Skript zur Vorlesung

Logik für Informatiker

SS 2008

Martin Hofmann

Lehr- und Forschungseinheit Theoretische Informatik

Institut für Informatik

Ludwig-Maximilians-Universität München

Inhaltsverzeichnis

Vorwort

Die Logik als Lehre vom *vernünftigen Schließen* beginnt in der Antike, mit Aristoteles und Euklid als bekannteste Vertreter. Aristoteles beobachtet in der menschlichen Argumentation vorkommende Schlussweisen und bezeichnet diese als *Syllogismen*. Ein einfaches Beispiel ist der *Modus Ponens*: wenn B von A impliziert wird und A vorliegt, dann darf auf B geschlossen werden. Beispiel: Aus "Wenn es regnet, dann wird die Straße nass." und "Es regnet." kann gefolgert werden, dass die Straße nass ist. Ein anderes Beispiel ist der *Modus tollens* mit dessen Hilfe aus "Wenn es regnet, dann wird die Straße nass." und "die Straße ist trocken" auf "Es regnet nicht" geschlossen werden kann. In der Sprache der modernen Aussagenlogik besagt der Modus tollens, dass aus $\phi \Rightarrow \psi$ und $\neg\psi$ auf $\neg\phi$ geschlossen werden kann.

Im allgemeinen nicht gültig ist der sog. Umkehrschluss von "Wenn A dann B" und "Nicht A" auf "Nicht B", also von "Wenn es regnet, dann wird die Straße nass." und "Es regnet nicht." auf "die Straße ist trocken."! Bei Vorschriften ist aber wegen impliziter Annahmen ein Umkehrschluss doch oft zulässig, etwa wenn es wie in Maxlrain heißt "An den gedeckten Tischen Bedienung", so kann man davon ausgehen, dass an den nicht gedeckten Tischen nicht bedient wird.

Ein Beispiel für einen komplizierteren Syllogismus, der auch Quantifizierung beinhaltet, ist der Modus Celarent, der z.B. erlaubt von "Kein vernünftiger Mensch glaubt an den Osterhasen." und "Teilnehmer dieser Veranstaltung sind vernünftige Menschen." den Schluss zu ziehen, dass kein Teilnehmer dieser Veranstaltung an den Osterhasen glaubt. In der Sprache der modernen Prädikatenlogik besagt der Modus Celarent, dass aus $\neg\exists x.A(x) \wedge B(x)$ und $\forall x.C(x) \Rightarrow A(x)$ auf $\neg\exists x.C(x) \wedge B(x)$ geschlossen werden kann.

Euklids Beitrag zur Logik liegt in der formalen Festlegung des Beweisbegriffs und der Einführung der axiomatischen Methode. Für ihn ist ein Beweis eine Reihe von Aussagen derart dass jede Aussage in der Reihe entweder ein Axiom (unbewiesene Annahme) ist, oder aber aus vorhergehenden Aussagen mit einer Schlussregel (z.B. Modus Ponens) gefolgert werden kann. Die nach ihm benannte Euklidische Geometrie beruht auf fünf Axiomen wie z.B. "durch zwei voneinander verschiedene Punkte geht genau eine Gerade". Aus ihnen folgert er rigoros die bekannten Sätze aus der Schulgeometrie, wie etwa "Stufenwinkel sind gleich", "Pythagoras", "Höhensatz".

Im zwanzigsten Jahrhundert wurde die Logik durch die Einführung der Prädikatenlogik durch Frege deutlich vereinfacht und auf ein solides Fundament gestellt. Dies erlaubte es, die gesamte Mathematik nach dem Beispiel Euklids auf wenige Grundannahmen zurückzuführen (Cantors Mengenlehre). Allerdings zeigten sich in Form von Gödels Unvollständigkeitssatz prinzipielle Grenzen der axiomatisch-logischen Methode: egal welche Axiome und (gültigen!) Schlussregeln man zugrundelegt, es gibt immer wahre Aussagen,

die sich nicht beweisen lassen.

Mit dem Aufkommen von Computern gewann die Logik eine zusätzliche Bedeutung, so z.B. in der Programmverifikation, der Spezifikation, der automatischen Programmanalyse, der Datenbanktheorie, der Wissensrepräsentation, der künstlichen Intelligenz. Darüberhinaus wurde es mit Computern erstmals möglich, auch große Beweise tatsächlich in dem von Euklid geforderten Detaillierungsgrad zu erzeugen und zu überprüfen. Solche als Theorembeweiser bezeichnete Systeme erlauben die interaktive Generierung von Beweisen am Computer, welcher die Verwaltung und Korrektheitsprüfung der einzelnen Schritte übernimmt und außerdem Teilaufgaben durch Entscheidungsprozeduren selbst lösen kann.

Detaillierteres Material zur Geschichte der Logik findet sich auf der gut gemachten Wikipediaseite `http://de.wikipedia.org/wiki/Logik`.

1 Aussagenlogik

1.1 Syntax der Aussagenlogik

Definition 1
Sei eine Menge $\mathcal{A}$ von Aussagenvariablen $A, B, C, D \ldots$ gegeben.

Die *aussagenlogischen Formeln* über $\mathcal{A}$ sind durch folgende BNF Grammatik definiert.

$$
\begin{array}{lcll}
\mathcal{F} & ::= & \mathcal{A} & \\
 & | & \neg\mathcal{F} & \text{(Negation, Verneinung)} \\
 & | & \mathcal{F} \wedge \mathcal{F} & \text{(Konjunktion, logisches Und)} \\
 & | & \mathcal{F} \vee \mathcal{F} & \text{(Disjunktion, logisches Oder)} \\
 & | & \mathcal{F} \Rightarrow \mathcal{F} & \text{(Implikation, Wenn-Dann-Beziehung)} \\
 & | & \mathcal{F} \Leftrightarrow \mathcal{F} & \text{(Äquivalenz, Genau–Dann-Wenn-Beziehung)} \\
 & | & \top & \text{Verum, wahre Formel} \\
 & | & \bot & \text{Falsum, falsche Formel}
\end{array}
$$

Die Symbole $\{\neg, \wedge, \vee, \Rightarrow, \Leftrightarrow\}$ heißen Junktoren.

Formal ist eine aussagenlogische Formel also ein Baum, dessen Blätter mit aussagenlogischen Variablen oder den Konstanten $\top, \bot$ beschriftet sind und dessen innere Knoten mit Junktoren beschriftet sind, sodass ein mit $\neg$ beschrifteter Knoten ein Kind hat und die mit $\vee, \wedge, \Rightarrow, \Leftrightarrow$ beschrifteten Knoten zwei Kinder haben.

Wir werden "aussagenlogische Formeln und Variablen" auch kurz als "Formeln und Variablen" bezeichnen.

Man notiert Formeln mithilfe von Klammern als Zeichenketten, wobei man, um Klammern zu sparen, festlegt, dass die Bindungskraft der Junktoren $\neg, \wedge, \vee, \Rightarrow, \Leftrightarrow$ von links nach rechts abnimmt. Außerdem assoziiert $\Rightarrow$ nach rechts und alle anderen Junktoren nach links. $\phi \Rightarrow \phi \Rightarrow \theta$ steht also für $\phi \Rightarrow (\psi \Rightarrow \theta)$.

Die im folgenden Syntaxbaum dargestellte Formel entspricht also der Zeichenkette $(A \wedge B) \vee ((C \Rightarrow D) \wedge (\neg A))$, was unter Berücksichtigung der Bindungskraft der Junktoren auch so notiert werden kann: $A \wedge B \vee (C \Rightarrow D) \wedge \neg A$,

1.2 Bedeutung der Aussagenlogik

Der Wahrheitsgehalt einer Formel ergibt sich aus dem Wahrheitsgehalt aller Variablen, die in ihr vorkommen gemäß folgender Festlegung:

- Eine Formel der Form $\phi \wedge \psi$ ist wahr, wenn ϕ und ψ beide wahr sind. Ist entweder ϕ oder ψ falsch, so ist $\phi \wedge \psi$ falsch.
- Eine Formel der Form $\phi \vee \psi$ ist wahr, wenn ϕ wahr ist oder wenn ψ wahr ist. Nur wenn ϕ und ψ beide falsch sind, ist die Formel $\phi \vee \psi$ falsch.
- Eine Formel der Form $\neg\phi$ ist wahr, wenn ϕ falsch ist. Ist dagegen ϕ wahr, so ist $\neg\phi$ falsch.
- Eine Formel der Form $\phi \Rightarrow \psi$ ist wahr, wenn entweder ϕ falsch ist, oder aber ϕ wahr ist und ψ dann auch wahr ist. Nur wenn ϕ wahr ist und zugleich ψ falsch ist, ist $\phi \Rightarrow \psi$ falsch.
- Eine Formel der Form $\phi \Leftrightarrow \psi$ ist wahr, wenn ϕ und ψ denselben Wahrheitsgehalt haben, also entweder beide wahr, oder beide falsch sind. falsch ist $\phi \Leftrightarrow \psi$, wenn ϕ und ψ entgegengesetzten Wahrheitsgehalt haben.
- Die Formel $\top$ ist wahr, die Formel $\bot$ ist nicht wahr; beides unabhängig vom Wahrheitsgehalt der Variablen.

Ist also etwa A falsch und B wahr und außerdem C falsch, so sind $A \wedge B$ falsch und $C \Rightarrow D$ wahr; außerdem ist $\neg A$ dann wahr, sodass unsere obige Beispielformel dann wahr ist.

1.2.1 Einfache Anwendungen

Jane und der Zug "Wenn der Zug verspätet ist und keine Taxis am Bahnhof stehen, dann kommt Jane zu spät."

Wir führen drei Variablen ein Z, T, S mit der folgenden intuitiven Bedeutung.

- Z bedeute: der Zug kommt zu spät.
- T bedeute: es stehen Taxis am Bahnhof.
- S bedeute: Jane kommt zu spät.

Die obige Aussage kann dann als die folgende Formel geschrieben werden.

$$\phi := Z \wedge \neg T \Rightarrow S$$

Nehmen wir an, dass ϕ wahr ist und außerdem Z wahr ist (Zug ist verspätet) und S falsch ist (Jane kommt rechtzeitig). Dann können wir schließen, dass T wahr sein muss (es stehen Taxis am Bahnhof). Wäre nämlich T falsch, so wäre $Z \wedge \neg T$ wahr, also müsste dann auch S wahr sein.

Vor der Wahl Drei Politiker Angela, Franz, Joschka machen folgende Aussagen:
Angela: "Franz oder ich werden an der Regierung beteiligt sein."
Franz: "Entweder Joschka oder ich werden an der Regierung beteiligt sein."
Joschka: "Entweder Angela oder ich werden in der Opposition sein."
Nehmen wir an, alle drei Vorhersagen bewahrheiten sich. Wie war der Wahlausgang?
Wir führen drei Variablen A, F, J mit der folgenden intuitiven Bedeutung ein:

- A bedeute: Angela ist an der Regierung beteiligt.
- F bedeute: Franz ist an der Regierung beteiligt.
- J bedeute: Joschka ist an der Regierung beteiligt.

Die drei Aussagen entsprechen dann der folgenden Formel:

$$\phi := (F \vee A) \wedge (J \Leftrightarrow \neg F) \wedge (A \Leftrightarrow \neg J)$$

Nehmen wir einmal an, dass F wahr ist. Dann muss wegen der zweiten Aussage J falsch sein und wegen der dritten Aussage A wahr sein.

Ist dagegen F falsch, so muss wegen der ersten Aussage A wahr sein und wegen der zweiten Aussage auch J wahr sein im Widerspruch zur dritten Aussage. Also ist F wahr, J falsch, A wahr.

Sudoku Will man ein Sudoku lösen, so kann man für $i, j, k \in \{1, 2, 3, \ldots, 9\}$ jeweils eine Variable A_{ijk} einführen, also insgesamt 9^3 Variablen mit der intuitiven Bedeutung
A_{ijk} bedeute: in der i-ten Zeile und j-ten Spalte steht eine k.
Also bedeutet A_{114}, dass links oben eine 4 steht.
Die Sudokuregeln kann man dann als Formeln aufschreiben; so besagt z.B. die Formel

$$A_{111} \Rightarrow \neg A_{211} \wedge \neg A_{311} \wedge \neg A_{411} \wedge \neg A_{511} \wedge \neg A_{611} \wedge \neg A_{711} \wedge \neg A_{811} \wedge \neg A_{911}$$

dass wenn links oben eine 1 steht, dann keine weitere 1 in der ersten Spalte vorkommen darf. Solch eine Formel braucht man natürlich auch für die anderen Ziffern und außerdem für jedes Feld und Zeile und Spalte und Neunerblock. Das macht insgesamt $9 \cdot 81 \cdot 3$ Formeln. Außerdem muss man noch sagen, dass in jedem Feld eine Ziffer stehen muss, also für alle i, j jeweils die Formel

$$A_{ij1} \vee A_{ij2} \vee A_{ij3} \vee A_{ij4} \vee A_{ij5} \vee A_{ij6} \vee A_{ij7} \vee A_{ij8} \vee A_{ij9}$$

Das macht noch einmal 81 Formeln. Man könnte jetzt noch fordern, dass für alle i, j höchstens eine der neun Variablen $A_{ij1}, \ldots A_{ij9}$ wahr ist, aber das folgt bereits aus den genannten Formeln (wieso? und wie würde man es formalisieren?).

Eine Sudokobelegung aus der Zeitung kann man dann als Konjunktion der entsprechenden Variablen formalisieren, also z.B. $A_{123} \wedge A_{458} \wedge A_{891} \wedge \ldots$. Das Sudoku zu lösen, läuft dann darauf hinaus, den Wahrheitsgehalt der Variablen so festzulegen, dass alle diese Formeln wahr sind.

1.3 Boolesche Algebra

Definition 2
Zwei Formeln sind äquivalent, wenn sie unabhängig vom Wahrheitsgehalt der Variablen immer denselben Wahrheitsgehalt haben. Man schreibt $\phi \iff \psi$, falls ϕ und ψ äquivalent sind.

Satz 1
Seien ϕ, ψ, θ beliebige Formeln. Es gilt:

1. $\phi \wedge \psi \iff \psi \wedge \phi$ *(Kommutativität)*
2. $\phi \vee \psi \iff \psi \vee \phi$ *(Kommutativität)*
3. $\phi \wedge (\psi \wedge \theta) \iff (\phi \wedge \psi) \wedge \theta$ *(Assoziativität)*
4. $\phi \vee (\psi \vee \theta) \iff (\phi \vee \psi) \vee \theta$ *(Assoziativität)*
5. $\phi \wedge (\psi \vee \theta) \iff \phi \wedge \psi \vee \phi \wedge \theta$ *(Distributivität)*
6. $\phi \vee (\psi \wedge \theta) \iff (\phi \vee \psi) \wedge (\phi \vee \theta)$ *(Distributivität)*
7. $\neg(\phi \vee \psi) \iff \neg\phi \wedge \neg\psi$ *(de Morgans Gesetz)*
8. $\neg(\phi \wedge \psi) \iff \neg\phi \vee \neg\psi$ *(de Morgans Gesetz)*
9. $\phi \vee \phi \iff \phi$ *(Idempotenz)*
10. $\phi \wedge \phi \iff \phi$ *(Idempotenz)*
11. $\phi \wedge \top \iff \phi$ *(Neutralität)*
12. $\phi \vee \bot \iff \phi$ *(Neutralität)*
13. $\phi \vee \top \iff \top$ *(Absorption)*
14. $\phi \wedge \bot \iff \bot$ *(Absorption)*
15. $\psi \Rightarrow \psi \iff \neg\phi \vee \psi$
16. $\phi \Leftrightarrow \psi \iff (\phi \Rightarrow \psi) \wedge (\psi \Rightarrow \phi)$
17. $\neg\neg\phi \iff \phi$
18. $\phi \Rightarrow \psi \iff \neg\psi \Rightarrow \neg\phi$ *(Kontraposition)*
19. $\neg\phi \iff \phi \Rightarrow \bot$ *(Widerspruch)*

Diese Äquivalenzen ähneln den Rechengesetzen der Algebra ($x \cdot (y + z) = x \cdot y + x \cdot z$, etc.) und werden deshalb nach G. Boole als Boolesche Algebra bezeichnet.

Die oben gegebene Bedeutung aussagenlogischer Formeln definieren wir nun noch einmal formal. Dies wird benötigt für Beweise über die Aussagenlogik und außerdem als Vorbereitung auf die Semantik der Prädikatenlogik.

Definition 3
Wir bezeichnen die Wahrheitswerte mit $\mathtt{tt}$ (wahr) und $\mathtt{ff}$ (falsch). Mit $\mathbf{B} = \{\mathtt{tt}, \mathtt{ff}\}$ bezeichnen wir die Menge dieser Wahrheitswerte. Durch die folgenden Wertetabellen sind auf der Menge $\mathbf{B}$ die Operationen $\neg$ (einstellig) und $\vee, \wedge, \Rightarrow, \Leftrightarrow$ (zweistellig, infix) erklärt:

x	y	$\neg x$	$x \vee y$	$x \wedge y$	$x \Rightarrow y$	$x \Leftrightarrow y$
ff	ff	tt	ff	ff	tt	tt
ff	tt	tt	tt	ff	tt	ff
tt	ff	ff	tt	ff	ff	ff
tt	tt	ff	tt	tt	tt	tt

Satz 2
Die Gesetze aus Satz **??** *gelten für die Menge der Wahrheitswerte sinngemäß, wobei die Äquivalenz durch Gleichheit ersetzt wird. D.h. für alle $x, y, z \in \mathbf{B}$ gilt:*

1. $x \wedge y = y \wedge x$ *(Kommutativität)*
2. $x \vee y = y \vee x$ *(Kommutativität)*
3. $x \wedge (y \wedge z) = (x \wedge y) \wedge z$ *(Assoziativität)*
4. $x \vee (y \vee z) = (x \vee y) \vee z$ *(Assoziativität)*
5. ...

Bemerkung 1
Die Menge $\mathbf{B}$ mit den Operationen $\vee, \wedge, \neg$ und den Konstanten $\mathtt{tt}, \mathtt{ff}$ bildet eine *Boolesche Algebra*. Allgemeiner bezeichnet man eine Menge mit Operationen $\neg, \vee, \wedge$ und Konstanten $\bot, \top$, welche den o.a. Gesetzen ohne den Gesetze mit $\Rightarrow, \Leftrightarrow$ genügen, als Boolesche Algebra. Eine weitere Boolesche Algebra bildet z.B. die Potenzmenge einer festen Menge X, also die Menge aller Teilmengen von X. Die Operationen $\wedge, \vee, \neg$ sind darauf durch Schnitt, Vereinigung, Komplement erklärt.

1.4 Formale Semantik

Wir wollen jetzt die Semantik, d.h. Bedeutung von aussagenlogischen Formeln formal definieren.

Definition 4
Eine Belegung (einer Menge $\mathcal{A}$ von Aussagenvariablen) ist eine Funktion $\eta : \mathcal{A} \to \mathbf{B}$. Durch die folgenden Klauseln ordnen wir jeder aussagenlogischen Formel ϕ über $\mathcal{A}$ und

Belegung η einen Wahrheitswert $[\![\phi]\!]\eta \in \mathbf{B}$ zu.

$$\begin{array}{lcl}
[\![A]\!]\eta & = & \eta(A) \\
[\![\neg\phi]\!]\eta & = & \neg([\![\phi]\!]\eta) \\
[\![\phi \vee \psi]\!]\eta & = & [\![\phi]\!]\eta \vee [\![\phi]\!]\eta \\
[\![\phi \wedge \psi]\!]\eta & = & [\![\phi]\!]\eta \wedge [\![\phi]\!]\eta \\
[\![\phi \Rightarrow \psi]\!]\eta & = & [\![\phi]\!]\eta \Rightarrow [\![\phi]\!]\eta \\
[\![\phi \Leftrightarrow \psi]\!]\eta & = & [\![\phi]\!]\eta \Leftrightarrow [\![\phi]\!]\eta \\
[\![\top]\!]\eta & = & \mathtt{tt} \\
[\![\bot]\!]\eta & = & \mathtt{ff}
\end{array}$$

Die logischen Symbole $\neg, \vee, \wedge, \Rightarrow$ auf der *rechten* Seite dieser Definitionsgleichungen bezeichnen die in Definition **??** eingeführten Operationen auf Wahrheitswerten in $\mathbf{B}$. Die logischen Symbole auf der *linken* Seite hingegen sind Teil der Syntax von aussagenlogischen Formeln. Manche Autoren heben diesen Unterschied besonders hervor, indem sie andere Symbole für die logischen Operationen auf den Wahrheitswerten verwenden, wie etwa $\cdot$ statt $\wedge$ und $+$ statt $\vee$.

Bemerkung 2
Häufig findet man die Notation $\eta \models \phi$, gelesen „η erfüllt ϕ“ für $[\![\phi]\!]\eta = \mathtt{tt}$.

Der Wahrheitswert einer Formel ϕ hängt nur vom Wahrheitswert derjenigen Variablen ab, die tatsächlich in ϕ vorkommen. Stimmen also η und η' auf allen in ϕ vorkommenden Variablen überein, so gilt $[\![\phi]\!]\eta = [\![\phi]\!]\eta'$.

Definition 5
Zwei Formeln ϕ, ψ sind *äquivalent*, wenn für alle Belegungen η der Variablen in ϕ und ψ gilt $[\![\phi]\!]\eta = [\![\psi]\!]\eta$.

Diese Definition stimmt mit der informellen Definition im letzten Abschnitt überein; insbesondere gelten die Äquivalenzen aus Satz **??** auch im Sinne der formal definierten Äquivalenz. Nehmen wir als Beispiel die Assoziativität. Seien ϕ, ψ, θ. Um zu zeigen, dass $(\phi \wedge \psi) \wedge \theta$ und $\phi \wedge (\psi \wedge \theta)$ äquivalent sind, müssen wir zeigen, dass sie unter jeder Belegung denselben Wahrheitswert haben. Sei also eine feste aber beliebige Belegung η vorgegeben. Es gilt dann

$$[\![\phi \wedge (\psi \wedge \theta)]\!]\eta = [\![\phi]\!]\eta \wedge ([\![\psi]\!]\eta \wedge [\![\theta]\!]\eta) = ([\![\phi]\!]\eta \wedge [\![\psi]\!]\eta) \wedge [\![\theta]\!]\eta = [\![(\phi \wedge \psi) \wedge \theta]\!]\eta$$

Die erste und die dritte Gleichung ergeben sich aus der Definition der semantischen Funktion $[\![-]\!]$; die zweite Gleichung ist das Assoziativgesetz für die Operation $\wedge$ auf der Menge der Wahrheitswerte $\mathbf{B}$.

Definition 6
Seien ϕ, ψ Formeln, A eine Variable. Die Formel $\phi[\psi/A]$ (Substitution von ψ für A in ϕ) erhält man, indem man in ϕ jedes Vorkommen von A durch ψ ersetzt.

Zum Beispiel ist $(A \Rightarrow B \wedge A)[(A \vee B)/A] = (A \vee B) \Rightarrow B \wedge (A \vee B)$.

Definition 7
Eine Formel ψ ist Teilformel einer Formel ϕ, wenn es eine Formel θ und eine Variable A gibt, sodass $\phi = \theta[\psi/A]$.

Zum Beispiel ist $A \vee B$ Teilformel von $(A \vee B) \Rightarrow C$, denn es gilt

$$(A \vee B) \Rightarrow C = (X \Rightarrow C)[(A \vee B)/X]$$

Satz 3
Ersetzt man in einer Formel ϕ eine Teilformel durch eine zu ihr äquivalente Formel, so ist die resultierende Formel äquivalent zu ϕ. Formal: Falls $\psi_1 \iff \psi_2$, so folgt $\theta[\psi_1/A] \iff \theta[\psi_2/A]$.

Beweis Sei eine Belegung η vorgegeben. Es gilt $[\![\theta[\psi_1/A]]\!]\eta = [\![\theta]\!]\eta[A{\mapsto}[\![\psi_1]\!]\eta] = [\![\theta]\!]\eta[A{\mapsto}[\![\psi_2]\!]\eta] = [\![\theta[\psi_2/A]]\!]\eta$. Die zweite Gleichung benutzt die Äquivalenz von ψ_1 und ψ_2. Die erste und die dritte Gleichung ist intuitiv klar; formal kann man sie durch Induktion über den Aufbau von θ beweisen. ∎

Zur Illustration des Satzes stellen wir fest, dass $A \wedge \neg(B \vee C)$ und $A \wedge (\neg B \wedge \neg C)$ äquivalent sind. (De Morgans Gesetz auf die Teilformel $\neg(B \vee C)$ angewendet.)

1.5 Tautologien und Erfüllbarkeit

Definition 8
Eine Formel ϕ ist eine *Tautologie*, wenn sie unabhängig vom Wahrheitsgehalt der Variablen stets wahr ist. Formal also, wenn für alle Belegungen η ihrer Variablen gilt $[\![\phi]\!]\eta = \mathtt{tt}$.

Eine Formel ϕ ist *erfüllbar*, wenn es eine Belegung ihrer Variablen gibt, die sie erfüllt (wahrmacht). Formal heißt das, dass eine Belegung η existiert, derart, dass $[\![\phi]\!]\eta = \mathtt{tt}$.

Eine Formel ϕ ist *unerfüllbar*, wenn sie nicht erfüllbar ist.

Beispiel 1
Die folgenden Formeln sind Tautologien

- $A \vee \neg A$
- $(A \Rightarrow B) \Rightarrow (B \Rightarrow C) \Rightarrow (A \Rightarrow C)$
- $\neg\neg A \Rightarrow A$
- $A \wedge B \vee A \wedge \neg B \vee \neg A \wedge B \vee \neg A \wedge \neg B$

Die folgenden Formeln sind erfüllbar

- A
- $A \wedge B$
- $(A \vee B) \wedge (\neg A \vee B) \wedge (A \vee \neg B)$

Die folgenden Formeln sind unerfüllbar

- $A \wedge \neg A$
- $(A \vee B) \wedge (\neg A \vee B) \wedge (A \vee \neg B) \wedge (\neg A \vee \neg B)$

Satz 4
Seien ϕ, ψ Formeln.

1. *ϕ ist eine Tautologie, genau dann, wenn $\neg\phi$ unerfüllbar ist.*
2. *ϕ ist erfüllbar, genau dann, wenn $\neg\phi$ keine Tautologie ist.*
3. *$\phi \wedge \psi$ ist eine Tautologie, genau dann, wenn sowohl ϕ, als auch ψ, Tautologien sind.*
4. *$\phi \vee \psi$ ist erfüllbar, genau dann, wenn ϕ erfüllbar ist oder ψ erfüllbar ist (oder beide).*
5. *ϕ und ψ sind äquivalent, genau dann, wenn $\phi \Leftrightarrow \psi$ eine Tautologie ist.*
6. *ϕ ist Tautologie, genau dann, wenn ϕ zu $\top$ äquivalent ist und ϕ ist unerfüllbar, genau dann, wenn ϕ zu $\bot$ äquivalent ist.*

Bemerkung 3
Tautologien werden auch als allgemeingültige Formeln bezeichnet.

1.6 Normalformen

Wenn $\phi_1, \ldots, \phi_n$ Formeln sind, so schreiben wir $\bigwedge_{i=1}^{n} \phi_i$ für die Formel $\phi_1 \wedge \phi_2 \wedge \cdots \wedge \phi_n$. Ist $n = 1$, so ist einfach ϕ_1 gemeint. Analog definieren wir $\bigvee_{i=1}^{n} \phi_i$, wobei auch wieder im Falle $n = 1$ die Formel ϕ_1 gemeint ist. Ist I eine endliche Menge und ϕ_i eine Formel für $i \in I$, dann schreibt man $\bigwedge_{i \in I} \phi_i$ für die Formel $\phi_{i_1} \wedge \cdots \wedge \phi_{i_n}$, wobei $i_0, i_1, \ldots, i_n$ eine Aufzählung der Elemente von I ist. Man muss diese Aufzählung irgendwie festlegen, aber wegen des Assoziativitäts- und Kommutativgesetzes hängt die Bedeutung der Formel $\bigwedge_{i \in I} \phi_i$ nicht von der Wahl dieser Aufzählung ab. Der Spezialfall einer leeren Konjunktion versteht sich als $\top$. Man kann das so begründen, dass eine Konjunktion $\bigwedge_{i \in I} \phi_i$ wahr ist, wenn ϕ_i wahr ist für alle $i \in I$. Ist aber I leer, so ist diese Bedingung trivialerweise erfüllt.

Analog definieren wir die Notation $\bigvee_{i \in I} \phi_i$, wobei nunmehr die leere Disjunktion als $\bot$ (Falsum) verstanden wird, denn $\bigvee_{i \in I} \phi_i$ ist ja wahr, wenn ein $i \in I$ angegeben werden kann, sodass ϕ_i wahr ist. Aber im Falle $I = \emptyset$ ist es unmöglich, überhaupt ein i anzugeben.

Eine weiteres Argument für diese Setzung ist das *Permanenzprinzip* (Fortgelten von Rechenregeln), denn es gilt nun $\bigvee_{i \in I} \phi_i \Longleftrightarrow \phi_{i_0} \vee \bigvee_{i \in I \setminus \{i_0\}} \phi_i$ für alle $i_0 \in I$, selbst wenn $I \setminus \{i_0\} = \emptyset$. Analog natürlich für die Konjunktion.

Merke: Leere Konjunktion = Verum; leere Disjunktion = Falsum.

Zur Illustration betrachten wir noch einmal das Sudoku-Beispiel aus **??**. Sei N die Menge $\{1, 2, \ldots, 9\}$. Die folgende Formel drückt dann aus, dass in keiner Zeile (i) an zwei verschiedenen Spalten (j und j') dieselbe Zahl z steht.

$$\bigwedge_{i\in N}\bigwedge_{j\in N}\bigwedge_{j'\in N\setminus\{j\}}\bigwedge_{z\in N}\neg(A_{ijz}\wedge A_{ij'z})$$

Die dritte Konjunktion $\bigwedge_{j'\in N\setminus\{j\}}$ dürfen wir auch in etwas lesbarer Form $\bigwedge_{j'\in N, j\neq j'}$ schreiben.

