereinigt*, wenn

- keine Variable in φ sowohl frei als auch gebunden vorkommt und
- keine Variable zweimal quantifiziert wird. ⊣

Definition 5.51 Eine Formel ψ ist in *Pränexnormalform* (PNF), wenn sie bereinigt ist und die Form $Q_1 x_1 \ldots Q_l x_l \psi(x_1, \ldots, x_l)$ hat, wobei ψ quantorenfrei und $Q_i \in \{\exists, \forall\}$ ist. $Q_1, ..., Q_l$ heißt der (Quantoren-)Präfix von ψ. ⊣

Durch Anwendungen der Äquivalenzen aus Lemma 5.45 kann man leicht folgenden Satz zeigen.

Theorem 5.52 *Jede Formel der Prädikatenlogik kann effektiv in eine äquivalente Formel in Pränexnormalform übersetzt werden.*

6. Spiele und Kalküle

6.1. Spiele

In diesem Abschnitt führen wir den Begriff der *endlichen Spiele* ein. Dieser Formalismus wir uns zum einen erlauben, eine alternative Definition der Semantik für die Prädikatenlogik anzugeben. Wichtiger aber noch wird er uns eine elegante Methode liefern um zu beweisen, dass bestimmte Sachverhalte in der Prädikatenlogik nicht ausdrückbar sind. Zuerst definieren wir formal, was ein endliches Spiel ist.

Definition 6.1 Ein *endliches Spiel* ist ein Spiel mit perfekter Information zwischen zwei Spielern, *Verifizierer* und *Falsifizierer*, auf einer Arena $\mathcal{G} := (V, V_0, E, v_0)$ bestehend aus

- der Menge V aller Positionen,
- der Menge $V_0 \subseteq V$ der Positionen für *Verifizierer*,
- der Menge $E \subseteq V \times V$ möglicher Züge,
- der Startposition v_0.

Falsifizierer zieht an Positionen in $V \setminus V_0$.

⊣

Das Spiel heißt *wohl-fundiert*, wenn (V, E) azyklisch ist (als gerichteter Graph) und keine unendlichen Pfade enthält. Wir werden Positionen, in denen Verifizierer zieht, mit Kreisen und Positionen, in denen Falsifizierer zieht, mit Boxen in unseren Bildern zeichnen.

Sei im Folgenden $\mathcal{G} := (V, V_0, E, v_0)$ ein endliches Spiel.

Partien. Eine Partie ist eine endliche oder unendliche Folge $v_0, v_1, \ldots$ von Positionen, so dass $(v_i, v_{i+1}) \in E$ für alle $i \geqslant 0$ gilt. Eine Partie in $\mathcal{G}$ beginnt also in $v := v_0$. Wenn im Spielverlauf die aktuelle Position v in der Menge V_0 ist, dann kann *Verifizierer* zu einer neuen Position w mit $(v, w) \in E$ ziehen. Ist $v \in V \setminus V_0$ zieht *Falsifizierer* zu neuer Position w mit $(v, w) \in E$. Wenn ein Spieler nicht ziehen kann, d.h. die aktuelle Position hat keine Nachfolger, verliert er. Wenn das Spiel für immer weitergeht, dann ist es unentschieden.

Strategien. Für einen Knoten v sei

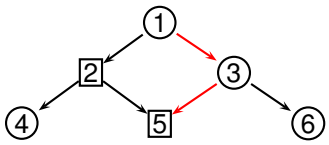
$$vE := \{w \in V : (v, w) \in E\}$$

die Menge der Nachfolger von v. Eine Strategie für *Verifizierer* ist eine Funktion f, die jedem $v \in V_0$ mit $vE \neq \varnothing$, einen Nachfolger $w \in vE$ zuordnet. Eine Strategie für *Falsifizierer* ist analog definiert für $V \setminus V_0$.

Eine Strategie sagt also dem Spieler für jeden Knoten v, an dem er ziehen muss, was zu tun ist, wenn die Partie v erreicht. Sei f eine Strategie für Verifizierer. Eine Partie $P = v_0, v_1 \ldots$ ist *konsistent* mit f, wenn $v_{i+1} = f(v_i)$ für alle $i \geqslant 0$ mit $v_i \in V_0$ gilt. Analog werden Partien definiert, die mit Strategien für Falsifizierer konsistent sind. Sei v_0 ein Knoten im Spiel. Eine Strategie ist eine *Gewinnstrategie* von v_0 aus, wenn der Spieler jede Partie gewinnt, in der er der Strategie folgt, also jede Partie, die mit der Strategie konsistent ist und mit v_0 beginnt.

Definition 6.2 Ein Spieler gewinnt ein Spiel $\mathcal{G}$ von einem Knoten v_0 aus, wenn er eine Gewinnstrategie auf $\mathcal{G}$ von v_0 aus hat. ⊣

Beispiel 6.3 Betrachten wir folgendes Spiel:



Hierbei bezeichnen Kreise Knoten für Verifizierer und Boxen Knoten für Falsifizierer. Verifizierer gewinnt von 1 aus, indem er von 1 zu 3 und von 3 zu 5 zieht. Da seine Knoten 4 und 6 keine Nachfolger haben, ist die Strategie auf diesen Knoten nicht definiert. ⊣

Endliche wohl-fundierte Spiele sind *determiniert*, d.h. in jedem solchen Spiel hat genau einer der Spieler eine Gewinnstrategie. Diese können auch sehr effizient berechnet werden, was für algorithmische Anwendungen von Spielen sehr nützlich ist.

Theorem 6.4 *In jedem endlichen, wohl-fundierten Spiel hat genau einer der Spieler eine Gewinnstrategie. Diese kann in Linearzeit berechnet werden.*

6.2. Eine spieltheoretische Semantik der Prädikatenlogik

Ziel dieses Abschnitts ist es, die Semantik der Prädikatenlogik über ein Zwei-Personen-Spiel zu definieren. Genauer werden wir für jede Formel $\varphi \in \text{FO}[\sigma]$ und σ-Struktur $\mathcal{A}$ ein Spiel $\mathfrak{G}(\mathcal{A}, \varphi)$ zwischen zwei Spielern, dem *Verifizierer* und dem *Falsifizierer*, definieren, so dass Verifizierer das Spiel $\mathfrak{G}(\mathcal{A}, \varphi)$ genau dann gewinnt, wenn $\mathcal{A} \models \varphi$.

Die Rolle der Spieler. Im Verlauf des Spiels versucht

- *Verifizierer* zu zeigen, dass die Formel in der Struktur gilt, und
- *Falsifizierer* zu zeigen, dass die Formel nicht in der Struktur gilt.

Man betrachte die Formel $\varphi := \exists x \forall y E(x, y)$ über der Signatur $\sigma := \{E\}$ der Graphen und einen Graph $\mathcal{G} := (V, E)$. Dann gilt $\mathcal{G} \models \varphi$, wenn es einen Knoten $v \in V$ gibt, der eine Kante zu allen Knoten hat. Um dies in ein Spiel umzuwandeln, lassen wir

(1) *Verifizierer* einen Knoten $v_1 \in V$ und dann

(2) *Falsifizierer* einen Knoten $v_2 \in V$ wählen.

Verifizierer gewinnt die Partie, wenn es eine Kante $(v_1, v_2) \in E$ gibt, sonst gewinnt Falsifizierer. Dies deutet darauf hin, dass Verifizierer bei Existenzquantoren und Falsifizierer bei Allquantoren ziehen und das Spiel bei atomaren Formeln enden sollte.

Seien eine Formel φ in NNF und Struktur $\mathcal{A}$ gegeben. Wir definieren nun formal das Spiel $\mathfrak{G}(\mathcal{A}, \varphi)$.

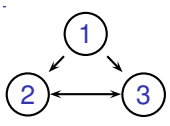
Positionen in $\mathfrak{G}(\mathcal{A}, \varphi)$ sind Paare (ψ, β), wobei $\psi \in \text{sub}(\varphi)$ und $\beta : \mathit{frei}(\psi) \to A$ eine Belegung der freien Variablen von ψ ist.

Die Startposition ist $(\varphi, \{\})$.

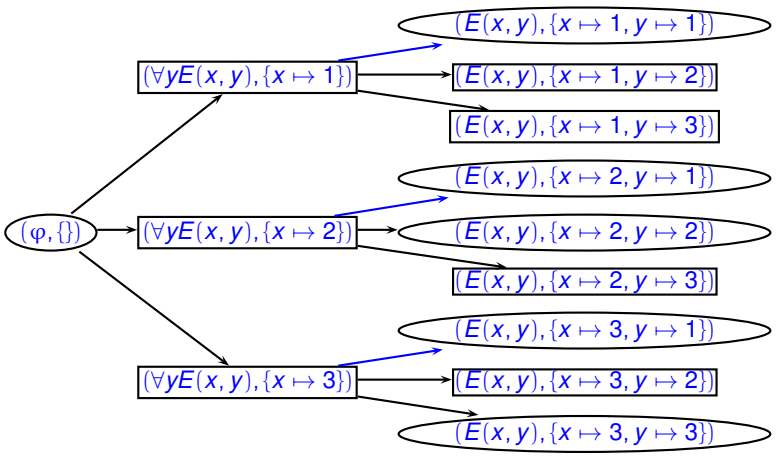
Züge des Spiels. Sei (ψ, β) eine Position.

- Wenn ψ ein Literal ist, endet das Spiel. *Verifizierer* gewinnt gdw. $(\mathcal{A}, \beta) \models \psi$.
- Wenn $\psi := (\vartheta_1 \vee \vartheta_2)$, kann *Verifizierer* zu (ϑ_1, β) oder (ϑ_2, β) ziehen.
- Wenn $\psi := (\vartheta_1 \wedge \vartheta_2)$, kann *Falsifizierer* zu (ϑ_1, β) oder (ϑ_2, β) ziehen.
- Wenn $\psi := \exists x \vartheta(x)$, kann *Verifizierer* für jedes $a \in A$ zur Position $(\vartheta, \beta[x/a])$ ziehen.
- Wenn $\psi := \forall x \vartheta(x)$, kann *Falsifizierer* für jedes $a \in A$ zu $(\vartheta, \beta[x/a])$ ziehen.

Beispiel 6.5 Betrachten wir die Formel $\varphi := \exists x \forall y E(x, y)$ und folgenden Graph $\mathcal{G}$:



Das Spiel $\mathfrak{G}(\mathcal{A}, \varphi)$ sieht wie folgt aus:



Falsifizierer hat eine Gewinnstrategie, in der er von jeder Position für die Formel $\forall y E(x, y)$ zur Nachfolgerposition mit der Belegung $[y/1]$ zieht. ⊣

Bemerkung 6.6 Auswertungsspiele $\mathfrak{G}(\mathcal{A}, \varphi)$ der Prädikatenlogik sind wohl-fundierte endliche Spiele. ⊣

Theorem 6.7 *Für jede Formel* $\varphi \in \text{FO}$ *und jede Struktur* $\mathcal{A}$

$$\mathcal{A} \models \varphi \quad \textit{gdw.} \quad \text{Verifizierer } \textit{gewinnt } \mathfrak{G}(\mathcal{A}, \varphi).$$

Auswertungsspiele bieten eine alternative Sicht auf die Semantik der Formeln. Sie können für viele Logiken definiert werden. Algorithmen zum Lösen von Spielen, wie z.B. in Theorem 6.4, können benutzt werden, um zu entscheiden, ob eine Formel in einer endlichen Struktur gilt.

6.3. Definierbarkeit in der Prädikatenlogik

Ein prädikatenlogischer Satz definiert die Klasse aller Strukturen über der entsprechenden Signatur, in denen der Satz gilt. In diesem Abschnitt befassen wir uns mit der Frage, welche Klassen von Strukturen in der Prädikatenlogik definierbar sind. Um zu zeigen, dass eine Klasse definierbar ist, reicht es einen Satz anzugeben, der sie definiert. Wie aber kann man zeigen, dass es keinen Satz gibt, der eine bestimmte Klasse von Strukturen definiert? Im Prinzip müssen wir ja für jeden einzelnen Satz der Prädikatenlogik über der entsprechenden Signatur zeigen, dass er die Klasse nicht definiert. Eine Methode zur Lösung dieses Problems sind die Ehrenfeucht-Fraïssé-Spiele, die wir im Folgenden betrachten werden.

Als einführendes Beispiel betrachten wir eine Datenbank mit Fluginformationen, die nur eine Tabelle *Flug*(Fluggesellschaft, Flugnummer, Zeit, Start, Ziel) enthält. Wir möchten wissen, ob es eine Möglichkeit gibt, von s nach t zu fliegen (eventuell mit Zwischenstopps). Da für diese Frage die Information über Fluggesellschaften, Flugnummern und über die Zeiten irrelevant ist, vereinfachen wir das Beispiel und betrachten die Tabelle *Flug*(Start, Ziel).

Wir können die Datenbank also als Struktur über der Signatur $\sigma := \{Flug, s, t\}$ auffassen, wobei s und t Konstantensymbole und $Flug$ ein 2-stelliges Relationssymbol ist. Das Relationssymbol $Flug$ beschreibt eine Relation F wobei $(a, b) \in F$ genau dann, wenn es einen direkten Flug von a nach b gibt. Wir suchen einen Satz φ, der die Konstantensymbole s und t und das Relationssymbol $Flug$ benutzt und in einer Struktur wahr ist, wenn es eine Verbindung von s nach t gibt. Zunächst überlegen wir uns, dass für jede feste Zahl von Zwischenstopps so ein Satz existiert:

- Direktflug: $\varphi_0 := Flug(s, t)$,
- Ein Stopp : $\varphi_1 := \exists x \Big(Flug(s, x) \wedge Flug(x, t) \Big)$,
- Zwei Stopps : $\varphi_2 := \exists x_1 \exists x_2 \Big(Flug(s, x_1) \wedge Flug(x_1, x_2) \wedge Flug(x_2, t) \Big)$.

Aber gibt es eine Formel φ, die genau dann in einer solchen Struktur gilt, wenn es eine Möglichkeit gibt, von s nach t zu fliegen, egal wieviele Stopps eingelegt werden müssen? Die Antwort ist „Nein“: Es gibt keinen Satz der Prädikatenlogik, der Erreichbarkeit in diesem Sinne definiert. Um das zu zeigen, müssen wir für jeden Satz der Prädikatenlogik zeigen, dass er Erreichbarkeit nicht definiert. Genauer, müssen wir für jeden Satz $\varphi \in \text{FO}[\{Flug, s, t\}]$ zeigen, dass es zwei σ-Strukturen $\mathcal{A}, \mathcal{B}$ gibt, so dass

(1) es in $\mathcal{A}$ einen Weg von s nach t gibt, in $\mathcal{B}$ aber nicht und

(2) $\mathcal{A} \models \varphi$ und $\mathcal{B} \models \varphi$ oder aber $\mathcal{A} \not\models \varphi$ und $\mathcal{B} \not\models \varphi$.

Wir werden als Nächstes ein Verfahren kennen lernen, das uns diese Arbeit stark vereinfacht. Zunächst treffen wir aber einige Vorbereitungen.

6.3.1. Elementare Äquivalenz

Definition 6.8 Zwei σ-Strukturen $\mathcal{A}, \mathcal{B}$ sind *elementar äquivalent*, geschrieben $\mathcal{A} \equiv \mathcal{B}$, wenn für alle σ-Sätze ψ gilt:

$$\mathcal{A} \models \psi \iff \mathcal{B} \models \psi\,. \qquad \dashv$$

Wenn zwei Strukturen elementar äquivalent sind, erfüllen sie dieselben Sätze der Prädikatenlogik. Wenn wir also zeigen wollen, dass Erreichbarkeit nicht in der Prädikatenlogik definierbar ist, würde es reichen, zwei σ-Strukturen $\mathcal{A}$ und $\mathcal{B}$ zu finden, so dass

- es in $\mathcal{A}$ einen Weg von s nach t (genauer, von $s^{\mathcal{A}}$ nach $t^{\mathcal{A}}$) gibt, in $\mathcal{B}$ aber nicht und
- $\mathcal{A} \equiv \mathcal{B}$.

Solche Strukturen kann es aber nicht geben. Angenommen, es gibt in $\mathcal{A}$ einen Weg von s nach t der Länge m. Dann existiert auch eine Formel φ_m, die die Exstenz dieses Weges ausdrückt und die in $\mathcal{A}$, aber nicht in $\mathcal{B}$ gilt. Dann sind also $\mathcal{A}$ und $\mathcal{B}$ nicht äquivalent. Wir werden daher die elementare Äquivalenz noch verfeinern.

Definition 6.9 Der *Quantorenrang* $qr(\psi)$ einer Formel $\psi \in \text{FO}$ ist induktiv definiert durch:

- $qr(\psi) := 0$ für quantorenfreie Formeln ψ,
- $qr(\neg\psi) := qr(\psi)$,
- $qr((\varphi * \psi)) := \max\{qr(\varphi), qr(\psi)\}$, für $* \in \{\vee, \wedge, \to, \leftrightarrow\}$,
- $qr(\exists x\varphi) = qr(\forall x\varphi) = 1 + qr(\varphi)$. ⊣

Der Quantorenrang einer Formel ist also die maximale Schachtelungstiefe der Quantoren in der Formel.

Beispiel 6.10

- $qr\Big(\exists x \forall y\big(x = y \vee R(x,y,z)\big)\Big) = 2$
- $qr\Big(\exists x\big(T(x) \vee \forall y\, R(x,y,z)\big)\Big) = 2$
- $qr\Big(\exists x\, T(x) \vee \forall y\big(R(y,y,z) \to y = z\big)\Big) = 1$ ⊣

Definition 6.11 Sei $m \in \mathbb{N}$. Zwei σ-Strukturen $\mathcal{A}, \mathcal{B}$ sind *m-äquivalent*, geschrieben $\mathcal{A} \equiv_m \mathcal{B}$, wenn für alle σ-Sätze ψ mit Quantorenrang $qr(\psi) \leqslant m$ gilt:

$$\mathcal{A} \models \psi \iff \mathcal{B} \models \psi\,. \qquad \dashv$$

Wenn zwei Strukturen elementar äquivalent sind, erfüllen sie also dieselben Sätze der Prädikatenlogik bis zum Quantorenrang m. So erlaubt m-Äquivalenz eine feinere Unterscheidung zwischen Strukturen. Offenbar sind Strukturen $\mathcal{A}$ und $\mathcal{B}$, die elementar äquivalent sind, auch m-äquivalent für alle m.

Wir erweitern die Begriffe der Äquivalenz und m-Äquivalenz noch auf Strukturen mit ausgezeichneten Elementen.

Definition 6.12 Seien $\mathcal{A}, \mathcal{B}$ σ-Strukturen und $\overline{a} \in A^k, \overline{b} \in B^k$ k-Tupel von Elementen.

(1) $(\mathcal{A}, \overline{a})$ und $(\mathcal{B}, \overline{b})$ sind *m-äquivalent*, geschrieben $(\mathcal{A}, \overline{a}) \equiv_m (\mathcal{B}, \overline{b})$, wenn für alle σ-Formeln $\psi(\overline{x})$ mit Quantorenrang $qr(\psi) \leqslant m$ und freien Variablen $\overline{x} := x_1, \ldots, x_k$ gilt:

$$\mathcal{A} \models \psi[\overline{a}] \iff \mathcal{B} \models \psi[\overline{b}]\,.$$

(2) $(\mathcal{A}, \overline{a})$ und $(\mathcal{B}, \overline{b})$ sind *elementar äquivalent*, geschrieben $(\mathcal{A}, \overline{a}) \equiv (\mathcal{B}, \overline{b})$, wenn für alle σ-Formeln $\psi(\overline{x})$ und freien Variablen $\overline{x} := x_1, \ldots, x_k$ gilt:

$$\mathcal{A} \models \psi[\overline{a}] \iff \mathcal{B} \models \psi[\overline{b}]\,. \qquad \dashv$$

Definierbarkeit in der Prädikatenlogik. m-Äquivalenz eignet sich besser als elementare Äquivalenz zum Beweis der Nicht-Definierbarkeit bestimmter Aussagen. Wenn wir zeigen wollen, dass Erreichbarkeit nicht in der Prädikatenlogik definierbar ist, reicht es, für alle m zwei σ-Strukturen $\mathcal{A}_m, \mathcal{B}_m$ zu finden, so dass

- es in $\mathcal{A}_m$ einen Weg von s nach t gibt, in $\mathcal{B}_m$ aber nicht und
- $\mathcal{A}_m \equiv_m \mathcal{B}_m$.

