Die Entwicklung einer Annotationssprache für natürlichsprachlich formulierte mathematische Beweise

Bernhard Fisseni

7. September 2003

Inhaltsverzeichnis

Abbildungsverzeichnis

Danksagung

Für die Stellung des interessanten Themas danke ich Herrn Professor Dr. Winfried Lenders.

Für die konstruktiven Diskussionen während der Erstellung dieser Arbeit danke ich Herrn Dr. Bernhard Schröder.

Für die Erlaubnis, in seiner noch nicht ganz fertigen Dissertation lesen zu dürfen, danke ich Herrn Dipl.-Inf. Claus Zinn.

Für die Überlassung des Code-Beispiels im Kapitel über Mizar (Kap. 4.4.2 auf Seite 39) danke ich Herrn Dipl.-Math. Patrick Braselmann.

Für die Auffrischung meiner Kenntnisse über die Bezeichnung von Objekten in der modernen Geometrie (Kap. 8.5.3.2 auf Seite 77), die seit der Schulzeit verblaßt waren, danke ich Frau Meike Jungebloed.

1. Einleitung: ProofML und mathematische Beweise

Mathematische Texte sind natürlichsprachlich, nicht formalsprachlich. Dennoch gilt als ein Kriterium für die Annahme ihrer Korrektheit, daß sie in einem Logik-Kalkül komplett formalisierbar sind. Formalisierte Beweise können mit einem Rechner verarbeitet werden, der sie dann mit der ihm eigenen Akribie – allerdings ohne jene Kreativität, die zur Schaffung von Beweisen notwendig ist – gegen allgemein akzeptierte Regeln verifizieren kann. van Benthem (2003 : Kap. 2) bezeichnet eine solche Formalisierung allerdings *'rather austere medicine'*; solche formalisierten Beweise sind für den Leser schwer verständlich, weil sie zu detailreich sind.

Eine solche formale Verifikation unvorbereiteter Texte ist bisher nicht einfachhin möglich, allerdings gibt es Ansätze zu Systemen, die genau dies versuchen (vgl. Kap. 4.3 auf Seite 36). Solche Ansätze entwerfen allerdings eine Repräsentation des Beweises, die eher der statischen Struktur eines formalisierten Beweises entspricht und nicht mehr an die sprachliche Struktur rückgekoppelt wird.

Ziel dieser Arbeit ist die Entwicklung einer Annotationssprache für natürlichsprachliche mathematische Beweise, die der natürlichsprachlichen Struktur durch die Annotation eine formal-logische Interpretation zuordnet. Eine solche Annotation kann den Anspruch der Formalisierbarkeit explizit machen. Wir bauen diese Sprache, ProofML, auf den Grundlagen auf, die Koepke und Schröder (2003) in ihrem Aufsatz gelegt haben.

In einem weiteren Schritt soll im Forschungsprojekt *Beweisformen* im Rahmen der Forschergruppe *Wissensformate* der Universität Bonn ein Programm entwickelt werden, das die annotierten Texte im Hinblick auf die formallogische Struktur validiert, aber auch eine Rückkoppelung an den Text erlaubt. Im Falle eines Fehlers in der Verarbeitung oder Überprüfung wäre es so beispielsweise leichter möglich einzugrenzen, wo er aufgetreten ist.

Um die Arbeit empirisch zu gestalten und die Möglichkeiten von ProofML zu demonstrieren, haben wir in der vorliegenden Arbeit ein kleines Korpus erstellt, das für einen bestimmten Bereich nach unserer Einschätzung (sprachlich) repräsentative Beweise beinhaltet (vgl. auch die Einleitung zum Praxis-Teil).

Teil I.

Mathematische Beweise als natürlichsprachliche Texte

2. Der „mathematische Beweis“ als Textsorte

Die textlinguistische Kategorie der *Textsorte* umfaßt Merkmale von Texten, die für ihre Analyse relevant sein können. Wir gehen hier von einer Einteilung von Brinker (2001) aus und gehen nach seinem Schema (Brinker 2001 : 149) vor.

Zunächst ist die *Textfunktion* zu bestimmen. In mathematischen Beweisen ist die informative Funktion eindeutig vorherrschend. Dies bestätigt in überraschendem Maße sogar Lamport (1993), indem er berichtet, Mathematiker seien daran interessiert, Fehler in ihren Texten zu vermeiden.[1]

Das Thema eines mathematischen Beweises sind mathematische Zusammenhänge, also zeitlose Verhältnisse außerhalb von Emittenten und Rezipienten.

Die *Themenentfaltung* ist im wesentlichen *explikativ* (vgl. Brinker 2001 : 70f), es wird zunächst das Explanandum genannt, welches anschließend belegt wird durch „singuläre Aussagen, die die Anfangsbedingungen beschreiben“ und „Gesetzesaussagen“ (Brinker 2001 : 70); abschließend wird es häufig noch einmal ausformuliert.[2]

Im mathematischen Beweis wird Kohärenz auf verschiedene Arten und Weisen hergestellt.

Spezifisch für den mathematischen Diskurs sind jedoch die Benennung und Verwendung von Variablen, bestimmte Schlußfiguren (etwa die vollständige Induktion) und eine starke Mischung von natürlicher Sprache mit Formelsprache; außerdem besitzt die Mathematik, wie jede Fachsprache, eine eigene Terminologie.

Baur (1999) und Zinn (1999, to appear) nennen einige Beispiele für spezifisch mathematische Konstruktionen, auf die wir im praktischen Teil der Arbeit als Ergänzung zu den Beweisen eingehen, die wir annotiert haben.

[1] Die Seitenhiebe gegen die Informatiker, diese seien stärker an der Menge der Veröffentlichungen als an ihrer Korrektheit interessiert, möchten wir hier außer Acht lassen.

[2] Daß das Explanandum am Anfang des Textes noch einmal ausdrücklich genannt wird, stellt eine der vielen Abweichungen vom zugrundeliegenden Schema (Brinker 2001 : 70) dar, die für diese Art der Themenentfaltung typisch sind (Brinker 2001 : 71).

Teil II.

Die Annotation und Repräsentation natürlichsprachlicher Beweise

3. Linguistische Methoden der Diskurs-Repräsentation

In der Linguistik sind einige Theorien entwickelt worden, die Einfluß auf die Entwicklung von ProofML gehabt haben, auch wenn ProofML natürlich möglichst nicht von einer bestimmten linguistischen Theorie abhängen soll.

3.1. Allgemeine Anforderungen für unser Repräsentationssystem

Kompositionalität und Dynamik der Bedeutungskonstitution Es ist ein Ziel in der formalen Beschreibung der Semantik natürlicher Sprache, Formalismen zu entwickeln, die *kompositional* vorgehen. Dies bedeutet — grob gesagt —, daß die Bedeutung einer Phrase das Produkt der Bedeutungen ihrer Konstituenten und der Art der Bildung der Phrase selbst ist.[1]

Dynamik bedeutet im Zusammenhang der semantischen Beschreibung, daß Äußerungen immer in einem bestimmten Kontext stattfinden und diesen Kontext auch verändern. Die Intuition, die zum Konzept der Dynamik führt, ist die Erfahrung, daß ein Gegenstand, sobald er auf geeignete Weise in den Diskurs(kontext) eingeführt worden ist, wieder aufgegriffen werden kann, weil er nun im Kontext vorhanden ist.

Aspekte des dynamischen Aufbaus der Bedeutung werden wir im Zusammenhang mit der *Dynamischen Prädikatenlogik* (Kap. 3.3 auf Seite 18) besprechen; einen wichtigen Aspekt der Kompositionalität für die Annotation werden wir im Abschnitt über *generalisierte Quantoren* (Kap. 3.2 auf Seite 16) behandeln.

[1] Ein Beispiel, in dem Kompositionalität nicht zutrifft, stellen Phraseologismen dar, wenn man sie „wörtlich nimmt“, also etwa in (1) für *mit Kind und Kegel* die Bedeutung: „mit ehelichem und unehelichem Kind“ erschließt — dagegen bedeutet die Phrase nur noch „mit allen Siebensachen“.

(1) Er verreiste [mit Kind und Kegel].

Kommunikationsmaximen Zinn (2000 : 2f) nennt Gründe, die letztlich die Bitterkeit der Medizin erklären mögen, von der van Benthem (2003 : Kap. 2) spricht; er bezieht sich dabei auf die Maximen, die Grice (1968) für Diskurse aufstellt: *Quality, Manner, Quantity, Relation*. Diesen folgen mathematische Beweise, wenn sie in natürlicher Sprache verfaßt sind; sie sind aber mit einer kompletten Formalisierung des Beweises nicht zu vereinbaren.

Zwei Maximen sind nach Zinn (2000 : Kap. 1) leicht auf den mathematischen Diskurs anzuwenden: *Quality* (Grice 1968 : 309) besagt nach Zinn (2000 : 2), daß Mathematiker nur Wahres behaupten. *Manner* bezeichnet die über Jahrhunderte ausgebildete mathematische Sprache.

Daß Mathematiker die Maximen von *Manner* und *Quality* befolgen, stellt für unsere Annotation kein Problem dar, sie beeinflußt sie allerdings auch nur insofern, als es nicht nötig ist, eine Annotation für „Lügen“ zu entwerfen. Auf Stilfragen muß man ggf. Rücksicht nehmen, wenn man ein diachronisches mathematisches Korpus aufstellt. Aber auch sie sind nicht besonders problematisch, da die mathematische Sprache auf allen fachlichen Niveaus und sogar über die Jahrhunderte von Euklid bis in ein Logik-Buch des 20. Jahrhunderts überraschend große Einheitlichkeit aufweist.[2]

Die Bedeutung der Maximen von *Quantity* (sei genau so informativ, wie nötig) und *Relation* (*'be relevant'*) sind — so Zinn (2000 : 2) — schwieriger anzuwenden. Sie setzen aber voraus, daß der Autor (allgemeiner: *Emittent*) über sein Publikum (die *Rezipienten*) genug weiß, um zu entscheiden, welche Informationen er voraussetzen kann — und damit, welche er weglassen oder verkürzt darstellen kann. Hier stellt der mathematische Diskurs natürlich keine besondere Ausnahme dar.

Die Ergänzung vorausgesetzter Informationen stellt in der Tat eines der Hauptprobleme dar, die bei vollständiger Formalisierung eines Beweises entstehen. Die angemessene Repräsentation von Informationen, die ergänzt werden, ist allerdings eine der wichtigsten Anforderungen an unser System.

Hintergrund und Bridging Die Annahmen des Emmitenten darüber, welchen gemeinsamen Kontext er mit dem Rezipienten besitzt, ermöglichen ihm also Schlüsse darauf, welche Informationen vorausgesetzt werden können, welche auf jeden Fall explizit gegeben werden müssen.

[2] Vgl. den Abschnitt über die Annotation von Namen (Kap. 8.5 auf Seite 67), der auf den mglw. größten Unterschied in der Veränderung der Struktur mathematischer Beweise eingeht.

Aus einer großen Überlappung des Kontextwissens, des *gemeinsamen Hintergrundwissens* der Kommunikanten erklärt sich die erstaunliche Kürze eines Beweises in Veröffentlichungen für ein Fachpublikum — oder auch die Länge „trivialer“ Beweise in Schulbüchern.

Mit dem Hintergrundwissen hängt das Problem des *Bridgings* zusammen, der kontextbedingten Verfügbarkeit implizit eingeführter Diskursreferenten.

(2) a. Sei ABC ein Dreieck.
b. Der Schwerpunkt heiße S.

Dieser Satz hat explizit das Dreieck ABC eingeführt, implizit aber auch die Punkte A, B, C, die Seiten $|AB|$, $|AC|$ und $|BC|$; *per definitionem* ist auch die Existenz der Mittelsenkrechten, eines Schwerpunktes und noch vieler anderer Objekte deklariert worden. Das mathematische Hintergrundwissen, die mathematische Theorie liefert also viele solcher im Kontext erschließbaren Entitäten (*Inferrables*, s. Prince 1981) .

Ansätze zur Behandlung des Kontextes in dem Umfange, wie er uns hier betrifft, finden wir in gewissem Maße in der *Diskursrepräsentations-Theorie* (Kap. 3.4 auf Seite 20), aber auch in Systemen der Künstlichen Intelligenz, die mit Ontologien[3] arbeiten, sowie in den *Proof Assistants* (Kap. 4.2 auf Seite 35) und in typenlogischen Grammatiken, wie sie Ranta (2002, 1999, 1996) vorschlägt (vgl. Kap. 4.5 auf Seite 40).

Andere Aspekte natürlichsprachlicher Diskurse, die sehr häufig diskutiert werden, stellen für unser Thema keine Frage dar. Dazu gehören etwa Zeit/Tempus und Aspekt/Aktionsart. Das Verhältnis von Tempora und zeitlichen Abfolgen, von Aspekten und Aktionsarten ist im mathematischen Beweis kaum wichtig, da mathematische Beweise prinzipiell zeitlos sind.

Zinn (2000 : Kap. 2.3) nennt *'Requirements for a theory of discourse representation'*, die geeignet sein soll, mathematische Beweise zu repräsentieren. Einige von ihnen sind auch für unsere Annotation sinnvoll, andere hingegen spielen nur in Zusammenhang mit seinem sprachverarbeitenden System (s. Kap. 4.3 auf Seite 36) eine Rolle.

Die Forderung, daß alle Entitäten in einer Datenstruktur verwaltet werden, weil es um die Entwicklung eines Programmes geht, ist für ProofML insofern relevant, als wir mit unserer Annotation die Struktur des Textes in einer Datenstruktur explizit machen. Die Datenstruktur soll nach Zinn wiederum selbst strukturiert sein, da der Diskurs strukturiert ist. Auch diese Forderung erfüllt eine XML-Annotation trivialerweise. Außerdem

[3] *Ontologie* im Sinne der Künstlichen Intelligenz, nicht der Philosophie.

müsse sie Sorten enthalten, da es Anaphern für verschiedene Typen von Ausdrücken gebe. Dies scheint für die Annotation nicht unbedingt notwendig.

Für die folgenden Anforderungen gibt Zinn (2000 : Kap. 2.3) keine weiteren Erläuterungen.

Zinn fordert, die Repräsentation müsse zwischen gebundenen und freien Variablen unterscheiden können.

Diese Forderung ist im Falle einer maschinellen Verarbeitung von Texten natürlich relevant, damit Koreferenzen aufgelöst werden können. Da wir Koreferenz zwischen Ausdrücken explizit markieren, ist die Markierung einer Bindung für uns allerdings nicht notwendig. (Vgl. auch Kap. 3.3 auf Seite 18 zum Konzept der Variablenbindung, das ProofML zugrundeliegt.)

Zinn verlangt, es müßten verschiedene Arten von Diskursreferenten verfügbar sein für Terme, Aussagen – sogar mit Modalitäten wie „angenommen“ (*'assumed'*), „gefordert“ (*'postulated'*) und „hergeleitet“ (*'derived'*).

Nach Zinn muß eine Repräsentationstheorie auch für „Argumentationsstrukturen“ (*'argumentation structures'*) wie etwa Fälle in Fallunterscheidungen eine eigene Art Diskursreferenten zur Verfügung stellen (auf diesen Punkt geht er leider nicht näher ein).

Dies ist wohl parallel zu den parametrisierten Schlußregeln in ProofML zu sehen (Kap. 9.2 auf Seite 93).

Ferner fordert Zinn, daß Namen für mathematische Objekte vergeben werden können. Dem schließen wir uns an und werden diese Fragestellung in Kap. 8.5 auf Seite 67 ausführlich behandeln.

Zinn fordert weiterhin, daß es möglich sein muß, die *'discourse relations'* zwischen Aussagen zu beschreiben; eine deduktive Komponente sollte diese rhetorischen Relationen herleiten oder überprüfen können. Diese Forderung ist natürlich für eine Annotationssprache nicht sinnvoll, für eine Automatisierung der Annotation oder eine Beweisüberprüfung sicherlich schon.

3.2. Generalisierte Quantoren

Die Theorie der *generalisierten Quantoren* stammt aus der Mathematik und wurde von Barwise und Cooper (1981) zum ersten Mal auf die natürliche Sprache angewandt. Sie dient auch in ProofML als Ansatzpunkt für die Behandlung von Nominalphrasen (vgl. hierzu auch Koepke und Schröder 2003 : 431f).

Während die Prädikatenlogik nur zwei Quantoren kennt (∀, ∃), kennt die natürliche Sprache viele Ausdrücke, die sich durch Verwendung von Quantoren modellieren lassen. Etwa kann man *wenige Quantoren* oder *einige Menschen* so verstehen, daß man sie eine Variable binden läßt. Das Ergebnis sähe dann etwa aus wie in (3-b).

(3) a. Wenige Menschen jodeln.
b. (wenig-mensch′ x)[jodel x]

Diese Darstellung kann man so anpassen, daß die Dreiteilung des Ausdrucks in der natürlichen Sprache auch in der Formel wiedergespiegelt wird:

(4) (wenig x)(mensch′)[jodel′(x)]

In der Terminologie der Theorie der generalisierten Quantoren nennt man *wenige* den *Determinierer*, *Menschen* den *Restriktor* und — wie bei ∀ und ∃ — *jodeln* den Geltungsbereich oder *Skopus* des Quantors.

Diese Einteilung werden wir in ProofML übernehmen und zur Grundlage der Annotation quantifizierter Nominalphrasen machen (s. Kap. 8.4.5 auf Seite 60).

Es ist unter Semantikern üblich, die Schreibweise aus (4) soweit zu verfeinern, daß man die Bedeutung der Determinierer noch genauer bestimmen kann und daß sie als „semantische Primitive“ nicht notwendig sind. Der Demonstration halber geben wir in Abb. 3.1 auf der nächsten Seite ein Lexikon an, das aus Lambda-Ausdrücken aufgebaut ist und darunter in (5-b) die Repräsentation des Satzes, die sich aus diesem Lexikon ergibt.[4] Im folgenden werden wir die Interpretation generalisierter Quantoren allerdings grundsätzlich ins Lexikon und damit ins Hintergrundwissen verbannen, ebenso wie die Definition mathematischer Theorien.

Wir übernehmen aber die kompositionale Analyse komplexer Nominalphrasen als generalisierte Quantoren insofern, als wir in der Annotation solche Nominalphrasen in Determinierer, Restriktor und Skopus aufteilen (vgl. Kap. 8.4.5 auf Seite 60).

[4] Um die Formel einfach zu halten, interpretieren wir „wenige x“ so, als bedeute es, „weniger als die Hälfte aller x“.

wenige :	$\langle\langle e,t\rangle,\langle\langle e,t\rangle,t\rangle\rangle$	$\lambda P.\lambda Q.0,5 \cdot \lvert(\lambda x.Px)\rvert > \lvert(\lambda x.(Px) \wedge (Qx))\rvert$
Menschen :	$\langle e,t\rangle$	$\lambda x.(\text{mensch}'\ x)$
jodeln :	$\langle\langle e,t\rangle,t\rangle$	$\lambda x.(\text{jodel}'\ x)$

(5) a. Wenige Menschen jodeln.
b. $0,5 \cdot \lvert\lambda x.(\text{mensch}'\ x)\rvert > \lvert(\lambda x.(\text{mensch}'\ x)(\text{jodel}'\ x)\rvert$

Abb. 3.1: **Jodeln aufgelöst im λ-Kalkül**

3.3. Dynamische Prädikatenlogik (DPL)

Aus der Sicht des Sprachwissenschaftlers liegt ein Defizit der Prädikatenlogik darin, daß sie gleiche Phrasen — auch wenn aus linguistischer Sicht nicht einmal Polysemie vorliegt — je nach Kontext vollkommen anders übersetzt und damit das Kriterium der *Kompositionalität* verletzt; dies hat hauptsächlich mit den Geltungsbereichen (Skopi) der Quantoren[5] zu tun. Thematisiert wurde dies zuerst von Geach mit Beispielsätzen über Bauern und Esel, daher heißen die entsprechenden Sätze im Englischen auch *'donkey sentences'* Ein kleines Beispiel:

(6) a. Wenn ein Mensch einen Computer hat, verflucht er ihn.
b. $(\exists x \exists y.[(\text{mensch}'\ x) \wedge (\text{compi}'\ y) \wedge ((\text{hab}'\ y)x)]) \Rightarrow ((\text{verfluch}'\ y)x)$
c. $(\exists x \exists y.[(\text{mensch}'\ x) \wedge (\text{compi}'\ y) \wedge ((\text{hab}'\ y)x)] \Rightarrow ((\text{verfluch}'\ y)x))$
d. $(\forall x \forall y.[(\text{mensch}'\ x) \wedge (\text{compi}'\ y) \wedge ((\text{hab}'\ y)x)] \Rightarrow ((\text{verfluch}'\ y)x))$

Eine übliche Übersetzung für den unbestimmten Artikel ist der Existenzquantor; es ist aber mit dieser Übersetzung nicht möglich, (6-a) adäquat zu beschreiben: (6-b) bedeutet lediglich, daß, sofern ein Mensch einen Computer besitzt, jemand — oder etwas — jemanden — oder etwas — verflucht; die Bindung der Variablen x und y ist aufgehoben. Man kann auch verschiedene andere Varianten ausprobieren — sofern man auf der gewünschten Interpretation einer indefiniten Nominalphrase als Variableneinführung mit dem Existenzquantor besteht, bleiben sie falsch, z.B. besagt (6-c), es gibt ein x und ein y, so daß, falls x ein Mensch und y ein Computer ist und y x gehört, x y verflucht; diese Formel ist wahr, sofern es zwei Dinge gibt, die den in der inneren Formel beschriebenen Sachverhalt erfüllen, also etwa einen Friseur, der eine Schere hat; ob er die Schere verflucht oder nicht, der Satz ist wahr. Man müßte zwei Allquantoren wählen und

[5] In der Standardprädikatenlogik verwendet man klassischerweise lediglich $\forall$ und $\exists$, aber siehe auch Kap. 3.2 auf Seite 16.

$$(\exists x.\alpha) \wedge \beta \quad \Leftrightarrow \quad (\exists x.\alpha \wedge \beta) \tag{3.1}$$
$$(\exists x.\alpha) \rightarrow \beta \quad \Leftrightarrow \quad (\exists x.\alpha \rightarrow \beta) \tag{3.2}$$
$$(\exists x.\alpha \rightarrow \beta) \quad \Leftrightarrow \quad \forall x.(\alpha \rightarrow \beta) \tag{3.3}$$

Dies ist unabhängig davon, ob im folgenden x in α oder β frei vorkommt.

Abb. 3.2: **Regeln der DPL nach Groenendijk und Stokhof (1991 : 65)**

ihren Geltungsbereich auf den gesamten Satz ausdehnen, um eine einigermaßen richtige Übersetzung des Satzes zu finden, und man müßte diesen Quantoren den gesamten Satz als Geltungsbereich zuordnen ((6-d)). Dies ist kontraintuitiv und wiederspricht, was den Geltungsbereich betrifft, der Kompositionalitätsforderung.

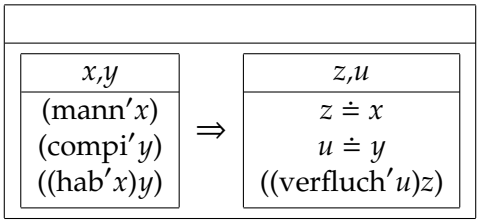
Dynamische Prädikatenlogik ist ein Formalismus, der syntaktisch der klassischen Prädikatenlogik entspricht, aber eine Semantik besitzt, die „dynamisch“ ist (für einen Vergleich verschiedener solcher dynamischer Ansätze zur Repräsentation von Diskursen vgl. Kuschert 1996 : Kap. 23), vor allem in Bezug darauf, wie der Geltungsbereich von Variablen behandelt wird, die durch Quantoren gebunden sind.

So ist die Übersetzung von (6-a) in DPL tatsächlich (6-b), aber mit der zusätzlichen Bedingung, daß die Variablen x und y auch nach dem Ende des Geltungsbereiches 3.1 des Quantors an die Werte gebunden sind, die ihnen zu Beginn der Formel zugewiesen wurden.

Wir nehmen uns vor, Variablennamen jeweils nur einmal zu verwenden, so daß wir die Probleme vermeiden können, die durch die einfache Handhabung der DPL entstehen können (*'destructive assignment'*, vgl. Kuschert 1996 : 32, van Eijck und Kamp 1997 : Kap. 6).

Wichtig ist in unserem Zusammenhang vor allem, daß der Skopus von Existenzquantoren in Konjunktionen und Implikationen über Formelgrenzen hinweg gilt und daß Existenzquantoren, die in generischen Aussagen auftreten, universell interpretiert werden ('they have "universal" force', Groenendijk und Stokhof 1991 : 65), es gelten also die Regeln aus Abb. 3.2.

Wir werden auch zunächst keine Regeln dafür aufstellen, unter welchen Bedingungen es möglich ist, auf Variablen aus vorangehenden Textsegmenten zuzugreifen. Solche Regeln sollten aus einem Korpus von mathematischen Texten ableitbar sein.



≐ bedeutet Gleichsetzung von Variablen

Eine äquivalente (standard)prädikatenlogische Formel ist:

$$\forall x.\forall y.((\text{mann}'x) \wedge ((\text{hab}'x)y)) \rightarrow ((\text{verfluch}'y)x)$$

Abb. 3.3: **(6-a) als DRS nach van Eijck und Kamp (1997)**

Die Verwendung dynamischer Existenzquantoren wird auch bereits von Koepke und Schröder (2003 : Kap. 2) vorgeschlagen.

3.4. Diskursrepräsentationstheorie (DRT)

Die Diskursrepräsentationstheorie (DRT) wurde von Hans Kamp und Uwe Reyle Ende der 80er Jahre entwickelt. Sie ist auch Grundlage der Repräsentation mathematischer Beweise, die Zinn (to appear, 2003, 2000, 1999) in seinem System vip verwendet; welches die Anforderungen von ProofML und vip an eine Repräsentationsstruktur sind, stellen wir in Kap. (2) auf Seite 15 dar. Es gibt recht viele Versionen der DRT.

Auch die DRT bemüht sich, das soeben erwähnte Skopus-Problem in Angriff zu nehmen. Für unsere Zwecke läßt sich eine in einigen Aspekten äquivalente Formalisierung zu derjenigen der DPL verwenden (vgl. van Eijck und Kamp 1997 : 10). Eine explizite Bindung von Variablen findet in der klassischen DRT nicht durch Quantoren statt, sondern durch den Kontext (eine „Box“).

van Eijck und Kamp (1997 : 45–48) stellen eine Version von DRT vor, in der Lexikon-Einträge für generalisierte Quantoren gegeben werden, die eine analoge Semantik erhalten wie in der DPL, wie wir es auch in unserer Annotation anstreben (s. Kap. 3.2 auf Seite 16).

Die Idee der DRT ist — ähnlich wie in der DPL —, daß Äußerungen in dem Kontext interpretiert werden, der ihnen vorangeht. Hat dieser Kontext z.B. bestimmte Variablen

gebunden — „Diskursreferenten“ (sie entsprechen den Variablen in DPL) eingeführt —, so sind diese prinzipiell in folgenden Sätzen verfügbar; allerdings gibt es bestimmte Einschränkungen für negierte Sätze, Implikationen etc. Ist eine Box das Antezedens einer Implikation, so sind, ähnlich wie in der DPL, die nur in ihr gebundenen Variablen implizit allquantifiziert.

Für uns besonders interessant ist, daß die DRT, zumindest wie sie van Eijck und Kamp (1997 : Kap. 9) vorstellen, Diskursreferenten zulassen kann, die im Kontext der Situation verankert sind, nicht nur solche, die im sprachlichen Kontext des Diskurses auftreten. An dieser Stelle kann man sich etwas vorstellen, das den mathematischen Kontext modelliert, etwa wie eine Ontologie. Eine solche Wissensbasis über das mathematische Umfeld findet sich auch in allen Computer-Systemen, die mathematische Beweise verarbeiten (s. Kap. 4 auf der nächsten Seite).[6]

Methoden zur Beschreibung des Hintergrundwissens werden wir in Kap. 6 auf Seite 47 näher besprechen.

3.5. Zusammenfassung

Wir fassen kurz zusammen, welche Prinzipien der lingustischen Theorien zur Diskursrepräsentation wir für unsere Annotation übernehmen:

Von der Theorie der generalisierten Quantoren, aber auch von der DPL (und in gewissem Maße der DRT neueren Datums) übernehmen wir den Anspruch, die Dynamik und Kompositionalität der natürlichen Sprache auch in der Repräsentation ihrer Inhalte wiederzugeben.

Die Berücksichtigung des Hintergrundwissens in einigen Versionen der DRT, die bedeutet, daß manche Diskursreferenten ohne Einführung verfügbar sind, übernehmen wir ebenfalls und ziehen daraus die Schlußfolgerung, daß wir ein Modell des mathematischen Hintergrundes zumindest andeuten müssen.