Definition 9

- Formeln der Gestalt A oder $\neg A$, wobei A eine Variable ist, heißen *Literale*. Literale sind also die negierten und die nichtnegierten aussagenlogischen Variablen.
- Eine Formel der Gestalt $\ell_1 \vee \ell_2 \vee \cdots \vee \ell_n$, wobei $\ell_1, \ldots, \ell_n$ Literale sind, heißt *Klausel*. Eine Klausel ist also eine Disjunktion von Literalen.
- Eine Formel der Gestalt $\ell_1 \wedge \ell_2 \wedge \cdots \wedge \ell_n$, wobei $\ell_1, \ldots, \ell_n$ Literale sind, heißt *Minterm*. Ein Minterm ist also eine Konjunktion von Literalen.
- Eine Formel der Gestalt $K_1 \wedge K_2 \wedge \cdots \wedge K_m$, wobei $K_1, \ldots, K_m$ Klauseln sind, heißt *konjunktive Normalform* (KNF). Eine KNF ist also eine Konjunktion von Klauseln.
- Eine Formel der Gestalt $M_1 \vee M_2 \vee \cdots \vee M_m$, wobei $M_1, \ldots, M_m$ Minterme sind, heißt *disjunktive Normalform* (DNF). Eine DNF ist also eine Disjunktion von Mintermen.

Satz 5
Zu jeder Formel ϕ existiert eine äquivalente KNF und eine äquivalente DNF.

Beweis Unter Verwendung von Satz **??** eliminiert man aus ϕ zunächst die Junktoren $\Rightarrow$ und $\Leftrightarrow$, indem man eine Teilformel $\alpha \Leftrightarrow \beta$ zunächst durch $(\alpha \Rightarrow \beta) \wedge (\beta \Rightarrow \alpha)$ ersetzt und sodann jede Teilformel der Gestalt $\rho \Rightarrow \theta$ durch $\neg\rho \vee \theta$.

Die resultierende zu ϕ äquivalente Formel enthält nur die Junktoren $\neg, \vee, \wedge$. Durch sukzessive Anwendung der De Morganschen Gesetze lassen sich alle Negationen nach innen verschieben, sodass man eine Formel enthält, in der die Negation nur unmittelbar vor Variablen vorkommt.

Durch Anwendung der Distributivgesetze erhält man schließlich eine KNF oder DNF, je nachdem, welches der beiden Distributivgesetze verwendet wird. ∎

Beispiel 2
Betrachten wir die Formel $\phi := A \wedge B \vee (C \Rightarrow D) \wedge \neg A$. Eine äquivalente DNF erhalten wir wie folgt:

$$\begin{aligned} \phi &\iff \\ A\wedge B \ \vee\ (\neg C \vee D) \wedge \neg A &\iff \\ A\wedge B \vee \neg C\wedge\neg A \vee D\wedge\neg A & \end{aligned}$$

Für eine KNF rechnen wir:

$$\begin{aligned} \phi &\iff \\ A\wedge B \ \vee\ (\neg C \vee D) \wedge \neg A &\iff \\ (A \ \vee\ (\neg C \vee D)\wedge\neg A) \wedge (B \ \vee\ (\neg C \vee D)\wedge\neg A) &\iff \\ (A \vee \neg C \vee D) \wedge (A \vee \neg A) \wedge (B \vee \neg C \vee D) \wedge (B \vee \neg A) & \end{aligned}$$

Satz 6
Ein Minterm $\ell_1 \wedge \cdots \wedge \ell_n$ ist erfüllbar, wenn er keine zwei entgegengesetzte Literale enthält (A und $\neg A$ sind entgegengesetzt).

Eine DNF ist erfüllbar, genau dann, wenn einer ihrer Minterme erfüllbar ist.

Eine Klausel $\ell_1 \vee \cdots \vee \ell_n$ ist tautologisch, wenn sie zwei entgegengesetzte Literale enthält.

Eine KNF ist tautologisch, genau dann, wenn jede ihrer Klauseln tautologisch ist.

Um festzustellen, ob eine Formel tautologisch (erfüllbar) ist, kann man sie also auf KNF (DNF) bringen. Leider werden bei der Anwendung der Distributivgesetze Teilformeln verdoppelt, sodass die Formel beim Bringen auf Normalform exponentiell anwachsen kann (und das meist auch tut).

Bemerkung 4
Durch Einführen von Abkürzungen für Teilformeln kann man das exponentielle Wachstum vermeiden; man erhält so eine KNF ϕ' die zur ursprünglich gegebenen Formel ϕ *erfüllungsäquivalent* ist, d.h. ϕ ist erfüllbar, genau dann, wenn ϕ' erfüllbar ist. Nehmen wir etwa an, dass

$$\phi = (A \wedge B \wedge C \wedge D) \vee (E \wedge F \wedge G \wedge H)$$

Eine zu ϕ äquivalente KNF besteht aus den sechzehn Klauseln $A \vee E, A \vee F, \ldots D \vee G, D \vee H$.

Eine erfüllungsäquivalente KNF ist hingegen durch

$$\phi' = (A\vee\neg X)\wedge(B\vee\neg X)\wedge(C\vee\neg X)\wedge(D\vee\neg X)\wedge(E\vee\neg Y)\wedge(F\vee\neg Y)\wedge(G\vee\neg Y)\wedge(H\vee\neg Y)\wedge(X\vee Y)$$

gegeben. Durch die Klausel $X \vee Y$ ist festgelegt, dass entweder X oder Y oder beide auf $\mathtt{tt}$ gesetzt werden müssen. Setzt man aber X auf $\mathtt{tt}$, so erzwingt man die Erfüllung von $A \wedge B \wedge C \wedge D$ und analog für Y.

1.6.1 Wahrheitstafeln

Definition 10
Sei $\mathcal{A}$ eine endliche Menge von Variablen. Eine Funktion f, die jeder Belegung η von $\mathcal{A}$ einen Wahrheitswert $f(\eta)$ zuordnet, heißt *Wahrheitstafel* über $\mathcal{A}$.

Man stellt Wahrheitstafeln üblicherweise als Tabellen dar, etwa so:

A	B	C	f
ff	ff	ff	tt
ff	ff	tt	ff
ff	tt	ff	tt
ff	tt	tt	ff
tt	ff	ff	tt
tt	ff	tt	ff
tt	tt	ff	ff
tt	tt	tt	tt

Diese Wahrheitstafel gehört zur Formel $\phi = X \wedge Y \Leftrightarrow Z$. Wir zeigen jetzt, dass alle Wahrheitstafeln durch Formeln dargestellt werden können.

Satz 7 (Konstruktion einer Formel zu gegebener Wahrheitstafel)
Sei f eine Wahrheitstafel über $\mathcal{A}$. Es existiert eine Formel ϕ, sodass $[\![\phi]\!]\eta = f(\eta)$.

Beweis Jeder Belegung η ordnen wir den entsprechenden Minterm

$$\overline{\eta} = \bigwedge_{A \in \mathcal{A}} \ell_A$$

zu, wobei $\ell_A = A$, falls $\eta(A) = \mathtt{tt}$ und $\ell_A = \neg A$ falls $\eta(A) = \mathtt{ff}$. Es gilt $[\![\overline{\eta}]\!]\eta = \mathtt{tt}$ und wenn $[\![\overline{\eta}]\!]\mu = \mathtt{tt}$, so folgt $\eta = \mu$.

Die gewünschte Formel ergibt sich zu

$$\phi = \bigvee_{\{\eta | f(\eta) = \mathtt{tt}\}} \overline{\eta}$$

Im Spezialfall, wo kein η existiert mit $f(\eta) = \mathtt{tt}$, ergibt sich per Definition $\phi = \bot$. Man erhält die Formel ϕ also als DNF, deren Minterme gerade den Zeilen der Wahrheitstafel entsprechen, bei denen tt "herauskommt". ∎

Bemerkung 5
Alternativ kann man eine Formel ϕ zu gegebener Wahrheitstafel auch als KNF konstruieren, wobei man dann für jede Zeile der Wahrheitstafel, bei der ff "herauskommt" eine entsprechende Klausel einführt. Gibt es überhaupt keine solche Zeile, dann setzt man $\phi = \top$.

Im Beispiel erhält man also die DNF

$$\neg A \wedge \neg B \wedge \neg C \vee \neg A \wedge B \wedge \neg C \vee A \wedge \neg B \wedge \neg C \vee A \wedge B \wedge C$$

und alternativ die KNF

$$(A \vee B \vee \neg C) \wedge (A \vee \neg B \vee \neg C) \wedge (\neg A \vee B \vee \neg C) \wedge (\neg A \vee \neg B \vee C)$$

Hier verbietet also z.B. die Klausel $A \vee B \vee \neg C$ gerade die zweite Zeile der Wahrheitstafel usw.

Die Konstruktion einer DNF bietet sich an, wenn die Zeilen, bei denen $\mathtt{tt}$ herauskommt, in der Minderheit sind und umgekehrt. In unserem Beispiel ist es egal.

Um festzustellen, ob eine Formel tautologisch / erfüllbar / unerfüllbar ist, kann man die zu ihr gehörige Wahrheitstafel tabellieren und nachsehen, ob bei allen /manchen /keiner Zeile(n) $\mathtt{tt}$ herauskommt. Bei wenigen Variablen ist das ein praktikables Verfahren; nachdem sich aber mit jeweils einer Variablen mehr die Größe der Wahrheitstafel verdoppelt, verbietet sich dieses Verfahren ab ca. 30 Variablen (Größe der Tafel dann ca. 4GB).

Hat man mit der Methode aus Satz **??** eine DNF gefunden, so kann man versuchen, diese mit Hilfe der Rechenregeln aus Satz **??** zu vereinfachen. Die Minimierungsverfahren von Quine-McCluskey und Karnaugh-Veitch systematisieren dies.

1.7 Kompaktheitssatz

Definition 11
Eine Menge von Formeln $\mathcal{F}$ heißt *erfüllbar*, wenn eine Belegung η existiert, sodass $[\![\phi]\!]\eta = \mathtt{tt}$ für alle Formeln $\phi \in \mathcal{F}$.

Eine (möglicherweise unendliche) Menge von Formeln $\mathcal{F}$ heißt *konsistent*, wenn jede endliche Teilmenge T von $\mathcal{F}$ erfüllbar ist.

Stellen wir uns vor, wir hätten eine Menge von quadratischen Fliesen, deren Seiten bestimmte Farben tragen ⊠. Von jeder Fliesensorte seien beliebig viele vorhanden, es gibt aber nur endlich viele Fliesensorten. Man darf zwei Fliesen nur aneinanderlegen, wenn die angrenzenden Farben übereinstimmen. Wir stellen uns die Frage, ob man mit den gegebenen Fliesensorten die gesamte Ebene auslegen kann, oder ob man irgendwann "steckenbleibt".[1]

Zu diesem Zweck führen wir für jede Fliesensorte t und Zahlen $i, j \in \mathbf{Z}$ eine aussagenlogische Variable $F_{t,i,j}$ ein, die bedeuten soll, dass an der Position i, j eine Fliese des Typs t liegt. Der Einfachheit halber nehmen wir an, dass man Fliesen nicht drehen darf. Das ist keine echte Einschränkung, denn man kann ja jede der vier Orientierungen als eigene Sorte betrachten. Die Bedingungen kann man jetzt wieder als Menge von Formeln ausdrücken und zwar nimmt man

- für jedes $i, j \in \mathbf{Z}$ und Sorten t, t' derart, dass die Farben der rechten Kante von t und der linken Kante von t' verschieden sind, die Formel $\neg(F_{t,i,j} \wedge F_{t,i,j+1})$;
- für jedes $i, j \in \mathbf{Z}$ und Sorten t, t' derart, dass die Farben der oberen Kante von t und der unteren Kante von t' verschieden sind, die Formel $\neg(F_{t,i,j} \wedge F_{t,i+1,j})$;

[1] Ein berühmtes Resultat besagt, dass kein Programm existieren kann, welches eine Beschreibung von solchen Fliesensorten einliest und dann ausrechnet, ob eine Auslegung der Ebene mit Fliesen dieser Sorten möglich ist, oder nicht. Man weiß auch, dass es Fliesensorten gibt, mit denen die Ebene zwar ausgelegt werden kann, aber nur auf unregelmäßige Weise (nichtperiodisch).

- für jedes $i, j \in \mathbf{Z}$ eine Formel $F_{t_1,i,j} \vee F_{t_2,i,j} \vee \cdots \vee F_{t_n,i,j}$, wobei $t_1, \ldots, t_n$ eine Aufzählung der Fliesensorten ist. Diese Formel besagt dann, dass an der Stelle i, j tatsächlich eine Fliese liegt.

Sei $\mathcal{F}$ die unendliche Menge all dieser Formeln. Die Menge $\mathcal{F}$ ist erfüllbar, genau dann, wenn mit den gegebenen Fliesensorten die Ebene ausgelegt werden kann und eine erfüllende Belegung gibt dann solch eine Auslegung an.

Konsistenz der Formelmenge $\mathcal{F}$ bedeutet hingegen, dass mit den gegebenen Fliesensorten endliche Bereiche beliebiger Form und Größe ausgelegt werden können.

Wir werden jetzt zeigen, dass Konsistenz und Erfüllbarkeit dasselbe ist, und zwar nicht nur bei dieser Formelmenge $\mathcal{F}$, sondern immer. Man beachte, dass das nicht unmittelbar offensichtlich ist, denn es könnte ja sein, dass jeder endliche Bereich auslegbar ist, aber jedesmal anders.

Hat man unendlich viele Farben und Fliesensorten, dann gilt die Äquivalenz in der Tat nicht, aber dann kann man die Spielregeln auch nicht als Formelmenge ausdrücken.

Lemma 1
Sei $\mathcal{F}$ eine konsistente Formelmenge und A eine Variable. Mindestens eine der beiden Formelmengen $\mathcal{F} \cup \{A\}$ und $\mathcal{F} \cup \{\neg A\}$ ist konsistent.

Beweis Nehmen wir widerspruchshalber an, dass beide inkonsistent sind. Dann gibt es also jeweils unerfüllbare endliche Teilmengen. Da eine Obermenge einer unerfüllbaren Menge erst recht unerfüllbar ist, bedeutet das also, dass eine endliche Teilmenge $T \subseteq \mathcal{F}$ existiert, sodass sowohl $T \cup \{A\}$, als auch $T \cup \{\neg A\}$ unerfüllbar sind. Dann ist aber auch T selbst unerfüllbar, denn eine erfüllende Belegung würde ja auch eine der beiden Mengen erfüllen. Das aber steht im Widerspruch zur Konsistenz von $\mathcal{F}$. ∎

Satz 8 (Kompaktheitssatz)
Sei $\mathcal{F}$ eine konsistente Menge von Formeln. Dann ist $\mathcal{F}$ erfüllbar.

Beweis Wir zeigen nur den Spezialfall in dem es höchstens abzählbar unendlich viele Variablen gibt. Wir fixieren also eine Nummerierung der aussagenlogischen Variablen $\mathcal{A} = \{A_1, A_2, A_3, \ldots\}$. Im Beispiel könnte man etwa von $(0, 0)$ ausgehend sich spiralförmig nach außen vorarbeiten und an jedem Punkt (i, j) jeweils die Variablen F_{tij} aufzählen.

Jetzt definieren wir eine Folge von konsistenten Formelmengen $\mathcal{F}_0, \mathcal{F}_1, \mathcal{F}_2, \ldots$ wie folgt:

- $\mathcal{F}_0 = \mathcal{F}$.
- Ist $\mathcal{F}_i$ schon definiert, dann betrachte die beiden Formelmengen $\mathcal{F}_i \cup \{A_i\}$ und $\mathcal{F}_i \cup \{\neg A_i\}$. Mindestens eine von beiden muss konsistent sein, denn sonst wäre wegen obigem Lemma bereits $\mathcal{F}_i$ inkonsistent gewesen. Setze also $\mathcal{F}_{i+1} = \mathcal{F}_i \cup \{\ell\}$, wobei $\ell \in \{A_i, \neg A_i\}$ so gewählt ist, dass $\mathcal{F}_i \cup \{\ell\}$ konsistent ist.

Wir definieren jetzt eine Belegung η durch $\eta(A_i) = \mathtt{tt}$, falls $A_i \in \mathcal{F}_i$ und $\eta(A_i) = \mathtt{ff}$, falls $\neg A_i \in \mathcal{F}_i$. Wir behaupten, dass diese Belegung alle Formeln in $\mathcal{F}$ erfüllt. Ist nämlich $\phi \in \mathcal{F}$ vorgegeben, dann wählen wir i so groß, dass die in ϕ vorkommenden Variablen alle

in $\{A_1, \ldots, A_i\}$ enthalten sind. Jede Belegung, die $\mathcal{F}_i$ erfüllt, muss aber diese Variablen genauso bewerten, wie η, weil das ja explizit in $\mathcal{F}_i$ durch Hinzunahme der entsprechenden Literale gefordert ist. Wegen der Konsistenz von $\mathcal{F}_i$ gibt es also so eine Belegung η', die wegen $\phi \in \mathcal{F} \subseteq \mathcal{F}_i$ auch ϕ wahrmacht. Dementsprechend muss also auch $[\![\phi]\!]\eta = [\![\phi]\!]\eta' = \mathtt{tt}$ gelten. ∎

1.8 Sequenzenkalkül

Um festzustellen, ob eine Formel eine Tautologie ist, kann man auch logisch argumentieren, zum Beispiel so.

Beweis von $A \wedge (A \Rightarrow B) \Rightarrow B \wedge A$**:** Wir nehmen an, es gelte $A \wedge (A \Rightarrow B)$, also A (H1) und $A \Rightarrow B$ (H2). Wir müssen zeigen $B \wedge A$, also zunächst einmal B. Das aber folgt aus H2 angewendet auf H1. Es bleibt noch A zu zeigen, aber das ist nichts anderes als unsere Annahme H1.

Es gibt mehrere logische Kalküle, mit deren Hilfe man solche Argumentationen formalisieren kann. Einer davon ist der Gentzensche Sequenzenkalkül, der für uns besonders geeignet ist, da er im Theorembeweiser PVS implementiert ist.

Definition 12 (Sequenz)
Eine *Sequenz* ist ein Paar von Listen von Formeln. Man schreibt eine Sequenz in der Form $\Gamma \Longrightarrow \Delta$, wobei Γ, Δ durch Komma getrennte Listen von Formeln sind. Die leere Liste lässt man weg. Man schreibt $\phi \in \Gamma$ um zu sagen, dass ϕ in Γ vorkommt.

Die Formeln in Γ heißen *Antezedentien* (Sg. Antezedens) der Sequenz; die Formeln in Δ heißen *Sukzedentien* (Sg. Sukzedens) der Sequenz.

Beispiele von Sequenzen sind

$$
\begin{array}{l}
A, A \Rightarrow B \Longrightarrow C \\
A \Longrightarrow B \\
A, \neg A \Longrightarrow \\
\Longrightarrow A, \neg A, C \Rightarrow D \\
\Longrightarrow
\end{array}
$$

Definition 13 (Bedeutung einer Sequenz)
Ist Γ eine Liste von Formeln, so schreiben wir $\bigwedge \Gamma$ für $\bigwedge_{\phi \in \Gamma} \phi$ und $\bigvee \Gamma$ für die Formel $\bigvee_{\phi \in \Gamma} \phi$. Einer Sequenz $\Gamma \Longrightarrow \Delta$ *ordnen wir die Formel* $\bigwedge \Gamma \Rightarrow \bigvee \Delta$ *zu.*

Die *Bedeutung einer Sequenz* ist die Bedeutung der ihr zugeordneten Formeln, also ist eine Sequenz $\Gamma \Longrightarrow \Delta$ unter einer gegebenen Belegung wahr, wenn entweder eine der Formeln in Γ falsch ist, oder aber eine der Formeln in Δ wahr ist. Man kann auch sagen, dass aus den Formeln in Γ wenigstens eine der Formeln in Δ folgen muss.

Eine Sequenz ist *Tautologie*, wenn die ihr zugeordnete Formel eine Tautologie ist.

Also ist der Sequenz $A, A \Rightarrow B \Longrightarrow B, C$ die Formel $A \wedge (A \Rightarrow B) \Rightarrow B \vee C$ zugeordnet. Der Sequenz $A, \neg A \Longrightarrow$ ist die Formel $A \wedge \neg A \Rightarrow \bot$ zugeordnet, also eine Tautologie.

$$\frac{\Gamma_1, \phi, \psi, \Gamma_2 \Longrightarrow \Delta}{\Gamma_1, \psi, \phi, \Gamma_2 \Longrightarrow \Delta}(\textsc{Perm}\text{-L}) \qquad \frac{\Gamma \Longrightarrow \Delta_1, \phi, \psi, \Delta_2}{\Gamma \Longrightarrow \Delta_1, \psi, \phi, \Delta_2}(\textsc{Perm}\text{-R})$$

$$\frac{\Gamma, \phi, \phi \Longrightarrow \Delta}{\Gamma, \phi \Longrightarrow \Delta}(\textsc{Contr}\text{-L}) \qquad \frac{\Gamma \Longrightarrow \Delta, \phi, \phi}{\Gamma \Longrightarrow \Delta, \phi}(\textsc{Contr}\text{-R})$$

Abbildung 1.1: Strukturelle Regeln

Der Sequenz $\Longrightarrow A, \neg A, C \Rightarrow D$ ist die Formel $\top \Rightarrow A \vee \neg A \vee (C \Rightarrow D)$ zugeordnet, ebenfalls eine Tautologie. Der Sequenz $\Longrightarrow$ ist die Formel $\top \Rightarrow \bot$ zugeordnet, welche äquivalent zu $\bot$ und dementsprechend unerfüllbar ist.

Definition 14 (Beweisregeln)
Die Beweisregeln des Sequenzenkalküls sind in Abb. **??** und **??** angegeben. Hinzu kommen noch zwei Regeln für die Biimplikation $\Leftrightarrow$, deren genaue Angabe als Übung verbleibt.

Die Sequenzen oberhalb des waagrechten Strichs heißen *Prämissen* der Beweisregel; die Sequenz unterhalb des waagrechten Strichs heißt *Konklusion* der Beweisregel. Ihre Bedeutung ist, dass aus der Gültigkeit der Prämissen auf die Gültigkeit der Konklusion geschlossen werden darf. Hierbei versteht sich eine Beweisregel gleich als ganze Familie von konkreten *Instanzen*: eine für jede konkrete Wahl der *Metavariablen* $\Gamma, \Delta, \phi, \psi$ etc. welche Formellisten $(\Gamma, \Delta, \dots)$, bzw. Formeln $(\phi, \psi, \dots)$ bezeichnen. Für mehrfache Vorkommen so einer Metavariablen in einer Regel ist jeweils die gleiche Formelliste, bzw. Formel einzusetzen. Dieselbe Metavariable in unterschiedlichen Regeln kann jeweils unterschiedlich gesetzt werden.

So bedeutet die Regel $\wedge$-R, dass aus $\Gamma \Longrightarrow \Delta, \phi$ und $\Gamma \Longrightarrow \Delta, \psi$ auf $\Gamma \Longrightarrow \Delta, \phi \wedge \psi$ geschlossen werden darf. Als konkrete Instanz der Regel dürfen wir von $A, B \Longrightarrow B$ und $A, B \Longrightarrow C$ auf $A, B \Longrightarrow B \wedge C$ schließen ($\Gamma = A, B$ und $\Delta =$ leer und $\phi = B$ und $\psi = C$).

Man beachte, dass manche Regeln zwei, andere Regeln eine, wieder andere gar keine Prämissen haben.

Definition 15 (Herleitung)
Eine Herleitung im Sequenzenkalkül ist ein endlicher Baum, dessen Knoten und Blätter mit Sequenzen beschriftet sind, so dass für jeden (inneren) Knoten gilt: Es gibt eine Instanz einer Regel, deren *Prämissen* genau die Beschriftungen der Kinder des Knotens sind und deren *Konklusion* gerade die Beschriftung des Knotens selbst ist. Diejenigen Blätter des Baumes, die nicht Konklusion einer Regel ohne Prämissen sind ($\top$-R, $\bot$-L) heißen *Prämissen* der Herleitung. Diejenigen Blätter des Baumes, die Konklusion einer Regel ohne Prämissen (also $\top$-R, $\bot$-L) sind, zählen nicht zu den Prämissen der Herleitung. Die Beschriftung des Wurzelknotens heißt *Konklusion* der Herleitung.

Definition 16 (Axiom)
Eine Sequenz $\Gamma \Longrightarrow \Delta$, wobei Γ und Δ eine Formel gemeinsam haben, heißt *Axiom*. Sie ist offensichtlich tautologisch.

$$\frac{\Gamma, \phi, \psi \Longrightarrow \Delta}{\Gamma, \phi \wedge \psi \Longrightarrow \Delta}(\wedge\text{-L}) \qquad \frac{\Gamma \Longrightarrow \Delta, \phi \qquad \Gamma \Longrightarrow \Delta, \psi}{\Gamma \Longrightarrow \Delta, \phi \wedge \psi}(\wedge\text{-R})$$

$$\frac{\Gamma, \phi \Longrightarrow \Delta \qquad \Gamma, \psi \Longrightarrow \Delta}{\Gamma, \phi \vee \psi \Longrightarrow \Delta}(\vee\text{-L}) \qquad \frac{\Gamma \Longrightarrow \Delta, \phi, \psi}{\Gamma \Longrightarrow \Delta, \phi \vee \psi}(\vee\text{-R})$$

$$\frac{\Gamma \Longrightarrow \Delta, \phi}{\Gamma, \neg\phi \Longrightarrow \Delta}(\neg\text{-L}) \qquad \frac{\Gamma, \phi \Longrightarrow \Delta}{\Gamma \Longrightarrow \Delta, \neg\phi}(\neg\text{-R})$$

$$\frac{\Gamma \Longrightarrow \Delta, \phi \qquad \Gamma, \psi \Longrightarrow \Delta}{\Gamma, \phi \Rightarrow \psi \Longrightarrow \Delta}(\Rightarrow\text{-L}) \qquad \frac{\Gamma, \phi \Longrightarrow \Delta, \psi}{\Gamma \Longrightarrow \Delta, \phi \Rightarrow \psi}(\Rightarrow\text{-R})$$

$$\frac{\Gamma \Longrightarrow \Delta}{\Gamma, \top \Longrightarrow \Delta}(\top\text{-L}) \qquad \frac{}{\Gamma \Longrightarrow \Delta, \top}(\top\text{-R})$$

$$\frac{}{\Gamma, \bot \Longrightarrow \Delta}(\bot\text{-L}) \qquad \frac{\Gamma \Longrightarrow \Delta}{\Gamma \Longrightarrow \Delta, \bot}(\bot\text{-R})$$

Abbildung 1.2: Logische Regeln

$$\dfrac{\dfrac{\dfrac{A \Longrightarrow B, A \quad B, A \Longrightarrow B}{A \Rightarrow B, A \Longrightarrow B}\Rightarrow\text{-L}}{A \Rightarrow B \Longrightarrow B, \neg A}\neg\text{-L}}{A \Rightarrow B, \neg B \Longrightarrow \neg A}\neg\text{-L}$$

Abbildung 1.3: Ein Beweis der Sequenz $A \Rightarrow B, \neg B \Longrightarrow \neg A$

Definition 17 (Beweis)
Eine Herleitung H, derart, dass alle Prämissen von H Axiome sind, heißt *Beweis*.

In der Beispielherleitung aus Abb. **??** haben wir zwei Prämissen, nämlich $A \Longrightarrow B, A$ und $B, A \Longrightarrow B$. Beide sind Axiome, also handelt es sich um einen Beweis und zwar um einen Beweis der Sequenz $A \Rightarrow B, \neg B \Longrightarrow \neg A$.

In der Beispielherleitung Abb. **??** haben wir vier Prämissen. Genau eine von ihnen, nämlich $A, C \Longrightarrow A \wedge B, D$ ist kein Axiom; daher handelt es sich bei dieser Herleitung um keinen Beweis.

1.8.1 Erläuterung der Beweisregeln

Die strukturellen Regeln aus Abb. **??** dienen der Verwaltung der Formellisten. In der Literatur werden Sequenzen häufig auch als Paare von Mengen von Formeln definiert. Dann sind alle strukturellen Regeln redundant.

Die logischen Regeln in Abb. **??** treten paarweise auf. Für jeden Junktor gibt es eine Linksregel, die angibt, was man mit einer Annahme dieser Form tun kann und eine

$$\dfrac{\dfrac{A, B \Longrightarrow A, A \wedge C \quad A, B \Longrightarrow B, A \wedge C}{A, B \Longrightarrow A \wedge B, A \wedge C} \quad \dfrac{A, C \Longrightarrow A \wedge B, A \quad A, C \Longrightarrow A \wedge B, D}{A, C \Longrightarrow A \wedge B, A \wedge D}}{A, (B \vee C) \Longrightarrow A \wedge B, A \wedge D}$$

Abbildung 1.4: Eine Herleitung, die kein Beweis ist.

Rechtsregel, die angibt, wie man eine Formel dieser Form beweisen kann.

Nehmen wir $\vee$-L. Um Δ unter der Annahme $\phi \vee \psi$ und Γ zu beweisen, genügt es Δ einmal unter der Annahme ϕ und dann noch einmal unter der Annahme ψ (jeweils natürlich unter Verwendung von Γ) zu beweisen. Die Regel formalisiert also die übliche Vorgehensweise der Fallunterscheidung.

Die Regel $\Rightarrow$-R gibt an, wie man eine Implikation $\phi \Rightarrow \psi$ beweist: man gibt ϕ zu den Annahmen hinzu und versucht dann ψ zu zeigen.

Die Regel $\neg$-R besagt, dass um $\neg\phi$ zu zeigen aus der Annahme ϕ ein Widerspruch (oder eine der Nebenformeln Δ) hergeleitet werden muss. Um eine negative Annahme $\neg\phi$ einzusetzen ($\neg$-L) versucht man ϕ zu beweisen, woraus ja dann Beliebiges folgt.

Dies sind natürlich nur Intuitionen; genaue Auskunft gibt der folgende Korrektheitssatz.

Satz 9
Sei H eine Herleitung mit Konklusion S und Prämissen $S_1, \ldots, S_\ell$. Erfüllt eine Belegung η jede der Prämissen $S_1, \ldots, S_\ell$, so erfüllt η auch die Konklusion S der Herleitung H.