Allgemeiner können wir Folgendes zeigen.

Lemma 6.13 *Sei σ eine Signatur und $\mathcal{C}$ eine Klasse von σ-Strukturen. Falls es für alle $m \geqslant 1$ zwei σ-Strukturen $\mathcal{A}_m$ und $\mathcal{B}_m$ gibt, so dass*

- *$\mathcal{A}_m \in \mathcal{C}$, $\mathcal{B}_m \notin \mathcal{C}$ und*
- *$\mathcal{A}_m \equiv_m \mathcal{B}_m$,*

dann gibt es keinen Satz $\varphi \in \text{FO}[\sigma]$, der $\mathcal{C}$ definiert, d.h. es gibt kein $\varphi \in \text{FO}[\sigma]$ mit $\text{Mod}(\varphi) = \mathcal{C}$.

Beweis. Angenommen, es gäbe ein $\varphi \in \text{FO}[\sigma]$ mit $\mathcal{C} = \text{Mod}(\varphi)$. Sei $m := qr(\varphi)$. Dann ist $\mathcal{A}_m \in \mathcal{C}$ und somit $\mathcal{A}_m \models \varphi$. Da aber $\mathcal{A}_m \equiv_m \mathcal{B}_m$ gilt auch $\mathcal{B}_m \models \varphi$, im Widerspruch zu $\mathcal{B}_m \notin \mathcal{C}$. □

6.3.2. Ehrenfeucht-Fraïssé Spiele

Wir werden jetzt eine spieltheoretische Methode kennen lernen, um elementare oder m-Äquivalenz zwischen Strukturen testen zu können. Dazu benötigen wir zunächst ein wenig Notation.

Vereinbarung. Zur Vereinfachung der Notation betrachten wir in diesem Abschnitt nur endliche, relationale Signaturen σ, d.h. endliche Signaturen, in denen nur Relationssymbole vorkommen. Die im Folgenden präsentierten Methoden lassen sich sehr einfach auf endliche Signaturen mit Konstantensymbolen erweitern. Für unendliche Signaturen gelten die Ergebnisse allerdings nicht mehr. Ebenso ändern Funktionssymbole die Situation deutlich, so dass wir hier nicht darauf eingehen wollen.

Definition 6.14 Sei σ eine (relationale) Signatur. Ein *partieller Isomorphismus* zwischen zwei σ-Strukturen $\mathcal{A}, \mathcal{B}$ ist eine injektive Abbildung $h : A' \to B$, wobei $A' \subseteq A$, so dass für alle

- $k \geqslant 0$ und alle
- k-stelligen Relationssymbole $R \in \sigma \cup \{=\}$ und alle
- $a_1, ..., a_k \in A'$

gilt:

$$(a_1, \ldots, a_k) \in R^{\mathcal{A}} \quad \text{gdw.} \quad (h(a_1), ..., h(a_k)) \in R^{\mathcal{B}} . \qquad \dashv$$

Beobachtungen.

- Ein partieller Isomorphismus sagt nur etwas über Teile der Strukturen aus, aber nichts über die Strukturen insgesamt!
- Die leere Abbildung, d.h. die Abbildung mit Definitionsbereich $\varnothing$ ist ein partieller Isomorphismus.
- Ein nicht-leerer partieller Isomorphismus von $\mathcal{A}$ nach $\mathcal{B}$ mit Definitionsbereich $A' \subseteq A$ und Bildbereich $B' \subseteq B$ ist ein Isomorphismus zwischen den von A' und B' induzierten Substrukturen von $\mathcal{A}$ und $\mathcal{B}$.

Beispiel 6.15 Sei $\sigma := \{E\}$ und seien $\mathcal{A}, \mathcal{B}$ wie folgt gegeben:

$\mathcal{A}$: $1 - 2 - 3 - 4$ $\qquad$ $\mathcal{B}$: $\begin{array}{ccc} a & - & b \\ | & \diagup & | \\ c & - & d \end{array}$

Dann ist $\pi : 2 \mapsto a, 3 \mapsto b, 4 \mapsto d$ ein partieller Isomorphismus zwischen $\mathcal{A}$ und $\mathcal{B}$. Die Abbildung definiert durch $\pi : 2 \mapsto a, 3 \mapsto b, 4 \mapsto c$ ist jedoch kein partieller Isomorphismus zwischen $\mathcal{A}$ und $\mathcal{B}$. ⊣

Proposition 6.16 *Sei σ eine relationale Signatur, $\mathcal{A}, \mathcal{B}$ σ-Strukturen und $\overline{a} := a_1, \ldots, a_k \in A^k$ und $\overline{b} := b_1, \ldots, b_k \in B^k$. Dann sind folgende Aussagen äquivalent:*

(1) *Die Abbildung $h : \{a_1, \ldots, a_k\} \to \{b_1, \ldots, b_k\}$ definiert durch $h(a_i) := b_i$ für alle $1 \leqslant i \leqslant k$ ist ein partieller Isomorphismus.*

(2) *Für alle atomaren Formeln $\psi(x_1, \ldots, x_k)$ mit frei$(\psi) \subseteq \{x_1, \ldots, x_k\}$ gilt:*

$$\mathcal{A} \models \psi[\overline{a}] \quad \text{gdw.} \quad \mathcal{B} \models \psi[\overline{b}]\,.$$

(3) *Für alle quantorenfreien Formeln $\psi(x_1, \ldots, x_k)$ mit frei$(\psi) \subseteq \{x_1, \ldots, x_k\}$ gilt:*

$$\mathcal{A} \models \psi[\overline{a}] \quad \text{gdw.} \quad \mathcal{B} \models \psi[\overline{b}]\,.$$

Die Existenz eines partiellen Isomorphismus zwischen Substrukturen $\mathcal{A}'$ von $\mathcal{A}$ und $\mathcal{B}'$ von $\mathcal{B}$ impliziert also $\mathcal{A}' \equiv_0 \mathcal{B}'$, sagt aber nichts darüber aus, ob $\mathcal{A}' \equiv_m \mathcal{B}'$ für $m > 0$ gilt.

Ehrenfeucht-Fraïssé-Spiele. Wir werden jetzt das Ehrenfeucht-Fraïssé-Spiel definieren. Dazu sei σ eine Signatur und seien $m, k \in \mathbb{N}$. Seien ferner $\mathcal{A}$ und $\mathcal{B}$ zwei σ-Strukturen und $\overline{a} := a_1, \ldots, a_k \in A^k$ und $\overline{b} := b_1, \ldots, b_k \in B^k$.

Das *m-Runden Ehrenfeucht-Fraïssé Spiel* $\mathfrak{G}_m(\mathcal{A}, \overline{a}, \mathcal{B}, \overline{b})$ wird von zwei Spielern, dem *Herausforderer* und der *Duplikatorin*, gespielt. Ziel des Herausforderers ist es zu zeigen, dass $(\mathcal{A}, \overline{a}) \not\equiv_m (\mathcal{B}, \overline{b})$. Im Gegenzug versucht die Duplikatorin zu zeigen, dass $(\mathcal{A}, \overline{a}) \equiv_m (\mathcal{B}, \overline{b})$. Zur Vereinfachung der Notation schreiben wir für $k = 0$ einfach $\mathfrak{G}_m(\mathcal{A}, \mathcal{B})$ statt $\mathfrak{G}(\mathcal{A}, (), \mathcal{B}, ())$.

Die Regeln des Spiels sind wie folgt definiert. Eine *Partie* des Spiels besteht aus m Runden. In Runde $i = 1, ..., m$

- wählt zunächst der Herausforderer entweder ein Element $a'_i \in A$ oder $b'_i \in B$.
- Danach antwortet die Duplikatorin. Hat der Herausforderer ein $a'_i \in A$ gewählt, wählt nun die Duplikatorin ein $b'_i \in B$. Anderenfalls wählt sie $a'_i \in A$.

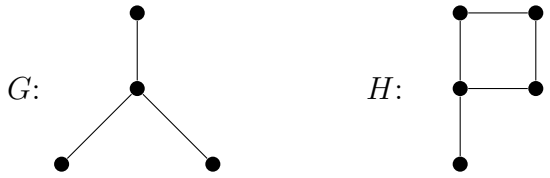
Nach Runde m wird der Gewinner ermittelt: Die Duplikatorin hat gewonnen, wenn die Abbildung

$$h : a_1 \mapsto b_1, \ldots, a_k \mapsto b_k, a'_1 \mapsto b'_1, \ldots, a'_m \mapsto b'_m$$

ein partieller Isomorphismus von $\mathcal{A}$ nach $\mathcal{B}$ ist.

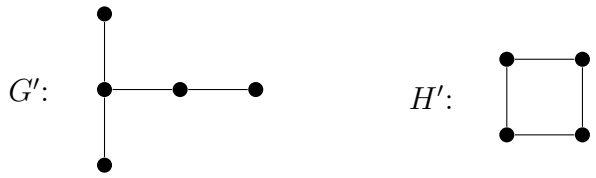
Beispiel 6.17

(1) *Herausforderer* gewinnt $\mathfrak{G}_2(G, H)$ für die Graphen



indem er in Runde 1 den mittleren Knoten a_1 in G wählt. In Runde 2 wählt der dann einen Knoten b_2 in H, der nicht zu Knoten b_1 benachbart ist.

(2) *Duplikator* gewinnt $\mathfrak{G}_2(G', H')$ für die Graphen



denn in beiden Graphen gibt es zu jedem Knoten sowohl einen Nachbarn als auch einen Nicht-Nachbarn.

(3) Sei $\mathcal{A} := ([0, 7], \leqslant^{\mathcal{A}})$ und $\mathcal{B} := ([0, 8], \leqslant^{\mathcal{B}})$, wobei $\leqslant^{\mathcal{A}}$ bzw. $\leqslant^{\mathcal{B}}$ die natürlichen linearen Ordnungen auf $[0, 7]$ bzw. $[0, 8]$ sind (mit $[0, 7], [0, 8] \subset \mathbb{N}$). Dann gewinnt die Duplikatorin das Spiel $\mathfrak{G}_3(\mathcal{A}, \mathcal{B})$ aber der Herausforderer gewinnt $\mathfrak{G}_4(\mathcal{A}, \mathcal{B})$. ⊣

Das letzte Beispiel ist ein Spezialfall der folgenden Aussage.

Theorem 6.18 *Seien $\mathcal{A}$ und $\mathcal{B}$ endliche $\{\leqslant\}$-Strukturen mit Universum A bzw. B, wobei $\leqslant^{\mathcal{A}}, \leqslant^{\mathcal{B}}$ lineare Ordnungen auf dem jeweiligen Universum sind. Dann gilt für alle $m \in \mathbb{N}$:*

Die Duplikatorin gewinnt das Spiel $\mathfrak{G}_m(\mathcal{A}, \mathcal{B}) \iff |A| = |B|$ oder $|A|, |B| \geqslant 2^m$.

Beweis. Wir zeigen zunächst die Rückrichtung. Wenn $|A| = |B|$, dann sind $\mathcal{A} \cong \mathcal{B}$. Die Duplikatorin gewinnt $\mathfrak{G}_m(\mathcal{A}, \mathcal{B})$ dann, indem sie die Züge des Herausforderers einfach „kopiert".

Sei daher $|A|, |B| \geqslant 2^m$. Für $\mathcal{C} \in \{\mathcal{A}, \mathcal{B}\}$ mit Universum C sei die *Distanzfunktion* $dist\colon C \times C \to \mathbb{N}$ definiert durch

$$dist(c, c') \;:=\; |\{d \in C : c <^{\mathcal{C}} d \leqslant^{\mathcal{C}} c' \text{ oder } c' <^{\mathcal{C}} d \leqslant^{\mathcal{C}} c\}|.$$

Wir definieren weiterhin für jedes Element $a \in A$ die Distanz $dist_s(a) := |\{a' \in A : a' < a\}|$ zum minimalen Element von $\leqslant^{\mathcal{A}}$ und die Distanz $dist_l(a) := |\{a' \in A : a < a'\}|$ zum maximalen Element in $\leqslant^{\mathcal{A}}$. Analog sind $dist_s(b)$, $dist_l(b)$ für Elemente $b \in B$ definiert.

Wir zeigen per Induktion über i, dass die Duplikatorin so spielen kann, dass für jedes $i \in [1, m]$ gilt:

Sind $a_1, \ldots, a_i$ und $b_1, \ldots, b_i$ die in den Runden $1, \ldots, i$ gewählten Elemente in $\mathcal{A}$ und $\mathcal{B}$ und sind a_s, b_s und a_l, b_l die jeweils kleinsten und größten Elemente aus $a_1, \ldots, a_i$ und $b_1, \ldots b_i$ dann gilt für alle $j, j' \in [1, i]$:

(1) $a_j <^{\mathcal{A}} a_{j'} \iff b_j <^{\mathcal{B}} b_{j'}$,

(2) $dist(a_j, a_{j'}) = dist(b_j, b_{j'})$ oder $dist(a_j, a_{j'}), dist(b_j, b_{j'}) \geqslant 2^{m-i}$,

(3) $dist_s(a_s) = dist_s(b_s)$ oder $dist_s(a_s), dist_s(b_s) \geqslant 2^{m-i}$ und

(4) $dist_l(a_l) = dist_l(b_l)$ oder $dist_l(a_l), dist_l(b_l) \geqslant 2^{m-i}$.

$(*)_i$

Man beachte, dass $(*)_i$ impliziert, dass $a_1, \ldots, a_i \mapsto b_1, \ldots, b_i$ ein partieller Isomorphismus von $\mathcal{A}$ nach $\mathcal{B}$ ist.

$i = 0$: Für $i = 0$ ist nichts zu zeigen.

$i \to i + 1$: Es gelte $(*)_i$ und seien $a_1, \ldots, a_i$ und $b_1, \ldots, b_i$ die in den Runden $1, \ldots, i$ gewählten Elemente in $\mathcal{A}$ und $\mathcal{B}$. Weiterhin seien a_s, b_s und a_l, b_l die jeweils kleinsten und größten Elemente aus $a_1, \ldots, a_i$ und $b_1, \ldots b_i$. Im Folgenden betrachten wir nur den Fall, dass der Herausforderer in Runde $i + 1$ ein Element a_{i+1} in $\mathcal{A}$ wählt. Den anderen Fall (dass der Herausforderer ein Element in $\mathcal{B}$ wählt) kann analog behandelt werden.

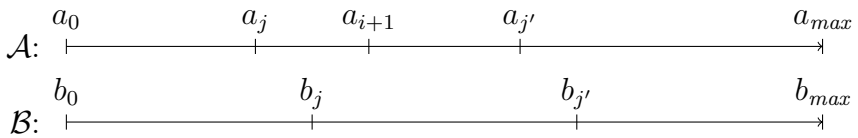
Falls $a_{i+1} = a_j$ für ein $j \in [1, i]$, so antwortet Duplikator mit $b_{i+1} := b_j$ und erfüllt damit $(*)_{i+1}$.

Wir betrachten als nächstes den Fall, dass der Herausforder „zwischen" zwei schon gezogene Elemente zieht. Seien also $j, j' \in [1, i]$ so, dass $a_j <^{\mathcal{A}} a_{i+1} <^{\mathcal{A}} a_{j'}$ und kein $j'' \in [1, i]$ existiert mit $a_j <^{\mathcal{A}} a_{j''} <^{\mathcal{A}} a_{j'}$. Wegen $(*)_i$ gilt

(1) $b_j <^{\mathcal{B}} b_{j'}$ und es gibt kein $j'' \in [1, i]$ mit $b_j <^{\mathcal{B}} b_{j''} <^{\mathcal{B}} b_{j'}$ und

(2) $dist(a_j, a_{j'}) = dist(b_j, b_{j'})$ oder $dist(a_j, a_{j'}), dist(b_j, b_{j'}) \geqslant 2^{m-i}$.

Die folgende Abbildung skizziert diese Situation.



Fall 1: $dist(a_j, a_{j'}) = dist(b_j, b_{j'})$.
Dann wählt die Duplikatorin ein $b_{i+1} \in B$ mit $b_j <^{\mathcal{B}} b_{i+1} <^{\mathcal{B}} b_{j'}$, $dist(b_j, b_{i+1}) = dist(a_j, a_{i+1})$ und $dist(b_{i+1}, b_{j'}) = dist(a_{i+1}, a_{j'})$. Offensichtlich ist $(*)_{i+1}$ dann erfüllt.

Fall 2: $dist(a_j, a_{j'}), dist(b_j, b_{j'}) \geqslant 2^{m-i}$.
Dann existiert mindestens ein $c \in B$ mit $b_j <^{\mathcal{B}} c <^{\mathcal{B}} b_{j'}$, $dist(b_j, c) \geqslant 2^{m-i}/2 = 2^{m-(i+1)}$ und $dist(c, b_{j'}) \geqslant 2^{m-i}/2 = 2^{m-(i+1)}$. Wähle solch ein c. Die Duplikatorin verfährt in Runde $i+1$ wie folgt:

- Falls $dist(a_j, a_{i+1}), dist(a_{i+1}, a_{j'}) \geqslant 2^{m-(i+1)}$, so wählt sie $b_{i+1} := c$.
- Falls $dist(a_j, a_{i+1}) < 2^{m-(i+1)}$, so wählt sie dasjenige $b_{i+1} \in B$, so dass $b_j <^{\mathcal{B}} b_{j+1}$ und $dist(b_j, b_{i+1}) = dist(a_j, a_{i+1})$.
- Falls $dist(a_{i+1}, a_{j'}) < 2^{m-(i+1)}$, so wählt sie dasjenige $b_{i+1} \in B$, so dass $b_{i+1} <^{\mathcal{B}} b_{j'}$ und $dist(b_{i+1}, b_{j'}) = dist(a_{i+1}, a_{j'})$.

Man kann leicht nachprüfen, dass in jedem Fall $(*)_{i+1}$ erfüllt ist.

Es bleiben die beiden Fälle zu betrachten, in denen der Herausforder ein Element $a_{i+1}) < a_s$ oder ein Element $a_{i+1} > a_l$ wählt. Die beiden Fälle sind völlig symmetrisch, so dass wir nur den Fall $a_{i+1} < a_s$ betrachten. Wegen $(*)_i$ gilt also $dist_s(a_s) = dist_s(b_s)$ oder $dist_s(a_s), dist_s(b_s) \geqslant 2^{m-i}$. Falls $dist_s(a_s) = dist_s(b_s)$, so wählt die Duplikatorin das Element b_{i+1}) mit $dist_s(b_{i+1}) = dist_s(a_{i+1})$. Die Wahl erfüllt offensichtlich $(*)_{i+1}$). Anderenfalls, d.h. $dist_s(a_s), dist_s(b_s) \geqslant 2^{m-i}$, wählt sie b_{i+1} wie folgt. Gilt $dist_s(a_{i+1}) < 2^{m-(i+1)}$, so wählt sie b_{i+1} so, dass $dist_s(b_{i+1}) = dist_s(a_{i+1})$. Anderenfalls wählt sie das Element b_{i+1} mit $dist_s(b_{i+1}) = 2^{m-(i+1)}$. In beiden Fällen ist $(*)_{i+1}$ erfüllt.

Es folgt, dass die Duplikatorin so spielen kann, dass $(*)_m$ gilt. Daher gewinnt sie $\mathfrak{G}_m(\mathcal{A}, \mathcal{B})$.

Zum Beweis der Hinrichtung genügt folgendes zu zeigen: Falls $|A| < |B|$ und $|A| < 2^m$, so hat *Herausforderer* eine Gewinnstrategie in $\mathfrak{G}_m(\mathcal{A}, \mathcal{B})$. Dies kann man beweisen, indem man zeigt, dass Herausforderer so spielen kann, dass für jedes $i \in$

$[1, m]$ gilt:

Sind $a_1, \ldots, a_i$ und $b_1, \ldots, b_i$ die in den Runden $1, \ldots, i$ gewählten Elemente in $\mathcal{A}$ und $\mathcal{B}$ und sind a_s, b_s und a_l, b_l die jeweils kleinsten und größten Elemente aus $a_1, \ldots, a_i$ und $b_1, \ldots, b_i$, so gilt $dist_s(a_s) < 2^{m-i}$ und $dist_s(a_s) < dist_s(b_s)$ oder $dist_l(a_l) < 2^{m-i}$ und $dist_l(a_l) < dist_l(b_l)$ oder es gibt $j, j' \in [1, i]$, so dass $(**)_i$

(1) $a_j <^{\mathcal{A}} a_{j'}$ und $b_j \geqslant^{\mathcal{B}} b_{j'}$,

(2) $a_j \geqslant^{\mathcal{A}} a_{j'}$ und $b_j <^{\mathcal{B}} b_{j'}$ oder

(3) $dist(a_j, a_{j'}) < 2^{m-i}$ und $dist(a_j, a_{j'}) < dist(b_j, b_{j'})$.

