Metamath

Eine Computersprache für mathematische Beweise

Norman Megill

mit umfangreichen Überarbeitungen durch
David A. Wheeler

Deutsche Übersetzung von
Georg M. van der Vekens und Alexander W. van der Vekens

18.11.2023

ISBN: 978-0-359-70223-7

Lulu Press
Morrisville, North Carolina
USA

Norman Megill
93 Bridge St., Lexington, MA 02421
E-Mail Adresse: `nm@alum.mit.edu`

David A. Wheeler
E-Mail Adresse: `dwheeler@dwheeler.com`

`http://metamath.org`

Titelseite: Aleph Null ($\aleph_0$) ist das Symbol für die erste unendliche Kardinalzahl, entdeckt von Georg Cantor im Jahr 1873. Wir verwenden ein rotes Aleph Null (mit dunklem Umriss und goldenem Schimmer) als das Logo für Metamath. Urheber: Norman Megill (1994) und Giovanni Mascellani (2019), Public Domain.

Diese Übersetzung ist Norman Dwight Megill,
dem Erfinder von Metamath, gewidmet.

Inhaltsverzeichnis

Vorwort

Übersicht

Metamath ist eine Computersprache und ein zugehöriges Computerprogramm zur Archivierung, Verifikation und Untersuchung mathematischer Beweise auf einer sehr detaillierten Ebene. Die Metamath-Sprache enthält keine Mathematik an sich, sondern betrachtet alle mathematischen Aussagen als reine Folgen von Symbolen. Bei der Nutzung von Metamath werden bestimmte, spezielle Symbolsequenzen (Axiome) vorgegeben, die Metamath sagen, welche Schlussfolgerungsregeln erlaubt sind. Metamath ist nicht auf ein bestimmtes mathematisches Gebiet beschränkt. Die Metamath-Sprache ist einfach und robust, sie besitzt so gut wie keine fest verdrahtete Syntax. Wir[1] glauben, dass sie vielleicht den einfachst möglichen Ansatz bietet, mit dem im Wesentlichen die gesamte Mathematik mit absoluter Strenge ausgedrückt werden kann.

Mit der Metamath-Sprache können formale oder mathematische Systeme[2] aufgebaut werden, die Schlussfolgerungen aus Axiomen umfassen. Obwohl zusammen mit Metamath eine Datenbasis bereitgestellt wird, die einen empfohlenen Satz von Axiomen für die Standardmathematik enthält, könnte man nach Belieben eigene Symbole, Syntax, Axiome, Regeln und Definitionen vorgeben.

Der Name „Metamath“ wurde gewählt um anzudeuten, dass die Sprache ein Mittel zur *Beschreibung* der Mathematik und nicht die Mathematik *selbst* ist. Tatsächlich ist in gewissem Sinne jede mathematische Sprache metamathematisch. Auf Papier geschriebene oder in einem Computer

[1] Sofern nicht anders angegeben, beziehen sich die Worte „Ich“, „mich“ und „mein“ auf Norman Megill, während „wir“, „uns“ und „unser“ auf Norman Megill und David A. Wheeler beziehen.

[2] Ein formales oder mathematisches System besteht aus einer Sammlung von Symbolen (solche wie 2, 4, + und =), Syntaxregeln, die beschreiben, wie Symbole kombiniert werden können um einen gültigen Ausdruck zu formen (solch ein Ausdruck wird auch 'wohlgeformte Formel', engl. 'well formed formula', genannt, oder kurz *wff*, gesprochen „whiff“), einige wffs (Axiome genannt), mit denen begonnen wird, und Schlussfolgerungsregeln, die beschreiben, wie Theoreme aus den Axiomen abgeleitet (bewiesen) werden können. Ein Theorem ist eine mathematische Tatsache so wie $2+2=4$. Streng genommen muss selbst solch eine offensichtliche Tatsache anhand von Axiomen bewiesen werden, um von einem Mathematiker formal akzeptiert werden zu können.

gespeicherte Symbole sind nicht die Mathematik selbst, sondern vielmehr eine Möglichkeit, mathematische Gegebenheiten und Zusammenhänge auszudrücken. Beispielsweise sind „7“ und „VII“ Symbole um die Zahl sieben in arabischen und römischen Ziffern zu bezeichnen; keines von beiden Symbolen *ist* die Zahl sieben.

Wenn Sie in der Lage sind, Computerprogramme zu verstehen und zu schreiben, sollten Sie in der Lage sein, abstrakte Mathematik mit Hilfe von Metamath nachzuvollziehen. In Verbindung mit Standard-Lehrbüchern kann Metamath Sie Schritt für Schritt zu einem Verständnis der abstrakten Mathematik von einem sehr rigorosen Standpunkt aus führen, auch wenn Sie keine formale Ausbildung in abstrakter Mathematik haben. Sobald Sie die grundlegenden Konzepte verstanden haben, bietet Metamath Ihnen durch die Verwendung einer einzigen, konsistenten Notation zur Darstellung von Beweisen die Möglichkeit, allen Beweisschritten sofort zu folgen und im Detail zu verstehen, und das selbst in Ihnen völlig unbekannten Bereichen.

Natürlich führt die bloße Fähigkeit einem Beweis zu folgen nicht unbedingt zu einer intuitive Vertrautheit mit der Mathematik. Das Auswendiglernen der Schachregeln gibt Ihnen nicht die Fähigkeit, die Spielweise eines Meisters zu würdigen, und zu wissen wie die Noten einer Partitur den Klaviertasten zugeordnet sind, gibt Ihnen nicht die Fähigkeit, in Ihrem Kopf zu hören, wie sich diese anhören würde. Aber jede dieser Tätigkeiten kann ein erster Schritt für den Einstieg in ein neues Themengebiet sein.

Metamath erlaubt es Ihnen, Beweise so zu erforschen, dass Sie den Beweis jedes Theorems, auf das in einem Beweisschritt verwiesen wird, wiederum im Detail betrachten können, und das immer weiter bis hin zu den zugrunde liegenden Axiomen der Logik und der Mengenlehre (im Falle der mitgelieferten Datenbasis für die Mengenlehre). Während Metamath nicht das nur durch Übung und harte Arbeit zu erlangende Verständnis der Mathematik auf höherer Ebene ersetzen kann, so hilft die Möglichkeit zu sehen, wie Lücken in einem Beweis geschlossen werden, den Lernprozess zu beschleunigen und Ihnen Zeit zu sparen, wenn Sie nicht selbst bei einem Beweis weiterkommen.

Die Metamath-Sprache zerlegt einen mathematischen Beweis in seine kleinstmöglichen Teile. Diese können wie Puzzleteile zusammengesetzt werden, und zwar so, dass korrekte und absolut strenge Mathematik entsteht.

Die Natur von Metamath erzwingt ein sehr präzises mathematisches Denken, ähnlich dem, das beim Schreiben eines Computerprogramms erforderlich ist. Ein entscheidender Unterschied ist jedoch, dass ein Beweis, sobald er (durch das Metamath-Programm) als korrekt verifiziert wurde, definitiv korrekt ist; er kann niemals einen versteckten „Fehler“ haben. Nachdem Sie sich an die Strenge und Genauigkeit von Metamath gewöhnt haben, könnten Sie sogar versucht sein die Haltung einzunehmen, dass ein Beweis niemals als korrekt angesehen werden sollte, bevor er nicht von einem Computer verifiziert wurde, genauso wie Sie einem Ausrechnen im Kopf oder auf Pa-

pier nicht völlig vertrauen, bis Sie die Berechnung auf einem Taschenrechner nachgeprüft haben.

Meine Zielvorstellung für Metamath war ein System zur Beschreibung und Verifikation der Mathematik, das vollständig universell und dennoch konzeptionell so einfach wie möglich ist. Da ich die Mathematik von einem axiomatischen, formalen Standpunkt aus betrachte, wollte ich erreichen, dass Metamath in der Lage ist, mit fast jedem mathematischen System umzugehen. Das ist nicht gerade einfach, aber zumindest im Prinzip möglich und hoffentlich anwendungstauglich. Ich wollte, dass es Beweise mit absoluter Strenge verifiziert, und aus diesem Grund ist Metamath eher eine „nur-kompilierbare“ Sprache als eine algorithmische oder Turing-Maschinensprache (wie Pascal, C, Prolog, Mathematica, usw.). Mit anderen Worten, eine in der Metamath-Sprache geschriebene Datenbasis „macht“ nichts; sie stellt lediglich mathematisches Wissen dar und erlaubt es, dieses Wissen als korrekt zu verifizieren. Ein Programm, das in einer algorithmischen Sprache geschrieben ist, kann potentiell versteckte Programmfehler(Bugs) haben und möglicherweise auch schwer zu verstehen sein. Aber jedes Eintrag in einer Metamath-Datenbasis muss konsistent mit den vorherigen Inhalt der Datenbasis sein, nach einfachen, festen Regeln. Wenn eine Datenbasis als korrekt verifiziert wurde,[3] dann ist der mathematische Inhalt korrekt, wenn der Verifizierer korrekt ist und die Axiome korrekt sind. Das Verifikationsprogramm könnte zwar prinzipiell falsch sein, der verwendete Verifikationsalgorithmus ist jedoch relativ einfach, so dass es unwahrscheinlich ist, dass er in einem Metamath-Programm falsch implementiert wird. Außerdem gibt es mehr als ein Dutzend Verifizierer für die Metamath-Datenbasen, geschrieben von verschiedenen Programmierern in unterschiedlichen Programmiersprachen, so dass diese unterschiedlichen Verifizierer als unabhängige Prüfer einer Datenbasis fungieren können. Die meistgenutzte Metamath-Datenbasis, der Metamath Proof Explorer (auch bekannt als `set.mm`), wird derzeit von vier verschiedenen Metamath-Verifikationsprogrammen verifiziert, die von vier unterschiedlichen Personen in vier verschiedenen Programmiersprachen geschrieben wurden, einschließlich des originalen Metamath-Programms, das in diesem Buch beschrieben wird. Deshalb könnten die einzig möglichen „Fehler“ in der Formulierung der Axiome liegen, zum Beispiel, wenn die Axiome inkonsistent sind (ein berühmtes Problem, das durch den Gödelschen Unvollständigkeitssatz als unlösbar aufgezeigt wurde). Reale mathematische Systeme haben jedoch nur sehr wenige Axiome, und diese können sorgfältig studiert werden. All dies bietet eine außerordentlich hohe Sicherheit, dass die geprüfte Datenbasis in der Tat korrekt ist.

Das Metamath-Programm beweist Theoreme nicht automatisch, sondern ist darauf ausgelegt, Beweise zu überprüfen, die ihm bereitgestellt wer-

[3] Dies beinhaltet die Verifikation, dass eine sequentielle Liste von Beweisschritten zu dem angegebenen Theorem führt.

den. Die Metamath zugrunde liegende Sprache ist völlig allgemein und hat keine eingebaute, vorgefasste Vorstellung über Ihr formales System, seine Logik oder seine Syntax. Für die Konstruktion von Beweisen verfügt das Metamath-Programm über einen Beweis-Assistenten, der Sie beim Ausfüllen einiger Details eines Beweisschritts unterstützt, und Ihnen die Möglichkeiten bei jedem Schritt zeigt. Weiterhin verifiziert er den Beweis, während Sie ihn erstellen; den Beweis führen müssen Sie aber immer noch selbst.

Es gibt viele andere Programme, die Informationen in der Metamath-Sprache verarbeiten oder erzeugen können, und es werden immer mehr geschrieben. Das liegt zum Teil daran, dass die Metamath-Sprache selbst sehr einfach ist und mit Absicht leicht automatisch verarbeitet werden kann. Einige Programme, wie z. B. `mmj2`, enthalten einen Beweis-Assistenten, der einige Beweisschritte automatisiert erstellen kann, die über das hinausgehen, was das Metamath-Programm kann. Mario Carneiro hat einen Algorithmus entwickelt, der Beweise im OpenTheory-Austauschformat, welches aus und in jede einzelne Beweissprache der HOL-Familie (HOL4, HOL Light, Proof-Power, und Isabelle) übersetzt werden kann, in die Metamath Sprache [10] überführt. Daniel Whalen hat Holophrasm entwickelt, das automatisch viele Metamath-Beweise mithilfe von Ansätzen des maschinellen Lernens (einschließlich mehrerer neuronaler Netze) beweisen kann[72]. Eine ausführliche Besprechung dieser anderen Programme würde jedoch den Rahmen dieses Buches sprengen.

Wie die meisten Computersprachen verwendet die Metamath-Sprache den Standardzeichensatz (ASCII), der auf jeder Computertastatur zur Verfügung steht. Daher kann sie viele der speziellen Symbole, die Mathematiker verwenden, nicht direkt darstellen. Eine nützliche Eigenschaft des Metamath-Programms ist sein Fähigkeit, die von ihm verwendete Notation in die Textsatzsprache LaTeX umzuwandeln. Mit dieser Funktion können Sie die von Ihnen definierten ASCII-Tokens in standardmäßig verwendete mathematische Symbole umwandeln, so dass Sie am Ende Symbole und Formeln erhalten, mit denen Sie vertraut sind, anstelle der etwas kryptischen ASCII-Darstellungen davon. Das Metamath-Programm kann auch HTML generieren, was die Veröffentlichung von Ergebnissen im Internet vereinfacht und die Bereitstellung weiterer Informationen zu einem Thema über Hypertext-Links ermöglicht.

Metamath ist wahrscheinlich konzeptionell anders als alles, was Sie bisher gesehen haben, und einige Aspekte sind vielleicht etwas gewöhnungsbedürftig. Dieses Buch wird Ihnen bei der Entscheidung helfen, ob Metamath Ihren speziellen Bedürfnissen entspricht.

Was Sie erwartet

Es ist wichtig, dass Sie verstehen, was Metamath ist und was nicht. Wie bereits erwähnt, ist das Metamath-Programm *kein* automatischer Theorembeweiser, sondern vielmehr ein Beweisprüfer. Die Entwicklung einer Datenbasis

kann eine langwierige, harte Arbeit sein, vor allem wenn man die Beweise so kurz wie möglich halten möchte. Aber es wird einfacher, wenn man bereits eine Sammlung nützlicher Theoreme aufgebaut hat. Der Zweck von Metamath ist es einfach, die bestehende Mathematik in einer absolut strengen, durch Computer überprüfbare Weise zu dokumentieren, nicht um direkt bei der Schaffung neuer mathematischer Sätze oder Erkenntnisse zu helfen. Es ist auch keine magische Lösung für das Erlernen abstrakter Mathematik, obwohl es hilfreich sein kann, die implizite Strenge hinter dem zu sehen, was man aus den Lehrbüchern lernt. Außerdem erhält man Hinweise um Beweise auszuarbeiten, bei denen man sonst nicht weiterkommt.

Bis zum Zeitpunkt der Erstellung dieses Buches wurde bereits eine umfangreiche Datenbasis für die Mengenlehre entwickelt, die eine Grundlage für viele Bereiche der Mathematik bietet. Aber es ist noch viel mehr Arbeit nötig, um nützliche Datenbasen für andere Bereiche zu entwickeln.

Metamath „kennt keine Mathematik"; es bietet lediglich Rahmenbedingungen zur Formulierung von Mathematik. Sein Sprachumfang ist sehr klein: man kann zwei Arten von Symbolen definieren, nämlich Konstanten und Variablen. Das Einzige, was Metamath beherrscht, ist das Ersetzen von Symbolen durch Zeichenketten für die Variablen in einem Ausdruck, der auf Anweisungen basiert, die man in einem Beweis angibt, vorbehaltlich bestimmter Einschränkungen, die man für die Variablen festlegt. Sogar die dezimale Darstellung einer Zahl ist lediglich eine Folge von bestimmten Konstanten (Ziffern), die zusammengenommen einem beliebigen mathematischen Objekt entsprechen, das man für sie in einem bestimmten Kontext definiert. Im Gegensatz zu anderen Computersprachen wird tatsächlich keine Zahl explizit im Computer gespeichert. In einem Beweis gibt man Metamath vor, welche Symbol-Substitutionen in vorherigen Axiomen oder Theoremen vorzunehmen sind, und fügt eine Folge von solchen Substitutionen zusammen, um das gewünschte Theorem zu erhalten. Diese Art der Symbolmanipulation erfasst das Wesentliche der Mathematik auf einer präaxiomatischen Ebene.

Metamath und mathematische Literatur

In der Literatur für höhere Mathematik werden die Beweise gewöhnlich in Form von kurzen Skizzen dargestellt, denen oft nur ein Experte folgen kann. Dies resultiert zum Teil aus dem Wunsch nach Kürze, aber es wäre auch unklug (selbst wenn es praktisch möglich wäre) Beweise mit allen formalen Details zu präsentieren, da das Gesamtbild verloren gehen würde.

Eine Lösung, so wie ich sie mir vorstelle, besteht aus einer Kombination des traditionellen kurzen, informellen Beweises in gedruckter Form, begleitet von einem vollständigen formalen Beweis, der in einer Computerdatenbasis gespeichert ist. Dies würde es ermöglichen, dass die Mathematik für den Experten akzeptabel bleibt, aber auch dem Nichtspezialisten zugänglich ist. In Analogie zu einem Computerprogramm kann man den informellen Beweis als Pseudocode ansehen, der die übergreifenden Schlussfolgerungen und den

Inhalt des Beweises beschreibt, während die Computerdatenbasis mit dem tatsächlichen Programmcode verglichen werden kann, die jedem, auch einem Laien, die Möglichkeit bietet, den Beweis so detailliert wie gewünscht nachzuvollziehen, indem man in immer tiefere Schichten von Theoremen bis hin zu den Axiomen der Theorie vordringt (wie bei Unterprogrammen, die wiederum andere Unterprogramme aufrufen). Darüber hinaus hätte die Computerdatenbasis den Vorteil, dass sie die absolute Sicherheit bietet, dass der Beweis korrekt ist, da jeder Schritt automatisch überprüft werden kann.

Neben Metamath gibt es mehrere andere Ansätze für ein Projekt wie dieses. Abschnitt 1.3.4 erörtert einige davon.

Ein hehres Ziel wäre für uns eine Datenbasis mit Hunderttausenden von Theoremen und ihren durch Computer überprüfbaren Beweisen, die einen bedeutenden Teil der bekannten Mathematik umfasst und für den sofortigen Zugriff zur Verfügung stellt. Diese würden von mehreren unabhängig voneinander implementierten Verifizierern vollständig geprüft werden, um ein extrem hohes Maß an Vertrauen in die vollständige Korrektheit der Beweise herzustellen. Die Datenbasis würde es allen ermöglichen, jegliche interessierende Details zu untersuchen, so dass man jeden gewünschten Teil eines Beweises bestätigen kann. Ob Metamath die richtige Wahl ist, bleibt abzuwarten, aber im Prinzip glauben wir, dass sie hinreichend angemessen ist.

Formalismus

In den letzten fünfzig Jahren hat eine Gruppe französischer Mathematiker, die unter dem Pseudonym Bourbaki zusammenarbeiten, eine Reihe von Monographien verfasst, die versuchen, große Teile der Mathematik konsequent von den Grundlagen her zu formalisieren. Einerseits hat ein solcher Versuch sicherlich seine Vorzüge; andererseits ist das Bourbaki-Projekt wegen seiner „Scholastik“ und „Hyperaxiomatik“, welche die intuitiven, zu den Ergebnissen führenden Schritte verbergen, kritisiert worden [3, S. 191].

Metamath treibt diese Philosophie ungeniert auf die Spitze und ist zweifellos der gleichen Art von Kritik ausgesetzt. Nichtsdestotrotz denke ich, dass in Verbindung mit konventionellen Ansätzen der Mathematik Metamath einen nützlichen Zweck erfüllen kann. Der Ansatz von Bourbaki ist im Wesentlichen pädagogisch und verlangt vom Leser, sich mit jedem Detail in einer sehr großen Hierarchie vertraut zu machen, bevor er oder sie zum nächsten Schritt übergehen kann. Der Unterschied zu Metamath besteht darin, dass der „Leser“ (Benutzer) weiß, dass alle Details in seiner Computerdatenbasis enthalten sind, die bei Bedarf abgerufen werden können; es wird nicht verlangt, dass der Benutzer alles weiß, sondern es werden ihm komfortabel die Teile zur Verfügung gestellt, die von Interesse sind. Da der Umfang des gesamten mathematischen Wissens immer größer wird, kann kein Einzelner seine Gesamtheit in voller Tiefe überblicken. Metamath kann alle Fragen über die Gültigkeit eines beliebigen Teils des Wissens abschließend klären und Zweifel ausräumen, und kann im Prinzip jeden Teil davon

für einen Nicht-Spezialisten zugänglich machen.

Eine persönliche Anmerkung

Warum habe ich Metamath entwickelt? Ich mag abstrakte Mathematik, aber manchmal verirre ich mich in einer Flut von Definitionen und verliere das Vertrauen in die Korrektheit meiner Beweise. Oder ich erreiche einen Punkt, an dem ich aus den Augen verliere, wie alles, was ich tue, mit den Axiomen zusammenhängt, auf denen eine Theorie beruht. Ich habe manchmal den Verdacht, dass unterwegs ein übersehenes implizites Axiom versehentlich eingebracht wurde (wie es historisch mit der euklidischen Geometrie, deren Auslassung des Pasch'schen Axioms für 2000 Jahre unbemerkt blieb [15, p. 160]!). Ich bin auch etwas faul und möchte den Aufwand vermeiden Lücken in informellen Beweisen, die „dem Leser überlassen" werden, erneut selbst nachzuprüfen. Ich ziehe es vor, sie nur einmal herauszufinden und mich nicht ein Jahr später erneut durch dieselbe Frustration zu kämpfen, wenn ich vergesse, was ich getan habe. Metamath bietet eine bessere Möglichkeit zur Wiederherstellung meiner Bemühungen als Papierfetzen, die ich nicht mehr entziffern kann. Aber vor allem finde ich die Idee sehr reizvoll, mathematisches Wissen in einer Computerdatenbasis zu archivieren, die Präzision, Gewissheit und die Eliminierung menschlicher Fehler gewährleistet.

Anmerkung zu Bibliographie und Index

Die Bibliographie enthält in der Regel die Library of Congress-Klassifikation für ein Werk, damit Sie es in einem Regal einer Universitätsbibliothek leichter finden. Der Index enthält Verweise auf Seiten, auf denen die Werke der Autoren zitiert werden, auch wenn die Namen der Autoren vielleicht nicht auf diesen Seiten erscheinen.

Danksagungen

Dank gebührt zunächst meiner Frau Deborah (die am 4. September 1998 verstorben ist), für die Kritik am Manuskript, aber vor allem für ihre Geduld und Unterstützung. Ich möchte auch Joe Wright, Richard Becker, Clarke Evans, Buddha Buck, und Jeremy Henty für hilfreiche Kommentare danken. Etwaige Fehler, Auslassungen und andere Unzulänglichkeiten liegen natürlich in meiner Verantwortung.

Notiz hinzugefügt am 22. Juni 2005

Die ursprüngliche, unveröffentlichte Version dieses Buches wurde 1997 geschrieben und über das Internet verbreitet. Die vorliegende Ausgabe wurde aktualisiert, um das aktuelle Metamath-Programm und die Datenbasen sowie aktuellere URLs für Internet-Seiten wiederzugeben. Dank an Josh Purinton, One Hand Clapping, Mel L. O'Cat und Roy F. Longton für das

Aufzeigen von typografischen und anderen Fehler. Ich habe auch von zahlreichen Diskussionen mit Raph Levien profitiert, der Metamaths Philosophie der Strenge erweitert hat, was in seine Beweissprache *Ghilbert* (`http://ghilbert.org`) resultierte.

Robert (Bob) Solovay teilte ein neues Ergebnis von A. R. D. Mathias über das System von Bourbaki mit, und der Text wurde entsprechend aktualisiert (S. 17).

Bob wies auch auf eine Klärung der Literatur bezüglich der Kategorientheorie und unzugängliche Kardinalzahlen (S. 40) hin, und eine missverständliche Aussage wurde aus dem Text entfernt. Genauer gesagt ist es im Gegensatz zu einer Aussage in früheren Ausgaben möglich, „Es gibt eine eigene Klasse von unzugänglichen Kardinalen“ in der Sprache von ZFC auszudrücken. Dies lässt sich wie folgt bewerkstelligen: „Für jede Menge x gibt es eine unzugängliche Kardinalzahl κ, so dass κ nicht in x liegt“. Bob schreibt:[4]

> Dieses Axiom ist die Art und Weise, wie Grothendieck die Kategorientheorie darstellt. Jedem unzugänglichen Kardinal κ ordnet man ein Grothendieck-Universum $U(\kappa)$ zu. $U(\kappa)$ besteht aus denjenigen Mengen, die in einer transitiven Menge der Kardinalität kleiner als κ liegen. Anstelle der „Kategorie aller Gruppen“ arbeitet man relativ zu einem Universum [unter Berücksichtigung der Kategorie der Gruppen mit Kardinalität kleiner als κ]. Nun ist die Kategorie, deren Objekte alle Kategorien „relativ zum Universum $U(\kappa)$“ sind, eine Kategorie nicht relativ zu diesem Universum, sondern zum nächsten Universum.
>
> All die Dinge, die Kategorientheoretiker gerne tun, können in diesem Rahmen getan werden. Der einzige strittige Punkt ist, ob das Grothen-Dieck-Axiom für die Bedürfnisse der Kategorientheoretiker zu stark ist. Mac Lane argumentiert, dass „ein Universum ausreicht“ und Feferman hat argumentiert, dass man mit der gewöhnlichen ZFC auskommen kann. Ich finde die Argumente von Feferman nicht überzeugend. Mac Lane mag recht haben, aber wenn ich über Kategorientheorie nachdenke, tue ich das à la Grothendieck.
>
> Übrigens fügt Mizar das Axiom „Es gibt eine eigene Klasse von Unzugänglichkeiten“ hinzu, genau um Kategorientheorie zu betreiben.

Die aktuellsten Informationen über das Metamath-Programm und die Datenbasen sind immer unter `http://metamath.org` zu finden.

[4] Private Korrespondenz, 30. November 2002.

Notiz hinzugefügt am 24. Juni 2006

Die Metamath-Spezifikation wurde leicht eingeschränkt, um das Schreiben von Parsern zu erleichtern. Siehe die Fußnote auf S. 129.

Notiz hinzugefügt 10 März 2007

Ich bin Anthony Williams dankbar für das Schreiben des LaTeX-Pakets namens `realref.sty` und dafür, dass er es der Public Domain zur Verfügung gestellt hat. Mit diesem Paket können die internen Hyperlinks in einer PDF-Datei auf bestimmte Seitenzahlen verankert werden, anstatt nur auf Abschnittsüberschriften, was die Navigation in der PDF-Datei für dieses Buch viel angenehmer und „logischer“ macht.

Ein von Martin Kiselkov gefundener Druckfehler wurde korrigiert. Eine verwirrende Bemerkung über die Vereinheitlichung wurde auf Anregung von Mel O'Cat entfernt.

Notiz hinzugefügt am 27. Mai 2009

Mehrere von Kim Sparre gefundene Tippfehler wurden korrigiert. Es wurde ein Hinweis hinzugefügt, dass die Poincar'e-Vermutung bewiesen wurde (S. 27).

Notiz hinzugefügt am 17 Nov. 2014

Die Aussage des Schröder-Bernstein-Theorems in Abschnitt 1.2.2 wurde korrigiert. Dank an Bob Solovay für den Hinweis auf den Fehler.

Notiz hinzugefügt am 25. Mai 2016

Dank an Jerry James für die Korrektur von 16 Tippfehlern.

Notiz hinzugefügt Februar 25, 2019

David A. Wheeler hat, in Zusammenarbeit mit mir, eine große Anzahl von Verbesserungen und Aktualisierungen vorgenommen. Die Axiome der Prädikatenlogik wurden neu nummeriert, und der Text macht nun deutlich, dass sie auf Tarskis System S2 beruhen; die einzige geringfügige Abweichung im Axiom ax-6 wird erklärt und begründet. Die Axiome für reelle und komplexe Zahlen wurden geändert, damit sie mit `set.mm` konsistent sind. Lang erwartete Änderungen der Spezifikation „1–8“ wurden vorgenommen, was zur Klärung von zuvor mehrdeutigen Punkten führte. Einige Fehler im Text, welche die `$f`- und `$d`-Anweisungen betreffen, wurden korrigiert (die Spezifikation war korrekt, aber die Erklärungen im Buch widersprachen versehentlich der Spezifikation). Wir haben jetzt ein System zur automatischen Erzeugung von schmalen PDFs, damit alle, die ein Smartphone besitzen, einen einfachen Zugang zur aktuellen Version des Dokuments haben.

Ein neuer Abschnitt über Deduktion wurde hinzugefügt; er behandelt das Standard-Deduktionstheorem, das Theorem der schwachen Deduktion, den Deduktionsstil und die natürliche Deduktion. Viele kleinere Korrekturen (zu zahlreich, um sie hier aufzulisten) wurden ebenfalls vorgenommen.

Notiz hinzugefügt am 7. März 2019

Eine Beschreibung der Metamath-Syntax in erweiterter Backus–Naur-Form (EBNF) wurde im Anhang E ergänzt, eine kurze Erklärung über Typcodes hinzugefügt, weitere Beispiele im Abschnitt über Deduktion eingefügt, und eine Vielzahl kleinerer Verbesserungen durchgeführt.

Notiz hinzugefügt am 7. April 2019

In dieser Version des Buches wird die Notation für die „echte Substitution“ geklärt, die Erläuterungen zu dem Theorem der schwachen Deduktion und zu der natürlichen Deduktion verbessert, der Befehl `undo` dokumentiert, die Informationen zu `write source` aktualisiert, der Typcode von `set` in `setvar` geändert, um mit der aktuellen Version von `set.mm` übereinzustimmen, weitere Erläuterungen über Kommentarauszeichnungen hinzugefügt (z.B. wurde dokumentiert, wie man Überschriften erstellt), und die Unterschiede zwischen den verschiedenen Behauptungsformen (insbesondere der Ableitungsform) klargestellt.

Notiz hinzugefügt am 2. Juni 2019

Diese Version behebt eine große Anzahl kleinerer Probleme, die von Benoît Jubin gemeldet wurden, wie z.B. redaktionelle Probleme und die Notwendigkeit, `verify markup` zu dokumentieren (Vielen Dank!). Außerdem enthält diese Version nun konkrete Beispiele für die Formen von Theoremen (Deduktionsform, Inferenzform und geschlossene Form). Wir nennen diese Version die „zweite Auflage“; die vorherige Ausgabe, die 2007 offiziell veröffentlicht wurde, hatte einen etwas anderen Titel (*Metamath: Eine Computersprache für die reine Mathematik*).

Kapitel 1

Einleitung

> I.M.: *Nein, nein. Da ist nichts Subjektives dran! Jeder weiß, was ein Beweis ist. Lesen Sie einfach ein paar Bücher, besuchen Sie Kurse bei einem kompetenten Mathematiker, und Sie werden es verstehen.*
>
> Schüler: *Sind Sie sicher?*
>
> I.M.: *Nun - es ist möglich, dass Sie es nicht verstehen, wenn Sie keine Begabung dafür haben. Das kann auch passieren.*
>
> Schüler: *Dann entscheiden Sie, was ein Beweis ist, und wenn ich nicht lerne auf dieselbe Art und Weise zu entscheiden, dann bestimmen Sie, dass ich keine Begabung habe.*
>
> I.M.: *Wenn nicht ich, wer dann?*
>
> „Der ideale Mathematiker“ [1]

Brillante Mathematiker haben nahezu unvorstellbar tiefgreifende Ergebnisse erzielt, die zu den krönenden intellektuellen Errungenschaften der Menschheit zählen. Allerdings hinkt die moderne abstrakte Mathematik in gewisser Weise der Zeit hinterher, und ist in einer Ära vor der Existenz von Computern stecken geblieben. Zwar bestreitet niemand die bemerkenswerten Ergebnisse, die erzielt wurden. Jedoch ist es praktisch unmöglich, diese Ergebnisse einem Uneingeweihten in präziser Weise zu vermitteln. Um diese Ergebnisse zu beschreiben, wird eine knappe, informelle Sprache verwendet, die trotz ihrer Eleganz sehr schwer zu erlernen ist. Diese informelle Sprache ist nicht unpräzise, ganz im Gegenteil, jedoch werden Details häufig einfach ausgelassen, und es werden Symbole mit verborgenem Kontext verwendet, die implizit von einem Experten verstanden werden, aber nur von wenigen anderen. Äußerst komplexe technische Bedeutungen werden verbunden mit harmlos klingenden Wörtern wie „kompakt“ und „messbar“, die kaum einen

[1] Frei übersetzt nach [15], „The Ideal Mathematician“ S. 40.

Hinweis darauf geben, was eigentlich damit ausgesagt wird. Wer sich die genaue technische Bedeutung nicht ständig vor Augen hält scheitert, und die Fähigkeit dazu kann nur durch viel Übung und harte Arbeit erworben werden. Nur die wenigen, welche diese notwendige, schmerzhafte Lernerfahrung machen, können der kleinen, geschlossenen Gruppe der reinen Mathematiker beitreten. Die informelle Sprache schottet die wahre Natur ihres Wissens von den meisten anderen ab.

Metamath macht abstrakte Mathematik konkreter. Es ermöglicht einem Computer, die mit jedem Wort oder Symbol verbundene Komplexität mit absoluter Strenge zu erfassen. Man kann diese Komplexität in aller Ruhe erforschen, und zwar bis zum gewünschten Detaillierungsgrad. Ob Sie nun glauben, dass Konzepte wie Unendlichkeit tatsächlich außerhalb des Verstandes „existieren“ oder nicht, mit Metamath können Sie das ergründen, was damit tatsächlich ausgesagt werden soll.

Metamath ermöglicht auch eine völlig rigorose und gründliche Überprüfung von Beweisen. Seine Sprache ist so einfach, dass man sich nicht auf die Autorität von Experten verlassen muss, sondern die Ergebnisse Schritt für Schritt selbst überprüfen kann. Wenn Sie versuchen wollen, Ihre eigenen Ergebnisse abzuleiten, lässt Metamath Sie keinen Denkfehler machen. Auch professionelle Mathematiker machen Fehler; Metamath dagegen ermöglicht es, die Korrektheit von Beweisen gründlich zu überprüfen.

Metamath ist eine Computersprache und ein zugehöriges Computerprogramm für das Archivieren, Überprüfen und Studieren mathematischer Beweise auf einer sehr detaillierten Ebene. Mit der Metamath-Sprache können formale mathematische Systeme beschrieben und Beweise für Theoreme in diesen Systemen formuliert werden. Eine solche Sprache wird von Mathematikern als Metasprache bezeichnet. Das Metamath-Programm ist ein Computerprogramm zur Überprüfung von Beweisen, die in der Metamath-Sprache geschrieben sind. Das Metamath-Programm verfügt nicht über die eingebaute Fähigkeit, logische Schlüsse zu ziehen; es führt lediglich eine Reihe von Ersetzungen von Symbolen gemäß den Anweisungen durch, die ihm in einem Beweis gegeben werden, und prüft, ob das Ergebnis mit dem erwarteten Theorem übereinstimmt. Es macht logische Schlussfolgerungen nur auf der Grundlage von Regeln der Logik, welche in einer Menge von Axiomen oder ersten Prinzipien enthalten sind, die ihm als Ausgangspunkt für Beweise vorgegeben werden.

Die vollständige Spezifikation der Metamath-Sprache ist nur vier Seiten lang (Abschnitt 4.1, S. 126). Ihre Einfachheit mag Sie zunächst fragen lassen, was man damit überhaupt erreichen kann. Aber in der Tat sind die verwendeten Symbolmanipulationen diejenigen, die in allen mathematischen Systemen auf der untersten Ebene implizit durchgeführt werden. Man kann sie relativ schnell lernen und volles Vertrauen in jeden mathematischen Beweis haben, den Metamath verifiziert. Andererseits ist die Metamath-Sprache leistungsfähig und allgemein genug, um mit ihr praktisch jede mathemati-

sche Theorie, von der einfachsten bis zur abstraktesten, zu beschreiben.

Obwohl Metamath im Prinzip für jede Art von Mathematik verwendet werden kann, ist es am besten für abstrakte oder „reine“ Mathematik geeignet, die sich hauptsächlich mit Theoremen und deren Beweisen befasst - im Gegensatz zu der Art von Mathematik, die sich mit der praktischen Handhabung von Zahlen beschäftigt. Beispiele für Teilgebiete der reinen Mathematik sind die Logik[2], Mengenlehre[3], Zahlentheorie[4], Gruppentheorie[5], abstrakte Algebra[6], Analysis[7] und Topologie[8]. Auch in der Physik könnte Metamath auf bestimmte Zweige angewandt werden, die sich der abstrakten Mathematik, wie z.B. der Quantenlogik (die zur Untersuchung von Aspekten der Quantenmechanik genutzt wird), bedienen.

Andererseits ist Metamath weniger geeignet für Anwendungen, die sich hauptsächlich mit intensiven numerischen Berechnungen befassen. Metamath hat keine eingebaute Darstellung von Zahlen; stattdessen muss eine bestimmte Folge von Symbolen (Ziffern) syntaktisch als Teil eines Beweises konstruiert werden, in dem eine gewöhnliche Zahl verwendet wird. Aus diesem Grund sind Zahlen in Metamath am besten auf spezifische Konstanten beschränkt, die im Verlauf eines Theorems oder seines Beweises auftauchen. Zahlen sind nur ein winziger Teil der Welt der abstrakten Mathematik. Der Ausschluss von eingebauten Zahlen war eine bewusste Entscheidung, um die Einfachheit von Metamath sicherzustellen. Es gibt andere Software-Tools für andere mathematische Fragestellungen. Wenn Sie schnell algebraische Probleme lösen möchten, sind die Computeralgebrasysteme Macsyma, Mathematica und Maple besonders gut geeignet, um mit Zahlen und Algebra effizient umzugehen. Wenn Sie einfach nur numerische Ausdrücke oder Matrixausdrücke bequem berechnen möchten, dürften Tools wie Octave eine

[2]Logik ist die Lehre von den Aussagen, die unabhängig von den Objekten, die sie beschreiben, universell wahr sind. Ein Beispiel ist die Aussage: „Wenn P Q impliziert, dann ist entweder P falsch oder Q wahr.“

[3]Die Mengenlehre ist die Lehre von allgemeinen mathematischen Objekten, den so genannten „Mengen“. Von ihr lässt sich im Wesentlichen die gesamte Mathematik ableiten. Zahlen können beispielsweise als spezifische Mengen definiert werden, und ihre Eigenschaften können mit den Werkzeugen der Mengenlehre erforscht werden.

[4]Die Zahlentheorie befasst sich mit den Eigenschaften von positiven und negativen ganzen Zahlen.

[5]Die Gruppentheorie untersucht die Eigenschaften von mathematischen Objekten, die als „Gruppen“ bezeichnet werden: Sie gehorchen einem einfachen Satz von Axiomen und haben Symmetrieeigenschaften, die sie für viele andere Bereiche nützlich machen.

[6]Abstrakte Algebra umfasst die Gruppentheorie und untersucht auch Gruppen mit zusätzlichen Eigenschaften, die sie als „Ringe“ und „Körper“ qualifizieren. Die Menge der reellen Zahlen ist ein bekanntes Beispiel für einen Körper.

[7]Analysis ist die Lehre von den reellen und komplexen Zahlen.

[8]Ein Bereich, der von der Topologie untersucht wird, sind Eigenschaften geometrischer Objekte, die unverändert bleiben, wenn sie Verformungen unterzogen werden. Zum Beispiel haben ein Donut und eine Kaffeetasse jeweils ein Loch (die Tasse hat ein Loch im Henkel) und werden daher als topologisch äquivalent betrachtet. Im Allgemeinen ist die Topologie jedoch die Lehre von abstrakten mathematischen Objekten, die einer bestimmten (erstaunlich einfachen) Reihe von Axiomen gehorchen. Siehe z. B. Munkres [48].

bessere Wahl sein.

Nach dem Erlernen der grundlegenden Anweisungstypen von Metamath sollte jeder technisch versierte Mensch, ob Mathematiker oder nicht, sofort in der Lage sein, jedes in der Metamath-Sprache bewiesene Theorem so weit wie gewünscht zurückzuverfolgen - bis hin zu den Axiomen, auf denen das Theorem beruht. Diese Möglichkeit erlaubt eine nicht-traditionelle Art und Weise des Lernens der reine Mathematik. In Verbindung mit traditionellen Methoden könnte Metamath die reine Mathematik für Menschen zugänglich machen, die nicht ausreichend qualifiziert sind, um die impliziten Details in gewöhnlichen Lehrbuchbeweisen zu verstehen. Sobald man die Axiome einer Theorie kennt, kann man sich darauf verlassen, dass alles vorhanden ist, was man zum Verstehen eines vorliegenden Beweises braucht. Somit kann man sich auf jeden Beweisschritt konzentrieren, den man nicht versteht - und zwar so tief wie nötig, ohne sich Sorgen machen zu müssen, dass man bei einem Schritt, den man sich nicht erklären kann, nicht mehr weiter kommt.[9]

Metamath ist wahrscheinlich anders als alles, was Ihnen bisher begegnet ist. In diesem ersten Kapitel werden wir uns mit der Philosophie und dem Einsatz von Computern in der Mathematik beschäftigen, um die Motivation hinter Metamath besser zu verstehen. Das Material in diesem Kapitel ist nicht erforderlich, um Metamath zu benutzen. Sie können es überspringen, wenn Sie ungeduldig sind, aber ich hoffe, Sie werden es lehrreich und unterhaltsam finden. Wenn Sie gleich mit dem Experimentieren mit dem Metamath-Programm beginnen wollen, gehen Sie direkt zu Kapitel 2 (S. 43). Um die Metamath-Sprache zu lernen, überfliegen Sie Kapitel 2 und fahren direkt fort mit Kapitel 4 (S. 125).

1.1 Mathematik als eine Computersprache

> *Das Studium der Mathematik beginnt oft mit einer Enttäuschung.*
> ...
> *Uns wird gesagt, dass mit ihrer Hilfe die Sterne gewogen und die Milliarden von Molekülen in einem Wassertropfen gezählt werden. Doch wie der Geist von Hamlets Vater entzieht sich diese große Wissenschaft den Bemühungen unserer geistigen Waffen, sie zu begreifen.*

[9] Andererseits ist das Schreiben von Beweisen in der Metamath-Sprache anspruchsvoll und erfordert einen Grad an Strenge, der weit über das hinausgeht, was Schülern oder Studenten normalerweise beigebracht wird. Ich bezweifle, dass im Mathematikunterricht das Schreiben von Metamath-Beweisen jemals die traditionellen Hausaufgaben mit informellen Beweisen ersetzen wird, denn aufgrund der Zeit, die für die Ausarbeitung der Details benötigt wird, könnten im Unterricht nur wenige Themen behandelt werden. Schülern mit Schwierigkeiten, die implizite Strenge im vorliegenden, traditionell verfassten Material zu verstehen, kann das Schreiben einiger einfacher Beweise in der Metamath-Sprache jedoch helfen, ungenaue Gedankengänge zu klären. Obwohl es anfangs etwas schwierig ist, macht dies aufgrund der sofortigen Rückmeldung durch den Computer sogar Spaß, wie das Lösen eines Rätsels.

Alfred North Whitehead[10]

1.1.1 Ist die Mathematik „benutzerfreundlich“?

Angenommen, Sie haben keine formale Ausbildung in abstrakter Mathematik erhalten. Aber populäre Bücher, die Sie gelesen haben, bieten verlockende Einblicke in diese Welt voller tiefgründiger Ideen, die den menschlichen Geist aufgewühlt haben. Sie sind nicht zufrieden mit den informellen, verwässerten Beschreibungen, die Sie gelesen haben, sondern halten es für wichtig, die zugrundeliegende Mathematik selbst zu begreifen, um ihre wahre Bedeutung zu verstehen. Es ist aber nicht sinnvoll, wieder zur Schule zu gehen, um sie zu lernen; Sie wollen nicht Jahre Ihres Lebens damit verbringen. Es gibt viele wichtige Dinge im Leben, und man muss Prioritäten für das setzen, was einem wichtig ist. Was würde passieren, wenn Sie versuchen würden, dieses Vorhaben allein in Ihrer Freizeit zu verfolgen?

Immerhin waren Sie in der Lage, eine Computer-Programmiersprache wie Pascal ohne große Schwierigkeiten selbst zu erlernen, obwohl Sie keine formale Ausbildung für Computer hatten. Sie behaupten nicht, ein Experte für Software-Design zu sein, aber Sie können ein passables Programm schreiben, das Ihren Bedürfnissen entspricht. Wichtiger ist sogar noch, dass Sie wissen, dass Sie sich das Pascal-Programm eines anderen ansehen können, egal wie komplex es ist, und mit genügend Geduld herausfinden können, wie es genau funktioniert, auch wenn Sie kein Spezialist sind. Mit Pascal können Sie alles tun, was ein Computer tun kann, zumindest im Prinzip. Sie wissen also, dass Sie die Fähigkeit haben, im Prinzip alles zu tun, was ein Computerprogramm tun kann: Sie müssen es nur in genügend kleine Stücke zerlegen.

Das folgende imaginäre Szenario könnte eintreten, wenn Sie sich unbedarft der gleichen Sichtweise der abstrakten Mathematik annehmen und versuchen würden, sie selbst zu lernen. Und zwar in einem Zeitraum, der vergleichbar ist mit dem Erlernen einer Programmiersprache für Computer.

Die Suche eines Nicht-Mathematikers nach der Wahrheit

> *... meine Töchter studieren schon seit mehreren Semestern (Chemie) und meinen, in der Schule Differential- und Integralrechnung gelernt zu haben, aber wissen doch bis heute nicht, warum* $x \cdot y = y \cdot x$ *gilt.*
>
> Edmund Landau[11]

Minus mal minus ergibt plus,
warum das so ist müssen wir nicht diskutieren.

[10] Frei übersetzt nach [73], Kap. 1.
[11] Frei übersetzt nach [35], S. vi.

W. H. Auden[12]

Nehmen wir an, Sie sind ein technisch orientierter Fachmann, vielleicht ein Ingenieur, ein Computerprogrammierer oder ein Physiker, aber eben kein Mathematiker. Sie halten sich für einigermaßen intelligent. Sie waren gut in der Schule und lernten eine Vielzahl von Methoden und Techniken der praktischen Mathematik, wie z. B. der Infinitesimalrechnung und das Lösen von Differentialgleichungen. Aber in Ihrem Unterricht ging es selten um moderne abstrakte Mathematik, und Beweise tauchten nur gelegentlich in Ihren Lehrbüchern auf - eine Art notwendiges Übel, das Sie von einem bestimmten Schlüsselergebnis überzeugen sollte. Die meisten Ihrer Hausaufgaben bestanden aus Übungen, in denen die Techniken geübt wurden, und Sie wurden kaum jemals aufgefordert, einen eigenen Beweis zu erstellen.

Sie sind neugierig auf fortgeschrittene, abstrakte Mathematik. Sie sind von der inneren Überzeugung getrieben, dass es wichtig ist, einige der tiefgreifendsten Erkenntnisse der Menschheit zu verstehen und zu schätzen. Aber es scheint sehr schwer erlernbar zu sein, etwas, das nur bestimmte begabte Fachidioten begreifen und verstehen können. Sie sind frustriert, dass Ihnen solche Erkenntnisse scheinbar für immer verschlossen bleiben.

Schließlich treibt Sie Ihre Neugierde dazu, etwas dagegen zu tun. Sie setzen sich das Ziel, Mathematik „wirklich“ zu verstehen: nicht nur wie man Gleichungen in Algebra oder Infinitesimalrechnung nach Kochbuchregeln manipuliert, sondern vielmehr ein tiefes Verständnis dafür zu erlangen, woher diese Regeln kommen. In Wirklichkeit geht es Ihnen gar nicht um diese Art von gewöhnlicher Mathematik, sondern über einen viel abstrakteren, feinstofflichen Bereich der reinen Mathematik, zu denen berühmte Ergebnisse wie der Gödelsche Unvollständigkeitssatz und Cantors verschiedene Arten von Unendlichkeiten gehören.

Sie haben wahrscheinlich eine Reihe populärer Bücher mit Titeln wie *Infinity and the Mind* [58] zu Themen wie diesen gelesen. Sie fanden sie inspirierend, aber gleichzeitig auch etwas unbefriedigend. Sie gaben Ihnen eine allgemeine Vorstellung davon, worum es bei diesen Ergebnissen geht, aber wenn jemand Sie bitten würde, sie zu beweisen, wüssten Sie nicht, wo man anfangen soll. Sicher, Sie könnten denselben allgemeinen Überblick geben, den Sie aus den populären Büchern gelernt haben, und in gewisser Weise haben Sie auch ein Verständnis. Aber tief in Ihrem Inneren wissen Sie, dass eine gewisse Strenge fehlt, dass es auf dem Weg dorthin wahrscheinlich viele subtile Schritte und Fallstricke gibt, und dass man sich letztlich auf die Experten auf diesem Fachgebiet verlassen muss. Das gefällt Ihnen nicht; Sie möchten diese Ergebnisse selbst überprüfen können.

Was tun Sie also als Nächstes? Als ersten Schritt beschließen Sie, in einigen der Originalarbeiten zu den Sie interessierenden Theoremen nachzuschlagen, oder besser noch, sich einige Standardlehrbücher auf diesem Gebiet zu beschaffen. Sie schlagen ein Theorem nach, das Sie verstehen wollen.

[12] Frei übersetzt nach dem Zitat in [20], S. 64.

Natürlich steht es da, aber es wird mit seltsamen Begriffen und merkwürdigen Symbolen ausgedrückt, die für Sie absolut nichts bedeuten. Es könnte genauso gut in einer Fremdsprache geschrieben sein, die Sie noch nie gesehen haben und deren Symbole Ihnen völlig fremd sind. Sie sehen sich den Beweis an und haben nicht die leiseste Ahnung, was die einzelnen Schritte bedeuten, geschweige denn, wie ein Schritt auf den anderen folgt. Nun, offensichtlich müssen Sie eine Menge lernen, wenn Sie diese Dinge verstehen wollen.

Sie denken, dass Sie es wahrscheinlich verstehen könnten, wenn Sie noch einmal drei bis sechs Jahre zur Uni gehen und ein Mathematikstudium absolvieren. Aber das passt nicht zu Ihrer Karriere und den anderen Dingen in Ihrem Leben und würde keinen praktischen Nutzen bringen. Sie beschließen, einen schnelleren Weg zu suchen. Sie denken sich, Sie gehen einfach zurück zum Anfang, Schritt für Schritt, wie bei einem Computerprogramm, bis Sie es verstehen. Aber Sie stellen schnell fest, dass dies nicht möglich ist, da Sie nicht einmal genug verstehen, um zu wissen, wohin man zurückgehen muss.

Vielleicht ist ein anderer Ansatz angebracht - vielleicht sollte man am Anfang beginnen und sich hocharbeiten. Zuerst lesen Sie die Einleitung des Buches um herauszufinden, was die Voraussetzungen dafür sind. Auf ähnliche Art und Weise verfolgen Sie Ihren Weg durch zwei oder drei weitere Bücher zurück und stoßen schließlich auf eines, das am Anfang zu stehen scheint: Es listet die Axiome der Arithmetik auf. „Aha!“, denken Sie naiv, „Das muss der Ausgangspunkt sein, die Quelle allen mathematischen Wissens“. Oder zumindest der Ausgangspunkt für die Mathematik, die sich mit Zahlen befasst; irgendwo muss man ja anfangen, und man hat keine Ahnung, wo der Ausgangspunkt für eine andere Mathematik sein könnte. Aber das Wort „Axiome“ sieht vielversprechend aus. Also liest man eifrig weiter und arbeitet sich durch einige elementare Übungen am Anfang des Buches durch. Sie fühlen sich vage beunruhigt: Diese scheinen überhaupt nicht wie Axiome zu sein, zumindest nicht in dem Sinne, den Sie sich vorstellen, wenn Sie an Axiome denken. Axiome implizieren einen Ausgangspunkt, von dem aus alles andere nach genauen, im Axiomensystem festgelegten Regeln aufgebaut werden kann. Auch wenn Sie die ersten Beweise auf informelle Weise verstehen können, und in der Lage sind, einige der Übungen zu erledigen, ist es schwer, die Regeln genau auszumachen. Sicher, jeder Schritt scheint logisch aus den anderen zu folgen, aber was bedeutet das genau? Ist die „Logik“ nur eine Frage des gesunden Menschenverstands, etwas Unbestimmtes, das wir alle verstehen, aber nie ganz genau benennen können?

Sie haben einige Jahre - mit Unterbrechungen - damit verbracht Computer zu programmieren, und Sie wissen, dass es bei Computersprachen keine Frage nach den Regeln gibt - sie sind präzise und kristallklar. Wenn Sie sie befolgen, wird Ihr Programm funktionieren, und wenn Sie es nicht tun, wird es nicht funktionieren. Egal wie komplex ein Programm ist, es kann jederzeit in immer einfachere Teile zerlegt werden, bis man schließlich die Bits iden-

tifizieren kann, die herumgeschoben werden, um eine bestimmte Funktion auszuführen. Einige Programme erfordern vielleicht viel Ausdauer, um sie zu schreiben, aber wenn Sie sich auf einen bestimmten Teil konzentrieren, müssen Sie nicht einmal unbedingt wissen, wie der Rest des Programms funktioniert. Sollte es nicht eine Analogie in der Mathematik geben?

Sie beschließen, den ultimativen Test durchzuführen: Sie fragen sich, wie ein Computer überprüfen oder sicherstellen könnte, dass die Schritte in diesen Beweisen aufeinander aufbauen. Sicherlich muss die Mathematik mindestens genauso genau definiert sein wie eine Computersprache, wenn nicht sogar noch präziser; schließlich basiert die Informatik selbst auf ihr. Wenn man einen Computer dazu bringen kann, diese Beweise zu überprüfen, dann sollte man im Grunde ebenfalls in der Lage sein, sie im Prinzip auch selbst kristallklar, in einer präzisen Weise zu verstehen.

Sie werden überrascht sein: Sie können sich keine Vorgehensweise vorstellen, wie Sie die Beweise, die in deutscher oder englischer Sprache abgefasst sind, in eine Form bringen können, die der Computer versteht. Die Beweise sind voll von Sätzen wie „Angenommen, es gibt ein eindeutiges x...“ und „Bei einem beliebigen y sei z eine solche Zahl, dass...“ Das ist nicht die Art von Logik, die man aus der Computerprogrammierung kennt, wo sich alles, sogar die Arithmetik, auf boolesche Einsen und Nullen reduziert, wenn man sie nur ausreichend aufschlüsselt. Auch wenn Sie glauben, dass Sie die Beweise verstehen, scheint eine Art höheres Denken notwendig zu sein und nicht präzise Regeln, die festlegen, wie man die Symbole in den Axiomen manipuliert. Was auch immer es ist, es ist einfach nicht offensichtlich, wie man es einem Computer gegenüber ausdrücken würde, und je mehr man darüber nachdenkt, desto verwirrter wird man. Schließlich gelangt man zu einem Punkt, an dem man sich sogar fragt, ob man es wirklich versteht. Es steckt viel mehr hinter diesen Axiomen der Arithmetik, als man auf den ersten Blick sieht.

Darüber hat in der Schule in den naturwissenschaftlichen Fächern nie jemand gesprochen. Sie haben nur die Regeln gelernt, die man Ihnen vorgab, ohne ganz zu verstehen, wie oder warum sie funktionierten. Manchmal waren Sie vage misstrauisch oder unsicher, und haben durch Hausaufgaben und Übertragung von dem Präsentierten gelernt, wie man Lösungen präsentiert, die den Dozenten zufriedenstellten und Ihnen eine „1“ einbrachten. Selten hat man tatsächlich etwas auf rigorose Weise „bewiesen“, und die Mathe-Studenten, die so etwas taten, schienen in einer anderen Welt zu leben.

Natürlich gibt es Computeralgebra-Programme, die Mathematik betreiben können, und zwar ziemlich beeindruckend. Sie können im Handumdrehen die Integrale lösen, mit denen man in dem Fach Infinitesimalrechnung zu kämpfen hatte, und können noch viel, viel mehr. Aber wenn man diese Programme anschaut, sieht man eine große Sammlung von Algorithmen und Techniken, die im Laufe der Zeit weiterentwickelt und ergänzt wurden, zusammen mit grundlegender Software, die Symbole manipuliert. Je-

der eingebaute Algorithmus ist das Ergebnis eines Theorems, dessen Beweis weggelassen wurde; man muss nur derjenigen Person vertrauen, die ihn bewiesen hat, und der Person, die ihn einprogrammiert hat, und hoffen, dass es keine Bugs gibt. Irgendwie scheint dies nicht die Essenz der Mathematik zu sein. Obwohl Computeralgebrasysteme Theoreme mit erstaunlicher Geschwindigkeit generieren können, können sie nicht ein einziges davon wirklich beweisen.

Nach einigem Grübeln schauen Sie sich einige populäre Bücher darüber an, worum es in der Mathematik geht. Irgendwo liest man, dass die gesamte Mathematik eigentlich von etwas abgeleitet ist, das sich „Mengenlehre“ nennt. Das ist ein wenig verwirrend, denn in dem Buch, in dem die Axiome der Arithmetik vorgestellt werden, wird nirgends die Mengenlehre erwähnt, oder wenn, dann nur als ein Werkzeug, das hilft, Dinge besser zu beschreiben - die Menge der geraden Zahlen und so weiter. Wenn Mengenlehre die Grundlage für die gesamte Mathematik ist, warum werden dann zusätzliche Axiome für die Arithmetik benötigt?

Irgendetwas stimmt nicht, aber Sie sind sich nicht sicher, was. Einer Ihrer Freunde ist ein reiner Mathematiker. Er weiß, dass er nicht in der Lage ist, Ihnen mitzuteilen, womit er sein Geld verdient und er scheint wenig Interesse daran zu haben, es zu versuchen. Sie wissen aber, dass für ihn Beweise das sind, worum es in der Mathematik geht. Sie fragen ihn, was ein Beweis ist, und er sagt Ihnen, dass er zwar auf Logik basiert, aber dass das Beweisen eigentlich etwas ist, das man lernt, indem man es immer und immer wieder macht, bis man es kapiert hat. Er verweist auf ein Buch, *How to Read and Do Proofs* [63]. Obwohl dieses Buch Ihnen hilft, traditionelle informelle Beweise zu verstehen, gibt es immer noch etwas, das Sie noch nicht ganz begreifen können.

Sie fragen Ihren Freund, wie Sie einen Beweis von einem Computer überprüfen lassen würden. Zuerst scheint er über die Frage verwirrt zu sein; warum sollte man das tun? Dann sagt er, dass es keinen Sinn ergeben würde, aber er hat gehört, dass man den Beweis in Tausende oder sogar Millionen von Einzelschritten zerlegen müsste, um so etwas zu tun, weil die damit verbundenen Überlegungen auf einer so hohen Abstraktionsebene liege. Er sagt, dass man so etwas vielleicht bis zu einem gewissen Punkt machen könne, aber dass der Computer völlig unpraktisch wäre, sobald man in eine sinnvolle Mathematik einsteigt. Dort kann man einen Beweis nur noch von Hand verifizieren, und die Fähigkeit dazu kann man nur erwerben, wenn man sich ein paar Jahre lang in der Universität auf das Gebiet spezialisiert. Wie auch immer, er glaubt, dass das alles mit der Mengenlehre zu tun hat, obwohl er nie einen formalen Kurs im Fach Mengentheorie belegt hat, sondern sich das, was er brauchte, einfach nach und nach angeeignet hat.

Sie sind fasziniert und erstaunt. Offenbar kann ein Mathematiker mit einem einzigen Konzept etwas erfassen, für das ein Computer tausend oder eine Million Schritte benötigen würde, um es zu verifizieren, und er hat

volles Vertrauen darin. Jeder einzelne dieser Schritte muss absolut korrekt sein, sonst ist der ganze Beweis sinnlos. Wenn Sie eine Million Zahlen von Hand addieren würden, würden Sie dem Ergebnis vertrauen? Woher wissen Sie wirklich, dass all diese Schritte korrekt sind, dass es in einem dieser Millionen Schritte nicht irgendeinen subtilen Fallstrick gibt, wie einen Bug in einem Computerprogramm? Immerhin haben Sie gelesen, dass berühmte Mathematiker gelegentlich Fehler gemacht haben, und Sie wissen sicherlich, dass Sie bei Ihren Mathehausaufgaben in der Schule auch schon Fehler gemacht haben.

Sie erinnern sich an die Analogie mit einem Computerprogramm. Sicher, Sie können verstehen, was ein großes Computerprogramm (z. B. ein Textverarbeitungsprogramm) tut, als ein einziges hochrangiges Konzept oder eine kleine Menge solcher Konzepte, aber Ihre Fähigkeit, es zu verstehen, garantiert keineswegs, dass das Programm korrekt ist und keine versteckten Fehler enthält. Sogar wenn Sie das Programm selbst geschrieben haben, können Sie das nicht wirklich wissen; die meisten großen Programme, die Sie geschrieben haben, hatten Fehler, die zu einem späteren Zeitpunkt auftauchten, egal wie sorgfältig Sie beim Schreiben waren.

Also gut, es scheint nun also, dass der Grund dafür ist, warum Sie nicht herausfinden können, wie man einen Computer Beweise verifizieren lassen kann, dass jeder Schritt in Wirklichkeit einer Million kleiner Schritte entspricht. Nun, Sie mögen sagen, ein Computer könne eine Million Berechnungen in einer Sekunde durchführen, also wäre es vielleicht trotzdem praktisch. Jetzt ist es also eine Rätselaufgabe, wie man die eine Million Schritte, die jedem der Schritte in deutscher oder englischer Sprache entsprechen, herausfinden kann. Ihr mathematischer Freund hat keine Ahnung, schlägt aber vor, dass man die Antwort vielleicht durch das Studium der Mengenlehre fände. Eigentlich findet Ihr Freund, dass Sie ein bisschen verrückt sind, dass Sie sich so etwas überhaupt fragen. Für ihn ist das nicht das, worum es in der Mathematik geht.

Das Thema Mengenlehre taucht immer wieder auf, also beschließen Sie, dass es an der Zeit ist, es sich genauer anzuschauen.

Sie beschließen, vorsichtig anzufangen, und beginnen mit der Lektüre einiger sehr elementarer Bücher über Mengenlehre. Vieles davon scheint ziemlich offensichtlich zu sein, wie Schnittmengen, Teilmengen und Venn-Diagramme. Sie blättern durch eines der Bücher; nirgends werden Axiome erwähnt. Ein anderes Buch verweist auf einen Anhang, eine kurze Diskussion, in der eine Reihe von Axiomen erwähnt wird, die „Zermelo-Fraenkel-Mengenlehre“ und gibt sie auf Deutsch oder Englisch wieder. Man sieht sie sich an und hat keine Ahnung, was sie wirklich bedeuten oder was man mit ihnen anfangen kann. Die Kommentare in diesem Anhang besagen, dass ihre Erwähnung dazu diene, Sie mit der Idee vertraut zu machen, dass sie aber für das grundlegende Verständnis nicht notwendig seien und dass sie eigentlich Gegenstand fortgeschrittener Abhandlungen sind, in denen Feinheiten

wie ein bestimmtes Paradoxon (Russells Paradoxon[13]) gelöst werden. Moment mal - sollten die Axiome nicht ein Ausgangspunkt und kein Endpunkt sein? Wenn es Paradoxien gibt, welche ohne die Axiome entstehen, woher weiß man dann, dass man nicht zufällig über eines stolpert, wenn man den informellen Ansatz verwendet?

Und nirgends wird in diesen Büchern beschrieben, wie sich „die gesamte Mathematik aus der Mengenlehre ableiten lässt“, was Sie inzwischen schon ein paar Mal gehört haben.

Sie finden ein fortgeschritteneres Buch über Mengenlehre. Dieses Buch listet die Axiome der ZF-Mengentheorie in einfachem Deutsch oder Englisch auf Seite eins auf. *Jetzt* denken Sie, Ihre Suche sei zu Ende und Sie haben endlich die Quelle allen mathematischen Wissens gefunden; Sie müssen nur noch verstehen, was sie bedeuten. Hier, an einem einzigen Ort, ist die Grundlage für die gesamte Mathematik! Sie starren voller Ehrfurcht auf die Axiome, rätseln über sie, lernen sie auswendig und hoffen, dass sie Ihnen klar werden, wenn Sie nur lange genug über sie nachdenken. Natürlich haben Sie nicht die geringste Ahnung, wie der Rest der Mathematik von ihnen „abgeleitet“ ist; insbesondere, wenn dies die Axiome der Mathematik sind, warum brauchen dann Arithmetik, Gruppentheorie und so weiter ihre eigenen Axiome?

Sie fangen an, dieses fortgeschrittene Buch sorgfältig zu lesen, und denken über die Bedeutung jedes Wortes nach, denn Sie wollen der Sache unbedingt auf den Grund gehen. Das erste, was das Buch tut, ist zu erklären, wie die Axiome zustande gekommen sind, nämlich um das Russellsche Paradoxon zu lösen. In der Tat scheint das der Hauptzweck ihrer Existenz zu sein; dass sie angeblich dazu verwendet werden können, die gesamte Mathematik abzuleiten, scheint irrelevant zu sein und wird nicht einmal erwähnt. Wie dem auch sei, Sie fahren fort. Sie hoffen, dass das Buch Ihnen klar und deutlich, Schritt für Schritt, erklären wird, wie man die Dinge aus den Axiomen ableitet. Schließlich ist dies der Ausgangspunkt der Mathematik, so wie ein Buch, das die Grundlagen einer Programmiersprache erklärt. Aber irgendetwas fehlt. Sie können nicht einmal den ersten Beweis verstehen oder die erste Übung machen. Symbole wie $\exists$ und $\forall$ durchziehen die Seite, ohne dass erwähnt wird, woher sie kommen oder wie man sie manipuliert. Der Autor geht davon aus, dass man mit ihnen völlig vertraut ist, und sagt Ihnen nicht einmal, was sie bedeuten. Inzwischen wissen Sie, dass $\exists$ „es gibt“ bedeutet und dass $\forall$ „für alle“ bedeutet, aber sollten nicht die Regeln für die Manipulation dieser Symbole Teil der Axiome sein? Sie haben immer noch keine Idee, wie man die Axiome überhaupt einem Computer beschreiben könnte.

[13]Russells Paradoxon setzt voraus, dass es eine Menge S gibt, die eine Sammlung aller Mengen ist, die sich selbst nicht enthalten. Also, entweder enthält S sich selbst oder nicht. Enthält diese Menge sich selbst, widerspricht sie ihrer Definition. Enthält sie sich aber nicht selbst, widerspricht sie ebenso ihrer Definition. Das Russellsche Paradoxon wird in der ZF-Mengenlehre Theorie aufgelöst, indem man ausschließt, dass eine solche Menge S existiert.

Sicherlich gibt es hier etwas ganz anderes als die technische Literatur, die Sie zu lesen gewohnt sind. In einem Handbuch für Computersprachen wird fast immer sehr deutlich, was alle Symbole bedeuten, was sie genau machen und welche Regeln es gibt, nach denen sie kombiniert werden, und man arbeitet sich von dort aus weiter vor.

Nach einem Blick in vier oder fünf andere Bücher dieser Art kommt man zu der Erkenntnis, dass es noch ein ganzes Studienfach gibt, das man braucht, um die Axiome der Mengenlehre zu verstehen. Dieses Gebiet wird „Logik“ genannt. In der Tat wurde es in einigen Büchern als Voraussetzung empfohlen, aber man hat es einfach nicht realisiert. Man nahm an, Logik sei, nun ja, einfach nur Logik, etwas, das ein Mensch mit gesundem Menschenverstand intuitiv versteht. Warum Ihre Zeit mit der Lektüre langweiliger Abhandlungen über symbolische Logik verschwenden, die Manipulation von 1en und 0en, die Computer machen, wenn man das schon weiß? Aber dies ist eine andere Art von Logik, die Ihnen völlig fremd ist. Das Thema von NAND und NOR-Gattern wird nicht einmal berührt oder hat ohnehin nur mit einem sehr kleinen Teil dieses Bereichs zu tun.

Ihre Suche geht also weiter. Wenn Sie die ersten einführenden Bücher überfliegen, bekommen Sie eine allgemeine Vorstellung davon, worum es in der Logik geht und was Quantoren („Für alle“, „Es gibt“) bedeuten, aber Sie finden die Beispiele etwas trivial und leicht nervig („Alle Hunde sind Tiere“, „Einige Tiere sind Hunde“ und so weiter). Aber alles, was Sie wissen wollen, sind die Regeln für die Manipulation der Symbole, damit man sie in der Mengenlehre anwenden kann. Einige Formeln, die die Beziehungen zwischen den Quantoren ($\exists$ und $\forall$) beschreiben, sind in Tabellen aufgelistet, zusammen mit einigen verbalen Begründungen, um sie zu rechtfertigen. Wenn Sie herausfinden wollen, ob eine Formel richtig ist, durchlaufen Sie vermutlich die gleiche Art von Denkprozessen, möglicherweise mit Bildern von Hunden und Tieren. Intuitiv scheinen die Formeln einen Sinn zu ergeben. Aber wenn Sie sich fragen „Was sind die Regeln, die ich brauche, um einen Computer dazu zu bringen, herauszufinden, ob diese Formel richtig ist?“, wissen Sie es immer noch nicht. Man bittet den Computer ja auch nicht, sich Hunde und Tiere vorzustellen.

Sie sehen sich einige fortgeschrittenere Logikbücher an. Viele von ihnen haben ein einführendes Kapitel mit einer Zusammenfassung der Mengenlehre, was sich als Voraussetzung erweist. Sie brauchen Logik, um die Mengenlehre zu verstehen, aber anscheinend braucht man auch die Mengenlehre, um Logik zu verstehen! Diese Bücher stürzen sich gleich auf den Beweis von ziemlich fortgeschrittenen Theoremen über Logik, ohne den geringsten Hinweis darauf zu geben, woher die Logik kommt, mit der sie diese Theoreme beweisen können.

Glücklicherweise stoßen Sie auf ein elementares Buch über Logik, das nach der Hälfte der Lektüre, nach den üblichen Wahrheitstabellen und Metaphern, auf klare und präzise Weise präsentiert, wonach Sie die ganze Zeit

gesucht haben: die Axiome! Sie sind unterteilt in Aussagenlogik (auch Satzlogik genannt) und Prädikatenlogik (auch Logik erster Ordnung genannt), mit Regeln, die so einfach und kristallklar sind, dass man jetzt endlich einen Computer programmieren kann, um sie zu verstehen. Sie sind in der Tat nicht schwieriger als das Erlernen einer Schachpartie. Soweit es um das geht, was Sie zu brauchen scheinen, hätte man das ganze Buch auf fünf Seiten schreiben können!

Jetzt glauben Sie, die ultimative Quelle der mathematischen Wahrheit gefunden zu haben. Also - die Axiome der Mathematik bestehen aus den Axiomen der Logik, zusammen mit den Axiomen der ZF-Mengentheorie. (Inzwischen haben Sie auch herausgefunden wie man die ZF-Axiome aus dem Deutschen oder Englischen in die eigentlichen Symbole der Logik übersetzen kann, die Sie nun nach präzisen, leicht verständlichen Regeln manipulieren können.)

Natürlich verstehen Sie immer noch nicht, wie „die gesamte Mathematik aus der Mengenlehre abgeleitet werden kann“, aber vielleicht wird sich das zu gegebener Zeit offenbaren.

Sie machen sich eifrig daran, die Axiome und Regeln in einen Computer zu programmieren und beginnen sich mit den Theoremen zu befassen, die Sie beweisen müssen, während die Logik entwickelt wird. Alle Arten von wichtigen Theoremen tauchen auf: das Deduktionstheorem, das Substitutionstheorem, der Vollständigkeitssatz der Aussagenlogik, der Vollständigkeitssatz der Prädikatenlogik. Oh-oh, da scheint es Probleme zu geben. Sie werden alle schwieriger und schwieriger, und nicht eine davon kann mit den Axiomen und Regeln der Logik, die Sie gerade erhalten haben, abgeleitet werden. Stattdessen benötigen sie alle eine „Metalogik“ für ihre Beweise, eine Art Mischung aus Logik und Mengenlehre, die es erlaubt, Dinge *über* die Axiome und Theoreme der Logik zu beweisen, anstatt *mit* ihnen.

Sie machen trotzdem weiter. Einen Monat später haben Sie einen Großteil Ihrer Freizeit damit verbracht, den Computer dazu zu bringen, Beweise in der Aussagenlogik zu verifizieren. Sie haben die Axiome einprogrammiert, aber Sie mussten auch den Deduktionssatz, den Substitutionssatz und den Satz von der Vollständigkeit der Aussagenlogik einprogrammieren, da Sie sich inzwischen damit abgefunden haben, diese Sätze als ziemlich komplexe Zusatzaxiome zu betrachten, da sie nicht aus den Axiomen bewiesen werden können. Sie können nun den Computer dazu bringen, vollständige, strenge, formale Beweise zu überprüfen und sogar zu erzeugen. Es macht nichts, dass die Beweise 100.000 Schritte haben können - zumindest können Sie jetzt vollständiges, absolutes Vertrauen in sie haben. Leider sind die einzigen Theoreme, die Sie bewiesen haben, ziemlich trivial, und Sie können sie in wenigen Minuten mit Wahrheitstabellen überprüfen, wenn nicht sogar durch direkte Überprüfung.

Es sieht so aus, als hätte Ihr Freund, der Mathematiker, recht gehabt. Den Computer dazu zu bringen, ernsthafte Mathematik mit dieser Art von

Strenge zu betreiben, scheint fast aussichtslos. Sogar schlimmer noch, je weiter man kommt, desto mehr „Axiome“ muss man hinzufügen, da jedes neue Theorem zusätzliche „metamathematische“ Argumentation zu beinhalten scheint, die nicht formalisiert wurde, und nichts davon lässt sich aus den Axiomen der Logik ableiten. Nicht nur, dass die Beweise exponentiell wachsen, je weiter man kommt, sondern auch das Programm zu ihrer Verifizierung wird immer größer, je mehr „Metatheoreme“[14] man einprogrammiert. Die bisher aufgetauchten Fehler in Ihrem Computerprogramm haben Sie bereits dazu gebracht, den Glauben an die Strenge zu verlieren, die Sie scheinbar erreicht haben, und Sie wissen, dass es nur noch schlimmer wird, je größer Ihr Programm wird.

1.1.2 Mathematik und der Nichtfachmann

> *Ein echter Beweis ist nicht von einer Maschine überprüfbar, und sogar nicht von einem Mathematiker, wenn er nicht in die Gestalt, die Denkweise des speziellen Bereichs der Mathematik eingeweiht ist, in dem der Beweis angesiedelt ist.*
>
> Davis und Hersh [15]

Der größte Teil der abstrakten oder theoretischen Mathematik liegt in der Regel außerhalb der Reichweite aller, mit Ausnahme von nur wenigen Spezialisten auf dem jeweiligen Gebiet, die eine notwendige, schwierige Ausbildung absolviert haben, um in diesen Kreis aufgenommen zu werden. Der typisch intelligente Laie hat keine begründete Hoffnung, viel davon zu verstehen, und auch der spezialisierte Mathematiker hat keine Chance, andere Bereiche zu verstehen. Es ist wie eine Fremdsprache, für die es kein Wörterbuch gibt, um die Übersetzung nachzuschlagen; die einzige Möglichkeit sie zu lernen besteht darin, ein paar Jahre in dem jeweiligen Land zu leben. Der Aufwand, der mit dem Erlernen eines Fachgebiets verbunden ist, wird als ein notwendiger Prozess angesehen, um ein tiefes Verständnis zu erlangen. Natürlich gilt dies sicherlich, wenn man einen bedeutenden Beitrag zu einem Fachgebiet leisten will; insbesondere Beweise zu „führen“, was wahrscheinlich der wichtigste Teil der Ausbildung eines Mathematikers ist. Aber ist es auch notwendig, Außenstehenden den Zugang dazu zu verwehren? Ist es notwendig, dass abstrakte Mathematik für einen Laien so schwer zu begreifen ist?

[14] Ein Metatheorem ist normalerweise eine Aussage, die zu allgemein ist, um direkt in einer Theorie bewiesen werden zu können. Zum Beispiel: „Wenn n_1, n_2 und n_3 ganze Zahlen sind, dann ist $n_1 + n_2 + n_3$ eine ganze Zahl“ ist ein Satz der Zahlentheorie. Aber „für jede ganze Zahl $k > 1$, wenn $n_1, \ldots, n_k$ ganze Zahlen sind, dann ist $n_1 + \ldots + n_k$ eine ganze Zahl“ ist ein Metatheorem, das heißt eine Familie von Theoremen, eines für jedes k. Warum dies kein Theorem ist, ist darin begründet, dass die allgemeine Summe $n_1 + \ldots + n_k$ (als Funktion von k) keine Operation ist, die direkt in der Zahlentheorie definiert werden kann.

[15] Frei übersetzt nach [15], S. 354.

Ein Computer ist normalerweise überhaupt keine Hilfe. Die meisten veröffentlichten Beweise sind eigentlich nur eine Reihe von Hinweisen, die in einem informellen Stil geschrieben sind, der ein beträchtliches Wissen auf dem Gebiet erfordert, um sie zu verstehen. Dies sind die „echten Beweise“, auf die Davis und Hersh hinweisen. Es gibt eine implizite Übereinkunft darüber, dass ein solcher Beweis im Prinzip in einen vollständigen formalen Beweis umgewandelt werden kann. Es heißt jedoch, dass niemand eine solche Umwandlung jemals versuchen würde, selbst wenn er es könnte, denn das würde vermutlich Millionen von Schritten erfordern (Abschnitt 1.1.3). Leider schließt der informelle Stil automatisch das Verstehen des Beweises für jeden aus, der nicht die notwendige Ausbildung durchlaufen hat. Es bleibt einem intelligenten Laien nichts anderes übrig als populäre Bücher über tiefe und berühmte Ergebnisse zu lesen; dies kann zwar hilfreich, aber auch irreführend sein, und der Mangel an Details lässt den Leser in der Regel ohne jegliche Möglichkeit zurück, einen Aspekt des beschriebenen Bereichs zu erforschen.

Die Aussagen der Theoreme verwenden oft eine komplizierte Notation, die sie für den Nichtfachmann unzugänglich machen. Für einen Nichtfachmann, der einen Beweises genauer verstehen möchte, wird der Prozess der Rückverfolgung von Definitionen und Lemmata durch ihre Hierarchie schnell verwirrend und entmutigend. Lehrbücher werden normalerweise geschrieben, um Mathematiker auszubilden oder um Mathematikern neues Wissen zu vermitteln, und große Lücken in Beweisen werden oft dem Leser als Aufgabe überlassen, der in eine Sackgasse gerät, wenn er oder sie damit nicht weiterkommt.

Ich glaube, dass Computer es irgendwann auch Nichtfachleuten und sogar intelligenten Laien ermöglichen, fast jeden mathematischen Beweis auf jedem Gebiet nachvollziehen zu können. Metamath ist ein Ansatz in diese Richtung. Wäre die gesamte Mathematik so leicht zugänglich wie eine Computerprogrammiersprache, könnte ich mir vorstellen, dass es Computerprogrammierern und Hobbyisten, denen es sonst an mathematischer Raffinesse fehlt, die Welt der Theoreme und Beweise in obskuren Fachgebieten zu erkunden und darüber zu staunen. Vielleicht gelingt es ihnen dann sogar, eigene Ergebnisse zu finden. Ein enormer Vorteil wäre, dass jeder mit Vermutungen auf jedem Gebiet experimentieren könnte - der Computer würde eine sofortige Rückmeldung darüber geben, ob ein Schritt in einer Schlussfolgerung korrekt ist.

Mathematiker müssen manchmal das Ärgernis über Spinner ertragen, denen es an einem grundlegenden Verständnis der Mathematik mangelt, die aber darauf bestehen, dass ihre Beweise für, sagen wir, den Großen Fermatschen Satz, ernst genommen werden. Ich denke, ein Teil des Problems ist, dass diese Leute von der informellen mathematischen Sprache in die Irre geführt werden und sie so behandeln, als ob sie gewöhnliches, erklärendes Deutsch oder Englisch lesen würden, und nicht die implizit zugrunde liegende Strenge beachten. Solche Verrückten sind im Bereich der Compu-

ter selten, denn Computersprachen sind viel expliziter, und der Beweis liegt letztlich darin, ob ein Computerprogramm funktioniert oder nicht. Mit leicht zugänglicher, computergestützter abstrakter Mathematik könnte ein Mathematiker zu einem Verrückten sagen, „Belästigen Sie mich nicht, bis Sie Ihre Behauptung auf dem Computer bewiesen haben!“

1.1.3 Ein unmöglicher Traum?

Die Darstellung der Beweise selbst ganz einfacher Theoreme würde unglaublich umfangreiche Bücher füllen.

Robert E. Edwards[16]

Oh, natürlich macht *das nie jemand wirklich. Das würde ewig dauern! Du zeigst nur, dass du es könntest, das reicht aus.*

„Der ideale Mathematiker“ [17]

Es gibt ein Theorem in der elementaren Notation der Mengenlehre, das dem arithmetischen Theorem ‘$1000 + 2000 = 3000$’ *entspricht. Die Formel wäre furchtbar lang, denn selbst wenn [man] die Definitionen kennt und gebeten wird, die lange Formel entsprechend zu vereinfachen, wird man wahrscheinlich Fehler machen und zu einem falschen Ergebnis kommen.*

Hao Wang[18]

Die Principia Mathematica *war die krönende Leistung der Formalisten. Es war auch der Todesstoß für die formalistische Sichtweise. . . . [Russell] gelang es in drei riesigen Bänden nicht, über die elementaren Fakten der Arithmetik hinauszukommen. Er zeigte, was prinzipiell möglich ist und was in der Praxis nicht möglich ist. Wäre der mathematische Prozess wirklich eine streng logische Abfolge, dann würden wir immer noch unsere Finger zählen. . . . Ein Theoretiker schätzt. . . , dass die Darstellung einer von Ramanujans Vermutungen mithilfe der Mengenlehre und elementarer Analysis etwa zweitausend Seiten umfassen würde; die Länge einer Ableitung aus ersten Prinzipien ist fast unvorstellbar . . . Die Wahrscheinlichkeitstheoretiker argumentieren, dass . . . jeder sehr lange Beweis bestenfalls als „wahrscheinlich richtig“ angesehen werden kann. . . .*

[16] Frei übersetzt nach [17], S. 68.
[17] Frei übersetzt nach [15], „The Ideal Mathematician“ S. 40.
[18] Frei übersetzt nach [71], S. 140.

Richard de Millo et. al.[19]

Einige Autoren haben den Eindruck erweckt, dass die Art von absoluter Strenge, wie sie Metamath bietet, ein unmöglicher Traum ist und suggeriert, dass ein vollständiger, formaler Beweis eines typischen Theorems Millionen von Schritten in unzähligen Bänden von Büchern erfordern würde. Selbst wenn es möglich wäre, dann wird manchmal angenommen, dass jegliche Bedeutung bei einer solchen monströsen, langwierigen Überprüfung verloren gehen würde.

Diese Autoren gehen jedoch davon aus, dass man für eine solche Art von vollständiger, formaler Verifikation einen Beweis in einzelne primitive Schritte herunterbrechen müsse, die sich direkt auf die Axiome beziehen. Dies ist nicht notwendig. Es gibt keinen Grund, nicht auf bereits bewiesene Theoreme zurückzugreifen, anstatt sie immer wieder aufs Neue zu beweisen.

Genauso wichtig ist es, dass Definitionen auf dem Beweisweg eingeführt werden können, so dass sehr komplexe Formeln mit wenigen Symbolen dargestellt werden können. Wird dies nicht getan, können absurd lange Formeln entstehen. Zum Beispiel würde allein das Formulieren des Gödelschen Unvollständigkeitssatzes, das mit einer überschaubaren Anzahl definierter Symbole ausgedrückt werden kann, etwa 20.000 primitive Symbole erfordern[20]. Ein extremes Beispiel ist Bourbakis Sprache für die Mengenlehre, die
4.523.659.424.929 Symbole plus 1.179.618.517.981 „unterscheidende Verbindungen“ (Linien, die Symbolpaare miteinander verbinden, die normalerweise unterhalb oder oberhalb der Formel gezogen werden) erfordert, um die Zahl „eins“ auszudrücken [40].

Eine Hierarchie von Theoremen und Definitionen ermöglicht es, dass exponentiell wachsende Formelgrößen und primitive Beweisschritte mit nur einer linear wachsenden Anzahl von verwendeten Symbole beschrieben werden können. Natürlich ist dies die Art und Weise, wie sie auch in der gewöhnlichen, informellen Mathematik normalerweise praktiziert wird, aber mit Metamath kann dies mit absoluter Strenge und Präzision geschehen.

1.1.4 Schönheit

> *Aus dem Paradies, das Cantor uns geschaffen, soll uns niemand vertreiben können.*
>
> David Hilbert[21]

[19] Frei übersetzt nach [16], S. 269, 271.

[20] George S. Boolos, Vortrag am Massachusetts Institute of Technology, Frühling 1990.

[21] „Über das Unendliche“ in *Mathematische Annalen*, 95. Band, Verlag von Julius Springer, Berlin 1926, S. 170. Außerdem zitiert in [47], S. 131.

Die Mathematik hat nicht nur Wahrheit inne, sondern auch eine höchste Schönheit — eine kalte und strenge Schönheit, wie die einer Skulptur.

Bertrand Russell[22]

Euklid allein hat die schiere Schönheit gesehen.

Edna Millay[23]

Das Wissen eines Menschen macht seine Miene strahlend, und seine strengen Züge lösen sich.

Kohelet/Prediger 8:1[24]

Für die meisten Menschen ist die abstrakte Mathematik weit weg, fremd und unverständlich. Viele populäre Bücher haben versucht, etwas von dem Sinn der Schönheit berühmter Theoreme zu vermitteln. Aber selbst ein intelligenter Laie hat nur eine allgemeine Vorstellung davon, worum es in einem Theorem geht, und erhält kaum die Werkzeuge, die er braucht um sie zu nutzen. Traditionell erfordert es erst ein jahrelanges, mühsames Studium, um die für ein tiefes Verständnis erforderlichen Konzepte zu erfassen. Metamath ermöglicht es, den Beweis des Satzes aus einer ganz anderen Perspektive anzugehen, indem man die Formeln und Definitionen Schicht für Schicht auseinandernimmt, bis man ein völlig anderes Verständnis gewonnen hat. Jeder Schritt des Beweises ist nachvollziehbar, mit absoluter Präzision zusammengesetzt, und kann sofort wie durch ein Mikroskop mit einer beliebigen Vergrößerung, so stark ist wie Sie es wünschen, weiter inspiziert werden.

Ein Beweis an sich kann als ein Kunstobjekt angesehen werden. Das Konstruieren eines eleganten Beweis ist eine Kunst. Sobald ein berühmtes Theorem bewiesen ist, werden oft beträchtliche Anstrengungen unternommen, um einfachere und leichter verständliche Beweise zu finden. Das Erstellen und Vermitteln eleganter Beweise ist ein Hauptanliegen von Mathematikern. Metamath ist eine Möglichkeit, eine gemeinsame Sprache für die Archivierung und Bewahrung dieser Informationen zu bieten.

Die Länge eines Beweises kann bis zu einem gewissen Grad als objektives Maß für seine „Schönheit“ sein, da kürzere Beweise gewöhnlich als eleganter gelten. In der Mengenlehre-Datenbasis`set.mm`, die zusammen mit dem Metamath-Programm bereitgestellt wird, war und ist es ein Ziel, alle Beweise so kurz wie möglich zu halten.

[22] Frei übersetzt nach [60].

[23] Frei übersetzt nach dem Zitat in [15], S. 150.

[24] Ergänzung der Übersetzer, aus *Die Bibel - Einheitsübersetzung*, Katholische Bibelanstalt, 1980, Stuttgart.

1.1.5 Einfachheit

Gott hat die Menschen einfach und aufrichtig erschaffen, aber manche wollen alles kompliziert haben.

KOHELET/PREDIGER 7:29[25]

Die ganzen Zahlen hat der liebe Gott gemacht, alles andere ist Menschenwerk.

LEOPOLD KRONECKER[26]

Denn das Klare und leicht Faßliche zieht uns an, das Verwickelte schreckt uns ab.

DAVID HILBERT[27]

Die Metamath-Sprache ist einfach und spartanisch. Metamath betrachtet alle mathematischen Ausdrücke als einfache Abfolgen von Symbolen, die an sich keine Bedeutung haben. Die übergeordneten oder „metamathematischen" Grundsätze, auf denen Metamath basiert, sind so einfach wie möglich gehalten. Jeder einzelne Schritt in einem Beweis basiert auf einem einzigen Grundkonzept, nämlich der Ersetzung einer Variablen durch einen Ausdruck, so dass im Prinzip fast jeder, ob Mathematiker oder nicht, komplett verstehen kann, wie er zustande gekommen ist.

Eine der grundlegendsten Anwendungen von Metamath ist die Entwicklung der Grundlagen der Mathematik von Anfang an. Dies geschieht in der Mengenlehre-Datenbasis, die in dem Metamath-Paket mitgeliefert wird und Gegenstand von Kapitel 3 ist. Jede Sprache (eine Metasprache) die zur Beschreibung der Mathematik verwendet wird (eine Objektsprache), muss selbst mathematische Konzepte beinhalten. Aber es ist wünschenswert, diese Konzepte auf ein absolutes Minimum zu beschränken, nämlich auf die für die Anwendung der axiomatisch spezifizierten Inferenzregeln benötigten. Für jede Metasprache gibt es ein „Henne-Ei'-Problem, ähnlich wie bei einem Zirkelschluss: Man muss die Gültigkeit der Mathematik der Metasprache voraussetzen, um die Gültigkeit der Mathematik der Objektsprache zu beweisen. Der mathematische Anteil von Metamath selbst ist recht begrenzt. Wie die Regeln eines Schachspiels sind die wesentlichen Konzepte so

[25] *Die Bibel im heutigem Deutsch*, Deutsche Bibelgesellschaft, 1982, Stuttgart. (Anm. der Übersetzer: Das hier wiedergegebene Zitat entspricht dem Zitat aus dem englischen Original; in anderen Ausgaben der Bibel wird diese Stelle häufig etwas anders übersetzt, z.B. *Gott hat die Menschen rechtschaffen gemacht, aber sie haben sich in allen möglichen Berechnungen versucht* in *Die Bibel - Einheitsübersetzung*, Katholische Bibelanstalt, 1980, Stuttgart.)

[26] *Jahresbericht der Deutschen Mathematiker-Vereinigung* , Ausg. 2, S. 19.

[27] *Mathematische Probleme* (1900), S. 254.

einfach, dass praktisch jeder in der Lage sein sollte, sie zu verstehen (auch wenn das an sich nicht dazu führt, dass man wie ein Meister spielt). Die Symbole, mit denen Metamath arbeitet, haben an sich keine eigene Bedeutung. Die Interpretation der Axiome, die Sie für Metamath festlegen, durch Sie selbst ist es, was ihnen Bedeutung verleiht. Metamath ist ein Versuch, das mathematisches Denken auf seine pure Essenz zu reduzieren und Ihnen genau zu zeigen, wie die Symbole manipuliert werden.

Philosophen und Logiker haben es aus verschiedenen Gründen oft für wichtig gehalten, „schwache“ Teilbereiche der Logik[2] [42], andere unkonventionelle Systeme der Logik (wie z.B. „modale“ Logik [8, Kap. 27]), und Quantenlogik in der Physik[51] zu untersuchen. Metamath bietet einen Rahmen, in dem solche Systeme mit einer absoluten Präzision ausgedrückt werden können, die alle zugrundeliegenden metamathematischen Annahmen präzise und kristallklar darlegt.

Einige philosophische Denkschulen wie der Intuitionismus und der Konstruktivismus fordern, dass die jedem mathematischen System zugrunde liegenden Begriffe so einfach und konkret wie möglich seien. Metamath sollte den Anforderungen dieser Philosophien entsprechen. Metamath muss die Symbole, Axiome, und Regeln für eine bestimmte Theorie, vom Skeptischen (wie beim Intuitionismus [28]) bis zum Kühnen (wie zum Beispiel das Auswahlaxiom in der Mengenlehre [29]) beigebracht werden.

Die Einfachheit der Metamath-Sprache erlaubt es dem Algorithmus (Computerprogramm), der die Gültigkeit eines Metamath-Beweises verifiziert, einfach und robust zu sein. Sie können darauf vertrauen, dass die Theoreme, die er verifiziert, wirklich aus Ihren Axiomen abgeleitet werden können.

1.1.6 Strenge

> *Rigorosität wurde bei den Griechen zu einem Ziel ... Aber die Bemühungen, die Strenge bis zum Äußersten zu treiben, haben in eine Sackgasse geführt, in der es keine Einigung mehr darüber*

[28] Der Intuitionismus akzeptiert nicht das Gesetz des ausgeschlossenen Dritten („entweder ist etwas wahr oder es ist nicht wahr“). Siehe [70, S. xi] für Diskussionen und Referenzen zu diesem Thema. Betrachten Sie das Theorem „Es gibt irrationale Zahlen a und b, so dass a^b rational ist“. Ein Intuitionist würde den folgenden Beweis ablehnen: Wenn $\sqrt{2}^{\sqrt{2}}$ rational ist, sind wir fertig. Ansonsten gilt $a = \sqrt{2}^{\sqrt{2}}$ und $b = \sqrt{2}$. Dann ist $a^b = 2$, was eine rational Zahl ist.

[29] Das Auswahlaxiom besagt, dass bei einer beliebigen Sammlung von paarweise disjunkten, nicht leeren Mengen eine Menge existiert, die mit jeder Menge der Sammlung genau ein Element gemeinsam hat. Es wird verwendet, um viele wichtige Theoreme in der Standardmathematik zu beweisen. Einige Philosophen lehnen es ab, weil die Existenz einer Menge behauptet wird, ohne ihre Elemente benennen zu können[18, S. 154]. In einer Grundlage für die Mathematik, die auf Quine zurückgeht und sich nicht als inkonsistent erwiesen hat, erweist sich das Auswahlaxiom als falsch[14, S. 23]. Der `show trace_back`-Befehl des Metamath-Programms ermöglicht es Ihnen herauszufinden, ob das Auswahlaxiom oder ein anderes Axiom in einem Beweis angenommen wurde.

gibt, was sie wirklich bedeutet. Die Mathematik bleibt lebendig und vital, aber nur auf einer pragmatischen Basis.

Morris Kline[30]

Kline bezieht sich auf eine viel tiefere Art von Strenge als die, die wir in diesem Abschnitt erörtern werden. Der Gödelsche Unvollständigkeitssatz zeigte, dass es unmöglich ist, in der Standardmathematik absolute Strenge zu erreichen, weil wir nie beweisen können, dass die Mathematik konsistent (frei von Widersprüchen) ist. Wenn die Mathematik konsistent ist, werden wir es nie wissen, sondern müssen uns auf den Glauben darauf verlassen. Wenn die Mathematik inkonsistent ist, können wir bestenfalls darauf hoffen, dass in der Zukunft ein kluger Mathematiker die Inkonsistenz entdecken wird. In diesem Fall würden die Axiome wahrscheinlich leicht überarbeitet werden, um die Inkonsistenz zu beseitigen, wie im Fall des Russell'schen Paradoxons geschehen ist. Aber der Großteil der Mathematik wäre von einer solchen Entdeckung wahrscheinlich nicht betroffen. Das Russell-Paradoxon hatte zum Beispiel keinen Einfluss auf die meisten bemerkenswerten Ergebnisse, die von Mathematikern des 19. Jahrhunderts und früher erzielt wurden. Es widerlegte hauptsächlich einige der Arbeiten von Gottlob Frege über die Grundlagen der Mathematik in den späten 1800er Jahren; tatsächlich inspirierte Freges Arbeit Russells Entdeckung. Trotz dieses Paradoxons enthält Freges Werk wichtige Konzepte, die die moderne Logik maßgeblich beeinflusst haben. Kline's *Mathematics, The Loss of Certainty* [31] enthält eine interessante Diskussion zu diesem Thema.