Beweis Wir zeigen die Behauptung durch Induktion über die Anzahl der in H vorkommenden Schlußregeln. Formal zeigen wir also für jedes n die Behauptung "Sei H wie in den Voraussetzungen des Satzes und beinhalte H höchstens n Anwendungen von Regeln des Sequenzenkalküls, so erfüllt H die Aussage des Satzes.".

Ist $n = 0$ (Induktionsverankerung), dann wurde überhaupt keine Schlussregel verwendet, es gibt also nur eine Prämisse und die ist identisch mit der Konklusion der Herleitung. Die Behauptung ist dann klar.

Gelte nun die Behauptung für alle $n' \leq n$ (Induktionsvoraussetzung) und sei H eine Herleitung mit $n+1$ Regelanwendungen. Wir unterscheiden, welches die letzte Regelanwendung ist. Handelt es sich z.B. um die Regel $\vee$-L, so hat die Konklusion die Form $\Gamma, \phi \vee \psi \Longrightarrow \Delta$ und wir haben Herleitungen H_1 mit Konklusion $\Gamma, \phi \Longrightarrow \Delta$ (bez. als K_1) und H_2 mit Konklusion $\Gamma, \psi \Longrightarrow \Delta$ (bez. als K_2). Beide enthalten weniger als n Regelanwendungen und somit lässt sich die Induktionsvoraussetzung auf sie anwenden. Deshalb erfüllt eine Belegung η, die die Prämissen von H und somit von H_1 und H_2 erfüllt, auch die Konklusionen K_1 und K_2. Wir behaupten, dass dann η auch $\Gamma, \phi \vee \psi \Longrightarrow \Delta$ erfüllt. Wir nehmen also an, dass $\eta \models \bigwedge \Gamma$ und außerdem $\eta \models \phi \vee \psi$. In allen anderen Fällen ist die Konklusion trivialerweise erfüllt. Nach Definition muss nun gelten $\eta \models \phi$ oder $\eta \models \psi$. Im ersten Fall sind alle Antezedentien von H_1 erfüllt und es folgt $\eta \models \bigvee \Delta$ nach Induktionsvoraussetzung. Im zweiten Fall argumentieren wir analog mit H_2.

Zur weiteren Illustration betrachten wir die Regel ¬-L. Hier lautet die Konklusion $\Gamma, \neg\phi \Longrightarrow \Delta$ und wir haben eine kleinere Herleitung H_1 mit Konklusion $\Gamma \Longrightarrow \Delta, \phi$ auf welche die Induktionsvoraussetzung anwendbar ist. Wir können also annehmen, dass η die Konklusion von H_1 erfüllt und haben daraus zu folgern, dass η auch die Konklusion von H erfüllt. Nehmen wir also an, dass $\eta \models \bigwedge \Gamma$ und $\eta \models \neg\phi$. Die Induktionsvoraussetzung liefert $\eta \models \bigvee \Delta \vee \phi$. Nachdem nun $\eta \models \phi$ *nicht* infrage kommt, muss tatsächlich $\eta \models \bigvee \Delta$ gelten, wie verlangt.

Das Muster sollte nun klar sein. Für jede Regel ist zu zeigen: erfüllt eine Belegung η jede ihrer Prämissen, so erfüllt sie auch ihre Konklusion. Die anderen Fälle verbleiben somit als Übung. ∎

Korollar 10
Ist ϕ eine Formel, sodass die Sequenz $\Longrightarrow \phi$ Konklusion eines Beweises *im Sequenzenkalkül, so ist ϕ eine Tautologie.*

Beweis Sei η eine beliebige Belegung. Axiome werden von jeder beliebigen Belegung, also auch von η erfüllt. Nachdem aber alle Prämissen eines *Beweises* Axiome sind, folgt mit dem vorhergehenden Satz, dass η die Konklusion $\Longrightarrow \phi$ erfüllt. Es folgt $\eta \models \phi$ wie gewünscht. ∎

1.8.2 Einführung in PVS

Der Theorembeweiser PVS basiert auf dem Sequenzenkalkül und wurde entwickelt, um aus noch nicht ausführbaren Spezifikationen Folgerungen abzuleiten, um diese vor ihrer Implementierung zu "testen" (PVS = prototype verification system). Das System ist Anfang der 90er Jahre am Stanford Research Institute (SRI) entstanden und wird seitdem hauptsächlich von N Shankar entwickelt.

Wir rufen PVS von der Linux-Kommandozeile mit dem Befehl `pvs` auf. Dies öffnet ein Emacs-Fenster PVS. Auf evtl. Fragen mit `yes` antworten. Man erzeuge nun eine Datei `sequent.pvs` wie in Abb. **??**. Man klickt auf die erste "Proposition" (Satz). Das öffnet ein Fenster mit der Eingabeaufforderung `Rule?`. PVS versucht nun einen Beweis für die entsprechende Sequenz zu finden. Der Trennungspfeil $\Longrightarrow$ ist in PVS als `|-------` wiedergegeben in Anlehnung an Freges Ableitungssymbol $\vdash$. Die Regeln mit einer Prämisse, also ∧-L, ∨-R, ⇒-R werden allesamt mit dem Befehl `(flatten)` aufgerufen; die Regeln mit zwei Prämissen, also ∧-R, ∨-L, ⇒-L, werden mit dem Befehl `(split)` aufgerufen. Die Regeln für die Negation und die Konstanten $\top, \bot$ werden automatisch aufgerufen, außerdem werden Axiome automatisch als solche erkannt.

Im ersten Beispiel `K` genügt also der Befehl `(flatten)`, da er auf die Sequenz $A \Longrightarrow B \Rightarrow A$ und dann $A, B \Longrightarrow A$ führt, welche ein Axiom ist. Man beachte, dass `(flatten)` solange Regeln mit einer Prämisse anwendet, wie es geht, nicht nur einmal.

Das zweite Beispiel `dist1` beginnt mit `(split)` gefolgt von `split`, was auf die beiden Unterziele $A, B \Longrightarrow A \wedge C, B \wedge C$ und $A, C \Longrightarrow A \wedge C, B \wedge C$ führt. Im Gegensatz zu `(flatten)`, wendet `(split)` jeweils nur eine Regel an. Von hier kommt man mit drei weiteren `(split)` Befehlen zum Ziel.

```
sequent_calculus: THEORY
BEGIN

A,B,C,D:  boolean
A11,A12,A21,A22,A31,A32: boolean

K : PROPOSITION
    A IMPLIES B IMPLIES A

dist1 : PROPOSITION
    A AND (B OR C) IMPLIES (A AND B) OR (A AND C)

dist2 : PROPOSITION
    A AND B OR C AND D IMPLIES (A OR C) AND  (B OR D)


S : PROPOSITION
    (A IMPLIES B IMPLIES C) IMPLIES (A IMPLIES B)
        IMPLIES  (A  IMPLIES C)

Peirce : PROPOSITION
    ((A IMPLIES B) IMPLIES A) IMPLIES A

Contra : PROPOSITION
     (A IMPLIES B) IMPLIES NOT B IMPLIES NOT A


schub : PROPOSITION
    NOT (
     (A11 OR A12) AND
     (A21 OR A22) AND
     (A31 OR A32) AND
     NOT (A11 AND A21) AND
     NOT (A11 AND A31) AND
     NOT (A21 AND A31) AND
     NOT (A12 AND A22) AND
     NOT (A12 AND A32) AND
     NOT (A22 AND A32))

END sequent_calculus
```

Abbildung 1.5: PVS Datei `sequent.pvs`

Mit M-x `(x-show-current-proof)` kann man den aktuellen partiellen Beweis grafisch anzeigen. Dieser zeigt zunächst nur die verwendeten Regeln; durch Klicken auf die $\vdash$ Symbole bekommt die jeweiligen Sequenzen zu sehen. Einen fertigen Beweis zeigt man mit M-x `(x-show-proof)`. Wie immer bei Emacs bedeutet M-x das gleichzeitige Drücken von ALT und x. Mit M-p und M-n kann man beim `Rule?` prompt die bisher abgesetzten Befehle zur Wiederverwendung durchgehen. Mit M-s kann man einen partiell eingegebenen Befehl komplettieren. Vollständige Dokumentation findet sich auf `pvs.csl.sri.com`.

1.8.3 Vollständigkeit des Sequenzenkalküls

Wir haben schon gesehen, dass die Regeln des Sequenzenkalküls korrekt sind in dem Sinne, dass tatsächlich nur tautologische Sequenzen bewiesen werden können. Wir wollen jetzt zeigen, dass die Regeln auch ausreichend sind in dem Sinne, dass alle tautologischen Sequenzen auch bewiesen werden können. Diese als *Vollständigkeit* bezeichnete Eigenschaft kann zusammen mit der Korrektheit als Überprüfung des Designs der Beweisregeln gedeutet werden. Hätten wir zum Beispiel die Regel $\vee$-R "vergessen", so würde das nicht die Korrektheit, wohl aber die Vollständigkeit kompromittieren, denn die tautologische Sequenz $A \Longrightarrow A \vee B$ wäre dann nicht beweisbar.

Etwas interessanter wird es, wenn man die Regeln Contr-L und Contr-R weglässt und außerdem $\vee$-R durch die folgenden zwei Regeln ersetzt: $\frac{\Gamma \Longrightarrow \Delta, \phi}{\Gamma \Longrightarrow \Delta, \phi \vee \psi}$ und $\frac{\Gamma \Longrightarrow \Delta, \psi}{\Gamma \Longrightarrow \Delta, \phi \vee \psi}$. Intuition: Um $\phi \vee \psi$ zu beweisen sollte man sich entscheiden, ob man ϕ oder ψ beweist.

In diesem Fall wäre zum Beispiel $\Longrightarrow A \vee \neg A$ nicht beweisbar.

Auf der anderen Seite sind die Regeln Contr-L, Contr-R ohne diese Modifikation tatsächlich redundant.

Satz 11 (Vollständigkeit)
Wenn ϕ eine Tautologie ist, so existiert ein Beweis im Sequenzenkalkül mit Konklusion $\Longrightarrow \phi$.

Beweis Durch systematische Rückwärtsanwendung der Regeln mit Ausnahme von Contr-L und Contr-R kann man eine Herleitung mit Konklusion $\Longrightarrow \phi$ erzeugen, deren Prämissen *atomar* sind, d.h. nur Formeln der Form A für A Variable enthalten. Dies deshalb, weil jede nicht atomare Formel durch Rückwärtsanwendung struktureller Regeln in eine Form gebracht werden kann, auf die eine logische Regel anwendbar ist.

Liege also eine Herleitung H mit Konklusion $\Longrightarrow \phi$ und atomaren Prämissen vor. Sind alle diese atomaren Prämissen Axiome, so handelt es sich um einen Beweis und es ist nichts zu zeigen. Sollte die Herleitung H aber kein Beweis sein, ist also eine ihrer Prämissen von der Form $\Gamma \Longrightarrow \Delta$ wobei Γ, Δ Listen von Variablen sind, so dass $\Gamma \cap \Delta = \emptyset$; dann können wir eine Belegung η finden, die diese atomare Prämisse falsifiziert, also nicht erfüllt. Hierzu setzen wir $\eta(A) = \mathtt{tt}$, falls $A \in \Gamma$ und $\eta(B) = \mathtt{ff}$, falls $B \in \Delta$. Auf denjenigen Variablen, die weder in Γ noch in Δ vorkommen, setzen wir η beliebig fest. Es ist klar, dass diese Belegung die Prämisse $\Gamma \Longrightarrow \Delta$ falsifiziert.

Wir behaupten nun, dass diese Belegung η alle Sequenzen auf dem Pfad in der Herleitung H von der Prämisse $\Gamma \Longrightarrow \Delta$ zur Wurzel $\Longrightarrow \phi$ falsifiziert. Insbesondere zeigt das,

dass die Konklusion $\Longrightarrow \phi$ gar nicht tautologisch sein kann, weil ja auch sie, also ϕ von der Belegung η falsifiziert wird.

Um zu zeigen, dass alle Sequenzen auf dem genannten Pfad falsifiziert werden, beweisen wir dass für jede Regel folgendes gilt: erfüllt eine Belegung die Konklusion einer Regel, so erfüllt sie alle Prämissen der Regel.

Ist dann nämlich eine der Prämissen nicht erfüllt, so kann auch die Konklusion der Regel nicht erfüllt sein.

Betrachten wir also die Regel $\wedge$-R. Ihre Prämissen haben die Form $\Gamma \Longrightarrow \Delta, \phi$ und $\Gamma \Longrightarrow \Delta, \psi$, während ihre Konklusion $\Gamma \Longrightarrow \Delta, \phi \wedge \psi$ ist. Erfüllt nun eine Belegung η diese Konklusion, so erfüllt sie offensichtlich auch beide Prämissen. Als weiteres Beispiel betrachten wir die Regel $\neg$-L. Hier haben wir eine Prämisse der Form $\Gamma \Longrightarrow \Delta, \phi$ und Konklusion $\Gamma, \neg\phi \Longrightarrow \Delta$. Erfüllt η diese Konklusion und gilt weiterhin $\eta \models \bigwedge \Gamma$, so muss $\eta \models \bigvee \Delta \vee \phi$ gelten. Wenn nämlich $\eta \not\models \phi$, also $\eta \models \neg\phi$, so folgt $\eta \models \bigvee \Delta$, da ja η die Konklusion erfüllt.

Auf ähnliche Weise argumentiert man auch bei den anderen Regeln, die als Übung verbleiben.

Wir fassen noch einmal zusammen. Alle Regeln des Sequenzenkalküls *reflektieren* die Gültigkeit in dem Sinne, dass wenn eine Belegung die Konklusion einer Regel erfüllt, sie dann auch alle ihre Prämissen erfüllt. Zu jeder Sequenz, tautologisch oder nicht, findet man eine Herleitung mit atomaren Prämissen. Ist eine dieser Prämissen kein Axiom, so kann man eine widerlegende Belegung angeben, die dann auch die gegebene Sequenz widerlegt. Somit muss jede Herleitung einer Tautologie mit atomaren Prämissen schon ein Beweis sein.

1.8.4 Zulässige und herleitbare Regeln

Wir können den Sequenzenkalkül auf zwei Arten um Regeln erweitern, ohne dabei die Korrektheit zu verlieren. Zum einen können wir eine Regel hinzugeben, die man durch mehrere vorhandene Regeln implementieren kann. So eine Regel heißt herleitbar. Zum Beispiel ist folgende Regel für die "NAND"-Verknüpfung herleitbar. $\dfrac{\Gamma, \phi, \psi \Longrightarrow \Delta}{\Gamma \Longrightarrow \Delta, \neg(\phi \wedge \phi)}$

Definition 18
Eine Regel ist herleitbar, wenn es zu jeder ihrer konkreten Instanzen eine Herleitung gibt, die dieselben Prämissen und Konklusion hat.

Interessanter sind zulässige Regeln. Hier verlangt man nur, dass ihre Hinzunahme den Kalkül nicht verstärkt.

Definition 19
Eine Regel ist *zulässig*, wenn jede Sequenz, die unter Hinzunahme der neuen Regel bewiesen werden kann, auch schon im ursprünglichen Kalkül bewiesen werden kann.

Satz 12
Eine Regel ist genau dann zulässig, wenn gilt: Sind alle ihre Prämissen Tautologien, so auch ihre Konklusion.

BEWEIS Wenn die genannte Bedingung zutrifft, so lässt sich der Beweis der Korrektheit des Sequenzenkalküls leicht auf die neue Regel erweitern. Man kann also mit der neuen Regel auch wieder nur die Tautologien beweisen, welche ja aufgrund der Vollständigkeit schon ohne dieselbe hergeleitet werden können.

Ist umgekehrt die Bedingung nicht erfüllt, gibt es also Tautologien $S_1, \ldots, S_\ell$ und eine Nicht-Tautologie S, sodass $\frac{S_1 \ \ldots \ S_\ell}{S}$ Instanz der Regel ist, so erhalten wir einen "Beweis" im Sequenzenkalkül der Sequenz S, indem wir die tautologischen Sequenzen $S_1, \ldots, S_n$ beweisen und dann diese Regel anwenden. Solch eine Regel kann also nicht zulässig sein.■

Ein wichtiges Beispiel für eine zulässige Regel im Sequenzenkalkül ist die folgende *Schnittregel*

$$\frac{\Gamma \Longrightarrow \Delta, \phi \qquad \Gamma, \phi \Longrightarrow \Delta}{\Gamma \Longrightarrow \Delta}(\text{CUT})$$

Intuitiv erlaubt es diese Regel, jederzeit ein Zwischenziel (hier ϕ) einzuführen um dann zunächst das Zwischenziel zu zeigen und anschließend das eigentliche Ziel unter Zuhilfenahme von ϕ. In PVS wird diese Regel durch den Befehl `(case ...)` repräsentiert.

Im Falle der Schnittregel kann man die Zulässigkeit auch syntaktisch begründen, indem zeigt, wie jeder Beweis, der die Schnittregel verwendet, so umbauen kann, dass die Schnittregel nicht benutzt wird. Hierzu verwendet man die Schnittregel in der folgenden etwas allgemeineren Form

$$\frac{\Gamma \Longrightarrow \Delta, \phi \qquad \Theta, \phi \Longrightarrow \Psi}{\Gamma, \Delta \Longrightarrow \Delta, \Psi}$$

Diese *Schnittelimination* bietet eine Möglichkeit, die Widerspruchsfreiheit eines Beweissystems mit rein syntaktischen Mitteln zu zeigen: Dass ein Beweissystem ohne Schnittregel widerspruchsfrei ist, ist oft unmittelbar klar; so sieht man im Sequenzenkalkül auch direkt ohne semantische Überlegungen, dass kein Beweis der Sequenz $\Longrightarrow$ existiert. Was sollte denn dessen letzte Regel sein? Solche Überlegungen sind Gegenstand der *Beweistheorie*, welche durch G. Gentzen begründet wurde. Gentzen hatte gezeigt, dass eine auf dem von ihm zu diesem Zweck eingeführten Sequenzenkalkül basierende Axiomatisierung der natürlichen Zahlen widerspruchsfrei ist und konnte außerdem die in dieser Axiomatisierung beweisbar terminierenden Rekursionsschemata hinsichtlich ihres Wachstums charakterisieren.

Zwei weitere im Sequenzenkalkül zulässige Regeln sind die Abschwächungsregeln

$$\frac{\Gamma \Longrightarrow \Delta}{\Gamma, \phi \Longrightarrow \Delta}(\text{WEAK-L})$$

$$\frac{\Gamma \Longrightarrow \Delta}{\Gamma \Longrightarrow \Delta, \phi}(\text{WEAK-R})$$

In PVS heißen diese `(del...)`.

1.9 Resolution

Die systematische Rückwärtsanwendung der Beweisregeln beinhaltet neben der Methode der Wahrheitstafeln ein weiteres Entscheidungsverfahren für Tautologien und auch die Erfüllbarkeit, nachdem ja eine Formel erfüllbar ist, genau dann, wenn ihre Negation nicht tautologisch ist. Um festzustellen, ob eine Formel ϕ erfüllbar ist, genügt es also, einen Beweis im Sequenzenkalkül für ihre Negation zu suchen. Findet sich ein Beweis für $\neg\phi$, so ist ϕ unerfüllbar. Führt der Beweisversuch auf eine atomare Sequenz, die kein Axiom ist, so ist $\neg\phi$ keine Tautologie, dementsprechend ist ϕ erfüllbar und die atomare Sequenz liefert wie im Beweis der Vollständigkeit eine erfüllende Belegung.

Nachdem bei Rückwärtsanwendung der konjunktiven Regeln ("split", $\Rightarrow$-L, $\wedge$-R, ...) die Gesamtgröße der aktuell zu beweisenden Sequenzen bis zu verdoppelt wird, ist die Laufzeit dieses Entscheidungsverfahrens im schlechtesten Falle exponentiell. Hat man zum Beispiel $\Gamma, \phi \Rightarrow \psi \Longrightarrow \Delta$ zu zeigen, dann führt das nach Rückwärtsanwendung von $\Rightarrow$-L auf $\Gamma \Longrightarrow \Delta, \phi$ und $\Gamma, \psi \Longrightarrow \Delta$. Man muss dann die Nebenformeln Γ, Δ in jedem der beiden Zweige separat bearbeiten, wodurch wie gesagt, der Gesamtaufwand exponentiell ansteigen kann.

Wir lernen jetzt ein Verfahren—die *Resolutionsmethode*—zur Entscheidung der Erfüllbarkeit kennen, welches einen höheren Grad an Wiederverwendung erlaubt, ohne dabei die Zielgerichtetheit zu verlieren. Leider ist auch dieses Verfahren im schlechtesten Falle exponentiell. In der Tat ist die Frage nach der Existenz eines Entscheidungsverfahrens für die Erfüllbarkeit aussagenlogischer Formeln mit polynomieller Laufzeit nach wie vor ungelöst und unter dem Kürzel P-NP als größtes offenes Problem der theoretischen Informatik bekannt.

Die Resolutionsmethode ermöglicht es, von einer vorgelegten KNF zu entscheiden, ob sie erfüllbar ist, oder nicht. Will man mithilfe der Resolutionsmethode entscheiden, ob eine gegebene Formel erfüllbar ist, so ist diese zunächst wie in Bemerkung **??** angegeben in eine erfüllungsäquivalente KNF zu übersetzen. Traditionell wird bei der Resolutionsmethode eine KNF als *Menge* von Klauseln repräsentiert, die sich natürlich implizit als Konjunktion über die in ihr enthaltenen Klauseln versteht. Ebenso werden Klauseln traditionell als Mengen von Literalen repräsentiert, die sich dann implizit als Disjunktion über die enthaltenen Literale versteht.

Dies erlaubt die Verwendung mengentheoretischer Notation, wie $C\backslash\{\ell\}$ für die Klausel, die entsteht, wenn man aus der Klausel C das Literal ℓ streicht.

Definition 20 (Resolvente)
Sei ℓ ein Literal. Die *Negation* $\neg\ell$ ist definiert als $\neg A$, falls $\ell = A$ und A, falls $\ell = \neg A$.

Seien C_1 und C_2 Klauseln, derart, dass ein Literal existiert, sodass ℓ in C_1 vorkommt und $\neg\ell$ in C_2 vorkommt. Die *Resolvente* von C_1 und C_2 ist dann definiert als die Klausel $C_1 \setminus \{\ell\} \cup C_2 \setminus \{\neg\ell\}$.

Zum Beispiel ist die Resolvente von $C_1 := \{A, B, \neg C\}$ und $C_2 := \{\neg D, C\}$ die Klausel $C := \{A, B, \neg D\}$, welche ja $A \vee B \vee \neg D$ bedeutet. Strenggenommen ist die Resolvente nicht eindeutig, denn es könnte ja sein, dass C_1 sowohl ℓ, als auch ℓ' und C_2 dann sowohl $\neg\ell$, als auch $\neg\ell'$ enthält, wodurch sich dann zwei mögliche Resolventen ergäben, einmal

```
RESO(Klauselmenge 𝒞){
  fertig = false;
  while (!fertig) {
    if (∅ ∈ 𝒞) fertig = true;
    else if (es gibt Klauseln C1, C2 ∈ 𝒞, deren Resolvente C nicht in 𝒞 ist)
      𝒞 = 𝒞 ∪ C;
    else fertig = true;
  }
  if (∅ ∈ 𝒞) return "unerfüllbar";
  else return "erfüllbar"
}
```

Abbildung 1.6: Die Resolutionsmethode

unter Entfernung von $\ell, \neg\ell$ und ein anderes Mal unter Entfernung von $\ell', \neg\ell'$. In diesem Fall enthält jedoch die Resolvente sowohl ℓ' als auch $\neg\ell'$, bzw. sowohl ℓ als auch $\neg\ell$ und ist somit äquivalent zu `tt` und deshalb uninteressant. (Eine Klausel, die nichts ausschließt, bringt nichts.)

Lemma 2 (Korrektheit der Resolution)
Erfüllt eine Belegung η zwei Klauseln C_1 und C_2, so erfüllt sie auch deren (strenggenommen jede ihrer) Resolvente(n).

Beweis Die Resolvente sei $C_1 \setminus \{\ell\} \cup C_2 \setminus \{\neg\ell\}$. Wenn $\eta(\ell) = \texttt{ff}$, so gilt sogar $\eta \models C_1 \setminus \{\ell\}$. Wenn hingegen $\eta(\ell) = \texttt{tt}$, so gilt sogar $\eta \models C_2 \setminus \{\neg\ell\}$. In jedem Fall wird also die Vereinigung dieser beiden Restklauseln erfüllt, die aber ist gerade die Resolvente. ∎

Die Resolutionsmethode besteht nun darin, aus einer gegebenen KNF geschrieben als Klauselmenge solange Resolventen herzuleiten und diese zur Klauselmenge hinzuzugeben, bis entweder die leere Klausel erzeugt wurde, oder aber keine neuen Klauseln mehr entstehen. Formal ist dieses Verfahren in Abb. **??** angegeben. Für Beweise günstiger ist folgende abstrakte Version:

Definition 21
Eine Klausel C ist aus einer Klauselmenge $\mathcal{C}$ herleitbar genau dann, wenn eine Folge von Klauseln $C_1, \ldots, C_n$ existiert, sodass $C_n = C$ und für alle $i = 1, \ldots, n$ gilt: Entweder ist $C_i \in \mathcal{C}$, oder es gibt $j, k < i$ sodass C_i Resolvente von C_j und C_k ist.

Es sollte klar sein, dass der Algorithmus in Abb. **??** geanu dann „$\mathcal{C}$ ist erfüllbar.“ antwortet, wenn keine Herleitung der leeren Klausel aus $\mathcal{C}$ existiert. Wir zeigen jetzt, dass dies zur Erfüllbarkeit von $\mathcal{C}$ äquivalent ist.

Satz 13 (Korrektheit und Vollständigkeit der Resolutionsmethode)
Eine Klauselmenge $\mathcal{C}$ ist genau dann erfüllbar, wenn die leere Klausel nicht aus $\mathcal{C}$ durch Resolution herleitbar ist.

BEWEIS Lässt sich die leere Klausel aus $\mathcal{C}$ herleiten, so ist $\mathcal{C}$ wg. Lemma **??** unerfüllbar. Umgekehrt nehmen wir an, $\mathcal{C}$ sei unerfüllbar und zeigen die Herleitbarkeit der leeren Klausel durch Induktion über die anzahl der Variablen in $\mathcal{C}$. Ist diese Null, so muss $\mathcal{C} = \{\emptyset\}$ sein, denn die andere Möglichkeit $\mathcal{C} = \{\}$ wird ja von jeder Belegung erfüllt. Aus $\{\emptyset\}$ kann man aber $\emptyset$ herleiten.

Ist die Anzahl der Variablen größer als Null, so greifen wir willkürlich eine Variable A heraus und bilden die Klauselmenge $\mathcal{C}_1$, welche entsteht, indem wir das Literal A aus allen Klauseln streichen, in denen es vorkommt und außerdem alle Klauseln, in denen das Literal $\neg A$ vorkommt, gänzlich aus $\mathcal{C}$ entfernen. Erfüllt eine Belegung η diese neue Klauselmenge $\mathcal{C}_1$, so ist auch $\mathcal{C}$ selbst erfüllbar und zwar durch $\eta[A \mapsto \mathtt{ff}]$. Also muss nach Annahme $\mathcal{C}_1$ unerfüllbar sein und man kann nach Induktionsvoraussetzung daraus die leere Klausel herleiten. Indem man zu der entsprechenden Herleitung das entfernte Literal A überall wieder hinzugibt, erhalt man eine Herleitung entweder von $\emptyset$, oder von $\{A\}$ aus $\mathcal{C}$. In ersterem Fall haben wir die gewünschte Herleitung von $\emptyset$ aus $\mathcal{C}$; ansonsten bilden wir analog die Klauselmenge $\mathcal{C}_2$, die aus $\mathcal{C}$ dadurch entsteht, dass wir das Literal $\neg A$ aus allen Klauseln streichen, die es enthalten und diejenigen Klauseln, die das entgegengesetzte Literal A enthalten, ganz entfernt. Wir erhalten auf diese Weise eine Herleitung der leeren Klausel oder aber der Klausel $\{\neg A\}$ aus $\mathcal{C}$. Im ersteren Fall ist wiederum $\emptyset$ herleitbar. Im zweiten Falle haben wir aber sowohl die Klausel $\{A\}$, als auch die Klausel $\{\neg A\}$ aus $\mathcal{C}$ hergeleitet. Ein weiterer Resolutionsschritt liefert dann die leere Klausel. ■

Aufgrund der Kompaktheit (Satz **??**) gilt dieses Ergebnis auch für unendliche Klauselmengen.

Beispiel 3
Sei

$$\mathcal{C} = \{C_1, C_2, C_3, C_4, C_5\}$$
$$C_1 = \{A, B, C\}\ C_2 = \{\neg A, B, C\}\ C_3 = \{B, \neg C\}\ C_4 = \{\neg B\}$$

Aus C_1 und C_2 erhalten wir die Resolvente $C_5 = \{B, C\}$. Zusammen mit C_3 ergibt sich dann $C_6 = \{B\}$ und dann die leere Klausel als Resolvente von C_6 und C_4. Die ursprüngliche Klauselmenge ist unerfüllbar.

Beispiel 4
Wir betrachten wieder die Aussagen der Politiker vor der Wahl. Auf KNF gebracht ergibt sich die folgende Klauselmenge:

$$\mathcal{C} = \{C_1, C_2, C_3, C_4, C_5\}$$
$$C_1 = \{F, A\}\ C_2 = \{\neg J, \neg F\}\ C_3 = \{F, J\}\ C_4 = \{\neg A, \neg J\}\ C_5 = \{J, A\}\}$$

Die Klauseln C_1, C_4 haben die Resolvente $C_6 = \{F, \neg J\}$. Die Klauseln C_4, C_5 könnte man auf zwei Arten resolvieren. Er ergibt sich aber wie angekündigt jeweils nur eine triviale Klausel. Aus C_2 und C_5 ergibt sich die Resolvente $C_7 = \{\neg F, A\}$. Aus C_6 und C_7 erhalten wir $C_8 = \{\neg J, A\}$ und dann mit C_5 die Klausel $C_9 = \{A\}$. Wegen Lemma **??** muss jede Belegung η, die die ursprüngliche Klauselmenge $\mathcal{C}$ erfüllt, auch die Klausel C_9

erfüllen. Das aber bedeutet, dass $\eta(A) = \mathtt{tt}$, also dass Angela der Regierung angehört. Aus C_9 und C_4 erhalten wir $C_{10} = \{\neg J\}$ also gehört Joschka nicht der Regierung an. Schließlich erhalten wir $C_{11} = \{F\}$ mit C_3.