Details: *Übung.* □

Eine wichtige Variante des m-Runden-Ehrenfeucht-Fraïssé-Spiels ist das Spiel $\mathfrak{G}(\mathcal{A}, \overline{a}, \mathcal{B}, \overline{b})$ mit unbeschränkter Zugzahl. Hier wählt der Herausforderer zunächst eine Zahl $m \geqslant 0$ und dann wird das m-Runden Spiel $\mathfrak{G}_m(\mathcal{A}, \overline{a}, \mathcal{B}, \overline{b})$ gespielt.

Theorem 6.19 (Satz von Ehrenfeucht) *Sei σ eine endliche, relationale Signatur und sei $k \in \mathbb{N}$. Seien ferner $\mathcal{A}$ und $\mathcal{B}$ zwei σ-Strukturen mit Universum A bzw. B seien und $\overline{a} := a_1, \ldots, a_k \in A^k$ und $\overline{b} := b_1, \ldots, b_k \in B^k$.*

(1) *Folgende Aussagen sind äquivalent:*
   (a) $(\mathcal{A}, \overline{a}) \equiv (\mathcal{B}, \overline{b})$.
   (b) *Die Duplikatorin gewinnt $\mathfrak{G}(\mathcal{A}, \overline{a}, \mathcal{B}, \overline{b})$.*

(2) *Für alle $m \geqslant 0$ sind folgende Aussagen sind äquivalent:*
   (a) $(\mathcal{A}, \overline{a}) \equiv_m (\mathcal{B}, \overline{b})$.
   (b) *Die Duplikatorin gewinnt $\mathfrak{G}_m(\mathcal{A}, \overline{a}, \mathcal{B}, \overline{b})$.*

Zum Beweis des Satzes benötigen wir zunächst einige Hilfsmittel.

Definition 6.20 (m-Isomorphietypen oder Hintikka-Formeln.) Wir definieren induktiv sogenannte *Hintikka-Formeln* $\varphi^m_{\mathcal{A},\overline{a}}(\overline{x})$, die beschreiben, welche Formeln mit Quantorenrang höchstens m in der Struktur $\mathcal{A}$ mit Konstanten $\overline{a}$ gelten. Sei σ eine Signatur und sei $k \in \mathbb{N}$. Sei ferner $\mathcal{A}$ eine σ-Struktur, $\overline{a} := a_1, \ldots, a_k \in A^k$ und $\overline{x} := x_1, \ldots, x_k$. Wir definieren

$$\varphi^0_{\mathcal{A},\overline{a}}(\overline{x}) := \bigwedge \Big\{ \psi(\overline{x}) : \psi \text{ ist atomar oder negiert atomar und } \mathcal{A} \models \psi[\overline{a}] \Big\}$$

und für $m \geqslant 0$

$$\varphi^{m+1}_{\mathcal{A},\overline{a}}(\overline{x}) := \bigwedge_{a \in A} \exists x_{k+1} \varphi^m_{\mathcal{A},\overline{a},a}(\overline{x}, x_{k+1}) \wedge \forall x_{k+1} \bigvee_{a \in A} \varphi^m_{\mathcal{A},\overline{a},a}(\overline{x}, x_{k+1}) \,. \qquad \dashv$$

Man beachte, dass der Quantorenrang einer Formel $\varphi^m_{\mathcal{A},\overline{a}}(\overline{x})$ genau m ist.

Lemma 6.21 *Sei σ eine Signatur und sei $k \in \mathbb{N}$. Seien ferner $\mathcal{A}$ und $\mathcal{B}$ zwei σ-Strukturen und $\overline{a} := a_1, \ldots, a_k \in A^k$ und $\overline{b} := b_1, \ldots, b_k \in B^k$. Für alle $m \geqslant 0$ sind folgende Aussagen äquivalent:*

(1) *Die Duplikatorin gewinnt $\mathfrak{G}_m(\mathcal{A}, \overline{a}, \mathcal{B}, \overline{b})$.*

(2) $\mathcal{B} \models \varphi^m_{\mathcal{A},\overline{a}}[\overline{b}]$.

(3) $(\mathcal{A}, \overline{a}) \equiv_m (\mathcal{B}, \overline{b})$.

Beweis.

Wir zeigen als erstes, dass *„3 $\Longrightarrow$ 2“*. Gelte $(\mathcal{A}, \overline{a}) \equiv_m (\mathcal{B}, \overline{b})$. Zusammen mit $\mathcal{A} \models \varphi^m_{\mathcal{A},\overline{a}}[\overline{a}]$ und $qr(\varphi^m_{\mathcal{A},\overline{a}}) = m$ folgt dann sofort $\mathcal{B} \models \varphi^m_{\mathcal{A},\overline{a}}[\overline{b}]$.

Als nächstes zeigen wir die Äquivalenz *„2 $\Longleftrightarrow$ 1“*. Der Beweis folgt per Induktion über m. Für $m = 0$ gilt: Die Duplikatorin gewinnt $\mathfrak{G}_0(\mathcal{A}, \overline{a}, \mathcal{B}, \overline{b})$ genau dann, wenn $\overline{a} \mapsto \overline{b}$ ein partieller Isomorphismus von $\mathcal{A}$ nach $\mathcal{B}$ ist, was genau dann der Fall ist, wenn $\mathcal{B} \models \varphi^0_{\mathcal{A},\overline{a}}[\overline{b}]$.

Für $m > 0$ gilt:

Die Duplikatorin gewinnt $\mathfrak{G}_m(\mathcal{A}, \overline{a}, \mathcal{B}, \overline{b})$

$\Longleftrightarrow$ für alle $a \in A$ gibt es ein $b \in B$, so dass die Duplikatorin $\mathfrak{G}_{m-1}(\mathcal{A}, \overline{a}a, \mathcal{B}, \overline{b}b)$ gewinnt und
für alle $b \in B$ gibt es ein $a \in A$, so dass die Duplikatorin $\mathfrak{G}_{m-1}(\mathcal{A}, \overline{a}a, \mathcal{B}, \overline{b}b)$ gewinnt

$\overset{\text{I.V.}}{\Longleftrightarrow}$ für alle $a \in A$ gibt es ein $b \in B$, so dass $\mathcal{B} \models \varphi^{m-1}_{\mathcal{A},\overline{a}a}[\overline{b}b]$ und
für alle $b \in B$ gibt es ein $a \in A$, so dass $\mathcal{B} \models \varphi^{m-1}_{\mathcal{A},\overline{a}a}[\overline{b}b]$

$$\Longleftrightarrow \mathcal{B} \models \left(\bigwedge_{a \in A} \exists x_{k+1}\, \varphi^{m-1}_{\mathcal{A},\overline{a}a}(\overline{x}, x_{k+1}) \;\wedge\; \forall x_{k+1} \bigvee_{a \in A} \varphi^{m-1}_{\mathcal{A},\overline{a}a}(\overline{x}, x_{k+1}) \right) [\overline{b}]$$

$$\Longleftrightarrow \mathcal{B} \models \varphi^m_{\mathcal{A},\overline{a}}[\overline{b}].$$

Als letztes zeigen wir per Induktion über m die Implikation *„1 $\Longrightarrow$ 3“*.

Für $m = 0$ ist der Beweis wie bei „2 $\Longleftrightarrow$ 1“. Sei nun $m > 0$. Angenommen, die Duplikatorin gewinnt $\mathfrak{G}_m(\mathcal{A}, \overline{a}, \mathcal{B}, \overline{b})$. Sei $\psi(\overline{x}')$ eine FO-Formel mit $qr(\psi) \leqslant m$, wobei $\overline{x}' = (x_{i_1}, \ldots, x_{i_n})$ und $i_1, \ldots, i_n \in \{1, \ldots, k\}$. Wir müssen zeigen, dass

$$\mathcal{A} \models \psi[\overline{a}'] \iff \mathcal{B} \models \psi[\overline{b}']. \tag{$*$}$$

wobei $\overline{a}' := (a_{i_1}, \ldots, a_{i_n})$ und $\overline{b}' := (b_{i_1}, \ldots, b_{i_n})$.

Falls $qr(\psi) < m$, so folgt $(*)$ direkt aus der Induktionsvoraussetzung. Da zudem die Menge der Formeln ψ, die $(*)$ erfüllen, unter Booleschen Kombinationen abgeschlossen ist, können wir o.B.d.A. annehmen, dass

$$\psi \;=\; \exists x_{k+1}\, \chi(\overline{x}', x_{k+1}),$$

wobei $qr(\chi) = m - 1$.

Angenommen, $\mathcal{A} \models \psi[\overline{a}']$. Dann gibt es ein $a \in A$ mit $\mathcal{A} \models \chi[\overline{a}', a]$. Wähle ein solches a. Da die Duplikatorin $\mathfrak{G}_m(\mathcal{A}, \overline{a}, \mathcal{B}, \overline{b})$ gewinnt, gibt es ein $b \in B$, so dass die Duplikatorin $\mathfrak{G}_{m-1}(\mathcal{A}, \overline{a}a, \mathcal{B}, \overline{b}b)$ gewinnt. Gemäß Induktionsannahme gilt also

$$\mathcal{A} \models \chi[\overline{a}', a] \iff \mathcal{B} \models \chi[\overline{b}', b].$$

Mit $\mathcal{A} \models \chi[\overline{a}', a]$ folgt daraus $\mathcal{B} \models \chi[\overline{b}', b]$ und daher $\mathcal{B} \models \psi[\overline{b}']$. Die Richtung „$\Longleftarrow$“ in $(*)$ kann man analog beweisen. □

Bemerkung 6.22 Sind $\mathcal{A}$ und $\mathcal{B}$ zwei Strukturen, die sich durch einen FO-Satz φ mit $qr(\varphi) = m$ unterscheiden lassen (also z.B. $\mathcal{A} \models \varphi$ und $\mathcal{B} \not\models \varphi$), so gibt es gemäß Satz 6.19 und Bemerkung 6.6 eine Gewinnstrategie für den Herausforderer in $\mathfrak{G}_m(\mathcal{A}, \mathcal{B})$. Der Satz φ gibt sogar direkt eine solche Gewinnstrategie an: Der Herausforderer gewinnt das Spiel, indem er Elemente in $\mathcal{A}$ wählt, die den $\exists$-Quantoren in φ entsprechen, und Elemente in $\mathcal{B}$, die den $\forall$-Quantoren in φ entsprechen.

Seien beispielsweise $\mathcal{G}$ und $\mathcal{H}$ die Graphen aus Beispiel 6.17(1) mit Universum G bzw. H und

$$\varphi \;:=\; \exists x \forall y \big(x = y \vee E(x, y)\big).$$

Dann gilt $\mathcal{G} \models \varphi$ und $\mathcal{H} \not\models \varphi$. Wegen $\mathcal{G} \models \varphi$ kann der Herausforderer in Runde 1 ein $a_1 \in G$ wählen, so dass

$$\mathcal{G} \models \forall y \big(x = y \vee E(x, y)\big)[a_1].$$

Da $\mathcal{H} \not\models \varphi$, muss für *jede* beliebige Antwort $b_1 \in H$ der Duplikatorin gelten:

$$\mathcal{H} \models \exists y \neg\big(x = y \vee E(x, y)\big)[b_1].$$

In der zweiten Runde kann der Herausforderer daher ein Element $b_2 \in H$ auswählen, für das gilt:

$$\mathcal{H} \models \neg\big(x = y \vee E(x, y)\big)[b_1, b_2].$$

Für jede mögliche Antwort $a_2 \in G$, die die Duplikatorin geben kann, gilt

$$\mathcal{G} \models \big(x = y \vee E(x, y)\big)[a_1, a_2].$$

Daher kann die Abbildung $a_1, a_2 \mapsto b_1, b_2$ kein partieller Isomorphismus sein. Die Duplikatorin hat die Partie also verloren. ⊣

Bemerkung 6.23 Sei σ eine Signatur und $m \in \mathbb{N}$. Aus Satz 6.19 und Lemma 6.21 folgt, dass $\equiv_m$ nur *endlich* viele Äquivalenzklassen auf der Klasse aller σ-Strukturen hat. Die Äquivalenzklasse, zu der eine gegebene Struktur $\mathcal{A}$ gehört, wird dabei durch den Satz $\varphi^m_{\mathcal{A}}$ definiert. Da die Modellklasse eines FO$[\sigma]$-Satzes vom Quantorenrang m auf der Klasse aller Strukturen eine Vereinigung von $\equiv_m$-Äquivalenzklassen ist, folgt daraus insbesondere, dass für jedes m und jede Signatur σ nur *endlich* viele nicht-äquivalente FO$[\sigma]$-Formeln vom Quantorenrang m existieren. (Man beachte dabei, dass wir hier nur endliche Signaturen betrachten.) ⊣

Ehrenfeucht-Fraïssé-Spiele bieten eine sehr elegante Methode um die Nicht-Definierbarkeit bestimmter Klassen von Strukturen zu beweisen. Wir werden dies anhang einiger Beispiele demonstrieren. Dazu brauchen wir zunächst noch ein wenig Notation.

Definition 6.24 Sei σ eine Signatur und $\mathcal{K}$ eine Klasse von σ-Strukturen. Eine Klasse $\mathcal{C} \subseteq \mathcal{K}$ ist *in* $\mathcal{K}$ FO-*definierbar*, wenn es einen Satz $\psi \in \mathrm{FO}[\sigma]$ gibt, so dass

$$\mathcal{C} \quad = \quad \{\mathcal{A} \in \mathcal{K} : \mathfrak{A} \models \psi\}\,. \qquad \dashv$$

Der Satz von Ehrenfeucht liefert nun sofort folgendes Korollar.

Korollar 6.25 *Sei σ eine endliche, relationale Signatur, $\mathcal{K}$ eine Klasse von σ-Strukturen und $\mathcal{C} \subseteq \mathcal{K}$. Dann sind folgende Aussagen äquivalent.*

(1) *$\mathcal{C}$ ist nicht in $\mathcal{K}$ FO-definierbar.*

(2) *Für alle $m \in \mathbb{N}$ gibt es $\mathcal{A}_m \in \mathcal{C}$ und $\mathcal{B}_m \in \mathcal{K} \setminus \mathcal{C}$ mit $\mathcal{A}_m \equiv_m \mathcal{B}_m$.*

(3) *Für alle $m \in \mathbb{N}$ gibt es $\mathcal{A}_m \in \mathcal{C}$ und $\mathcal{B}_m \in \mathcal{K} \setminus \mathcal{C}$, so dass die Duplikatorin das Spiel $\mathfrak{G}_m(\mathcal{A}_m, \mathcal{B}_m)$ gewinnt.*

Als Anwendung des Korollars zeigen wir nun einige Beispiele von Klassen von Strukturen, die nicht FO-definierbar sind.

Theorem 6.26 *Sei $\sigma := \{<\}$ und $\mathcal{K}$ die Klasse aller endlichen linearen Ordnungen. Die Klasse $\mathcal{C} \subseteq \mathcal{K}$ der endlichen linearen Ordnungen gerader Länge ist in $\mathcal{K}$ nicht* FO-*definierbar.*

Beweis. Wir haben schon gesehen, dass wenn $\mathcal{A} := (A, <^{\mathcal{A}}), \mathcal{B} := (B, <^{\mathcal{B}})$ zwei endliche lineare Ordnungen sind, so gewinnt die Duplikatorin das Spiel $\mathfrak{G}_m(\mathcal{A}, \mathcal{B})$ genau dann, wenn $|A| = |B|$ oder aber $|A|, |B| \geqslant 2^m$.

Wir brauchen also für $m \geqslant 0$ nur $\mathcal{A}_m$ als lineare Ordnung der Länge 2^m und $\mathcal{B}_m$ als lineare Ordnung der Länge $2^m + 1$ zu wählen.

Dann gilt $\mathcal{A}_m \equiv_m \mathcal{B}_m$ aber $\mathcal{A}_m \in \mathcal{C}$ und $\mathcal{B}_m \in \mathcal{K} \setminus \mathcal{C}$. □

Betrachten wir noch einmal das Beispiel vom Anfang des Abschnitts. Sei $\sigma := \{E, S, T\}$, wobei E ein 2-stelliges Relationssymbol ist und S, T einstellige Relationssymbole sind.

Sei $\mathcal{K}$ die Klasse aller σ-Strukturen, in denen S, T durch Relationen interpretiert werden, die nur ein einziges Element enthalten. Sei *Reach* $\subseteq \mathcal{K}$ die Klasse aller σ-Strukturen, in denen es einen Pfad vom Element in S zum Element in T gibt.

Theorem 6.27 *Reach ist nicht* FO-*definierbar in $\mathcal{K}$.*

Beweis. Für jedes $m \geqslant 0$ sei $\mathcal{A}_m$ ein Kreis der Länge 2^{m+1} und $\mathcal{B}_m$ die disjunkte Vereinigung zweier Kreise der Länge 2^m. In $\mathcal{A}_m$ wählen wir für $S^{\mathcal{A}}$ und $T^{\mathcal{A}}$ zwei Elemente, die maximalen Abstand voneinander haben, d.h. sich auf dem Kreise „gegenüber“ liegen. In $\mathcal{B}$ wählen wir für $S^{\mathcal{B}}$ ein Element des einen Kreises und fuur $T^{\mathcal{B}}$ ein Element des anderen Kreises aus. Ähnlich zum Beweis von Satz 6.18, kann man nun leicht zeigen, dass die Duplikatorin das Spiel $\mathfrak{G}_m(\mathcal{A}_m, \mathcal{B}_m)$ gewinnt, obschon $\mathcal{A}_m \in$ *Reach* aber $\mathcal{B}_m \notin$ *Reach*. Die Details sind dem Leser und der Leserin zur Übung überlassen. □

6.4. Sequenzenkalkül

Im Abschnitt 5.4 über Modellklassen und Theorien haben wir gesehen, dass z.B. die Klasse aller Gruppen FO-definierbar ist. Um zu entscheiden, ob eine in der Prädikatenlogik formalisierbare Aussage φ über Gruppen in allen Gruppen gilt, muss man also im Prinzip nur zeigen, dass φ aus den Gruppenaxiomen logisch folgt. Wir brauchen dazu also Methoden, die es uns erlauben, logische Folgerungen der Art $\Phi \models \varphi$ herleiten zu können. Die gleiche Anwendung stellt sich natürlich auch in der Aussagenlogik. Dieser Abschnitt ist daher der Frage gewidmet, wie man logisches Schließen formalisieren kann. Wir werden zunächst einen Kalkül für die Aussagenlogik vorstellen, der es erlaubt, logische Folgerungen herleiten zu können. Den Kalkül werden wir dann auf die Prädikatenlogik erweitern. Dazu betrachten wir zunächst noch einmal das einfache Beispiel ganz zu Anfang dieses Skriptes.

Beispiel 6.28 Gegeben sei eine Menge von Voraussetzungen: {„Wenn der Zug zu spät ist und es keine Taxis am Bahnhof gibt, kommt Peter zu spät zu seinem Termin“, „Peter kam nicht zu spät”, „Der Zug war zu spät“}. Können wir daraus herleiten, dass „es Taxis gab”? ⊣

Allgemeiner kann man fragen, ob es Verfahren gibt, um solche Schlüsse aus einer Menge von Voraussetzungen zu ziehen. Wir suchen also eine Methode,

(1) mit der nur korrekte Schlüsse gezogen werden können und

(2) die allgemein genug ist, alle gültigen Schlüsse ziehen zu können.

Eine solche Methode ist die Resolution. Wir werden jetzt eine weitere Methode kennen lernen, den *Sequenzenkalkül*. Der Sequenzenkalkül formalisiert die Art, in der Mathematiker oder Logiker aus Aussagen Folgerungen ziehen.