[6] Solches Hintergrundwissen erwähnen Groenendijk und Stokhof (1991) nicht explizit; sie schließen es aber auch nicht aus.

4. Repräsentation mathematischer Beweise für die maschinelle Verarbeitung

In diesem Kapitel wollen wir untersuchen, inwieweit wir für die Annotation natürlichsprachlicher Beweise Ideen aus Computer-Systemen übernehmen können, die mathematische Beweise verarbeiten, oder aus Systemen, die zur Annotation oder Übersetzung von Beweisen geschrieben wurden, die zumindest teilweise formalisiert sind.[1]

Wenn wir von maschineller Verarbeitung mathematischer Beweise sprechen, können wir verschiedene Ansätze unterscheiden. Zunächst gibt es Systeme, die Beweise verarbeiten, die soweit formalisiert sind, daß ihre Sprache nicht mehr natürlich erscheint; die Beweise sind oft sehr explizit und soweit aufbereitet, daß ihre maschinelle Verarbeitung recht einfach ist.

Auf der anderen Seite stehen Systeme, die natürlichsprachliche Beweise verarbeiten, wie sie in mathematischen Publikationen zu finden sind.

Zwischen diesen beiden Ansätzen der Verarbeitung komplett formalisierter und der Verarbeitung natürlichsprachlicher Beweise finden sich einige Zwischenstufen. Dies liegt auch an der Form mathematischer Beweise.

Prinzipiell kann man nämlich bereits die Form, in der mathematische Beweise üblicherweise publiziert werden, als „hybrid“ bezeichnen, da meist Formel-Ausdrücke und „rein natürlichsprachliche“ Ausdrücke gemischt sind.[2]

Einige der formalen Methoden, einen Beweis aufzuschreiben, sind dagegen mit dem Ziel entwickelt worden, daß ihre Syntax sich an die der natürlichen Sprache anlehnt und intuitiv verständlich ist; allerdings sind ambige Formulierungen üblicherweise ausgeschlossen, ebenso Synonymie (also Ambiguität in der Wortwahl).

[1] Ein System, das zur Annotation natürlichsprachlicher mathematischer Beweise gedacht ist, kennen wir nicht.

[2] Einen Beispielbeweis in natürlicher Sprache gibt Lamport (1993), um zu zeigen, wie vorteilhaft ein gewisser Grad an Formalisierung, vor allem in der Benennung mathematischer Entitäten ist.

4.1. Bisherige Markup-Sprachen

Es gibt bisher bereits einige Markup-Sprachen für mathematische Beweise. Diese sind vor allem für formalisierte Beweise gedacht. Wir werden kurz die uns bekannten Markup-Sprachen darstellen und aufzeigen, inwieweit sie für uns interessant sind.

4.1.1. XML

XML (Bray et al. 1997) steht nicht in einem besonders engen Verhältnis zur Annotation mathematischen Wissens; allerdings ist es als allgemeine Annotationssprache auch bereits auf mathematische Texte angewandt worden. Auch ProofML wird, und darin folgen wir Koepke und Schröder (2003), XML verwenden; daher beschreiben wir kurz die Grundlagen von XML, wie sie für ProofML relevant sind. Eine erschöpfende Beschreibung würde natürlich den Rahmen dieser Arbeit sprengen, sie liegt im XML-Standard des *World Wide Web Consortium*s vor (Bray et al. 1997).

XML ist das „schlanke Geschwister“ von SGML, der *Standardized General Mark-up Language*, die in den achtziger Jahren für die Annotation von Texten entwickelt wurde.

SGML und XML sind Metasprachen, die die Definition von Sprachen erlauben („SGML- oder XML-Anwendungen“). Die Metadaten werden um die Daten herum gesetzt und regelhaft durch spitze Klammern (<,>) markiert (vgl. Abb. 4.1 auf der nächsten Seite). Der Text wird eingeteilt in (möglicherweise verschachtelte) *Elemente*, die auch Text enthalten können. Elemente werden von einem öffnenden und einem schließenden ‘Tag’ mit seinem Namen umschlossen (die <`element`*n*> in Abb. 4.1 auf der nächsten Seite). Kommentare können zwischen Elementen eingeschoben werden und werden zwischen <!-- und --> geschrieben. Ein solches XML-Dokument kann man also als eine Baumstruktur verstehen (s. Abb. 4.2 auf Seite 25. Es darf dabei nur ein Element geben, das die Wurzel des Baumes darstellt (<`element1`> ist das *Wurzel-Element* in Abb. 4.1 auf der nächsten Seite).

Weitere Daten, üblicherweise Meta-Informationen, über die Elemente können in *Attributen* (`attribut1` in Abb. 4.1 auf der nächsten Seite) angegeben werden, denen Werte zugeordnet werden (`"wert1"` in Abb. 4.1 auf der nächsten Seite). Die Struktur der Informationen in Attributen ist „flach“, da nur fortlaufender Text angegeben werden kann. Es verstößt — außer in wenigen, geregelten Ausnahmen — gegen die Konventionen, in Attributen Daten abzulegen, die hierarchisch strukturiert sind; solche Inhalte werden meist in Elementen abgelegt.

```
  <element1>
    <element2/>
<!-- alternativ:
<element2></element2>
-->
    <element3>
      <element4 attribut1="wert1">text</element4>
    </element3>
  </element1>
```

Abb. 4.1: **einfaches XML/SGML-Dokument: Quellcode**

Wenn man ein XML-Dokument als Baumstruktur auffaßt, spricht man von Element-Knoten, die wiederum Element-Knoten, aber auch Attribut-Knoten, Kommentar-Knoten und Text-Knoten als Kinder haben können. Vor allem die Behandlung von Text-Knoten ist nicht in allen Datenstrukturen, die zur Verarbeitung von XML verwandt werden, gleich geregelt. Elemente, die keine Kinder haben, heißen „leer“ (`element2` im Beispiel); für sie gibt es eine verkürzte Schreibweise.

Während SGML viele Möglichkeiten bietet, Ausdrücke zu verkürzen und so Speicherplatz zu sparen, ist XML in dieser Hinsicht einfacher und anspruchsvoller. Weniger anspruchsvoll als SGML ist XML im Hinblick auf sogenannte „kreuzende Strukturen“; ein Beispiel dafür wäre der Versuch, sowohl die physikalische Seite als auch den textlichen Absatz als Grundlage für die Struktur der Annotation zu nehmen. Es kann passieren, daß ein Absatz über eine Seitengrenze hinausläuft, so daß also eine Struktur entsteht, die in Pseudo-XML (oder SGML) so beschrieben werden kann wie in Abb. 4.3 auf der nächsten Seite.

Dennoch ist es auch in XML möglich, Standard-Werte für Attribute zu definieren und so für den „Normalfall“ die Annotation zu verkürzen, da für ihn dann das Attribut nicht mehr gesondert angegeben werden muß.

Damit man zu bestimmten Elementen eindeutigen Bezug herstellen kann, ist es üblich, Kennungen einzuführen, engl. *identity*, die häufig in einem Attribut `id` untergebracht werden; man spricht dann oft von der ID des Elements. XML stellt Möglichkeiten zur Verfügung, die sicherstellen, daß eine ID in einem Dokument jeweils nur einmal vorkommen darf. Es ist jedoch auch üblich, IDs zu verwenden, welche nur in einem lokalen, relevanten Kontext eindeutig sein müssen. In diesem Falle muß man allerdings diese Eindeutigkeit selbst sicherstellen.

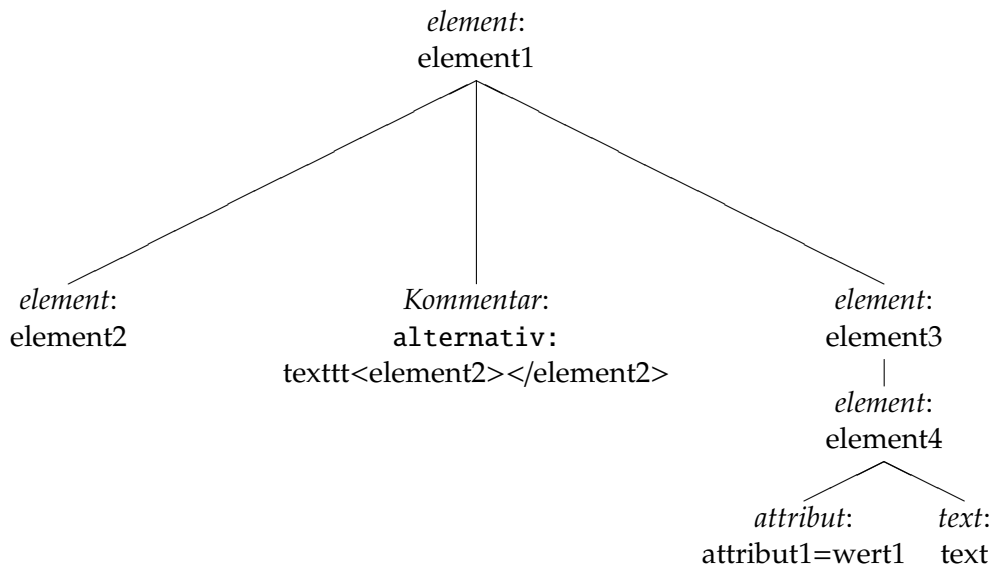


Abb. 4.2: **einfaches XML/SGML-Dokument: eine Baumstruktur**

```
<seite nr="1">
...
<absatz>Dieser Absatz geht über die Seite hinaus und kann daher
</seite>
<seite nr="2">
schlecht in XML annotiert werden.
</absatz>
</seite>
```

Abb. 4.3: **„kreuzende Struktur“**

Es existieren auch weitere Methoden, XML-Dokumente zu verlinken und ineinander einzubetten, auf die wir hier aber nicht weiter eingehen (z.B. XLink, DeRose et al. 2001, XPath, Clark und DeRose 1999, XInclude, Marsh und Orchard 2002).

Es gibt verschiedene XML-Anwendungen, die für mathematische Texte verwandt werden, und zwar sowohl für die Präsentation des Layouts (MathML) als auch für die Darstellung mathematischer Inhalte (OpenMath, OMDoc) oder gar nur für Beweise (HELM). Im folgenden werden diese Ansätze vorgestellt, insoweit sie für ProofML relevant sind.

4.1.2. HELM

HELM (Sacerdoti Coen 1999/2000; Asperti et al. 2003b) ist ein Projekt, das die Infrastruktur für verteilte Datenbanken für mathematisches Wissen zur Verfügung stellen möchte. Dazu sollen möglichst offene Standards wie XML verwandt werden, damit das Wissen, das mit *Proof Assistants* erworben wird, nicht (aus Sicht der Datenverarbeitung) verloren geht, wenn das entsprechende Programm aus der Mode kommt. (So geschehen mit van Benthem-Juttings Automath-System.) Ein Datenaustausch zwischen *Proof Assistants* ist das ein wesentliches Ziel; bisher ist in HELM jedoch vor allem für Coq (The Logical Project), ein konkretes Format für die Darstellung von Beweisen entwickelt worden Sacerdoti Coen (1999/2000 : Kap. 3).

Dieses Format läßt natürlichsprachliche Annotationen für Beweise zu, die in Coq entwickelt wurden; es ist allerdings nicht möglich natürlichsprachliche Beweise in Coq zu verarbeiten.[3]

4.1.3. MathML

MathML (Carlisle et al. 2003) ist eine XML-Anwendung, die einerseits das Layout des traditionellen mathematischen Formelsatzes unterstützt (Präsentationsform), andererseits aber auch für einen fest definierten Satz von Funktionen ein am Inhalt orientiertes Markup ('Content-Markup') definiert. Man kann einer Präsentationsform eine semantische Annotation zuordnen (Carlisle et al. 2003 : Kap. 5.3), indem man beide Formen explizit annotiert. Ein solches Dokument wird allerdings recht schnell unübersichtlich. Markup für die sprachliche Struktur mathematischer Texte bietet MathML nicht an.

[3] Allerdings gibt es ein Modul, das aus Coq-Beweisen natürlichsprachliche Beweise generiert.

Präsentationsform

```
<mrow>
  <msup>
    <mfenced>
      <mrow>
        <mi>a</mi>
        <mo>+</mo>
        <mi>b</mi>
      </mrow>
    </mfenced>
    <mn>2</mn>
  </msup>
</mrow>
```

Content-Markup

```
<mrow>
 <apply>
   <power/>
   <apply>
     <plus/>
     <ci>a</ci>
     <ci>b</ci>
   </apply>
   <cn>2</cn>
 </apply>
</mrow>
```

Abb. 4.4: $(a + b)^2$ **in MathML (Carlisle et al. 2003 : Kap. 2.1.1)**

Den Bereich der Mathematik, den das inhaltsbasierte Markup von MathML abdeckt, bezeichnet Kohlhase (to appear : Kap. 3.1.1) als 'K-12 fragment', da es das Fragment der Mathematik sei, das man vom Kindergarten bis zum Schulabschluß braucht.[4]

Variablen können seit MathML 2.0 nicht nur, falls sie frei sind, sondern auch wenn sie gebunden sind, über einen Pointer zu einer `definitionURL` identifiziert werden.

Es ist theoretisch möglich und wird von Carlisle et al. (2003 : Kap. 4.2.7) vorgeschlagen, die Lücken in MathML zu schließen, indem man für eine semantische Annotation — vor allem eine solche, die das K-12-Fragment sprengt — OpenMath verwendet.

4.1.4. OpenMath

OpenMath (Caprotti et al. 2000). ist gedacht als eine Sprache zur Annotation mathematischer Inhalte, vor allem innerhalb der Algebra. Es soll semiformales Hintergrundwissen in einer maschinenlesbaren Form aufbereitet werden.

Die grundlegende Einheit von OpenMath sind Objekte, welche zwischen Programmen ausgetauscht werden können. Solche Objekte können einfach sein oder komplex. Einfache Objekte ('basic objects') sind grundlegende oder besonders übliche mathematische Objekte: ganze Zahlen, Fließkommazahlen, Zeichenketten, Bytearrays, Symbole und Variablen. Die Struktur von OpenMath-Texten besteht aus Variablenbindungen

[4] Auf deutsch müßte man wohl sagen: K-13- oder besser K-Abi-Fragment, um allen gerecht zu werden.

(<OMBIND>), Funktionsanwendungen (<OMA>), Attributionen (<OMATTR>) und Fehlermeldungen (<OME>).

Variablenbindungen werden so angezeigt, daß der Geltungsbereich einer oder mehrerer Variablen eindeutig abgegrenzt wird (<OMBIND> mit zwei Kindern: einer Variablenliste und dem Geltungsbereich. Eine 'Attribution' (<OMATTR>) kann noch hinzutreten; ihre Funktion ist ähnlich wie die von XML-Attributen. Auf Variablen wird mit einem Namen verwiesen (<OMV name="*var-name*">).

Bei Funktionsanwendungen steht in einem <OMA>-Element der Funktor (üblicherweise ein Symbol *sym* aus einem 'Content Dictionary' *contdic*, annotiert als <OMS cd="*contdic*" name="*sym*">) den Argumenten voran.

Möchte man in OpenMath die Präsentationsform einer Formel darstellen, so kann man in einem <CMP> (*'commented mathematical property'*, Kohlhase to appear : Kap. 3.1.2) zwar LaTeX, MathML etc. verwenden, sollte aber zusätzlich zur Repräsentation des Layouts auch eine Beschreibung der semantischen Struktur in einem <FMP>-Element geben, denn diese ist der eigentliche Gegenstand des OpenMath-„Textes".

4.1.5. OMDoc

OMDoc (Kohlhase to appear)[5] erweitert OpenMath soweit, daß es auch Layout beinhaltet, Elemente zur Strukturierung von Texten enthält und auch vollständig formalisierte Beweise zuläßt. Es scheint möglich, ProofML als eine Erweiterung von OMDoc zu konzipieren.

OMDoc bietet auch die Möglichkeit zur Strukturierung mathematischer Beweise (Kohlhase to appear : Kap. 22) auf verschiedene Art und Weise für menschliche Leser oder für Rechner. Wir werden darauf allerdings nicht näher eingehen, da die vorliegende Arbeit versucht, die Verschränkung der logischen und der sprachlichen Struktur mathematischer Beweise zu annotieren, nicht beide Strukturen nebeneinander zu annotieren.

4.1.5.1. Repräsentation von Beweisen

OMDoc erlaubt zwei verschiedene Herangehensweisen an die Beschreibung von Beweisen: einerseits die Klassifikation mittels beschreibender Rollenbestimmungen, wie

[5] Der ursprüngliche Entwurf ist Kohlhase (2000). Wir beziehen uns, soweit wir über OMDoc sprechen und nichts anderes angegeben ist, auf Anraten des Autors von OMDoc — Michael Kohlhase, *International University Bremen* — immer auf den Entwurf der Version 1.2.

```
<CDDefinition>
  <Name> plus </Name>
  <Description> An binary commutative function plus. </Description>
  <CMP> a + b = b + a </CMP>
  <FMP>
    <OMOBJ>
      <OMBIND>
        <OMS cd="quant1" name="forall"/>
        <OMBVAR>
          <OMV name="a"/>
          <OMV name="b"/>
        </OMBVAR>
        <OMA>
          <OMS cd="relation1" name="eq"/>
          <OMA>
            <OMS cd="arith1" name="plus"/>
            <OMV name="a"/>
            <OMV name="b"/>
          </OMA>
          <OMA>
            <OMS cd="arith1" name="plus"/>
            <OMV name="b"/>
            <OMV name="a"/>
          </OMA>
        </OMA>
      </OMBIND>
    </OMOBJ>
  </FMP>
</CDDefinition>
```

Abb. 4.5: **Ein Beispiel für die Definition eines Operators in OpenMath (Caprotti et al. 2000 : 49)**

Prämissen, Lemmata, Korollare etc.[6] im Modul *MTXT* (für: *'Mathematical Text*, beschrieben in Kohlhase to appear : Kap. 15), andererseits die formalere Kodierung durch die Elemente der Art <derive>, <hypothesis>, <conclude> und <metacomment> aus dem Modul *PF* (*'proof'*, vgl. Kohlhase to appear : Kap. 19).

Eine Verbindung zwischen beiden Annotationen wird nicht hergestellt. Es scheint, als sei die MTXT-Annotation in einem gewissen Maße eher am Layout der Texte orientiert, so enthalten die Annotations-Beispiele des Kapitels 15 auch wenige Beispiele für <assumption>, <formula> etc., sondern demonstrieren die Möglichkeiten zur Hervorhebung von Text (<highlight>) oder zur multilingualen Gestaltung von Texten (mehrere <CMP xml:lang="XX">, wo XX für einen Sprachen-Code steht). Am Ende von Kap. 15 weist Kohlhase (to appear) auch darauf hin, daß die funktionale Rolle eines Textes — außer für Definitionen und Axiome — sehr stark beweistheoretisch und pragmatisch (d.h. hier wohl: durch den konkreten Kontext) bedingt ist:

> Note that the differences between all but the first two are largely pragmatic or proof-theoretic (conjectures become theorems once there is a proof). Kohlhase (to appear : Ende Kap.15)

Insbesondere ist die Strukturierung von Aussagen, die einen Beweis in mehreren Schritten beschreiben mit dem OMDoc-Modul MTXT nicht möglich.

Die Annotation von Beweisen in OMDoc mit den Elementen aus dem Modul PF ist soweit entwickelt, daß sie für die Codierung der logischen Struktur mathematischer Beweise, die verschieden stark formalisiert sind, geeignet ist.[7] Solche Beweise beschreiben dann einen gerichteten azyklischen Graphen.

Der Aufbau eines Beweises in OMDoc geschieht folgendermaßen: Zunächst wird eine Behauptung aufgestellt (<assertion>), daran schließt sich ein Beweis (<proof>) an. Innerhalb dieses Beweises sind <metacomment>-Elemente möglich, also Aussagen über den Ablauf des Beweises selbst; der Hauptteil des Beweises besteht allerdings aus Ableitungsschritten (<derive>). Grundlage für einen Ableitungsschritt kann unter anderem eine Annahme (<hypothesis>) sein, die im entsprechenden Schritt verbraucht wird (der Ort wird im Wert des Attributs discharged-in angegeben). Darüber hinaus ist

[6] Kohlhase (to appear : Kap 15.2) nennt: 'axiom', 'definition', 'example', 'theorem', 'proposition', 'lemma', 'corollary', 'postulate', 'conjecture', 'false-conjecture', 'obligation', 'assumption', 'formula'.

[7] Für einen Versuch aus HELM nach OMDoc zu übersetzen s. Asperti et al. (2003a). Die dort angesprochene Schwierigkeit, den Geltungsbereich einer Hypothese einzuschränken spielt für unsere Annotation keine Rolle; sie soll außerdem spätestens in der nächsten Version von OMDoc behoben sein.

es möglich, in einem Ableitungsschritt anzugeben, welche Prämissen verwandt werden (<premise xref=Prämissen-ID">) und welche Schlußregel zum Einsatz kommt (<method xref="Regel-ID"> *Parameter für die Regel* </method>).

4.1.5.2. Hintergrundwissen

OMDoc erlaubt, Theorien auf anderen Theorien aufzubauen, aber auch, Theorien miteinander zu verbinden (Kohlhase to appear : Kap. 16, 17).

Insbesondere ist auch eine Klassifikation von Aussagen danach möglich, inwiefern sie für eine Theorie grundlegend sind (Kohlhase to appear : Kap. 16) und es kann eine Hierarchie von Theorien angelegt werden (Kohlhase to appear : Kap. 17).

4.1.5.3. Strukturteilung und Anaphern

OMDoc erlaubt Strukturteilung ('structure sharing') zwischen Elementen; d.h. es ist semantisch äquivalent, ob eine Strukturteilungsangabe an einer Stelle steht oder der entsprechende XML-Code.

Die Verwendung von Strukturteilung als Repräsentation etwa für sprachliche Anaphern scheint uns nicht sinnvoll, da sie den entscheidenden Punkt einer Anapher, nämlich die Tatsache, daß sie nur verweist und nicht etwas Bekanntes wiederholt, nicht wiedergibt.

Die Strukturteilung ist für natürlichsprachliche Texte aber ggf. dort nützlich, wo tatsächlich gleiche sprachliche Strukturen vorliegen, die auch eine gleiche Semantik besitzen.[8] Ein Beispiel wären komplexe Namen, ggf. mit einem Determinierer und einer Kategorienbestimmung, also etwa «das Dreieck ABC» oder die Wiederholung des Demonstrandums nach Abschluß des Beweises.

4.1.5.4. Zuordnung von Semantik und Formel

OMDoc erlaubt, sofern man das Modul MT verwendet, einem Ausdruck in natürlicher Sprache eine mathematisch-theoretische Semantik zuzuordnen. In den Beispielen ist dies allerdings nur für Propositionen erfolgt.

[8] Daß eine bloße Übereinstimmung in der sprachlichen Oberflächenstruktur nicht ausreicht, ergibt sich von selbst, wenn man etwa oberflächlich gleiche anaphorische Ausdrücke betrachtet, die verschiedene Referenz haben.

Eine Zergliederung auf niedrigerer Ebene mit den Mitteln von OMDoc wäre sicherlich möglich, sofern man sich auf die grundsätzliche Beschreibung aller sprachlicher Zusammenhänge als Funktionsapplikation einläßt;[9] allerdings würde man außerdem den Unterschied zwischen mathematischer Funktionsapplikation und sprachlicher Komposition aufgeben müssen, da sie durch das gleiche Mittel repräsentiert würde.

4.1.5.5. Rhetorische Relationen

OMDoc sieht noch nicht vor, daß eine linguistische Analyse der OMDoc-Texte vorgenommen wird.

Bis OMDoc 1.0 gab es aber einen Versuch, rhetorische Relationen in der Art der *Rhetorical Strucure Theory* (für eine Einführung Mann 1999) zu annotieren. Allerdings ist diese Unterstützung mittlerweile wieder aus der Sprache entfernt worden (Kohlhase to appear erwähnt sie nicht mehr), denn, wie Michael Kohlhase in einer Email erklärte „die RST-Relationen sind nie richtig unterstützt oder verwendet worden". Dazu paßt, daß auch in der Dokumentation zu OMDoc 1.0 die RST selten erwähnt wird (v.a. Kohlhase 2000 : 10,14,41) und es wird keine Liste der Relationen angegeben.

4.1.5.6. Layout

OMDoc erlaubt, die typographische Struktur von Texten, also die Einteilung in Absätze auf einem relativ einfachen Niveau etc. zu annotieren (Kohlhase to appear : Kap. 21); außerdem besteht die Möglichkeit, für eine semantische Annotation von Formeln anzugeben, wie sie präsentiert werden sollen (Kohlhase to appear : Kap. 22). Auch die typographische Strukturierung eines Textes zu untersuchen, würde jedoch den Rahmen der Arbeit sprengen und ein deutlich größeres Korpus von Beweistexten verlangen. Auf einen Nutzen der Stylesheet-gesteuerten Präsentation von Formeln für das Anliegen von ProofML werden wir aber in Kap. 8.6.1 auf Seite 79 kurz eingehen.

4.1.6. Unsere Anforderungen an eine Annotationssprache

Obwohl OMDoc deutlich näher an dem ist, was für uns eine geeignete Struktur für eine Annotation ist, erfüllt es noch nicht alle unsere Anforderungen.

[9] Dies tut etwa die typenlogische Kategorialgrammatik, vgl. z.B. Morril 1994; Moortgat 1996; mit besonderem Bezug zur Mathematik: Ranta 1999.

4.1.6.1. Linguistische Erscheinungen

Für unsere Annotation halten wir aber folgende linguistisch-motivierte Zuordnungen für sinnvoll:

Einführung von Diskursreferenten Während es für ZINN (to appear : 101ff) wichtig ist, in seinem System erkennen zu können, wann und wie ein Diskursreferent eingeführt wird und wo ihm eine mathematische Kategorie zugeordnet wird, ist dies für die Annotation keine Schwierigkeit: Es reicht, ein Objekt bei seiner Einführung eindeutig zu kennzeichnen und zukünftig über diese Kennzeichnung darauf zu referieren.

Anaphern Bezüge innerhalb des Textes sollten erkennbar sein. Vor allem bei Pronomina sollte aus den Annotationen hervorgehen, auf welche Entitäten sie verweisen. Besonders schwierig sind natürlich Fälle sogenannter „gespaltener Anaphern".

Aber auch bei definiten Nominalphrasen sollte der Bezug klar sein.

'Bridging' Erkennbar sein sollte auch, wenn ein Element aufgrund von *Bridging* verfügbar ist.

Viele Diskursreferenten sind aufgrund bestimmter Eigenschaften schon eingeführter Referenten bereits implizit verfügbar. Diese Gruppe nennt PRINCE (1981) 'Inferrables', in der formalen Semantik bezeichnet man den Schluß auf die Existenz des entsprechenden Referenten als *Bridging*. Das Konzept von Bridging in dieser Arbeit ist möglichst allgemein gehalten, damit möglichst alle problematischen Fälle abgedeckt werden können.

Ein ganz einfaches Beispiel aus der Geometrie ist die Benennung von Vielecken:

(7) a. das Dreieck ABC
b. der Punkt A

Der Punkt A ist ist miteingeführt und gleich benannt, wenn das Dreieck einmal als ABC eingeführt und benannt ist.

Ebenso sind aber auch alle Eigenschaften von Dreiecken verfügbar und alle daraus entstehenden Referenzpunkte — etwa der Schnittpunkt der Mittelsenkrechten der Seiten, und auch die Tatsache, daß alle Seiten überhaupt Mittelsenkrechten haben!

Der Grund dafür, daß der Punkt verfügbar ist, sollte aus der Annotation erkennbar sein.

Kontextgebundende Ausdrücke Ähnliches gilt natürlich für Ausdrücke, die kontextgebunden sind, etwa (8)

(8) die gegenüberliegenden Seiten

Plurale Man unterscheidet in der Linguistik zwischen einer *distributiven* Verwendung des Plurals wie in (9-a), die sich durch konjungungierte Sätze wie (9-b) umschreiben läßt und einer *kollektiven* wie in (10-a), bei dem die Paraphrase (10-b) sicherlich nicht die intendierte Bedeutung wiedergibt. Auf die Wichtigkeit dieser Unterscheidung weisen bereits Koepke und Schröder (2003) hin; die konkrete Umsetzung in ProofML unterscheidet sich ein wenig von der dort (Koepke und Schröder 2003 : 34) vorgeschlagenen.

(9) a. Die Dreiecke *ABC* und *DEF* sind gleichschenklig.
b. Das Dreieck *ABC* ist gleichschenklig und das Dreieck *DEF* ist gleichschenklig.

(10) a. Die Dreiecke *ABC* und *DEF* sind gleich groß.
b. Das Dreieck *ABC* ist gleich groß und das Dreieck *DEF* ist gleich groß.

Bindung Schwierig ist, bei natürlichsprachliche Ausdrücke, die „Diskursreferenten“ einführen, einen Geltungsbereich für die eingeführten Diskursreferenten zu bestimmen.