In diesem Beispiel hatten wir nur Zweier- und Einerklauseln. Diese haben die gute Eigenschaft, dass Resolventen von solchen auch wieder welche (2er & 1er) sind. Daher ist in diesem Fall die Resolutionsmethode besonders effizient: ihre Laufzeit ist $O(n^3)$, wenn n die Zahl der Variablen ist.

Ein ähnlicher Fall liegt vor, wenn alle beteiligten Klauseln *Hornklauseln* sind.

Definition 22
Eine Klausel, die höchstens eine nichtnegierte Variable enthält, heißt *Hornklausel*. Hornklauseln sind äquivalent zu Formeln der Gestalt $A_1 \wedge \cdots \wedge A_\ell \Rightarrow B$ (eine nichtnegierte Variable, nämlich B), bzw. $A_1 \wedge \cdots \wedge A_\ell \Rightarrow \bot$.

Resolventen von Hornklauseln sind auch wieder Hornklauseln. Auch für Mengen von Hornklauseln lässt die Resolutionsmethode eine effiziente Implementierung zu, die darauf basiert, eine Resolution unter Beteiligung von $A_1 \wedge \cdots \wedge A_\ell \Rightarrow B$, bzw. $A_1 \wedge \cdots \wedge A_\ell \Rightarrow \bot$ nur dann durchzuführen, wenn die Einerklauseln $\{A_1\}, \ldots \{A_n\}$ bereits alle hergeleitet worden sind. In diesem Fall führen n Resolutionsschritte auf die Einerklausel B, bzw. auf die leere Klausel. Man kann sich überlegen, dass man durch diese Strategie nichts verliert. Entweder dadurch, dass man einen vorgelegten Resolutionsbeweis entsprechend umbaut, oder dadurch, dass man zeigt, dass auch schon diese eingeschränkte Methode vollständig ist.

Leider hat die Resolutionsmethode im allgemeinen Falle exponentielle Laufzeit selbst bei geschicktester Wahl der Resolventen. In der Tat wurde von Haken gezeigt, dass jeder Resolutionsbeweis des Schubfachprinzips exponentielle Länge hat. Die Instanz (2,3) des Schubfachprinzips ist gerade die Formel `schub` aus der Beispieldatei `sequent.pvs`: „Drei Briefe können nicht so auf zwei Schubfächer verteilt werden, dass in keinem Schubfach mehr als ein Brief liegt.“

1.9.1 DPLL-Algorithmus

Die Resolutionsmethode hat den Nachteil, dass man die zu resolvierenden Klauseln geschickt wählen muss, bzw. wie im Algorithmus aus Abb. **??** alle möglichen Resolventen berechnen muss. Eine Möglichkeit, etwas zielgerichteter vorzugehen, bietet der DPLL-Algorithmus (Davis-Putnam-Loveland-Logemann). Letztendlich läuft er auf das Prinzip “Vorrechnen und wenn nichts mehr geht Probieren” hinaus, welches man ja auch beim Lösen von Sudokus anwendet. Wir präsentieren das Verfahren hier als Spezialfall der Resolutionsmethode.

Definition 23
Eine Einerklausel ist eine Klausel, die nur aus einem Literal besteht. Eine Einerresolution ist ein Resolutionsschritt bei dem eine der beiden beteiligten Klauseln eine Einerklausel ist.

```
DPLL(C) {                                                          (1)
  C=UNIT(C);                                                       (2)
  if (∅ ∈ C)                                                       (3)
    return "unerfüllbar";                                          (4)
  else if (es gibt Variable A, sodass {A} ∉ C und {¬A} ∉ C){       (5)
    wähle solch eine Variable A;                                   (6)
    if (DPLL(C ∪ {A}) == "unerfüllbar")                            (7)
      return DPLL(C ∪ {¬A});                                       (8)
    else return "erfüllbar";                                       (9)
  }                                                               (10)
}                                                                 (11)
UNIT(Klauselmenge C){                                             (12)
  Schließe C unter Einerresolution ab;                            (13)
  return C;}                                                      (14)
```

Abbildung 1.7: DPLL Algorithmus

Mit Einerresolution alleine kommt man bei Hornklauseln zum Ziel, nicht aber bei beliebigen Klauseln, so etwa bei der Formalisierung von "Vor der Wahl".

Die Einerresolution entspricht letztendlich dem Propagieren einfacher Konsequenzen. Etwa beim Sudoku "Wenn hier eine drei steht, dann kann dort keine drei stehen". Der DPLL-Algorithmus basiert nun darauf, die Einerresolution zu verstärken, dadurch dass man, wenn die Einerresolution nicht mehr weiterführt, eine Fallunterscheidung über eine beliebige Variable A durchführt und also der aktuellen Klauselmenge zunächst die Einerklausel $\{A\}$ und dann separat noch einmal die Einerklausel $\{\neg A\}$ hinzufügt. Sind beide so entstandenen neuen Klauselmengen unerfüllbar, dann auch die ursprüngliche.

Lemma 3
Ist eine bestimmte Klausel K , z.B. die leere Klausel, sowohl aus $\mathcal{C} \cup \{A\}$, als auch aus $\mathcal{C} \cup \{\neg A\}$ herleitbar, so ist K auch aus $\mathcal{C}$ selbst herleitbar und umgekehrt.

Beweis Betrachten wir die Herleitung H von K aus $\mathcal{C} \cup \{A\}$. Wir wollen mit denselben Schritten eine Herleitung ausgehend von $\mathcal{C}$ konstruieren. Schritte, die die Klausel $\{A\}$ benutzen, lassen wir einfach weg. Da die Anwendbarkeit der Resolution immer nur vom Vorhandensein, nie aber die Abwesenheit bestimmter Literale abhängt, erhalten wir so eine Herleitung von K selbst oder aber $K \cup \{\neg A\}$ aus $\mathcal{C}$. Analog erhalten wir eine Herleitung von $K \cup \{A\}$ bzw. K selbst. Maximal ein weiterer Resolutionsschritt liefert dann K. ∎

Aus diesen Betrachtungen ergibt sich der DPLL-Algorithmus, dessen Pseudocode in Abb. **??** angegeben ist. Es ist klar, dass der DPLL-Algorithmus korrekt und vollständig ist. Schlimmstenfalls wird eben solange verzweigt, bis alle Variablen entschieden sind. In der Praxis funktioniert DPLL erstaunlich gut, was u.a. auch daran liegt, dass er sehr effizient implementiert werden kann.

Als Beispiel nehmen wir nochmal die Politiker nach der Wahl aus Beispiel **??**. Einerresolution bringt zunächst einmal nichts Neues, da keine Einerklauseln vorhanden sind. Wir wählen zunächst die Variable J und geben zeitweilig die Klausel $\{J\}$ hinzu. Dies führt per Einerresolution auf die Klauseln $\{\neg F\}$, $\{A\}$, $\{\neg J\}$, $\emptyset$. Also ergibt sich in Zeile (7) das Ergebnis "unerfüllbar" und wir probieren es durch Hinzunahme der Klausel $\{\neg J\}$. Einerresolution liefert zusätzlich $\{F\}, \{A\}$. Die Bedingung aus Programmzeile (5) ist nunmehr nicht erfüllt, den alle Variaben sind besetzt, wir liefern daher in Zeile (9) das Ergebnis "erfüllbar" und sind fertig. Die erfüllende Belegung lässt sich nun auch direkt aus den vorhandenen Einerklauseln ablesen.

Effiziente Implementierungen des DPLL-Algorithmus sind als SAT-Solver bekannt und erzielen erstaunlich gute Ergebnisse. So kann ein Sudoku nach Ubersetzung in Aussagenlogik mit einem SAT-Solver in kurzer Zeit automatisch gelöst werden. Auch in der automatischen Programmanalyse, -verifikation und -generierung werden SAT-Solver erfolgreich eingesetzt. Frei verfügbare SAT-Solver sind z.B. zChaff und Minisat.

2 Prädikatenlogik

Die Prädikatenlogik, um die es in diesem Kapitel geht, erlaubt es, über Elemente, Eigenschaften von Elementen, Termen und Operationen zu sprechen. Typische Beispiele von prädikatenlogischen Sätzen sind

- Jeder Studierende hat eine Matrikelnummer.
- Wenn ein Studierender sich nicht zurückmeldet, so wird er exmatrikuliert.
- Jeder Mensch ist ein Philosoph.
- Zu jeder natürlichen Zahl n existiert eine natürliche Zahl d, sodass $2^d \leq n$ und $n < 2^{d+1}$.
- In jeder Vorlesung gibt es einen Studierenden, der alles versteht, aber es gibt keine Studierenden, die alle Vorlesungen verstehen.

Als Formeln der Prädikatenlogik lesen sich diese Sätze so:

- $\forall x.\text{studierender}(x) \Rightarrow \exists n.\text{zahl}(x) \wedge \text{hat\_matrikelnummer}(x, n)$
- $\forall x.\text{studierender}(x) \Rightarrow \neg\text{hat\_sich\_zurückgemeldet}(x) \Rightarrow \text{wird\_exmatrikuliert}(x)$
- $\forall x.\text{mensch}(x) \Rightarrow \text{philosoph}(x)$
- $\forall n.\text{zahl}(n) \Rightarrow \exists d.\text{zahl}(d) \wedge \text{klgl}(\text{hoch}(\text{zwei}(), d), n) \wedge \text{kl}(n, \text{hoch}(\text{zwei}(), \text{plus}(d, \text{eins}())))$
- $(\forall x.\text{vorlesung}(x) \Rightarrow \exists y.\text{studierender}(y) \wedge \text{versteht}(x, y)) \wedge \neg\exists y.\text{studierender}(y).\forall x.\text{vorlesung}(x) \Rightarrow \text{versteht}(x, y)$

Die Symbole $\forall, \exists$ heißen *Quantoren* und werden "für alle", bzw. "es gibt" gelesen. Wie in der Aussagenlogik schreibt man diesen Formeln einen Wahrheitsgehalt zu, welcher natürlich davon abhängt, wie die einzelnen Prädikate (hier "studierender", "hat_matrikelnummer",...) interpretiert werden. Genau wie bei der Prädikatenlogik gibt es aber auch Formeln, die unabhängig von der Interpretation immer gelten, z.B.: $(\forall x.P(x) \wedge Q(x)) \Rightarrow (\forall x.P(x)) \wedge (\forall x.Q(x))$.

2.1 Syntax der Prädikatenlogik

Wir legen nun formal die Syntax der Prädikatenlogik fest. Hierzu muss zunächst festgelegt werden, welche Prädikate und Operationen überhaupt zugelassen werden.

Definition 24 (Prädikatenlogische Sprache)
Eine *prädikatenlogische Sprache* (kurz Sprache) ist ein Tripel $(\mathcal{P}, \mathcal{F}, ar)$, wobei $\mathcal{P}$ und $\mathcal{F}$ voneinander disjunkte Mengen sind und $ar : \mathcal{P} \cup \mathcal{F} \to \mathbf{N}$ eine Funktion ist. Die Elemente von $\mathcal{P}$ heißen *Prädikatsymbole*, die Elemente von $\mathcal{F}$ heißen *Funktionssymbole* der Sprache $\mathcal{L} = (\mathcal{P}, \mathcal{F}, ar)$.

Die Funktion ar weist jedem Prädikatssymbol P, bzw. Funktionssymbol f eine *Stelligkeit* (engl. *arity*) $ar(P)$, bzw. $ar(f)$ zu, welche angibt, auf wieviele Argumente man das Symbol anwenden muss.

Eine passende Sprache für die obigen Beispiele mit Studierenden ist gegeben durch $(\mathcal{P}, \mathcal{F}, ar)$, wobei:

- $\mathcal{P} = \{\text{studierender}, \text{hat\_matrikelnummer}, \text{zahl}, \text{hat\_sich\_zurückgemeldet}, \text{wird\_exmatrikuliert}, \text{vorlesung}, \text{versteht}(x, y)\}$
- $\mathcal{F} = \emptyset$,
- $ar(\text{studierender}) = 1$, $ar(\text{hat\_matrikelnummer}) = 2$, $ar(\text{zahl}) = 1$, $ar(\text{hat\_sich\_zurückgemeldet}) = 2$, $ar(\text{wird\_exmatrikuliert}) = 2$, $ar(\text{vorlesung}) = 1$, $ar(\text{versteht})$

Eine prädikatenlogische Sprache entspricht einer Java-Schnittstelle. Sie gibt an, welche Operationen es gibt und wieviele Argumente sie erwarten. Anders als in Java haben alle Argumente denselben Typ, den man deshalb weglässt und auf deren Anzahl man sich also beschränken kann.

Ein Funktionssymbol f mit $ar(f) = n$ heißt n-stellig, ebenso für Prädikatsymbole.

Für das arithmetische Beispiel verwenden wir die Sprache $(\mathcal{P}, \mathcal{F}, ar)$, wobei:

- $\mathcal{P} = \{\text{zahl}, \text{klgl}, \text{kl}\}$,
- $\mathcal{F} = \{\text{hoch}, \text{plus}, \text{eins}, \text{zwei}\}$
- $ar(\text{zahl}) = 1$, $ar(\text{klgl}) = 2$, $ar(\text{kl}) = 1$, $ar(\text{hoch}) = 2$, $ar(\text{plus}) = 2$, $ar(\text{eins}) = 0$, $ar(\text{zwei}) = 0$

Zu beachten ist der Spezialfall Stelligkeit Null. Ein nullstelliges Funktionssymbol entspricht einer Konstante (im Bsp. die Symbole eins und zwei). Ein nullstelliges Prädikatsymbol entspricht einer aussagenlogischen Variablen.

Wir geben uns nun eine unendliche Menge von Variablen $\mathcal{X}$ vor, welche über Individuen (Studierenden, Zahlen, Menschen, ...) rangieren sollen. Wir verwenden in der Regel Buchstaben $x, y, z, \ldots$ für Variablen.

Definition 25 (Terme)
Es sei $\mathcal{L} = (\mathcal{P}, \mathcal{F}, ar)$ eine prädikatenlogische Sprache und X eine endliche Teilmenge von $\mathcal{X}$. Die *Terme* (über $\mathcal{L}$) sind wie folgt induktiv definiert.

- Jede Variable $x \in \mathcal{X}$ ist ein Term.

- Ist f ein k-stelliges Funktionssymbol und sind $t_1, \ldots, t_k$ Terme, so ist $f(t_1, \ldots, t_k)$ ein Term. Insbesondere ist jedes nullstellige Funktionssymbol (Konstante) ein Term.

Im obigen Beispiel mit Zahlen wäre also z.B. hoch(plus(x, eins())) ein Term.

Manchmal vereinbart man für Funktionssymbole kleiner Stelligkeit besondere Notationen, z.B. Infixnotation mit Bindekonventionen für + und ×, Juxtaposition (Nebeneinanderschreibung) für ×. Exponentialnotation für "hoch", etc. Auch lässt man bei 0-stelligen Funktionssymbolen gern die Klammern weg, schreibt also 0 statt 0(). All das ist konkrete Syntax, die verwendet wird, um Terme lesbar zu repräsentieren. Formal fassen wir Terme genauseo wie aussagenlogische Formeln als Syntaxbäume auf.

Definition 26 (Formeln)
Wiederum sei $\mathcal{L} = (\mathcal{P}, \mathcal{F}, ar)$ eine prädikatenlogische Sprache und X eine endliche Teilmenge von $\mathcal{X}$. Die *Formeln* über $\mathcal{L}$ sind wie folgt induktiv definiert.

- Ist P ein k-stelliges Prädikatsymbol und sind $t_1, \ldots, t_k$ Terme, so ist $P(t_1, \ldots, t_k)$ eine Formel. Formeln dieser Gestalt heißen *atomare Formeln.*
- Ist ϕ Formel, so auch $\neg\phi$.
- Sind ϕ, ψ Formeln, so auch $\phi \star \psi$, wobei $\star$ einer der aussagenlogischen Junktoren $\wedge, \vee, \Rightarrow, \Leftrightarrow$ ist.
- Ist ϕ Formel und x Variable, so sind auch $\forall x.\phi$ und $\exists x.\phi$ Formeln.

Sind zum Beispiel P, Q jeweils zweistellige Prädikatsymbole, so ist

$$\psi := (\forall x.P(x, y)) \Rightarrow \exists z.Q(y, z)$$

eine Formel.

2.1.1 Freie und gebundene Variablen

Ein Quantor $\forall x$, bzw. $\exists x$, *bindet* alle Vorkommen der Variablen x in seinem Geltungsbereich. Dieser Geltungsbereich erstreckt sich immer so weit nach rechts wie möglich, d.h. bis eine früher geöffnete Klammer geschlossen wird, oder das Ende der Formel erreicht ist.

Eine Variable, die nicht durch einen Quantor gebunden wird, ist *frei.* In der obigen Beispielformel ist also y freie Variable, nicht aber x und z, die ja durch die Quantoren gebunden sind.

Die später noch formal festzulegende Bedeutung einer Formel hängt von den Werten ihrer freien Variablen, nicht aber von den Werten ihrer gebundenen Variablen ab, da letztere ja durch die Quantoren mit Werten belegt werden.

Es kann passieren, dass eine freie Variable auch noch einmal an anderer Stelle einer Formel als gebundene Variable eingesetzt wird.

So ist zum Beispiel x freie Variable in der Formel $P(x, y) \wedge \forall x.Q(x, y)$, wird aber im zweiten Konjunkt in gebundener Form eingesetzt.

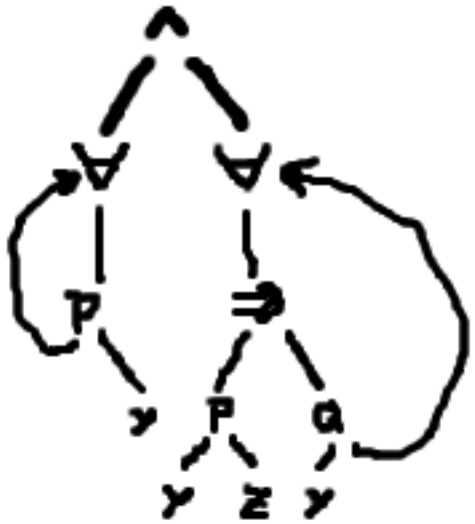

Abbildung 2.1: Syntaxbaum mit Rückwärtsverweisen für die Formel $(\forall z.P(z,y)) \wedge \forall x.P(y,z) \Rightarrow Q(y,x)$

Gebundene Variablen dürfen umbenannt werden, ohne die Bedeutung einer Formel zu verändern, so ist die Formel $P(x,y) \wedge \forall x.Q(x,y)$ in diesem Sinne äquivalent zur Formel $P(x,y) \wedge \forall z.Q(z,y)$.

Eine gebundene Variable sollte eigentlich gar nicht als Variable aufgefasst werden, sondern vielmehr als Zeiger auf den Quantor durch den sie eingeführt wurde. Dies wird deutlich, wenn man Formeln als Syntaxbäume mit Rückwärtsverweisen, wie in Figur **??** repräsentiert. Wir werden nunmehr Formeln, die sich nur in der Benennung ihrer gebundenen Variablen unterscheiden, miteinander identifizieren. D.h. also, dass zum Beispiel die Formeln $\forall x.P(x,x)$ und $\forall y.P(y,y)$ identifiziert werden, da beide Repräsentanten ein und desselben Syntaxbaums mit Rückwärtsverweisen sind. Ebenso identifizieren wir ja z.B. $A \wedge B \vee C$ mit $(A \wedge B) \vee C$, da beide denselben Syntaxbaum haben.

Definition 27 (Geschlossener Term/Formel)
Eine Formel ohne freie Variablen heißt *geschlossen*. Ebenso heißt ein Term t, in dem keine Variablen vorkommen, *geschlossener Term*.

Ein geschlossener Term wird oft auch als *Grundterm* bezeichnet.

2.1.2 Substitution

Ist ϕ eine Formel, x eine Variable und t ein Term so können wir x in ϕ durch t ersetzen und erhalten eine Formel $\phi[t/x]$. Diese ist wie folgt durch Induktion über Term- und

Formelaufbau definiert.

Definition 28
Sei t ein Term und x eine Variable. Die Operation $(--)[t/x]$ auf Formeln und Termen ist wie folgt definiert:

- $x[t/x] = t$,
- $y[t/x] = y$, falls $y \neq x$,
- $f(t_1, \ldots, t_k)[t/x] = f(t_1[t/x], \ldots, t_k[t/x])$,
- $P(t_1, \ldots, t_k)[t/x] = P(t_1[t/x], \ldots, t_k[t/x])$,
- $(\neg\phi)[t/x] = \neg(\phi[t/x])$,
- $(\phi \star \psi)[t/x] = \phi[t/x] \star \psi[t/x]$, falls $\star \in \{\vee, \wedge, \Rightarrow, \Leftrightarrow\}$
- $(\forall z.\phi)[t/x] = \forall z.(\phi[t/x])$
- $(\exists z.\phi)[t/x] = \exists z.(\phi[t/x])$

In den letzten beiden Klauseln ist die gebundene Variable verschieden von den Variablen, die in t vorkommen und auch verschieden von der zu substituierenden Variablen x zu wählen.

Zum Beispiel gilt

$$(\forall y.P(y, f(x)))[g(z)/x] = \forall y.P(y, f(g(z)))$$

Es gilt auch

$$(\forall y.P(y, f(x)))[g(y)/x] = \forall w.P(w, f(g(y)))$$

Im letzten Beispiel muss man vor der wörtlichen Substitution die gebundene Variable y umbenennen, damit die freie Variable y im substitutierten Term t nicht vom Binder $\forall y.$ "eingefangen" wird. Dies wird durch die Erklärung am Ende von Definition **??** verlangt. Ebenso gilt z.B. $(\forall x.P(x))[t/x] = \forall y.P(y)$ und nicht etwa $\forall x.P(t)$, da die gebundene Variable vor der Substitution auch von x verschieden zu wählen ist.

In der Regel werden wir nur geschlossene Terme substituieren (also für Variablen einsetzen), in welchem Falle keine Umbenennung erforderlich ist.

2.2 Semantik der Prädikatenlogik

Wir wollen nun Formeln der Prädikatenlogik ähnlich wie den aussagenlogischen Formeln Wahrheitswerte zuordnen. Dieser Wahrheitswert wird im allgemeinen davon abhängen, über welchen Bereich die Variablen rangieren, welche Prädikate und Funktionen die Prädikat- und Funktionssymbole bezeichnen, und welchen aktuellen Wert die freien Variablen haben. So ist etwa die Formel $\forall x.P(x, y)$ wahr, wenn Variablen über die natürlichen Zahlen rangieren, P die "größer-oder-gleich" Relation ist und y den Wert 0 hat,

denn alle natürlichen Zahlen sind ja ≥ 0. Rangieren die Variablen aber über Studierenden und Vorlesungen und bedeutet $P(x, y)$, dass x ein Studierender ist, der die Vorlesung y versteht, so ist im allgemeinen davon auszugehen, dass die Formel nicht wahr ist, egal welchen Wert y hat.

Definition 29
Sei $\mathcal{L} = (\mathcal{P}, \mathcal{F}, ar)$ eine Sprache. Eine *Struktur* $\mathcal{A}$ für $\mathcal{L}$ ist durch folgende Daten gegeben:

- Eine nichtleere Menge $|\mathcal{A}|$,
- Für jedes Funktionssymbol $f \in \mathcal{F}$ eine $ar(f)$-stellige Funktion

$$f_{\mathcal{A}} : \underbrace{|\mathcal{A}| \times \cdots \times |\mathcal{A}|}_{ar(f) \text{ Stück}} \to |\mathcal{A}|$$

- Für jedes Prädikatsymbol $P \in \mathcal{F}$ eine $ar(P)$-stellige Relation

$$P_{\mathcal{A}} \subseteq \underbrace{|\mathcal{A}| \times \cdots \times |\mathcal{A}|}_{ar(P) \text{ Stück}}$$

Definition 30
Sei $\mathcal{L} = (\mathcal{P}, \mathcal{F}, ar)$ eine Sprache und $\mathcal{A}$ eine Struktur für $\mathcal{L}$. Eine *Belegung* ist eine Funktion η, die jeder Variablen $x \in \mathcal{X}$ einen Wert $\eta(x) \in |\mathcal{A}|$ zuordnet.

Ist t ein Term über $\mathcal{L}$ und η eine Belegung, so ist die Bedeutung des Terms $[\![t]\!]_{\mathcal{A}}\eta$ in der Struktur $\mathcal{A}$ und unter der Belegung η definiert durch

- $[\![x]\!]_{\mathcal{A}}\eta = \eta(x)$
- $[\![f(t_1, \ldots, t_k)]\!]_{\mathcal{A}}\eta = f_{\mathcal{A}}([\![t_1]\!]_{\mathcal{A}}\eta, \ldots, [\![t_k]\!]_{\mathcal{A}}\eta)$

Ist ϕ eine Formel über $\mathcal{L}$ und η eine Belegung, so ist ihre Bedeutung $[\![\phi]\!]_{\mathcal{A}}\eta \in \mathbf{B}$ in der Struktur $\mathcal{A}$ und unter der Belegung η definiert durch

- $[\![P(t_1, \ldots, t_n)]\!]_{\mathcal{A}}\eta = \mathtt{tt}$, falls $([\![t_1]\!]_{\mathcal{A}}\eta, \ldots, [\![t_n]\!]_{\mathcal{A}}\eta) \in P_{\mathcal{A}}$.
- $[\![P(t_1, \ldots, t_n)]\!]_{\mathcal{A}}\eta = \mathtt{ff}$, falls $([\![t_1]\!]_{\mathcal{A}}\eta, \ldots, [\![t_n]\!]_{\mathcal{A}}\eta) \notin P_{\mathcal{A}}$.
- $[\![\neg\phi]\!]_{\mathcal{A}}\eta = \neg[\![\phi]\!]_{\mathcal{A}}\eta$
- $[\![\phi \star \psi]\!]_{\mathcal{A}}\eta = [\![\phi]\!]_{\mathcal{A}}\eta \star [\![\psi]\!]_{\mathcal{A}}\eta$, falls $\star \in \{\vee, \wedge, \Rightarrow, \Leftrightarrow\}$.
- $[\![\forall x.\phi]\!]_{\mathcal{A}}\eta = \mathtt{tt}$, falls $[\![\phi]\!]_{\mathcal{A}}(\eta[x \mapsto v]) = \mathtt{tt}$ für alle $v \in |\mathcal{A}|$ gilt.
- $[\![\forall x.\phi]\!]_{\mathcal{A}}\eta = \mathtt{ff}$, falls $[\![\phi]\!]_{\mathcal{A}}(\eta[x \mapsto v]) = \mathtt{ff}$ auch nur für ein $v \in |\mathcal{A}|$ gilt.
- $[\![\exists x.\phi]\!]_{\mathcal{A}}\eta = \mathtt{tt}$, falls mindestens ein $v \in |\mathcal{A}|$ existiert derart, dass $[\![\phi]\!]_{\mathcal{A}}(\eta[x \mapsto v]) = \mathtt{tt}$.
- $[\![\exists x.\phi]\!]_{\mathcal{A}}\eta = \mathtt{ff}$, falls $[\![\phi]\!]_{\mathcal{A}}(\eta[x \mapsto v]) = \mathtt{ff}$ für alle $v \in |\mathcal{A}|$ gilt.

Bei geschlossenen Formeln ist die Belegung irrelevant, wir schreiben in diesem Falle $[\![\phi]\!]_{\mathcal{A}}$ anstelle von $[\![\phi]\!]_{\mathcal{A}}\eta$ wobei ja η beliebig wäre.

Definition 31
Eine geschlossene Formel ϕ, die in allen Strukturen für ihre Sprache wahr ist, für die also gilt $[\![\phi]\!]_{\mathcal{A}} = \mathtt{tt}$ für alle $\mathcal{A}$, heißt *allgemeingültig*.

Eine geschlossene Formel ϕ, die zumindest in einer Struktur $\mathcal{A}$ wahr ist, heißt *erfüllbar*.

Die allgemeingültigen Formeln entsprechen den Tautologien der Aussagenlogik; die Strukturen der Prädikatenlogik entsprechen den Belegungen η der Aussagenlogik, wenn man jeweils bedenkt, dass aussagenlogische Variablen ja den nullstelligen Prädikatsymbolen entsprechen. Die Belegungen der Prädikatenlogik haben keine Entsprechung in der Aussagenlogik. Dass beide mit dem Buchstaben η bezeichnet werden und "Belegung" heißen, ist eine Koinzidenz.

Beispiele für allgemeingültige Formeln sind

- $\forall x.P(x) \Rightarrow \exists x.P(x)$
- $(\forall x.\forall y.R(x,y) \Rightarrow R(y,x)) \wedge (\forall x.\forall y.\forall z.R(x,y) \wedge R(y,z) \Rightarrow R(x,z)) \Rightarrow \forall x.\forall y.R(x,y) \Rightarrow R(x,x)$
- $\exists x.P(x) \Rightarrow \forall x.D(x) \Rightarrow P(x)$
- $(\forall x.P(x) \vee Q) \Rightarrow (\forall x.P(x)) \vee Q$
- $(\forall x.R(x, f(x))) \Rightarrow \forall x.\exists y.R(x,y)$

Wieder gilt: ϕ ist allgemeingültig genau dann, wenn $\neg\phi$ nicht erfüllbar ist; ϕ ist erfüllbar, genau dann, wenn $\neg\phi$ nicht allgemeingültig ist.

Definition 32 (Modell)
Sei ϕ eine geschlossene Formel. Eine Struktur $\mathcal{A}$, derart dass $[\![\phi]\!]_{\mathcal{A}} = \mathtt{tt}$ heißt *Modell* der Formel ϕ. Man schreibt in diesem Fall $\mathcal{A} \models \phi$. Eine Formel ist also erfüllbar, genau dann, wenn sie ein Modell hat.

2.3 Sequenzenkalkül für die Prädikatenlogik

Wir wollen jetzt den Sequenzenkalkül auf die Prädikatenlogik erweitern. Die Regeln für die aussagenlogischen Junktoren werden unverändert beibehalten, nur für die beiden Quantoren benötigen wir jeweils eine Links- und eine Rechtsregel.