Wir führen dazu zunächst den Begriff der *Sequenz* ein, d.h. Aussagen der Form

$$\underbrace{\left\{\begin{array}{c}\text{„Zug zu spät und keine Taxis impliziert Peter zu spät“,}\\ \text{„Zug zu spät“,}\\ \text{„Peter nicht zu spät“}\end{array}\right\}}_{\text{Voraussetzung}} \Rightarrow \underbrace{\text{„es gab Taxis“}}_{\text{Konklusion}}$$

Die Bedeutung einer Sequenz ist, dass aus den Voraussetzungen die Konklusion geschlossen werden kann. Wir werden auch unendliche Mengen von Voraussetzungen zulassen, interessieren uns jedoch meistens nur für eine Konklusion. In der Mitte eines Beweises ist es aber oft nützlich, verschiedene Möglichkeiten zu betrachten, also Hypothesen aufzustellen. D.h. wir werden zwischenzeitlich mehr als eine Folgerung betrachten.

Zum Beispiel könnten wir in dem Zugbeispiel als Zwischenschritt annehmen

$$\underbrace{\left\{\begin{array}{c}\text{„Zug zu spät und keine Taxis impliziert Peter zu spät“},\\ \text{„Zug zu spät“},\\ \text{„Peter nicht zu spät“}\end{array}\right\}}_{\text{Voraussetzung}} \Rightarrow \underbrace{\begin{array}{c}\text{„es gab Taxis“},\\ \text{„Peter zu spät“}\end{array}}_{\text{Konklusion}}$$

Die Bedeutung ist, dass wenn die Voraussetzungen gelten, *mindestens eine Konklusion* gilt. Wir können dann aus der Voraussetzung „Peter nicht zu spät“ die eigentliche Folgerung ableiten.

Wir stellen zunächst einen Sequenzenkalkül für die Aussagenlogik vor und erweitern ihn später auf die Prädikatenlogik. Es gibt viele verschiedene Sequenzenkalküle, abhängig von den in der Logik verwendeten Verknüpfungen. Wir werden hier einen Kalkül für Formeln ohne $\leftrightarrow$ vorstellen. Wie bereits gezeigt, ist dies keine Einschränkung der Allgemeinheit.

6.4.1. Sequenzenkalkül der Aussagenlogik

Notation Im Folgenden benutzen wir Φ, Δ für Mengen von Formeln. Dabei wird Δ immer endlich sein, Φ kann endlich oder unendlich sein. Wenn Φ und Δ endlich sind, schreiben wir

(1) $\bigwedge \Phi$ für die Konjunktion $\varphi_1 \wedge \cdots \wedge \varphi_n$ aller Formeln in Φ und

(2) $\bigvee \Delta$ für die Disjunktion $\psi_1 \vee \cdots \vee \psi_m$ aller Formeln in Δ.

Definition 6.29 Eine *Sequenz* ist eine Aussage der Form

$$\Phi \Rightarrow \Delta.$$

Wir nennen Φ die *Voraussetzungen* und Δ die *Konklusionen* (bisweilen auch *Antezedenz* und *Sukzedenz*). Eine Sequenz $\Phi \Rightarrow \Delta$ ist *gültig*, wenn jede Belegung β, die alle Formeln in Φ erfüllt, mindestens eine Formel in Δ erfüllt. Ist $\Phi \Rightarrow \Delta$ nicht gültig, so gibt es eine Belegung β die alle Formeln in Φ erfüllt, aber keine in Δ. Wir sagen, dass β *die Sequenz falsifiziert*. ⊣

Beispiel 6.30 (1) Die folgende Sequenz ist gültig:

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

(2) Jede Sequenz $\Phi \Rightarrow \Delta$ mit $\Phi \cap \Delta \neq \varnothing$ ist gültig.

(3) Eine Sequenz $\Phi \Rightarrow \varnothing$ ist gültig gdw. Φ unerfüllbar ist.

(4) Eine Sequenz $\varnothing \Rightarrow \Delta$ (mit endlicher Formelmenge Δ) ist gültig gdw. $\bigvee \Delta$ allgemeingültig ist. ⊣

Basierend auf Sequenzen können wir nun Axiome und Theoreme definieren.

Definition 6.31 Ein *Axiom* des Sequenzenkalküls ist eine Sequenz $\Phi \Rightarrow \Delta$, so dass $\Phi \cap \Delta \neq \varnothing$. Ein *Theorem* ist eine gültige Sequenz $\varnothing \Rightarrow \{\psi\}$, d.h. eine Sequenz ohne Voraussetzungen mit genau einer Konklusion. ⊣

Das Ziel des Sequenzenkalküls ist das Ableiten gültiger Sequenzen aus anderen gültigen Sequenzen durch *Regeln* der Form

$$\frac{\Phi_0 \Rightarrow \Delta_0 \qquad \Phi_1 \Rightarrow \Delta_1 \qquad \Phi_2 \Rightarrow \Delta_2 \qquad \dots}{\Phi' \Rightarrow \Delta'},$$

wobei die Folge $(\Phi_i \Rightarrow \Delta_i)_{i\in\{0,1,\dots\}}$ endlich ist und mindestens ein Element enthält. Wir nennen $\Phi_i \Rightarrow \Delta_i$ die *Prämissen* der Regel und $\Phi' \Rightarrow \Delta'$ ihre *Konsequenz*. Wir werden nur Regeln betrachten, die eine oder zwei Prämissen haben.

Die Idee ist, dass wenn man einmal gezeigt hat, dass alle Prämissen $\Phi_i \Rightarrow \Delta_i$ gültig sind, dann stellt die Regel sicher, dass auch $\Phi' \Rightarrow \Delta'$ gültig ist. Natürlich garantiert nicht jede Regel diese Eigenschaft.

Definition 6.32 Eine Regel

$$\frac{\Phi_0 \Rightarrow \Delta_0 \qquad \Phi_1 \Rightarrow \Delta_1 \qquad \Phi_2 \Rightarrow \Delta_2 \qquad \dots}{\Phi' \Rightarrow \Delta'}$$

ist *korrekt*, wenn die Gültigkeit der Prämissen die Gültigkeit der Konsequenz impliziert. ⊣

Notation. Wir schreiben Mengen von Formel $\{\varphi_1, \dots, \varphi_n\}$ auch als $\varphi_1, \dots, \varphi_n$. Wenn Φ eine Menge von Formeln ist und ψ eine Formel, dann schreiben wir Φ, ψ für $\Phi \cup \{\psi\}$.

Beispiel 6.33 Die Regel

$$\frac{\{p, q\} \Rightarrow \{q\}}{\{p \wedge q\} \Rightarrow \{q\}}$$

wird also notiert als

$$\frac{p, q \Rightarrow q}{p \wedge q \Rightarrow q}.$$

Die Prämisse ist $p, q \Rightarrow q$ und die Konsequenz $p \wedge q \Rightarrow q$. Die Regel ist korrekt. Um das zu sehen, muss man beweisen, dass $p \wedge q \Rightarrow q$ gültig ist, wobei man benutzen darf, dass $p, q \Rightarrow q$ gültig ist. Sei nun β eine Belegung, die $p \wedge q$ erfüllt. (Wir wollen zeigen, dass β die Formel q erfüllt.) Dann erfüllt β auch p und q einzeln. Doch das ist genau die Voraussetzung der Prämisse $p, q \Rightarrow q$. Da die Prämisse gültig ist, erfüllt β die Formel q, was zu zeigen war. ⊣

Wir werden im Folgenden korrekte Regeln vorstellen, die es erlauben, die Formeln, die in den zu beweisenden Sequenzen vorkommen, zu vereinfachen. Für jedes der Symbole $\wedge$, $\vee$, $\neg$ und $\rightarrow$ betrachten wir jeweils zwei Regeln: eine für den Fall, dass das Symbol in der Prämisse steht, und eine für die Konklusion.

Konjunktion auf der linken Seite:

$$(\wedge \Rightarrow) \ \frac{\Phi, \psi, \varphi \Rightarrow \Delta}{\Phi, \psi \wedge \varphi \Rightarrow \Delta}$$

Beispiel 6.34 Die Regel erlaubt Beweise wie den folgenden

$$(\wedge \Rightarrow) \frac{p, q \Rightarrow q}{p \wedge q \Rightarrow q} \quad .$$ ⊣

Die folgende Regel kann benutzt werden, um Konjunktionen auf der rechten Seite einzuführen. Sie hat zwei Prämissen.

Konjunktion auf der rechten Seite.

$$(\Rightarrow \wedge) \frac{\Phi \Rightarrow \Delta, \psi \qquad \Phi \Rightarrow \Delta, \varphi}{\Phi \Rightarrow \Delta, \psi \wedge \varphi}$$

Beispiel 6.35 Die Regel erlaubt Beweise wie den folgenden

$$(\Rightarrow \wedge) \frac{p, q, r \Rightarrow q \qquad p, q, r \Rightarrow r}{p, q, r \Rightarrow q \wedge r} \quad .$$ ⊣

Man könnte nicht viel beweisen, wenn Beweise nur einzelne Regeln wären. Wir können aber Regeln zu komplizierteren Beweisen kombinieren.

Beispiel 6.36

$$\dfrac{\dfrac{p, q, r \Rightarrow q \quad p, q, r \Rightarrow r}{p, q, r \Rightarrow q \wedge r}}{p \wedge q, r \Rightarrow q \wedge r}$$

Aus diesem Beweis können wir folgende Aussage ablesen: Wenn

- die Sequenz $p, q, r \Rightarrow q$ gültig ist und
- die Sequenz $p, q, r \Rightarrow r$ gültig ist,

dann ist auch $p \wedge q, r \Rightarrow q \wedge r$ gültig. Wir wissen bereits, dass $p, q, r \Rightarrow q$ und $p, q, r \Rightarrow r$ gültig sind, da sie Axiome sind, also können wir folgern, dass $p \wedge q, r \Rightarrow q \wedge r$ gültig ist. ⊣

Definition 6.37 Ein *Beweis* im Sequenzenkalkül ist ein Baum, dessen Knoten wie folgt mit Sequenzen beschriftet sind.

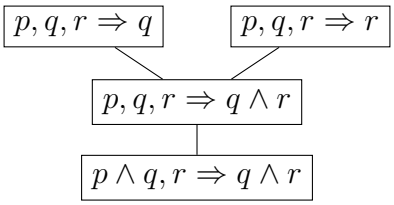
- Die Blätter sind mit Axiomen beschriftet.
- Jeder innere Knoten ist mit der Konsequenz einer korrekten Regel beschriftet. Die Kinder des Knotens sind mit den Prämissen der Regel beschriftet. ⊣

Wir werden in unserer Version des Sequenzenkalküls keine Regeln mit mehr als zwei Prämissen betrachten. Deswegen hat jeder innere Knoten des Baums entweder ein oder zwei Kinder.

Beispiel 6.38 Folgender Baum ist ein Beweis:

$$\dfrac{\dfrac{\overline{p, q, r \Rightarrow q} \quad \overline{p, q, r \Rightarrow r}}{p, q, r \Rightarrow q \wedge r}}{p \wedge q, r \Rightarrow q \wedge r}$$

Wir zeichnen den Baum auch explizit als einen Graphen:



⊣

Notation. Wir „überstreichen“ Axiome, um das Ende des Astes zu markieren.

Definition 6.39 Eine Sequenz $\Phi \Rightarrow \Delta$ ist *beweisbar*, oder kann im Sequenzenkalkül *abgeleitet* werden, wenn sie als Beschriftung eines Knotens in einem Beweis vorkommt. ⊣

Beispiel 6.40 Die Sequenz $p \wedge q, r \Rightarrow q \wedge r$ ist im Sequenzenkalkül beweisbar. ⊣

Der vorherige Beweis zeigt also, dass $p \wedge q, r \Rightarrow q \wedge r$ gültig ist und somit ist

$$(p \wedge q) \wedge r \to (q \wedge r)$$

eine Tautologie.

Wir werden später beweisen, dass alle ableitbaren Sequenzen gültig sind. In dieser Art wird der Sequenzenkalkül zum Beweis logisch korrekter Schlüsse benutzt.

Regeln der Negation.

$$(\neg \Rightarrow) \dfrac{\Phi \Rightarrow \Delta, \psi}{\Phi, \neg\psi \Rightarrow \Delta} \qquad (\Rightarrow \neg) \dfrac{\Phi, \psi \Rightarrow \Delta}{\Phi \Rightarrow \Delta, \neg\psi}$$

Beispiel 6.41 Die Regeln erlauben es, Beweise wie den folgenden zu bilden

$$\dfrac{\dfrac{\overline{\varphi \Rightarrow \varphi}}{\Rightarrow \neg\varphi, \varphi}}{\neg\neg\varphi \Rightarrow \varphi} \qquad \dfrac{\dfrac{\overline{\varphi \Rightarrow \varphi}}{\neg\varphi, \varphi \Rightarrow}}{\varphi \Rightarrow \neg\neg\varphi}$$

⊣

Disjunktion auf der rechten Seite.

$$(\Rightarrow \vee) \dfrac{\Phi \Rightarrow \Delta, \varphi, \psi}{\Phi \Rightarrow \Delta, \varphi \vee \psi}$$

$$(\neg\Rightarrow)\ \frac{\Phi\Rightarrow\Delta,\psi}{\Phi,\neg\psi\Rightarrow\Delta} \qquad (\Rightarrow\neg)\ \frac{\Phi,\psi\Rightarrow\Delta}{\Phi\Rightarrow\Delta,\neg\psi}$$

$$(\wedge\Rightarrow)\ \frac{\Phi,\psi,\varphi\Rightarrow\Delta}{\Phi,\psi\wedge\varphi\Rightarrow\Delta} \qquad (\Rightarrow\wedge)\ \frac{\Phi\Rightarrow\Delta,\psi \qquad \Phi\Rightarrow\Delta,\varphi}{\Phi\Rightarrow\Delta,\psi\wedge\varphi}$$

$$(\vee\Rightarrow)\ \frac{\Phi,\varphi\Rightarrow\Delta \qquad \Phi,\psi\Rightarrow\Delta}{\Phi,\varphi\vee\psi\Rightarrow\Delta} \qquad (\Rightarrow\vee)\ \frac{\Phi\Rightarrow\Delta,\varphi,\psi}{\Phi\Rightarrow\Delta,\varphi\vee\psi}$$

$$(\rightarrow\Rightarrow)\ \frac{\Phi\Rightarrow\Delta,\varphi \qquad \Phi,\psi\Rightarrow\Delta}{\Phi,\varphi\rightarrow\psi\Rightarrow\Delta} \qquad (\Rightarrow\rightarrow)\ \frac{\Phi,\varphi\Rightarrow\Delta,\psi}{\Phi\Rightarrow\Delta,\varphi\rightarrow\psi}$$

Abbildung 6.1.: Die Regeln des aussagenlogischen Sequenzenkalküls.

Beispiel 6.42 Die Regel erlaubt Beweise wie den folgenden:

$$\dfrac{\dfrac{\overline{\varphi\Rightarrow\varphi}}{\Rightarrow\varphi,\neg\varphi}}{\Rightarrow\neg\varphi\vee\varphi}$$

Die Formel $\neg\varphi\vee\varphi$ drückt eines der wichtigsten Prinzipien der Logik aus und wird *tertium non datur* genannt (auf Englisch „law of excluded middle“). ⊣

Disjunktion auf der linken Seite.

$$(\vee\Rightarrow)\ \frac{\Phi,\varphi\Rightarrow\Delta \qquad \Phi,\psi\Rightarrow\Delta}{\Phi,\varphi\vee\psi\Rightarrow\Delta}$$

Beispiel 6.43 Die Regel erlaubt Beweise wie den folgenden:

$$\dfrac{\dfrac{\dfrac{\overline{\varphi\Rightarrow\psi,\varphi}}{\varphi\Rightarrow\psi\vee\varphi} \qquad \dfrac{\overline{\psi\Rightarrow\psi,\varphi}}{\psi\Rightarrow\psi\vee\varphi}}{\varphi\vee\psi\Rightarrow\psi\vee\varphi}$$

⊣

Implikation auf der linken Seite.

$$(\rightarrow\Rightarrow)\ \frac{\Phi\Rightarrow\Delta,\varphi \qquad \Phi,\psi\Rightarrow\Delta}{\Phi,\varphi\rightarrow\psi\Rightarrow\Delta}$$

Implikation auf der rechten Seite.

$$(\Rightarrow\rightarrow)\ \frac{\Phi,\varphi\Rightarrow\Delta,\psi}{\Phi\Rightarrow\Delta,\varphi\rightarrow\psi}$$

Abbildung 6.1 fasst alle Regeln des Sequenzenkalküls noch einmal zusammen.

Beispiel 6.44 Wir wollen zeigen, dass folgende Formel eine Tautologie ist:

$$(X \wedge Y) \to (X \vee Y)\,.$$

Wir beweisen, dass die Sequenz $\Rightarrow (X \wedge Y) \to (X \vee Y)$ gültig ist.
Beweis:

$$\dfrac{\dfrac{\dfrac{\dfrac{}{X, Y \Rightarrow X, Y}}{X, Y \Rightarrow X \vee Y}}{X \wedge Y \Rightarrow X \vee Y}}{\Rightarrow (X \wedge Y) \to (X \vee Y)}$$ ⊣

Beispiel 6.45 Wir beweisen die Gültigkeit folgender Sequenz:

$$\Rightarrow (X \to Y) \to (\neg Y \to \neg X)$$

Beweis:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{X, \neg Y \Rightarrow X}}{\neg Y \Rightarrow X, \neg X}}{\Rightarrow X, (\neg Y \to \neg X)} \quad \dfrac{\dfrac{\dfrac{}{Y \Rightarrow Y, \neg X}}{Y, \neg Y \Rightarrow \neg X}}{Y \Rightarrow (\neg Y \to \neg X)}}{(X \to Y) \Rightarrow (\neg Y \to \neg X)}}{\Rightarrow (X \to Y) \to (\neg Y \to \neg X)}$$ ⊣

Zwei Schlüsseleigenschaften eines Kalküls sind *Korrektheit* und *Vollständigkeit*. Ein Kalkül ist korrekt, wenn aus gültigen Sequenzen nur gültige Sequenzen abgeleitet werden können. Dazu reicht es zu zeigen, dass jede Regel des Kalküls korrekt ist. Die Vollständigkeit bedeutet, dass alle gültigen Sequenzen aus Axiomen ableitbar sind. Wir werden sehen, dass der Sequenzenkalkül vollständig und korrekt ist.

Lemma 6.46 *Die Regeln des Sequenzenkalküls sind korrekt.*

Wir werden eine etwas stärkere Behauptung beweisen.

Definition 6.47 Sei $\Phi \Rightarrow \Delta$ eine Sequenz und β eine zu $\Phi \cup \Delta$ passende Belegung.

(1) β *falsifiziert* die Sequenz $\Phi \Rightarrow \Delta$, falls sie alle $\varphi \in \Phi$ erfüllt, aber kein $\delta \in \Delta$.

(2) β *erfüllt* die Sequenz $\Phi \Rightarrow \Delta$, falls sie ein $\varphi \in \Phi$ nicht erfüllt oder aber mindestens ein $\delta \in \Delta$ erfüllt. ⊣

Lemma 6.48 *Für jede Regel des Sequenzenkalküls und jede Belegung β, die zu allen Formeln der Regel passt, gilt: β erfüllt die Konsequenz der Regel genau dann, wenn β alle Prämissen erfüllt.*

Beweis. Wir beweisen das Lemma exemplarisch für die beiden Regeln der Disjunktion. Die anderen Regeln sind zur Übung empfohlen.

Wir betrachten zunächst die Regel

$$(\Rightarrow \vee)\ \frac{\Phi \Rightarrow \Delta, \varphi, \psi}{\Phi \Rightarrow \Delta, \varphi \vee \psi}\ .$$

Sei β eine zu allen Formeln der Regel passende Belegung. Angenommen, β falsifiziert die Prämisse $\Phi \Rightarrow \Delta, \varphi, \psi$. Also gilt $\beta \models \vartheta$ für alle $\vartheta \in \Phi$ aber $\beta \not\models \delta$ für alle $\delta \in \Delta, \varphi, \psi$. Dann gilt aber $\beta \not\models \varphi \vee \psi$ und somit falsifiziert β die Sequenz $\Phi \Rightarrow \Delta, \varphi \vee \psi$.