Was mit absoluter Gewissheit erreicht werden *kann* ist das Wissen, dass unter der Annahme, dass die Axiome konsistent und wahr sind, die Ergebnisse, die aus ihnen abgeleitet werden, wahr sind. Ein Teil der Schönheit der Mathematik ist, dass sie der einzige Bereich menschlichen Strebens ist, in dem absolute Gewissheit in diesem Sinne erreicht werden kann. Eine mathematische Wahrheit wird bis in alle Ewigkeit wahr bleiben. Dennoch ist unser tatsächliches Wissen darüber, ob eine bestimmte Aussage eine mathematische Wahrheit darstellt, jedoch nur so sicher wie die Korrektheit des Beweises, der dieser Aussage zugrunde liegt. Wenn der Beweis einer Aussage fragwürdig oder vage ist, können wir nicht absolutes Vertrauen in die von der Aussage behaupteten Wahrheit haben.

Schauen wir uns einige traditionelle Arten an, Beweise auszudrücken.

Außer auf dem Gebiet der formalen Logik sind fast alle traditionellen Beweise in der Mathematik eigentlich gar keine Beweise, sondern eher Beweisskizzen oder Hinweise darauf, wie der Beweis zu konstruieren ist. Viele Lücken werden dem Leser zum Ausfüllen überlassen. Dafür gibt es mehrere Gründe. Erstens wird in der mathematischen Literatur gewöhnlich angenommen, dass die Person, die den Beweis liest, ein Mathematiker ist, der mit dem beschriebenen Spezialgebiet vertraut ist, und dass die fehlenden

[30] Frei übersetzt nach [30], S. 1209.

Schritte für einen solchen Leser offensichtlich sind oder er zumindest in der Lage ist, sie zu ergänzen. Diese Haltung ist für professionelle Mathematiker des Fachgebiets in Ordnung, aber leider hat sie oft den Nachteil, dass sie den Rest der Welt, einschließlich der Mathematiker anderer Fachgebiete, vom Verstehen des Beweises ausschließt. Wir haben eine mögliche Lösung für dieses Problem auf S. xv diskutiert. Zweitens wird oft angenommen, dass ein vollständig formaler Beweis indexformaler Beweis unzählige Millionen von Symbolen erfordern würde (Abschnitt 1.1.3). Dies könnte richtig sein, wenn der Beweis direkt durch die Axiome der Logik und der Mengenlehre ausgedrückt würde. Aber es ist normalerweise nicht richtig, wenn wir uns eine Hierarchie von Definitionen und Theoremen erlauben, auf denen wir aufbauen können, und eine Notation verwenden, die es uns erlaubt, neue Symbole, Definitionen und Theoreme auf eine genau spezifizierte Weise einzuführen.

Selbst in der formalen Logik enthalten formale Beweise, die als vollständig angesehen werden, immer noch versteckte oder implizite Informationen. Zum Beispiel wird ein „Beweis“ gewöhnlich als eine Folge von wffs, [31] definiert, von denen jedes ein Axiom ist oder aus einer Regel folgt, die auf vorherige wffs in der Sequenz angewendet wird. Der implizite Teil des Beweises ist der Algorithmus, mit dem eine Folge von Symbolen als gültige wff verifiziert wird, wenn die Definition einer wff gegeben ist. Der Algorithmus ist in diesem Fall recht einfach, aber damit ein Computer den Beweis verifizieren kann, muss er den Algorithmus in sein Verifikationsprogramm eingebaut haben. [32] Wenn man sich ausschließlich mit Axiomen und elementaren wffs beschäftigt, ist es einfach, einen solchen Algorithmus zu implementieren. Aber wenn mehr und mehr Definitionen zur Theorie hinzugefügt werden, um diese kompakter zu machen, wird der Algorithmus immer komplizierter. Ein Computerprogramm, das den Algorithmus implementiert, wird größer und mit jeder neuen Definition schwieriger zu verstehen und damit anfälliger für Bugs. Je größer das Programm, desto misstrauischer wird der Mathematiker gegenüber der Gültigkeit seiner Algorithmen. Dies gilt insbesondere, weil Computerprogramme von Natur aus schwer nachzuvollziehen sind, und nur wenige Menschen Spaß daran haben, sie manuell im Detail zu überprüfen.

[31] Eine *wff* oder eine wohlgeformte Formel ist ein mathematischer Ausdruck (eine Kette von Symbolen), der nach einigen präzisen Regeln aufgebaut ist. Ein formales mathematisches System enthält (1) die Regeln für die Konstruktion von syntaktisch korrekten wffs, (2) eine Liste von Ausgangs-wffs, genannt Axiome, und (3) eine oder mehrere Regeln, die vorschreiben, wie man neue wffs, genannt Theoreme, aus den Axiomen oder vorher abgeleiteten Theoremen ableitet. Ein Beispiel für ein solches System ist in Metamaths Mengenlehre-Datenbasis enthalten, die ein formales System definiert, aus dem die gesamte Standardmathematik abgeleitet werden kann. Abschnitt 2.2.1 führt Sie durch ein vollständiges Beispiel für ein formales Systems, und Sie sollten ihn überfliegen, wenn Sie mit dem Konzept nicht vertraut sind.

[32] Es ist natürlich möglich, die Syntax der wff-Konstruktion außerhalb des Programms mit einer geeigneten Eingabesprache selbst zu spezifizieren (die Metamath-Sprache ist ein Beispiel dafür), aber einige Programme zur Beweisverifikation oder zum Beweisen von Theoremen sind nicht in der Lage, die wff-Syntax auf eine solche Weise zu erweitern.

Metamath verfolgt einen anderen Ansatz. Das „Wissen“ von Metamath beschränkt sich auf die Fähigkeit, Variablen durch Ausdrücke zu ersetzen, und zwar unter einigen einfachen Einschränkungen. Sobald der grundlegende Algorithmus von Metamath als getestet und vielleicht unabhängig bestätigt wurde, kann man ihm ein für alle Mal vertrauen. Die Informationen, die Metamath benötigt, um Mathematik zu verstehen, ist vollständig in dem Wissensbestand enthalten, der Metamath zugrunde gelegt wird. Jegliche Fehler in der Argumentation können nur Fehler in den Axiomen oder Definitionen sein, die in diesem Wissensbestand enthalten sind. Als eine „konstruktive“ Sprache hat Metamath keine bedingte Verzweigungen oder Schleifen, wie sie Computerprogramme schwer entschlüsselbar machen; stattdessen kann die Sprache nur neue Folgen von Symbolen aus vorherigen Folgen von Symbolen aufbauen.

Die Einfachheit der Regeln, die Metamath zugrunde liegen, macht Metamath nicht nur leicht zu erlernen, sondern verleiht Metamath auch ein hohes Maß an Flexibilität. Metamath ist zum Beispiel nicht auf die Beschreibung der Standardlogik erster Ordnung beschränkt; Logik höherer Ordnung und Teilbereiche der Logik können ebenso einfach beschrieben werden. Metamath gibt Ihnen die Freiheit, jede beliebige wff-Notation zu definieren, die Sie bevorzugen; es hat keine eingebaute Interpretation der Syntax einer wff. Mit geeigneten Axiomen und Definitionen kann Metamath sogar Dinge über sich selbst beschreiben und beweisen. (John Harrison erörtert das „Reflexions“-Prinzip in selbstbeschreibenden Systemen in [22].)

Die Flexibilität von Metamath erfordert, dass die Beweise sehr viele Details enthalten, viel mehr als in einem gewöhnlichen „formalen“ Beweis. Zum Beispiel besteht in einem gewöhnlichen formalen Beweis ein einzelner Schritt aus der Anzeige der wff, die diesen Schritt darstellt. Damit ein Computerprogramm verifizieren kann dass der Schritt gültig ist, muss es zunächst prüfen, ob die angezeigte Symbolfolge eine gültige wff ist. Die meisten Prüfprogramme haben zumindest eine grundlegende wff-Syntax eingebaut. Metamath hat kein fest verdrahtetes Wissen darüber eingebaut, was eine wff ist; stattdessen muss jede wff explizit konstruiert werden, basierend auf Regeln, die in einer Datenbasis vorhanden wffs definieren. Daher kann ein einzelner Schritt in einem gewöhnlichen formalen Beweis vielen Schritten in einem Metamath-Beweis entsprechen. Trotz der größeren Anzahl von Schritten bedeutet dies jedoch nicht, dass ein Metamath-Beweis wesentlich größer sein muss als ein gewöhnlicher formaler Beweis. Der Grund dafür ist, dass wir wissen, da wir die wff von Grund auf neu konstruiert haben, was die wff ist - also gibt es keinen Grund, sie anzuzeigen. Wir müssen nur auf eine Folge von Aussagen verweisen, die sie konstruieren. In gewissem Sinne ist die Darstellung der wff in einem gewöhnlichen formalen Beweis ein impliziter Beweis für seine eigene Gültigkeit als wff; Metamath macht den Beweis einfach explizit. (Abschnitt 4.3 beschreibt die Beweisnotation von Metamath.)

1.2 Computer und Mathematiker

Der Computer ist wichtig, aber nicht für die Mathematik.

Paul Halmos[33]

Reine Mathematiker stehen Computern seit jeher gleichgültig gegenüber, bis hin zur Verachtung. Die Computerwissenschaft/Informatik selbst wird manchmal in den banalen Bereich der „angewandten“ Mathematik eingeordnet, die vielleicht für die reale Welt wichtig, aber für diejenigen, die nach den tiefsten Wahrheiten der Mathematik suchen, intellektuell wenig aufregend ist. Vielleicht liegt ein Grund für diese Einstellung gegenüber Computern darin, dass es wenig oder gar keine Computersoftware gibt, die ihren Bedürfnissen gerecht wird, und es mag die allgemeine Meinung vorherrschen, dass eine solche Software gar nicht existieren kann. Auf der einen Seite gibt es die praktischen Computeralgebrasysteme, die erstaunliche Symbolmanipulationen in den Bereichen Algebra und Infinitesimalrechnung durchführen können, aber nicht einmal den einfachsten Existenzsatz beweisen, wenn die Vorstellung eines Beweises überhaupt vorhanden ist. Andererseits gibt es spezialisierte automatische Theorembeweiser, die technisch gesehen korrekte Beweise generieren können. Aber manchmal sind ihre speziellen Eingabenotationen sehr kryptisch, und ihre Ausgaben werden als lange, unelegant wirkende, unverständliche Beweise wahrgenommen. Die Ausgabe kann mit Misstrauen betrachtet werden, da das Programm, das sie erzeugt, in der Regel sehr groß ist und seine Größe das Potenzial für Bugs erhöht. Ein solcher Beweis kann nur dann als vertrauenswürdig angesehen werden, wenn er von einem Menschen unabhängig verifiziert und „verstanden“ wurde, aber niemand möchte seine Zeit mit einer solch langweiligen, undankbaren Aufgabe verschwenden.

1.2.1 Dem Computer vertrauen

. . . ich halte die quasi-empirische Interpretation von Computerbeweisen nach wie vor für die plausiblere. . . . Da nicht alles, was angeblich ein Computerbeweis ist, als gültig anerkannt werden kann, was sind die mathematischen Kriterien für akzeptable Computerbeweise?

Thomas Tymoczko[34]

In einigen Fällen waren Computer unverzichtbare Werkzeuge für den Beweis berühmter Theoreme. Aber wenn ein Beweis so lang und obskur ist, dass er praktisch nur mit einem Computer überprüft werden kann, wird er vage als verdächtig empfunden. Zum Beispiel der Beweis des berühmten

[33] Frei übersetzt nach dem Zitat in [1], S. 121.
[34] Frei übersetzt nach [70], S. 245.

Vier-Farben-Satzes („es werden nicht mehr als vier Farben benötigt, um eine Landkarte so einzufärben, dass zwei benachbarte Länder nicht die gleiche Farbe haben"), kann derzeit nur mit Hilfe eines sehr komplexen Computerprogramms durchgeführt werden, das ursprünglich 1200 Stunden Rechenzeit erforderte. Es hat eine beträchtliche Debatte darüber gegeben, ob man einem solchen Beweis trauen kann und ob ein solcher Beweis „echte" Mathematik ist [65].

Unter normalen Umständen würde jedoch selbst ein skeptischer Mathematiker ein sehr großes Vertrauen in das Ergebnis der Multiplikation zweier Zahlen auf einem Taschenrechner haben, auch wenn die genauen Einzelheiten des Vorgangs vor dem Benutzer verborgen sind. Selbst der Überprüfung, dass eine große Zahl eine Primzahl ist, durch einen Supercomputer wird vertraut, vor allem, wenn es eine unabhängige Überprüfung gibt; niemand macht sich die Mühe über die philosophische Bedeutung des „Beweises" zu diskutieren, obwohl der eigentliche Beweis so umfangreich wäre, dass es völlig unpraktisch wäre, ihn jemals auf Papier zu bringen. Es scheint, dass man dann dem Computer vertrauen kann, wenn der vom Computer verwendete Algorithmus einfach genug ist, um leicht verstanden zu werden.

Metamath macht sich diese Philosophie zu eigen. Die Einfachheit der Sprache macht es leicht sie zu erlernen, und aufgrund seiner Einfachheit kann man prinzipiell absolut sicher sein, dass ein Beweis korrekt ist. Alle Axiome, Regeln und Definitionen sind jederzeit einsehbar, da sie vom Benutzer selbst definiert werden. Es gibt keine versteckten oder eingebauten Regeln, die anfällig für subtile Bugs sein könnten. Der grundlegende Algorithmus, im Herzstück von Metamath ist einfach und feststehend, und es kann mit einem an Gewissheit grenzenden Grad an Vertrauen davon ausgegangen werden, dass er fehlerfrei und robust ist. Unabhängig geschriebene Implementierungen des Metamath-Verifizierers können den Restzweifel eines Skeptikers noch weiter reduzieren. Es gibt inzwischen mehr als ein Dutzend solcher Implementierungen, die von vielen verschiedenen Personen geschrieben wurden.

1.2.2 Dem Mathematiker vetrauen

> *Es gibt keinen Algebraiker oder Mathematiker, der in seiner Wissenschaft so bewandert ist, dass er sofort volles Vertrauen in eine von ihm entdeckte Gültigkeit setzt oder sie als etwas anderes als eine bloße Wahrscheinlichkeit betrachtet. Jedes Mal, wenn er seine Beweise durchgeht, wächst sein Vertrauen; aber noch mehr durch die Zustimmung seiner Freunde; und wird durch die allgemeine Zustimmung und den Beifall der gelehrten Welt zu seiner höchsten Vollkommenheit erhoben.*
>
> David Hume[35]

[35] *A Treatise of Human Nature*, Frei übersetzt nach dem Zitat in [16], S. 267.

Stanislaw Ulam schätzt, dass Mathematiker jedes Jahr 200.000 Theoreme veröffentlichen. Einige davon werden später widerlegt oder anderweitig nicht anerkannt, andere werden angezweifelt, und die meisten werden ignoriert.

RICHARD DE MILLO ET. AL.[36]

Unabhängig davon, ob man dem Computer vertrauen kann oder nicht, werden sich Menschen natürlich gelegentlich irren. Nur die denkwürdigsten Beweise werden von unabhängiger Seite verifiziert, und von diesen erlangen nur eine Handvoll wirklich großartiger Beweise den Status von „bekannten" mathematischen Wahrheiten, die ohne einen zweiten Gedanken an ihre Richtigkeit verwendet werden.

Es gibt viele berühmte Beispiele für falsche Theoreme und Beweise in der mathematischen Literatur.

- Es gibt Tausende von angeblichen Beweisen für den Großen Fermatschen Satz („keine ganzzahligen Lösungen existieren zu $x^n + y^n = z^n$ für $n > 2$"), von Amateuren, Spinnern und angesehenen Mathematikern [64, S. 5]. Fermat schrieb eine Notiz in sein Exemplar von Bachets *Diophantus*, dass er „hierfür einen wahrhaft wunderbaren Beweis entdeckt [hat], doch ist dieser Rand hier zu schmal, um ihn zu fassen."[33, S. 507]. Ein neuerer, viel beachteter Beweis von Yoichi Miyaoka wurde als falsch erwiesen (*Science News*, April 9, 1988, S. 230). Das Theorem wurde schließlich von Andrew Wiles (*Science News*, 3. Juli 1993, S. 5) bewiesen, aber der Beweis wies anfangs einige Lücken auf und es dauerte über ein Jahr nach seiner Bekanntgabe, bis er gründlich von Experten überprüft wurde. Am 25. Oktober 1994 gab Wiles bekannt, dass die letzte Lücke in seinem Beweis geschlossen worden sei.

- 1882 entdeckte M. Pasch, dass in Euklids Formulierung der Geometrie ein Axiom ausgelassen wurde; ohne dieses Axiom sind die Beweise der wichtigen Theoreme von Euklid nicht gültig. Das Axiom von Pasch besagt, dass eine Linie, die eine Seite eines Dreiecks schneidet, auch eine andere Seite schneiden muss, vorausgesetzt, sie berührt keine der Scheitelpunkte. Das Fehlen des Axioms von Pasch blieb, trotz der (vermutlich) Tausenden von Studenten, Lehrern und Mathematikern, die Euklid studierten, 2000 Jahre lang unbemerkt.

- Der erste veröffentlichte Beweis des berühmten Schröder–Bernstein-Theorems in der Mengenlehre war unkorrekt[18, p. 148]. Dieses Theorem besagt, dass wenn es eine 1-zu-1-Funktion [37] von der Menge A in

[36] Frei übersetzt nach [16], S. 269.

[37] Eine *Menge* ist eine beliebige Sammlung von Objekten. Eine *Funktion* oder *Zuordnung* ist eine Regel, die jedem Element einer Menge (dem sogenannten *Definitionsbereich*) ein Element aus einer anderen Menge zuordnet.

die Menge B und umgekehrt gibt, dann haben die Mengen A und B eine 1-zu-1-Entsprechung. Obwohl es einfach und offensichtlich klingt, ist der Standardbeweis recht lang und komplex.

- In den frühen 1900er Jahren veröffentlichte Hilbert einen angeblichen Beweis für die Kontinuumshypothese, die schließlich 1963 von Cohen als unbeweisbar festgestellt wurde[18, S. 166]. Die Kontinuumshypothese besagt, dass keine Unendlichkeit („transfinite Kardinalzahl“) existiert, deren Größe (oder „Kardinalität“) zwischen der Größe der Menge der ganzen Zahlen und der Größe der Menge der reellen Zahlen liegt. Diese Hypothese geht auf den deutschen Mathematiker Georg Cantor in den späten 1800er Jahren zurück, und seine Unfähigkeit, sie zu beweisen soll zu seiner Geisteskrankheit beigetragen haben, die ihn in seinen späteren Jahren plagte.

- Ein falscher Beweis für den Vier-Farben-Satz wurde von Kempe 1879 veröffentlicht[13, S. 582]; er bestand 11 Jahre lang, bevor sein Fehler entdeckt wurde. Dieses Theorem besagt, dass jede Karte mit nur vier Farben gefärbt werden kann, so dass keine zwei benachbarten Länder die gleiche Farbe haben. Im Jahr 1976 wurde das Theorem schließlich durch den berühmten computergestützten Beweis von Haken, Appel und Koch bewiesen [65]. Zumindest scheint es so zu sein. Der Mathematiker H. S. M. Coxeter hat Zweifel [15, S. 58]: „Ich habe das Gefühl, dass das eine unsaubere Art der Nutzung der Computer ist, und je mehr man mit Haken und Appel korrespondiert, desto wackeliger scheint sie zu sein.“

- Viele falsche „Beweise“ der Poincaré-Vermutung sind im Laufe der Jahre vorgeschlagen worden. Diese Vermutung besagt, dass jedes Objekt, das sich mathematisch wie eine dreidimensionale Kugel verhält, topologisch eine dreidimensionale Kugel ist, unabhängig davon, wie sehr es verzerrt ist. Im März 1986 haben die Mathematiker Colin Rourke und Eduardo Rêgo Aufsehen in der mathematischen Gemeinschaft erregt, als sie verkündeten, einen Beweis gefunden zu haben; im November desselben Jahres stellte sich heraus, dass der Beweis falsch war [53, S.218]. Sie wurde schließlich 2003 von Grigory Perelman bewiesen [66].

Viele Gegenbeispiele zu „Theoremen“ in der neueren mathematischen Literatur im Zusammenhang mit Clifford-Algebren wurden von Pertti Lounesto (der 2002 verstorben ist) gefunden. Siehe dazu `http://mathforum.org/library/view/4933.html`.

Einer der Verwendungszwecke von Metamath ist es, Beweise mit absoluter Präzision zu formulieren. Die Entwicklung eines Beweises in der Metamath-Sprache kann eine Herausforderung sein, weil Metamath nicht einmal den kleinsten Fehler zulässt. Ist der Beweis jedoch einmal erstellt,

kann man sich sofort auf seine Korrektheit verlassen, ohne dass man zur Bestätigung auf den Prozess der Peer Reviews angewiesen ist.

1.3 Der Einsatz von Computern in der Mathematik

1.3.1 Computeralgebrasysteme

In den meisten Fällen werden Sie feststellen, dass Metamath kein praktisches Werkzeug zum Rechnen mit Zahlen ist. (Selbst der Beweis, dass $2 + 2 = 4$, kann ziemlich komplex sein, wenn Sie mit der Mengenlehre beginnen). Mehrere kommerzielle Softwareprogramme für Mathematik sind ziemlich gut in Arithmetik, Algebra und Infinitesimalrechnung, und als praktische Werkzeuge sind sie von unschätzbarem Wert. Aber sie haben keine Möglichkeiten mit Beweisen umzugehen, und können keine Aussagen verstehen, die mit „Es gibt solche und solche...“ beginnen.

Softwareprogramme wie Mathematica[76] beschäftigen sich nicht mit Beweisen, sondern arbeiten direkt mit bekannten Ergebnissen. Diese Programme setzen vor allem auf heuristische Regeln wie die Ersetzung von Gleichen für Gleiches, um einfachere Ausdrücke oder Ausdrücke in einer anderen Form zu erhalten. Ausgehend von einer reichhaltigen Sammlung eingebauter Regeln und Algorithmen können die Benutzer mit Hilfe einer leistungsfähigen Programmiersprache weitere solche ergänzen. Allerdings können Ergebnisse wie z. B. die Existenz eines bestimmten abstrakten Objekts ohne dass das tatsächliche Objekt angegeben wird in ihren Sprachen nicht (direkt) ausgedrückt werden. Die Möglichkeit, einen Beweis aus einer kleinen Menge von Axiomen zu erstellen, fehlt. Stattdessen geht diese Software einfach davon aus, dass jedes Faktum oder jede Regel, die Sie der eingebauten Sammlung von Algorithmen hinzufügen, gültig ist. Somit stellt solch eine Software eine große Sammlung von Axiomen dar, aus der die Software unter gegebenen Zielsetzungen versucht, neue Theoreme abzuleiten, z. B. die Gleichheit eines komplexen Ausdrucks mit einem einfacheren Äquivalent. Aber die Begriffe „Theorem“ und „Beweis“ werden zum Beispiel nicht einmal im Index des Benutzerhandbuchs für Mathematica erwähnt. Aus philosophischer Sicht unbefriedigend ist auch, dass es keine Möglichkeit gibt, die Gültigkeit der Ergebnisse zu gewährleisten, als dem Autor jedes einzelnen Anwendungsmoduls zu vertrauen oder jedes Modul mühsam von Hand zu überprüfen, ähnlich wie ein Computerprogramm auf Fehler zu überprüfen. [38] Obwohl sie

[38] Zwei Beispiele verdeutlichen, warum die Wissensdatenbasis von Computeralgebrasystemen manchmal mit einer gewissen Vorsicht zu betrachten ist. Wenn Sie Mathematica (Version 3.0) auffordern: `Solve[x^n + y^n == z^n , n]`, dann wird es antworten: `{{n->-2}, {n->-1}, {n->1}, {n->2}}`. Mit anderen Worten, Mathematica scheint zu „wissen“, dass der Große Fermatsche Satz wahr ist! Zum Zeitpunkt als diese Version von Mathematica veröffentlicht wurde, war diese Tatsache jedoch noch nicht bekannt. Wenn

in der angewandten Mathematik natürlich äußerst hilfreich sind, sind Computeralgebrasysteme für den theoretischen Mathematiker eher von geringem Interesse, außer als Hilfsmittel zur Erforschung bestimmter spezifischer Probleme.

Wegen möglicher Fehler würde ein bloßes Vertrauen in die Ausgabe eines Computeralgebrasystems zur Verwendung als Theorem in einem Beweisverifizierer dessen Ziel der Strenge verfehlen. Andererseits ist zwar eine Tatsache, dass eine bestimmte relativ große Zahl eine Primzahl ist, für ein Computeralgebrasystem leicht herzuleiten, könnte aber einen langen, mühsamen Beweis haben, der einen Beweisverifizierer überfordern könnte. Ein Ansatz zur Verknüpfung von Computeralgebrasystemen mit einem Beweisverifizierer unter Beibehaltung der Vorteile beider Systeme besteht darin, jedem dieser Theoreme eine Hypothese hinzuzufügen, die seine Quelle angibt. Zum Beispiel könnte eine Konstante MAPLE anzeigen, dass das Theorem aus dem Maple Paket stammt, und würde bedeuten: „Angenommen Maple ist konsistent, dann...“ Dieses und viele andere Themen, welche die Formalisierung der Mathematik betreffen, werden in John Harrisons sehr interessanter Dissertation [23] diskutiert.

1.3.2 Automatische Theorembeweiser

Eine mathematische Theorie ist „entscheidbar“, wenn eine mechanische Methode oder ein Algorithmus existiert, der garantiert feststellen kann, ob eine bestimmte Formel ein Theorem ist. Zu den wenigen Theorien, die entscheidbar sind, gehört die Elementargeometrie, wie ein klassisches Ergebnis des Logikers Alfred Tarski im Jahr 1948 [68] zeigt. [39] Aber die meisten Theorien, einschließlich der elementaren Arithmetik, sind unentscheidbar. Diese Tatsache trägt dazu bei, die Mathematik am Leben zu erhalten, da viele Mathematiker glauben, dass sie niemals durch Computer ersetzt werden (wenn sie Roger Penroses Argument glauben, dass ein Computer niemals das Gehirn ersetzen kann [52]). In der Tat wird die Elementargeometrie oft als „toter“ Bereich betrachtet, und zwar aus dem einfachen Grund, dass sie entscheidbar ist.

Andererseits bedeutet die Unentscheidbarkeit einer Theorie nicht, dass man nicht einen Computer für die Suche nach Beweisen verwenden kann.

Sie Maple auffordern: `solve(x^2 = 2^x)`, danach: `simplify({"})`, gibt es die Lösungsmenge `{2, 4}` zurück, offenbar nicht wissend, dass $-0.7666647\ldots$ auch eine Lösung ist.

[39] Tarskis Ergebnis gilt eigentlich für eine Teilmenge der Geometrie, die in elementaren Lehrbüchern behandelt wird. Diese Untermenge umfasst das meiste von dem, was man als elementare Geometrie bezeichnen würde, aber sie ist nicht stark genug, um unter anderem die Begriffe Umfang und Fläche eines Kreises auszudrücken. Die Theorie so zu erweitern, dass sie Begriffe wie diese enthält, macht die Theorie unentscheidbar, wie auch von Tarski gezeigt wurde. Tarskis Algorithmus ist viel zu ineffizient, um ihn praktisch auf einem Computer zu implementieren. Ein praktischer Algorithmus zum Beweisen einer kleineren Untermenge von Geometrie-Theoremen (die nicht die Konzepte der „Ordnung“ oder „Kontinuität“ beinhalten) wurde von dem chinesischen Mathematiker Wu Wen-tsün im Jahr 1977 [12] entdeckt.

Man muss jedoch bereit sein, die Suche nach einer angemessenen Zeitspanne abzubrechen, wenn bis dahin kein Beweis gefunden wurde. Das Gebiet des automatischen Theorembeweisens ist spezialisiert auf die Durchführung solcher Suchen mittels Computern. Zu den bisher erfolgreichsten Ergebnissen gehören diejenigen, die auf einem Algorithmus, der als Robinsons Resolutionsprinzip bekannt ist[56] basieren.

Automatische Theorembeweiser können hervorragende Werkzeuge sein, wenn man bereit ist zu lernen, wie man sie benutzt. Aber sie sind in der reinen Mathematik nicht weit verbreitet, obwohl sie wahrscheinlich in vielen Bereichen nützlich sein könnten. Es gibt dafür mehrere Gründe. Der wohl wichtigste ist, dass das Hauptziel in der reinen Mathematik ist, zu Ergebnissen zu gelangen, die als tiefgründig oder wichtig wahrgenommen werden; sie zu beweisen ist zwar wichtig, aber zweitrangig. Normalerweise kann ein automatischer Theorembeweiser bei diesem Hauptziel nicht helfen, und wenn das Hauptziel erreicht ist, hat der Mathematiker vielleicht schon den Beweis als Nebenprodukt. Außerdem gibt es ein Problem mit der Notation. Mathematiker sind daran gewöhnt eine sehr kompakte Syntax zu verwenden, bei der ein oder zwei Symbole (die stark vom Kontext abhängen) sehr komplexe Konzepte darstellen können; dies ist Teil der Hierarchie, die sie aufgebaut haben, um schwierige Probleme zu bewältigen. Ein Theorembeweiser hingegen könnte voraussetzen, dass ein Theorem in „Logik erster Ordnung“ ausgedrückt wird, der Logik, auf die der größte Teil der Mathematik beruht, die aber normalerweise nicht direkt verwendet wird, da die Ausdrücke sehr lang werden können. Einige automatische Theorembeweiser sind experimentelle Programme, die in ihrer Anwendung auf sehr spezielle Bereiche beschränkt sind, und das Ziel vieler Programme ist einfach die Erforschung der Natur des automatischen Theorembeweisens selbst. Schließlich muss noch viel Forschung betrieben werden, um sehr tiefgreifende Theoreme zu beweisen. Ein wichtiges Ergebnis war ein Computerbeweis von Larry Wos und Kollegen, dass jede Robbins-Algebra eine Boolesche Algebra ist (*New York Times*, Dec. 10, 1996). [40]

Wie verhält sich Metamath zu automatischen Theorembeweisern? Ein

[40]Im Jahr 1933 präsentierte, E. V. Huntington das folgende Axiomensystem für die Boolesche Algebra mit einer unären Operation n und einer binären Operation $+$:

$$\begin{gathered} x + y = y + x \\ (x + y) + z = x + (y + z) \\ n(n(x) + y) + n(n(x) + n(y)) = x \end{gathered}$$

Herbert Robbins, ein Schüler von Huntington, vermutete, dass die letzte Gleichung durch eine einfachere Gleichung ersetzt werden kann:

$$n(n(x + y) + n(x + n(y))) = x$$

Robbins und Huntington konnten keinen Beweis finden. Das Problem wurde später erfolglos von Tarski und seinen Schülern untersucht, und es blieb ein ungelöstes Problem, bis ein Computer im Jahr 1996 den Beweis fand. Weitere Informationen zum Robbins-Algebra-Problem finden Sie unter [77].

Theorembeweiser befasst sich in erster Linie mit jeweils einem Theorem (vielleicht einer kleinen Datenbasis mit bekannten Theoremen), während Metamath eher wie ein Theoremarchivierungssystem, das sowohl das Theorem als auch seinen Beweis in einer Datenbasis für den Zugriff und die Überprüfung speichert. Metamath ist eine Antwort auf die Frage „Was macht man mit der Ausgabe eines Theorembeweisers?“ und könnte als der nächste Schritt in diesem Prozess betrachtet werden. Automatische Theorembeweiser könnten nützliche Werkzeuge sein, um bei der Entwicklung seiner Datenbasis zu helfen. Beachten Sie, dass sehr lange, automatisch generierte Beweise die Datenbasis fett und hässlich machen und dazu führen, dass Metamaths Beweisverifikation sehr viel Zeit in Anspruch nimmt. Wenn Sie nicht ein besonders gutes Programm haben, das sehr prägnante Beweise erzeugt, ist es vielleicht am besten, die Verwendung automatisch generierter Beweise als eine unsaubere Schnelllösung zu betrachten, die zu einem späteren Zeitpunkt manuell neu zu schreiben wäre.

Das Programm OTTER[41], später ersetzt durch prover9[42], hat einen historischen Einfluss gehabt. Der E-Prover[43] ist ein immer noch gepflegter automatischer Theorembeweiser für vollständige Logik erster Ordnung mit Gleichheit. DAneben gibt es noch viele andere automatische Theorembeweiser.

Wenn Sie automatische Theorembeweiser mit Metamath kombinieren wollen, sollten Sie sich mit dem Buch *Automated Reasoning: Introduction and Applications*[77] beschäftigen. In diesem Buch geht es darum wie man OTTER in einer solchen Weise einsetzt, dass es nicht nur in der Lage ist relativ effiziente Beweise zu erzeugen, sondern sogar angewiesen werden kann nach kürzeren Beweisen zu suchen. Die effektive Verwendung von OTTER (und ähnlicher Werkzeuge) erfordert jedoch ein gewisses Maß an Erfahrung, Sachkenntnis und Geduld. Das Axiomensystem, das in der `set.mm` Mengenlehre-Datenbasis verwendet wird, kann mit Hilfe einer Methode für OTTER ausgedrückt werden, die in [41] [44] beschrieben wird. Bei Erfolg neigt diese Methode in der Regel dazu, kurze und clevere Beweise zu generieren, aber meine Experimente mit ihr zeigen, dass die Methode in angemessener Zeit Beweise nur für relativ einfache Theoreme findet. Es macht trotzdem Spaß, damit zu experimentieren.

Referenz [7] gibt einen Überblick über eine Reihe von Ansätzen, die auf dem Gebiet des automatischen Theorembeweisens erforscht wurden.

[41] `http://www.cs.unm.edu/~mccune/otter/`.

[42] `https://www.cs.unm.edu/~mccune/mace4/`.

[43] `https://github.com/eprover/eprover`.

[44] Um diese Axiome mit OTTER zu verwenden, müssen sie in einer Weise umformuliert werden, die die Notwendigkeit von „Dummy Variablen“ beseitigt. Siehe den Kommentar auf S. 140.

1.3.3 Interaktive Theorembeweiser

Das vollautomatische Finden von Beweisen ist schwierig, daher gibt es einige interaktive Theorembeweiser, bei denen ein Mensch den Computer bei der Suche nach einem Beweis anleitet. Beispiele hierfür sind HOL Light[45], Isabelle[46], HOL[47], und Coq[48].

Ein wesentlicher Unterschied zwischen den meisten dieser Werkzeuge und Metamath ist, dass die „Beweise“ eigentlich Programme sind, die das Programm anleiten, einen Beweis zu finden, und nicht der Beweis selbst. Zum Beispiel könnte ein Isabelle/HOL-Beweis einen Schritt `apply (blast dest: rearrange reduction)` anwenden. Die `blast` Anweisung verwendet einen automatischen Tableaux-Beweiser und wird beendet, wenn eine Folge von gültigen Beweisschritten gefunden wurde... aber diese Folge wird nicht als Teil des Beweises betrachtet.

Ein guter Überblick zu übergeordneten Sprachen für Beweisverifikationen (wie LCF und HOL) ist in [22] angegeben. Alle diese Sprachen unterscheiden sich grundlegend von Metamath dadurch, dass ein Großteil des mathematischen Grundwissens in das zugrundeliegende Beweisprogramm und nicht direkt in die zu prüfende Datenbasis eingebettet ist. Diese Sprachen können eine steile Lernkurve für diejenigen erfordern, die keinen mathematischen Hintergrund haben. Zum Beispiel muss man in der Regel ein gewisses Verständnis für mathematische Logik haben, um ihren Beweisen folgen zu können.

1.3.4 Beweisverifizierer

Ein Beweisverifizierer ist ein Programm, das keine Beweise erzeugt, sondern Beweise die im vorgegeben werden verifiziert. Viele Beweisverifizierer haben eingeschränkte eingebaute automatisierte Beweisfähigkeiten, wie z. B. das Herausfinden einfacher logischer Schlüsse (wobei sie immer noch von einer Person angeleitet werden, die den Gesamtbeweis liefert). Metamath hat keine eingebauten automatischen Beweisfähigkeiten, außer der begrenzten Fähigkeit seines Beweis-Assistenten.

Sprachen zur Beweisverifikation werden nicht so häufig verwendet, wie sie es könnten. Reine Mathematiker sind mehr damit beschäftigt, neue Ergebnisse zu erzielen, und solche Detailgenauigkeit und Strenge würden diesem Ziel im Wege stehen. Der Einsatz von Computern in der reinen Mathematik konzentriert sich in erster Linie auf automatische Theorembeweiser (nicht auf Verifizierer), wiederum mit dem Ziel, die Schaffung neuer Mathematik zu unterstützen. Automatische Theorembeweiser befassen sich in der Regel damit, jeweils ein Theorem anzugehen, anstatt dem Benutzer eine

[45] `https://www.cl.cam.ac.uk/~jrh13/hol-light/`.
[46] `http://www.cl.cam.ac.uk/Research/HVG/Isabelle`.
[47] `https://hol-theorem-prover.org/`.
[48] `https://coq.inria.fr/`.

große, organisierte Datenbasis zur Verfügung zu stellen. Metamath ist eine Möglichkeit, diese Lücke zu schließen.

An sich ist Metamath hauptsächlich ein Beweisverifizierer. Das bedeutet nicht, dass andere Ansätze nicht verwendet werden können; der Unterschied ist, dass in Metamath die Ergebnisse der verschiedenen Beweise Schritt für Schritt aufgezeichnet werden müssen, damit sie verifiziert werden können.

Eine weitere Sprache zur Überprüfung von Beweisen ist Mizar,, die ihre Beweise in der informellen Sprache darstellen kann, an die Mathematiker gewöhnt sind. Informationen über die Mizar-Sprache finden Sie unter `http://mizar.org`.

Für den tätigen Mathematiker ist Mizar ein ausgezeichnetes Werkzeug um Beweise streng zu dokumentieren. Mizar stellt seine Beweise in dem informellen Englisch dar, das von Mathematikern verwendet wird (und, obwohl sie für diese ausreichen, für Laien genauso unverständlich sind!). Der Preis für Mizar ist eine relativ steile Lernkurve von einigen Wochen. Mehrere Mathematiker arbeiten aktiv an der Formalisierung verschiedener Bereiche der Mathematik mit Mizar und veröffentlichen die Beweise in einer speziellen Zeitschrift. Leider wird die Aufgabe, Mathematik zu formalisieren, immer noch in gewissem Maße geringgeschätzt, da sie keine „neue“ Mathematik hervorbringt.

Das System, das Metamath am nächsten kommt, ist die *Ghilbert*-Beweissprache (`http://ghilbert.org`), entwickelt von Raph Levien. Ghilbert ist ein formaler Beweisprüfer, der stark von Metamath inspiriert ist. Ghilbert-Anweisungen sind s-Ausdrücke (à la Lisp), die für Computer leicht zu analysieren sind, aber viele Menschen finden sie schwer zu lesen. Es gibt eine Reihe von Unterschieden in ihren spezifischen Konstrukten, aber es gibt zumindest ein Werkzeug, um einige Metamath-Daten in Ghilbert zu übersetzen. Nach dem Stand von 2019 ist die Ghilbert-Gemeinschaft kleiner und weniger aktiv als die Metamath-Gemeinschaft. Dennoch gibt es Überschneidungen zwischen der Metamath- und der Ghilbert-Gemeinschaft, und im Laufe der Jahre haben viele Male fruchtbare Gespräche zwischen ihnen stattgefunden.

1.3.5 Erstellung einer Datenbasis für formalisierte Mathematik

Neben Metamath gibt es mehrere andere laufende Projekte mit dem Ziel, die Mathematik in durch Computer verifizierbare Datenbasen zu formalisieren. Um sie zu verstehen hilft ein Rückblick auf deren Historie.

Das QED[49] Projekt wurde 1993 ins Leben gerufen und seine Ziele wurden in dem QED Manifest umrissen. Das QED Manifest war ein Vorschlag für eine computergestützte Datenbasis für das gesamte mathematische Wissen, streng formalisiert und mit allen Beweisen, die automatisch überprüft

[49] `http://www-unix.mcs.anl.gov/qed`.

wurden. Zu diesem Projekt fand eine Konferenz im Jahr 1994 und eine weitere im Jahr 1995 statt. Außerdem gab es einen Workshop „twenty years of the QED manifesto“ im Jahr 2014. Seine Ideale werden regelmäßig wieder aufgegriffen.

In einem Papier aus dem Jahr 2007 nennt Freek Wiedijk zwei Gründe für das Scheitern des QED Projekts in der ursprünglich geplanten Form:[75]

- Nur sehr wenige Menschen arbeiten an der Formalisierung der Mathematik. Es gibt keine zwingende Anwendung für vollständig mechanisierte Mathematik.
- Formalisierte Mathematik entspricht noch nicht der traditionellen Mathematik. Das liegt zum einen an der Komplexität der mathematischen Notation und zum anderen an den Grenzen der vorhandenen Theorembeweiser und Beweis-Assistenten.

Doch damit war der Traum von der Formalisierung der Mathematik in durch Computer verifizierbaren Datenbasen noch nicht beendet. Die Probleme, die zum QED Manifest führten, sind immer noch aktuell, auch wenn die Herausforderungen schwieriger waren als ursprünglich angenommen. Stattdessen sind verschiedene unabhängige Projekte entstanden, die an der Formalisierung der Mathematik in durch Computer verifizierbaren Datenbasen gearbeitet haben, die gleichzeitig miteinander konkurrieren und kooperieren.

Eine konkrete Möglichkeit, diese Projekte zu vergleichen, ist Freek Wiedijks Liste „Formalisierung von 100 Theoremen“,[50] die zeigt, welche Fortschritte verschiedene Systeme beim Finden von Beweisen für eine Liste von 100 mathematischen Theoremen gemacht haben.[51] Die Top-Systeme im Februar 2019 (in der Reihenfolge der Anzahl der abgeschlossenen Beweise) sind HOL Light, Isabelle, Metamath, Coq und Mizar[52].

Die Metamath 100[53] Seite (betreut von David A. Wheeler) zeigt den Fortschritt von Metamath (insbesondere der `set.mm` Datenbasis) im Vergleich zu dieser von Freek Wiedijk geführten Challenge-Liste. Die Metamath-`set.mm`Datenbasis hat im Laufe der Jahre große Fortschritte gemacht, zum Teil deshalb, weil die Arbeit an den Beweisen dieser Theoreme die Definition verschiedener Begriffe und die Beweise ihrer Eigenschaften als Voraussetzung erforderlich machten. Hier sind nur einige der vielen Sätze, die formal mit Metamath bewiesen wurden[54]:

- 1. Die Irrationalität der Quadratwurzel aus 2 (`sqr2irr`, von Norman Megill, 20. August 2001)

[50] `http://www.cs.ru.nl/%7Efreek/100/`.

[51] Dies ist nicht die einzige Liste von „interessanten“ Theoremen. Eine weitere interessante Liste wurde von Oliver Knill veröffentlicht [32].

[52] Stand 16.10.2023: Isabelle(89), HOL Light(87), Coq(79), Lean(76), Metamath(74) und Mizar(69)

[53] `http://us.metamath.org/mm_100.html`

[54] Anm. der Übersetzer: Eine vollständige Liste befindet sich in Anhang F

- 2. Der Fundamentalsatz der Algebra (`fta`, von Mario Carneiro, 15. September 2014)
- 22. Die Nicht-Abzählbarkeit des Kontinuums (`ruc`, von Norman Megill, 13. August 2004)
- 54. Das Königsberger Brückenproblem (`konigsberg`, von Mario Carneiro, 16. April 2015)
- 83. Der Freundschaftssatz (`friendship`, von Alexander W. van der Vekens, 9. Oktober 2018)

Wir danken all denen, die mindestens einen der Metamath 100-Beweise entwickelt haben, und wir danken insbesondere Mario Carneiro, der im Jahr 2019 die meisten Metamath 100-Beweise beigesteuert hat. Die Metamath 100-Seite zeigt die Liste aller Personen, die einen Beweis beigetragen haben, sowie Links zu Grafiken und Diagrammen, die den Fortschritt im Laufe der Zeit zeigen. Wir ermutigen andere, an Beweisen für Theoreme zu arbeiten, die noch nicht in Metamath bewiesen wurden, da dies das Werk insgesamt verbessert.

Jedes der mathematischen Formalisierungssysteme (einschließlich Metamath) hat unterschiedliche Stärken und Schwächen, je nachdem, worauf Sie Wert legen. Die wichtigsten Aspekte, die Metamath von den anderen Top-Systemen unterscheiden, sind:

- Metamath ist nicht an einen bestimmten Satz von Axiomen gebunden.
- Metamath kann jeden Schritt von jedem Beweis anzeigen, ohne Ausnahmen. Die meisten anderen Beweiser behaupten nur, dass ein Beweis gefunden werden kann, und zeigen nicht jeden Schritt. Das macht auch die Verifikation schnell, da das System die Details eines Beweises nicht neu ermitteln muss.
- Der Metamath-Prüfer wurde in vielen verschiedenen Programmiersprachen erneut implementiert, so dass die Verifikation durch mehrere Implementierungen durchgeführt werden kann. Insbesondere die `set.mm` Datenbasis wird von vier verschiedenen Verifizierern verifiziert, die in vier verschiedenen Sprachen von vier verschiedenen Autoren geschrieben wurden[55] . Dadurch wird das Risiko der Annahme eines ungültigen Beweises aufgrund eines Fehlers im Verifizierer reduziert.

[55] Anm. der Übersetzer: Damit sind die vier Verifizierer gemeint, die bei jedem Git-Merge für Änderungen an `set.mm` durchgeführt werden: der originale C-Verifizierer von Norman Megill, der Rust-Verifizierer von Stefan O'Rear, der Java-Verifizierer von Mel L. O'Cat und Mario Carneiro und der Python-Verifizierer von Raph Levin. Es gibt daneben noch viele andere Verifizierer, siehe `http://us.metamath.org/other.html`.

- Beweise bleiben bewiesen. In einigen Systemen können Änderungen an der Syntax des Systems oder an der Funktionsweise einer Taktik dazu führen, dass Beweise in späteren Versionen nicht mehr funktionieren, wodurch ältere Arbeiten im Grunde verloren gehen. Die Metamath-Sprache ist extrem klein und stabil, so dass sobald ein Beweis einmal zu einer Datenbasis hinzugefügt wurde, kann die Datenbasis mit späteren Versionen des Metamath-Programms und mit anderen Verifizierern von Metamath-Datenbasen überprüft werden. Wenn ein Axiom oder eine Schlüsseldefinition geändert werden muss, ist es einfach, die Datenbasis als Ganzes zu manipulieren, um die Änderung zu verarbeiten ohne den zugrunde liegenden Verifizierer zu verändern. Da die erneute Verifizierung einer gesamten Datenbasis nur Sekunden dauert, gibt es nie einen Grund, die vollständige Überprüfung hinauszuzögern. Dieser Aspekt ist besonders überzeugend, wenn man eine langfristig nutzbare Datenbasis mit Beweisen haben möchte.

- Die Lizenzierung ist großzügig. Die wichtigsten Metamath-Datenbasen sind öffentlich verfügbar, und das Metamath-Hauptprogramm ist Open-Source-Software unter einer standardisierten, weit verbreiteten Lizenz.

- Substitutionen sind leicht zu verstehen, auch für diejenigen, die keine professionellen Mathematiker sind.

Natürlich können andere Systeme Vorteile gegenüber Metamath haben, die geeigneter sind, je nachdem, worauf Sie Wert legen. In jedem Fall hoffen wir, dass Ihnen diese Ausführungen helfen, Metamath in einem größeren Kontext zu verstehen.

1.3.6 Zusammenfassung

Unsere Diskussion über Computer und Mathematik kann wie folgt zusammengefasst werden: Computeralgebrasysteme können als Theoremgeneratoren betrachtet werden, die sich auf einen eingeschränkten Bereich der Mathematik (Zahlen und ihre Eigenschaften) konzentrieren, automatische Theorembeweiser als Beweisgeneratoren für spezifische Theoreme in einem viel breiteren Bereich, der durch ein eingebautes formales System wie die Logik erster Ordnung abgedeckt wird, interaktive Theorembeweiser erfordern menschliche Anleitung, Beweisverifizierer verifizieren Beweise, aber historisch gesehen waren sie auf die Logik erster Ordnung beschränkt. Im Gegensatz dazu ist Metamath ein Beweisverifizierer und Dokumentierer, dessen Anwendungsbereich im Wesentlichen unbegrenzt ist.

1.4 Die Mathematik und Metamath

1.4.1 Standard-Mathematik

Es gibt eine Reihe von Möglichkeiten, Metamath für die Standard-Mathematik zu benutzen. Der philosophisch befriedigendste Weg ist, ganz am Anfang zu beginnen und die gewünschte Mathematik aus den Axiomen der Logik und Mengenlehre zu entwickeln. Dies ist der Ansatz, der in der `set.mm` Datenbasis (auch bekannt als Metamath Proof Explorer) angewandt wird. Diese Datenbasis baut unter anderem auch auf den Axiomen der reellen und komplexen Zahlen auf (siehe Abschnitt 3.7), und eine Standardentwicklung der Analysis könnte beispielsweise an diesem Punkt ansetzen, basierend auf den übrigen Theoremen und Sätzen dieser Datenbasis. Neben diesem philosophischen Vorteil gibt es auch praktische Vorteile, alle Werkzeuge der Mengenlehre in der unterstützenden Infrastruktur zur Verfügung zu haben.

Andererseits möchte man vielleicht mit den Standardaxiomen einer mathematischen Theorie beginnen, ohne die mengentheoretischen Beweise dieser Axiome berücksichtigen zu müssen. Sie werden die mathematische Logik benötigen, um Schlussfolgerungen zu ziehen, aber wenn Sie möchten können Sie einfach die Theoreme der Logik wo immer man sie braucht als „Axiome“ einführen, unter der impliziten Annahme, dass sie im Prinzip bewiesen werden können, wenn sie für Sie offensichtlich sind. Wenn Sie diesen Ansatz wählen, werden Sie wahrscheinlich die in `set.mm` verwendete Notation überprüfen, damit sie mit Ihrer eigenen Notation zusammenpasst.

1.4.2 Andere formale Systeme

Im Gegensatz zu anderen Programmen ist Metamath weder auf einen bestimmten Bereich der Mathematik beschränkt, noch einer bestimmten mathematischen Philosophie verpflichtet wie z.B. der klassischen oder intuitionistischer Logik, noch beschränkt auf Ausdrücke der Logik erster Ordnung. Obwohl die Datenbasis `set.mm` die Standardlogik und Mengenlehre beschreibt, ist Metamath eigentlich eine Allzwecksprache zur Beschreibung einer Vielzahl formaler Systeme. Nicht-Standard-Systeme wie die modale Logik, intuitionistische Logik, Logik höherer Ordnung, Quantenlogik, und Kategorientheorie können alle mit der Metamath-Sprache beschrieben werden. Sie definieren die von Ihnen bevorzugten Symbole und teilen Metamath die Axiome und Regeln mit, von denen Sie ausgehen wollen, und Metamath verifiziert alle Schlüsse, die Sie aus diesen Axiomen und Regeln ziehen. Ein einfaches Beispiel für ein nicht standardisiertes formales System ist Hofstadters MIU-System, dessen Metamath-Version im Anhang D dargestellt wird.

Diese Nutzungsmöglichkeiten von Metamatrh sind nicht nur hypothetisch. Die größte Metamath-Datenbasis ist `set.mm`, auch bekannt als der Metamath Proof Explorer, der die gebräuchlichsten Axiome für mathema-

tische Grundlagen verwendet (insbesondere die klassische Logik kombiniert mit der Zermelo–Fraenkel Mengenlehre zusammen mit dem Auswahlaxiom). Es sind aber auch andere Metamath-Datenbasen verfügbar:

- Die Datenbasis `iset.mm`, auch bekannt als Intuitionistic Logic Explorer, verwendet intuitionistische Logik (eine konstruktivistische Sichtweise) anstelle der klassischen Logik.
- Die Datenbasis `nf.mm`, auch bekannt als New Foundations Explorer, konstruiert die Mathematik von Grund auf neu, ausgehend von Quines New Foundations (NF) Axiomen der Mengenlehre.
- Die Datenbasis `hol.mm`, auch bekannt als Higher-Order Logic (HOL) Explorer, beginnt mit HOL (auch einfache Typentheorie genannt), leitet daraus Äquivalente zu den ZFC-Axiomen ab und verbindet so die beiden Ansätze.

Seit der Zeit von David Hilbert haben sich die Mathematiker darüber Gedanken gemacht, ob die Metasprache, die zur Beschreibung der Mathematik verwendet wird, mächtiger sein muss als die Mathematik, die beschrieben wird. Die finitistische, konstruktive Natur von Metamath bietet eine gute philosophische Grundlage für das Studium selbst der schwächsten Logiken.

Die Beschreibung von vielen, nicht-standardisierten formaler Systeme wird die Modelltheorie oder die Beweistheorie verwendet; diese Theorien basieren ihrerseits auf der Standard-Mengentheorie. Mit anderen Worten: Ein nicht-standardisiertes formales System ist definiert als eine Menge mit bestimmten Eigenschaften, und die Standard-Mengentheorie wird zur Herleitung zusätzliche Eigenschaften dieser Menge genutzt. Die Datenbasis der Standard-Mengentheorie, die mit Metamath zur Verfügung gestellt wird, kann für diesen Zweck verwendet werden. Und wenn sie auf diese Weise genutzt wird, ist die Entwicklung eines speziellen Axiomensystems für das nicht-standardisierte formale System überflüssig. Der modell- oder beweistheoretische Ansatz erlaubt es oft, viel tiefgreifendere Ergebnisse mit weniger Aufwand zu beweisen.

Metamath unterstützt beide Ansätze. Sie können das nicht-standardisierte formale System direkt definieren, oder das nicht-standardisierte formale System als eine Menge mit bestimmten Eigenschaften definieren, je nachdem, was Sie am geeignetsten finden.

1.4.3 Metamath und seine Philosophie

Metamath steht im Zusammenhang mit einer bestimmten Philosophie oder Betrachtungsweise der Mathematik. Diese Philosophie ist verwandt mit der formalistischen Philosophie von Hilbert und seinen Anhängern [30, S. 1203–1208] [4, S. 6]. In dieser Philosophie ist die Mathematik nichts weiter als eine Reihe von Regeln zur Handhabung von Symbolen, zusammen mit den

Folgerungen aus der Anwendung dieser Regeln. Während die beschriebene Mathematik komplex sein kann, sollten die Regeln, mit denen sie beschrieben wird (die „Metamathematik“) so einfach wie möglich sein. Insbesondere sollten sich die Beweise auf konkrete Objekte beziehen (die Symbole, die wir auf Papier schreiben, und nicht die abstrakten Konzepte, die sie repräsentieren) und mit ihnen in einer konstruktiven Weise umgehen; solche Beweise werden „finitistisch“ genannt [62, S. 2–3].

Ob Sie Metamath interessant oder nützlich finden, hängt zum Teil von der Anziehungskraft ab, die seine Philosophie auf Sie ausübt, und diese Anziehungskraft hängt wahrscheinlich von Ihren besonderen Zielen in Bezug auf die Mathematik ab. Wenn Sie zum Beispiel ein reiner Mathematiker sind, der an vorderster Front bei der Entdeckung neuer mathematischer Erkenntnisse steht, werden Sie wahrscheinlich Ihre Kreativität durch den starren Formalismus von Metamath als einschränkt erachten. Andererseits würden wir argumentieren, dass es vorteilhaft ist, solches Wissen in einem Standardformat zu dokumentieren, sobald es entdeckt ist, damit es daurch für andere zugänglich ist. In sechzig Jahren mag Ihr Wissensgebiet vielleicht eingeschlafen sein, und dann würden Ihre „Schriften noch schwieriger übersetzbar sein als die der Maya“, wie Davis und Hersh es ausdrücken[15, S. 37].

1.4.4 Die Hintergründe des Metamath-Ansatzes

Den wahrscheinlich stärksten Einfluss auf Metamath hatte Whiteheads und Russells monumentales Werk *Principia Mathematica* [74], dessen Ziel es war, die gesamte Mathematik aus einer kleinen Anzahl von primitiven Konzepten abzuleiten, und zwar auf eine sehr expliziten Weise, die im Prinzip jeder verstehen und nachvollziehen konnte. Während dieses Werk zu seiner Zeit sehr einflussreich war, hat es aus heutiger Sicht mehrere Nachteile. Sowohl die Notation als auch die zugrunde liegenden Axiome gelten heute als veraltet und werden nicht mehr verwendet. Von unserem Standpunkt aus ist ihre Entwicklung nicht wirklich so zugänglich, wie wir es gerne hätten; aus praktischen Gründen werden die Beweise mit fortschreitender mathematischer Tiefe immer skizzenhafter, und ihre Ausarbeitung im Detail erfordert ein Maß an mathematischem Geschick und Geduld, über die viele Menschen nicht verfügen. Es gibt zahlreiche kleine Fehler, was angesichts der langwierigen, technischen Natur der Beweise und dem Fehlen eines Computers zur Überprüfung der Details verständlich ist. Dennoch ist *Principia Mathematica* auch heute noch das Werk, das dem Geist von Metamath am nächsten kommt. Es bleibt ein verblüffendes Werk, und man kann nicht anders als staunen, wenn man sieht, dass „$1 + 1 = 2$“ schließlich auf Seite 83 von Band II erscheint (Theorem *110.643).

Der Ursprung der von Metamath verwendeten Beweisnotation geht auf die 1950er Jahre zurück, als der Logiker C. A. Meredith seine Beweise in einer kompakten Notation ausdrückte, die „kondensierte Ablösung“ (engl. „condensed detachment„) [25] [29] [45] [54] genannt wird. Diese Notation

ermöglicht die eindeutige Wiedergabe von Beweisen durch bloße Bezugnahme auf das Axiom, die Regel oder das Theorem, das bei jedem Schritt verwendet wird, ohne eine explizite Angabe der Substitutionen, die für die Variablen in diesem Axiom, dieser Regel oder diesem Theorem vorgenommen werden müssen. Gewöhnlich ist die kondensierte Ablösung mehr oder weniger auf die Aussagenlogik beschränkt. Das Konzept ist in [41] auf Logik erster Ordnung erweitert worden, wodurch das Schreiben eines kleinen Computerprogramms zur Überprüfung von Beweisen einfacher Theoreme der Logik erster Ordnung vereinfacht wird.

Ein Schlüsselkonzept hinter der Notation der kondensierte Ablösung ist die sogenannte „Vereinheitlichung“ (engl. „unification“), ein Algorithmus zur Bestimmung der Substitutionen, die für Variablen vorgenommen werden müssen, damit zwei Ausdrücke miteinander übereinstimmen. Die Vereinheitlichung wurde erstmals von dem Logiker J. A. Robinson genau definiert, der sie bei der Entwicklung einer leistungsfähigen Theorem-Beweis-Technik namens „Auflösungsprinzip“ (engl. „resolution principle“) verwendete[56]. Metamath macht keinen Gebrauch von diesem Auflösungsprinzip, das für Systeme der Logik erster Ordnung gedacht ist. Die Verwendung von Metamath ist nicht auf die Logik erster Ordnung beschränkt, und wie wir bereits erwähnt haben, findet es nicht automatisch Beweise. Allerdings ist die Vereinheitlichung eine Schlüsselidee hinter Metamaths Beweisnotation, und Metamath macht von einer sehr einfachen Version davon Gebrauch (Abschnitt 4.3.1).

1.4.5 Metamath und die Logik erster Ordnung

Die Logik erster Ordnung ist die grundlegende Struktur für die Standardmathematik. Darauf aufbauend gibt es die Mengenlehre mit den Axiomen, aus denen sich praktisch die gesamte Mathematik ableiten lässt — eine bemerkenswerte Tatsache.[56]

Einer der Aspekte, die Metamath für Theorien erster Ordnung praktikabler machen, ist ein Satz von Axiomen für die Logik erster Ordnung, der speziell für den Metamath-Ansatz entwickelt wurde. Diese sind enthalten in der Datenbasis `set.mm`. Siehe Kapitel 3 für eine detaillierte Beschreibung; die Axiome sind in Abschnitt 3.3 aufgeführt. Während es logisch äquivalent zu Standard-Axiomensystemen ist, bricht unser Axiomensystem die Standardaxiome in kleinere Teile auf, so dass man aus ihnen direkt ableiten kann, was in anderen Systemen nur als übergeordnete „Metatheoreme“ abgeleitet werden kann. Mit anderen Worten, es ist mächtiger als die Standardaxiome vom metalogischen Standpunkt aus gesehen. Eine strenge Rechtfertigung für

[56] Eine Ausnahme scheint die Kategorientheorie zu sein. Es gibt verschiedene Denkschulen darüber, ob die Kategorientheorie aus der Mengenlehre ableitbar ist. Zumindest scheint es so, dass ein zusätzliches Axiom erforderlich ist, das die Existenz eines „unzugänglichen Kardinals“ (eine Art von Unendlichkeit, die so groß ist, dass die Standard-Mengenlehren-Theorie ihre Existenz weder beweisen noch leugnen kann).

Für weitere Informationen siehe[24, pp. 328–331] und [6].

dieses System und seine „metalogische Vollständigkeit“ findet sich in [41]. Das System ist eng verwandt mit einem System, das von Monk und Tarski im Jahr 1965 [46] entwickelt wurde.

Zum Beispiel ist die Formel $\exists x\, x = y$ (zu einem gegebenen y existiert ein x, das diesem gleicht) ist ein Satz der Logik,[57], egal ob x und y verschiedene Variablen sind oder nicht. In vielen Systemen der Logik müsste man zwei Theoreme beweisen, um zu diesem Ergebnis zu kommen. Zuerst würden wir beweisen, dass „$\exists x\, x = x$“, dann würden wir separat beweisen: „$\exists x\, x = y$, wobei x und y verschiedene Variablen sind“. Wir würden dann diese beiden Spezialfälle „außerhalb des Systems“ (d.h. in unseren Köpfen) kombinieren, um behaupten zu können: „$\exists x\, x = y$, unabhängig davon, ob x und y verschieden sind“. Mit anderen Worten, die Kombination der beiden Spezialfälle ist ein Metatheorem. In dem System der Logik, das in der Mengenlehre von Metamath verwendet wird, sind die Axiome der Logik in kleine Teile zerlegt, die es erlauben, sie so zusammenzusetzen, dass Theoreme wie diese direkt bewiesen werden können.

Wenn man die Axiome auf diese Weise aufschlüsselt, sehen sie auf den ersten Blick seltsam und nicht sehr intuitiv aus, aber seien Sie versichert, dass sie korrekt und vollständig sind. Ihre Korrektheit ist gewährleistet, weil es sich um Theoremschemata der Standardlogik erster Ordnung handelt (was Sie leicht überprüfen können, wenn Sie Logiker sind). Ihre Vollständigkeit folgt aus der Tatsache, dass wir die Standardaxiome der Logik erster Ordnung explizit als Theoreme ableiten. Die Ableitung der Standardaxiome ist etwas schwierig, aber wenn wir es geschafft haben, haben wir ein System zur Verfügung, das weniger unbequem für die Arbeit mit formalen Beweisen ist. Ausgedrückt in technischen Begriffen der Logiker eliminieren wir die umständlichen Konzepte der „freien Variable“, der „gebundenen Variable“, und der „echten Substitution“ als primitive Begriffe. Diese Begriffe sind in unserem System vorhanden, werden aber durch konzeptionelle Begriffe definiert, die durch die Axiome ausgedrückt werden, und können im Prinzip eliminiert werden. In Standardsystemen werden diese Begriffe tatsächlich wie zusätzliche, implizite Axiome verwendet, die etwas komplex sind und nicht eliminiert werden können.

Die traditionelle Herangehensweise an die Logik, bei der freie Variable und die echte Substitution definiert ist, kann auch direkt in der Metamath-Sprache nachgebildet werden. Allerdings ist die Notation eher umständlich,

[57] Speziell ist es ein Satz derjenigen Systeme der Logik, die nicht-leere Wertebereiche voraussetzen. Es ist kein Theorem von allgemeineren Systemen, die den leeren Wertebereich einschließen, in dem nichts existiert, Punkt! Solche Systeme nennt man „freie Logiken.“ Für eine Diskussion dieser Systeme, siehe [36]. Da die Logik eine Grundlage für die Mengenlehre ist, die einen nicht leeren Wertebereich hat, ist es bequemer (und traditioneller), ein weniger allgemeines System zu verwenden. Eine interessante Kuriosität bei der Verwendung einer freie Logik als Grundlage für die Zermelo-Fraenkel-Mengenlehre (wobei das redundante Axiom der Existenz einer leeren Menge weggelassen wird) ist, dass nicht einmal die Existenz einer einzigen Menge bewiesen werden kann, ohne das Axiom der Unendlichkeit anzunehmen!

und es gibt Nachteile: zum Beispiel ist die Erweiterung der Definition einer wff mit einer Definition umständlich, weil die Definition der Konzepte der freien Variablen und der echten Substitution ebenfalls erweitert werden müssen. Unsere Wahl der Axiome für `set.mm` ist bis zu einem gewissen Grad eine Frage des Stil bei dem Versuch, eine übergreifende Einfachheit zu gewährleisten mit dem Bewusstsein, dass auch der traditionelle Ansatz, falls gewünscht, möglich ist.

Kapitel 2

Verwendung des Metamath-Programms

2.1 Installation

Die Art und Weise, wie Sie Metamath auf Ihrem Computersystem installieren, ist von Computer zu Computer verschieden. Aktuelle Anweisungen werden mit dem Download des Metamath-Programms bereitgestellt unter `http://metamath.org`. Im Allgemeinen ist die Installation einfach. Es gibt eine Datei, die das Metamath-Programm selbst enthält. Diese Datei heißt normalerweise `metamath` oder `metamath.`*xxx*, wobei *xxx* der Konvention (wie `exe`) für ein ausführbares Programm auf Ihrem Betriebssystem entspricht. Es gibt mehrere zusätzliche Dateien mit Beispielen für die Metamath-Sprache, die alle mit `.mm` enden. Die Datei `set.mm` enthält Logik und Mengenlehre und kann als Ausgangspunkt für andere Bereiche der Mathematik verwendet werden.

Sie benötigen außerdem einen Texteditor, mit dem Sie einfachen ASCII[1] Text bearbeiten können, um Ihre Eingabedateien zu erzeugen. Auf den meisten Computern stehen solche Texteditoren standardmäßig zur Verfügung. Beachten Sie, dass einfacher Text nicht unbedingt der Standard für einige Textverarbeitungsprogramme ist, insbesondere wenn sie mit verschiedenen Schriftarten umgehen können; zum Beispiel müssen Sie bei Microsoft Word die Datei im Format „Nur Text“ abspeichern, um eine einfache Textdatei zu erhalten.[2]

[1] American Standard Code for Information Interchange.

[2] Es wird empfohlen, dass alle Zeilen in einer Metamath-Quelldatei eine Länge von höchstens 79 Zeichen haben, um die Kompatibilität zwischen verschiedenen Computerterminals zu gewährleisten. Beim Erstellen einer Quelldatei in einem Editor wie Word, wählen Sie eine nichtproportionale Schriftart wie Courier oder Monaco, um dies zu erleichtern. Oder noch besser, verwenden Sie einfach einen einfachen Texteditor wie Notepad.

Auf einigen Computersystemen kann Metamath seine Ausgaben nicht direkt ausdrucken; stattdessen senden Sie die Ausgabe in eine Datei (mit den `open`-Befehlen, die später beschrieben werden). Die Art und Weise, wie Sie diese Ausgabedatei ausdrucken, hängt von Ihrem Computer ab. Einige Computer haben einen Druckbefehl, während Sie bei anderen Computern die Datei möglicherweise in einen Editor einlesen und von dort aus drucken müssen.

Wenn Sie Ihre Metamath-Quelldateien mit professionell formatierten Formeln, die mathematische Standardsymbole enthalten, drucken möchten, benötigen Sie das LaTeX Satzprogramm, das für die meisten Betriebssysteme frei verfügbar ist. Es läuft von Haus aus auf Unix und Linux und kann unter Windows als Teil des freien Cygwin-Pakets installiert werden (`http://cygwin.com`).

Sie können auch HTML-Webseiten[3] erstellen. Der Befehl `help html` im Metamath-Programm unterstützt Sie bei dieser Funktion.

2.2 Ihr erstes formales System

2.2.1 Vom Nichts zur Zahl Null

Um Ihnen ein Gefühl dafür zu vermitteln, wie die Metamath-Sprache aussieht, sehen wir uns ein sehr einfaches Beispiel aus der formalen Zahlentheorie an. Dieses Beispiel stammt aus Mendelson [43, S. 123].[4] Wir werden eine kleine Teilmenge dieser Theorie betrachten, nämlich den Teil, der für den ersten in [43] bewiesenen Satz der Zahlentheorie benötigt wird.

Zuerst werden wir uns einen standardmäßigen formalen Beweis für das ausgewählte Beispiel anschauen, und dann einen Blick auf die Metamath-Version werfen. Wenn Sie noch nie mit formalen Beweisen in Berührung gekommen sind, mag Ihnen die Notation als ein Overkill für die Darstellung so einfacher Begriffe vorkommen, so dass Sie sich vielleicht fragen, ob Sie etwas übersehen haben. Das haben Sie nicht. Die verwendeten Konzepte sind in der Tat sehr einfach, und eine detaillierte Aufschlüsselung auf diese Art ist notwendig, um den damit formulierten Beweis mechanisch verifizieren zu können. Und wie Sie sehen werden, zerlegt Metamath den Beweis in noch feinere Teile, so dass der mechanische Verifikationsprozess so einfach wie möglich gehalten werden kann.

Bevor wir die Axiome der Theorie einführen können, müssen wir die Syntaxregeln zur Bildung legaler Ausdrücke (Kombinationen von Symbolen) definieren, mit denen diese Axiome verwendet werden können. Die Zahl 0 ist ein **Term**; und wenn t und r Terme sind, so ist auch $(t + r)$ ein solcher. Hier sind t und r „Metavariablen“, die sich über Terme erstrecken;

[3] HyperText Markup Language.

[4] Um das Beispiel einfach zu halten, haben wir den Formalismus leicht verändert, und was wir als Axiome bezeichnen, sind streng genommen Theoreme in [43].

sie selbst erscheinen nicht als Symbole in einem eigentlichen Term. Einige Beispiele für konkrete Terme sind $(0 + 0)$ und $((0 + 0) + 0)$. (Beachten Sie, dass unsere Theorie nur die Zahl Null und Summen von Nullen beschreiben kann. Natürlich kann man mit einer so trivialen Theorie nicht viel anfangen, aber wir haben uns für unser Beispiel eine sehr kleine Teilmenge der kompletten Zahlentheorie ausgewählt. Das Wichtigste, worauf Sie sich konzentrieren sollten, sind unsere Definitionen, die beschreiben, wie Symbole kombiniert werden, um gültige Ausdrücke zu bilden, und nicht auf den Inhalt oder die Bedeutung dieser Ausdrücke). Wenn t und r Terme sind, ist ein Ausdruck der Form $t = r$ eine **wff** (wohlgeformte Formel); und wenn P und Q wffs sind, so ist dies auch $(P \to Q)$ (was „P impliziert Q“ oder „wenn P dann Q“ bedeutet). Hier sind P und Q Metavariablen, die sich über wffs erstrecken. Beispiele für konkrete wffs sind $0 = 0$, $(0 + 0) = 0$, $(0 = 0 \to (0 + 0) = 0)$, und $(0 = 0 \to (0 = 0 \to 0 = (0 + 0)))$. (Unsere Notation verwendet mehr Klammern als üblich, aber die Verhinderung von Mehrdeutigkeiten auf diese Weise vereinfacht unser Beispiel, da die Notwendigkeit entfällt, die Vorrangigkeit von Operatoren zu definieren.)

Die **Axiome** unserer Theorie sind alle wffs der folgenden Form, wobei t, r und s beliebige Terme sind:

$$(t = r \to (t = s \to r = s)) \tag{A1}$$

$$(t + 0) = t \tag{A2}$$

Man beachte, dass es eine unendliche Anzahl von Axiomen gibt, da es eine unendliche Anzahl von möglichen Termen gibt. A1 und A2 müssten korrekterweise „Axiomenschemata“ genannt werden, aber der Kürze halber werden sie als „Axiome“ bezeichnet.

Ein Axiom ist ein **Theorem**; und wenn P und $(P \to Q)$ Theoreme sind (wobei P und Q wffs sind), dann ist Q auch ein Theorem. Der zweite Teil dieser Definition wird als Modus ponens-Schlussregel (MP) bezeichnet. Sie erlaubt es uns, neue Theoreme aus alten Theoremen zu gewinnen.

Der **Beweis** eines Satzes ist eine Folge von einem oder mehreren Theoremen, von denen jedes entweder ein Axiom oder das Ergebnis des Modus ponens ist, das auf zwei vorhergehende Theoreme in der Folge angewandt wird, und das letzte davon das zu beweisende Theorem ist.

Das Theorem, das wir für unser Beispiel beweisen wollen, ist sehr einfach: $t = t$. Der Beweis unseres Theorems folgt. Studieren Sie ihn sorgfältig, bis Sie sicher sind, dass Sie ihn verstehen.

1.	$(t + 0) = t$	(nach Axiom A2)
2.	$(t + 0) = t$	(nach Axiom A2)
3.	$((t + 0) = t \to ((t + 0) = t \to t = t))$	(nach Axiom A1)
4.	$((t + 0) = t \to t = t)$	(nach MP angewandt auf Schritte 2 und 3)
5.	$t = t$	(nach MP angewandt auf Schritte 1 und 4)

(Sie fragen sich vielleicht, warum Schritt 1 zweimal wiederholt wird. Dies ist nicht notwendig in der formalen Sprache, die wir definiert haben. Aber wegen der in Metamath verwendeten „umgekehrten polnischen Notation“ für Beweise kann auf einen vorherigen Schritt nur einmal verwiesen werden. Die Wiederholung von Schritt 1 wird es Ihnen ermöglichen, diesen Beweis mit der Metamath-Version auf S. 56 besser abgleichen zu können).

Unser Theorem müsste richtigerweise als „Theoremschema“ bezeichnet werden, denn es stellt eine unendliche Anzahl von Theoremen dar, eines für jeden möglichen Term t. Zwei Beispiele für konkrete Theoreme wären $0 = 0$ und $(0+0) = (0+0)$. Selten beweisen wir eigentliche Theoreme, da wir durch den Nachweis von Schemata eine unendliche Anzahl von Sätzen auf einen Schlag beweisen können. In ähnlicher Weise sollte unser Beweis eigentlich als „Beweisschema“ bezeichnet werden. Um einen konkreten Beweis zu erhalten, wählen Sie einen konkreten Term, den Sie anstelle von t verwenden, und setzen ihn im gesamten Beweis für t ein.

Lassen Sie uns darüber diskutieren, was wir gerade getan haben. Die Axiome unserer Theorie, A1 und A2, sind trivial und offensichtlich. Jeder weiß, dass die Addition von Null zu etwas nichts verändert, und auch, dass, wenn zwei Dinge gleich einem dritten sind, sie einander gleichen. Die Feststellung des Trivialen und Offensichtlichen ist ein Ziel, das in jedem axiomatischen System angestrebt werden sollte. Aus trivialen und offensichtlichen Wahrheiten, über die sich alle einig sind, können wir Ergebnisse beweisen, die nicht so offensichtlich sind, und dennoch absolutes Vertrauen in sie haben. Wenn wir den Axiomen und den Regeln vertrauen, müssen wir per Definition auch den Konsequenzen dieser Axiome und Regeln vertrauen, wenn Logik überhaupt eine Bedeutung haben soll.

Unsere Schlussregel, der Modus ponens, ist ebenfalls ziemlich offensichtlich, sobald man versteht, was sie bedeutet. Wenn wir eine Tatsache P beweisen, und wir auch beweisen, dass P Q impliziert, dann folgt Q notwendigerweise als eine neue Tatsache. Die Regel gibt uns ein Mittel an die Hand, um neue Fakten zu erhalten (d.h. Theoreme) aus alten Fakten zu gewinnen.

Das Theorem $t = t$, das wir bewiesen haben, ist so grundlegend, dass man sich fragt warum es nicht zu den Axiomen gehört. In einigen Axiomensystemen der Arithmetik *ist* es ein Axiom. Die Wahl der Axiome in einer Theorie ist bis zu einem gewissen Grad willkürlich und sogar eine Art Kunst, die nur durch die Anforderung eingeschränkt wird, dass zwei gleichwertige Axiomensysteme in der Lage sein müssen, sich gegenseitig als Theoreme

ableiten zu können. Wir könnten uns vorstellen, dass der Erfinder unseres Axiomensystems ursprünglich $t = t$ als Axiom aufnahm und dann entdeckte, dass es sich als Theorem aus den anderen Axiomen ableiten lässt. Aus diesem Grund war es nicht notwendig, es als Axiom beizubehalten. Durch die Streichung dieses Axioms wurde der endgültige Satz von Axiomen sehr viel einfacher.

Wenn Sie noch nie mit formalen Beweisen gearbeitet haben, war es Ihnen wahrscheinlich nicht klar, dass $t = t$ aus unseren beiden Axiomen abgeleitet werden kann, bis Sie den Beweis gesehen haben. Obwohl Sie sicherlich glauben, dass $t = t$ wahr ist, könnten Sie einen imaginären Skeptiker, der nur an unsere beiden Axiome glaubt, nicht überzeugen, solange Sie nicht den Beweis erbringen. Formale Beweise wie dieser sind schwer zu finden, wenn man anfängt, mit ihnen zu arbeiten. Aber wenn man sich an sie gewöhnt hat, können sie interessant werden und Spaß machen. Sobald Sie die Idee hinter formalen Beweisen verstanden haben, haben Sie das grundlegende Prinzip, das der gesamten Mathematik zugrunde liegt, begriffen. Je anspruchsvoller die Mathematik wird, desto anspruchsvoller werden auch die Beweise, aber letztlich lassen sie sich alle in einzelne Schritte zerlegen, die so einfach sind wie die in unserem obigen Beweis.

Das Buch von Mendelson, dem unser Beispiel entnommen wurde, enthält eine Reihe detaillierter formaler Beweise wie diese, und es könnte Sie interessieren, es dort nachzuschlagen. Das Buch ist für Mathematiker gedacht, und das meiste davon hat ein ziemlich fortgeschrittenes Niveau. Zur populären Literatur, die sich mit der Beschreibung von formalen Beweisen befasst, gehören unter anderem [58, S. 296] und [26, S. 204–230].

2.2.2 Konvertierung des Beweises nach Metamath

Formale Beweise, wie der in unserem Beispiel, zerlegen die logischen Schlussfolgerungen in kleine, präzise Schritte, die keinen Zweifel daran lassen, dass die ihre Ergebnisse aus den Axiomen folgen. Man könnte meinen, dass dies die feinste Aufschlüsselung ist, die wir in der Mathematik erreichen können. Es steckt jedoch mehr hinter dem Beweis, als man auf den ersten Blick vermuten würde. Obwohl unsere Axiome recht einfach waren, war eine Menge Wortklauberei nötig, bevor wir sie überhaupt formulieren konnten: Wir mussten Begriffe wie „Term“, „wff“ und so weiter definieren. Darüber hinaus gibt es eine Reihe von impliziten Regeln, die wir noch nicht einmal erwähnt haben. Woher wissen wir zum Beispiel, dass Schritt 3 unseres Beweises aus dem Axiom A1 folgt? Es gibt einige versteckte Argumentationen, um dies zu bestimmt sagen zu können. Axiom A1 hat zwei Vorkommen des Buchstabens t. Eine der impliziten Regeln besagt, dass alles, was wir für t ersetzen, ein legaler Term[5] sein muss. Der Ausdruck $t + 0$ ist ganz offensichtlich

[5]Einige Autoren machen diese implizite Regel explizit, indem sie nach der Definition von „Term“ sagen: „Nur Ausdrücke der obigen Form sind Terme“.

ein legaler Term, wenn t ein solcher ist. Aber nehmen wir an, wir wollten solche Ersetzungen in einem riesigen Term mit Tausenden von Symbolen durchführen? Sicherlich wäre eine Menge Arbeit damit verbunden festzustellen, dass es sich wirklich bei dem Ergebnis um einen Term handelt, aber in gewöhnlichen formalen Beweisen würde man all diese Arbeit als einen einzigen „Schritt“ betrachten.

Um unser Axiomensystem in der Metamath-Sprache auszudrücken, müssen wir diese Zusatzinformationen zusätzlich zu den eigentlichen Axiomen beschreiben. Metamath weiß nicht, was ein „Term“ oder eine „wff“ ist. In Metamath ist die Spezifikation der Art und Weise, in der wir Symbole kombinieren können, um Terme und wffs zu erhalten, selbst wie kleine Axiome zu verstehen. Diese Hilfsaxiome werden in der gleichen Notation ausgedrückt wie die „echten“ Axiome, und Metamath unterscheidet nicht zwischen diesen beiden. Die Unterscheidung wird von Ihnen getroffen, d.h. durch die Art und Weise, wie Sie die Notation interpretieren, die Sie gewählt haben, um diese beiden Arten von Axiomen auszudrücken.

Die Metamath-Sprache zerlegt mathematische Beweise in winzige Teile, viel mehr als in gewöhnlichen formalen Beweisen. Wenn ein einzelner Schritt die Substitution eines komplexen Terms für eine seiner Variablen beinhaltet, muss Metamath diesen einzelnen Schritt in viele kleine Schritte unterteilt behandeln. Diese feinkörnige Aufschlüsselung ist es, die Metamath Allgemeingültigkeit und Flexibilität gibt, da es sich nicht auf eine bestimmte mathematische Notation beschränkt.

Die Beweisnotation von Metamath ist nicht dazu gedacht, von Menschen gelesen zu werden, sondern ihr kompaktes Format ist für eine Maschine gedacht. Das Metamath-Programm konvertiert diese Notation in eine Form, die Sie verstehen können, indem es den `show proof` Befehl nutzt. Sie können dem Programm mitteilen, wie detailliert Sie den Beweis betrachten möchten. Vielleicht möchten Sie sich nur die logischen Folgerungsschritte ansehen, die den normalen formalen Beweisschritten entsprechen, oder Sie möchten die feingranularen Schritte sehen, die beweisen, dass ein Ausdruck ein Term ist.

Hier ist schließlich, ohne weitere Erläuterungen, unser Beispiel, das in die Metamath-Sprache konvertiert wurde:

```
$( Festlegen der konstanten Symbole, die wir nutzen wollen $)
    $c 0 + = -> ( ) term wff |- $.
$( Festlegen der Metavariablen, die wir nutzen wollen $)
    $v t r s P Q $.
$( Spezifizieren der Eigenschaften der Metavariablen $)
    tt $f term t $.
    tr $f term r $.
    ts $f term s $.
    wp $f wff P $.
    wq $f wff Q $.
$( Definieren von "Term" und "wff" $)
```

```
    tze $a term 0 $.
    tpl $a term ( t + r ) $.
    weq $a wff t = r $.
    wim $a wff ( P -> Q ) $.
$( Festlegen der Axiome $)
    a1 $a |- ( t = r -> ( t = s -> r = s ) ) $.
    a2 $a |- ( t + 0 ) = t $.
$( Definieren der Schlussregel Modus ponens $)
    ${
       min $e |- P $.
       maj $e |- ( P -> Q ) $.
       mp  $a |- Q $.
    $}
$( Beweisen eines Theorems $)
    th1 $p |- t = t $=
  $( Hier ist sein Beweis: $)
       tt tze tpl tt weq tt tt weq tt a2 tt tze tpl
       tt weq tt tze tpl tt weq tt tt weq wim tt a2
       tt tze tpl tt tt a1 mp mp
     $.
```

Eine „Datenbasis“ ist ein Satz von einer oder mehreren ASCII Quelldateien. Es folgt eine kurze Beschreibung solch einer Metamath-Datenbasis (die aus einer einzigen Quelldatei besteht), damit Sie allgemein verstehen können, was vor sich geht. Um die Quelldatei im Detail zu verstehen, sollten Sie Kapitel 4 lesen.

Die Datenbasis ist eine Folge von „Token“, die normalerweise durch Leerzeichen oder Zeilenumbrüche voneinander getrennt sind. Die einzigen Token, die in der Metamath-Sprache fest vorgegeben sind, sind diejenigen die mit `$` beginnen. Diese Token werden „Schlüsselwörter“ genannt. Alle anderen Token sind benutzerdefiniert, und ihre Namen sind beliebig.

Wie Sie vielleicht schon vermutet haben, beginnt mit dem Metamath-Token `$(` ein Kommentar und wird mit `$)` beendet.

Die Metamath-Token `$c`, `$v`, `$e`, `$f`, `$a` und `$p` spezifizieren „Anweisungen“, die mit `$.` enden.

Die Metamath-Token `$c` und `$v` deklarieren jeweils eine Liste von benutzerdefinierten Token, genannt „Mathematische Symbole“, die später in der Datenbasis verwendet werden. Alle mathematischen Symbole für unser Beispiel werden so definiert, außer dem Drehkreuzsymbol `|-` (⊢), das von Logikern üblicherweise verwendet wird um zu sagen: „Ein Beweis existiert für“. Für uns ist das Drehkreuz nur ein praktisches Symbol zur Unterscheidung zwischen Ausdrücken, die Axiome oder Theoreme darstellen, und Ausdrücken, die Terme oder wffs sind.

Die Anweisung `$c` deklariert „Konstanten“, und die Anweisung `$v` deklariert „Variablen“ (oder genauer gesagt, Metavariablen). Eine Variable kann

durch Folgen von mathematischen Symbolen ersetzt werden, während eine Konstante nicht ersetzt werden kann.

Es mag redundant erscheinen, dass sowohl `$c` als auch `$v` erforderlich sind (da jedes mathematische Symbol, das nicht mit einer `$c`-Anweisung spezifiziert wurde, als Variable angesehen werden könnte). Aber dies ermöglicht eine bessere Prüfung auf Fehler und erlaubt es auch, mathematische Symbole neu zu deklarieren (Abschnitt 4.2.8).

Das Token `$f` spezifiziert eine Anweisung, die als „Hypothese vom Variablentyp“ (auch als „fließende Hypothese“ bekannt) und `$e` spezifiziert eine „logische Hypothese“ (auch „essentielle Hypothese“). Das Token `$a` spezifiziert eine „axiomatische Behauptung“, und `$p` spezifiziert eine „beweisbare Behauptung“. Links von jedem Vorkommen dieser vier Token befindet sich ein „Label“ , das die Hypothese oder Behauptung für eine spätere Bezugnahme identifiziert. Zum Beispiel ist das Label der ersten axiomatischen Behauptung `tze`. Eine `$f`-Anweisung muss genau zwei mathematische Symbole enthalten, eine Konstante gefolgt von einer Variable. Die Anweisungen `$e`, `$a`, und `$p` beginnen jeweils mit einer Konstanten, auf die im Allgemeinen eine beliebige Folge von mathematischen Symbolen folgt.

Jeder Behauptung ist ein Satz von Hypothesen zugeordnet die erfüllt sein müssen, damit die Behauptung in einem Beweis verwendet werden kann. Diese werden „obligatorische Hypothesen“ der Behauptung genannt. Zu den Hypothesen, deren „Gültigkeitsbereich“ (siehe unten) die Behauptung umfasst, sind `$e` Hypothesen immer obligatorisch und `$f` Hypothesen sind obligatorisch, wenn sie ihre Variable mit der Behauptung oder ihren `$e`-Hypothesen teilen. Die genauen Regeln zur Bestimmung, welche Hypothesen obligatorisch sind, werden in den Abschnitten
refframes und 4.2.8 im Detail beschrieben. Zum Beispiel sind `tt` und `tr` die obligatorischen Hypothesen der Behauptung `tpl`, während die Behauptung `tze` keine obligatorischen Hypothesen hat, weil sie keine Variablen enthält und keine `$e` Hypothese hat. Metamaths `show statement`-Befehl, der im nächsten Abschnitt beschrieben wird, zeigt Ihnen die obligatorischen Hypothesen einer Behauptung.

Manchmal soll eine Hypothese nur für bestimmte Behauptungen relevant sein. Die Menge der Behauptungen, für die eine Hypothese relevant ist, wird ihr „Gültigkeitsbereich“ genannt. Die Metamath-Klammern, `${` und `$char‘}`, definieren einen „Block“, der den Gültigkeitsbereich jeder dazwischen liegenden Hypothese abgrenzt. Die Behauptung `mp` hat obligatorische Hypothesen `wp`, `wq`, `min` und `maj`. Die einzige obligatorische Hypothese von `th1` ist dagegen `tt`, da `th1` außerhalb des Blocks auftritt, der `min` und `maj` enthält.

Beachten Sie, dass `$${` und `$$}` keine Auswirkungen auf den Gültigkeitsbereich von Behauptungen (`$a` und `$p`) haben. Behauptung sind immer verfügbar, um von jedem späteren Beweis in der Quelldatei referenziert zu werden.

Jede beweisbare Behauptung (`$p` Behauptung) besteht aus zwei Teilen. Der erste Teil ist die Behauptung selbst: eine Folge von mathematischen Symbolen, die zwischen dem `$p`-Token und einem `$=`-Token platziert ist. Der zweite Teil ist ein „Beweis“: eine Liste von Label-Token, die zwischen dem `$=`-Token und dem `$.`-Token, das die Behauptung beendet, liegt.[6] Der Beweis fungiert als eine Reihe von Anweisungen für das Metamath-Programm, die ihm sagen, wie die Abfolge der mathematischen Symbole, die im Behauptungsteil der Anweisung enthalten ist, unter Verwendung der Hypothesen der `$p`-Anweisung und der vorherigen Behauptungen. Die Konstruktion erfolgt nach genauen Regeln. Wenn die Liste der Label im Beweis gegen diese Regeln verstößt oder wenn die sich ergebende Endsequenz nicht mit der Behauptung übereinstimmt, gibt das Metamath-Programm eine Fehlermeldung aus.

Wenn Sie mit der umgekehrten polnischen Notation (engl. „reverse Polish notation“; RPN) vertraut sind, die manchmal auf Taschenrechnern verwendet wird, wissen Sie, wie ein Beweis in Kurzform funktioniert. Jedes Hypothesenlabel im Beweis wird auf den RPN-Stapel geschoben sobald es auftaucht. Jedes Label einer Behauptung entfernt so viele Einträge vom Stapel, wie die referenzierte Behauptung zwingende Hypothesen hat. Variablenersetzungen werden aus den obligatorischen Hypothesen der referenzierten Behauptung so berechnet, dass diese Hypothesen mit den Stack-Einträgen übereinstimmen. Die gleichen Substitutionen werden dann an den Variablen in der referenzierten Aussage selbst vorgenommen, die dann auf den Stapel abgelegt wird. Am Ende des Beweises sollte es nur noch einen Eintrag im Stapel geben, nämlich die zu beweisende Behauptung. Dieser Vorgang wird im Abschnitt 4.3 ausführlich erläutert.

Die Beweisnotation von Metamath ist für Menschen nicht einfach lesbar, aber sie erlaubt es, den Beweis kompakt in einer Datei zu speichern. Das Programm Metamath hat Funktionen zur Anzeige von Beweisen in einer lesbareren Art, mit der man besser sehen kann, was vor sich geht, wie Sie im nächsten Abschnitt sehen werden.

Die Regeln, die bei der Überprüfung eines Beweises verwendet werden, basieren weder auf einer eingebauten Syntax der Symbolsequenz in einer Behauptung noch auf irgendwelchen eingebauten Bedeutungen, die mit bestimmten Symbolnamen verbunden sind. Sie basieren ausschließlich auf Symbolübereinstimmungen: Konstanten müssen mit sich selbst übereinstimmen, und Variablen können durch alles ersetzt werden, was eine Übereinstimmung ermöglicht. Anstelle von `term`, `0` und `|-` könnten wir zum Beispiel ebenso gut `gelb`, `Null` und `beweisbar` verwenden, solange wir dies in der gesamten Datenbasis einheitlich tun. Wir hätten auch `ist beweisbar` (zwei Token)

[6] Wenn Sie die `set.mm`-Datenbasis angesehen haben, ist Ihnen vielleicht eine andere Notation aufgefallen, die für Beweise genutzt wird. Die andere Notation wird „komprimiert“ genannt. Sie reduziert den Platzbedarf zur effizienten Speicherung eines Beweises in der Datenbasis, und wird in Anhang B beschrieben. In dem obigen Beispiel verwenden wir die „normale“ Notation.

anstelle von `|-` (ein Token) in der gesamten Datenbasis verwenden können. In jedem dieser Fälle wäre der Beweis genau derselbe. Die Unabhängigkeit von Beweisen und Notation bedeutet, dass Sie viele Möglichkeiten haben die verwendete Notation zu ändern, ohne die Beweise ändern zu müssen.

2.3 Ein Probelauf

Jetzt sind Sie bereit, das Metamath-Programm auszuprobieren.

Metamath verfügt auf allen Computersystemen über eine standardmäßige „Befehlszeilen Schnittstelle“ (command line interface; CLI), die es Ihnen ermöglicht, mit dem Programm zu interagieren. Sie geben Befehle in die CLI ein, indem Sie sie auf der Tastatur eintippen und nach jeder Zeile die *Eingabe*-Taste Ihrer Tastatur drücken. Die CLI ist einfach zu bedienen und verfügt über integrierte Hilfe-Funktionen.

Als erstes sollten Sie einen Texteditor verwenden, um eine Datei mit dem Namen `demo0.mm` zu erstellen und in diese den Metamath-Quelltext einzugeben, der auf S. 48 gezeigt wird. Eigentlich ist diese Datei im Metamath Software-Paket enthalten, also überprüfen Sie das zuerst. Wenn Sie den Quelltext eintippen, stellen Sie sicher, dass Sie ihn in Form von „reinem ASCII Text mit Zeilenumbrüchen“ speichern. Die meisten Textverarbeitungsprogramme verfügen über diese Funktion.

Als nächstes müssen Sie das Metamath-Programm ausführen. Abhängig von Ihrem Computer System und der Art der Installation von Metamath kann dies von einem Mausklick auf das Metamath-Symbol, die Eingabe von `run metamath` oder durch die einfache Eingabe von `metamath` reichen. (Der Befehl `help invoke` von Metamath beschreibt alternative Möglichkeiten, das Metamath-Programm aufzurufen.)

Wenn Sie Metamath zum ersten Mal eingeben, befindet es sich in der CLI und wartet auf Ihre Eingabe. Auf Ihrem Bildschirm sehen Sie dann etwas wie das Folgende:

```
Metamath - Version 0.177 27-Apr-2019
Type HELP for help, EXIT to exit.
MM>
```

Die Eingabeaufforderung `MM>` bedeutet, dass Metamath auf einen Befehl wartet. Bei Befehlsschlüsselwörtern wird nicht zwischen Groß- und Kleinschreibung unterschieden; wir werden in unseren Beispielen Befehle in Kleinbuchstaben verwenden. Die Versionsnummer und das Veröffentlichungsdatum werden auf Ihrem System wahrscheinlich von der oben gezeigten Version abweichen.

Als erstes müssen Sie Ihre Datenbasis einlesen:[7]

[7] Wenn in Unix ein Verzeichnispfad benötigt wird, sollten Sie den Pfad/Dateinamen in Anführungszeichen setzen, damit Metamath nicht denkt, dass das `/` im Pfadnamen ein Befehlszeilenparameter ist, z.B., `read "‘db/set.mm"’`. Anführungszeichen sind optional,

```
MM> read demo0.mm
```

Denken Sie daran, nach der Eingabe dieses Befehls die *Eingabe*taste zu drücken. Wenn Sie den Dateinamen weglassen, wird Metamath Sie nach einem Namen fragen. Die Syntax für die Angabe eines Macintosh-Dateipfades ist in einer Fußnote auf S. 168. zu finden.