Eindeutig ist, daß die Einführung eines Elementes des gleichen Namens einen globalen Diskursreferenten verdecken kann. Fraglich wäre, ob diese Verdeckung aufzuheben ist, etwa mit „im allgemeinen“ etc., oder ob zumindest die Variablenzuweisung mit einem Stapel zu modellieren wäre.[10]

Relationen zwischen Sätzen Nicht nur zwischen einzelnen mathematischen Entitäten oder zwischen „atomaren“ Bestandteilen der Sprache bestehen Beziehungen, die für das Verständnis von Texten wichtig sind. Auch zwischen Sätzen bestehen solche annotierenswerten Beziehungen. Wichtig sind für uns vor allem jene Relationen, die in der Annotation von OMDoc noch nicht erfaßt werden. Aus der Orientierung von

[10] Groenendijk und Stokhof (1991 : 45) verwenden ganz offensichtlich einen „flachen“ Variablenraum, in dem eine erneute Einführung eines Diskursreferenten einen alten gleichen Namens zerstört.

OMDoc ergibt sich, daß es sich um diejenigen Relationen zwischen Sätzen handelt, die nicht relevant sind, um die Struktur des Beweises in einem mathematischen Kalkül beschreiben zu können. Wir nehmen zunächst an, daß es sich um folgende Relationen handelt:

Wiederholung: Ein Satz wiederholt inhaltlich oder wörtlich einen anderen.

Parallele/Analogie: Zwischen zwei Sätzen besteht eine Analogie oder eine Parallele, die entweder explizit angegeben wird oder implizit für das Verständnis des Textes wichtig ist.

Kontradiktion Zwei Sätze widersprechen einander; der eine ist die Negation des anderen.

4.1.6.2. Lücken

Im mathematischen natürlichsprachlichen Text treten „Löcher“ auf, das heißt, herleitende Schritte fehlen, da sie — für die Zielgruppe — „trivial“ sind. Wir haben es hier also mit kontextuell bedingter Ausdrucksweise zu tun.[11]

Die Annotation solcher Lücken in einer Art und Weise, die eine möglichst vollständige logische Formalisierung angibt, nennen Koepke und Schröder (2003) bereits als eine vordringliche Aufgabe einer Annotationssprache für mathematische Beweise.

4.2. Verarbeitung formaler Beweise

Mathematisch und logisch vollständig und explizit formalisierte Beweise sind maschinell sehr leicht zu überprüfen.

Als ein frühes System für die maschinelle Prüfung mathematischer Beweise kann der *Proofchecker* von Abrahams (1964) gelten, bei dessen Entwicklung sich der Autor anfangs noch das Ziel gesetzt hatte, natürlichsprachliche Beweise aus Lehrbüchern (*'textbook proofs'*, Abrahams 1964 : 167) zu verifizieren, aber sich bald darauf verlegte, formale Beweise zu überprüfen:

[11] Versuche, Beweise zu formalisieren, weisen darauf hin, daß nicht ganz eindeutig ist, welches Wissen weggelassen wurde, sondern daß die Annahme eines Beweises als richtig auch mit mathematischer Intuition zu tun hat.

> If a computer were to check a textbook proof verbatim, it would require far more intelligence than is possible with the present state of the programming art. (Abrahams 1964 : 167f)

Umwandlung solcher natürlichsprachlicher Beweise bedeutet für Abrahams (1964 : 168), eine LISP-Implementation für eine formallogische Sprache zu entwickeln.

Es gibt viele Systeme, die sich Proof Assistants (Beweis-Assistenten) oder *Proof Checker* nennen. Sie verwenden meist eine formale Sprache zur Spezifikation der Beweise.

Spätere Beweis-Assistenten erweitern das Konzept von „Proofchecker" um Automatismen, die helfen, Beweise zu finden.[12] Es handelt sich dabei um sogenannte tactics (Beweisschritte) und tacticals, Beweisregeln höherer Ordnung, die es letztlich erlauben, ein wenig von der Kreativität in den maschinellen Prozeß der Beweissuche zu integrieren, die den menschlichen Denker auszeichnet.[13]

Was ist nun interessant an den *Proof Assistants* für unsere Arbeit? Sicherlich nicht das Vokabular der Beweis-„Texte" selbst, denn es ist — mit voller Absicht — allenfalls an die natürliche Sprache angelehnt.

Interessant hingegen ist die Strukturierung mathematischer Beweise. So baut die Strukturierung von Beweisen in OMDoc (vgl. Kap. 4.1.5 auf Seite 28 in einem gewissem Maße auf derjenigen von *Proof Assistants* auf. Auch Hintergrundwissen wird in Beweis-Assistenten repräsentiert. In Omega (Siekmann et al. 1998) werden auch domänenspezifische Beweispläne (sehr abstrakte Regeln, Beweise zu führen) angenommen; allerdings ist die Annotation in ProofML, wie wir sie hier präsentieren, auf einer deutlich niedrigeren Abstraktionsstufe.

4.3. Verarbeitung natürlicher Sprache: Zinns vip

Zinn (2003, to appear) stellt vip vor, ein System, das Beweise in natürlicher Sprache „versteht" und überprüfen kann.

Das System besteht aus drei Modulen (Parser, *Discourse Update Engine* mit *Proof Planner* und *Method Expander*) und einer allgemeinen Wissensbasis.

[12] Dies wird dem Begriff von Computer-Assisted Proving eher gerecht.

[13] Zinn (to appear : 20f) weist etwa darauf hin, daß kreative Mathematiker verschiedene Beweismethoden kreativ kombinieren. Wiewohl ein solches Vorgehen maschinell noch nicht simuliert werden kann, können solche Kombinationen im nachhinein als zusätzliche Beweistechniken integriert werden, sofern das System die Hinzufügung von Beweisregeln erlaubt.

Ein Satz wird zunächst immer vom Parser zerlegt, anschließend versucht die *Discourse Update Engine* ihn in die bisherige Repräsentation des Beweisverlaufes zu integrieren. Sollte dies nicht ohne weiteres möglich sein, werden *Proof Plans* zu Rate gezogen. Diese enthalten gewissermaßen Schluß- und Argumentationsfiguren, die vorgeben, wie ein Beweis fortgeführt werden kann. Es können parallel mehrere Repräsentationen für einen Beweis aufgebaut werden; sobald Ungereimtheiten auftreten, werden sie nach und nach ausgeschlossen. Ist ein Beweis erfolgreich verarbeitet, wird anschließend der Beweisplan soweit explizit gemacht, daß er einem Beweis in einem logischen Kalkül des Natürlichen Schließens (Gentzen 1934/35, Jaśkowski 1934) entspricht.

Die Einbettung in den Kalkül des Natürlichen Schließens ermöglicht es, eine klassische Form des Beweises in natürlicher Sprache mit einer fast ebenso klassischen Form des formalen Beweises zu verbinden. Allerdings ist offensichtlich, daß sich eine Darstellung eines Beweises im Kalkül des Natürlichen Schließens nicht für die Annotation mathematischer Beweise eignet. Schwierigkeiten treten im Bereich der Quantoren, der Kompositionalität überhaupt und der Anaphern auf.

Eines der Ziele unserer Annotation ist, daß die Struktur des natürlichsprachlichen Beweises durch die Annotation zwar um formal-logische Information angereichert wird; grundsätzlich verändert werden soll sie aber nicht.

Da dem Kalkül die klassische Prädikatenlogik zugrunde liegt, zeigen sich natürlich genau diejenigen Schwierigkeiten mit der Umsetzung des Skopus eines natürlichsprachlichen Quantors, die Groenendijk und Stokhof (1991) dazu veranlaßt haben, die Dynamische Prädikatenlogik zu entwickeln (vgl. Kap. 3.3 auf Seite 18). Da Zinn das Ziel verfolgt, mathematische Beweise zu verstehen, aber nicht, dieses Verständnis an den ursprünglichen Text rückzukoppeln, stellt die eingeschränkte Verwendung der Quantoren für ihn kein Problem dar. Eine Annotation, die der Form der natürlichen Sprache angemessen sein soll, ist aber unseres Erachtens nicht nur auf eine kompositionale Analyse angewiesen, sondern auch darauf, daß die resultierende Repräsentation kompositional ist und daß sie möglichst gut kompositional darstellbar ist.

In ProofML fehlt nämlich der prozedurale Aspekt von vip, da die Annotation zunächst manuell erfolgt. Obwohl also die formal-logische Repräsentation, die vip erstellt, für unsere Zwecke nicht geeignet ist, können wir doch einiges vom Konzept der *Proof Representation Structures* (PRS) übernehmen, deren Name bereits darauf hinweist, daß es sich um eine den Erfordernissen der Repräsentation eines mathematischen Beweises angepaßter Version der DRSen erinnert, wie wir sie aus der DRT kennen.

PRSen unterscheiden sich von DRSen dadurch, daß statt der Verschachtelungen von

Boxen Numerierungen verwandt werden[14] und daß es möglich ist, Aussagen danach zu klassifizieren, ob sie Annahmen oder Konsequenzen sind, oder ob sie noch bewiesen werden müssen (*'goal'*). Eine ähnliche Klassifizierung verwendet OMDoc und damit ProofML (vgl. Kap. 4.1.5 auf Seite 28, Kap. 9 auf Seite 89). Außerdem können in PRSen auch — wie in anderen Varianten der DRS— Sätze als Diskursreferenten fungieren und anaphorisch aufgegriffen werden.

In ProofML sollte allerdings direkt erkennbar sein, welche Schritte des Beweises explizit ausgeführt wurden und welche offengelassen wurden oder gar vergessen wurden; während dies für ein textverstehendes System wie vip nicht so wichtig ist, ist gerade die Untersuchung der Lücken ein Hauptinteresse der weiteren Erforschung von „Beweisformaten" im entsprechenden Teilprojekt der „Wissensformate".

4.4. Ein Mittelweg: Kontrollierte Sprache

Wenn man dem Computer nicht ganz zutraut, die menschliche Sprache zu verarbeiten, aber einen reinen Formalismus als eine zu bittere Medizin (van Benthem 2003 : Kap. 2) empfindet, kann man deren Bitterkeit durch syntaktischen Zucker zu vermindern suchen. Auch dieser Weg ist beschritten worden.

Für die Entwicklung von ProofML ist natürlich vor allem interessant, was wir aus diesen Ansätzen in ProofML übernehmen können.

4.4.1. Student

Bobrow (1968) stellt Student vor, ein mittlerweile klassisches Systemvor, (Bobrow 1968 : 135), von dem er sagt: '[it] accepts input in a comfortable but restricted subset of English'. In dieser Untermenge des Englischen können mathematische Textaufgaben beschrieben sein, die zu linearen Gleichungssystemen führen. Das System kann diese Gleichungssysteme extrahieren und lösen. Hier begegnen wir also dem Versuch, zur Beschreibung eines Sachverhaltes eine „kontrollierte Sprache" zu verwenden, wie

[14] Dies ist durchaus keine unübliche Variation: Jaśkowski (1934) (vgl. Pelletier 2000 für eine Übersicht über dieses und ähnliche Systeme) entwickelt zwei verschiedene Varianten des Natürlichen Schließens: eine grafische, die auf der Verschachtelungen von Boxen besteht, eine andere, die auf der Numerierung der Formeln basiert.

Auch die von Lamport (1993) vorgeschlagene Methode, Beweise zu schreiben, verwendet ebenfalls Numerierungen zur Gliederung und Markierung der Zugänglichkeit unter den Aussagen.

es auch in manchen Firmen üblich ist, wenn Texte (etwa Betriebsanleitungen) für die maschinelle Übersetzung natürlicher Sprache vorbereitet werden sollen.

Das Hintergrundwissen ist in STUDENT offensichtlich „festverdrahtet" und ist nur für eine Art Textaufgabe geeignet, von Beweisen kann keine Rede sein. Insofern belassen wir es bei dieser historischen Fußnote.

4.4.2. MIZAR

MIZAR (Mizar Projekt; RUDNICKI 1992) stellt mathematische Beweise ebenfalls in einer Art kontrollierter Sprache dar. Auch diese Sprache kann man in einem gewissen Maße als *'comfortable but restricted subset of English'* (BOBROW 1968 : 135) sehen. In Bezug auf die Natürlichkeit der Sprache ist allerdings der Komfort etwas stärker eingeschränkt als bei STUDENT; dafür ist der Anwendungsbereich von MIZAR deutlich weiter: Nach dem Willen der Autoren soll MIZAR dazu dienen, eine Bibliothek mathematischer Beweise aufzubauen, die die gesamte Mathematik abdeckt.[15]

Im Vergleich zu den rein formal-logischen *Proof Assistants* ist MIZAR nicht darauf ausgelegt, dem Menschen, der einen Beweis entwickelt, bei der Entwicklung des Beweises zur Seite zu stehen. Nutzer sollen vielmehr den Beweis schreiben, wie sie es ohne Rechner auch täten; lediglich deutlich formaler. Hilfssätze und Vorwissen werden im *environment* erfaßt, also der „Umgebung", dem mathematischen Kontext, in dem ein Beweis steht. Dieser Kontext umfaßt einige Sätze, Definitionen von Konstanten und Hintergrundwissen.

Die Bereitstellung dieses Kontextes zeigt durchaus Parallelen dazu, wie Vorwissen in informellen Beweisen verwandt wird; genauso wie im informellen Beweis auf Sätze Bezug genommen wird, ohne — normalerweise — den Beweis für ihre Gültigkeit noch einmal zu nennen oder gar zu führen, wird in MIZAR aus dem formalen Beweis ein *Abstract* erstellt, das die „öffentlichen" Sätze eines Beweiskonvoluts enthält.[16]

Interessant ist auch, daß MIZAR eine geringe Zahl vordefinierter Möglichkeiten zu Rückbezügen bereitstellt; diese unterscheiden sich natürlich insofern von natürlichsprachlichen Anaphern als der Bezug hier eindeutig definiert ist.

[15] SACERDOTI COEN (1999/2000 : 9) meldet Zweifel bezüglich der Korrektheit des MIZAR-Systems an, da nicht nur der Code nicht frei zugänglich ist, sondern die vorhandene Dokumentation recht spärlich ausfällt.

[16] Leider teilt MIZAR auch eine andere Eigenschaft der natürlichsprachlichen mathematischen Beweise: die große Freiheit für die Beweisautoren und damit verbunden einen gewissen Wildwuchs in der Beweisbibliothek, der bedeutet, daß manche Sätze mehrfach, gelegentlich auf verschiedene Weise in der Bibliothek vorhanden sind. Zu den etwas unbequemen Konsequenzen vgl. RUDNICKI und TRYBULEC (1996).

Übrigens schließen Mizar-ähnliche Sprachen und *Computer-Assisted Proving* — also computerseitige Hilfestellung beim Beweisen — einander nicht aus: es gibt einige Ansätze, Mizars Sprache in andere Proof Assistants zu integrieren (vgl. Harrison 1996).

4.4.3. Zum Begriff des 'Mathematical Vernacular'

Von NG de Bruijn stammt der Ausdruck des *'mathematical vernacular'*.

Er wird offensichtlich verschieden gebraucht: zunächst bezeichnet er gelegentlich die Sprache der Mathematiker, wie sie sie gebrauchen. Manchmal scheint der Begriff nur die Sprache zu bezeichnen, die die Mathematiker gebrauchen *sollten*.

Zum dritten kann *'mathematical vernacular'* auch etwas sein, das es noch zu erstellen gilt, eine Art vereinfachtes Esperanto für Mathematiker: natürlicher Sprache ähnlich, aber möglichst ohne Ambiguität; vor allem im letzteren Punkt formalen Beweiskalkülen ähnlich, aber flexibler als sie und in der Struktur der natürlichen Sprache näher. Unter diese letztere Kategorie fällt in gewissem Maße, was Ranta (1999, 1994) als Repräsentationssystem für die Sprache der Mathematik vorschlägt (vgl. Kap. 4.5) und die *Weak Type Theory* (Nederpelt 2002), auf die wir hier nicht näher eingehen können.

4.5. Eine Interlingua: Rantas Grammatik der mathematischen Sprache

Ein sehr interessanter Ansatz in der Betrachtung der Sprache der Mathematiker ist zu finden in den Artikeln von Ranta (1999, 1996, 1995, 1994).

Es ist nicht ganz einfach, Rantas Ansatzpunkt einzuordnen. Einerseits spricht er davon, mathematische Sprache analysieren zu wollen und zieht Vergleiche zwischen der mathematischen Sprache und dem Repräsentationsformalismus, den er gebraucht: eine besondere Form der Typentheorie (die 'constructive type theory' von Per Marin-Löf), die vermittels einer Curry-Howard-van-Benthem-Korrespondenz an eine semantische Repräsentation gekoppelt wird. Andererseits wird zwar in den Texten dargestellt, wie bestimmte natürlich-sprachliche Phänomene in Rantas System repräsentiert werden können, aber eine regelrechte Parsing-Strategie wird nicht angegeben, sondern nur an einer Stelle skizziert und als eher nebensächlich abgetan.

Vielmehr scheint die Repräsentation primär zu sein. Ranta versteht sie offensichtlich als Ersatz für die unvollkommenen Ansätze Freges und Montagues, die Semantik na-

```
# formalisiert und verifiziert von Patrick Braselmann, Uni Bonn, 2003

environ # Umgebung / Hintergrundwissen
 vocabulary RELAT_1, FUNCT_2, ...;   notation TARSKI, XBOOLE_0, ...;
 constructors QC_LANG3, DOMAIN_1, ...; clusters NAT_1, ...
 requirements ARYTM, NUMERALS, SUBSET, BOOLE; theorems AXIOMS, TARSKI, ...;
 definitions TARSKI, XBOOLE_0,...;   schemes FUNCT_1, FINSEQ_1,...;

begin
 reserve a for set; reserve Y,Z for Subset of CQC-WFF;
 #...
 theorem
    Th1:  X |- p implies X |= p by CORRECT:1;
 definition let Z;
   attr Z is consistent means
   :Def5: for p holds not (Z |- p & Z |- 'not' p);
 end;
 definition let Z;
   attr Z is inconsistent means
   :Def6: Z is not consistent;
 end;

 theorem
   Th2: Z is inconsistent iff for p holds Z |- p
   proof
     thus Z is inconsistent implies for p holds Z |- p
     proof
       assume Z is inconsistent; then Z is not consistent by Def6;
       then consider q such that
A1:    (Z |- q & Z |- 'not' q) by Def5; # benannte Aussage
       let p;
A2:    q in Cn(Z) by A1,CQC_THE1:def 9;
A3:    'not' q in Cn(Z) by A1,CQC_THE1:def 9;
       q => ('not' q => p) in Cn(Z) by CQC_THE1:29;
       then  ('not' q => p) in Cn(Z) by A2,CQC_THE1:32;
       then p in Cn(Z) by A3,CQC_THE1:32; hence thesis by CQC_THE1:def 9;
     end;
    thus (for p holds Z |- p) implies Z is inconsistent
    proof
      assume for p holds Z |- p; then Z |- VERUM & Z |- 'not' VERUM;
      then not Z is consistent by Def5;  hence thesis by Def6;
end; end;
```

Abb. 4.6: **Beispiel für Mizar-Code (gekürzt): «(a) ergibt sich unmittelbar aus (b)» aus Lemma 7.2 aus dem Korpus (s. Anhang)**

türlicher Sprache formal darzustellen. In seinem Aufsatz zum *Grammatical Framework* (Ranta 2002), das die Weiterentwicklung seiner Ideen beschreibt, die er mit besonderer Rücksicht auf die mathematischen Texte entwickelt hat, schlägt Ranta eben dieses 'Framework' als eine Interlingua vor, die zum *multilingual editing* dienen soll; ein Benutzer könne also in der abstrakten Repräsentation einen Text schreiben, der über entsprechende Grammatikregeln in einer anderen Sprache ebenfalls syntaktisch korrekt ausgegeben werden kann.[17]

In einem Aufsatz zum Thema der mathematischen Sprache (Ranta 1996) stellt er ein System vor, das es erlaubt, mathematische Texte in einem Proof-Assistant (ALF) zu erstellen und dabei gleichzeitig die mathematische und die grammatische Korrektheit zu gewährleisten.[18]

Mehrsprachigkeit Aus Rantas Aufsätzen können wir die Idee übernehmen, daß die semantische Struktur der Texte für alle Sprachen anpaßbar sein soll. Daß sie für unsere Zwecke nicht ganz so abstrakt sein kann, wie sie es bei Ranta ist, ist offensichtlich; es geht immerhin darum, die syntaktische Struktur und die semantische Struktur auf einer Ebene zu annotieren. Wir brauchen also keine Linearisierungsregeln. Außerdem sollten notwendige syntaktische Merkmale auch in der Annotation vorhanden sein. Im Unterschied zu Rantas typenlogischer Interlingua wird allerdings in ProofML die Annotation nicht für alle Sprachen gleich sein, denn vor allem in der Annotation grammatischer Merkmale unterscheiden die Sprachen sich zu sehr.

Kontextabhängige Kategorien / 'Dependent Types' Ranta (2002 : Kap. 4) erklärt 'Dependent Types' als:

> *Dependent Types* are types that depend on objects. Ranta (2002 : 14)

[17] Auf das „Interlinguaproblem", also die Schwierigkeit eine allgemeine Interlingua für alle Sprachen zu definieren, geht Ranta nicht ein. Dies mag damit zusammenhängen, daß er das 'Framework' vornehmlich zur Verarbeitung mathematischer Sprache eingesetzt hat und in dieser die spezifisch mathematischen Konzepte soweit definiert sind, daß sie austauschbar sind.

[18] Interessant ist, daß Ranta einerseits die bekannten Sätze von Geach ohne Probleme behandelt, andererseits „nicht genau weiß" (Ranta 1994 : Kap. 8), welche Rolle das Wörtchen 'all' in (11) spielt, obwohl er soeben den Unterschied zwischen kollektiven und distributiven Pluralen erklärt hat. Leider sagt Ranta nicht genau, worin seine Schwierigkeiten bestehen. Da er bis zu diesem Punkt in seinem Text bereits kollektive Plurale besprochen hat, könnte das Problem darin liegen, daß das erste Argument von *all* das durch den Relativsatz modifizierte Substantiv ist und daß im Relativsatz die Verwendung des Plurals eindeutig distributiv ist, wohingegen sie im Hauptsatz kollektiv ist.

(11) All lines that pass through the center of a circle converge.

Vermittels dieser Typen ist es im 'Grammatical Framework' möglich, es nur dann zuzulassen über bestimmte Dinge zu sprechen, wenn andere Dinge vorausgesetzt werden können und genau dies bewiesen werden kann. Ein unmathematisches Beispiel von (Ranta 2002 : 16) ist die Angabe einer Zugverbindung mit Umsteigeangaben, die die Existenz und Verträglichkeit der jeweiligen Verbindungen voraussetzen (ohne daß diese in der tatsächlichen Fahrempfehlung erwähnt würden). Auf unseren Bereich angewandt würde dies bedeuten, daß man etwa über den Mittelpunkt der Strecke $\overline{AB}$ nur dann sprechen darf, wenn beweisbar ist, daß es den Punkt A und den Punkt B gibt und daß man zwischen beiden Punkten eine Strecke ziehen kann (d.h. daß sie verschieden sind).

Auf diese Weise können Hintergrundinformationen bei der Beweiserstellung explizit gemacht werden; dies ist in Euklid-Editionen durchaus üblich, dort werden ‚triviale' Konstruktionen wie Strecken und Kreise um einen Punkt durch Axiome gerechtfertigt.

Auch in Rantas Ansatz ist keine Rückkopplung der semantischen Repräsentation und der sprachlichen Realisierung vorgesehen. Zwar kann man mittels GF-Grammatiken theoretisch auch parsen, dies ist jedoch nicht das Ziel bei ihrer Entwicklung gewesen und ist auch nicht ganz einfach (Ranta 2002 : 9ff). Außerdem ist die Entsprechung zwischen „linearisiertem" Text und typenlogischer Interlingua bei Ranta nicht bijektiv — sowohl werden in der Linearisierung bestimmte Ausdrücke unterdrückt (*suppression*) als auch bestimmte Informationen hinzugefügt (*reduplication*); außerdem kann die Reihenfolge der semantischen und der syntaktischen Argumente verschieden sein (*permutation*, vgl. zu den Begriffen Ranta 2002 : 6f).

Der Aufwand, sich einer relativ unbekannten Repräsentation anzuschließen, wäre wahrscheinlich höher als der Gewinn, den man davon hätte.

Bindungen In Rantas Theorie sind Variablen immer explizit Geltungsbereiche zugeordnet; die Bindungen sind also statisch und nicht, wie in ProofML, dynamisch. Für den Zweck der Annotation scheinen uns aber die Vorteile einer dynamischen Variablenbindung zu überwiegen (s. Kap. 3.3 auf Seite 18).

Teil III.

Praxis: Die Annotation

Dieser Teil der Arbeit erläutert den Aufbau von ProofML und illustriert die Anwendung von ProofML an einigen Beispielsätzen, die ergänzt werden durch Beispiele, die wir am Ende der Arbeit anhängen.

ProofML ist eine XML-Anwendung, soviel ist bereits gesagt worden. Die Annotation der Beweise an sich baut auf dem Modul PF von OMDoc auf, wie in Kap. 4.1.5 auf Seite 28 angedeutet; allerdings muß PF um eine Annotation für die sprachliche Dimension der Beweise ergänzt und erweitert werden. Diese Ergänzungen werden mit ihrer Syntax und Semantik in Kap. 8 und 9 beschrieben.

ProofML muß darüberhinaus auf Hintergrundwissen zurückgreifen, wie etwa ein Lexikon und eine mathematische Wissensbank (à la OMDoc oder HELM). Dieses Hintergrundwissen wird abstrakt im folgenden Kapitel beschrieben; da wir uns insgesamt für die Beschreibung der logischen Struktur an OMDoc orientieren, nehmen wir OpenMath-Dictionaries oder OMDoc-Dokumente als Basis des Hintergrundwissens an. Für die Formulierung des Hintergrundwissens bieten wir aber keine Formalisierung an; eine informelle Beschreibung des Hintergrundwissens, das für das erste Beispiel notwendig wäre, findet sich im Anhang.

Formale Syntax-Definitionen sind im fortlaufenden Text schwierig zu präsentieren. Daher findet sich im Anhang eine kommentierte *Document Type Definition* der für ProofMLin Verbindung mit OMDoc, im Text sind dagegen nur informelle Beschreibungen der Elemente gegeben.

5. Die Beispiele

ProofML wurde hauptsächlich an einem kleinen Korpus von Beweisen entwickelt, aber es wurden auch Ergänzungen aus der Sekundärliteratur vorgenommen.

Die Beweise des Beispiel-Korpus stammen einerseits aus den *Elementen* von Euklid, andererseits aus der *Einführung in die mathematische Logik* von Ebbinghaus et al. (1996). Sie decken also zwei sehr unterschiedliche und grundlegende Bereiche der Mathematik ab. Es fällt selbst bei unserem kleinen Korpus auf, daß der Stil beider Bücher recht unterschiedlich ist; so verwendet Euklid viel weniger Formeln, und die Namensgebung ist bei ihm noch nicht so bedeutungstragend wie bei Ebbinghaus et al..

Eine Gemeinsamkeit der beiden verwandten Beispiele liegt darin, daß es sich um einführende Werke handelt, auch wenn man Euklids Werk einen deutlich größeren Einfluß zuschreiben darf.[1] Da sie aus Einführungswerken stammen, sind die präsentierten Beweise recht detailliert; dennoch finden sich bereits in diesen Texten für Neulinge einige Vereinfachungen der Darstellung („Lücken“) in den Beweisen, die für menschliche Leser mehr oder minder offensichtlich sind, aber in einem formalisierten Beweis ergänzt werden müssen. Die Methoden, die wir in dieser Arbeit demonstrieren, werden hoffentlich ausreichen, um auch größere Lücken zu schließen, oder zumindest Anregungen dazu bieten, dies zu erweitern.

[1] Gerade wegen dieser Bedeutung haben wir die emphElemente ausgewählt. Daß die annotierten Beispiele der englischen Ausgabe folgen, ist nicht nur ein Zugeständnis an geringe Griechischkenntnisse des Autors, die Lesbarkeit und die Einfachheit der Textein- und vor allem -ausgabe; es steht wohl ohnehin zu erwarten, daß im weiteren Gebrauch von ProofML eher englische als altgriechische Texte annotiert werden. Die englische Übersetzung haben wir auch deshalb ausgewählt, weil sie deutlich näher am griechischen Original bleibt als die deutsche. Darüberhinaus können wir so demonstrieren, daß ProofML für die Annotation von Texten in verschiedenen Sprachen geeignet ist.

6. Hintergrundwissen in ProofML

Wir brauchen zweierlei Hintergrundwissen für unsere Annotation: einerseits Hintergrundwissen über die mathematischen Zusammenhänge – andererseits linguistisches und sprachliches Wissen.