Umgekehrt nehmen wir an, dass β die Konklusion $\Phi \Rightarrow \Delta, \varphi \vee \psi$ falsifiziert. Dann gilt $\beta \models \vartheta$ für alle $\vartheta \in \Phi$ aber $\beta \not\models \delta$ für alle $\delta \in \Delta, \varphi \vee \psi$. Dann gilt aber $\beta \not\models \varphi$ und $\beta \not\models \psi$ und somit falsifiziert β die Sequenz $\Phi \Rightarrow \Delta, \varphi, \psi$.

Als zweites Beispiel beweisen wir die Korrektheit der Regel

$$(\vee \Rightarrow)\ \frac{\Phi, \varphi \Rightarrow \Delta \qquad \Phi, \psi \Rightarrow \Delta}{\Phi, \varphi \vee \psi \Rightarrow \Delta}\ .$$

Sei dazu β eine Belegung, die zu allen Formeln der Regel passt. Wenn β die Sequenz $\Phi, \varphi \vee \psi \Rightarrow \Delta$ falsifiziert, gilt also $\beta \models \varphi'$ für alle $\varphi' \in \Phi \cup \{\varphi \vee \psi\}$ und $\beta \not\models \delta$ für alle $\delta \in \Delta$. Es gilt also $\beta \models \varphi$ oder $\beta \models \psi$. Im ersten Fall wird aber die erste, im zweiten Fall die zweite Prämisse falsifiziert.

Umgekehrt, wenn β die Prämissen $\Phi, \varphi \Rightarrow \Delta$ und $\Phi, \psi \Rightarrow \Delta$ falsifiziert, gilt also $\beta \models \varphi'$ für alle $\varphi' \in \Phi \cup \{\varphi, \psi\}$ und $\beta \not\models \delta$ für alle $\delta \in \Delta$. Es gilt also auch $\beta \models \varphi \vee \psi$ und somit wird die Konklusion falsifiziert.

Analog zeigt man die Korrektheit aller anderen Regeln des Sequenzenkalküls. □

Als Folgerung des Lemmas erhält man folgenden Satz.

Theorem 6.49 *Jede im Sequenzenkalkül beweisbare Sequenz ist gültig.*

Man kann sogar mehr zeigen: Nicht nur gibt es zu jeder Aussage, die in der Aussagenlogik formalisierbar ist, einen Beweis im Sequenzenkalkül, dieser ist durch eine systematische Suche zu finden. Wir werden einen Algorithmus angeben, der zu jeder Sequenz $\Phi \Rightarrow \Delta$ entweder

- einen Beweis im Sequenzenkalkül konstruiert oder
- eine Belegung findet, die alle Konsequenzen falsifiziert, aber alle Voraussetzungen erfüllt.

Im zweiten Fall ist die Belegung ein Beweis, dass die Sequenz nicht gültig ist.

Bemerkung 6.50 Es gibt gültige Sequenzen, die mehrere Beweise besitzen, aber für jede nicht-atomare Formel in jeder Sequenz gibt es genau eine anwendbare Regel. Als Beispiel betrachte man die Sequenz

$$\Rightarrow (X \to Y) \to (\neg Y \to \neg X)\,.$$

Man kann zwei verschiedene Beweise dieser Sequenz angeben.

Beweis 1:

$$\dfrac{\dfrac{\dfrac{\overline{X, \neg Y \Rightarrow X}}{\neg Y \Rightarrow X, \neg X}}{\Rightarrow X, (\neg Y \to \neg X)} \qquad \dfrac{\dfrac{\overline{Y \Rightarrow Y, \neg X}}{Y, \neg Y \Rightarrow \neg X}}{Y \Rightarrow (\neg Y \to \neg X)}}{\dfrac{(X \to Y) \Rightarrow (\neg Y \to \neg X)}{\Rightarrow (X \to Y) \to (\neg Y \to \neg X)}}$$

Beweis 2:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\overline{X \Rightarrow X, Y} \qquad \overline{X, Y \Rightarrow Y}}{(X \to Y), X \Rightarrow Y}}{(X \to Y), \Rightarrow \neg X, Y}}{(X \to Y), \neg Y \Rightarrow \neg X}}{(X \to Y) \Rightarrow (\neg Y \to \neg X)}}{\Rightarrow (X \to Y) \to (\neg Y \to \neg X)}$$

⊣

Beispiel 6.51 Wir wollen die folgende Sequenz beweisen: $X \vee Y \Rightarrow X \wedge Y$. Ein Beweisversuch führt zu folgendem Ableitungsbaum:

$$\dfrac{\dfrac{\overline{X \Rightarrow X} \quad Y \Rightarrow X}{X \vee Y \Rightarrow X} \qquad \dfrac{X \Rightarrow Y \quad \overline{Y \Rightarrow Y}}{X \vee Y \Rightarrow Y}}{X \vee Y \Rightarrow X \wedge Y}$$

Wir erhalten einen Baum, dessen Blätter alle mit Sequenzen beschriftet sind, die nur noch atomare Formeln enthalten. Es sind also keine weiteren Regeln mehr anwendbar. Allerdings sind nur zwei Blätter Axiome, die anderen können falsifiziert werden. Wählen wir β durch $\beta(X) := 1$ und $\beta(Y) := 0$. Diese Belegung falsifiziert ein Blatt, aber auch die Originalsequenz. Der Versuch, die Sequenz $X \vee Y \Rightarrow X \wedge Y$ zu beweisen, führt also zu einer falsifizierenden Belegung eines Blattes. Nach Lemma 6.48, falsifiziert dies auch die Originalsequenz. ⊣

Definition 6.52 Ein *Ableitungsbaum für eine Sequenz* $\mathcal{S}$ ist ein Baum, dessen Knoten wie folgt mit Sequenzen beschriftet sind:

- Jeder innere Knoten ist mit der Konsequenz einer korrekten Regel beschriftet. Die Kinder des Knotens sind mit den Prämissen der Regel beschriftet.
- Die Wurzel des Baumes ist mit $\mathcal{S}$ beschriftet.

Ein Blatt des Baums heißt *positiv*, wenn es mit einem Axiom beschriftet ist. Es ist *negativ*, wenn es mit einer Sequenz $\Phi \Rightarrow \Delta$ beschriftet ist, so dass Φ, Δ disjunkte Mengen von Variablen sind. Ein Ableitungsbaum für $\mathcal{S}$ ist *vollständig*, wenn jedes Blatt positiv oder negativ ist. Ein vollständiger Ableitungsbaum für $\mathcal{S}$ ist eine *Widerlegung*, wenn er ein negatives Blatt enthält. ⊣

Bemerkung 6.53 Ein Beweis für $\mathcal{S}$ ist ein vollständiger Ableitungsbaum dessen Blätter alle positiv sind. ⊣

Wir können nun einen Algorithmus angeben, der für eine aussagenlogische Sequenz entscheidet, ob diese gültig ist.

Algorithmus

Eingabe: Eine Sequenz $\Phi \Rightarrow \Delta$

Ausgabe: Ein Beweis für $\Phi \Rightarrow \Delta$ oder eine falsifizierende Belegung der Sequenz.

Wir konstruieren induktiv einen Ableitungsbaum für $\Phi \Rightarrow \Delta$ wie folgt. Zunächst besteht der Baum aus einem mit $\Phi \Rightarrow \Delta$ beschrifteten unmarkierten Knoten.

Solange es unmarkierte Blätter gibt:

(1) Wähle ein unmarkiertes Blatt t. Sei $\Phi' \Rightarrow \Delta'$ die Beschriftung des Blattes.

(2) Ist t negativ, konstruiere die Belegung, die alle Variablen in Φ' mit 1 belegt und alle anderen mit 0. Gebe diese Belegung aus. Terminiere.

(3) Ist t positiv, markiere es mit (+).

(4) Anderenfalls wähle eine nicht-atomare Formel $\varphi \in \Phi' \cup \Delta'$ und wende die (eindeutig bestimmte) anwendbare Regel an.

(5) Füge ein oder zwei Nachfolger zu t hinzu, beschriftet mit den Prämissen der Regel mit Konklusion $\Phi' \Rightarrow \Delta'$.

Sind alle Blätter markiert, gib den Baum als Beweis für $\Phi \Rightarrow \Delta$ zurück.

Theorem 6.54 *Der Algorithmus terminiert auf jeder Sequenz $\Phi \Rightarrow \Delta$, in der $\Phi \cup \Delta$ endlich ist. Er findet einen Beweis für $\Phi \Rightarrow \Delta$, wenn es einen gibt. Anderenfalls gibt er eine Widerlegung für $\Phi \Rightarrow \Delta$ zurück.*

Beweis. Wir zeigen zunächst die *Terminierung*. Sei die *Komplexität* einer Sequenz $\Phi \Rightarrow \Delta$ definiert als die Zahl der Verknüpfungen, die in Formeln aus $\Phi \cup \Delta$ vorkommen. Jede Regel des Sequenzenkalküls erhöht die Komplexität, d.h. die Konklusion einer Regel ist komplexer als ihre Prämissen. Der Algorithmus muss also terminieren.

Angenommen der Algorithmus findet einen Ableitungsbaum, in dem alle Blätter markiert und somit positiv sind. Dann ist der Baum ein Beweis im Sequenzenkalkül und nach dem Korrektheitssatz ist die Sequenz $\Phi \Rightarrow \Delta$ gültig.

Anderenfalls findet der Algorithmus ein negatives Blatt und konstruiert eine falsifizierende Belegung. Nach Lemma 6.48, falsifiziert diese Belegung auch $\Phi \Rightarrow \Delta$. □

Der Sequenzenkalkül stellt eine Verbindung zwischen der *semantischen* Erfüllbarkeitsrelation $\models$ und einem Verfahren, das völlig syntaktisch auf einer gegeben Sequenz arbeitet. Um diese Verbindung präzise auszudrücken, definieren die Ableitbarkeit als eine Relation zwischen Formelmengen und Formeln.

Definition 6.55 Sei $\Phi \subseteq \mathrm{AL}$ eine Menge von Formeln. Eine Formel $\psi \in \mathrm{AL}$ kann aus Φ *hergeleitet* werden, geschrieben $\Phi \vdash_S \psi$, wenn es eine endliche Teilmenge $\Phi' \subseteq \Phi$ gibt, so dass $\Phi' \Rightarrow \psi$ im Sequenzenkalkül beweisbar ist. ⊣

Insbesondere kann ψ aus der leeren Menge hergeleitet werden, d.h. $\vdash_S \psi$, wenn $\varnothing \Rightarrow \psi$ im Sequenzenkalkül beweisbar ist.

Definition 6.56 Sei $\Phi \subseteq \mathrm{AL}$ eine endliche oder unendliche Formelmenge, dann ist Φ *inkonsistent*, wenn es eine Formel $\psi \in \mathrm{AL}$ gibt, so dass $\Phi \vdash_S \psi$ und $\Phi \vdash_S \neg\psi$. Anderenfalls ist Φ *konsistent*. ⊣

Theorem 6.54 und der Algorithmus zeigen die Vollständigkeit des Sequenzenkalküls für endliche Formelmengen. Mit Hilfe des Kompaktheitssatzes folgt die Vollständigkeit für beliebige Formelmengen.

Theorem 6.57 (Vollständigkeit und Korrektheit) *Sei $\Phi \subseteq \mathrm{AL}$ eine Menge von Formeln und sei $\psi \in \mathrm{AL}$.*

(1) *Φ ist konsistent genau dann, wenn Φ erfüllbar ist.*

(2) *$\Phi \vdash_S \psi$ gilt genau dann, wenn $\Phi \models \psi$ gilt.*

Welche syntaktische Begriffe welchen semantischen Begriffen entsprechen, fassen wir in folgender Tabelle zusammen.

Sequenzenkalkül	*Semantische Konzepte*
Sequenz $\Phi \Rightarrow \psi$ ist gültig	Aus Φ folgt logisch ψ
$\vdash$	$\models$
Φ ist konsistent	Φ ist erfüllbar
$\varnothing \Rightarrow \psi$ gültig	ψ allgmeingültig
$\psi \Rightarrow \varnothing$ gültig	ψ unerfüllbar

6.4.2. Sequenzenkalkül der Prädikatenlogik

Wir erweitern nun den Sequenzenkalkül für die Prädikatenlogik. Dazu können wir alle bisherigen Regeln und Definitionen direkt übernehmen und brauchen nur noch neue Regeln für die Quantoren. Dies erfordert allerdings etwas Vorarbeit.

Bei der Definition von Substitutionen haben wir gesehen, dass die Kombination von freien und gebundenen Variablen Probleme bereiten kann. Um ähnliche Probleme im Sequenzenkalkül zu vermeiden, werden wir freie Variablen durch Konstantensymbole substituieren und somit nur mit Sätzen arbeiten.

Lemma 6.58 *Sei σ eine Signatur. Sei $\varphi(x_1, \ldots, x_k)$ eine Formel mit freien Variablen $\mathrm{frei}(\varphi) \subseteq \{x_1, \ldots, x_k\}$. Seien $c_1, \ldots, c_k$ Konstantensymbole mit $c_i \notin \sigma$ für alle $1 \leqslant i \leqslant k$. Dann gilt, dass φ erfüllbar ist gdw. $\varphi[x_1/c_1, \ldots, x_k/c_k]$ erfüllbar ist.*

Beweis. Wenn φ erfüllbar ist, dann gibt es eine σ-Struktur $\mathcal{A}$ und Belegung β mit $\{x_1, \ldots, x_k\} \subseteq \mathrm{def}(\beta)$, so dass $(\mathcal{A}, \beta) \models \varphi$.

Aber dann erfüllt die $\sigma \cup \{c_1, \ldots, c_k\}$-Struktur $\mathcal{B}$ mit $\mathcal{B}_{|\sigma} = \mathcal{A}$ und $c_i^{\mathcal{B}} := \beta(x_i)$ die Formel $\varphi[x_1/c_1, \ldots, x_k/c_k]$. Die Umkehrung ist analog. □

Das Lemma zeigt, dass Erfüllbarkeit und Gültigkeit von Formeln auf entsprechende Aussagen über Sätzen reduziert werden können.

Notation. In der Formulierung des Sequenzenkalküls werden wir ausschließlich mit Sätzen arbeiten, d.h. Formeln ohne freie Variablen. Das heißt, wann immer wir $\Phi, \Delta, \varphi, \psi, ...$ schreiben, meinen wir Sätze oder Mengen von Sätzen. Wenn wir jedoch $\varphi(x)$ oder $\psi(x)$ schreiben, dann soll das bedeuten, dass die Formeln eine freie Variable x haben.

Für den Rest des Abschnitts fixieren wir eine Signatur σ und eine abzählbar unendliche Menge $c_0, c_1, \dots$ von Konstantensymbolen $c_i \notin \sigma$, für alle $i \geqslant 0$. Das heißt, formal arbeiten wir in der Signatur $\tau := \sigma \cup \{c_0, c_1, \dots\}$.

Analog zum aussagenlogischen Sequenzenkalkül definieren wir Sequenzen, deren Gültigkeit, Regeln, deren Korrektheit usw. für die Prädikatenlogik.

Definition 6.59 Sei σ eine Signatur.

(1) Eine *Sequenz* ist eine Aussage der Form

$$\Phi \Rightarrow \Delta$$

wobei $\Phi, \Delta \subseteq \mathrm{FO}[\sigma]$ Mengen von Formeln sind. Wir nennen Φ die *Voraussetzungen* und Δ die *Konklusionen*.

(2) Eine Sequenz $\Phi \Rightarrow \Delta$ ist *gültig*, wenn jede σ-Interpretation $\mathcal{I}$, die alle Formeln in Φ erfüllt, auch eine Formel aus Δ erfüllt.

(3) Ein *Axiom* des Sequenzenkalküls ist eine Sequenz $\Phi \Rightarrow \Delta$, so dass $\Phi \cap \Delta \neq \varnothing$.

(4) Wenn $\Phi \Rightarrow \Delta$ nicht gültig ist, dann gibt es eine σ-Interpretation $\mathcal{I}$, die alle $\varphi \in \Phi$, aber kein $\psi \in \Delta$ erfüllt. Wir sagen: $\mathcal{I}$ *falisifiziert die Sequenz.* ⊣

Die Regeln des Sequenzenkalküls sind die Regeln des aussagenlogischen Kalküls erweitert um die *Gleichheitsregel*, die *Substitutionsregel* und die *Quantorenregeln.*

Die Regel für Gleichheit.

$$(=)\ \frac{\Phi, t = t \Rightarrow \Delta}{\Phi \Rightarrow \Delta}$$

Die Regeln der Substitution.

$$(S \Rightarrow)\ \frac{\Phi, \psi(t) \Rightarrow \Delta}{\Phi, t \doteq t', \psi(t') \Rightarrow \Delta} \qquad (\Rightarrow S)\ \frac{\Phi, \Rightarrow \Delta, \psi(t)}{\Phi, t \doteq t' \Rightarrow \Delta, \psi(t')}$$

Die Notation $t \doteq t'$ bedeutet, dass wir $t = t'$ oder $t' = t$ verwenden können. Hierbei sind t, t' Terme ohne freie Variablen und $\psi(x)$ ist eine Formel, wobei x die einzige freie Variable ist.

Beispiel 6.60 Sei $\sigma := \{R, f, c\}$, wobei R ein Relationssymbol, f ein Funktionssymbol und c ein Konstantensymbol ist. Setzt man $\psi(x) := Rfx$ in der Regel $(\Rightarrow S)$, so erhält man die Regel

$$\frac{Rfc \Rightarrow Rfc}{Rfc, fc = c \Rightarrow Rffc}.$$ ⊣

Die Regeln für den Existenzquantor.

$(\exists \Rightarrow)$ $\dfrac{\Phi, \psi(c) \Rightarrow \Delta}{\Phi, \exists x\psi(x) \Rightarrow \Delta}$, wenn c nicht in Φ, Δ und ψ vorkommt;

$(\Rightarrow \exists)$ $\dfrac{\Phi \Rightarrow \Delta, \psi(t)}{\Phi \Rightarrow \Delta, \exists x\psi(x)}$

Beispiel 6.61 Das Folgende ist ein Beweis für die Sequenz $\exists x\exists yRxy \Rightarrow \exists y\exists xRxy$.

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{R(c,d) \Rightarrow R(c,d)}}{R(c,d) \Rightarrow \exists xR(x,d)}}{R(c,d) \Rightarrow \exists y\exists xR(x,y)}}{\exists yR(c,y) \Rightarrow \exists y\exists xR(x,y)}}{\exists x\exists yR(x,y) \Rightarrow \exists y\exists xR(x,y)}$$ ⊣

Die Regeln für den Allquantor.

$(\forall \Rightarrow)$ $\dfrac{\Phi, \psi(t) \Rightarrow \Delta}{\Phi, \forall x\psi(x) \Rightarrow \Delta}$

$(\Rightarrow \forall)$ $\dfrac{\Phi \Rightarrow \Delta, \psi(c)}{\Phi \Rightarrow \Delta, \forall x\psi(x)}$, wenn c nicht in Φ, Δ und $\psi(x)$ vorkommt

Beispiel 6.62 Das Folgende ist ein Beweis für $\forall x\exists yRxy \Rightarrow \exists y\forall xRxy$.

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{R(c,d) \Rightarrow R(c,d)}}{R(c,d) \Rightarrow \exists xR(x,d)}}{\forall yR(c,y) \Rightarrow \exists xR(x,d)}}{\forall yR(c,y) \Rightarrow \forall y\exists xR(x,y)}}{\exists x\forall yR(x,y) \Rightarrow \forall y\exists xR(x,y)}$$ ⊣

Abbildung 6.2 fasst noch einmal alle Regeln des prädikatenlogischen Sequenzenkalküls zusammen.

Beispiel 6.63 Wir zeigen, dass $\varphi \vee \exists x\psi(x) \equiv \exists x(\varphi \vee \psi(x))$ für alle Formeln φ und ψ gilt, falls x nicht in φ vorkommt. Dazu geben wir Beweise für $\varphi \vee \exists x\varphi(x) \Rightarrow \exists x(\varphi \vee \psi(x))$ und $\exists x(\varphi \vee \psi(x)) \Rightarrow \varphi \vee \exists x\varphi(x)$ an.