Definition 33
Sei $\mathcal{L}$ eine prädikatenlogische Sprache. Eine Sequenz über $\mathcal{L}$ ist ein Paar von Listen geschlossener Formeln über $\mathcal{L}$. Man schreibt solch eine Sequenz in der Form $\Gamma \Longrightarrow_{\mathcal{L}} \Delta$, wobei Γ, Δ durch Komma getrennte Listen von Formeln sind. Notationen und Bezeichnungen aus Def. **??** werden sinngemäß übernommen.

$$\frac{\Gamma \Longrightarrow_{\mathcal{L}} \Delta, \phi[t/x] \qquad t \text{ geschlossener Term über } \mathcal{L}}{\Gamma \Longrightarrow_{\mathcal{L}} \Delta, \exists x.\phi}(\exists\text{-R})$$

$$\frac{\Gamma, \phi[t/x] \Longrightarrow_{\mathcal{L}} \Delta \qquad t \text{ geschlossener Term über } \mathcal{L}}{\Gamma, \forall x.\phi \Longrightarrow_{\mathcal{L}} \Delta}(\forall\text{-L})$$

$$\frac{\Gamma \Longrightarrow_{\mathcal{L}\cup\{c\}} \Delta, \phi[c/x] \qquad \text{Das Symbol } c \text{ kommt nicht in } \mathcal{L} \text{ vor.}}{\Gamma \Longrightarrow_{\mathcal{L}} \Delta, \forall x.\phi}(\forall\text{-R})$$

$$\frac{\Gamma, \phi[c/x] \Longrightarrow_{\mathcal{L}\cup\{c\}} \Delta \qquad \text{Das Symbol } c \text{ kommt nicht in } \mathcal{L} \text{ vor.}}{\Gamma, \exists x.\phi \Longrightarrow_{\mathcal{L}} \Delta}(\exists\text{-L})$$

Abbildung 2.2: Beweisregeln für die Quantoren

Definition 34 (Herleitung in der Prädikatenlogik)
Die Regeln des Sequenzenkalküls für die Prädikatenlogik sind die Regeln des Sequenzenkalküls für die Aussagenlogik (entsprechend auf prädikatenlogische Sequenzen angepasst), sowie die vier Quantorenregeln aus Abb. **??**, die im folgenden motiviert werden. Die Begriffe Herleitung und Beweis sind wie in der Aussagenlogik definiert.

Instanzierung von Quantoren Wie beweist man typischerweise eine Existenzaussage $\exists x.\phi$? Nun, man gibt einen Zeugen t in Form eines geschlossenen Terms für die postulierte Existenz an und zeigt dementsprechend $\phi[t/x]$. Dies motiviert also die Beweisregel $\exists$-R aus Abb. **??** zur Einführung eines Existenzquantors auf der rechten Seite. Man kann sich fragen, ob ein solcher Zeuge immer in Form eines geschlossenen Terms vorliegen muss, oder ob nicht auch aus anderen Gründen die Existenzaussage $\exists x.\phi$ gelten sollte. Dies berührt Fragen der Vollständigkeit, die wir erst später behandeln.

Ähnlich verhält es sich mit dem Allquantor auf der linken Seite. Will man eine Hypothese der Form $\forall x.\phi$ einsetzen, so gibt man einen geschlossenen Term t an und darf dann $\phi[t/x]$ als weitere Hypothese hinzunehmen. Dies motiviert die Regel $\forall$-L.

Betrachten wir die Sequenz

$$\forall x.P(x) \vee Q(x), \forall x.P(x) \Rightarrow R(x, f(x)), \forall x.Q(x) \Rightarrow R(x, g(x)) \Longrightarrow_{\mathcal{L}} \exists y.R(c, y)$$

wobei $\mathcal{L}$ passend gewählt ist. Insbesondere ist c eine Konstante (nullstelliges Funktionssymbol).

Hier können wir die Regel $\exists$-R probieren, was auf

$$\forall x.P(x) \vee Q(x), \forall x.P(x) \Rightarrow R(x, f(x)), \forall x.Q(x) \Rightarrow R(x, g(x)) \Longrightarrow_{\mathcal{L}} R(c, t)$$

für einen geeignet zu wählenden Term t führt. Aber welcher Term soll das sein? Intuitiv sollte es ja $f(c)$ oder $g(c)$ sein, je nachdem, ob $P(c)$ oder $Q(c)$ gilt. Solche Fallunterscheidungen sind aber nicht Teil der Termsyntax. Die Lösung besteht darin, zunächst *nicht* die Regel $\exists$-R zu bringen, sondern vielmehr die Fallunterscheidung mit $\vee$-L vorzuziehen. Hierzu muss allerdings zunächst die allquantifizierte Hypothese $\forall x.P(x) \vee Q(x)$ durch

$$\mathcal{H}_1 \left\{ \dfrac{\dfrac{\dfrac{}{P(c), R(c,f(c)), \forall x.Q(x) \Rightarrow R(x,g(x)) \Longrightarrow_{\mathcal{L}} R(c,f(c))}\,\exists\text{-R}}{P(c), \forall x.Q(x) \Rightarrow R(x,g(x)) \Longrightarrow_{\mathcal{L}} \exists y.R(c,y), P(c) \quad P(c), R(c,f(c)), \forall x.Q(x) \Rightarrow R(x,g(x)) \Longrightarrow_{\mathcal{L}} \exists y.R(c,y)}\,\Rightarrow\text{-L}}{P(c), P(c) \Rightarrow R(c,f(c)), \forall x.Q(x) \Rightarrow R(x,g(x)) \Longrightarrow_{\mathcal{L}} \exists y.R(c,y)}\,\forall\text{-L}}{P(c), \forall x.P(x) \Rightarrow R(x,f(x)), \forall x.Q(x) \Rightarrow R(x,g(x)) \Longrightarrow_{\mathcal{L}} \exists y.R(c,y)} \right.$$

$$\mathcal{H}_2 \left\{ \dfrac{\dfrac{\dfrac{}{Q(c), \forall x.P(x) \Rightarrow R(x,f(x)), R(c,g(c)) \Longrightarrow_{\mathcal{L}} R(c,g(c))}\,\exists\text{-R}}{Q(c), \forall x.P(x) \Rightarrow R(x,f(x)) \Longrightarrow_{\mathcal{L}} \exists y.R(c,y), Q(c) \quad Q(c), \forall x.P(x) \Rightarrow R(x,f(x)), R(c,g(c)) \Longrightarrow_{\mathcal{L}} \exists y.R(c,y)}\,\Rightarrow\text{-L}}{Q(c), \forall x.P(x) \Rightarrow R(x,f(x)), Q(c) \Rightarrow R(c,g(c)), \Longrightarrow_{\mathcal{L}} \exists y.R(c,y)}\,\forall\text{-L}}{Q(c), \forall x.P(x) \Rightarrow R(x,f(x)), \forall x.Q(x) \Rightarrow R(x,g(x)) \Longrightarrow_{\mathcal{L}} \exists y.R(c,y)} \right.$$

$$\dfrac{\dfrac{\dfrac{\mathcal{H}_1}{P(c), \forall x.P(x) \Rightarrow R(x,f(x)), \forall x.Q(x) \Rightarrow R(x,g(x)) \Longrightarrow_{\mathcal{L}} \exists y.R(c,y)} \quad \dfrac{\mathcal{H}_2}{Q(c), \forall x.P(x) \Rightarrow R(x,f(x)), \forall x.Q(x) \Rightarrow R(x,g(x)) \Longrightarrow_{\mathcal{L}} \exists y.R(c,y)}}{P(c) \vee Q(c), \forall x.P(x) \Rightarrow R(x,f(x)), \forall x.Q(x) \Rightarrow R(x,g(x)) \Longrightarrow_{\mathcal{L}} \exists y.R(c,y)}\,\vee\text{-L}}{\forall x.P(x) \vee Q(x), \forall x.P(x) \Rightarrow R(x,f(x)), \forall x.Q(x) \Rightarrow R(x,g(x)) \Longrightarrow_{\mathcal{L}} \exists y.R(c,y)}\,\forall\text{-L}$$

Abbildung 2.3: Beweis einer prädikatenlogischen Sequenz

$\forall$-R mit $x = c$ instanziert werden. Der so entstehende Beweis ist in Abb.**??** angegeben. Die (Rückwärts-)Anwendung einer Regel $\exists$-R oder $\forall$-L bezeichnet man als Instanzierung eines Quantors. Die richtigen Instanzierungen zu finden, macht die Schwierigkeit prädikatenlogischer Beweise aus.

Feste aber beliebige Konstanten Als nächstes betrachten wir den Beweis einer allquantifizierten Aussage. Um $\forall x.\phi$ zu beweisen, sollte man $\phi[c/x]$ für festes aber beliebiges c herleiten. Dies wird formal durch die Regel $\forall$-R geleistet, wobei die "feste aber beliebige" Konstante c verschieden von den in $\mathcal{L}$ bereits vorhandenen Konstanten (nullstelligen Funktionssymbolen) zu wählen ist. Die Prämisse der Regel $\forall$-R verwendet also eine andere Sprache, in der diese frische Konstante hinzugefügt ist. Die dort verwendete Notation $\mathcal{L} \cup \{c\}$ bezeichnet die Sprache, die man erhält, indem man zu $\mathcal{L}$ das nullstellige Funktionssymbol c hinzufügt.

Möchte man beispielsweise die Sequenz

$$\forall x.P(x) \vee Q(x), \forall x.P(x) \Rightarrow R(x,f(x)), \forall x.Q(x) \Rightarrow R(x,g(x)) \Longrightarrow_{\mathcal{L}_0} \forall x.\exists y.R(x,y)$$

beweisen, wobei nunmehr $\mathcal{L}_0$ die Symbole P, Q, f, g mit entsprechenden Stelligkeiten enthält, so wählt man eine "frische" Konstante c, bildet die Sprache $\mathcal{L} := \mathcal{L}_0 \cup \{c\}$ und wird dann auf die bereits in Abb.**??** hergeleitete Sequenz geführt.

Schließlich hat man die Regel $\exists$-L zur *Verwendung* existentiell quantifizierter Hypothesen. Weiß man $\exists x.\phi$, so darf man $\phi[c/x]$ für festes aber beliebiges c verwenden. Diese "frische" Konstante c repräsentiert den vorhandenen, aber unbekannten Zeugen für die Existenzaussage. Außer dass der Zeuge eben die aufgeführte Eigenschaft hat, darf man keine weiteren Annahmen über ihn machen.

$$\dfrac{\dfrac{\dfrac{\dfrac{R(d,c) \Longrightarrow_{\mathcal{L}\cup\{c,d\}} R(d,c)}{\forall y.R(d,y) \Longrightarrow_{\mathcal{L}\cup\{c,d\}} R(d,c)}\,\forall\text{-L}}{\forall y.R(d,y) \Longrightarrow_{\mathcal{L}\cup\{c,d\}} \exists x.R(x,c)}\,\exists\text{-R}}{\exists x.\forall y.R(x,y) \Longrightarrow_{\mathcal{L}\cup\{c\}} \exists x.R(x,c)}\,\exists\text{-L}}{\exists x.\forall y.R(x,y) \Longrightarrow_{\mathcal{L}} \forall y.\exists x.R(x,y)}\,\forall\text{-R}$$

Abbildung 2.4: Beweis mit allen vier Quantorenregeln

Als Beispiel einer Sequenz, deren Beweis (siehe Abb. **??**) alle vier Quantorenregeln verwendet, nehmen wir

$$\exists x.\forall y.R(x,y) \Longrightarrow_{\mathcal{L}} \forall y.\exists x.R(x,y)$$

Beliebige Konstanten Manche allgemeingültigen Sequenzen können nur bewiesen werden, wenn die zugrundeliegende Sprache mindestens einen geschlossenen Term enthält. Letzteres ist genau dann der Fall, wenn es mindestens eine Konstante gibt. Ein banales Beispiel ist die Formel $existsx.\top$. Etwas interessanter ist $Q \vee \exists x.P(x) \Rightarrow \exists x.Q \vee P(x)$. Hier macht man nach $\Rightarrow$-R zunächst eine Fallunterscheidung mit $\vee$-L. Im ersten Fall wird der Existenzquantor mit einem beliebigen geschlossenen Term instanziert. Im zweiten Fall wird die existentielle Hypothese mit $\exists$-L geöffnet.

Diese Sequenzen sind auch nur dann allgemeingültig, wenn man, wie geschehen, voraussetzt, dass der Grundbereich $|\mathcal{A}|$ jeder Struktur $\mathcal{A}$ nichtleer sei. Auf der Ebene der Syntax wird das eben durch das Vorhandensein einer Konstanten repräsentiert.

Verwendung der Kontraktion Bisweilen möchte man eine allquantifizierte Hypothese mehrfach und mit verschiedenen Einsetzungen verwenden. In diesem Fall muss sie vor Einsatz der Quantorenregel durch Contr-L dupliziert werden. Als Beispiel betrachten wir die Sequenz

$$\forall x.P(x) \Rightarrow P(f(x)) \Longrightarrow_{\mathcal{L}} \forall x.P(x) \Rightarrow P(f(f(x)))$$

deren Beweis in Abb. **??** gegeben ist. Der dort ebenfalls gegebene Beweis der "*drinker's formula*" $\exists x.P(x) \Rightarrow \forall x.P(x)$ benutzt außerdem noch eine beliebige Konstante c.

Bemerkung zum Existenzquantor Die Regel $\exists$-R suggeriert, dass zum Beweis einer existentiell quantifizierten Formel ein Zeuge angegeben werden muss. Das ist aber nicht immer so. So ist die Sequenz $\neg\forall x.\neg\phi \Longrightarrow_{\mathcal{L}} \exists x.\phi$, weil allgemeingültig, herleitbar. Sie besagt, dass wenn die Annahme eines Gegenbeispiels widersprüchlich ist, dann auf die Existenz eines Beispiels geschlossen werden darf. Um sie herzuleiten, verwendet man zunächst $\neg$-L, was auf $\Longrightarrow_{\mathcal{L}} \forall x.\neg\phi, \exists x.\phi$ führt. Alsdann $\forall$-R gefolgt von $\neg$-R: $\phi[c] \Longrightarrow_{\mathcal{L}\cup\{c\}} \exists x.\phi$ und dann schließlich $\exists$-R.

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{P(c) \Rightarrow P(f(c)), P(f(c)) \Rightarrow P(f(f(c))), P(c) \Longrightarrow_{\mathcal{L}\cup\{c\}} P(f(f(c)))}\,\textit{Auss.logik}}{P(c) \Rightarrow P(f(c)), \forall x.P(x) \Rightarrow P(f(x)), P(c) \Longrightarrow_{\mathcal{L}\cup\{c\}} P(f(f(c)))}\,\forall\text{-L} + \textit{Perm.}}{\forall x.P(x) \Rightarrow P(f(x)), \forall x.P(x) \Rightarrow P(f(x)), P(c) \Longrightarrow_{\mathcal{L}\cup\{c\}} P(f(f(c)))}\,\forall\text{-L}}{\forall x.P(x) \Rightarrow P(f(x)), \forall x.P(x) \Rightarrow P(f(x)) \Longrightarrow_{\mathcal{L}} \forall x.P(x) \Rightarrow P(f(f(x)))}\,\forall\text{-R} + \Rightarrow\text{-R}}{\forall x.P(x) \Rightarrow P(f(x)) \Longrightarrow_{\mathcal{L}} \forall x.P(x) \Rightarrow P(f(f(x)))}\,\textsc{Contr}\text{-L}$$

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{P(d), P(c) \Longrightarrow_{\mathcal{L}\cup\{d\}} P(d), \forall x.P(x)}{P(c) \Longrightarrow_{\mathcal{L}\cup\{d\}} P(d), P(d) \Rightarrow \forall x.P(x)}\,\Rightarrow\text{-R}}{P(c) \Longrightarrow_{\mathcal{L}\cup\{d\}} P(d), \exists x.P(x) \Rightarrow \forall x.P(x)}\,\exists\text{-R}}{P(c) \Longrightarrow_{\mathcal{L}} \forall x.P(x), \exists x.P(x) \Rightarrow \forall x.P(x)}\,\forall\text{-R}}{\Longrightarrow_{\mathcal{L}} \exists x.P(x) \Rightarrow \forall x.P(x), \exists x.P(x) \Rightarrow \forall x.P(x)}\,\exists\text{-R} + \Rightarrow\text{-R}}{\Longrightarrow_{\mathcal{L}} \exists x.P(x) \Rightarrow \forall x.P(x)}\,\textsc{Contr}\text{-R}$$

Abbildung 2.5: Obligatorische Verwendung der Kontraktion

2.4 Prädikatenlogik in PVS

In PVS sind alle Quantoren und Variablen typisiert. D.h. man deklariert etwa

```
D : TYPE+
```

und kann dann eine Sprache z.B. wie folgt deklarieren:

```
P : [D -> boolean]
R : [D,D -> boolean]
f : [D->D]
c : D
```

Nunmehr ist P einstelliges Prädikatsymbol; R ist zweistelliges Prädikatsymbol, also Relationssymbol; f ist einstelliges Funktionssymbol und c ist nullstelliges Funktionssymbol, also Konstante.

Die Formel $(\forall x.R(x, f(x))) \Rightarrow \forall x.R(f(c), f(f(c)))$ schreibt sich in PVS dann so:

```
(FORALL (x:D): R(x,f(x))) IMPLIES FORALL (x:D):R(f(c),f(f(c)))
```

Man kann in PVS auch mehrere Typen deklarieren und entsprechend typisierte Funktions- und Prädikatsymbole einführen. Deklariert man etwa `D,E : TYPE+` und dann `f : [D,E -> E]`. So bezeichnet f eine zweistellige Funktion, die ein Argument vom Typ D und ein Argument vom Typ E nimmt und ein Ergebnis vom Typ E liefert. Man kann dann etwa schreiben:

```
FORALL (x:D): EXISTS (y:E): P(f(x,y))
```

wenn `P:[E->boolean]` deklariert wurde.

In der untypisierten Prädikatenlogik lassen sich Typen durch einstellige Prädikatensymbole repräsentieren, wobei Quantoren jeweils durch diese relativiert werden und für Funktionssymbole entsprechende Annahmen hinzugegeben werden, die die Erhaltung der Typen ausdrücken. Obige Formel würde man also unter Zuhilfenahme zweier Prädikatsymbole D, E wie folgt schreiben:

$$(\forall d.D(d) \Rightarrow \forall e.E(e) \Rightarrow E(f(d,e))) \Rightarrow (\forall x.D(x) \Rightarrow \exists y.E(y) \wedge P(f(x,y)))$$

Die instanzierenden Quantorenregeln $\exists$-R und $\forall$-L sind in PVS durch den Befehl `inst` repräsentiert, welcher als Argumente die Nummer der zu instanzierenden Formel, sowie den instanzierenden Term (in Anführungszeichen) nimmt. In der Situation

```
{-1} P(d)
  |-------
{1}  EXISTS (x:D): P(x)
```

führt also der Befehl `(inst 1 "d")` auf

```
{-1} P(d)
  |-------
{1}  P(d)
```

welches ein Axiom ist.

Die beiden anderen Quantorenregeln $\forall$-R und $\exists$-L sind durch den Befehl `skolem` repräsentiert. Dieser nimmt als Argument die Nummer der Formel und die frische Konstante. Ist diese bereits in Verwendung, so gibt es eine Fehlermeldung.

In der Situation

```
  |-------
{1}    FORALL(x:D):P(x) IMPLIES EXISTS(x:D): P(x)
```

führt `(skolem 1 "d")` auf

```
  |-------
{1}    P(d) IMPLIES EXISTS(x:D): P(x)
```

Der Befehl `M-x show-skolem-constants` zeigt alle durch Quantorenregeln eingeführten Konstanten.

Die Befehle `inst` und `skolem` haben eine Reihe von Varianten, siehe Dokumentation.

2.5 Korrektheit und Vollständigkeit

Wenn $\mathcal{A}$ eine Struktur über $\mathcal{L}$ ist und $S \equiv \Gamma \Longrightarrow_{\mathcal{L}} \Delta$ eine Sequenz über $\mathcal{L}$ ist, so sagen wir, $\mathcal{A}$ *erfüllt* die Sequenz S, wenn entweder $\mathcal{A} \not\models \phi$ für mindestens ein $\phi \in \Gamma$, oder aber $\mathcal{A} \models \psi$ für mindestens ein $\psi \in \Delta$. Intuitiv müssen in $\mathcal{A}$ die Antezedentien in Γ zusammen die Disjunktion der Sukzedentien in Δ implizieren. Man schreibt in diesem Fall $\mathcal{A} \models S$. Die Sequenz S ist allgemeingültig, wenn $\mathcal{A} \models S$ für alle Strukturen $\mathcal{A}$ über $\mathcal{L}$.

Satz 14 (Korrektheit des Sequenzenkalküls für die Prädikatenlogik)
Hat eine Sequenz S einen Beweis im Sequenzenkalkül, so ist sie allgemeingültig.

Beweis Wir zeigen dass jede Regel die Allgemeingültigkeit erhält, also dass falls die Prämissen einer Regel allgemeingültig sind, dies auch für die Konklusion der Regel zutrifft. Die Behauptung folgt dann wie im Falle der Aussagenlogik durch Induktion über die Größe des Beweises.

Bei den Regeln genübt es, die neu hinzugekommenen Quantorenregeln betrachten; die Korrektheit der aussagenlogischen Regeln überträgt sich sinngemäß auf den prädikatenlogischen Fall.

Wir beginnen mit der Regel $\exists$-R. Sei also die Sequenz $\Gamma \Longrightarrow_{\mathcal{L}} \Delta, \phi[t/x]$ allgemeingültig. Sei eine Struktur $\mathcal{A}$ vorgegeben mit $\mathcal{A} \models \bigwedge \Gamma$. Nach Annahme muss $\mathcal{A}$ dann entweder $\bigvee \Delta$ erfüllen, oder aber die Formel $\phi[t/x]$. In ersterem Fall erfüllt $\mathcal{A}$ auch $\bigvee \Delta \vee \exists x.\phi$ und wir sind fertig. Im zweiten Falle gilt $\mathtt{tt} = [\![\phi[t/x]]\!]_{\mathcal{A}} = [\![\phi]\!]_{\mathcal{A}}[x \mapsto [\![t]\!]_{\mathcal{A}}]$. Letztere Gleichung ist strenggenommen beweisbedüftig und heißt "Substitutionslemma". Wegen $[\![\phi]\!]_{\mathcal{A}}[x \mapsto v] = \mathtt{tt}$ mit $v = [\![t]\!]_{\mathcal{A}}$ hat man dann auch $\mathcal{A} \models \exists x.\phi$. Die Regel $\forall$-L verbleibt als Übung.

Betrachten wir nun die Regel $\forall$-R. Es sei $\Gamma \Longrightarrow_{\mathcal{L} \cup \{c\}} \Delta, \phi[c/x]$ allgemeingültig und c komme weder in $\mathcal{L}$ noch in Γ, Δ, noch in der Formel ϕ selbst vor (nur dann haben wir eine wohlgeformte Prämisse einer Instanz der Regel $\forall$-R. Sei $\mathcal{A}$ eine Struktur für $\mathcal{L}$, welche alle Formeln in Γ erfüllt und keine der Formeln in Δ erfüllt. Wir müssen zeigen, dass $\mathcal{A} \models \forall x.\phi$. Sei also $v \in |\mathcal{A}|$ beliebig vorgegeben. Wir betrachten die Struktur $\mathcal{A}'$ für $\mathcal{L} \cup \{c\}$, die genau wie $\mathcal{A}$ definiert ist und außerdem die neue Konstante c als v interpretiert. Also $|\mathcal{A}'| = |\mathcal{A}|$ und $c_{\mathcal{A}'} = v$ und $X_{\mathcal{A}'} = X_{\mathcal{A}}$, falls $X \neq c$. Für jede Formel ϕ über $\mathcal{L}$ gilt dann $\mathcal{A} \models \phi$ gdw. $\mathcal{A}' \models \phi$, sodass nach Voraussetzung $\mathtt{tt} = [\![\phi[c/x]]\!]_{\mathcal{A}'}$ gelten muss. Nun ist aber $[\![\phi[c/x]]\!]_{\mathcal{A}'} = [\![\phi]\!]_{\mathcal{A}'}[x \mapsto v] = [\![\phi]\!]_{\mathcal{A}}[x \mapsto v]$. Erstere Gleichung mit Substitutionslemma; die zweite folgt aus der Tatsache, dass c nicht in ϕ vorkommt. Damit ist gezeigt $[\![\phi]\!]_{\mathcal{A}}[x \mapsto v] = \mathtt{tt}$ w.z.b.w. Die Regel $\exists$-L verbleibt wiederum als Übung. ∎

Satz 15 (Vollständigkeit des Sequenzenkalküls für die Prädikatenlogik)
Sei $\mathcal{L}$ eine Sprache mit mindestens einem Konstantensymbol und sei S eine allgemeingültige Sequenz über $\mathcal{L}$. Dann existiert ein Beweis von S im Sequenzenkalkül.

Die Idee des Beweises ist ähnlich wie bei der Aussagenlogik: zu beliebiger Sequenz S konstruiert man durch systematische Rückwärtsanwendung der Regeln einen Beweis"versuch", welcher so angelegt ist, dass, falls S überhaupt einen Beweis hat, dieser auch so gefunden wird. Aus einem erfolglosen Beweisversuch lässt sich dann eine Struktur konstruieren, die S falsifiziert.

Ein gewisses Problem besteht darin, dass bei den instanzierenden Regeln ja ein passender Term geschickt geraten werden muss. Wie soll das in einem generischen Beweisversuch auf systematische Weise geschehen? Die Antwort besteht darin, dass man vor jeder Anwendung einer instanzierenden Regel $\forall$-L oder $\exists$-R die entsprechende Formel durch Kontraktion verdoppelt, sodass sie nach der Instanzierung noch vorhanden ist. Sodann legt man es so an, dass jede $\exists$-Formel auf der rechten Seite irgendwann und immer wieder mit jedem nur möglichen Term instanziert wird und gleiches für $\forall$-Formeln links. Trifft man in einem Zweig auf ein Axiom, so hört man dort natürlich auf.

Führt nun der Beweisversuch nicht auf einen Beweis, so wird er sich zumindest entlang eines Zweiges ins Unendliche fortsetzen. Aus diesem unendlichen Zweig kann dann das gewünschte Gegenmodell konstruiert werden.

Formal ersetzt man also die instanzierenden Regeln durch die folgenden Varianten:

$$\frac{\Gamma \Longrightarrow_{\mathcal{L}} \Delta, \exists x.\phi, \phi[t/x] \qquad t \text{ geschlossener Term über } \mathcal{L}}{\Gamma \Longrightarrow_{\mathcal{L}} \Delta, \exists x.\phi}(\exists\text{-R'})$$

$$\frac{\Gamma, \forall x.\phi, \phi[t/x] \Longrightarrow_{\mathcal{L}} \Delta \qquad t \text{ geschlossener Term über } \mathcal{L}}{\Gamma, \forall x.\phi \Longrightarrow_{\mathcal{L}} \Delta}(\forall\text{-L'})$$

Ist nun eine Sequenz S nicht beweisbar, so kann man durch systematische Rückwärtsanwendung der Regeln und schließliche Instanzierung mit allen Termen, auch denen, die erst später eingeführte Konstanten verwenden, einen unendlichen Beweisbaum erhalten, welcher insbesondere einen unendlichen Zweig $S_0, S_2, S_2, S_3, \ldots$ folgender Gestalt enthält:

1. $S_0 = S$
2. Keine der Sequenzen S_i ist ein Axiom.
3. S_{i+1} ist Prämisse einer Regel R_i mit S_i als Konklusion.
4. Die Regel R_i ist nur dann von der Form $\exists$-R' oder $\forall$-L', wenn keine andere Regel anwendbar ist, also wenn S_i links nur allquantifizierte und rechts nur existentiell quantifizierte Formeln enthält.
5. Hat S_i die Form $\Gamma_i \Longrightarrow_{\mathcal{L}_i} \Delta_i$ und kommt eine Formel $\forall x.\phi$ in Γ vor und ist t ein geschlossener Term über $\mathcal{L}_i$, so existiert $j \geq i$ derart dass $\phi[t/x]$ in S_j links vorkommt.
6. Hat S_i die Form $\Gamma_i \Longrightarrow_{\mathcal{L}_i} \Delta_i$ und kommt eine Formel $\exists x.\phi$ in Δ vor und ist t ein geschlossener Term über $\mathcal{L}_i$, so existiert $j \geq i$ derart dass $\phi[t/x]$ in S_j rechts vorkommt.

Wir konstruieren nun eine Struktur $\mathcal{A}$ über $\mathcal{L} = \mathcal{L}_0$ wie folgt. Die Grundmenge von $\mathcal{A}$ enthält alle geschlossenen Terme über den Sprachen $\mathcal{L}_i$. Beachte, dass die Sprachen der Sequenzen entlang des Zweiges nicht immer dieselben sind, sondern durch Einführung fester aber beliebiger Konstanten anwachsen.

Funktionssymbole einschl. Konstantensymbole in $\mathcal{L} = \mathcal{L}_0$, der Sprache von S interpretieren sich selbst, d.h. $f_{\mathcal{A}}(t_1, \ldots, t_k) = f(t_1, \ldots, t_k)$.

Ein k-stelliges Prädikatsymbol P interpretieren wir als die Menge aller k-Tupel $(t_1, \ldots, t_k)$, derart, dass die atomare Formel $P(t_1, \ldots, t_k)$ in irgendeiner Sequenz S_i links vorkommt. Man beachte, dass sie dann nicht auch noch rechts vorkommen kann, da ja atomare Formeln nie verschwinden und somit ein Axiom vorläge.

Durch Induktion über den Formelaufbau zeigen wir nun: Kommt ϕ in einer der Sequenzen S_i links vor (also $\phi \in \Gamma_i$, so folgt $\mathcal{A} \models \phi$. Kommt ϕ in einer der Sequenzen S_i rechts vor, so folgt $\mathcal{A} \not\models \phi$.

Fall ϕ atomar. Das folgt direkt aus der Interpretation der Prädikatsymbole, die ja extra so gewählt wurde.