6.1. Das mathematische Hintergrundwissen

Die Notwendigkeit des mathematischen Hintergrundwissens ergibt sich daraus, daß gewisse Inferenzen aus der Einführung bestimmter Diskursreferenten gezogen werden können.

> Im Gegensatz etwa zur Technik ist Normung bei Mathematikern verpönt, so daß — wenigstens im Prinzip — alles, wovon gesprochen wird, erst definiert werden muß, wenn es sich nicht gerade um Dinge handelt, die in gängige Lehrbücher eingegangen sind, oder wenn man sich nicht auf entsprechende Monographien oder sonstige Arbeiten bezieht. Eisenreich (1998 : 1223)

Das mathematische Hintergrundwissen sollte also verfügbar sein, etwa als formalisiertes OMDoc-Dokument.

(12) a. Das Dreieck *ABC*
b. die Seite *AB* ...

Es ist offensichtlich, daß durch die Einführung des Dreiecks *ABC* auch die Seiten *AB*, *BC*, *AC* implizit eingeführt sind, ebenso die Winkel *ABC*, *BCA*, *CAB* etc. (Zum Problem der Namen s. 8.5).

Die Zusammenhänge zwischen der expliziten Einführung einer Entität und der Menge der damit implizit eingeführten Entitäten ist keinesfalls trivial (für einen frühen Versuch einer Klassifikation von Diskurs-Entitäten zwischen „alt“ und „neu“ s. Prince 1981). Wissen über den mathematischen Zusammenhang, den Zusammenhang mit

dem „Weltwissen“, in dem eine Entität steht, kann nicht *ad hoc* erschlossen werden. Es ist daher sinnvoll, schon für die Annotation anzunehmen, daß der mathematische Hintergrund als *Wissensbasis* oder *Ontologie* modelliert ist. Wir möchten hier vorerst keine vollständige Ontologie angeben, nehmen aber an, daß eine solche vorhanden ist und werden auch bei einzelnen Abschnitten Hinweise beschreiben, welche Zusammenhänge sie enthalten muß.

Für unsere Annotation ist vor allem wichtig, daß die Ontologie folgende Informationen enthält:

1. Definitionen der mathematischen Kategorien und Relationen, wie sie Euklid unter anderem zu Beginn des ersten Buches der Elemente vornimmt.
2. Informationen über die Bedeutung, die in der Struktur von Namen liegt.[1]
3. Informationen, die Entitäten verbinden: mereologische Beziehungen, *Bridging*-Relationen etc.
4. Informationen über Stelligkeit und Argumentstruktur der Relationen zwischen Entitäten, aber auch Propositionen.
5. Allgemeine Schlußregeln für das mathematische Schließen.
6. Schlußregeln, die für ein Fachgebiet typisch sind.

6.2. Das Lexikon

Weniger offensichtlich mag scheinen, daß ein Lexikon vonnöten ist. Dennoch scheint es uns ein Erfordernis der Effizienz zu sein, ein solches Lexikon anzunehmen.

Vor allem in stärker flektierenden Sprachen kann ein Wort viele verschiedene Formen annehmen, die grammatisch oder phonologisch bedingt sind.

Es ist nicht unbedingt sinnvoll, alle Wortformen zu erfassen (in agglutinierenden Sprachen kann dies unmöglich sein), da dies zu einem hohen Maß an Redundanz führt und Regularitäten verdeckt, die im Bereich des Wortschatzes und der Flexion bestehen.

Sprachen zur Repräsentation von Lexika, wie etwa DATR (Evans und Gazdar 1996), bieten die Möglichkeit, den Aufbau eines Lexikons mit möglichst wenig Redundanz zu

[1] Wir haben in 8.5 zusammengesetzte Namen als Tupel einfacher Namen analysiert; daraus folgt: wenn der Name des Dreiecks aus den Namen der Eckpunkte besteht, sind auch die Namen der Strecken zwischen diesen Eckpunkten bekannt.

beschreiben. Ähnliche Ansätze verfolgen Grammatiktheorien, etwa die *Kopfgesteuerte Phrasenstrukturgrammatik* (*Head-driven Phrase Structure Grammar*, HPSG), die *lexikalische Regeln* anwenden, um im Lexikon aus einer bestimmten Zahl von Wortformen (oder Stämmen) andere abzuleiten. Zumindest in der HPSG verändern lexikalische Regeln auch die Valenz eines Wortes, beispielsweise bei Passiv-Formen.

Das Lexikon sollte also folgende Informationen enthalten:

- Wörter
  - Lemma, Schreibung
  - Valenz-Rahmen
  - Zuordnung zu Flexionsklassen etc.
- evtl. lexikalische Regeln zur Ableitung bestimmter Wortformen.

Für die Konstruktion von ProofML nehmen wir an, daß ein solches Lexikon besteht und daß in diesem Lexikon eine Abbildung der semantischen Kategorien auf die mathematischen vorliegt.[2]

Wir gehen auf die Struktur des Lexikons und die genaue grammatikalische Analyse in dieser Arbeit aber nicht näher ein, da dies den Rahmen sprengen würde.

Sowohl Zinn (to appear) als auch Baur (1999) verwenden in ihren jeweiligen Programmen zur Verarbeitung natürlichsprachlicher Beweise Parse-Mechanismen und Lexikonstrukturen, die für natürlichsprachliche Texte im allgemeinen entwickelt worden sind.

[2] Die Zuordnung der syntaktischen Kategorien zu den semantischen müßte bei einer Automatisierung von einem Parser übernommen werden; dieser wäre im weitesten Sinne auch als ein Teil des sprachlichen Hintergrundwissens zu betrachten.

7. Standardinterpretationen

Wir können die Syntax von ProofML so definieren, daß sie maximal explizit sein kann. Es fördert allerdings die Lesbarkeit eines Textes, wenn man nicht alle Beziehungen zwischen Elementen und alle Attribut-Werte explizit macht. Damit aber diese Beziehungen wohl eindeutig definiert sind, werden wir einige Standardinterpretationen festlegen, die die Annotation vereinfachen.

Es gibt zwei Arten solcher Festlegungen: auf der syntaktischen Ebene ist es möglich, in der *Document Type Declaration* (DTD) Standardwerte für Attribute festzulegen; inhaltliche Beziehungen zwischen den Elementen sind allerdings nur in der Semantik einer XML-Anwendung definiert, über die die DTD nichts aussagt.

8. Annotation auf phrasaler Ebene

8.1. Allgemeines zur sprachlichen Struktur

Das Ziel der Annotation ist es, wie gesagt, die logische und die sprachlich-syntaktische Struktur deutlich zu machen. In Bezug auf die logische Struktur bietet es sich an, sich an den Beschreibungs-Modellen der formalen Semantik zu orientieren. Diese beruhen (seit Freges „Begriffschrift“, etwa im Band Frege 1998) meist auf der Funktor-Argument-Struktur.

Die syntaktische Beschreibung der sprachlichen Struktur eines Textes beruht hingegen im wesentlichen auf zwei Prinzipien: Beschreibung der *Dependenz*, also der Bestimmung des „Kopfes“ einer Phrase und seiner Argumente — sie ist parallel zur Funktor-Argument-Struktur in der Semantik zu sehen —, und Beschreibung der *Konstituenz* innerhalb einer Phrase, also die Bestimmung und Kategorisierung ihrer Bestandteile.

Konstituenz Es ist offensichtlich, daß bezüglich der Konstituenz für diese Arbeit nur eine an der Oberfläche orientierte Beschreibung möglich ist, da die Annahme einer Tiefenstruktur die Annotation theorielastiger machen würde als nötig.

Die Konstituenz einer Phrase in der Oberflächenstruktur der Sprache ist in XML relativ einfach zu annotieren, sofern man den Bereich der Phrase abgrenzt und auch ihre Teile hierarchisch annotiert. Es ergibt sich, daß die Struktur eines XML-Baumes denjenigen Strukturen entspricht, die mit einer kontextfreien Grammatik erzeugt werden können.

Dependenz Die Annotation der Dependenz-Strukturen gestaltet sich schwieriger.

Als Grundlage der Dependenz-Beschreibungen verwendet ProofML die Funktor-Argument-Struktur.[1] Gelegentlich nennt man den Funktor auch den *(syntaktischen) Kopf*

[1] In der Dependenz-Grammatik werden die Argumente eines Funktors üblicherweise als *actants* (nach Tesnière 1965), dt. Aktanten oder Mitspieler bezeichnet, während der Funktor oft nicht als solcher benannt wird, sondern nach seiner grammatischen Funktion. Man spricht also von „Aktanten des Verbs“.

einer Phrase. In dieser Arbeit ist meist die Rede von Funktor, wenn es nur um die semantische Struktur geht, sonst von Kopf.

Modifikation vs. Ergänzung Wichtig ist in diesem Zusammenhang in manchen Syntax-Theorien der Unterschied zwischen (notwendiger) Ergänzung und (fakultativer) Modifikation. In der Ergänzung werden dem (syntaktischen) Kopf seine Argumente untergeordnet. In der Modifikation wird der syntaktische Kopf durch ein anderes Element näher bestimmt (z.B. ein Substantiv durch ein Adjektiv); gelegentlich wird dieser Modifikator dann als semantischer Kopf bezeichnet.

Wir annotieren der Einfachheit halber immer nur die semantischen Köpfe mit ihren Argumenten; ein Modifikator ist für uns also immer der semantische Kopf der modifizierten Phrase.

8.2. Die Elemente <word> <arg>, <keyword> und <ling>

<keyword> Das Element <keyword> kann Funktionswörter enthalten, die anzeigen, daß die annotierte Phrase zu einer bestimmten Kategorie gehört. In einem <arg>-Element kann zum Beispiel eine Präposition bei einem Präpositionalobjekt ein <keyword> sein (vgl. Abb. 8.2 auf Seite 54). <keyword> kann auch in einem Element vorkommen, daß die logische Struktur beschreibt (Kap. 9 auf Seite 89), etwa in Prämissen.

Dabei gibt es mindestens einen Fall, in dem es vorkommen kann, daß zwei <keyword>-Elemente als Kinder desselben Elements vorkommen, nämlich im Falle zweier Prämissen, die sowohl mit einer koordinierenden Konjunktion verbunden sind als auch durch eine Konjunktion eingeleitet werden, die sie (beide) als Prämissen (Begründungen) ausweist (Abb. 8.1 auf der nächsten Seite), die mit dem Element <nl-premise> annotier werden. Für diesem Fall ist es vorgesehen, daß mindestens ein <keyword>-Element ein Attribut `type`, aus dessen Wert ersichtlich ist, wofür es steht; dies ist entweder der Name des Elements, auf das es hinweist, oder der Wert `conj` für die Konjunktion.

Für den Fall, daß ein Argument zwar durch ein Funktionswort markiert ist, aber es aufgrund der Oberflächenstruktur nicht als Kind seines Funktors annotiert werden kann, kann man mit dem `refid`-Attribut auf seine ID verweisen und gibt <keyword refid="*ID-des-Funktors*" type="arg"> an.

Die Markierung von Argumenten erfolgt prinzipiell durch die Angabe eines Verweises in einem <arg>-Element.

(13) Since DB equals AC and BD is common, the two sides DB and BC equal the two sides AC and CB respectively, and the angle DBC equals the angle ACB.

```
<derive id="pre-I.4">
  <method xref="obvious"/>
  <nl-premise form="conj">
    <keyword>Since</keyword>
    <nl-premise xref="cut-off-DB=AC">
      <expression type="proposition">
        <name refid="DB" type="compound">DB</name>
        <expression type="predicate:equal">
          equals
        </expression>
        <name refid="AC" type="compound">AC</name>,
      </expression>
    </nl-premise>
    <keyword type="conj">and</keyword>
    <nl-premise>
      <expression type="proposition">
        <name refid="BC">BC</name>
        <expression type="predicate:common-list">
          <keyword>is</keyword>
          <word>common</word>
          <arg refid="DBC"/>
          <arg refid="ABC"/>
        </expression>,
      </expression>
    </nl-premise>
  </nl-premise>

  [...]
</derive>
```

Abb. 8.1: **Zwei <keyword>-Elemente**

(14) a. *AB* equals *BC*
b. *AB* is equal to *BC*

```
<!-- Wir müssen hier nicht besonders annotieren, da die Argumente in
  der richtigen Reihenfolge stehen. -->

<expression type="proposition">
  <name type="compound">AB</name>
  <expression type="predicate:equal">equals</expression>
  <name type="compound">BC</name>
</expression>


<!-- Diese Klammerung erlaubt, die Funktion des 'to' deutlich zu
  machen -->

<expression type="proposition">
  <name type="compound" refid="AB">AB</name>
  <expression type="predicate:equal">
    <keyword>is</keyword>
    <word>equal</word>
    <arg refid="AB"/>
    <arg><keyword>to</arg><name type="compound">BC</name></arg>
  </expression>
</expression>
```

Abb. 8.2: **Allerlei Gleichheit**

Das <arg>-Element kann also entweder auf ein Argument verweisen oder ein Argument enthalten (vgl. Abb. 8.2).

<word> In dem Fall, daß einem Wort Argumente zugeordnet werden, würde das Wort als Textknoten im XML-Baum ein Geschwister der <arg>-Elemente sein. Dies ist für die Verarbeitung des XML-Dokuments nicht besonders angenehm. Wir führen daher das Element <word> ein.

Annotationsregel 1 (Das Element <word>) *Das Element <word> hat keine Semantik, es dient dazu, Wörter zu klammern, die andernfalls als Textknoten im* XML*-Baum Geschwister von Elementknoten wären.*

Werden mehrere Wörter als Einheit aufgefaßt (z.B. Phraseologismen), so empfiehlt es sich — gegen die intuitive Bedeutung des Element-Namens —, ein gemeinsames <word>-Element für sie zu verwenden.

<ling> Das Element <ling> enthält Wörter, welche sprachlich vorhanden (und oft für den Fluß des Textes relevant) sind, aber für die formale Semantik keinen wichtigen Beitrag leisten.

Ein Beispiel sind „Füllwörter“ und Gradpartikeln wie „auch“, „aber“ (v.a. als Adverb) oder etwa griechisch „*μέν* ... *δέ*“, die einen Kontrast ausdrücken, der in den wenigsten Fällen wichtig ist, aber auch „und“, wenn es einfach als (sprachliche) Konjunktion auftritt, z.B. in „und daher“.[2]

Es ergibt sich von selbst, daß man die Zahl der <ling>-Elemente so gering halten möchte, wie nur möglich, um möglichst viel von der natürlichsprachlichen Struktur zu erhalten.

8.3. Annotation von Funktor-Argument-Strukturen

Ein <expression>-Element kann Kinder vom Typ <arg/> haben; diese leeren Elemente enthalten im Attribute name einen Verweis auf die <expression>, welche ein semantisches Argument eines Funktors enthält. Gibt es mehrere Argumente, bieten sich hier zwei Möglichkeiten, die Argumente eines Funktors zu identifizieren: über die Position im XML-Baum oder über einen Rollen-Namen (etwa „Subjekt“, „Objekt“ oder auch — inspiriert von der thematischen Rollen [θ-Rollen] in der Generativen Grammatik oder von den Tiefenkasus in der Kasusgrammatik, „Agens“, „Patiens“). ProofML verwendet die Position zur Identifikation, da die Zuweisung von Namen für die Argumente entweder vollkommen willkürlich sein müßte, oder — wie die Suche nach Tiefenkasus oder θ-Rollen wahrscheinlich kontrovers sein könnte.[3]

In Abb. 8.3 auf der nächsten Seite sehen wir ein Beispiel für eine Argument-Verweisung mit dem Element <arg>. *haben* ist im Hintergrundwissen definiert als eine zweistellige

[2] Vgl. Zinn (to appear : 83) für ein Beispiel mit „and“ in vielen Funktionen.

[3] Ein Ausweg wäre es, Namen zu verwenden, die die syntaktische Rolle wiedergeben (vgl. die syntaktischen Relationen in Eisenberg 1999 : Kap. 2.3.2); dann wäre zwar bei Verben, die im Aktiv und im Passiv auftreten können, verschiedene Argument-Strukturen zu erwarten, aber die ergibt sich ja ohnehin durch die Optionalisierung des Aktiv-Subjekts. Dieser Aufwand steht allerdings einem sehr geringen Nutzen gegenüber, da nun anstelle der Reihenfolge die entsprechenden syntaktischen Rollen annotiert werden müßten.

```
<expression form="predicate" num="plural"
  pl-type="distributive" id="pred-have-1">
  <arg name="T1"/>
  <arg name="S1"/>
  <keyword>have</keyword>
</expression>
```

Abb. 8.3: **Funktor-Argument-Struktur**

Relation zwischen etwas, das hat (die beiden Dreiecke mit dem `name="T1"`), und etwas, das besessen wird (die beiden Seiten `name="S1"`).

Das Element <keyword> wird dazu verwandt, Wörter oder Phrasen zusammenzuhalten, die in der internen Struktur einer Relation den Funktor bezeichnet: etwa ein *light verb* oder ein *cue word*.

8.3.1. Reihenfolge der Argumente

Im Hintergrundwissen (vgl. Kap. 6 auf Seite 47) müssen den sprachlichen Ausdrücken Argumentstrukturen zugeordnet sein.

Da der günstigste Fall ist, daß ein Funktor mit seinen Argumenten zusammen eine Phrase bildet, keine anderen potentiellen Argumente diese unterbrechen und die Argumente (evtl. unterbrochen durch den Funktor) in der „Standardreihenfolge“ auftreten, nehmen wir diesen Fall als Standardfall an. Für diesen Fall können wir dem semantischen Funktor eindeutig seine Argumente zuordnen, da wir ja den Valenzrahmen über das `type`-Attribut eindeutig angeben (s. Kap. 8.4.2 auf Seite 58).

Annotationsregel 2 (Funktor-Argument-Vereinfachung) *Die Stelligkeit eines semantischen Funktors ist in der Ontologie festgelegt, ebenso die Typen der Argumente. Grundsätzlich sind die Argumente eines Funktors Geschwister- oder Kindelemente des Elements, das den Funktor einrahmt, und zwar in der Reihenfolge ihres Auftretens; relevant werden nur Elemente, die Argumente darstellen können: <`expression`>, <`name`>.*

Es ist offensichtlich, daß dieser Fall häufiger in Sprachen auftritt, die in der Satzgliedreihenfolge sehr festgelegt sind, wie etwa das Englische. In Sprachen, die in diesem Punkt variantenreicher sind, müßte zumindest eine morphologische Analyse vorgenommen

werden, die meist eine eindeutige Zuweisung der Argumente zu ihren Rollen vereinfacht. Für den Fall, daß eine solche Analyse durchgeführt wird, könnten die Syntax und Semantik von ProofML natürlich entsprechend angepaßt werden, oder der Parser kann eine Zuweisung der Argumente nach morphologischen Kriterien vornehmen, die von menschlichen Annotatoren kontrolliert wird.

Wenn der Standardfall eintritt, muß also die Argumentstruktur nicht gesondert annotiert werden; wenn allerdings die Argumente in einer anderen Reihenfolge stehen oder einige Argumente etwa implizit gegeben sind, wie im Falle von Partizipialkonstruktionen (92), müssen wir die Argumente ihren Funktoren explizit zuordnen.

Annotationsregel 3 (Explizite Funktor-Argument-Zuordnung) *Explizit durch ein Element* `<arg refid="id">` *zugeordnet werden Argumente nur, wenn sie nicht in der Reihenfolge stehen, die im Hintergrundwissen deklariert ist. Wird ein Argument explizit zugeordnet, so müssen alle anderen Argumente auch explizit zugeordnet werden.*

Haben wir beispielsweise das Prädikat einer Proposition, so gehen wir davon aus, daß seine Argumente im allgemeinen seine Geschwister in der XML-Hierarchie sind, und zwar in der Reihenfolge ihres Auftretens.[4]

8.4. Nominalphrasen (NPs)

Eine Notwendigkeit, gemäß dem Vorgehen von Montague anzunehmen, daß jede Nominalphrase ein generalisierter Quantor ist, besteht nicht, wenn man semantische Prinzipien (Partee 1987) oder syntaktische Operationen (Argumentation mit Hypothesen in der kategorialen Grammatik à la Moortgat) annimmt, die ggf. den semantischen Typ einer Nominalphrase anpassen können.

Außerdem ist das Ergebnis einer Anwendung von Prädikaten, die Individuen als Argumente fordern, auf NPs, die Individuen denotieren, identisch mit der Anwendung des generalisierten Quantors, der einer NP (nach einer Typen-Anhebung) entspricht, auf das entsprechende Prädikat.

Wir nehmen daher für Nominalphrasen im allgemeinen eine Denotation vom Typ „Individuum“ an, in der Mathematik können wir auch von „Objekten“ sprechen. Wir gehen

[4] Während die Reihenfolge im Englischen das am häufigsten zutreffende, also sinnvollste Kriterium für eine Standardinterpretation sein dürfte, könnte man für das Deutsche oder Griechische wohl den Kasus heranziehen; dies würde wahrscheinlich ein wenig komplizierter.

davon aus, daß die Denotation von Variablen, Namen und Pronomina gleichartig zu derjenigen von Nominalphrasen ist.

8.4.1. NPs als Individuen/Objekte

Normalerweise werden Objekte (auch Verbindungen von Objekten, s. Kap. 9.1.1 auf Seite 90) mit einem <expression>-Element annotiert, welches beim ersten Auftreten des Objekts ein id-Attribut bekommt, wir nennen dies die ID des Objekts. Wird wieder über dieses Objekt gesprochen, erhält das entsprechende <expression>-Element refid-Element, welches sich auf die ID des Objekts bezieht.

8.4.2. Das Attribut type des Elements <expression>

Das Attribut type erhält einen Wert, der anzeigt, von welcher Art die entsprechende Phrase bzw. die entsprechende Wortform ist. Es reicht offensichtlich aus, diese Annotation bei der ersten Einführung des Referenten vorzunehmen, da im folgenden über das refid-Attribut das Objekt (und damit sein Typ) bestimmt ist.

Im ProofML-Korpus haben wir einen Unterschied zwischen den Werten für type bei Wortformen und denjenigen bei Phrasen gemacht.

Für Phrasen nehmen wir eine Grobgliederung der Denotation vor: "proposition", "object" etc. bedeuten jeweils, daß die entsprechende Phrase als Denotation eine Proposition, ein Objekt hat.

Für Wortformen folgen wir dem Muster type="*klasse:typ*".[5] Eine solche Schablone für den Wert von type ist natürlich nicht notwendig, sondern zum Zwecke der Illustration gewählt worden.[6] Die Idee ist, daß "*klasse:typ*" entweder den Lexikoneintrag für die entsprechende Wortform bestimmt, oder zumindest die Valenz der annotierten Wortform. Der (imaginäre) Lexikoneintrag sollte bestimmt werden, solange keine Valenzklassen gebildet worden sind, wie etwa in dieser Arbeit.

Es scheint vielleicht verwunderlich, daß bei Phrasen nicht ein klarer bestimmter Typ der Denotation angegeben wird, etwa "object:triangle". Dies würde allerdings bei jeder Annotation die Frage aufwerfen, wie speziell etwa ein „Objekt“ zu annotieren

[5] Die Kursivschreibung soll andeuten, daß es sich um variable Bestandteile des Namens handelt.

[6] Man könnte sich vorstellen, daß *klasse:* ein gemeinsames XML-Namespace-Präfix (Bray et al. 1999) für alle Definitionen von Worten einer Klasse ist, etwa im Rahmen einer RDF- oder DAML+OIL-Ontologie.

wäre: als Dreieck oder als geometrische Figur? Oder zumindest als Vieleck? Was, wenn sich eine neue Eigenschaft des Objekts im Beweis zeigt?

Es scheint also sinnvoll, die Informationen da zu annotieren, wo sie auftreten. Bei der Einführung eines Diskursreferenten wird üblicherweise auf eine von zwei Arten und Weisen eine semantische Klasse für dieses Objekt angegeben: entweder, indem sie explizit genannt wird oder indem man bestimmten Namenskonventionen gehorcht, die man vorher definiert hat oder die allgemein bekannt sind.[7] Zu den Namenskonventionen siehe Kap. 8.5 auf Seite 67.

8.4.3. Numeri

Es ist für die Semantik einer NP natürlich wichtig, welchen Numerus sie hat; dies wird in der Annotation von ProofML durch das Attribut `nr` angegeben, das die Werte `"sg"` (Singular) und `"pl"` (Plural) annehmen kann. Das Attribut `pl-type` gibt den Typ des Plurals an, sofern die NP im Plural steht (andernfalls ist der Wert dieses Attributs irrelevant), `"distributive"` oder `"collective"` für distributive oder kollektive Plurale. Der Default-Wert ist `pl-type="collective"`.[8]

8.4.4. Konjunktion

Das Attribut Form gibt Beziehungen zwischen Elementen wieder; der einzige Wert dieses Attributs, der an dieser Stelle bereits relevant ist, ist `form="conj"`. Dieser Wert steht für die Konjunktion, wie sie in der natürlichen Sprache mit „und“ vorgenommen wird, und kann bei den Elementen <`expression`> und <`name`> vorkommen, sowie zwischen <`nl-premise`>-Elementen. Bei <`expression type="object"`> oder Namen kann auch bei Konjunktionen ein Attribut `pl-type` auftreten, welches distributiven und kollektiven Gebrauch der Konjunktion anzeigt. Zur weiteren Verwendung des Attributs `form` vgl. Kap. 9.1.2 auf Seite 92.

[7] Ein solcher geläufiger Name ist ε, das in der Analysis oft für beliebig kleine, aber positive Zahlen verwandt wird. Für den Nichtmathematiker ist es beeindruckend, welche Heiterkeitsausbrüche der Satz „Sei ε kleiner Null.“ unter Mathematikern auslösen kann.

[8] Wie der auf S. 42 (Fußnote) zitierte Beispielsatz von Ranta zeigt, sind offensichtlich Anaphern nicht unbedingt darauf beschränkt, denselben `pl-type` zu haben. Insofern ist es immer notwendig, `pl-type` anzugeben und es kann keine Übernahme von den <`expression`>-Elementen erfolgen, auf die mit einem `refid`-Attribut verwiesen wird.

8.4.5. NPs als generalisierte Quantoren

Daß wir gewisse Phrasen als generalisierte Quantoren annotieren wollen, haben wir bereits begründet und auch die entsprechende Struktur bereits angedeutet:

Annotationsregel 4 (Generalisierte Quantoren) *Phrasen, die als generalisierte Quantoren annotiert werden, werden in ein* <quant>*-Element geschachtelt und, soweit möglich, aufgeteilt in* <det>, <restr>*. Dem Quantor wird ein Geltungsbereich* <scope> *zugeordnet. Das* <quant>*-Element erhält ein Attribut* binds*, dessen Wert den Namen (die ID) der Variable angibt, die der Quantor bindet.*

Wie auch Koepke und Schröder (2003) lassen wir in unserer Annotation zu, daß der Quantor im Geltungsbereich (Skopus) eingeschachtelt ist.

Es bietet sich an, zu diesem Zweck verschiedene Arten von Quantoren zu unterscheiden, unter anderem, weil indefinite Nominalphrasen ohne Determinierer im Plural als Existenzquantoren zu interpretieren sind und die Annotation in diesen Fällen ohne Unterscheidung der Typen von Quantoren ein leeres <det>-Element o.ä. erfordern würde.

Annotationsregel 5 (<quant>-Typen) *Das Attribut* type *bei Quantoren gibt an, ob es sich um einen Existenz-(*type="exist"*), All-(*type="all"*) oder einen anderen Typ generalisierten Quantor (*type="gen"*) handelt. Der* Default*-Typ ist* type="exist"*.*

Außerdem gilt, daß, wenn zwei Existenz- oder zwei All-Quantoren gleicher Art auftreten, es unerheblich ist, welcher von beiden Skopus über den anderen hat. Wir führen daher ein, daß in einem solchen Fall einem Skopus zwei Quantoren-Variablen zugeordnet werden können. Dies geschieht mit dem Element <scope-of>, dessen Attribut refid den Wert der ID der entsprechenden gebundenen Variable erhält. Überhaupt gilt:

Annotationsregel 6 (Skopus von Quantoren) *Ein* <scope>*-Element bestimmt immer den Geltungsbereich desjenigen Quantors, der sein Kind ist oder dessen Kind er ist. Im Zweifelsfall ist der Bezug mit einem* <scope-of>*-Element zu disambiguieren, dessen* refid*-Attribut auf die* ID *der Variable verweist, die durch den Quantor gebunden ist.*

Kommt also nur ein generalisierter Quantor in der Formel vor und bindet er die Variable, die das nächste noch ausstehende semantische Argument des entsprechenden semantischen Kopfes / Funktors liefert, so reicht es prinzipiell aus, die Bindung nicht anzuzeigen, sondern dem Quantor nur einen entsprechenden Skopus zuzuordnen.