Sequenzenkalkülbeweis von $\varphi \vee \exists x\varphi(x) \Rightarrow \exists x(\varphi \vee \psi(x))$:

$$\dfrac{\dfrac{\dfrac{\dfrac{}{\varphi \Rightarrow \varphi, \psi(c)}}{\varphi \Rightarrow \varphi \vee \psi(c)}}{\varphi \Rightarrow \exists x(\varphi \vee \psi(x))} \quad \dfrac{\dfrac{\dfrac{\dfrac{}{\psi(c) \Rightarrow \varphi, \psi(c)}}{\psi(c) \Rightarrow \varphi \vee \psi(c)}}{\psi(c) \Rightarrow \exists x(\varphi \vee \psi(x))}}{\exists x\psi(x) \Rightarrow \exists x(\varphi \vee \psi(x))}}{\varphi \vee \exists x\psi(x) \Rightarrow \exists x(\varphi \vee \psi(x))}$$

$$(\neg\Rightarrow)\ \frac{\Phi\Rightarrow\Delta,\psi}{\Phi,\neg\psi\Rightarrow\Delta} \qquad (\Rightarrow\neg)\ \frac{\Phi,\psi\Rightarrow\Delta}{\Phi\Rightarrow\Delta,\neg\psi}$$

$$(\wedge\Rightarrow)\ \frac{\Phi,\psi,\varphi\Rightarrow\Delta}{\Phi,\psi\wedge\varphi\Rightarrow\Delta} \qquad (\Rightarrow\wedge)\ \frac{\Phi\Rightarrow\Delta,\psi \qquad \Phi\Rightarrow\Delta,\varphi}{\Phi\Rightarrow\Delta,\psi\wedge\varphi}$$

$$(\vee\Rightarrow)\ \frac{\Phi,\varphi\Rightarrow\Delta \qquad \Phi,\psi\Rightarrow\Delta}{\Phi,\varphi\vee\psi\Rightarrow\Delta} \qquad (\Rightarrow\vee)\ \frac{\Phi\Rightarrow\Delta,\varphi,\psi}{\Phi\Rightarrow\Delta,\varphi\vee\psi}$$

$$(\to\Rightarrow)\ \frac{\Phi\Rightarrow\Delta,\varphi \qquad \Phi,\psi\Rightarrow\Delta}{\Phi,\varphi\to\psi\Rightarrow\Delta} \qquad (\Rightarrow\to)\ \frac{\Phi,\varphi\Rightarrow\Delta,\psi}{\Phi\Rightarrow\Delta,\varphi\to\psi}$$

$$(\forall\Rightarrow)\ \frac{\Phi,\psi(t)\Rightarrow\Delta}{\Phi,\forall x\psi(x)\Rightarrow\Delta} \qquad (\Rightarrow\forall)\ \frac{\Phi\Rightarrow\Delta,\psi(c)}{\Phi\Rightarrow\Delta,\forall x\psi(x)}\ c\ \textbf{n. i.}\ \Phi,\Delta,\psi$$

$$(\exists\Rightarrow)\ \frac{\Phi,\psi(c)\Rightarrow\Delta}{\Phi,\exists x\psi(x)\Rightarrow\Delta}\ c\ \textbf{n.i.}\ \Phi,\Delta,\psi \qquad (\Rightarrow\exists)\ \frac{\Phi\Rightarrow\Delta,\psi(t)}{\Phi\Rightarrow\Delta,\exists x\psi(x)}$$

$$(S\Rightarrow)\ \frac{\Phi,\psi(t)\Rightarrow\Delta}{\Phi,t\doteq t',\psi(t')\Rightarrow\Delta} \qquad (\Rightarrow S)\ \frac{\Phi,\Rightarrow\Delta,\psi(t)}{\Phi,t\doteq t'\Rightarrow\Delta,\psi(t')} \qquad (=)\ \frac{\Phi,t=t\Rightarrow\Delta}{\Phi\Rightarrow\Delta}$$

Abbildung 6.2.: Die Regeln des prädikatenlogischen Sequenzenkalküls.

Der Sequenzenkalkülbeweis von $\exists x(\varphi\vee\psi(x))\Rightarrow\varphi\vee\exists x\varphi(x)$ ist ähnlich. Die Annahme, dass x nicht in φ vorkommt, wird im linken Zweig benutzt, da wir sonst folgenden Nicht-Beweis erhalten:

$$\frac{\dfrac{\varphi(x)\Rightarrow\varphi(c),\psi(c)}{\varphi(x)\Rightarrow\varphi(c)\vee\psi(c)}}{\varphi(x)\Rightarrow\exists x(\varphi(x)\vee\psi(x))}\qquad\dashv$$

Wir definieren die Korrektheit der prädikatenlogischen Regeln genauso wie für die Aussagenlogik.

Definition 6.64 Eine Regel

$$\frac{\Phi_0\Rightarrow\Delta_0 \qquad \Phi_1\Rightarrow\Delta_1 \qquad \Phi_2\Rightarrow\Delta_2 \qquad \dots}{\Phi'\Rightarrow\Delta'}$$

ist *korrekt*, wenn aus der Gültigkeit von allen $\Phi_i\Rightarrow\Delta_i$ für $i\geqslant 0$ die Gültigkeit von $\Phi'\Rightarrow\Delta'$ folgt. $\dashv$

Lemma 6.65 *Die Regeln für den Allquantor sind korrekt.*

Beweis. Wir zeigen zuerst, dass die $(\forall\Rightarrow)$-Regel korrekt ist.

Sei dazu $\mathcal{I}:=(\mathcal{A},\beta)$ eine σ-Interpretation, so dass $\mathcal{I}\models\Phi$ und $\mathcal{I}\models\forall x\psi(x)$. Sei $a:=[\![t]\!]^{\mathcal{I}}$. Dann gilt also $\mathcal{I}\models\psi[a]$. Es folgt, dass $\mathcal{I}\models\psi(t)$ und, da nach Voraussetzung $\Phi,\psi(t)\Rightarrow\Delta$ gültig ist, gilt $\mathcal{I}\models\delta$ für ein $\delta\in\Delta$.

Jetzt betrachten wir die Regel $(\Rightarrow\forall)$. Sei τ die Signatur, die alle (Funktions-, Relations- und Konstanten-) Symbole enthält, die in $\Phi,\Delta,\psi(x)$ vorkommen, außer c. Sei $\mathcal{I}:=(\mathcal{A},\beta)$ eine τ-Interpretation mit $\mathcal{I}\models\Phi$. Wenn $\mathcal{I}\models\delta$, für ein $\delta\in\Delta$, dann sind wir fertig.

Anderenfalls, gilt $\mathcal{I} \not\models \delta$ für alle $\delta \in \Delta$. Für $a \in A$ sei $\mathcal{I}_a$ die $\tau \cup \{c\}$-Expansion von $\mathcal{I}$ mit $c^{\mathcal{I}_a} := a$ (d.h. $(\mathcal{I}_a)_{|\tau} = \mathcal{I}$). Da c nur in ψ vorkommt, folgt aus dem Koinzidenzlemma, dass $\mathcal{I}_a \models \Phi$, aber $\mathcal{I}_a \not\models \delta$ für alle $\delta \in \Delta$. Also gilt wegen der Gültigkeit der Prämisse $\mathcal{I}_a \models \psi(c)$. Es folgt, dass $\mathcal{I}_a \models \psi(c)$ für alle $a \in A$ und somit $\mathcal{I} \models \forall x \psi(x)$. □

Wir wollen uns überzeugen, dass die Voraussetzung „c kommt nicht in Γ, Δ und $\psi(x)$ vor“ notwendig ist.

Beispiel 6.66 Sei $\Delta := \varnothing$ und $\Phi := \{R(c)\}$ und $\psi(x) := R(x)$. Dann ist $\Phi \Rightarrow \psi(c)$ sicherlich gültig. Aber $\Phi \Rightarrow \forall x R(x)$ ist nicht gültig, wie die folgende Struktur $\mathcal{A} := (\{0,1\}, c^{\mathcal{A}} := 0, R^{\mathcal{A}} := \{0\})$ zeigt. ⊣

Die Begriffe des Beweises, der Ableitbarkeit (Beweisbarkeit), der Inkonsistenz und der Konsistenz werden für dir Prädikatenlogik genauso definiert, wie für die Aussagenlogik.

Bemerkung 6.67 Die Klasse der ableitbaren Sequenzen ist also die kleinste Klasse, die alle Axiome enthält und wann immer sie die Prämissen einer Regel enthält, dann auch deren Konsequenz. ⊣

Theorem 6.68 (Korrektheitssatz) *Die Regeln des Sequenzenkalküls sind korrekt, d.h. jede beweisbare Sequenz ist gültig.*

Beweis. Der Satz folgt sofort aus der Korrektheit der einzelnen Regeln. Wir haben die Korrektheit der aussagenlogischen Regeln sowie der Regeln für den Allquantor bewiesen. Die anderen Regeln sind zur Übung empfohlen. □

Das folgende Theorem ist einer der wichtigsten Sätze der Prädikatenlogik, denn er verbindet die Semantik der Prädikatenlogik mit einem rein syntaktischen Verfahren, um die Semantik zu überprüfen.

Theorem 6.69 (1) *Sei* $\Phi \subseteq \mathrm{FO}[\sigma]$ *eine Menge von Formeln und* $\psi \in \mathrm{FO}[\sigma]$. $\Phi \models \psi$ *genau dann, wenn* $\Phi \Rightarrow \psi$ *beweisbar ist.*

(2) *Sei* $\Phi \subseteq \mathrm{FO}[\sigma]$ *eine Menge von Formeln.* Φ *ist erfüllbar genau dann, wenn* Φ *konsistent ist.*

Beweis. Die Richtung von „rechts nach links“ ist im wesentlichen der Korrektheitssatz. Die andere Richtung ist erheblich schwieriger und ist als *Gödelscher Vollständigkeitssatz* bekannt (nicht zu verwechseln mit *Gödelschem Unvollständigkeitssatz*). □

Ein anderer wichtiger Satz über die Prädikatenlogik ist der Kompaktheitssatz, der wie für die Aussagenlogik formuliert wird. Der Beweis folgt aber jetzt aus dem Vollständigkeitssatz mit relativ simplen Argumenten. Die komplizierten Argumente sind so im Beweis des Vollständigkeitssatzes versteckt.

Theorem 6.70 (Kompaktheit der Prädikatenlogik) (1) *Sei $\Phi \subseteq \text{FO}[\sigma]$ eine Menge von Formeln. Φ ist erfüllbar genau dann, wenn jede endliche Teilmenge von Φ erfüllbar ist.*

(2) *Sei $\Phi \subseteq \text{FO}[\sigma]$ und $\psi \in \text{FO}[\sigma]$. $\Phi \models \psi$ genau dann, wenn es eine endliche Teilmenge $\Phi_0 \subseteq \Phi$ gibt, so dass $\Phi_0 \models \psi$.*

Beweis. Wir zeigen zuerst Teil (1) des Satzes. Sei $\Phi \subseteq \text{FO}[\sigma]$ eine Menge von Formeln. Wir wollen zeigen, dass Φ genau dann erfüllbar ist, wenn jede endliche Teilmenge von Φ erfüllbar ist.

Wenn Φ erfüllbar ist, so ist natürlich auch jede endliche Teilmenge erfüllbar. Sei also jede endliche Teilmenge von Φ erfüllbar. Angenommen, Φ sei nicht erfüllbar. Dann ist Φ nach Theorem 6.69 inkonsistent und es gibt ein $\psi \in \text{FO}[\sigma]$, so dass $\Phi \models \psi$ und $\Phi \models \neg\psi$. Aus dem Vollständigkeitssatz folgt nun, dass $\Phi \vdash_S \psi$ und $\Phi \vdash_S \neg\psi$. Beide Beweise sind aber endlich und verwenden daher nur endlich viele Formeln aus Φ. Seien Φ_1, Φ_2 die in den beiden Beweisen benutzen Formeln aus Φ. Dann sind also ψ und $\neg\psi$ bereits aus $\Phi_1 \cup \Phi_2$ ableitbar. Somit ist $\Phi_1 \cup \Phi_2$ inkonsistent und damit unerfüllbar, im Widerspruch zur Annahme.

Als nächstes zeigen wir Teil (2) des Satzes. Sei $\Phi \subseteq \text{FO}[\sigma]$ und $\psi \in \text{FO}[\sigma]$. Die Behauptung folgt sofort aus dem ersten Teil. Denn $\Phi \models \psi$ gdw. $\Phi \cup \{\neg\psi\}$ unerfüllbar ist gdw. es eine unerfüllbare endliche Teilmenge $\Phi_0 \subseteq \Phi \cup \{\neg\psi\}$ gibt gdw. $\Phi_0 \setminus \{\neg\psi\} \models \psi$. □

Wir haben gesehen, dass es einen Algorithmus gibt, der als Eingabe eine aussagenlogische Sequenz $\Phi \Rightarrow \Delta$ bekommt und entscheidet, ob $\Phi \Rightarrow \Delta$ im aussagenlogischen Sequenzenkalkül beweisbar ist. Der Algorithmus terminiert auf jeder endlichen Eingabe. Insbesondere kann der Algorithmus benutzt werden, um für eine Formel $\psi \in \text{AL}$ zu entscheiden, ob $\Rightarrow \psi$ beweisbar und somit ψ eine Tautologie ist. Wir erhalten also folgende algorithmische Aussage für die Aussagenlogik.

Korollar 6.71 (1) *Das Allgemeingültigkeitsproblem der Aussagenlogik ist entscheidbar.*

(2) *Das Erfüllbarkeitsproblem der Aussagenlogik ist entscheidbar.*

Die Frage, on es einen solchen Algorithmus auch für die Prädikatenlogik gibt, war eines der wichtigsten Probleme der mathematischen Logik.

> *Das Entscheidungsproblem ist gelöst, wenn man ein Verfahren kennt, das bei einem vorgelegten logischen Ausdruck durch endlich viele Operationen die Entscheidung über die Allgemeingültigkeit bzw. Erfüllbarkeit erlaubt. (...) Das Entscheidungsproblem muss als das Hauptproblem der mathematischen Logik betrachtet werden.*
>
> (D. Hilbert, W. Ackermann: Grundzüge der theoretischen Logik, Vol. 1, Berlin 1928, S.73 ff.)

Eine positive Antwort auf das Entscheidungsproblem, also ein Algorithmus, der Allgemeingültigkeit oder Erfüllbarkeit prädikatenlogischer Formeln entscheiden könnte, hätte weitreichende Folgen für die Mathematik. Wir könnten dann zum Beispiel die Gruppentheorie entscheiden, d.h. für jede Formel in der Signatur der Gruppen entscheiden, ob sie in allen Gruppen gilt.

Für die Prädikatenlogik kann ein solcher Algorithmus allerdings nicht existieren.

Theorem 6.72 (Church, Turing 1936, 1937) *Das Allgemeingültigkeits- und somit das Erfüllbarkeitsproblem der Prädikatenlogik ist unentscheidbar. Das heißt, es gibt keinen Algorithmus, der auf Eingabe $\psi \in \mathrm{FO}$, korrekt entscheidet, ob ψ gültig ist.*

Wir werden uns im nächsten Kapitel genauer mit Unentscheidbarkeit befassen und sehen, dass nicht nur die Prädikatenlogik unentscheidbar ist, sondern viele natürliche und wichtige algorithmische Probleme in der Informatik unentscheidbar sind.

Teil IV.

Berechenbarkeit

7. Turingmaschinen und der Begriff der Berechenbarkeit

Thema dieses Teils des Skripts ist eine Einführung in die Rekursions- oder Berechenbarkeitstheorie. Die Rekursionstheorie beschäftigt sich mit der Frage nach der Berechenbarkeit algorithmischer Probleme bzw. der Berechenbarkeit von Funktionen.

7.1. Was heißt eigentlich berechenbar?

Die grundlegende Frage ist natürlich zunächst einmal, was eigentlich berechenbar bedeuten soll. Intuitiv wollen wir eine Funktion „berechenbar“ nennen, wenn wir für jedes mögliche Argument der Funktion den Funktionswert durch eine endliche Zahl von Schritten berechnen können. Für einen genauen, formalen Berechenbarkeitsbegriff reicht diese Definition natürlich nicht aus, da es ja nicht klar ist, was ein „Rechenschritt“ denn sein soll.

Zu Beginn des vorherigen Jahrhunderts haben sich daher viele Wissenschaftler (meistens aus der Mathematik) Gedanken gemacht, wie man Berechenbarkeit formal definieren kann, und damit die moderne Informatik mitbegründet.

Die für diese Vorlesung wichtigsten Ideen stammen von dem britischen Mathematiker *Alan Mathison Turing* (23. Juni 1912 – 7. Juni 1954). Turing betrachtete 1936 die Frage, was berechenbar eigentlich bedeuten soll. Er argumentierte, dass *jede* Berechnung durch die folgenden Schritte (auf einem Blatt Papier) ausgeführt werden kann:

- Man betrachtet jeweils nur einen Teil des Problems, d.h. nur ein Symbol auf dem Papier.
- Je nachdem was dort gelesen wird, wechselt man in einen neuen (Geistes-) Zustand, z.B. in dem man sich die dort gelesene Information merkt. Da wir uns aber nur endlich viele Dinge merken können, gibt es auch nur endlich viele verschiedene Zustände, in denen wir uns befinden können.
- Wir modifizieren dann diesen Teil des Problems, z.B. indem wir das gerade gelesene Symbol durch ein anderes überschreiben. Auch hier gibt es nur endlich viele verschiedene Zeichen, die wir benutzen.
- Danach betrachtet man einen anderen Teil des Problems und wiederholt diese Schritte.

- Dies wird solange gemacht, bis das Ergebnis berechnet ist.

Turing argumentierte (überzeugend), dass dies abstrakt gesehen genau das ist, was Mathematiker machen, wenn sie etwas berechnen. Diese Idee überführte er dann in folgendes Maschinen-Modell, welches heute zu seinen Ehren als Turingmaschinen bezeichnet wird. Wir haben Turingmaschienen schon im Kapitel 3.3 gesehen. Da wir dort die feine Unterscheidung zwischen P und NP brauchten, haben wir dort nicht-deterministische Turingmaschinen betrachtet. Für die hier betrachtete Rekursionstheorie ist das aber unerheblich, so dass wir im Folgenden nur deterministische Turingmaschinen einführen werden.

7.2. Turingmaschinen

7.2.1. 1-Band-Turingmaschinen

Definition 7.1 Eine *deterministische Turingmaschine* ist ein 6-Tupel

$$M := (Q, \Sigma, \Gamma, \delta, q_0, F),$$

wobei

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

Eine Turingmaschine arbeitet auf einem beidseitig unbeschränkten *Arbeitsband*. Die Bandpositionen sind indiziert durch ganze Zahlen. An jeder Bandposition steht ein Element aus Γ. Wir beschreiben daher den Zustand eines Bandes durch eine Funktion $f\colon \mathbb{Z} \to \Gamma$. Zu Beginn der Berechnung steht die Eingabe $w \subseteq \Sigma^*$ auf den Positionen 0 bis $|w|$ des Eingabebandes, wobei $|w|$ wie üblich die Länge von w bezeichnet. Die Turingmaschine hat einen Lese/Schreib-Kopf, der zu Beginn der Berechnung auf Position 0 des Bandes steht. Darüber hinaus hat die Turingmaschine die Menge Q von Zuständen, in denen sie sich befinden kann. In jedem Berechnungsschritt kann die Maschine nun das Symbol an der aktuellen Position lesen und abhängig vom gelesenen Symbol und ihrem aktuellen Zustand das Symbol an der aktuellen Position durch ein anderes ersetzen, ihren eigenen Zustand ändern und danach den Lese/Schreib-Kopf nach links oder rechts bewegen (oder aber einfach stehen lassen). Sobald die Maschine in einen Endzustand $F \subseteq Q$ wechselt, ist die Berechnung beendet und die Ausgabe der Berechnung ist dann der Inhalt des Arbeitsbandes.

Nach obiger Beschreibung ist der aktuelle Zustand einer Turingmaschine also vollständig durch den Inhalt des Arbeitsbandes, der aktuellen Position des Lese/Schreibkopfes sowie dem aktuellen Zustand der Maschine beschrieben. Dies wird als *Konfiguration* bezeichnet und wie folgt formal definiert.