Wenn es in der Datenbasis Syntaxfehler gibt, wird Metamath Sie beim Einlesen der Datei darauf hinweisen. Das Einzige, was Metamath beim Einlesen einer Datenbasis nicht prüft, ist die Korrektheit aller Beweise, denn das würde es zu sehr verlangsamen. Es ist jedoch ratsam, die Beweise in einer Datenbasis, an der Sie Änderungen vornehmen, regelmäßig zu überprüfen. Verwenden Sie dazu den folgenden Befehl (und führen Sie ihn jetzt für Ihre Datei `demo0.mm` aus). Beachten Sie, dass `*` ein 'Platzhalter" ist, der alle Beweise in der Datei repräsentiert.

```
MM> verify proof *
```

Metamath meldet alle ungültigen Beweise.

Es ist oft nützlich, die Informationen zu speichern, die das Metamath-Programm auf dem Bildschirm anzeigt. Sie können alles speichern, was auf dem Bildschirm passiert, indem Sie eine Protokolldatei öffnen. Sie sollten dies tun, bevor Sie eine Datenbasis einlesen, damit Sie später eventuelle Fehler untersuchen können. Um eine Protokolldatei zu öffnen, geben Sie Folgendes ein:

```
MM> open log abc.log
```

Dadurch wird eine Datei namens `abc.log` geöffnet, und alles, was ab diesem Zeitpunkt auf dem Bildschirm erscheint, wird in dieser Datei gespeichert. Der Name der Protokolldatei ist frei wählbar. Um die Protokolldatei zu schließen, geben Sie Folgendes ein:

```
MM> close log
```

Sie können mit mehreren Befehlen untersuchen, was in Ihrer Datenbasis enthalten ist. Abschnitt 3.10 enthält eine Übersicht über einige nützliche Befehle. Der Befehl `show labels` lässt Sie sehen, welche Label vorhanden sind. Ein `*` entspricht einer beliebigen Kombination von Zeichen, und `t*` bezieht sich auf alle Labels, die mit dem Buchstaben `t` beginnen. Das `/all` ist ein „Befehlszeilenparameter" , der Metamath anweist, die Beschriftungen der Hypothesen einzuschließen. (Um die Syntax erklärt zu bekommen, geben Sie`help show labels` ein .)

Geben Sie Folgendes ein:

```
MM> show labels t* /all
```

Metamath antwortet mit

wenn keine Zweideutigkeit besteht.

```
The statement number, label, and type are shown.
3 tt $f       4 tr $f       5 ts $f       8 tze $a
9 tpl $a      19 th1 $p
```

Sie können den Befehl `show statement` verwenden, um Informationen über eine bestimmte Anweisung zu erhalten. Sie können zum Beispiel Informationen über die Anweisung mit der Bezeichnung `mp` bekommen, wenn Sie Folgendes eingeben:

```
MM> show statement mp /full
```

Metamath antwortet mit

```
  Statement 17 is located on line 43 of the file
  "demo0.mm".
  "Define the modus ponens inference rule"
  17 mp $a |- Q $.
  Its mandatory hypotheses in RPN order are:
    wp $f wff P $.
    wq $f wff Q $.
    min $e |- P $.
    maj $e |- ( P -> Q ) $.
  The statement and its hypotheses require the
        variables:  Q P
  The variables it contains are:  Q P
```

Die obligatorischen Hypothesen und ihre Reihenfolge sind nützliche Informationen, wenn Sie versuchen, einen Beweis zu verstehen oder zu debuggen.

Jetzt sind Sie bereit, sich anzusehen, was wirklich in unserem Beweis enthalten ist. Zunächst wird hier gezeigt, wie man sich jeden Schritt des Beweises ansieht - nicht nur die Schritte eines gewöhnlichen formalen Beweises, sondern auch diejenigen, die die Formeln aufbauen, die in jedem Schritt eines gewöhnlichen formalen Beweises auftauchen.

```
MM> show proof th1 /lemmon /all
```

Dadurch wird der Beweis in folgendem Format auf dem Bildschirm angezeigt:

```
 1 tt            $f term t
 2 tze           $a term 0
 3 1,2 tpl       $a term ( t + 0 )
 4 tt            $f term t
 5 3,4 weq       $a wff ( t + 0 ) = t
 6 tt            $f term t
 7 tt            $f term t
 8 6,7 weq       $a wff t = t
 9 tt            $f term t
```

```
10 9 a2          $a |- ( t + 0 ) = t
11 tt            $f term t
12 tze           $a term 0
13 11,12 tpl     $a term ( t + 0 )
14 tt            $f term t
15 13,14 weq     $a wff ( t + 0 ) = t
16 tt            $f term t
17 tze           $a term 0
18 16,17 tpl     $a term ( t + 0 )
19 tt            $f term t
20 18,19 weq     $a wff ( t + 0 ) = t
21 tt            $f term t
22 tt            $f term t
23 21,22 weq     $a wff t = t
24 20,23 wim     $a wff ( ( t + 0 ) = t -> t = t )
25 tt            $f term t
26 25 a2         $a |- ( t + 0 ) = t
27 tt            $f term t
28 tze           $a term 0
29 27,28 tpl     $a term ( t + 0 )
30 tt            $f term t
31 tt            $f term t
32 29,30,31 a1   $a |- ( ( t + 0 ) = t -> ( ( t + 0 )
                                        = t -> t = t ) )
33 15,24,26,32 mp  $a |- ( ( t + 0 ) = t -> t = t )
34 5,8,10,33 mp  $a |- t = t
```

Der Befehlszeilenparameter `/lemmon` spezifiziert eine Anzeige im sogenannten Lemmon-Stil. Das Weglassen der Option `/lemmon` führt zu einer Baumdarstellung des Beweises (siehe S. 156 für ein Beispiel), der etwas weniger eindeutig ist, aber leichter zu folgen ist, wenn man sich daran gewöhnt hat.

Die erste Zahl in jeder Zeile ist die Schrittnummer des Beweises. Alle folgenden Zahlen sind Schrittnummern, die den Hypothesen der Aussage zugeordnet sind, auf die dieser Schritt verweist. Die nächste Zahl ist die Bezeichnung der, auf die der Schritt verweist. Der Typ der Aussage, auf die verwiesen wird, kommt als nächstes, gefolgt von der Folge mathematischer Symbole, die durch den Beweis bis zu diesem Schritt konstruiert wurde.

Der letzte Schritt, 34, enthält die Aussage, die bewiesen wird.

Betrachtet man einen kleinen Teil des Beweises, so stellt man fest, dass die Schritte 3 und 4 aussagen, dass `( t + 0 )` und `t` `term` e sind, und Schritt 5 nutzt die Schritte 3 und 4, um festzustellen, dass `( t + 0 ) = t` eine `wff` ist. Lassen Sie Metamath selbst im Detail erklären, was in Schritt 5 geschieht. Beachten Sie, dass die „Zielhypothese“ sich darauf bezieht, wo Schritt 5 letztendlich verwendet wird, d.h. in Schritt 34.

```
MM> show proof th1 /detailed_step 5
Proof step 5:  wp=weq $a wff ( t + 0 ) = t
This step assigns source "weq" ($a) to target "wp"
($f).  The source assertion requires the hypotheses
"tt" ($f, step 3) and "tr" ($f, step 4).  The parent
assertion of the target hypothesis is "mp" ($a,
step 34).
The source assertion before substitution was:
    weq $a wff t = r
The following substitutions were made to the source
assertion:
    Variable  Substituted with
     t         ( t + 0 )
     r         t
The target hypothesis before substitution was:
    wp $f wff P
The following substitution was made to the target
hypothesis:
    Variable  Substituted with
     P         ( t + 0 ) = t
```

Der soeben gezeigte vollständige Beweis ist nützlich für das Verständnis, was im Detail vor sich geht. Die meiste Zeit werden Sie jedoch nur an den „wesentlichen“ oder logischen Schritten eines Beweises, d.h. die Schritte, die einem gewöhnlichen formalen Beweisindex entsprechen, interessiert sein. Wenn Sie Folgendes eingeben

```
MM> show proof th1 /lemmon /renumber
```

dann sehen Sie

```
1 a2             $a |- ( t + 0 ) = t
2 a2             $a |- ( t + 0 ) = t
3 a1             $a |- ( ( t + 0 ) = t -> ( ( t + 0 )
                                   = t -> t = t ) )
4 2,3 mp         $a |- ( ( t + 0 ) = t -> t = t )
5 1,4 mp         $a |- t = t
```

Vergleichen Sie dies mit dem formalen Beweis auf S. 45 und beachten Sie die Ähnlichkeit. Standardmäßig zeigt Metamath für einen Beweis nicht `$f`-Hypothesen und alles, was von ihnen im Beweisbaum abzweigt; dadurch sieht der Beweis eher wie ein gewöhnlicher mathematischen Beweis aus, der normalerweise keine explizite Konstruktion von Ausdrücken beinhaltet. Dies wird die „essentielle“ Ansicht genannt (früher musste man den Parameter `/essential` im Befehl `show proof` hinzufügen, um diese Ansicht zu erhalten, aber das ist jetzt die Standardeinstellung). Sie können den Parameter `/all` im Befehl `show proof` verwenden, um auch die explizite Konstruktion

von Ausdrücken anzuzeigen. Durch den Parameter `/renumber` werden die Schritte neu nummeriert, damit sie dem entsprechen, was angezeigt wird.

Um Metamath zu verlassen, geben Sie Folgendes ein:

```
MM> exit
```

2.3.1 Einige Hinweise zur Verwendung der Befehlszeilenschnittstelle

Wir schließen diese kurze Einführung in Metamath mit einigen hilfreichen Hinweisen, wie Sie sich durch die Befehle navigieren können.

Wenn Sie Befehle in Metamaths CLI eingeben, müssen Sie nur so viele Zeichen eines Befehlsschlüsselworts eingeben, die für die Eindeutigkeit notwendig sind. Wenn Sie zu wenige Zeichen eingeben, wird Metamath Ihnen mitteilen, welche die Auswahlmöglichkeiten sind. Im Fall des Befehls `read` ist nur das `r` nötigt, um ihn eindeutig zu spezifizieren. Sie hätten also Folgendes eingeben können

```
MM> r demo0.mm
```

anstelle von

```
MM> read demo0.mm
```

In unserer Beschreibung geben wir immer die vollständigen Befehlswörter an. Bei Verwendung der Metamath CLI-Befehle in einer Befehlsdatei (zum Einlesen mit dem Befehl `submit`), ist es ratsam, den ungekürzten Befehl zu verwenden, um sicherzustellen, dass Ihre Anweisungen nicht mehrdeutig werden, wenn dem Metamath-Programm in Zukunft weitere Befehle hinzugefügt werden.

Die Befehlsschlüsselwörter unterscheiden nicht zwischen Groß- und Kleinschreibung; Sie können entweder `read` oder `ReAd` eingeben. Bei Dateinamen wird je nach Betriebssystem Ihres Computers zwischen Groß- und Kleinschreibung unterschieden. Metamath Label und mathematische Symbole-Tokens unterscheiden zwischen Groß- und Kleinschreibung.

Der Befehl `help` bietet Ihnen eine Liste der Themen, zu denen Sie Hilfe erhalten können. Sie können dann `help` *topic* eingeben, um Hilfe zu diesem Thema zu erhalten.

Wenn Sie sich über die Schreibweise eines Befehls nicht sicher sind, geben Sie einfach die Zeichen des Befehls ein, an die Sie sich erinnern. Wenn Sie nicht genug Zeichen eingetippt haben, um ihn eindeutig zu spezifizieren, wird Metamath Ihnen sagen, welche Auswahlmöglichkeiten Sie haben.

```
MM> show s
         ^
?Ambiguous keyword - please specify SETTINGS,
STATEMENT, or SOURCE.
```

Wenn Sie nicht wissen, welches Argument Sie als Teil eines Befehls verwenden sollen, geben Sie ein `?` an der Position des Arguments ein. Metamath wird Ihnen sagen, was es dort erwartet.

```
MM> show ?
         ^
?Expected SETTINGS, LABELS, STATEMENT, SOURCE, PROOF,
MEMORY, TRACE_BACK, or USAGE.
```

Schließlich können Sie auch nur das erste Wort oder die ersten Wörter eines Befehls eingeben, gefolgt von *return*. Metamath fragt Sie nach dem restlichen Teil des Befehls und zeigt Ihnen bei jedem Schritt die Auswahlmöglichkeiten an. Sie könnten anstelle der Eingabe von `show statement th1 /full` beispielsweise folgendermaßen vorgehen:

```
MM> show
SETTINGS, LABELS, STATEMENT, SOURCE, PROOF,
MEMORY, TRACE_BACK, or USAGE <SETTINGS>? st
What is the statement label <th1>?
/ or nothing <nothing>? /
TEX, COMMENT_ONLY, or FULL <TEX>? f
/ or nothing <nothing>?
19 th1 $p |- t = t $= ... $.
```

Nach jedem `?` in diesem Modus müssen Sie Metamath die Informationen geben, die es anfordert. Manchmal gibt Metamath Ihnen eine Liste von Auswahlmöglichkeiten an, wobei die Standardauswahl durch Klammern `< >` angezeigt wird. Durch das Drücken von *return* nach dem `?` wird die Standardauswahl ausgewählt. Wenn Sie eine andere Antwort geben, wird die Standardauswahl außer Kraft gesetzt. Beachten Sie, dass der `/` in Befehlszeilenparametern als ein separates Token betrachtet wird und deshalb separat abgefragt wird.

2.4 Ihr erster Beweis

Beweise werden mit Hilfe des Beweis-Assistenten erstellt. Wir werden Ihnen nun zeigen, wie der Beweis des Satzes `th1` aufgebaut wurde. Damit Sie diese Schritte selbst nachvollziehen können, lassen wir zunächst den Beweis-Assistenten den Beweis im Quellpuffer von Metamath löschen und ihn dann rekonstruieren. (Der Quellpuffer ist die Stelle im Speicher wo Metamath die Informationen aus der Datenbasis speichert, wenn sie eingelesen werden (mit dem `read` Befehl). Neue oder geänderte Beweise werden im Quellpuffer gehalten, bis ein `write source` Befehl erteilt wird). In der Praxis würde man ein `?` zwischen `$=` und `$.` in der Datenbasis platzieren, um Metamath mitzuteilen, dass der Beweis unbekannt ist, und das wäre Ihr Ausgangspunkt. Wann immer der Befehl `verify proof` auf einen Beweis mit einem

`?` anstelle eines Beweisschritts trifft, wird die Aussage als nicht bewiesen gekennzeichnet.

Als ich anfing, Metamath-Beweise zu erstellen, habe ich mir auf ein Stück Papier den vollständigen formalen Beweis, wie er mit einem `show proof` Befehl ausgegeben würde (siehe die Anzeige von `show proof th1 /lemmon /renumber` oben als ein Beispiel). Nachdem Sie sich an den Umgang mit dem Beweis-Assistenten gewöhnt haben, können Sie den Beweis in Ihrem Kopf „sehen“ und sich beim Ausfüllen der Details vom Beweis-Assistenten leiten lassen, zumindest bei einfacheren Beweisen. Aber bis Sie diese Erfahrung gesammelt haben, können es für die sehr hilfreich sein alle Details vorher aufzuschreiben. Andernfalls könnten viel Zeit verschwendet werden, wenn Sie sich vom Assistenten auf einen falschen Weg führen lassen. Andere finden diesen Ansatz jedoch nicht so hilfreich. Zum Beispiel findet Thomas Brendan Leahy, dass es für ihn hilfreicher ist, interaktiv rückwärts von einer maschinenlesbaren Aussage zu arbeiten. David A. Wheeler schreibt sich zuerst einen allgemeinen Ansatz auf, entwickelt den Beweis aber interaktiv durch Umschalten zwischen vorwärts (ausgehend von Hypothesen und Fakten, die nützlich sein könnten) und rückwärts (vom Ziel ausgehend), bis sich die vorwärts- und rückwärtsgerichteten Ansätze treffen. Am Ende sollten Sie den Ansatz wählen, der für Sie am besten geeignet ist.

Ein Beweis wird mit dem Beweis-Assistenten entwickelt, indem rückwärts gearbeitet wird, beginnend mit dem zu beweisenden Theorem. Danach wird jeder unbekannte Schritt einem Theorem oder einer Hypothese zugeordnet, bis keine unbekannten Schritte mehr übrig bleiben. Der Beweis-Assistent lässt Sie nur dann eine Zuordnung vornehmen, wenn diese mit dem unbekannten Schritt „vereinheitlicht“ werden kann. Das bedeutet, dass eine Substitution von Variablen existiert, die die Zuordnung mit dem unbekannten Schritt übereinstimmen lässt. Andererseits ist in der Mitte eines Beweises, wenn man rückwärts arbeitet, oft mehr als eine Vereinheitlichung (Menge von Substitutionen) möglich, da zu diesem Zeitpunkt nicht genügend Informationen vorhanden sind, um sie eindeutig zu bestimmen. In diesem Fall kann man Metamath mitteilen, welche Vereinheitlichung zu wählen ist, oder man kann weiterhin unbekannte Schritte zuweisen, bis genügend Informationen verfügbar sind, um die Vereinheitlichung eindeutig zu machen.

Wir gehen davon aus, dass Sie Metamath gestartet und die Datenbasis wie oben beschrieben eingelesen haben. Der folgende Dialog zeigt, wie der Beweis erstellt wird. Weitere Einzelheiten zu den Funktionen einiger Befehle finden Sie in Abschnitt 5.6.

```
MM> prove th1
Entering the Proof Assistant.  Type HELP for help, EXIT
to exit.  You will be working on the proof of statement th1:
  $p |- t = t
Note:  The proof you are starting with is already complete.
MM-PA>
```

Die Eingabeaufforderung `MM-PA>` bedeutet, dass wir uns innerhalb des Beweis-Assistenten befinden. Die meisten der regulären Metamath-Befehle (`show statement`, etc.) sind weiterhin verfügbar, falls Sie sie benötigen.

```
MM-PA> delete all
The entire proof was deleted.
```

Wir haben den gesamten Beweis gelöscht, damit wir von vorne beginnen können.

```
MM-PA> show new_proof/lemmon/all
1 ?              $? |- t = t
```

Der Befehl `show new_proof` verhält sich wie `show proof`, außer dass wir keine Aussage angeben; stattdessen wird der Beweis, an dem wir gerade arbeiten, angezeigt.

```
MM-PA> assign 1 mp
To undo the assignment, DELETE STEP 5 and INITIALIZE, UNIFY
if needed.
3   min=?  $? |- $2
4   maj=?  $? |- ( $2 -> t = t )
```

Der obige `assign`-Befehl bedeutet „Weise die Aussage mit der Bezeichnung `mp` dem Schritt 1 zu.“ Beachten Sie, dass die Schritte ständig neu nummeriert werden, wenn Sie Schritte in der Mitte eines Beweises zuweisen; im Allgemeinen werden alle Schritte von dem Schritt, den Sie zuweisen, bis zum Ende des Beweises nach oben verschoben. In diesem Fall ist der frühere Schritt 1 jetzt Schritt 5, weil der (Teil-)Beweis jetzt fünf Schritte hat: die vier Hypothesen der `mp`-Aussage und die `mp` Aussage selbst. Schauen wir uns alle Schritte in unserem Teilbeweis an:

```
MM-PA> show new_proof/lemmon/all
1 ?              $? wff $2
2 ?              $? wff t = t
3 ?              $? |- $2
4 ?              $? |- ( $2 -> t = t )
5 1,2,3,4 mp     $a |- t = t
```

Das Symbol `$2` ist eine temporäre Variable, die eine noch nicht bekannte Symbolfolge darstellt. In dem endgültigen Beweis müssen alle temporären Variablen eliminiert sein. Das allgemeine Format für eine temporäre Variable ist `$` gefolgt von einer ganzen Zahl. Beachten Sie, dass `$` kein zulässiges Zeichen in einem mathematischen Symbol ist (siehe Abschnitt 4.2.1, S. 132), also wird es niemals einen Namenskonflikt zwischen realen Symbolen und temporären Variablen geben.

Die unbekannten Schritte 1 und 2 sind Konstruktionen der beiden wffs, die von dem Modus ponens verwendet werden. Wie Sie am Ende dieses Abschnitts sehen werden, kann der Beweis-Assistent diese Schritte normalerweise selbst herausfinden, und wir müssen uns nicht um sie kümmern. Deshalb werden wir von hier an nur noch die „essentiellen“ Hypothesen anzeigen, d.h. die Schritte, die den traditionellen formalen Beweisen entsprechen.

```
MM-PA> show new_proof/lemmon
3 ?              $? |- $2
4 ?              $? |- ( $2 -> t = t )
5 3,4 mp         $a |- t = t
```

Die unbekannten Schritte 3 und 4 sind die, auf die wir uns konzentrieren müssen. Sie entsprechen den Neben- und Hauptprämissen des Modus ponens. Wir werden sie wie folgt zuordnen. Beachten Sie, dass es wegen der Neunummerierung der Schritte nach einer Zuordnung vorteilhaft ist, die unbekannten Schritte in umgekehrter Reihenfolge zuzuordnen, da frühere Schritte nicht neu nummeriert werden.

```
MM-PA> assign 4 mp
To undo the assignment, DELETE STEP 8 and INITIALIZE, UNIFY
if needed.
3   min=?  $? |- $2
6     min=?  $? |- $4
7     maj=?  $? |- ( $4 -> ( $2 -> t = t ) )
```

Wir werden jetzt eine obskure Funktionalität beschreiben, die Sie wahrscheinlich nie benutzen werden, die Sie aber kennen sollten. Die Metamath-Sprache erlaubt es, dass Variable durch leere Symbolsequenzen ersetzt werden können, aber in den meisten formalen Systemen wird dies nie eingesetzt. Eines der wenigen Beispiele, in denen sie verwendet wird, ist das MIU-System, das in Anhang D beschrieben wird. Aber solche Systeme sind selten, und standardmäßig ist diese Funktion im Beweis-Assistenten ausgeschaltet (sie ist immer erlaubt für `verify proof`). Schalten wir sie ein und schauen, was was passiert.

```
MM-PA> set empty_substitution on
Substitutions with empty symbol sequences is now allowed.
```

Wenn diese Funktionalität aktiviert ist, werden mehr Vereinheitlichungen in der Mitte eines Beweises mehrdeutig sein, weil die Substitution von Variablen mit leeren Symbolfolgen eine zusätzliche Möglichkeit darstellt. Schauen wir uns an, was passiert, wenn wir unsere nächste Zuweisung vornehmen.

```
MM-PA> assign 3 a2
There are 2 possible unifications.  Please select the correct
```

```
    one or Q if you want to UNIFY later.
Unify:  |- $6
 with:  |- ( $9 + 0 ) = $9
Unification #1 of 2 (weight = 7):
  Replace "$6" with "( + 0 ) ="
  Replace "$9" with ""
  Accept (A), reject (R), or quit (Q) <A>? r
```

Die erste der vorgestellten Möglichkeiten ist die falsche. Hätten wir sie gewählt, wäre der temporären Variablen `$6` eine abgeschnittene wff, und der temporären Variablen `$9` eine leere Folge zugewiesen worden (was in unserem System nicht zulässig ist). Bei dieser Wahl kämen wir irgendwann an einen Punkt, an dem wir nicht mehr weiterkämen, weil wir mit Schritten enden würden, die nicht mehr zu beweisen sind. (Probieren Sie es aus.) Wir haben `r` eingegeben, um die Wahl zu verwerfen.

```
Unification #2 of 2 (weight = 21):
  Replace "$6" with "( $9 + 0 ) = $9"
  Accept (A), reject (R), or quit (Q) <A>? q
To undo the assignment, DELETE STEP 4 and INITIALIZE, UNIFY
if needed.
 7     min=?  $? |- $8
 8     maj=?  $? |- ( $8 -> ( $6 -> t = t ) )
```

Die zweite Wahlmöglichkeit ist richtig, und normalerweise würden wir `a` eingeben, um sie zu akzeptieren. Stattdessen haben wir `q` eingegeben, um zu zeigen, was passieren wird: Der Schritt wird mit einer unbekannten Vereinheitlichung verlassen, die wie folgt aussehen kann:

```
MM-PA> show new_proof/not_unified
 4   min    $a |- $6
        =a2  = |- ( $9 + 0 ) = $9
```

Später können wir diesen Schritt mit dem Befehl `unify all/interactive` vereinheitlichen.

Es ist wichtig, sich daran zu erinnern, dass Ihnen bei der Eingabe eines Beweises gelegentlich mehrere Vereinheitlichungsmöglichkeiten angeboten werden, wenn das Programm feststellt, dass noch nicht genügend Informationen vorhanden sind, um automatisch eine eindeutige Wahl zu treffen (und das kann sogar bei ausgeschalteter `set empty_substitution` passieren). Normalerweise ist es während der Inspektion der Auswahlmöglichkeiten offensichtlich, welche Wahl die richtige ist, da die falsche Wahl zu sinnlosen Fragmenten von wffs führt. Außerdem ist die richtige Wahl normalerweise die erste, die präsentiert wird, im Gegensatz zu unserem obigen Beispiel.

Genug der Abschweifung. Kehren wir zur Standardeinstellung zurück.

```
MM-PA> set empty_substitution off
The ability to substitute empty expressions for variables
has been turned off.  Note that this may make the Proof
Assistant too restrictive in some cases.
```

Wenn wir den Beweis löschen, neu beginnen und zu dem Punkt gelangen, an dem wir oben abgeschweift sind, gibt es keine mehrdeutige Vereinheitlichung mehr.

```
MM-PA> assign 3 a2
To undo the assignment, DELETE STEP 4 and INITIALIZE, UNIFY
if needed.
 7     min=?  $? |- $4
 8     maj=?  $? |- ( $4 -> ( ( $5 + 0 ) = $5 -> t = t ) )
```

Schauen wir uns unseren bisherigen Beweis an und fahren wir fort.

```
MM-PA> show new_proof/lemmon
 4 a2            $a |- ( $5 + 0 ) = $5
 7 ?             $? |- $4
 8 ?             $? |- ( $4 -> ( ( $5 + 0 ) = $5 -> t = t ) )
 9 7,8 mp        $a |- ( ( $5 + 0 ) = $5 -> t = t )
10 4,9 mp        $a |- t = t
MM-PA> assign 8 a1
To undo the assignment, DELETE STEP 11 and INITIALIZE, UNIFY
if needed.
 7     min=?  $? |- ( t + 0 ) = t
MM-PA> assign 7 a2
To undo the assignment, DELETE STEP 8 and INITIALIZE, UNIFY
if needed.
MM-PA> show new_proof/lemmon
 4 a2            $a |- ( t + 0 ) = t
 8 a2            $a |- ( t + 0 ) = t
12 a1            $a |- ( ( t + 0 ) = t -> ( ( t + 0 ) = t ->
                                                t = t ) )
13 8,12 mp       $a |- ( ( t + 0 ) = t -> t = t )
14 4,13 mp       $a |- t = t
```

Nun sind alle temporären Variablen und unbekannten Schritte aus dem „wesentlichen“ Teil des Beweises entfernt worden. Wenn dieser Zustand erreicht ist, kann der Beweis-Assistent den Rest des Beweises in der Regel automatisch bestimmen (angestoßen durch den Befehl `improve` - beachten Sie, dass der Befehl `improve` gelegentlich auch zum Ausfüllen wesentlicher Schritte genutzt werden kann. Es wird dabei aber nur versucht Aussagen zu verwenden, die keine neuen Variablen in ihren Hypothesen einführen, was bei `mp` nicht der Fall ist. Es wird auch nicht versucht, Schritte zu verbessern,

die temporäre Variablen enthalten). Schauen wir uns den vollständigen Beweis nochmals an, führen dann den Befehl `improve` aus und sehen ihn uns dann noch einmal an.

```
MM-PA> show new_proof/lemmon/all
 1 ?             $? wff ( t + 0 ) = t
 2 ?             $? wff t = t
 3 ?             $? term t
 4 3 a2          $a |- ( t + 0 ) = t
 5 ?             $? wff ( t + 0 ) = t
 6 ?             $? wff ( ( t + 0 ) = t -> t = t )
 7 ?             $? term t
 8 7 a2          $a |- ( t + 0 ) = t
 9 ?             $? term ( t + 0 )
10 ?             $? term t
11 ?             $? term t
12 9,10,11 a1    $a |- ( ( t + 0 ) = t -> ( ( t + 0 ) = t ->
                                                  t = t ) )
13 5,6,8,12 mp   $a |- ( ( t + 0 ) = t -> t = t )
14 1,2,4,13 mp   $a |- t = t

MM-PA> improve all
A proof of length 1 was found for step 11.
A proof of length 1 was found for step 10.
A proof of length 3 was found for step 9.
A proof of length 1 was found for step 7.
A proof of length 9 was found for step 6.
A proof of length 5 was found for step 5.
A proof of length 1 was found for step 3.
A proof of length 3 was found for step 2.
A proof of length 5 was found for step 1.
Steps 1 and above have been renumbered.
CONGRATULATIONS!  The proof is complete.  Use SAVE
NEW_PROOF to save it.  Note:  The Proof Assistant does
not detect $d violations.  After saving the proof, you
should verify it with VERIFY PROOF.
```

Der `save new_proof` speichert den Beweis in der Datenbasis. Hier wird er nur in einer Form angezeigt, die aus einer Protokolldatei ausgeschnitten und mit einem Texteditor manuell in die Quelldatei der Datenbasis eingefügt werden kann.

```
MM-PA> show new_proof/normal
---------Clip out the proof below this line:
      tt tze tpl tt weq tt tt weq tt a2 tt tze tpl tt weq
      tt tze tpl tt weq tt tt weq wim tt a2 tt tze tpl tt
```

```
      tt a1 mp mp $.
---------The proof of 'th1' to clip out ends above this line.
```

Es gibt ein weiteres Beweis-Format, der „komprimierte“ Beweis, das Sie in Datenbasen sehen werden. Es ist nicht wichtig zu verstehen, wie es kodiert ist, sondern nur es zu erkennen, wenn Sie es sehen. Sein einziger Zweck ist es, den Speicherbedarf für große Beweise zu verringern. Ein komprimierter Beweis kann immer in einen normalen umgewandelt werden und umgekehrt, und der Metamath-Befehl `show proof` funktioniert genauso gut mit komprimierten Beweisen. Das komprimierte Beweisformat wird im Anhang B beschrieben.

```
MM-PA> show new_proof/compressed
---------Clip out the proof below this line:
      ( tze tpl weq a2 wim a1 mp ) ABCZADZAADZAEZJJKFLIA
      AGHH $.
---------The proof of 'th1' to clip out ends above this line.
```

Nun beenden wir den Beweis-Assistenten. Da wir Änderungen an dem Beweis vorgenommen haben, werden wir gewarnt, dass wir ihn nicht gespeichert haben. In diesem Fall ist uns das egal.

```
MM-PA> exit
Warning:  You have not saved changes to the proof.
Do you want to EXIT anyway (Y, N) <N>? y
Exiting the Proof Assistant.
Type EXIT again to exit Metamath.
```

Der Beweis-Assistent verfügt über verschiedene andere Befehle, die Ihnen bei der Erstellung von Beweisen helfen können. Siehe Abschnitt 5.6 für eine Liste dieser Befehle.

Ein oft nützlicher Befehl ist `minimize_with*/brief`, der versucht, den Beweis zu verkürzen. Er kann den Prozess des Beweisens effizienter machen, indem er Sie einen etwas „schlampigen“ Beweis schreiben lässt und ihn dann durch einige feine Optimierungsdetails für Sie bereinigt (obwohl er keine Wunder vollbringen kann, wie z.B. die Umstrukturierung des gesamten Beweises).

2.5 Hinweise zur Bearbeitung einer Datenbasis

Sobald die Quelldatei ihrer Datenbasis Beweise enthält, gibt es einige Einschränkungen für deren Bearbeitung, damit die Beweise gültig bleiben. Diese Regeln sollten Sie besonders beachten, da Sie sonst mühsam erzielte Ergebnisse verlieren können. Es ist sinnvoll, alle Beweise regelmäßig mit `verify proof *` zu überprüfen, um ihre Integrität sicherzustellen.

Wenn Ihre Datei nur normale (im Gegensatz zu komprimierten) Beweise enthält, besteht die Hauptregel darin, dass Sie die Reihenfolge der obligatorischen Hypothesen im Index einer Aussage, auf die in einem späteren Beweis verwiesen wird, nicht ändern dürfen. Wenn Sie zum Beispiel die Reihenfolge der Haupt- und Nebenhypothese in dem Modus ponens vertauschen, werden alle Beweise, die diese Regel verwenden, falsch. Der Befehl `show statement` zeigt Ihnen die obligatorischen Hypothesen einer Aussage und ihre Reihenfolge.

Wenn eine Aussage einen komprimierten Beweis hat, dürfen Sie auch nicht die Reihenfolge *ihrer* obligatorischen Hypothesen ändern. Das komprimierte Beweisformat verwendet diese Information als Teil der Komprimierungstechnik. Beachten Sie, dass das Vertauschen der Namen zweier Variablen in einem Theorem die Reihenfolge der zwingenden Hypothesen ändert.

Der sicherste Weg, eine Aussage, z. B. `mytheorem`, zu bearbeiten besteht darin, sie zu duplizieren und das Original in der gesamten Datenbasis in `mytheoremOLD` umzubenennen. Sobald die bearbeitete Version erneut bewiesen ist, können alle Aussagen, die auf `mytheoremOLD` verweisen, im Beweis-Assistenten mit dem Befehl `minimize_with mytheorem/allow_growth` aktualisiert werden.

Kapitel 3

Abstrakte Mathematik enthüllt

3.1 Logik und Mengenlehre

Die Mengenlehre kann als eine Form einer exakten Theologie betrachtet werden.

Rudy Rucker[1]

Trotz ihrer vermeintlichen Komplexität lässt sich die gesamte Standardmathematik, egal wie tief oder abstrakt sie ist, erstaunlicherweise aus einem relativ kleinen Satz von Axiomen oder ersten Prinzipien ableiten. Die Entwicklung dieser Axiome gehört zu den beeindruckendsten und wichtigsten Errungenschaften der Mathematik im 20. Jahrhundert. Letztlich lassen sich diese Axiome in eine Reihe von Regeln für die Handhabung von Symbolen herunterbrechen, denen jeder technisch orientierte Mensch folgen kann.

Wir werden nicht viel Zeit darauf verwenden, ein tiefes, übergeordnetes Verständnis der Bedeutung der Axiome zu vermitteln. Diese Art von Verständnis erfordert ein gewisses Maß an mathematischer Raffinesse sowie ein Verständnis der Philosophie, die den Grundlagen der Mathematik zugrunde liegt, und entwickelt sich in der Regel im Laufe der Zeit, wenn Sie mit Mathematik arbeiten. Unser Ziel ist es stattdessen, Ihnen die unmittelbare Fähigkeit zu vermitteln zu verstehen, wie Theoreme aus den Axiomen und aus anderen Theoremen abgeleitet werden. Dies ist vergleichbar mit dem Erlernen der Syntax einer Computersprache, die es Ihnen ermöglicht, die Details eines Programms zu verstehen, Ihnen aber nicht unbedingt die Fähigkeit verleiht, nicht-triviale Programme selbst zu schreiben - eine Fähigkeit, die sich erst mit der Zeit entwickelt. Lassen Sie sich vorerst nicht von den

[1] Frei übersetzt nach [3], S. 31.

abstrakt klingenden Namen der Axiome beunruhigen, sondern konzentrieren Sie sich auf die Regeln zur Manipulation der Symbole, die den einfachen Konventionen der Metamath-Sprache folgen.

Die Axiome, die der gesamten Standardmathematik zugrunde liegen, bestehen aus Axiomen der Logik und Axiomen der Mengenlehre. Die Axiome der Logik sind in zwei Unterkategorien unterteilt, die Aussagenlogik (manchmal auch Satzlogik genannt) und die Prädikatenlogik (manchmal auch Logik erster Ordnung oder Quantentheorie genannt). Die Aussagenlogik ist eine Voraussetzung für die Prädikatenlogik, und die Prädikatenlogik ist eine Voraussetzung für die Mengenlehre. Die am häufigsten verwendete Version der Mengenlehre ist die Zermelo-Fraenkel-Mengenlehre mit dem Auswahlaxiom (engl. „axiom of choice"), oft abgekürzt als ZFC.

Hier ist in aller Kürze dargestellt, worum es bei den Axiomen geht, und zwar auf informelle Art und Weise. Die Verbindung zwischen dieser Beschreibung und den Symbolen, die wir Ihnen zeigen werden, wird nicht sofort offensichtlich sein und muss es im Prinzip auch nicht. Unsere Beschreibung versucht lediglich zusammenzufassen, worüber Mathematiker nachdenken, wenn sie mit den Axiomen arbeiten.

Logik ist eine Reihe von Regeln, die es uns ermöglichen, Wahrheiten aus anderen Wahrheiten abzuleiten. Anders ausgedrückt, ist Logik mehr oder weniger die Übersetzung dessen, was wir als gesunden Menschenverstand betrachten würden, in einen strengen Satz von Axiomen. Angenommen, φ, ψ und χ (die griechischen Buchstaben phi, psi und chi) stehen für Aussagen, die entweder wahr oder falsch sind, und x ist eine Variable, die sich über eine Gruppe mathematischer Objekte (Mengen, ganze Zahlen, reelle Zahlen usw.) erstreckt. In der Mathematik ist eine „Aussage" eigentlich eine Formel, und ψ könnte z.B. „$x = 2$" sein. Die Aussagenlogik erlaubt uns, Variablen zu verwenden, die entweder wahr oder falsch sind, und Schlüsse zu ziehen wie: „Wenn ψ aus φ und χ aus ψ folgt, dann folgt χ aus φ." Die Prädikatenlogik erweitert die Aussagenlogik, indem sie auch Aussagen über Objekte (nicht nur über Wahrheitswerte) erlaubt, einschließlich Aussagen über „alle" Objekte oder „wenigstens ein" Objekt. Die Prädikatenlogik erlaubt es zum Beispiel zu sagen: „Wenn φ für alle x wahr ist, dann ist φ für einige x wahr." Die in `set.mm` verwendete Logik ist die klassische Standardlogik (im Gegensatz zu anderen Logiksystemen wie der intuitionistischen Logik).

Die Mengenlehre befasst sich mit der Handhabung von Objekten und Sammlungen von Objekten, insbesondere mit den abstrakten, imaginären Objekten, mit denen sich die Mathematik beschäftigt, wie z. B. Zahlen. Alles, was es in der Mathematik geben soll, wird als Menge betrachtet. Eine Menge, die als leere Menge bezeichnet wird, enthält nichts. Wir stellen die leere Menge durch $\varnothing$ dar. Viele Mengen können aus der leeren Menge aufgebaut werden. Es gibt eine durch $\{\varnothing\}$ dargestellte Menge, die die leere Menge enthält, eine weitere durch $\{\varnothing, \{\varnothing\}\}$ dargestellte Menge, die diese Menge sowie die leere Menge enthält, eine weitere durch $\{\{\varnothing\}\}$ dargestellte Menge,

die nur die Menge enthält, die die leere Menge enthält, und so weiter ad infinitum. Alle mathematischen Objekte, egal wie komplex sie sind, werden als identisch mit bestimmten Mengen definiert: Die ganze Zahl 0 ist definiert als die leere Menge, die ganze Zahl 1 ist definiert als $\{\varnothing\}$, die ganze Zahl 2 ist definiert als $\{\varnothing, \{\varnothing\}\}$. (Wie diese Definitionen gewählt wurden, spielt jetzt keine Rolle: die Idee dahinter ist, dass diese Mengen die Eigenschaften haben, die wir von ganzen Zahlen erwarten, sobald geeignete Operationen definiert sind.) Mathematische Operationen, wie z. B. die Addition, werden in Form von Operationen auf Mengen definiert - ihre Vereinigung, ihre Schnittmenge usw. - Operationen, die Sie vielleicht schon in der Grundschule verwendet haben, als Sie mit Gruppen von Äpfeln und Orangen gearbeitet haben.

Die Axiome postulieren auch die Existenz unendlicher Mengen, wie z.B. die Menge aller nichtnegativen ganzen Zahlen $(0, 1, 2, \ldots$, auch „natürliche Zahlen“ genannt). Diese Menge kann nicht mit der soeben gezeigten Klammerschreibweise dargestellt werden, sondern erfordert eine kompliziertere Schreibweise, die „Klassenabstraktion“. Zum Beispiel bedeutet die unendliche Menge $\{x|$„x ist eine natürliche Zahl“$\}$ die „Menge aller Objekte x, so dass x eine natürliche Zahl ist“, d.h. die Menge der natürlichen Zahlen; dabei ist „x eine natürliche Zahl“ eine ziemlich komplizierte Formel, wenn man sie mit den primitiven Symbole ausdrückt. [2] Die primitiven Symbole enthalten tatsächlich noch nicht einmal die Klammerschreibweise. Die Klammerschreibweise ist eine übergeordnete Definition, die Sie im Abschnitt 3.4 finden können.

Interessanterweise können die Arithmetik der ganzen Zahlen und die Arithmetik der rationalen Zahlen entwickelt werden, ohne sich auf die Existenz einer unendlichen Menge zu berufen, während die Arithmetik der reellen Zahlen dies erfordert.

Jede Variable in den Axiomen der Mengenlehre stellt eine beliebige Menge dar, und die Axiome geben auf einer sehr primitiven Ebene die zulässigen Möglichkeiten an, die man mit diesen Variablen tun kann.

Sie denken jetzt vielleicht, dass Zahlen und Arithmetik viel intuitiver und grundlegender sind als Mengen und daher die Grundlage der Mathematik sein sollten. Aber in Wirklichkeit haben Sie Ihr ganzes Leben lang

[2] Die Aussage „x ist eine natürliche Zahl“ wird formal ausgedrückt als „$x \in \omega$“, wobei $\in$ (stilisiertes Epsilon) bedeutet „ist in“ oder „ist ein Element von“ und ω (omega) bedeutet „die Menge der natürlichen Zahlen“. Wenn „$x \in \omega$“ vollständig durch die primitiven Symbole der Mengenlehre ausgedrückt wird, ist das Ergebnis $\neg$ ($\neg$ ($\forall$ z ($\neg$ $\forall$ w ($z \in w$ $\rightarrow$ $\neg$ $w \in x$) $\rightarrow$ $z \in x$) $\rightarrow$ ($\forall$ z ($\neg$ ($\forall$ w ($w \in z \rightarrow w \in x$) $\rightarrow$ $\forall$ w $\neg w \in z$) $\rightarrow$ $\neg$ $\forall$ w ($w \in z \rightarrow$ $\neg$ $\forall$ v ($v \in z \rightarrow \neg v \in w$))) $\rightarrow$ $\neg$ $\forall$ z $\forall$ w($\neg$ ($z \in x \rightarrow \neg w \in x$)$\rightarrow$ ($\neg$ $z \in w$ $\rightarrow$ ($\neg$ $z = w \rightarrow w \in z$))))) $\rightarrow$ $\neg\forall$ y ($\neg$ ($\neg$ ($\forall$ z ($\neg$ $\forall w$ ($z \in w \rightarrow \neg w \in y$) $\rightarrow z \in y$) $\rightarrow$ ($\forall$ z ($\neg$ ($\forall w$ ($w \in z \rightarrow w \in y$) $\rightarrow$ $\forall w$ $\neg w \in z$) $\rightarrow$ $\neg$ $\forall$ w ($w \in z \rightarrow$ $\neg$ $\forall$ v ($v \in z \rightarrow \neg v \in w$))) $\rightarrow$ $\neg$ $\forall$ $z\forall$ w ($\neg$ ($z \in y \rightarrow \neg w \in y$) $\rightarrow$ ($\neg$ $z \in w \rightarrow$ ($\neg$ $z = w \rightarrow w \in z$)))) $\rightarrow$ ($\forall$ z $\neg$ $z \in y \rightarrow$ $\neg$ $\forall$ w ($\neg$ ($w \in y \rightarrow \neg$ $\forall$ z ($w \in z \rightarrow \neg$ $z \in y$)) $\rightarrow$ $\neg$ ($\neg$ $\forall$ z ($w \in z \rightarrow \neg$ $z \in y$) $\rightarrow$ $w \in y$)))) $\rightarrow$ $x \in y$))). Abschnitt 3.4 zeigt die Hierarchie der Definitionen, die zu diesem Ausdruck führt.

mit Zahlen zu tun gehabt und sind mit einigen Regeln für ihre Handhabung vertraut, wie z. B. Addition und Multiplikation. Diese Regeln decken nur einen kleinen Teil dessen ab, was man mit Zahlen tun kann, und nur einen winzigen Teil der übrigen Mathematik. Wenn Sie sich ein beliebiges Buch über Zahlentheorie ansehen, werden Sie schnell verloren sein, wenn dies die einzigen Regeln sind, die Sie kennen. Auch wenn solche Bücher eine Liste von „Axiomen“ für die Arithmetik enthalten, erfordert die Fähigkeit, die Axiome zu verwenden und Beweise von Theoremen (Fakten) über Zahlen zu verstehen - ein implizites mathematisches Talent, das viele Menschen davon abhält, abstrakte Mathematik zu studieren. Die Art von Mathematik, die die meisten Menschen kennen, beschränkt sie auf den praktischen, alltäglichen Gebrauch der blinden Manipulation von Zahlen und Formeln, ohne dass sie verstehen, warum diese Regeln richtig sind, und ohne weiter darüber nachzudenken. Wissen Sie zum Beispiel, warum die Multiplikation von zwei negativen Zahlen eine positive Zahl ergibt? Wenn Sie mit der Mengenlehre beginnen, werden Sie ebenfalls damit beginnen, Symbole blind nach den von uns vorgegebenen Regeln zu manipulieren, allerdings mit dem Vorteil, dass diese Regeln Ihnen im Prinzip den Zugang zur *gesamten* Mathematik ermöglichen, nicht nur zu einem winzigen Teil davon.

Natürlich sind konkrete Beispiele oft hilfreich für den Lernprozess. So kann man zum Beispiel überprüfen, dass $2 \cdot 3 = 3 \cdot 2$ ist, indem man Objekte gruppiert, und man kann leicht „sehen“, wie dies zu $x \cdot y = y \cdot x$ verallgemeinert wird, auch wenn man nicht in der Lage ist, es rigoros zu beweisen. In ähnlicher Weise kann es in der Mengenlehre hilfreich sein zu verstehen, wie die Axiome der Mengenlehre auf kleine endliche Sammlungen von Objekten anwendbar (und korrekt) sind. Sie sollten sich darüber im Klaren sein, dass die Intuition in der Mengenlehre für unendliche Sammlungen irreführend sein kann, und dass deshalb strenge Beweise wichtiger werden. Zum Beispiel ist $x \cdot y = y \cdot x$ zwar für endliche Ordinalzahlen (die natürlichen Zahlen) richtig, aber nicht für unendliche Ordinalzahlen.

3.2 Die Axiome für die gesamte Mathematik

In diesem Abschnitt zeigen wir Ihnen die Axiome für die gesamte Standardmathematik (d.h. Logik und Mengenlehre), wie sie traditionell dargestellt werden. Die traditionelle Darstellung ist nützlich für jemanden, der über die nötige mathematische Erfahrung verfügt, um abstrakte Konzepte auf hohem Niveau korrekt zu handhaben. Für jemanden, der nicht über dieses Talent verfügt, kann es schwierig sein zu wissen, wie man diese Axiome tatsächlich anwendet. Der Zweck dieses Abschnitts ist es, Ihnen zu zeigen, wie sich die Version der Axiome, die in der Standard-Metamath-Datenbasis `set.mm` verwendet wird, zu der typischen Version in Lehrbüchern verhält, damit Sieein informelles Gespür dafür entwickeln.

3.2.1 Aussagenlogik

Die Aussagenlogik beschäftigt sich mit Aussagen, die entweder als wahr oder falsch interpretiert werden können. Einige Beispiele für entweder wahre oder falsche Aussagen (außerhalb der Mathematik) sind „Es regnet heute“ und „Die Vereinigten Staaten haben einen weiblichen Präsidenten“. In der Mathematik sind Aussagen, wie wir bereits erwähnt haben, eigentlich Formeln.

In der Aussagenlogik ist uns der Inhalt der Aussagen egal. Auch eine logische Kombination von Aussagen, wie „Es regnet heute und die Vereinigten Staaten haben eine weibliche Präsidentin“, wird nicht anders behandelt als eine Einzelaussage. Aussagen und ihre Kombinationen werden wohlgeformte Formeln (wffs) genannt. Wir definieren wffs nur in Bezug auf andere wffs und definieren nicht, was eine „initiale“ wff ist. Wie in der Literatur üblich, verwenden wir kleine griechische Buchstaben zur Darstellung von wffs.

Nehmen wir konkret an, dass φ und ψ wffs sind. Dann sind die Kombinationen $\varphi \to \psi$ („φ impliziert ψ“, auch gelesen als „wenn φ dann ψ“ oder „aus φ folgt ψ“) und $\neg\varphi$ („nicht φ“) ebenfalls wffs.

Die drei Axiome der Aussagenlogik sind alle wffs der folgenden Form:[3]

$$\varphi \to (\psi \to \varphi)$$
$$(\varphi \to (\psi \to \chi)) \to ((\varphi \to \psi) \to (\varphi \to \chi))$$
$$(\neg\varphi \to \neg\psi) \to (\psi \to \varphi)$$

Diese drei Axiome sind weit verbreitet. Sie werden Jan Łukasiewicz (sprich: wu-kah-schei-witsch) zugeschrieben und wurden von Alonzo Church veröffentlicht, der sie als System P2 bezeichnete. (Vielen Dank an Ted Ulrich für diese Information.)

Es gibt eine unendliche Anzahl von Axiomen, eines für jede mögliche wff der obigen Form. (Aus diesem Grund werden Axiome wie die obigen oft als „Axiomenschema“) bezeichnet.) Jeder griechische Buchstabe in den Axiomen kann durch eine komplexeres wff ersetzt werden, um ein anderes Axiom zu erhalten. Ersetzt man beispielsweise φ im ersten Axiom durch $\neg(\varphi \to \chi)$, erhält man $\neg(\varphi \to \chi) \to (\psi \to \neg(\varphi \to \chi))$, was immer noch ein Axiom ist.

Um aus den Axiomen neue wahre Aussagen (Theoreme) abzuleiten, wird eine Schlussregel namens „Modus ponens“ verwendet. Diese Regel besagt, dass, wenn die wff φ ein Axiom oder ein Theorem ist, und die wff $\varphi \to \psi$ ein Axiom oder ein Theorem ist, dann ist die wff ψ auch ein Theorem.

Ein nicht-mathematisches Beispiel für den Modus ponens: Nehmen wir an, wir haben bewiesen (oder als Axiom genommen): „Bob ist ein Mann“, und haben separat bewiesen (oder als Axiom genommen): „Wenn Bob ein Mann ist, dann ist Bob ein Mensch.“ Mit Hilfe des Modus ponens können wir logisch folgern: „Bob ist ein Mensch.“

[3]Ein bemerkenswertes Ergebnis von C. A. Meredith quetscht diese drei Axiome in ein einziges Axiom $((((\varphi \to \psi) \to (\neg\chi \to \neg\theta)) \to \chi) \to \tau) \to ((\tau \to \varphi) \to (\theta \to \varphi))$ [44], das als das kürzest mögliche gilt.

Aus der Sicht von Metamath definieren die Axiome und die Schlussregel Modus ponens lediglich ein mechanisches Mittel zur Ableitung neuer wahrer Aussagen aus bestehenden wahren Aussagen, und das ist der vollständige Inhalt der Aussagenlogik, soweit es Metamath betrifft. Sie können ein Logik-Lehrbuch lesen, um ein besseres Verständnis ihrer Bedeutung zu erlangen, oder Sie können sich ihre Bedeutung einfach langsam erschließen, nachdem Sie sie eine Weile benutzt haben.

Es ist eigentlich recht einfach zu prüfen, ob eine Formel ein Satz (oder Theorem) der Aussagenlogik ist. Theoreme der Aussagenlogik werden auch als „Tautologien“ bezeichnet. Die Technik, mit der man überprüfen kann, ob eine Formel eine Tautologie ist, wird als „Wahrheitstabellenmethode“ bezeichnet und funktioniert folgendermaßen. Eine wff $\varphi \rightarrow \psi$ ist falsch, wenn φ wahr und ψ falsch ist. Andernfalls ist sie wahr. Eine wff $\neg\varphi$ ist immer dann falsch, wenn φ wahr ist und ansonsten falsch. Um eine Tautologie wie $\varphi \rightarrow (\psi \rightarrow \varphi)$ zu verifizieren, zerlegt man sie in Teil-wffs und konstruiert eine Wahrheitstabelle, die alle möglichen Kombinationen von wahr (W) und falsch (F) berücksichtigt, die den wff-Metavariablen zugeordnet sind:

φ	ψ	$\psi \rightarrow \varphi$	$\varphi \rightarrow (\psi \rightarrow \varphi)$
W	W	W	W
W	F	W	W
F	W	F	W
F	F	W	W

Wenn alle Einträge in der letzten Spalte wahr (W) sind, ist die Formel eine Tautologie.

Die Wahrheitstabellen-Methode sagt Ihnen nicht, wie Sie die Tautologie aus den Axiomen beweisen können, sondern nur, dass ein Beweis existiert. Einen tatsächlichen Beweis zu finden (insbesondere einen, der kurz und elegant ist), kann eine Herausforderung sein. Es gibt zwar Methoden zur automatischen Generierung von Beweisen in der Aussagenlogik, aber die daraus resultierenden Beweise können manchmal sehr lang sein. In der Metamath-Datenbasis `set.mm` wurden die meisten oder sogar alle Beweise manuell erstellt.

In Abschnitt 3.4.1 werden verschiedene Definitionen erörtert, die die Verwendung der Aussagenlogik erleichtern. Wir definieren zum Beispiel:

- $\varphi \vee \psi$ ist wahr, wenn entweder φ oder ψ (oder beide) wahr sind (dies ist die Disjunktion alias logisches ODER).
- $\varphi \wedge \psi$ ist wahr, wenn sowohl φ als auch ψ wahr sind (dies ist die Konjunktion alias logisches UND).
- $\varphi \leftrightarrow \psi$ ist wahr, wenn φ und ψ denselben Wert haben, d. h. beide wahr oder beide falsch sind (dies ist die Äquivalenz oder das Bikonditional).

3.2.2 Prädikatenlogik

Die Prädikatenlogik führt das Konzept der „individuellen Variablen“ ein, die wir in der Regel einfach „Variablen“ nennen werden. Diese Variablen können etwas anderes als wahr oder falsch (wffs) darstellen und werden immer Mengen repräsentieren, sobald wir zur Mengenlehre kommen. Es gibt auch drei neue Symbole $\forall$, $=$ und $\in$, die jeweils „für alle“, „gleich“ und „ist ein Element von“ bedeuten. Wir werden Variablen mit lateinischen Kleinbuchstaben wie x, y, z und w darstellen, wie es in der Literatur üblich ist. Zum Beispiel bedeutet $\forall x\varphi$: „Für alle möglichen Werte von x ist φ wahr.“

In der Prädikatenlogik erweitern wir die Definition einer wff. Wenn φ eine wff ist und x und y Variablen sind, dann sind $\forall x\,\varphi$, $x = y$, und $x \in y$ wffs. Man beachte, dass diese drei neuen Arten von wffs als „initiale wffs“ betrachtet werden können, aus denen man andere wffs mit $\rightarrow$ und $\neg$ aufbauen kann. Das Konzept einer initialen wff war in der Aussagenlogik nicht vorhanden. Aber egal ob initiale wff oder nicht, uns interessiert wirklich nur, ob unsere wffs korrekt nach diesen mechanischen Regeln konstruiert sind.

Eine kurze Anmerkung: Um Verwirrung zu vermeiden, ist es an dieser Stelle vielleicht am besten, sich die Variablen von Metamath als „Metavariablen“ vorzustellen, weil sie nicht ganz dasselbe sind wie die Variablen, die wir hier vorstellen. Eine (Meta-)Variable in Metamath kann ein wff oder eine individuelle Variable sein, sowie viele andere Dinge; im Allgemeinen stellt sie eine Art Platzhalter für eine nicht spezifizierte Folge von mathematischen Symbolen dar.

Anders als in der Aussagenlogik gibt es kein Entscheidungsverfahren analog zur Wahrheitstabellenmethode (und kann es theoretisch auch nicht geben), mit dem man sicher feststellen kann, ob eine Formel ein Satz der Prädikatenlogik ist. Ein großer Teil der Arbeit auf dem Gebiet des automatischen Theorembeweisens wurde der Entwicklung cleverer Heuristiken zum Beweisen von Theoremen der Prädikatenlogik gewidmet, aber es kann nie garantiert werden, dass sie immer funktionieren.

In Abschnitt 3.4.2 werden verschiedene Definitionen erörtert, die die Anwendung der Prädikatenlogik erleichtern. Zum Beispiel definieren wir $\exists x\varphi$ als „Es gibt mindestens einen möglichen Wert von x, bei dem φ wahr ist.“

Wir wenden uns nun der Frage zu, wie die Prädikatenklogik formal aufgebaut werden kann.

Gängige Axiome

Es gibt eine neue Schlussregel in der Prädikatenlogik: Wenn φ ein Axiom oder ein Theorem ist, dann ist $\forall x\,\varphi$ auch ein Theorem. Dies nennt man die „Regel der Verallgemeinerung“. Dies lässt sich in Metamath leicht darstellen.

In Standardtexten der Logik gibt es oft zwei Axiome der Prädikatenlogik:

$\forall x\,\varphi(x) \rightarrow \varphi(y)$, wobei „$y$ echt durch x ersetzt wird“.
$\forall x(\varphi \rightarrow \psi) \rightarrow (\varphi \rightarrow \forall x\,\psi)$, wobei „$x$ in φ nicht frei ist“.

Auf den ersten Blick erscheint dies einfach: nur zwei Axiome. Allerdings sind an jedes Axiom Bedingungsklauseln angehängt, die rätselhaft erscheinende Anforderungen beschreiben. Außerdem setzt das erste Axiom nach jeder wff ein variables Symbol in Klammern, was offensichtlich gegen unsere Definition einer wff verstößt; dies ist nur eine informelle Art, auf eine beliebige Variable zu verweisen, die in der wff vorkommen kann. Die Konditionalklauseln haben natürlich eine genaue Bedeutung, aber wie sich herausstellt, ist die genaue Bedeutung etwas kompliziert und schwer in einer Weise zu formalisieren, die ein Computer leicht handhaben kann. Anders als bei der Aussagenlogik ist ein gewisses Maß an mathematischer Raffinesse und Übung erforderlich, um diese Konzepte leicht zu erfassen und korrekt zu handhaben.

Die Prädikatenlogik kann mit oder ohne Gleichheitsaxiome dargestellt werden. Wir werden die Gleichheitsaxiome als Voraussetzung für die von uns verwendete Version der Mengenlehre benötigen. Die Gleichheitsaxiome werden, wenn sie enthalten sind, oft durch diese beiden Axiome dargestellt:

$$x = x$$
$$x = y \to (\varphi(x, x) \to \varphi(x, y)),$$
wobei sich „$\varphi(x, y)$ aus $\varphi(x, x)$ ergibt, indem einige, aber nicht notwendigerweise alle Vorkommen von x durch y ersetzt werden, vorausgesetzt, dass y für x in $\varphi(x, x)$ frei ist“.

Das erste Gleichheitsaxiom ist einfach, aber auch hier ist die Bedingung für das zweite Axiom etwas umständlich auf einem Computer umzusetzen.

Tarski-System S2

Natürlich sind wir nicht die Ersten, die die Komplikationen dieser Axiome der Prädikatenlogik bemerken, wenn man es genau nimmt.

Der bekannte Logiker Alfred Tarski veröffentlichte 1965 ein System, das er als System S2 bezeichnete [69, S. 77]. Tarskis System ist *exakt äquivalent* zu der traditionellen Lehrbuchformalisierung, aber (durch geschickten Gebrauch von Gleichheitsaxiomen) eliminiert es die primitiven Begriffe „echte Substitution“ und „freie Variable“ und ersetzt sie durch eine direkte Substitution und den Begriff einer Variable, die nicht in einer Formel vorkommt (was wir mit Nebenbedingungen für verschiedene Variablen ausdrücken).

Als er für sein System plädierte, schrieb Tarski: „Der relativ komplizierte Charakter von [freien Variablen und echter Substitution] ist eine Quelle gewisser Unannehmlichkeiten sowohl praktischer als auch theoretischer Natur; dies zeigt sich deutlich sowohl beim Unterrichten eines Grundkurses der mathematischen Logik als auch bei der Formalisierung der Syntax der Prädikatenlogik für einige theoretische Zwecke“[69, S. 61].

Entwicklung einer Metamath-Darstellung

Die Standard-Lehrbuch-Axiome der Prädikatenlogik sind aufgrund der komplexen Begriffe der „freien Variablen“ und der „echten Substitution“ etwas umständlich auf einem Computer zu implementieren. Obwohl es möglich ist, diese Konzepte mit der Metamath-Sprache umzusetzen, haben wir uns dafür entschieden, sie nicht als primitive Konstrukte in der `set.mm`-Datenbasis der Mengenlehre zu implementieren. Stattdessen haben wir sie aus den Axiomen eliminiert, indem wir, auf Tarskis System S2 aufbauend, die Axiome sorgfältig umformuliert haben, dass sie vermieden werden. Dies macht es einem Anfänger leicht, den Schritten eines Beweises zu folgen, ohne irgendwelche fortgeschrittenen Konzepte zu kennen, außer dem einfachen Konzept der Ersetzung von Variablen durch Ausdrücke.

Um die Konzepte der freien Variablen und der echten Substitution aus den Axiomen zu entwickeln, verwenden wir einen zusätzlichen Metamath-Anweisungstyp namens „disjunkte Variableneinschränkung“, der uns bisher noch nicht begegnet ist. Im Zusammenhang mit den Axiomen bedeutet die Aussage `$d` $x\,y$ einfach, dass x und y verschiedene sein müssen, d.h. sie dürfen nicht gleichzeitig durch dieselbe Variable substituiert werden. Die Aussage `$d` $x\,\varphi$ bedeutet, dass die Variable x nicht in der wff φ vorkommen darf. Für die genaue Definition von `$d` siehe Abschnitt 4.2.4.

Metamath-Darstellung

Das in set.mm definierte Metamath-Axiomensystem für die Prädikatenlogik verwendet das System S2 von Tarski. Wie oben erwähnt, hat dieses eine andere Darstellung als die traditionelle Lehrbuchformalisierung, aber es ist *exakt äquivalent* zur Lehrbuchformalisierung, und es ist *viel* einfacher damit zu arbeiten. Dies wird als System S3 in Abschnitt 6 von Megills Formalisierung [41] wiedergegeben.

Es gibt eine Ausnahme, nämlich Tarskis Axiom der Existenz, das wir als Axiom ax-6 bezeichnen. Im Fall von ax-6 ist Tarskis Version schwächer, weil sie eine disjunkte Variableneinschränkung enthält. Wenn wir wollen, können wir auch unsere Version auf diese Weise abschwächen und haben trotzdem ein metalogisch vollständiges System. Theorem ax-6 zeigt dies, indem es bei Vorhandensein der anderen Axiome unser ax-6 aus Tarskis schwächerer Version ax6v ableitet. Wir haben jedoch die stärkere Version für unser System gewählt, weil sie einfacher zu formulieren und leichter zu benutzen ist.

Tarskis System war eher für den Beweis spezifischer Theoreme als für allgemeinere Theoremschemata konzipiert. Theoremschemata sind jedoch sehr viel effizienter als spezifische Theoreme, wenn es darum geht, einen Bestand an mathematischem Wissen aufzubauen, da sie je nach Bedarf als verschiedene Instanzen wiederverwendet werden können. Tarski leitet zwar einige Theoremschemata aus seinen Axiomen ab, aber ihre Beweise erfordern

Konzepte, die „außerhalb“ des Systems liegen, wie z.B. die Induktion über die Länge von Formeln. Die Verifikation solcher Beweise lässt sich nur schwer mit einem Beweisverifizierer automatisieren. (Konkret behandelt Tarski die Formeln seines Systems als mengentheoretische Objekte. Um die Beweise seiner Theoremschemata zu verifizieren, müsste ein Beweisverifizierer eine beträchtliche Menge an Mengenlehre fest implementiert haben).

Das Metamath-Axiomensystem für die Prädikatenlogik erweitert Tarskis System, um diese Schwierigkeit zu beseitigen. Die zusätzlichen „unterstützenden“ Axiomenschemata (wie wir sie in diesem Abschnitt nennen werden; siehe unten) verleihen Tarskis System eine nette Eigenschaft, die wir metalogische Vollständigkeit nennen [41, Remark 9.6]. Infolgedessen können wir jedes Theoremschema beweisen, das in der „einfachen Metalogik“ des Tarski-Systems ausgedrückt werden kann, indem wir nur die direkte Substitutionsregel von Metamath verwenden, die auf das Axiomensystem angewendet wird (und keine anderen metalogischen oder mengentheoretischen Begriffe „außerhalb“ des Systems). Einfache Metalogik besteht aus Schemata, die wff-Metavariablen (ohne Argumente) und/oder Mengenmetavariablen (auch „individuelle Variablen“ genannt) enthalten, begleitet von optionalen Vorschriften, die jeweils besagen, dass zwei spezifizierte Mengenmetavariablen verschieden sein müssen oder dass eine spezifizierte Mengenmetavariable nicht in einer spezifizierten wff-Metavariable vorkommen darf. Die Axiom- und Regelschemata der Metamath-Logik und der Mengenlehre sind allesamt Beispiele für eine einfache Metalogik. Die Schemata der traditionellen Prädikatenlogik mit Gleichheit sind Beispiele, die keine einfache Metalogik sind, weil sie wff-Metavariablen mit Argumenten verwenden und „frei für“ und „nicht frei in“ als Nebenbedingungen haben.

Eine strenge Begründung für dieses System, das einen älteren, aber genau gleichwertigen Satz von Axiomen verwendet, ist in [41] zu finden.

Dies ermöglicht es uns, in der Metamath-Datenbasis `set.mm` einen anderen Ansatz zu wählen. Wir verwenden keinen der primitiven Begriffe „freie Variable“ und „echte Substitution“ als primitive Konstrukte. Stattdessen verwenden wir eine Reihe von Axiomen, die fast so einfach zu handhaben sind wie die der Aussagenlogik. Unser Axiomensystem vermeidet komplexe primitive Begriffe, indem es die Komplexität wirksam in die Axiome selbst einbettet. Das Ergebnis ist eine größere Anzahl von Axiomen, die jedoch ideal für eine Computersprache wie Metamath geeignet sind. (Abschnitt 3.3 zeigt diese Axiome.)

Wir werden hier nicht weiter auf die Begriffe „freie Variable“ und „echte Substitution“ eingehen. Eine genaue Erklärung dieser Konzepte finden Sie in [21, ch. 3–4] (sowie in vielen anderen Büchenr). Wenn Sie beabsichtigen, ernsthaft mathematisch zu arbeiten, ist es ratsam, sich mit dem traditionellen Lehrbuchansatz vertraut zu machen; auch wenn die in den Axiomen enthaltenen Konzepte ein höheres Maß an Raffinesse erfordern, können diese für den täglichen, informellen Umgang praktikabler sein. Selbst wenn Sie nur

Metamath-Beweise entwickeln, kann Ihnen die Vertrautheit mit dem traditionellen Ansatz helfen, viel schneller zu einem Beweisentwurf zu gelangen, den Sie dann in die von Metamath geforderten Details umsetzen können.

Wir entwickeln später eigene Substitutionsregeln, aber in set.mm sind sie als abgeleitete Konstrukte definiert; sie sind keine Primitive.

Sie sollten auch beachten, dass unser System der Prädikatenlogik speziell auf die Mengenlehre zugeschnitten ist; daher gibt es nur zwei spezifische Prädikate $=$ und $\in$ und keine Funktionen oder Konstanten im Gegensatz zu allgemeineren Systemen. Wir fügen diese später hinzu.

3.2.3 Mengenlehre

Die traditionelle Zermelo-Fraenkel-Mengenlehre mit dem Auswahlaxiom hat 10 Axiome, die in der Sprache der Prädikatenlogik ausgedrückt werden können. In diesem Abschnitt werden wir nur die Namen und kurze deutsche (und englische) Beschreibungen dieser Axiome aufführen, da wir Ihnen später die genauen Formeln der Metamath-Mengenlehre-Datenbasis `set.mm` vorstellen werden.

In den Beschreibungen der Axiome gehen wir davon aus, dass x, y, z, w und v Mengen darstellen. Dies sind die gleichen Variablen wie in unserem obigen System der Prädikatenlogik mit dem Unterschied, dass wir uns die Variablen jetzt informell als Platzhalter für Mengen vorstellen. Beachten Sie, dass die Begriffe „Objekt“, „Menge“, „Element“, „Sammlung“ und „Familie“ synonym sind, ebenso wie „ist ein Element von“, „ist ein Mitglied von“, „ist enthalten in“ und „gehört zu“. Die verschiedenen Begriffe werden der Einfachheit halber verwendet; zum Beispiel ist „eine Sammlung von Mengen“ weniger verwirrend als „eine Menge von Mengen“. Eine Menge x ist eine „Teilmenge“ von y, wenn jedes Element von x auch ein Element von y ist; wir sagen auch, dass x „in y enthalten“ ist.

Die Axiome sind sehr allgemein und gelten für fast alle denkbaren mathematischen Objekte, und diese Abstraktionsebene kann zunächst erdrückend sein. Um ein intuitives Gefühl für das Konzept zu bekommen, kann es hilfreich sein, ein Bild zu zeichnen, welches das Konzept veranschaulicht; ein Kreis mit Punkten könnte beispielsweise eine Sammlung von Mengen darstellen, und ein kleinerer Kreis, der innerhalb des Kreises gezeichnet wird, könnte eine Teilmenge darstellen. Sich überschneidende Kreise können Schnittmenge und Vereinigung veranschaulichen. Kreise, die die Konzepte der Mengenlehre veranschaulichen, werden häufig in Grundschulbüchern verwendet und als Venn-Diagramme bezeichnet.

1. Extensionalitätsaxiom (engl. Axiom of Extensionality): Zwei Mengen sind identisch, wenn sie dieselben Elemente enthalten.

2. Paarmengenaxiom (engl. Axiom of Pairing): Die Menge $\{x, y\}$ existiert.

3. Potenzmengenaxiom (engl. Axiom of Power Sets): Die Potenzmenge

einer Menge (die Sammlung aller ihrer Teilmengen) existiert. Zum Beispiel ist die Potenzmenge von $\{x, y\}$ $\{\varnothing, \{x\}, \{y\}, \{x, y\}\}$, und sie existiert.

4. Leermengenaxiom (engl. Axiom of the Null Set): Die leere Menge $\varnothing$ existiert.

5. Vereinigungsaxiom (engl. Axiom of Union): Die Vereinigung einer Menge (die Menge, welche die Elemente ihrer Mitglieder enthält) existiert. Zum Beispiel ist die Vereinigung von $\{x, y\}, \{z\}$ die existierende Menge $\{x, y, z\}$.

6. Fundierungsaxiom (engl. Axiom of Regularity): Grob gesagt kann keine Menge sich selbst enthalten, noch kann es zyklische Zugehörigkeiten geben, wie z.B. dass eine Menge ein Element eines ihrer Mitglieder ist.

7. Unendlichkeitsaxiom (engl. Axiom of Infinity): Eine unendliche Menge existiert. Ein Beispiel für eine unendliche Menge ist die Menge aller ganzen Zahlen.

8. Aussonderungsaxiom (engl. Axiom of Separation): Es existiert die Menge, die man erhält, wenn man x mit einer bestimmten Eigenschaft einschränkt. Wenn zum Beispiel die Menge aller ganzen Zahlen existiert, dann existiert auch die Menge aller geraden ganzen Zahlen.

9. Ersetzungsaxiom (engl. Axiom of Replacement): Der Wertebereich einer Funktion, deren Definitionsbereich auf die Elemente einer Menge x beschränkt ist, ist ebenfalls eine Menge. Zum Beispiel gibt es eine Funktion von den ganzen Zahlen (der Definitionsbereich der Funktion) zu ihren Quadraten (ihr Wertebereich). Schränkt man den Definitionsbereich auf gerade Zahlen ein, so wird ihr Wertebereich zur Menge der Quadrate der geraden Zahlen, so dass dieses Axiom besagt, dass die Menge der Quadrate der geraden Zahlen existiert. Technische Anmerkung: Im Allgemeinen muss die „Funktion“ keine Menge sein, sondern kann eine echte Klasse sein.

10. Auswahlaxiom: Sei x eine Menge, deren Mitglieder paarweise disjunkte Mengen sind. (d.h. deren Mitglieder keine gemeinsamen Elemente enthalten). Dann gibt es eine andere Menge, die ein Element von jedem Mitglied von x enthält. Wenn x beispielsweise $\{\{y, z\}, \{w, v\}\}$ ist, wobei y, z, w und v verschiedene Mengen sind, dann existiert eine Menge wie $\{z, w\}$ (das Axiom sagt uns aber nicht welche). (Eigentlich ist das Auswahlaxiom überflüssig, wenn die Menge x, wie in diesem Beispiel, eine endliche Anzahl von Elementen hat.)

Das Auswahlaxiom wird in der Regel als Erweiterung der ZF-Mengenlehre betrachtet und nicht als deren originärer Bestandteil. Es wird manchmal als philosophisch umstritten angesehen, weil es die Existenz einer Menge festlegt, ohne sie zu spezifizieren. Konstruktive Logiken, einschließlich der intuitionistischen Logik, akzeptieren das Auswahlaxiom nicht. Da die Kontroverse darüber anhält, bevorzugen wir oft Beweise, die das Auswahlaxiom nicht verwenden (wenn es eine bekannte Alternative gibt), und in einigen Fällen werden wir schwächere Axiome als das vollständige Auswahlaxiom verwenden. Dennoch ist das Auswahlaxiom ein mächtiges und weithin akzeptiertes

Werkzeug, so dass wir es bei Bedarf verwenden. Die ZF-Mengenlehre, die das Auswahlaxiom enthält, wird Zermelo-Fraenkel-Mengenlehre mit Auswahlaxiom (ZFC) genannt.

Symbolisch ausgedrückt enthalten das Aussonderungsaxiom und das Ersetzungsaxiom wff-Symbole und stellen daher jeweils unendlich viele Axiome dar, eines für jede mögliche wff. Aus diesem Grund werden sie oft als Axiomenschemata bezeichnet.

Es stellt sich heraus, dass das Leermengenaxiom, das Paarmengenaxiom und das Aussonderungsaxiom aus den anderen Axiomen abgeleitet werden können und daher unnötig sind, obwohl sie aus verschiedenen Gründen (historisch, philosophisch und möglicherweise, weil einige Autoren dies nicht wissen) in Standardtexten enthalten sind. In der Metamath-Mengenlehre-Datenbasis werden diese überflüssigen Axiome von den anderen abgeleitet, anstatt wirklich als Axiome betrachtet zu werden. Dies entspricht unserem allgemeinen Ziel, die Anzahl der Axiome, von denen wir abhängig sind, zu minimieren.

3.2.4 Andere Axiome

Oben haben wir die Formulierung „die gesamte Mathematik“ mit „im Wesentlichen“ relativiert. Das wichtigste fehlende Element ist die Fähigkeit zur Kategorientheorie, die riesige Mengen (unzugängliche Kardinalzahlen) erfordert, die größer sind als die, die von den ZFC-Axiomen abgeleitet werden können. Das Tarski-Grothendieck-Axiom postuliert die Existenz solcher Mengen. Man beachte, dass dies dasselbe Axiom ist, das von Mizar zur Unterstützung der Kategorientheorie verwendet wird. Das Tarski-Grothendieck-Axiom kann als ein sehr starker Ersatz für das Unendlichkeitsaxiom, das Auswahlaxiom und das Potenzmengenaxiom angesehen werden. Die Datenbasis `set.mm` enthält dieses Axiom; Einzelheiten dazu finden Sie in der Datenbasis. Auch dieses Axiom wird nur verwendet, wenn es absolut notwendig ist. Sie werden diesem Axiom wahrscheinlich nur begegnen oder es verwenden, wenn Sie sich mit der Kategorientheorie beschäftigen, da seine Verwendung hochspezialisiert ist. Daher werden wir das Tarsky-Grothendieck-Axiom in der folgenden kurzen Liste von Axiomen nicht aufführen.

Kann es noch mehr Axiome geben? Ja, natürlich. Gödel hat gezeigt, dass keine endliche Menge von Axiomen oder Axiomenschemata eine konsistente Theorie, die stark genug ist, um die Arithmetik einzuschließen, vollständig beschreiben kann. Aber praktisch gesehen sind die oben genannten Axiome die anerkannte Grundlage, auf der fast alle Mathematiker explizit oder implizit ihre Arbeit aufbauen.

3.3 Die Axiome in der Metamath-Sprache

Hier führen wir die Axiome so auf, wie sie in der Datenbasis der Mengenlehre `set.mm` erscheinen, damit Sie diese dort leicht nachschlagen können. Übrigens wurde der Befehl `show statement /tex` verwendet, um sie darzustellen.

3.3.1 Aussagenlogik

Axiom der Vereinfachung.
`ax-1 $a` $\vdash (\varphi \to (\psi \to \varphi))$
Axiom der Verteilung.
`ax-2 $a` $\vdash ((\varphi \to (\psi \to \chi)) \to ((\varphi \to \psi) \to (\varphi \to \chi)))$
Axiom der Kontraposition.
`ax-3 $a` $\vdash ((\neg \varphi \to \neg \psi) \to (\psi \to \varphi))$
Die Schlussregel Modus ponens.
`min $e` $\vdash \varphi$
`maj $e` $\vdash (\varphi \to \psi)$
`ax-mp $a` $\vdash \psi$

3.3.2 Axiome der Prädikatenlogik mit Gleichheit — Tarskis S2

Regel der Verallgemeinerung.
`ax-g.1 $e` $\vdash \varphi$
`ax-gen $a` $\vdash \forall x \varphi$
Axiom der quantifizierten Implikation.
`ax-4 $a` $\vdash (\forall x (\forall x \varphi \to \psi) \to (\forall x \varphi \to \forall x \psi))$
Axiom der Unterscheidbarkeit.
`ax-5 $a` $\vdash (\varphi \to \forall x \varphi)$ *mit* `$d` x φ (x *kommt in* φ *nicht vor*)
Axiom der Existenz.
`ax-6 $a` $\vdash (\forall x (x = y \to \forall x \varphi) \to \varphi)$
Axiom der Gleichheit.
`ax-7 $a` $\vdash (x = y \to (x = z \to y = z))$
Axiom der Linksgleichheit für binäre Prädikate.
`ax-8 $a` $\vdash (x = y \to (x \in z \to y \in z))$
Axiom der Rechtsgleichheit für binäre Prädikate.
`ax-9 $a` $\vdash (x = y \to (z \in x \to z \in y))$

3.3.3 Axiome der Prädikatenlogik mit Gleichheit — Hilfsaxiome

Axiom der quantifizierten Negation.
`ax-10 $a` $\vdash (\neg \forall x \neg \forall x \varphi \to \varphi)$

Axiom der Quantifizierungskommutativität.
`ax-11 $a` $\vdash (\forall x \forall y \varphi \rightarrow \forall y \forall x \varphi)$
Axiom der Substitution.
`ax-12 $a` $\vdash (\neg \forall x \; x = y \rightarrow (x = y \rightarrow (\varphi \rightarrow \forall x (x = y \rightarrow \varphi))))$
Axiom der quantifizierten Gleichheit.
`ax-13 $a` $\vdash (\neg \forall z \; z = x \rightarrow (\neg \forall z \; z = y \rightarrow (x = y \rightarrow \forall z \; x = y)))$

3.3.4 Mengenlehre

Um die Axiome der Mengenlehre etwas kompakter zu gestalten, gibt es einige Definitionen aus der Logik, die wir implizit verwenden, nämlich „logisches UND“, „logische Äquivalenz“, und „Es gibt“.

$$\begin{array}{rll} (\varphi \wedge \psi) & \text{steht für} & \neg(\varphi \rightarrow \neg\psi) \\ (\varphi \leftrightarrow \psi) & \text{steht für} & ((\varphi \rightarrow \psi) \wedge (\psi \rightarrow \varphi)) \\ \exists x \, \varphi & \text{steht für} & \neg\forall x \neg\varphi \end{array}$$

Darüber hinaus verlangen die Axiome der Mengenlehre, dass alle Variablen unterscheidbar sind,[4] also nehmen wir auch an:

$$\texttt{\$d} \; x \, y \, z \, w$$

Extensionalitätsaxiom.
`ax-ext $a` $\vdash (\forall x (x \in y \leftrightarrow x \in z) \rightarrow y = z)$
Ersetzungsaxiom.
`ax-rep $a` $\vdash (\forall w \exists y \forall z (\forall y \varphi \rightarrow z = y) \rightarrow \exists y \forall z (z \in y \leftrightarrow \exists w (w \in x \wedge \forall y \varphi)))$
Vereinigungsaxiom.
`ax-un $a` $\vdash \exists x \forall y (\exists x (y \in x \wedge x \in z) \rightarrow y \in x)$
Potenzmengenaxiom.
`ax-pow $a` $\vdash \exists x \forall y (\forall x (x \in y \rightarrow x \in z) \rightarrow y \in x)$
Fundierungsaxiom.
`ax-reg $a` $\vdash (\exists x \; x \in y \rightarrow \exists x (x \in y \wedge \forall z (z \in x \rightarrow \neg z \in y)))$
Unendlichkeitsaxiom.
`ax-inf $a` $\vdash \exists x (y \in x \wedge \forall y (y \in x \rightarrow \exists z (y \in z \wedge z \in x)))$
Auswahlaxiom.
`ax-ac $a` $\vdash \exists x \forall y \forall z ((y \in z \wedge z \in w) \rightarrow \exists w \forall y (\exists w ((y \in z \wedge z \in w) \wedge (y \in w \wedge w \in x)) \leftrightarrow y = w))$

[4] Die Axiome der Mengenlehre können so entwickelt werden, dass *keine* Variablen unterscheidbar sein müssen, vorausgesetzt, wir ersetzen `ax-c16` durch ein Axiom, das besagt, dass „mindestens zwei Dinge existieren“, wodurch `ax-5` das einzige andere Axiom wird, das die Anweisung `$d` erfordert. Diese Axiome sind unkonventionell und werden hier nicht vorgestellt, aber sie können auf der Website `http://metamath.org` gefunden werden. Siehe auch den Kommentar zu S. 140.

3.3.5 Das war's

Das waren sie, die Axiome für (im Wesentlichen) die gesamte Mathematik! Bestaunen Sie sie und schauen Sie sie ehrfürchtig an. Stecken Sie ein Exemplar in Ihre Brieftasche und Sie werden die Kodierung aller Theoreme, die jemals bewiesen wurden und die jemals bewiesen werden, in Ihrer Tasche tragen - von den banalsten bis zu den tiefgründigsten.

3.4 Eine Hierarchie von Definitionen

Die Axiome im vorigen Abschnitt ermöglichen im Prinzip alles, was man in der Standardmathematik erreichen kann. Allerdings ist es in den meisten Fällen unpraktisch, mit ihnen direkt zu arbeiten, da selbst einfache Konzepte (aus menschlicher Sicht) nur mit extrem langen, unverständlichen Formeln ausgedrückt werden können. Die Mathematik wird deshalb erst durch die Einführung von Definitionen praktikabel. Definitionen führen in der Regel neue Symbole oder zumindest neue Beziehungen zwischen bestehenden Symbolen ein, um komplexere Formeln abzukürzen. Eine wichtige Voraussetzung für eine Definition ist, dass es eine einfache (algorithmische) Methode gibt, um die Abkürzung zu eliminieren, indem sie durch die primitivere Symbolkette, die sie repräsentiert, ersetzt wird. Einige wichtige Definitionen, die in der Datei `set.mm` enthalten sind, werden in diesem Abschnitt als Referenz aufgeführt, und auch, um Ihnen ein Gefühl dafür zu geben, warum etwas wie ω (die Menge der natürlichen Zahlen 0, 1, 2,...) sehr kompliziert wird, wenn es vollständig in primitiven Symbolen ausgedrückt wird.

Was ist die Motivation für Definitionen, abgesehen davon, dass komplizierte Ausdrücke einfacher ausgedrückt werden können? Im Falle von ω besteht ein Ziel darin, eine Grundlage für die Theorie der natürlichen Zahlen zu schaffen. Vor der Erfindung der Mengenlehre wurde eine Reihe von Axiomen für die Arithmetik, die so genannten Peano-Postulate indexPeanos Postulate, entwickelt und gezeigt, dass sie die erwarteten Eigenschaften für natürliche Zahlen haben. Nun kann jeder einen Satz von Axiomen postulieren, aber wenn die Axiome inkonsistent sind, können daraus Widersprüche abgeleitet werden. Sobald ein Widerspruch abgeleitet ist, kann alles trivialerweise bewiesen werden, einschließlich aller Fakten der Arithmetik und ihrer Negationen. Um sicherzustellen, dass ein Axiomensystem mindestens so zuverlässig ist wie die Axiome der Mengenlehre, können wir Mengen und Operationen auf diesen Mengen definieren, die die neuen Axiome erfüllen. In der `set.mm` Metamath-Datenbasis beweisen wir, dass die Elemente von ω die Peano-Postulate erfüllen, und es ist ein langer und harter Weg, um von den Axiomen der Mengenlehre direkt dorthin zu gelangen. Aber das Ergebnis ist das Vertrauen in die Grundlagen der Arithmetik. Und es gibt noch einen weiteren Vorteil: Wir haben jetzt alle Werkzeuge der Mengenlehre zur Verfügung, um Objekte zu manipulieren, die den Axiomen der Arithmetik

gehorchen.

Was sind die Kriterien für unsere Definitionen? Erstens, und das ist von größter Bedeutung, sollte die Definition nicht *kreativ* sein, d.h. sie sollte nicht zulassen, dass ein Ausdruck, der als wff klassifiziert, aber nicht beweisbar war, beweisbar wird. Zweitens sollte die Definition *eliminierbar* sein, d.h. es sollte eine algorithmische Methode geben, um jeden als wff klasifizierten Ausdruck, der die Definition verwendet, in einen logisch äquivalenten Ausdruck umzuwandeln.

In fast allen folgenden Fällen verbinden die Definitionen zwei Ausdrücke entweder mit $\leftrightarrow$ oder $=$. Die Eliminierung[5] einer solchen Definition ist ein einfaches Ersetzen des Ausdrucks auf der linken Seite (*Definiendum* oder Sache, die definiert wird) durch den äquivalenten, primitiveren Ausdruck auf der rechten Seite (*Definiens* oder Definition).

Häufig enthält eine Definition auf der rechten Seite Variablen, die auf der linken Seite nicht vorkommen; diese werden als *Dummy-Variable* bezeichnet. In diesem Fall kann jede zulässige Substitution (z. B. eine neue, unterschiedliche Variable) verwendet werden, wenn die Definition eliminiert wird. Dummy-Variablen dürfen nur verwendet werden, wenn sie *effektiv gebunden* sind, was bedeutet, dass die Definition bei jeder Ersetzung einer Dummy-Variablen durch einen anderen *qualifizierenden Ausdruck* logisch äquivalent bleibt, d.h. eine beliebige Zeichenkette (z.B. eine andere Variable), die den Beschränkungen für die Dummy-Variable durch die Anweisungen `$d` und `$f` entspricht. Wir könnten zum Beispiel eine Konstante $\bot$ (invertiertes T, d.h. logisch „falsch“) als $(\varphi \wedge \neg\varphi)$ definieren, d.h. „phi und nicht phi“. Hier ist φ effektiv gebunden, weil die Definition logisch äquivalent bleibt, wenn wir φ durch irgendeine andere wff ersetzen. (Eigentlich wird $\bot$ in `set.mm` durch `df-fal` definiert.)

Es gibt zwei Fälle, in denen die Eliminierung von Definitionen ein wenig komplexer ist. Diese Fälle sind die Definitionen `df-bi` und `df-cleq`. Der erste Fall fasst das Konzept einer Definition ein wenig weiter, da er in der Tat eine „Definition definiert“; diese Definition erfüllt jedoch unsere Anforderungen an eine Definition, da sie eliminierbar ist und die Mächtigkeit der Sprache nicht erhöht. Theorem `bii` zeigt die notwendige Substitution, um das Symbol $\leftrightarrow$ zu eliminieren.

Definition `df-cleq` erweitert die Verwendung des Gleichheitssymbols auf „Klassen“ in der Mengenlehre. Dies könnte potenziell problematisch sein, weil es zu Aussagen führen kann, die nicht allein aus der Logik folgen, sondern das Extensionalitätsaxiom voraussetzen. Deshalb nehmen wir dieses Axiom als Hypothese in die Definition mit auf. Wir hätten `df-cleq` direkt eliminierbar machen können, indem wir ein neues Gleichheitssymbol eingeführt hätten. Wir haben uns aber entschieden, dies nicht zu tun, um

[5] Hier ist die Eliminierung gemeint, die ein Mensch in seinem Kopf durchführen könnte. Um sie als Teil eines Metamath-Beweises zu eliminieren, würden wir uns auf eines der Theoreme berufen, die sich mit der Transitivität von Äquivalenz oder Gleichheit befassen; es gibt viele solcher Beispiele in den Beweisen in `set.mm`.

der üblichen Lehrbuchpraxis zu entsprechen. Definitionen wie `df-cleq`, die die Bedeutung bestehender Symbole erweitern, müssen sorgfältig eingeführt werden, damit sie nicht zu Widersprüchen führen. Die Definition `df-clel` erweitert ebenfalls die Bedeutung eines bestehenden Symbols ($\in$); sie erhöht zwar nicht die Mächtigkeit der Sprache so wie `df-cleq`, aber das ist nicht offensichtlich, weshalb sie ebenfalls einer sorgfältigen Prüfung unterzogen werden muss.

Übung: Untersuchen Sie, wie die wff $x \in \omega$, die besagt, dass „x eine natürliche Zahl ist“, in Form von primitiven Symbolen ausgedrückt werden könnte, indem Sie mit den Definitionen `df-clel` auf S. 89 und `df-om` auf S. 93 beginnen und sich dann rückwärts vorarbeiten. Machen Sie sich nicht die Mühe, die Details auszuarbeiten; stellen Sie nur sicher, dass Sie verstehen, wie Sie es im Prinzip tun könnten. Die Antwort finden Sie in der Fußnote auf S. 69. Wenn Sie dies tatsächlich durchführen, werden Sie nicht genau die gleiche Antwort erhalten, weil wir einige Vereinfachungen verwendet haben, wie z.B. das Weglassen von $\neg\neg$ (doppelte Negation).

In den untenstehenden Definitionen haben wir die ASCII Metamath-Quelle unter jede der Formeln gesetzt, damit Sie sich mit der Notation in der Datenbasis vertraut machen können. Der Einfachheit halber werden die notwendigen `$f`- und `$d`-Anweisungen nicht gezeigt. Im Zweifelsfall sollten Sie den Befehl `show statement` im Metamath-Programm verwenden, um die vollständige Aussage zu sehen. Eine Auswahl dieser Notation ist im Anhang A zusammengefasst.

Um die Motivation für diese Definitionen zu verstehen, sollten Sie die angegebenen Referenzen konsultieren: Takeuti und Zaring [67], Quine [55], Bell und Machover [5], und Enderton [18]. Unsere Liste der Definitionen dient eher als Referenz denn als Lernhilfe. Anhand einiger Definitionen können Sie jedoch ein Gefühl dafür bekommen, wie die Hierarchie aufgebaut ist. Die Definitionen sind eine repräsentative Auswahl der vielen Definitionen in `set.mm`, aber sie sind vollständig in Bezug auf die Beispieltheoreme, die wir in Abschnitt 3.6 vorstellen werden. Außerdem unterscheiden sich einige Definitionen geringfügig von denen in `set.mm`, sind aber logisch äquivalent zu denen in `set.mm` (von denen einige im Laufe der Zeit überarbeitet wurden, um sie z. B. zu kürzen).

3.4.1 Definitionen für die Aussagenlogik

Die Symbole φ, ψ und χ stehen für wffs.

Unsere erste Definition führt den bikonditionalen Junktor ein[6] (auch

[6]Der Begriff „Junktor“ wird informell verwendet, um ein Symbol zu bezeichnen, das zwischen zwei Variablen oder neben einer Variable platziert ist, während eine mathematische „Konstante“ normalerweise ein Symbol wie die Zahl 0 bezeichnet, das eine Variable oder Metavariable ersetzen kann. Aus der Sicht von Metamath gibt es keine Unterscheidung zwischen einem Junktor und einer Konstante; beide sind in der Metamath-Sprache Konstanten.

logische Äquivalenz genannt). Im Gegensatz zu den meisten traditionellen Vorgehensweisen haben wir uns entschieden, kein separates Symbol wie „Df." für „ist definiert als" zu verwenden. Stattdessen werden wir den bikonditionalen Junktor für diesen Zweck verwenden, da er uns erlaubt, Definitionen direkt mithilfe der Logik zu manipulieren. Hier geben wir die Eigenschaften des bikonditionalen Junktors mit einer sorgfältig formulierten `$a`-Anweisung an, die den bikonditionalen Junktor probat dazu verwendet, sich selbst zu definieren. Das Symbol $\leftrightarrow$ kann mit Hilfe des Theorems `bii`, das später hergeleitet wird, aus einer Formel entfernt werden.

Definition des **bikonditionalen Junktors**.

`df-bi $a` $\vdash \neg (((\varphi \leftrightarrow \psi) \rightarrow \neg ((\varphi \rightarrow \psi) \rightarrow \neg (\psi \rightarrow \varphi))) \rightarrow \neg (\neg ((\varphi \rightarrow \psi) \rightarrow \neg (\psi \rightarrow \varphi)) \rightarrow (\varphi \leftrightarrow \psi)))$

```
df-bi $a |- -. ( ( ( ph <-> ps ) -> -. ( ( ph -> ps ) -> -.
         ( ps -> ph ) ) ) -> -. ( -. ( ( ph -> ps ) -> -. (
         ps -> ph ) ) -> ( ph <-> ps ) ) ) $.
```

Das folgende Theorem stellt eine Beziehung zwischen dem bikonditionalen Junktor und den primitiven Junktoren her und kann verwendet werden, um das $\leftrightarrow$-Symbol aus jeder wff zu eliminieren.

`bii $p` $\vdash ((\varphi \leftrightarrow \psi) \leftrightarrow \neg ((\varphi \rightarrow \psi) \rightarrow \neg (\psi \rightarrow \varphi)))$

```
bii $p |- ( ( ph <-> ps ) <-> -. ( ( ph -> ps ) -> -. ( ps
       -> ph ) ) ) $= ... $.
```

Definition der **Disjunktion** (ODER).

`df-or $a` $\vdash ((\varphi \vee \psi) \leftrightarrow (\neg \varphi \rightarrow \psi))$

```
df-or $a |- ( ( ph \/ ps ) <-> ( -. ph -> ps ) ) $.
```

Definition der **Konjunktion** (UND).

`df-an $a` $\vdash ((\varphi \wedge \psi) \leftrightarrow \neg (\varphi \rightarrow \neg \psi))$

```
df-an $a |- ( ( ph /\ ps ) <-> -. ( ph -> -. ps ) ) $.
```

Definition der **Disjunktion (ODER) von 3 wffs**.

`df-3or $a` $\vdash ((\varphi \vee \psi \vee \chi) \leftrightarrow ((\varphi \vee \psi) \vee \chi))$

```
df-3or $a |- ( ( ph \/ ps \/ ch ) <-> ( ( ph \/ ps ) \/ ch )
          ) $.
```

Definition der **Konjunktion (UND) von 3 wffs**.