Variablennamen Es ergibt sich sozusagen von selbst, daß ein Name, der eine von einem Quantor gebundene Variable bezeichnet (er ist üblicherweise dem Restriktor beigeordnet), mit einem =refid-Attribut auf sie verweist. Bei Kardinalzahl-Quantoren kann es geschehen, daß mehrere Namen angegeben werden (s. (15) in Abb. 8.4 auf der nächsten Seite). In diesem Fall ist es oft durchaus sinnvoll, den Quantor nur eine Variable binden zu lassen, denn sonst müssen wir immer, wenn wir über beide Dreiecke zusammen sprechen, so vorgehen, als hätten wir es mit *Split Antecedents* zu tun. Wir können aber angeben, daß dieser eine Name für die Konjunktion zweier Namen steht (<name form="conj">). Ähnlich können wir vorgehen, wenn keine Namen angegeben sind (s. (16) in Abb. 8.4 auf der nächsten Seite); die andere mögliche Lösung, nämlich ein <bridged-from>-Element zu verwenden (s. Kap. 8.9.3 auf Seite 82), kann zu recht unübersichtlichen Ergebnissen führen; allerdings scheint diese Lösung uns intuitiv plausibler.

Dies ist die übliche Vorgehensweise in der formalen Semantik, die es erlaubt (sofern Peter′ und Paul′ als generalisierte Quantoren Peter′ : $\lambda P.(P\ \text{peter}')$, Paul′ : $\lambda P.(P\ \text{paul}')$ verstanden werden)

(17) (Peter′(Paul′ lieb′)) zu schreiben und entsprechend

(18) ((lieb′ paul′)peter′) zu erhalten (so als ob peter′ und paul′ als Argumente vom Typ Individuum gegeben worden wären).

Es kann es passieren, daß Argumente nicht in der gewünschten Reihenfolge erscheinen oder wenn verschachtelte generalisierte Quantoren in einem Satz auftauchen (etwa ein Subjekt- und ein Objekt-Quantor). Wenn man in diesen Fällen die Oberflächenstruktur respektiert (dazu zwingt uns die XML-Struktur in gewissem Maße), muß dem Quantor eine Variablenbindung zugeordnet werden, und man muß mit <arg> auf die entsprechende Variable verweisen. Dies ist dann parallel zum Vorgehen in der klassischen logischen Schreibweise für Quantoren zu sehen, wo man immer explizit die gebundene Variable angibt:

(19) $\forall x : x > 2.$

Kardinalzahlen und Quantoren Quantoren der Art „einige“, „zwei“,... behandeln wir als Existenzquantoren.

(15) zwei Dreiecke *ABC* und *DEF*.

```
<quant binds="dreiecke">
  <det type="det:zwei">zwei</det>
  <restr>
    <expression type="property:dreieck">Dreiecke</expression>
  </restr>
  <name refid="dreiecke" form="conj">
    <name id="ABC" type="compound">ABC</name>
    <keyword>und</keyword>
    <name id="DEF" type="compound">DEF</name>
  </name>
</quant>
```

(16) zwei Dreiecke

```
<quant binds="dreiecke">
  <det type="det:zwei">zwei</det>
  <restr>
    <expression type="property:dreieck">Dreiecke</expression>
  </restr>
  <name refid="dreiecke" form="conj">
    <name id="ABC"/>
    <name id="DEF"/>
  </name>
</quant>
```

Abb. 8.4: **Zweimal zwei Dreiecke**

8.4.6. Beziehungsweise

Zu beachten ist, daß *beziehungsweise*, *jeweils*, *respectively* und ihre Entsprechungen in anderen Sprachen erzwingen können, daß die genannten Objekte sozusagen als Listen behandelt werden und einzeln miteinander in Relation gesetzt werden, etwa wie mit einer `map`-Funktion in funktionalen Programmiersprachen:

(20) a. *Ἔστω δύο τρίγωνα τὰ ΑΒΓ, ΔΕΖ τὰς δύο πλευρὰς τὰς ΑΒ, ΑΓ ταῖς δυσὶ πλευραῖς ταῖς ΔΕ, ΔΖ ἴσας ἔχοντα ἑκατέραν ἑκατέρᾳ [...]*

b. Let *ABC* and *DEF* be two triangles having the two sides *AB* and *AC* equal to the two sides *DE* and *DF*,[...] Elemente, I.4

Hier soll gelten: *AB* = *DE* und *AC* = *DF*, nicht aber (notwendigerweise) *AB* = *DF* oder *DE* = *DF*. Dies scheint auf Englisch unmißverständlich; im Griechischen scheint die Konstruktion nicht ganz so eindeutig zu sein, denn Euklid stellt noch die Klarstellung nach:

(21) a. *τὴν μὲν ΑΒ τῇ ΔΕ τὴν δὲ ΑΓ τῇ ΔΖ*

b. namely *AB* equal to *DE* and *DE* equal to *AC*.

In ProofMLwerden *respectively* etc. in dieser Verwendung als Verbmodifikatoren annotiert.

8.4.7. Einführung und Fortführung von Diskursreferenten

Wie bereits in Kap. 4.1.6.1 auf Seite 33 ausgeführt, ist es für die maschinelle Analyse eines mathematischen Textes eine Schwierigkeit, alle Arten und Weisen zu erfassen, wie Diskursreferenten in den Text eingeführt werden können, während für die Annotation folgende Regel ausreicht.

Annotationsregel 7 (Einführung von Diskursreferenten) *Das erste Mal, wenn ein Objekt im Kontext eines Beweises erscheint, wird ihm eine* id *zugeordnet, welche es für den gesamten Text eindeutig identifiziert.*

Dies ist ähnlich zu verstehen, wie die Einführung einer existenzquantifizierten Variable in der DPL, die ihrem Geltungsbereich in der Proposition hat, in der sie das erste Mal auftritt (Proposition ist hier so eng wie möglich zu fassen; im Zweifelsfall ist die Phrase

als Quantor zu annotieren und ihr ist ein entsprechender Geltungsbereich zuzuordnen). Üblicherweise verläuft der Schluß in der Mathematik ja so, daß man etwa beginnt:

(22) Sei A ein Punkt[...]

und am Ende des Beweises schließt:

(23) Also gilt für alle Punkte, daß [...]

Zu beachten ist natürlich, daß solchen Konstanten, die in einem Beweis das erste Mal auftauchen, deren Name aber aus dem Hintergrundwissen stammt, etwa π oder e, keine ID zugeordnet wird, sondern die ID aus der Hintergrund-Theorie im `refid`-Element referenziert wird.

Wir gehen dennoch auf einige Fälle ein, in denen wir die Anwendung der gerade gegebenen Regel demonstrieren.

BAUR (1999) nennt einige Methoden in der Mathematik, Diskursreferenten einzuführen.

Wir wollen sie — entgegen BAUR (1999) — in zwei Gruppen aufteilen: in formelsprachliche und natürlichsprachliche Methoden der Einführung; dies bietet sich in unserem Fall an, da in der Geometrie die Einführung von Diskursreferenten durch eine natürlichsprachliche Wendung üblicher ist als in der Analysis, die BAUR (1999) betrachtet.

BAUR (1999 : 33ff) nennt folgende Beispiele, die wir als formelsprachlich bezeichnen wollen (wir passen die Beispielnummern unserem Text an):

(24) *Let* **a, b, c** be elements of $\mathbb{N}$.

(25) There *exists* $\mathbf{a} \in \mathbf{R}$ such that $\cdots$

(26) If $\mathbf{a} \in \mathbb{R}$ *and* $\mathbf{a} \neq 0$, then $\cdots$

(27) If $\mathbf{a} \in \mathbb{R}$ is *such that* $\mathbf{a} \neq 0$, then $\cdots$

(28) If **a** *is any* element of $\mathbb{N}$ $\cdots$

(29) If **a** *is an* element $\cdots$

(30) *A* number A is a positive $\cdots$

Aus den Texten, die wir betrachtet haben, ergänzen wir noch:

(31) Let $\triangle ABC$ be a triangle, ...

(32) Euklid, Elemente I,1

a. *Ἐπὶ τῆς δοθείσης εὐθείας πεπερασμένης τρίγωνον ἰσόπλευρον συστέσασθαι. Ἔστω ἡ δοθεῖσα εὐθεῖα πεπερασμένη ἡ AB.*

b. On a given finite straight line to construct an equilateral triangle. Let AB be the given finite straight line.

c. In data recta terminata triangulum sequilatrum construere. Sit data recta terminata *AB*.

d. *Über einer gegebenen Strecke ein gleichseitiges Dreieck zu errichten.* Die gegebene Strecke sei *A B*.

(33) Euklid, *Elemente* I.4:[9]

a. *Ἐὰν δύο τρίγωνα τὰς δύο πλευρὰς [ταῖς] δυσὶ πλευραῖς ἴσας ἔχῃ [...] Ἔστω δύο τρίγωνα τὰ ΑΒΓ, ΔΕΖ τὰς δύο πλευρὰς τὰς ΑΒ, ΑΓ ταῖς δυσὶ πλευραῖς ταῖς ΔΕ, ΔΖ ἴσας ἔχοντα ἑκατέραν ἑκατέρᾳ [...]*

b. Si duo trianguli duo latera duobos lateribus alterum alteri aequalia habent [...] duo trianguli *ΑΒΓ*, *ΔΕΖ* duo latera *AB*, *ΑΓ* duobus lateribus *ΔΕ*, *ΔΖ* aequalia habentes alterum alteri [...]

c. If two triangles have the two sides equal to two sides respectively, [...] Let *ABC*, *DEF* be two triangles having the two sides *AB*, *AC* equal to the two sides *DE*, *DF* respectively.

Baur (1999 : 33) weist darauf hin, daß Diskursreferenten in solchen Sätzen nicht als indefinite NPs eingeführt werden. Dies ist allerdings nicht weiter verwunderlich, wenn man die Variablen-Namen als Eigennamen sieht.[10]

[9] Auffällig sind die definiten Artikel bei *δύο πλευράς* und [ergänzt vom Herausgeber] bei *δυσὶ πλευραῖς*; dies ist nur möglich wegen des besonderen Gebrauchs des Wortes *βάσις* bei Euklid; vgl. Heaths Fußnote 3 zu El. I.4.

[10] Es ist hier nicht weiter von Belang, ob man solche Formeln im Text wie einen Relativsatz wertet (Baur 1999 : 26) oder einfach als ein Attribut (das natürlich durch einen Relativsatz wiedergegeben werden kann, aber ggf. auch durch eine andere Phrase — wie im Falle von $a \in \mathbb{R}$ „a aus R“).

Klar ist, daß ein Diskursreferent eingeführt und dabei näher bestimmt wird. Ein analoges Beispiel aus der natürlichen Sprache:

(34) Ein fünfunddreißigjähriger indischer Dickhäuter aus Sri Lanka betrat einen Laden für echtes

```
<relation type="particularisation">
  <arg refid="T1"/> <!-- indefinite NP -->
  <arg refid="T2"/> <!-- bezieht sich auf ABC und DEF -->
</relation>
```

Abb. 8.5: **Partikularisierung**

Variablen können als Diskursreferenten also in formelsprachlichen Ausdrücken einfach dadurch eingeführt werden, daß sie das erste Mal erwähnt werden. Die Semantik dieser Einführung ist analog zu verstehen wie die der Einführung von NP-Referenten (vgl. S. 63); die Annotation ist daher auch analog vorzunehmen. Baur (1999) weist darauf hin, daß es häufig ähnliche Kontexte gibt, in denen solche Formeln eingeführt werden, diese sind für uns hier allerdings irrelevant, da sie die Annotation nicht beeinflussen; für die Annotation ist es ausreichend, eine Variable mit Namen a einzuführen, zu welcher bestimmte Bedingungen (Zugehörigkeit zur Menge $\mathbb{R}$, Ungleichheit mit 0) angegeben werden.

In (32) haben wir den Fall, daß ein allgemeines, als „gegeben“ vorausgesetztes Dreieck durch eine (außer im lateinischen Text natürlich) indefinite Nominalphrase eingeführt und dann im folgenden Satz benannt wird.

In (33) werden zwei Dreiecke durch eine indefinite Nominalsprache eingeführt, obwohl auch hier vorher die Rede von solchen Dreiecken war — sie wurden allerdings nicht als gegeben benannt.

Wir annotieren die beiden Fälle verschieden: Während in (32) die beiden Nominalphrasen, die die Dreiecke bezeichnen, als koreferent annotiert werden, annotieren wir die Dreiecks-NP in (33) als eine Partikularisierung der Dreiecks-NP des Demonstrandums (s. Abb. 8.5 und Kap. 9.5 auf Seite 100).

8.4.8. Definite Nominalphrasen

Definite Nominalphrasen werden als `<expression type="object">` annotiert. Sie enthalten meist ein `<det>`-Element sowie eine Annotation der entsprechenden näheren Bestimmungen (Substantive, Adjektive, Präpositionalphrasen, Namen etc.).

Meißner Geschirr.

Nominalphrasen, die mit anderen Nominalphrasen koreferent sind, verweisen mit dem Attribut `refid` auf die entsprechenden IDs. Entsprechendes gilt, wenn sie durch *Bridging* erschlossen werden.

Gelegentlich werden bei der Einführung benannten Objekten definite NPs verwandt. In diesem Fall wird die Annotation wie bisher beibehalten, aber es tritt ein Attribut `id` auf, ohne daß ein *Bridging* erfolgt (also wie in Kap. 8.4.7 auf Seite 63 beschrieben). Die Interpretation solcher Konstruktionen, die Diskursreferenten einführen, ist ähnlich der von Namen (s. S. 66).

8.5. Namen

Einen großen Fortschritt in der Sprache der Mathematik, so Lamport (1993), stellt die Benennung von mathematischen Entitäten dar. Er schreibt diese Errungenschaft für den Bereich der Zahlenvariablen der Neuzeit zu (Lamport 1993 : 1). Für die Geometrie scheint diese Praxis nicht so neu — immerhin finden wir sie bereits bei Euklid.[11]

Die Namensgebung in der Geometrie verhält sich allerdings in einigen Punkten anders als beispielsweise in der Logik oder Arithmetik. So kommen bloße Reihen von Referenzpunkten als Namen nur in der Geometrie vor. Wir werden zunächst eine Annotation für Namen in der Geometrie geben und sie anschließend für Namen in der Logik und anderen Bereichen der Mathematik erweitern.

Zunächst stellen wir da, wie Namen bei Euklid und allgemein in der Geometrie verwandt werden, und zeigen dann, wie wir sie in ProofML annotieren.

8.5.1. Namen bei Euklid

Bei Euklid finden wir oft Bezeichnungen der folgenden Art für geometrische Entitäten:

(35)
a. der Punkt *A*
b. die Strecke *AB*
c. die Strecke *C*
d. der Kreis *ABC*
e. der Winkel *ABC*.

(36)
a. the point *A*
b. the straight line *AB*
c. the straight line *C*
d. the circle *ABC*

[11] Wir stimmen also prinzipiell mit Zinn (1999) überein, daß ein Repräsentationssystem für mathematische Beweise einen Typ von Diskurs-Referenten für Namen bereitstellen sollte. Es ist allerdings für die Zwecke unserer Annotation nicht erforderlich, daß dieser Typ von Diskursreferent von dem definiter Nominalphrasen verschieden ist.

e. the angle *ABC*

(37) a. τὸ *B* σημεῖον
b. (i) ἡ *AB* εὐθεῖα
(ii) ἡ εὐθεῖα ἡ *AB*
(iii) ἡ *AB*
c. ἡ *Γ* εὐθεῖα
d. τὸ *ABΓ* κύκλον
e. ἡ ὑπὸ *ABΓ* γωνία

Während im Deutschen und Englischen Namen von geometrischen Objekten an der gleichen Stelle im Satz stehen wie Eigennamen, scheint es sich im Griechischen um eine nähere, attributive Bestimmung zu handeln, also etwa in dem Sinne von „das Dreieck mit den Eckpunkten *A*, *B* und *C*“ oder, der griechischen Wortstellung angepaßt: „Das *ABC*-Dreieck“. Sie steht an derselben Stelle wie ein Adjektiv.

Dennoch werden die Namen geometrischer Objekte bei Euklid oft so verwandt, als identifizierten sie diese Objekte eindeutig, insbesondere wenn es sich um Strecken handelt. Das mag daran liegen, daß es keine anderen geometrischen Objekte gibt, die mit zwei einfachen Buchstaben benannt sind und insofern eine Benennung wie *AB* eindeutig ist. In den meisten Fällen gilt dies auch für Namen von Punkten; bei ihnen setzt Euklid allerdings immer das Substantiv σημεῖον hinzu. Zu dieser Praxis paßt, daß gelegentlich auch Strecken mit einem Einzelbuchstaben gekennzeichnet werden können.[12] Auch dann kann die Bestimmung „Strecke“ entfallen.

Kurze Erläuterung zu geometrischen Entitäten mit drei Referenzpunkten Bei Euklid können folgende geometrischen Entitäten durch drei Referenzpunkte bezeichnet werden:

Kreise Beim Kreis stehen drei beliebige — meist bekannte — Punkte, die auf ihm liegen.[13]

Dreiecke Beim Dreieck sind es die drei „Eckpunkte“, wobei konventionellerweise die Punkte im „mathematisch positiven Sinn“ angegeben werden, also entgegen der

[12] Ein Beispiel findet sich im folgenden Beispiel aus *Elemente* I,3. Bei der Übesetzung ins Deutsche hat der Übersetzer es offensichtlich als ungewöhnlich empfunden, eine Strecke mit einem Großbuchstaben zu bezeichnen. Vgl. a. 8.5.3:

(38) a. *Ἔστωσαν αἱ δοθεῖσαι δύο εὐθεῖαι ἄνισοι αἱ AB, Γ, ὧν μείζων ἔστω ἡ AB·*
b. Die zwei gegebenen ungleichen Strecken seien *AB*, *c*; von ihnen sei *AB* die größere.

[13] Oft werden auch (gemäß Postulat I,4) der Mittelpunkt und der Radius genannt.

Richtung des Uhrzeigers; es macht allerdings keinen Unterschied, mit welchem Punkt man bei der Aufzählung beginnt: das Dreieck *ABC* ist auch das Dreieck *BCA* und *CAB*.

Winkel Auch bei einem Winkel spielt die Abfolge der Punkte eine wichtige Rolle (der Winkel *ABC* ergänzt sich zu 360° mit dem Winkel *CBA*), zu beachten ist allerdings außerdem, daß man hier in der Mitte den Scheitelpunkt des Winkels nennt und die beiden anderen Punkte auf den Schenkeln des Winkels liegen (wiederum im „mathematisch positiven Sinn").

Für Winkelbezeichnungen mit drei Referenzpunkten sieht es im Englischen und Deutschen etwas anders aus als im Griechischen; während nämlich hier eine Präpositionalphrase mit *ὑπό* steht — mit einem Artikel wohlgemerkt — (vgl. (39)), steht im Deutschen und Englischen keine Präposition.

(39) a. *λέγω, ὅτι ἡ μὲν ὑπὸ ΑΒΓ γωνία τῇ ὑπὸ ΑΓΒ ἴση ἐστιν, ἡ δὲ ὑπὸ ΓΒΔ τῇ ὑπὸ ΒΓΕ.*

b. I say that the angle ABC is equal to the angle ACB, and the angle CBD to the angle BCE.

c. dico, esse ∠ *ΑΒΓ* = *ΑΓΒ* et ∠ *ΓΒΔ* = *ΒΓΕ*.

d. Ich behaupte, daß ∠ *ABC* = *ACB* und *CBD* =*BCE*.

Der englische Übersetzer von (39) hält hier also die Referenzpunkte für nicht ausreichend, um den Winkel als solchen zu kennzeichnen, auch dort, wo im griechischem das Wort *γωνία* („Winkel") ausfallen kann, da aus dem Zusammenhang klar ist, daß es sich um Winkel handelt und dies durch die Verwendung des Artikels/Demonstrativpronomens mit der Präposition in einer elliptischen Konstruktion deutlich wird.[14] Die Präposition gehört also zum Namen.

Mehrdeutige Buchstabenkombinationen stehen also nur in solchen Fällen ohne Substantiv, in denen entweder die Form des Namens eine eindeutige Zuordnung zu einer Kategorie ermöglicht (Streckenbezeichnungen) oder es durch eine inhaltliche und oft auch syntaktische Parallelität zu vorangehenden Phrasen ergänzt wird[15] und dieser

[14] Vgl. Heath (1920 : 167, Fn. 12 zu I,3) für eine Erklärung, weshalb eine Präpositionalphrase steht und weshalb die Punkte in der gegebenen Reihenfolge stehen.

[15] Sehr interessant ist auch die Praxis Heibergs, in seiner lateinischen Übersetzung, unabhängig vom Vorkommen im griechischen Text, grundsätzlich vor die Referenzpunkte (bei zwei aufeinanderfolgenden vor den ersten; vgl. oben (39-c)) von Dreiecken „△" und vor Winkel „∠" zu setzen, während der deutsche Übersetzer sparsamer mit den Symbolen umgeht.

sprachliche Kontext eine Implikatur bezüglich der Kategorie des durch den Namen bezeichneten Objekts erzeugt.

(40) a. *Εἰλήφθω γὰρ ἐπὶ τῆς ΒΔ τυχὸν σημεῖον τὸ Ζ, καὶ ἀφῃρήσθω ἀπὸ τῆς μείζονος τῆς ΑΕ τῇ ἐλάσσονι τῇ ΑΖ ἴση ἡ ΑΗ, καὶ ἐπεζεύχθωσαν αἱ ΖΓ, ΗΒ εὐθεῖαι.*

b. Let a point F be taken at random on BD; from AE the greater let AG be cut off equal to AF the less; and let the straight lines FC, GB be joined.

8.5.2. Eine grundlegende Annotation für Namen

Wir wollen das bisher Gesagte nun interpretierend zusammenfassen und eine Annotation für benannte Entitäten vorstellen. Wir gehen davon aus, daß einfache Namen eine Entität eindeutig identifizieren, wenn sie ihr vorher zugewiesen worden sind.

Bei zusammengesetzten Namen haben wir in den Beweisen, die wir untersuchen, zwei Fälle zu unterscheiden: eindeutige und mehrdeutige.

Namen aus einzelnen Buchstaben oder aus zwei Buchstaben identifizieren Strecken eindeutig; längere Namen können, wie gezeigt, als eine konventionalisierte Abfolge von Referenzpunkten gedeutet werden, reichen aber im allgemeinen nicht aus, um eine Entität eindeutig zu bestimmen; dies gelingt nur in einem günstigen Kontext.

Wir werden daher den Namen mit dem Element <name> annotieren, dessen Attribut `form` den Default-Wert `"simple"` für einfache Namen hat; `form="compound"` wird für zusammengesetzte Namen verwandt. In der Geometrie steht <name form=compound"> für ein Tupel von Referenzpunkten; es kann also weiter aufgelöst werden in einzelne Namen. Da eine vollständige Auflösung zusammengesetzter Namen die Annotation sehr unübersichtlich macht, haben wir in manchen Beispielen im Korpus und in manchen Beispielen dieser Arbeit darauf verzichtet, eine solche Auflösung vorzunehmen. Werden die annotierten Texte maschinell verarbeitet, ist dies natürlich sehr leicht nachzuholen.[16] Eine weitere Vereinfachung wäre, daß man nur die Einführung von Namen eine ID vergibt und im folgenden davon ausgeht, daß sich Namen mit dem letzten Namen koreferent sind, dem eine ID zugewiesen wurde und der den selben Inhalt hat wie sie.

Wir führen außerdem für die Annotation von Namen das Attribut `class` ein, dessen Wert die speziellste Eigenschaft benennt, die man aus dem Namen erschließen kann.

[16] Sollte einmal ein Name aus mehreren Buchstaben bestehen, die aber selbst nicht einzelne Namen sind, so wäre dieser Name natürlich nicht als zusammengesetzter Name zu annotieren.

Steht allerdings ein Name alleine und kann aus ihm nicht geschlossen werden, welcher Art das Objekt ist, das er benennt, und geht diese Information auch nicht aus einer beigeordneten oder prädikativen NP hervor, so kann es sinnvoll sein, dem Namen ein — bezüglich der damit verbundenen Zeichenkette — leeres <expression>-Element beizuordnen und in diesem mittels <expression type="property:*typ*"> anzugeben, welches der Typ der entsprechenden Phrase ist. Ggf. kann man in einem <bridged-from>-Element angeben, woraus man auf den Typ schließt. Abb. 8.6 auf der nächsten Seite ist einmal so dargestellt, als wäre ein Name, der aus zwei einbuchstabigen Namen zusammengesetzt, eindeutig, und einmal so, als wäre der Name *DC* oder $\Delta\Gamma$ nicht eindeutig, sondern man habe die Tatsache, daß es sich um eine Strecke handelt, nur aus dem Verb schließen können.

Da Namen oft für sich stehen und aus ihrer Form auf die Art von Objekt geschlossen werden kann, für die sie stehen, scheint es unnötig, sie in einem solchen Fall in ein <expression>-Element zu verschachteln, das genau diese Angaben liefert. Wir legen also fest:

Annotationsregel 8 (Vereinfachung der Namens-Annotation) *Wenn ein Name nicht durch eine beigeordnete NP näher bestimmt wird und die Denotation eines Namens über ein* refid*-Attribut aufgelöst ist oder aus der Art des Namens hinreichend genau zu ersehen ist, von welcher Art das Objekt ist, für das er steht, so muß der Name nicht in ein* <expression>*-Element geschachtelt werden.*

8.5.3. Moderne Namensgebung im allgemeinen: *Sei* $\Xi_0^1 = \widetilde{\mathcal{X}}(\mathfrak{x}')$

Wir können Ebbinghaus et al. (1996) insofern als paradigmatisch für den moderenen mathematischen Sprachgebrauch auch außerhalb der Logik betrachten, als der Text eine sehr differenzierte Namensgebung aufweist, die viele der typographischen Möglichkeiten nutzt, die den Autoren mathematischer Texte in einem modernen Textverarbeitungs- (*Word*[*Perfect*]) oder Satzsystem (LaTeX, ConTeXt etc.) zur Verfügung stehen.

Es werden nicht mehr nur Entitäten mit einfachen Buchstaben bezeichnet; vielmehr mehrere Alphabete verwendet (griechische und lateinische Groß- und Kleinbuchstaben; in anderen Textteilen Fraktur; kalligraphische Schrift). Diesen Schriften sind auch bestimmte Funktionen zugeordnet. Außerdem werden Buchstaben nach bestimmten Regeln mit diakritischen Zeichen, Sub- und Superskripten ausgestattet.

In einleitenden Werken ist es üblich, diese Konventionen explizit einzuführen; zumin-

(41) a. ἐπεζεύχθω ἡ $\Delta\Gamma$
b. ducatur $\Delta\Gamma$.
c. ziehe DC.

```
<!-- Falls zweiteilige Namen eindeutig sind: -->
<expression type="proposition">
  <expression type="predicate:ziehe">
    ziehe
  </expression>
  <expression type="object" id="DC">
    <name class="property:straight-line" type="compound">
      <name>D</name><name>C</name>
    </name>
    <expression type="property:straight-line"/>
  </expression>
</expression>

<!-- Falls zweiteilige Namen nicht eindeutig sind -->
<expression type="proposition">
  <expression type="predicate:ziehe" id="zieh">
    ziehe
  </expression>
  <expression type="object" id="DC">
    <name type="compound"><name>D</name><name>C</name></name>
    <expression type="property:straight-line">
      <bridged-from>
        <arg refid="zieh"/>
      </brigded-from>
    </expression>
  </expression>
</expression>
```

Abb. 8.6: **Ein Objekt-Typ aus dem Kontext**

dest in diesen Fällen kann man aus der Art des Namens auch gewisse Schlüsse über die Art der benannten Entität ziehen. Ebbinghaus et al. (1996 : 15) geben gleich zu Anfang eine Bestimmung der für ihr Buch verbindlichen Konventionen.

(42) Fortan stehen $P, Q, R, \ldots$ für Relationssymbole, $f, g, h, \ldots$ für Funktionssymbole, $c, c_0, c_1, \ldots$ für Konstanten und $x, y, z, \ldots$ für Variablen.

Solche Konventionen bei der Benennung sollten im Hintergrundwissen definiert sein, und sie können dann auch bei der maschinellen Verarbeitung von Namen ausgenutzt werden.

8.5.3.1. Alphabete im allgemeinen

Ebbinghaus et al. (1996 : 11ff) unterscheiden endliche von unendlichen Alphabeten. Endliche Alphabete haben definitionsgemäß nur eine endliche Anzahl von Zeichen; unter diese Kategorie fallen etwa das lateinische oder das griechische Alphabet, aber auch allerlei Zeichen, die in der Mathematik üblich sind ($<, >, +, -, *, \cdot, \vdash, \ldots$). Unendliche Alphabete besitzen im allgemeinen einen Lauf-Index, der als Sub- oder Superskript (oder beides) steht; sie enthalten also keine atomaren Zeichen; während dies für Ebbinghaus et al. (1996) für die Betrachtung der Größe des Alphabets unerheblich ist, da diese „Alphabete“ einer Schablone folgen und eben keine atomaren Zeichen enthalten, werden wir dies für ProofML ausnutzen.