Definition 7.2 Eine *Konfiguration* (f, n, q) einer Turingmaschine besteht aus dem Zustand des Bandes, d.h. einer Funktion $f\colon \mathbb{Z} \to \Gamma$, der Position der Turingmaschine auf dem Band, d.h. einer Zahl $n \in \mathbb{Z}$, sowie dem Zustand $q \in Q$ der Turingmaschine. Es sind ausschließlich die Konfigurationen zulässig, in denen f nur auf endlich vielen Stellen ungleich $\square$ ist. ⊣

Die Start- oder Anfangskonfiguration einer TM ist wie folgt definiert.

Definition 7.3 Auf einem Eingabewort $w \in \Sigma^*$ ist die *Startkonfiguration* einer Turingmaschine $M := (Q, \Sigma, \Gamma, \delta, q_0, F)$ das Tupel $(f, 0, q_0)$, wobei $f(i) = w_i$ ist für alle $i \in \{0, \ldots, |w| - 1\}$ und $f(i) = \square$ für alle restlichen i. ⊣

Definition 7.4 Zu jeder Konfiguration (f, n, q) gibt es höchstens eine *Nachfolgekonfiguration*. Falls $q \in F$, dann hat (f, n, q) keine Nachfolgekonfiguration. Anderenfalls sei $\delta(q, f(n)) = (q', x, s)$. Dann ist die Nachfolgekonfiguration $(f[n \mapsto x], n + s, q')$. Hierbei bezeichnet $f[n \mapsto x]$ die Funktion f' mit $f'(n) = x$ und $f'(i) = f(i)$ für alle $i \neq n$. ⊣

Definition 7.5 Der *Lauf* einer Turingmaschine M auf einem Wort $w \in \Sigma^*$ beginnt in der Startkonfiguration und geht solange wie möglich zur Nachfolgekonfiguration über. Falls irgendwann eine Konfiguration erscheint, die keine Nachfolgekonfiguration hat, dann *terminiert* M auf der Eingabe w. Das Ergebnis von M ist der Bandzustand in der letzten Konfiguration. ⊣

7.2.2. Varianten von Turingmaschinen

Das hier beschriebene Turingmaschinenmodell wird als 1-Band Turingmaschine bezeichnet. Daneben gibt es noch Mehrband-Turingmaschinen, die statt eines Eingabebandes k verschiedene Bänder haben. Jedes Band hat einen eigenen Schreib/Lese Kopf. In jedem Schritt kann jeder dieser Köpfe separat bewegt werden, sodass die Maschine simultan auf allen Bändern arbeitet. Daher ist die Transitionsfunktion in diesem Modell eine Funktion

$$\delta : (Q \setminus F) \times \Gamma^k \to Q \times (\Gamma \times \{-1, 0, 1\})^k.$$

Allerdings können mit k-Band-Turingmaschinen nur dieselben Funktionen berechnet werden, wie durch 1-Band-Turingmaschinen.

7.3. Entscheidbare und semi-entscheidbare Sprachen

Die bisher beschriebenen Turingmaschinen berechnen Funktionen, da sie zu einer Eingabe eine passende Ausgabe berechnen (falls sie terminieren). In dieser Vorlesung interessieren wir uns mehr für Entscheidungsprobleme, also Probleme, die eine Eingabe bekommen und entscheiden sollen, ob diese Eingabe bestimmte Eigenschaften haben. Ein Beispiel eines solchen Problems ist das 3-Färbbarkeitsproblem für Graphen, bei denen die Eingabe ein Graph ist und wir entscheiden müssen, ob die Knoten des Graphen so mit drei Farben gefärbt werden können, dass für keine Kante beide Endpunkte gleich gefärbt sind. Ein anderes Beispiel wäre ein Syntax-Checker für die Programmiersprache C, wo die Eingabe ein Programm ist und wir entscheiden müssen, ob es ein syntaktisch korrektes C-Programm ist.

Für solche Entscheidungsprobleme besteht die Ausgabe einfach aus "Ja" oder "Nein". Wir werden daher im Folgenden mit einer Variante von Turingmaschinen arbeiten, den sogenannten *Turing-Akzeptoren*, die die Eingabe entweder *akzeptieren* oder *verwerfen*, aber selbst keine Ausgabe liefern.

Definition 7.6 Ein *Turing-Akzeptor* ist eine Turingmaschine, deren Endzustandsmenge F in zwei disjunkte Mengen F_a, F_r partitioniert ist. Die Zustände in F_a werden *akzeptierend* genannt, die in F_r *verwerfend*. ⊣

Ein Turing-Akzeptor operiert auf einer Eingabe $w \in \Sigma^*$ genau wie eine Turingmaschine. Wenn die Maschine in einen Endzustand q geht, dann sagen wir, dass die Eingabe akzeptiert wurde, wenn q ein akzeptierender Zustand ist, ansonsten verwirft die Maschine.

Definition 7.7 Sei $\mathcal{M} := (Q, \Sigma, \Gamma, \delta, q_o, F_a, F_r)$ ein Turing-Akzeptor. Die durch $\mathcal{M}$ *akzeptierte Sprache* $\mathcal{L}(\mathcal{M}) \subseteq \Sigma^*$ ist definiert als

$$\mathcal{L}(\mathcal{M}) := \{w \in \Sigma^* : \mathcal{M} \text{ akzeptiert } w\}.$$ ⊣

Man beachte, dass ein Turing-Akzeptor ein Wort als Eingabe verwerfen kann, indem er in einem verwerfenden Zustand anhält oder aber indem er einfach unendlich lange läuft, d.h. nicht terminiert.

Definition 7.8 Sei Σ ein endliches Alphabet.

(1) Eine Sprache $\mathcal{L} \subseteq \Sigma^*$ heißt *akzeptierbar*, oder *semi-entscheidbar*, wenn es einen Turing-Akzeptor $\mathcal{M}$ mit $\mathcal{L} = \mathcal{L}(\mathcal{M})$ gibt.

(2) Eine Sprache $\mathcal{L} \subseteq \Sigma^*$ heißt *entscheidbar*, wenn es einen Turing-Akzeptor $\mathcal{M}$ gibt mit $\mathcal{L} = \mathcal{L}(\mathcal{M})$, der auf allen Eingaben terminiert. Wir sagen, dass $\mathcal{M}$ die Sprache $\mathcal{L}$ *entscheidet*. ⊣

Der Unterschied zwischen semi-entscheidbaren und entscheidbaren Sprachen besteht also darin, dass eine Turingmaschine, die eine Sprache entscheidet, für jede

Eingabe wirklich „entscheiden“ muss, ob sie in der Sprache ist oder nicht. D.h. sie muss irgendwann anhalten und diese Entscheidung treffen und kann nicht einfach dadurch verwerfen, dass sie unendlich lange läuft und die Entscheidung vermeidet. Dadurch wird zum Beispiel folgende Strategie einer Turingmaschine ein Problem zu entscheiden verhindert: Auf einer Eingabe sucht die Maschine einfach nacheinander alle möglichen Lösungen durch. Findet sie eine, so terminiert sie und akzeptiert. Ansonsten läuft sie einfach weiter und versucht die nächste mögliche Lösung. Bei Problemen, bei denen die Zahl der potentiellen Lösungen unendlich ist, läuft die Maschine also unter Umständen unendlich lange.

Wir könnten zum Beispiel eine Turingmaschine bauen, die C-Programme als Eingabe nimmt und entscheiden soll, ob diese niemals in eine Endlosschleife geraten können. Sie soll also nur die Programme akzeptieren, die nur endlich lange laufen. Die Turingmaschine könnte das dadurch machen, dass sie das C-Programm einfach laufen lässt und abwartet, ob es endet. Bei Programmen, die nur endlich lange laufen, kann die TM also terminieren und akzeptieren. Bei anderen Programmen aber läuft sie selbst unendlich lange und verwirft daher nur durch Nicht-Terminierung. Für praktische Anwendungen ist das Verwerfen durch Nicht-Terminierung natürlich uninteressant, da wir ja irgendwann eine Antwort haben wollen. Wir werden später aber sehen, dass es keine Turingmaschine gibt, die diese Sprache entscheidet.

Die folgende Proposition liefert eine erste leichte Klassifizierung von Sprachen bezüglich ihrer Berechenbarkeit.

Proposition 7.9 *Sei Σ ein endliches Alphabet.*

(1) *Jede entscheidbare Sprache $\mathcal{L} \subseteq \Sigma^*$ ist semi-entscheidbar.*

(2) *Eine Sprache $\mathcal{L} \subseteq \Sigma^*$ ist genau dann entscheidbar, wenn $\mathcal{L}$ und $\Sigma^* \setminus \mathcal{L}$ semi-entscheidbar sind.*

Beweis. Teil 1 folgt sofort aus Definition 7.8. Für Teil 2 seien also $\mathcal{L}$ und $\Sigma^* \setminus \mathcal{L}$ semi-entscheidbar. Also existieren Turing-Akzeptoren $\mathcal{M}$ und $\mathcal{M}'$ mit $\mathcal{L}(\mathcal{M}) = \mathcal{L}$ und $\mathcal{L}(\mathcal{M}') = \Sigma^* \setminus \mathcal{L}$.

Wir benutzen $\mathcal{M}$ und $\mathcal{M}'$, um einen neuen Turing-Akzeptor $\mathcal{N}$ zu konstruieren, der $\mathcal{L}$ entscheidet. Die Idee dabei ist, auf Eingabe $w \in \Sigma^*$, die beiden Maschinen $\mathcal{M}$ und $\mathcal{M}'$ *parallel* auf w laufen zu lassen. Da $\mathcal{L}(\mathcal{M}) = \Sigma^* \setminus \mathcal{L}(\mathcal{M}')$, muss genau eine der beiden Maschinen terminieren und akzeptieren. Wenn $\mathcal{M}$ akzeptiert, dann akzeptiert $\mathcal{N}$, da dann $w \in \mathcal{L}$, und wenn $\mathcal{M}'$ akzeptiert, dann verwirft $\mathcal{N}$, da dann $w \notin \mathcal{L}$. □

Manchmal werden wir nicht nur entscheidbare Sprachen, sondern auch berechenbare Funktionen benötigen. Sei Σ ein endliches Alphabet.

Definition 7.10 Eine Funktion $f\colon \Sigma^* \to \Sigma^*$ heißt *berechenbar*, wenn es eine Turingmaschine $\mathcal{M}$ gibt, die auf jeder Eingabe $w \in \Sigma^*$ terminiert und zum Schluss $f(w)$ auf ihr Band schreibt, sodass auf allen restlichen Bandpositionen □ steht. ⊣

8. Berechenbarkeit

In diesem Abschnitt werden wir zeigen, dass es Sprachen (algorithmische Probleme) gibt, die nicht entschieden werden können, egal wieviel Zeit wir dafür erlauben. Eines der ersten solcher Probleme war das *Halteproblem*. Dieses Problem ist die Frage „Hält eine gegebene Turingmaschine auf einer gegebenen Eingabe in endlicher Zeit an?“.

Um dieses Problem formal beschreiben zu können, müssen wir erklären, was es bedeutet, einer Turingmaschine eine andere Turingmaschine und ihre Eingabe zu übergeben.

8.1. Kodierung von Turingmaschinen

Wenn wir Turingmaschinen als Wörter über einem Alphabet kodieren können, können wir Turingmaschinen als Eingabe an andere Turingmaschinen übergeben. Hierzu benötigen wir zuerst ein geeignetes Alphabet. Solange das Alphabet mindestens zwei verschiedene Symbole enthält, damit Binärkodierung möglich ist, ist es letztendlich irrelevant, welches Alphabet wir nehmen. Ein größeres Alphabet kann ggf. die Notation vereinfachen, daher werden wir das Alphabet

$$\Sigma = \{0, \ldots, 9, a, \ldots, z, (,), \mapsto, \square\}$$

nehmen.

Angenommen, $\mathcal{M} := (Q, \Sigma', \Gamma, \delta, q_0, F)$ ist eine 1-Band-Turingmaschine mit

$$\delta : \big(Q \setminus F\big) \times \Gamma \to Q \times \Gamma \times \{-1, 0-1\},$$
$$\delta := \left\{ \begin{array}{lcl} (q_0, a) & \mapsto & (q_1, a, 1) \\ \ldots & \mapsto & \ldots \end{array} \right\}.$$

Wir können immer annehmen, dass $Q := \{0, \ldots, n\}$ für irgendein n, d.h. die Zustände sind natürliche Zahlen von 0 bis n. Falls die Zustände anders heißen sollten, können wir die Zustände vorher umbenennen, ohne die Berechnungen der Turingmaschine zu verändern.

Wir kodieren nun $\mathcal{M}$ als

$$(n)\#q_0\#(\text{Liste der akzeptierenden Zustände})\#(\delta \text{ zeilenweise}).$$

Listen kodieren wir als eine #-getrennte Aufzählung ihrer Elemente. Elemente aus Γ kodieren wir binär mit 0 und 1.

Sei zum Beispiel $\mathcal{M}_1 = (\{0,1\}, \{a,b\}, \{a,b,\square\}, \delta, 0, \{1\})$ eine 1-Band-Turingmaschine mit

$$\delta := \left\{ \begin{array}{lcl} (0, a) & \mapsto & (0, a, 1) \\ (0, b) & \mapsto & (1, b, 0) \\ (0, \square) & \mapsto & (0, \square, 0). \end{array} \right.$$

Um $\mathcal{M}_1$ zu kodieren, müssen wir zuerst das Alphabet von $\mathcal{M}_1$ kodieren. Wir nehmen als Kodierung

$$a \mapsto 00 \qquad b \mapsto 01 \qquad \square \mapsto 10.$$

Die Kodierung von $\mathcal{M}_1$ ist dann

$$(1)\#0\#(1)\#(\underbrace{(0\#00) \mapsto (0\#00\#1)}_{\text{die erste Zeile von } \delta}\#$$
$$(0\#01) \mapsto (1\#01\#0)\#(0\#10) \mapsto (0\#10\#0)).$$

Wie bereits oben angedeutet, kann $\mathcal{M}$ ebensogut über dem Binäralphabet $\{0,1\}$ kodiert werden. Am einfachsten ist das zu erreichen, indem wir jedes Symbol mit einer festen Anzahl Bits kodieren und dann die oben angegebene Kodierung ins Binäralphabet übertragen.

Wir bezeichnen die Kodierung von $\mathcal{M}$ im Binäralphabet mit $\text{enc}(\mathcal{M})$. Wir können auch beliebige Wörter w über dem Eingabealphabet von $\mathcal{M}$ im Binäralphabet kodieren. Eine solche Kodierung bezeichnen wir analog mit $\text{enc}(w)$.

Analog zu 1-Band-Turingmaschinen können wir auch k-Band-Turingmaschinen kodieren, indem wir als erstes die Zahl der Bänder in die Kodierung schreiben und danach genauso wie oben die Maschine kodieren.

8.1.1. Ein Zählargument

Wir haben im letzten Abschnitt gesehen, dass über dem Alphabet $\{0,1\}$ jede Turingmaschine als Wort kodiert werden kann. Die Menge der Wörter über $\{0,1\}$ ist abzählbar unendlich.

Andererseits gibt es aber überabzählbar viele Sprachen.

Lemma 8.1 *Es gibt überabzählbar viele Sprachen über* $\{0,1\}$.

Beweis. Nach Definition sind Sprachen Teilmengen von $\{0,1\}^*$. Also ist die Menge der Sprachen genau die Potenzmenge von $\{0,1\}^*$. Die Potenzmenge einer abzählbar unendlichen Menge ist überabzählbar. □

Es gibt also mehr Sprachen als Turingmaschinen. Daraus folgt direkt die Existenz mindestens einer Sprache, die von keiner Turingmaschine entschieden wird.

Proposition 8.2 *Es gibt unentscheidbare Sprachen.*

8.2. Das Halteproblem

Das Argument im vorhergehenden Abschnitt zeigt nur, dass es unentscheidbare Sprachen gibt. Es bleibt die Frage, ob wir auch ein konkretes Beispiel einer solchen Sprache finden können. Eine der berühmtesten unentscheidbaren Sprachen ist das sogenannte *Halteproblem*.

HALTEPROBLEM	
Eingabe:	Eine Turingmaschine $\mathcal{M}$ und eine Eingabe w
Problem:	Hält $\mathcal{M}$ auf Eingabe w?

Wenn wir uns auf eine Kodierung von Turingmaschinen festgelegt haben, können wir dieses Problem auch beschreiben durch die Sprache

$$\text{HALTEPROBLEM} := \{\text{enc}(\mathcal{M})\#\text{enc}(w) : \mathcal{M} \text{ hält auf } w\}.$$

Theorem 8.3 *Es gilt:*

(1) *Das Halteproblem ist semi-entscheidbar.*

(2) *Das Halteproblem ist unentscheidbar.*

Wir führen den Beweis in mehreren Schritten. Der wichtigste Schritt ist die Konstruktion einer universellen Turingmaschine, d.h. ein Interpreter für Turingmaschinen. Um diese formal anzugeben, ist es sehr nützlich, Turingmaschinen mit mehreren Bändern zu verwenden. Wir werden daher zuerst zeigen, dass jede k-Band-Turingmaschine auf eine 1-Band-Maschine reduziert werden kann.

Lemma 8.4 *Sei $\mathcal{L} \subseteq \Sigma^*$ eine Sprache. Wenn $\mathcal{L}$ durch einen k-Band Turing-Akzeptor entschieden werden kann, dann kann $\mathcal{L}$ durch einen 1-Band Turing-Akzeptor entschieden werden.*

Beweis. Sei $\mathcal{M}$ ein k-Band-Turing-Akzeptor für $\mathcal{L}$. Wir möchten einen 1-Band-Turing-Akzeptor $\mathcal{M}'$ konstruieren, der ebenfalls $\mathcal{L}$ entscheidet. Eine Möglichkeit ist, dass $\mathcal{M}'$ auf sein eines Band die k Bänder von $\mathcal{M}$ abwechselnd schreibt.

Sei $b_{i,j}$ das Symbol an der j-ten Bandposition auf Band i für irgendeine k-Bandkonfiguration von $\mathcal{M}$. Das heißt $i \in \{1, \ldots, k\}$ und $j \in \mathbb{Z}$. Aus diesen k Bändern können wir ein einziges Band definieren, indem wir die k Bänder abwechselnd schreiben:

$$\cdots b_{1,-1}b_{2,-1}\cdots b_{k,-1}b_{1,0}b_{2,0}\cdots b_{k,0}b_{1,1}b_{2,1}\cdots b_{k,1}b_{1,2}b_{2,2}\cdots b_{k,2}\cdots,$$

wobei $b_{1,0}$ an der Position 0 auf dem Band stehen soll.

Außerdem verdoppeln wir das Arbeitsalphabet Γ und verwenden $\Gamma' := \Gamma \times \{0, 1\}$ für die 1-Band-Turingmaschine. Die zweite Komponente der Elemente aus Γ' markiert die aktuelle Position des Kopfes auf Band $i \in \{1, \ldots, k\}$, indem für jedes i genau ein $b_{i,j}$ in der zweiten Komponente eine 1 stehen hat. □

Im nächsten Schritt werden wir eine *universelle Turingmaschine $\mathcal{U}$* definieren. Eine solche Maschine bekommt als Eingabe Wörter der Form $\text{enc}(\mathcal{M})\#\text{enc}(w)$, wobei $\text{enc}(\mathcal{M})$ die Kodierung einer Turingmaschine $\mathcal{M}$ und $\text{enc}(w)$ die Kodierung einer Eingabe w für $\mathcal{M}$ ist.

Die Maschine $\mathcal{U}$ simuliert nun die Berechnung von $\mathcal{M}$ auf w und akzeptiert genau dann, wenn $\mathcal{M}$ akzeptiert. Universelle Turingmaschinen sind in diesem Sinne

Interpreter; sie interpretieren eine gegebene Turingmaschine. Universelle Turingmaschinen sind daher vergleichbar mit Interpretern, die man aus der Praxis kennt. Beispielsweise gibt es Interpreter für die Programmiersprache Lisp, die selbst in Lisp geschrieben sind. Es ist daher nicht weiter schwer zu zeigen, dass eine universelle Turingmaschine existiert.