`df-3an $a` $\vdash ((\varphi \wedge \psi \wedge \chi) \leftrightarrow ((\varphi \wedge \psi) \wedge \chi))$

```
df-3an $a |- ( ( ph /\ ps /\ ch ) <-> ( ( ph /\ ps ) /\ ch )
          ) $.
```

3.4.2 Definitionen für die Prädikatenlogik

Die Symbole x, y und z stehen für individuelle Variablen der Prädikatenlogik. In diesem Abschnitt sind sie nicht notwendigerweise verschieden, es sei denn, es wird ausdrücklich erwähnt.

Definition der **existentiellen Quantifizierung**.

Der Ausdruck $\exists x \varphi$ bedeutet „Es existiert ein x, bei dem φ wahr ist."

```
df-ex $a ⊢ ( ∃ x φ ↔ ¬ ∀ x ¬ φ )
```

```
df-ex $a |- ( E. x ph <-> -. A. x -. ph ) $.
```

Definition der **echten Substitution**.

In unserer Notation verwenden wir $[y/x]\varphi$, um „die wff zu bezeichnen, die sich ergibt, wenn y in der wff φ echt durch x ersetzt wird".[7] Zum Beispiel ist $[y/x]z \in x$ das gleiche wie $z \in y$. Man kann sich diese Notation leicht merken, wenn man sie mit einer Division vergleicht, bei der $(y/x) \cdot x\ y$ ist (wenn $x \neq 0$). Die Notation unterscheidet sich von der Notation $\varphi(x|y)$, die manchmal verwendet wird, weil letztere Notation für uns mehrdeutig ist: Wir wissen zum Beispiel nicht, ob $\neg\varphi(x|y)$ als $\neg(\varphi(x|y)$ oder $(\neg\varphi)(x|y)$ zu interpretieren ist.[8] In anderen Texten wird oft $\varphi(y)$ verwendet, um unser $[y/x]\varphi$ zu bezeichnen, aber diese Schreibweise ist noch mehrdeutiger, da es keinen ausdrücklichen Hinweis darauf gibt, was ersetzt wird. Man beachte, dass unsere Definition auch dann gültig ist, wenn x und y die gleiche Variable repräsentieren. Die erste Konjunktion in der folgenden formalen Definition ist ein „Trick", um diese Eigenschaft zu erreichen, was die Definition auf den ersten Blick etwas merkwürdig erscheinen lässt.

```
df-sb $a ⊢ ( [ y / x ] φ ↔ ( ( x = y → φ ) ∧ ∃ x ( x = y ∧ φ ) ) )
```

```
df-sb $a |- ( [ y / x ] ph <-> ( ( x = y -> ph ) /\ E. x ( x
        = y /\ ph ) ) ) $.
```

Definition der **existentiellen Eindeutigkeit** („Es existiert genau einer").

Man beachte, dass y eine Variable ist, die sich von x unterscheidet und nicht in φ vorkommt.

---

[7] Dies kann auch so beschrieben werden, dass x durch y ersetzt wird, y x echt ersetzt, oder x echt durch y ersetzt wird.

[8] Aufgrund der Art und Weise, wie wir wffs ursprünglich definiert haben, ist dies der Fall bei jedem Postfix-Konnektor (einer, der nach den zu verbindenden Symbolen auftritt) oder einem Infix-Konnektor (einer, der zwischen den zu verbindenden Symbolen vorkommt). Metamath hat keine eingebaute Regel für die Vorrangigkeit einer Operatorausführung, die die Mehrdeutigkeit beseitigen könnte. Die öffnende Klammer stellt einen effektiven Präfix-Konnektor dar, um die Mehrdeutigkeit zu beseitigen. Einige Konventionen, wie z. B. die polnische Notation, die in den 1930er und 1940er Jahren von polnischen Logikern verwendet wurde, verwenden nur Präfix-Konnektoren und ermöglichen so den vollständigen Verzicht auf Klammern, was allerdings auf Kosten der Lesbarkeit geht. In Metamath könnten wir, wenn wir wollten, die gesamte Notation auf die polnische Notation umstellen, ohne irgendwelche Beweise ändern zu müssen!

`df-eu $a` $\vdash (\exists! x\, \varphi \leftrightarrow \exists y \forall x\, (\varphi \leftrightarrow x = y))$

```
df-eu $a |- ( E! x ph <-> E. y A. x ( ph <-> x = y ) ) $.
```

3.4.3 Definitionen für die Mengenlehre

Die Symbole x, y, z und w stellen individuelle Variablen der Prädikatenlogik dar, die in der Mengenlehre als Mengen verstanden werden. Allerdings wäre es sehr unpraktisch, nur die bisher gezeigten Konstrukte zu verwenden.

Um die Mengenlehre praktikabler zu machen, führen wir den Begriff der „Klasse“ ein. Eine Klasse ist entweder eine Mengenvariable (wie x) oder ein Ausdruck der Form $\{x|\varphi\}$ (genannt eine „Abstraktionsklasse“). Man beachte, dass Mengen (d.h. individuelle Variablen) immer existieren (dies ist ein Satz der Logik, nämlich $\exists y\, y = x$ für jede beliebige Menge x), während Klassen existieren können oder nicht (d.h. $\exists y\, y = A$ kann wahr sein oder nicht). Wenn eine Klasse nicht existiert, wird sie als „echte Klasse“ bezeichnet. Die Definitionen `df-clab`, `df-cleq` und `df-clel` können verwendet werden, um einen Klassen enthaltenden Ausdruck in einen Ausdruck umzuwandeln, der nur Mengenvariablen und wff-Metavariablen enthält.

Die Symbole A, B, C, D, F, G und R sind Metavariablen, die sich über Klassen erstrecken. Eine Klassenmetavariable A kann aus einer wff eliminiert werden, indem sie durch $\{x|\varphi\}$ ersetzt wird, wobei weder x noch φ in der wff vorkommen.

Die Klassentheorie erweist sich als eliminierbare und konservative Erweiterung der Mengenlehre. Die Eigenschaft der **Eliminierbarkeit** bedeutet, dass wir für jede Formel in der erweiterten Sprache eine logisch äquivalente Formel in der Basissprache bilden können. Auch wenn die erweiterte Sprache die Vermittlung und Formulierung mathematischer Ideen für die Mengenlehre erleichtert, stärkt ihre Ausdruckskraft nicht die Ausdruckskraft der Basissprache. Die Eigenschaft der **Konservativität** bedeutet, dass wir für jeden Beweis einer Formel der Basissprache, der im erweiterten System geführt wird, einen anderen Beweis derselben Formel, der ausschließlich im Basissystem geführt wird, konstruieren können; so dass die deduktiven Möglichkeiten des erweiterten Systems und des Basissystems identisch sind, wenn es nur um Theoreme über Mengen geht. Zusammen bedeuten diese Eigenschaften, dass die erweiterte Sprache als eine definitorische Erweiterung behandelt werden kann, die **gesund** ist.

Eine strenge Begründung, die wir hier nicht geben werden, findet sich bei Levy [38, pp. 357-366], der seine informelle Einführung in die Klassentheorie auf S. 7-17 ergänzt. Zwei weitere gute Abhandlungen der Klassentheorie finden sich bei Quine [55, pp. 15-21] und auch bei [67, pp. 10-14]. Quines Ausführungen (er nennt sie virtuelle Klassen) sind elegant geschrieben und sehr lesenswert.

Im weiteren Verlauf dieses Abschnitts wird immer davon ausgegangen, dass die einzelnen Variablen voneinander verschieden sind, sofern nicht anders angegeben. Darüber hinaus kommen Dummy-Variablen auf der rechten Seite einer Definition nicht in den zu definierenden Metavariablen für Klassen und wffs vor.

Die hier vorgestellten Definitionen sind eine unvollständige, aber in sich geschlossene Auswahl aus mehreren hundert Definitionen, die in der aktuellen Datenbasis `set.mm` enthalten sind. Sie sind ausreichend für eine grundlegende Herleitung der elementaren Mengenlehre.

Definition einer **Abstraktionsklasse**. x und y müssen nicht verschieden sein. Definition 2.1 von Quine, S. 16. Diese Definition mag rätselhaft erscheinen, da sie kürzer ist als der zu definierende Ausdruck und uns in Bezug auf die Kürze keinen Vorteil bringt. Warum wir diese Definition einführen, ist dadurch begründet, dass sie gut zu der von `df-clel` bereitgestellten Erweiterung des $\in$-Symbols passt.

`df-clab $a` $\vdash (x \in \{ y \mid \varphi \} \leftrightarrow [x / y] \varphi)$

```
df-clab $a |- ( x e. { y | ph } <-> [ x / y ] ph ) $.
```

Definition des **Gleichheitszeichen zwischen Klassen**. Siehe Quine oder Kapitel 4 von Takeuti und Zaring für die Rechtfertigung und die Methoden zu ihrer Eliminierung. Dies ist ein Beispiel für eine etwas „gefährliche“ Definition, denn sie erweitert die Verwendung des bestehenden Gleichheitssymbols, anstatt ein neues Symbol einzuführen, und erlaubt uns, in der ursprünglichen Sprache Aussagen zu machen, die möglicherweise nicht wahr sind. Zum Beispiel erlaubt sie uns, $y = z \leftrightarrow \forall x(x \in y \leftrightarrow x \in z)$ abzuleiten, was kein Satz der Logik ist, sondern das Extensionalitätsaxiom voraussetzt, das wir als Hypothese einfügen, damit wir wissen, wann dieses Axiom in einem Beweis vorausgesetzt wird (mit dem Befehl `show trace_back`). Wir könnten die Gefahr vermeiden, indem wir ein anderes Symbol, sagen wir $=\!\!\!=$, anstelle von $=$ einführen; dies hätte auch den Vorteil, dass die Definition einfach zu eliminieren wäre und die Notwendigkeit der Extensionalität als Hypothese entfiele. Wir hätten dann auch den Vorteil, dass wir genau feststellen könnten, wo die Extensionalität wirklich ins Spiel kommt. Eines unserer Theoreme wäre $x =\!\!\!= y \leftrightarrow x = y$, indem wir uns auf die Extensionalität berufen. In Übereinstimmung mit der üblichen Praxis behalten wir jedoch die „gefährliche“ Definition bei.

`df-cleq.1 $e` $\vdash (\forall x (x \in y \leftrightarrow x \in z) \rightarrow y = z)$
`df-cleq $a` $\vdash (A = B \leftrightarrow \forall x (x \in A \leftrightarrow x \in B))$

```
df-cleq.1 $e |- ( A. x ( x e. y <-> x e. z ) -> y = z ) $.
```

```
df-cleq $a |- ( A = B <-> A. x ( x e. A <-> x e. B ) ) $.
```

Definition des **Elementprädikates zwischen Klassen**. Theorem 6.3 von Quine, S. 41, das wir als Definition übernehmen. Man beachte, dass er die Verwendung des bestehenden Zugehörigkeitssymbols erweitert, aber im Gegensatz zu `df-cleq` nicht die Menge der gültigen wffs der Logik erweitert, wenn die Klassenmetavariablen durch Mengenvariablen ersetzt werden.

`df-clel $a` $\vdash (A \in B \leftrightarrow \exists x (x = A \wedge x \in B))$

```
df-clel $a |- ( A e. B <-> E. x ( x = A /\ x e. B ) ) $.?
```

Definition der **Ungleichheit**.

`df-ne $a` $\vdash (A \neq B \leftrightarrow \neg A = B)$

```
df-ne $a |- ( A =/= B <-> -. A = B ) $.
```

Definition der **eingeschränkten Allquantifizierung**. Enderton, S. 22.

`df-ral $a` $\vdash (\forall x \in A \, \varphi \leftrightarrow \forall x (x \in A \rightarrow \varphi))$

```
df-ral $a |- ( A. x e. A ph <-> A. x ( x e. A -> ph ) ) $.
```

Definition der **eingeschränkten Existenzquantifizierung**. Enderton, S. 22.

`df-rex $a` $\vdash (\exists x \in A \, \varphi \leftrightarrow \exists x (x \in A \wedge \varphi))$

```
df-rex $a |- ( E. x e. A ph <-> E. x ( x e. A /\ ph ) ) $.
```

Definition der **universellen Klasse**. Definition 5.20, S. 21, von Takeuti und Zaring.

`df-v $a` $\vdash \mathrm{V} = \{ x \mid x = x \}$

```
df-v $a |- _V = { x | x = x } $.
```

Definition der **Unterklassen-Beziehung zwischen zwei Klassen** (die so genannte Untermengen-Beziehung, wenn die Klassen Mengen sind, d. h. keine echten Klassen). Definition 5.9 von Takeuti und Zaring, S. 17.

`df-ss $a` $\vdash (A \subseteq B \leftrightarrow \forall x (x \in A \rightarrow x \in B))$

```
df-ss $a |- ( A C_ B <-> A. x ( x e. A -> x e. B ) ) $.
```

Definition der **Vereinigung von zwei Klassen**. Definition 5.6 von Takeuti und Zaring, S. 16.

`df-un $a` $\vdash (A \cup B) = \{ x \mid (x \in A \vee x \in B) \}$

```
df-un $a ( A u. B ) = { x | ( x e. A \/ x e. B ) } $.
```

Definition des **Schnitts zwischen zwei Klassen**. Definition 5.6 von Takeuti und Zaring, S. 16.

```
df-in $a |- ( A i^i B ) = { x | ( x e. A /\ x e. B ) } $.
```

Definition der **Klassendifferenz**. Definition 5.12 von Takeuti und Zaring, S. 20. In der Literatur werden verschiedene Schreibweisen verwendet; wir haben die Konvention \ anstelle eines Minuszeichens gewählt, um letzteres für die spätere Verwendung z.B. in der Arithmetik zu reservieren.

```
df-dif $a |- ( A \ B ) = { x | ( x e. A /\ -. x e. B ) } $.
```

Definition der **leeren Menge**. Vergleiche Definition 5.14 von Takeuti und Zaring, S. 20.

```
df-nul $a |- (/) = ( _V \ _V ) $.
```

Definition der **Potenzklasse**. Definition 5.10 von Takeuti und Zaring, S. 17, aber wir lassen sie auch für echte Klassen gelten. (Beachten Sie, dass `~P` das Symbol für das kalligraphische P ist, wobei die Tilde für „lockig“ steht; siehe Anhang A.)

```
df-pw $a |- ~P A = { x | x C_ A } $.
```

Definition einer **einelementigen Klasse (Singleton)**. Definition 7.1 von Quine, S. 48. Sie ist auch für echte Klassen wohldefiniert, obwohl sie in diesem Fall nicht sehr aussagekräftig ist, da sie zur leeren Menge ausgewertet wird.

```
df-sn $a |- { A } = { x | x = A } $.
```

Definition eines **ungeordneten Klassenpaares**. Definition 7.1 von Quine, S. 48.

```
df-pr $a |- { A , B } = ( { A } u. { B } ) $.
```

Definition eines **ungeordneten Klassentripels**. Definition von Enderton, S. 19.

```
df-tp $a |- { A , B , C } = ( { A , B } u. { C } )
```

```
df-tp $a |- { A , B , C } = ( { A , B } u. { C } ) $.
```

Definition von Kuratowskis **geordneten Klassenpaares**-Definition. Definition 9.1 von Quine, S. 58. Für echte Klassen ist sie nicht sinnvoll, aber der Einfachheit halber wohldefiniert. (Man beachte, dass `<.` für $\langle$ steht, während `<` für $<$ steht.)

`df-op $a` $\vdash \langle A , B \rangle = \{ \{ A \} , \{ A , B \} \}$

```
df-op $a |- <. A , B >. = { { A } , { A , B } } $.
```

Definition der **Vereinigung einer Klasse**. Definition 5.5, S. 16, von Takeuti und Zaring.

`df-uni $a` $\vdash \bigcup A = \{ x \mid \exists y (x \in y \wedge y \in A) \}$

```
df-uni $a |- U. A = { x | E. y ( x e. y /\ y e. A ) } $.
```

Definition des **Schnittes einer Klasse**. Definition 7.35, S. 44, von Takeuti und Zaring.

`df-int $a` $\vdash \bigcap A = \{ x \mid \forall y (y \in A \rightarrow x \in y) \}$

```
df-int $a |- |^| A = { x | A. y ( y e. A -> x e. y ) } $.
```

Definition einer **transitiven Klasse**. Dies sollte nicht mit einer transitiven Beziehung verwechselt werden, die ein anderes Konzept ist. Definition aus S. 71 von Enderton, erweitert auf Klassen.

`df-tr $a` $\vdash (\mathrm{Tr}\, A \leftrightarrow \bigcup A \subseteq A)$

```
df-tr $a |- ( Tr A <-> U. A C_ A ) $.
```

Definition einer Notation für eine **allgemeine binäre Relation**. Definition 6.18, S. 29, von Takeuti und Zaring, verallgemeinert auf beliebige Klassen. Diese Definition ist wohldefiniert, wenn auch nicht sehr aussagekräftig, wenn die Klassen A und/oder B echte Klassen sind. Das Fehlen von Klammern (oder eines anderen Konnektors) erzeugt keine Mehrdeutigkeit, da wir eine atomare wff definieren.

`df-br $a` $\vdash (A \; R \; B \leftrightarrow \langle A , B \rangle \in R)$

```
df-br $a |- ( A R B <-> <. A , B >. e. R ) $.
```

Definition einer **Abstraktionsklasse von geordneten Paaren** . Ein Spezialfall der Definition 4.16, S. 14, von Takeuti und Zaring. Man beachte, dass z von x und y verschieden sein muss und z nicht in φ vorkommen darf, aber x und y können identisch sein und in φ vorkommen.

`df-opab $a` $\vdash \{ \langle x , y \rangle \mid \varphi \} = \{ z \mid \exists x \exists y (z = \langle x , y \rangle \wedge \varphi) \}$

```
df-opab $a |- { <. x , y >. | ph } = { z | E. x E. y ( z =
          <. x , y >. /\ ph ) } $.
```

Definition der **Epsilon-Relation**. Ähnlich der Definition 6.22, S. 30, von Takeuti und Zaring.

`df-eprel $a` $\vdash \mathrm{E} = \{ \langle x , y \rangle \mid x \in y \}$

```
df-eprel $a |- _E = { <. x , y >. | x e. y } $.
```

Definition einer fundierten Relation. R ist eine fundierte Relation auf A, genau dann, wenn (wenn und nur dann, wenn) jede nichtleere Teilmenge von A ein „R-minimales Element“ hat. Ähnlich der Definition 6.21, S. 30, von Takeuti und Zaring.

`df-fr $a` $\vdash (R \mathrm{Fr} A \leftrightarrow \forall x ((x \subseteq A \wedge \neg x = \varnothing) \rightarrow \exists y (y \in x \wedge (x \cap \{ z \mid z \; R \; y \}) = \varnothing)))$

```
df-fr $a |- ( R Fr A <-> A. x ( ( x C_ A /\ -. x = (/) ) ->
         E. y ( y e. x /\ ( x i^i { z | z R y } ) = (/) ) )
         ) $.
```

Definition einer **Wohlordnung**. R ist eine Wohlordnung von A genau dann, wenn sie auf A fundiert ist und die Elemente von A paarweise R-vergleichbar sind. Ähnlich der Definition 6.24(2), S. 30, von Takeuti und Zaring.

`df-we $a` $\vdash (R \mathrm{We} A \leftrightarrow (R \mathrm{Fr} A \wedge \forall x \forall y ((x \in A \wedge y \in A) \rightarrow (x \; R \; y \vee x = y \vee y \; R \; x))))$

```
df-we $a ( R We A <-> ( R Fr A /\ A. x A. y ( ( x e. A /\ y
         e. A ) -> ( x R y \/ x = y \/ y R x ) ) ) ) $.
```

Definition des **Ordinalprädikats**, das für eine Klasse gilt, die transitiv ist und durch die Epsilon-Relation wohlgeordnet ist. Ähnlich der Definition auf S. 468, Bell und Machover.

`df-ord $a` $\vdash (\mathrm{Ord} A \leftrightarrow (\mathrm{Tr} A \wedge E \mathrm{We} A))$

```
df-ord $a |- ( Ord A <-> ( Tr A /\ E We A ) ) $.
```

Definition der **Klasse aller Ordinalzahlen**. Eine Ordinalzahl ist eine Menge, die das Ordinalprädikat erfüllt. Definition 7.11 von Takeuti und Zaring, S. 38.

`df-on $a` $\vdash \mathrm{On} = \{ x \mid \mathrm{Ord} \; x \}$

```
df-on $a |- On = { x | Ord x } $.
```

Definition des **Limes-Prädikats**, das für eine nicht leere Ordinalzahl gilt, die kein Nachfolger ist (d.h. die die Vereinigung ihrer selbst ist). Vergleiche Bell und Machover, S. 471 und Übung (1), S. 42 von Takeuti und Zaring.

```
df-lim $a ⊢ ( Lim A ↔ ( Ord A ∧ ¬ A = ∅ ∧ A = ⋃ A ) )

df-lim $a |- ( Lim A <-> ( Ord A /\ -. A = (/) /\ A = U. A )
          ) $.
```

Definition eines **Nachfolgers** einer Klasse. Definition 7.22 von Takeuti und Zaring, S. 41. Unsere Definition ist eine Verallgemeinerung auf Klassen, obwohl sie für echte Klassen bedeutungslos ist.

```
df-suc $a ⊢ suc A = ( A ∪ { A } )

df-suc $a |- suc A = ( A u. { A } ) $.
```

Definition der **Klasse der natürlichen Zahlen**. Vergleiche Bell und Machover, S. 471.

```
df-om $a ⊢ ω = { x | ( Ord x ∧ ∀ y ( Lim y → x ∈ y ) ) }

df-om $a |- om = { x | ( Ord x /\ A. y ( Lim y -> x e. y ) )
         } $.
```

Definition eines **kartesischen Produkts** (auch **Kreuzprodukt** genannt) von zwei Klassen. Definition 9.11 von Quine, S. 64.

```
df-xp $a ⊢ ( A × B ) = { ⟨ x , y ⟩ | ( x ∈ A ∧ y ∈ B ) }

df-xp $a |- ( A X. B ) = { <. x , y >. | ( x e. A /\ y e. B)
         } $.
```

Definition einer **Relation**. Definition 6.4(1) von Takeuti und Zaring, S. 23.

```
df-rel $a ⊢ ( Rel A ↔ A ⊆ ( V × V ) )

df-rel $a |- ( Rel A <-> A C_ ( _V X. _V ) ) $.
```

Definition eines **Definitionsbereichs** einer Klasse[9]. Definition 6.5(1) von Takeuti und Zaring, S. 24.

```
df-dm $a ⊢ dom A = { x | ∃ y ⟨ x , y ⟩ ∈ A }

df-dm $a |- dom A = { x | E. y <. x , y >. e. A } $.
```

[9]Anm. der Übersetzer: Der Begriff „Definitionsbereich“ und die folgenden Begriffe wie „Wertebereich“ etc. werden üblicherweise für Funktionen oder zumindest Relationen verwendet, können aber so wie hier für beliebige Klassen definiert werden.

Definition des **Wertebereichs** einer Klasse. Definition 6.5(2) von Takeuti und Zaring, S. 24.

`df-rn $a` $\vdash \operatorname{ran} A = \{ y \mid \exists x \langle x , y \rangle \in A \}$

```
df-rn $a |- ran A = { y | E. x <. x , y >. e. A } $.
```

Definition einer **Einschränkung** einer Klasse. Definition 6.6(1) von Takeuti und Zaring, S. 24.

`df-res $a` $\vdash (A \restriction B) = (A \cap (B \times \mathrm{V}))$

```
df-res $a |- ( A |` B ) = ( A i^i ( B X. _V ) ) $.
```

Definition des **Bildes** einer Klasse. Definition 6.6(2) von Takeuti und Zaring, S. 24.

`df-ima $a` $\vdash (A \mathbin{“} B) = \operatorname{ran} (A \restriction B)$

```
df-ima $a |- ( A " B ) = ran ( A |` B ) $.
```

Definition der **Komposition** zweier Klassen. Definition 6.6(3) von Takeuti und Zaring, S. 24.

`df-co $a` $\vdash (A \circ B) = \{ \langle x , y \rangle \mid \exists z (\langle x , z \rangle \in B \wedge \langle z , y \rangle \in A) \}$

```
df-co $a |- ( A o. B ) = { <. x , y >. | E. z ( <. x , z >.
          e. B /\ <. z , y >. e. A ) } $.
```

Definition einer **Funktion**. Definition 6.4(4) von Takeuti und Zaring, S. 24.

`df-fun $a` $\vdash (\operatorname{Fun} A \leftrightarrow (\operatorname{Rel} A \wedge \forall x \exists z \forall y (\langle x , y \rangle \in A \rightarrow y = z)))$

```
df-fun $a |- ( Fun A <-> ( Rel A /\ A. x E. z A. y ( <. x ,
          y >. e. A -> y = z ) ) ) $.
```

Definition einer **Funktion mit Definitionsbereich**. Definition 6.15(1) von Takeuti und Zaring, S. 27.

`df-fn $a` $\vdash (A \operatorname{Fn} B \leftrightarrow (\operatorname{Fun} A \wedge \operatorname{dom} A = B))$

```
df-fn $a |- ( A Fn B <-> ( Fun A /\ dom A = B ) ) $.
```

Definition einer **Funktion mit Definitionsbereich und Zielbereich**. Definition 6.15(3) von Takeuti und Zaring, S. 27.

`df-f $a` $\vdash (F : A \longrightarrow B \leftrightarrow (F \operatorname{Fn} A \wedge \operatorname{ran} F \subseteq B))$

```
df-f $a |- ( F : A --> B <-> ( F Fn A /\ ran F C_ B ) ) $.
```

Definition einer **injektiven** oder **Eins-zu-eins-Funktion**. Vergleiche Definition 6.15(5) von Takeuti und Zaring, S. 27.

`df-f1 $a` $\vdash (F : A \xrightarrow{1\text{-}1} B \leftrightarrow (F : A \longrightarrow B \wedge \forall y \exists z \forall x (\langle x , y \rangle \in F \rightarrow x = z)))$

```
df-f1 $a |- ( F : A -1-1-> B <-> ( F : A --> B /\ A. y E. z
         A. x ( <. x , y >. e. F -> x = z ) ) ) $.
```

Definition einer **surjektiven** oder **rechtstotalen Funktion**[10]. Definition 6.15(4) von Takeuti und Zaring, S. 27.

`df-fo $a` $\vdash (F : A \xrightarrow[\text{onto}]{} B \leftrightarrow (F \text{ Fn } A \wedge \text{ran } F = B))$

```
df-fo $a |- ( F : A -onto-> B <-> ( F Fn A /\ ran F = B ) )
         $.
```

Definition einer **bijektiven Funktion**. Vergleiche Definition 6.15(6) von Takeuti und Zaring, S. 27.

`df-f1o $a` $\vdash (F : A \xrightarrow[\text{onto}]{1\text{-}1} B \leftrightarrow (F : A \xrightarrow{1\text{-}1} B \wedge F : A \xrightarrow[\text{onto}]{} B))$

```
df-f1o $a |- ( F : A -1-1-onto-> B <-> ( F : A -1-1-> B
          /\ F : A -onto-> B ) ) $.
```

Definition eines **Funktionswertes**. Diese Definition gilt für jede Klasse und wird zur leeren Menge ausgewertet, wenn sie nicht sinnvoll ist. Beachten Sie, dass $F`A$ dasselbe bedeutet wie die bekanntere Notation $F(A)$ für den Wert einer Funktion an der Stelle $A$. Die Notation $F`A$ ist in der formalen Mengenlehre gebräuchlich.

`df-fv $a` $\vdash (F ` A) = \bigcup \{ x \mid (F \text{ “ } \{ A \}) = \{ x \} \}$

```
df-fv $a |- ( F ` A ) = U. { x | ( F " { A } ) = { x } } $.
```

Definition des **Ergebnisses einer Operation**. Hier ist F eine Operation für zwei Operanden (z. B. + für reelle Zahlen). Dies ist auch für echte Klassen A und B definiert, auch wenn es in diesem Fall nicht sinnvoll ist[11]. Die Definition kann jedoch zu einem sinnvollen Ergebnis führen, wenn F eine echte Klasse ist.

`df-opr $a` $\vdash (A \; F \; B) = (F ` \langle A , B \rangle)$

```
df-opr $a |- ( A F B ) = ( F ` <. A , B >. ) $.
```

[10] Im Englischen **onto function**.

[11] Anm. der Übersetzer: das Ergebnis ist in diesem Fall wie bei einem Funktionswert die leere Menge.

3.5 Tricks des Verfahrens

In der Regel war es unser Ziel, in der Datenbasis `set.mm` die moderne Notation zu verwenden. In einigen Fällen wurde aber in unkonventioneller Weise von der in den Standardlehrbüchern verwendeten Sprache abgewichen, um deren Weiterentwicklung zu vereinfachen und die Vorteile der Metamath-Sprache besser zu nutzen. In diesem Abschnitt werden wir einige allgemeine, in `set.mm` verwendete Konventionen beschreiben.

- Das Drehkreuzsymbol $\vdash$, das „es ist beweisbar, dass“ bedeutet, ist das erste Token aller Behauptungen und Hypothesen, die keine Syntaxkonstruktionen sind. Dies ist eine Standardkonvention in der Logik. (Wir haben dies bereits erwähnt, aber dieses Symbol ist für manche Menschen ohne Logikkenntnisse etwas verstörend. Es hat keine tiefere Bedeutung, sondern dient nur dazu, Syntaxkonstruktionen von gewöhnlichen mathematischen Aussagen zu unterscheiden).

- Eine Annahme der Form

  `$e` $\vdash (\varphi \rightarrow \forall x \, \varphi)$

  sollte als „unter der Annahme, dass die Variable x in wff φ (effektiv) nicht frei ist“ verstanden werden. Wörtlich heißt das: „Angenommen, es ist beweisbar, dass $\varphi \rightarrow \forall x \, \varphi$.“ Auf diese Weise können wir die Komplexität vermeiden, die mit der Standardbehandlung von freien und gebundenen Variablen verbunden ist. In der Fußnote auf S. 226 wird dies näher erläutert.

- Eine Aussage in einer der Formen

  `$a` $\vdash (\neg \forall x \; x = y \rightarrow \ldots)$
  `$p` $\vdash (\neg \forall x \; x = y \rightarrow \ldots)$

  sollte als „Wenn x und y verschiedene Variablen sind, dann...“ verstanden werden. Mit solch einer Voraussetzung können wir in der frühen Entwicklung der Prädikatenlogik auf die `$d`-Anweisung verzichten, so dass Symbolmanipulationen konzeptionell so einfach sind wie in der Aussagenlogik. Sobald die `$d`-Anweisung jedoch mehr und mehr zum Einsatz gekommen ist, wird dieses Konstrukt nur noch selten verwendet.

- Die Aussage

  `$d` $x \; y$

  sollte als „Angenommen x und y sind unterschiedliche Variablen“ verstanden werden.

- Die Anweisung

`$d` $x\ \varphi$

sollte als „angenommen x kommt in φ nicht vor“ verstanden werden.

- Die Anweisung

  `$d` $x\ A$

  sollte als „angenommen, die Variable x kommt in der Klasse A nicht vor“ verstanden werden.

- Die folgende Gruppe von Variableneinschränkungen und Hypothesen

  `$d` $x\ A$
  `$d` $x\ \psi$
  `$e` $\vdash (x = A \rightarrow (\varphi \leftrightarrow \psi))$

  wird häufig anstelle der expliziten Substitution verwendet, was bedeutet: „angenommen, ψ ergibt sich aus der echten Substitution von A für x in φ.“[12] Manchmal wird „`$e` $\vdash (\psi \rightarrow \forall x\, \psi)$“ anstelle von „`$d` $x\, \psi$“[13] verwendet, was nur voraussetzt, dass x effektiv nicht frei in φ ist, aber nicht notwendigerweise nicht darin vorkommt. Die Verwendung der impliziten Substitution ist zum Teil eine Frage des persönlichen Stils, obwohl sie Beweise etwas kürzer machen kann, als es bei expliziter Substitution der Fall wäre.

- Die Annahme

  `$e` $\vdash A \in \mathrm{V}$

  sollte als „angenommen, die Klasse A ist eine Menge (d.h. sie existiert)“ verstanden werden. Dies ist eine praktische Konvention, die von Quine verwendet wurde.

- Die Bariablenbeschränkung und die Annahme

  `$d` $x\ y$
  `$e` $\vdash (y \in A \rightarrow \forall x\ y \in A)$

  sollte als „angenommen, die Variable x ist (effektiv) nicht frei in der Klasse A“ verstanden werden.

3.6 Einige Beispiele für Theoreme

In diesem Abschnitt werden einige der wichtigsten Theoreme aufgelistet, die in der Datenbasis `set.mm` bewiesen werden, und sie veranschaulichen, was

[12] Anm. der Übersetzer: dies wird dann eine „implizite Substitution“ genannt.

[13] Anm. der Übersetzer: Solche Hypothesen werden neuerdings durch „`$e` $\vdash F/\, x\, \psi$“ ersetzt.

man mit Metamath alles machen kann. Während alle diese Fakten bekannte Ergebnisse sind, bietet Metamath den Vorteil, dass man ihre Herleitung leicht zu den Axiomen zurückverfolgen kann. Wir wollen hier nicht versuchen, die Details oder die Motivation zu erklären; dafür verweisen wir auf die Lehrbücher, die in den Beschreibungen erwähnt werden. (Die Datei `set.mm` enthält bibliografische Angaben zu den Textverweisen.) Ihre Beweise enthalten oft wichtige Konzepte, die Sie vielleicht mit dem Programm Metamath untersuchen möchten (siehe Abschnitt 3.10). Alle Symbole, die hier verwendet werden, sind in Abschnitt 3.4 definiert. Der Kürze halber haben wir die `$d`-Beschränkungen oder `$f`-Hypothesen für diese Theoreme nicht aufgenommen; wenn Sie unsicher sind, konsultieren Sie die `set.mm`-Datenbasis.

Wir beginnen mit `syl` (dem Prinzip des Syllogismus). In *Principia Mathematica* nennen Whitehead und Russell dies „das Prinzip des Syllogismus... weil... der Syllogismus in Barbara von ihnen abgeleitet ist“ [74, Zitat nach Theorem *2.06 S. 101]. Einige Autoren nennen dieses Gesetz einen „hypothetischen Syllogismus“. Ab 2019 ist `syl` die am häufigsten referenzierte bewiesene Behauptung in der `set.mm`-Datenbasis.[14]

Theorem syl (das Prinzip des Syllogismus).

`syl.1 $e` $\vdash (\varphi \rightarrow \psi)$
`syl.2 $e` $\vdash (\psi \rightarrow \chi)$
`syl $p` $\vdash (\varphi \rightarrow \chi)$

Das folgende Theorem ist nicht sehr tiefgründig, bietet uns aber eine häufig verwendete Notationshilfe. Es erlaubt uns, den Ausdruck „$A \in V$“ als eine kompakte Art zu sagen, dass die Klasse A existiert, d.h. eine Menge ist.

Es gibt zwei Möglichkeiten zu sagen, dass A eine Menge ist: A ist ein Element des Universums V genau dann, wenn A existiert (d.h. wenn es eine Menge gibt, die A entspricht). Theorem 6.9 von Quine, S. 43.

`isset $p` $\vdash (A \in \mathrm{V} \leftrightarrow \exists x \; x = A)$

Als nächstes beweisen wir die Axiome der Standard-ZF-Mengenlehre, die in unserem Axiomensystem fehlen. Aus unserer Sicht sind sie Theoreme, da sie aus den anderen Axiomen abgeleitet werden können.

Das Aussonderungsaxiom, bewiesen aus den anderen Axiomen der ZF-Mengenlehre. Vergleiche Übung 4 von Takeuti und Zaring, S. 22.

`inex1.1 $e` $\vdash A \in \mathrm{V}$
`inex $p` $\vdash (A \cap B) \in \mathrm{V}$

[14] Der Metamath-Programmbefehl `show usage` zeigt die Anzahl der Verwendungen. Am 29.04.2019 (commit 71cbbbdb387e [im GitHub-Repository metamath/set.mm]) wurde `syl` 10.819 Mal direkt referenziert. Die am zweithäufigsten referenzierte bewiesene Assertion war `eqid`, die 7.738 Mal direkt referenziert wurde.

Das Leermengenaxiom, bewiesen aus den anderen Axiomen der ZF-Mengenlehre. Korollar 5.16 von Takeuti und Zaring, S. 20.

`0ex $p` $\vdash \varnothing \in \mathrm{V}$

Das Paarmengenaxiom, bewiesen aus den anderen Axiomen der ZF Mengenlehre. Theorem 7.13 von Quine, S. 51.

`prex $p` $\vdash \{ A , B \} \in \mathrm{V}$

Als nächstes werden wir einige berühmte oder wichtige Theoreme auflisten, die in der Datenbasis `set.mm` bewiesen sind. Keines von ihnen außer `omex` erfordert das Unendlichkeitsaxiom, wie Sie mit dem Metamath-Befehl `show trace_back` überprüfen können.

Die Auflösung des Russell'schen Paradoxons. Es gibt keine Menge, die der Klasse aller Mengen, die nicht Mitglieder ihrer selbst sind, entspricht. Proposition 4.14 von Takeuti und Zaring, S. 14.

`ru $p` $\vdash \neg \exists x \; x = \{ y \mid \neg \, y \in y \}$

Satz von Cantor. Keine Menge kann auf ihre Potenzmenge abgebildet werden. Vergleiche Theorem 6B(b) von Enderton, S. 132.

`canth.1 $e` $\vdash A \in \mathrm{V}$

`canth $p` $\vdash \neg \, F : A \xrightarrow[\text{onto}]{} \mathcal{P} \, A$

Das Burali-Forti-Paradoxon. Keine Menge enthält alle Ordinalzahlen. Enderton, S. 194. (Burali-Forti war eine Person, nicht zwei.)

`onprc $p` $\vdash \neg \, \mathrm{On} \in \mathrm{V}$

Peano-Postulate für die Arithmetik. Satz 7.30 von Takeuti und Zaring, S. 42–43. Die zu beschreibenden Objekte sind die Elemente von ω, d.h. die natürlichen Zahlen 0, 1, 2, ... Die Nachfolger-Operation suc bedeutet „plus eins". `peano1` besagt, dass 0 (die als leere Menge definiert ist) eine natürliche Zahl ist. `peano2` besagt, dass wenn A eine natürliche Zahl ist, $A + 1$ auch eine natürliche Zahl ist. `peano3` besagt, dass 0 nicht der Nachfolger einer natürlichen Zahl ist. `peano4` besagt, dass zwei natürliche Zahlen genau dann gleich sind, wenn ihre Nachfolger gleich sind. `peano5` ist im Wesentlichen dasselbe wie die vollständige Induktion.

`peano1 $p` $\vdash \varnothing \in \omega$

`peano2 $p` $\vdash (A \in \omega \rightarrow \mathrm{suc} \, A \in \omega)$

`peano3 $p` $\vdash (A \in \omega \rightarrow \neg \, \mathrm{suc} \, A = \varnothing)$

`peano4 $p` $\vdash ((A \in \omega \wedge B \in \omega) \rightarrow (\mathrm{suc} \, A = \mathrm{suc} \, B \leftrightarrow A = B))$

`peano5 $p` $\vdash ((\varnothing \in A \wedge \forall x \in \omega \, (x \in A \rightarrow \mathrm{suc} \, x \in A)) \rightarrow \omega \subseteq A)$

Finite Induktion (vollständige Induktion). Die erste Hypothese ist der Induktionsanfang und die zweite ist der Induktionsschritt. Theorem Schema

22 von Suppes, S. 136.

`findes.1 $e` $\vdash [\varnothing / x]\varphi$
`findes.2 $e` $\vdash (x \in \omega \rightarrow (\varphi \rightarrow [\operatorname{suc} x / x]\varphi))$
`findes $p` $\vdash (x \in \omega \rightarrow \varphi)$

Transfinite Induktion mit expliziter Substitution. Die erste Hypothese ist der Induktionsanfang, die zweite ist der Induktionsschritt für Nachfolger und die dritte ist der Induktionsschritt für Grenzzahlen. Theorem Schema 4 von Suppes, S. 197.

`tfindes.1 $e` $\vdash [\varnothing / x]\varphi$
`tfindes.2 $e` $\vdash (x \in \mathrm{On} \rightarrow (\varphi \rightarrow [\operatorname{suc} x / x]\varphi))$
`tfindes.3 $e` $\vdash (\operatorname{Lim} y \rightarrow (\forall x \in y\, \varphi \rightarrow [y / x]\varphi))$
`tfindes $p` $\vdash (x \in \mathrm{On} \rightarrow \varphi)$

Prinzip der transfiniten Rekursion. Theorem 7.41 von Takeuti und Zaring, S. 47. Die transfinite Rekursion ist der grundlegende Satz für eine strenge Definition der Arithmetik von Ordinalzahlen, und hat auch viele andere wichtige Anwendungen. Die Annahmen `tfr.1` und `tfr.2` spezifizieren eine bestimmte (echte) Klasse F. Die komplizierte Definition von F ist an sich nicht wichtig; wichtig ist, dass es ein solches F mit den erforderlichen Eigenschaften gibt, und wir zeigen dies, indem wir F explizit angeben. `tfr1` besagt, dass F eine Funktion ist, deren Definitionsbereich die Menge der Ordnungszahlen ist. `tfr2` besagt, dass jeder Wert von F vollständig durch seine vorherigen Werte und die Werte einer Hilfsfunktion, G, bestimmt ist. `tfr3` besagt, dass F eindeutig ist, d.h. es ist die einzige Funktion, die `tfr1` und `tfr2` erfüllt. Beachten Sie, dass f eine individuelle Variable wie x und y ist; es ist nur eine Gedächtnisstütze, um uns daran zu erinnern, dass A eine Sammlung von Funktionen ist.

`tfr.1 $e` $\vdash A = \{ f \mid \exists x \in \mathrm{On}\, (f \operatorname{Fn} x \wedge \forall y \in x\, (f\,`\,y) = (G\,`\,(f \restriction y))) \}$
`tfr.2 $e` $\vdash F = \bigcup A$
`tfr1 $p` $\vdash F \operatorname{Fn} \mathrm{On}$
`tfr2 $p` $\vdash (z \in \mathrm{On} \rightarrow (F\,`\,z) = (G\,`\,(F \restriction z)))$
`tfr3 $p` $\vdash ((B \operatorname{Fn} \mathrm{On} \wedge \forall x \in \mathrm{On}\, (B\,`\,x) = (G\,`\,(B \restriction x))) \rightarrow B = F)$

Die Existenz von omega (die Klasse der natürlichen Zahlen). Axiom 7 von Takeuti und Zaring, S. 43. (Dies ist das einzige Theorem in diesem Abschnitt, das das Unendlichkeitsaxiom erfordert).

`omex $p` $\vdash \omega \in \mathrm{V}$

3.7 Axiome für reelle und komplexe Zahlen

In diesem Abschnitt werden die Axiome für reelle und komplexe Zahlen vorgestellt und kommentiert. Analysis-Lehrbücher verwenden implizit oder explizit diese Axiome oder ihre Entsprechungen als Ausgangspunkt. In der Datenbasis `set.mm` definieren wir reelle und komplexe Zahlen als (ziemlich

komplizierte) spezifische Mengen und leiten diese Axiome als *Theoreme* aus den Axiomen der ZF-Mengenlehre ab, indem wir eine Methode der Dedekindschen Schnitte verwenden. Wir lassen die Details dieser Konstruktion weg, die Sie bei Bedarf mit Hilfe der Datenbasis `set.mm` in Verbindung mit den darin referenzierten Lehrbüchern nachvollziehen können.

Sobald wir diese Theoreme bewiesen haben, formulieren wir die bewiesenen Theoreme als Axiome neu. Auf diese Weise können wir leicht erkennen, welche Axiome für einen bestimmten Beweis komplexer Zahlen benötigt werden, ohne durch die Komplexität ihrer Herleitung durch die Mengenlehre abgelenkt zu werden. Infolgedessen ist die Konstruktion eigentlich unwichtig, außer um zu zeigen, dass es Mengen gibt, die den Axiomen genügen, und dass die Axiome folglich konsistent sind, wenn die Mengenlehre konsistent ist. Wenn man mit reellen Zahlen arbeitet, kann man sie tatsächlich als die Mengen betrachten, die sich aus der Konstruktion ergeben (für die Definitheit), oder man kann sie als nicht weiter spezifizierte Mengen betrachten, die zufälligerweise die Axiome erfüllen. Die Herleitung ist nicht einfach, aber die Tatsache, dass sie funktioniert, ist bemerkenswert und unterstützt die Idee, dass die ZFC-Mengenlehre alles ist, was wir brauchen, um eine Grundlage für die gesamte Mathematik zu schaffen.

3.7.1 Die Axiome für reelle und komplexe Zahlen selbst

Für die Axiome werden uns 8 Klassen vorgegeben (oder vorausgesetzt): $\mathbb{C}$ (die Menge der komplexen Zahlen), $\mathbb{R}$ (die Menge der reellen Zahlen, eine Teilmenge von $\mathbb{C}$), 0 (Null), 1 (Eins), i (Quadratwurzel aus -1), $+$ (plus), $\cdot$ (mal) und $<_{\mathbb{R}}$ (kleiner als, nur für die reellen Zahlen). Subtraktion und Division sind definierte Begriffe und werden nicht in den Axiomen verwendet. Für ihre Definitionen siehe `set.mm`.

Man beachte, dass die Notation $(A+B)$ (und ähnlich $(A\cdot B)$) eine Klasse bezeichnet, die als *Operation*, bezeichnet wird und den Funktionswert der Klasse $+$ für das geordnete Paar $\langle A, B\rangle$ darstellt. Eine Operation ist durch die Aussage `df-opr` auf Seite 95 definiert. Die Notation $A <_{\mathbb{R}} B$ bezeichnet eine wff, die als *binäre Relation* bezeichnet wird und $\langle A, B\rangle \in <_{\mathbb{R}}$ bedeutet, wie durch `df-br` auf Seite 91 definiert.

Wir gehen davon aus, dass die 8 vorgegebenen Klassen die folgenden 22 Axiome erfüllen (in den unten aufgeführten Axiomen wird kurz $<$ statt $<_{\mathbb{R}}$ verwendet).

1. Die reellen Zahlen sind eine Teilmenge der komplexen Zahlen.
   `ax-resscn $p` $\vdash \mathbb{R} \subseteq \mathbb{C}$
2. Eins ist eine komplexe Zahl.
   `ax-1cn $p` $\vdash 1 \in \mathbb{C}$
3. Die imaginäre Einheit i ist eine komplexe Zahl.
   `ax-icn $p` $\vdash i \in \mathbb{C}$
4. Komplexe Zahlen sind bzgl. der Addition abgeschlossen.

`ax-addcl $p` $\vdash ((A \in \mathbb{C} \wedge B \in \mathbb{C}) \rightarrow (A + B) \in \mathbb{C})$

5. Reelle Zahlen sind bzgl. der Addition abgeschlossen.
`ax-addrcl $p` $\vdash ((A \in \mathbb{R} \wedge B \in \mathbb{R}) \rightarrow (A + B) \in \mathbb{R})$

6. Komplexe Zahlen sind bzgl. der Multiplikation abgeschlossen.
`ax-mulcl $p` $\vdash ((A \in \mathbb{C} \wedge B \in \mathbb{C}) \rightarrow (A \cdot B) \in \mathbb{C})$

7. Reelle Zahlen sind bzgl. der Multiplikation abgeschlossen.
`ax-mulrcl $p` $\vdash ((A \in \mathbb{R} \wedge B \in \mathbb{R}) \rightarrow (A \cdot B) \in \mathbb{R})$

8. Die Multiplikation von komplexen Zahlen ist kommutativ.
`ax-mulcom $p` $\vdash ((A \in \mathbb{C} \wedge B \in \mathbb{C}) \rightarrow (A \cdot B) = (B \cdot A))$

9. Die Addition von komplexen Zahlen ist assoziativ.
`ax-addass $p` $\vdash ((A \in \mathbb{C} \wedge B \in \mathbb{C} \wedge C \in \mathbb{C}) \rightarrow ((A + B) + C) = (A + (B + C)))$

10. Die Multiplikation von komplexen Zahlen ist assoziativ.
`ax-mulass $p` $\vdash ((A \in \mathbb{C} \wedge B \in \mathbb{C} \wedge C \in \mathbb{C}) \rightarrow ((A \cdot B) \cdot C) = (A \cdot (B \cdot C)))$

11. Die Multiplikation der komplexen Zahlen ist distributiv bzgl. der Addition.
`ax-distr $p` $\vdash ((A \in \mathbb{C} \wedge B \in \mathbb{C} \wedge C \in \mathbb{C}) \rightarrow (A \cdot (B + C)) = ((A \cdot B) + (A \cdot C)))$

12. Das Quadrat von i ist gleich -1 (ausgedrückt als i-Quadrat plus Eins ist Null).
`ax-i2m1 $p` $\vdash ((i \cdot i) + 1) = 0$

13. Eins und Null sind verschieden.
`ax-1ne0 $p` $\vdash 1 \neq 0$

14. Eines ist ein neutrales Element für die reelle Multiplikation.
`ax-1rid $p` $\vdash (A \in \mathbb{R} \rightarrow (A \cdot 1) = A)$

15. Zu jeder reellen Zahl gibt es eine entsprechende negative Zahl (additives Inverses).
`ax-rnegex $p` $\vdash (A \in \mathbb{R} \rightarrow \exists x \in \mathbb{R} (A + x) = 0)$

16. Jede reelle Zahl ungleich Null hat einen Kehrwert.
`ax-rrecex $p` $\vdash (A \in \mathbb{R} \rightarrow (A \neq 0 \rightarrow \exists x \in \mathbb{R} (A \cdot x) = 1))$

17. Eine komplexe Zahl kann durch zwei reelle Zahlen ausgedrückt werden.
`ax-cnre $p` $\vdash (A \in \mathbb{C} \rightarrow \exists x \in \mathbb{R} \exists y \in \mathbb{R} A = (x + (y \cdot i)))$

18. Die Ordnung der reellen Zahlen erfüllt die strenge Trichotomie.
`ax-pre-lttri $p` $\vdash ((A \in \mathbb{R} \wedge B \in \mathbb{R}) \rightarrow (A < B \leftrightarrow \neg (A = B \vee B < A)))$

19. Die Ordnung der reellen Zahlen ist transitiv.
`ax-pre-lttrn $p` $\vdash ((A \in \mathbb{R} \wedge B \in \mathbb{R} \wedge C \in \mathbb{R}) \rightarrow ((A < B \wedge B < C) \rightarrow A < C))$

20. Die Ordnung der reellen Zahlen ist invariant bzgl. der Addition.
`ax-pre-ltadd $p` $\vdash ((A \in \mathbb{R} \wedge B \in \mathbb{R} \wedge C \in \mathbb{R}) \rightarrow (A < B \rightarrow (C + A) < (C + B)))$

21. Das Produkt zweier positiver reeller Zahlen ist positiv.
`ax-pre-mulgt0 $p` $\vdash ((A \in \mathbb{R} \wedge B \in \mathbb{R}) \rightarrow ((0 < A \wedge 0 < B) \rightarrow 0 < (A \cdot$

B)))

22. Eine nicht leere, nach oben begrenzte Menge von reellen Zahlen hat ein Supremum.

`ax-pre-sup $p` $\vdash ((A \subseteq \mathbb{R} \wedge A \neq \varnothing \wedge \exists x \in \mathbb{R} \, \forall y \in A \; y < x) \rightarrow \exists x \in \mathbb{R} \, (\forall y \in A \, \neg x < y \wedge \forall y \in \mathbb{R} \, (y < x \rightarrow \exists z \in A \; y < z)))$

Dies ist der vollständige Satz der Axiome für reelle und komplexe Zahlen. Sehen Sie sich an, wie Subtraktion, Division und Dezimalzahlen in `set.mm` definiert sind, und schauen Sie sich zum Spaß den Beweis von $2 + 2 = 4$ (Theorem `2p2e4` in `set.mm`) an, wie in Abschnitt 3.8 besprochen.

In `set.mm` definieren wir die positiven ganzen Zahlen $\mathbb{N}$, die nichtnegativen ganzen Zahlen $\mathbb{N}_0$, die ganzen Zahlen $\mathbb{Z}$ und die rationalen Zahlen $\mathbb{Q}$ als Teilmengen von $\mathbb{R}$. Dies führt zu der schönen Teilmengenkette $\mathbb{N} \subseteq \mathbb{N}_0 \subseteq \mathbb{Z} \subseteq \mathbb{Q} \subseteq \mathbb{R} \subseteq \mathbb{C}$, was uns einen einheitlichen Rahmen für die Arithmetik gibt, in dem zum Beispiel eine Eigenschaft wie die Kommutativität der Addition komplexer Zahlen automatisch für ganze Zahlen gilt. Die natürlichen Zahlen $\mathbb{N}$[15] unterscheiden sich von der zuvor definierten Menge ω, aber beide erfüllen die Peanoschen Postulate.

3.7.2 Axiome für komplexe Zahlen in Texten zur Analysis

Die meisten Texte zur Analysis konstruieren komplexe Zahlen als geordnete Paare von reellen Zahlen, was zu konstruktionsabhängigen Eigenschaften führt, die diese Axiome erfüllen, aber nicht in ihrer reinen Form angegeben werden. (Dies geschieht auch in `set.mm`, aber unsere Axiome abstrahieren von dieser Konstruktion.) In anderen Texten heißt es einfach, dass $\mathbb{R}$ ein „komplettes geordnetes Teilfeld von $\mathbb{C}$ ist“, was zu redundanten Axiomen führt, wenn man diese Phrase vollständig ausformuliert. Tatsächlich habe ich noch keinen Text gesehen, der die Axiome in der obigen expliziten Form enthält. Keines dieser Axiome ist individuell einzigartig, aber diese sorgfältig ausgearbeitete Sammlung von Axiomen ist das Ergebnis jahrelanger Arbeit der Metamath-Gemeinschaft.

3.7.3 Beseitigung unnötiger Axiome für komplexe Zahlen

Metamath hatte ursprünglich mehr Axiome für reelle und komplexe Zahlen, aber im Laufe der Zeit haben wir (die Metamath-Gemeinschaft) Wege gefunden, unnötige Axiome zu eliminieren (indem wir sie anhand anderer Axiome bewiesen haben) oder sie abzuschwächen (indem wir schwächere

[15] Anm. der Übersetzer: sowohl im Deutschen als auch im Englischen ist nicht eindeutig festgelegt, ob mit dem Begriff „natürliche Zahlen“ die positiven ganzen Zahlen $\mathbb{N}$ oder die nichtnegativen ganzen Zahlen $\mathbb{N}_0$ gemeint werden.

Behauptungen aufgestellt haben, ohne die Beweisbarkeit der auf sie aufbauenden Theoreme zu reduzieren). Es folgen einige Aussagen, die früher Axiome für komplexe Zahlen waren, die aber inzwischen (mit Metamath) formal als überflüssig nachgewiesen wurden:

- $\mathbb{C} \in V$. Früher wurde dies als „Axiom der komplexen Zahlen“ aufgeführt. Es handelt sich jedoch eigentlich nicht um ein Axiom der komplexen Zahlen, und sein Beweis verwendet in jedem Fall Axiome der Mengenlehre. Von Mario Carneiro am 17-Nov-2014 als redundant bewiesen (siehe `axcnex`).
- $((A \in \mathbb{C} \wedge B \in \mathbb{C}) \rightarrow (A + B) = (B + A))$. Von Eric Schmidt am 19-Jun-2012 als redundant bewiesen und von Scott Fenton am 3-Jan-2013 formalisiert (siehe `addcom`).
- $(A \in \mathbb{C} \rightarrow (A + 0) = A)$. Von Eric Schmidt am 19. Juni 2012 als überflüssig bewiesen und von Scott Fenton am 3. Januar 2013 formalisiert (siehe `addid1`).
- $(A \in \mathbb{C} \rightarrow \exists x \in \mathbb{C}(A + x) = 0)$. Von Eric Schmidt für überflüssig bewiesen und am 21. Mai 2007 formalisiert (siehe `cnegex`).
- $((A \in \mathbb{C} \wedge A \neq 0) \rightarrow \exists x \in \mathbb{C}(A \cdot x) = 1)$. Von Eric Schmidt für überflüssig bewiesen und am 22. Mai 2007 formalisiert (siehe `recex`).
- $0 \in \mathbb{R}$. Von Eric Schmidt am 19-Feb-2005 als überflüssig bewiesen und am 21-Mai-2007 formalisiert (siehe `0re`).

Wir könnten 0 als axiomatisches Objekt eliminieren, indem wir es als $((i \cdot i) + 1)$ definieren und es in den Axiomen durch diesen Ausdruck ersetzen. In diesem Fall wird das Axiom ax-i2m1 überflüssig. Die übrigen Axiome würden jedoch länger und weniger intuitiv werden.

Eric Schmidts Arbeit, in der er dieses Axiomensystem[61] analysiert, enthält einen Beweis dafür, dass die verbleibenden Axiome, mit der eventuellen Ausnahme von ax-mulcom, unabhängig von den anderen sind. Es ist derzeit eine offene Frage, ob ax-mulcom unabhängig von den anderen Axiomen ist.

3.8 Zwei plus zwei ist gleich vier

Es folgt ein Beweis, dass $2 + 2 = 4$, wie im Theorem `2p2e4` in der Datenbasis `set.mm` bewiesen wird. Damit wird anschaulich demonstriert, wie ein Metamath-Beweis aussehen kann. Dieser Beweis hat vielleicht mehr Schritte, als Sie gewohnt sind, aber jeder Schritt ist streng bewiesen, bis hin zu den Axiomen der Logik und Mengenlehre. Diese Darstellung wurde ursprünglich vom Metamath-Programm als HTML-Datei erzeugt (siehe `https://us.metamath.org/mpeuni/2p2e4.html`).

In der Tabelle, die den Beweis zeigt, ist „Schritt“ die sequentielle Nummer des entsprechenden Schritts, während der zugehörige „Ausdruck“ ein Ausdruck ist, den wir bewiesen haben. Unter „Ref“ (Referenz) ist der Name eines Theorems oder Axioms, das diesen Ausdruck rechtfertigt, und „Hyp“ bezieht sich auf vorangegangene Schritte (falls vorhanden), die das Theorem oder Axiom benötigt, damit wir es verwenden können. Ausdrücke werden weiter eingerückt als die von ihnen abhängigen Ausdrücke, um ihre Abhängigkeiten zu verdeutlichen.

Tabelle 3.1: Zwei plus zwei ist gleich vier

Step	**Hyp**	**Ref**	**Expression**
1		df-2	$\vdash 2 = 1 + 1$
2	1	oveq2i	$\vdash (2 + 2) = (2 + (1 + 1))$
3		df-4	$\vdash 4 = (3 + 1)$
4		df-3	$\vdash 3 = (2 + 1)$
5	4	oveq1i	$\vdash (3 + 1) = ((2 + 1) + 1)$
6		2cn	$\vdash 2 \in \mathbb{C}$
7		ax-1cn	$\vdash 1 \in \mathbb{C}$
8	6,7,7	addassi	$\vdash ((2 + 1) + 1) = (2 + (1 + 1))$
9	3,5,8	3eqtri	$\vdash 4 = (2 + (1 + 1))$
10	2,9	eqtr4i	$\vdash (2 + 2) = 4$

Schritt 1 besagt, dass wir behaupten können, dass $2 = 1 + 1$ ist, weil es durch `df-2` gerechtfertigt ist. Was ist `df-2`? Es ist einfach die Definition von 2, die in unserem System als gleich $1 + 1$ definiert ist. Dies zeigt, wie wir Definitionen in Beweisen verwenden können.

Sehen Sie sich Schritt 2 des Beweises an. In der Spalte „Ref“ sehen wir, dass er sich auf ein zuvor bewiesenes Theorem, `oveq2i`, bezieht. Es stellt sich heraus, dass das Theorem `oveq2i` eine Annahme erfordert, und in der Spalte Hyp von Schritt 2 geben wir an, dass Schritt 1 diese Annahme erfüllt (entspricht). Wenn wir uns `oveq2i` ansehen, stellen wir fest, dass es beweist, dass wir bei einer Annahme $A = B$ beweisen können, dass $(CFA) = (CFB)$. Wenn wir `oveq2i` benutzen und das Ergebnis von Schritt 1 als Annahme verwenden, bedeutet das, dass $A = 2$ und $B = (1 + 1)$ innerhalb dieser Verwendung von `oveq2i` gesetzt wird. Für C und F können wir beliebige Werte einsetzen (vorbehaltlich der syntaktischen Einschränkungen), also können wir $C = 2$ und $F = +$ wählen, was zu unserem gewünschten Ergebnis $(2 + 2) = (2 + (1 + 1))$ führt.

Schritt 2 ist ein Beispiel für eine Substitution. Letztendlich verwendet jeder Schritt in jedem Beweis nur diese eine Substitutionsregel. Alle Regeln der Logik und alle Axiome sind so ausgedrückt, dass sie mittels dieser einen Substitutionsregel verwendet werden können. Wenn Sie also einmal die Substitution beherrschen, können Sie jeden Metamath-Beweis beherrschen, ohne Ausnahmen.

Jeder Schritt ist klar und kann sofort überprüft werden. In der HTML-Anzeige können Sie sogar auf jeden Verweis klicken, um zu sehen, warum er gerechtfertigt ist, so dass Sie leicht erkennen können, warum der Beweis funktioniert.

3.9 Deduktion

Streng genommen ist eine Deduktion (auch Inferenz genannt) eine Art von Aussage, bei der einige Annahmen wahr sein müssen, damit ihre Schlussfolgerung wahr ist. Ein Theorem hingegen hat keine Hypothesen. Informell werden beide Arten von Aussagen oft als Theoreme bezeichnet, aber in diesem Abschnitt werden wir uns an die strengen Definitionen halten.

Es kommt manchmal vor, dass wir bereits eine Deduktion der Form $\varphi \Rightarrow \psi$ bewiesen haben (bei gegebener Annahme φ können wir ψ beweisen) und wir wollen dann ein Theorem der Form $\varphi \rightarrow \psi$ beweisen.

Die Umwandlung einer Deduktion (die eine Annahme verwendet) in ein Theorem (das dies nicht tut) ist nicht so einfach, wie man vielleicht denkt. Die Deduktion besagt: „Wenn wir φ beweisen können, dann können wir auch ψ beweisen“, was in gewisser Weise schwächer ist als die Aussage „φ impliziert ψ“. Es gibt kein Axiom der Logik, das uns erlaubt, das Theorem direkt aus der Deduktion zu erhalten.[16]

Dies steht im Gegensatz zum umgekehrten Weg. Wenn wir das Theorem $(\varphi \rightarrow \psi)$ haben, ist es einfach, die Deduktion $(\varphi \Rightarrow \psi)$ mit Hilfe des Modus ponens (`ax-mp`; siehe Abschnitt 3.3.1) wiederherzustellen.

In den folgenden Unterabschnitten besprechen wir zunächst das Standard-Deduktionstheorem (die traditionelle, aber umständliche Art, Deduktionen in Theoreme umzuwandeln) und das Theorem der schwachen Deduktion (eine eingeschränkte Version des Standard-Deduktionstheorems, die einfacher zu handhaben ist und früher in der Mengenlehre-Datenbasis `set.mm` weit verbreitet war. In Abschnitt 3.9.3 besprechen wir den Deduktionsstil, den neueren Ansatz, den wir jetzt in den meisten Fällen empfehlen. Der Deduktionsstil verwendet die „Deduktionsform“, eine Form, bei der jeder Annahme (außer Definitionen) und der Schlussfolgerung eine universelle Prämisse vorangestellt wird („$\varphi \rightarrow$“). Der Deduktionsstil ist in `set.mm` weit verbreitet, so dass es nützlich ist, ihn zu verstehen und zu begreifen, warum er weit verbreitet ist. In Abschnitt 3.9.4 wird kurz unser Ansatz zur Verwendung der natürlichen Deduktion in `set.mm` erörtert, da dieser Ansatz eng mit dem Deduktionsstil verbunden ist. Wir schließen mit einer Zusammenfassung der Stärken unseres Ansatzes, die wir für überzeugend halten.

[16] Die Umwandlung einer Deduktion in ein Theorem gilt nicht einmal allgemein für die Quantenlogik, die eine schwache Untermenge der klassischen Aussagenlogik ist. Es wurde gezeigt, dass das Hinzufügen des Standard-Deduktionstheorems (siehe unten) zur Quantenlogik diese zur klassischen Aussagenlogik macht!

3.9.1 Das Standard-Deduktionstheorem

Die Informationen, die in der Deduktion oder ihrem Beweis enthalten sind, können genutzt werden, um uns beim Beweis des zugehörigen Theorems zu helfen. In traditionellen Logikbüchern gibt es ein Metatheorem, das sogenannte Deduktionstheorem, das von Herbrand und Tarski um 1930 unabhängig voneinander entdeckt wurde. Das Deduktionstheorem, das wir oft als Standard-Deduktionstheorem bezeichnen, liefert einen Algorithmus für die Konstruktion eines Beweises eines Theorems aus dem Beweis seiner entsprechenden Deduktion. Siehe z. B. [39, S. 56]. Um einen Beweis für ein Theorem zu konstruieren, betrachtet der Algorithmus jeden Schritt im Beweis der ursprünglichen Deduktion und ersetzt den Schritt durch mehrere Schritte, wobei die Annahme eliminiert und zu einer Prämisse wird.

In der gewöhnlichen Mathematik führt niemand den Algorithmus tatsächlich aus, weil er (in seiner einfachsten Form) eine exponentielle Explosion der Anzahl der Beweisschritte mit sich bringt, je mehr Annahmen eliminiert werden. Stattdessen beruft man sich auf das Standard-Deduktionstheorem, um zu behaupten, dass der Algorithmus prinzipiell durchführbar ist, ohne ihn tatsächlich auszuführen. Außerdem ist der Algorithmus nicht so einfach, wie er auf den ersten Blick erscheinen mag, wenn man ihn rigoros anwendet. Es gibt eine subtile Einschränkung des Standard-Deduktionstheorems, die bei der Arbeit mit der Prädikatenlogik berücksichtigt werden muss, nämlich das Axiom der Verallgemeinerung (weitere Einzelheiten finden Sie in der Literatur).

Eines der Ziele von Metamath ist es, mit möglichst wenigen zugrundeliegenden Konzepten deutlich zu machen, wie Mathematik direkt aus den Axiomen abgeleitet werden kann, und nicht indirekt nach irgendwelchen versteckten Regeln, die in einem Programm vergraben sind oder nur von Logikern verstanden werden. Wenn wir das Standard-Deduktionstheorem zur Sprache und zum Beweisverifizierer hinzufügen würden, würde das beides stark verkomplizieren und Metamaths Ziel der Einfachheit weitgehend zunichte machen. Im Prinzip könnten wir direkt Beweise erstellen, indem wir die vom Algorithmus des Standard-Deduktionstheorems generierten Beweisschritte erweitern. Aber das ist in der Praxis kaum machbar, weil die Anzahl der Beweisschritte schnell riesig, ja sogar astronomisch groß wird. Da der Algorithmus des Standard-Deduktionstheorems durch den Beweis gesteuert wird, müssten wir diesen Beweis noch einmal von vorne durchgehen - ausgehend von den Axiomen -, um das entsprechende Theorem zu erhalten. In Bezug auf die Länge des Beweises würde es keine Einsparungen geben, wenn man das Theorem direkt beweist, anstatt zuerst die Deduktionsform zu beweisen.

3.9.2 Das Theorem der schwachen Deduktion

Wir haben eine effizientere Methode entwickelt, um ein Theorem aus einer Deduktion zu beweisen, die in vielen (aber nicht allen) Fällen anstelle des Standard-Deduktionstheorems verwendet werden kann. Wir nennen diese effizientere Methode das Theorem der schwachen Deduktion.[17] Im Gegensatz zum Standard-Deduktionstheorem erzeugt das Theorem der schwachen Deduktion das Theorem direkt aus einer speziellen Substitutionsinstanz der Deduktion, wobei eine kleine, feste Anzahl von Schritten verwendet wird, die ungefähr proportional zur Länge des endgültigen Theorems ist.

Wenn Sie auf einen Beweis stoßen, der auf das Theorem der schwachen Deduktion `dedth` (oder eine seiner Varianten `dedthxx`) verweist, können Sie dem Beweis folgendermaßen folgen, ohne sich in die Details zu vertiefen: Klicken Sie einfach auf das Theorem, auf das in dem Schritt direkt vor dem Verweis auf `dedth` verwiesen wird, und ignorieren Sie alles andere. Das Theorem `dedth` verwandelt einfach eine Annahme in eine Prämisse (d.h. die Annahme, gefolgt von $\rightarrow$, wird vor die Behauptung gestellt, und die Annahme selbst wird eliminiert), wenn bestimmte Bedingungen erfüllt sind.

Das Theorem der schwachen Deduktion eliminiert eine Annahme φ und macht sie zu einer Prämisse. Dies geschieht durch den Beweis eines Ausdrucks $\varphi \rightarrow \psi$ bei zwei Annahmen: (1) $(A = \text{if}(\varphi, A, B) \rightarrow (\varphi \leftrightarrow \chi))$ und (2) χ. Man beachte, dass es für φ einen Beweis geben muss, wenn die Klassenvariable A durch eine bestimmte Klasse B ersetzt wird. Die Annahme χ sollte der Inferenz zugewiesen werden. Die Details des Beweises des Theorems der schwachen Deduktion können Sie im Theorem `dedth` sehen.

Das Theorem der schwachen Deduktion ist wahrscheinlich einfacher zu verstehen, wenn man Beweise studiert, die es verwenden. Sehen wir uns zum Beispiel den Beweis von `renegcl` an, der beweist, dass $\vdash (A \in \mathbb{R} \rightarrow -A \in \mathbb{R})$:

Step	Hyp	Ref	Expression
1		negeq	$\vdash (A = \text{if} (A \in \mathbb{R} , A , 1) \rightarrow - A = - \text{if} (A \in \mathbb{R} , A , 1))$
2	1	eleq1d	$\vdash (A = \text{if} (A \in \mathbb{R} , A , 1) \rightarrow (- A \in \mathbb{R} \leftrightarrow - \text{if} (A \in \mathbb{R} , A , 1) \in \mathbb{R}))$
3		1re	$\vdash 1 \in \mathbb{R}$
4	3	elimel	$\vdash \text{if}(A \in \mathbb{R}, A, 1) \in \mathbb{R}$
5	4	renegcli	$\vdash -\text{if}(A \in \mathbb{R}, A, 1) \in \mathbb{R}$
6	2,5	dedth	$\vdash (A \in \mathbb{R} \rightarrow -A \in \mathbb{R})$

Die etwas seltsam aussehenden Schritte in `renegcl` vor Schritt 5 sind technischer Natur, die dafür sorgen, dass dieser Zauber funktioniert, und sie können für einen schnellen Überblick über den Beweis ignoriert werden. Um

[17] Es gibt auch ein davon unabhängiges „Theorem der schwachen Deduktion“ im Bereich der Relevanzlogik, so dass wir, um Verwirrung zu vermeiden, unser Theorem „Theorem der schwachen Deduktion für die klassische Logik“ nennen könnten.

den „wichtigen“ Teil des Beweises von `renegcl` weiter zu verfolgen, können Sie sich den Verweis auf `renegcli` in Schritt 5 ansehen.

Nachdem dies geklärt ist, wollen wir uns kurz ansehen, wie für `renegcl` das Theorem der schwachen Deduktion (`dedth`) verwendet wird, um seine Aufgabe zu erfüllen, falls Sie etwas Ähnliches tun oder es besser verstehen wollen. Lassen Sie uns im Beweis von `renegcl` rückwärts arbeiten. Schritt 6 wendet `dedth` an, um unser Zielergebnis $\vdash (A \in \mathbb{R} \rightarrow -A \in \mathbb{R})$ zu erzeugen. Dies erfordert zum einen die (substituierte) Deduktion `renegcli` in Schritt 5. Von sich aus beweist `renegcli` die Deduktion $\vdash A \in \mathbb{R} \Rightarrow \vdash -A \in \mathbb{R}$; dies ist die Deduktionsform, die wir in Theoremform zu bringen versuchen, und somit hat `renegcli` eine eigene Annahme, die erfüllt werden muss. Um die Annahme des Aufrufs von `renegcli` in Schritt 5 zu erfüllen, wird sie schließlich auf das bereits bewiesene Theorem $1 \in \mathbb{R}$ in Schritt 3 reduziert. Schritt 4 verbindet die Schritte 3 und 5; Schritt 4 ruft `elimel` auf, einen Spezialfall von `elimhyp`, der eine Ist-Element-Von-Hypothese für das Theorem der schwachen Deduktion eliminiert[18].

Andererseits muss die Äquivalenz der Schlussfolgerung von `renegcl` ($-A \in \mathbb{R}$) und der substituierten Schlussfolgerung von `renegcli` bewiesen werden, was in Schritt 2 und 1 geschieht.

Das Theorem der schwachen Deduktion hat seine Grenzen. Insbesondere müssen wir in der Lage sein, einen Spezialfall der Annahme der Deduktion als eigenständiges Theorem zu beweisen. Wir haben zum Beispiel $1 \in \mathbb{R}$ in Schritt 3 von `renegcl` verwendet.

Früher haben wir das Theorem der schwachen Deduktion ausgiebig in `set.mm` verwendet. Inzwischen empfehlen wir jedoch in den meisten Fällen die Anwendung des „Deduktionsstils“, da der Deduktionsstil oft eine einfachere und klarere Vorgehensweise ermöglicht. Daher werden wir nun den Deduktionsstil beschreiben.

3.9.3 Deduktionsstil

Wir ziehen es jetzt vor, Behauptungen in „Deduktionsform“ zu schreiben, um die Verwendung des Standard-Deduktionstheorems oder des Theorems der schwachen Deduktion in Beweisen zu vermeiden. Wir nennen diesen Ansatz „Deduktionsstil“.

Es wird einfacher sein, dies zu erklären, wenn man zunächst einige Begriffe definiert:

- **geschlossene Form**: Behauptungen (Theoreme) ohne Hypothesen. Normalerweise hat ihre Bezeichnung kein spezielles Suffix. Ein Beispiel ist `unss`, das besagt: $\vdash ((A \subseteq C \wedge B \subseteq C) \leftrightarrow (A \cup B) \subseteq C)$

[18] Anm. der Übersetzer: Das Theorem `elimel` besagt $B \in C \Rightarrow \text{if}(A \in C, A, B) \in C$, also dass entweder der „then“-Teil des if-Statements in C enthalten ist, was wegen der „if“-Bedingungung der Fall ist, oder der „else“-Teil, was aus der Annahme $B \in C$ folgt.

- **Deduktionsform**: Behauptungen mit einer oder mehreren Hypothesen, bei der die Schlussfolgerung eine Implikation mit einer wff-Variablen als Prämisse (normalerweise φ) ist und jede Hypothese ($e Aussage) entweder (1) eine Implikation mit derselben Prämisse wie die der Schlussfolgerung oder (2) eine Definition ist. Eine Definition kann für eine Klassenvariable (dies ist eine Klassenvariable, gefolgt von „=“) oder eine wff-Variable (dies ist eine wff-Variable, gefolgt von $\leftrightarrow$) erfolgen; Klassenvariablendefinitionen sind häufiger. In der Praxis wird ein Beweis in Deduktionsform auch viele Schritte enthalten, die Implikationen sind, bei denen die Prämisse entweder diese wff-Variable ist (normalerweise φ) oder eine Konjunktion (...$\wedge$...), die diese wff-Variable (φ) enthält. Wenn eine Behauptung in Deduktionsform vorliegt, und auch andere Formen möglich sind, dann fügen wir ihrer Bezeichnung den Suffix „d“ hinzu. Ein Beispiel dafür ist `unssd`, das besagt[19]: $\vdash (\varphi \rightarrow A \subseteq C) \quad \& \quad \vdash (\varphi \rightarrow B \subseteq C) \quad \Rightarrow \quad \vdash (\varphi \rightarrow (A \cup B) \subseteq C)$

- **Inferenzform**: Behauptungen mit einer oder mehreren Annahmen, die nicht in Deduktionsform vorliegen (z.B. gibt es keine gemeinsame Prämisse). Liegt eine Behauptung in der Inferenzform vor und sind auch andere Formen möglich, so fügen wir der Bezeichnung ein „i“ hinzu. Ein Beispiel ist `unssi`, das besagt: $\vdash A \subseteq C \quad \& \quad \vdash B \subseteq C \quad \Rightarrow \quad \vdash (A \cup B) \subseteq C$

Wenn wir den Deduktionsstil verwenden, drücken wir eine Behauptung in der Deduktionsform aus. In dieser Form wird jeder Annahme (mit Ausnahme von Definitionen) und der Schlussfolgerung eine universelle Prämisse („$\varphi \rightarrow$“) vorangestellt. Die Prämisse (z.B. φ) ahmt den Kontext nach, der im Deduktionstheorem behandelt wird, so dass es nicht notwendig ist, das Deduktionstheorem direkt zu verwenden.

Sobald Sie eine Behauptung in Deduktionsform haben, können Sie sie leicht in die Inferenzform oder geschlossene Form umwandeln:

- Um eine Behauptung Ti in Inferenzform zu beweisen, wenn die Behauptung Td in Deduktionsform vorliegt, gibt es einen einfachen mechanischen Prozess, den man anwenden kann. Zuerst nimmt man jede Annahme Ti und fügt ein `T.` $\rightarrow$-Präfix („wahr impliziert“) unter Verwendung von `a1i` ein. Sie können dann die vorhandene Behauptung Td verwenden, um die resultierende Schlussfolgerung mit einem `T.` $\rightarrow$-Präfix beweisen. Schließlich können Sie dieses Präfix mit `trud` entfernen, was zu der Schlussfolgerung führt, die Sie beweisen wollten[20].

[19] Der Kürze halber zeigen wir hier (und an anderen Stellen) ein & zwischen Hypothesen und ein $\Rightarrow$ zwischen den Hypothesen und der Schlussfolgerung. Diese Notation ist technisch gesehen nicht Teil der Metamath-Sprache, sondern eine bequeme Abkürzung, um sowohl die Hypothesen als auch die Schlussfolgerung darzustellen.

[20] Anm. der Übersetzer: Die Schlussfolgerung der Behauptung Ti muss mittels `mptru`

- Um eine Behauptung T in geschlossener Form zu beweisen, wenn die Behauptung Td in Deduktionsform vorliegt, gibt es ein weiteres einfaches mechanisches Verfahren, das Sie anwenden können. Wählen Sie zunächst einen Ausdruck, der die Konjunktion (...∧...) aller Folgerungen jeder Annahme von Td ist. Beweisen Sie dann, dass dieser Ausdruck jede der einzelnen Annahmen von Td impliziert, indem Sie die Konjunktionen eliminieren (es gibt eine Reihe von bewiesenen Behauptungen, um dies zu tun, einschließlich `simpl`, `simpr`, `3simpa`, `3simpb`, `3simpc`, `simp1`, `simp2`, und `simp3`). Wenn der Ausdruck verschachtelte Konjunktionen hat, können die inneren Konjunktionen durch Verkettung der obigen Theoreme mit `syl` herausgebrochen werden (siehe Abschnitt 3.6).[21] Als letzten Schritt können Sie dann die bereits bewiesene Behauptung Td (die in Deduktionsform vorliegt) anwenden und die Behauptung T in geschlossener Form beweisen.

Wir können auch jede Behauptung T in geschlossener Form leicht in die zugehörige Behauptung Ti in Inferenzform umwandeln, indem wir den Modus ponens anwenden (siehe Abschnitt 3.3.1) [22].

Die gemeinsame Prämisse in der Deduktionsform kann auch verwendet werden, um den Kontext darzustellen, der für die Unterstützung von Systemen des natürlichen Schließens notwendig ist. Daher werden wir nun die natürliche Deduktion diskutieren.

3.9.4 Natürliche Deduktion

Systeme des natürlichen Schließens oder der natürlichen Deduktion (ND) als solche wurden ursprünglich 1934 von zwei unabhängig voneinander arbeitenden Logikern eingeführt: Jaśkowski und Gentzen. ND-Systeme sollen auf formal ordentliche Weise traditionelle Methoden des mathematischen Schließens (wie den bedingten Beweis, den indirekten Beweis und den Beweis durch Fallunterscheidung) rekonstruieren. Als Rekonstruktionen wurden sie natürlich durch frühere Arbeiten beeinflusst, und viele spezifische ND-Systeme und Notationen wurden seit ihrem Ursprung entwickelt.

Es gibt viele ND-Varianten, aber Indrzejczak[27, S. 31-32] schlägt vor,

---

um den `T.` →-Präfix erweitert werden, damit Td darauf angewendet werden kann. Siehe zum Beispiel `hadbi123i`.

[21] Es gibt tatsächlich viele Theoreme (mit simp* gekennzeichnet, wie z.B. `simp333`), die innere Konjunktionen in einem Schritt auflösen. Aber anstatt sie alle zu lernen, können Sie einfach die gerade beschriebene Verkettung für den Beweis verwenden, und dann den Metamath-Programmbefehl `minimize_with` die richtigen speziellen simp*-Theoreme herausfinden lassen, die die Verkettungen aufzulösen.

[22] Anm. der Übersetzer: Eine Behauptung T in geschlossener Form kann auch leicht in die zugehörige Behauptung Td in Deduktionsform umgewandelt werden, indem die Prämisse von T durch φ ersetzt wird und für jedes Konjunkt aus der ursprünglichen Prämisse eine Hypothese bestehend aus dem Konjunkt mit vorangestelltem $\varphi \rightarrow$ ergänzt wird.

dass jedes System des natürlichen Schließens zumindest diese drei Kriterien erfüllen muss:

- „Es gibt Möglichkeiten, um Annahmen in einen Beweis einzufügen und auch um sie zu eliminieren. Gewöhnlich bedarf es einiger buchhalterischer Hilfsmittel, um den Gültigkeitsbereich einer Annahme anzugeben und zu zeigen, dass ein Teil eines Beweises, der von einer eliminierten Annahme abhängt, entlastet wird.
- Es gibt keine (oder zumindest eine sehr begrenzte Menge von) Axiomen, weil ihre Rolle von der Menge der primitiven Regeln für die Einführung und Eliminierung logischer Konstanten übernommen wird, was bedeutet, dass elementare Schlussfolgerungen anstelle von Formeln als primitiv angesehen werden.
- (Ein echtes) ND-System erlaubt eine große Freiheit bei der Konstruktion von Beweisen und die Möglichkeit, verschiedene Strategien der Beweissuche anzuwenden, wie den bedingten Beweis, den Beweis durch Fallunterscheidungen, den Beweis durch reductio ad absurdum usw.“

Der Metamath Proof Explorer (MPE), wie er in `set.mm` definiert ist, ist im Grunde ein System im Hilbert-Stil. Das heißt, MPE basiert auf einer größeren Anzahl von Axiomen (im Vergleich zu Systemen der natürlichen Deduktion), einer sehr kleinen Menge von Schlussregeln (Modus ponens), und der Kontext wird nicht durch die Inferenzregeln in der Mitte eines Beweises verändert. Abgesehen davon können MPE-Beweise mit dem Ansatz der natürlichen Deduktion (ND), wie er ursprünglich von Jaśkowski und Gentzen entwickelt wurde, erstellt werden.

Der gebräuchlichste und empfohlene Ansatz für die Anwendung von ND in MPE ist die Verwendung der Deduktionsform und die Anwendung der in MPE bewiesenen Behauptungen, die den ND-Regeln entsprechen. Zum Beispiel ist MPE's `jca` äquivalent zur ND-Regel $\wedge$-I (and-insertion). Wir haben eine Liste von Äquivalenzen erstellt, die Sie einsehen können. Dieser Ansatz für die Anwendung eines ND-Ansatzes innerhalb von MPE stützt sich im Wesentlichen auf Metamaths wff-Metavariablen und wird in der Präsentation „Natural Deductions in the Metamath Proof Language“ von Mario Carneiro [11] näher beschrieben.

In diesem Stil sind viele Schritte eine Implikation, deren Prämissen den Kontext (Γ) der meisten ND-Systeme nachahmt. Um eine Annahme hinzuzufügen, fügen Sie sie einfach der Implikationsprämisse hinzu (typischerweise unter Verwendung von `simpr`) und verwenden diese neue Prämisse für alle späteren Behauptungen im selben Bereich. Wenn Sie eine Behauptung in einem ND-Hypothesenbereich verwenden wollen, der außerhalb des aktuellen ND-Hypothesenbereichs liegt, ändern Sie die Behauptung so, dass die ND-Hypothesenannahme zu ihrer Prämisse hinzugefügt wird (typischerweise mit `adantr`). Die meisten Beweisschritte werden mit Hilfe von Regeln bewiesen, die Hypothesen und Ergebnisse der Form $\varphi \rightarrow ...$ haben.

Ein Beispiel mag dies deutlicher machen. Schauen wir uns Theorem 5.5 von [34, S. 18] zusammen mit einer zeilenweisen Übersetzung unter Verwendung der üblichen Übersetzung der natürlichen Deduktion (ND) in die Metamath Proof Explorer (MPE) Notation an (dies ist Beweis `ex-natded5.5`). Das ursprüngliche Ziel des Beweises war der Beweis von $\neg\psi$ unter zwei Annahmen, $(\psi \to \chi)$ und $\neg\chi$. Wir werden diese Aussagen in die MPE-Deduktionsform übersetzen, indem wir ihnen allen das Präfix $\varphi \to$ voranstellen. In MPE lautet das Ziel also $(\varphi \to \neg\psi)$, und die beiden Hypothesen lauten $(\varphi \to (\psi \to \chi)$ und $(\varphi \to \neg\chi)$.

Die folgende Tabelle zeigt den Beweis im Stil der natürlichen Deduktion von Fitch und seine MPE-Entsprechung. Die Spalte # zeigt die ursprüngliche Nummerierung, *MPE#* zeigt die Nummer im äquivalenten MPE-Beweis (den wir später zeigen werden), *ND-Ausdruck* zeigt die ursprüngliche Beweisbehauptung in ND-Notation, und *MPE-Übersetzung* zeigt ihre Übersetzung in MPE, wie in diesem Abschnitt diskutiert. Die letzten Spalten zeigen die Begründung in ND bzw. MPE.

#	MPE#	ND-Ausdruck	MPE-Übersetzung	ND-Begründung	MPE-Begründung
1	2;3	$(\psi \to \chi)$	$(\varphi \to (\psi \to \chi))$	gegeben	\$e; `adantr` um die ND-Hypothese einzufügen
2	5	$\neg\chi$	$(\varphi \to \neg\chi)$	gegeben	\$e; `adantr` um die ND-Hypothese einzufügen
3	1	... | ψ	$(\varphi \to \psi)$	Annahme der ND-Hypothese	`simpr`
4	4	... χ	$((\varphi \wedge \psi) \to \chi)$	$\to$E 1,3	`mpd` 1,3
5	6	... $\neg\chi$	$((\varphi \wedge \psi) \to \neg\chi)$	IT 2	`adantr` 5
6	7	$\neg\psi$	$(\varphi \to \neg\psi)$	$\wedge$I 3,4,5	`pm2.65da` 4,6

Im Original wurden lateinische Buchstaben verwendet; wir haben sie durch griechische Buchstaben ersetzt, um den Metamath-Namenskonventionen zu folgen und um die Metamath-Übersetzung leichter nachvollziehen zu können. Die zeilenweise Metamath-Übersetzung dieses Ansatzes der natürlichen Deduktion stellt jeder Zeile eine Prämisse mit φ voran und verwendet

die Metamath-Äquivalente der natürlichen Deduktionsregeln. Um eine Annahme hinzuzufügen, wird die Prämisse so modifiziert, dass sie sie enthält (typischerweise durch Verwendung von `adantr`; `simpr` ist nützlich, wenn man eine direkte Abhängigkeit von der neuen Annahme haben möchte, wie hier gezeigt).

In Metamath können wir die beiden gegebenen Aussagen als folgende Hypothesen darstellen:

- ex-natded5.5.1 $\vdash (\varphi \to (\psi \to \chi))$
- ex-natded5.5.2 $\vdash (\varphi \to \neg\chi)$

Hier ist der Beweis in Metamath als zeilenweise Übersetzung:

Step	**Hyp**	**Ref**	**Expression**
1		simpr	$\vdash ((\varphi \wedge \psi) \to \psi)$
2		ex-natded5.5.1	$\vdash (\varphi \to (\psi \to \chi))$
3	2	adantr	$\vdash ((\varphi \wedge \psi) \to (\psi \to \chi))$
4	1, 3	mpd	$\vdash ((\varphi \wedge \psi) \to \chi)$
5		ex-natded5.5.2	$\vdash (\varphi \to \neg\chi)$
6	5	adantr	$\vdash ((\varphi \wedge \psi) \to \neg\chi)$
7	4, 6	pm2.65da	$\vdash (\varphi \to \neg\psi)$

Die direkte Verwendung spezifischer Regeln für die natürliche Deduktion kann zu sehr langen Beweisen führen, und zwar aus genau demselben Grund, aus dem die direkte Verwendung von Axiomen in Beweisen im Hilbert-Stil zu sehr langen Beweisen führen kann. Wenn das Ziel kurze und klare Beweise sind, dann ist es besser, bereits bewiesene Behauptungen in Deduktionsform wiederzuverwenden, als jedes Mal von vorne anzufangen und nur grundlegende natürliche Deduktionsregeln zu verwenden.

3.9.5 Die Stärken unseres Ansatzes

Soweit wir wissen, gibt es in der Literatur weder das Theorem der schwachen Deduktion noch die natürliche Deduktionsmethode von Mario Carneiro. Um eine Annahme in eine Prämisse umzuwandeln, benötigt das in der Literatur übliche „Deduktionstheorem“ eine Metalogik außerhalb der vom Axiomensystem bereitgestellten Begriffe. Stattdessen bevorzugen wir im Allgemeinen die Methode der natürlichen Deduktion von Mario Carneiro, verwenden dann das schwache Deduktionstheorem in Fällen, in denen es schwierig ist, es anzuwenden, und verwenden erst dann das vollständige Standard-Deduktionstheorem als letzten Ausweg.

Das Theorem der schwachen Deduktion erfordert keine zusätzliche Metalogik, sondern wandelt eine Schlussfolgerung direkt in ein Theorem in geschlossener Form um, mit einem strengen Beweis, der nur das Axiomensystem verwendet. Im Gegensatz zum Standard-Deduktionstheorem gibt es

keine implizite externe Rechtfertigung, auf die wir vertrauen müssen, um es anzuwenden.

Die Methode der natürlichen Deduktion von Mario Carneiro erfordert ebenfalls keine neuen metalogischen Begriffe. Sie umgeht die Metalogik des Deduktionstheorems, indem sie den Annahmen und Schlussfolgerungen jeder möglichen Schlussfolgerung von Anfang an eine universelle Prämisse („$\varphi \rightarrow$“) voranstellt.

Wir finden es beeindruckend und befriedigend, dass wir so viel im praktischen Sinne tun können, ohne unser Hilbert-artiges Axiomensystem zu verlassen. Natürlich enthält unsere Axiomatisierung, die in Form von Schemata vorliegt, eine eigene Metalogik, die wir ausnutzen. Aber diese Metalogik ist relativ einfach, und für unsere Alternativen zum Deduktionstheorem verwenden wir in erster Linie nur die direkte Substitution von Ausdrücken für Metavariablen.

3.10 Erforschung der Mengenlehre-Datenbasis

An dieser Stelle möchten Sie vielleicht die Datei `set.mm` genauer studieren. Achten Sie insbesondere auf die Annahmen, die zur Definition von wffs (die oben nicht enthalten sind) benötigt werden, auf die Variablentypen (`$f`-Anweisungen) und auf die eingeführten Definitionen. Beginnen Sie mit einigen einfachen Theoremen der Aussagenlogik und stellen Sie sicher, dass Sie jeden Schritt eines Beweises im Detail verstehen. Sobald Sie die ersten paar Beweise hinter sich gebracht haben und mit der Metamath-Sprache vertraut sind, wird jeder Teil der `set.mm`-Datenbasis Schritt für Schritt genauso einfach zu verstehen sein wie jeder andere Teil - Sie müssen keinen „Quantensprung“ bezüglich der mathematischen Raffinesse durchmachen, um einem tiefgehenden Beweis in der Mengenlehre folgen zu können.

Als Nächstes möchten Sie vielleicht untersuchen, wie Konzepte wie die natürlichen Zahlen definiert und beschrieben werden. Dies geschieht wahrscheinlich am besten in Verbindung mit Standard-Lehrbüchern der Mengenlehre, die Ihnen ein besseres Verständnis vermitteln können. Die Datenbasis `set.mm` bietet Referenzen, mit denen Sie beginnen können. Von dort aus beginnt Ihr Weg zu einem sehr tiefgehenden, rigorosen Verständnis der abstrakten Mathematik.

Das Programm Metamath kann Ihnen dabei helfen, eine Metamath-Datenbasis durchzugehen, sei es um herauszufinden, wie ein bestimmter Schritt in einem Beweis erfolgt, oder nur aus allgemeiner Neugier. Wir werden einige Beispiele für die Befehle durchgehen und dabei die Datenbasis der Mengenlehre `set.mm` verwenden, die mit der Metamath-Software geliefert wird. Diese sollten Ihnen den Einstieg erleichtern. Siehe Kapitel 5 für eine detailliertere Beschreibung der Befehle. Beachten Sie, dass wir die

vollständige Schreibweise aller Befehle angegeben haben, um Mehrdeutigkeiten bei zukünftigen Befehlen zu vermeiden. In der praktischen Arbeit brauchen Sie nur die Zeichen eingeben, die erforderlich sind, um jeden Befehl keyword eindeutig zu identifizieren, oft nur ein oder zwei Zeichen pro Schlüsselwort, und Sie brauchen sie nicht in Großbuchstaben zu schreiben.

Führen Sie zunächst das Programm Metamath wie oben beschrieben aus. Sie sollten die Eingabeaufforderung `MM>` sehen. Lesen Sie die Datei `set.mm` ein:

```
MM> read set.mm
Reading source file "set.mm"... 34554442 bytes
34554442 bytes were read into the source buffer.
The source has 155711 statements; 2254 are $a and 32250 are $p.
No errors were found.  However, proofs were not checked.
Type VERIFY PROOF * if you want to check them.
```

Wie bei den meisten Beispielen in diesem Buch wird das, was Sie sehen werden, leicht von den dargestellten Inhalten abweichen, da wir unsere Datenbasen (einschließlich `set.mm`) ständig verbessern.

Prüfen wir die Integrität der Datenbasis. Dieser Vorgang kann ein oder zwei Minuten dauern, wenn Ihr Computer langsam ist.

```
MM> verify proof *
0 10%  20%  30%  40%  50%  60%  70%  80%  90% 100%
..............................................
All proofs in the database were verified in 2.84 s.
```

Es wurden keine Fehler gemeldet, so dass jeder Beweis korrekt ist.

Sie müssen die Namen (Bezeichnungen) der Theoreme kennen, bevor Sie sich diese ansehen können. Oft ist das Durchsuchen der Datenbasisdatei(en) mit einem Texteditor die beste Vorgehensweise. In `set.mm` gibt es viele detaillierte Kommentare, vor allem am Anfang, die Ihnen helfen können. Der Befehl `search` im Programm Metamath ist ebenfalls sehr nützlich. Die Option `comments` listet die Aussagen auf, deren zugehöriger Kommentar (der unmittelbar vor der Aussage steht) eine von Ihnen angegebene Zeichenfolge enthält. Wenn Sie zum Beispiel Endertons *Elements of Set Theory* studieren [18], möchten Sie vielleicht die Verweise darauf in der Datenbasis finden. Bei der zu suchenden Zeichenfolge `enderton` wird nicht zwischen Groß- und Kleinschreibung unterschieden. (Auf diese Weise werden nicht alle Theoreme in der Datenbasis angezeigt, die in Endertons Buch enthalten sind, da es für ein bestimmtes Theorem, das in mehreren Lehrbüchern vorkommen kann, in der Regel nur ein einziges Zitat gibt.)

```
MM> search * "enderton" / comments
12067 unineq $p "... Exercise 20 of [Enderton] p. 32 and ..."
12459 undif2 $p "...Corollary 6K of [Enderton] p. 144. (C..."
12953 df-tp $a "...s. Definition of [Enderton] p. 19. (Co..."
```

```
13689 unissb $p ".... Exercise 5 of [Enderton] p. 26 and ..."
```

(etc.)