Fall $\phi = \neg\phi_1$. Kommt ϕ links vor, so muss irgendwann die Regel $\neg$-L verwendet werden, es gibt also i mit $\phi \in \Gamma_i$ und $\phi_1 \in \Delta_{i+1}$. Nach Induktionsvoraussetzung gilt $\mathcal{A} \not\models \phi_1$, also $\mathcal{A} \models \neg\phi_1$. Der Fall, wo ϕ rechts vorkommt, ist analog.

Fall $\phi = \forall x.\phi_1$. Kommt ϕ rechts vor, so muss irgendwann die Regel $\forall$-R verwendet werden, es gibt also i mit $\phi \in \Delta_i$ und $\mathcal{L}_{i+1} = \mathcal{L} \cup \{c\}$ für frisches c und $\phi_1[c/x] \in \Delta_{i+1}$. Nach Induktionsvoraussetzung gilt dann $\mathcal{A} \not\models \phi_1[c/x]$, also $[\![\phi]\!]_{\mathcal{A}}[x{\mapsto}c] = \mathtt{ff}$. Beachte, dass $c \in |\mathcal{A}|$. Es folgt $\mathcal{A} \not\models \forall x.\phi$.

Kommt ϕ hingegen auf der rechten Seite vor, so sei $t \in |\mathcal{A}|$ beliebig vorgegeben. Irgendwann muss ϕ mit diesem Term t instanziert werden, d.h., es existiert i sodass $\phi \in \Gamma_i$ und $\phi[t/x] \in \Gamma_{i+1}$. Nach Induktionsvoraussetzung gilt dann $\mathcal{A} \models \phi[t/x]$, also $\mathtt{tt} = [\![\phi[t/x]]\!]_{\mathcal{A}} = [\![\phi]\!]_{\mathcal{A}}[x{\mapsto}[\![t]\!]_{\mathcal{A}} = [\![\phi]\!]_{\mathcal{A}}[x{\mapsto}t]$. Letztere Gleichung verwendet, dass Funktionssymbole und folglich Terme durch sich selbst interpretiert werden.

Da t beliebig war, folgt also $\mathcal{A} \models \forall x.\phi$.

Die restlichen Fälle verbleiben als Übung.

Nachdem nun $\mathcal{A}$ alle linksstehenden Formeln erfüllt und alle rechtsstehenden Formeln falsifiziert, falsifiziert $\mathcal{A}$ auch alle Sequenzen S_i, also insbesondere $S = S_0$. Somit war S gar nicht allgemeingültig und wir können schließen, dass für jede allgemeingültige Sequenz der generische Beweisversuch in einen Beweis münden muss.

Darüberhinaus können wir aufgrund der Konstruktion der Struktur $\mathcal{A}$ folgende Verstärkung ablesen:

Korollar 16
Hat eine Formel ϕ (also die Sequenz $\Longrightarrow_{\mathcal{L}} \phi$) keinen Beweis, so gibt es eine Struktur $\mathcal{A}$, derart, dass $|\mathcal{A}|$ höchstens abzählbar unendlich ist und $\mathcal{A} \not\models \phi$ gilt.

Kontrapositiv ausgedrückt ergibt sich daraus der Satz von Löwenheim und Skolem

Satz 17 (Löwenheim-Skolem)
Hat ϕ ein Modell, so hat ϕ auch ein abzählbares Modell.

Beweis Hat ϕ ein Modell, so kann $\neg\phi$ nicht beweisbar sein. Aufgrund des voranstehenden Korollars gibt es also eine höchstens abzählbare Struktur $\mathcal{A}$ mit $\mathcal{A} \not\models \neg\phi$. Also ist $\mathcal{A}$ Modell von ϕ. Sollte $\mathcal{A}$ ausnahmsweise endlich sein, so kann man es durch Hinzunahme neuer Konstantensymbole künstlich aufblähen. ∎

Man beachte, dass es Formeln der Prädikatenlogik gibt, die keine endlichen Modelle, wohl aber unendliche Modelle haben. Das einfachste Beispiel ist die Formel

$$(\forall x.\exists y.R(x,y)) \wedge (\forall x.\forall y.\forall z.R(x,y) \wedge R(y,z) \Rightarrow R(x,z)) \wedge \forall x.\neg R(x,x)$$

Sie besagt, dass R eine transitive Relation ist, derart, dass zu jedem x and y existiert mit $R(x,y)$. In jedem endlichen Modell findet sich dann ein R-Zyklus und somit wegen der Transitivität ein x mit $R(x,x)$.

Nach dem Satz von Löwenheim-Skolem kann es aber keine Formel geben, die keine abzählbar unendlichen Modelle hat, aber doch überabzählbare besitzt. Das heißt also auch, dass man keine Formel angeben kann, deren einziges Modell gerade die Menge der reellen Zahlen ist.

2.6 Pränex Normalform und Skolemisierung

Definition 35 (Äquivalenz)
Zwei Formeln ϕ, ψ sind äquivalent, wenn gilt $[\![\phi]\!]_{\mathcal{A}}\eta = [\![\psi]\!]_{\mathcal{A}}\eta$ für alle $\mathcal{A}$ und η.

Ersetzt man in einer Formel ϕ ein Vorkommen einer Teilformel durch eine äquivalente Formel, so resultiert daraus eine zu ϕ äquivalente Formel

Definition 36
Eine Formel ϕ ist in *Pränex Normalform*, wenn ϕ von der Form

$$Q_1 x_1 . Q_2 x_2 . \cdots Q_n x_n . \psi$$

ist, wobei die Q_i Quantoren sind (also $\forall$ oder $\exists$) und ψ keine Quantoren enthält.

Satz 18
Zu jeder Formel ϕ kann eine äquivalente Formel ϕ' angegeben (und effizient berechnet) werden, welche in Pränex Normalform ist.

Beweis Dies geschieht durch sukzessive Anwendung der folgenden Äquivalenzumformungen, deren Gültigkeit leicht zu sehen ist.

$$\begin{array}{lcl}
(\exists x.\phi) \wedge \psi & \iff & \exists x.(\phi \wedge \psi) \\
(\exists x.\phi) \vee \psi & \iff & \exists x.(\phi \vee \psi) \\
\neg(\exists x.\phi) & \iff & \forall x.\neg\phi \\
\neg(\forall x.\phi) & \iff & \exists x.\neg\phi \\
(\exists x.\phi) \Rightarrow \psi & \iff & \forall x.\phi \Rightarrow \psi \\
(\forall x.\phi) \Rightarrow \psi & \iff & \exists x.\phi \Rightarrow \psi \\
\phi \Rightarrow \exists x.\psi & \iff & \exists x.\phi \Rightarrow \psi \\
\phi \Rightarrow \forall x.\psi & \iff & \forall x.\phi \Rightarrow \psi
\end{array}$$

Durch geeignete Umbenennung ist hier sicherzustellen, dass eine möglicherweise freie Variable x nicht unbeabsichtigt in den Geltungsbereich des Quantors gerät. So ist natürlich $(\exists x.P(x)) \wedge Q(x)$ nicht äquivalent zu $\exists x.P(x) \wedge Q(x)$, wohl aber zu $\exists y.P(y) \wedge Q(x)$.

Als Beispiel betrachten wir die Formel

$$(\forall x.\exists y.R(x,y)) \wedge (\forall x.\forall y.\forall z.R(x,y) \wedge R(y,z) \Rightarrow R(x,z)) \wedge \forall x.\neg R(x,x)$$

Die folgenden Formeln sind äquivalente Pränex Normalformen (je nachdem in welcher Reihenfolge man die Quantoren nach vorne zieht):

$$\begin{array}{l}
\forall a.\exists b.\forall x.\forall y.\forall z.\forall u.R(a,b) \wedge (R(x,y) \wedge R(y,z) \Rightarrow R(x,z)) \wedge \neg R(u,u) \\
\forall x.\forall y.\forall z.\forall a.\exists b.\forall u.R(a,b) \wedge (R(x,y) \wedge R(y,z) \Rightarrow R(x,z)) \wedge \neg R(u,u) \\
\forall z.\forall u.\forall a.\exists b.\forall x.\forall y.R(a,b) \wedge (R(x,y) \wedge R(y,z) \Rightarrow R(x,z)) \wedge \neg R(u,u) \\
\ldots
\end{array}$$

Man beachte aber, dass die Quantoren $\forall a$ und $\exists b.$ nicht miteinander vertauscht werden können. Man beachte in diesem Zusammenhang auch, dass die folgende Formel allgemeingültig ist:

$$(\forall x.\exists y.P(x) \wedge Q(y)) \Rightarrow \exists y.\forall x.P(x) \wedge Q(y)$$

Definition 37
Zwei geschlossene Formeln ϕ und ψ heißen erfüllungsäquivalent, wenn gilt: ϕ ist erfüllbar, genau dann, wenn ψ erfüllbar ist.

So sind zum Beispiel $\forall x.\exists y.\phi$ und $\forall x.\phi[f(x)/y]$ erfüllungsäquivalent, wenn f ein Funktionssymbol ist, welches nicht in ϕ vorkommt. Ist nämlich $\mathcal{A}$ ein Modell für $\forall x.\exists y.\phi$, so bedeutet das ja, dass für jeden Wert $u \in |\mathcal{A}$ ein Wert $v \in |\mathcal{A}|$ existiert, derart, dass $[\![\phi]\!]_{\mathcal{A}}[x{\mapsto}u, y{\mapsto}v] = \mathtt{tt}$. Das aber bedeutet, dass für jedes u die Menge $V_u = \{v \mid [\![\phi]\!]_{\mathcal{A}}[x{\mapsto}u, y{\mapsto}v] = \mathtt{tt}\}$ nicht leer ist. Man kann nun $\mathcal{A}$ zu einem Modell $\mathcal{A}'$ von $\forall x.\phi[f(x)/y]$ erweitern, indem man für jedes u den Funktionswert $f_{\mathcal{A}'}(u)$ so festlegt, dass $f_{\mathcal{A}'}(u) \in V_u$. Umgekehrt liefert $\mathcal{A}'_f$ jeweils die gewünschten Zeugen für die Existenzaussage.

Dieses Beispiel lässt sich wie folgt verallgemeinern.

Satz 19
Zu jeder geschlossenen Formel ϕ existiert eine erfüllungsäquivalente Formel von der Form

$$\forall x_1.\cdots\forall x_n.\psi$$

wobei ψ keine Quantoren enthält.

Beweis Zunächst bringt man ϕ auf Pränex-Normalform

$$Q_1x_1.Q_2x_2.\cdots Q_nx_n.\psi$$

Ist nun $Q_i = \exists$, so führe man ein neues Funktionssymbol f ein, dessen Stelligkeit gerade der Zahl der Allquantoren vor Q_i entspricht. Sei $\vec{x}$ der Vektor bestehend aus denjenigen x_j, sodass $j < i$ und $Q_j = \forall$. Nun streicht man den Existenzquantor Q_i und ersetzt in ψ die Variable x_i durch $f(\vec{x})$. Dies tut man für alle Existenzquantoren unter den Q_i.

Man sieht leicht, dass die so erhaltene Formel erfüllungsäquivalent ist: Hat man ein Modell für die ursprünglich gegebene Formel, so erweitert man auf ein Modell der neuen Formel, indem man die neu hinzugekommenen Funktionssymbole so interpretiert, dass die zugesicherte Eigenschaft wahr wird. Dass es so eine Funktion gibt, sagt ja gerade der Existenzquantor. Hat man umgekehrt ein Modell für die neue Formel, so kann man die Interpretation der neuen Funktionssymbole streichen und erhält so ein Modell für die alte Formel. ∎

Zu Ehren von Th. Skolem bezeichnet man die Elimination von Existenzquantoren durch neue Funktionssymbole als *Skolemisierung.*

Definition 38
Ein prädikatenlogisches Literal ist eine atomare Formel oder eine negierte atomare Formel. Eine prädikatenlogische Klausel ist eine Formel der Gestalt $\forall x_1.\forall x_2.\cdots\forall x_n \bigvee_{j=1}^{m} \phi_{ij}$, wobei die ϕ_{ij} prädikatenlogische Literale sind.

Satz 20
Jede Formel ist erfüllungsäquivalent zu einer Konjunktion prädikatenlogischer Klauseln.

Beweis Durch Skolemisierung stellt man zunächst eine allquantifizierte Formel her. Den quantorenfreien Rumpf dieser Formel bringt man dann mit den Regeln der Aussagenlogik auf KNF. Unter Verwendung der Äquivalenz $\forall x.\phi \wedge \phi \iff (\forall x.\phi) \wedge (\forall x.\psi)$ distribuiert man dann die Allquantoren auf die einzelnen Klauseln. ∎

Betrachten wir als Beispiel wieder die Formel

$$(\forall x.\exists y.R(x,y)) \wedge (\forall x.\forall y.\forall z.R(x,y) \wedge R(y,z) \Rightarrow R(x,z)) \wedge \forall x.\neg R(x,x)$$

Eine Pränex Normalform lautet

$$\forall a.\exists b.\forall x.\forall y.\forall z.\forall u.R(x,y) \wedge (R(x,y) \wedge R(y,z) \Rightarrow R(x,z)) \wedge \neg R(u,u)$$

Die Skolemisierung ergibt sich also zu

$$\forall a.\forall x.\forall y.\forall z.\forall u.R(a,f(a)) \wedge (R(x,y) \wedge R(y,z) \Rightarrow R(x,z)) \wedge \neg R(u,u)$$

Als Konjunktion von Klauseln geschrieben, wird das zu

$$\forall a.\forall x.\forall y.\forall z.\forall u.R(a,f(a)) \wedge (\neg(R(x,y) \vee \neg R(y,z) \vee R(x,z)) \wedge \neg R(u,u)$$

Es ist auch in der Prädikatenlogik üblich, Klauseln und KNFs als Mengen von Literalen bzw. Klauseln zu schreiben. Die Allquantoren lässt man dann weg. Unsere Beispielformel als Klauselmenge wird also zu

$$\{\{R(a,f(a))\},\{\neg R(x,y),\neg R(y,z),R(x,z)\},\{\neg R(u,u)\}\}$$

Ein Modell für diese Klauselmenge ist durch $|\mathcal{A}| = \mathbb{N}$ und $R_{\mathcal{A}} =$ "$<$" und $f_{\mathcal{A}}(x) = x+1$ gegeben.

Beachte, dass a, x, y, z, u keine Konstanten sondern allquantifizierte Variablen sind. Die entsprechenden Quantoren sind ja nur konventionshalber weggelassen worden.

2.7 Satz von Herbrand

Wir stellen nun eine Beziehung zwischen Prädikatenlogik und Aussagenlogik her, welche es uns erlaubt, die Kompaktheitseigenschaft und die Resolutionsmethode auf die Prädikatenlogik zu übertragen.

Sei Γ eine Menge prädikatenlogischer Formeln der Gestalt $\forall x_1.\cdots\forall x_n.\phi$ mit quantorenfreiem ϕ.

Wir nehmen an, dass die zugrundeliegende Sprache mindestens ein Konstantensymbol enthält (falls nicht, eines dazunehmen!). Wir betrachten nunmehr die Menge $\mathcal{H}(\Gamma)$ aller geschlossenen Formeln, die man erhält, indem man für jede Formel $\forall x_1.\cdots\forall x_n.\phi \in \Gamma$ die Menge aller Formeln $\phi[t_1/x_1,\ldots,t_n/x_n]$ für beliebige geschlossene Terme t_i über $\mathcal{L}$ hinzugibt.

Formal:

$$\mathcal{H}(\Gamma) = \{\phi[t_1/x_1, \ldots, t_n/x_n] \mid \forall \vec{x}.\phi \in \Gamma; t_1, \ldots, t_n \text{geschl. Terme}\}$$

Nehmen wir als Beispiel die Formelmenge

$$\Gamma = \{\forall a.R(a, f(a)), \forall x.\forall y.\forall z.R(x,y) \wedge R(y,z) \Rightarrow R(x,z), \forall u.\neg R(u,u)\}$$

wobei die Sprache neben R und f ein Konstantensymbol c enthalte. Wir erhalten:

$$\begin{aligned}\mathcal{H}(\Gamma) = \{\ & R(c, f(c)),\\ & R(c, f(c)) \wedge R(f(c), f(f(c))) \Rightarrow R(c, f(c)),\\ & \neg R(c,c), R(f(c), f(f(c))),\\ & R(f(c), f(f(c))) \wedge R(f(f(c)), f(f(f(c)))) \Rightarrow R(f(c), f(f(f(c)))),\\ & \neg R(f(c), f(c)), R(f(f(c)), f(f(f(c)))),\\ & R(c, f(f(c))) \wedge R(f(f(c)), f(f(f(c)))) \Rightarrow R(c, f(f(f(c)))),\\ & \neg R(f(f(c)), f(f(c))), \ldots\}\end{aligned}$$

Wir fassen jetzt $\mathcal{H}(\Gamma)$ als Menge *aussagenlogischer Formeln* auf, wobei wir die geschlossenen atomaren Formeln als aussagenlogische Variablen nehmen.

Falls eine Struktur $\mathcal{A}$ existiert, die alle Formeln in Γ erfüllt, wenn also Γ erfüllbar ist, dann ist $\mathcal{H}(\Gamma)$ auch als Menge aussagenlogischer Formeln erfüllbar: Man belegt eine "Variable" $P(t_1, \ldots, t_n)$ einfach mit $\eta(P(t_1, \ldots, t_n)) := [\![P(t_1, \ldots, t_n)]\!]_{\mathcal{A}}$. Eine Struktur, die die Formelmenge im Beispiel erfüllt ist ja gegeben durch die natürlichen Zahlen mit $R =$ "$<$" und $f(x) = x{+}1$. Die hinzugefügte Konstante kann man beliebig interpretieren, z.B. $c_{\mathcal{A}} = 0$. Die resultierende Belegung der aussagenlogischen Variablen ergibt sich dann zu $\eta(R(c, f(c))) = \mathtt{tt}$, $\eta(R(f(c), f(c))) = \mathtt{ff}$, $\eta(R(f(c), f(f(c)))) = \mathtt{tt}$, ...

Der Satz von Herbrand macht nun die umgekehrte Aussage: Ist $\mathcal{H}(\Gamma)$ im aussagenlogischen Sinne erfüllbar, so existiert eine Struktur $\mathcal{A}$, die alle Formeln in Γ erfüllt.

Satz 21
Sei Γ eine Menge allquantifizierter prädikatenlogischer Formeln, also von der Form $\forall x_1 \ldots . \forall x_n.\phi$. Die zugerundeliegende Sprache enthalte mindestens ein Konstantensymbol. Sei $\mathcal{H}(\Gamma)$ die Menge der geschlosssenen Instanzen der Formeln ϕ, wobei $\forall x_1 \ldots . \forall x_n.\phi$. Ist $\mathcal{H}(\Gamma)$ aufgefasst als aussagenlogische Formelmenge über den geschlossenen atomaren Formeln als Variablen erfüllbar, so existiert eine Struktur $\mathcal{A}$ derart dass $\mathcal{A} \models \forall x_1 \ldots . \forall x_n.\phi$ für alle Formeln $\forall x_1 \ldots . \forall x_n.\phi \in \Gamma$.

Beweis Es sei η eine Belegung, die alle aussagenlogischen Formeln in $\mathcal{H}(\Gamma)$ wahrmacht.

Wir definieren eine Struktur $\mathcal{A}$, das *Herbrand Modell*, wie folgt. Die Grundmenge $|\mathcal{A}|$ enthält alle geschlossenen Terme. Nach Annahme ist diese ja nicht leer. Funktionssymbole interpretieren sich selbst, also $f_{\mathcal{A}}(t) = f(t)$. Prädikatsymbole werden gemäß der aussagenlogischen Belegung η interpretiert, also

$$P_{\mathcal{A}} = \{(t_1, \ldots, t_n) \mid \eta(P(t_1, \ldots, t_n)) = \mathtt{tt}\}$$

Jetzt ist klar, dass $\mathcal{A}$ alle Formeln in Γ erfüllt, denn die einzigen infrage kommenden Werte für die allquantifizierten Variablen sind ja wiederum nur die geschlossenen Terme. ∎

Korollar 22 (Kompaktheit der Prädikatenlogik)
Sei Γ eine möglicherweise unendliche Menge prädikatenlogischer Formeln. Existiert zu jeder endlichen Teilmenge T von Γ eine Struktur, die alle Formeln in T erfüllt, so gibt es auch eine feste Struktur, die simultan alle Formeln in Γ erfüllt.

Beweis Indem wir die Formeln in Γ zunächst auf Pränex Normalform bringen und dann skolemisieren, dürfen wir annehmen, dass Γ nur allquantifizierte Formeln enthält. Nach Annahme ist nun die aussagenlogische Formelmenge $\mathcal{H}(\Gamma)$ konsistent und somit nach dem Kompaktheitssatz der Aussagenlogik auch erfüllbar. Nach dem Satz von Herbrand ist damit Γ im prädikatenlogischen Sinne erfüllbar. ∎

Mit dem Kompaktheitssatz kann man folgern, dass bestimmte Dinge nicht in Prädikatenlogik ausgedrückt werden können. Zum Beispiel gibt es keine Formelmenge Γ, geschweige denn eine einzige Formel ϕ, die ausdrückt, dass eine Relation R Äquivalenzrelation mit endlich vielen Klassen ist. Beachte, dass die folgende Formel ausdrückt, dass R Äquivalenzrelation ist:

$$(\forall x.R(x,x)) \wedge (\forall x.\forall y.R(x,y) \Rightarrow R(y,x)) \wedge (\forall x.\forall y.\forall z.R(x,y) \wedge R(y,z) \Rightarrow R(x,z))$$

Wäre nun Γ eine Formelmenge, die ausdrückt, dass eine Relation R Äquivalenzrelation mit endlich vielen Klassen ist. Nun sei ϕ_n eine Formel, die ausdrückt, dass R mindestens n Klassen hat. Zum Beispiel:

$$\phi_n = \exists x_1. \ldots \exists x_n. \bigwedge_{i,j \leq n; i \neq j} \neg R(x_i, x_j)$$

Jede endliche Teilmenge von $\Gamma \cup \{\phi_n \mid n \geq 0\}$ ist erfüllbar, denn in jeder solchen Teilmenge sind nur endlich viele der ϕ_n dabei. Eine Äquivalenzrelation mit ausreichend vielen Klassen liefert dann ein Modell. Gemäß dem Kompaktheitssatz müsste dann auch $\Gamma \cup \{\phi_n \mid n \geq 0\}$ simultan erfüllbar sein, was ein Widerspruch ist.

2.8 Resolution für die Prädikatenlogik

Wenn wir wissen wollen, ob eine gegebene prädikatenlogische Formel ϕ erfüllbar ist, so können wir sie skolemisieren und dann die aussagenlogische Formelmenge $\mathcal{H}(\phi)$ mithilfe der Resolutionsmethode für die Aussagenlogik auf Erfüllbarkeit untersuchen.

Natürlich wird man nicht $\mathcal{H}(\phi)$ vollständig aufschreiben, bevor man mit der Resolutionsmethode beginnt, sondern die Klauseln in $\mathcal{H}(\phi)$ erst bei Bedarf erzeugen.

Daraus ergibt sich die folgende Definition:

Definition 39
Sei $\mathcal{C}$ eine Menge von prädikatenlogischen Klauseln. Eine Herleitung einer Klausel C aus $\mathcal{C}$ ist eine Folge von Klauseln

$$C_1, C_2, C_3, \ldots, C_n$$

sodass $C_n = C$ und für jedes $i \leq n$ eine der folgenden drei Bedingungen erfüllt ist.

- $C_i \in \mathcal{C}$
- es gibt $k, k' < i$ sodass C_i Resolvente von C_k und $C_{k'}$ ist. D.h. $C_k = D \cup \{\ell\}$ und $C_{k'} = D' \cup \{\neg\ell\}$ und $C_i = D \cup D'$
- es gibt $k < i$ und Terme $t_1, \ldots, t_m$ (möglicherweise mit freien Variablen), sodass $C_i = C_k[t_1/x_1, \ldots, t_m/x_m]$.

Existiert eine Herleitung von C aus $\mathcal{C}$, so ist C aus $\mathcal{C}$ herleitbar.

Satz 23
Ist $\mathcal{A}$ eine Struktur, die alle Klauseln in $\mathcal{C}$ erfüllt (beachte, dass Variablen in Klauseln als allquantifiziert verstanden werden), so erfüllt $\mathcal{A}$ auch jede Klausel, die aus $\mathcal{C}$ herleitbar ist.

Beweis Man rechnet leicht nach, dass dies für jeden der drei Fälle gilt.
Dem zweiten Falle liegt die offensichtlich allgemeingültige Formel

$$(\forall x_1 \ldots \forall x_n.\phi) \Rightarrow \forall y_1 \ldots \forall y_m.\phi[t_1/x_1, \ldots t_n/x_n]$$

zugrunde, wobei die y_i die freien Variablen der Terme t_i sind. ∎

Sei zum Beispiel

$$\mathcal{C} = \{\{R(a, f(a))\}, \{\neg R(x, y), \neg R(y, z), R(x, z)\}, \{\neg R(u, u)\}\}$$

Hier ist eine Herleitung von $R(c, f(f(c)))$ aus $\mathcal{C}$:

$$\begin{array}{l}
\{R(a, f(a))\} \\
\{R(x, f(x))\} \\
\{R(f(x), f(f(x))\} \\
\{\neg R(x, f(x)), \neg R(f(x), f(f(x))), R(x, f(f(x)))\} \\
\{\neg R(f(x), f(f(x))), R(x, f(f(x)))\} \\
\{R(x, f(f(x)))\} \\
\{R(c, f(f(c)))\}
\end{array}$$

Satz 24
Aus einer Klauselmenge $\mathcal{C}$ ist die leere Klausel herleitbar genau dann wenn $\mathcal{C}$ unerfüllbar ist.

Beweis Die Klauselmenge besteht aus universell quantifizierten Formeln. Nach dem Satz von Herbrand muss also die assoziierte Menge aussagenlogischer Klauseln $\mathcal{H}(\mathcal{C})$, welche alle geschlossenen Instanzen der Klauseln umfasst, im Sinne der Aussagenlogik unerfüllbar sein. Deshalb ist mithilfe der aussagenlogischen Resolutionsmethode die leere Klausel herleitbar. Man erhält dann eine Herleitung der leeren Klausel, indem man zunächst die in dieser Herleitung benutzten geschlossenen Instanzen erzeugt und dann die aussagenlogischen Resolutionsschritte nachspielt. ∎

Hier ist ein Beispiel einer Herleitung der leeren Klausel. Wir betrachten die Formel

$$(\forall x.\forall y.\forall z.R(x,y)\wedge R(y,z) \Rightarrow R(x,z))\wedge(\forall x,\forall y.R(x,y) \Rightarrow R(y,x))\wedge\exists x.\exists y.R(x,y)\wedge\neg R(x,x)$$

In Klauselform und nach der Skolemisierung wird das zu

$$\begin{array}{l}\{\{\neg R(x,y),\neg R(y,z),R(x,z)\},\\ \{\neg R(x,y),R(y,x)\},\\ \{R(c,d)\},\\ \{\neg R(c,c)\}\end{array}$$

Jetzt also die Herleitung:

$$\begin{array}{l}\{\neg R(x,y),\neg R(y,z),R(x,z)\}\\ \{\neg R(c,d),\neg R(d,c),R(c,c)\}\\ \{\neg R(x,y),R(y,x)\}\\ \{\neg R(c,d),R(d,c)\}\\ \{\neg R(c,d),R(d,c)\}\\ \{\neg R(c,d),R(c,c)\}\\ \{R(c,d)\}\\ \{R(c,c)\}\\ \{\neg R(c,c)\}\\ \{\}\end{array}$$

2.8.1 Unifikation

In der Praxis instanziert man Klauseln immer nur unmittelbar bevor man mit ihnen resolviert und dann auch nur gerade soweit, dass die zu resolvierenden Literale komplementär sind.

Das heißt, man sucht sich zunächst zwei Klauseln heraus, sodass die eine ein Literal der Form $P(u_1,\ldots,u_n)$ enthält und die andere ein Literal $\neg P(t_1,\ldots,t_n)$. Sollte darüberhinaus gelten $u_i = t_i$ für alle i, so kann man sofort resolvieren. Ist das nicht der Fall, so versucht man, zwei Instanzierungen $\vec{v}$ und $\vec{s}$ zu finden, sodass $u_i[\vec{v}/\vec{x}] = t_i[\vec{s}/\vec{y}]$ gilt. Darüberhinaus sollten diese Substitutionen so allgemein wie möglich sein.

Sind beispielsweise die Klauseln

$$\begin{array}{l}C_1 = \{P(f(x,g(y)),g(z),z)\}\cup C_1'\\ C_2 = \{\neg P(x,g(x),y)\cup C_2'\end{array}$$

gegeben und man möchte die beiden Literale mit P gegeneinander resolvieren, so wird man in C_2 für die Variable x einen Term der Form $f(\cdot,\cdot)$ einsetzen müssen. Damit wird also das erste Literal von C_2 zu $\neg P(f(\cdot,\cdot),g(f(\cdot,\cdot)),y)$. Dann aber muss auch in C_1 für z dieser selbe Term eingesetzt werden, damit die zweiten Argumente übereinstimmen und über die dritte Argumentposition wird das dann wieder nach C_2 reflektiert. Die allgemeinstmögliche Substitution, die die beiden Literale komplementär macht, ergibt sich also zu

$$\begin{array}{l}\{P(f(a,g(b)),g(f(a,g(b))),f(a,g(b)))\}\cup C_1'[a/x,b/y,f(a,g(b))/z]\\ \{\neg P(f(a,g(b)),g(f(a,g(b))),f(a,g(b)))\}\cup C_2'[f(a,g(b))/x,f(a,g(b))/y]\end{array}$$

Hier sind a, b wohlgemerkt Variablen, die wiederum in jeder der Klauseln separat allquantifiziert sind. Dass sie in C_1 und C_2 gleich heißen, ist "Zufall".

Nach einem Resolutionsschritt ergibt sich dann

$$C_1'[a/x, b/y, f(a, g(b))/z] \cup C_2'[f(a, g(b))/x, f(a, g(b))/y]$$

Die beiden Substitutionen $\sigma_1 := [a/x, b/y, f(a, g(b))/z]$ und $\sigma_2 := [f(a, g(b))/x, f(a, g(b))/y]$ bilden den *allgemeinsten Unifikator* der beiden atomaren Formeln $P(f(x, g(y)), g(z), z)$ und $P(x, g(x), y)$. Es gilt

$$P(f(x, g(y)), g(z), z)\sigma_1 = P(x, g(x), y)\sigma_2$$

und jedes andere Paar von Substitutionen, welches die beiden Formeln *unifiziert* entsteht als Instanz von (σ_1, σ_2), also durch Einsetzen spezieller Terme für die Parametervariablen a und b.