Theorem 8.5 *Es gibt eine universelle Turingmaschine.*

Sei $\mathcal{U}$ eine solche universelle Turingmaschine. Wir können nun eine neue Turingmaschine $\mathcal{A}$ definieren, die das Halteproblem semi-entscheidet. Dazu verhält sich $\mathcal{A}$ auf Eingabe $\text{enc}(\mathcal{M})\#\text{enc}(w)$ genauso wie $\mathcal{U}$, nur mit dem Unterschied, dass $\mathcal{A}$ immer akzeptiert, wenn die Simulation von $\mathcal{M}$ auf w anhält. Das bedeutet, dass es drei mögliche Ergebnisse von $\mathcal{A}$ gibt:

(1) $\mathcal{A}$ akzeptiert. Das impliziert, dass $\mathcal{M}$ auf w terminiert.

(2) $\mathcal{A}$ terminiert und verwirft. Das impliziert, dass die Eingabe nicht die Form $\text{enc}(\mathcal{M})\#\text{enc}(w)$ hatte.

(3) $\mathcal{A}$ terminiert nicht. Das impliziert, dass $\mathcal{M}$ auf w nicht terminiert.

Die Maschine $\mathcal{A}$ ist also ein Turing-Akzeptor für das Halteproblem. Damit haben wir gezeigt

Theorem 8.6 *Das Halteproblem ist semi-entscheidbar.*

Unser Ziel war jedoch die Unentscheidbarkeit des Halteproblems zu beweisen. Hierfür führen wir einen Beweis durch Widerspruch. Angenommen, es gibt eine Maschine $\mathcal{B}$, die das Halteproblem entscheidet. Wir konstruieren eine neue Maschine $\mathcal{B}'$ wie folgt.

(1) Auf Eingabe $\text{enc}(\mathcal{M})$ simuliert $\mathcal{B}'$ die Maschine $\mathcal{B}$ auf Eingabe $\text{enc}(\mathcal{M})\#\text{enc}(\mathcal{M})$.

(2) Falls $\mathcal{B}$ akzeptiert, geht $\mathcal{B}'$ in eine Endlosschleife. Falls $\mathcal{B}$ verwirft, akzeptiert $\mathcal{B}'$.

Die Maschine $\mathcal{B}'$ verhält sich dann auf der Eingabe $\text{enc}(\mathcal{B}')$ interessant. Auf dieser Eingabe simuliert $\mathcal{B}'$ im ersten Schritt die Maschine $\mathcal{B}$ mit Eingabe $\text{enc}(\mathcal{B}')\#\text{enc}(\mathcal{B}')$. Die Maschine $\mathcal{B}$ entscheidet nach Annahme das Halteproblem, d.h. $\mathcal{B}$ akzeptiert genau dann, wenn $\mathcal{B}'$ auf der Eingabe $\text{enc}(\mathcal{B}')$ terminiert. Wir behaupten, dass, egal wie $\mathcal{B}$ antwortet, ein Widerspruch entsteht.

Falls $\mathcal{B}$ akzeptiert, dann geht $\mathcal{B}'$ in eine Endlosschleife und terminiert nicht. Falls $\mathcal{B}$ verwirft, terminiert $\mathcal{B}'$. In jedem Fall ist die Antwort von $\mathcal{B}$ falsch. Das heißt, entgegen unserer Annahme entscheidet $\mathcal{B}$ das Halteproblem nicht. Wir haben damit das folgende Theorem bewiesen.

Theorem 8.7 *Das Halteproblem ist unentscheidbar.*

Wir erinnern uns an Proposition 7.9: Eine Sprache $\mathcal{L} \subseteq \Sigma^*$ ist entscheidbar genau dann, wenn $\mathcal{L}$ und $\Sigma^* \setminus \mathcal{L}$ semi-entscheidbar sind. Damit bekommen wir als Korollar aus Theorem 8.7, dass das Komplement des Halteproblems nicht semi-entscheidbar ist.

Korollar 8.8 *Die Sprache*

$$\overline{\textsc{Halteproblem}} = \{\text{enc}(\mathcal{M})\#\text{enc}(w) : \mathcal{M} \textit{ hält nicht auf } w\}$$

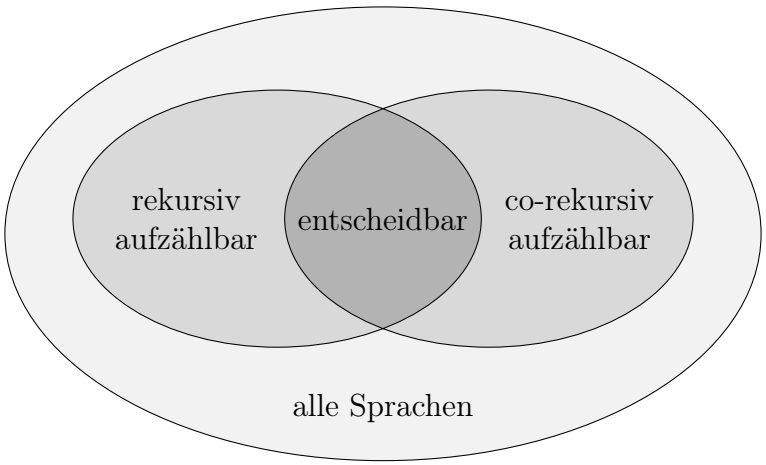
ist nicht semi-entscheidbar.

8.3. Klassifikation von Sprachen

Definition 8.9 Eine Sprache $\mathcal{L} \subseteq \Sigma^*$ ist *co-rekursiv aufzählbar*, oder *co-r.e.*, wenn $\Sigma^* \setminus \mathcal{L}$ rekursiv aufzählbar ist. ⊣

Beispielsweise ist die Sprache $\overline{\textsc{Halteproblem}}$ co-rekursiv aufzählbar, aber nicht rekursiv aufzählbar.

Die Relationen zwischen rekursiv aufzählbar, co-rekursiv aufzählbar und entscheidbar sind noch einmal in dem folgenden Diagramm dargestellt.



Wir wollen zeigen, dass die folgenden Probleme ebenfalls unentscheidbar sind.

ε-Halteproblem	
Eingabe:	Turing-Akzeptor $\mathcal{M}$
Problem:	Hält $\mathcal{M}$ auf dem leeren Wort als Eingabe?

Äquivalenz	
Eingabe:	Turing-Akzeptoren $\mathcal{M}$ und $\mathcal{M}'$
Problem:	Gilt $\mathcal{L}(\mathcal{M}) = \mathcal{L}(\mathcal{M}')$?

Leerheit	
Eingabe:	Turing-Akzeptor $\mathcal{M}$
Problem:	Ist $\mathcal{L}(\mathcal{M}) = \varnothing$?

Damit wir nicht für jedes dieser Probleme einen Unentscheidbarkeitsbeweis ähnlich wie beim Halteproblem führen müssen, führen wir in diesem Abschnitt das Werkzeug der *Reduktionen* ein. Reduktionen sind ein sehr mächtiges Werkzeug, um die Unentscheidbarkeit von Problemen zu beweisen.

Die Grundidee einer Reduktion ist, ein Problem $\mathcal{A}$ bekannter Schwierigkeit zurückzuführen auf ein Problem $\mathcal{B}$ unbekannter Schwierigkeit. Gelingt uns das, dann wissen wir, dass $\mathcal{B}$ mindestens so schwierig ist wie $\mathcal{A}$. In unserem Fall heißt „schwierig“ normalerweise „unentscheidbar“, und das Problem $\mathcal{A}$ ist oft das Halteproblem.

Solche Reduktionen bezeichnen wir als *Many-One-Reduktionen.*

8.3.1. Many-One-Reduktionen

Wir erinnern uns dazu an das Konzept der Polynomialzeit-Reduktionen. Wir werden dieses Konzept hier geeignet erweitern.

Definition 8.10 Eine Sprache $\mathcal{A}$ ist *many-one-reduzierbar* auf eine Sprache $\mathcal{B}$, wenn es eine totale und berechenbare Funktion $f\colon \Sigma^* \to \Sigma^*$ gibt, so dass für alle $w \in \Sigma^*$ gilt

$$w \in \mathcal{A} \quad \Longleftrightarrow \quad f(w) \in \mathcal{B}.$$

Wir schreiben hierfür $\mathcal{A} \leqslant_m \mathcal{B}$. ⊣

Lemma 8.11 *Wenn $\mathcal{A} \leqslant_m \mathcal{B}$ und $\mathcal{B}$ entscheidbar ist, dann ist auch $\mathcal{A}$ entscheidbar.*

Beweis. Angenommen $\mathcal{A} \leqslant_m \mathcal{B}$ und $\mathcal{B}$ wird entschieden von einem Turing-Akzeptor $\mathcal{M}$. Wir definieren einen Turing-Akzeptor $\mathcal{M}'$, der auf einer Eingabe x zuerst $f(x)$ berechnet und dann die Maschine $\mathcal{M}$ auf dem Wort $f(x)$ ausführt.

Die Maschine $\mathcal{M}$ akzeptiert $f(x)$ genau dann, wenn $f(x) \in \mathcal{B}$. Außerdem gilt nach Annahme $f(x) \in \mathcal{B}$ genau dann, wenn $x \in \mathcal{A}$. Das heißt, $\mathcal{M}'$ entscheidet die Sprache $\mathcal{A}$. □

Korollar 8.12 *Wenn $\mathcal{A}$ unentscheidbar ist und $\mathcal{A} \leqslant_m \mathcal{B}$, dann ist $\mathcal{B}$ ebenfalls unentscheidbar.*

Wenn wir also eine Sprache $\mathcal{B}$ haben und es uns gelingt, HALTEPROBLEM $\leqslant_m \mathcal{B}$ zu zeigen, dann ist $\mathcal{B}$ unentscheidbar.

Wir beobachten ein paar weitere Eigenschaften von Many-One-Reduktionen:

Lemma 8.13 $\leqslant_m$ *ist* reflexiv *und* transitiv*, d.h. wenn $\mathcal{A} \leqslant_m \mathcal{B}$ und $\mathcal{B} \leqslant_m \mathcal{C}$, dann $\mathcal{A} \leqslant_m \mathcal{C}$.*

Beweis. Durch Kompositionen der Funktionen. □

Lemma 8.14 *Wenn $\mathcal{A}$ entscheidbar und $\mathcal{B}$ eine Sprache außer $\varnothing$ und Σ^* ist, dann ist $\mathcal{A} \leqslant_m \mathcal{B}$.*

Beweis. Da $\mathcal{B} \neq \varnothing$ und $\mathcal{B} \neq \Sigma^*$, gibt es $w_a \in \mathcal{B}$ und $w_r \notin \mathcal{B}$.

Für $w \in \Sigma^*$ definieren wir

$$f(w) := \begin{cases} w_a & \text{wenn } w \in \mathcal{A}, \\ w_r & \text{wenn } w \notin \mathcal{A}. \end{cases}$$

Offensichtlich ist die Funktion f berechenbar. □

Das gerade bewiesene Lemma zeigt, dass Many-One-Reduktionen zu schwach sind, um zwischen entscheidbaren Sprachen zu unterscheiden.

Um unser ursprüngliches Problem zu lösen, werden wir die folgende Reduktionskette zeigen:

$$\textsc{Halteproblem} \leqslant_m \varepsilon\text{-}\textsc{Halteproblem} \leqslant_m \textsc{Äquivalenz}.$$

Weil wir damit das Halteproblem auf ε-Halteproblem und Äquivalenz reduziert haben, sind dann diese beiden Probleme unentscheidbar.

Lemma 8.15 $\textsc{Halteproblem} \leqslant_m \varepsilon\text{-}\textsc{Halteproblem}$

Beweis. Wir definieren eine Funktion f, sodass

$$w \in \textsc{Halteproblem} \iff f(w) \in \varepsilon\text{-}\textsc{Halteproblem}.$$

Für $w := \text{enc}(\mathcal{M})\#\text{enc}(v)$ berechnen wir eine Turingmaschine $\mathcal{M}_w$ mit folgendem Berechnungsverhalten:

(1) Schreibe v auf das Eingabeband.

(2) Simuliere $\mathcal{M}$.

Offensichtlich akzeptiert $\mathcal{M}_w$ das leere Wort genau dann, wenn $\mathcal{M}$ die Eingabe v akzeptiert.

Sei $\mathcal{M}'$ eine Turingmaschine, die auf dem leeren Wort nicht terminiert.

Definiere

$$f(w) := \begin{cases} \mathcal{M}_w & \text{wenn } w = \text{enc}(\mathcal{M})\#\text{enc}(v), \\ \mathcal{M}' & \text{wenn } w \text{ nicht die gewünschte Form hat.} \end{cases}$$

Die Funktion f ist eine Many-One-Reduktion. Sei $\mathcal{M}$ eine Turingmaschine, v eine Eingabe für $\mathcal{M}$ und sei $w := \text{enc}(\mathcal{M})\#\text{enc}(v)$. Falls $\mathcal{M}$ auf v terminiert, dann ist $w \in \textsc{Halteproblem}$ und $f(w) = \mathcal{M}_w$. Die Maschine $\mathcal{M}_w$ wird dann auf dem leeren Wort terminieren, denn sie führt $\mathcal{M}$ auf v aus.

Für die andere Richtung nehmen wir an, dass $\mathcal{M}_w$ auf dem leeren Wort terminiert. Das bedeutet aber genau, dass $\mathcal{M}$ auf v terminiert, also $w \in \textsc{Halteproblem}$. □

Lemma 8.16

$$\varepsilon\text{-Halteproblem} \leqslant_m \text{Äquivalenz}.$$

Beweis. Wir definieren eine Funktion f, sodass

$$w \in \varepsilon\text{-Halteproblem} \iff f(w) \in \text{Äquivalenz}.$$

Sei $\mathcal{M}_a$ eine Turingmaschine, die alle Eingaben akzeptiert. Für eine Turingmaschine $\mathcal{M}$ konstruieren wir eine Turingmaschine $\mathcal{M}^*$, die die folgende Berechnung durchführt.

(1) Lasse $\mathcal{M}$ auf der leeren Eingabe laufen.

(2) Wenn $\mathcal{M}$ hält, akzeptiere.

$\mathcal{M}^*$ ist äquivalent zu $\mathcal{M}_a$ genau dann, wenn $\mathcal{M}$ auf der leeren Eingabe hält. Definiere

$$f(w) := \begin{cases} \text{enc}(\mathcal{M}^*)\#\text{enc}(\mathcal{M}_a) & \text{wenn } w = \text{enc}(\mathcal{M}), \\ \text{enc}(w)\#\text{enc}(\mathcal{M}_a) & \text{wenn } w \text{ nicht die gewünschte Form hat.} \end{cases}$$

□

8.3.2. Weitere unentscheidbare Probleme

Definition 8.17 Sei Σ ein endliches Alphabet.

Das Postsche Korrespondezproblem (PCP) ist das folgende Entscheidungsproblem:

PCP	
Eingabe:	Eine Menge $K = \{(x_1, y_1), \ldots, (x_k, y_k)\}$ von Wortpaaren, $x_i, y_i \in \Sigma^*$
Problem:	Gibt es eine Folge $i_1, \ldots, i_n \in \{1, \ldots, k\}$, sodass $x_{i_1} \cdots x_{i_n} = y_{i_1} \cdots y_{i_n}$?

Theorem 8.18 *Das Postsche Korrespondenzproblem ist unentscheidbar.*

Definition 8.19 Das Schnittproblem für kontextfreie Grammatiken ist folgendes Entscheidungsproblem.

Schnittproblem für kontextfreie Grammatiken	
Eingabe:	Zwei kontextfreie Grammatiken G_1, G_2
Problem:	Gilt $\mathcal{L}(G_1) \cap \mathcal{L}(G_2) = \varnothing$?

Theorem 8.20 *Das Schnittproblem für kontextfreie Grammatiken ist unentscheidbar.*

8.4. Der Satz von Rice

Eine Eigenschaft von Turingmaschinen ist eine Eigenschaft, die nicht von der konkreten Implementierung einer Maschine abhängt, sondern nur davon, was die Maschine macht. Das heißt, eine Eigenschaft einer Turingmaschine $\mathcal{M}$ darf nur von $\mathcal{L}(\mathcal{M})$ abhängen.

Formal ist eine Eigenschaft von Turingmaschinen eine Menge $\mathcal{P}$ von rekursiv aufzählbaren Sprachen. Wir nennen eine Eigenschaft $\mathcal{P}$ *trivial*, wenn $\mathcal{P} = \varnothing$ oder $\mathcal{P}$ die Menge der rekursiv aufzählbaren Sprachen ist. Eine Eigenschaft $\mathcal{P}$ ist entscheidbar, wenn die Sprache $\{\text{enc}(\mathcal{M}) \mid \mathcal{L}(\mathcal{M}) \in \mathcal{P}\}$ entscheidbar ist.

Das nächste Theorem ist auch bekannt als der Satz von Rice.

Theorem 8.21 *Jede nicht-triviale Eigenschaft von Turingmaschinen ist unentscheidbar.*

Beweis. Wir werden ε-HALTEPROBLEM auf $\mathcal{P}$ reduzieren. Ohne Beschränkung der Allgemeinheit sei $\varnothing \notin \mathcal{P}$. Sei $\mathcal{L} \in \mathcal{P}$ und sei $\mathcal{M}_L$ ein Turing-Akzeptor, der $\mathcal{L}$ akzeptiert.

Für eine gegebene Turingmaschine $\mathcal{M}$ konstruieren wir die Turingmaschine $\mathcal{M}^*$; auf Eingabe $w \in \Sigma^*$ macht $\mathcal{M}^*$ die folgende Berechnung.

(1) Simuliere $\mathcal{M}$ auf der leeren Eingabe.

(2) Wenn $\mathcal{M}$ hält, simuliere $\mathcal{M}_L$ auf w.

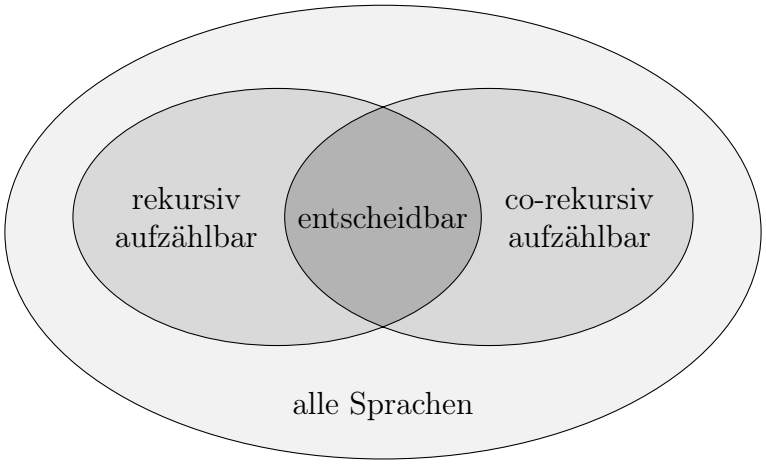
$$\text{Wenn} \quad \begin{array}{lll} \mathcal{M} \text{ auf } \varepsilon \text{ hält}, & \text{dann ist} & \mathcal{L}(\mathcal{M}^*) = \mathcal{L} \in \mathcal{P}. \\ \mathcal{M} \text{ auf } \varepsilon \text{ nicht hält}, & \text{dann ist} & \mathcal{L}(\mathcal{M}^*) = \varnothing \notin \mathcal{P}. \end{array}$$

□

8.5. Zusammenfassung

In diesem Kapitel haben wir uns mit Berechenbarkeitstheorie beschäftigt. Die Berechenbarkeitstheorie analysiert die Grenze zwischen entscheidbaren und unentscheidbaren Sprachen.

Wir erinnern noch einmal an die Relationen zwischen entscheidbaren, (co-)rekursiv aufzählbaren und allen Sprachen.



Teil V.

Anhang

A. Griechische Symbole

Anbei eine Liste einiger häufig benutzter griechischer Symbole, teilweise in Klein- und Großschreibweise, sortiert nach Verwendungsarten.

α	alpha
β	beta
γ, Γ	gamma
δ, Δ	delta
ε	epsilon
φ, Φ	phi
ψ, Ψ	psi
ϑ, Θ	theta
χ	chi
ξ, Ξ	xi
λ, Λ	lambda
ω, Ω	omega
μ	mü
ν	nü

Symbole

Index