Oder Sie möchten nachsehen, welche Theoreme etwas mit Konjunktionen (logisches UND) zu tun haben. Die Anführungszeichen um den Suchstring sind optional, wenn es keine Mehrdeutigkeit gibt.

```
MM> search * conjunction / comments
120 a1d $p "...be replaced with a conjunction ( ~ df-an )..."
662 df-bi $a "...viated form after conjunction is introdu..."
1319 wa $a "...ff definition to include conjunction ('and')."
1321 df-an $a "Define conjunction (logical 'and'). Defini..."
1420 imnan $p "...tion in terms of conjunction. (Contribu..."
```

(etc.)

Nun werden wir uns mit einigen Details befassen. Schauen wir uns das erste Axiom der Aussagenlogik an (wir könnten `sh st` als Abkürzung für `show statement` verwenden).

```
MM> show statement ax-1/full
Statement 49 is located on line 11182 of the file "set.mm".
Its statement number for HTML pages is 6.
"Axiom _Simp_.  Axiom A1 of [Margaris] p. 49.  One of the 3
axioms of propositional calculus.  The 3 axioms are also
given as Definition 2.1 of [Hamilton] p. 28.
..."
49 ax-1 $a |- ( ph -> ( ps -> ph ) ) $.
Its mandatory hypotheses in RPN order are:
  wph $f wff ph $.
  wps $f wff ps $.
The statement and its hypotheses require the variables:
  ph ps
The variables it contains are:  ph ps
```

Vergleichen Sie dies mit `ax-1` auf Seite 80. Sie sehen, dass zum Beispiel das Symbol `ph` die ASCII-Notation für φ ist. Um die mathematischen Symbole für einen beliebigen Ausdruck zu sehen, können Sie ihn in LaTeX umsetzen (geben Sie `help tex` für Anweisungen dazu ein) oder, was einfacher ist, verwenden Sie einfach einen Texteditor, um sich die Kommentare anzusehen, in denen die Symbole zuerst in `set.mm` eingeführt werden. Die Annahmen `wph` und `wps`, die von `ax-1` verlangt werden, bedeuten, dass die Variablen `ph` und `ps` wffs sein müssen.

Als nächstes wählen wir ein einfaches Theorem der Aussagenlogik, das Gesetz der Selbstimplikation[23], das direkt aus den Axiomen bewiesen wird. Wir werden uns die Aussage und dann ihren Beweis ansehen.

```
MM> show statement id1/full
Statement 116 is located on line 11371 of the file "set.mm".
Its statement number for HTML pages is 22.
"Principle of identity.  Theorem *2.08 of [WhiteheadRussell]
p. 101.  This version is proved directly from the axioms for
demonstration purposes.
..."
116 id1 $p |- ( ph -> ph ) $= ... $.
Its mandatory hypotheses in RPN order are:
  wph $f wff ph $.
Its optional hypotheses are:  wps wch wth wta wet
      wze wsi wrh wmu wla wka
The statement and its hypotheses require the variables:  ph
These additional variables are allowed in its proof:
      ps ch th ta et ze si rh mu la ka
The variables it contains are:  ph
```

Die optionalen Variablen `ps`, `ch`, etc. stehen bei Bedarf zur Verwendung in einem Beweis dieser Aussage zur Verfügung, und wenn diese verwendet würden, würden die entsprechenden optionalen Hypothesen `wps`, `wch`, etc. ebenfalls verwendet werden. (Siehe Abschnitt 4.2.5 für die Bedeutung von „optionaler Hypothese“.) Der Grund dafür, dass diese in der Anzeige der Aussage auftauchen, ist, dass die Aussage `id1` zufällig in ihrem Gültigkeitsbereich liegt (siehe Abschnitt 4.2.8 für die Definition von „Gültigkeitsbereich“), aber tatsächlich werden wir in der Aussagenlogik niemals optionale Hypothesen oder Variablen verwenden. Dies wird wichtig, nachdem Quantoren eingeführt wurden, wo „Dummy“-Variablen oft in der Mitte eines Beweises benötigt werden.

Schauen wir uns den Beweis der Aussage `id1` an. Wir verwenden den Befehl `show proof`, der standardmäßig die „unwesentlichen“ Schritte unterdrückt, die die wffs konstruieren. Wir zeigen den Beweis im „lemmon“-Format an (ein Format ohne Einrückungen mit expliziten Verweisen auf vorherige Schrittnummern) und nummerieren die angezeigten Schritte neu:

```
MM> show proof id1 /lemmon/renumber
1 ax-1           $a |- ( ph -> ( ph -> ph ) )
2 ax-1           $a |- ( ph -> ( ( ph -> ph ) -> ph ) )
3 ax-2           $a |- ( ( ph -> ( ( ph -> ph ) -> ph ) ) ->
```

[23]Anm. der Übersetzer: Im Originaltext wird dieses Gesetz, wie in *Principia Mathematica* [74], „principle of identity“ genannt, was aber oft mit dem „law of identity“ verwechselt wird, das im Deutschen „Identitätsprinzip“ genannt wird.

```
                    ( ( ph -> ( ph -> ph ) ) -> ( ph -> ph )
                                                        ) )
4 2,3 ax-mp      $a |- ( ( ph -> ( ph -> ph ) ) -> ( ph -> ph
                                                        ) )
5 1,4 ax-mp      $a |- ( ph -> ph )
```

Wenn Sie Abschnitt 2.3 gelesen haben, werden Sie wissen, wie Sie diesen Beweis interpretieren können. Schritt 2 zum Beispiel ist eine Anwendung des Axioms `ax-1`. Dieser Beweis ist identisch mit demjenigen in Hamiltons *Logic for Mathematicians* [21, S. 32].

Vielleicht möchten Sie sich ansehen, welche Substitutionen in `ax-1` vorgenommen werden, um zu Schritt 2 zu gelangen. Der entsprechende Befehl muss die „echte“ Schrittnummer kennen, also zeigen wir den Beweis noch einmal ohne die Option `renumber` an.

```
MM> show proof id1 /lemmon
 9 ax-1          $a |- ( ph -> ( ph -> ph ) )
20 ax-1          $a |- ( ph -> ( ( ph -> ph ) -> ph ) )
24 ax-2          $a |- ( ( ph -> ( ( ph -> ph ) -> ph ) ) ->
                     ( ( ph -> ( ph -> ph ) ) -> ( ph -> ph )
                                                        ) )
25 20,24 ax-mp   $a |- ( ( ph -> ( ph -> ph ) ) -> ( ph -> ph
                                                        ) )
26 9,25 ax-mp    $a |- ( ph -> ph )
```

Die „echte“ Nummer des zu betrachtenden Schrittes ist 20. Schauen wir uns die Details an.

```
MM> show proof id1 /detailed_step 20
Proof step 20:  min=ax-1 $a |- ( ph -> ( ( ph -> ph ) -> ph )
  )
This step assigns source "ax-1" ($a) to target "min" ($e).
The source assertion requires the hypotheses "wph" ($f, step
18) and "wps" ($f, step 19).  The parent assertion of the
target hypothesis is "ax-mp" ($a, step 25).
The source assertion before substitution was:
    ax-1 $a |- ( ph -> ( ps -> ph ) )
The following substitutions were made to the source
assertion:
    Variable  Substituted with
     ph        ph
     ps        ( ph -> ph )
The target hypothesis before substitution was:
    min $e |- ph
The following substitution was made to the target hypothesis:
    Variable  Substituted with
```

```
    ph          ( ph -> ( ( ph -> ph ) -> ph ) )
```

Dies zeigt die Substitutionen, die an den Variablen in `ax-1` vorgenommen wurden. Es wird auf die Schritte 18 und 19 verwiesen, die in unserer Beweisdarstellung nicht gezeigt werden. Um diese Schritte zu sehen, können Sie den Beweis mit der Option `all` anzeigen.

Sehen wir uns nun einen etwas fortgeschritteneren Beweis der Aussagenlogik an. Beachten Sie, dass `/\` das Symbol für $\wedge$ (logisches UND, auch Konjunktion genannt) ist.

```
MM> show statement prth/full
Statement 1791 is located on line 15503 of the file "set.mm".
Its statement number for HTML pages is 559.
"Conjoin antecedents and consequents of two premises.  This
is the closed theorem form of ~ anim12d .  Theorem *3.47 of
[WhiteheadRussell] p. 113.  It was proved by Leibniz,
and it evidently pleased him enough to call it
_praeclarum theorema_ (splendid theorem).
..."
1791 prth $p |- ( ( ( ph -> ps ) /\ ( ch -> th ) ) -> ( ( ph
      /\ ch ) -> ( ps /\ th ) ) ) $= ... $.
Its mandatory hypotheses in RPN order are:
  wph $f wff ph $.
  wps $f wff ps $.
  wch $f wff ch $.
  wth $f wff th $.
Its optional hypotheses are:  wta wet wze wsi wrh wmu wla wka
The statement and its hypotheses require the variables:  ph
      ps ch th
These additional variables are allowed in its proof:  ta et
      ze si rh mu la ka
The variables it contains are:  ph ps ch th


MM> show proof prth /lemmon/renumber
1 simpl          $p |- ( ( ( ph -> ps ) /\ ( ch -> th ) ) ->
                                                ( ph -> ps ) )
2 simpr          $p |- ( ( ( ph -> ps ) /\ ( ch -> th ) ) ->
                                                ( ch -> th ) )
3 1,2 anim12d    $p |- ( ( ( ph -> ps ) /\ ( ch -> th ) ) ->
                           ( ( ph /\ ch ) -> ( ps /\ th ) ) )
```

Es gibt Verweise auf eine Reihe unbekannter Aussagen. Um zu sehen, um welche es sich handelt, können Sie Folgendes eingeben:

```
MM> show proof prth /statement_summary
```

```
Summary of statements used in the proof of "prth":

Statement simpl is located on line 14748 of the file
"set.mm".
"Elimination of a conjunct.  Theorem *3.26 (Simp) of
[WhiteheadRussell] p. 112. ..."
  simpl $p |- ( ( ph /\ ps ) -> ph ) $= ... $.

Statement simpr is located on line 14777 of the file
"set.mm".
"Elimination of a conjunct.  Theorem *3.27 (Simp) of
[WhiteheadRussell] ..."
  simpr $p |- ( ( ph /\ ps ) -> ps ) $= ... $.

Statement anim12d is located on line 15445 of the file
"set.mm".
"Conjoin antecedents and consequents in a deduction.
..."
  anim12d.1 $e |- ( ph -> ( ps -> ch ) ) $.
  anim12d.2 $e |- ( ph -> ( th -> ta ) ) $.
  anim12d $p |- ( ph -> ( ( ps /\ th ) -> ( ch /\ ta ) ) )
      $= ... $.
```

(etc.)

Natürlich können Sie jede dieser Aussagen und ihre Beweise und so weiter bis zu den Axiomen der Aussagenlogik zurückverfolgen, wenn Sie möchten.

Der Befehl `search` ist nützlich, um Aussagen zu finden, deren Inhalt Sie ganz oder teilweise kennen. Das folgende Beispiel findet alle Aussagen, die `ph -> ps` gefolgt von `ch -> th` enthalten. Das `$*` ist ein Platzhalter, der auf alles passt; das `$` vor dem `*` verhindert Konflikte mit Token-Namen für mathematische Symbole. Das `*` nach `SEARCH` ist ebenfalls ein Platzhalter, der in diesem Fall „passt auf jede Bezeichnung“ bedeutet.

```
MM> search * "ph -> ps $* ch -> th"
1791 prth $p |- ( ( ( ph -> ps ) /\ ( ch -> th ) ) -> ( ( ph
    /\ ch ) -> ( ps /\ th ) ) )
2455 pm3.48 $p |- ( ( ( ph -> ps ) /\ ( ch -> th ) ) -> ( (
    ph \/ ch ) -> ( ps \/ th ) ) )
117859 pm11.71 $p |- ( ( E. x ph /\ E. y ch ) -> ( ( A. x (
    ph -> ps ) /\ A. y ( ch -> th ) ) <-> A. x A. y ( ( ph /\
    ch ) -> ( ps /\ th ) ) ) )
```

Drei Aussagen, `prth`, `pm3.48` und `pm11.71`, wurden als übereinstimmend befunden.

Um zu sehen, von welchen Axiomen und Definitionen der Beweis von `prth` letztlich abhängt, können Sie das Programm die Hierarchie der Theoreme und Definitionen zurückverfolgen lassen.

```
MM> show trace_back prth /essential/axioms
Statement "prth" assumes the following axioms ($a
statements):
  ax-1 ax-2 ax-3 ax-mp df-bi df-an
```

Beachten Sie, dass die 3 Axiome der Aussagenlogik und der Modus ponens benötigt werden (wie erwartet); außerdem gibt es eine Reihe von Definitionen, die unterwegs verwendet werden. Beachten Sie, dass Metamath keinen Unterschied zwischen Axiomen und Definitionen macht. In `set.mm` wurden sie künstlich unterschieden, indem man ihren Bezeichnungen jeweils `ax-` und `df-` voranstellte. Zum Beispiel definiert `df-an` die Konjunktion (logisch UND), die durch das Symbol `/\` dargestellt wird. In Abschnitt 4.5 wird die Philosophie von Definitionen erörtert, und die Metamath-Sprache verfolgt einen besonders einfachen, konservativen Ansatz, indem sie die `$a`-Anweisungen sowohl für Axiome als auch für Definitionen verwendet.

Sie können das Programm auch berechnen lassen, wie viele Schritte ein Beweis hat, wenn wir ihn bis zu den `$a`-Anweisungen zurückverfolgen würden.

```
MM> show trace_back prth /essential/count_steps
The statement's actual proof has 3 steps.  Backtracking, a
total of 79 different subtheorems are used.  The statement
and subtheorems have a total of 274 actual steps.  If
subtheorems used only once were eliminated, there would be a
total of 38 subtheorems, and the statement and subtheorems
would have a total of 185 steps.  The proof would have 28349
steps if fully expanded back to axiom references.  The
maximum path length is 38.  A longest path is:  prth <-
anim12d <- syl2and <- sylan2d <- ancomsd <- ancom <- pm3.22
<- pm3.21 <- pm3.2 <- ex <- sylbir <- biimpri <- bicomi <-
bicom1 <- bi2 <- dfbi1 <- impbii <- bi3 <- simprim <- impi <-
con1i <- nsyl2 <- mt3d <- con1d <- notnot1 <- con2i <- nsyl3
<- mt2d <- con2d <- notnot2 <- pm2.18d <- pm2.18 <- pm2.21 <-
pm2.21d <- a1d <- syl <- mpd <- a2i <- a2i.1 .
```

Daraus ergibt sich, dass wir 274 Schritte überprüfen müssten, wenn wir den Beweis ausgehend von den Axiomen vollständig verifizieren wollen. Es werden auch einige weitere Statistiken angezeigt. Es gibt einen oder mehrere Pfade zurück zu den Axiomen, die am längsten sind; dieser Befehl sucht einen von ihnen heraus und zeigt ihn an. Die Länge des längsten Pfades kann in gewisser Weise ausdrücken, wie „tief“ das Theorem ist.

Wir könnten uns auch fragen, welche Beweise von dem Theorem `prth` abhängen. Wenn er später nie verwendet wird, könnten wir ihn als überflüssig streichen, wenn er an sich nicht von Interesse ist.

```
MM> show usage prth
Statement "prth" is directly referenced in the proofs of 18
statements:
  mo3 moOLD 2mo 2moOLD euind reuind reuss2 reusv3i opelopabt
  wemaplem2 rexanre rlimcn2 o1of2 o1rlimmul 2sqlem6 spanuni
  heicant pm11.71
```

Somit wird `prth` von 18 Beweisen direkt verwendet. Wir können die Option `/recursive` verwenden, um die indirekte Verwendung einzuschließen:

```
MM> show usage prth /recursive
Statement "prth" directly or indirectly affects the proofs of
24214 statements:
  mo3 mo mo3OLD eu2 moOLD eu2OLD eu3OLD mo4f mo4 eu4 mopick
...
```

3.10.1 Eine Anmerkung zum „kompakten“ Beweisformat

Das Programm Metamath zeigt Beweise in einem „kompakten“ Format an, wenn der Beweis in komprimiertem Format in der Datenbasis gespeichert ist. Dies kann etwas verwirrend sein, wenn man nicht weiß, wie dies zu interpretieren ist. Wenn Sie zum Beispiel den vollständigen Beweis des Theorems `id1` anzeigen lassen, wird er wie folgt beginnen:

```
MM> show proof id1 /lemmon/all
 1 wph           $f wff ph
 2 wph           $f wff ph
 3 wph           $f wff ph
 4 2,3 wi    @4: $a wff ( ph -> ph )
 5 1,4 wi    @5: $a wff ( ph -> ( ph -> ph ) )
 6 @4            $a wff ( ph -> ph )
```

etc.

Schritt 4 wird ein „lokales Label“, `@4`, zugewiesen. Später, bei Schritt 6, wird auf dieses Label `@4` verwiesen, anstatt den expliziten Beweis für diesen Schritt anzuzeigen. Diese Technik macht sich die Tatsache zunutze, dass sich Schritte in einem Beweis häufig wiederholen, insbesondere bei der Konstruktion von wffs. Das kompakte Format reduziert die Anzahl der Schritte in der Darstellung des Beweises und wird von manchen Benutzern bevorzugt.

Wenn Sie das normale Format mit den „wahren“ Schrittzahlen sehen wollen, können Sie folgende Abhilfe verwenden:

```
MM> save proof id1 /normal
The proof of "id1" has been reformatted and saved internally.
Remember to use WRITE SOURCE to save it permanently.
MM> show proof id1 /lemmon/all
 1 wph           $f wff ph
 2 wph           $f wff ph
 3 wph           $f wff ph
 4 2,3 wi        $a wff ( ph -> ph )
 5 1,4 wi        $a wff ( ph -> ( ph -> ph ) )
 6 wph           $f wff ph
 7 wph           $f wff ph
 8 6,7 wi        $a wff ( ph -> ph )
```

etc.

Beachten Sie, dass aus den ursprünglichen 6 Schritten nun 8 Schritte geworden sind. Das Format ist jetzt jedoch dasselbe wie in Kapitel 2 beschrieben.

Kapitel 4

Die Metamath-Sprache

So kann die Mathematik als das Fach definiert werden, in dem wir nie wissen, wovon wir sprechen, noch ob das, was wir sagen, wahr ist.

Bertrand Russell[1]

Das wohl auffälligste Merkmal der Metamath-Sprache ist das fast vollständige Fehlen einer fest verdrahteten Syntax. Metamath versteht keine andere Mathematik oder Logik als die, die für die Konstruktion endlicher Symbolfolgen nach einer kleinen Menge einfacher, eingebauter Regeln erforderlich ist. Die einzige Regel für Beweise ist die Ersetzung einer Variablen durch einen Ausdruck (Symbolfolge) mit einer einfachen Variablenbinschränkung, um Kollisionen zwischen gebundenen Variablen zu verhindern. Die primitiven Konzepte, die in Metamath eingebaut sind, beinhalten die einfache Manipulation von endlichen Objekten (Symbolen), die wir uns als Menschen leicht vergegenwärtigen können und mit denen Computer leicht umgehen können. Sie scheinen so ziemlich die einfachsten Konzepte zu sein, die für die Standardmathematik erforderlich sind.

Dieses Kapitel dient als Nachschlagewerk für die Sprache Metamath. Es behandelt die langwierigen technischen Details der Sprache, von denen Sie einige beim ersten Lesen vielleicht überfliegen möchten. Andererseits sollten Sie den definierten Begriffen in **Fettschrift** große Aufmerksamkeit schenken; sie haben präzise Bedeutungen, die Sie sich für das spätere Verständnis merken sollten. Am besten machen Sie sich zunächst mit den Beispielen in Kapitel 2 vertraut, um eine gewisse Motivation für die Sprache zu erhalten.

Wenn Sie eine gewisse Kenntnis über die Mengenlehre haben, sollten Sie dieses Kapitel in Verbindung mit der formalen mengentheoretischen Beschreibung der Metamath-Sprache im Anhang C studieren.

[1] Frei übersetzt nach [59, S. 84].

Wir werden den Namen „Metamath“ verwenden, um entweder die Metamath-Computersprache oder die mit der Computersprache verbundene Metamath-Software zu bezeichnen. Wir werden nicht zwischen diesen beiden unterscheiden, wenn der Kontext klar ist.

Der nächste Abschnitt enthält die vollständige Spezifikation der Metamath-Sprache. Sie dient als maßgebliche Referenz und stellt die Syntax detailliert genug dar, um einen Parser und einen Beweisverifizierer zu schreiben. Die Spezifikation ist knapp und es ist wahrscheinlich schwer, die Sprache direkt aus ihr heraus zu lernen. Aber wir nehmen sie hier für die ungeduldigen Leute auf, die lieber alles im Voraus sehen wollen, bevor sie sich mit ausführlichen Erklärungen beschäftigen wollen. Spätere Abschnitte erläutern dieses Material und liefern Beispiele. Wir werden die Definitionen in diesen Abschnitten wiederholen, und Sie können den nächsten Abschnitt beim ersten Lesen überspringen und mit Abschnitt 4.2 (S. 130) fortfahren.

4.1 Spezifikation der Metamath-Sprache

Manchmal muss man schwierige Dinge sagen, aber man sollte sie so einfach wie möglich sagen.

G. H. Hardy[2]

4.1.1 Vorbereitungen

Eine Metamath-**Datenbasis** wird aus einer übergeordneten Quelldatei zusammen mit allen Quelldateien aufgebaut, die durch Anweisungen zum Einbinden von Dateien (siehe unten) eingebunden werden. Die einzigen Zeichen, die in einer Metamath-Quelldatei vorkommen dürfen, sind die 94 druckbaren ASCII-Zeichen ohne Whitespace-Zeichen, d.h. Ziffern, Groß- und Kleinbuchstaben, und die folgenden 32 Sonderzeichen:

```
! ’’ # $ % & ’ ( ) * + , - . / :
; < = > ? @ [ \ ] ^ _ ‘ { | } ~
```

plus die folgenden Zeichen, die als „Whitespace“-Zeichen bezeichnet werden: Leerzeichen (ein druckbares Zeichen) und die nicht druckbaren Steuerzeichen Tabulator, Wagenrücklauf, Zeilenvorschub und Seitenvorschub.[3] Wir verwenden die Schriftart `typewriter`, um die druckbaren Zeichen anzuzeigen.

[2] Frei übersetzt nach dem Zitat in [16], S. 273.

[3] Anm. der Übersetzer: Im Deutschen wird ein „Whitespace“-Zeichen manchmal auch als „Leerraum“ oder „Weißraum“oder noch sperriger „Zwischenraumzeichen“ übersetzt. Wir benutzen hier aber durchgängig die auch im Deutschen üblicherweise verwendete englische Bezeichnung „Whitespace“.

Eine Metamath-Datenbasis besteht aus einer Folge von drei Arten von **Token**, die durch **Whitespace** (eine beliebige Folge von einem oder mehreren „Whitespace“-Zeichen) getrennt sind. Die Menge der **Schlüsselwort**-Token ist `$[`, `$]`, `$c`, `$v`, `$f`, `$e`, `$d`, `$a`, `$p`, `$.`, `$=`, `$(`, `$)`, `$[`, und `$]`. Die letzten vier werden als **Hilfs-** oder vorverarbeitende Schlüsselwörter bezeichnet. Ein **Label**-Token besteht aus einer beliebigen Kombination von Buchstaben, Ziffern und den Zeichen Bindestrich, Unterstrich und Punkt. Ein Token für ein **mathematisches Symbol** kann aus einer beliebigen Kombination der 93 druckbaren Standard-ASCII-Zeichen außer Leerzeichen und `$` bestehen. Bei allen Token wird zwischen Groß- und Kleinschreibung unterschieden.

4.1.2 Vorverarbeitung

Mit dem Token `$(` beginnt ein **Kommentar** und `$)` beendet einen Kommentar. Kommentare können jedes der 94 druckbaren Nicht-Whitespace-Zeichen und Whitespace enthalten, mit der Ausnahme, dass sie nicht die 2-Zeichen-Sequenzen `$(` oder `$)` enthalten dürfen (Kommentare lassen sich nicht verschachteln). Kommentare werden beim Parsen ignoriert (wie Whitespace behandelt), z. B. ist `$( $[ $)` ein Kommentar. Siehe S. 159 für die Konventionen zur Darstellung von Kommentaren; diese Konventionen können für die Zwecke des Parsens ignoriert werden.

Eine **Anweisung zum Einbinden von Dateien** besteht aus `$[`, gefolgt von einem Dateinamen, gefolgt von `$]`. Es ist nur im äußersten Bereich zulässig (d. h. nicht zwischen `${` und `$}`) und darf nicht innerhalb einer Aussage stehen (z. B. darf es nicht zwischen dem Label einer `$a`-Anweisung und ihrem abschließenden `$.` vorkommen). Der Dateiname darf weder ein `$` noch ein Whitespace enthalten. Die Datei muss existieren. Die Unterscheidung zwischen Groß- und Kleinschreibung des Namens folgt den Konventionen des Betriebssystems. Der Inhalt der Datei ersetzt die Anweisung zum Einbinden von Dateien beim Einlesen der übergeordneten Datei. Eingebundene Dateien können andere Dateien einbinden. Nur der erste Verweis auf eine bestimmte Datei wird eingebunden; alle späteren Verweise auf dieselbe Datei (ob in der Datei der obersten Ebene oder in eingebundene Dateien) führen dazu, dass die Anweisung zum Einbinden von Dateien ignoriert wird (wie Whitespace behandelt wird). Ein Prüfprogramm kann davon ausgehen, dass Dateinamen bestehend aus unterschiedlichen Zeichenketten sich auf unterschiedliche Dateien beziehen, um spätere Verweise zu ignorieren. Ein Selbstverweis auf eine Datei wird ignoriert, ebenso wie jeder Verweis auf die übergeordnete Datei (um Schleifen zu vermeiden). Eingebundene Dateien dürfen kein `$(` ohne ein passendes `$)`, kein `$[` ohne ein passendes `$]` und keine unvollständigen Aussagen (z. B. ein `$a` ohne ein passendes `$.`) enthalten. Es ist derzeit nicht spezifiziert, ob Pfadreferenzen relativ zum aktuellen Verzeichnis des Prozesses oder zum Verzeichnis, das die Datei enthält, zu verstehen sind, so dass Datenbasen die Verwendung von Pfadnamen-Trennzeichen (z.B. „/“)

in Dateinamen vermeiden sollten.

Wie alle Token müssen die Schlüsselwörter `$(`, `$)`, `$[` und `$]` von Whitespace umgeben sein.

4.1.3 Grundlegende Syntax

Nach der Vorverarbeitung besteht eine Datenbasis aus einer Folge von **Anweisungen**. Dies sind die Gültigkeitsbereichsanweisungen `${` und `$}` sowie die Anweisungen `$c`, `$v`, `$f`, `$e`, `$d`, `$a` und `$p`.

Eine **Gültigkeitsbereichsanweisung** besteht nur aus ihrem Schlüsselwort: `${` oder `$}`. Mit `${` beginnt ein **Block**, und ein zugehöriges `$}` beendet den Block. Jedes `${` muss ein zugehöriges `$char‘}` haben. Rekursiv definiert ist ein Block eine Folge von keinem, einem oder mehreren Token außer `${` und `$}` und möglicherweise anderen enthaltenen Blöcken. Es gibt einen **äußersten Block**, der nicht von `${ $}` eingeklammert ist; das Ende des äußersten Blocks ist das Ende der Datenbasis.

Eine **$v- oder $c-Anweisung** besteht aus dem Schlüsselwort-Token `$v` bzw. `$c`, gefolgt von einem oder mehreren mathematischen Symbolen, gefolgt von dem `$.`-Token. Diese Anweisungen **erklären** die mathematischen Symbole zu **Variablen** bzw. **Konstanten**. Dasselbe mathematische Symbol darf nicht zweimal in ein und derselben `$v`- oder `$c`-Anweisung vorkommen.

Ein mathematisches Symbol wird **aktiv**, sobald es deklariert wird, und bleibt es bis zum Ende des Blocks, in dem es deklariert wird. Eine Variable kann nicht ein zweites Mal deklariert werden, solange sie aktiv ist, aber sie kann erneut deklariert werden (als Variable, aber nicht als Konstante), nachdem sie inaktiv geworden ist. Eine Konstante muss im äußersten Block deklariert werden und darf nicht ein zweites Mal deklariert werden.

Eine **$f-Anweisung** besteht aus einem Label, gefolgt von `$f`, gefolgt von seinem Typcode (eine aktive Konstante), gefolgt von einer aktiven Variablen, gefolgt von dem `$.`-Token. Eine **$e-Anweisung** besteht aus einem Label, gefolgt von `$e`, gefolgt von seinem Typcode (einer aktiven Konstante), gefolgt von keinem, einem oder mehreren aktiven mathematischen Symbolen, gefolgt von dem `$.`-Token. Eine **Hypothese** ist eine `$f`- oder `$e`-Anweisung. Der durch eine `$f`-Anweisung für ein bestimmtes Label deklarierte Typ ist global, auch wenn die Variable es nicht ist (z. B. kann eine Datenbasis nicht `wff P` in einem lokalen Bereich und `class P` in einem anderen haben).

Eine **einfache $d-Anweisung**besteht aus `$d`, gefolgt von zwei verschiedenen aktiven Variablen, gefolgt von dem Token `$.`Eine zusammengesetzte Anweisung besteht aus `$d`, gefolgt von drei oder mehr Variablen (alle unterschiedlich), gefolgt von dem `$.` Token. Die Reihenfolge der Variablen in einer `$d`-Anweisung ist unerheblich. Eine zusammengesetzte `$d`-Anweisung ist äquivalent zu einer Reihe von einfachen `$d`-Anweisungen, eine für jedes mögliche Paar von Variablen, die in der zusammengesetzten `$d`-Anweisung vorkommen. In dieser Spezifikation nehmen wir an, dass alle `$d`-Anweisungen

einfach sind. Eine `$d`-Anweisung wird auch als **disjunkte Variableneinschränkung** bezeichnet.

Eine **$a-Anweisung** besteht aus einem Label, gefolgt von `$a`, gefolgt von seinem Typcode (einer aktiven Konstante), gefolgt von keinem, einem oder mehreren aktiven mathematischen Symbolen, gefolgt von dem `$.` Token. Eine **$p-Anweisung** besteht aus einem Label, gefolgt von `$p`, gefolgt von seinem Typcode (einer aktiven Konstanten), gefolgt von keinem, einem oder mehreren aktiven mathematischen Symbolen, gefolgt von `$=`, gefolgt von einer Folge von Labels, gefolgt von dem `$.` Token. Eine **Behauptung** ist eine `$a`- oder `$p`-Anweisung.

Eine `$f`-, `$e`- oder `$d`-Anweisung ist von der Stelle, an der sie auftritt, bis zum Ende des Blocks, in dem sie auftritt, **aktiv**. Eine `$a`- oder `$p`-Anweisung ist **aktiv** von der Stelle, an der sie auftritt, bis zum Ende der Datenbasis. Es darf nicht zwei aktive `$f`-Anweisungen geben, die die gleiche Variable enthalten. Jede Variable in einer `$e`-, `$a`- oder `$p`-Anweisung muss in einer aktiven `$f`-Anweisung vorkommen.[4]

Jedes Label-Token muss eindeutig sein, und kein Label-Token darf mit einem Token eines mathematischen Symbols übereinstimmen.[5]

Die Menge der **obligatorischen Variablen**, die mit einer Behauptung verbunden ist, ist die (möglicherweise leere) Menge der Variablen in der Behauptung und in allen aktiven `$e`-Anweisungen. Die (möglicherweise leere) Menge der **obligatorischen Hypothesen** ist die Menge aller aktiven `$f`-Anweisungen, die obligatorische Variablen enthalten, zusammen mit allen aktiven `$e`-Anweisungen. Die Menge der **obligatorischen $d-Anweisungen**, die mit einer Behauptung verbunden ist, sind die aktiven `$d`-Anweisungen, deren Variablen beide zu den obligatorischen Variablen der Behauptung gehören.

4.1.4 Beweisverifizierung

Die Folge von Labeln zwischen den Token in einer `$p`-Anweisung ist ein **Beweis**. Jedes Label in einem Beweis muss das Label einer anderen aktiven Aussage als die der `$p`-Anweisung selbst sein; eine Kennzeichnung muss sich also entweder auf eine aktive Hypothese der `$p`-Anweisung oder auf eine frühere Behauptung beziehen.

Ein **Ausdruck** ist eine Folge von mathematischen Symbolen. Eine **Substitutionsabbildung** assoziiert eine Menge von Variablen mit einer Menge von Ausdrücken. Es ist zulässig, dass eine Variable auf einen Ausdruck abgebildet wird, der sie enthält. Eine **Substitution** ist die gleichzeitige Ersetzung aller Variablen in einem oder mehreren Ausdrücken durch die Ausdrücke, denen die Variablen zugeordnet sind.

[4] Diese Anforderung kann den Vereinheitlichungsalgorithmus (Berechnung der Substitutionen), der für die Beweisüberprüfung erforderlich ist, erheblich vereinfachen.

[5] Diese Einschränkung wurde am 24. Juni 2006 hinzugefügt. Sie ist theoretisch nicht notwendig, wird aber eingeführt, um das Schreiben bestimmter Parser zu erleichtern.

Ein Beweis wird in der Reihenfolge seiner Labelfolge gescannt. Wenn sich das Label auf eine aktive Hypothese bezieht, wird der Ausdruck in der Hypothese auf einen Stapel geschoben. Bezieht sich das Label auf eine Behauptung, so muss eine (eindeutige) Substitution vorhanden sein, die, wenn sie an den obligatorischen Hypothesen der referenzierten Behauptung vorgenommen wird, bewirkt, dass diese mit den obersten (d. h. jüngsten) Einträgen des Stapels in der Reihenfolge des Auftretens der obligatorischen Hypothesen übereinstimmen, wobei der oberste Stapel-Eintrag mit der letzten obligatorischen Hypothese der referenzierten Behauptung übereinstimmt. Anschließend werden so viele Stapel-Einträge wie obligatorischen Hypothesen vorhanden sind, vom Stapel entfernt. Die gleiche Ersetzung wird an der referenzierten Behauptung vorgenommen, und das Ergebnis wird auf den Stapel geschoben. Nachdem das letzte Label im Beweis verarbeitet wurde, muss der Stapel einen einzigen Eintrag enthalten, der mit dem Ausdruck in der `$p`-Anweisung übereinstimmt, die den Beweis enthält.

Ein Beweis kann ein `?` anstelle eines Labels enthalten, um einen unbekannten Schritt anzuzeigen (Abschnitt 4.4.6). Ein Beweisverifizierer kann jeden Beweis ignorieren, der `?` enthält, sollte aber den Benutzer warnen, dass der Beweis unvollständig ist.

Ein **komprimierter Beweis** ist eine alternative Beweis-Notation, die im AnhangB beschrieben wird; siehe auch Verweise auf „komprimierte Beweise“ im Index. Komprimierte Beweise sind eine Erweiterung der Metamath-Sprache, die ein vollständiger Beweisverifizierer analysieren und verifizieren können sollte.

Überprüfung von disjunkten Variableneinschränkungen

Jede in einem Beweis vorgenommene Substitution muss wie folgt überprüft werden, um sicherzustellen, dass alle disjunkten Variableneinschränkungen erfüllt sind.

Wenn zwei durch eine Substitution ersetzte Variablen in einer obligatorischen `$d`-Anweisung der referenzierten Behauptung vorhanden sind, müssen die beiden aus der Substitution resultierenden Ausdrücke folgende Bedingungen erfüllen. Erstens dürfen die beiden Ausdrücke keine Variablen gemeinsam haben. Zweitens muss jedes mögliche Paar von Variablen, eine von jedem Ausdruck, in einer aktiven `$d`-Anweisung der `$p`-Anweisung, die den Beweis enthält, vorhanden sein.

Damit ist die Spezifikation der Metamath-Sprache abgeschlossen; siehe Anhang E für ihre Syntax in erweiterter Backus–Naur-Form (EBNF).

4.2 Die grundlegenden Schlüsselwörter

Im folgenden werden die im letzten Abschnitt vorgestellten Konstrukte der Metamath-Sprache genauer erläutert.

Wie die meisten Computersprachen bezieht Metamath seine Eingaben aus einer oder mehreren **-Quelldateien**, die Zeichen im Standardcode ASCII (American Standard Code for Information Interchange) für Computer enthalten. Eine Quelldatei besteht aus einer Reihe von **Tokens**, d.h. Zeichenketten aus druckbaren Zeichen ohne Whitespace (aus dem auf S. 126 gezeigten Satz von 94 Zeichen), die durch **Whitespace** (Leerzeichen, Tabulatoren, Wagenrücklauf, Zeilenvorschub und Seitenvorschub) getrennt sind. Jede Zeichenkette, die nur aus diesen Zeichen besteht, wird wie ein einzelnes Leerzeichen behandelt. Die druckbaren Zeichen ohne Whitespace , die Metamath erkennt, sind die 94 Zeichen auf Standard-ASCII-Tastaturen.

Metamath hat die Möglichkeit, mehrere Dateien zusammenzufügen, um die Eingabe zu bilden (Abschnitt 4.4.4). Wir nennen den Gesamtinhalt aller Dateien, nachdem sie zusammengefügt wurden, eine **database**, um sie von einer einzelnen Quelldatei zu unterscheiden. Die Tokens in einer Datenbasis bestehen aus **Schlüsselwörter**, die in die Sprache eingebaut sind, sowie aus zwei Arten von benutzerdefinierten Tokens, den **Labels** und die **mathematischen Symbole**. (Der Kürze halber werden wir oft einfach **symbol** anstelle von mathematischem Symbol sagen). Die Menge der **grundlegenden Schlüsselwörter** ist `$c`, `$v`, `$e`, `$f`, `$d`, `$a`, `$p`, `$=`, `$.`, `${` und `$}`. Dies ist der vollständige Satz der syntaktischen Elemente der sogenannten **Grundsprache** von Metamath, und mit ihnen können Sie die gesamte Mathematik ausdrücken, so wie es bei der Entwicklung von Metamath beabsichtigt war. Sie sollten sich unbedingt mit ihnen vertraut machen. Tabelle 4.1 listet die grundlegenden Schlüsselwörter zusammen mit einer kurzen Beschreibung ihrer Funktionen auf. Für den Moment wird diese Beschreibung Ihnen nur eine vage Vorstellung von der Bedeutung der Schlüsselwörter geben; später werden wir die Schlüsselwörter im Detail beschreiben.

Tabelle 4.1: Zusammenfassung der grundlegenden Schlüsselwörter von Metamath

Schlüsselwort	*Beschreibung*
`$c`	Konstantensymbol-Deklaration
`$v`	Variablensymbol-Deklaration
`$d`	disjunkte Variableneinschränkung
`$f`	(„Fließende“) Hypothese vom Variablentyp
`$e`	Logische („wesentliche“) Hypothese
`$a`	Axiomatische Behauptung
`$p`	Beweisbare Behauptung
`$=`	Beginn des Beweises in einer `$p`-Anweisung
`$.`	Ende einer der obigen Anweisungen
`${`	Beginn eines Blocks
`$}`	Ende eines Blocks

Es gibt einige zusätzliche Schlüsselwörter, genannt **Hilfsschlüsselwörter**, die hilfreich für das Arbeiten mit Metamath sind. Diese sind Teil der **erweiterten Sprache**. Sie bieten Ihnen die Möglichkeit, Kommentare in eine Metamath-Quelldatei einzufügen und auf andere Quelldateien zu verweisen. Wir werden diese in späteren Abschnitten vorstellen. Tabelle 4.2 fasst sie zusammen, so dass Sie diese beim Durchsehen einiger Quelldateien zum Lernen der grundlegenden Schlüsselwörter erkennen können.

Tabelle 4.2: Hilfsschlüsselwörter in Methamath

Schlüsselwort	*Beschreibung*
`$(`	Beginn eines Kommentars
`$)`	Ende eines Kommentars
`$[`	Anfang des Namens einer eingeschlossenen Quelldatei
`$]`	Ende des Namens einer eingeschlossenen Quelldatei

Im Gegensatz zu den Schlüsselwörtern in einigen Computersprachen handelt es sich bei den Schlüsselwörtern nicht um englische Wörter, sondern um kurze zweistellige Zeichenfolgen. Dadurch sind sie zwar anfangs etwas schwerer zu merken, aber durch ihre Kürze fügen sie sich in die beschriebene Mathematik ein und lenken nicht davon ab, so wie es etwa Satzzeichen tun.

4.2.1 Benutzerdefinierte Tokens

Wie Sie vielleicht bemerkt haben, beginnen alle Schlüsselwörter mit dem Zeichen `$`. Dieses banale Währungssymbol wird normalerweise nicht in der höheren Mathematik verwendet (abgesehen von Förderanträgen), daher haben wir es ausgewählt, um die Metamath-Schlüsselwörter von gewöhnlichen mathematischen Symbolen zu unterscheiden. Das Zeichen `$` wird daher als Sonderzeichen betrachtet und darf nicht als Zeichen in einem benutzerdefinierten Token verwendet werden. Bei allen Tokens und Schlüsselwörtern wird zwischen Groß- und Kleinschreibung unterschieden; so wird beispielsweise `n` als ein anderes Zeichen als `N` betrachtet. Durch die Unterscheidung zwischen Groß- und Kleinschreibung wird der verfügbare ASCII-Zeichensatz so umfangreich wie möglich genutzt.

Mathemathische Symbole

Mathematische Symbole sind Tokens, die zur Darstellung von Symbolen verwendet werden, die in gewöhnlichen mathematischen Formeln vorkommen. Sie können aus einer beliebigen Kombination der 93 druckbaren ASCII-Zeichen (das Leerzeichen ausgeschlossen) außer `$` bestehen. Einige Beispiele sind `x`, `+`, `(`, `|-`, `!%@?&`, und `bounded`. Aus Gründen der Lesbarkeit ist es am besten, wenn diese Symbole den tatsächlichen mathematischen Symbolen

im Rahmen des ASCII-Zeichensatzes so ähnlich wie möglich sind, um die Lesbarkeit der resultierenden mathematischen Ausdrücke zu verbessern.

In der Metamath-Sprache drückt man gewöhnliche mathematische Formeln und Aussagen als Sequenzen von mathematischen Symbolen aus, wie z.B. `2 + 2 = 4` (fünf Symbole, alles Konstanten).[6] Es kann sich sogar um englische oder auch deutsche Sätze handeln, wie in `E ist geschlossen und begrenzt` (fünf Symbole) - hier wäre `E` eine Variable und die anderen vier Konstantensymbole. Im Prinzip könnte eine Metamath-Datenbasis so konstruiert werden, dass sie mit fast jeder eindeutigen mathematischen Aussage in englischer Sprache arbeitet. In der Praxis wären die Definitionen, die erforderlich wären, um alle möglichen Syntax-Variationen zu berücksichtigen, jedoch umständlich und verwirrend und hätten möglicherweise subtile, zufällig eingebaute Tücken. Wir empfehlen generell, mathematische Aussagen, wann immer möglich, mit kompakten mathematischen Standardsymbolen auszudrücken und ihre englischsprachigen Beschreibungen in Kommentare zu setzen. Axiome und Definitionen (`$a`-Anweisungen) sind die einzigen Stellen, an denen Metamath keinen Fehler erkennt, und dies hilft, die Anzahl der benötigten Definitionen zu reduzieren.

Es steht Ihnen frei, beliebige Tokens für mathematische Symbole zu verwenden. Im Anhang A werden Token-Namen für Symbole in der Mengenlehre empfohlen, und wir schlagen vor, dass Sie diese übernehmen, um die Mengenlehre-Datenbasis `set.mm` in Ihre Datenbasis aufnehmen zu können. Für Ausdrucke können Sie die Tokens in einer Datenbasis mit dem Satzprogramm LaTeX in mathematische Standardsymbole umwandeln. Der Metamath-Befehl `open tex` *Dateinamen* erzeugt eine Ausgabe, die von LaTeX. gelesen werden kann. Die Korrespondenz zwischen den Token und den tatsächlichen Symbolen wird durch `latexdef`-Anweisungen innerhalb eines speziellen Datenbasiskommentars hergestellt, der mit `$t` gekennzeichnet ist. Sie können diesen Kommentar bearbeiten, um die Definitionen zu ändern oder neue hinzuzufügen. Im Anhang A wird genauer beschrieben, wie das funktioniert.

Labels

Label-Token werden verwendet, um Metamath-Anweisungen für eine spätere Referenz zu identifizieren[7]. Label-Token dürfen nur Buchstaben, Ziffern und die drei Zeichen Punkt, Bindestrich und Unterstrich enthalten:

```
. - _
```

[6] Um Mehrdeutigkeiten mit anderen Ausdrücken zu vermeiden, wird dies in der Mengenlehre-Datenbasis `set.mm` als `|- ( 2 + 2 ) = 4` ausgedrückt, dessen LaTeX-Äquivalent $\vdash (2 + 2) = 4$ ist. Das $\vdash$ bedeutet „ist ein Satz“ und die Klammern ermöglichen eine explizite assoziative Gruppierung.

[7] Anm. der Übersetzer: deshalb werden „Labels“ im Deutschen oft auch als „Sprungmarken“ bezeichnet.

Ein Label wird deklariert, indem man es unmittelbar vor das Schlüsselwort der Anweisung setzt, die es kennzeichnet. Zum Beispiel könnte das Label `axiom.1` wie folgt deklariert werden:

```
axiom.1 $a |- x = x $.
```

Jede `$e`, `$f`, `$a`, und `$p`-Anweisung in einer Datenbasis muss ein Label für sie deklariert haben. Keine anderen Anweisungstypen dürfen Label-Deklarationen haben. Jedes Label muss eindeutig sein.

Auf ein Label (und die damit bezeichnete Anweisung) wird **referenziert**, indem das Label zwischen den `$=` und `$.` Schlüsselwörtern in eine `$p`-Anweisung aufgenommen wird. Die Folge von Labels zwischen diesen beiden Schlüsselwörtern wird als **Beweis** bezeichnet. Ein Beispiel für eine Anweisung mit einem Beweis, dem wir später begegnen werden (Abschnitt 4.3), ist

```
wnew $p wff ( s -> ( r -> p ) )
     $= ws wr wp w2 w2 $.
```

Sie müssen noch nicht wissen, was das bedeutet, aber Sie sollten wissen, dass das Label `wnew` durch diese `$p`-Anweisung deklariert wird und dass davon ausgegangen wird, dass die Labels `ws`, `wr`, `wp` und `w2` früher in der Datenbasis deklariert wurden und hier referenziert werden.

4.2.2 Konstanten und Variablen

Ein **Ausdruck** ist eine beliebige Folge von mathematischen Symbolen, die auch leer sein kann.

Die grundlegende Metamath-Sprache hat zwei Arten von mathematischen Symbolen: **Konstanten** und **Variablen**. In einem Metamath-Beweis kann eine Konstante nicht durch einen beliebigen Ausdruck ersetzt werden. Eine Variable kann durch einen beliebigen Ausdruck ersetzt werden. Diese Sequenz kann andere Variablen und sogar die zu ersetzende Variable einschließen. Diese Substitution findet statt, wenn Beweise verifiziert werden, und wird in Abschnitt 4.3 beschrieben. Die Anweisung `$f` (später in Abschnitt 4.2.5 beschrieben) wird verwendet, um den **Typ** einer Variablen anzugeben (d.h. um welche Art von Variable es sich handelt) zu spezifizieren und ihr eine Bedeutung zu geben, die typischerweise mit einer „Metavariablen“ assoziiert wird. [8] in der gewöhnlichen Mathematik; eine Variable kann z. B. als wff oder wohlgeformte Formel (in der Logik), als Menge (in der Mengenlehre) oder als nichtnegative ganze Zahl (in der Zahlentheorie) angegeben werden.

[8] Eine Metavariable ist eine Variable, die sich über die syntaktischen Elemente der diskutierten Objektsprache erstreckt; zum Beispiel könnte eine Metavariable eine Variable der Objektsprache und eine andere Metavariable eine Formel in der Objektsprache repräsentieren.

4.2.3 Die $c- und $v-Deklarationsanweisungen

Konstanten werden mit $c-Anweisungen eingeführt oder **deklariert**, und Variablen werden mit $v-Anweisungen deklariert. Eine **einfache** Deklarationsanweisung führt eine einzelne Konstante oder Variable ein. Ihre Syntax ist eine der folgenden:

$c *math-symbol* $.
$v *math-symbol* $.

Die Notation *math-symbol* bedeutet ein beliebiges Token für ein mathematisches Symbol.

Einige Beispiele für einfache Anweisungen sind:

```
$c + $.
$c -> $.
$c ( $.
$v x $.
$v y2 $.
```

Die Zeichen in einem mathematischen Symbol, das deklariert wird, sind für Metamath irrelevant; zum Beispiel könnten wir eine rechte Klammer als Variable deklarieren,

```
$v ) $.
```

obwohl dies unkonventionell wäre.

Die Anweisung **zusammengesetzte** Deklaration ist eine Kurzform für die gleichzeitige Deklaration mehrerer Symbole. Ihre Syntax ist eine der folgenden:

$c *math-symbol* $\cdots$ *math-symbol* $.
$v *math-symbol* $\cdots$ *math-symbol* $.

Hier bedeutet die Ellipse (. . .) eine beliebige Anzahl von *math-symbol*en.

Ein Beispiel für eine zusammengesetzte Deklaration ist:

```
$v x y mu $.
```

Dies entspricht den drei einfachen Deklarationsanweisung

```
$v x $.
$v y $.
$v mu $.
```

Es gibt bestimmte Regeln dafür, wo in der Datenbasis mathematische Symbole deklariert werden dürfen, welche Abschnitte der Datenbasis von ihnen Kenntnis haben (d.h. wo sie „aktiv“ sind) und wann sie mehr als einmal deklariert werden dürfen. Diese werden in Abschnitt 4.2.8 und speziell auf S. 151 diskutiert.

4.2.4 Die `$d`-Anweisung

Die Anweisung `$d` wird als eine **disjunkte Variableneinschränkung** bezeichnet. Die Syntax der **einfachen** Version dieser Anweisung lautet

`$d` *variable variable* `$.`,

wobei jede *variable* eine zuvor deklarierte Variable ist und die beiden *variable*n unterschiedlich sind. (Genauer gesagt muss jede *variable* eine **aktive** Variable sein, was bedeutet, dass es eine vorherige `$v`-Anweisung geben muss, deren **Gültigkeitsbereich** die `$d`-Anweisung einschließt. Diese Begriffe werden definiert, wenn wir in Abschnitt 4.2.8 über Anweisungen mit Gültigkeitsbereich sprechen).

In der gewöhnlichen Mathematik können Formeln auftreten, die wahr sind, wenn die Variablen in ihnen unterschiedlich sind, aber falsch werden, wenn diese Variablen identisch gemacht werden. Zum Beispiel ist die prädikatenlogische Formel $\exists x\, x \neq y$, was bedeutet: „Für ein gegebenes y existiert ein x, das nicht gleich y ist“, in den meisten mathematischen Theorien wahr (nämlich in allen nicht-trivialen Theorien, d.h. solchen mit mehr als einem Individuum, wie z.B. die Arithmetik). Wenn wir jedoch y durch x ersetzen, erhalten wir $\exists x\, x \neq x$, was immer falsch ist, da es bedeutet, dass „etwas existiert, das nicht gleich sich selbst ist.“[9] Mit der Anweisung `$d` können Sie eine Beschränkung angeben, die die Ersetzung einer Variablen durch eine andere verbietet. In diesem Fall würden wir die Anweisung

```
$d x y $.
```

verwenden, um diese Beschränkung zu spezifizieren.

Die Reihenfolge, in der die Variablen in einer `$d`-Anweisung erscheinen, ist nicht wichtig. Wir könnten auch

```
$d y x $.
```

verwenden.

Die Anweisung `$d` ist eigentlich noch allgemeiner, wie das „disjunkt“ in ihrem Namen vermuten lässt. Die volle Bedeutung besteht darin, dass bei einer Substitution der beiden Variablen (im Verlauf eines Beweises, der sich auf eine `$a`- oder `$p`-Anweisung bezieht, die mit der `$d`-Anweisung verknüpft ist), die beiden Ausdrücke, die sich aus der Substitution ergeben, keine gemeinsamen Variablen haben dürfen. Außerdem muss jedes mögliche Paar von Variablen, eine von jedem Ausdruck, in einer `$d`-Anweisung enthalten sein, die mit der zu beweisenden Anweisung verbunden ist. (Diese Anforderung zwingt die zu beweisende Anweisung dazu, die ursprüngliche disjunkte Variableneinschränkung zu „vererben“).

[9] Wenn Sie Logiker sind, werden Sie dies als die unzulässige Substitution einer freien Variablen mit einer gebundenen Variablen erkennen. Metamath macht keinen inhärenten Unterschied zwischen freien und gebundenen Variablen; stattdessen lässt man Metamath wissen, welche Substitutionen zulässig sind, indem man `$d`-Anweisungen in der richtigen Weise in seinem Axiomensystem verwendet.

Nehmen wir zum Beispiel an, dass `u` eine Variable ist. Wenn die Beschränkung

```
$d A B $.
```

für ein Theorem angegeben wurde, auf das in einem Beweis verwendet wird, dürfen wir nicht `A` durch `a + u` und `B` durch `b + u` ersetzen, da diese beiden Symbolfolgen die Variable `u` gemeinsam haben. Außerdem, wenn `a` und `b` Variablen sind, können wir nicht `A` durch `a` und `B` durch `b` ersetzen, es sei denn, wir haben auch `$d a b` für das zu beweisende Theorem angegeben. Mit anderen Worten, die Eigenschaft von `$d`, die mit einem Paar von Variablen verbunden ist, muss nach der Substitution wirksam erhalten bleiben.

Die `$d`-Anweisung bedeutet *nicht*, dass die beiden Variablen nicht durch dasselbe ersetzt werden dürfen, wie man zunächst denken könnte. Wenn man beispielsweise `A` und `B` im obigen Beispiel durch identische Symbolsequenzen ersetzt, die nur aus Konstanten bestehen, entsteht kein disjunkter Variablenkonflikt, da die beiden Symbolsequenzen keine Variablen gemeinsam haben (da sie überhaupt keine Variablen haben). Ebenso tritt kein Konflikt auf, wenn die beiden Variablen in einer Anweisung `$d` durch die leere Symbolfolge ersetzt werden.

Die `$d`-Anweisung hat keine direkte Entsprechung in der gewöhnlichen Mathematik: zum Teil deshalb, weil die Variablen von Metamath nicht wirklich dasselbe sind wie die Variablen der gewöhnlichen Mathematik, sondern vielmehr Metavariablen, die sich über gewöhnliche Variablen erstrecken (sowie über andere Arten von Symbolen und Gruppen von Symbolen). Je nach Situation können wir die Anweisung `$d` informell auf verschiedene Weise interpretieren. Nehmen wir zum Beispiel an, dass `x` und `y` Variablen sind, die sich über Zahlen erstrecken (genauer gesagt, dass `x` und `y` Metavariablen sind, die sich über Variablen erstrecken, die beliebige Zahlen repräsentieren), und dass `ph` (φ) und `ps` (ψ) Variablen (genauer gesagt, Metavariablen) sind, die sich über Formeln erstrecken. Wir können die folgenden Interpretationen durchführen, die der informellen Sprache der gewöhnlichen Mathematik entsprechen:

`$d x y $.` bedeutet „unter der Voraussetzung, dass x und y unterschiedliche Variablen sind.“

`$d x ph $.` bedeutet „unter der Voraussetzung, dass x nicht in φ vorkommt.“

`$d ph ps $.` bedeutet „unter der Voraussetzung, dass φ und ψ keine gemeinsamen Variablen haben.“

Zusammengesetzte `$d` Anweisungen

Die **zusammengesetzte** Version der `$d`-Anweisung ist eine Kurzform für die Angabe mehrerer Variablen, deren Ersetzungen paarweise disjunkt sein müssen. Ihre Syntax lautet:

`$d` *variable* ⋯ *variable* `$.`

Hier steht *variable* für das Token einer zuvor deklarierten Variable (genauer gesagt, einer aktiven Variable) und alle *variable*n sind unterschiedlich. Die zusammengesetzte `$d`-Anweisung wird von Metamath intern in eine einfache `$d`-Anweisung für jedes mögliche Paar von Variablen in der ursprünglichen `$d`-Anweisung zerlegt. Zum Beispiel ist

```
$d w x y z $.
```

gleichbedeutend mit

```
$d w x $.
$d w y $.
$d w z $.
$d x y $.
$d x z $.
$d y z $.
```

Zwei oder mehr einfache `$d`-Anweisungen, die das gleiche Variablenpaar angeben, werden intern zu einer einzigen `$d`-Anweisung zusammengefasst. So ist die Menge der drei Anweisungen

```
$d x y $. $d x y $. $d y x $.
```

gleichbedeutend mit

```
$d x y $.
```

In ähnlicher Weise werden bei zusammengesetzten `$d`-Anweisungen nach der internen Zerlegung die gemeinsamen Variablenpaare kombiniert. Zum Beispiel ist die Menge der Anweisungen

```
$d x y A $. $d x y B $.
```

gleichbedeutend mit

```
$d x y $. $d x A $. $d y A $. $d x y $. $d x B $. $d y B $.
```

welches gleichbedeutend ist mit

```
$d x y $. $d x A $. $d y A $. $d x B $. $d y B $.
```

Metamath verifiziert beim Verifizieen eines Beweises automatisch, dass alle `$d`-Beschränkungen erfüllt sind. `$d`-Anweisungen werden in Beweisen nie direkt referenziert (deshalb haben sie keine Labels), aber Metamath weiß immer, welche erfüllt sein müssen (d.h. aktiv sind) und meldet einen Fehler, wenn eine Verletzung auftritt.

Zur Veranschaulichung, wie Metamath eine fehlende `$d`-Anweisung erkennt, betrachten wir das folgende Beispiel aus der Datenbasis `set.mm`.

```
$d x z $.  $d y z $.
$( Theorem to add distinct quantifier to atomic formula. $)
ax5eq $p |- ( x = y -> A. z x = y ) $=...
```

Diese Aussage benötigt offensichtlich die Bedingung, dass z von x verschieden sein muss in dem Theorem `ax5eq`, das $x = y \to \forall z\, x = y$ besagt (das ist offensichtlich, wenn man Logiker ist, denn sonst könnte man $x = y \to \forall x\, x = y$ folgern, was falsch ist, wenn die freien Variablen x und y gleich sind).

Schauen wir uns an, was passiert, wenn wir in der Datenbasis diese `$d`-Anweisung auszukommentieren.

```
$( $d x z $. $) $d y z $.
$( Theorem to add distinct quantifier to atomic formula. $)
ax5eq $p |- ( x = y -> A. z x = y ) $=...
```

Wenn Metamath versucht, den Beweis zu verifizieren, wird es Ihnen sagen, dass `x` und `z` disjunkt sein müssen, weil einer seiner Schritte auf ein Axiom oder Theorem verweist, das diese Bedingung erfüllt.

```
MM> verify proof ax5eq
ax5eq ?Error at statement 1918, label "ax5eq", type "$p":
      vz wal wi vx vy vz ax-13 vx vy weq vz vx ax-c16 vx vy
                                               ^^^^^
There is a disjoint variable ($d) violation at proof step 29.
Assertion "ax-c16" requires that variables "x" and "y" be
disjoint.  But "x" was substituted with "z" and "y" was
substituted with "x".  The assertion being proved, "ax5eq",
does not require that variables "z" and "x" be disjoint.
```

Wir können die Ersetzungen in `ax-c16` mit dem folgenden Befehl sehen.

```
MM> show proof ax5eq / detailed_step 29
Proof step 29:  pm2.61dd.2=ax-c16 $a |- ( A. z z = x -> ( x =
  y -> A. z x = y ) )
This step assigns source "ax-c16" ($a) to target "pm2.61dd.2"
($e).  The source assertion requires the hypotheses "wph"
($f, step 26), "vx" ($f, step 27), and "vy" ($f, step 28).
The parent assertion of the target hypothesis is "pm2.61dd"
($p, step 36).
The source assertion before substitution was:
    ax-c16 $a |- ( A. x x = y -> ( ph -> A. x ph ) )
The following substitutions were made to the source
assertion:
    Variable  Substituted with
     x         z
```

```
     y         x
     ph        x = y
The target hypothesis before substitution was:
    pm2.61dd.2 $e |- ( ph -> ch )
The following substitutions were made to the target
hypothesis:
    Variable  Substituted with
     ph        A. z z = x
     ch        ( x = y -> A. z x = y )
```

Die disjunkten Variableneinschränkungen von `ax-c16` können mit dem Befehl `show statement` ermittelt werden. Die Zeile, die mit „`Its mandatory disjoint variable pairs are:...`“ beginnt, listet alle Variablenpaare in spitzen Klammern auf.

```
MM> show statement ax-c16/full
Statement 3033 is located on line 9338 of the file "set.mm".
"Axiom of Distinct Variables. ..."
  ax-c16 $a |- ( A. x x = y -> ( ph -> A. x ph ) ) $.
Its mandatory hypotheses in RPN order are:
  wph $f wff ph $.
  vx $f setvar x $.
  vy $f setvar y $.
Its mandatory disjoint variable pairs are:  <x,y>
The statement and its hypotheses require the variables:  x y
      ph
The variables it contains are:  x y ph
```

Da Metamath immer erkennt, wenn `$d`-Anweisungen für einen Beweis benötigt werden, brauchen Sie sich keine Sorgen zu machen, dass Sie vergessen haben, eine entsprechende Anweisung einzufügen; sie kann immer hinzugefügt werden, wenn Sie die obige Fehlermeldung sehen. Wenn Sie unnötige `$d`-Anweisungen einfügen, kann es schlimmstenfalls passieren, dass Ihr Theorem nicht so allgemein ist wie es sein könnte, und dies kann seine spätere Verwendung einschränken.

Andererseits muss man bei der Einführung von Axiomen (`$a`-Anweisungen) sehr vorsichtig sein, um die notwendigen zugehörigen `$d`-Anweisungen richtig anzugeben, da Metamath keine Möglichkeit hat zu prüfen, ob die Axiome korrekt sind. Metamath hätte zum Beispiel keine Kenntnis darüber, dass `ax-c16`, das wir als ein Axiom der Logik betrachten, zu Widersprüchen führen würde, wenn wir seine zugehörige `$d`-Anweisung weglassen.

Sie fragen sich vielleicht, ob es möglich ist, Standardmathematik in der Metamath-Sprache ohne die Anweisung `$d` zu entwickeln, da sie wie ein Ärgernis erscheint, das die Überprüfung von Beweisen erschwert. Die Anweisung `$d` wird in bestimmten Teilbereichen der Mathematik wie der Aussagenlogik nicht benötigt. Dummy-Variablen und die damit verbundenen

`$d`-Anweisungen lassen sich jedoch in Beweisen in der Standardlogik erster Ordnung sowie in der in `set.mm` verwendeten Variante nicht vermeiden. Tatsächlich gibt es keine Obergrenze für die Anzahl der Dummy-Variablen, die in einem Beweis eines Satzes der Logik erster Ordnung mit 3 oder mehr Variablen benötigt werden, wie H. Andreka [50] gezeigt hat. Ein System erster Ordnung, das sie vollständig vermeidet, findet sich in [41]; der Trick dort besteht einfach darin, die notwendigen Dummy-Variablen bedenkenlos in ein zu beweisendes Theorem einzubetten, so dass sie nicht mehr „dummy“ sind, und dann das resultierende längere Theorem so zu interpretieren, dass die eingebetteten Dummy-Variablen ignoriert werden. Falls Sie das interessiert, das System in `set.mm`, das sich aus `ax-1` bis `ax-c14` in `set.mm` ergibt, und das Entfernen von `ax-c16` und `ax-5` erfordert keine `$d`-Anweisungen, ist aber logisch vollständig in dem in [41] beschriebenen Sinne. Das bedeutet, dass damit jeder Satz der Logik erster Ordnung bewiesen werden kann, solange wir dem Satz eine Prämisse hinzufügen, das Dummy- und alle anderen Variablen, die unterschiedlich sein müssen, einschließt. In ähnlicher Weise können Axiome für die Mengenlehre entwickelt werden, die keine disjunkten Variableneinschränkungen erfordern, wie unter `http://us.metamath.org/mpeuni/mmzfcnd.html` erläutert. Zusammen ermöglichen diese im Prinzip die Entwicklung der gesamten Mathematik unter Metamath ohne eine `$d`-Anweisung, während die Länge der resultierenden Theoreme wachsen würde, je mehr Dummy-Variablen in ihren Beweisen benötigt werden.

4.2.5 Die `$f`- und `$e`-Anweisungen

Metamath kennt zwei Arten von Hypothesen, die `$f`- oder **Variablentyp**-Hypothese und die `$e`- oder **logische** Hypothese. [10] Die Buchstaben `f` und `e` stehen für „fließend“ (was in etwa bedeutet, dass sie nur verwendet wird, wenn sie relevant ist) bzw. „essentiell“ (was bedeutet, dass sie immer verwendet wird), aus Gründen, die deutlich werden, wenn wir Frames in Abschnitt 4.2.7 und Gültigkeitsbereiche in Abschnitt 4.2.8 besprechen. Die Syntax dieser Anweisungen ist wie folgt:

label `$f` *typecode variable* `$.`
label `$e` *typecode math-symbol* $\cdots$ *math-symbol* `$.`

Eine Hypothese muss ein *Label* haben. Der Ausdruck in einer `$e`-Hypothese besteht aus einem Typcode (einem aktiven konstanten mathematischen Symbol), gefolgt von einer Folge von keinem, einem oder mehreren mathematischen Symbolen. Jedes mathematische Symbol (einschließlich *constant* und

[10] Streng genommen handelt es sich bei der `$d`-Anweisung auch um eine Hypothese, aber sie wird in einem Beweis nie direkt referenziert, weshalb wir sie eher als Beschränkung denn als Hypothese bezeichnen, um Verwirrung zu vermeiden. Die Überprüfung auf Verletzungen von `$d`-Restriktionen erfolgt automatisch und ist in den Algorithmus zur Überprüfung von Beweisen in Metamath integriert.

variable) muss eine zuvor deklarierte Konstante oder Variable sein. (Außerdem muss jedes mathematische Symbol aktiv sein, was bei der Besprechung von Gültigkeitsbereichsanweisungen in Abschnitt 4.2.8 behandelt wird). Eine `$f`-Hypothese wird verwendet, um die Art oder den **Typ** einer Variablen zu spezifizieren (wie z.B. „sei x eine ganze Zahl“), und eine `$e`-Hypothese wird verwendet, um eine logische Wahrheit auszudrücken (wie z.B. „angenommen x ist prim“), die festgelegt sein muss, damit eine Behauptung, die sie erfordert, auch wahr ist.

Der Typ einer Variablen muss in einer `$f`-Anweisung angegeben sein, bevor sie in einer `$e`-, `$a`- oder `$p`-Anweisung verwendet werden kann. Es darf nur eine (aktive) `$f`-Anweisung für eine bestimmte Variable geben. („Aktiv“ ist in Abschnitt 4.2.8 definiert.)

In der gewöhnlichen Mathematik werden Theoreme oft in der Form „Annahme P; dann Q“ ausgedrückt, wobei Q eine Aussage ist, die man ableiten kann, wenn man von der Aussage P ausgeht. [11] In der Metamath-Sprache würde man die mathematische Aussage P als Hypothese (in diesem Fall eine `$e`-Anweisung in der Metamath-Sprache) und die Aussage Q als beweisbare Behauptung (eine `$p`-Anweisung in der Metamath-Sprache) ausdrücken.

Hier einige Beispiele für Hypothesen, denen Sie in der Logik und Mengenlehre begegnen könnten:

```
stmt1 $f wff P $.
stmt2 $f setvar x $.
stmt3 $e |- ( P -> Q ) $.
```

Informell würde dies lauten: „Angenommen P sei eine wohlgeformte Formel“, „Angenommen x sei eine (individuelle) Variable“ und „Angenommen, wir haben $P \rightarrow Q$ bewiesen.“ Das Drehkreuz-Symbol $\vdash$ wird in Logik-Texten häufig verwendet, um anzuzeigen, dass „ein Beweis existiert für“.

Zusammengefasst:

- Eine `$f`-Hypothese teilt Metamath den Typ oder die Art seiner Variablen mit. Sie ist vergleichbar mit einer Variablendeklaration in einer Computersprache, die dem Compiler mitteilt, dass eine Variable eine Ganzzahl oder eine Gleitkommazahl ist.

- Die `$e`-Hypothese entspricht dem, was man in der gewöhnlichen Mathematik eine „Hypothese“ nennt.

[11] Eine stärkere Version eines Theorems wie dieses wäre die *einzelne* Formel $P \rightarrow Q$ (P impliziert Q), aus der die schwächere Version oben durch den Modus ponens in der Logik folgt. Wir diskutieren diese stärkere Form hier nicht. In der schwächeren Form sagen wir nur, dass, wenn wir P beweisen können, wir auch Q beweisen können. Wenn x die einzige freie Variable in P und Q ist, ist in der Sprache der Logiker die stärkere Form äquivalent zu $\forall x(P \rightarrow Q)$ (für alle x impliziert P Q), während die schwächere Form äquivalent zu $\forall x P \rightarrow \forall x Q$ ist. Die stärkere Form impliziert die schwächere, aber nicht umgekehrt. Um genau zu sein, wird die schwächere Form des Satzes richtigerweise als „Inferenz“ und nicht als Satz bezeichnet.

Bevor eine Behauptung (`$a`- oder `$p`-Aussage) in einem Beweis referenziert werden kann, müssen alle zugehörigen `$f`- und `$e`-Hypothesen (d.h. jene `$e`-Hypothesen, die aktiv sind) erfüllt sein (d.h. durch den Beweis nachgewiesen werden). Die Bedeutung des Begriffs „assoziiert“ (den wir in Abschnitt 4.2.7 als **obligatorisch** bezeichnen werden) wird klar, wenn wir später über Gültigkeitsbereiche sprechen.

Beachten Sie, dass nach jedem `$f`-, `$e`-, `$a`- oder `$p`-Token ein *typecode* stehen muss. Der Typcode ist eine Konstante, die verwendet wird, um die Typen von Ausdrücken festzulegen. Dies wird deutlicher, wenn wir mehr über Behauptungen (`$a`- und `$p`-Anweisungen) erfahren. Ein Beispiel kann auch ihren Zweck verdeutlichen. In der Datenbasis `set.mm` werden die folgenden Typcodes verwendet:

- `wff` : Symbol einer wohlgeformten Formel (wff) (sprich: „die folgende Symbolfolge ist eine wff“).
- $\vdash$: Drehkreuz (sprich: „die folgende Symbolfolge ist beweisbar“ oder „ein Beweis existiert für“).
- `setvar` : Typ der individuellen Mengenvariable (sprich: „das Folgende ist eine individuelle Mengenvariable“). Beachten Sie, dass dies *nicht* der Typ eines beliebigen Mengenausdrucks ist, sondern dazu dient, sicherzustellen, dass nur ein einziges Symbol nach Quantoren wie „für alle“ ($\forall$) und „Es gibt ein“ ($\exists$) verwendet wird.
- `class` : Ein Ausdruck, der ein syntaktisch gültiger Klassenausdruck ist. Alle gültigen Mengenausdrücke sind auch gültige Klassenausdrücke, daher haben Mengenausdrücke normalerweise den Typcode `class`. Verwenden Sie den Typcode `class` und *nicht* den `setvar`-Typcode für den Typ von Mengenausdrücken, es sei denn, Sie wollen gezielt eine einzelne Mengenvariable bestimmen.

4.2.6 Behauptungen (`$a`- und `$p`-Anweisungen)

Es gibt zwei Arten von Behauptungen, `$a`-Anweisungen (**axiomatische Behauptungen**) und `$p`-Anweisungen (**beweisbare Behauptungen**). Ihre Syntax lautet wie folgt:

label `$a` *typecode math-symbol ... math-symbol* `$.`
label `$p` *typecode math-symbol ... math-symbol* `$=` *proof* `$.`

Eine Behauptung erfordert immer ein *Label*. Der Ausdruck in einer Behauptung besteht aus einem Typcode (einer aktiven Konstante), gefolgt von einer Folge von keinem, einem oder mehreren mathematischen Symbolen. Jedes mathematische Symbol, einschließlich aller *Konstanten*, muss eine zuvor deklarierte Konstante oder Variable sein. (Außerdem muss jedes mathematische Symbol aktiv sein, was bei der Besprechung von Gültigkeitsbereichsanweisungen in Abschnitt 4.2.8 behandelt wird).

Eine `$a`-Anweisung ist in der Regel eine syntaktische Definition (z. B. wenn P und Q wffs sind, dann ist $(P \rightarrow Q)$ auch eine wwf), ein Axiom der gewöhnlichen Mathematik (z. B. $x = x$) oder eine Definition der gewöhnlichen Mathematik (z. B. $x \neq y$ bedeutet $\neg x = y$). Eine `$p`-Anweisung ist eine Behauptung, dass eine bestimmte Kombination von mathematischen Symbolen aus früheren Behauptungen folgt, und wird durch einen Beweis ergänzt, der dies zeigt.

Behauptungen können auch in (späteren) Beweisen referenziert werden, um neue Behauptungen aus ihnen abzuleiten. Das Label einer Behauptung wird verwendet, um in einem Beweis auf sie zu verweisen. In Abschnitt 4.3 wird der Beweis im Detail beschrieben.

Behauptungen sind auch das wichtigste Mittel, um Menschen die mathematischen Ergebnisse in der Datenbasis mitzuteilen. Beweise (wenn sie in geeigneter Weise angezeigt werden) vermitteln Menschen, wie die Ergebnisse zustande gekommen sind.

Die `$a`-Anweisung

Axiomatische Behauptungen (`$a`-Anweisungen) stellen die Ausgangspunkte dar, von denen andere Behauptungen (`$p`-Anweisungen) abgeleitet werden. Ihr offensichtlichster Verwendungszweck ist die Spezifizierung gewöhnlicher mathematischer Axiome, aber sie werden auch für zwei andere Zwecke verwendet.

Zunächst muss Metamath die Syntax der Symbolfolgen kennen, die gültige mathematische Aussagen darstellen. Ein Metamath-Beweis muss viel detaillierter aufgeschlüsselt werden als gewöhnliche mathematische Beweise, an die Sie vielleicht gewöhnt sind (sogar die „vollständigen“ Beweise der formalen Logik). Dies ist einer der Faktoren, die Metamath zu einer Allzwecksprache machen, die unabhängig von jedem Logiksystem oder sogar irgendeiner Syntax ist. Wenn Sie einen Substitution für eine Behauptung als Schritt in einem Beweis durchführen wollen, müssen Sie zuerst beweisen, dass die Substitution syntaktisch korrekt ist (oder, wenn Sie es vorziehen, müssen Sie sie „konstruieren“), indem Sie zum Beispiel zeigen, dass der Ausdruck, den Sie für eine wff-Metavariable ersetzen, eine gültige wff ist. Die Anweisung `$a` wird verwendet, um die Kombinationen von Symbolen anzugeben, die als syntaktisch gültig angesehen werden, wie z.B. die legalen Formen von wffs.

Zweitens werden `$a`-Anweisungen verwendet, um das zu spezifizieren, was man sich normalerweise als Definitionen vorstellt, d.h. neue Kombinationen von Symbolen, die andere Kombinationen von Symbolen abkürzen. Metamath unterscheidet nicht zwischen Axiomen und Definitionen. In der Tat wurde argumentiert, dass eine solche Unterscheidung nicht einmal in der gewöhnlichen Mathematik gemacht werden sollte; siehe Abschnitt 4.5, der die Philosophie der Definitionen diskutiert. In Abschnitt 3.4 werden einige technische Anforderungen an Definitionen erörtert. In `set.mm` übernehmen

wir die Konvention, den Labels für Axiome `ax-` und den Labels für Definitionen `df-` voranzustellen.

Die Ergebnisse, die mit der Metamath-Sprache abgeleitet werden können, sind nur so gut wie die `$a`-Anweisungen, die als deren Ausgangspunkt verwendet werden. Wir können dies nicht genug betonen. Metamath wird Sie zum Beispiel nicht daran hindern, $x \neq x$ als ein Axiom der Logik anzugeben. Es ist wichtig, dass Sie alle `$a`-Anweisungen mit großer Sorgfalt prüfen. Da sie eine Quelle potenzieller Fallstricke sind, ist es am besten, keine neuen (normalerweise neue Definitionen) beiläufig hinzuzufügen; vielmehr sollten Sie die Notwendigkeit und die Vorteile jeder einzelnen sorgfältig bewerten.

Wenn Sie alle grundlegenden Axiome und Regeln einer mathematischen Theorie aufgestellt haben, sind die einzigen `$a`-Anweisungen, die Sie hinzufügen werden, die so genannten Definitionen. Im Prinzip sollten Definitionen in gewissem Sinne aus der Sprache einer Theorie eliminierbar sein, und zwar gemäß einer bestimmten Konvention (die normalerweise logische Äquivalenz oder Gleichheit beinhaltet). Die gebräuchlichste Konvention ist, dass jede Formel, die vor der Einführung der Definition syntaktisch gültig, aber nicht beweisbar war, nach der Einführung der Definition nicht beweisbar wird. In einer idealen Welt sollten Definitionen überhaupt nicht vorhanden sein, wenn man absolutes Vertrauen in ein mathematisches Ergebnis haben will. Sie sind jedoch notwendig, um die Mathematik praktikabel zu machen, da die resultierenden Formeln sonst extrem lang und unverständlich wären. Da es die Natur von Definitionen (im allgemeinsten Sinne) nicht gestattet, dass sie automatisch als „zulässig“ überprüft werden können, ist das Urteil des Mathematikers erforderlich, um dies sicherzustellen. (In `set.mm` wurden Anstrengungen unternommen, um fast alle Definitionen direkt eliminierbar zu machen und so die Notwendigkeit eines solchen Urteils zu minimieren.)

Wenn Sie kein Mathematiker sind, ist es vielleicht am besten, keine `$a`-Anweisungen hinzuzufügen oder zu ändern, sondern stattdessen die mathematische Sprache zu verwenden, die bereits in Standarddatenbasen zur Verfügung gestellt wird. Auf diese Weise wird Metamath nicht zulassen, dass Sie einen Fehler machen (d.h. ein falsches Ergebnis beweisen).

4.2.7 Frames

Wir führen nun das Konzept einer Sammlung zusammengehöriger Metamath-Anweisungen ein, die als Frame bezeichnet wird. Jede Behauptung (`$a-` oder `$p`-Anweisung) in der Datenbasis hat einen zugehörigen Frame.

Ein **Frame** ist eine Folge von `$d-`, `$f-` und `$e`-Anweisungen (keine, eine oder mehrere von jeder), gefolgt von einer `$a-` oder `$p`-Anweisung, vorbehaltlich bestimmter, im Folgenden beschriebenen Bedingungen. Der Einfachheit halber nehmen wir an, dass alle verwendeten Token, die mathematischen Symbole darstellen, am Anfang der Datenbasis mit den Anweisungen `$c` und `$v` deklariert werden (die eigentlich nicht Teil eines Frames sind). Der Einfachheit halber nehmen wir auch an, dass es nur einfache `$d`-Anweisungen

gibt (solche mit nur zwei Variablen) und stellen uns alle zusammengesetzten `$d`-Anweisungen (solche mit mehr als zwei Variablen) als in einfache zerlegt vor.

Ein Frame fasst die Hypothesen (und `$d`-Anweisungen) zusammen, die für eine Behauptung (`$a`- oder `$p`-Anweisung) relevant sind. Die Anweisungen in einem Frame können in einer Datenbasis physisch benachbart sein, müssen es aber nicht; wir werden dies in unserer Diskussion über Gültigkeitsbereichsanweisungen in Abschnitt 4.2.8 behandeln.

Ein Frame hat die folgenden Eigenschaften:

1. Die Menge der in den Anweisungen von `$f` enthaltenen Variablen muss mit der Menge der in seinen `$e`-, `$a`- und/oder `$p`-Anweisungen enthaltenen Variablen identisch sein. Mit anderen Worten: Für jede Variable in einer `$e`-, `$a`- oder `$p`-Anweisung muss in einer `$f`-Anweisung ein zugehöriger „Variablentyp“ definiert sein.

2. Keine zwei `$f`-Anweisungen dürfen die gleiche Variable enthalten.

3. Jede `$f`-Anweisung muss vor einer `$e`-Anweisung stehen, in der ihre Variable vorkommt.

Die erste Eigenschaft bestimmt die Menge der in einem Frame vorkommenden Variablen. Dies sind die **obligatorischen Variablen** des Frames. Die zweite Eigenschaft besagt, dass für eine Variable nur ein Typ angegeben werden darf. Die letzte Eigenschaft ist keine theoretische Anforderung, aber sie erleichtert das Parsen der Datenbasis.

Für unsere Beispiele gehen wir davon aus, dass unsere Datenbasis die folgenden Deklarationen hat:

```
$v P Q R $.
$c -> ( ) |- wff $.
```

Die folgende Folge von Anweisungen, die die Modus ponens Schlussregel beschreibt, ist ein Beispiel für ein Frame:

```
wp  $f wff P $.
wq  $f wff Q $.
maj $e |- ( P -> Q ) $.
min $e |- P $.
mp  $a |- Q $.
```

Die folgende Folge von Anweisungen ist kein Frame, da `R` weder in den `$e`'s noch in den `$a`'s vorkommt:

```
wp  $f wff P $.
wq  $f wff Q $.
wr  $f wff R $.
```

```
maj $e |- ( P -> Q ) $.
min $e |- P $.
mp  $a |- Q $.
```

Die folgende Folge von Anweisungen ist kein Frame, da `Q` nicht in einem `$f` vorkommt:

```
wp  $f wff P $.
maj $e |- ( P -> Q ) $.
min $e |- P $.
mp  $a |- Q $.
```

Die folgende Folge von Anweisungen ist kein Frame, da die Anweisung `$a` nicht die letzte ist:

```
wp  $f wff P $.
wq  $f wff Q $.
maj $e |- ( P -> Q ) $.
mp  $a |- Q $.
min $e |- P $.
```

Einem Frame ist eine Folge von **obligatorischen Hypothesen** zugeordnet. Dies ist einfach die Menge aller `$f`- und `$e`-Anweisungen im Frame, in der Reihenfolge, in der sie angegeben werden. Auf einen Frame kann in einem späteren Beweis über das Label der `$a`- oder `$p`-Aussage Bezug genommen werden, und der Beweis nimmt eine Zuordnung zu jeder zwingenden Hypothese in der Reihenfolge vor, in der sie angegeben wurde. Das bedeutet, dass die einmal gewählte Reihenfolge der Hypothesen nicht mehr geändert werden darf, um spätere Beweise, die sich auf die Anweisung des Frames beziehen, nicht zu beeinflussen. (Der Beweisverifizierer von Metamath wird natürlich einen Fehler anzeigen, wenn ein Beweis dadurch falsch wird.) Da Beweise die „umgekehrte Polnische Notation“ verwenden, die in Abschnitt 4.3 beschrieben wird, nennen wir diese Reihenfolge die **RPN-Reihenfolge** der Hypothesen.

Beachten Sie, dass `$d`-Anweisungen nicht zur Menge der obligatorischen Hypothesen gehören und ihre Reihenfolge keine Rolle spielt (solange sie die oben beschriebene vierte Eigenschaft für einen Frame erfüllen). Die `$d`-Anweisungen geben Beschränkungen für Variablen an, die erfüllt sein müssen (und vom Beweisverifizierer überprüft werden), wenn Ausdrücke in einem Beweis für sie ersetzt werden, und die `$d`-Anweisungen selbst werden in einem Beweis nie direkt referenziert.

Ein Frame mit einer (beweisbaren) `$p`-Anweisung erfordert einen Beweis als Teil der `$p`-Anweisung. Manchmal wollen wir in einem Beweis temporäre Variablen oder Dummy-Variablen verwenden, die nicht in der `$p`-Anweisung oder ihren obligatorischen Hypothesen vorkommen. Um dies zu ermöglichen,

definieren wir einen **erweiterten Frame** als einen Frame mit keinem, einem oder mehreren `$d`- und `$f`-Anweisungen, die auf Variablen verweisen, die nicht zu den obligatorischen Variablen des Frames gehören. Alle neuen Variablen, auf die verwiesen wird, werden als **optionale Variablen** des erweiterten Frames bezeichnet. Wenn eine `$f`-Anweisung auf eine optionale Variable verweist, wird sie als **optionale Hypothese** bezeichnet, und wenn eine oder beide der Variablen in einer `$d`-Anweisung optionale Variablen sind, wird sie als **optionale disjunkte Variableneinschränkung** bezeichnet. Die Eigenschaften 2 und 3 für einen Frame gelten auch für einen erweiterten Frame.

Das Konzept der optionalen Variablen ist für Frames mit `$a`-Anweisungen nicht sinnvoll, da diese Anweisungen keine Beweise haben, die von ihnen Gebrauch machen könnten. Es gibt kein Verbot für die Aufnahme optionaler Hypothesen in den erweiterten Frame für eine `$a`-Anweisung, aber sie haben keinen Zweck.

Der folgende Satz von Anweisungen ist ein Beispiel für einen erweiterten Frame, das eine optionale Variable `R` und eine optionale Hypothese `wr` enthält. In diesem Beispiel nehmen wir an, dass der Modus ponens kein Axiom ist, sondern als Theorem aus früheren Aussagen abgeleitet wurde (wir lassen den angenommenen Beweis weg). Die Variable `R` kann, falls gewünscht, in ihrem Beweis verwendet werden (obwohl dies in der Aussagenlogik wahrscheinlich keinen Vorteil hätte). Beachten Sie, dass die Reihenfolge der obligatorischen Hypothesen in der RPN-Reihenfolge immer noch `wp`, `wq`, `maj`, `min` lautet (d.h. `wr` wird weggelassen), und diese Reihenfolge wird immer noch angenommen, wenn in einem nachfolgenden Beweis auf die Behauptung `mp` verwiesen wird.

```
wp  $f wff P $.
wq  $f wff Q $.
wr  $f wff R $.
maj $e |- ( P -> Q ) $.
min $e |- P $.
mp  $p |- Q $= ... $.
```

Jeder Frame ist ein erweiterter Frame, aber nicht jeder erweiterte Frame ist ein Frame, wie dieses Beispiel zeigt. Den zugrundeliegenden Frame für einen erweiterten Frame erhält man, indem man einfach alle Anweisungen entfernt, die optionale Variablen enthalten. Jeder Beweis, der sich auf eine Behauptung bezieht, ignoriert alle Erweiterungen ihres Frames, was bedeutet, dass wir optionale Hypothesen nach Belieben hinzufügen oder löschen können, ohne dass dies Auswirkungen auf nachfolgende Beweise hat.

Die konzeptionell einfachste Art, eine Metamath-Datenbasis zu organisieren, ist eine Folge von erweiterten Frames. Die Gültigkeitsbereichsanweisungen `${` und `$}` können verwendet werden, um den Anfang und das Ende eines erweiterten Frames abzugrenzen, was zu der folgenden möglichen Struktur für eine Datenbasis führt.

(`$v-` *und* `$c-`*Anweisungen*)
`${`
  erweiterter Frame
`$}`
`${`
  erweiterter Frame
`$}`
⋮

In der Praxis ist diese Struktur unzweckmäßig, weil wir alle `$f-`, `$e-` und `$d-`Anweisungen immer und immer wieder wiederholen müssen, anstatt sie einmal für die Verwendung durch mehrere Annahmen anzugeben. Die Gültigkeitsbereichsanweisungen, die wir als nächstes besprechen werden, ermöglichen dies. Im Prinzip kann jede Metamath-Datenbasis in das obige Format konvertiert werden, und das obige Format ist am bequemsten zu verwenden, wenn man eine Metamath-Datenbasis als formales System betrachtet (Appendix C). Tatsächlich konvertiert Metamath die Datenbasis intern in das oben genannte Format. Der Befehl `show statement` im Metamath-Programm zeigt Ihnen den Inhalt des Frames für jede `$a-` oder `$p`-Anweisung sowie die Erweiterung im Falle einer `$p`-Anweisung.

Bei der Diskussion von Anweisungen mit Gültigkeitsbereich kann es hilfreich sein, sich die äquivalente Folge von Frames vorzustellen, die beim Parsen der Datenbasis entsteht. Gültigkeitsbereiche sind (abgesehen von der oben erwähnten begrenzten Verwendung zur Abgrenzung von Frames) keine theoretische Voraussetzung für Metamath, macht den Umgang damit aber bequemer.

4.2.8 Gültigkeitsbereichsanweisungen ($\{ und $\})

Die **Gültigkeitsbereich**-Anweisungen, `${` (**beginn eines Blocks**) und `$}` (**Ende eines Blocks**), bieten ein Mittel zur Steuerung des Teils einer Datenbasis, in dem bestimmte Anweisungstypen erkannt werden. Die Syntax einer Gültigkeitsbereichsanweisung ist sehr einfach; sie besteht lediglich aus dem Schlüsselwort der Anweisung:

```
${
$}
```

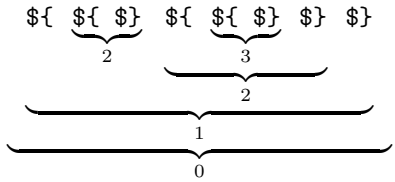
Betrachten wir zum Beispiel die folgende Datenbasis, in der wir alle Token außer den Schlüsselwörtern der Anweisung entfernt haben. Für die folgende Diskussion haben wir die Anweisungen mit Indizes (tiefgestellten Ziffern) versehen; diese Indizes erscheinen nicht in der eigentlichen Datenbasis.

$${_1\ ${_2\ $}_2\ ${_3\ ${_4\ $}_4\ $}_3\ $}_1$$

Zu jeder `${`-Anweisung in diesem Beispiel **gehört** die `$}`-Anweisung mit demselben Index. Jedes Paar von so zusammengehörenden Anweisungen de-

finiert einen Bereich der Datenbasis, der als **Block** bezeichnet wird. Blöcke können innerhalb anderer Blöcke **verschachtelt** werden; im Beispiel, ist der Block, der durch `${`$_4$ und `$}`$_4$ definiert ist, innerhalb des Blocks verschachtelt, der durch `${`$_3$ und `$}`$_3$ sowie innerhalb des durch `${`$_1$ und `$}`$_1$ definierten Blocks. Im Allgemeinen kann ein Block leer sein, er kann nur Anweisungen enthalten, die keine Gültigkeitsbereichsanweisungen sind[12] oder er kann eine beliebige Mischung aus anderen Blöcken und Anweisungen, die keine Gültigkeitsbereichsanweisungen sind, enthalten. (Dies wird eine „rekursive“ Definition eines Blocks genannt.)

Jedem Block ist eine Nummer zugeordnet, die als seine **Verschachtelungstiefe** bezeichnet wird und angibt, wie tief der Block verschachtelt ist. Die Verschachtelungstiefe der Blöcke in unserem Beispiel sind wie folgt:



Die gesamte Datenbasis wird als ein einziger großer Block (der äußerste Block) mit einer Verschachtelungstiefe von 0 betrachtet. Der äußerste Block wird nicht durch Gültigkeitsbereichsanweisungen eingeklammert.[13]

Alle Metamath-Anweisungen, die unabhängig von einem Gültigkeitsbereich sind, werden ab der Stelle, an der sie angegeben werden, gültig oder **aktiv**.[14] Bestimmte dieser Anweisungstypen werden am Ende des Blocks, in dem sie enthalten sind, inaktiv; diese Anweisungstypen sind:

`$c`, `$v`, `$d`, `$e`, und `$f`.

Die anderen Anweisungstypen bleiben für immer aktiv (d.h. bis zum Ende der Datenbasis); dies sind:

`$a` und `$p`.

Jede Anweisung (dieser 7 Typen), die sich im äußersten Block befindet, bleibt bis zum Ende der Datenbasis aktiv und ist somit effektiv eine „globale“ Anweisung.

Alle `$c`-Anweisungen müssen im äußersten Block platziert werden. Da sie also immer global sind, könnte man sie als zu beiden oben genannten Kategorien gehörig betrachten.

Der **Gültigkeitsbereich** einer Anweisung ist die Menge der Anweisungen, die sie als aktiv erkennen.

[12] Die Anweisungen, die nicht `${` und `$}` sind.

[13] Die Sprache wurde deshalb so konzipiert, damit mehrere Quelldateien leichter zusammengefügt werden können.

[14] Um die Dinge etwas zu vereinfachen, machen wir uns nicht die Mühe, den Begriff „aktiv“ für die Gültigkeitsbereichsanweisungen zu definieren.

Der Begriff „aktiv“ ist auch für mathematische Symbole definiert. Mathematische Symbole (Konstanten und Variablen) werden **aktiv** in den `$c`-Anweisungen und `$v`-Anweisungen, in den sie deklariert werden. Eine Variable wird inaktiv, wenn ihre Deklarationsanweisung inaktiv wird. Da alle `$c`-Anweisungen im äußersten Block stehen müssen, wird eine Konstante niemals inaktiv, nachdem sie deklariert wurde.

Umdeklarierung von mathematischen Symbolen

Eine Variable kann nicht ein zweites Mal deklariert werden, solange sie aktiv ist, aber sie kann erneut deklariert werden, nachdem sie inaktiv geworden ist. Auf diese Weise lassen sich auf bequeme Weise „lokale“ Variablen einführen, d.h. temporäre Variablen, die im Rahmen einer Behauptung oder eines Beweises verwendet werden können, ohne dass man sie für immer beibehalten muss. Eine zuvor deklarierte Variable kann nicht neu als Konstante deklariert werden.

Eine Konstante darf nicht neu deklariert werden. Und, wie oben erwähnt, müssen Konstanten im äußersten Block deklariert werden.

Der Grund dafür, dass Variablen einen begrenzten Gültigkeitsbereich haben können, Konstanten jedoch nicht, liegt darin, dass eine Behauptung (`$a`- oder `$p`-Anweisung) bis zum Ende der Datenbasis für die Verwendung in Beweisen verfügbar bleibt. Variablen im Rahmen einer Behauptung können in einem Beweisschritt, der sich auf die Behauptung bezieht, durch alles ersetzt werden, was benötigt wird, während Konstanten fest bleiben und durch nichts ersetzt werden können. Das spezielle Token, das für eine Variable im Rahmen einer Behauptung verwendet wird, ist irrelevant, wenn die Behauptung in einem Beweis referenziert wird, und es spielt keine Rolle, wenn dieses Token außerhalb des Frames der referenzierten Behauptung nicht verfügbar ist. Konstanten hingegen müssen global festgelegt werden.

Theoretisch ist es nicht notwendig, dass Variablen für begrenzte Bereiche aktiv sind, anstatt global zu sein. Es handelt sich lediglich um eine Annehmlichkeit, die es beispielsweise ermöglicht, sie lokal mit den entsprechenden Deklarationen vom `$f` Variablentyp zu gruppieren.

Zurück zu Frames

Nachdem wir nun die Gültigkeitsbereiche behandelt haben, werden wir uns ansehen, wie eine beliebige Metamath-Datenbasis in die einfache Folge von erweiterten Frames konvertiert werden kann, die auf S. 148 beschrieben ist. Dies ist auch die Art und Weise, wie Metamath die Datenbasis intern speichert, wenn es die Datenbasisquelle einliest. Die Methode ist einfach. Zunächst werden alle Konstanten- und Variablendeklarationen (`$c` und `$v`) in der Datenbasis gesammelt, wobei doppelte Deklarationen derselben Variable in verschiedenen Bereichen ignoriert werden. Dann setzen wir unsere gesammelten `$c`- und `$v`-Deklarationen an den Anfang der Datenbasis, so

dass ihr Gültigkeitsbereich die gesamte Datenbasis ist. Als Nächstes bestimmen wir für jede Behauptung in der Datenbasis ihren Frame und ihren erweiterten Frame. Der erweiterte Frame ist einfach die Sammlung der `$f`, `$e` und `$d`-Anweisungen, die aktiv sind. Der Frame ist der erweiterte Frame, aus dem alle optionalen Hypothesen entfernt wurden.