Man beachte, dass die Substitutionen für die beiden Klauseln jeweils unterschiedlich gewählt werden dürfen, selbst wenn die zu substituierenden Variablen (zufälligerweise) gleich lauten. Sie sind ja in jeder Klausel separat allquantifiziert.

Wir verzichten darauf, den allgemeinsten Unifikator formal einzuführen und verweisen auch für den Algorithmus zu seiner Berechnung auf die Literatur.

2.8.2 Entscheidbarkeit

Im Gegensatz zur Aussagenlogik bilden weder der Sequenzenkalkül, noch das Resolutionsverfahren Entscheidungsverfahren für die Gültigkeit, bzw. Unerfüllbarkeit prädikatenlogischer Formeln. Bei der Aussagenlogik waren irgendwann alle nur denkbaren Klauseln erzeugt, bzw. alle Möglichkeiten der Rückwärtsanwendung von Beweisregeln erschöpft. In der Prädikatenlogik gibt es außer in den allereinfachsten Fällen immer unendlich viele mögliche Klauseln (schon in unseren Beispielen war das der Fall) und so kann man zwar hoffen, irgendwann auf die leere Klausel zu stoßen, bzw. eine Herleitung im Sequenzenkalkül zu finden, und dabei verschiedene heuristische Hilfsmittel einsetzen; findet man die leere Klausel aber nicht, so hat man im allgemeinen keinen Grund zur Annahme, dass die gegebene Klauselmenge erfüllbar sei. Manchmal kann man die gewonnene Information nutzen, um ein endliches Modell zu konstruieren, aber es gibt ja, wie wir gesehen haben, auch Formeln, die nur unendliche Modelle haben.

In der Tat hat bereits Gödel gezeigt, dass Allgemeingültigkeit in der Prädikatenlogik unentscheidbar ist, d.h., dass prinzipiell kein Algorithmus existieren kann, der zu jeder beliebigen Formel nach endlicher Rechenzeit feststellt, ob sie allgemeingültig ist, oder nicht.

2.8.3 Implementierungen

Der prädikatenlogische Resolutionskalkül ist in zahlreichen Werkzeugen implementiert, welche zu einer gegebenen prädikatenlogischen Formel oder Klauselmenge vollautomatisch nach Resolutionsbeweisen suchen. SPASS (spass.mpi-sb.mpg.de), Otter, Waldmeister sind bekannte solche Werkzeuge, die in vielen Beispielen schön funktionieren, aber

wegen der Unentscheidbarkeit in anderen Fällen nicht terminieren, bzw. ohne Antwort abbrechen.

Hier ist eine SPASS Eingabedatei für die Komposition stetiger Funktionen (reelle Zahlen und Umgebungen sind hier abstrakt gegeben):

```
begin_problem(Cont).
list_of_descriptions.
name({*Cont*}).
author({*MH*}).
status(unknown).
description({*klik*}).
end_of_list.

list_of_symbols.
        functions[(f,1)].
        predicates[(in,2)].
        sorts[real,interval].
end_of_list.

list_of_declarations.
forall([real(x)],real(f(x))).
predicate(in,real,interval).
end_of_list.

list_of_formulae(axioms).
formula(forall([real(x)],forall([interval(U)],implies(in(f(x),U),
        exists([interval(V)],and(in(x,V),forall([real(y)],implies(in(y,V),in(f(y),U)))))))).
end_of_list.
list_of_formulae(conjectures).
formula(forall([real(x)],forall([interval(U)],implies(in(f(f(x)),U),
        exists([interval(V)],and(in(x,V),forall([real(y)],implies(in(y,V),in(f(f(y)),U))))))))).
end_of_list.
end_problem.
```

SPASS findet in weniger als einer Sekunde einen Resolutionsbeweis für die negierte Formel, so dass also die Formel selbst bewiesen ist. Auf der anderen Seite führt folgende SPASS-Version unseres *running example* zur Nichttermination:

```
begin_problem(Infinite).
list_of_descriptions.
name({*Infinite*}).
author({*MH*}).
status(satisfiable).
description({**}).
end_of_list.

list_of_symbols.
        predicates[(rel,2)].
end_of_list.

list_of_formulae(axioms).
formula(forall([x,y,z],implies(and(rel(x,y),rel(y,z)),rel(x,z)))).
formula(forall([x],exists([y],rel(x,y)))).
end_of_list.
list_of_formulae(conjectures).
formula(exists([x],rel(x,x))).
end_of_list.
end_problem.
```

Die entsprechenden WWW-Seiten bieten weitere Beispiele und Tutorials.

3 Gleichheit

In diesem Kapitel erweitern wir die Prädikatenlogik um die Gleichheit. Damit kommen wir dem allgemeinen mathematischen Sprachgebrauch schon sehr nahe. Insbesondere können wir damit algebraische Strukturen wie Ringe, Körper, Gruppen formalisieren. Im Prinzip ist es möglich, die Gleichheit durch Hinzunahme eines besonderen zweistelligen Prädikatsymbols, welches geeignet axiomatisiert wird. Diese Axiome, in Form von jeweils hinzuzunehmenden Formeln, setzen fest, dass die Gleichheit reflexiv, symmetrisch und transitiv ist und mit allen Komponenten der Sprache (Funktions- und Prädikatsymbolen) verträglich ist.

Auf der Ebene der Strukturen kann dann aber die Gleichheit auch als mit der Interpretation aller Symbole verträgliche Äquivalenzrelation interpretiert werden, was nicht wirklich der Intention entspricht. Daher werden wir die Gleichheit als neues logisches Symbol hinzunehmen, also mit demselben Status wie die Quantoren und die Junktoren und Syntax, Semantik, Beweisregeln, etc. entsprechend erweitern.

3.1 Syntax der Prädikatenlogik mit Gleichheit

Sprachen und Terme bleiben unverändert. Zu den Formeln nehmen wir Gleichungen hinzu:

Definition 40 (Formeln mit Gleichheit)
Sei $\mathcal{L}$ eine prädikatenlogische Sprache. Formeln mit Gleichheit über $\mathcal{L}$ sind entweder Formeln im Sinne von Definition **??** oder aber Gleichungen der Form $t_1{=}t_2$, wobei t_1 und t_2 Terme.

Ab jetzt bezeichnen wir mit "Formeln" immer die Formeln mit Gleichheit.

Ist etwa $\mathcal{L}_{\text{mon}}$ die Sprache mit einem zweistelligen Funktionssymbol $\circ$ (infixnotiert) und einer Konstanten e, so gibt es z.B. folgende Formeln:

$\forall x.\forall y.\forall z.(x \circ y) \circ z = x \circ (y \circ z)$	(Ass)
$\forall x.e \circ x = x$	(Neutr-L)
$\forall x.x \circ e = x$	(Neutr-R)
$\exists y.y \circ x = e$	(Inv-L(x))
$\exists y.x \circ y = e$	(Inv-R(x))
$\forall x.\forall y.x \circ y = e \wedge \text{Inv-L}(x) \Rightarrow y \circ x = e$	(Inv-LR)

3.2 Semantik der Prädikatenlogik mit Gleichheit

Die Semantik der Prädikatenlogik wird auf Gleichheit dahingehend erweitert, dass Formeln $s{=}t$ als Gleichheit der Bedeutungen von s und t interpretiert werden. Strukturen

und Belegungen sind hierbei dieselben wie für die Prädikatenlogik ohne Gleichheit.

Definition 41
Ist ϕ eine Formel, $\mathcal{A}$ eine Struktur, η eine Belegung. Die Bedeutung $[\![\phi]\!]_{\mathcal{A}}\eta \in \mathbf{B}$ der Formel ϕ in der Struktur $\mathcal{A}$ und unter der Belegung η ist definiert durch die Klauseln aus Definition **??** zuzüglich der folgenden Klausel für die Gleichheit:

$$[\![s{=}t]\!]_{\mathcal{A}}\eta = \begin{cases} \mathtt{tt}, \text{ wenn } [\![s]\!]\eta = [\![t]\!]\eta, \\ \mathtt{ff}, \text{ wenn } [\![s]\!]\eta \neq [\![t]\!]\eta \end{cases}$$

Die abgeleiteten Begriffe wie "erfüllen einer Formel oder Formelmenge", "Allgemeingültigkeit", "Modell" verallgemeinern sich analog.

Wir setzen

$$\textsc{Mon} = \{\textsc{Ass}, \textsc{Neutr-L}, \textsc{Neutr-R}\}$$

Eine Modell der Formelmenge Mon, also eine Struktur, in der alle drei Formeln in Mon gelten, ist ein *Monoid*. D.h. eine Menge zusammen mit einer assoziativen binären Operation und einem neutralen Element.

Umgekehrt kann jedes Monoid als Modell der Formelmenge Mon aufgefasst werden.

Sei ein Monoid $\mathcal{M} = (M, \circ, e)$ vorgegeben und $m \in M$. Ein Element $n \in M$ ist Linksinverses von m, wenn gilt $n \circ m = e$. Das Element m heißt *linksinvertierbar*, wenn so ein n existiert. Solch ein Element y heißt dann *Linksinverses* von x.

Ein Element m ist also linksinvertierbar, wenn $[\![\textsc{Inv-L}(x)]\!]_{\mathcal{M}}[x{\mapsto}m] = \mathtt{tt}$. Man notiert dies auch als $\mathcal{M} \models \textsc{Inv-L}(m)$.

Analog definiert man auch die Rechtsinversen.

Ist ein Element m linksinvertierbar und ist n' Rechtsinverses zu m, also $m \circ n' = e$, so ist n' auch gleichzeitig Linksinverses und dieses ist eindeutig. Ist nämlich $n \circ m = e$, so folgt $(n \circ m) \circ n' = n'$. Die linke Seite ist aber gleich $n \circ (m \circ n') = n \circ e = n$, also ist $n = n'$ und insbesondere gilt $n' \circ m = e$.

Das bedeutet, dass jedes Monoid $\mathcal{M}$ Modell der Formel Inv-LR ist und auch, dass die Formel

$$\bigwedge \textsc{Mon} \Rightarrow \textsc{Inv-LR}$$

allgemeingültig ist, also von allen Strukturen der zugrundeliegenden Sprache erfüllt wird.

3.3 Sequenzenkalkül für die Prädikatenlogik mit Gleichheit

Wir erweitern den Sequenzenkalkül um die drei Beweisregeln in Abb. **??**. Die Regel Refl besagt, dass Formeln der Form $t{=}t$ wahr sind. Die Regel Subst-R erlaubt es, in Gegenwart einer Gleichung $s{=}t$ unter den Antezedentien ein Vorkommen von s in einem Sukzedens durch t zu ersetzen, also die linke Seite der Gleichung durch die rechte zu ersetzen. Die drei weiteren Subst-Regeln erlauben dies auch von rechts nach links und in den Antezedentien zu tun. Manchmal ist es nützlich, mithilfe der Schnittregel Cut aus Abschnitt **??**, die auch hier zulässig ist, Gleichungen zunächst zu verwenden, um sie dann anschließend zu beweisen.

$$\frac{t \text{ geschlossener Term über } \mathcal{L}}{\Gamma \Longrightarrow_{\mathcal{L}} \Delta, t{=}t}(\textsc{Refl}) \qquad \frac{\Gamma, s{=}t \Longrightarrow_{\mathcal{L}} \Delta, \phi[t/x]}{\Gamma, s{=}t \Longrightarrow_{\mathcal{L}} \Delta, \phi[s/x]}(\textsc{Subst-R})$$

$$\frac{\Gamma, s{=}t, \phi[t/x] \Longrightarrow_{\mathcal{L}} \Delta}{\Gamma, s{=}t, \phi[s/x] \Longrightarrow_{\mathcal{L}} \Delta}(\textsc{Subst-L}) \qquad \frac{\Gamma, s{=}t \Longrightarrow_{\mathcal{L}} \Delta, \phi[s/x]}{\Gamma, s{=}t \Longrightarrow_{\mathcal{L}} \Delta, \phi[t/x]}(\textsc{Subst-R-RL})$$

$$\frac{\Gamma, s{=}t, \phi[s/x] \Longrightarrow_{\mathcal{L}} \Delta}{\Gamma, s{=}t, \phi[t/x] \Longrightarrow_{\mathcal{L}} \Delta}(\textsc{Subst-L-RL})$$

Abbildung 3.1: Regeln für die Gleichheit

$$\dfrac{\dfrac{\dfrac{\dfrac{\textsc{Ass}, \textsc{Neutr-L}, \textsc{Neutr-R}, e \circ c = c, e \circ c = e \Longrightarrow_{\mathcal{L}\cup\{c\}} e \circ c = e}{\textsc{Ass}, \textsc{Neutr-L}, \textsc{Neutr-R}, e \circ c = c, e \circ c = e \Longrightarrow_{\mathcal{L}\cup\{c\}} c = e}\textsc{Subst-L-RL}}{\textsc{Ass}, \textsc{Neutr-L}, \textsc{Neutr-R}, e \circ c = e \Longrightarrow_{\mathcal{L}\cup\{c\}} c = e}}{\textsc{Ass}, \textsc{Neutr-L}, \textsc{Neutr-R}, \forall x.x \circ c = x \Longrightarrow_{\mathcal{L}\cup\{c\}} c = e}}{\textsc{Ass}, \textsc{Neutr-L}, \textsc{Neutr-R} \Longrightarrow_{\mathcal{L}} \forall y.(\forall x.x \circ y = x) \Rightarrow y = e}$$

$$\dfrac{\dfrac{\dfrac{}{\Gamma, s{=}t \Longrightarrow_{\mathcal{L}} t{=}t, \Delta}\textsc{Refl}}{\Gamma, s{=}t \Longrightarrow_{\mathcal{L}} t{=}s, \Delta}\textsc{Subst-L}}{\Gamma, s{=}t, t{=}u \Longrightarrow_{\mathcal{L}} t{=}u, \Delta}\textsc{Subst-L}$$

$$\frac{\Gamma, s{=}t, t{=}u \Longrightarrow_{\mathcal{L}} t{=}u, \Delta}{\Gamma, s{=}t, t{=}u \Longrightarrow_{\mathcal{L}} s{=}u, \Delta}\textsc{Subst-L}$$

Abbildung 3.2: Beispielherleitung mit Gleichheit

In Abb. **??** finden sich Beispielherleitungen. Diese Erweiterung des Sequenzenkalküls ist wiederum korrekt und vollständig:

Satz 25 (Korrektheit und Vollständigkeit des Sequenzenkalküls mit Gleichheit) *Eine Sequenz S der Prädikatenlogik mit Gleichheit ist genau dann im Sequenzenkalkül mit Gleichheit beweisbar, wenn sie allgemeingültig ist.*

Beweis (Beweis (Skizze, nicht prüfungsrelevant)) Die Korrektheit (beweisbare Sequenzen sind allgemeingültig) ergibt sich aus der offensichtlichen Korrektheit der neu hinzugenommenen Regeln.

Für die Vollständigkeit bildet man wie im Fall ohne Gleichheit einen generischen Beweisversuch für S. Dieser ist so anzulegen, dass anwendbare Rückwärtsanwendungen der Subst-Regeln getätigt werden und dies jeweils nach vorheriger Anwendung von Contr, ähnlich wie bei den Quantorenregeln.

Aus einem unendlichen Zweig $S = S_0, S_1, S_2, \ldots$ über Sprachen $\mathcal{L} = \mathcal{L}_0 \subseteq \mathcal{L}_1 \subseteq \mathcal{L}_2 \ldots$ konstruiert man dann wiederum ein Gegenmodell, indem man zunächst die Struktur $\mathcal{A}$, deren Grundmenge die geschlossenen Terme über den Sprachen $\mathcal{L}_i$ sind, bildet. Damit diese nun alle Antezedentien auf dem unendlichen Zweig einschließlich der dort vorhanden Gleichheiten erfüllt, muss diese noch nach der von den linksstehenden Gleichungen erzeugte Kongruenzrelation faktorisiert werden. "Linksstehende Gleichungen" sind die Gleichungen $s{=}t$, welche Antezedens einer der Sequenzen S_i sind. "Erzeugte Kongruenzrelation" ist die kleinste mit den Funktionssymbolen verträgliche Äquivalenzrelation, die alle linksstehenden Gleichungen enthält. Man muss jetzt zeigen, dass die Interpretation der Prädikatsymbole mit dieser Faktorisierung verträglich ist. Dies sieht man leicht mithilfe der Subst-Regeln, die ja systematisch immer wieder angewandt werden. Des-

weiteren muss gezeigt werden, dass alle rechtsstehenden Gleichungen falsifiziert werden. Hierzu zeigt man, dass wenn $u \sim v$, dann existiert eine endliche Teilmenge Γ der linksstehenden Gleichungen, sodass $\Gamma \Longrightarrow_{\mathcal{L}_i} u{=}v$ beweisbar ist, wobei i hinreichend groß gewählt ist. Die Details verbleiben als Übung. ∎

3.4 Gleichheit in PVS

Die Regel Refl wird in PVS wie ein Axiom behandelt: kommt eine Gleichung $t{=}t$ in einer Sequenz rechts vor, so ist der entsprechende Zweig fertiggestellt.

Für die Regeln Subst-R or Subst-L verwendet man den Befehl

$$(\texttt{replace}\,\langle womit\rangle\,\langle wo\rangle)$$

Hier ist $\langle womit\rangle$ die (negative) Nummer einer Gleichung unter den Antezedentien; $\langle wo\rangle$ ist die Nummer eines Antezedens (entsprechend Subst-R) oder eines Sukzedens (entsprechend Subst-L). Um eine der Regeln Subst-L-RL oder Subst-R-RL anzuwenden, benutzt man

$$(\texttt{replace}\,\langle womit\rangle\,\langle wo\rangle : \texttt{dir RL})$$

3.4.1 Größeres Beispiel: Monoide

Das folgende größere Beispiel ist dem Skript "Rechnergestütztes Beweisen" entnommen und daher auf Englisch.

We assume a set M̈ with an associative operation * in infix notation.

```
M : TYPE+
* : [M,M->M]
assoc : AXIOM
   FORALL(x,y,z:M):(x*y)*z = x*(y*z)
```

Of course, all this must be placed in a .pvs file and within something like monoids : THEORY BEGIN... END.

Generalised associativity We want to prove an extended law of associativity:

```
assoc4 : THEOREM
    FORALL(x,y,z,w:M): ((x*y)*z)*w = x*(y*(z*w))
```

We will first do it the basic way and then using some more advanced commands.

After starting the prover we introduce constants for the universally quantified variables with the command (skolem!):

```
  |-------
{1}   ((x!1 * y!1) * z!1) * w!1 = x!1 * (y!1 * (z!1 * w!1))
```

where x!1, y!1, z!1, w!1 are fresh constants of type M.

Recall, that in this case (skolem!) is equivalent to `(skolem 1 (\x!1\ \y!1\ \z!1\ \w!1\))` which in turn is equivalent to (skolem 1 "x!1") followed by (skolem 1 "y!1") followed by (skolem 1 "z!1") followed by (skolem 1 "w!1").

We first want to rewrite the subterm ((x!1 * y!1) * z!1) using assoc. To that end we add assoc to our antecedents with (lemma "assoc").

```
{-1}  FORALL (x, y, z: M): (x * y) * z = x * (y * z)
  |-------
[1]   ((x!1 * y!1) * z!1) * w!1 = x!1 * (y!1 * (z!1 * w!1))
```

Normally, the lemma command applies to something already proved, a "lemma". In that case it corresponds to the CUT rule. With axioms it's a bit different. We can think of them as being implicitly added to the antecedents. In that case the lemma command simply highlights them. It is also possible to extend the sequent calculus by real axioms.

At any rate, we will need our axiom more than once, so we start by copying it corresponding to CONTR: copy -1.

```
{-1}  FORALL (x, y, z: M): (x * y) * z = x * (y * z)
[-2]  FORALL (x, y, z: M): (x * y) * z = x * (y * z)
  |-------
[1]   ((x!1 * y!1) * z!1) * w!1 = x!1 * (y!1 * (z!1 * w!1))
```

The -1 formula must now be instantiated with the inst command: (inst -1 "x!1" "y!1" "z!1").

```
{-1}  (x!1 * y!1) * z!1 = x!1 * (y!1 * z!1)
[-2]  FORALL (x, y, z: M): (x * y) * z = x * (y * z)
  |-------
[1]   ((x!1 * y!1) * z!1) * w!1 = x!1 * (y!1 * (z!1 * w!1))
```

Now we can use an equality rule: (replace -1 1):

```
[-1]  (x!1 * y!1) * z!1 = x!1 * (y!1 * z!1)
[-2]  FORALL (x, y, z: M): (x * y) * z = x * (y * z)
  |-------
{1}   (x!1 * (y!1 * z!1)) * w!1 = x!1 * (y!1 * (z!1 * w!1))
```

The parentheses around the just replaced subterm are not displayed which is irritating. Next, we must apply associativity to the whole left hand side, the middle term being this time not just a constant, but y!1 * z!1. This time we use a slightly more powerful command: inst-cp which works like inst but copies the formula to be instantiated beforehand. So,

```
(inst-cp -2 "x!1" "y!1*z!1" "w!1")
```

brings us to

```
-1]  (x!1 * y!1) * z!1 = x!1 * (y!1 * z!1)
[-2]  FORALL (x, y, z: M): (x * y) * z = x * (y * z)
{-3}  (x!1 * (y!1 * z!1)) * w!1 = x!1 * (y!1 * z!1 * w!1)
  |-------
[1]   x!1 * (y!1 * z!1) * w!1 = x!1 * (y!1 * (z!1 * w!1))
```

Now (replace -3 1) gives

```
[-1]  (x!1 * y!1) * z!1 = x!1 * (y!1 * z!1)
[-2]  FORALL (x, y, z: M): (x * y) * z = x * (y * z)
[-3]  (x!1 * (y!1 * z!1)) * w!1 = x!1 * (y!1 * z!1 * w!1)
  |-------
{1}   x!1 * ((y!1 * z!1) * w!1) = x!1 * (y!1 * (z!1 * w!1))
```

where again, I've inserted some parens. This is getting close; instantiating the remaining copy of associativity with

```
(inst -2 "y!1" "z!1" "w!1")
```

followed by (replace -2 1) completes the proof.

High-level proof Now let's do the same proof again with more powerful commands: After (skolem!) we get as before

```
  |-------
{1}   ((x!1 * y!1) * z!1) * w!1 = x!1 * (y!1 * (z!1 * w!1))
```

Now rather than bringing in assoc, instantiating, and then rewriting our goal with it, we can use the command rewrite-lemma (p. 65 of `prover-guide.ps`) which does these two steps in one go:

```
(rewrite-lemma "assoc" ("x" "x!1" "y" "y!1" "z" "z!1"))
```

results in

```
Rewriting using assoc where
   x gets x!1,
   y gets y!1,
   z gets z!1,
this simplifies to:
assoc4 :

  |-------
{1}   x!1 * (y!1 * z!1) * w!1 = x!1 * (y!1 * (z!1 * w!1))
```

The command rewrite-lemma takes as second argument a substitution which is an even length list providing the required values for all the bound variables in the lemma. After perfoming this instantiation it must become an equation with which rewriting then

takes place. Admittedly, this syntax is somewhat inconsistent with the syntax of the (inst) command.

Now, we want to perform the same procedure again, but with a different substitution:

```
(rewrite-lemma "assoc" ("x" "x!1" "y" "y!1 * z!1" "z" "w!1"))
```

Fortunately, we don't need to type in again from scratch: the keystroke M-p, that is the Alt key and the P key together, brings up the last command entered. We only need to edit the substitution. Further M-p bring up even earlier commands. If we've gone too far, we can use M-n to go back again. There's another way to ease typing: If we start to type a command like so

```
(rewri
```

and then type M-s it will be completed to the last command typed with the same beginning, i.e., in our case the last `rewrite-lemma` command which again can then be edited.

However we enter the command, it brings us to

```
  |-------
{1}   x!1 * (y!1 * z!1 * w!1) = x!1 * (y!1 * (z!1 * w!1))
```

at which point

```
(rewrite-lemma "assoc" ("x" "y!1" "y" "z!1" "z" "w!1"))
```

completes the job.

The rewrite-lemma command can also be given the :dir RL optional argument, so we could have worked on the right hand side instead like so:

```
  |-------
{1}   ((x!1 * y!1) * z!1) * w!1 = x!1 * (y!1 * (z!1 * w!1))

Rule? (rewrite-lemma "assoc" ("x" "y!1" "y" "z!1" "z" "w!1") :dir RL)
  |-------
{1}   ((x!1 * y!1) * z!1) * w!1 = x!1 * ((y!1 * z!1) * w!1)
```

Even quicker proofs: Filling in the instantiations is tedious and can partly be automated. That's what the command rewrite (p.64 of `prover-guide.ps`) does for us. Unfortunately, not always successfully, which is why it's good to know the more basic commands. In the example at hand it works, however, and we can do the entire proof by issuing the following four commands:

```
(skolem!)
(rewrite "assoc")
(rewrite "assoc")
(rewrite "assoc")
```

Even quicker is the following approach: using the command (p. 89 f. of prover-guide.ps)

```
(auto-rewrite "assoc")
```

we tell PVS that it should consider all instances of assoc as automatic rewrite rules. After that command, (grind) completes the task.

Uniqueness of neutral elements We postulate a neutral element by adding

```
e : M
neutral_left : AXIOM
    FORALL(x:M):e*x=x
neutral_right : AXIOM
    FORALL(x:M):x*e=x
```

Our goal is

```
neutral_unique : THEOREM
    FORALL(e1:M):
      (FORALL(x:M): e1*x=x) AND
      (FORALL(x:M): x*e1=x) IMPLIES e=e1
```

Here it is useful to first get an idea of how this proof should be done informally:

If e1 is also a neutral element then e = e*e1. By neutrality of e the right hand side equals e1 and we're done.

In PVS after (skolem!) and (flatten) or, more compactly, (skosimp), we get

```
{-1}  FORALL (x: M): e1!1 * x = x
{-2}  FORALL (x: M): x * e1!1 = x
  |-------
{1}   e = e1!1
```

We want to expand e as e * e1!1 using -2. We instantiate...

```
(inst -2 "e")

{-1}  FORALL (x: M): e1!1 * x = x
{-2}  e * e1!1 = e
  |-------
{1}   e = e1!1
```

and replace

```
(replace -2 1 :dir RL)
```

bringing us to

```
[-1]  FORALL (x: M): e1!1 * x = x
[-2]  e * e1!1 = e
  |-------
{1}   e * e1!1 = e1!1
```

which is an instance of neutral_left. The way to convince PVS of this is

```
(use "neutral_left")
```

A more pedestrian way would be to use lemma and inst.

A slightly quicker proof After (skosimp) we can use (rewrite-with-fnum -2 ("x" "e") :dir RL) to achieve the expansion of the left hand side. The command rewrite-with-fnum, is like rewrite, so doesn't normally require a substitution (instantiation). In this case, we have to give it because otherwise the replacement is applied to the right hand side, too!.

This brings us to

```
[-1]  FORALL (x: M): e1!1 * x = x
[-2]  FORALL (x: M): x * e1!1 = x
  |-------
{1}   e * e1!1 = e1!1
```

At which point we conclude using (use "neutral_left").

I couldn't find a more efficient proof of that one. Can you?

Invertible elements An element of M is invertible if it has an inverse:

```
  Invertible(x:M) : boolean = EXISTS(y:M): x*y=e AND y*x = e
```

We can prove that the neutral element is invertible:

```
   inv_neutral : THEOREM
     Invertible(e)
```

We see here, how, abbreviations a.k.a. definitions are introduced.

After invoking the prover the first command must be expand "Invertible" to open the definition. This brings us to

```
  |-------
{1}   EXISTS (y: M): e * y = e AND y * e = e
```

Now we have to come up with an alleged inverse to e. Surprise, it's going to be e itself.

```
(inst 1 "e")
```

```
  |-------
{1}   e * e = e AND e * e = e
```

Here we could also have used (inst? 1) which would leave it to PVS to find the correct instantiation. It sometimes does....

We conclude with (rewrite "neutral_left") followed by (split).

The last theorem in this series is that invertibles are closed under product:

```
  |-------
{1}   FORALL (x, y: M):
        Invertible(x) AND Invertible(y) IMPLIES Invertible(x * y)
```

(skosimp) then (expand "Invertible") brings us to

```
{-1}  EXISTS (y: M): x!1 * y = e AND y * x!1 = e
{-2}  EXISTS (y: M): y!1 * y = e AND y * y!1 = e
  |-------
{1}   EXISTS (y: M): x!1 * y!1 * y = e AND y * (x!1 * y!1) = e
```

Notice the then "strategy" (p. 111 of prover-guide.ps). It sequences commands.

Before being able to instantiate 1, i.e., come up with an alleged inverse to (x!1*y!1) we must "open" the assumptions, i.e., introduce fresh constants for the inverses of x!1 and y!1, respectively, which are guaranteed by -1 and -2.

I find it better to give suggestive names to these, so we do

```
(skolem -1 "xinv") then (skolem -2 "yinv") then (flatten)
```

to get

```
{-1}  x!1 * xinv = e
{-2}  xinv * x!1 = e
{-3}  y!1 * yinv = e
{-4}  yinv * y!1 = e
  |-------
[1]   EXISTS (y: M): x!1 * y!1 * y = e AND y * (x!1 * y!1) = e
```

Now, we have to think a bit as to what the inverse to x!1*y!1 should be. Well, thinking of * as sequencing of "actions" it becomes clear that the inverse ought to be yinv * xinv. That's the "rule of sock and shoe". Therefore, (inst 1 "yinv*xinv") is the command of choice.

```
{-1}  x!1 * xinv = e
{-2}  xinv * x!1 = e
{-3}  y!1 * yinv = e
{-4}  yinv * y!1 = e
  |-------
[1]   x!1 * y!1 * (yinv * xinv) = e AND (yinv * xinv) * (x!1 * y!1) = e
```

Relying on PVS' cleverness and doing (inst? 1) isn't a good idea here.