Endliche Alphabete Die einfachste Verwendung von Alphabeten zur Bezeichnung mathematischer Entitäten finden wir bei Euklid: in den mir zugänglichen Edititionen werden grundsätzlich nur griechische Großbuchstaben als Namen für mathematische Gegenstände verwandt. Dies ist nicht weiter verwunderlich, insofern Euklid noch keine Kleinbuchstaben zur Verfügung standen und die Kenntnis und Verwendung ausländischer Alphabete beschränkt gewesen sein dürfte.

Zeichensysteme Für moderne Texte stehen verschiedenste Zeichensysteme zur Verfügung: einerseits Alphabete im alltagssprachlichen Sinn, etwa das griechische, hebräische und lateinische, mit Groß- und Kleinbuchstaben; aber auch mathematische Symbole (die sich oft aus alphabetischen Zeichen ableiten, wie etwa $\int$, aber im allgemeinen nicht mehr als solche gesehen werden). Für diese Zeichen bietet es sich im

allgemeinen an, die entsprechende Repräsentation im *Unicode*-Standard (Unicode Inc. 2003) zu verwenden, der nach und nach alle jemals gebrauchten Zeichen enthalten soll.

Annotationsregel 9 (Zeichensystem) *Für Wechsel im Zeichensystem ist die Annotation von Math*ML *zu verwenden.*[17]

Schriftarten Nicht nur verschiedene Zeichensysteme, sondern auch verschiedene Schriftarten können in mathematischen Texten mit einer Bedeutung besetzt werden: so bietet das Satzsystem TEX etwa kalligraphische ($\mathcal{A}, \mathcal{B}, \mathcal{C}$) und Fraktur-Schrift ($\mathfrak{a}, \mathfrak{b}, \mathfrak{c}$) als Varianten der lateinischen Schrift, Fraktur, außerdem noch Varianten mit einem verdoppelten Stamm, die vor allem für Mengenbezeichnungen verwandt wird ($\mathbb{I}, \mathbb{R}$ oft ersetzt durch $\mathbb{I}, \mathbb{R}$).

Annotationsregel 10 (Schriftarten) *Für Wechsel in Schriftarten ist die Annotation von Math*ML *zu verwenden.*[18]

Diakritika Neben Schriftwechseln sind in der mathematischen Darstellung auch Diakritika weit verbreitet. Inhaltlich lassen sich zwei Arten von Diakritika unterscheiden, die wir *statische Diakritika* und *dynamische Diakritika* nennen wollen. Außerdem gibt es Diakritika, die keine Bedeutung haben und die wir *Pseudo-Diakritika* nennen wollen.

Statisch soll in diesem Zusammenhang bedeuten, daß das Diakritikum eine bestimmte Bedeutung hat, die feststeht (etwa «′» — ausgesprochen: „Strich“ — für die Ableitung von Funktionen: $f(x)$ und folglich $f'(x)$); in $\mathfrak{x}'$ ein statisches Diakritikum ′ also für eine Funktion, die auf diejenige Entität angewandt wird, für die $\mathfrak{x}$ steht. Neben der Ableitung ist $\bar{x}$ für den Mittelwert in der Statistik aller x ein Beispiel, das bereits andeutet, daß eine Definition der Semantik von Diakritika nicht immer einfach ist: x könnte hier für das Multiset der erhobenen Daten stehen, dies wird selten so vereinbart; ¯ ist dann die Operation, die das arithmetische Mittel der Daten berechnet. Eine ähnliche Vorgehensweise wählt Zinn (to appear : 108) für sein vip-System als eine „alternative Repräsentation“.

Dynamische Diakritika (meist Zahlen oder Buchstaben) dagegen stehen häufig für einen Teil einer Aufzählung $(a_1, \ldots, a_n, \ldots, a_m)$, aber auch $\sum$- und $\int$-Laufvariablen können ähnlich gedeutet werden. Diese dynamischen Diakritika sind offensichtlich selbst keine Funktionen, können aber als Argumente einer Funktion gedeutet werden, so daß

[17] alternativ sollte eine semantische Annotation möglich sein.
[18] Alternativ sollte eine semantische Annotation möglich sein (Kap. 8.6.1 auf Seite 79).

etwa a_1^i (wir nehmen einmal an, es stehe für ein bestimmtes Element einer Folge) als Anwendung der Funktion $\mathfrak{a}$ auf die Argumente 1 und i verstanden werden kann, die das entsprechende Folgenelement liefert. Dieses Vorgehen mag ein wenig kontraintuitiv scheinen; für Anwendungen des Summen- oder Produktzeichens ist jedoch eine Umformung in eine Funktionsschreibweise direkt einsichtig — und die Definition einer solchen Funktion eine der ersten Aufgaben in Informatik-Einführungen.[19] Und auch für die Exponentiation, etwa $6,48^2$, ist es üblich, in Programmiersprachen `exp(6.48,2)` zu schreiben.

Allerdings ergibt sich, daß gelegentlich nur bestimmte Elemente, z.B. einer Folge in einer Aufzählungsform, angegeben werden.

(43) $a_1, \ldots, a_m, a_{m+1}, \ldots, a_n$

Üblicherweise verfährt man so, wenn man über a_m, a_{m+1} Aussagen treffen möchte. Es ist nun einfach, eine formalisierte Semantik dieser Formel anzugeben, und auch eine Annotation des Textes mit vier oder fünf Diskursreferenten (nämlich: a_1, a_m, a_{m+1}, a_n und ggf. a als Referenten für die gesamte Aufzählung) ist recht einfach. Eine semantische Annotation, die auch die „richtigen“ Diskursreferenten zur Verfügung stellt, ist relativ schwierig, vor allem, wenn diese Annotation beliebige Fälle solcher Schreibweisen erfassen soll. Man müßte versuchen, eine Schablone zu definieren, welche aus einer entsprechenden Funktionsangabe eine solche Darstellung ableitet. Es fällt uns jedoch kein Weg ein, eine solche Schablone zu definieren, deren Anwendung einfacher wäre als der folgende, manuelle Weg, der von der OMDoc-Praxis inspiriert ist, Texte auf dem Niveau von Ableitungsschritten parallel formalisiert und natürlichsprachlich zu annotieren: Man annotiert die Formel mittels eines OMDoc-<FMP>-Elements mit einer Semantik entsprechend der Hintergrund-Theorie und mittels eines OMDoc-<CMP>-Elements mit einer Präsentationsform in MathML. Innerhalb des <CMP>-Elementes können dann auch diejenigen Diskursreferenten, über die gesprochen wird, entsprechend annotiert werden.

Annotationsregel 11 (statische Diakritika) *Für statische Diakritika wird die Annotation aus Math*ML *verwendet oder ggf. eine semantische Repräsentation mit Style-Sheets (vgl. Kap. 8.6.1 auf Seite 79), in der die Diakritika als Funktion auftreten.*

[19] Die Summenfunktion würde für einstelliges f umschrieben mit: $\sum_{i=0}^{2} f(i) = \mathrm{sum}(0, 2, f)$, also als Funktion höherer Ordnung; f kann beliebige Stellenzahl haben, vgl. etwa die Beispiele bei Zinn (to appear : 69).

Im ersten Fall muß die Bedeutung der Diakritika im Hintergrundwissen definiert sein, im zweiten die Präsentationsform der entsprechenden Funktion.

Annotationsregel 12 (dynamische Diakritika) *Analog zu Regel 11 verfährt man mit dynamischen Diakritika, die allerdings Funktionsargumente darstellen.*

Annotationsregel 13 (dynamische Diakritika in Aufzählungen) *Dynamischen Diakritika, die Diskursreferenten einführen, welche in der Beschreibung der Semantik nicht notwendig sind, aber auf die im späteren Text Bezug genommen wird, ordnet man mithilfe der* OMDoc*-Elemente <FMP> und mittels <CMP> eine Semantik und eine Präsentationsform in Math*ML *zu, in der die entsprechenden Diskursreferenten gemäß Regel 12 markiert werden.*

Ähnlich verfährt auch Zinn (to appear : 110f), indem er einen Diskursreferenten für die einzelnen Teilformeln einführt und einen für die Gesamtheit.

Ein interessantes Beispiel, das er (Zinn to appear : 111) zitiert, ist:

(44) We have $n = p_1p_2p_3 \ldots = q_1q_2 \ldots$, where the p and q are primes, no p is a q and no q is a p.

In diesem Falle steht p (ohne Index) offensichtlich für alle Faktoren p_i (mit Index i), welche durch die Funktion $\mathfrak{p}(i)$ geliefert wird.

Pseudo-Diakritika Als Pseudo-Diakritika möchten wir solche diakritischen Zeichen bezeichnen, die keine regelhafte Funktion haben, sondern sozusagen aus Verlegenkeit gewählt werden. So heißt ein Mittelpunkt in der Geometrie heute üblicherweise M; treten aber mehrere Mittelpunkte auf, so nennt man sie häufig M_1, M_2,... Wir schlagen vor, solche Diakritika als einfache Namen zu annotieren.

Annotationsregel 14 (Pseudo-Diakritika) *Diakritika, die keine regelhafte Funktion haben, werden mit der Präsentationsform aus Math*ML *als Bestandteile eines einfachen Namens annotiert.*

Eine Gefahr, diese Pseudo-Diakritika mit echten Diakritika zu verwechseln, besteht selten: Üblicherweise werden nämlich Entitäten mit „echten“ dynamischen Diakritika direkt hintereinander (in einer Aufzählung etwa) eingeführt, solche mit Pseudo-Diakritika jedoch nicht. „Echte“ statische Diakritika sollten im Hintergrundwissen definiert sein, und es sollte für den Annotator aus dem Zusammenhang klar sein, wann sie verwandt werden.

8.5.3.2. Moderne Namensgebung in der Geometrie

Nachdem wir das Grundsätzliche geklärt haben, können wir nachtragen, wie sich die Namensgebung in der Geometrie heute verhält. Seit EUKLID hat sich einiges geändert: Es ist heute üblicher, Entitäten mit einfachen (also nicht zusammengesetzten) Namen zu versehen. Ein Grund dafür ist sicherlich, daß heute, wie bereits erwähnt, Großbuchstaben und verschiedene Alphabete zur Verfügung stehen.

Insgesamt ist es auch üblich geworden, daß Namen eine Art Abkürzung darstellen oder aus einem Bereich um den abkürzenden Buchstaben gewählt werden: Mittelpunkte heißen M, Geraden g, f, h etc.

gerade Linien Für „gerade Linien“[20] verwendet man heutzutage immer noch Kombinationen aus zwei Buchstaben, allerdings mit diakritischen Zeichen:

Strecken Die Strecke an sich, als begrenzt durch die beiden Eckpunkte, wird durch $|AB|$ dargestellt.

Streckenlängen Ist die Streckenlänge gemeint, wird dies durch eine übergeschriebene Länge verdeutlicht: $\overline{AB}$.

Geraden Eine Gerade wird einfach nur durch AB bezeichnet.

Es ist allerdings auch üblich, für Strecken, Streckenlängen (v.a. Radien) oder Geraden einfache Namen aus Kleinbuchstaben zu vergeben: v.a. s (*Strecke*), g (*Gerade*), r (*Radius*).

Kreise Bei Kreisen ist es üblich, Mittelpunkt und Radius anzugeben; Angaben mit drei Referenzpunkten sind eher unüblich. Als Namen werden oft c oder k (*circle*, *circulus*, *Kreis*, κύκλοσ) verwandt.

Winkel Winkel werden heute häufig entweder so bezeichnet, wie wir es in Heibergs lateinischer Übersetzung der *Elemente* gesehen haben, $\angle ABC$,[21] oder man bezeichnet die Winkel mit kleinen griechischen Buchstaben; bei den inneren Winkeln eines Dreiecks vorzugsweise mit der griechischen Entsprechung des Scheitelpunktes (α für den Winkel $\angle CAB$).

[20] EUKLID spricht von εὐθεῖαι γραμμαῖ, und meint damit „breitenlose Länge“ (μῆκοσ ἀπλατέσ), begrenzt durch Punkte (πέρατα σημεῖα), die mit ihren Punkten auf gleicher Höhe liegt (*[…]* "τισ ἐξ ἴσου τοῖσ ἐφ'ἑαυτῆσ σημείοισ κεῖται.; hierzu vgl. HEATH 1920 : 118, Fn. 4 zu Def. I,4); insgesamt klingt dies eher nach Streckenlänge als nach Strecke.

[21] Allerdings schreibt man das Winkelzeichen dann vor jeden Winkel, also: $\angle ABC = \angle ACB$.

Eine Bezeichnung mit Winkelzeichen wäre in MathML zu beschreiben; vereinbarungsgemäß stellen die einzelnen Großbuchstaben in dieser Formel dann Namen dar.

8.6. Formeln im mathematischen Text

Eine wichtige Erscheinung in mathematischen Texten ist die Mischung von Formeln und natürlichsprachlichem Text. Diese Arbeit beschäftigt sich vornehmlich mit dem natürlichsprachlichen Anteil der mathematischen Beweise, jedoch ist dieser so eng mit den darin vorkommenden Formeln verwoben, daß sie oftmals kaum zu trennen sind.

Wichtig ist, daß in Formeln auch Referenten eingeführt werden können, auf die später Bezug genommen wird. Die Einführung der entsprechenden Referenten muß aus der Annotation ersichtlich sein.

Für die Annotation von Formeln ist das Präsentations-Markup von MathML eine Ausgangsbasis; allerdings müssen die neu eingeführten Diskursreferenten besonders markiert werden, auf die im folgenden Text Bezug genommen wird.

Für ProofML reicht es zunächst, davon auszugehen, daß Formeln ein eigener im fortlaufenden Text durch ein <formula>-Tag angezeigt werden; einer weiteren Annotation bedarf eine Formel nicht, sofern keine Diskursreferenten ein- oder fortgeführt werden; werden allerdings Diskursreferenten eingeführt, so müssen sie genauso annotiert werden, wie dies im fortlaufenden Text der Fall wäre, mit einem Typ und einer ID. Da Formeln für ganze Sätze, aber auch für einzelne Objekte stehen können, ist es möglich, sie in einem <expression>-Element zu klammern, welches anzeigt, welche konkrete Rolle eine Formel spielt.

Inhalt einer Formel kann ein <FMP>-Element aus OpenMath/OMDocsein. In den Beispielen im Anhang haben wir auf eine solche Darstellung verzichtet, damit in dieser Arbeit eigentlich interessanten Annotationen für natürlichsprachliche Anteile mathematischer Texte zur Geltung kommen.

Annotationsregel 15 (Formeln) *Formeln stehen in einem <formula>-Element. Dieses kann ein <FMP>-Element enthalten. Die Rolle einer Formel kann man anzeigen, indem man sie in ein <expression>-Element setzt.*

8.6.1. Regelhafte Darstellung mathematischer Formeln

Die Style-Schablonen in OMDoc (Kohlhase to appear : Kap. 22) können die Grundlage einer Annotation sein, die stärker an der Semantik der Texte orientiert ist, aber eine konsistente Präsentation der Semantik erzeugen kann, die dem Druckbild entspricht.

Für die Absichten der ProofML-Annotation eignet sich eine solche semantische Annotation jedoch aus zwei Gründen im Prinzip nicht:

Einerseits steht die Verwendung einer rein semantischen Annotation im Widerspruch zur Absicht von ProofML, den Text lediglich anzureichern: eine semantische Annotation ist in jedem Fall eine Transformation des bestehenden Textes, so daß der Aspekt des Layouts, der im Druckbild präsent ist, durch den der Semantik ersetzt wird, die lediglich im günstigen Fall aus dem Druckbild interpretiert werden kann.

Andererseits liegen die Unzulänglichkeiten der Style-Sprache genau dort, wo das Layout dazu beiträgt, daß bestimmte Referenten eingeführt werden, die in der Denotation des Formelausdrucks keine besondere Rolle spielen, etwa in Aufzählungen (vgl. Kap. 8.5.3.1 auf Seite 74). Problematisch sind auch Inkonsistenzen, die in der Notation auftreten können, aber natürlich durch Styles gerade nicht erfaßt werden (da Styles die regelmäßige Präsentation erfassen).

Wir geben hier kein Beispiel für die Anwendung dieser Schablonen, sondern verweisen auf das schon angesprochene Kapitel 22 der OMDoc-Dokumentation.

8.6.2. Mehrdeutigkeit von Funktionssymbolen

Ein notorisches Problem in der Betrachtung mathematischer Notation ist die Mehrdeutigkeit von Funktionssymbolen, etwa in der Formel $f(x)$ (vgl. Zinn to appear : 70): sie können für die Funktion stehen, für einen konkreten Funktionswert und können natürlich auch als Variablennamen höherer Ordnung gebunden sein wie f im folgenden Beispiel von Zinn (to appear : 70):

(45) A function $f(m)$ is said to be multiplicative if $(m, m') = 1$ implies $f(mm') = f(m)f(m')$.

In ProofML kann diese Mehrdeutigkeit dadurch aufgelöst werden, daß der entsprechende `type` für <`expression`> angegeben wird, nämlich "`function-definition`" und

"function" für die ersten beiden Fälle, oder der Ausdruck als Variable annotiert wird.[22]

8.7. Verben

Verben werden im wesentlichen genauso annotiert wie Nominalphrasen. In unseren Beispielen haben wir als Präfix für das Attribut type immer predicate: verwendet.[23]

Argumente von Verben sind, wie üblich, entweder Kinder und Geschwister in der Reihenfolge des Auftretens oder müssen mit <arg>-Verweisen angegeben werden. Der Numerus eines Verbs muß nicht gesondert annotiert werden, da er keine semantische Relevanz hat, sondern im allgemeinen mit dem Numerus des Subjekts des Satzes kongruiert, dessen Numerus bei seinem Auftreten bereits annotiert ist.[24]

Wie gesagt, kann es sein, daß es günstig ist, Phraseologismen als ein Verb zu annotieren.

Da es auch Anaphern gibt, die sich auf Verben beziehen, können auch diese ein id-Attribut erhalten, auf das sich das refid der entsprechenden Anapher bezieht.

8.8. Negation

In den mathematischen Texten, die uns vorlagen, kommen vor allem Satznegationen vor, keine fokussensitive Konstruktionen.

Wir können also Negationen ähnlich behandeln wie Quantoren. Sollte eine fokussensitive Konstruktion auftreten, so kann man der Negation als Argument den entsprechenden semantischen Fokus zuweisen.

Annotationsregel 16 (Negation) *Die Negationspartikel wird als <neg> annotiert. Es erhält ein Attribut* id*, das dazu dient, ihm einen <scope> zuordnen zu können (wie bei Quantoren mit* <scope-of refid="*neg-id*">*.*

Die Besonderheit, daß zumindest im Deutschen und Niederländischen, wenn Negation und indefinite Nominalphrase zusammentreffen, „kein“, „geen“ erscheinen können,

[22] Für die Definition einer Funktion verwendet man natürlich die Bindungs-Operatoren von OpenMath oder MathML, d.h. eine statische Bindung.

[23] Zur Bedeutung und Arbitrarität dieses Wertes gilt natürlich entsprechend das in Kap. 8.4.2 auf Seite 58 Gesagte.

[24] Wir haben im Rahmen dieser Arbeit keine größeren Beispiele in einer Sprache annotiert, die es erlaubt, Subjektpronomina wegzulassen. Es scheint aber sinnvoll, in solchen Fällen das Subjekt explizit durch einen Verweis mit einem <arg>-Element aufzulösen.

```
<expression type="object:straight-line" id="AB">AB</expression>
...
<expression refid="AB">Diese Strecke</expression>
```

Abb. 8.7: **Koreferenz**

auch wenn kein Existenzquantor in der Übersetzung dieser Phrase in eine prädikatenlogische Sprache vorkommt (46), wird in ProofML so aufgelöst, daß kein zwar als <det> annotiert wird, aber type="neg" erhält.

(46) Also ist *ABC* kein gleichseitiges Dreieck.

8.9. Koreferenz

Koreferenz bezeichnet den Sachverhalt, daß sich zwei natürlichsprachliche Ausdrücke auf dasselbe (sei es ein Gegenstand oder ein Sachverhalt) in der Realität beziehen.

Wir können Fälle, in denen Koreferenz durch grammatische Mittel hergestellt wird (*Kohäsion*), von solchen unterscheiden, in denen Koreferenz „durch den Zusammenhang", durch die semantische Verknüpfung des Textes erzeugt wird (*Kohärenz*). Es scheint allerdings nicht geraten, für diese beiden Fälle verschiedene Annotationsregeln einzuführen.[25]

Koreferenz ist also in der Annotation vor allem auf der Ebene der Nominalphrasen (einschließlich der Pronomina) zu erfassen. Dies geschieht, indem in den entsprechenden entsprechenden <expression>- oder <name>-Elementen das Attribut refid mit dem Wert des Attributs name des ersten Elements im selben Textabschnitt belegt wird, das mit ihm koreferent ist. Ein Beispiel findet sich in Abb. 8.7.

8.9.1. 'Split Antecedents'

'Split antecedents' nennt man das Phänomen, bei dem sich Anaphern auf zwei getrennt stehende Antezendentia bezieht (47).

[25] Bemerkenswert ist sicherlich, daß Mathematiker eine Technik entwickelt haben, Koreferenz anzuzeigen, die die Ambiguitäten, die im alltäglichen Sprachgebrauch entstehen (vgl. hierzu auch Ranta 1999), vermeiden: den Gebrauch von Variablen-Namen. Sie haben den Pronomina und den anaphorischen Nominalphrasen voraus, daß ihr Bezug in einem Kontext eindeutig ist.

(47) If two triangles have $[_{NP}$ two sides$]_1$ equal to $[_{NP}$ two sides$]_2$ respectively, and have the angles contained by $[_{NP}$ the equal straight lines$]_{1+2}$ equal then they also have the base equal to the base, the triangle equals the triangle, and the remaining angles equal the remaining angles respectively, namely those opposite the equal sides.

In ProofML stellen *split antecedents* kein prinzipielles Problem dar. Sie sind so zu annotieren, daß der Anapher als Argumente alle Antezedenten zugeordnet werden. Ein Beispiel zu (47) findet sich in Abb. 8.8 auf der nächsten Seite.

Es bietet sich allerdings an, ein eigenes Element zu verwenden, um die Koreferenz zu mehreren Antezendentia zu annotieren, da (theoretisch) sehr viele Antezendentia möglich sind und die Einführung unbegrenzt vieler Attributen `refid2`, `refid3`, `refid4`,... in XMLnicht möglich ist und die Regelhaftigkeit der Attributnamen für die Verarbeitung des einzelnen Textes nicht nutzbar gemacht werden kann.

Wir führen daher das Element <`coreference`> ein, das mehrere <`arg`>-Kinder haben kann, die auf die Antezedentia der entsprechenden Ausdrücke verweisen. Man kann die Angabe der Koreferenz mit dem Attribut `refid` als eine Abkürzung für ein <`coreference`>-Element mit nur einem <`arg`>-Kind sehen, die in der Mehrzahl der Fälle ausreicht.

Annotationsregel 17 (Split Antecedents) *Besteht Koreferenz zu mehreren Objekten (*Split Antecedents*), so enthält die entsprechende Phrase als Kind-Element ein* <`coreference`>*-Element, das in* <`arg`>*-Elementen Verweise auf diese Objekte enthält.*

8.9.2. Pronomina

Pronomina werden annotiert wie Nominalphrasen (d.h. über ein `refid`-Attribut wird ihr Bezug hergestellt.

Reziprokpronomina (*einander*, *each other*) werden mit dem Attribut `ref-type="recip"` markiert.

8.9.3. Bridging

Von Bridging im allgemeinen haben wir bereits gesprochen. Wir annotieren Bridging ähnlich wie Existenzbeweise. Für diese ist es in den Editionen der *Elemente* ist üblich,

```
<expression type="ex" form="quant" binds="S1">
  <det>two</det>
  <restr>sides</restr>
  <scope>
    <scope-of qvar="S1"/>
    <scope-of qvar="S2"/>
    <expression type=predicate:equal">
      <arg name="S1"/>
      <arg name="S2"/>
      <word>equal to</word>
    </expression>
    <expression type="ex" form="quantor" binds="S2">
      <det>two</det>
      <restr>sides</restr>
    </expression>
    <expression type="object" form="NP">
      <coreference>
        <arg name="S1"/>
        <arg name="S2"/>
      </coreference>
      <word>the equal sides</word>
    </expression>,
...
```

Abb. 8.8: **Die *Split Antecedents* aus (47) in ProofML.**

die Möglichkeit einer Konstruktion dadurch anzuzeigen, daß der entsprechende Satz des Textes angegeben wird, der sie erlaubt.

(48) a. *Ἐπεζεύχθω γὰρ ἀπὸ τοῦ A σημείου ἐπὶ τὸ B σημεῖον εὐθεῖα ἡ AB, καὶ συνεστάτω ἐπ' αὐτης τρίγωνον ἰσόπλευρον τὸ ΔAB, [...]*

b. ducatur enim a puncto *A* ad *B* punctum recta *AB* [*αἴτ 1*], et in ea construatur triangulus equilaterus $\triangle \Delta AB$ [prop. I], [...]

prop. I verweist darauf, daß im ersten Beweis der Elemente die Existenz des entsprechenden Dreiecks bewiesen wurde.

αἴτ 1 verweist auf das erste Postulat.

Die Editionen lesten hier also sozusagen der Forderung von Ranta (1996, 2002) Folge, daß für den Verweis auf bestimmte Punkte die entsprechenden Existenz-Präsuppositionen „bewiesen" werden müssen. Dies mag für den allgemeinen Fall eine sehr starke Forderung zu sein,[26] für den Zusammenhang der mathematischen Theorie ist er wohl zu verwenden.

Allerdings kommen wir gelegentlich an Stellen, an denen ein regelrechtes Beweisen schwerfällt: Wieso ist der Punkt *A* (mit genau diesem Namen!) eingeführt, wenn vom Dreieck *ABC* die Rede war?

Auch wenn Euklid es nicht für nötig hält, zu erklären, daß ein Dreieck drei Eckpunkte[27] hat und die Namensgebung für Dreiecke nicht explizit einführt, gehen wir davon aus, daß dieses Wissen im Hintergrund vorhanden sein muß. Wir können also aufgrund der Definition des Dreiecks jeden beliebigen Punkt und jede beliebige Seite des Dreiecks herbei*bridgen* (denn offensichtlich nehmen wir deren Existenz an, wenn wir die Existenz des Dreiecks annehmen). Im Beispiel haben wir als Namen für die Regel, die aus dem Namen eines Dreiecks erlaubt, die entsprechenden Namen der Punkte zu erschließen `"naming:triangle"` angenommen, also Parameter sind die ID des Dreiecks (dessen Namen seit seinem ersten Auftreten zum Hintergrundwissen hinzugekommen sein muß), die Regeln und der Punkt *A*, also im wesentlichen das Wissen, daß auch einem

[26] Immerhin kann die Ableitung auch genau umgekehrt erfolgen; vgl. die nähere Erläuterung des Besprochenen durch definite NPs in Zeitungsartikeln.

[27] In El., Def. 19 führt er die *σχήματα εὐθύγραμμά* („Geradlinige Figuren") ein, sondert aus diesen u.a. die *τρίπλευρα* („dreiseitigen") aus und aus diesen wiederum in Def. 20 das *ἰσόπλευρον* („gleichseitige"), das *ἰσοσκελές* („gleichschenklige") und das *σκαληνὸν τρίγωνον* („schiefe") und in Def. 21 das *ὀρθογώνιον* („rechtwinklige", das *ἀμβλυγώνιον* („spitzwinklige") und das *οξυγώνιον τρίγωνον* („stumpfwinklige") — von der Benennung und den Punkten ist keine Rede. Aber Euklid dachte sicherlich noch nicht an die maschinelle Verarbeitung seiner Beweise.

menschlichen Leser des Beweises vorliegen muß, wenn er wissen will, wie er vom Dreieck *ABC* auf den Punkt *A* kommen soll.

Annotationsregel 18 (Bridging) *Ist ein Referent durch Bridging verfügbar, so wird dies durch ein* <bridged-from>*-Element angezeigt, dessen Attribut* by *als Wert den Namen der Regel erhält, aufgrund derer gebridget wurde. Argumente für diese Regel können mittels* <arg>*-Elementen angegeben werden.*

8.9.4. Sätze als Diskursreferenten

Es ist in mathematischen Texten üblich, auf Sätze zu verweisen, wenn diese als Rechtfertigung für einen Schluß dienen.

Während Euklid zur Markierung dieses Bezugs noch das Zitat einsetzt, verwendet die moderne Mathematik meist explizite Benennung und Verweisung auf Sätze.[28]

Wir müssen also in ProofML einen Mechanismus vorsehen, der eine explizite Benennung von Sätzen und einen Verweis auf sie erlaubt, mit anderen Worten: auch Elemente der Art <expression type="proposition"> können ein id-Attribut erhalten. Verweise auf Sätze können zu verschiedenen Zwecken erfolgen.