Eine äquivalente Formulierung ist, dass der erweiterte Frame einer Behauptung die Sammlung aller `$f`-, `$e`- und `$d`-Anweisungen ist, deren Gültigkeitsbereich die Behauptung umfasst. Die Anweisungen `$f` und `$e` treten in der Reihenfolge auf, in der sie angegeben werden (die Reihenfolge ist für `$d`-Anweisungen irrelevant).

4.3 Die Anatomie eines Beweises

Jede beweisbare Behauptung (`$p`-Anweisung) in einer Datenbasis muss einen **Beweis** enthalten. Der Beweis befindet sich zwischen den Schlüsselwörtern `$=` und `$.` in der `$p`-Anweisung.

In der Metamath-Basissprache ist ein Beweis eine Folge von Anweisungs-Labels. Diese Label-Sequenz dient als eine Menge von Anweisungen, die das Metamath-Programm verwendet, um eine Reihe von mathematischen Symbolsequenzen zu konstruieren. Die Konstruktion muss letztlich zu der Foilge mathematischer Symbole führen, die zwischen den Schlüsselwörtern `$p` und `$=` der `$p`-Anweisung enthalten ist. Andernfalls wird das Metamath-Programm beim Verifizieren des Beweises diesen als falsch betrachten und Sie mit einer entsprechenden Fehlermeldung darauf hinweisen.[15] Jedes Label in einem Beweis **referenziert** seine entsprechende Anweisung.

Zu jeder Behauptung (`$p`- oder `$a`-Anweisung) gehört eine Reihe von Hypothesen (`$f`- oder `$e`-Anweisungen), die in Bezug auf diese Behauptung aktiv sind. Einige sind obligatorisch, die anderen sind optional. Sie sollten sich diese Konzepte bei Bedarf nochmals in Erinnerung rufen.

Jedes Label in einem Beweis muss entweder das Label einer vorhergehenden Behauptung (`$a`- oder `$p`-Anweisung) oder das Label einer aktiven Hypothese (`$e`- oder `$f`-Anweisung) der `$p`-Anweisung sein, die den Beweis enthält. Labels für Hypothesen können sowohl auf die obligatorischen als auch auf die optionalen Hypothesen der Anweisung `$p` verweisen.

Die Label-Sequenz in einem Beweis spezifiziert eine Konstruktion in **umgekehrter polnischer Notation** (RPN). Sie sind vielleicht mit RPN vertraut, wenn Sie ältere Hewlett-Packard- oder ähnliche Taschenrechner benutzt haben. In der Analogie zum Taschenrechner ist ein Label einer Hypothese wie eine Zahl und ein Label für eine Behauptung wie eine Operation (genauer gesagt, eine n-äre Operation, wenn die Behauptung n `$e`-

[15] Um das Laden zu beschleunigen, verifiziert das Metamath-Programm Beweise nicht automatisch, wenn Sie mit `read` eine Datenbasis einlesen, es sei denn, Sie verwenden die Option `/verify`. Nachdem eine Datenbasis eingelesen wurde, können Sie den Befehl `verify proof *` verwenden, um Beweise zu verifizieren.

Hypothesen hat) zu verstehen. Auf einem RPN-Rechner nimmt eine Operation eine oder mehrere vorherige Zahlen in einer Eingabefolge, führt eine Berechnung mit ihnen durch und ersetzt diese Zahlen und sich selbst durch das Ergebnis der Berechnung. Zum Beispiel ergibt die Eingabefolge $2, 3, +$ auf einem RPN-Rechner 5, und die Eingabefolge $2, 3, 5, \times, +$ ergibt $2, 15, +$, was 17 letztendlich ergibt.

Zum Verständnis der RPN-Verarbeitung gehört das Konzept eines **Stapels**, den man sich als eine Reihe von temporären Speicherplätzen vorstellen kann, die Zwischenergebnisse enthalten. Wenn Metamath auf ein Label für eine Hypothese stößt, wird die mathematische Symbolfolge der Hypothese auf den Stapel gelegt oder bf geschoben. Wenn Metamath auf ein Label für eine Behauptung stößt, ordnet es die obersten Stapel-Einträge den *obligatorischen* Hypothesen der Behauptung zu, und zwar in der Reihenfolge, in der der oberste Stapel-Eintrag der letzten obligatorischen Hypothese der Behauptung zugeordnet ist. Anschließend wird ermittelt, welche Substitutionen in den Variablen der obligatorischen Hypothesen der Behauptung vorgenommen werden müssen, damit sie mit den zugehörigen Stapel-Einträgen übereinstimmen. Dann werden die gleichen Substitutionen in der Behauptung selbst vorgenommen. Schließlich entfernt Metamath die übereinstimmenden Hypothesen vom Stapel und schiebt die substituierte Behauptung auf den Stapel.

Für den Zweck des Abgleichs der obligatorischen Hypothese mit den obersten Stapel-Einträgen ist es unerheblich, ob eine Hypothese eine `$e`- oder `$f`-Anweisung ist. Wichtig ist nur, dass es eine Menge von Substitutionen[16] gibt, die eine Übereinstimmung ermöglichen (und wenn nicht, wird der Beweisverifizierer eine Fehlermeldung ausgeben). Die Metamath-Sprache ist so spezifiziert, dass, wenn eine Menge von Substitutionen existiert, diese eindeutig ist. Insbesondere die Anforderung, dass jede Variable einen Typ hat, der mit einer Anweisung `$f` spezifiziert ist, stellt die Eindeutigkeit sicher.

Wir werden dies anhand eines Beispiels veranschaulichen. Betrachten Sie die folgende Metamath-Quelldatei:

```
$c ( ) -> wff $.
$v p q r s $.
wp $f wff p $.
wq $f wff q $.
wr $f wff r $.
ws $f wff s $.
w2 $a wff ( p -> q ) $.
wnew $p wff ( s -> ( r -> p ) ) $= ws wr wp w2 w2 $.
```

Dieses Metamath-Beispiel zeigt die Definition und den Beweis (d.h. die Konstruktion) einer wohlgeformten Formel (wff) in der Aussagenlogik. (Viel-

[16] In der Metamath-Spezifikation (Abschnitt 4.1) wird der Begriff „Substitution“ im Singular verwendet, um sich auf die Menge der Substitutionen zu beziehen, über die wir hier sprechen.

leicht möchten Sie dieses Beispiel in eine Datei eingeben, um mit dem Metamath-Programm zu experimentieren.) Die ersten beiden Anweisungen deklarieren (führen die Namen ein von) vier Konstanten und vier Variablen. Die nächsten vier Anweisungen spezifizieren die Variablentypen, nämlich dass jede Variable für eine wff steht. Die Anweisung `w2` definiert (postuliert) eine Möglichkeit, um aus zwei gegebenen wffs `p` und `q` eine neue wff, `( p -> q )`, zu erzeugen. Die obligatorischen Hypothesen von `w2` sind `wp` und `wq`. Die Anweisung `wnew` behauptet, dass `( s -> ( r -> p ) )` eine wff ist, wenn drei wff `s`, `r` und `p` gegeben sind. Genauer gesagt behauptet `wnew`, dass die Folge von zehn Symbolen `wff ( s -> ( r -> p ) )` aus früheren Behauptungen und den Hypothesen von `wnew` beweisbar ist. Metamath weiß nicht oder kümmert sich nicht darum, was eine wff ist, und soweit es sie betrifft, ist der Typcode `wff` einfach ein beliebiges konstantes Symbol in einer mathematischen Symbolfolge. Die obligatorischen Hypothesen von `wnew` sind `wp`, `wr`, und `ws`; `wq` ist eine optionale Hypothese. In unserem speziellen Beweis wird die optionale Hypothese nicht referenziert, aber im Allgemeinen könnte jede Kombination von aktiven (d.h. optionalen und obligatorischen) Hypothesen referenziert werden. Der Beweis der Anweisung `wnew` ist die Folge von fünf Labels, die mit `ws` (Schritt 1) beginnt und mit `w2` (Schritt 5) endet.

Wenn Metamath den Beweis verifiziert, scannt es den Beweis von links nach rechts. Wir werden untersuchen, was bei jedem Schritt des Beweises passiert. Der Stapel ist zu Beginn leer. In Schritt 1 sucht Metamath nach dem Label `ws` und stellt fest, dass es sich um eine Hypothese handelt, also schiebt es die Symbolfolge der Anweisung `ws` auf den Stapel:

Stapelposition	Inhalt
1	`wff s`

Metamath sieht, dass die Labels `wr` und `wp` in den Schritten 2 und 3 ebenfalls Hypothesen sind, also schiebt es sie auf den Stapel. Nach Schritt 3 sieht der Stapel wie folgt aus:

Stapelposition	Inhalt
3	`wff p`
2	`wff r`
1	`wff s`

In Schritt 4 sieht Metamath, dass das Label `w2` eine Behauptung ist, also muss es etwas verarbeiten. Zunächst werden die obligatorischen Hypothesen von `w2`, nämlich `wp` und `wq`, mit den Stapelpositionen 2 und 3 verknüpft, *in dieser Reihenfolge*. Metamath stellt fest, dass die einzige Möglichkeit, die Hypothese `wp` mit dem Inhalt auf Stapelposition 2 und `wq` mit dem Inhalt auf Stapelposition 3 übereinstimmen zu lassen, darin besteht, die Variable `p` durch `r` und `q` durch `p` zu ersetzen. Metamath nimmt diese Substitutionen

in `w2` vor und erhält die Symbolfolge `wff ( r -> p )`. Es entfernt die Hypothesen von den Stapelpositionen 2 und 3 und legt das Ergebnis auf der Stapelposition 2 ab:

Stapelposition	Inhalt
2	`wff ( r -> p )`
1	`wff s`

In Schritt 5 sieht Metamath, dass das Label `w2` eine Behauptung ist, also muss wieder eine Verarbeitung stattfinden. Zunächst werden die obligatorischen Hypothesen von `w2`, d.h. `wp` und `wq`, den Stapelpositionen 1 und 2 zugeordnet. Metamath stellt fest, dass die einzige Möglichkeit, die Hypothesen zur Übereinstimmung zu bringen, darin besteht, die Variable `p` durch `s` und `q` durch `( r -> p )` zu ersetzen. Metamath führt diese Substitutionen in `w2` durch und erhält die Symbolfolge `wff ( s -> ( r -> p ) )`. Es entfernt die Stapelposition 1 und 2 und legt das Ergebnis auf Stapelposition 1 ab:

Stapelposition	Inhalt
1	`wff ( s -> ( r -> p ) )`

Nachdem Metamath die Verarbeitung des Beweises abgeschlossen hat[17], prüft es, ob der Stapel genau ein Element enthält und ob dieses Element mit der mathematischen Symbolfolge in der `$p`-Anweisung übereinstimmt. Dies ist bei unserem Beweis von `wnew` der Fall, also haben wir `wnew` erfolgreich bewiesen. Wenn das Ergebnis davon abweicht, wird Metamath Sie mit einer Fehlermeldung informieren. Eine Fehlermeldung wird auch ausgegeben, wenn der Stapel am Ende des Beweises mehr als einen Eintrag enthält, oder wenn der Stapel an irgendeiner Stelle des Beweises nicht genügend Einträge enthält, um alle obligatorischen Hypothesen einer Behauptung zu erfüllen. Schließlich wird Metamath Sie mit einer Fehlermeldung benachrichtigen, wenn keine Substitution möglich ist, die die Hypothese einer referenzierten Behauptung mit den Stapeleinträgen übereinstimmen lässt. Sie können mit den verschiedenen Arten von Fehlern, die Metamath erkennt, experimentieren, indem Sie einige kleine Änderungen am Beweis unseres Beispiels vornehmen.

Die Notation von Metamath für Beweise wurde in erster Linie entwickelt, um Beweise in einer relativ kompakten Weise auszudrücken, nicht um sie für Menschen lesbar zu machen. Metamath kann Beweise mit dem Befehl `show proof` auf verschiedene Arten anzeigen. Die Option `/lemmon` zeigt sie in einem Format an, das leichter zu lesen ist, wenn die Beweise kurz sind, und Sie haben Beispiele für seine Verwendung in Kapitel 2 gesehen. Bei längeren Beweisen ist es nützlich, die Baumstruktur des Beweises zu sehen. Eine Baumstruktur wird angezeigt, wenn die Option `/lemmon` weggelassen wird.

[17] Anm. der Übersetzer: Da der Beweis kein weiteres Label enthält.

Wenn Sie sich an diese Darstellung gewöhnt haben, werden Sie sie wahrscheinlich bequemer finden. Die Baumdarstellung des Beweises in unserem Beispiel sieht wie folgt aus:

```
1     wp=ws    $f wff s
2        wp=wr    $f wff r
3        wq=wp    $f wff p
4     wq=w2    $a wff ( r -> p )
5  wnew=w2  $a wff ( s -> ( r -> p ) )
```

Die Zahl links von jeder Zeile ist die Schrittnummer. Es folgt eine **Hypothesenzuordnung**, bestehend aus zwei Labels, die durch `=` getrennt sind. Links von `=` steht (außer im letzten Schritt) das Label einer Hypothese einer Behauptung, auf die später im Beweis Bezug genommen wird; hier sind die Schritte 1 und 4 die Hypothesenzuordnungen für die Behauptung `w2`, auf die in Schritt 5 Bezug genommen wird. Eine Hypothesenzuordnung ist eine Stufe weiter eingerückt als die Behauptung, die sie verwendet. Dadurch ist es einfach, die entsprechende Behauptung zu finden, indem man sich direkt nach unten bewegt, bis die Einrückungsstufe um eine Stufe niedriger ist als die, von der aus man begonnen hat. Rechts von jedem `=` befindet sich das Label des Beweisschritts für diesen Schritt. Das Schlüsselwort der Anweisung im Label des Beweisschritts wird als nächstes aufgeführt, gefolgt vom Inhalt des obersten Stapeleintrags (dem neusten Stapeleintrag), wie er nach der Verarbeitung dieses Beweisschritts vorliegt. Mit ein wenig Übung sollten Sie keine Probleme haben, Beweise in diesem Format zu lesen.

Metamath-Beweise beinhalten die Syntaxkonstruktion einer Formel. In der Standardmathematik wird diese Art der Konstruktion nicht als Teil des Beweises angesehen, und sie wird nach einer Weile sicherlich ziemlich langweilig. Daher zeigt der Befehl `show proof` standardmäßig die Syntaxkonstruktion nicht an. Früher *zeigte* der Befehl `show proof` die Syntaxkonstruktionen an, und man musste die Option `/essential` hinzufügen, um sie auszublenden, aber heute ist `/essential` der Standard, und man muss `/all` verwenden, um die Syntaxkonstruktionen zu sehen.

Bei der Überprüfung eines Beweises prüft Metamath, dass keine obligatorische `$d`-Anweisung einer Behauptung, auf die in einem Beweis verwiesen wird, verletzt wird, wenn Substitutionen an den Variablen in der Behauptung vorgenommen werden. Für Einzelheiten siehe Abschnitt 4.1.4 oder 4.2.4.

4.3.1 Das Konzept der Vereinheitlichung

Wenn Metamath während der Überprüfung eines Beweises auf ein Label einer Behauptung stößt, assoziiert es die obligatorischen Hypothesen der Behauptung mit den obersten Einträgen des RPN-Stapels. Metamath bestimmt dann, welche Substitutionen es an den Variablen in den obligatori-

schen Hypothesen der Behauptung vornehmen muss, damit diese Hypothesen mit ihren entsprechenden Stapeleinträgen korrespondieren. Dieser Vorgang wird als **Vereinheitlichung** bezeichnet. (Wir verwenden den Begriff „Vereinheitlichung“ auch informell, um eine Reihe von Substitutionen zu bezeichnen, die sich aus diesem Prozess ergeben, wie in „zwei Vereinheitlichungen sind möglich“). Nachdem die Substitutionen vorgenommen wurden, werden die Hypothesen als **vereinheitlicht** bezeichnet.

Ist eine solche Ersetzung nicht möglich, hält Metamath den Beweis für fehlerhaft und gibt eine Fehlermeldung aus.

Der in der Literatur beschriebene allgemeine Algorithmus für eine Vereinheitlichung ist etwas komplex. Im Fall von Metamath ist er jedoch absichtlich einfach gehalten. Obligatorische Hypothesen müssen in der Reihenfolge ihres Auftretens auf den Beweisstapel geschoben werden. Darüber hinaus muss der Typ jeder Variablen mit einer `$f`-Hypothese spezifiziert werden, bevor sie verwendet wird, und jede `$f`-Hypothese muss die eingeschränkte Syntax eines Typcodes (einer Konstanten) gefolgt von einer Variablen aufweisen. Der Typcode in der `$f`-Hypothese muss mit dem ersten Symbol des entsprechenden RPN-Stack-Eintrags übereinstimmen (der ebenfalls eine Konstante ist), so dass die einzige mögliche Übereinstimmung für die Variable in der `$f`-Hypothese die Folge von Symbolen im Stapeleintrag nach der anfänglichen Konstanten ist.

Im Beweis-Assistenten wird ein allgemeinerer Vereinheitlichungsalgorithmus verwendet. Während ein Beweis entwickelt wird, sind manchmal nicht genügend Informationen verfügbar, um eine eindeutige Vereinheitlichung zu bestimmen. In diesem Fall bittet Metamath Sie, die richtige zu wählen.

4.4 Erweiterungen der Metamath-Sprache

4.4.1 Kommentare in der Metamath-Sprache

Die Kommentarfunktion ermöglicht es Ihnen, den Inhalt einer Datenbasis mit Anmerkungen zu versehen. Wie bei den meisten Computersprachen werden Kommentare bei der Interpretation des Inhalts der Datenbasis ignoriert. Kommentare fungieren beim Parsen einer Datenbasis effektiv als zusätzlicher Whitespace zwischen den Token.

Ein Kommentar kann am Anfang, am Ende oder zwischen zwei beliebigen Token in einer Quelldatei stehen.

Kommentare haben die folgende Syntax:

`$(` *text* `$)`

Hier ist *text* eine, möglicherweise leere, Zeichenkette aus beliebigen Zeichen des Metamath-Zeichensatzes (p. 126), mit der Ausnahme, dass die Zeichen-

ketten `$(` und `$)` nicht in *text* vorkommen dürfen. Daher sind verschachtelte Kommentare nicht erlaubt:[18] Metamath wird sich beschweren, wenn

```
$( This is a $( nested $) comment. $)
```

in einer Datenbasis vorkommt. Um diese fehlende Verschachtelungsmöglichkeit zu kompensieren, ändere ich oft alle `$`'s in `@`'s in Abschnitten des Metamath-Codes, die ich auskommentieren möchte.

Das Metamath-Programm unterstützt eine Reihe von Auszeichnungsmechanismen und Konventionen, um gut aussehende Ergebnisse in LaTeX und HTML zu generieren, wie unten beschrieben wird. Diese Auszeichnungsfunktionen haben ausschließlich damit zu tun, wie die Kommentare ausgegeben werden, und haben keinen Einfluss darauf, wie Metamath die Beweise in der Datenbasis verifiziert. Ihre unsachgemäße Verwendung kann zu einer falsch angezeigten Ausgabe führen, aber es werden keine Metamath-Fehlermeldungen bei den Befehlen `read` und `verify proof` ausgegeben. (Der Befehl `write theorem_list` prüft jedoch als Nebeneffekt seiner HTML-Generierung auf Auszeichnungsfehler.) Abschnitt 5.7 enthält Anweisungen zur Erstellung von LaTeX-Ausgaben, und Abschnitt 5.8 enthält Anweisungen zur Erstellung von HTML-Ausgaben.

Überschriften

Wenn unmittelbar nach dem `$(` eine neue Zeile folgt, die mit einer Kennzeichnung für Überschriften beginnt, handelt es sich um eine Überschrift. Diese kann beginnen mit:

`####` - Hauptteilüberschrift

`#*#*` - Abschnittsüberschrift

`=-=-` - Unterabschnittsüberschrift

`-.-.` - Unterunterabschnittsüberschrift

Die auf die Zeile mit der Kennzeichnung folgende Zeile wird nach dem Abschneiden der Leerzeichen für den Eintrag im Inhaltsverzeichnisses verwendet. Die nächste Zeile sollte eine weitere Zeile mit einer (abschließenden) passenden Kennzeichnung sein. Jeglicher Text danach, aber vor dem abschließenden `$`, wird in die Seite `mmtheoremsNNN.html` aufgenommen. Dies kann z. B. eine ausführliche Beschreibung des Abschnitts sein.

Weitere Informationen erhalten Sie, wenn Sie `help write theorem_list` ausführen.

[18] Computersprachen haben unterschiedliche Standards für verschachtelte Kommentare, und anstatt sich für einen zu entscheiden, ist es am einfachsten, sie überhaupt nicht zu erlauben, zumindest in der aktuellen Version (0.177) von Metamath.

Mathe-Modus

Innerhalb von Kommentaren wird eine Zeichenkette von Token, die von einfachen Anführungszeichen (`) eingeschlossen ist, während des HTML oder LaTeX-Ausgabesatzes in standardmäßige mathematische Symbole umgewandelt, entsprechend den Informationen in der speziellen `$t` in der Datenbasis (siehe Abschnitt 4.4.2 für Informationen über den Schriftsatzkommentar und Anhang A für Beispiele seiner Ergebnisse).

Das erste einfache Anführungszeichen ` veranlasst den Ausgabeprozessor, in den **Mathe-Modus** einzutreten, und der zweite verlässt ihn. In diesem Modus werden die auf ` folgenden Zeichen als eine Folge von mathematischen Symbolen interpretiert, die durch Whitespace getrennt sind. Die Zeichen werden in dem `$t`-Kommentar gesucht und, wenn sie gefunden werden, durch die mathematischen Standardsymbole ersetzt, denen sie entsprechen, bevor sie in die Ausgabedatei geschrieben werden. Werden sie nicht gefunden, wird das Symbol so ausgegeben, wie es ist, und es wird eine Warnung ausgegeben. Die Token müssen nicht in der Datenbasis aktiv sein, obwohl eine Warnung ausgegeben wird, wenn sie nicht mit den Anweisungen `$c` oder `$v` deklariert sind.

Zwei aufeinanderfolgende einfache Anführungszeichen `` werden als ein einziges einfaches Anführungszeichen ausgegeben (sowohl innerhalb als auch außerhalb des Mathe-Modus) und führen nicht dazu, dass der Ausgabeprozessor in den Mathe-Modus eintritt oder diesen verlässt.

Hier ist ein Beispiel für seine Anwendung:

```
$( Pierce's axiom, ` ( ( ph -> ps ) -> ph ) -> ph ` ,
                  is not very intuitive. $)
```

wird ausgegeben als

$(Pierce's axiom, $((\varphi \to \psi) \to \varphi) \to \varphi$, is not very intuitive. $)

Beachten Sie, dass das mathematische Symbol von Whitespace umgeben sein muss. Whitespace sollte auch die Begrenzungszeichen ` umgeben.

Die Funktion „Mathe-Modus“ bietet Ihnen auch eine schnelle und einfache Möglichkeit, Text mit mathematischen Symbolen zu erzeugen, unabhängig vom Verwendungszweck von Metamath. Dazu erstellen Sie einfach Ihren Text mit einfachen Anführungszeichen, die Ihre Formeln umgeben, nachdem Sie sichergestellt haben, dass Ihre mathematischen Symbole auf LaTeX-Symbole abgebildet sind, wie in Anhang A beschrieben. Es ist einfacher, wenn Sie mit einer Datenbasis mit vordefinierten Symbolen wie `set.mm` beginnen. Ersetzen Sie einen vorhandenen Kommentar durch eine mathematische Zeichenkette in einfachen Anführungszeichen und setzen Sie dann die diesem Kommentar entsprechende Anweisung gemäß den Anweisungen des Befehls `help tex` im Metamath-Programm. Sie werden dann wahrscheinlich die resultierende Datei mit einem Texteditor bearbeiten wollen, um sie genau auf Ihre Bedürfnisse abzustimmen.

Label-Modus

Außerhalb des Mathe-Modus zeigt eine Tilde ~ dem Ausgabeprozessor von Metamath an, dass das folgende Token (d.h. die Zeichen bis zum nächsten Whitespace) ein Label oder eine URL darstellt. Dieser Formatierungsmodus wird als **Label-Modus** bezeichnet. Wenn anstelle des Label-Modus tatsächlich eine Tilde ausgegeben werden soll (außerhalb des Mathe-Modus), verwenden Sie zwei Tilden in einer Reihe, um sie darzustellen.

Bei der Erzeugung einer LaTeX-Ausgabedatei wird das folgende Token in der Schriftart `typewriter` formatiert und die Tilde entfernt, damit es sich vom restlichen Text abhebt. Diese Formatierung wird auf alle Zeichen nach der Tilde bis zum ersten Whitespace angewendet. Es wird nicht geprüft, ob es sich bei dem Token um ein Label für eine Anweisung handelt oder nicht, und das Token muss nicht die korrekte Syntax für ein Label haben; es werden keine Fehlermeldungen ausgegeben. Die einzige Auswirkung des Label-Modus auf die Ausgabe ist, dass für die Token, die in die Ausgabedatei LaTeX eingefügt werden, eine Schreibmaschinenschrift verwendet wird.

Bei der Erzeugung von HTML *müssen* die Token nach der Tilde eine URL (entweder http: oder https:) oder ein gültiges Label sein. Ist dies nicht der Fall, werden bei der Ausgabe Fehlermeldungen ausgegeben. Es wird ein Hyperlink zu dieser URL oder diesem Label erzeugt.

Querverweis zum Literaturverzeichnis

Querverweis zum Literaturverzeichnis werden bei der Erstellung von HTML besonders behandelt, wenn sie speziell formatiert sind. Text in der Form `[`*author*`]` wird als Querverweis zum Literaturverzeichnis betrachtet. Siehe `help html` und `help write bibliography` im Metamath-Programm für weitere Informationen. Siehe auch Abschnitte 4.4.2 und 5.8.2.

Die Notation `[`*author*`]` erzeugt auch einen Eintrag in der Datei für das Literaturverzeichnis, die von `write bibliography` (Abschnitt 5.8.2) für HTML erzeugt wird. Damit dies richtig funktioniert, muss der umgebende Kommentar wie folgt formatiert sein:

keyword label noise-word `[`*author*`]` `p.` *number*

zum Beispiel

```
Theorem 5.2 von [Monk] S. 223
```

Beim *keyword* wird nicht zwischen Groß- und Kleinschreibung unterschieden, und es muss einer der folgenden Begriffe verwendet werden[19]:

```
Axiom Chapter Claim Compare Conclusion Condition Conjecture
Corollary Definition Equation Example Exercise Fact Figure
Introduction Item Lemma Lemmas Line Lines Notation Note
```

[19] Anm. der Übersetzer: angepasst an Metamath - Version 0.198 7-Aug-2021

```
Observation Paragraph Part Postulate Problem Proof Property
Proposition Remark Result Rule Scheme Scolia Scolion Section
Statement Subsection Table Theorem
```

Das optionale *label* kann aus mehr als einem Wort (nicht *keyword* und nicht *noise-word*) bestehen. Das optionale *noise-word* ist eines der folgenden:

```
from in of on
```

und wird bei der Erstellung der Datei für das Literaturverzeichnis ignoriert. Der Befehl `write bibliography` führt eine Fehlerprüfung durch, um das obige Format zu überprüfen.[20]

Parenthesen

Das Ende eines Kommentars kann eine oder mehrere Klammerausdrücke enthalten, d.h. spezielle Beschreibungen, die in Klammern eingeschlossen sind. Das Metamath-Programm sucht nach bestimmten Klammerausdrücken und kann daraufhin Warnungen ausgeben. Diese sind:

`(Contributed by` *NAME*`,` *DATE*`.)` - dokumentiert den Namen des ursprünglichen Verfassers und das Erstellungsdatum.

`(Revised by` *NAME*`,` *DATE*`.)` - dokumentiert den Namen des Mitwirkenden und das Erstellungsdatum, das zu einer signifikanten Überarbeitung geführt hat (nicht nur zu einer automatischen Minimierung oder Kürzung eines Beweises).

`(Proof shortened by` *NAME*`,` *DATE*`.)` - dokumentiert den Namen und das Datum desjenigen, der eine erhebliche Verkürzung des Beweises erzielt hat (nicht nur eine automatische Minimierung).

`(Proof modification is discouraged.)` - Hinweis, dass dieser Beweis normalerweise nicht geändert werden sollte.

`(New usage is discouraged.)` - Hinweis, dass diese Behauptung normalerweise nicht verwendet werden sollte.

Das Datum *DATE* muss in der Form (D)D-MMM-YYYY angegeben werden, wobei MMM die englische Abkürzung für den Monat ist.[21]

[20] Anm. der Übersetzer: oft soll ein Wort/Text in eckigen Klammern kein Querverweis sein. Dann sollte statt der einfachen öffnenden eckigen Klammer eine doppelte öffnende eckige Klammer verwendet werden, z.B. `[[`*wieder*`]`. Ansonsten wird ein Fehler gemeldet, wenn zwischen den eckigen Klammern kein Autor (aus dem Literaturverzeichnis) steht.

[21] Anm. der Übersetzer: Und (D)D der ein- oder zweistellige Tag im Monat ist.

Sonstige Textauszeichnungen

Neben dem Mathe-Modus und dem Label-Modus gibt es noch weitere Textauszeichnungen, um ansprechende Ergebnisse zu erzeugen:

*space*_*non-space* (i.e. _

_ (Unterstrich) - Kursive Darstellung von Text ab *space*_*non-space* (d.h. _ mit einem Leerzeichen davor und einem Nicht-Leerzeichen danach) bis zum nächsten *non-space*_*space*. Normale Interpunktion (z.B. ein nachgestelltes Komma oder ein Punkt) wird bei der Bestimmung von *space* ignoriert.

_ (Unterstrich) - *non-space*_*non-space-string*, wobei *non-space-string* eine Zeichenkette aus Nicht-Leerzeichen ist, stellt *non-space-string* tiefer.

`<HTML>`...`</HTML>` - konvertiert nicht „<“ und „>“ im enthaltenden Text, wenn HTML generiert wird. Ansonsten wird die Textauszeichnungen normal verarbeitet. Dies ermöglicht das direkte Einfügen von HTML-Befehlen.

„`&ref;`“ - fügt eine HTML-Zeichenreferenz ein. So können Sie beliebige Unicode-Zeichen einfügen (z. B. Zeichen mit Akzent). Derzeit nur direkt unterstützt, wenn HTML erzeugt wird.

Es wird empfohlen, alle ~- und ‘-Token im Kommentar mit Leerzeichen zu umgeben und ein Leerzeichen an das einem ~-Token folgenden *Label* anzuhängen. Dadurch werden globale Ersetzungen zum Ändern von Labels und Symbolnamen viel einfacher, und auch die Gefahr von Mehrdeutigkeiten wird in Zukunft ausgeschlossen. Leerzeichen um diese Zeichen herum werden in der endgültigen Ausgabe automatisch entfernt, um den normalen Interpunktionsregeln zu entsprechen; zum Beispiel wird ein Leerzeichen zwischen einem nachgestellten ‘ und einer linken Klammer entfernt.

Eine gute Möglichkeit, sich mit den Textauszeichnungen vertraut zu machen, ist die Betrachtung der umfangreichen Beispiele in der Datenbasis `set.mm`.

4.4.2 Der Schriftsatz-Kommentar (`$t`)

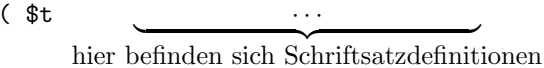
Der Schriftsatzkommentar `$t` in der Datenbasisdatei liefert die erforderlichen Informationen, um gut aussehende Ausgaben zu erzeugen. Er bietet LaTeX- und HTML-Definitionen für mathematische Symbole sowie Unterstützung und Anpassungen bei der Generierung von Web-Seite. Wenn Sie ein neues Token zu einer Datenbasis hinzufügen und später eine Ausgabe in LaTeX oder HTML erstellen möchten, dann sollten Sie auch die `$t`-Kommentarinformationen aktualisieren. In der Datenbasisdatei `set.mm` finden Sie ein ausführliches Beispiel für einen `$t`-Kommentar, das viele der unten beschriebenen Funktionen verdeutlicht.

Programme, die keine gut aussehenden Ausgaben für eine Präsentation erzeugen müssen, wie z. B. Programme, die nur Metamath-Datenbasen überprüfen, können Schriftsatzkommentare vollständig ignorieren und sie einfach als normale Kommentare behandeln. Selbst das Metamath-Programm konsultiert die `$t`-Kommentarinformationen nur dann, wenn es eine gesetzte Ausgabe in LaTeX oder HTML erzeugen muss (z. B. wenn Sie eine LaTeX-Ausgabedatei mit dem Befehl `open tex` öffnen).

Wir werden zunächst die Syntax von Schriftsatzkommentaren besprechen und dann kurz darauf eingehen, wie diese innerhalb des Metamath-Programms verwendet werden können.

Übersicht über die Syntax von Schriftsatzkommentaren

Der Schriftsatzkommentar wird durch das Token `$t` im Kommentar identifiziert, und der Schriftsatzkommentar endet an dem passenden `$)`:



Zwischen dem `$(`, mit dem der Kommentar beginnt, und dem Symbol `$t` muss ein Whitespace enthalten sein, und zwar ausschließlich, und auf `$t` muss wieder ein Whitespace folgen (siehe Abschnitt 4.1.1 zur Definition von Whitespace). Der Schriftsatzkommentar wird bis zum Kommentarende-Token `$)` fortgesetzt (dem ein Whitespace vorausgehen muss).

In der Version 0.177 des Metamath-Programms darf es nur einen `$t`-Kommentar in einer Datenbasis geben. Diese Einschränkung kann in Zukunft aufgehoben werden, um mehrere `$t`-Kommentare in einer Datenbasis zu ermöglichen.

Zwischen dem Symbol `$t` (und dem folgenden Whitespace) und dem Kommentarende-Token `$)` (und dem vorangehenden Whitespace) befindet sich eine Folge von einer oder mehreren Schriftsatzdefinitionen, wobei jede Definition die Form *definition-type arg arg ... ;* hat. Jeder der null oder mehr *arg*-Werte kann entweder ein Schriftsatzdatenwert oder ein Schlüsselwort sein (welche Schlüsselwörter wo erlaubt sind, hängt vom spezifischen *definition-type* ab). Der *definition-type* und jedes Argument *arg* werden durch einen Whitespace getrennt. Jede Definition endet mit einem Semikolon; ein Whitespace ist vor dem abschließenden Semikolon einer Definition nicht erforderlich. Jede Definition sollte in einer neuen Zeile beginnen.[22]

Als Beispiel für solch eine Schriftsatzdefinition definiert

```
latexdef "C_" as "\subseteq";
```

für das Token `C_` das LaTeX-Symbol $\subseteq$ (was „Teilmenge“ bedeutet).

[22] Diese Einschränkung der aktuellen Version von Metamath (0.177) wird vielleicht in einer zukünftigen Version entfernt, aber man sollte es aus Gründen der Lesbarkeit trotzdem tun.

Schriftsatzdaten sind eine Folge von einer oder mehreren Zeichenkette in Anführungszeichen (bei mehreren Zeichenkette in Anführungszeichen werden sie durch `+` verbunden). Häufig wird eine einzelne Zeichenkette in Anführungszeichen verwendet, um Daten für eine Definition bereitzustellen, wobei entweder doppelte (`"`) oder einfache (`'`) Anführungszeichen verwendet werden. *Eine in Anführungszeichen eingeschlossene Zeichenkette darf jedoch keine Zeilenumbrüche enthalten.* Eine in Anführungszeichen gesetzte Zeichenkette kann ein Anführungszeichen enthalten, das mit den eingeschlossenen Anführungszeichen übereinstimmt, indem das Anführungszeichen zweimal wiederholt wird. Hier sind einige Beispiele:

Beispiel	**Bedeutung**
`"a""b"`	`a"b`
`'c''d'`	`c'd`
`"e''f"`	`e''f`
`'g""h'`	`g""h`

Schließlich kann eine lange in Anführungszeichen gesetzte Zeichenkette in mehrere in Anführungszeichen gesetzte Zeichenketten aufgeteilt werden (die als Ganzes als eine einzige in Anführungszeichen gesetzte Zeichenkette betrachtet werden) und mit `+` verbunden werden. Sie können sogar mehrere Zeilen verwenden, solange sich am Ende jeder Zeile außer der letzten ein '+' befindet. Vor und nach `+` sollte ein Whitespace stehen. Also zum Beispiel,

```
"ab" + "cd" +
    'ef'
```

ist dasselbe wie

```
"abcdef"
```

Kommentare im C-Stil `/*...*/` werden ebenso unterstützt.

In der Praxis werden Sie oft Satzdefinitionen mit `latexdef`, `htmldef` und `althtmldef` wie unten beschrieben hinzufügen wollen, wenn Sie ein neues mathematisches Token ergänzen. Auf diese Weise werden sie alle auf dem neuesten Stand sein. Ob Sie alle drei Definitionen verwenden wollen oder nicht, hängt natürlich davon ab, wie die Datenbasis verwendet werden soll.

Im Folgenden werden die verschiedenen möglichen *definition-type*-Optionen erörtert. Wir zeigen die Daten in doppelten Anführungszeichen (in der Praxis können sie auch in einfachen Anführungszeichen stehen und/oder eine durch `+`s verbundene Sequenz sein). Wir werden spezifische Namen für die *data* verwenden, um zu verdeutlichen, wofür die Daten verwendet werden, z. B. *Mathe-Token* (für ein Metamath-Mathematik-Token), *Latex-string* (für eine Zeichenkette, die in einen LaTeX-Text eingefügt werden soll), HTML-*code* (für HTML-Code) und *filename* (für einen Dateinamen).

Schriftsatzkommentar - LaTeX

Die Syntax für eine LaTeX-Definition lautet:

`latexdef "`*math-token*`" as "`*latex-string*`";`

token-string und *latex-string* sind die Daten (Zeichenketten) für das Token bzw. die LaTeX-Definition des Tokens.

Diese LaTeX-Definitionen werden vom Metamath-Programm verwendet, wenn es mit dem Befehl `write tex` eine LaTeX-Ausgabe erzeugen soll.

Schriftsatzkommentar - HTML

Die wichtigsten Arten von HTML-Definitionen haben die folgende Syntax:

`htmldef "`*math-token*`" as "`HTML-*code*`";`
`althtmldef "`*math-token*`" as "`HTML-*code*`";`

Beachten Sie, dass es in HTML zwei mögliche Definitionen für mathematische Token gibt. Diese Funktionalität ist nützlich, wenn eine alternative Darstellung von Symbolen gewünscht wird, zum Beispiel eine, die Unicode-Entities verwendet, und eine andere, die GIF-Bilder verwendet.

Es gibt viele andere Schrifsatzdefinitionen, die HTML steuern können. Dazu gehören:

`htmltitle "`HTML-*code*`";`
`htmlhome "`HTML-*code*`";`
`htmlvarcolor "`HTML-*code*`";`
`htmlbibliography "`*filename*`";`

Der `htmltitle` ist der HTML Code für einen allgemeinen Titel, wie z.B. „Metamath Proof Explorer“. Das Feld `htmlhome` ist der Code für einen Link zurück zur Startseite. `htmlvarcolor` ist der Code für einen Farbschlüssel, der am unteren Rand jedes Beweises erscheint. Die durch *filename* angegebene Datei ist eine HTML-Datei, die ein `<A NAME=...>`-Tag für jeden Verweis in das Literaturverzeichnis in den Datenbasiskommentaren enthäten sollte. Wenn zum Beispiel `[Monk]` im Kommentar für ein Theorem vorkommt, muss `<A NAME='Monk'>` in der Datei vorhanden sein; andernfalls wird eine Warnmeldung ausgegeben.

Mit `htmldef` und `althtmldef` verbunden sind die Anweisungen

`htmldir "`*directoryname*`";`
`althtmldir "`*directoryname*`";`

geben die Verzeichnisse der GIF- bzw. Unicode-Versionen an; ihr Zweck ist es, Querverbindungen zwischen den beiden Versionen in den erzeugten Webseiten herzustellen.

Wenn zwei verschiedene Arten von Seiten aus einer einzigen Datenbasis erzeugt werden müssen, wie z.B. der Hilbert Space Explorer, der den

Metamath Proof Explorer erweitert, können „erweiterte“ Variablen in dem `$t`-Kommentar deklariert werden:

```
exthtmltitle "HTML-code";
exthtmlhome "HTML-code";
exthtmlbibliography "filename";
```

Wenn diese deklariert werden, müssen Sie auch Folgendes deklarieren

```
exthtmllabel "label";
```

welches die Anweisung der Datenbasis identifiziert, mit der der „erweiterte“ Abschnitt der Datenbasis (in unserem Beispiel der Hilbert Space Explorer) beginnt. Bei der Generierung von Webseiten für diese erste Anweisung und die nachfolgenden Anweisungen wird der HTML-Code, der `exthtmltitle` und `exthtmlhome` zugewiesen ist, anstelle des Codes verwendet, der `htmltitle` bzw. `htmlhome` zugewiesen ist.

4.4.3 Zusatzinformationskommentar (`$j`)

Der Zusatzinformationskommentar, auch bekannt als `$j`-Kommentar, bietet eine Möglichkeit, zusätzliche strukturierte Informationen hinzuzufügen, die optional von Systemen geparst werden können.

Der Zusatzinformationskommentar wird auf die gleiche Weise geparst wie der Schriftsatzkommentar (`$t`) (siehe Abschnitt 4.4.2). Das heißt, der Zusatzinformationskommentar beginnt mit dem Token `$j` innerhalb eines Kommentars und wird bis zum Kommentarschluss `$)` fortgesetzt. Innerhalb eines Zusatzinformationskommentars befindet sich eine Folge von einem oder mehreren Befehlen der Form `command arg arg ... ;`, wobei jeder der null oder mehr `arg`-Werte entweder eine Zeichenkette in Anführungszeichen oder ein Schlüsselwort sein kann. Beachten Sie, dass jeder Befehl mit einem nicht in Anführungszeichen gesetztes Semikolon endet. Wenn ein Prüfprogramm einen Zusatzinformationskommentar parst, aber einen bestimmten Befehl nicht erkennt, muss er den Befehl überspringen, indem er das Ende des Befehls (ein nicht in Anführungszeichen gesetztes Semikolon) sucht.

Eine Datenbasis kann keine, eine oder mehr Zusatzinformationskommentare haben. Beachten Sie jedoch, dass ein Prüfprogramm diese Kommentare vollständig ignorieren oder nur bestimmte Befehle in einem Zusatzinformationskommentar verarbeiten kann. Der `mmj2` Verifier unterstützt viele Befehle in Zusatzinformationskommentaren. Wir empfehlen Systemen, die Zusatzinformationskommentare verarbeiten, sich abzustimmen, so dass sie denselben Befehl für denselben Effekt verwenden.

Beispiele für Zusatzinformationskommentare mit verschiedenen Befehlen (aus der Datenbasis `set.mm`) sind:

- Definition der Syntax und der logischen Typcodes und Deklaration, dass unsere Grammatik eindeutig ist (überprüfbar mit dem KLR-Parser, mit Kompositionstiefe 5).

```
$( $j
  syntax 'wff';
  syntax '|-' as 'wff';
  unambiguous 'klr 5';
$)
```

- Registrierung von $\neg$ und $\rightarrow$ als primitive Ausdrücke (ohne Definitionen).

```
$( $j primitive 'wn' 'wi'; $)
```

- Für `df-bi` gibt es eine besondere Rechtfertigung.

```
$( $j justification 'bijust' for 'df-bi'; $)
```

- Registrierung von $\leftrightarrow$ als eine Gleichheit für seinen Typ (wff).

```
$( $j
  equality 'wb' from 'biid' 'bicomi' 'bitri';
  definition 'dfbi1' for 'wb';
$)
```

- Theorem `notbii` ist das Kongruenzgesetz für die Negation.

```
$( $j congruence 'notbii'; $)
```

- Ergänzung von `setvar` als Typcode.

```
$( $j syntax 'setvar'; $)
```

- Registrierung von $=$ als Gleichheit für seinen Typ (`class`).

```
$( $j equality 'wceq' from 'eqid' 'eqcomi' 'eqtri'; $)
```

4.4.4 Einbindung anderer Dateien in eine Metamath-Quelldatei

Die Schlüsselwörter `$[` und `$]` spezifizieren eine einzubindende Datei an dieser Stelle in einer Metamath-Quelldatei. Die Syntax für die Einbindung einer Datei lautet wie folgt:

`$[` *file-name* `$]`

Der Dateiname *file-name* sollte ein einzelnes Token mit der gleichen Syntax wie ein mathematisches Symbol sein (d. h. alle 93 druckbaren Zeichen ohne Leerzeichen außer `$` sind zulässig, vorbehaltlich der Dateinamensbeschränkungen Ihres Betriebssystems). Zwischen den Schlüsselwörtern `$[` und `$]` können Kommentare stehen. Eingebundene Dateien können andere Dateien einbinden, die wiederum andere Dateien einbinden können, und so weiter.

Nehmen wir zum Beispiel an, Sie möchten die Datenbasis der Mengenlehre als Ausgangspunkt für Ihre eigene Theorie verwenden. Die erste Zeile in Ihrer Datei könnte lauten:

```
$[ set.mm $]
```

Alle Informationen (Axiome, Theoreme usw.) in `set.mm` und alle Dateien, die *sie selbst wiederum* einschließt, stehen dann Ihnen zur Verfügung, damit Sie in Ihrer Datei darauf verweisen können. Dies ermöglicht eine modularen Aufbau Ihrer eigenen Arbeit. Ein Nachteil des Einbebindens von Dateien ist, dass Sie, wenn Sie den Namen eines Symbols oder das Label einer Anweisung ändern, auch daran denken müssen, alle Verweise in jeder Datei, die sie einbindet, zu aktualisieren.

Die Namenskonventionen für eingebundene Dateien entsprechen denen Ihres Betriebssystems.[23] Um die Kompatibilität zwischen verschiedenen Betriebssystemen zu gewährleisten, sollten Sie die Dateinamen so einfach wie möglich halten. Eine gute Konvention ist *file*`.mm`, wobei *file* aus acht Zeichen oder weniger in Kleinbuchstaben besteht.

Es gibt keine Begrenzung für die Verschachtelungstiefe von eingebundenen Dateien. Sie sollten jedoch beachten, dass, wenn zwei eingebundene Dateien selbst eine gemeinsame dritte Datei einbinden, diese nur beim *ersten* Verweis auf diese gemeinsame Datei eingelesen wird. Dies ermöglicht es Ihnen, zwei oder mehr Dateien einzubinden, die auf einer gemeinsamen Ausgangsdatei aufbauen, ohne sich um Label- und Symbolkonflikte sorgen zu müssen, die auftreten würden, wenn die gemeinsame Datei mehr als einmal eingelesen würde. (Wenn eine Datei sich selbst einbindet, wobei das natürlich nicht sinnvoll wäre, wird die Selbstreferenz ignoriert). Dieses Vorgehen bedeutet jedoch auch, dass das Ergebnis möglicherweise nicht das ist, was Sie erwarten, wenn Sie versuchen, eine gemeinsame Datei in mehreren inneren Blöcken einzubinden, da nur die erste Referenz durch die eingebundene Datei ersetzt wird (im Gegensatz zur include-Anweisung in den meisten anderen Computersprachen). Daher würden Sie normalerweise gemeinsame Dateien nur im äußersten Block einbinden.

[23] Auf dem Macintosh, vor Mac OS X, wird ein Doppelpunkt verwendet, um Laufwerke- und Ordnernamen von Ihrem Dateinamen zu trennen. Zum Beispiel bezieht sich *volume*:*file-name* auf das Stammverzeichnis, *volume*:*folder-name*:*file-name* auf einen Ordner im Stammverzeichnis, und *volume*:*folder-name*:...:*file-name* auf einen Unterordner. Ein einfacher *Dateiname* verweist auf eine Datei in dem Ordner, aus dem Sie die Metamath-Anwendung starten. Unter Mac OS X und später wird das Metamath-Programm unter der Terminal-Anwendung ausgeführt, die den Unix-Namenskonventionen entspricht.

4.4.5 Komprimiertes Beweisformat

Die in Abschnitt 4.3 vorgestellte Beweisschreibweise wird als **normaler Beweis** bezeichnet und ist im Prinzip ausreichend, um jeden Beweis vollständig zu beschreiben. Beweise enthalten jedoch oft Schritte und Unterbeweise, die identisch sind. Dies gilt insbesondere für typische Metamath-Anwendungen, da Metamath verlangt, dass die mathematische Symbolfolge (die in der Regel eine Formel enthält) bei jedem Schritt separat konstruiert, d. h. Stück für Stück aufgebaut wird. Daraus ergibt sich oft eine große Anzahl von Wiederholungen. Das **komprimierte Beweisformat** ermöglicht es Metamath, diese Redundanz zu nutzen, um Beweise zu verkürzen.

Die Spezifikation für das komprimierte Format der Beweise ist in Anhang B enthalten.

Normalerweise brauchen Sie sich nicht mit den Details des komprimierten Beweisformats zu befassen, da das Programm Metamath eine bequeme Konvertierung vom normalen Format in das komprimierte Format ermöglicht und auch automatisch vom komprimierten Format in das normale Format konvertiert, wenn Beweise angezeigt werden. Die allgemeine Struktur des komprimierten Formats ist wie folgt:

`$=` `(` *label-list* `)` *compressed-proof* `$.`

Die erste Klammer `(` dient als Kennzeichen für Metamath, dass ein komprimierter Beweis folgt. Die *label-list* enthält alle Anweisungen, auf die sich der Beweis bezieht, mit Ausnahme der obligatorischen Hypothesen. Das *compressed-proof* ist eine kompakte Kodierung des Beweises unter Verwendung von Großbuchstaben und kann als eine große ganze Zahl zur Basis 26 angesehen werden. Das Whitespace innerhalb eines *compressed-proof* ist optional und wird ignoriert.

Es ist wichtig zu beachten, dass die Reihenfolge der obligatorischen Hypothesen der zu beweisenden Anweisung nicht geändert werden darf, wenn das komprimierte Beweisformat verwendet wird, da der Beweis sonst falsch wird. Der Grund dafür ist, dass die obligatorischen Hypothesen im komprimierten Beweis nicht explizit erwähnt werden, um die Komprimierung effizienter zu gestalten. Wenn Sie die Reihenfolge der obligatorischen Hypothesen ändern möchten, müssen Sie den Beweis zunächst wieder in das normale Format konvertieren, indem Sie die Anweisung `save proof` *`statement`* `/normal` ausführen. Später können Sie mit dem Befehl `save proof` *`statement`* `/compressed` wieder in das komprimierte Format wechseln.

Bei der Fehlerprüfung mit dem Befehl `verify proof` kann ein in einem komprimierten Beweis gefundener Fehler auf ein Zeichen in *compressed-proof* verweisen, das für Sie möglicherweise nicht sehr aussagekräftig ist. Versuchen Sie in diesem Fall zuerst `save proof /normal` und führen Sie dann den Befehl `verify proof` erneut aus. Im Allgemeinen ist es am besten sich zu vergewissern, dass ein Beweis korrekt ist, bevor man ihn im komprimier-

ten Format speichert, da schwere Fehler mit geringerer Wahrscheinlichkeit wiederhergestellt werden können als im normalen Format.

4.4.6 Unbekannte Beweise oder Teilbeweise

In einem in Entwicklung befindlichen Beweis kann jeder Schritt oder Teilbeweis, der noch nicht bekannt ist, mit einem einzelnen `?` dargestellt werden. Beim Parsen des Beweises wird bei einem `?` ein einzelner Eintrag auf den RPN-Stapel geschoben, als wäre es eine Hypothese. Während der Entwicklung eines Beweises mit dem Beweis-Assistenten kann ein teilweise entwickelter Beweis mit dem Befehl `save new_proof` gespeichert werden, und die `?`'s werden an die entsprechenden Stellen gesetzt.

Für alle `$p`-Anweisungen müssen Beweise vorliegen, auch wenn sie völlig unbekannt sind. Bevor Sie einen Beweis mit dem Beweis-Assistenten erstellen, sollten Sie einen völlig unbekannten Beweis wie folgt angeben:

label `$p` *statement* `$= ? $.`

Der Befehl `verify proof` prüft die bekannten Teile eines Teilbeweises auf Fehler, warnt Sie aber, dass die Anweisung nicht bewiesen ist.

Beachten Sie, dass teilweise entwickelte Beweise auf Wunsch im komprimierten Format gespeichert werden können. In diesem Fall sehen Sie einen oder mehrere `?`'s im Teil *compressed-proof*.

4.5 Axiome vs. Definitionen

Die Metamath *zugrunde liegende* Sprache und das Metamath-Programm unterscheiden nicht zwischen Axiomen und Definitionen. Die `$a`-Anweisung wird für beides verwendet. Auf den ersten Blick mag dies seltsam erscheinen. In den Augen vieler Mathematiker ist die Unterscheidung klar, sogar offensichtlich, und kaum eine Diskussion wert. Eine Definition wird lediglich als eine Abkürzung betrachtet, die durch den Ausdruck, für den sie steht, ersetzt werden kann; wenn man dies jedoch nicht tut, sollte man sagen, dass ein Theorem eine Folge der Axiome *und* der Definitionen ist, die bei der Formulierung des Theorems verwendet werden [4, S. 20].

4.5.1 Was ist eine Definition?

Was ist eine Definition? In ihrer einfachsten Form führt eine Definition ein neues Symbol ein und liefert eine eindeutige Regel zur Umwandlung eines Ausdrucks, der das neue Symbol enthält, in einen Ausdruck ohne dieses Symbol. Das Konzept einer „zulässigen Definition“ (im Gegensatz zu einer kreativen Definition), auf das man sich in der Regel einigt, ist, dass (1) die Definition die Sprache nicht mächtiger machen sollte und (2) alle durch die Definition eingeführten Symbole aus der Sprache eliminierbar sein sollten

[49]. Mit anderen Worten, sie sind bloße typografische Bequemlichkeiten, die nicht zum System gehören und theoretisch überflüssig sind. Dies mag offensichtlich erscheinen, aber in der Tat kann die Natur von Definitionen subtil sein und erfordert manchmal komplexe Metatheoreme, um zu beweisen, dass sie nicht kreativ sind.

Eine konservativere Haltung vertrat der Logiker S. Leśniewski.

> Leśniewski betrachtet Definitionen als Thesen des Systems. In dieser Hinsicht unterscheiden sie sich weder von den Axiomen noch von den Theoremen, d.h. von den Thesen, die dem System auf der Grundlage der Substitutionsregel oder der Abtrennregel [Modus ponens] hinzugefügt werden. Sobald Definitionen als Thesen des Systems akzeptiert wurden, müssen sie als wahre Sätze in demselben Sinne betrachtet werden, in dem auch Axiome wahr sind [37].

Sehen wir uns einige einfache Beispiele für Definitionen in der Aussagenlogik an. Betrachten wir die Definition des logischen OR (Disjunktion): „$P \vee Q$ steht für $\neg P \to Q$ (nicht P impliziert Q)“. Eine Anweisung, die von dieser Definition Gebrauch macht, ist sehr leicht zu erkennen, weil sie das neue Symbol $\vee$ verwendet, das es vorher in der Sprache nicht gab. Es ist leicht zu erkennen, dass sich aus dieser Definition keine neuen Theoreme der ursprünglichen Sprache ergeben.

Betrachten wir nun eine Definition, bei der die Klammern entfallen: „$P \to Q \to R$ bedeutet $P \to (Q \to R)$.“ Dies ist subtiler, da keine neuen Symbole eingeführt werden. Der Grund, warum diese Definition als richtig angesehen wird, ist, dass sich aus der Definition keine neuen Symbolfolgen ergeben, die in der ursprünglichen Sprache gültige wffs (wohlgeformte Formeln) sind, da „$P \to Q \to R$“ keine wff in der ursprünglichen Sprache ist. Wir machen hier implizit Gebrauch von der Tatsache, dass es ein Entscheidungsverfahren gibt, mit dem wir feststellen können, ob eine Symbolfolge eine wff ist oder nicht, und diese Tatsache erlaubt es uns, Symbolfolgen, die keine wffs sind, zu verwenden, um andere Dinge (wie wffs) mit Hilfe der Definition darzustellen. Um jedoch zu rechtfertigen, dass die Definition nicht kreativ ist, müssen wir beweisen, dass „$P \to Q \to R$“ in der ursprünglichen Sprache tatsächlich keine wff ist, und das ist schwieriger als in dem Fall, in dem wir einfach ein neues Symbol einführen.

Was eine Definition von einem Axiom unterscheidet ist in der mathematischen Literatur manchmal willkürlich. Zum Beispiel werden die Junktoren $\vee$ (ODER), $\wedge$ (UND) und $\leftrightarrow$ (äquivalent zu) in der Aussagenlogik gewöhnlich als definierte Symbole betrachtet, die als Abkürzungen für Ausdrücke verwendet werden können, die die „primitiven“ Junktoren $\to$ und $\neg$ enthalten. So werden sie auch in der Standard-Datenbasis für Logik und Mengenlehre `set.mm` behandelt. Die ersten drei Junktoren können jedoch auch als „primitiv“ betrachtet werden, und es wurden Axiomensysteme entwickelt, die alle diese Junktoren als solche behandeln. So enthält z. B. [19, S. 35] 15

Axiome, von denen einige mit dem übereinstimmen, was wir in `set.mm` als Definitionen bezeichnet haben. In bestimmten Teilmengen der klassischen Aussagenlogik, wie dem intuitionistischen Fragment, kann man zeigen, dass man nicht nur mit $\rightarrow$ und $\neg$ auskommen kann, sondern zusätzliche Junktoren als primitiv behandeln muss, damit das System einen Sinn ergibt.[24]

4.5.2 Der Ansatz für Definitionen in `set.mm`

In der Mengenlehre definieren rekursive Definitionen ein neu eingeführtes Symbol in Bezug auf sich selbst. Die Begründung rekursiver Definitionen mit Hilfe mehrerer „Rekursionstheoreme“ ist normalerweise einer der ersten anspruchsvollen Beweise, mit denen ein Student beim Erlernen der Mengenlehre konfrontiert wird, und hinter einer rekursiven Definition steckt eine beträchtliche Menge impliziter Metalogik, obwohl die Definition selbst normalerweise einfach zu formulieren ist.

Metamath selbst hat keine eingebauten technischen Einschränkungen, die mehrteilige rekursive Definitionen im traditionellen Lehrbuchstil verhindern. Da die rekursive Definition jedoch eine fortgeschrittene Metalogik erfordert, um sie zu begründen, ist die Eliminierung einer rekursiven Definition sehr schwierig und wird in Lehrbüchern oft nicht einmal gezeigt.

Direkte Definitionen anstelle von rekursiven Definitionen

Es ist jedoch möglich, eine Art von Komplexität durch eine andere zu ersetzen. Wir können die Notwendigkeit einer metalogischen Begründung vermeiden, indem wir die Operation direkt mit einem expliziten (aber komplizierten) Ausdruck definieren und dann die rekursive Definition direkt als Theorem ableiten, indem wir ein Rekursionstheorem „in die andere Richtung“ verwenden. Die Eliminierung einer direkten Definition erfolgt diurch eine einfache mechanische Substitution. Wir tun dies in `set.mm` wie folgt.

In `set.mm` war es unser Ziel, fast alle Definitionen in Form von zwei Ausdrücken einzuführen, die entweder durch $\leftrightarrow$ oder $=$ verbunden sind, wobei das Definierte nicht auf der rechten Seite erscheint. Quine nennt diese Form „eine echte oder direkte Definition“ [55, S. 174], wodurch die Definitionen sehr leicht zu eliminieren sind und die Metalogik, die zu ihrer Rechtfertigung erforderlich ist, so einfach wie möglich ist. Anders ausgedrückt: Unser Ziel war es, alle Definitionen durch direkte mechanische Substitutionen zu eliminieren und die Fundiertheit der Definitionen leicht zu überprüfen.

[24] Zwei schöne Systeme, die den Übergang von dem intuitionistischen und anderen schwachen Fragmenten zur klassischen Logik nur durch Hinzufügen von Axiomen schaffen, sind in [57] angegeben.

Beispiel für direkte Definitionen

Wir haben dieses Ziel in fast allen Fällen in `set.mm` erreicht. Manchmal macht dies die Definitionen komplexer und weniger intuitiv. Die traditionelle Art, die Addition natürlicher Zahlen zu definieren, besteht zum Beispiel darin, eine Operation namens *Nachfolger* zu definieren (was „plus eins“ bedeutet und mit „suc“ bezeichnet wird), dann die Addition rekursiv zu definieren mit den beiden Definitionen $n+0=n$ und $m+\operatorname{suc} n=\operatorname{suc}(m+n)$. Obwohl diese Definition einfach und offensichtlich erscheint, ist die Methode zur Eliminierung der Definition nicht offensichtlich: Im zweiten Teil der Definition wird die Addition in Bezug auf sich selbst definiert. Mit der Eliminierung der Definition ist nicht gemeint, dass man sie wiederholt auf bestimmte m und n anwendet, sondern dass man den expliziten, in sich geschlossenen mengentheoretischen Ausdruck angibt, den $m+n$ darstellt, der für beliebige m und n gilt und der kein $+$-Zeichen auf der rechten Seite hat. Damit eine rekursive Definition wie diese nicht zirkulär (kreativ) ist, müssen wir einige versteckte, zugrundeliegende Annahmen machen, zum Beispiel, dass die natürlichen Zahlen eine bestimmte Ordnung haben.

In `set.mm` haben wir uns entschieden, mit der direkten (wenn auch komplexen und nicht intuitiven) Definition zu beginnen und daraus die rekursive Standarddefinition abzuleiten. Die geschlossene Definition, die in `set.mm` für die Additionsoperation von Ordinalzahlen (von denen die natürlichen Zahlen eine Teilmenge sind) verwendet wird, lautet beispielsweise

`df-oadd $a` $\vdash +_o = (x \in \mathrm{On} , y \in \mathrm{On} \mapsto (\mathrm{rec} ((z \in \mathrm{V} \mapsto \mathrm{suc}\, z) , x) \mathbin{`} y))$

welche vom Operator rec (siehe folgenden Abschnitt) abhängt.

Rekursionsoperatoren

Die obige Definition von `df-oadd` hängt von der Definition von rec ab, einem „Rekursionsoperator“ mit der Definition `df-rdg`:

`df-rdg $a` $\vdash \mathrm{rec} (F , I) = \mathrm{recs} ((g \in \mathrm{V} \mapsto \mathrm{if} (g = \varnothing , I , \mathrm{if} (\mathrm{Lim}\, \mathrm{dom}\, g , \bigcup \mathrm{ran}\, g , (F \mathbin{`} ( g \mathbin{`} \bigcup \mathrm{dom}\, g))))))$

die anhand der Definitionen in Abschnitt 3.4.3 weiter heruntergebrochen werden könnte.

Diese Definition von rec definiert einen rekursiven Definitionsgenerator auf On (der Klasse der Ordinalzahlen) mit charakteristischer Funktion F und Anfangswert I. Diese Operation erlaubt es uns, mit kompakten direkten Definitionen Funktionen zu definieren, die normalerweise in Lehrbüchern mit rekursiven Definitionen definiert werden. Der Preis, den wir mit unserem Ansatz zahlen, ist die Komplexität unserer rec-Operation (insbesondere, wenn `df-recs`, auf dem sie aufbaut, ebenfalls eliminiert werden soll). Aber sobald wir diese Hürde überwunden haben, werden Definitionen, die sonst rekursiv wären, relativ einfach, wie zum Beispiel `oav`, aus dem wir die rekursive Lehrbuchdefinition als Theoreme `oa0`, `oasuc` und `oalim` beweisen (mit Hilfe der Theoreme `rdg0`, `rdgsuc` und `rdglim2a`). Wir können die rec-

Operation auch einschränken, um rekursive Funktionen auf den natürlichen Zahlen ω zu definieren; siehe `fr0g` und `frsuc`. Unsere rec-Operation taucht in der veröffentlichten Literatur offenbar nicht auf, obwohl sie eng mit der Definition 25.2 von [Quine] S. 177 verwandt ist, die er verwendet, „um eine Rekursion in eine echte oder direkte Definition zu verwandeln“ (S. 174). Man beachte, dass die if-Operationen (siehe `df-if`) Fälle danach auswählen, ob der Definitionsbereich von g die leere Menge, ein Nachfolger oder eine Grenzordinale ist.

Eine wichtige Anwendung dieser Definition rec ist der rekursive Sequenzgenerator `df-seq` auf den natürlichen Zahlen (als Teilmenge der komplexen unendlichen Sequenzen, wie der Fakultätsfunktion `df-fac` und den ganzzahligen Potenzen `df-exp`).

Die Definition von rec hängt von recs ab. Von der direkten Verwendung des mächtigeren (und primitiveren) recs-Konstrukts wird abgeraten, es ist aber bei Bedarf verfügbar. Dies definiert eine Funktion recs(F) auf On, der Klasse der Ordinalzahlen, durch transfinite Rekursion unter der Voraussetzung einer Regel F, die den nächsten Wert unter Berücksichtigung aller bisherigen Werte bestimmt. Im Gegensatz zu `df-rdg`, das die Aktualisierungsregel darauf beschränkt, nur den vorherigen Wert zu verwenden, erlaubt diese Version der Aktualisierungsregel, alle vorherigen Werte zu verwenden, weshalb sie als „stark“ bezeichnet wird, obwohl sie eigentlich primitiver ist. Siehe `recsfnon` und `recsval` für die primären Eigenschaften dieser Definition. Sie ist definiert als:

`df-recs $a` $\vdash \mathrm{recs}(F) = \bigcup \{ f \mid \exists x \in \mathrm{On} (f \mathrm{Fn} x \wedge \forall y \in x (f ` y ) = ( F ` (f \restriction y))) \}$

Abschließende Bemerkungen zu direkten Definitionen

Aus diesen direkten Definitionen wird die einfachere, intuitivere rekursive Definition als eine Reihe von Theoremen abgeleitet. Das Endergebnis ist dasselbe, aber wir können dadurch vollständig auf die recht komplexe Metalogik verzichten, die die rekursive Definition rechtfertigt.

Rekursive Definitionen werden oft als effizienter und intuitiver angesehen als direkte Definitionen, sobald die Metalogik erlernt oder möglicherweise einfach als korrekt akzeptiert wurde. Man war jedoch der Ansicht, dass die direkte Definition in `set.mm` die Strenge maximiert, indem sie die Metalogik minimiert. Eine solche Definition kann mühelos eliminiert werden, was bei einer rekursiven Definition nur schwer möglich ist.

Auch hier gilt, dass Metamath selbst keine eingebauten technischen Einschränkungen hat, die mehrteilige rekursive Definitionen im traditionellen Lehrbuchstil verhindern. Stattdessen ist es unser Ziel, alle Definitionen mit direkter mechanischer Substitution zu eliminieren und die Fundiertheit der Definitionen leicht zu überprüfen.

4.5.3 Hinzufügen von Einschränkungen für Definitionen

Die Metamath-Basissprache und das Metamath-Programm haben keine eingebauten einschrämkungen für Definitionen, da sie nur `$a`-Anweisungen sind.

Nichts hindert jedoch ein Verifikationssystem daran, zusätzliche Regeln zu verifizieren, um weitere Einschränkungen für Definitionen vorzunehmen. Das `mmj2`-Programm unterstützt zum Beispiel verschiedene Arten von zusätzlichen Informationskommentaren (siehe Abschnitt 4.4.3). Eine ihrer Verwendungen ist die optionale Überprüfung zusätzlicher Einschränkungen, einschließlich der Überprüfung, ob Definitionen bestimmte Anforderungen erfüllen. Diese zusätzlichen Prüfungen werden von der kontinuierlichen Integration (CI) der `set.mm` Datenbasis benötigt. Dieser Ansatz ermöglicht es uns, optional zusätzliche Anforderungen an Definitionen zu stellen, wenn wir dies wünschen, ohne dass diese Regeln notwendigerweise für alle Datenbasen gelten oder von allen Verifikationssystemen verlangen zu müssen diese zu prüfen. Außerdem können wir auf diese Weise spezielle, auf eine Datenbasis zugeschnittene Einschränkungen vornehmen, ohne dass andere Systeme diese speziellen Einschränkungen berücksichtigen müssen.

Es gibt zwei solcher Einschränkungen in der Datenbasis `set.mm`, die meit dem Programm `mmj2` geprüft werden, die es wert sind, hier besprochen zu werden: eine Syntax-Prüfung und eine Prüfung der Fundiertheit von Definitionen.

Erstens aktivieren wir in `mmj2` (über den Befehl `SetParser`) eine Syntax-Prüfung, die vorschreibt, dass alle neuen Definitionen für einen KLR(5)-Parser kein mehrdeutiges Ergebnis erzeugen dürfen. Dies verhindert einige Fehler wie z. B. Definitionen mit unausgeglichenen Klammern.

Zweitens führen wir eine Definitionsprüfung durch, die spezifisch für `set.mm` oder ähnliche Datenbasen ist (über das Makro `definitionCheck`). Einige `$a`-Anweisungen (einschließlich aller ax-*-Statements) sind von diesen Prüfungen ausgenommen, da sie diese einfache Prüfung immer nicht bestehen werden, aber sie sind für die meisten Definitionen geeignet. Diese Prüfung erzwingt eine Reihe von zusätzlichen Regeln:

1. Neue Definitionen müssen mit $=$ oder $\leftrightarrow$ eingeführt werden.

2. Keine vor dieser Anweisung eingeführte `$a`-Anweisung darf das in dieser Definition definierte Symbol verwenden, und die Definition darf sich selbst nicht verwenden (außer einmal im Definiendum).

3. Die im Definiens verwendeten Variablen müssen distinkt sein.

4. Alle Dummy-Variablen im Definiendum müssen untereinander und mit den Variablen im Definiendum distinkt sein. Um dies festzustellen, sucht das System in der Datenbasis nach einem „Rechtfertigungs-

satz“[25]. Wenn dieser nicht vorhanden ist, versucht es intern, für jede Dummy-Variable x $(\varphi \rightarrow \forall x \varphi)$ zu beweisen.

5. Jede Dummy-Variable sollte eine Mengenvariable sein, es sei denn, es gibt einen Rechtfertigungssatz.

6. Jede Dummy-Variable muss gebunden sein (wenn das System dies nicht bestimmen kann, muss ein Rechtfertigungssatz angegeben werden).

4.5.4 Zusammenfassung des Metamath-Ansatzes für Definitionen

Kurz gesagt, bei einem rigorosen Vorgehen stellt sich heraus, dass Definitionen subtil sein können und manchmal schwierige Metatheoreme erfordern, um zu beweisen, dass sie nicht kreativ sind.

Anstatt solche Komplikationen in die Metamath-Sprache selbst einzubauen, behandeln die grundlegende Metamath-Sprache und das Programm die traditionellen Axiome und Definitionen einheitlich als `$a`-Anweisungen. Wir haben dann verschiedene Werkzeuge entwickelt, die es jedem ermöglichen, für seine spezifischen, selbst angelegten Datenbasen zusätzliche, selbst festgelegte Bedingungen zu überprüfen, ohne die grundlegenden Eigenschaften und Funktionalitäten von Metamath zu verkomplizieren.

[25] Anm. der Übersetzer: Ein „Rechtfertigungssatz“ („justification theorem“) ist ein Theorem, dass die Korrektheit der Definition rechtfertigt. Der Name des Theorems ist der Teil des Namens der Definition nach „df-“ ergänzt um „just“, siehe z.B. `df-mo` und `mojust`.

Kapitel 5

Das Metamath-Programm

Dieses Kapitel ist als ein Referenzhandbuch für das Metamath-Programm zu verstehen.

Aktuelle Anweisungen für den Bezug und die Installation des Metamath-Programms finden Sie auf der Website `http://metamath.org`. Für Windows gibt es eine vorkompilierte Version namens `metamath.exe`. Für Unix, Linux und Mac OS X (die wir zusammenfassend als „Unix“ bezeichnen) kann das Metamath-Programm aus seinem Quellcode mit dem Befehl

```
gcc *.c -o metamath
```

unter Verwendung des auf diesen Systemen verfügbaren `gcc` C-Compilers kompiliert werden.

In den nachfolgenden Beschreibungen der Befehlssyntax sind die in eckige Klammern [ ] eingeschlossenen Angaben optional. Dateinamen können optional in einfache oder doppelte Anführungszeichen gesetzt werden. Dies ist nützlich, wenn der Dateiname Leerzeichen oder Schrägstriche (`/`) enthält, wie z.B. in Unix-Pfadnamen, , die mit Metamath-Befehlszeilenparameter verwechselt werden könnten.

5.1 Aufruf von Metamath

Unix, Linux und Mac OS X verfügen über eine Befehlszeilenschnittstelle, die *Bash Shell*. (Unter Mac OS X wählen Sie das Programm Terminal aus Anwendungen/Dienstprogramme.) Um Metamath von der Bash-Shell-Eingabeaufforderung aus aufzurufen, geben Sie Folgendes ein, vorausgesetzt das Metamath-Programm befindet sich im aktuellen Verzeichnis:

```
bash$ ./metamath
```

Um Metamath von einem Windows-Konsolenfenster aus aufzurufen, geben Sie Folgendes ein, vorausgesetzt das Metamath-Programm befindet sich

im aktuellen Verzeichnis (oder in einem Verzeichnis, das in der Systemumgebungsvariablen Path enthalten ist):

```
C:\metamath>metamath
```

Um Befehlszeilenargumente beim Aufruf zu verwenden, sollten die Befehlszeilenargumente eine Liste von Metamath-Befehlen sein, die von Anführungszeichen umgeben sind, wenn sie Leerzeichen enthalten. Unter Windows müssen die umgebenden Anführungszeichen doppelte (nicht einfache) Anführungszeichen sein. Um zum Beispiel die Datenbasisdatei `set.mm` zu lesen, alle Beweise zu überprüfen und das Programm zu beenden, geben Sie (unter Unix) Folgendes ein:

```
bash$ ./metamath 'read set.mm' 'verify proof *' exit
```

Beachten Sie, dass unter Unix jeder Verzeichnispfad mit `/`'s von Anführungszeichen umgeben sein muss, damit Metamath das `/` nicht als Befehlszeilenparameter interpretiert. Wenn also `set.mm` im Verzeichnis `/tmp` liegt, verwenden Sie für das obige Beispiel

```
bash$ ./metamath 'read "/tmp/set.mm"' 'verify proof *' exit
```

Wenn die Befehlszeile nur ein Argument und keine Leerzeichen enthält, wird implizit angenommen, dass der Befehl `read` lautet. In diesem einen Sonderfall werden `/` nicht als Befehlszeilenparameter interpretiert, so dass Sie keine Anführungszeichen um einen Unix-Dateinamen herum benötigen. Also

```
bash$ ./metamath /tmp/set.mm
```

und

```
bash$ ./metamath "read '/tmp/set.mm'"
```

sind gleichwertig.

5.2 Steuerung von Metamath

Das Metamath-Programm wurde zuerst auf einem VAX/VMS-System entwickelt, und einige Aspekte seines Befehlszeilenverhaltens spiegeln dieses Erbe wider. Wir hoffen, dass Sie es einigermaßen benutzerfreundlich finden, sobald Sie sich daran gewöhnt haben.

Jede Befehlszeile besteht aus einer Folge englischsprachiger Wörter, die durch Leerzeichen getrennt sind, wie in `show settings`. Bei Befehlswörtern wird nicht zwischen Groß- und Kleinschreibung unterschieden, und es werden nur so viele Buchstaben benötigt, wie nötig sind, um Mehrdeutigkeiten auszuschließen; so würde beispielsweise bereits `sh se` für die Ausführung des Befehls `show settings` ausreichen. In einigen Fällen sind Argumente

wie Dateinamen, Label von Anweisungen oder Symbolnamen erforderlich; bei diesen wird zwischen Groß- und Kleinschreibung unterschieden (obwohl Dateinamen auf einigen Betriebssystemen möglicherweise nicht unterschieden werden).

Eine Befehlszeile wird eingegeben, indem man sie eintippt und dann die *Return-* (*Enter-*)Taste drückt. Um herauszufinden, welche Befehle verfügbar sind, geben Sie `?` an der Eingabeaufforderung `MM>` ein. Um herauszufinden, welche Möglichkeiten Sie an einer beliebigen Stelle eines Befehls haben, drücken Sie *return* und Sie werden dazu aufgefordert eine angegebene Möglichkeit auszuwählen. Die Standardauswahl (diejenige, die ausgewählt wird, wenn Sie nur *Return* drücken) wird in Klammern angezeigt (`<>`).

Sie können auch `?` anstelle eines Befehlsworts eingeben, um Metamath dazu zu bringen, Ihnen die verfügbaren Möglichkeiten mitzuteilen. Die Methode `?` funktioniert allerdings nicht, wenn an dieser Stelle ein Argument erwartet wird, das kein Schlüsselwort ist, wie z. B. ein Dateiname, weil das Programm dann das `?` als Wert des Arguments interpretiert.

Einige Befehle haben einen oder mehrere optionale Parameter, die das Verhalten des Befehls beeinflussen. Parameter werden mit einem Schrägstrich (`/`) eingeleitet, wie z. B. in `read set.mm / verify`. Leerzeichen um das `/` sind optional. Wenn Sie ein Leerzeichen oder einen Schrägstrich in einem Befehlsargument verwenden müssen, wie in einem Unix-Dateinamen, setzen Sie das Befehlsargument in einfache oder doppelte Anführungszeichen.

Der Befehl `open log` speichert alles, was Sie auf dem Bildschirm sehen, und ist nützlich, wenn Sie etwas wiederherstellen wollen, falls bei einem Beweis etwas schief geht, oder wenn Sie einen Fehler dokumentieren wollen.

Wenn ein Befehl mit mehr Zeilen als auf einen Bildschirm passen antwortet, werden Sie aufgefordert, `<return> zum Fortfahren, Q zum Beenden oder S zum Scrollen zum Ende` einzugeben. `Q` oder `q` (Groß- und Kleinschreibung wird nicht beachtet) führt den Befehl intern zu Ende, unterdrückt aber weitere Ausgaben bis zur nächsten Eingabeaufforderung `MM>`. `s` unterdrückt weitere Pausen bis zur nächsten Eingabeaufforderung `MM>`. Nach der ersten Bildschirmseite haben Sie auch die Möglichkeit, mit `b` eine Bildschirmseite zurückzugehen. Beachten Sie, dass `b` auch an der Eingabeaufforderung `MM>` unmittelbar nach einem Befehl eingegeben werden kann, um durch die Ausgabe dieses Befehls zurückzublättern.

Eine in Anführungszeichen eingeschlossene Befehlszeile wird von Ihrem Betriebssystem ausgeführt. Siehe Abschnitt 5.2.12.

Warnung: Wenn Sie CTRL-C drücken, wird das Metamath-Programm sofort abgebrochen. Das bedeutet, dass alle nicht gespeicherten Daten verloren gehen.

5.2.1 `exit`-Befehl

Syntax: `exit` [`/force`]

Mit diesem Befehl verlässt man Metamath. Wenn mit den Befehlen `proof` oder `save new_proof` Änderungen am Quelltext vorgenommen wurden, erhalten Sie die Möglichkeit, mit `write source` die Änderungen dauerhaft zu speichern.

Im Modus „Beweis-Assistent“ kehrt man mit dem Befehl `exit` zur Eingabeaufforderung `MM>` zurück. Wenn Änderungen am Beweis vorgenommen wurden, erhalten Sie die Möglichkeit, den neuen Beweis mit `save new_proof` zu speichern.

Der Befehl `quit` ist ein Synonym für `exit`.

Optionaler Befehlszeilenparameter: `/force` - Keine Eingabeaufforderung, wenn Änderungen nicht gespeichert wurden. Dieser Qualifizierer ist in `submit`-Befehlsdateien (siehe Abschnitt 5.2.4) nützlich, um ein vorhersehbares Verhalten zu gewährleisten.

5.2.2 `open log`-Befehl

Syntax: `open log` *Dateiname*

Mit diesem Befehl wird eine Protokolldatei geöffnet, in der alles gespeichert wird, was Sie auf dem Bildschirm sehen. Sie ist nützlich, um einen Fehler während einer langen Sitzung mit dem Beweis-Assistenten zu beheben oder um Fehler zu dokumentieren.

Die Protokolldatei kann mit `close log` geschlossen werden. Sie wird automatisch beim Beenden von Metamath geschlossen.

5.2.3 `close log`-Befehl

Syntax: `close log`

Der Befehl `close log` schließt eine Protokolldatei, falls eine geöffnet ist. Siehe auch `open log`.

5.2.4 `submit`-Befehl

Syntax: `submit` *Dateiname*

Dieser Befehl bewirkt, dass weitere Befehlszeilen aus der angegebenen Datei entnommen und ausgeführt werden. Beachten Sie, dass jede Zeile, die mit einem Ausrufezeichen (`!`) beginnt, als Kommentar behandelt (d.h. ignoriert) wird. Beachten Sie auch, dass die Bildschirmausgabe kontinuierlich durchläuft, so dass Sie eventuell eine Protokolldatei öffnen sollten (siehe `open log`), um die auf dem Bildschirm vorbeiziehenden Ergebnisse aufzuzeichnen. Nachdem alle Befehlszeilen in der Datei ausgeführt wurden, kehrt Metamath in den normalen Modus der Benutzeroberfläche zurück.

Der Befehl `submit` kann rekursiv aufgerufen werden (d.h. ebenfalls innerhalb einer `submit` Befehlsdatei).

Optionaler Befehlszeilenparameter: `/silent` - unterdrückt die Bildschirmausgabe, zeichnet die Ausgabe aber dennoch in einer Protokolldatei auf, falls eine solche geöffnet ist.

5.2.5 `erase`-Befehl

Syntax: `erase`

Dieser Befehl setzt Metamath auf seinen Ausgangszustand zurück und löscht alle Datenbasen, die mit `read` eingelesen wurden. Wenn mit den Befehlen `save proof` oder `save new_proof` Änderungen an der Quelldatei vorgenommen wurden, wird Ihnen die Möglichkeit gegeben, `write source` zu verwenden, um die Änderungen dauerhaft zu speichern.

5.2.6 `set echo`-Befehl

Syntax: `set echo on` or `set echo off`

Der Befehl `set echo on` bewirkt, dass die Befehlszeilen mit erweiterten Abkürzungen wiedergegeben werden. Beim Erlernen der Metamath-Befehle zeigt Ihnen diese Funktion genau den Befehl an, dem Ihre abgekürzte Eingabe entspricht.

5.2.7 `set scroll`-Befehl

Syntax: `set scroll prompted` oder `set scroll continuous`

Die Metamath-Befehlszeilenschnittstelle startet im Modus `prompted`, was bedeutet, dass Sie nach jedem Vollbild in einer langen Auflistung zum Fortfahren oder Beenden aufgefordert werden. Im Modus `continuous` werden lange Auflistungen ohne Pause durchlaufen.

5.2.8 `set width`-Befehl

Syntax: `set width` *Zahl*

Metamath geht davon aus, dass die Breite Ihres Bildschirms 79 Zeichen beträgt (dies wurde gewählt, weil die Eingabeaufforderung in Windows XP einen Umbruchfehler bei Spalte 80 aufweist). Wenn Ihr Bildschirm breiter oder schmaler ist, können Sie mit diesem Befehl die Standardbreite des Bildschirms ändern. Eine größere Breite ist vorteilhaft für die Protokollierung von Beweisen in einer Ausgabedatei, die auf einem breiten Drucker gedruckt werden soll. Auf manchen Terminals kann eine geringere Breite erforderlich sein; in diesem Fall kann der Umbruch der Informationsmeldungen jedoch manchmal etwas unnatürlich wirken. In LaTeX gibt es normalerweise maximal 61 Zeichen pro Zeile mit Schreibmaschinenschrift (die Beispiele in diesem Buch wurden mit 61 Zeichen pro Zeile erstellt).

5.2.9 `set height`-Befehl

Syntax: `set height` *Zahl*

Metamath geht davon aus, dass Ihre Bildschirmhöhe 24 Zeilen an Zeichen beträgt. Wenn Ihr Bildschirm größer oder kleiner ist, können Sie mit diesem Befehl die Anzahl der Zeilen ändern, bei denen die Anzeige pausiert und Sie zum Fortfahren auffordert.

5.2.10 `beep`-Befehl

Syntax: `beep`

Bei diesem Befehl ertönt ein Piepton. Wenn Sie ihn vor dem Start eines lang laufenden Befehls eingeben, werden Sie darauf hingewiesen, dass der Befehl beendet ist. Der Einfachheit halber ist `b` eine Abkürzung für `beep`.

Hinweis: Wenn `b` an der Eingabeaufforderung `MM>` unmittelbar nach dem Ende einer mehrseitigen Anzeige mit der Aufforderung „`Press <return> for more...`“ eingegeben wird, kehrt der `b` zur vorherigen Seite zurück, anstatt den Befehl `beep` auszuführen. In diesem Fall müssen Sie die ungekürzte `beep`-Form des Befehls eingeben.

5.2.11 `more`-Befehl

Syntax: `more` *Dateiname*

Dieser Befehl zeigt den Inhalt einer ASCII-Datei auf dem Bildschirm an.

(Dieser Befehl dient der Bequemlichkeit, ist aber nicht sehr leistungsfähig. Siehe Abschnitt 5.2.12, um den entsprechenden Befehl Ihres Betriebssystems aufzurufen, z. B. den Befehl `more` unter Unix).

5.2.12 Betriebssystem-Befehle

Eine in einfache oder doppelte Anführungszeichen eingeschlossene Zeile wird vom Betriebssystem Ihres Computers ausgeführt, wenn dieses über eine Befehlszeilenschnittstelle verfügt. Auf einem VAX/VMS-System wird beispielsweise `MM> 'dir'` den Inhalt des Festplattenverzeichnisses ausgeben. Beachten Sie, dass diese Funktion auf Macintosh-Systemen vor Mac OS X, die keine Befehlszeilenschnittstelle haben, nicht funktioniert.

Der Einfachheit halber ist das Anführungszeichen am Ende optional.

5.2.13 Größenbeschränkungen in Metamath

Im Allgemeinen gibt es keine festen, vordefinierten Grenzen dafür, wie viele Labels, Token, Anweisungen usw. Sie in einer Datenbasisdatei verwenden können. Das Metamath-Programm verwendet 32-Bit-Variablen (64-Bit auf 64-Bit-CPUs) als Indizes für fast alle internen Arrays, die bei Bedarf dynamisch zugewiesen werden.

5.3 Lesen und Schreiben von Dateien

Die folgenden Befehle erstellen neue Dateien: die `open`-Befehle; die `write`-Befehle; die Optionen `/html`, `/alt_html`, `/brief_html`, `/brief_alt_html` von `show statement`, und `midi`. Die folgenden Befehle werden an zuvor geöffnete Dateien angehängt: die Option `/tex` von `show proof` und `show new_proof`; die Optionen `/tex` und `/simple_tex` von `show statement`; die Befehle `close`; und alle Bildschirmdialoge zwischen `open log` und `close log`.

Die Befehle, die neue Dateien erstellen, überschreiben keine vorhandenen *Dateinamen*, sondern benennen die vorhandene Datei in *Dateiname*`~1` um. Ein vorhandener *Dateiname*`~1` wird umbenannt in *Dateiname*`~2`, usw. bis zu *Dateiname*`~9`. Ein vorhandener *Dateiname*`~9` wird gelöscht. Dies erleichtert die Wiederherstellung nach Fehlern, bringt aber auch Unordnung in Ihr Verzeichnis, so dass Sie gelegentlich diese `~`*n*-Dateien bereinigen (löschen) sollten.

5.3.1 `read`-Befehl

Syntax: `read` *Dateiname* [`/verify`]

Mit diesem Befehl wird eine Quelldatei in der Metamath-Sprache und alle darin referenzierten Dateien eingelesen. Normalerweise ist es das erste, was Sie tun, wenn Sie mit Metamath beginnen. Die Syntax der Anweisungen wird überprüft, die Syntax der Beweise jedoch nicht. Beachten Sie, dass der Dateiname in einfache oder doppelte Anführungszeichen eingeschlossen werden kann; dies ist nützlich, wenn der Dateiname Schrägstriche enthält, wie es unter Unix der Fall sein kann.

Falls Sie eine „`?Expected VERIFY`“ Fehlermeldung erhalten, wenn Sie versuchen einen Unix-Dateinamen mit Schrägstrichen zu lesen, haben Sie ihn wahrscheinlich nicht in Anführungszeichen gesetzt.

Wenn Sie zur Eingabe des Dateinamens aufgefordert werden (durch Drücken von *Enter* nach `read`), sollten Sie ihn *nicht* in Anführungszeichen setzen, auch wenn es sich um einen Unix-Dateinamen mit Schrägstrichen handelt.

Optionaler Befehlszeilenparameter:

`/verify` - Überprüft alle Beweise beim Einlesen der Datenbasis. Diese Option verlangsamt das Einlesen der Datei. Siehe `verify proof` für weitere Informationen zur Fehlerprüfung von Dateien.

Siehe auch `erase`.

5.3.2 `write source`-Befehl

Syntax: `write source` *filename* [`/rewrap`] [`/split`] [`/keep_includes`] [`/no_versioning`]

Mit diesem Befehl wird der Inhalt einer Metamath-Datenbasis in eine Datei geschrieben.

Optionale Befehlszeilenparameter:

`/rewrap` - Formatiert Anweisungen und Kommentare entsprechend der in der set.mm-Datenbasis verwendeten Konvention um. Es hebt den Zeilenumbruch im Kommentar vor jeder `$a`- und `$p`-Anweisung auf und bricht die Zeile dann neu um. Sie sollten die Ausgabe mit dem Original vergleichen, um sicherzustellen, dass der gewünschte Effekt erzielt wird; falls nicht, gehen Sie zurück zum Original. Die Länge der umgebrochenen Zeile berücksichtigt den aktuell gültigen Parameter `set width`.

Hinweis: Text, der in `<HTML>...</HTML>`-Tags eingeschlossen ist, wird durch den Qualifier `/rewrap` nicht verändert. Beweise selbst werden nicht umformatiert; verwenden Sie dazu `save proof * / compressed`. Eine isolierte Tilde (~) wird immer in derselben Zeile wie das folgende Symbol gehalten, so dass Sie alle Kommentarverweise auf ein Symbol finden können, indem Sie nach ~, gefolgt von einem Leerzeichen und dem Symbol, suchen (dies ist nützlich, um Querverweise zu finden). Übrigens, `save proof` beachtet auch den derzeit gültigen Parameter `set width`.

`/split` - Dateien, die mit dem Ausdruck $[*inclfile* $] in den Quelltext inkludiert werden, werden in separate Dateien geschrieben, anstatt in eine einzige Ausgabedatei aufgenommen zu werden. Der Name jeder separat geschriebenen Datei ist das Argument *inclfile* des Include-Befehls.

`/keep_includes` - Wenn eine Quelldatei inkludierte Dateien hat, aber durch Weglassen von `/split` als einzelne Datei geschrieben wird, werden die inkludierten Dateien standardmäßig gelöscht (eigentlich nur mit einem Suffix ~1 umbenannt, es sei denn, `/no_versioning` ist gesetzt), um die möglicherweise verwirrende Quelldateiduplikation sowohl in der Ausgabedatei als auch in der inkludierten Datei zu verhindern. Die Option `/keep_includes` verhindert diese Löschung.

`/no_versioning` - Sicherungsdateien mit dem Suffix ~1 werden nicht erstellt.

5.4 Anzeige von Status und Anweisungen

5.4.1 `show settings`-Befehl

Syntax: `show settings`

Dieser Befehl zeigt den Zustand verschiedener Parameter an.

5.4.2 `show memory`-Befehl

Syntax: `show memory`

Dieser Befehl zeigt den noch verfügbaren Speicher an. Er ist auf den meisten modernen Betriebssystemen, die über virtuellen Speicher verfügen, nicht aussagekräftig.

5.4.3 `show labels`-Befehl

Syntax: `show labels` *label-match* [`/all`] [`/linear`]

Dieser Befehl zeigt die Labels von `$a`- und `$p`-Anweisungen an, die auf *label-match* passen. Ein `*` in label-match ist ein Platzhalter für null oder mehr beliebige Zeichen. Zum Beispiel passt `*abc*def` auf alle Labels, die `abc` enthalten und mit `def` enden.

Optionale Befehlszeilenparameter:

`/all` - Übereinstimmungen für `$e`- und `$f`-Anweisungslabels einschließen.

`/linear` - Nur ein Label pro Zeile anzeigen. Dies kann für die Erstellung von Skripten in Verbindung mit den Dienstprogrammen unter dem Befehl `tools` nützlich sein.

5.4.4 `show statement`-Befehl

Syntax: `show statement` *label-match* [*qualifiers* (siehe unten)]

Dieser Befehl liefert Informationen über eine Anweisung. Es können nur Anweisungen mit Labels (`$f`, `$e`, `$a`, und `$p`) angegeben werden. Wenn *label-match* Platzhalterzeichen (`*`) enthält, werden alle übereinstimmenden Anweisungen in der Reihenfolge angezeigt, in der sie in der Datenbasis vorkommen.

Optionale Befehlszeilenparameter (*qualifiers*, es ist jeweils nur ein Parameter zulässig):

`/comment` - Diese Option schließt den Kommentar ein, der der Anweisung unmittelbar vorausgeht.

`/full` - Zeigt vollständige Informationen zu jeder Anweisung an, und zwar für alle Anweisungen, die mit *label* übereinstimmen (einschließlich der `$e`- und `$f`-Anweisungen).

`/tex` - Diese Option schreibt die Anweisungsinformationen in die LaTeX-Datei, die zuvor mit `open tex` geöffnet wurde. Siehe Abschnitt 5.7.

`/simple_tex` - Wie `/tex` mit dem Unterschied, dass LaTeX-Makros nicht für die Formatierung von Gleichungen verwendet werden, was eine einfachere manuelle Bearbeitung der Ausgabe für Folienpräsentationen usw. ermöglicht.

`/html`, `/alt_html`, `/brief_html`, `/brief_alt_html` - Diese Optionen aktivieren einen speziellen Modus von `show statement`, der eine Webseite für die Anweisung erstellt. Sie dürfen nicht zusammen mit einem anderen Qualifizierer verwendet werden. Siehe Abschnitt 5.8 oder `help html` im Programm.

5.4.5 `search`-Befehl

Syntax: search *label-match* "*symbol-match*" [`/all`] [`/comments`] [`/join`]

Dieser Befehl durchsucht alle `$a`- und `$p`-Anweisungen, die mit *label-match* übereinstimmen, nach Vorkommen von *symbol-match*. Ein `*` in *label-match* entspricht einem beliebigen Label-Zeichen. Ein `$*` in *symbol-match* passt auf eine beliebige Folge von Symbolen. Die Symbole in *symbol-match* müssen durch einen Whitespace getrennt sein. Die Anführungszeichen, die *symbol-match* umgeben, können einfache oder doppelte Anführungszeichen sein. Zum Beispiel listet `search b* "-> $* ch"` alle Anweisungen auf, deren Label mit `b` beginnen und die Symbole `->` und `ch` enthalten, die eine beliebige Symbolfolge umgeben (einschließlich keiner Symbolfolge). Die Platzhalter `?` und `$?` sind auch verfügbar, um einzelne Zeichen in Labels bzw. Symbolen zu finden; siehe `help search` im Metamath-Programm für Details zu ihrer Verwendung.

Optionale Befehlszeilenparameter:

`/all` - Suche auch nach `$e`- und `$f`-Anweisungen.