Splitting (∧-R) brings us two subgoals of which we'll only treat the first here:

```
{-1}  x!1 * xinv = e
{-2}  xinv * x!1 = e
{-3}  y!1 * yinv = e
{-4}  yinv * y!1 = e
  |-------
[1]   x!1 * y!1 * (yinv * xinv) = e
```

Now we must first "rebracket" our goal to

```
x!1 * (y!1 * yinv) * xinv = e
```

While this can certainly be done by successive application of associativity, it is easier to just claim this and prove it separately. To do this, we issue the command

```
(case "x!1 * (y!1 * yinv) * xinv = e")
```

This presents us with two subgoals. One asking us to prove our goal under the extra assumption of the "claim":

```
[-1]  x!1 * (y!1 * yinv) * xinv = e
[-2]  x!1 * xinv = e
[-3]  xinv * x!1 = e
[-4]  y!1 * yinv = e
[-5]  yinv * y!1 = e
  |-------
[1]   x!1 * y!1 * (yinv * xinv) = e
```

This follows from associativity, so

```
(auto-rewrite "assoc") then (grind)
```

does the job. Next, we must prove our claim:

```
[-1]  x!1 * xinv = e
[-2]  xinv * x!1 = e
[-3]  y!1 * yinv = e
[-4]  yinv * y!1 = e
  |-------
{1}   x!1 * (y!1 * yinv) * xinv = e
[2]   x!1 * y!1 * (yinv * xinv) = e
```

The old goal is still there, we can remove it with (delete 2) corresponding to rule WEAK-R. The rest is a rewriting consequence of neutral_left and neutral_right, so we install these and grind.

We could have turned off associativity with the command (stop-rewrite "assoc"), but this wasn't even necessary here.

4 Induktion und Unvollständigkeit

In diesem Kapitel wollen wir die Induktion für natürliche Zahlen und andere Datentypen im Rahmen der Prädikatenlogik formal studieren. Wir beginnen mit PVS-Beispielen.

4.1 Natürliche Zahlen in PVS

Wir definieren eine rekursive Funktion durch die Deklaration

```
summe(n:nat) : RECURSIVE nat =
IF n=0 THEN 0 ELSE
  n + summe(n-1) ENDIF
MEASURE n
```

Diese rekursive Definition ist gleichbedeutend mit der Deklaration eines Funktionssymbols `summe` und der beiden Formeln:

```
summe(0) = 0
n > 0 IMPLIES summe(n= n + summe(n-1)
```

Einfache Rechenregeln der Arithmetik mit natürlichen, rationalen und reellen Zahlen sind in PVS eingebaut; will man Arithmetik im Rahmen der Prädikatenlogik formal untersuchen, so muss man diese in Form von Axiomen hinzunehmen.

Mit der Phrase

```
MEASURE n
```

wird eine *Abstiegsfunktion* angegeben, also eine Funktion auf den Argumenten der rekursiv definierten Funktion, deren Wert bei rekursiven Aufrufen echt abnimmt. Auf diese Weise ist gesichert, dass tatsächlich eine Funktion existiert, die die Gleichungen erfüllt. Es ist eine Designentscheidung von PVS bei der Deklaration rekursiver Funktion auf solch einer Abstiegsfunktion zu bestehen.

Wir können die definierenden Gleichungen nun benutzen, um zum Beispiel zu zeigen, dass gilt

```
test : PROPOSITION
    summe(4) = 10
```

Will man aber eine allquantifizierte Aussage beweisen, so braucht man die Induktion:

```
gauss : PROPOSITION
   FORALL (n:nat): summe(n) = n * (n+1) / 2
```

In PVS ist die Induktion durch den Befehl `induct` repräsentiert: Nach `(induct n)` erhalten wir die beiden Unterziele

```
  |-------
{1}   summe(0) = 0 * (0 + 1) / 2
```

und

```
  |-------
{1}   FORALL j:
        summe(j) = j * (j + 1) / 2 IMPLIES
         summe(j + 1) = (j + 1) * (j + 1 + 1) / 2
```

Das erste verwandelt sich nach `(expand "summe")` in

```
  |-------
{1}   0 = 0 / 2
```

welches mit `(assert)` oder `(grind)` durch automatische Anwendung arithmetischer Rechenregeln gelöst werden kann. Die Taktik `(grind)` führt sogar automatisch die Expansion aus.

Das zweite wird nach `(skosimp)` zu

```
{-1}  summe(j!1) = j!1 * (j!1 + 1) / 2
  |-------
{1}   summe(j!1 + 1) = (j!1 + 1) * (j!1 + 1 + 1) / 2
```

Der Befehl `(expand "summe")` wäre hier fehl am Platze, da beide Vorkommen von `summe` expandiert würden. Stattdessen verwendet man

```
(expand "summe" :if-simplifies T)
```

wodurch nur solche Vorkommen expandiert werden, die mit einer Vereinfachung eines if-then-else Termes einhergehen. Der Befehl führt dann auf

```
[-1]  summe(j!1) = j!1 * (j!1 + 1) / 2
  |-------
{1}   1 + summe(j!1) + j!1 = (2 + j!1 + (j!1 * j!1 + 2 * j!1)) / 2
```

Jetzt können wir mit `(replace -1 1)` die Induktionshypothese einsetzen und erhalten

```
[-1]  summe(j!1) = j!1 * (j!1 + 1) / 2
  |-------
{1}   1 + j!1 * (j!1 + 1) / 2 + j!1 = (2 + j!1 + (j!1 * j!1 + 2 * j!1)) / 2
```

was wiederum mit `(assert)` gelöst werden kann.

Wir bemerken, dass das ursprüngliche Ziel in einem einzigen Schritt mit dem mächtigen Befehl `(induct-and-simplify "n")` gelöst werden kann.

Für ein weiteres Beispiel definieren wir die Fibonacci Zahlen durch

```
fib(n:nat) : RECURSIVE nat =
 IF n=0 OR n=1 THEN 1 ELSE fib(n-1) + fib(n-2) ENDIF
MEASURE n
```

und eine Hilfsfunktion durch

```
c(n:nat): RECURSIVE int = IF n=0 THEN 1 ELSE -c(n-1) ENDIF MEASURE n
```

Wir versuchen nun, den folgenden Satz von Cassini zu beweisen:

```
 THEOREM FORALL (n:nat):
      fib(n+2)*fib(n) - fib(n+1) * fib(n+1) = c(n)
```

Nach `(induct "n")`, der Erledigung des Induktionsanfangs und `(skosimp)` werden wir auf

```
{-1}  fib(j!1 + 2) * fib(j!1) - fib(j!1 + 1) * fib(j!1 + 1) = c(j!1)
  |-------
{1}   fib(j!1 + 1 + 2) * fib(j!1 + 1) - fib(j!1 + 1 + 1) * fib(j!1 + 1 + 1)
       = c(j!1 + 1)
```

geführt, was nach zweimaliger Expansion mit `if-simplifies T` auf

```
[-1]  fib(1 + j!1) * fib(j!1) + fib(j!1) * fib(j!1) -
       fib(1 + j!1) * fib(1 + j!1)
       = c(j!1)
  |-------
{1}   fib(1 + j!1) * fib(1 + j!1) - fib(1 + j!1) * fib(j!1) -
       fib(j!1) * fib(j!1)
       = c(1 + j!1)
```

führt. Expandieren wir nun auch noch die Hilfsfunktion `c`, so kommt:

```
[-1]  fib(1 + j!1) * fib(j!1) + fib(j!1) * fib(j!1) -
       fib(1 + j!1) * fib(1 + j!1)
       = c(j!1)
  |-------
{1}   fib(1 + j!1) * fib(1 + j!1) - fib(1 + j!1) * fib(j!1) -
       fib(j!1) * fib(j!1)
       = -c(j!1)
```

Das aber ist eine einfache arithmetische Folgerung, die mit `(assert)` gelöst wird.

4.2 Listen in PVS

Listen sind ein in PVS fest eingebauter Datentyp. Wir werden die Syntax nicht formal einführen, sondern anhand von Beispielen kennenlernen. Durch die folgende PVS Deklaration wird ein Typ `t` (also ein einstelliges Prädikat) und die "append"-Funktion auf Listen mit Einträgen in `t` definiert.

```
t : TYPE
append(l1,l2:list[t]) : RECURSIVE list[t] =
     IF null?(l1) THEN l2
     ELSE cons(car(l1),append(cdr(l1),l2))
    ENDIF
MEASURE length(l1)
```

Dies entspricht in etwa der SML Definition

```
fun append(l1,l2) = if null l1 then l2 else hd l1 :: append(tl l1, l2)
```

Ein wichtiger Unterschied ist, dass PVS wie schon gesagt eine Abstiegsfunktion verlangt und versucht, automatisch zu prüfen ob die gegebene Abstiegsfunktion tatsächlich eine ist. Kann PVS das nicht tun, so kann man versuchen es selbst interaktiv wie andere PVS Theoreme zu beweisen. Außerdem sind `hd` und `tl`, die in PVS `car` und `cdr` heißen *partielle* Funktionen, die nur auf nichtleeren Listen definiert sind. Bei jeder Anwendung ist zu prüfen, ob die Argumente tatsächlich im Definitionsbereich liegen. Auch das versucht PVS automatisch zu tun. Mit `M-x show-tccs` kann man diese ganzen Seitenbedingungen anzeigen.

Wir können nun versuchen, zu beweisen

```
append0 : PROPOSITION
FORALL (l1:list[t]):append(null,l1) = l1
```

Dies lässt sich mit `(skosimp)` und `(expand "append")` erledigen. Für das ähnlich aussehende

```
append1 : PROPOSITION
FORALL (l1:list[t]):append(l1,null) = l1
```

benötigen wir die Induktion über Listen. Mit `(induct "l1")` erhalten wir zunächst

```
  |-------
{1}   append(null, null) = null
```

was mit `expand` gelöst wird und dann, nach

```
(then (skolem 1 ("h1" "t1")) (flatten))
```

den Induktionsschritt:

```
{-1}  append(t1, null) = t1
  |-------
{1}   append(cons(h1, t1), null) = cons(h1, t1)
```

Expandieren führt auf

```
[-1]  append(t1, null) = t1
  |-------
{1}   cons(h1, append(t1, null)) = cons(h1, t1)
```

was wir mit `(replace -1 1)` zum Abschluss bringen.

Ähnlich beweist man

```
append_assoc: LEMMA
     FORALL(l1,l2,l3:list[t]):
     append(append(l1, l2), l3) = append(l1, append(l2, l3))
```

Man kann nun eine "reverse"-Funktion definieren durch

```
rev(l:list[t]) : RECURSIVE list[t] =
     IF null?(l) THEN null
       ELSE append(rev(cdr(l)),cons(car(l),null))
     ENDIF
     MEASURE length(l)
```

und zum Beispiel beweisen

```
rev_append : LEMMA FORALL(l1,l2:list[t]):
             rev(append(l1,l2)) = append(rev(l2),rev(l1))
```

Hierfür empfiehlt es sich, mit `(auto-rewrite)` die Lemmas `append1` und `append_assoc` zu installieren.

Wer will kann versuchen, andere Funktionen aus der Funktionalen Programmierung in PVS zu repräsentieren und Eigenschaften von diesen zu beweisen. Beliebte Beispiele sind Sortieren durch Einfügen und Sortieren durch Mischen.

4.3 Peano Arithmetik

Die Peano Arithmetik ist eine Formalisierung der natürlichen Zahlen (also etwas weniger, als in den obigen PVS-Beispielen, die ja auch negative Zahlen verwendet haben) innerhalb der Prädikatenlogik. Sie basiert auf der Sprache $\mathcal{L}_{\mathrm{PA}}$ mit den Konstanten 0,1, den binären Funktionssymbolen $+$ und $\cdot$ (beide infixnotiert und letzteres auch oft weggelassen). Das einzige Relationssymbol ist die eingebaute Gleichheit.

Die Axiome der Peanoarithmetik (PA) sind die Formeln, die in Abb. **??** angegeben sind. Wir bezeichnen diese Formelmenge mit PA.

Definition 42
Eine Formel ϕ in der Sprache der Peanoarithmetik heißt in der Peanoarithmetik beweisbar, wenn es eine endliche Teilmenge $\Gamma \subseteq \mathrm{PA}$ gibt, sodass die Formel $\bigwedge \Gamma \Rightarrow \phi$ allgemeingültig ist.

Man schreibt dann $\mathrm{PA} \vdash \phi$.

Man beachte, dass wegen der Vollständigkeit der Prädikatenlogik $\mathrm{PA} \vdash \phi$ gleichbedeutend ist damit, dass eine endliche Teilmenge $\Gamma \subseteq \mathrm{PA}$ existiert, sodass $\Gamma \Longrightarrow_{\mathcal{L}_{\mathrm{PA}}} \phi$ im Sequenzenkalkül herleitbar ist. Dies wiederum ist äquivalent dazu, dass die Sequenz

$\forall x.0 + x = x$
$\forall x.\forall y.x + y = y + x$
$\forall x.\forall y.\forall z.x + (y + z) = (x + y) + z$
$\forall x.0 \cdot x = 0$
$\forall x.1 \cdot x = x$
$\forall x.\forall y.xy = yx$
$\forall x.\forall y.\forall z.x(yz) = (xy)z$
$\forall x.\forall y.\forall z.x(y + z) = xy + xz$
$\neg 0 = 1$
$\phi[0/x] \wedge (\forall x.\phi \Rightarrow \phi[x + 1/x]) \Rightarrow \forall x.\phi$ für jede Formel ϕ mit einer freien Variablen x.

Abbildung 4.1: Axiome der Peanoarithmetik PA

$\Longrightarrow_{\mathcal{L}_{\mathrm{PA}}} \phi$ in einer Version des Sequenzenkalküls herleitbar ist, in der Formeln aus PA stets "geladen" werden können, in dem also noch die Regel

$$\frac{\Gamma, \psi \Longrightarrow_{\mathcal{L}} \Delta \qquad \psi \in \mathrm{PA}}{\Gamma \Longrightarrow_{\mathcal{L}} \Delta}$$

verfügbar ist.

Beispiel: Es gilt:

$$\mathrm{PA} \vdash \forall x.\exists y.x = 2y \vee x = 2y + 1$$

Hier steht 2 für den Term 1+1. Für den Beweis verwenden wir die Abkürzung

$$\phi \equiv \exists y.x = 2y \vee x = 2y + 1$$

Beachte, dass die herzuleitende Formel gerade $\forall x.\phi$ ist.

Nun gilt

$$\mathrm{PA} \vdash \phi[0/x]$$

indem wir den Existenzquantor mit 0 instanziieren und so auf $2 \cdot 0 = 0$ geführt werden, was mit Gleichungslogik aus PA folgt.

Weiters gilt

$$\mathrm{PA} \vdash \forall x.\phi \Rightarrow \phi[x + 1/x]$$

denn wenn $2y = x$ so folgt $2y+1 = x+1$ und wenn $2y+1 = x$, dann folgt $2(y+1) = x+1$. Dies kann leicht in einen entsprechenden Beweis im Sequenzenkalkül umgebaut werden.

Mithilfe der entsprechenden Instanz des Induktionsschemas, welche ja in PA enthalten ist, kann dann $\forall x.\phi$ geschlossen werden.

Definition 43
Es sei $\mathcal{N}$ die Struktur über $\mathcal{L}_{\mathrm{PA}}$ mit $|\mathcal{N}| = \mathbb{N}$ und $x +_{\mathcal{N}} y = x + y$ und $x \cdot_{\mathcal{N}} y = xy$ und $0_{\mathcal{N}} = 0$ und $1_{\mathcal{N}} = 1$. Diese Struktur heißt das *Standardmodell* der PA.

Mithilfe des Kompaktheitssatzes lassen sich noch weitere echt verschiedene Modelle konstruieren, etwa eines, in dem es eine Zahl gibt, die größer ist als alle Zahlen der Form $1 + \cdots + 1$.

Definition 44
Eine k-stellige Relation $R \subseteq \mathbb{N}^k$ heißt PA-definierbar, wenn es eine Formel $\phi_R(\vec{x})$ mit freien Variablen $x_1, \ldots, x_k$ gibt, sodass

$$(v_1, \ldots, v_k) \in R \iff [\![\phi]\!]_{\mathcal{N}}[x_1 \mapsto v_1, \ldots, x_k \mapsto v_k] = \mathtt{tt}$$

So ist zum Beispiel die Kleiner-Relation definierbar durch

$$\phi_< \equiv \exists z.x + (z + 1) = y$$

Ist eine Relation R PA-definierbar, so kann man für sie *abkürzungshalber* ein neues Prädikatsymbol einführen.

Tut man dies etwa für die Kleiner-Relation, so steht dann beispielsweise die (in PA beweisbare) Formel

$$\forall x.\forall y.x < y \vee x{=}y \vee y < x$$

als Abkürzung für die Formel

$$\forall x.\forall y.(\exists z.x + (z + 1) = y) \vee x{=}y \vee (\exists z.y + (z + 1) = x)$$

Ebenso sieht man leicht, dass die Relationen $>, \leq, \geq$ und "kongruent modulo" PA-definierbar sind.

Definition 45
Eine k-stellige Funktion $f : \mathbb{N}^k \to \mathbb{N}$ heißt PA-definierbar, wenn es eine Formel $\phi_f(\vec{x}, y)$ mit freien Variablen $x_1, \ldots, x_k, y$ gibt, so dass gilt

$$f(v_1, \ldots, v_k) = u \iff [\![\phi]\!]_{\mathcal{N}}[x_1 \mapsto v_1, \ldots, x_k \mapsto v_k, y \mapsto u] = \mathtt{tt}$$

und außerdem

$$\mathrm{PA} \vdash \forall x_1.\cdots.\forall x_k.\exists y.\phi(\vec{x}, y)$$
$$\mathrm{PA} \vdash \forall x_1.\cdots.\forall x_k.\forall y'.\phi(\vec{x}, y) \wedge \phi(\vec{x}, y') \Rightarrow y{=}y'$$

So ist zum Beispiel die Funktion $f(x) = [x/2]$ (eckige Klammern = größte ganze Zahl kleiner oder gleich) PA-definierbar. Man wählt hier $\phi_f(x, y) \equiv 2x{=}y \vee 2x{+}1{=}y$. Wir haben schon gezeigt, dass

$$\mathrm{PA} \vdash \forall x_1.\cdots.\forall x_k.\exists y.\phi(\vec{x}, y)$$

Die zweite Eigenschaft verbleibt als Übung.

Ist eine Funktion f PA-definierbar, so kann man für sie *abkürzungshalber* ein neues Funktionssymbol einführen.

So steht dann beispielsweise die Formel

$$\forall x.x \leq 2 \vee f(x) \cdot f(x) > x$$

für

$$\forall x.\forall t.\phi_f(x,t) \Rightarrow x \leq 1+1 \vee t \cdot t > x$$

Die beiden geforderten Eigenschaften (E) und (U) stellen sicher, dass man mit diesen "Abkürzungen" genauso rechnen kann, wie mit echten Funktionssymbolen. Hier noch ein Beispiel:

$$\forall x.x < 8 \Rightarrow f(f(f(x))) < y$$

steht für

$$\forall x.\forall t_1.\forall t_2.\forall t_3.\phi_f(x,t_1) \wedge \phi_f(t_1,t_2) \wedge \phi_f(t_2,t_3) \wedge x < 1+1+1+1+1+1+1+1 \Rightarrow t_3 < y$$

Satz 26 (Paarungsfunktion)
Die sog. Paarungsfunktion $\langle -,- \rangle : \mathbb{N}^2 \to \mathbb{N}$*, welche durch*

$$\langle x,y \rangle = (x+y)(x+y+1)/2 + x$$

definiert ist, ist bijektiv. Die Paarungsfunktion und ihre Inversen $fst : \mathbb{N} \to \mathbb{N}$ *sowie* $snd : \mathbb{N} \to \mathbb{N}$ *sind PA-definierbar.*

Für die Inversen gilt:

$$fst(\langle x,y \rangle) = x$$
$$snd(\langle x,y \rangle) = x$$

Beweis (Skizze) PA-Definierbarkeit der Paarungsfunktion ist klar. Sie zählt die Gitterpunkte im nichtnegativen Quadranten in immer größeren Scheiben, die von links oben nach rechts unten verlaufen, ab. Es gilt also $\langle 0,0 \rangle = 0, \langle 1,0 \rangle = 1, \langle 0,1 \rangle = 2, \langle 0,2 \rangle = 3, \langle 1,1 \rangle = 4, \langle 0,2 \rangle = 5, \langle 0,3 \rangle = 6, \langle 1,2 \rangle = 7, \langle 2,1 \rangle = 8, \langle 3,0 \rangle = 9, \langle 0,4 \rangle = 10, \langle 1,3 \rangle = 11, \ldots$.

Diese Intuition kann zu einem rigorosen Beweis der Bijektivität ausgebaut werden.

Die Graphen der Inversen definiert man durch

$$\phi_{\text{fst}}(x,t) \equiv \exists s.\langle s,t \rangle = x$$
$$\phi_{\text{snd}}(x,t) \equiv \exists s.\langle t,s \rangle = x$$

Indem man den Beweis der Bijektivität in PA formalisiert, erhält man die gewünschten Eigenschaften (E) und (U) im Sinne von Def. **??** für diese Relationen. ∎

Der Beweis des folgenden Satzes ist etwas umständlich und wird daher ausgelassen.

Satz 27 (Iterationssatz)
Sei

$$f : \mathbb{N} \to \mathbb{N}$$

PA-definierbare Funktion. Dann ist auch die Funktion

$$F : \mathbb{N}^2 \to \mathbb{N}$$
$$F(x,n) = \underbrace{f(f(\ldots f}_{n \text{ Stück}}(x)\cdots)$$

PA-definierbar.

Nunmehr können wir in der Peano Arithmetik Listen, Bäume und andere Datentypen durch natürliche Zahlen codieren. Z.B. können wir die Liste $[3,4,5]$ durch $\langle\langle\langle 0,3\rangle,4\rangle,5\rangle$, also 2272 repräsentieren. Dies erlaubt uns auch rekursive Definitionen in PA nachzubilden und die im letzten Abschnitt in PVS durchgeführten Beweise auch formal in der Peano-Arithmetik zu führen.

4.4 Der Unvollständigkeitssatz von Gödel

In diesem letzten Abschnitt werden wir eine geschlossene Formel ϕ_G angeben, sodass weder PA $\vdash \phi_G$ noch PA $\vdash \neg\phi_G$ gilt. Da entweder ϕ_G oder $\neg\phi_G$ im Standardmodell wahr sein muss, bedeutet dies, dass es eine (im Standardmodell) wahre Formel gibt, für die in PA kein Beweis existiert. Es zeigt sich im übrigen, dass ϕ_G und nicht $\neg\phi_G$ wahr ist.

Man könnte dann vielleicht eine Erweiterung von PA um zusätzliche Axiome und Schlussregeln in Betracht ziehen; allerdings ist die Konstruktion der Formel ϕ_G so uniform, dass sie sich auch für solch eine Erweiterung wieder durchführen ließe.

Dieses fundamentale Resultat wurde 1931 von Kurt Gödel gezeigt und ist seitdem als *Erster Gdelscher Unvollständigkeitssatz* bekannt.

In gewisser Hinsicht ist das Ergebnis nicht so überraschend. Gäbe es für jede Formel einen Beweis, so könnte man von einer gegebenen Formel ϕ feststellen, ob sie wahr ist, indem man einfach systematisch Beweisbäume generiert, bis ein Beweis für ϕ oder aber für $\neg\phi$ gefunden ist. Das würde dann im Prinzip die Mathematiker arbeitslos machen.

Man muss dazu bedenken, dass viele derzeit offene mathematische Probleme als Formeln der Peano Arithmetik geschrieben werden können. So zum Beispiel die Goldbachsche Vermutung: Es gibt unendlich viele x, sodass sowohl x, als auch $x+2$ Primzahlen sind. Primzahl zu sein ist PA-definierbar:

$$\mathrm{prim}(x) \equiv \forall y.\forall z.yz = x \Rightarrow y = 1 \vee z = 1$$

Also entspricht die Goldbachsche Vermutung der Formel

$$\forall x \exists y.y > x \wedge \mathrm{prim}(y) \wedge \mathrm{prim}(y+2)$$

Wir wissen derzeit nicht, ob die Goldbachsche Vermutung gilt und selbst wenn sie gilt, wissen wir nicht, ob sie beweisbar ist, sei es in der Peano Arithmetik sei es in einem stärkeren System, etwa der Mengenlehre.

Die Gödel-Formel ϕ_G ist nun so konstruiert, dass ϕ_G wahr ist, genau dann wenn ϕ_G selbst keinen Beweis in PA hat. Dass man so eine Formel bauen kann ist natürlich nicht offensichtlich. Wenn es aber gelingt, dann würde ein hypothetischer Beweis von ϕ_G ja gerade das Gegenteil von ϕ_G nach sich ziehen. Das kann nicht sein. Wäre dagegen $\neg\phi_G$ beweisbar, so wäre $\neg\phi_G$ insbesondere wahr, also gäbe es einen Beweis von ϕ_G selbst. Auch ein Widerspruch. Wir wollen uns jetzt einen informellen Überblick über die Konstruktion so einer Formel ϕ_G verschaffen.

4.4.1 Kodierung der Syntax

Wenn man Listen und Bäume als natürliche Zahlen codieren kann, ist das auch für Syntaxbäume und Herleitungsbäume möglich. Ohne detaillierte Begründung halten wir folgendes fest

- Zu jeder geschlossenen Formel ϕ gibt es eine natürliche Zahl $\ulcorner\phi\urcorner$, die die Formel eindeutig codiert. Man bezeichnet $\ulcorner\phi\urcorner$ als *Gödelnummer*, manchmal auch *Gödelisierung* von ϕ.
- Auch zu jeder Formel ϕ mit einer freien Variablen x gibt es eine eindeutige Codierung $\ulcorner\phi\urcorner^x$.
- Eine zweistellige Funktion subst sodass gilt

  $$\text{subst}(n, \ulcorner\phi\urcorner^x) = \ulcorner\phi[n/x]\urcorner$$

  ist PA-definierbar und zwar so, dass diese definierende Gleichung für konkretes ϕ und n auch in PA beweisbar ist.
- Zu jeder Herleitung H in der Peano Arithmetik gibt es eine natürliche Zahl $\ulcorner H\urcorner$, die diese Herleitung eindeutig codiert.
- Es gibt eine PA-definierbare Relation $\text{bew}(x, y)$ sodass $\text{bew}(\ulcorner H\urcorner, \ulcorner\phi\urcorner)$ genau dann wenn H eine gültige Herleitung der Formel ϕ ist.
- Für jede geschlossene Formel ϕ gilt:

  $$\text{PA} \vdash \phi \text{ genau dann wenn } \text{PA} \vdash \exists x.\text{bew}(x, \ulcorner\phi\urcorner)$$

Der letzte Spiegelstrich bedarf vielleicht der Erläuterung. Gilt $\text{PA} \vdash \phi$, so nenne man die zugrundeliegende Herleitung H. Bei vernünftiger Implementierung von bew sollte dann $\text{bew}(\ulcorner H\urcorner, \ulcorner\phi\urcorner)$ und also mit $\exists$-R dann auch $\exists x.\text{bew}(x, \ulcorner\phi\urcorner)$ beweisbar sein. Die umgekehrte Richtung verlangt nur, dass bew korrekt implementiert ist.

4.4.2 Eine wahre aber nicht beweisbare Formel

Wir definieren jetzt eine feste Formel ψ mit einer freien Variablen x durch

$$\psi \equiv \neg\exists y.\text{bew}(y, \text{subst}(x, x))$$

und bezeichnen ihre Codierung mit g

$$g := \ulcorner\psi\urcorner^x$$

Unsere "Gödel-Formel" ϕ_G definieren wir jetzt durch

$$\phi_G \equiv \psi[g/x]$$

Es gilt also

$$\phi_G \equiv \neg\exists y.\mathrm{bew}(y, \mathrm{subst}(g, g))$$

Außerdem aufgrund der Spezifikation von subst

$$\ulcorner \phi_G \urcorner = \mathrm{subst}(g, g)$$

Demnach folgt mit unserer Annahme an subst

$$\mathrm{PA} \vdash \mathrm{subst}(g, g) = \ulcorner \phi_G \urcorner$$

weswegen

$$\mathrm{PA} \vdash \phi_G \Leftrightarrow \neg\exists y.\mathrm{bew}(x, \ulcorner \phi_G \urcorner)$$

Die Formel ϕ_G besagt also, dass sie selbst keinen Beweis in PA besitzt. Wäre jetzt $\mathrm{PA} \vdash \phi_G$, dann erhielten wir sowohl $\mathrm{PA} \vdash \exists y.\mathrm{bew}(y, \ulcorner \phi_G \urcorner)$ wegen des vorgelegten Beweises, als auch $\mathrm{PA} \vdash \neg\exists y.\mathrm{bew}(y, \ulcorner \phi_G \urcorner)$ wegen der eben hergeleiteten Äquivalenz. Ein Widerspruch. Wäre hingegen $\mathrm{PA} \vdash \neg\phi_G$, so auch wegen der eben hergeleiteten Äquivalenz $\mathrm{PA} \vdash \exists y.\mathrm{bew}(y, \ulcorner \phi_G \urcorner)$, also $\mathrm{PA} \vdash \phi_G$, auch ein Widerspruch.

Da wir insbesondere gesehen haben, dass kein Beweis von ϕ_G existiert, ist also die Formel $\neg\exists y.\mathrm{bew}(y, \ulcorner \phi_G \urcorner)$ im Standardmodell wahr und damit auch die Formel ϕ_G, die ja dazu äquivalent ist. Sie hat aber eben keinen Beweis.

Wir bemerken, dass Paris und Harrington eine echt-mathematische und nicht künstlich syntaktische Formel angegeben haben, welche im Standardmodell wahr, aber auch nicht in PA beweisbar ist.

Es sei noch einmal wiederholt, dass die Konstruktion der Gödel-Formel ϕ_G genauso auch in jedem anderen Beweissystem für die Mathematik nachgespielt werden kann, solange dieses Beweissystem nur korrekt ist und die Codierung von Syntax und Beweisen erlaubt. Der Gödelsche Unvollständigkeitssatz zeigt also eine prinzipielle Grenze des rigorosen Beweisens.

ENDE DES SKRIPTUMS