Eine Art des Verweises würde Kohlhase (to appear) wahrscheinlich unter „Meta-Kommentar“ fassen:

(49) Wie bereits in der Einleitung zu diesem Kapitel ausgeführt, gewinnen wir aus 2.4 (für höchstens abzählbares *S*) bzw. 3.3 (für beliebiges *S*) die Vollständigkeit des Sequenzenkalküls: . . . (Ebbinghaus et al. 1996 : 92)

Dieser Verweis stellt (Makro-)Kohärenz innerhalb des Textes her, indem die Skizze vom Anfang des Kapitels (Ebbinghaus et al. 1996 : 81) weiter ausgeführt wird; inhaltlich fügt der Verweis an sich nichts Neues hinzu.

Verweise zwischen Sätzen („Sätze“ hier verstanden als größere logische Einheiten, Lemmata, Sätze etc.) erfolgen geregelt über den „Namen“ des Satzes, bei Euklid auch oft über ein wörtliches Zitat. Der Name eines Satzes kann auch aus einer Nummer oder einer Kategorie, etwa „Lemma“, „Satz“, bestehen. In ProofML kann dies einfach über die Attribute id (bei der Einführung des Satzes) und refid (bei Verweisen) geregelt werden. Im Gegensatz zu Namen einfacher NP-Diskursreferenten stehen Namen von

[28] Es ist interessant, daß mathematische Texte wohl die Prämissen von Schlüssen angeben, nicht aber die angewandte Schlußregel.

```
<expression type="object:triangle" id="ABC">
  <det>das</det>
  <expression type="property">
    Dreieck
  </expression>
  <name form="compound">
    <name>A</name> <name>B</name> <name>C</name>
  </name>
</expression>

<expression type="object:point id="A">
  <bridged-from by="naming:triangle">
   <arg refid="ABC"/>
   <arg refid="A/>
  </bridged-from
  <det>der</det>
  <expression type="property">Punkt</expression>
  <name">A</name>
</expression>

<expression type="object:point id="ABC">
  <bridged from="AB" by="elem-prop-10"/>
  <!-- der entsprechende Beweis bezieht sich auf die Halbierung einer
       gegebenen Strecke -->
  <det>der</det>
  <expression type="property">Mittelpunkt</expression>
  <name>M</name>
  <expression type="object:straight-line id="AB">
    <bridged-from by=naming:triangle">
     <arg refid="ABC"/>
     <arg refid="def-I-19/>
     <!-- Def. 19 in Elemente I: Vielecke -->
    </bridged-from>
    <det>der</det>
    <expression type="property">
      <arg refid="ABC"/>
      Seite
    </expression>
    <name form="compound">AB</expression>
  </expression>
</expression>
```

Abb. 8.9: **Bridging**

Sätzen diesen meist voran. ProofML verwendet für benannte Sätze Elemente der Art <expression type="named-proposition">, die wiederum einen Namen und ein Element vom Typ <expression type="proposition"> enthält.

8.10. Formalisierte Metasprache

Bei Ebbinghaus et al. (1996) ist bereits zu erkennen, daß die mathematische Sprache dazu tendiert, formelhaft zu werden: Zwar führt Ebbinghaus et al. (1996) „gdw“ als Abkürzung für „genau dann wenn“ ein, allerdings stehen bei „gdw“ anschließend lediglich Hauptsätze; es gibt sogar eine Form „:gdw“, die Definitionen einleitet, analog zu := oder $=_{\text{def}}$ in der Formelsprache.[29]

(50) a. $\mathcal{I}^{\Phi} \vDash \exists x_1 \ldots \exists x_n \varphi$ *gdw. es gibt* $t_1, \ldots, t_2 \in T^S$ mit $\mathcal{I}^{\Phi} \vDash \varphi \frac{t_1 \ldots t_t 2}{x_1 \ldots x_n}$
(Ebbinghaus et al. 1996 : 83)

b. $t_1 \sim t_2$ *:gdw* $\Phi \vdash t_1 \equiv t_2$.
hbox(Ebbinghaus et al. 1996 : 82)

c. $\Phi \vDash \varphi$ gdw jede Interpretation, die Modell von Φ ist, ist Modell von φ.
(Ebbinghaus et al. 1996 : 38)

Dies läßt sich wohl so erklären, daß „gdw“ hier nur noch einen Junktor darstellt, der Propositionen (paradigmatische Form: Hauptsätze) verbindet — keine Konjunktion mehr, die Sätze verbindet.[30]

Dies ist zwar durchaus kurios, kann aber genauso annotiert werden, wie natürlichsprachliche Konjunktionen auch.

8.11. Ellipsen

In den Texten des Korpus, das wir für die Erstellung von ProofML verwandt haben, kamen nur eine Ellipse vor (das Beispiel haben wir bereits als (47) genannt).

(51) If two triangles have two sides equal to two sides respectively, and have the angles contained by the equal straight lines equal then they also have the base

[29] Man kann noch weiter gehen und sagen, daß in (50-a) auch „es gibt“ formelhaft ist und keinen regelrechten Hauptsatz einleitet.

[30] In manchen Büchern ist die Meta-Sprache so weit formalisiert, daß ihre Symbole in einem Anhang erklärt werden (z.B. Essler 1969 : 322f).

> equal to the base, the triangle equals the triangle, and the remaining angles equal the remaining angles respectively, namely those opposite the equal sides.

Es ist offensichtlich, daß man bei „those“ „angles“ ergänzen muß. Wir schlagen vor, in diesen Fällen durch ein leeres <`expression`>-Element (zumindest eines, das keinen Text enthält) das Wort nachzutragen, das „fehlt“. (ähnlich wie wir mit Lücken verfahren, vgl. Kap. 9.3 auf Seite 96). Mit einem `refid`-Attribut kann man auf die ID der parallelen Elemente verweisen, von denen man bei der Ergänzung ausgeht (also hier etwa „angles“).

Annotationsregel 19 (Ellipsen) *Ellipsen sind zu annotieren, indem Ausgelassenes durch ein leeres* <`expression`>*-Element ergänzt wird. Ein* `refid`*-Element kann Parallelen andeuten.*

9. Annotation auf logischer Ebene

Dieser Abschnitt behandelt die Annotation der logischen Verhältnisse innerhalb mathematischer Beweise in ProofML.

Dabei gehen wir zunächst auf die Annotation der logischen Struktur der Verknüpfung von Propositionen ein. Dieser Abschnitt ist im wesentlichen unabhängig von der Annotation auf phrasaler Ebene und lehnt sich daran an, wie in OMDoc die entsprechenden Relationen zwischen formalisierten Propositionen beschrieben werden.

In Kap. 9.2 auf Seite 93 werden wir dann erläutern, was wir von OMDoc übernehmen und was wir daran verändern müssen. Eine entsprechende DTD findet sich im Anhang.

9.1. Propositionen oder Sätze?

Die logische und die sprachliche Form gehen nicht immer konform. So können Sachverhalte nominalisiert ausgedrückt werden oder in ganzen Sätzen. Nebensätze können Hauptsachen enthalten — und sie können auch Informationen enthalten, die eine andere Rolle in der logischen Struktur des Satzes enthalten: z.B. Prämissen oder Folgerungen.

Da in unserer Annotation auch die logische Struktur wesentlich ist, müssen wir also Sätze in Propositionen einteilen, die der sprachlichen Struktur widersprechen, sofern diese eine bestimmte Rolle in der logischen Struktur spielen.

Die Einteilung von Propositionen nehmen Koepke und Schröder (2003) im wesentlichen ähnlich vor, wie wir es hier beschreiben, ebenso die Markierung logischer Konjunktionen; allerdings thematisieren sie diese Einteilungen noch nicht explizit.

9.1.1. Einteilung in Propositionen

Es ist notwendig, den Text in Propositionen einzuteilen. Das wichtigste Kriterum lautet:

Annotationsregel 20 (Propositionen und Rollen) *Propositionen, die in Relation zueinander stehen, füllen Rollen in Schlußregeln aus (Prämisse, Sukzedenz, s. Kap. 9.2 auf Seite 93).*

Dennoch kann es nicht ganz einfach sein, zu entscheiden, inwieweit eine einzelne Prämisse vorliegt, die aus vielen Konjunkten besteht, oder mehrere Prämissen.

Es scheint sinnvoll, der sprachlichen Struktur des Beweises soweit zu folgen, wie möglich. Da der Satz allgemein als eine wichtige Einheit von Texten wahrgenommen wird, legen wir fest:

Annotationsregel 21 (Maximal-Grenze für Propositionen) *Als eine Proposition (*`<expression type="proposition">`*) ist maximal ein Satz zu annotieren, der durch ein Satzendezeichen beendet wird.*

Eine minimale Proposition ist, was in der Prädikatenlogik als solche verwandt wird:

Annotationsregel 22 (Minimal-Grenze für Propositionen) *Eine minimale Proposition im natürlichsprachlichen Text besteht aus einem (logischen) Prädikat oder Quantor und seinen Argumenten.*

Für Mischsprache aus Formeln und Text ergibt sich:

Annotationsregel 23 (Formeln als Propositionen) *Formeln haben grundsätzlich den Wert von Propositionen, sofern sie als mathematische Aussagen auftreten können.*

Die letzte Regel besagt, daß etwa a (als einfacher Variablenname) keine Proposition sein kann, wohl aber $a = 42$.

Eine Schwierigkeit ergibt sich bei der Behandlung von Kon- und Disjunktion. Diese können einerseits Propositionen, andererseits Entitäten verbinden.

Da für beide Funktionen dieselben Wörter verwandt werden und es in der Annotation offensichtlich ist, welches die Argumente der (linguistisch so genannten) Konjunktionen sind, behalten wir für die Konjunktion `form="conj"` für beide Fälle bei.

Annotationsregel 24 (Konjunktion) *In der Annotation ist Konjunktion auf allen Ebenen zu annotieren, indem das Attribut* `form="conj"` *gesetzt wird.*

Für die Disjunktion zwischen Namen oder Objekten scheint es günstiger, den Satz zu zwei (oder mehr) Propositionen zu ergänzen (vgl. Kap. 8.11 auf Seite 87). Allerdings ist ein solcher Fall in den untersuchten Texten nicht aufgetreten. Auch Zinn (to appear) und Ranta (1999) geben kein entsprechendes Beispiel.

Wie bereits beschrieben (vgl. auch Kap. 8.9.1 auf Seite 81), ist der Unterschied zwischen kollektiver und distributiver Konjunktion wichtig.

Annotationsregel 25 (Typen der Konjunktion) *Es ist zu unterscheiden zwischen kollektiver und distributiver Konjunktion.*

Eigenschaften in Appositionen oder Adjektivphrasen Eine Schwierigkeit tritt auf bei der Koordination (in)definiter NPs mit Adjektivphrasen oder Appositionen.

Wir beginnen mit dem Fall einer definiten NP. Die klassische Übersetzung (allerdings mit dynamischer Interpretation, siehe Kap. 3.3 auf Seite 18), wie sie in (52) dargestellt ist, zeigt zwei Propositionen, die durch eine Konjunktion verbunden sind.[1]

(52) a. Der Punkt A liege auf g.
b. $\exists A : ((\text{punkt}'A) \wedge (\text{lieg}'g)A)$

Wir haben in unseren Beispielen allerdings in solchen Appositionen immer „Punkt“ und andere Eigenschaften als `<expression type="property:point">` etc. annotiert, und eine explizite Konjunktion ist nicht vorgenommen. Dies läßt sich dadurch rechtfertigen, daß in keinem Fall die Eigenschaft alleine als Prämisse oder Annahme für einen Schluß auftritt. Der einzige Fall, wo dies auftritt, ist beim Bridging, und dort ist ein Verweis auf die entsprechende Eigenschaft unproblematisch.[2]

Im Falle einer indefiniten NP ist die Interpretation eines solchen Satzes in mathematischen Beweisen — wie sie uns vorgelegen haben — generisch:

(54) Zwei nichtparallele Geraden AB und DE schneiden einander.

Bei solchen Sätzen ist es zwei mögliche Verfahrensweisen:

Es ist möglich, die Nominalphrase (sonst annotiert wie immer) in einem Element der Art `<expression type="proposition">` zu kapseln und den restlichen Satz (nachdem

[1] Den Existenzquantor verwenden wir, weil wir davon ausgehen, daß A als Diskursreferent eingeführt wird. Daß dennoch oft der bestimmte Artikel gesetzt wird, erklärt sich aus dem Gebrauch eines Namens.

[2] Analog verhält es sich mit Namen; man könnte die Semantik von (52) auch folgendermaßen wiedergeben:

(53) $\exists x : ((\text{punkt}'x) \wedge (\text{lieg}'y)x)$

Dann würde also auch die Benennung als Prädikat zu Variablen aufgefaßt, die eingeführt werden (und y wäre die Variable, deren Name g ist). Es ergibt sich, daß auch der Name niemals als Grund für einen Schluß ausreicht und daher die Benennung nicht als eigene Proposition annotiert werden muß.

man dem Prädikat das entsprechende Argument mittels <arg> zugewiesen hat) mit der NP durch eine Implikation zu verbinden.

Die elegantere Lösung besteht darin, die NP als generalisierten Quantor zu annotieren, wobei die Eigenschaft als Restriktor fungiert. Der Quantor (v.a. die Semantik des Determinierers) muß natürlich im Hintergrundwissen definiert sein.

Für Adektivphrasen verfährt man sinngemäß ebenso wie für beigeordnete Eigenschaften. Für Adjektivphrasen, die Eigenschaften beigeordnet sind, nehmen wir — wie üblich — an, daß sie mit einer Konjunktion mit ihnen verbunden sind und sie sich ein Argument teilen (das durch `id` oder `refid` in der übergeordneten Phrase gegeben ist). Weitere Argumente werden wie üblich angezeigt.

Zusammenfassungen Es kann auch sinnvoll sein, mehrere Wörter unanalysiert zusammenzufassen, etwa „gleichschenklige Dreiecke“ oder Euklids «εὐθεῖαι γραμμαῖ» („gerade Linien“), vor allem, wenn einzelne Bestandteile keine semantische Funktion mehr haben wie im letzteren Fall (daher fällt γραμμή, „Linie“, meist weg).

Relativsätze, Partizipialkonstruktionen, Präpositionalphrasen etc. In ProofML werden Relativsätze, Partizipialkonstruktionen u.ä. genauso annotiert wie Hauptsätze; wir gehen davon aus, daß sie mit ihnen durch eine (logische) Konjunktion oder eine Implikation verbunden sind, werden als eigenständige Propositionen annotiert. Relativpronomina (soweit vorhanden) werden wie andere Anaphern annotiert, die sich auf das näher bestimmte Objekt beziehen. Auch semantische Präpositionen, also Präpositionen, die nicht nur Präpositionalobjekte angeben, werden analog annotiert.

9.1.2. Logische Verknüpfungen zwischen Propositionen

Sind verschiedene Propositionen jedoch verknüpft, etwa mit Kon- oder Disjunktion, so müssen wir diese Verknüpfungen annotieren können. Diese werden im `form`-Attribut des <expression>-Elements mit den entsprechenden Namen angegeben.

Die wesentlichen Verknüpfungen sind Kon- und Disjunktion, Implikation und Äquivalenz. Weitere logische Relationen sind uns in unserem Korpus nicht begegnet, aber es ist offensichtlich, wie ProofML gegebenenfalls zu erweitern wäre.

Bei Implikationen ist zu beachten, daß die Form nicht immer $A \rightarrow B$ („wenn, dann“) lautet, sondern auch die Form $A \leftarrow B$ („gilt, wenn“) vorkommen kann. Für dieses

Problem gibt es zwei Möglichkeiten: einerseits könnte man die Reihenfolge *Implikant Implikat* vorschreiben und notfalls mittels <arg/>-Elementen wiederherstellen; andererseits kann man zusätzlich zum form="implication" für den gewöhnlichen Fall eine form="rimplication" (*r*echtsseitige Implikation) einführen. Das letztere Verfahren ist prinzipiell als Vereinfachung der Annotation zu verstehen, die mit einem form="implication" und einer <arg/>-Korrektur äquivalent ist; es ist nur gerechtfertigt, wenn eine rechtsseitige Implikation häufig vorkommt und man die Lesbarkeit des Textes für einen menschlichen Annotierer erhöhen möchte.

Wir sehen in ProofML als Relationen vor: "conj", "disj", "implication", "rimplication" und "equivalence".

9.2. Schlußregeln

Wir nehmen die Beweis-Annotation von OMDoc (vgl. Kap. 4.1.5.1 auf Seite 28) als Grundlage unserer Annotation; wir übernehmen also auch die Annotation für Schlußregeln von OMDoc (s. Kap. 4.1.5 auf Seite 28; Kohlhase to appear : Kap. 19).[3]

Daher besitzt ProofML die Elemente <hypothesis> für Hypothesen, <assertion> für Behauptungen, <proof> für Beweise (üblicherweise für Behauptungen),<method> für Schlußregeln, <derive> für Beweisschritte, <conclude> für den letzten Schluß eines Beweises,[4] <metacomment> für Kommentare zum Beweisgeschehen, die keinen direkten Beitrag zum logischen Verlauf des Beweises leisten; die genannten Elemente haben in ProofML dieselbe Syntax und Semantik wie in OMDoc, wir müssen allerdings die Reihenfolge der Kindelemente flexibilisieren und zusätzlich einige weitere Elemente als Inhalt erlauben.

Wir müssen dieses Repertoire nämlich ein wenig erweitern, denn Sätze können den Beweisfluß unterbrechen oder können die Beweisführung kommentieren, etwa indem sie eine Schlußregel angeben (s. Abb. 9.1 auf der nächsten Seite). In OMDoc ist eine solch tiefgehende Analyse der Semantik natürlicher Sprache noch nicht vorgesehen.

[3] Dieses Vorgehen ist im wesentlichen äquivalent zum Vorgehen von Koepke und Schröder (2003), soweit es aus dem relativ kurzen Beispiel ersichtlich ist.

[4] Kohlhase (to appear : Kap. 19.1) schließt für den Inhalt von <conclude>-Elementen <FMP>-Elemente aus, die ja in OMDoc den Inhalt einer Aussage repräsentieren; da in ProofML auch die <CMP>-Elemente mathematische Aussagen beinhalten können, fallen <conclude> mit einem verschachtelten <formula>-Element und <derive> mit einem <FMP>-Element sozusagen zusammen. Es bleibt aber die Beschränkung, daß ein <conclude>-Element an das Demonstrandum (üblicherweise eine <assertion>) gebunden ist, dessen Beweis es abschließt; wenn es einen Inhalt hat, muß er also dem des Demonstandums entsprechen.

```
<derive id="III-4.6-a">
  <nl-method  xref="Induktion">
     <keyword>durch</keyword> Induktion
     <arg><keyword>über</keyword>
       <expression type="object">
         <det>den</det>
         <expression type="property:structure">Aufbau
           <arg>
             <det>der</det>
             <expression type="property:S-Terme" nr="pl">
                S-Terme.
             </expression>
           </arg>
         </expression>
  </nl-method>
  ...
</derive>
```

Abb. 9.1: **Schlußregel (Ebbinghaus et al. 1996 : 39)**

Solche Aussagen sind in ProofML zu annotieren, indem das Element <nl-method> verwandt wird (eine Erweiterung des <method>-Elements von OMDoc), das eine Referenz auf die Schlußregel (analog zum OMDoc-Element <method>) erhält, die im Satz erwähnt wird. Eine Illustration hierfür findet sich in Abb. 9.1. Eventuelle Parameter der Regel können, wie immer in ProofML, mit <arg>-Elementen annotiert werden.

Beweispläne (s. Kap. 4.2 auf Seite 36) können wir in unserem Zusammenhang als eine Art erweiterte parametrisierte Regeln verstehen. Ein Beispiel für die Annotation einer Induktion findet sich in Abb. 9.1.

Von OMDoc übernehmen wir die Unterscheidung zwischen <CMP>-Elementen, die natürlichsprachliche Texte enthalten, und <FMP>-Elementen, welche formalisierte Darstellungen enthalten. Im allgemeinen kommen <FMP>-Elemente in ProofML relativ selten vor; vor allem, wenn Lücken (Kap. 9.3 auf Seite 96) gefüllt werden, sind sie notwendig. Aber sie können auch auftreten, wenn Stylesheets für die regelhafte Darstellung formalisierter Inhalte verwandt werden. Insbesondere können <CMP>-Elemente auch <formula>-Elemente enthalten.

Prämissen In OMDoc können Prämissen grundsätzlich nur über einen Verweis mittels dem `xref`-Attribut eines <premise>-Elements angegeben werden. Dies ist für natürlichsprachliche Texte nicht ausreichend, da in ihnen Prämissen oft noch einmal textlich aufgegriffen werden, auch wenn sie bereits einmal vorgekommen sind. Daher führen wir zusätzlich zum OMDoc-Element <premise> ein Element <nl-premise> ein, das natürlichsprachlichen Text und einen Verweis (wie <premise> mit `xref`) auf eine andere Proposition oder einen Beweisschritt enthalten kann, deren Inhalt sie wiederholt und sonst dieselbe Syntax enthält wie das OMDoc-Element.

Eine Schwierigkeit könnte sich hier ergeben: Ist die Prämisse ein Satz (<expression type="proposition">) oder ein Beweisschritt, eine Hypothese oder eine Behauptung (<derive>, <hypothesis>, <assertion>)? Da es angebracht scheint, die Ebenen nicht mehr zu vermischen als ohnehin notwendig, verwenden wir, sooft möglich, die Elemente der logischen Ebene zur Bezeichnung der Prämisse.

„Meta-Kommentare“ In OMDoc gibt es eine Möglichkeit, *'meta commentaries'* zu annotieren. Diese „Meta-Kommentare“ sind solche, die nicht für den Verlauf des Beweises nötig sind, aber einem menschlichen Leser sein Verständnis erleichtern. Ein Beispiel wäre:

(55) Wir bedienen uns dabei einer ähnlichen Notierung wir bei den Ableitungen in früheren Kalkülen (vgl II.3). (Ebbinghaus et al. 1996 : 68)

Dieser Satz fügt an sich zu den Fakten des Gesamt-Beweises nichts hinzu — aber er kann das Verständnis eines Lesers durchaus fördern.

Aussagen wie (55) gehen über das hinaus, was für die logische Struktur eines mathematischen Beweises relevant ist. Wir gehen daher auf solche Bemerkungen nicht ein, sondern verwenden für sie das <metacomment>-Element von OMDoc.

Parametrisierte Regeln Im Beweis durch Widerspruch wird um p zu beweisen zunächst $\neg p$ angenommen, und es wird gezeigt, daß — wenn alle Aussagen gelten, die man als Prämissen voraussetzt — ein Widerspruch entsteht; daraus schließt man $\neg\neg p$, also p.[5] Selbst wenn man angegeben hat, daß der Beweis nach der Regel „Beweis durch

[5] Der Beweis durch Widerspruch erfreut sich nur zweifelhafter Beliebtheit; Euklid etwa verwendet zum Beweis der Existenz eines geometrischen Beweises ausschließlich konstruktive Beweise (vgl. Kommentar zu *I.6* von D. C. Joyce.).

Widerspruch“ geführt wird, macht es Sinn, anzeigen zu können, wo der Widerspruch besteht.

In OMDoc ist es möglich, Regeln für Schlüsse zu parametrisieren und Verweise auf Aussagen als Parameter an Regeln zu übergeben. Dies ermöglicht es im Falle eines Beweises durch Widerspruch (z.B. *Elemente*, I,4), die Aussagen, die p und $\neg p$ darstellen, als Parameter an die Regel zu übergeben.

Andererseits kann man auch aus der Anwendung einer Definition schließen. Es kann sinnvoll sein, hierfür eine allgemeine Regel zu definieren, die als Parameter nur das Definierte nimmt und als Prämissen die entsprechenden Voraussetzungen. Ein Beispiel dafür gibt Kohlhase (to appear : Kap. 19.2).

9.3. Lücken

Wie bereits erwähnt, treten in mathematischen Beweisen gelegentlich Lücken auf.

Diese Lücken kann man in zwei Kategorien einteilen: Die Kategorie der Analogien, die wir in Kap. 9.4.4 auf Seite 98 behandeln und die Kategorie „echte Lücken“ / „wie man leicht sieht“, zu der Eisenreich vermerkt:

> Berüchtigt ist dagegen die Bemerkung *wie man leicht sieht*, die gern dann gemacht wird, wenn die explizite Ausführung (um die sich der Autor möglicherweise gedrückt hat) technisch zu aufwendig wird.
> (Eisenreich 1998 : 1223)

Manche solcher Lücken sind allerdings auch tatsächlich für den eingeweihten Leser leicht zu sehen (in diesem Falle sind sie auch häufig nicht explizit als Lücken markiert). Einen solchen Fall haben wir in unseren Beispielen.

(56) (a) ergibt sich unmittelbar aus (b) Ebbinghaus et al. (1996 : 77).

Dieser unmittelbare Beweis ist allerdings, wenn man ihn voll formalisiert, durchaus einige Schritte lang. Es ist möglich, diesen Satz als Verweis auf eine logische Ableitung zu annotieren, welche dann die Lücke schließt. Es muß allerdings erkennbar sein, daß diese Ableitung im Text selbst nicht vorhanden ist.

Annotationsregel 26 (Lücken) *Echte Lücken können geschlossen werden, indem man eine*

Ableitung aus mehreren <derive type="lacuna">*-Schritten einführt, die* <FMP>*-Elemente enthalten.*

In den <FMP>-Elementen kann die formale Semantik der Beweisschritte ausgedrückt werden; das leere <CMP>-Element zeigt an, daß der semantischen Repräsentation keine textliche entspricht. Notwendig ist das <CMP>-Element, falls Stylesheets für mathematische Funktionen (Kap. 8.6.1 auf Seite 79) verwandt werden, die es ermöglichen, daß ein Ableitungsschritt, der in einem Text nur in Formelsprache vorliegt, auch nur als <FMP> dargestellt wird und durch die Stylesheets in eine angemessene Präsentationsform überführt werden kann.

9.4. Pragmatisch relevante, logisch irrelevante Relationen

9.4.1. Motivation

Zwischen Sätzen können Beziehungen bestehen, die in der streng formal-logischen Struktur keine Bedeutung haben, aber in der Diskurs-Struktur eine bestimmte Rolle erfüllen. Man kann hier von rhetorischen Relationen sprechen, allerdings nicht unbedingt im Sinne der RST, sondern im Sinne einer Relation, die auf der rhetorischen oder poetisch-ästhetischen Ebene besteht.

Es kann sein, daß unser Korpus zu klein war, um alle solche Relationen zu enthalten. Sie sprengen in gewissem Maße auch den Rahmen der vorliegenden Arbeit, da sie nicht unbedingt eine logische Entsprechung in der Struktur des Beweises haben. In der weiteren Untersuchung natürlichsprachlicher Beweise wird zu untersuchen sein, welche Rolle solche Beziehungen zwischen Sätzen für die Struktur von Beweisen spielen.

9.4.2. Zitat

Eine solche Relation zwischen Propositionen ist das *Zitat*. Bei Euklid findet es sich in zwei Verwendungsweisen: einerseits zur Kennzeichnung der Verwendung eines Postulats oder eines Axioms, andererseits am Ende eines Beweises zur Kennzeichnung dessen, daß (genau) der anfangs genannte Satz bewiesen ist. Zumindest in der letzten Funktion ist die Wiederholung auch heute noch gebräuchlich. Wir haben in diesen Fällen im Beispielkorpus darauf verzichtet, den Satz noch einmal zu annotieren, sondern angegeben, daß es sich um eine wörtliche Wiederholung handelt.

Eine Verbindung zu den Postulaten oder Axiomen der Theorie herzustellen, ist prinzipiell auf dieselbe Weise möglich.

Da wir das Attribut `refid` grundsätzlich dazu verwenden, um eine anaphorische Beziehung zu bezeichnen, und da diese Verwendung von Zitaten eher ästhetische als logische Gründe zu haben scheint, haben wir für ihre Annotation das Element <relation> mit dem Attribut `type` eingeführt, das die erwartete Bedeutung hat. Für eine wörtliche Wiederholung erhält `type` den Wert `"lit-restatement"`. Der Inhalt eines <relation>-Elements besteht aus <arg>-Elementen, welche auf die Argumente der Relation verweisen. Es ist offensichtlich, daß hier eine Parallele zu den RST-Relationen in OMDoc 1.0 gibt (Kohlhase 2000 : 10,14,41), jedoch ist es in der ProofML-Syntax leicht möglich, solche Relationen zwischen mehr als zwei Propositionen bestehen zu lassen.