`/comments` - Sucht auch in dem Kommentar, der jeder Anweisung mit Label-Matching unmittelbar vorausgeht, nach *symbol-match*. In diesem Fall ist *symbol-match* eine beliebige, nicht von Groß- und Kleinschreibung abhängige Zeichenkette. Anführungszeichen um *symbol-match* sind optional, wenn es keine Mehrdeutigkeit gibt.

`/join` - Im Falle einer `$a`- oder `$p`-Anweisung werden deren `$e`-Hypothesen für die Suche vorangestellt. Die Option `/join` hat im Modus `/comments` keine Wirkung.

5.5 Anzeigen und Verifizieren von Beweisen

5.5.1 `show proof`-Befehl

Syntax: `show proof` *label-match* [*qualifiers* (siehe unten)]

Mit diesem Befehl wird der Beweis der angegebenen `$p`-Anweisung in verschiedenen Formaten angezeigt. Der Parameter *label-match* kann Platzhalterzeichen (`$*`) enthalten, um mehrere Anweisungen anzuzeigen. Ohne Optionen (*qualifiers*) werden nur die logischen Schritte in einem eingerückten Format angezeigt (d.h. die Syntaxkonstruktionsschritte werden weggelassen).

In den meisten Fällen werden Sie `show proof` *label* verwenden, um nur die Beweisschritte zu sehen, die logischen Schlussfolgerungen entsprechen.

Optionale Befehlszeilenparameter:

`/essential` - Der Beweisbaum wird vor der Anzeige um alle Hypothesen (`$f`-Anweisungen) bereinigt. (Dies ist die Voreinstellung und muss deshalb nicht angegeben werden).

`/all` - Der Beweisbaum wird vor der Anzeige nicht von allen `$f`-Hypothesen bereinigt. `/essential` und `/all` schließen sich gegenseitig aus.

`/from_step` *step* - Die Anzeige beginnt mit dem angegebenen Schritt. Wird diese Option weggelassen, beginnt die Anzeige beim ersten Schritt.

`/to_step` *step* - Die Anzeige endet mit dem angegebenen Schritt. Wird diese Option weggelassen, endet die Anzeige beim letzten Schritt.

`/tree_depth` *number* - Es werden nur Schritte angezeigt, die weniger als die angegebene Tiefe des Beweisbaums haben. Manchmal nützlich, um einen Überblick über den Beweis zu erhalten.

`/reverse` - Die Schritte werden in umgekehrter Reihenfolge angezeigt.

`/renumber` - Bei Verwendung mit `/essential` werden die Schritte neu nummeriert und entsprechen nur den wesentlichen Schritten.

`/tex` - Der Beweis wird in LaTeX konvertiert und in der mit `open tex` geöffneten Datei gespeichert. Siehe Abschnitt 5.7 oder `help tex` im Programm.

`/lemmon` - Der Beweis wird in einem nicht eingerückten Format angezeigt, das als Lemmon-Stil bekannt ist, mit expliziten Verweisen auf vorherige Schrittnummern. Wird diese Option weggelassen, werden die Schritte in einem Baumformat eingerückt.

`/start_column` *number* - Setzt die Standardspalte (16) außer Kraft, bei der die Formelanzeige in einer Lemmon-Anzeige beginnt. Kann nur in Verbindung mit `/lemmon` verwendet werden.

`/normal` - Der Beweis wird in einem normalen Format angezeigt, das sich zur Aufnahme in eine Metamath-Quelldatei eignet. Darf nicht mit einer anderen Option verwendet werden.

`/compressed` - Der Beweis wird in einem komprimierten Format angezeigt, das sich zur Aufnahme in eine Metamath-Quelldatei eignet. Darf nicht mit einer anderen Option verwendet werden.

`/statement_summary` - Gibt einen Überblick über alle Anweisungen (wie mit `show statement`), die im Beweis verwendet werden. Er darf nicht mit einer anderen Option außer `/essential` verwendet werden.

`/detailed_step` *step* - Zeigt an, was in einem bestimmten Schritt des Beweises im Einzelnen geschieht. Darf nicht mit einer anderen Option verwendet werden. Der Schritt *step* ist die Schrittnummer, die angezeigt wird, wenn ein Beweis ohne die Option `/renumber` dargestellt wird.

5.5.2 `show usage`-Befehl

Syntax: `show usage` *label-match* [`/recursive`]

Dieser Befehl listet die Anweisungen auf, deren Beweise sich direkt auf die angegebene Anweisung beziehen.

Optionaler Befehlszeilenparameter:

`/recursive` - Dazu gehören auch Anweisungen, deren Beweise letztlich von der angegebenen Anweisung abhängen.

5.5.3 `show trace_back`-Befehl

Syntax: `show trace_back` *label-match* [`/essential`] [`/axioms`] [`/tree`] [`/depth` *number*]

Dieser Befehl listet alle Anweisungen auf, von denen der Beweis der durch *label-match* angegebenen Anweisung(en) abhängt.

Optionale Befehlszeilenparameter:

`/essential` - Beschränkt die Rückverfolgung auf `$e`-Hypothesen von Beweisbäumen.

`/axioms` - Führt nur die Axiome auf, von denen der Beweis letztlich abhängt.

`/tree` - Anzeige der Rückverfolgung in einem eingerückten Baumformat.

`/depth` *number* - Schränkt die `/tree`-Rückverfolgung auf die angegebene Einrückungstiefe ein.

`/count_steps` - Zählt die Anzahl der Schritte, die der Beweis bis zu den Axiomen zurückführt. Wenn `/essential` angegeben ist, werden Expansionen von Hypothesen vom Variablentyp (Syntaxkonstruktionen) nicht gezählt.

5.5.4 `verify proof`-Befehl

Syntax: `verify proof` *label-match* [`/syntax_only`]

Mit diesem Befehl werden die Beweise der angegebenen Aussagen überprüft. Die Option *label-match* kann Platzhalterzeichen (`*`) enthalten, um mehr als einen Beweis zu überprüfen; zum Beispiel wird `*abc*def` auf alle Labels passen, die `abc` enthalten und mit `def` enden. Der Befehl `verify proof *` prüft alle Beweise in der Datenbasis.

Optionaler Befehlszeilenparameter:

`/syntax_only` - Nit dieser Option wird nur auf Syntax- und RPN-Stack-Verletzungen geprüft. Es wird nicht geprüft, ob der Beweis korrekt ist. Diese Option ist nützlich um schnell festzustellen, welche Beweise unvollständig sind (d.h. in der Entwicklung sind und `?`'s in ihnen enthalten sind).

Anmerkung: `read`, gefolgt von `verify proof *`, stellt sicher, dass die Datenbasis frei von Fehlern in der Metamath-Sprache ist, überprüft aber nicht die Auszeichnungsnotation in Kommentaren. Sie können die Auszeichnungsnotation auch überprüfen, indem Sie `verify markup *` ausführen, wie in Abschnitt 5.5.5 beschrieben; siehe auch die Diskussion über die Erzeugung von HTML in Abschnitt 5.8.

5.5.5 `verify markup`-Befehl

Syntax: `verify markup` *label-match* [`/date_skip`] [`/top_date_skip`] [`/file_skip`] [`/verbose`]

Dieser Befehl überprüft Kommentarauszeichnungen und andere informelle Konventionen, die wir festgesetzt haben. Er prüft die latexdef-, htmldef- und althtmldef-Anweisungen in der `$t`-Anweisung einer Metamath-Quelldatei auf Fehler. Es prüft alle `‘...‘`, `~` *Label* und bibliographischen Markierungen in Anweisungsbeschreibungen auf Fehler. Es wird geprüft, ob `$p`- und `$a`-Anweisungen den gleichen Inhalt haben, wenn ihre Labels mit „ax“ bzw.

„ax-“ beginnen, aber ansonsten identisch sind, zum Beispiel ax4 und ax-4. Er überprüft auch die Datumsübereinstimmung von „(Contributed by...)“, „(Revised by...)“ und „(Proof shortened by...)“ in dem Kommentar über jeder `$a`- und `$p`-Anweisung.

Optionale Befehlszeilenparameter:

`/date_skip` - Mit dieser Option wird die Prüfung der Datumsübereinstimmung übersprungen, die normalerweise für andere Datenbasen als `set.mm` nicht erforderlich ist.

`/top_date_skip` - Mit dieser Option wird die Datumsübereinstimmung geprüft, mit der Ausnahme, dass das Versionsdatum am Anfang der Datenbasisdatei nicht geprüft wird. Es kann nur eine der beiden Optionen `/date_skip` und `/top_date_skip` angegeben werden.

`/file_skip` - Mit dieser Option werden Prüfungen übersprungen, die das Vorhandensein externer Dateien voraussetzen, wie z. B. die Überprüfung des Vorhandenseins von GIFs und bibliografischen Links zu mmset.html oder Ähnlichem. Dies ist nützlich, um eine schnelle Prüfung aus einem Verzeichnis ohne diese Dateien durchzuführen.

`/verbose` - Liefert mehr Informationen. Gegenwärtig wird eine Liste der Übereinstimmungen zwischen axXXX und ax-XXX angezeigt.

5.5.6 `save proof`-Befehl

Syntax: `save proof` *label-match* [`/normal`] [`/compressed`]

Der Befehl `save proof` formatiert einen Beweis in einem von zwei Formaten neu und ersetzt den vorhandenen Beweis im Quellpuffer. Er ist nützlich, um zwischen verschiedenen Formaten von Beweisen zu konvertieren. Beachten Sie, dass ein Beweis erst dann dauerhaft gespeichert wird, wenn Sie den Befehl `write source` ausführen.

Optionale Befehlszeilenparameter:

`/normal` - Der Beweis wird im normalen Format gespeichert (d. h. als eine Folge von Labels, was dem definierten Format der Metamath-Basissprache entspricht). Dies ist das Standardformat, das verwendet wird, wenn keine Option angegeben wird.

`/compressed` - Der Beweis wird im komprimierten Format gespeichert, was den Speicherbedarf für eine Datenbasis reduziert. Siehe Anhang B.

5.6 Beweise erstellen

Bevor Sie den Beweis-Assistenten verwenden, müssen Sie (mit einem Texteditor) eine `$p`-Anweisung in Ihre Quelldatei einfügen, der die zu beweisende Aussage enthält. Der Beweis sollte aus einem einzigen `?` bestehen, was „unbekannter Schritt“ bedeutet. Beispiel:

```
equid $p x = x $= ? $.
```

Um den Beweis-Assistenten aufzurufen, geben Sie `prove` *label* ein, z.B. `prove equid`. Metamath antwortet mit der Eingabeaufforderung `MM-PA>`.

Beweise werden ausgehend von der zu beweisenden Anweisung rückwärts erstellt, wobei hauptsächlich eine Reihe von `assign`-Befehlen verwendet wird. Ein Beweis ist vollständig, wenn allen Schritten Anweisungen zugewiesen sind und alle Schritte vereinheitlicht wurden und vollständig bekannt sind. Während der Erstellung eines Beweises lässt Metamath nur Operationen zu, die aufgrund der bis dahin bekannten Daten zulässig sind. So wird zum Beispiel kein `assign` mit einer Anweisung zugelassen, die nicht in den unbekannten Beweisschritt, für den die Zuweisung erfolgen soll, eingesetzt werden kann.

Wichtig: Der Beweis-Assistent ist kein Werkzeug, das Ihnen hilft, Beweise zu finden. Er ist nur ein Hilfsmittel, das Ihnen hilft, Beweise zur Datenbasis hinzuzufügen. Eine Anleitung dazu finden Sie in Abschnitt 2.4. Um die Verwendung des Beweis-Assistenten zu üben, können Sie ein bestehendes Theorem mit `prove` bearbeiten, dann alle Schritte mit `delete all` löschen und den Bewqeis dann mit dem Beweis-Assistenten neu erstellen, während Sie den (vor dem Löschen) angezeigten Beweises betrachten. Es kann sinnvoll sein, die ersten Beweise vollständig selbst zu erarbeiten und von Hand aufzuschreiben, bevor Sie den Beweis-Assistenten benutzen, auch wenn das nicht für jeden geeignet ist.

Wichtig: Der Befehl `undo` ist sehr hilfreich bei der Eingabe eines Beweises, da Sie damit einen zuvor eingegebenen Schritt rückgängig machen können. Außerdem empfehlen wir Ihnen, Ihre Arbeit in einer Protokolldatei (`open log`) festzuhalten und sie regelmäßig zu speichern (`save new_proof`, `write source`). Sie können `delete` verwenden, um ein `assign` rückgängig zu machen. Sie können auch `delete floating_hypotheses`, dann `initialize all` und dann `unify all /interactive` verwenden, um ungewollte Vereinheitlichungen, die versehentlich oder durch unpassende `assign`s gemacht wurden, zu reinitialisieren. Sie können ein `delete` nicht rückgängig machen, es sei denn, Sie verwenden ein entsprechendes `undo` oder `exit /force` und rufen dann den Beweis-Assistenten erneut auf, um den letzten `save new_proof` wiederherzustellen.

Die folgenden Befehle stehen im Proof-Assistenten (an der Eingabeaufforderung `MM-PA>`) zur Verfügung, um Sie bei der Erstellung Ihres Beweises zu unterstützen. Siehe die einzelnen Befehle für weitere Details.

> `show new_proof` [`/all`,...] - Zeigt den aktuellen Beweis an. Sie werden diesen Befehl häufig verwenden; siehe `help show new_proof`, um sich mit seinen Optionen vertraut zu machen. Die Optionen `/unknown` und `/not_unified` sind nützlich, um die noch zu erledigende Arbeit anzuzeigen. Die Kombination `/all/unknown` ist nützlich, um Dummy-Variablen zu identifizieren, die zugewiesen werden müssen, oder Versuche, ungültige Syntax zu verwenden, wenn `improve all` nicht in der Lage ist, die Syntaxkonstruktionen abzuschließen. Unbekannte Varia-

blen werden als `$1`, `$2`, ... angezeigt.

`assign` *step label* - Weist einem noch nicht zugeordneten Beweisschritt mit der *step*-Nummer die durch *label* angegebene Anweisung zu.

`let variable` *variable* `= "`*symbol sequence*`"` - Erzwingt die Ersetzung einer unbekannten Variablen (z. B. `$1`) in einem Beweis durch eine Symbolfolge. Dies ist nützlich, um schwierige Vereinheitlichungen zu erleichtern, und es ist notwendig, wenn Sie Dummy-Variablen nutzen, denen schließlich ein Name zugewiesen werden muss.

`let step` *step* `= "`*symbol sequence*`"` - Erzwingt, dass eine Symbolsequenz den Inhalt eines Beweisschritts ersetzt, sofern sie mit dem vorhandenen Schrittinhalt vereinheitlicht werden kann. (Ich verwende dies selten.)

`unify step` *step* (oder `unify all`) - Vereinheitlicht die Quelle und das Ziel eines Schrittes. Wenn Sie einen bestimmten Schritt angeben, werden Sie aufgefordert, eine der möglichen Vereinheitlichungen auszuwählen. Wenn Sie `all` angeben, werden alle Schritte mit eindeutigen Vereinheitlichungen vereinheitlicht, aber nur diese Schritte. `unify all /interactive` geht durch alle nicht vereinheitlichten Schritte.

`initialize` *step* (oder `all`) - Entkoppelt das Ziel und die Quelle eines Schritts (oder aller Schritte) sowie die Hypothesen der Quelle und macht alle Variablen in der Quelle unbekannt. Nützlich, um einen `assign`- oder `let`-Fehler zu beheben, der zu falschen Vereinheitlichungen führte.

`delete` *step* (oder `all` oder `floating_hypotheses`) - Löscht den/die angegebenen Schritt(e). `delete floating_hypotheses`, dann `initialize all`, dann `unify all /interactive` ist nützlich, um Fehler zu beheben, bei denen falsche Vereinheitlichungen den Variablen falsche mathematische Symbolfolgen zugewiesen haben.

`improve` *step* (oder `all`) - Erstellt automatisch einen Beweis für Schritte (ohne unbekannte Variablen), deren Beweis keine Anweisungen mit `$e`-Hypothesen erfordert. Nützlich zum Ausfüllen von Beweisen für `$f`-Hypothesen. Die Option `/depth` versucht dies auch mit Anweisungen, deren `$e`-Hypothesen keine neuen Variablen enthalten. Warnung: Speichern Sie Ihre Arbeit (mit `save new_proof` und dann `write source`), bevor Sie `/depth = 2` oder größer verwenden, da die Suchzeit exponentiell ansteigt und möglicherweise nie in einer angemessenen Zeit beendet wird, und Sie können die Suche nicht unterbrechen. Ich habe festgestellt, dass `/depth = 3` oder größer nur selten nützlich ist.

`save new_proof` - Speichert den laufenden Beweis im internen Datenbasispuffer des Programms. Um ihn dauerhaft in der Datenbasisdatei

zu speichern, verwenden Sie `write source` nach `save new_proof`. Um zum letzten `save new_proof` zurückzukehren, beenden Sie mit `exit /force` den Beweis-Assistenten und starten Sie ihn dann erneut.

`match step` *step* (oder `match all`) - Zeigt an, welche Anweisungen für die Anweisung `assign` möglich sind. (Dieser Befehl ist in seiner jetzigen Form nicht sehr nützlich und wird hoffentlich in Zukunft verbessert werden. Verwenden Sie in der Zwischenzeit die Anweisung `search` für Kandidaten, die auf bestimmte Kombinatiuonen von mathematische Token passen.)

`minimize_with` - Nachdem ein Beweis abgeschlossen ist, können mit diesem Befehl andere Datenbasistheoreme mit dem Beweis abgeglichen werden, um damit ggf. die Größe des Beweises zu verringern. Siehe `help minimize_with` im Metamath-Programm für seine Verwendung.

`undo` - Macht die Wirkung eines den Beweis verändernden Befehls rückgängig (alle Befehle außer den oben genannten `show` und `save`).

`redo` - Macht ein vorheriges `undo` wieder rückgängig.

Die folgenden Befehle setzen Parameter, die für Ihren Beweis relevant sein können. Konsultieren Sie die einzelnen `help set`... Befehle.

`set unification_timeout`

`set search_limit`

`set empty_substitution` - Beachten Sie, dass der Standardwert `off` ist.

Geben Sie `exit` ein, um die Eingabeaufforderung `MM-PA>` zu verlassen und zur Eingabeaufforderung `MM>` zurückzukehren. Ein weiteres `exit` beendet dann Metamath komplett.

5.6.1 `prove`-Befehl

Syntax: `prove` *label*

Mit diesem Befehl wird der Beweis-Assistent aufgerufen, mit dem Sie den Beweis der angegebenen Aussage erstellen oder bearbeiten können. Die Eingabeaufforderung in der Befehlszeile ändert sich von `MM>` zu `MM-PA>`.

Hinweis: In der aktuellen Version (0.177) von Metamath prüft der Beweis-Assistent nicht, ob die Einschränkungen für `$d`-Anweisungen eingehalten werden, während ein Beweis erstellt wird. Nachdem Sie einen Beweis abgeschlossen haben, sollten Sie `save new_proof` gefolgt von `verify proof` *label* (wobei *label* die Anweisung ist, die Sie mit dem Befehl `prove` beweisen) eingeben, um die `$d`-Einschränkungen zu überprüfen.

Siehe auch: `exit`

5.6.2 `set unification_timeout`-Befehl

Syntax: `set unification_timeout` *number*

(Dieser Befehl ist auch außerhalb des Beweis-Assistenten verfügbar, wirkt sich aber nur auf den Beweis-Assistenten aus).

Manchmal meldet der Beweis-Assistent, dass eine Zeitüberschreitung beim Vereinheitlichen aufgetreten ist. Dies kann passieren, wenn Sie versuchen, Formeln mit vielen temporären Variablen (`$1`, `$2`, usw.) zu vereinheitlichen, da die Zeit für die Berechnung aller möglichen Vereinheitlichungen exponentiell mit der Anzahl der Variablen wachsen kann. Wenn Sie möchten, dass Metamath sich mehr Mühe gibt (und Sie bereit sind, länger zu warten), können Sie diesen Parameter erhöhen. `show settings` zeigt Ihnen den aktuellen Wert an.

5.6.3 `set empty_substitution`-Befehl

Syntax: `set empty_substitution on` oder `set empty_substitution off`

(Dieser Befehl ist auch außerhalb des Proof-Assistenten verfügbar, wirkt sich aber nur auf den Beweis-Assistenten aus).

Die Metamath-Sprache erlaubt es, Variablen durch leere Symbolfolgen zu ersetzen. In vielen formalen Systemen wird dies jedoch nie in einem gültigen Beweis vorkommen. Die Berücksichtigung dieser Möglichkeit erhöht die Wahrscheinlichkeit mehrdeutiger Vereinheitlichungen während der Beweiserstellung. Standardmäßig sind leere Substitutionen nicht erlaubt; für formale Systeme, die sie erfordern, müssen Sie `set empty_substitution on` setzen. Ein Beispiel, bei dem dieser Parameter `on` sein muss, wäre ein System, das eine Deduktionsregel implementiert und in dem Deduktionen von leeren Annahmelisten zulässig wären. Das im Anhang D beschriebene MIU-System ist ein weiteres Beispiel.

Es ist besser, diesen Befehl auszuschalten (auf `off` zu setzen oder zu belassen), wenn Sie mit `set.mm` arbeiten. Beachten Sie, dass dieser Befehl keinen Einfluss darauf hat, wie Beweise mit dem Befehl `verify proof` überprüft werden. Außerhalb des Beweis-Assistenten ist die Ersetzung von leeren Sequenzen für mathematische Symbole immer erlaubt.

5.6.4 `set search_limit`-Befehl

Syntax: `set search_limit` *number*

(Dieser Befehl ist auch außerhalb des Proof-Assistenten verfügbar, wirkt sich aber nur auf den Beweis-Assistenten aus).

Dieser Befehl legt einen Parameter fest, der bestimmt, wann der Befehl `improve` im Modus Beweis-Assistent seine Suche nach Vereinheitlichungen beendet. Wenn Sie möchten, dass `improve` intensiver sucht, können Sie den Wert erhöhen. Der Befehl `show settings` zeigt Ihnen den aktuellen Wert an.

5.6.5 `show new_proof`-Befehl

Syntax: `show new_proof` [*Optionen* (siehe unten)]

Dieser Befehl (nur im Modus Beweis-Assistent verfügbar) zeigt den aktuellen Beweis an. Er ist identisch mit dem Befehl `show proof` mit dem Unterschied, dass es kein Argument für die Aussage gibt (da es sich um die zu beweisende Aussage handelt). Außerdem sind die folgenden Optionen nicht verfügbar:

`/statement_summary`

`/detailed_step`

Es sind aber die folgenden zusätzlichen Optionen verfügbar:

`/unknown` - Zeigt nur Schritte an, denen keine Anweisung zugewiesen ist.

`/not_unified` - Zeigt nur Schritte an, die noch nicht vereinheitlicht wurden.

Beachten Sie, dass `/essential`, `/depth`, `/unknown` und `/not_unified` in jeder beliebigen Kombination verwendet werden können; jede von ihnen filtert effektiv zusätzliche Schritte aus der Beweisanzeige heraus.

Siehe auch: `show proof`

5.6.6 `assign`-Befehl

Syntax: `assign` *step label* [`/no_unify`]

und: `assign first` *label*

und: `assign last` *label*

Dieser Befehl, der nur im Beweis-Assistenten verfügbar ist, ordnet einem unbekannten (d.h. noch nicht zugeordneten) Schritt (einen mit `?` in der Auflistung `show new_proof`) die durch *label* angegebene Anweisung zu. Die Zuordnung wird nicht zugelassen, wenn die Anweisung nicht mit dem Schritt vereinheitlicht werden kann.

Wenn `last` anstelle der *step*-Nummer angegeben wird, wird der letzte Schritt, der von `show new_proof /unknown` angezeigt wird, verwendet. Dies kann für die Erstellung eines Beweises mit einer Befehlsdatei nützlich sein (siehe `help submit`). Es beschleunigt auch das Erstellen von Beweisen, wenn Sie die Zuordnung für den letzten Schritt kennen.

Wenn `first` anstelle der *step*-Nummer angegeben wird, wird der erste Schritt verwendet, der durch `show new_proof /unknown` angezeigt wird.

Wenn *step* 0 oder negativ ist, wird der -*step*-te von dem letzten unbekannten Schritt, wie durch `show new_proof /unknown` gezeigt, verwendet. `assign -1` *label* weist den vorletzten unbekannten Schritt zu, `assign -2` *label* den vorvorletzten, und `assign 0` *label* ist dasselbe wie `assign last` *label*.

Optionaler Befehlszeilenparameter:

`/no_unify` - der Benutzer wird nicht aufgefordert, eine Vereinheitlichung zu wählen, wenn es mehr als eine Möglichkeit gibt. Dies ist nützlich für nicht-interaktive Befehlsdateien. Später kann der Benutzer `unify all`

`/interactive` wählen. (Die Zuweisung wird immer noch automatisch vereinheitlicht, wenn es nur eine Möglichkeit gibt, und wird abgelehnt, wenn eine Vereinheitlichung nicht möglich ist).

5.6.7 `match`-Befehl

Syntax: `match step` *step* [`/max_essential_hyp` *number*]
und: `match all` [`/essential`] [`/max_essential_hyp` *number*]

Dieser Befehl, der nur im Beweis-Assistenten verfügbar ist, zeigt an, welche Anweisungen mit dem/den angegebenen Schritt(en) vereinigt werden können. *Hinweis:* In seiner jetzigen Form ist dieser Befehl nicht sehr nützlich, da er eine große Anzahl von Übereinstimmungen anzeigt. Er kann in Zukunft verbessert werden. In der Zwischenzeit kann der Befehl `search` oft eine bessere Möglichkeit für das Auffinden von Theoremen von Interesse bieten.

Optionale Befehlszeilenparameter:

`/max_essential_hyp` *number* - filtert aus der Liste alle Anweisungen mit mehr als der angegebenen Anzahl von `$e`-Anweisungshypothesen heraus.

`/essential_only` - in der Anweisung `match all` werden nur die Schritte abgeglichen, die in der Anzeige `show new_proof /essential` aufgelistet wären.

5.6.8 `let`-Befehl

Syntax: `let variable` *variable* = "*symbol-sequence*"
und: `let step` *step* = "*symbol-sequence*"

Diese Befehle, die nur im Beweis-Assistenten verfügbar sind, weisen einer temporären Variable oder einem unbekannten Schritt eine bestimmte Symbolfolge zu. Sie sind während der Erstellung eines Beweises nützlich, wenn Sie wissen, was in dem Beweisschritt enthalten sein soll, der Vereinheitlichungsalgorithmus aber noch nicht genügend Informationen hat, um die temporären Variablen vollständig zu ermitteln. Eine „temporäre Variable“ ist eine Variable, die in der Beweisanzeige die Form `$`*nn* hat, wie z.B. `$1`, `$2`, usw. Die *Symbolfolge* kann auch andere unbekannte Variablen enthalten, falls gewünscht. Beispiele:

```
let variable $32 = "A = B"
let variable $32 = "A = $35"
let step 10 = '|- x = x'
let step -2 = "|- ( $7 = ph )"
```

Für den Befehl `let variable` wird jede beliebige Symbolfolge akzeptiert. Für `let step` werden nur solche Symbolfolgen akzeptiert, die mit dem Schritt vereinheitlicht werden können.

Die `let`-Befehle „knallen“ Informationen in den Beweis, die nur verifiziert werden können, wenn der Beweis weiter aufgebaut wird. Wenn Sie einen Fehler machen, kann die Befehlssequenz `delete floating_hypotheses`,

`initialize all` und `unify all /interactive` eine falsche `let`-Zuweisung rückgängig machen.

Wenn *step* 0 oder negativ ist, wird der -*step*-te vom letzten unbekannten Schritt, wie durch `show new_proof /unknown` gezeigt, verwendet. Der Befehl `let step 0 =` "*symbol-sequence*" verwendet den letzten unbekannten Schritt, `let step -1 =` "*symbol-sequence*" den vorletzten, usw. Wenn *step* positiv ist, kann `let step` verwendet werden, um sowohl bekannte (im Sinne von zuvor mit `assign` ein Label zugewiesen) als auch unbekannte Schritte zuzuweisen.

Einfache oder doppelte Anführungszeichen können *symbol-sequence* umgeben, solange sie sich von allen Anführungszeichen innerhalb einer *symbol-sequence* unterscheiden. Wenn *symbol-sequence* beide Arten von Anführungszeichen enthält: siehe die Anweisungen am Ende von `help let` im Metamath-Programm.

5.6.9 `unify`-Befehl

Syntax: `unify step` *step*

und: `unify all` [`/interactive`]

Diese Befehle, die nur im Beweis-Assistenten verfügbar sind, vereinheitlichen die Quelle und das Ziel des/der angegebenen Schrittes/Schritte. Wenn Sie einen bestimmten Schritt angeben, werden Sie aufgefordert, eine der möglichen Vereinheitlichungen auszuwählen. Wenn Sie `all` angeben, werden nur die Schritte mit eindeutigen Vereinheitlichungen vereinheitlicht.

Optionaler Befehlszeilenparameter für `unify all`:

`/interactive` - Sie werden aufgefordert, eine der möglichen Vereinheitlichungen für alle Schritte auszuwählen, die keine eindeutigen Vereinheitlichungen haben. (Andernfalls wird `unify all` diese Schritte übergehen).

Siehe auch `set unification_timeout`. Der Standardwert ist 100000, aber eine Erhöhung auf 1000000 kann in problematischen Fällen helfen. Die manuelle Zuweisung einiger oder aller unbekannten Variablen mit dem Befehl `let variable` hilft ebenfalls in schwierigen Fällen.

5.6.10 `initialize`-Befehl

Syntax: `initialize step` *step*

und: `initialize all`

Diese Befehle, die nur im Beweis-Assistenten verfügbar sind, „ent-unifizieren“ das Ziel und die Quelle eines Schrittes (oder aller Schritte), sowie die Hypothesen der Quelle, und machen alle Variablen in der Quelle und die Hypothesen der Quelle unbekannt. Dieser Befehl ist nützlich, um einen Beweis nach einer falsche Vereinheitlichungen, die durch ein falsches `assign`, ein falsches `let` oder einer falschen automatischen Zuordnung entstanden sind, wiederherzustellen. Ein Teil oder die gesamte Befehlssequenz `delete`

`floating_hypotheses`, `initialize all` und `unify all /interactive` wird den Beweis nach falschen Vereinheitlichungen wiederherstellen.

Siehe auch: `unify` und `delete`

5.6.11 `delete`-Befehl

Syntax: `delete step` *step*

und: `delete all` – *Achtung: Gefährlich!*

und: `delete floating_hypotheses`

Diese Befehle sind nur im Beweis-Assistenten verfügbar. Der Befehl `delete step` löscht den Abschnitt des Beweisbaums, der von dem angegebenen Schritt abzweigt, und lässt den Schritt unbekannt werden. `delete all` ist äquivalent zu `delete step` *step*, wobei *step* der letzte Schritt im Beweis ist (d.h. der Anfang des Beweisbaums).

In den meisten Fällen ist der Befehl `undo` die beste Methode, um einen vorherigen Schritt rückgängig zu machen. Eine Alternative ist, den letzten Beweis zu speichern, indem Sie den Beweis-Assistenten verlassen und erneut aufrufen. Damit dies funktioniert, sollten Sie eine Protokolldatei öffnen, um Ihre Arbeit zu protokollieren, und den Befehl `save new_proof` häufig ausführen, insbesondere vor `delete`.

`delete floating_hypotheses` löscht alle Abschnitte des Beweises, die von `$f`-Anweisungen abzweigen. Es ist manchmal nützlich, dies vor einem `initialize`-Befehl zu tun, um einen Fehler zu beheben. Sobald ein Beweisschritt mit einer `$f`-Hypothese als Ziel vollständig bekannt ist, kann der Befehl `improve` normalerweise den Beweis für diesen Schritt ausfüllen. Im Gegensatz zum Löschen von logischen Schritten ist `delete floating_hypotheses` ein relativ sicherer Befehl, nach dem der Beweis normalerweise leicht wiederhergestellt werden kann.

5.6.12 `improve`-Befehl

Syntax: `improve` *step* [`/depth` *number*] [`/no_distinct`]

und: `improve first` [`/depth` *number*] [`/no_distinct`]

und: `improve last` [`/depth` *number*] [`/no_distinct`]

und: `improve all` [`/depth` *number*] [`/no_distinct`]

Diese Befehle, die nur im Beweis-Assistenten verfügbar sind, versuchen, automatisch Beweise für unbekannte Schritte zu finden, deren Symbolfolgen vollständig bekannt sind. Sie sind in erster Linie zum Ausfüllen von Beweisen für `$f`-Hypothesen nützlich. Die Suche wird auf Anweisungen beschränkt, die keine `$e`-Hypothesen enthalten.

Hinweis: Wenn der Speicher begrenzt ist, kann `improve all` bei einem großen Beweis den Speicher überlaufen lassen. Wenn Sie `set unification_timeout 1` vor `improve all` verwenden, wird in der Regel eine ausreichende Verbesserung erzielt, um den Beweis später auf einem

größeren Computer leicht wiederherzustellen und mittels `improve` zu vervollständigen. Warnung: Wenn der Speicher einmal übergelaufen ist, gibt es keine Möglichkeit für eine Wiederherstellung mehr. Speichern Sie im Zweifelsfall den Zwischenbeweis (`save new_proof` und danach `write source`) vor `improve all`.

Wenn `last` anstelle von *step* number angegeben wird, wird der letzte Schritt, der durch `show new_proof /unknown` angezeigt wird, verwendet.

Wenn `first` anstelle der *step* Nummer angegeben wird, wird der erste Schritt verwendet, der durch `show new_proof /unknown` angezeigt wird.

Wenn *step* 0 oder negativ ist, wird der -*step*-te von dem letzten unbekannten Schritt, wie durch `show new_proof /unknown` gezeigt, verwendet. `improve -1` verwendet den vorletzten unbekannten Schritt, `improve -2` *label* den vorvorletzten, und `improve 0` ist dasselbe wie `improve last`.

Optionaler Befehlszeilenparameter:

`/depth` *number* - Diese Option bewirkt, dass bei der Suche auch Anweisungen mit `$e`-Hypothesen (aber keine neuen Variablen in den `$e`-Hypothesen) berücksichtigt werden, sofern das Backtracking die angegebene Tiefe nicht überschritten hat. *Warnung:* Versuchen Sie `/depth 1`, dann `2`, dann `3` usw. der Reihe nach wegen möglicher exponentieller Blowups. Speichern Sie Ihre Arbeit, bevor Sie `/depth` größer als `1` ausprobieren!

`/no_distinct` - Überspringt Anweisungen, welche `$d`-Anforderungen haben. Diese Option verhindert Zuweisungen, die gegen `$d`-Anforderungen verstoßen könnten, aber er könnten dann auch mögliche legale Zuweisungen übersehen werden.

Siehe auch: `set search_limit`

5.6.13 `save new_proof`-Befehl

Syntax: `save new_proof` *label* [`/normal`] [`/compressed`]

Der Befehl `save new_proof` ist nur im Beweis-Assistenten verfügbar. Er speichert den laufenden Beweis in den Quellpuffer. `save new_proof` kann verwendet werden, um einen fertigen Beweis zu speichern, oder um einen sich in Bearbeitung befindenden Beweis zu speichern, um ihn später weiter zu bearbeiten. Wenn ein unvollständiger Beweis gespeichert wird, gehen alle Benutzerzuweisungen mit `let step` oder `let variable` verloren, ebenso wie alle mehrdeutigen Vereinheitlichungen, die manuell aufgelöst wurden. Um die Wiederherstellung zu erleichtern, kann es hilfreich sein, `improve all` vor `save new_proof` zu verwenden, damit der unvollständige Beweis so viele Informationen wie möglich enthält.

Beachten Sie, dass der Beweis erst dann dauerhaft gespeichert wird, wenn der Befehl `write source` aufgwerufen wird.

Optionale Befehlszeilenparameter:

`/normal` - Der Beweis wird im normalen Format gespeichert (d. h. als eine Folge von Labels, was dem definierten Format der Metamath-Basissprache

entspricht). Dies ist das Standardformat, das verwendet wird, wenn ein Qualifier weggelassen wird.

`/compressed` - Der Beweis wird im komprimierten Format gespeichert, was den Speicherbedarf für eine Datenbasis reduziert. (Siehe Anhang B.)

5.7 Erstellen von LaTeX-Ausgaben

Sie können LaTeX-Ausgaben anhand der Informationen in einer Datenbasis erzeugen. Die Datenbasis muss bereits die erforderlichen Schriftsatzinformationen enthalten (siehe Abschnitt 4.4.2 für die Bereitstellung dieser Informationen).

Die Befehle `show statement` und `show proof` haben jeweils einen speziellen `/tex` Befehlszeilenparameter, der eine LaTeX-Ausgabe erzeugt (Der Befehl `show statement` verfügt auch über die Option `/simple_tex` für eine Ausgabe, die leichter von Hand zu bearbeiten ist). Bevor Sie diese Befehle verwenden können, müssen Sie eine LaTeX-Datei öffnen, an die Sie ihre Ausgabe senden können. Eine typische vollständige Sitzung verwendet diese Abfolge von Metamath-Befehlen:

```
read set.mm
open tex example.tex
show statement a1i /tex
show proof a1i /all/lemmon/renumber/tex
show statement uneq2 /tex
show proof uneq2 /all/lemmon/renumber/tex
close tex
```

Siehe Abschnitt 4.4.1 für Informationen über Kommentarauszeichnungen und Anhang A für Informationen darüber, wie die Übersetzung von mathematischen Symbolen angegeben wird.

Um den LaTeX-Quelltext zu formatieren und zu drucken, benötigen Sie das Programm LaTeX das auf den meisten Linux-Installationen standardmäßig vorhanden und für Windows verfügbar ist. Um unter Linux eine pdf-Datei zu erstellen, geben Sie normalerweise an der Shell-Eingabeaufforderung Folgendes ein

```
$ pdflatex example.tex
```

5.7.1 `open tex`-Befehl

Syntax: `open tex` *Dateiname* [`/no_header`]

Dieser Befehl öffnet eine Datei zum Schreiben von LaTeX-Quelltext und schreibt einen LaTeX-Header in die Datei. LaTeX-Quelltext kann mit den Befehlen `show proof`, `show new_proof` und `show statement` unter Verwendung der Option `/tex` geschrieben werden.

Die Zuordnung zu LaTeX-Symbolen wird in einem speziellen Kommentar definiert, der ein `$t`-Token enthält, so wie im Anhang A beschrieben.

Es gibt einen optionalen Befehlszeilenparameter:

`/no_header` - Diese Option verhindert, dass ein standardmäßiger LaTeX-Header und -Trailer in den ausgegebenen LaTeX-Code aufgenommen wird.

5.7.2 `close tex`-Befehl

Syntax: `close tex`

Dieser Befehl schreibt einen Trailer in jede LaTeX-Datei, die mit `open tex` geöffnet wurde (sofern nicht `/no_header` mit `open tex` verwendet wurde) und schließt die LaTeX-Datei.

5.8 Erstellen von HTML-Ausgaben

Sie können anhand der Informationen in einer Datenbasis Webseiten generieren. Die Datenbasis muss bereits die notwendigen Schriftsatzinformationen enthalten (siehe Abschnitt 4.4.2, wie man diese Informationen bereitstellt). Die Fähigkeit, HTML-Webseiten zu erzeugen, wurde in Metamath Version 0.07.30 hinzugefügt.

Um (eine) HTML Ausgabedatei(en) für `$a`- oder `$p`-Anweisung(en) zu erstellen, verwenden Sie

`show statement` *label-match* `/html`

Die Ausgabedatei erhält für jede Übereinstimmung den Namen *label-match*`.html`. Wenn *label-match* Platzhalterzeichen (`*`) enthält, werden für alle Anweisungen mit übereinstimmenden Labels HTML-Dateien erzeugt. Wenn *label-match* einen Platzhalter (`*`) enthält, werden außerdem zwei zusätzliche Dateien, `mmdefinitions.html` und `mmascii.html`, erzeugt. Um nur diese beiden zusätzlichen Dateien zu erzeugen, können Sie anstelle von *label-match* `?*` verwenden, das mit keinem Label einer Anweisung übereinstimmt.

Es gibt drei weitere Optionen für `show statement`, die ebenfalls HTML-Code erzeugen. Diese sind `/alt_html`, `/brief_html` und `/brief_alt_html`, welche im nächsten Abschnitt beschrieben werden.

Der Befehl

`show statement` *label-match* `/alt_html`

bewirkt das Gleiche wie `show statement` *label-match* `/html`, außer dass der HTML-Code für die Symbole aus `althtmldef`-Anweisungen anstelle von `htmldef`-Anweisungen im `$t`-Kommentar entnommen wird.

Der Befehl

```
show statement * /brief_html
```

ruft einen speziellen Modus auf, der nur Definitions- und Theoremlisten zusammen mit den zugehörigen Symbolen in einem Format ausgibt, das sich zum Kopieren und Einfügen in eine andere Webseite eignet (z. B. in die Tutorial-Seiten auf der Metamath-Website).

Der Befehl

```
show statement * /brief_alt_html
```

bewirkt schließlich dasselbe wie `show statement * / brief_html` für die alternative HTML Darstellung des Symbols.

Der Kommentar einer Anweisung kann eine spezielle Notation enthalten, die eine gewisse Kontrolle über die HTML-Version des Kommentars bietet. Siehe Abschnitt 4.4.1 (p. 159) für die Kommentarauszeichnungsfunktionen.

Die Befehle `write theorem_list` und `write bibliography`, die im Folgenden beschrieben werden, bieten als Nebeneffekt eine vollständige Fehlerprüfung für alle in diesem Abschnitt beschriebenen Funktionen.

5.8.1 `write theorem_list` -Befehl

Syntax: `write theorem_list` [`/theorems_per_page` *number*]

Dieser Befehl schreibt eine Liste aller `$a`- und `$p`-Anweisungen in der Datenbasis in eine Webseitendatei namens `mmtheorems.html`. Wenn weitere Dateien benötigt werden, heißen sie `mmtheorems2.html`, `mmtheorems3.html`, usw.

Optionaler Befehlszeilenparameter:

`/theorems_per_page` *number* - Diese Option gibt die Anzahl der Anweisungen an, die pro Webseite geschrieben werden sollen. Der Standardwert ist 100.

Anmerkung: In Version 0.177 von Metamath setzen die „Nearby theorems“-Links auf den einzelnen Webseiten 100 Theoreme pro Seite voraus, wenn sie auf eine Seite mit einer Theoremliste verweisen. Daher führt die Option `/theorems_per_page`, wenn sie eine andere Zahl als 100 angibt, dazu, dass die einzelnen Webseiten nicht mehr synchron sind, und sollte nicht verwendet werden, um die Haupttheoremliste für die Website zu erzeugen. Dieses Problem wird möglicherweise in einer zukünftigen Version behoben.

5.8.2 `write bibliography` -Befehl

Syntax: `write bibliography` *filename*

Dieser Befehl liest eine bestehende bibliografische Querverweisdatei ein, die normalerweise `mmbiblio.html` heißt, und aktualisiert sie anhand der bibliografischen Links in den Datenbasiskommentaren. Die Datei wird zwischen den HTML Kommentarzeilen `<!-- #START# -->` und `<!-- #END# -->` aktualisiert. Die ursprüngliche Eingabedatei wird in *Dateiname*`~1` umbenannt.

Ein bibliografischer Verweis wird mit dem Namen der Referenz in Klammern angegeben, z. B. `Theorem 3.1 aus [Monk] S. 22`. Siehe Abschnitt 5.8 (p. 200) für Einzelheiten zur Syntax.

5.8.3 `write recent_additions` -Befehl

Syntax: `write recent_additions` *filename* [`/limit` *number*]

Dieser Befehl liest eine vorhandene Datei „Recent Additions“ HTML ein, die normalerweise `mmrecent.html` heißt, und aktualisiert sie mit den Beschreibungen der Theoreme, die der Datenbasis zuletzt hinzugefügt wurden. Die Datei wird zwischen den HTML Kommentarzeilen `<!-- #START# -->` und `<!-- #END# -->` aktualisiert. Die ursprüngliche Eingabedatei wird in *Dateiname*`~1` umbenannt.

Optionaler Befehlszeilenparameter:

`/limit` *number* - Dieser Qualifier gibt die Anzahl der neuesten Theoreme an, die in die Ausgabedatei geschrieben werden sollen. Der Standardwert ist 100.

5.9 Text File Utilities

5.9.1 `tools`-Befehl

Syntax: `tools`

Dieser Befehl ruft ein einfach zu bedienendes, universelles Dienstprogramm zur Bearbeitung des Inhalts von Textdateien auf. Nach der Eingabe von `tools` ändert sich die Eingabeaufforderung in `TOOLS>`, bis Sie `exit` eingeben. Mit den `tools`-Befehlen können Sie einfache, globale Änderungen an einer Ein-/Ausgabedatei vornehmen, z. B. eine Zeichenkettenersetzung in jeder Zeile vornehmen, eine Zeichenkette zu jeder Zeile hinzufügen usw. Eine typische Verwendung dieses Dienstprogramms ist die Erstellung einer `submit`-Eingabedatei, um eine allgemeine Operation an einer Liste von Anweisungen durchzuführen, die mittels `show label` oder `show usage` erzeugt wurde.

Die Aktionen der meisten `tools`-Befehle können auch mit äquivalenten (und leistungsfähigeren) Unix-Shell-Befehlen ausgeführt werden, und manche Benutzer finden diese vielleicht effizienter. Aber für Windows-Benutzer oder Benutzer, die mit Unix nicht vertraut sind, bietet `tools` eine leicht zu erlernende Alternative, die für die meisten Aufgaben der Skripterstellung ausreicht, die für die effektive Nutzung des Metamath-Programms erforderlich sind.

5.9.2 `help`-Befehl (in `tools`)

Syntax: `help`

Der Befehl `help` listet die im Dienstprogramm `tools` verfügbaren Befehle zusammen mit einer kurzen Beschreibung auf. Für jeden Befehl gibt es wiederum eine eigene Hilfe, wie z. B. `help add`. Wie bei der `MM>`-Eingabeaufforderung von Metamath kann ein kompletter Befehl auf einmal eingegeben werden, oder es kann nur das Befehlswort eingegeben werden, woraufhin das Programm nach jedem Argument fragt.

Befehle für eine zeilenweise Bearbeitung:

`add` - Fügt eine angegebene Zeichenkette zu jeder Zeile einer Datei hinzu.

`clean` - Schneidet Leerzeichen und Tabulatoren in jeder Zeile einer Datei ab; konvertiert Zeichen.

`delete` - Löschen eines Abschnitts jeder Zeile in einer Datei.

`insert` - Fügt eine Zeichenkette in einer bestimmten Spalte in jeder Zeile einer Datei ein.

`substitute` - Führt eine einfache Ersetzung in jeder Zeile der Datei durch.

`tag` - Wie `add`, aber beschränkt auf einen Bereich von Zeilen.

`swap` - Vertauscht die beiden Hälften jeder Zeile in einer Datei.

Andere Befehle zur Dateiverarbeitung:

`break` - Zerlegt (tokenisiert) eine Datei in eine Liste von Token (eines pro Zeile).

`build` - Erzeugt eine Datei mit mehreren Token pro Zeile aus einer Liste.

`count` - Zählt die Vorkommen einer bestimmten Zeichenfolge in einer Datei.

`number` - Erstellt eine Liste von Zahlen.

`parallel` - Schaltet zwei Dateien nebeneinander.

`reverse` - Kehrt die Reihenfolge der Zeilen in einer Datei um.

`right` - Richtet Zeilen in einer Datei rechtsbündig aus (nützlich vor dem Sortieren von Zahlen).

`sort` - Sortiert die Zeilen in einer Datei nach dem Schlüssel, der mit der angegebenen Zeichenfolge beginnt.

`match` - Extrahiert Zeilen, die eine bestimmte Zeichenfolge enthalten (oder nicht).

`unduplicate` - Eliminiert doppelt vorkommende Zeilen in einer Datei.

`duplicate` - Extrahiert das erste Vorkommen einer Zeile, die mehr als einmal in einer Datei vorkommt, und verwirft Zeilen, die genau einmal vorkommen.

`unique` - Extrahiert Zeilen, die genau einmal in einer Datei vorkommen.

`type` (10 Zeilen) - Zeigt die ersten paar Zeilen einer Datei an. Ähnlich wie bei Unix `head`.

`copy` - Ähnlich wie Unix `cat`, aber sicher (gleiche Eingabe- und Ausgabedatei erlaubt).

`submit` - Führt ein Skript aus, das `tools` Befehle enthält.

Hinweis: `unduplicate`, `duplicate` und `unique` sortieren als Nebeneffekt auch die Zeilen.

5.9.3 Verwendung von `tools` zur Erstellung von Metamath `submit`-Skripten

Der Befehl `break` Befehl wird normalerweise verwendet, um eine Reihe von Anweisungs-Labels, wie z.B. die Ausgabe von Metamaths `show usage`, in ein Label pro Zeile aufzubrechen. Die anderen `tools`-Befehle können dann verwendet werden, um Zeichenketten vor und nach jedem Label einer Anweisung hinzuzufügen, um Befehle anzugeben, die für die Anweisung ausgeführt werden sollen. Der Befehl `parallel` ist nützlich, wenn ein Label für eine Anweisung mehr als einmal in einer Zeile erwähnt werden muss.

Sehr oft erfordert ein Skript für Metamath mehrere Befehlszeilen für jede zu verarbeitende Anweisung. Sie möchten zum Beispiel den Beweis-Assistenten starten, `minimize_with` für Ihren zuletzt bearbeitetes Theorem ausführen, mit `save` den neuen Beweis speichern und mit `exit` den Beweis-Assistenten beenden. Um dies zu erreichen, können Sie eine Datei mit diesen vier Befehlen für jede Anweisung in einer einzigen Zeile erstellen und jeden Befehl durch ein bestimmtes Zeichen wie `@` abtrennen. Am Ende können Sie dann jedes `@` mit `\n` ersetzen, um die Zeilen in einzelne Befehlszeilen aufzulösen (siehe `help substitute`).

5.9.4 Beispiel für eine `tools`-Sitzung

Um Ihnen ein schnelles Gefühl für das Dienstprogramm `tools` zu vermitteln, zeigen wir eine einfache Sitzung, in der wir eine Datei `n.txt` mit 3 Zeilen erstellen, Zeichenketten vor und nach jeder Zeile hinzufügen und die Zeilen auf dem Bildschirm anzeigen. Sie können mit den verschiedenen Befehlen experimentieren, um Erfahrungen mit dem Dienstprogramm `tools` zu sammeln.

```
MM> tools
Entering the Text Tools utilities.
Type HELP for help, EXIT to exit.
TOOLS> number
Output file <n.tmp>? n.txt
First number <1>?
Last number <10>? 3
Increment <1>?
TOOLS> add
Input/output file? n.txt
String to add to beginning of each line <>? This is line
String to add to end of each line <>? .
The file n.txt has 3 lines; 3 were changed.
```

```
First change is on line 1:
This is line 1.
TOOLS> type n.txt
This is line 1.
This is line 2.
This is line 3.
TOOLS> exit
Exiting the Text Tools.
Type EXIT again to exit Metamath.
MM>
```

Anhang A

Beispielhafte Darstellungen

Dieser Anhang enthält eine Auswahl von ASCII-Darstellungen, die entsprechenden traditionellen mathematischen Symbole und eine Erläuterung ihrer Bedeutungen in der Datenbasis `set.mm`. Die Symbole sind in der Reihenfolge ihres Auftretens aufgeführt. Dies ist nur eine unvollständige Liste, und es werden laufend neue Definitionen hinzugefügt. Eine vollständige Liste finden Sie unter `http://metamath.org`.

Diese ASCII-Darstellungen sowie Informationen zu ihrer Anzeige werden in der Datenbasisdatei `set.mm` in einem speziellen Kommentar namens `$t` *comment* oder *Schriftsatzkommentar* definiert. Ein Schriftsatzkommentar ist durch die zweistellige Zeichenfolge `$t` am Anfang des Kommentars gekennzeichnet. Weitere Informationen finden Sie in Abschnitt 4.4.2, S. 162.

In der folgenden Tabelle zeigt die Spalte „ASCII“ die ASCII-Darstellung, „Symbol“ die mathematisch-symbolische Darstellung, die dieser ASCII-Darstellung entspricht, „Labels“ die Schlüssel-Labels, die die Darstellung definieren, und „Beschreibung“ liefert eine Beschreibung des Symbols. Wie üblich ist „gdw“ die Abkürzung für „genau dann, wenn“ oder „dann und nur dann, wenn“. In den meisten Fällen zeigt die Spalte „ASCII“ nur das Schlüssel-Token an, aber manchmal wird auch eine Folge von Token angezeigt, wenn dies für die Verständlichkeit notwendig ist.

ASCII	Symbol	Labels	Beschreibung
`|-`	$\vdash$		„Es ist beweisbar, dass...“
`ph`	φ	`wph`	Die (boolesche) wff-Variable Phi, üblicherweise die erste wff-Variable.

ASCII	Symbol	Labels	Beschreibung
`ps`	ψ	`wps`	Die (boolesche) wff-Variable Psi, üblicherweise die zweite wff-Variable.
`ch`	χ	`wch`	Die (boolesche) wff-Variable Chi, üblicherweise die dritte wff-Variable.
`-.`	$\neg$	`wn`	Logisch nicht. Wenn z.B. φ wahr ist, dann ist $\neg\varphi$ falsch.
`->`	$\rightarrow$	`wi`	Impliziert, auch als materielle Implikation bezeichnet. In der klassischen Logik ist der Ausdruck $\varphi \rightarrow \psi$ wahr, wenn entweder φ falsch oder ψ wahr ist (oder beides), d.h. $\varphi \rightarrow \psi$ hat die gleiche Bedeutung wie $\neg\varphi \vee \psi$ (wie in Theorem `imor` bewiesen).
`<->`	$\leftrightarrow$	`df-bi`	Bikonditional (auch bekannt als istgleich für boolesche Werte). $\varphi \leftrightarrow \psi$ ist wahr, wenn und nur dann wenn φ und ψ den gleichen Wert haben.
`\/`	$\vee$	`df-or`, `df-3or`	Disjunktion (logisches „oder“). $\varphi \vee \psi$ ist wahr, wenn φ, ψ, oder beide wahr sind.
`/\`	$\wedge$	`df-an`, `df-3an`	Konjunktion (logisches „und“). $\varphi \wedge \psi$ ist wahr, wenn sowohl φ als auch ψ wahr sind.
`A.`	$\forall$	`wal`	Für alle; die wff $\forall x\varphi$ ist wahr, wenn φ für alle Werte von x wahr ist.
`E.`	$\exists$	`df-ex`	Es existiert; die wff $\exists x\varphi$ ist wahr, wenn es mindestens ein x gibt, für das φ wahr ist.
`[ y / x ]`	$[y/x]$	`df-sb`	Die wff $[y/x]\varphi$ ist das Ergebnis, wenn y in φ echt durch x ersetzt wird (y ersetzt x). Zum Beispiel ist $[x/y]z \in y$ das Gleiche wie $z \in x$.
`E!`	$\exists!$	`df-eu`	Es gibt genau ein; $\exists!x\varphi$ ist wahr, wenn es genau ein x gibt, bei dem φ wahr ist.
`{ y | phi }`	$\{y\|\varphi\}$	`df-clab`	Die Klasse aller Mengen, in denen φ wahr ist.

ASCII	Symbol	Labels	Beschreibung
`=`	$=$	`df-cleq`	Klassengleichheit; $A = B$ gdw A gleich B ist.
`e.`	$\in$	`df-clel`	Klassenzugehörigkeit; $A \in B$ gdw A ein Element von B ist.
`_V`	V	`df-v`	Klasse aller Mengen (selbst keine Menge).
`C_`	$\subseteq$	`df-ss`	Unterklasse (Untermenge); $A \subseteq B$ ist wahr gdw A eine Unterklasse von B ist.
`u.`	$\cup$	`df-un`	$A \cup B$ ist die Vereinigung der Klassen A und B.
`i^i`	$\cap$	`df-in`	$A \cap B$ ist die Schnittmenge der Klassen A und B.
`\`	$\setminus$	`df-dif`	$A \setminus B$ (Klassendifferenz) ist die Klasse aller Mengen in A mit Ausnahme derjenigen in B.
`(/)`	$\varnothing$	`df-nul`	$\varnothing$ ist die leere Menge.
`~P`	$\mathcal{P}$	`df-pw`	Potenzklasse.
`<. A , B >.`	$\langle A, B \rangle$	`df-op`	Das geordnete Paar $\langle A, B \rangle$.
`( F ` ` A )`	$(F`A)$	`df-fv`	Der Wert der Funktion F an der Stelle A.
`_i`	i	`df-i`	Die Quadratwurzel von minus eins.
`x.`	$\cdot$	`df-mul`	Multiplikation komplexer Zahlen; $2 \cdot 3 = 6$.
`CC`	$\mathbb{C}$	`df-c`	Die Menge der komplexen Zahlen.
`RR`	$\mathbb{R}$	`df-r`	Die Menge der reellen Zahlen.

Anhang B

Komprimierte Beweise

Die Beweise in der Mengenlehre-Datenbasis `set.mm` werden aus Effizienzgründen in einem komprimierten Format gespeichert. Normalerweise brauchen Sie sich nicht um das komprimierte Format zu kümmern, da Sie es mit den üblichen Werkzeugen zur Anzeige von Beweisen im Metamath-Programm (`show proof...`) anzeigen oder in das normale RPN-Beweisformat konvertieren können, so wie in Abschnitt 4.3 beschrieben (mit `save proof` *label* `/normal`). Der Vollständigkeit halber beschreiben wir hier jedoch das Format und zeigen, wie es auf das normale RPN-Beweisformat abgebildet wird.

Ein komprimierter Beweis, der sich zwischen den Schlüsselwörtern `$=` und `$.` befindet, besteht aus einer linken Klammer, einer Folge von Anweisungs-Labels, einer rechten Klammer und einer Folge von Großbuchstaben `A` bis `Z` (mit optionalem Whitespace dazwischen). Die Klammern und die Labels müssen von Whitespace umgeben sein. Die linke Klammer sagt Metamath, dass ein komprimierter Beweis folgt (Ein normaler RPN-Beweis besteht nur aus einer Folge von Labels, und eine Klammer ist kein zulässiges Zeichen in einem Label).

Die Folge von Großbuchstaben entspricht einer Folge von ganzen Zahlen mit der folgenden Zuordnung. Jede ganze Zahl entspricht einem Beweisschritt, wie später beschrieben.

$$
\begin{aligned}
\texttt{A} &= 1 \\
\texttt{B} &= 2 \\
&\dots \\
\texttt{T} &= 20 \\
\texttt{UA} &= 21 \\
\texttt{UB} &= 22 \\
&\dots \\
\texttt{UT} &= 40 \\
\texttt{VA} &= 41
\end{aligned}
$$

$$\begin{array}{c} \texttt{VB} = 42 \\ \ldots \\ \texttt{YT} = 120 \\ \texttt{UUA} = 121 \\ \ldots \\ \texttt{YYT} = 620 \\ \texttt{UUUA} = 621 \\ \text{etc.} \end{array}$$

Mit anderen Worten: `A` bis `T` stehen für die niedrigstwertige Ziffer zur Basis 20, und `U` bis `Y` stehen für null oder mehr höchstwertige Ziffern zur Basis 5, wobei die Ziffern bei 1 anstelle der üblichen 0 beginnen. Bei diesem Schema brauchen wir keinen Whitespace zwischen diesen „Ziffern".

(Beim Entwurf des komprimierten Beweisformats wurden nur Großbuchstaben gewählt, im Gegensatz zu allen druckbaren nicht-Whitespace ASCII-Zeichen außer `$`, um nicht mit der Suchfunktion der meisten Texteditorn in Konflikt zu geraten. Dadurch wird ein Kompressionsverlust von typischerweise 20% i Kauf genommen. Die Gruppierung Basis 5/Basis 20 wurde gewählt, z.B. statt Basis 6/Basis 19, nachdem experimentell die Gruppierung ermittelt wurde, die in typischen Fällen die beste Kompression ergab).

Der Buchstabe `Z` kennzeichnet („taggt") einen Beweisschritt, der mit einem später im Beweis vorkommenden Schritt übereinstimmt; dadurch wird ein Beweis verkürzt, indem identische Beweisschritte nicht immer wieder erneut bewiesen werden müssen (was bei der Erstellung von wff's häufig vorkommt). Das `Z` wird unmittelbar nach der niedrigstwertigen Ziffer (Buchstaben `A` bis `T`) platziert, die die ganze Zahl beendet, die dem Schritt entspricht, auf den später verwiesen werden soll.

Die ganzen Zahlen, denen die Großbuchstaben entsprechen, werden wie folgt auf Labels abgebildet. Wenn die zu beweisende Aussage m zwingende Hypothesen hat, entsprechen die ganzen Zahlen 1 bis m den Labels dieser Hypothesen in der Reihenfolge, die durch den Befehl `show statement ... / full` angezeigt werden, d.h. der RPN-Reihenfolge der zwingenden Hypothesen. Die ganzen Zahlen $m + 1$ bis $m + n$ entsprechen den in den Klammern des komprimierten Beweises eingeschlossenen Labels, und zwar in der Reihenfolge ihres Auftretens, wobei n die Anzahl dieser Labels ist. Ganze Zahlen ab $m + n + 1$ entsprechen nicht direkt den Labels der Anweisung, sondern verweisen auf Beweisschritte, die mit dem Buchstaben `Z` gekennzeichnet sind, so dass auf diese Beweisschritte später im Beweis Bezug genommen werden kann. Die ganze Zahl $m+n+1$ entspricht dem ersten Schritt, der mit einem `Z` gekennzeichnet ist, $m + n + 2$ dem zweiten Schritt, der mit einem `Z` gekennzeichnet ist, usw. Wenn der komprimierte Beweis in einen normalen Beweis umgewandelt wird, ersetzt der gesamte Teilbeweis eines mit `Z` gekennzeichneten Schritts die Referenz auf diesen Schritt.

Aus Effizienzgründen arbeitet Metamath direkt mit komprimierten Beweisen, ohne sie intern in normale Beweise umzuwandeln. Zusätzlich zur

üblichen Fehlerprüfung wird eine Fehlermeldung ausgegeben, wenn (1) ein Label in der Label-Liste in Klammern nicht auf eine vorherige `$p`- oder `$a`-Anweisung oder eine nicht zwingende Hypothese der zu beweisenden Anweisung verweist und (2) ein mit `Z` markierter Beweisschritt vor dem mit `Z` markierten Schritt referenziert wird.

Wie bei einem normalen Beweis in der Entwicklung (Abschnitt 4.4.6) kann jeder Schritt oder Teilbeweis, der noch nicht bekannt ist, mit einem einzigen `?` dargestellt werden. Zwischen dem `?` und den Großbuchstaben (oder anderen `?`'s), die den Rest des Beweises darstellen, muss kein Whitespace eingefügt sein.

Anhang C

Das formale System von Metamath

C.1 Einführung

Vollkommenheit ist, wenn es nichts mehr wegzunehmen gibt.

Antoine de Saint-Exupery[1]

Dieser Anhang beschreibt die Theorie hinter der Metamath-Sprache in einer abstrakten Form, die für Mathematiker gedacht ist. Konkret konstruieren wir zwei Mengen-theoretische Objekte: ein „formales System“ (grob gesagt, eine Menge von Syntaxregeln, Axiomen und logischen Regeln) und sein „Universum“ (grob gesagt, die Menge der Theoreme, die im formalen System ableitbar sind). Die Computersprache Metamath bietet uns eine Möglichkeit, bestimmte formale Systeme zu beschreiben und mit Hilfe eines vom Benutzer bereitgestellten Beweises zu überprüfen, ob gegebene Theoreme zu ihren Universen gehören.

Um diesen Anhang zu verstehen, benötigen Sie Grundkenntnisse der informellen Mengenlehre. Es sollte ausreichen, z.B. Kap. 1 von Munkres' *Topology* zu verstehen[48] oder das einführende Kapitel zur Mengenlehre in vielen Lehrbüchern, die in die abstrakte Mathematik einführen. (Beachten Sie, dass es zwischen den Autoren kleinere Unterschiede in der Schreibweise gibt; z.B. verwendet Munkres $\subset$ anstelle unseres $\subseteq$ für „subset“. Wir verwenden „enthalten in“ für „eine Teilmenge von“ und „gehört zu“ oder „ist enthalten in“ für „ist ein Element von“). Was wir hier als „formale“ Beschreibung bezeichnen, ist anders als früher, eigentlich eine informelle Beschreibung in der gewöhnlichen Sprache der Mathematiker. Wir geben jedoch genügend Details an, so dass ein Mathematiker sie leicht formalisieren kann, sogar in

[1] Nach [9, S. 3-25].

der Sprache von Metamath selbst, falls gewünscht. Um die Logikbeispiele am Ende dieses Anhangs zu verstehen, wäre die Kenntnis eines einführenden Buches über mathematische Logik hilfreich.

C.2 Die formale Beschreibung

C.2.1 Vorbereitende Maßnahmen[2]

Mit ω bezeichnen wir die Menge aller natürlichen Zahlen (nichtnegative ganze Zahlen). Jede natürliche Zahl n wird mit der Menge aller kleineren Zahlen identifiziert: $n = \{m|m < n\}$. Die Formel $m < n$ ist also äquivalent zu der Bedingung: $m \in n$ und $m, n \in \omega$. Insbesondere ist 0 die Zahl Null und zugleich die leere Menge $\varnothing$, $1 = \{0\}$, $2 = \{0, 1\}$ usw. ${}^{B}A$ bezeichnet die Menge aller Funktionen von B nach A (d.h. mit dem Definitionsbereich B und einem in A enthaltenen Wertebereich). Die Elemente von ${}^{\omega}A$ sind so genannte *einfache unendliche Folgen*, mit allen *Gliedern* in A. Für den Fall $n \in \omega$ werden die Elemente von ${}^{n}A$ als *endliche n-gliedrige Folgen*, bezeichnet, wiederum mit Gliedern in A. Die aufeinanderfolgenden Glieder (Funktionswerte) einer endlichen oder unendlichen Folge f werden mit $f_0, f_1, \ldots, f_n, \ldots$ bezeichnet. Jede endliche Folge $f \in \bigcup_{n\in\omega} {}^{n}A$ bestimmt eindeutig die Zahl n, so dass $f \in {}^{n}A$; n heißt die *Länge* von f und wird mit $|f|$ bezeichnet. $\langle a\rangle$ ist die Folge f mit $|f| = 1$ und $f_0 = a$; $\langle a, b\rangle$ ist die Folge f mit $|f| = 2$, $f_0 = a$, $f_1 = b$; usw. Für zwei gegebene endliche Folgen f und g bezeichnen wir mit $f \frown g$ ihre *Verkettung*, d.h. die endliche Folge h, die durch die folgende Bedingungen bestimmt ist:

$$\begin{aligned} |h| &= |f| + |g|; \\ h_n &= f_n \qquad \text{for } n < |f|; \\ h_{|f|+n} &= g_n \qquad \text{for } n < |g|. \end{aligned}$$

C.2.2 Konstanten, Variablen und Ausdrücke

Ein formales System hat eine Menge von *Symbolen*, die mit *SM* bezeichnet wird. Eine genaue mengentheoretische Definition dieser Menge ist unwichtig; ein Symbol kann als primitives oder atomares Element betrachtet werden, wenn man will. Wir nehmen an, dass diese Menge in zwei disjunkte Teilmengen unterteilt ist: eine Menge *CN* von *Konstanten* und eine Menge *VR* von *Variablen*. *CN* und *VR* bestehen jeweils aus abzählbar vielen Symbolen, die in endlichen oder einfachen unendlichen Folgen $c_0, c_1, \ldots$ bzw. $v_0, v_1, \ldots$ ohne Wiederholumgen angeordnet werden können. Beliebige Symbole werden wir durch Metavariablen α, β usw. darstellen.

[2] Dieser Abschnitt ist größtenteils wörtlich von Tarski[69, p. 63] übernommen und frei übersetzt.

Kommentar: Die Variablen $v_0, v_1, \ldots$ unseres formalen Systems entsprechen dem, was in der Literatur zu spezifischen formalen Systemen gewöhnlich als „Metavariablen“ bezeichnet wird. Typischerweise wird bei der Beschreibung eines bestimmten formalen Systems in einem Buch eine Reihe von primitiven Objekten postuliert, die Variablen genannt werden, und dann werden deren Eigenschaften mit Hilfe von Metavariablen beschrieben, die sich über diese erstrecken, wobei die eigentlichen Variablen selbst nie wieder erwähnt werden. Unser formales System erwähnt diese primitiven, Variablen genannte Objekte überhaupt nicht, sondern befasst sich von Anfang an direkt mit Metavariablen als seine primitiven Objekten. Dies ist ein subtiler, aber wichtiger Unterschied, den man im Auge behalten sollte. Denn dadurch unterscheidet sich unsere Definition von „formalem System“ etwas von denen, die man normalerweise in der Literatur findet. (So sind die oben genannten Metavariablen α, β usw. eigentlich „Metametavariablen“, wenn sie zur Darstellung von $v_0, v_1, \ldots$ verwendet werden.)

Endliche Folgen, bei denen alle Glieder Symbole sind, heißen -*Ausdrücke*. EX ist die Menge aller Ausdrücke; also

$$EX = \bigcup_{n \in \omega} {}^n SM.$$

Ein *konstant-gepräfixter Ausdruck* ist ein Ausdruck der Länge ungleich Null, dessen erstes Glied eine Konstante ist. Wir bezeichnen die Menge aller konstant-gepräfixter Ausdrücke mit $EX_C = \{e \in EX | (|e| > 0 \wedge e_0 \in CN)\}$.

Ein *Konstante-Variable-Paar* ist ein Ausdruck der Länge 2, dessen erstes Glied eine Konstante ist und dessen zweites Glied eine Variable ist. Wir bezeichnen die Menge aller Konstanten-Variablen-Paare mit $EX_2 = \{e \in EX_C | (|e| = 2 \wedge e_1 \in VR)\}$.

Beziehung zu Metamath: Im Allgemeinen entspricht die Menge SM der Menge der deklarierten mathematischen Symbole in einer Metamath-Datenbasis, die Menge CN denjenigen Symbolen, die mit `$c`-Anweisungen deklariert sind, und die Menge VR denjenigen Symbolen, die mit `$v`-Anweisungen deklariert sind. Natürlich kann eine Metamath-Datenbasis nur eine endliche Anzahl von mathematischen Symbolen haben, während formale Systeme im Allgemeinen eine unendliche Anzahl haben können, obwohl die Anzahl der in Metamath verfügbaren mathematischen Symbole im Prinzip unbegrenzt ist. Die Menge EX_C entspricht der Menge der zulässigen Ausdrücke für `$e`-, `$a`- und `$p`-Anweisungen. Die Menge EX_2 entspricht der Menge der zulässigen Ausdrücke für `$f`-Anweisungen.

Wir bezeichnen mit $\mathcal{V}(e)$ die Menge aller Variablen in einem Ausdruck $e \in EX$, d.h. die Menge aller $\alpha \in VR$, so dass $\alpha = e_n$ für mindestens ein $n < |e|$. Wir bezeichnen auch (unter Missbrauch der Notation) mit $\mathcal{V}(E)$ die Menge aller Variablen in einer Sammlung von Ausdrücken $E \subseteq EX$, d.h. $\bigcup_{e \in E} \mathcal{V}(e)$.

C.2.3 Substitution

Bei einer Funktion F von VR nach EX bezeichnen wir mit σ_F oder einfach σ die Funktion von EX nach EX, die rekursiv für nichtleere Folgen durch

$$\sigma(< \alpha >) = F(\alpha) \qquad \text{mit } \alpha \in VR;$$

$$\sigma(<\alpha>) = <\alpha> \quad \text{mit } \alpha \notin VR;$$
$$\sigma(g \frown h) = \sigma(g) \frown \sigma(h) \quad \text{mit } g, h \in EX.$$

definiert ist.

Wir definieren außerdem $\sigma(\varnothing) = \varnothing$. Wir nennen σ eine *simultane Substitution* (oder einfach *Substitution*) mit *Substitutionsabbildung* F.

Mit $\sigma(E)$ bezeichnen wir (unter Missbrauch der Notation) auch eine Substitution auf einer Sammlung von Ausdrücken $E \subseteq EX$, d.h. die Menge $\{\sigma(e)|e \in E\}$. Die Sammlung $\sigma(E)$ kann natürlich weniger Ausdrücke als E enthalten, weil durch die Substitution doppelte Ausdrücke entstehen könnten.

C.2.4 Aussagen

Wir bezeichnen mit DV die Menge aller ungeordneten Paare $\{\alpha, \beta\} \subseteq VR$, so dass $\alpha \neq \beta$. DV steht für „unterscheidbare Variablen“.

Eine *Prä-Aussage* ist ein Quadrupel $\langle D, T, H, A\rangle$ derart, dass $D \subseteq DV$, $T \subseteq EX_2$, $H \subseteq EX_C$ und H endlich ist, $A \in EX_C$, $\mathcal{V}(H \cup \{A\}) \subseteq \mathcal{V}(T)$, und $\forall e, f \in T\ \mathcal{V}(e) \neq \mathcal{V}(f)$ (oder äquivalent, $e_1 \neq f_1$), wann immer $e \neq f$. Die Terme des Quadrupels werden respektive *disjunkte Variableneinschränkungen*,, *Variablentyp-Hypothesen*,, *logische Hypothesen*, und die *Behauptung* genannt. Wir bezeichnen mit T_M (*obligatorische Variablentyp-Hypothesen*) die Teilmenge von T, so dass $\mathcal{V}(T_M) = \mathcal{V}(H \cup \{A\})$. Wir bezeichnen mit $D_M = \{\{\alpha, \beta\} \in D|\{\alpha, \beta\} \subseteq \mathcal{V}(T_M)\}$ die *obligatorischen disjunkte Variableneinschränkung* der Prä-Aussage. Die Menge der *obligatorischen Hypothesen* ist $T_M \cup H$. Wir nennen das Quadrupel $\langle D_M, T_M, H, A\rangle$ das *Redukt* der Prä-Aussage $\langle D, T, H, A\rangle$.

Eine *Aussage* ist das Redukt einer Prä-Aussage. Eine Aussage ist also eine besondere Art von Prä-Aussage; insbesondere ist eine Aussage das Redukt ihrer selbst.

> *Kommentar:* T ist eine Menge von Ausdrücken der Länge 2, die eine Menge von Konstanten („Variablentypen“) mit einer Menge von Variablen verknüpfen. Die Bedingung $\mathcal{V}(H \cup \{A\}) \subseteq \mathcal{V}(T)$ bedeutet, dass jede Variable, die in den logischen Hypothesen oder Behauptungen einer Aussage vorkommt, eine zugehörige Variablentyp-Hypothese oder eine „Typendeklaration“ haben muss, in Analogie zu einer Programmiersprache für Computer, in der eine Variable beispielsweise als String oder Integer deklariert werden muss. Die Anforderung, dass $\forall e, f \in T\, e_1 \neq f_1$ für $e \neq f$ bedeutet, dass jede Variable eindeutig einer Konstanten zugeordnet sein muss, die ihren Variablentyp bezeichnet; z.B. kann eine Variable ein „wff“ oder ein „set“ sein, aber nicht beides.
>
> Disjunkte Variableneinschränkungen werden verwendet, um anzugeben, welche Variablensubstitutionen zulässig sind, damit die Anweisung gültig bleibt. In dem Theoremschema der Mengenlehre $\neg\forall x\, x = y$ dürfen wir zum Beispiel nicht dieselbe Variable sowohl für x als auch für y einsetzen. Andererseits verlangt das Theoremschema $x = y \to y = x$ nicht, dass x und y verschieden sein müssen, so dass wir keine disjunkte Variableneinschränkungen brauchen, obwohl eine solche Einschränkung nur dazu führen würde, dass das Schema weniger allgemein wäre.

Eine obligatorische Variablentyp-Hypothese ist eine, deren Variable in einer logischen Hypothese oder der Behauptung vorkommt. Eine beweisbare Prä-Aussage (siehe unten) kann nicht-obligatorische Variablentyp-Hypothesen erfordern, die im Endeffekt „Dummy"-Variablen zur Verwendung in ihrem Beweis einführen. Jede mögliche Anzahl von Dummy-Variablen kann für einen bestimmten Beweis erforderlich sein; tatsächlich wurde von H. Andréka [50] gezeigt, dass es keine endliche Obergrenze für die Anzahl der Dummy-Variablen gibt, die benötigt werden, um einen beliebiges Theorem in der Logik erster Ordnung (mit Gleichheit) zu beweisen, der eine feste Anzahl $n > 2$ von individuellen Variablen hat. (Siehe auch den Kommentar zu S. 140.) Aus diesem Grund setzen wir keine endliche Größenbeschränkung für die Sammlungen D und T, obwohl diese in einer tatsächlichen Anwendung (Metamath-Datenbasis) natürlich endlich sein werden und deren Anzahl wenn nötig vergrößert werden muss, wenn mehr Beweise hinzugefügt werden.

Beziehung zu Metamath: Eine Prä-Aussage eines formalen Systems entspricht einem erweiterten Rahmen in einer Metamath-Datenbasis (Abschnitt 4.2.7). Die Sammlungen D, T und H entsprechen den Sammlungen der Anweisungen `$d`, `$f` und `$e` in einem erweiterten Rahmen. Der Ausdruck A entspricht der Anweisung `$a` (oder `$p`) in einem erweiterten Rahmen. Eine Ausage eines formalen Systems entspricht einem Frame in einer Metamath-Datenbasis.

C.2.5 Formale Systeme

Ein *formales System* ist ein Tripel $\langle CN, VR, \Gamma \rangle$ wobei Γ eine Menge von Aussagen ist. Die Elemente von Γ werden *axiomatische Aussagen* genannt. Manchmal wird ein formales System nur mit Γ bezeichnet, wenn CN und VR klar sind.[3]

In einem formalen System Γ ist der *Abschluss*[4] einer Prä-Aussage $\langle D, T, H, A \rangle$ die kleinste Menge C von Ausdrücken, die so beschaffen ist, dass:

1. $T \cup H \subseteq C$ und
2. Wenn für eine axiomatische Aussage $\langle D'_M, T'_M, H', A' \rangle \in \Gamma$ und eine Substitution σ gilt
   a. $\sigma(T'_M \cup H') \subseteq C$ und
   b. für alle $\{\alpha, \beta\} \in D'_M$, für alle $\gamma \in \mathcal{V}(\sigma(\langle\alpha\rangle))$ und für alle $\delta \in \mathcal{V}(\sigma(\langle\beta\rangle))$ gilt $\{\gamma, \delta\} \in D$,

   dann gilt $\sigma(A') \in C$.

[3] Anm. der Übersetzer: Üblicherweise ist ein *formales System* ein Quadrupel $\langle A, B, \Gamma, R \rangle$ wobei A ein Alphabet (hier also $A = SM = CN \cup VR$), B eine Teilmenge aller Wörter, die sich über dem Alphabet A bilden lassen, also die Menge aller „wohlgeformten Formeln" oder eine „formale Sprache" über dem Alphabet A, Γ eine Menge von Aussagen, die als „Axiome" aufgefasst werden (es gilt $\Gamma \subseteq B$) und R eine Menge von zwei- oder mehrstelligen Relationen über Wörtern aus B (hier also die Substitution σ als Funktion/zweistellige Releation) ist.

[4] Diese Definition des Abschlusses enthält eine Vereinfachung, die Josh Purinton zu verdanken ist..

Eine Prä-Aussage $\langle D, T, H, A\rangle$ ist *beweisbar*, wenn $A \in C$ d.h. wenn ihre Behauptung zu ihrem Abschluss gehört. Eine Aussage ist *beweisbar*, wenn sie das Redukt einer beweisbaren Prä-Aussage ist. Das *Universum* eines formalen Systems ist die Sammlung aller seiner beweisbaren Aussagen. Man beachte, dass die Menge der axiomatischen Aussagen Γ in einem formalen System eine Teilmenge seines Universums ist.

Kommentar: Die erste Bedingung in der Definition des Abschlusses besagt einfach, dass die Hypothesen der Prä-Aussage in ihrem Abschluss enthalten sind.

Bedingung 2(a) besagt, dass es eine Substitution gibt, die dafür sorgt, dass die obligatorischen Hypothesen einer axiomatischen Aussage genau mit mindestens einem Element des Abschlusses übereinstimmt. Dies zeigen wir explizit in einem Beweis mittels der Metamath-Sprache.

Bedingung 2(b) beschreibt, wie die disjunkte Variableneinschränkung in der axiomatischen Aussage erfüllt werden müssen. Sie besagt, dass nach einer Substitution für zwei Variablen, die verschieden sein müssen, die beiden resultierenden Ausdrücke entweder keine Variablen enthalten dürfen oder, falls doch, keine Variablen gemeinsam haben dürfen, und dass jedes Paar von Variablen, die sie haben, mit einer Variablen aus jedem Ausdruck, in der ursprünglichen Anweisung als distinkt angegeben werden muss.

Beziehung zu Metamath: Axiomatische Aussagen und beweisbare Aussagen in einem formalen System entsprechen den Frames für `$a`- bzw. `$p`-Aussagen in einer Metamath-Datenbasis. Die Menge der axiomatischen Aussagen ist eine Teilmenge der Menge der beweisbaren Aussagen in einem formalen System, obwohl in einer Metamath-Datenbasis eine `$a`-Aussage dadurch gekennzeichnet ist, dass sie keinen Beweis hat. Ein Beweis in der Metamath-Sprache für eine `$p`-Anweisung sagt dem Computer, wie er explizit eine Folge von Elementen des Abschlusses konstruieren soll, was schließlich zu dem Nachweis führt, dass die zu beweisende Behauptung in dem Abschluss enthalten ist. Der tatsächliche Abschluss enthält normalerweise eine unendliche Anzahl von Ausdrücken. Ein formales System selbst hat kein explizites Objekt, das als „Beweis“ bezeichnet wird, sondern die Existenz eines Beweises wird indirekt durch die Zugehörigkeit einer Behauptung zum Abschluss einer beweisbaren Ausage impliziert. Wir tun dies, um das formale System leichter in der Sprache der Mengenlehre beschreiben zu können.

Wir weisen auch darauf hin, dass eine einmal als beweisbar nachgewiesene Aussage denselben Status wie eine axiomatische Aussage erhält, denn wenn die Menge der axiomatischen Aussagen um eine beweisbare Aussage erweitert wird, bleibt das Universum des formalen Systems unverändert (vorausgesetzt, dass VR unendlich ist). In der Praxis bedeutet dies, dass wir eine Hierarchie von beweisbaren Aussagen aufbauen können, um effizienter weitere beweisbare Aussagen zu ermitteln. Genau das tun wir in Metamath, wenn wir zulassen, dass Beweise auf vorherige `$p`-Anweisungen sowie auf vorherige `$a`-Anweisungen verweisen.

C.3 Beispiele für formale Systeme

Beziehung zu Metamath: Die Beispiele in diesem Abschnitt, mit Ausnahme von Beispiel 2, entsprechen größtenteils exakt dem Vorgehen in der Mengenlehre-Datenbasis `set.mm`. Vergleichen Sie die Beispiele 1, 3 und 5 mit Abschnitt 3.3, Beispiel 4 mit den Abschnitten 3.4.1 und 3.4.2 und Beispiel 6 mit Abschnitt 3.4.3.

C.3.1 Beispiel 1 — Aussagenlogik

Die klassische Aussagenlogik kann durch das folgende formale System beschrieben werden. Wir nehmen an, dass die Menge der Variablen unendlich ist. Anstatt die Konstanten und Variablen mit $c_0, c_1, \ldots$ und $v_0, v_1, \ldots$ zu bezeichnen, werden wir aus Gründen der Lesbarkeit stattdessen gängigere Symbole verwenden, wobei wir natürlich davon ausgehen, dass sie unterschiedliche primitive Objekte bezeichnen. Der Lesbarkeit halber können wir auch Kommas zwischen aufeinanderfolgenden Glieder einer Folge weglassen; so steht $\langle \text{wff } \varphi \rangle$ für $\langle \text{wff}, \varphi \rangle$.

Sei

$CN = \{\text{wff}, \vdash, \to, \neg, (,)\}$

$VR = \{\varphi, \psi, \chi, \ldots\}$

$T = \{\langle \text{wff } \varphi \rangle, \langle \text{wff } \psi \rangle, \langle \text{wff } \chi \rangle, \ldots\}$, d.h. diejenigen Ausdrücke der Länge 2, deren erstes Glied wff ist und deren zweites Glied zu VR gehört.[5]

Dann besteht Γ aus den axiomatischen Aussagen, die die Redukte der folgenden Prä-Aussagen sind:

$\langle \varnothing, T, \varnothing, \langle \text{wff } (\varphi \to \psi) \rangle \rangle$

$\langle \varnothing, T, \varnothing, \langle \text{wff } \neg\varphi \rangle \rangle$

$\langle \varnothing, T, \varnothing, \langle \vdash (\varphi \to (\psi \to \varphi)) \rangle \rangle$

$\langle \varnothing, T, \varnothing, \langle \vdash ((\varphi \to (\psi \to \chi)) \to ((\varphi \to \psi) \to (\varphi \to \chi))) \rangle \rangle$

$\langle \varnothing, T, \varnothing, \langle \vdash ((\neg\varphi \to \neg\psi) \to (\psi \to \varphi)) \rangle \rangle$

$\langle \varnothing, T, \{\langle \vdash (\varphi \to \psi) \rangle, \langle \vdash \varphi \rangle\}, \langle \vdash \psi \rangle \rangle$

Zum Beispiel ist das Redukt von $\langle \varnothing, T, \varnothing, \langle \text{wff } (\varphi \to \psi) \rangle \rangle$

$\langle \varnothing, \{\langle \text{wff } \varphi \rangle, \langle \text{wff } \psi \rangle\}, \varnothing, \langle \text{wff } (\varphi \to \psi) \rangle \rangle$,

und von $\langle \varnothing, T, \varnothing, \langle \vdash ((\varphi \to (\psi \to \chi)) \to ((\varphi \to \psi) \to (\varphi \to \chi))) \rangle \rangle$

$\langle \varnothing, \{\langle \text{wff } \varphi \rangle, \langle \text{wff } \psi \rangle, \langle \text{wff } \chi \rangle\}, \varnothing, \langle \vdash ((\varphi \to (\psi \to \chi)) \to ((\varphi \to \psi) \to (\varphi \to \chi))) \rangle \rangle$,

[5] Der Einfachheit halber lassen wir T eine unendliche Menge sein; die Definition einer Aussage erlaubt dies im Prinzip. Da eine Metamath-Quelldatei eine endliche Größe hat, müssen wir in der Praxis natürlich geeignete endliche Teilmengen dieses T verwenden, und zwar solche, die zumindest die obligatorischen Variablentyp-Hypothesen enthalten. In ähnlicher Weise führen wir in der Quelldatei nach Bedarf neue Variablen ein, wobei wir davon ausgehen, dass eine potenziell unendliche Anzahl von ihnen verfügbar ist.

welches die erste und die vierte axiomatischen Aussagen sind.

Wir nennen die Elemente von *VR wff-Variablen* oder (im Kontext der Logik erster Ordnung, die wir gleich beschreiben werden) *wff-Metavariablen*. Man beachte, dass die Symbole ϕ, ψ usw. tatsächliche spezifische Elemente der Menge *VR* bezeichnen; sie sind keine Metavariablen unserer Beschreibungssprache (die wir mit α, β usw. bezeichnen), sondern sind stattdessen (meta)konstante Symbole (Elemente der Menge *SM*) aus der Sicht unserer Beschreibungssprache. Das in [69] beschriebene äquivalente System der Aussagenlogik verwendet ebenfalls die Symbole ϕ, ψ usw. um wff-Metavariablen zu bezeichnen, aber in [69] sind dies im Gegensatz zu hier Metavariablen der Beschreibungssprache und keine primitiven Symbole des formalen Systems.

Die ersten beiden Aussagen definieren wffs: wenn φ und ψ wffs sind, dann ist $(\varphi \to \psi)$ auch eine wff; wenn φ eine wff ist, dann ist $\neg\varphi$ auch eine wff. Die nächsten drei sind die Axiome der Aussagenlogik: Wenn φ und ψ wffs sind, dann ist $\vdash (\varphi \to (\psi \to \varphi))$ ein (axiomatisches) Theorem; usw. Die letzte ist der Modus ponens: wenn φ und ψ wffs sind, und $\vdash (\varphi \to \psi)$ und $\vdash \varphi$ Theoreme sind, dann ist $\vdash \psi$ ein Theorem.

Die Entsprechung zur gewöhnlichen Aussagenlogik ist wie folgt. Wir betrachten nur beweisbare Aussagen der Form $\langle \varnothing, T, \varnothing, A \rangle$ mit T definiert wie oben. Der erste Term der Behauptung A einer solchen Aussage ist entweder „wff“ oder „$\vdash$“. Eine Aussage, bei der der erste Term „wff“ ist, ist eine *wff* der Aussagenlogik, und eine, bei der der erste Term „$\vdash$“ ist, ist ein *Theorem (Schema)* der Aussagenlogik.

Das Universum dieses formalen Systems enthält auch viele andere beweisbare Aussagen. Diejenigen mit Beschränkungen für unterschiedliche Variablen sind irrelevant, da die Aussagenlogik keine Beschränkungen für Substitutionen kennt. Diejenigen, die logische Hypothesen haben, nennen wir *Inferenzen*, wenn die logischen Hypothesen von der Form $\langle \vdash \rangle \frown w$ sind, wobei w eine wff ist (wobei der führende konstante Term „wff“ entfernt wurde). Inferenzen (mit Ausnahme des Modus ponens) sind kein eigentlicher Bestandteil der Aussagenlogik, lassen sich aber beim Aufbau einer Hierarchie von beweisbaren Aussagen gut verwenden. Eine beweisbare Aussage mit einer unsinnigen Hypothese wie $\langle \to, \vdash, \neg \rangle$ und demselben Ausdruck als Behauptung betrachten wir als irrelevant; sie kann beim Beweis von Theoremen nicht verwendet werden, da es keine Möglichkeit gibt, die unsinnige Hypothese zu eliminieren.

> *Kommentar:* Unsere Verwendung von Klammern in der Definition einer wff zeigt, dass axiomatische Aussagegen sorgfältig so formuliert werden sollten, dass sie eindeutig mit den vom formalen System erlaubten Substitutionen zusammenpassen. Es gibt viele Möglichkeiten, wffs zu definieren - die polnische Präfix-Notation hätte es uns beispielsweise erlaubt, die Klammern ganz wegzulassen, was allerdings zu Lasten der Lesbarkeit gegangen wäre -, aber wir müssen sie auf eine Weise definieren, die eindeutig ist. Hätten wir z.B. die Klammern in der Definition von $(\varphi \to \psi)$ weggelassen, hätte die wff $\neg\varphi \to \psi$ entweder als $\neg(\varphi \to \psi)$ oder $(\neg\varphi \to \psi)$ interpretiert werden können und hätte uns erlaubt, Unsinn zu beweisen. Es ist zu beachten, dass unser

formales System kein Konzept der Vorrangigkeit von Operatoren enthält.

C.3.2 Beispiel 2 — Prädikatenlogik mit Gleichheit

Hier erweitern wir Beispiel 1 um die Prädikatenlogik mit Gleichheit zu erhalten und veranschaulichen damit die Verwendung von disjunkte Variableneinschränkungen. Dieses System ist dasselbe wie Tarskis System $\mathfrak{S}_2$ in [69] (mit der Ausnahme, dass die Axiome der Aussagenlogik unterschiedlich, aber äquivalent sind, und dass ein redundantes Axiom weggelassen wird). Wir erweitern *CN* um die Konstanten $\{\mathrm{var}, \forall, =\}$ und *VR* um eine unendliche Menge von *individuelle Metavariablen* $\{x, y, z, \ldots\}$ und bezeichnen diese Teilmenge als *Vr*.

Wir erweitern *CN* auch um eine möglicherweise unendliche Menge *Pr* von *Prädikaten* $\{R, S, \ldots\}$. Wir assoziieren mit *Pr* eine Funktion rnk von *Pr* nach ω, und für $\alpha \in Pr$ nennen wir $\mathrm{rnk}(\alpha)$ den *Rang* des Prädikats α, der einfach die Anzahl der „Argumente“ ist, die das Prädikat hat. (Die meisten Anwendungen der Prädikatenlogik haben eine endliche Anzahl von Prädikaten; in der Mengenlehre gibt es z.B. ein einziges Prädikat mit zwei Argumenten (auch binäres Prädikat genannt) $\in$, das üblicherweise mit seinen Argumenten um das Prädikatssymbol herum geschrieben wird und nicht mit der Präfix-Notation, die wir für den allgemeinen Fall verwenden). Um unsere Diskussion zu erleichtern sei *Vs* eine beliebige feste injektive Funktion von ω nach *Vr*; somit ist *Vs* eine beliebige einfache unendliche Folge von einzelnen Metavariablen ohne Wiederholungen.

In diesem Beispiel verzichten wir auf die Funktionssymbole, die häufig Teil von Formalisierungen der Prädikatenlogik sind. Unter Verwendung metalogischer Argumente, die den Rahmen unserer Diskussion sprengen würden, kann gezeigt werden, dass unsere Formalisierung äquivalent ist, wenn Funktionen über geeignete Definitionen eingeführt werden.

Wir erweitern die in Beispiel 1 definierte Menge T um die Ausdrücke $\{\langle \mathrm{var}\ x\rangle, \langle \mathrm{var}\ y\rangle, \langle \mathrm{var}\ z\rangle, \ldots\}$ und das obige Γ um die axiomatischen Aussagen, die die Redukte der folgenden Prä-Aussagen sind:

$\langle \varnothing, T, \varnothing, \langle \mathrm{wff}\ \forall x\, \varphi\rangle\rangle$

$\langle \varnothing, T, \varnothing, \langle \mathrm{wff}\ x = y\rangle\rangle$

$\langle \varnothing, T, \{\langle \vdash \varphi\rangle\}, \langle \vdash \forall x\, \varphi\rangle\rangle$

$\langle \varnothing, T, \varnothing, \langle \vdash ((\forall x(\varphi \to \psi) \to (\forall x\, \varphi \to \forall x\, \psi))\rangle\rangle$

$\langle \{\{x, \varphi\}\}, T, \varnothing, \langle \vdash (\varphi \to \forall x\, \varphi)\rangle\rangle$

$\langle \{\{x, y\}\}, T, \varnothing, \langle \vdash \neg\forall x \neg x = y\rangle\rangle$

$\langle \varnothing, T, \varnothing, \langle \vdash (x = z \to (x = y \to z = y))\rangle\rangle$

$\langle \varnothing, T, \varnothing, \langle \vdash (y = z \to (x = y \to x = z))\rangle\rangle$

Dies sind die Axiome, die keine Prädikatssymbole beinhalten. Die ersten beiden Anweisungen erweitern die Definition einer wff. Die dritte ist die Regel

der Verallgemeinerung. Die fünfte besagt: „Für eine wff φ und die Variable x gilt: $\vdash (\varphi \to \forall x\, \varphi)$, sofern x nicht in φ vorkommt." Die sechste lautet: „Für die Variablen x und y gilt $\vdash \neg\forall x \neg x = y$, sofern x und y verschieden sind." (Dieser Vorbehalt ist nicht notwendig, wurde aber von Tarski eingefügt, um das Axiom abzuschwächen und trotzdem zu zeigen, dass das System logisch vollständig ist.)

Schließlich fügen wir für jedes Prädikatssymbol $\alpha \in Pr$ eine axiomatische Aussage zu Γ hinzu, die die Definition von wff erweitert und die das Redukt der folgenden Prä-Ausage ist:

$$\langle \varnothing, T, \varnothing, \langle \text{wff}, \alpha \rangle \rangle \frown Vs \restriction \text{rnk}(\alpha) \rangle$$

und für jedes $\alpha \in Pr$ und jedes $n < \text{rnk}(\alpha)$ fügen wir zu Γ ein Gleichheitsaxiom hinzu, das das Redukt der folgenden Prä-Aussage ist:

$$\begin{aligned} \langle \varnothing, T, \varnothing, \langle \vdash, (, Vs_n, =, Vs_{\text{rnk}(\alpha)}, \to, (, \alpha \rangle \frown Vs \restriction \text{rnk}(\alpha) \\ \frown \langle \to, \alpha \rangle \frown Vs \restriction n \frown \langle Vs_{\text{rnk}(\alpha)} \rangle \\ \frown Vs \restriction (\text{rnk}(\alpha) \setminus (n+1)) \frown \langle),) \rangle \rangle \end{aligned}$$

wobei $\restriction$ die Einschränkung des Definitionsbereichs und $\setminus$ die Mengendifferenz bezeichnet. Erinnern Sie sich daran, dass ein tiefgestellter Index in Vs einen seiner Terme kennzeichnet. (In den beiden obigen axiomatischen Aussagen werden Kommas zwischen aufeinanderfolgende Terme von Sequenzen gesetzt, um Mehrdeutigkeit zu vermeiden, und wenn Sie sie genau betrachten, werden Sie in der Lage sein, die Klammern, die konstante Symbole bezeichnen, von den Klammern unserer Beschreibungssprache, die Funktionsargumente abgrenzen, zu unterscheiden. Es wäre vielleicht besser gewesen, unsere primitiven Symbole fett zu schreiben, aber leider waren nicht für alle Zeichen in dem LaTeX-System, das für den Schriftsatz dieses Textes verwendet wurde, Fettschrift verfügbar). Diese scheinbar verbotenen Axiome lassen sich in Analogie zur Verkettung von Teilzeichenfolgen in einer Computersprache verstehen. Tatsächlich sind sie für jeden spezifischen Fall relativ einfach und werden deutlicher, wenn man den Spezialfall eines binären Prädikats $\alpha = R$ betrachtet, bei dem $\text{rnk}(R) = 2$ ist. Wenn Vs die Folge $\langle x, y, z, \ldots \rangle$ ist, wären die Axiome, die wir für diesen Fall zu Γ hinzufügen würden, die wff-Erweiterung und zwei Gleichheitsaxiome, die die Redukte der folgenden Aussagen sind:

$$\langle \varnothing, T, \varnothing, \langle \text{wff } Rxy \rangle \rangle$$
$$\langle \varnothing, T, \varnothing, \langle \vdash (x = z \to (Rxy \to Rzy)) \rangle \rangle$$
$$\langle \varnothing, T, \varnothing, \langle \vdash (y = z \to (Rxy \to Rxz)) \rangle \rangle$$

Studieren Sie diese sorgfältig, um zu sehen, wie sie aus den obigen allgemeinen Axiome entstehen. In der Praxis werden typischerweise nur wenige

Spezialfälle wie dieser benötigt, und in jedem Fall erlaubt uns die Metamath-Sprache nur die Beschreibung einer endlichen Anzahl von Prädikaten, im Gegensatz zu der unendlichen Anzahl, die das formale System erlaubt. (Sollte aus irgendeinem Grund eine unendliche Anzahl benötigt werden, könnten wir das formale System nicht direkt in der Metamath-Sprache definieren, sondern könnten es stattdessen metalogisch unter der Mengenlehre definieren, wie wir es in diesem Anhang tun, und nur die zugrunde liegende Mengenlehre mit ihrem einzigen binären Prädikat würde direkt in der Metamath-Sprache definiert).

Kommentar: Wie bereits erwähnt, handelt es sich bei den spezifischen Variablen, die durch die Symbole $x, y, z, \ldots \in Vr \subseteq VR \subseteq SM$ in Beispiel 2 dargestellt werden, nicht um die eigentlichen Variablen der gewöhnlichen Prädikatenlogik, sondern sie sind als Metavariablen zu betrachten, die sich über diese erstrecken. Zum Beispiel wäre eine disjunkte Variableneinschränkung für eigentliche Variablen der gewöhnlichen Prädikatenlogik bedeutungslos, da zwei verschiedene eigentliche Variablen per Definition unterschiedlich sind. Und wenn wir über einen beliebigen Repräsentanten $\alpha \in Vr$ sprechen, ist α eine Metavariable (in unserer Erklärungssprache), die sich über Metavariablen erstreckt (die Primitive unseres formalen Systems sind), von denen sich jede über die einzelnen eigentlichen Variablen der Prädikatenlogik erstreckt (die in unserem formalen System nie erwähnt werden).

Die oben genannte Konstante „var“ heißt in der `set.mm`-Datenbasis `setvar`, aber sie bedeutet dasselbe. Ich war der Meinung, dass „var“ im Kontext der Prädikatenlogik, deren Verwendung nicht auf die Mengenlehre beschränkt ist, ein sinnvollerer Name ist. Aus Gründen der Konsistenz bleiben wir in diesem Anhang bei dem Namen „var“, auch nachdem die Mengenlehre eingeführt wurde.

C.3.3 Freie Variablen und echte Substitution

Typische Darstellungen mathematischer Axiome verwenden Konzepte wie „freie Variable“, „gebundene Variable“ und „echte Substitution“ als primitive Begriffe. Eine freie Variable ist eine Variable, die kein Parameter eines Containerausdrucks ist. Eine gebundene Variable ist das Gegenteil einer freien Variable; sie ist eine Variable, die in einem Containerausdruck gebunden wurde. Zum Beispiel ist in dem Ausdruck $\forall x \varphi$ (für alle x ist φ wahr) die Variable x in dem „für alle“-Ausdruck ($\forall$) gebunden. Es ist möglich, eine Variable durch eine andere zu ersetzen, und diesen Vorgang nennt man „echte Substitution“. In den meisten Büchern hat die echte Substitution eine etwas komplizierte rekursive Definition mit mehreren Fällen, die auf dem Vorkommen von freien und gebundenen Variablen basieren. Sie können in [21, ch. 3–4] (sowie in vielen anderen Texten) für weitere formale Details zu diesen Begriffen nachschauen.

Die Verwendung dieser Konzepte als `primitives` schafft Komplikationen für Computerimplementierungen.

In dem System von Beispiel 2 gibt es keine primitiven Begriffe für freie Variablen und die echten Substitution. Tarski [69] zeigt, dass dieses System

logisch äquivalent zu den typischeren Lehrbuchsystemen ist, die diese primitiven Begriffe haben, wenn wir diese Begriffe mit geeigneten Definitionen und Metalogik einführen. Wir könnten auch direkt Axiome für solche Systeme definieren, obwohl die rekursiven Definitionen der freien Variablen und der echten Substitution unübersichtlich und umständlich zu handhaben wären. Stattdessen weisen wir auf zwei Hilfsmittel hin, die in der Praxis verwendet werden können, um diese Begriffe zu imitieren. (1) Anstatt eine spezielle Notation einzuführen, um (als logische Hypothese) „wobei x nicht frei in φ ist“ auszudrücken, können wir die logische Hypothese $\vdash (\varphi \to \forall x\, \varphi)$ verwenden.[6] (2) Es kann gezeigt werden, dass die wff $((x = y \to \varphi) \wedge \exists x(x = y \wedge \varphi))$ (mit den üblichen Definitionen von $\wedge$ und $\exists$; siehe Beispiel 4 unten) logisch äquivalent ist zu „die wff, die sich aus der echten Substitutiong von y für x in φ ergibt“. Das funktioniert unabhängig davon, ob x und y verschieden sind oder nicht.

C.3.4 Metalogische Vollständigkeit

In dem System von Beispiel 2 sind die folgenden Prä-Aussagen beweisbar (und ihre Redukte sind beweisbare Aussagen):

$$\langle \{\{x, y\}\}, T, \varnothing, \langle \vdash \neg\forall x \neg x = y \rangle \rangle$$
$$\langle \varnothing, T, \varnothing, \langle \vdash \neg\forall x \neg x = x \rangle \rangle$$

wohingegen die folgende Prä-Aussage meines Wissens nach nicht beweisbar ist (aber wir werden in der folgenden Diskussion so tun, als ob sie es nicht wäre:

$$\langle \varnothing, T, \varnothing, \langle \vdash \neg\forall x \neg x = y \rangle \rangle$$

Mit anderen Worten, wir können „$\neg\forall x \neg x = y$, wobei x und y verschieden sind“ und separat „$\neg\forall x \neg x = x$“ beweisen, aber wir können den kombinierten allgemeinen Fall „$\neg\forall x \neg x = y$“ nicht beweisen, der keine zusätzlich Bedingung hat. Dies beeinträchtigt jedoch nicht die logische Vollständigkeit, da die Variablen wirklich Metavariablen sind und die beiden beweisbaren Fälle zusammen alle möglichen Fälle abdecken. Der dritte Fall kann als ein Metatheorem betrachtet werden, dessen direkter Beweis mit dem System von Beispiel 2 außerhalb der Möglichkeiten des formalen Systems liegt.

Außerdem ist im System von Beispiel 2 die folgende Anweisung meines Wissens nicht beweisbar (wiederum eine Vermutung, die wir als wahr unterstellen werden):

$$\langle \varnothing, T, \varnothing, \langle \vdash (\forall x\, \varphi \to \varphi) \rangle \rangle$$

[6] Dies ist eine etwas schwächere Anforderung als „wobei x nicht frei in φ ist“. Ersetzen wir φ durch $x = x$, so haben wir den Satz $(x = x \to \forall x\, x = x)$, der die Hypothese erfüllt, obwohl x in $x = x$ frei ist. In einem solchen Fall sagen wir, dass x *effektiv nicht frei* in $x = x$ ist, da $x = x$ logisch äquivalent zu $\forall x\, x = x$ ist, in dem x gebunden ist.

Stattdessen können wir nur spezielle Fälle von φ mit individuellen Metavariablen beweisen und durch Induktion über die Formellänge die obige allgemeine Anweisung als Metatheorem außerhalb unseres formalen Systems beweisen. Die Einzelheiten dieses Beweises finden sich in [28].

Es gibt jedoch ein System der Prädikatenlogik, in dem alle derartigen „einfachen Metatheoreme“ wie die obigen direkt bewiesen werden können, und wir stellen es in Beispiel 3 vor. Ein *einfaches Metatheorem* ist jede Aussage des formalen Systems aus Beispiel 2, in dem alle disjukte Variableneinschränkung entweder aus zwei individuellen Metavariablen oder einer individuellen Metavariablen und einer wff-Metavariablen bestehen, und die durch Kombination von Fällen außerhalb des Systems wie oben bewiesen werden kann. Ein System ist *metalogisch vollständig*, wenn alle seine einfachen Metatheoreme (direkt) beweisbare Aussagen sind. Die genaue Definition von „einfachem Metatheorem“ und der Beweis der „metalogischen Vollständigkeit“ von Beispiel 3 findet sich in Bemerkung 9.6 und Theorem 9.7 von [41].

C.3.5 Beispiel 3 — Metalogisch vollständige Prädikatenlogik mit Gleichheit

Der Einfachheit halber nehmen wir an, dass es ein binäres Prädikat R gibt; dieses System reicht für die Mengenlehre aus, wobei das R natürlich das Prädikat $\in$ ist. Wir enennen die Axiome so, wie sie in [41] vorkommen. Dieses System ist logisch äquivalent zu dem in Beispiel 2 (wenn letzteres auf dieses eine binäre Prädikat beschränkt wird), ist aber auch metalogisch vollständig.

Angenommen

$CN = \{\text{wff}, \text{var}, \vdash, \to, \neg, (,), \forall, =, R\}$.

$VR = \{\varphi, \psi, \chi, \ldots\} \cup \{x, y, z, \ldots\}$.

$T = \{\langle \text{wff } \varphi\rangle, \langle \text{wff } \psi\rangle, \langle \text{wff } \chi\rangle, \ldots\} \cup \{\langle \text{var } x\rangle, \langle \text{var } y\rangle, \langle \text{var } z\rangle, \ldots\}$.

Dann besteht Γ aus den Redukten der folgenden Prä-Aussagen:

$\langle \varnothing, T, \varnothing, \langle \text{wff } (\varphi \to \psi)\rangle\rangle$

$\langle \varnothing, T, \varnothing, \langle \text{wff } \neg\varphi\rangle\rangle$

$\langle \varnothing, T, \varnothing, \langle \text{wff } \forall x\, \varphi\rangle\rangle$

$\langle \varnothing, T, \varnothing, \langle \text{wff } x = y\rangle\rangle$

$\langle \varnothing, T, \varnothing, \langle \text{wff } Rxy\rangle\rangle$

(C1′) $\langle \varnothing, T, \varnothing, \langle \vdash (\varphi \to (\psi \to \varphi))\rangle\rangle$

(C2′) $\langle \varnothing, T, \varnothing, \langle \vdash ((\varphi \to (\psi \to \chi)) \to ((\varphi \to \psi) \to (\varphi \to \chi)))\rangle\rangle$

(C3′) $\langle \varnothing, T, \varnothing, \langle \vdash ((\neg\varphi \to \neg\psi) \to (\psi \to \varphi))\rangle\rangle$

(C4′) $\langle \varnothing, T, \varnothing, \langle \vdash (\forall x(\forall x\, \varphi \to \psi) \to (\forall x\, \varphi \to \forall x\, \psi))\rangle\rangle$

(C5′) $\langle \varnothing, T, \varnothing, \langle \vdash (\forall x\, \varphi \to \varphi) \rangle \rangle$

(C6′) $\langle \varnothing, T, \varnothing, \langle \vdash (\forall x \forall y\, \varphi \to \forall y \forall x\, \varphi) \rangle \rangle$

(C7′) $\langle \varnothing, T, \varnothing, \langle \vdash (\neg\varphi \to \forall x \neg \forall x\, \varphi) \rangle \rangle$

(C8′) $\langle \varnothing, T, \varnothing, \langle \vdash (x = y \to (x = z \to y = z)) \rangle \rangle$

(C9′) $\langle \varnothing, T, \varnothing, \langle \vdash (\neg \forall x\, x = y \to (\neg \forall x\, x = z \to (y = z \to \forall x\, y = z))) \rangle \rangle$

(C10′) $\langle \varnothing, T, \varnothing, \langle \vdash (\forall x (x = y \to \forall x\, \varphi) \to \varphi)) \rangle \rangle$

(C11′) $\langle \varnothing, T, \varnothing, \langle \vdash (\forall x\, x = y \to (\forall x\, \varphi \to \forall y\, \varphi)) \rangle \rangle$

(C12′) $\langle \varnothing, T, \varnothing, \langle \vdash (x = y \to (Rxz \to Ryz)) \rangle \rangle$

(C13′) $\langle \varnothing, T, \varnothing, \langle \vdash (x = y \to (Rzx \to Rzy)) \rangle \rangle$

(C15′) $\langle \varnothing, T, \varnothing, \langle \vdash (\neg \forall x\, x = y \to (x = y \to (\varphi \to \forall x (x = y \to \varphi)))) \rangle \rangle$

(C16′) $\langle \{\{x, y\}\}, T, \varnothing, \langle \vdash (\forall x\, x = y \to (\varphi \to \forall x\, \varphi)) \rangle \rangle$

(C5) $\langle \{\{x, \varphi\}\}, T, \varnothing, \langle \vdash (\varphi \to \forall x\, \varphi) \rangle \rangle$

(MP) $\langle \varnothing, T, \{\langle \vdash (\varphi \to \psi) \rangle, \langle \vdash \varphi \rangle\}, \langle \vdash \psi \rangle \rangle$

(Gen) $\langle \varnothing, T, \{\langle \vdash \varphi \rangle\}, \langle \vdash \forall x\, \varphi \rangle \rangle$

Es ist zwar bekannt, dass diese Axiome „metalogisch vollständig“ sind, aber es ist nicht bekannt, ob sie im metalogischen Sinne unabhängig sind (d.h. keines ist redundant); insbesondere, ob irgendein Axiom (möglicherweise mit zusätzlichen optionalen disjunkte Variableneinschränkungen zur Verwendung von beliebigen Dummy-Variablen in seinem Beweis) aus den anderen beweisbar ist. Beachten Sie, dass metalogische Unabhängigkeit eine schwächere Anforderung ist als Unabhängigkeit im üblichen logischen Sinne. Nicht alle der oben genannten Axiome sind logisch unabhängig: beispielsweise kann C9′ als Metatheorem aus den anderen bewiesen werden, und zwar außerhalb des formalen Systems, indem die möglichen Fälle von unterscheidbaren Variablen kombiniert werden.

C.3.6 Beispiel 4 — Hinzufügen von Definitionen

Es gibt mehrere Möglichkeiten, einem formalen System Definitionen hinzuzufügen. Der wahrscheinlich beste Weg ist, Definitionen überhaupt nicht als Teil des formalen Systems zu betrachten, sondern als Abkürzungen, die Teil der erklärenden Metalogik außerhalb des formalen Systems sind. Der Einfachheit halber können wir jedoch das formale System selbst verwenden, um Definitionen einzubeziehen, indem wir sie als axiomatische Erweiterungen zum System hinzufügen. Dies könnte durch das Hinzufügen einer Konstante geschehen, die den Begriff „ist definiert als“ zusammen mit Axiomen für diesen Begriff repräsentiert. Aber es gibt einen schöneren Weg, zumindest meiner Meinung nach, der Definitionen als direkte Erweiterungen der Sprache und nicht als extralogische primitive Begriffe einführt. Wir führen zusätzliche logische Junktoren ein und stellen Axiome für sie bereit. Für

Logiksysteme wie die Beispiele 1 bis 3 müssen die zusätzlichen Axiome in dem Sinne konservativ sein, dass keine wff des ursprünglichen Systems, das kein Theorem war (wenn der ursprüngliche Begriff „wff“ natürlich durch „$\vdash$“ ersetzt wird), zu einem Theorem des erweiterten Systems wird. In diesem Beispiel erweitern wir Beispiel 3 (oder 2) mit Standardabkürzungen der Logik.

Wir erweitern *CN* aus Beispiel 3 um neue Konstanten $\{\leftrightarrow, \wedge, \vee, \exists\}$, die der logischen Äquivalenz, Konjunktion, Disjunktion und dem Existenzquantor entsprechen. Wir erweitern Γ um die axiomatischen Ausagen, die die Redukte der folgenden Prä-Aussagen sind:

$$\langle \varnothing, T, \varnothing, \langle \text{wff } (\varphi \leftrightarrow \psi) \rangle \rangle$$
$$\langle \varnothing, T, \varnothing, \langle \text{wff } (\varphi \vee \psi) \rangle \rangle$$
$$\langle \varnothing, T, \varnothing, \langle \text{wff } (\varphi \wedge \psi) \rangle \rangle$$
$$\langle \varnothing, T, \varnothing, \langle \text{wff } \exists x\, \varphi \rangle \rangle$$
$$\langle \varnothing, T, \varnothing, \langle \vdash ((\varphi \leftrightarrow \psi) \rightarrow (\varphi \rightarrow \psi)) \rangle \rangle$$
$$\langle \varnothing, T, \varnothing, \langle \vdash ((\varphi \leftrightarrow \psi) \rightarrow (\psi \rightarrow \varphi)) \rangle \rangle$$
$$\langle \varnothing, T, \varnothing, \langle \vdash ((\varphi \rightarrow \psi) \rightarrow ((\psi \rightarrow \varphi) \rightarrow (\varphi \leftrightarrow \psi))) \rangle \rangle$$
$$\langle \varnothing, T, \varnothing, \langle \vdash ((\varphi \wedge \psi) \leftrightarrow \neg(\varphi \rightarrow \neg\psi)) \rangle \rangle$$
$$\langle \varnothing, T, \varnothing, \langle \vdash ((\varphi \vee \psi) \leftrightarrow (\neg\varphi \rightarrow \psi)) \rangle \rangle$$
$$\langle \varnothing, T, \varnothing, \langle \vdash (\exists x\, \varphi \leftrightarrow \neg\forall x \neg\varphi) \rangle \rangle$$

Die ersten drei logischen Axiome (Aussagen, die „$\vdash$“ enthalten) führen die logische Äquivalenz, „$\leftrightarrow$“, ein und definieren sie effektiv. Die letzten drei verwenden „$\leftrightarrow$“ effektiv in der bedeutung von „ist definiert als“.

C.3.7 Beispiel 5 — ZFC Mengenlehre

Hier fügen wir dem System in Beispiel 4 die Axiome der Zermelo-Fraenkel-Mengenlehre mit Auswahlaxionm hinzu. Der Einfachheit halber verwenden wir die Definitionen aus Beispiel 4.

In der *CN* aus Beispiel 4 (die Beispiel 3 erweitert), ersetzen wir das Symbol R durch das Symbol $\in$. Genauer gesagt, wir entfernen aus Γ aus Beispiel 4 die drei axiomatischen Aussagegen, die R enthalten, und ersetzen sie durch die Redukte der folgenden:

$$\langle \varnothing, T, \varnothing, \langle \text{wff } x \in y \rangle \rangle$$
$$\langle \varnothing, T, \varnothing, \langle \vdash (x = y \rightarrow (x \in z \rightarrow y \in z)) \rangle \rangle$$
$$\langle \varnothing, T, \varnothing, \langle \vdash (x = y \rightarrow (z \in x \rightarrow z \in y)) \rangle \rangle$$

Unter der Annahme, dass $D = \{\{\alpha, \beta\} \in DV | \alpha, \beta \in Vr\}$ (mit anderen Worten müssen alle einzelnen Variablen verschieden sein), erweitern wir Γ

um die ZFC-Axiome, genannt Extensionalität, Ersetzung, Vereinigung, Potenzmenge, Regelmäßigkeit, Unendlichkeit und dem Auswahlaxiom, die die Redukte der folgenden Prä-Aussagen sind:

Ext $\langle D, T, \varnothing, \langle \vdash (\forall x(x \in y \leftrightarrow x \in z) \to y = z)\rangle\rangle$

Rep $\langle D, T, \varnothing, \langle \vdash \exists x(\exists y \forall z(\varphi \to z = y) \to \forall z(z \in x \leftrightarrow \exists x(x \in y \wedge \forall y\, \varphi)))\rangle\rangle$

Un $\langle D, T, \varnothing, \langle \vdash \exists x \forall y(\exists x(y \in x \wedge x \in z) \to y \in x)\rangle\rangle$

Pow $\langle D, T, \varnothing, \langle \vdash \exists x \forall y(\forall x(x \in y \to x \in z) \to y \in x)\rangle\rangle$

Reg $\langle D, T, \varnothing, \langle \vdash (x \in y \to \exists x(x \in y \wedge \forall z(z \in x \to \neg z \in y)))\rangle\rangle$

Inf $\langle D, T, \varnothing, \langle \vdash \exists x(y \in x \wedge \forall y(y \in x \to \exists z(y \in z \wedge z \in x)))\rangle\rangle$

AC $\langle D, T, \varnothing, \langle \vdash \exists x \forall y \forall z((y \in z \wedge z \in w) \to \exists w \forall y(\exists w((y \in z \wedge z \in w) \wedge (y \in w \wedge w \in x)) \leftrightarrow y = w))\rangle\rangle$

C.3.8 Beispiel 6 — Begriff der Klasse in der Mengenlehre

Ein leistungsfähiges Hilfsmittel, das die Mengenlehre vereinfacht (und das wir die ganze Zeit in unserer informellen Beschreibungssprache verwendet haben), ist die Notation der *Klassenabstraktion*. Die von uns eingeführten Definitionen werden von Takeuti und Zaring [67] oder Quine [55] rigoros als konservativ nachgewiesen. Die Schlüsselidee ist die Einführung der Notation $\{x|—\}$ für Abstraktionsklassen, was „die Klasse aller x, so dass —“ bedeutet, und die Einführung von (Meta-)Variablen, die sich über sie erstrecken. Eine Abstraktionsklasse kann eine Menge sein oder auch nicht, je nachdem, ob sie (als Menge) existiert. Eine Klasse, die nicht (als Menge) existiert, nennt man eine *echte Klasse*.

Zur Veranschaulichung der Verwendung von Abstraktionsklassen geben wir einige Beispiele für Definitionen, die von ihnen Gebrauch machen: die leere Menge, die Klassenvereinigung und das ungeordnete Paar. Viele weitere derartige Definitionen finden sich in der Metamath-Datenbasis für Mengenlehre, `set.mm`.

Wir erweitern *CN* aus Beispiel 5 um neue Symbole { class, {, |, }, $\varnothing$, $\cup$, , }, wobei die inneren Klammern und das letzte Komma konstante Symbole sind. (Wie zuvor sollte unsere doppelte Verwendung einiger mathematischer Symbole sowohl für unsere Beschreibungssprache als auch als Primitive des formalen Systems aus dem Kontext heraus klar sein).

Wir erweitern *VR* aus Beispiel 5 mit einer Menge von *Klassenvariablen* $\{A, B, C, \ldots\}$. Wir erweitern das T aus Beispiel 5 mit $\{\langle \text{class } A\rangle, \langle \text{class } B\rangle, \langle \text{class } C\rangle, \ldots\}$.

Um unsere Definitionen einzuführen, fügen wir zu Γ aus Beispiel 5 die axiomatischen Aussagen hinzu, die die Redukte der folgenden Prä-Aussagen sind:

$\langle \varnothing, T, \varnothing, \langle \text{class } x\rangle\rangle$

$\langle \varnothing, T, \varnothing, \langle \text{class } \{x|\varphi\}\rangle\rangle$

$\langle \varnothing, T, \varnothing, \langle \text{wff } A = B\rangle\rangle$

$\langle \varnothing, T, \varnothing, \langle \text{wff } A \in B\rangle\rangle$

Ab $\langle \varnothing, T, \varnothing, \langle \vdash (y \in \{x|\varphi\} \leftrightarrow ((x = y \to \varphi) \wedge \exists x(x = y \wedge \varphi)))\rangle\rangle$

Eq $\langle \{\{x, A\}, \{x, B\}\}, T, \varnothing, \langle \vdash (A = B \leftrightarrow \forall x(x \in A \leftrightarrow x \in B))\rangle\rangle$

El $\langle \{\{x, A\}, \{x, B\}\}, T, \varnothing, \langle \vdash (A \in B \leftrightarrow \exists x(x = A \wedge x \in B))\rangle\rangle$

Hier sagen wir, dass eine individuelle Variable eine Klasse ist; $\{x|\varphi\}$ ist eine Klasse; und wir erweitern die Definition einer wff, um Klassengleichheit und -zugehörigkeit einzuschließen. Axiom Ab definiert die Zugehörigkeit einer Variablen zu einer Klassenabstraktion; die rechte Seite kann gelesen werden als „die wff, die sich aus der echten Substitution von y für x in φ ergibt.“[7] Die Axiome Eq und El erweitern die Bedeutung des bestehenden Gleichheitszeichens und des Elementprädikats. Dies ist potenziell gefährlich und muss sorgfältig begründet werden. Zum Beispiel können wir aus Eq das Extensionalitätsaxiom allein mit Prädikatenlogik ableiten; daher sollten wir das Extensionalitätsaxiom im Prinzip als logische Hypothese aufnehmen. Wir machen uns jedoch nicht die Mühe, dies zu tun, da wir dieses Axiom bereits vorher vorausgesetzt haben. Die disjunkte Variableneinschränkungen sollten verstanden werden als: „wobei x nicht in A oder B vorkommt.“ Wir tun dies typischerweise, wenn die rechte Seite einer Definition eine individuelle Variable beinhaltet, die nicht in dem zu definierenden Ausdruck vorkommt; dies geschieht, damit die rechte Seite unabhängig von der speziellen „Dummy'-Variable bleibt, die wir verwenden.

Wir fügen Γ weiterhin die folgenden Definitionen (d.h. die Reduktionen der folgenden Prä-Aussagen) für die leere Menge,, die Klassenvereinigung, und das ungeordnete Paar hinzu. Sie sollten selbsterklärend sein. Analog zu unserer Verwendung von „$\leftrightarrow$“ zur Definition neuer wffs in Beispiel 4, verwenden wir „$=$“ zur Definition neuer Abstraktionsbegriffe, und beide können in diesem Zusammenhang informell als „ist definiert als“ gelesen werden.

$\langle \varnothing, T, \varnothing, \langle \text{class } \varnothing\rangle\rangle$

$\langle \varnothing, T, \varnothing, \langle \vdash \varnothing = \{x|\neg x = x\}\rangle\rangle$

$\langle \varnothing, T, \varnothing, \langle \text{class } (A \cup B)\rangle\rangle$

$\langle \{\{x, A\}, \{x, B\}\}, T, \varnothing, \langle \vdash (A \cup B) = \{x|(x \in A \vee x \in B)\}\rangle\rangle$

$\langle \varnothing, T, \varnothing, \langle \text{class } \{A, B\}\rangle\rangle$

$\langle \{\{x, A\}, \{x, B\}\}, T, \varnothing, \langle \vdash \{A, B\} = \{x|(x = A \vee x = B)\}\rangle\rangle$

[7] Anmerkung: Diese Definition macht die Einführung einer separaten Notation ähnlich $\varphi(x|y)$ für die echte Substitution überflüssig, obwohl wir dies aus Gründen der Konvention tun könnten. Übrigens wäre $\varphi(x|y)$ in seiner jetzigen Form in den formalen Systemen unserer Beispiele mehrdeutig, da wir nicht wissen würden, ob $\neg(\varphi(x|y)$ entweder $\neg(\varphi(x|y))$ oder $(\neg\varphi)(x|y)$ bedeutet. Stattdessen müssten wir eine eindeutige Variante wie $(\varphi\, x|y)$ verwenden.

C.4 Metamath als formales System

Dieser Abschnitt setzt die Kenntnis der Computersprache Metamath voraus.

Unsere Theorie beschreibt formale Systeme und ihre Universen. Die Metamath-Sprache bietet eine Möglichkeit, diese mengentheoretischen Objekte auf einem Computer darzustellen. Eine Metamath-Datenbasis, die aus einer endlichen Menge von ASCII-Zeichen besteht, kann in der Regel nur eine Teilmenge eines formalen Systems und seines Universums, die normalerweise unendlich sind, beschreiben. Allerdings kann die Datenbasis eine beliebig große endliche Teilmenge des formalen Systems und seines Universums enthalten. (Natürlich kann eine Metamath-Mengenlehre-Datenbasis im Prinzip indirekt ein ganzes unendliches formales System beschreiben, indem sie die Beschreibungssprache in diesem Anhang formalisiert).

Für unsere Diskussion gehen wir davon aus, dass die Metamath-Datenbasis die auf S. 148 beschriebene einfache Form hat, die aus allen Konstanten- und Variablendeklarationen am Anfang besteht, gefolgt von einer Folge erweiterter Frames, die jeweils durch `${` und `$}` begrenzt sind. Jede Metamath-Datenbasis kann in diese Form konvertiert werden, wie auf S. 151 beschrieben.

Die mathematischen Symbol-Token einer Metamath-Quelldatei, die mit den Anweisungen `$c` und `$v` deklariert werden, sind Namen, die wir den Repräsentanten von *CN* und *VR* zuweisen. Der Eindeutigkeit halber könnten wir annehmen, dass das erste mathematische Symbol, das als Variable deklariert wird, v_0 entspricht, das zweite v_1 usw., obwohl die gewählte Zuordnung nicht von bedeutung ist.

In der Metamath-Sprache entspricht jede `$d`-, `$f`- und `$e`-Quellanweisung in einem erweiterten Rahmen (Abschnitt 4.2.7) jeweils einem Element der Sammlungen D, T und H in einer Aussage $\langle D_M, T_M, H, A\rangle$ des formalen Systems. Die auf diese Metamath-Schlüsselwörter folgenden Zeichenketten mit mathematischen Symbolen entsprechen einem Variablenpaar (im Fall von `$d`) oder einem Ausdruck (für die beiden anderen Schlüsselwörter). Die mathematische Symbolkette nach einer `$a`-Quellanweisung entspricht dem Ausdruck A in einer axiomatischen Aussage des formalen Systems; die nach einer `$p`-Quellanweisung entspricht A in einer beweisbaren Aussage, die nicht axiomatisch ist. Mit anderen Worten: Jeder erweiterte Rahmen in einer Metamath-Datenbasis entspricht einer Prä-Aussage des formalen Systems, und ein Rahmen entspricht einer Aussage des formalen Systems[8].

Damit der Computer überprüfen kann, ob eine Aussage des formalen Systems beweisbar ist, wird jede entsprechende Anweisung von einem Beweis begleitet. Der Beweis hat jedoch keine Entsprechung im formalen System, sondern ist lediglich eine Möglichkeit, dem Computer die für seine Verifika-

[8] Anm. der Übersetzer: Im englischen Originaltext folgt hier ein Hinweis auf die doppelte Bedeutung des Wortes „statement“, was wir durch die unterschiedliche Übersetzung („Aussage“ im formalen System, „Anweisung“ in der Metamath-Datenbasis) vermieden haben.

tion benötigten Informationen mitzuteilen. Der Beweis sagt dem Computer, *wie* er bestimmte Glieder des Abschlusses der Prä-Aussage des formalen Systems konstruieren soll, die dem erweiterten Rahmen der `$p`-Anweisung entspricht. Das Endergebnis der Konstruktion ist das Element des Abschlusses, das mit der Anweisung `$p` übereinstimmt. Das abstrakte formale System hingegen befasst sich nur mit der *Existenz* von Elementen des Abschlusses.

Wie auf S. 220 erwähnt, entsprechen die Beispiele 1 und 3–6 im vorigen Abschnitt der Entwicklung der Logik und Mengenlehre in der Metamath-Datenbasis `set.mm`. Es ist sicherlich aufschlussreich, sie zu vergleichen.

Anhang D

Das MIU-System

Es folgt eine (übersetzte[1]) Auflistung der Datei `miu.mm`. Sie ist selbsterklärend.

```
$( Das MIU-System: Ein einfaches formales System $)

$( Hinweis: Dieses formale System ist insofern ungewöhnlich,
als es leere wffs zulässt.  Um mit einem Beweis zu arbeiten,
müssen Sie SET EMPTY_SUBSTITUTION ON eingeben, bevor Sie den
Befehl PROVE verwenden. Standardwert ist OFF, um die Anzahl
der mehrdeutigen Vereinheitlichungsmöglichkeiten zu reduzieren,
die während der Konstruktion eines Beweises bereitgestellt
werden müssen.  $)

$( Hofstadters MIU-System ist ein einfaches Beispiel für ein
formales System, das einige Konzepte von Metamath illustriert.
Siehe Douglas R. Hofstadter, _Goedel, Escher, Bach: An Eternal
Golden Braid_ (Vintage Books, New York, 1979), S. 33ff. für
eine Beschreibung des MIU-Systems.

Das System hat 3 konstante Symbole, M, I, und U. Das einzige
Axiom des Systems ist MI. Es gibt 4 Regeln:
      Regel I:  Wenn Sie eine Zeichenkette besitzen, deren
      letzter Buchstabe I ist, können Sie am Schluß ein U
      zufügen.
      Regel II: Angenommen Sie haben Mx.  Dann können Sie
      Ihrer Sammlung Mxx zufügen.
      Regel III: Wenn in einer der Zeichenketten Ihrer
```

[1] Anm. der Übersetzer: Übersetzung angelehnt an die deutschen Übersetzung von *Gödel, Escher, Bach: ein Endloses Geflochtenes Band* (Klett-Cotta, Stuttgart, 1986), S. 37ff.

```
      Sammlung III vorkommt, können Sie eine neue Kette mit
      U anstelle von III bilden.
      Regel IV: Wenn UU in einer Ihrer Ketten vorkommt,
      kann man es streichen.
Leider haben die Regeln III und IV keine eindeutigen
Ergebnisse: (Zeichen-)Ketten können mehr als ein Vorkommen
von III oder UU haben. Daher müssen wir das Konzept der
"wohlgeformten MIU-Formel" oder wff einführen, das es uns
ermöglicht, eindeutige Symbolfolgen zu konstruieren, auf
die die Regeln III und IV angewendet werden können. $)

$( Zuerst deklarieren wir die konstanten Symbole der Sprache.
Man beachte, dass wir zwei Symbole brauchen, um die Behauptung,
dass eine Folge eine wff ist, von der Behauptung, dass sie ein
Theorem ist, zu unterscheiden; wir haben willkürlich "wff" und
"|-" gewählt. $)
 $c M I U |- wff $. $( Konstanten deklarieren $)

$( Als nächstes deklarieren wir einige Variablen. $)
 $v x y $.

$( In unserer gesamten Theorie gehen wir davon aus, dass diese
Variablen wffs darstellen. $)
 wx   $f wff x $.
 wy   $f wff y $.

$( Definition von MIU-wffs. Wir erlauben, dass die leere Folge
eine wff ist. $)

$( Die leere Folge ist eine wff. $)
 we $a wff $.
$( "M" nach einer beliebigen wff ist eine wff. $)
 wM $a wff x M $.
$( "I" nach einer beliebigen wff ist eine wff. $)
 wI $a wff x I $.
$( "U" nach einer beliebigen wff ist eine wff. $)
 wU $a wff x U $.

$( Festlegung des Axioms. $)
 ax   $a |- M I $.

$( Festlegung der Regeln. $)
 ${
   Ia   $e |- x I $.
```

```
$( Ein beliebiger Satz, der mit "I" endet, bleibt auch dann ein
Satz, wenn ein "U" angehängt wird.  (Wir unterscheiden das Label
I_ vom mathematischen Symbol I, um der Metamath-Spezifikation
vom 24. Juni 2006 zu entsprechen.) $)
   I_    $a |- x I U $.
 $}
 ${
IIa  $e |- M x $.
$( Jeder Satz, der mit "M" beginnt, bleibt ein Satz, wenn der
Teil nach dem "M" nochmals hinzugefügt wird. $)
   II   $a |- M x x $.
 $}
 ${
   IIIa $e |- x I I I y $.
$( Jeder Satz mit "III" in der Mitte bleibt ein Satz, wenn das
 "III" durch "U" ersetzt wird. $)
   III  $a |- x U y $.
 $}
 ${
   IVa  $e |- x U U y $.
$( Ein beliebiger Satz mit "UU" in der Mitte bleibt ein Satz,
wenn das "UU" gelöscht wird. $)
   IV   $a |- x y $.
  $}

$( Nun beweisen wir das Theorem MUIIU.  Vielleicht ist es
für Sie interessant, diesen Beweis mit dem von Hofstadter
(S. 35 - 36 bzw. S. 40 in der deutschen Ausgabe) zu
vergleichen. $)
 theorem1  $p |- M U I I U $=
      we wM wU wI we wI wU we wU wI wU we wM we wI wU we wM
      wI wI wI we wI wI we wI ax II II I_ III II IV $.
```

Der Befehl `show proof /lemmon/renumber` erzeugt die folgende Anzeige. Sie ist derjenigen in [26, S. 35–36][2] sehr ähnlich.

```
1 ax             $a |- M I
2 1 II           $a |- M I I
3 2 II           $a |- M I I I I
4 3 I_           $a |- M I I I I U
5 4 III          $a |- M U I U
6 5 II           $a |- M U I U U I U
7 6 IV           $a |- M U I I U
```

[2] Anm. der Übersetzer: S. 40 in der deutschen Ausgabe

Wir stellen fest, dass Hofstadters „MU-Rätsel“, indem es um die Frage geht, ob MU ein Satz des MIU-Systems ist, nicht mit dem obigen System beantwortet werden kann, weil das MU-Rätsel eine Frage *über* das System ist. Um die Antwort auf das MU-Rätsel zu beweisen, ist ein viel ausgefeilteres System erforderlich, nämlich eines, das das MIU-System innerhalb der Mengenlehre modelliert. (Die Antwort auf das MU-Rätsel ist übrigens nein.)

Anhang E

Metamath-Sprache EBNF

Dieser Anhang enthält eine formale Beschreibung der grundlegenden Syntax der Metamath-Sprache (mit komprimierten Beweisen und Unterstützung für unbekannte Beweisschritte). Sie ist definiert unter Verwendung der Erweiterten Backus–Naur-Form (EBNF), eine Notation, so wie sie in W3C *Extensible Markup Language (XML) 1.0 (Fifth Edition)* (W3C Recommendation 26 November 2008) unter `https://www.w3.org/TR/xml/#sec-notation` beschrieben und verwendet wird.

Die Regel `database` wird bis zum Ende der Datei (`EOF`) verarbeitet. Die Regeln erfordern schließlich das Lesen von Token, die durch ein Whitespace getrennt sind. Ein Token hat eine Großbuchstaben-Definition (siehe unten) oder ist eine String-Konstante in einem Nicht-Token (wie `'$a'`). Wir hoffen, dass dies korrekt ist, aber wenn es einen Konflikt gibt, gelten die Regeln des Abschnitts 4.1. In diesem Abschnitt werden auch nicht-syntaktische Einschränkungen erörtert, die hier nicht gezeigt werden (z. B. dass jedes neue Label-Token, das in einem `hypothesis-stmt` oder `assert-stmt` definiert wird, eindeutig sein muss).

```
database ::= outermost-scope-stmt*

outermost-scope-stmt ::=
  include-stmt | constant-stmt | stmt

/* Anweisung zum Einbinden von Dateien; behandelt eine Datei als
   (einen Teil einer) Datenbasis. Innerhalb des Dateinamens darf
   sich KEIN Kommentar befinden. */
include-stmt ::= '$[' filename '$]'

/* Deklaration von Symbolen für Konstanten. */
constant-stmt ::= '$c' constant+ '$.'
```

```
/* Eine normale Anweisung kann sich in jedem Scope befinden. */
stmt ::= block | variable-stmt | disjoint-stmt |
  hypothesis-stmt | assert-stmt

/* Ein Block, der auch leer sein kann. */
block ::= '${' stmt* '$}'

/* Deklaration von Symbolen für Variablen. */
variable-stmt ::= '$v' variable+ '$.'

/* Disjunkte Variablen. Einfache disjunkte Variableneinschränkung
   bestehen aus 2 Variables, d.h. "variable*" ist in diesem Fall
   leer. */
disjoint-stmt ::= '$d' variable variable variable* '$.'

hypothesis-stmt ::= floating-stmt | essential-stmt

/* Fließende (Variablentyp-)Hypothese. */
floating-stmt ::= LABEL '$f' typecode variable '$.'

/* Essenzielle (logische) Hypothese. */
essential-stmt ::= LABEL '$e' typecode MATH-SYMBOL* '$.'

assert-stmt ::= axiom-stmt | provable-stmt

/* Axiomatische Behauptung. */
axiom-stmt ::= LABEL '$a' typecode MATH-SYMBOL* '$.'

/* Beweisbare Behauptung. */
provable-stmt ::= LABEL '$p' typecode MATH-SYMBOL*
  '$=' proof '$.'

/* Ein Beweis. Innerhalb von Beweisen können sich auch
   Kommentare befinden. Wenn ein '?' in dem Beweis
   enthalten ist, dann handelt es sich um einen
   "unvollständigen" Beweis. */
proof ::= uncompressed-proof | compressed-proof
uncompressed-proof ::= (LABEL | '?')+
compressed-proof ::= '(' LABEL* ')' COMPRESSED-PROOF-BLOCK+

typecode ::= constant

filename ::= MATH-SYMBOL /* enthält kein Whitespace oder '$' */
constant ::= MATH-SYMBOL
```

```
variable ::= MATH-SYMBOL
```

Ein `frame` ist eine Folge von keinem, einem oder mehreren `disjoint-stmt`- und `hypotheses-stmt`-Anweisungen (möglicherweise verschachtelt mit anderen Anweisungen, die keine `assert-stmt`-Anweisungen sind), gefolgt von einem `assert-stmt`.

Hier sind die Regeln für die lexikalische Verarbeitung (Tokenisierung) über die oben gezeigten konstanten Token hinaus. Konventionell werden diese Tokenisierungsregeln in Großbuchstaben geschrieben. Jedes Token wird so lang wie möglich gelesen. Durch ein Whitespace getrennte Token werden nacheinander gelesen; beachten Sie, dass der trennende Whitespace und die Kommentare `$(` ... `$)` übersprungen werden.

Wenn eine Token-Definition eine andere Token-Definition verwendet, wird das Ganze als ein einziges Token betrachtet. Ein Muster, das nur Teil eines vollständigen Tokens ist, hat einen Namen, der mit einem Unterstrich („_“) beginnt. Eine Implementierung könnte viele Token als `PRINTABLE-SEQUENCE` tokenisieren und dann prüfen, ob sie die hier dargestellte spezifischere Regel erfüllen.

Kommentare lassen sich nicht verschachteln, und sowohl `$(` als auch `$)` müssen von mindestens einem Whitespace-Zeichen (`_WHITECHAR`) umgeben sein. Technisch gesehen enden Kommentare vor dem abschließenden `_WHITECHAR`, aber das abschließende `_WHITECHAR` wird sowieso ignoriert, so dass wir dieses Detail hier ignorieren. Metamath-Sprachprozessoren müssen kein `$)` unterstützen, das unmittelbar vor dem Dateiende steht, da auf das abschließende Kommentarsymbol ein `_WHITECHAR`, wie ein Zeilenumbruch, folgen muss.

```
PRINTABLE-SEQUENCE ::= _PRINTABLE-CHARACTER+

MATH-SYMBOL ::= (_PRINTABLE-CHARACTER - '$')+

/* druckbare ASCII-Zeichen ohne Whitespace-Zeichen */
_PRINTABLE-CHARACTER ::= [#x21-#x7e]

LABEL ::= ( _LETTER-OR-DIGIT | '.' | '-' | '_' )+

_LETTER-OR-DIGIT ::= [A-Za-z0-9]

COMPRESSED-PROOF-BLOCK ::= ([A-Z] | '?')+

/* Definition von Whitespace zwischen Token. Das -> SKIP
   bedeutet, dass ein auftretendes Whitespace übersprungen
   und mit dem Lesen weiterer Zeichen fortgefahren wird. */
WHITESPACE ::= (_WHITECHAR+ | _COMMENT) -> SKIP
```

```
/* Kommentare. $( ... $), die nicht verschachtelt sind. */
_COMMENT ::= '$(' (_WHITECHAR+ (PRINTABLE-SEQUENCE - '$)')*
  _WHITECHAR+ '$)' _WHITECHAR

/* Whitespace: (' ' | '\t' | '\r' | '\n' | '\f') */
_WHITECHAR ::= [#x20#x09#x0d#x0a#x0c]
```

Anhang F

Metamath 100

Die folgende Tabelle enthält alle Theoreme der Liste „Formalisierung von 100 Theoremen“ („Formalizing 100 Theorems“, siehe `http://www.cs.ru.nl/%7Efreek/100/`) von Freek Wiedijk, die bereits mit Metamath bewiesen wurden (Stand 12.11.2023). Die Nr. entspricht der Nummer in der Liste von Freek Wiedijk, die Bezeichnung wurde ins Deutsche übersetzt[1]

Nr.	Bezeichnung des Theorems	Label in set.mm	Autor	Datum
1.	Die Quadratwurzel aus 2 ist irrational	sqrt2irr	Norman Megill	2001-08-20
2.	Der Fundamentalsatz der Algebra	fta	Mario Carneiro	2014-09-15
3.	Die Abzählbarkeit der rationalen Zahlen	qnnen	Norman Megill	2004-07-31
4.	Der Satz von Pythagoras	pythi	Norman Megill	2008-04-17
5.	Der Primzahlensatz	pnt	Mario Carneiro	2016-06-01
7.	Das quadratische Reziprozitätsgesetz	lgsquad	Mario Carneiro	2015-06-19
9.	Die Kreisfläche	areacirc	Brendan Leahy	2017-08-31
10.	Der Satz von Euler (Verallgemeinerung des kleinen Fermatschen Satzes)	eulerth	Mario Carneiro	2014-02-28
11.	Der Satz von Euklid (Existenz unendlich vieler Primzahlen)	infpn2	Norman Megill	2005-05-05

[1] Anm. der Übersetzer: Dieser Anhang ist im Original nicht vorhanden

Nr.	Bezeichnung des Theorems	Label in set.mm	Autor	Datum
14.	Das Baseler Problem (Summe der reziproken Quadratzahlen)	basel	Mario Carneiro	2014-07-30
15.	Der Fundamentalsatz der Analysis	ftc1, ftc2	Mario Carneiro	2014-09-03
17.	Der Moivresche Satz	demoivreALT	Steve Rodriguez	2006-11-10
18.	Der Approximationssatz von Liouville	aaliou	Stefan O'Rear	2014-11-22
19.	Der Vier-Quadrate-Satz von Lagrange	4sq	Mario Carneiro	2014-07-16
20.	Der Zwei-Quadrate-Satz von Fermat	2sq	Mario Carneiro	2015-06-20
22.	Die Überabzählbarkeit des Kontinuums (der reelen Zahlen)	ruc	Norman Megill	2004-08-13
23.	Formel zur Bildung der pythagoreischen Tripel	pythagtrip	Scott Fenton	2014-04-19
25.	Der Cantor-Bernstein-Schrödersche Äquivalenzssatz	sbth	Norman Megill	1998-06-08
26.	Die Leibniz-Reihe für Pi	leibpi	Mario Carneiro	2015-04-16
27.	Der Innenwinkelsatz für Dreiecke	ang180	Mario Carneiro	2014-09-24
30.	Bertrand's Ballot Problem	ballotth	Thierry Arnoux	2016-12-07
31.	Der Satz von Ramsey (Kombinatorik)	ramsey	Mario Carneiro	2015-04-23
34.	Die Divergenz der harmonischen Reihe	harmonic	Mario Carneiro	2014-07-11
35.	Der Satz von Taylor	taylth	Mario Carneiro	2017-01-01
37.	Die Lösungsformel für kubische Gleichungen	cubic	Mario Carneiro	2015-04-26
38.	Die Ungleichung vom arithmetischen und geometrischen Mittel	amgm	Mario Carneiro	2015-06-21
39.	Lösungen der Pellschen Gleichung	rmxycomplete	Stefan O'Rear	2014-11-22
42.	Die Summe der Kehrwerte aller Dreieckszahlen	trirecip	Scott Fenton	2014-05-02

Nr.	Bezeichnung des Theorems	Label in set.mm	Autor	Datum
44.	Der binomische Lehrsatz	binom	Norman Megill	2005-12-07
45.	Der Partitionssatz von Euler (als Spezialfall des Satzes von Glaisher)	eulerpart	Thierry Arnoux	2018-08-30
46.	Die Lösungsformel für allgemeine quartische Gleichungen	quart	Mario Carneiro	2015-05-06
48.	Der Dirichletsche Primzahlsatz	dirith	Mario Carneiro	2016-05-12
49.	Der Satz von Cayley-Hamilton	cayleyhamilton	Alexander van der Vekens	2019-11-25
51.	Der Satz von Wilson	wilth	Mario Carneiro	2015-01-28
52.	Die Anzahl der Teilmengen von Mengen	pw2en	Norman Megill	2004-01-29
54.	Das Königsberger Brückenproblem	konigsberg	Mario Carneiro	2015-04-16
55.	Der Sehnensatz	chordthm	David Moews	2017-02-28
57.	Der Satz von Heron	heron	Mario Carneiro (Jon Pennant, Thierry Arnoux)	2019-03-10
58.	Die Formel für die Anzahl von Kombinationen	hashbc	Mario Carneiro	2014-07-13
60.	Das Lemma von Bézout	bezout	Mario Carneiro	2014-02-22
61.	Der Satz von Ceva	cevath	Saveliy Skresanov	2017-09-24
63.	Der Satz von Cantor	canth2	Norman Megill	1994-08-07
64.	Die Regel von de L'Hospital	lhop	Mario Carneiro	2016-12-30
65.	Der Basiswinkelsatz in gleichschenkligen Dreiecken	isosctr	Saveliy Skresanov	2017-01-01
66.	Die (endlichen) Partialsummen einer geometrischen Reihe	geoser	Norman Megill	2006-05-09

Nr.	Bezeichnung des Theorems	Label in set.mm	Autor	Datum
67.	Die Eulersche Zahl e ist transzendent	etransc	Glauco Siliprandi	2020-04-05
68.	Die Gaußsche Summenformel	arisum	Frédéric Liné	2006-11-16
69.	Der Euklidische Algorithmus	eucalg	Paul Chapman	2011-03-31
70.	Der Euklid-Euler-Satz über vollkommene Zahlen	perfect	Mario Carneiro	2016-05-17
71.	Der Satz von Lagrange	lagsubg, lagsubg2	Mario Carneiro	2014-07-11
72.	Die Sylow-Sätze	sylow1, sylow2, sylow2b, sylow3	Mario Carneiro	2015-01-19
73.	Der Satz von Erdős und Szekeres	erdsze, erdsze2	Mario Carneiro	2015-01-28
74.	Das Prinzip der vollständigen Induktion	finds	Norman Megill	1995-04-14
75.	Der Mittelwertsatz der Differentialrechnung	mvth	Mario Carneiro	2014-09-14
76.	Fourierreihen	fourier	Glauco Siliprandi	2019-12-11
77.	Die Faulhabersche Formel für Potenzsummen	fsumkthpow	Scott Fenton	2014-05-16
78.	Die Cauchy-Schwarz-Ungleichung	sii	Norman Megill	2008-01-12
79.	Der Zwischenwertsatz (der reellen Analysis)	ivth	Paul Chapman	2008-01-22
80.	Der Fundamentalsatz der Arithmetik	1arith2	Paul Chapman	2012-11-17
81.	Erdős Beweis der Divergenz der Reihe der Kehrwerte der Primzahlen (Satz von Euler)	prmrec	Mario Carneiro	2014-08-10
83.	Der Freundschaftssatz	friendship	Alexander van der Vekens	2018-10-09
85.	Quersummenregel für die Teilbarkeit durch 3	3dvds	Mario Carneiro	2014-07-14
86.	Lebesgue-Maß und -Integral	itgcl	Mario Carneiro	2014-06-29

Nr.	Bezeichnung des Theorems	Label in set.mm	Autor	Datum
87.	Der Satz von Desargues	dath	Norman Megill	2012-08-20
88.	Die Anzahl der fixpunktfreien Permutationen	derangfmla, subfaclim	Mario Carneiro	2015-01-28
89.	Der Restpolynom-Satz	facth, plyrem	Mario Carneiro	2014-07-26
90.	Die Stirling-Formel	stirling	Glauco Siliprandi	2017-06-29
91.	Die Dreiecksungleichung	abstrii	Norman Megill	1999-10-02
93.	Das Geburtstagsproblem	birthday	Mario Carneiro	2015-04-17
94.	Der Kosinussatz	lawcos	David A. Wheeler	2015-06-12
95.	Der Satz des Ptolemäus	ptolemy	David A. Wheeler	2015-05-31
96.	Das Prinzip von Inklusion und Exklusion	incexc	Mario Carneiro	2017-08-07
97.	Die Cramersche Regel	cramer	Alexander van der Vekens (Stefan O'Rear)	2019-02-21
98.	Das Bertrandsche Postulat	bpos	Mario Carneiro	2014-03-15

Anhang G

Glossar

G.1 Deutsch - Englisch

Deutsch	**Englisch**
Abschluss	closure
abstrakte Algebra	abstract algebra
Abstraktionsklasse	abstraction class
Addition	addition
aktive Anweisung	active statement
aktives mathematisches Symbol	active math symbol
Allquantor	universal quantifier
Analysis	analysis
Anweisung	statement
äußerster Block	outermost block
Ausdruck	expression
Aussage	statement
Aussagenlogik	propositional calculus
Aussonderungssaxiom	Axiom of Separation
Auswahlaxiom	Axiom of Choice
Auszeichnungsnotation	markup notation
automatisches Theorembeweisen	automated theorem proving
automatisierte Beweisverifizierung	automated proof verification
Axiom	axiom
axiomatische Aussagen	axiomatic statement
axiomatische Behauptung	axiomatic assertion
Axiome der Aussagenlogik	axioms of propositional calculus
Axiome der Logik	axioms of logic
Axiome der Mengenlehre	axioms of set theory
Axiome der Prädikatenlogik	axioms of predicate calculus

Deutsch	**Englisch**
Axiome für die Gleichheit	axioms for equality
Axiome für die Mathematik	axioms for mathematics
Axiomenschema	axiom scheme
Axiom von Pasch	Pasch's axiom
Baumdarstellung eines Beweises	tree-style proof
Befehlsschlüsselwort	command keyword
Befehlszeilenparameter	command qualifier
Befehlszeilenschnittstelle (CLI)	command line interface
Behauptung	assertion
Behauptungslabel	assertion label
Betriebssystem-Befehl	operating system command
Beweis	proof
Beweis-Assistent	Proof Assistant
beweisbare Aussage	provable statement
beweisbare Behauptung	provable assertion
Beweislänge	proof length
Beweisschema	proof scheme
Beweisschritt	proof step
Beweistheorie	proof theory
bijektive Funktion	one-to-one, onto function
Bikonditional	biconditional
Bild	image
binäre Relation	binary relation
Block	block
Boolesche Algebra	Boolean algebra
Bug	software bug
Burali-Forti Paradoxon	Burali-Forti paradox
Clifford-Algebren	Clifford algebras
Computeralgebrasystem	computer algebra system
Courier Schriftart	Courier font
Dateieinbindung	file inclusion
Dateiname	file name
Datenbasis	database
Deduktionsform	deduction form
Deduktionsstil	deduction style
Deduktionstheorem	deduction theorem
Definiendum	definiendum
Definiens	definiens
Definition	definition
Definitionsbereich	domain

Deutsch	**Englisch**
Deklaration	declaration
disjunkte Mengen	disjoint sets
disjunkte Variablen	disjoint variables
disjunkte Variableneinschränkung	disjoint-variable restriction
Disjunktion	disjunction
Drehkreuz	turnstile
druckbares Zeichen	printable character
Drucker	printer
Dummy-Variable	dummy variable
echte Klasse	proper class
echte Substitution	proper substitution
effektiv gebundene Variable	effectively bound variable
effektiv nicht frei	effectively not free
einelementige Menge	singleton
Eindeutigkeitsquantor	existential uniqueness quantifier
einfache Deklaration	simple declaration
einfache unendliche Folge	simple infinite sequence
einfacher Text	plain text
einfaches Anführungszeichen	grave accent
einfaches Metatheorem	simple metatheorem
eingeschränkter Allquantor	restricted universal quantifier
eingeschränkter Existenzquantor	restricted existential quantifier
eingebundene Datei	included file
Einschränkung	restriction
Element	element
endliche n-gliedrige Folge	finite n-termed sequence
entfernen	pop
entscheidbare Theorie	decidable theory
Entscheidungsverfahren	decision procedure
Epsilon-Relation	epsilon relation
Ersetzung	substitution
Ersetzungsaxiom	Axiom of Replacement
erweiterte Backus-Naur-Form	Extended Backus-Naur Form
erweiterte Sprache	extended language
erweiterter Frame	extended frame
essentielle Hypothese	essential hypothesis
euklidische Geometrie	Euclidean geometry
Existenzquantor	existential quantifier
Extensionalitätsaxiom	Axiom of Extensionality
Familie	family
Fehler in Beweisen	errors in proofs

Deutsch	**Englisch**
Fehlerprüfung	error checking
finite Induktion	finite induction
finitistischer Beweis	finitary proof
fließende Hypothese	floating hypothesis
formale Logik	formal logic
formaler Beweis	formal proof
formales System	formal system
Formalismus	formalism
Formen	forms
Frame	frame
freie Logik	free logic
freie Variable	free variable
fundierte Relation	founded relation
Fundierungsaxiom	Axiom of Regularity
Funktion	function
Funktionswert	function value
ganze Zahl	integer
gebundene Variable	bound variable
genau dann, wenn	iff
geordnetes Paar	ordered pair
geschlossene Form	closed form
Gleichheit	equality
Glied	member
globale Anweisung	global statement
Gödelscher Unvollständigkeitssatz	Gödel's incompleteness theorem
Grenzzahl	limit ordinal
Großer Fermatscher Satz	Fermat's Last Therorem
Grundlagen der Mathematik	foundations of mathematics
grundlegende Sprache	basic language
grundlegendes Schlüsselwort	basic keyword
Gruppentheorie	group theory
Gültigkeitsbereich	scope
Gültigkeitsbereichsanweisung	scoping statement
Hierarchie	hierarchy
Hilfsschlüsselwort	auxiliary keyword
Hypothese	hypothesis
Hypothesenlabel	hypothesis label
Hypothesenzuordnung	hypothesis association
Implikation	implication
implizites Axiom	implicit axiom

Deutsch	**Englisch**
implizite Substitution	implicit substitution
individuelle Metavariable	individual metavariable
individuelle Variable	individual variable
Inferenz	inference
Inferenzform	inference form
Inferenzregel	inference rule
Infix-Konnektor	inflix connective
informeller Beweis	informal proof
injektive Funktion	one-to-one function
Intuitionismus	intuitionism
Junktor	connective
Kardinalität	cardinality
Kardinalzahl	cardinal
Kartesisches Produkt	Cartesian product
Kategorientheorie	category theory
Klammerschreibweise	brace notation
Klasse	class
Klassenabstraktion	class abstraction
Klassendifferenz	class difference
Klassengleichheit	class equality
Klassenvariable	class variable
Klassenzugehörigkeit	class membership
Kommentar	comment
Kommentar mit zusätzlichen Informationen	additional information comment
kompakter Beweis	compact proof
komplexe Zahl	complex number
Komposition	composition
komprimierter Beweis	compressed proof
kondensierte Ablösung	condensed detachment
Konjunktion	conjunction
Konstante	constant
Konstantendeklaration	constant declaration
Konstante-Variable-Paar	constant-variable pair
konstant-gepräfixter Ausdruck	constant-prefixed expression
konstruktive Sprache	constructive language
Konstruktivismus	constructivism
kontinuierliche Integration	continuous integration
Kontinuumshypothese	continuum hypothesis
kreative Definintion	creative definition
Kreuzprodukt	cross product

Deutsch	**Englisch**
künstliche Intelligenz	artificial intelligence
Label	label
Label-Deklaration	label declaration
Label-Modus	label mode
Label-Referenz	label reference
Label-Sequenz	label sequence
leere Menge	empty set, null set
leerer Definitionsbereich	empty domain
leere Substitution	empty substitution
Leermengenaxiom	Axiom of the Null Set
Leerraum	white space
Lemmon-Stil Beweis	Lemmon-style proof
Logik	logic
Logik erster Ordnung	first-order logic (FOL)
Logik höherer Ordnung	higher-order logic (HOL)
logische Äquivalenz	logical equivalenz
logische Hypothese	logical hypothesis
logisches ODER	logical OR
logisches UND	logical AND
lokale Variable	local variable
lokales Label	local label
Lücken in Beweisen	gaps in proofs
Macintosh-Dateiname	Macintosh file name
maschinelles Lernen	machine learning
mathematisches Symbol	math symbol
Mathe-Modus	math mode
mehrdeutige Vereinheitlichung	ambiguous unification
Menge	set
Mengendifferenz	set difference
Mengenlehre	set theory
Metalogik	metalogic
metalogische Vollständigkeit	metalogical completeness
Metamath	Metamath
Metamath Proof Explorer	Metamath Proof Explorer
Metamathematik	metamathematics
Metamath-Sprache EBNF	Metamath Language EBNF
Metasprache	metalanguage
Metatheorem	metatheorem
Metavariable	metavariable
Mitglied	member
MIU-System	MIU-system

Deutsch	**Englisch**
modale Logik	modal logic
Modelltheorie	model theory
Modus ponens	modus ponens
Monaco Schriftart	Monaco font
Nachfolger	successor
natürliche Deduktion	natural deduction
natürliche Zahl	natural number
Negation	negation
nichtproportionale Schriftart	monospaced font
nicht-triviale Theorie	non-trivial theory
normaler Beweis	normal proof
Objekt	object
Objektsprache	object language
obligatorische `$d`-Anweisung	mandatory `$d` statement
obligatorische disjunkte Variableneinschränkung	mandatory distinct-/disjoint-variable restriction
obligatorische Hypothese	mandatory hypothesis
obligatorische Variable	mandatory variable
obligatorische Variablentyp-Hypothese	mandatory variable-type hypothesis
Omega	omega
Operation	operation
optionale disjunkte Variableneinschränkung	optional disjoint-variable restriction
optionale Hypothese	optional hypothesis
optionale Variable	optional variable
ordinale Addition	ordinal addition
Ordinalprädikat	ordinal predicate
Ordinalzahl	ordinal number
Paar	pair
Paarmengenaxiom	Axiom of Pairing
Parser	parser
Peano-Postulate	Peano's postulates
Pierces Axiom	Pierce's axiom
Poincaré-Vermutung	Poincaré conjecture
polnische Notation	Polish notation
Postfix-Konnektor	postfix connective
Potenzklasse	power class
Potenzmenge	power set
Potenzmengenaxiom	Axiom of Power Sets

Deutsch	**Englisch**
Prä-Aussage	pre-statement
Prädikatenlogik	predicate calculus
Präfix-Konnektor	prefix connective
Principia Mathematica	Principia Mathematica
Programmfehler	software bug
qualifizierender Ausdruck	qualifying expression
Quantenlogik	quantum logic
Quantenmechanik	quantum mechanics
Quantifizierungstheorie	quantifier theory
Quelldatei	source file
Quellpuffer	source buffer
Querverweis zu Literaturangaben	bibliographical references
rationale Zahl	rational number
Redukt	reduct
reelle Zahl	real number
Reflexionsprinzip	reflection principle
Regel	rule
Regel der Verallgemeinerung	rule of generalization
reine Mathematik	pure mathematics
Rekursionsoperator	recursion operator
rekursive Definition	recursive definition
Relation	relation
Robbins-Algebra	Robbins algebra
Robinsons Resolutionsprinzip	Robinson's resolution principle
RPN-Reihenfolge	RPN order
RPN-Stapel	RPN stack
Russells Paradoxon	Russell's paradox
Sammlung	collection
Satz von Cantor	Cantor's theorem
Satzlogik	sentential logic
schieben	push
Schlüsselwort	keyword
Schlussfolgerung	conclusion
Schnittmenge	intersection
Schriftsatzanweisung	typesetting comment
Schröder-Bernstein-Theorem	Schröder-Bernstein theorem
schwache Logik	weak logic
Sonderzeichen	special characters
Spinner	crank
Standard-Deduktionstheorem	Standard Deduction Theorem

Deutsch	**Englisch**
Stapel	stack
stilisiertes Epsilon	stylized epsilon
Substitution	substitution
Substitutionsabbildung	substitution map
Substitutionstheorem	substitution theorem
surjektive Funktion	onto function
Syllogismus	Syllogism
Symbol	symbol
Syntax-Regeln	syntax rules
Tautologie	tautology
Teilmenge	subset
temporäre Variable	temporary variable
Term	term
Texteditor	text editor
Textverarbeitung	word processing
Theorem	theorem
Theorem der schwachen Deduktion	Weak Deduction Theorem
Theoremschema	theorem scheme
Tilde	tilde
Token	token
Topologie	topology
transfinite Kardinalzahl	transfinite cardinal
transfinite Rekursion	transfinite recoursion
transitive Klasse	transitive class
transitive Menge	transitive set
Typ	type
Typcode	typecode
Umdeklarierung von Symbolen	redeclaration of symbols
umgekehrte polnische Notation	reverse polish notation (RPN)
unendliche Menge	infinite set
Unendlichkeit	infinity
Unendlichkeitsaxiom	Axiom of Infinity
ungeordnetes Paar	unordered pair
ungeordnetes Tripel	unordered tripel
universelle Klasse	universal class
Universum eines formalen Systems	universe of a formal system
Unix-Dateiname	Unix file name
Unterklasse	subclass
unterschiedliche Variablen	distinct variables
unzugängliche Kardinalzahl	inaccessible cardinal

Deutsch	**Englisch**
Variable	variable
Variablendeklaration	variable declaration
Variablensubstitution	variable substitution
Variablentyp	variable type
Variablentyp-Hypothese	variable-type hypothesis
Venn-Diagramm	Venn diagram
Vereinheitlichung	unification
Vereinigung	union
Vereinigungsaxiom	Axiom of Union
Vereinigungsmenge	set union
Verkettung	concatenation
verschachtelter Block	nested block
Verschachtelungstiefe	nesting level
verschiedene Variablen	distinct variables
Vier-Farben-Satz	four-color theorem
vollständige Induktion	mathematical induction
Vollständigkeitssatz der Aussagenlogik	completeness theorem of propositional calculus
Vorrangigkeit eines Operators	operator precedence
Wahrheitstabelle	truth table
Wertebereich	range
Whitespace	white space
widerspruchsfreie Theorie	consistent theory
wohlgeformte Formel (wff)	well-formed formula
Wohlordnung	well-ordering
Zahlentheorie	number theory
Zermelo-Fraenkel-Mengenlehre	Zermelo-Fraenkel set theory
ZFC-Mengenlehre	ZFC set theory
Zitat	citation
zulässige Definition	proper definition
Zuordnung	mapping
zusammengesetzte Deklaration	compound declaration

G.2 Englisch - Deutsch

Englisch	**Deutsch**
abstract algebra	abstrakte Algebra
abstraction class	Abstraktionsklasse
active math symbol	aktives mathematisches Symbol
active statement	aktive Anweisung
addition	Addition
additional information comment	Kommentar mit zusätzlichen Informationen
ambiguous unification	mehrdeutige Vereinheitlichung
artificial intelligence	künstliche Intelligenz
analysis	Analysis
assertion	Behauptung
assertion label	Behauptungslabel
automated proof verification	automatisierte Beweisverifizierung
automated theorem proving	automatisches Theorembeweisen
auxiliary keyword	Hilfsschlüsselwort
Axiom	axiom
Axiom of Choice	Auswahlaxiom
Axiom of Extensionality	Extensionalitätsaxiom
Axiom of Infinity	Unendlichkeitsaxiom
Axiom of Pairing	Paarmengenaxiom
Axiom of Power Sets	Potenzmengenaxiom
Axiom of Regularity	Fundierungsaxiom
Axiom of Replacement	Ersetzungsaxiom
Axiom of Separation	Aussonderungssaxiom
Axiom of the Null Set	Leermengenaxiom
Axiom of Union	Vereinigungsaxiom
axiom scheme	Axiomenschema
axiomatic assertion	axiomatische Behauptung
axiomatic statement	axiomatische Aussagen
axioms for equality	Axiome für die Gleichheit
axioms for mathematics	Axiome für die Mathematik
axioms of logic	Axiome der Logik
axioms of predicate calculus	Axiome der Prädikatenlogik
axioms of propositional calculus	Axiome der Aussagenlogik
axioms of set theory	Axiome der Mengenlehre
basic keyword	grundlegendes Schlüsselwort
basic language	grundlegende Sprache
bibliographical references	Querverweis zu Literaturangaben
biconditional	Bikonditional

Englisch	**Deutsch**
binary relation	binäre Relation
block	Block
Boolean algebra	Boolesche Algebra
bound variable	gebundene Variable
brace notation	Klammerschreibweise
Burali-Forti paradox	Burali-Forti Paradoxon
Cantor's theorem	Satz von Cantor
cardinal	Kardinalzahl
cardinality	Kardinalität
Cartesian product	Kartesisches Produkt
category theory	Kategorientheorie
citation	Zitat
class	Klasse
class abstraction	Klassenabstraktion
class difference	Klassendifferenz
class equality	Klassengleichheit
class membership	Klassenzugehörigkeit
class variable	Klassenvariable
Clifford algebras	Clifford-Algebren
closed form	geschlossene Form
closure	Abschluss
collection	Sammlung
command keyword	Befehlsschlüsselwort
command line interface	Befehlszeilenschnittstelle (CLI)
command qualifier	Befehlszeilenparameter
comment	Kommentar
compact proof	kompakter Beweis
completeness theorem of propositional calculus	Vollständigkeitssatz der Aussagenlogik
complex number	komplexe Zahl
composition	Komposition
compound declaration	zusammengesetzte Deklaration
compressed proof	komprimierter Beweis
computer algebra system	Computeralgebrasystem
concatenation	Verkettung
conclusion	Schlussfolgerung
condensed detachment	kondensierte Ablösung
conjunction	Konjunktion
connective	Junktor
consistent theory	widerspruchsfreie Theorie
constant	Konstante
constant declaration	Konstantendeklaration

Englisch	**Deutsch**
constant-prefixed expression	konstant-gepräfixter Ausdruck
constant-variable pair	Konstante-Variable-Paar
constructive language	konstruktive Sprache
constructivism	Konstruktivismus
continuous integration	kontinuierliche Integration
continuum hypothesis	Kontinuumshypothese
Courier font	Courier Schriftart
crank	Spinner
creative definition	kreative Definintion
cross product	Kreuzprodukt
database	Datenbasis
decidable theory	entscheidbare Theorie
decision procedure	Entscheidungsverfahren
declaration	Deklaration
deduction form	Deduktionsform
deduction style	Deduktionsstil
deduction theorem	Deduktionstheorem
definiendum	Definiendum
definiens	Definiens
definition	Definition
disjoint sets	disjunkte Mengen
disjoint variables	disjunkte Variablen
disjoint-variable restriction	disjunkte Variableneinschränkung
disjunction	Disjunktion
distinct variables	unterschiedliche/verschiedene Variablen
domain	Definitionsbereich
dummy variable	Dummy-Variable
effectively bound variable	effektiv gebundene Variable
effectively not free	effektiv nicht frei
element	Element
empty domain	leerer Definitionsbereich
empty set	leere Menge
empty substitution	leere Substitution
epsilon relation	Epsilon-Relation
equality	Gleichheit
error checking	Fehlerprüfung
errors in proofs	Fehler in Beweisen
essential hypothesis	essentielle Hypothese
Euclidean geometry	euklidische Geometrie
existential quantifier	Existenzquantor

Englisch	**Deutsch**
existential uniqueness quantifier	Eindeutigkeitsquantor
expression	Ausdruck
Extended Backus-Naur Form	erweiterte Backus-Naur-Form
extended frame	erweiterter Frame
extended language	erweiterte Sprache
family	Familie
Fermat's Last Therorem	Großer Fermatscher Satz
file inclusion	Dateieinbindung
file name	Dateiname
finitary proof	finitistischer Beweis
finite induction	finite Induktion
finite n-termed sequence	endliche n-gliedrige Folge
first-order logic (FOL)	Logik erster Ordnung
floating hypothesis	fließende Hypothese
formal logic	formale Logik
formal proof	formaler Beweis
formal system	formales System
formalism	Formalismus
forms	Formen
foundations of mathematics	Grundlagen der Mathematik
founded relation	fundierte Relation
four-color theorem	Vier-Farben-Satz
frame	Frame
free logic	freie Logik
free variable	freie Variable
function	Funktion
function value	Funktionswert
gaps in proofs	Lücken in Beweisen
global statement	globale Anweisung
Gödel's incompleteness theorem	Gödelscher Unvollständigkeitssatz
grave accent	einfaches Anführungszeichen
group theory	Gruppentheorie
hierarchy	Hierarchie
higher-order logic (HOL)	Logik höherer Ordnung
hypothesis	Hypothese
hypothesis association	Hypothesenzuordnung
hypothesis label	Hypothesenlabel
iff	genau dann, wenn
image	Bild

Englisch	**Deutsch**
implication	Implikation
implicit axiom	implizites Axiom
implicit substitution	implizite Substitution
inaccessible cardinal	unzugängliche Kardinalzahl
included file	eingebundene Datei
individual metavariable	individuelle Metavariable
individual variable	individuelle Variable
inference	Inferenz
inference form	Inferenzform
inference rule	Inferenzregel
infinite set	unendliche Menge
infinity	Unendlichkeit
inflix connective	Infix-Konnektor
informal proof	informeller Beweis
integer	ganze Zahl
intersection	Schnittmenge
intuitionism	Intuitionismus
keyword	Schlüsselwort
label	Label
label declaration	Label-Deklaration
label mode	Label-Modus
label reference	Label-Referenz
label sequence	Label-Sequenz
Lemmon-style proof	Lemmon-Stil Beweis
limit ordinal	Grenzzahl
local label	lokales Label
local variable	lokale Variable
logical AND	logisches UND
logical equivalenz	logische Äquivalenz
logical hypothesis	logische Hypothese
logical OR	logisches ODER
machine learning	maschinelles Lernen
Macintosh file name	Macintosh-Dateinamen
mandatory `$d` statement	obligatorische `$d`-Anweisung
mandatory disjoint-variable restriction	obligatorische disjunkte Variableneinschränkung
mandatory distinct-variable restriction	obligatorische disjunkte Variableneinschränkung
mandatory hypothesis	obligatorische Hypothese
mandatory variable	obligatorische Variable

Englisch	**Deutsch**
mandatory variable-type hypothesis	obligatorische Variablentyp-Hypothese
mapping	Zuordnung
markup notation	Auszeichnungsnotation
math mode	Mathe-Modus
math symbol	mathematisches Symbol
mathematical induction	vollständige Induktion
member	Glied
member	Mitglied
metalanguage	Metasprache
metalogic	Metalogik
metalogical completeness	metalogische Vollständigkeit
Metamath	Metamath
Metamath Language EBNF	Metamath-Sprache EBNF
Metamath Proof Explorer	Metamath Proof Explorer
metamathematics	Metamathematik
metatheorem	Metatheorem
metavariable	Metavariable
MIU-system	MIU-System
modal logic	modale Logik
model theory	Modelltheorie
modus ponens	Modus ponens
Monaco font	Monaco Schriftart
monospaced font	nichtproportionale Schriftart
natural deduction	natürliche Deduktion
natural number	natürliche Zahl
negation	Negation
nested block	verschachtelter Block
nesting level	Verschachtelungstiefe
non-trivial theory	nicht-triviale Theorie
normal proof	normaler Beweis
null set	leere Menge
number theory	Zahlentheorie
object	Objekt
object language	Objektsprache
omega	Omega
one-to-one function	injektive Funktion
one-to-one, onto function	bijektive Funktion
onto function	surjektive Funktion
operating system command	Betriebssystem-Befehl
operation	Operation

Englisch	**Deutsch**
operator precedence	Vorrangigkeit eines Operators
optional disjoint-variable restriction	optionale disjunkte Variableneinschränkung
optional hypothesis	optionale Hypothese
optional variable	optionale Variable
ordered pair	geordnetes Paar
ordinal addition	ordinale Addition
ordinal number	Ordinalzahl
ordinal predicate	Ordinalprädikat
outermost block	äußerster Block
pair	Paar
parser	Parser
Pasch's axiom	Axiom von Pasch
Peano's postulates	Peano-Postulate
Pierce's axiom	Pierces Axiom
plain text	einfacher Text
Poincaré conjecture	Poincaré-Vermutung
Polish notation	polnische Notation
pop	entfernen
postfix connective	Postfix-Konnektor
power class	Potenzklasse
power set	Potenzmenge
predicate calculus	Prädikatenlogik
prefix connective	Präfix-Konnektor
pre-statement	Prä-Aussage
Principia Mathematica	Principia Mathematica
printable character	druckbares Zeichen
printer	Drucker
proof	Beweis
Proof Assistant	Beweis-Assistent
proof length	Beweislänge
proof scheme	Beweisschema
proof step	Beweisschritt
proof theory	Beweistheorie
proper class	echte Klasse
proper definition	zulässige Definition
proper substitution	echte Substitution
propositional calculus	Aussagenlogik
provable assertion	beweisbare Behauptung
provable statement	beweisbare Aussage
pure mathematics	reine Mathematik

Englisch	**Deutsch**
push	schieben
qualifying expression	qualifizierender Ausdruck
quantifier theory	Quantifizierungstheorie
quantum logic	Quantenlogik
quantum mechanics	Quantenmechanik
range	Wertebereich
rational number	rationale Zahl
real number	reelle Zahl
recursion operator	Rekursionsoperator
recursive definition	rekursive Definition
redeclaration of symbols	Umdeklarierung von Symbolen
reduct	Redukt
reflection principle	Reflexionsprinzip
relation	Relation
restricted universal quantifier	eingeschränkter Allquantor
restricted existential quantifier	eingeschränkter Existenzquantor
restriction	Einschränkung
reverse polish notation (RPN)	umgekehrte polnische Notation
Robbins algebra	Robbins-Algebra
Robinson's resolution principle	Robinsons Resolutionsprinzip
RPN order	RPN-Reihenfolge
RPN stack	RPN-Stapel
rule	Regel
rule of generalization	Regel der Verallgemeinerung
Russell's paradox	Russells Paradoxon
Schröder-Bernstein theorem	Schröder-Bernstein-Theorem
scope	Gültigkeitsbereich
scoping statement	Gültigkeitsbereichsanweisung
sentential logic	Satzlogik
set	Menge
set difference	Mengendifferenz
set theory	Mengenlehre
set union	Vereinigungsmenge
simple declaration	einfache Deklaration
simple infinite sequence	einfache unendliche Folge
simple metatheorem	einfaches Metatheorem
singleton	einelementige Menge
software bug	Programmfehler, Bug
source buffer	Quellpuffer
source file	Quelldatei

Englisch	**Deutsch**
special characters	Sonderzeichen
stack	Stapel
Standard Deduction Theorem	Standard-Deduktionstheorem
statement	Anweisung, Aussage
stylized epsilon	stilisiertes Epsilon
subclass	Unterklasse
subset	Teilmenge
substitution	Substitution, Ersetzung
substitution map	Substitutionsabbildung
substitution theorem	Substitutionstheorem
successor	Nachfolger
Syllogism	Syllogismus
symbol	Symbol
syntax rules	Syntax-Regeln
tautology	Tautologie
temporary variable	temporäre Variable
term	Term
text editor	Texteditor
theorem	Theorem
theorem scheme	Theoremschema
tilde	Tilde
token	Token
topology	Topologie
transfinite cardinal	transfinite Kardinalzahl
transfinite recoursion	transfinite Rekursion
transitive class	transitive Klasse
transitive set	transitive Menge
tree-style proof	Baumdarstellung eines Beweises
truth table	Wahrheitstabelle
turnstile	Drehkreuz
type	Typ
typecode	Typcode
typesetting comment	Schriftsatzanweisung
unification	Vereinheitlichung
union	Vereinigung
universal class	universelle Klasse
universal quantifier	Allquantor
universe of a formal system	Universum eines formalen Systems
Unix file name	Unix-Dateiname
unordered pair	ungeordnetes Paar
unordered tripel	ungeordnetes Tripel

Englisch	**Deutsch**
variable	Variable
variable declaration	Variablendeklaration
variable substitution	Variablensubstitution
variable type	Variablentyp
variable-type hypothesis	Variablentyp-Hypothese
Venn diagram	Venn-Diagramm
Weak Deduction Theorem	Theorem der schwachen Deduktion
weak logic	schwache Logik
well-formed formula	wohlgeformte Formel (wff)
well-ordering	Wohlordnung
white space	Whitespace, Leerraum
word processing	Textverarbeitung
Zermelo-Fraenkel set theory	Zermelo-Fraenkel-Mengenlehre
ZFC set theory	ZFC-Mengenlehre

Literaturverzeichnis

[1] Donald J. Albers and G. L. Alexanderson, editors. *Mathematical People*. Contemporary Books, Inc., Chicago, 1985. [QA28.M37].

[2] Alan Ross Anderson and Nuel D. Belnap. *Entailment*, volume 1. Princeton University Press, Princeton, 1975. [QA9.A634 1975 v.1].

[3] John D. Barrow. *Theories of Everything: The Quest for Ultimate Explanation*. Oxford University Press, Oxford, 1991. [Q175.B225].

[4] H. Behnke, F. Backmann, K. Fladt, and W. Süss, editors. *Fundamentals of Mathematics*, volume I. The MIT Press, Cambridge, Massachusetts, 1974. [QA37.2.B413].

[5] J. L. Bell and M. Machover. *A Course in Mathematical Logic*. North-Holland, Amsterdam, 1977. [QA9.B3953].

[6] Andrea Blass. The interaction between category theory and set theory. In John Walter Gray, editor, *Mathematical Applications of Category Theory (Proceedings of the Special Session on Mathematical Applications Category Theory, 89th Annual Meeting of the American Mathematical Society, held in Denver, Colorado January 5–9, 1983)*, pages 5–29, Providence, Rhode Island, 1983. American Mathematical Society. [QA169.A47 1983].

[7] W. W. Bledsoe and D. W. Loveland, editors. *Automated Theorem Proving: After 25 Years (Proceedings of the Special Session on Automatic Theorem Proving, 89th Annual Meeting of the American Mathematical Society, held in Denver, Colorado January 5–9, 1983)*, Providence, Rhode Island, 1983. American Mathematical Society. [QA76.9.A96.S64 1983].

[8] George S. Boolos and Richard C. Jeffrey. *Computability and Logic*. Cambridge University Press, Cambridge, third edition, 1989. [QA9.59.B66 1989].

[9] John Campbell. *Programmer's Progress*. White Star Software, Box 51623, Palo Alto, CA 94303, 1991.

[10] Mario Carneiro. Conversion of HOL light proofs into metamath. *CoRR*, abs/1412.8091, 2014.

[11] Mario Carneiro. Natural deductions in the metamath proof language. 2014.

[12] Shang-Ching Chou. Proving elementary geometry theorems using Wu's algorithm. In W. W. Bledsoe and D. W. Loveland, editors, *Automated Theorem Proving: After 25 Years (Proceedings of the Special Session on Automatic Theorem Proving, 89th Annual Meeting of the American Mathematical Society, held in Denver, Colorado January 5–9, 1983)*, pages 243–286, Providence, Rhode Island, 1983. American Mathematical Society. [QA76.9.A96.S64 1983].

[13] Richard Courant and Herbert Robbins. Topology. In James R. Newman, editor, *The World of Mathematics, Volume One*, pages 573–590. Simon and Schuster, New York, 1956. [QA3.W67 1988].

[14] Haskell B. Curry. *Foundations of Mathematical Logic*. Dover Publications, Inc., New York, 1977. [QA9.C976 1977].

[15] Philip J. Davis and Reuben Hersh. *The Mathematical Experience*. Birkhäuser Boston, Boston, 1981. [QA8.4.D37 1982].

[16] Richard de Millo, Richard Lipton, and Alan Perlis. Social processes and proofs of theorems and programs. In Thomas Tymoczko, editor, *New Directions in the Philosophy of Mathematics*, pages 267–285. Birkhäuser Boston, Inc., Boston, 1986. [QA8.6.N48 1986].

[17] Robert E. Edwards. *A Formal Background to Mathematics*. Springer-Verlag, New York, 1979. [QA37.2.E38 v.1a].

[18] Herbert B. Enderton. *Elements of Set Theory*. Academic Press, Inc., San Diego, 1977. [QA248.E5].

[19] R. L. Goodstein. *Development of Mathematical Logic*. Springer-Verlag New York Inc., New York, 1971. [QA9.G6554].

[20] Michael Guillen. *Bridges to Infinity*. Jeremy P. Tarcher, Inc., Los Angeles, 1983. [QA93.G8].

[21] Alan G. Hamilton. *Logic for Mathematicians*. Cambridge University Press, Cambridge, revised edition, 1988. [QA9.H298].

[22] John Robert Harrison. Metatheory and reflection in theorem proving: A survey and critique. Technical Report CRC-053. SRI Cambridge, Millers Yard, Cambridge, UK, 1995. Available on the Web as `http://www.cl.cam.ac.uk/users/jrh/papers/reflect.html`.

[23] John Robert Harrison. Theorem proving with the real numbers. Technical Report 408, University of Cambridge Computer Laboratory, New Museums Site, Pembroke Street, Cambridge, CB2 3QG, UK, 1996. Author's PhD thesis, available on the Web at `http://www.cl.cam.ac.uk-/users/jrh/papers/thesis.html`.

[24] Horst Herrlich and George E. Strecker. *Category Theory: An Introduction.* Allyn and Bacon Inc., Boston, 1973. [QA169.H567].

[25] J. Roger Hindley and David Meredith. Principal type-schemes and condensed detachment. *The Journal of Symbolic Logic*, 55:90–105, 1990. [QA.J87].

[26] Douglas R. Hofstadter. *Gödel, Escher, Bach.* Basic Books, Inc., New York, 1979. [QA9.H63 1980].

[27] Andrzej Indrzejczak. Natural deduction, hybrid systems and modal logic. *Trends in Logic*, 30, 2010.

[28] D. Kalish and R. Montague. On Tarski's formalization of predicate logic with identity. *Archiv für Mathematische Logik und Grundlagenforschung*, 7:81–101, 1965. [QA.A673].

[29] J. A. Kalman. Condensed detachment as a rule of inference. *Studia Logica*, 42(4):443–451, 1983. [B18.P6.S933].

[30] Morris Kline. *Mathematical Thought from Ancient to Modern Times.* Oxford University Press, New York, 1972. [QA21.K516 1990 v.3].

[31] Morris Kline. *Mathematics, The Loss of Certainty.* Oxford University Press, New York, 1980. [QA21.K525].

[32] Oliver Knill. Some fundamental theorems in mathematics. 2018.

[33] Edna E. Kramer. *The Nature and Growth of Modern Mathematics.* Princeton University Press, Princeton, New Jersey, 1981. [QA93.K89 1981].

[34] Daniel Clemente Laboreo. *Introduction to natural deduction.* 2014.

[35] Edmund Landau. *Foundations of Analysis.* Chelsea Publishing Company, New York, second edition, 1960. [QA241.L2541 1960].

[36] Hugues Leblanc. On Meyer and Lambert's quantificational calculus FQ. *The Journal of Symbolic Logic*, 33:275–280, 1968. [QA.J87].

[37] Czeslaw Lejewski. On implicational definitions. *Studia Logica*, 8:189–208, 1958. [B18.P6.S933].

[38] Azriel Levy. *Basic Set Theory.* Dover Publications, Mineola, NY, 2002.

[39] Angelo Margaris. *First Order Mathematical Logic*. Blaisdell Publishing Company, Waltham, Massachusetts, 1967. [QA9.M327].

[40] Adrian R. D. Mathias. A term of length 4,523,659,424,929. *Synthese*, 133:75–86, 2002. [Q.S993].

[41] Norman D. Megill. A finitely axiomatized formalization of predicate calculus with equality. *Notre Dame Journal of Formal Logic*, 36:435–453, 1995. [QA.N914].

[42] Norman D. Megill and Martin W. Bunder. Weaker D-complete logics. *Journal of the IGPL*, 4:215–225, 1996. Available on the Web at `http://www.mpi-sb.mpg.de/igpl/Journal/V4-2/#Megill`.

[43] Elliott Mendelson. *Introduction to Mathematical Logic*. D. Van Nostrand Company, Inc., New York, second edition, 1979. [QA9.M537 1979].

[44] C. A. Meredith. Single axioms for the systems (C,N), (C,O) and (A,N) of the two-valued propositional calculus. *The Journal of Computing Systems*, 3:155–164, 1953.

[45] David Meredith. In memoriam Carew Arthur Meredith (1904-1976). *Notre Dame Journal of Formal Logic*, 18:513–516, 1977. [QA.N914].

[46] J. Donald Monk. Substitutionless predicate logic with identity. *Archiv für Mathematische Logik und Grundlagenforschung*, 7:103–121, 1965.

[47] A. W. Moore. *The Infinite*. Routledge, New York, 1989. [BD411.M59].

[48] James R. Munkres. *Topology: A First Course*. Prentice-Hall, Inc., Englewood Cliffs, New Jersey, 1975. [QA611.M82].

[49] E. Z. Nemesszeghy and E. A. Nemesszeghy. On strongly creative definitions: A reply to V. F. Rickey. *Logique et Analyse (N. S.)*, 20:111–115, 1977. [BC.L832].

[50] I. Németi. Algebraizations of quantifier logics, an overview. Version 11.4, preprint, Mathematical Institute, Budapest, 1994. A shortened version without proofs appeared in „Algebraizations of quantifier logics, an introductory overview“, *Studia Logica*, 50:485–569, 1991 [B18.P6.S933].

[51] M. Pavičić. A new axiomatization of unified quantum logic. *International Journal of Theoretical Physics*, 31:1753 –1766, 1992. [QC.I626].

[52] Roger Penrose. *The Emperor's New Mind*. Oxford University Press, New York, 1989. [Q335.P415].

[53] Ivars Peterson. *The Mathematical Tourist.* W. H. Freeman and Company, New York, 1988. [QA93.P475].

[54] Jeremy George Peterson. An automatic theorem prover for substitution and detachment systems. *Notre Dame Journal of Formal Logic*, 19:119–122, 1978. [QA.N914].

[55] Willard Van Orman Quine. *Set Theory and Its Logic.* The Belknap Press of Harvard University Press, Cambridge, Massachusetts, revised edition, 1969. [QA248.Q7 1969].

[56] J. A. Robinson. A machine-oriented logic based on the resolution principle. *Journal of the Association for Computing Machinery*, 12:23–41, 1965.

[57] T. Thacher Robinson. Independence of two nice sets of axioms for the propositional calculus. *The Journal of Symbolic Logic*, 33:265–270, 1968. [QA.J87].

[58] Rudy Rucker. *Infinity and the Mind: The Science and Philosophy of the Infinite.* Bantam Books, Inc., New York, 1982. [QA9.R79 1982].

[59] Bertrand Russell. Recent work on the principles of mathematics. *International Monthly*, 4:84, 1901.

[60] Bertrand Russell. *Mysticism and Logic, and Other Essays.* Barnes & Noble Books, Totowa, New Jersey, 1981. [B1649.R963.M9 1981].

[61] Eric Schmidt. Reductions in norman megill's axiom system for complex numbers. 2012.

[62] Joseph R. Shoenfield. *Mathematical Logic.* Addison-Wesley Publishing Company, Inc., Reading, Massachusetts, 1967. [QA9.S52].

[63] Daniel Solow. *How to Read and Do Proofs: An Introduction to Mathematical Thought Process.* John Wiley & Sons, New York, 1982. [QA9.S577].

[64] Harold M. Stark. *An Introduction to Number Theory.* Markham Publishing Company, Chicago, 1970. [QA241.S72 1978].

[65] E. R. Swart. The philosophical implications of the four-color problem. *American Mathematical Monthly*, 87:697–707, November 1980. [QA.A5125].

[66] George G. Szpiro. *Poincaré's Prize: The Hundred-Year Quest to Solve One of Math's Greatest Puzzles.* Penguin Books Ltd, London, 2007. [QA43.S985 2007].

[67] Gaisi Takeuti and Wilson M. Zaring. *Introduction to Axiomatic Set Theory*. Springer-Verlag New York Inc., New York, second edition, 1982. [QA248.T136 1982].

[68] Alfred Tarski. What is elementary geometry. In Leon Henkin, Patrick Suppes, and Alfred Tarski, editors, *The Axiomatic Method, with Special Reference to Geometry and Physics (Proceedings of an International Symposium held at the University of California, Berkeley, December 26, 1957 — January 4, 1958)*, pages 16–29, Amsterdam, 1959. North-Holland Publishing Company.

[69] Alfred Tarski. A simplified formalization of predicate logic with identity. *Archiv für Mathematische Logik und Grundlagenforschung*, 7:61–79, 1965. [QA.A673].

[70] Thomas Tymoczko, editor. *New Directions in the Philosophy of Mathematics*. Birkhäuser Boston, Inc., Boston, 1986. [QA8.6.N48 1986].

[71] Hao Wang. Theory and practice in mathematics. In Thomas Tymoczko, editor, *New Directions in the Philosophy of Mathematics*, pages 129–152. Birkhäuser Boston, Inc., Boston, 1986. [QA8.6.N48 1986].

[72] Daniel Whalen. Holophrasm: a neural automated theorem prover for higher-order logic. *CoRR*, abs/1608.02644, 2016.

[73] Alfred North Whitehead. *An Introduction to Mathematics*, 1911.

[74] Alfred North Whitehead and Bertrand Russell. *Principia Mathematica*. Cambridge University Press, Cambridge, second edition, 1927. (3 vols.) [QA9.W592 1927].

[75] Freek Wiedijk. The qed manifesto revisited. 2007.

[76] Stephen Wolfram. *Mathematica: A System for Doing Mathematics by Computer*. Addison-Wesley Publishing Co., Redwood City, California, second edition, 1991. [QA76.95.W65 1991].

[77] Larry Wos, Ross Overbeek, Ewing Lusk, and Jim Boyle. *Automated Reasoning: Introduction and Applications*. McGraw-Hill, Inc., New York, second edition, 1992. [QA76.9.A96.A93 1992].

Index