9.4.3. Klarstellung

Eine weitere Relation, die im Korpus (einmal) vorkam, war die Relation der Klarstellung. Hier haben wir als `type="clarification"` gesetzt und die beiden Propositionen als Argumente angegeben. Es wäre prinzipiell möglich, noch weiter zu bestimmen, was klargestellt wurde, dies schien aber für das Grundkonzept von ProofML nicht notwendig.

9.4.4. Analogien

Eine wichtige Technik in der Mathematik, um Beweise abzukürzen, ist die Analogie: In einer Fallunterscheidung zeigt man einen Fall und gibt dann (gelegentlich) an, daß in den anderen Fällen „analog“ zu verfahren ist.

Für die Annotation ist es hier natürlich wünschenswert, solche Lücken zu schließen. Üblicherweise besteht die Analogie darin, daß bestimmte Diskursreferenten oder Prädikate ausgetauscht werden. ProofML stellt dafür die Relation <sem-relation type="symmetry"> zur Verfügung, innerhalb deren der Austausch von Diskursreferenten durch die Angabe von Verweisen (Werten des Attributs `refid`) annotiert werden kann.

In Abb. 9.2 auf der nächsten Seite findet sich ein Beispiel, das illustriert, wie eine Lücke im Beweis geschlossen wird; der erste Fall ist gezeigt, der zweite ist symmetrisch zum ersten und daher so klar, daß er nicht eigens erklärt werden muß. Dennoch wäre für einen formalisierten Beweis gerade diese Symmetrie wichtig. Also geben wir an, daß

```
<sem-relation type="symmetry">
  <arg refid="AB !&gt; AC"/>
  <arg refid="AC !&gt; AB"/>
  <exchange>
    <arg refid="AB"/>
    <arg refid="AC"/>
  </exchange>
  <exchange>
    <arg refid="BD"/>
    <arg refid="DC"/>
  </exchange>
</sem-relation>
```

Abb. 9.2: **Analogieschluß**

im Inhalt des Beweisschritts, dessen `id="AB !> AC"` ist, AB gegen AC ausgetauscht werden muß. In unserem Beispiel-Korpus besteht diese Relation, wie gesagt, zwischen zwei Beweisschritten. Natürlich muß die entsprechende Ersetzung auch in allen Beweisschritten vorgenommen werden, von denen der angegebene Beweisschritt abhängt, also in seinen Prämissen.

Es ergibt sich die Schwierigkeit, daß auch alle von AB oder AC abhängigen Entitäten entsprechend umbenannt werden müssen, sofern sie im Text auftreten. Wir sehen vor, daß dies in ProofML explizit geschieht, da uns dies vor allem im Hinblick auf eine maschinelle Verarbeitung wesentlich einfacher zu sein scheint als eine implizite Auflösung. Es mag aber sein, daß sich bei einer breiter angelegten Untersuchung herausstellt, daß eine implizite Auflösung mit vertretbarem Aufwand möglich ist.

Partikularisierungen und Generalisierungen Oft kommt es in mathematischen Texten vor, daß zunächst ein Satz generell als Demonstrandum eingeführt wird und dann aus seinen generell genannten Voraussetzungen Prämissen gemacht werden, in denen die Objekte meist Namen bekommen. Wenn es gewünscht ist, solche Verhältnisse zu annotieren (auch hier besteht keine Notwendigkeit, aber es könnte hilfreich sein, um die Struktur des Satzes zu verstehen), so kann man zwischen den Objekten Relationen der Art `type="particularisation"` o.ä. definieren (vgl. für ein Beispiel Abb. 8.5 auf Seite 66).

9.5. Eine Grenze von ProofML: Hypothesen und unmögliche Objekte

Das Modell für Beweise, das OMDoc verwendet und dem wir uns anschließen, folgt u.a. dem Natürlichen Schließen nach Gentzen (1934/35) (Kohlhase to appear : Kap. 19.2).

Beweise dieser Art lassen sich in einer Baumstruktur (oder, wenn man mehrfach verwandte Prämissen immer durch Verweise anzeigt, durch einen gerichteten Graphen) darstellen. In ProofML wird diese Struktur durch die `xref`-Attribute der `<premise>`-Elemente dargestellt.

Ein wesentliches Merkmal dieses Ansatzes ist, daß gelegentlich zu Beginn eines Beweiszweiges eine Annahme (in ProofML/OMDoc als `<hypothesis>` markiert) gemacht wird, und anschließend Beweise geführt werden, die von diesen Annahmen abhängen. Wenn man etwa beweisen will, daß A B impliziert, nimmt man A an, zeigt, daß B gilt. Schließlich folgert man daraus $A \to B$, also A impliziert B.

Beweist man, daß eine Annahme A in einer Menge von Aussagen, die man als wahr annimmt, zu einem Widerspruch führt, gilt diese Annahme als widerlegt. Insofern müssen alle Schlußfolgerungen, die in dem entsprechenden Beweis-Teilbaum gezogen wurden, der mit dem Widerspruch endet, aus dem Kontext entfernt werden, insbesondere natürlich auch A.[6]

Der Beweis durch Widerspruch kann in dieser Art verwandt werden, um generische Annahmen zu widerlegen, allerdings nicht, um über einzelne unmögliche Objekte zu sprechen, etwa über „den Punkt p auf dem Kreis k, der dem Mittelpunkt am nächsten ist“, sofern p und k als Individuen angenommen werden, oder den „Durchmesser eines Einhorn-Horns“ (sofern man Einhörner nicht für reale Wesen hält, sonden allenfalls für mögliche). Um solche und andere Aussagen über (un)mögliche Objekte und ihre (Un)Möglichkeit machen zu können, bräuchte man eine intensionale Logik, die Veränderungen im Beweis-Modell von OMDoc/ProofML verlangen würde.

Wir haben aber bereits in Kap. 8.5 auf Seite 67 darauf hingewiesen, daß in mathematischen Beweisen häufig benannte Entitäten (auch in definiten Nominalphrasen) eingeführt und so verwandt werden, als seien sie in einer DPL mit einem dynamischen Existenzquantor eingeführt worden, so daß der Beweis letztlich generisch wird; man spricht also letztlich über alle Punkte auf einem Kreis, die dem Mittelpunkt am näch-

[6] Frank Veldman kündigt auf seiner Homepage (`http://turing.wins.uva.nl/~veltman/research.htm`) einen Artikel an, der für diesen Aspekt der Betrachtung von Beweisen relevant sein kann: *'How natural is natural deduction in DPL?'*

sten sind und zeigt, daß es solche Punkte nicht geben kann. Eine solche generische Interpretation der Hypothesen war für alle Beweise möglich, die für die Erstellung von ProofML betrachtet wurden.

Dennoch mag es sein, daß für eine größer angelegte Untersuchung mathematischer Beweise eine intensionale Logik notwendig ist.

Teil IV.

Schluß

In dieser Arbeit haben wir eine Annotationssprache vorgestellt, die es erlaubt, die sprachliche und die logische Struktur mathematischer Beweise so zu annotieren, daß in einem weiteren Schritt eine genauere Untersuchung ihrer Interaktion möglich sein sollte.

Wir sind dazu von Beispielen aus realen mathematischen Texten ausgegangen, deren Annotation sich als Beispiel der Anwendung von ProofML im Anhang findet. Als besonders wichtig haben sich die Einbettung mathematischer Beweise in das mathematische Hintergrundwissen eines bestimmten Bereiches und die Benennung von Objekten erwiesen; auch der Beitrag der Benennung von Objekten zur Semantik eines Textes ist ganz offensichtlich groß.

Es bleibt zu hoffen, daß sich ProofML im weiteren Gebrauch bewährt, auch wenn sicherlich einige Anpassungen und Erweiterungen notwendig sein mögen.

Literaturverzeichnis

Abrahams, Paul W. (1964): Application of LISP to Checking Mathematical Proofs. In Berkeley, Edmund C. und Daniel G. Bobrow (Hg.), *The Programming Language* LISP*: Its Operations and Applications*, S. 137–160. Cambridge (USA), London: The MIT Press.

Asperti, Andrea, Michael Kohlhase und Claudio Sacerdoti Coen (2003a): Prototype n. D2.b Document Type Descriptors: OMDoc Proofs. Mowgli deliverable, The MoWGLI Project.

Asperti, Andrea, Luca Padovani, Claudio Sacerdoti Coen, Ferruccio Guidi und Irene Schena (2003b): Mathematical Knowledge Management in HELM. *Annals of Mathematics and Aritificial Intelligence*, 38(1):27–46.

Barwise, Jon und Robin Cooper (1981): Generalized quantiers and natural language. *Linguistics and Philosophy*, 4:159–219.

Baur, Judith (1999): *Syntax und Semantik mathematischer Texte — ein Prototyp*. Diplomarbeit, Saarland University.

van Benthem, Johan (2003): Mathematical Logic and Natural Language. In Löwe, B., W. Malzkorn und T. Rasch (Hg.), *Foundations of the Formal Sciences II: Applications of Mathematical logic in Philosophy and Linguistics*, S. 25–38. Dordrecht: Kluwer.

Bobrow, Daniel G. (1968): Natural Language Input for a Computer Problem-Solving System. In Minsky, Marvin (Hg.), *Semantic Information Processing*, S. 135–215. Cambridge (USA): The MIT Press.

Bray, Tim, Dave Hollander und Andrew Layman (1999): Namespaces in XML. W3c recommendation, The World Wide Web Consortium.
URL: `http://www.w3.org/TR/REC-xml-names`

Bray, Tim, Jean Paoli und C. M. Sperberg-McQueen (1997): Extensible Markup Language (XML). W3C Recommendation TR-XML, The World Wide Web Consortium.
URL: `http://www.w3.org/TR/PR-xml.html`

Brinker, Klaus (2001): *Linguistische Textanalyse: eine Einführung in Grundbegriffe und Methoden*. Nr. 29 in Grundlagen der Germanistik, 5. Aufl. Berlin: Erich Schmidt Verlag.

Caprotti, Olga, David P. Carlisle und Arjeh M. Cohen (2000): The Open Math Standard, Version 1.0. Techn. Ber., The Open Math Society.
URL: `http://monet.nag.co.uk/cocoon/openmath/standard/omstd.pdf`

Carlisle, David, Patrick Ion, Robert Miner und Nico Poppelier (2003): Mathematical Markup Language (MathML) Version 2.0 (second edition). W3c last call, World Wide Web Consortium. Available at http://www.w3.org/TR/MathML2.

Clark, James und Steve DeRose (1999): XML Path Language (XPath) Version 1.0. W3c recommendation, The World Wide Web Consortium.
URL: `http://www.w3.org/TR/xpath`

DeRose, Steve, Eve Maler, David Orchard und Ben Trafford (2001): XML Linking Language (XLink). W3c recommendation, The World Wide Web Consortium.
URL: `http://www.w3.org/TR/xlink`

Ebbinghaus, Heinz-Diether, Jörg Flum und Wolfgang Thomas (1996): *Einführung in die mathematische Logik*. 4. Aufl., korrigierter Nachdruck 1998. Heidelberg: Spektrum, Akademischer Verlag.

van Eijck, Jan und Hans Kamp (1997): Representing Discourse in Context. In *Handbook of Logic and Linguistics*, Kap. 3, S. 179–237. Elsevier.
URL: `ftp://ftp.cwi.nl/pub/jve/reports/rdcrep.ps.Z`

Eisenberg, Peter (1999): *Grundriß der deutschen Grammatik. Band 2: Der Satz*. Stuttgart, Weimar: J.B. Metzler.

Eisenreich, Günther (1998): Die neuere Fachsprache der Mathematik seit Carl Friedrich Gauß. In Hoffmann, Lother, Hartwig Kalverkämper, Herbert Ernst Wiegand, Christian Galinski und Werner Hüllen (Hg.), *Fachsprachen / Languages for Special Purposes: ein internationales Handbuch zur Fachsprachenforschung und Terminologiewissenschaft / An International Handbook of Special-Language and Terminology Research, 1. Halbband*, Nr. 14 in Handbücher zur Sprach- und Kommunikationswissenschaft, Kap. 136, S. 1222–1230. Berlin, New York: de Gruyter.

Essler, Wilhelm K. (1969): *Einführung in die Logik*. zweite, erweiterte Auflage. Stuttgart: Alfred Körner.

Evans, Roger und Gerald Gazdar (1996): DATR: A language for lexical knowledge representation. *Computational Linguistics*, 22.2:167–216.

Frege, Gottlob (1998): Begriffsschrift, eine der arithmetischen nachgebildete Formelsprache des reinen Denkens. In Angelli, Ignacio (Hg.), *Begriffsschrift und andere Aufsätze*, Zweite Auflage. Hildesheim, Zürich, New York: Georg Olms.

Gentzen, Gerhard (1934/35): Über das mathematische Schließen. *Mathematische Zeitschrift*, 39:176–210; 405–431.

Grice, Herbert Paul (1968): Logic and Conversation. wieder abgedruckt in. In Davis, Steven (Hg.), *Pragmatics. A Reader*, S. 305–315. Oxford: Oxford University Press, 1991.

Groenendijk, Jeroen und Martin Stokhof (1991): Dynamic Predicate Logic. *Linguistics and Philosophy*, 14(1):39–100.
URL: `http://citeseer.nj.nec.com/article/groenendijk90dynamic.html`

Harrison, John (1996): A Mizar Mode for HOL. In von Wright, Joakim, Jim Grundy und John Harrison (Hg.), *Theorem Proving in Higher Order Logics: 9th International Conference, TPHOLs'96*, Bd. 1125 von *Lecture Notes in Computer Science*, S. 203–220. Turku, Finland: Springer-Verlag.

Heath, Sir Thomas (1920): *Euclid in Greek. Book I. With introduction and notes.* Cambridge: Cambridge University Press.

Jaśkowski, Stanisław (1934): On the Rules of Suppositions in Formal Logic. *Studia Logica*, 1. Wiederabgedruckt in S. McCall (Hrsg.) (1967): *Polish Logic 1920–39, Oxford: Oxford University Press*.

Koepke, Peter und Bernhard Schröder (2003): ProofML – eine Annotationssprache für natürliche Beweise. In Seewald-Heeg, Uta (Hg.), *Sprachtechnologie für die multilinguale Kommunikation. Textproduktion, Recherche, Übersetzung Lokalisierung. Beiträge der* gldv-*Frühjahrstagung 2003*, Nr. 5 in Sprachwissenschaft, Computerlinguistik, Neue Medien, hrsg. von Nico Weber, S. 428–441. St. Augustin: Gardez!

Kohlhase, Michael (2000): OMDoc: An Open Markup Format for Mathematical Documents. Seki Report SR-00-02, Fachbereich Informatik, Universität des Saarlandes.
URL: `http://www.mathweb.org/omdoc`

— (to appear): OMDoc An open markup format for mathematical documents (Version 1.2). Techn. Ber., Computer Science, Carnegie Mellon University.

ANMERKUNG: Für diese Arbeit wurde die Dokumentation nach dem Stand vom 27. Juli 2003 verwandt; die Zählung der Kapitel bezieht sich auf eine mit LaTeX erstellte Version der Dokumentation, die alle Kapitel einbindet.

KUSCHERT, Susanna (1996): *Dynamic Meaning and Accomodation*. Dissertation, Universität des Saarlandes, Saarbrücken.
URL: `http://www.coli.uni-sb.de/~kuschert/{vorspann,thesis-te%xt}.ps.gz`

LAMPORT, Leslie (1993): How to write a proof. SRC Research Report 94, DEC Systems Research Center (SRC), Palo Alto.
URL: `http://gatekeeper.research.compaq.com/pub/DEC/SRC/research-reports/abstracts/src-rr-094.html`

MANN, Bill (1999): Introduction to RST.
URL: `http://www.sil.org/~mannb/rst/rintro99.htm`

MARSH, Jonathan und David ORCHARD (2002): XML Inclusions (XInclude) Version 1.0. W3c recommendation, W3C.
URL: `http://www.w3.org/TR/xinclude`

Mizar Projekt (2003): *Mizar Homepage*.
URL: `http://www.mizar.org`

MOORTGAT, Michael (1996): Categorial Type Logics. In VAN BENTHEM, Johan und Alice TER MEULEN (Hg.), *Handbook of Logic and Language*, S. 93–177. Amsterdam: Elsevier.
URL: `citeseer.nj.nec.com/moortgat97categorial.html`

MORRIL, Glyn V. (1994): *Type Logical Categorial Logical Grammar. Categorial Grammar of signs*. Dordrecht: Kluver Academic Publishers.

NEDERPELT, Rob (2002): Weak Type Theory: A formal language for mathematics. Computing Science Report 02-05, Techische Universiteit Eindhoven, Faculteit Wiskunde en Informatica.
URL: `http://www.win.tue.nl/~wsinrpn/mv-report.ps`

PARTEE, Barbara H. (1987): Noun Phrase Interpretation and Type-Shifting Principles. In GROENENDIJK, Jeroen und Martin STOKHOF (Hg.), *Studies in Discourse Representation Theory and the Theory of Generalized Quantifiers*, S. 115–144. Dordrecht: Foris.

PELLETIER, Francis Jeffry (2000): A History of Natural Deduction and Elementary Logic Textbooks. In WOODS, J. und B. BROWN (Hg.), *Logical Consequence: Rival Approaches and New Studies*, Bd. 1, S. 105–138. Oxford: Hermes Science.

Prince, Ellen F. (1981): Toward a Taxonomy of Given–New Information. In Cole, Peter (Hg.), *Radical Pragmatics*, S. 223–255. New York: Oxford University Press.

Ranta, Aarne (1994): Type theory and the informal language of mathematics. In *Types for Proofs and Programs*, Nr. 806 in Lecture Notes in Computer Science. Heidelberg: Springer-Verlag.
URL: `http://www.math.chalmers.se/~aarne/articles/nijmegen.ps.gz`

— (1995): Syntactic categories in the language of mathematics. In *Types for Proofs and Programs*, Nr. 996 in Lecture Notes in Computer Science, S. 162–182. Heidelberg: Springer-Verlag.
URL: `http://www.math.chalmers.se/~aarne/articles/baastad.ps.gz`

— (1996): Context-relative syntactic categories and the formalization of mathematical text. In *Types for Proofs and Programs*, Nr. 1158 in Lecture Notes in Computer Science, S. pp. 231–248. Heidelberg: Springer-Verlag.
URL: `http://www.math.chalmers.se/~aarne/articles/torino.ps.gz`

— (1999): Structures grammaticales dans le français mathématique. *Mathématiques, informatique et Sciences Humaines*, S. 138:5–56; 139:5–36.

— (2002): Grammatical Framework: A Type-Theoretical Grammar Formalism (Manuscript). *Journal of Functional Programming*. To appear.
URL: `http://www.math.chalmers.se/~aarne/articles/gf-jfp.ps.gz`

Rudnicki, Piotr (1992): An Overview of the MIZAR Project. In *Proceedings of the 1992 Workshop on Types and Proofs as Programs*, S. 311–332.

Rudnicki, Piotr und Andrzej Trybulec (1996): A Note on "How to Write a Proof". Techn. Ber. 1996-08, University of Alberta.
URL: `ftp://ftp.cs.ualberta.ca/pub/TechReports/1996/TR96-08/`

Sacerdoti Coen, Claudio (1999/2000): *Progettazione e realizzazione con tecnologia di XML di basi distribuite di conoscenza matematica formalizzata*. Diplomarbeit, Università degli Studi di Bologna.
URL: `http://www.cs.unibo.it/~sacerdot/tesi/tesi.ps.gz`

Siekmann, Jörg, Michael Kohlhase und Erica Melis (1998): Omega: Ein mathematisches Assistenzsystem. *Kognitionswissenschaft*.

Tesnière, Lucien (1965): *Éléments de syntaxe structurale*. 2, überarbeitete und verbesserte Aufl. Paris: Librairie C. Klincksieck.

The Logical Project (2003): The Coq proof assistant. Institut national de recherche en informatique et en automatique (INRIA).
URL: `http://coq.inria.fr`

Unicode Inc. (Hg.) (2003): *The Unicode Standard, Version 4.0*. Addison-Wesley.

Zinn, Claus (1999): Understanding Mathematical Discourse. In *Proceedings Amstelogue '99, Workshop on the Semantics and Pragmatics of Dialogue, Amsterdam University, 7-9 May 1999. 15*.
URL: `http://citeseer.nj.nec.com/zinn99understanding.html`

— (2000): Computing Presuppositions and Implicatures in Mathematical Discourse. In Bos, J. und M. Kohlhaase (Hg.), *Workshop 'Inference in Computational Semantics' (ICoS-2)*. Schloss Dagstuhl.

— (2003): A Computational Framework for Understanding Mathematical Discourse. *Research on Language and Computation*. Revised version accepted for publication. To appear.
URL: `http://www.cogsci.ed.ac.uk/~zinn/private/zinn-icos2si.ps`

— (to appear): *Understanding Mathematical Discourse (vorläufige Version)*. Dissertation.

Anmerkung: Die Zitate beziehen sich auf die Fassung vom 12. Juni 2003

URL: `http://www.cogsci.ed.ac.uk/~zinn/private/`

Verwandte Euklid-Ausgaben

Euclidis Elementa edidit et Latine interpretatus est I. L. Heiberg, Uol. 1, Libros I–IV continens. Leipzig: Teubner, 1883.

Euclidis Elementa post I. L. Heiberg edidit E. S. Stammatis, Uol. 1, Libri I–IV cum appendicibus. Leipzig: Teubner, 1969.

Euclid. Aristotelis opera. J. L. Heiberg. Berlin. Reimer. 1831. `http://perseus.mpiwg-berlin.mpg.de/cgi-bin/ptext?lookup=Euc.+1`[7]

Euclid in Greek. Book I. With introduction and notes by Sir Thomas L. Heath. Cambridge: University Press, 1920.

The thirteen books of Euclid's Elements translated from the text of Heiberg with introduction and commentary. Three volumes. University Press, Cambridge, 1908. Second edition. Cambridge: University Press, 1925.

`http://perseus.mpiwg-berlin.mpg.de/cgi-bin/ptext?lookup=Euc.+1`

Euclid's Elements. D.E. Joyce:`http://aleph0.clarku.edu/~djoyce/java/elements/elements.html`. 1996–1998. [8]

Euclides: *Die Elemente*: Bücher I–XIII / von Euklid. (Oswalds Klassiker der exakten Wissenschaften 235). Aus dem Griechischen übersetzt und herausgegeben von Clemens Thaer mit einem Vorwort von W. Trageser. Reprint, 3. Aufl. Thun, Frankfurt/M.: Deutsch, 1997.

Der griechische Text ist nach der Ausgabe von Stammatis zitiert, sofern diese von den anderen abweicht.

---

[7] Diese bibliographische Angabe zu Heiberg bei Perseus ist zweifelhaft. 1831 erschien eine Ausgabe *Aristotelis opera*, edidit Academia regia borussica, ed. Immanuel Bekker. 5 Bd. Berlin, G. Reimer, 1831–1870. Der spätere Euklid-Editor J. L. Heiberg (1854–1928) war damals noch nicht geboren, und der Dichter J. L. Heiberg (1791–1860) scheint keine solche Ausgabe erstellt zu haben. Es wäre ohnehin fraglich, was eine Euklid-Ausgabe in einer Ausgabe der Werke von Aristoteles zu suchen hätte. Der Text scheint allerdings im wesentlichen mit der „klassischen" Druck-Ausgabe von 1883–88 übereinzustimmen. Eine Rückfrage an das Perseus-Team wurde eingereicht.

[8] Der Verfasser vermerkt auf der Internet-Seite, `http://aleph0.clarku.edu/~djoyce/java/elements/aboutText.html`

> This text of this version of Euclid's Elements is similar to Heath's edition which he translated from Heiberg's definitive edition in Greek, but it is slightly less literal to make it more readable.

Anhang

A. Die ProofML-DTD

Obwohl ProofML prinzipiell als eine Erweiterung von OMDoc konzipiert ist, erwies sich die Erweiterung der OMDoc-DTD als untauglich für die Präsentation in dieser Arbeit. Dies liegt daran, daß die OMDoc-DTD modular aufgebaut ist und auf raffinierte Art und Weise Parameter-Entitäten verwendet; dadurch wird sie leider schwer lesbar.

Daher ist in diesem Abschnitt eine „traditionelle“ XML-DTD für ProofML gegeben, die mit Kommentaren versehen ist und alle Erscheinungen abdeckt, die in den Beispielen auftreten und diese noch weiter generalisiert, wo nötig. Die Elemente, die die Präsentation von Beweisen betreffen (`class`, `style`) haben wir beiseite gelassen, ebenso das Element <`proofobject`>, für das wir in unserem Modell keine Verwendung haben.

Für den realen Einsatz würde sich die Modifikation der OMDoc-DTD empfehlen, da damit auch alle anderen Module von OMDoc zur Verfügung stehen.

B. Beweise

B.1. Euklid: *Elemente*, Satz 6

Es handelt sich um den sechsten Beweis aus dem ersten Buch der *Elemente*. Des Leseflusses wegen haben wir die Anpassung der Heath-Übersetzung durch D.C. Joyce gewählt. In eckigen Klammern geben wir diejenigen Prämissen einer Konstruktion an, die Joyce an der Seite vermerkt; wir fügen als Fußnote den Text der Postulate, Definitionen und Demonstranda der Sätze an, auf die der Verweis sich bezieht.

Die Skizze findet sich so oder ähnlich in allen Druck-Ausgaben und in Joyce' Ausgabe (dort allerdings interaktiv).

B.1.1. Der Text

If in a triangle two angles equal one another, then the sides opposite the equal angles also equal one another.

Let ABC be a triangle having the angle ABC equal to the angle ACB.

I say that the side AB also equals the side AC.

If AB does not equal AC, then one of them is greater. [C.N.][1]

Let AB be greater. Cut off DB from AB the greater equal to AC the less [I.3][2], and join DC [Post. 1][3].

Since DB equals AC, and BC is common, therefore the two sides DB and BC equal the two sides AC and CB respectively, and the angle DBC equals the angle ACB. Therefore the base DC equals the base AB, and the triangle DBC equals the triangle ACB [I.4],[4] the

[1] Joyce weist in seinem Kommentar zu diesem Beweis auf *'the law of trichotomy'* hin, welches besagt, daß es bei der Gleichheit drei Fälle gibt: $AB < AC$, $AB = AC$ und $AB > AC$. Dies ist eine der Selbstverständlichkeiten, die in den Elementen selbst nicht definiert sind.

[2] To cut off from the greater of two given unequal straight lines a straight line equal to the less.

[3] To draw a straight line from any point to any point.

[4] If two triangles have two sides equal to two sides respectively, and have the angles contained by the equal straight lines equal, then they also have the base equal to the base, the triangle equals the triangle,

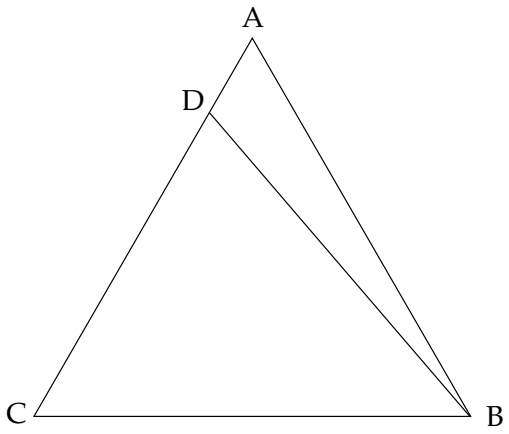


Abb. B.1: **Skizze zu *Elemente* I.6**

less equals the greater, which is absurd [CN.5][5]. Therefore AB is not unequal to AC, it therefore equals it.

Therefore *if in a triangle two angles equal one another, then the sides opposite the equal angles also equal one another.*

B.1.2. Hintergrundwissen

In diesem Abschnitt wollen wir kurz skizzieren, welches Hintergrundwissen für das Verständnis dieses Beweises vorhanden sein muß.

Ganz offensichtlich müssen wir zum Verständnis dieses Satzes nicht über das gesamte Wissen verfügen, das Euklid bis hierher definiert hat. Denn von nicht-geraden Linien oder vierseitigen „geradlinigen Figuren“ ist keine Rede.

Der Leser muß wissen, was ein Dreieck ist,[6] was Punkte und gerade Linien (Strecken) sind. Wir müssen wissen, was Gleichheit bedeutet. Euklid versteht Strecken primär als Länge (Def. I.2); zwei Strecken sind als gleich, wenn sie gleich lang sind. Dreiecke

and the remaining angles equal the remaining angles respectively, namely those opposite the equal sides.

[5] The whole is greater than the part.

[6] Auf die Schwierigkeit, daß Euklid nicht erwähnt, daß Dreiecke drei Ecken haben, haben wir bereits hingewiesen. Allerdings kann ein menschlicher Leser dies durchaus daraus erschließen, daß ein Dreieck durch drei gerade Linien begrenzt ist (Def. I.19).

sind offensichtlich gleich, wenn sie gleiche Seiten(längen) haben; denn darauf, daß sie gleiche Winkel haben, weist Euklid explizit hin.[7]

Man muß wissen, unter welchen Umständen welche Sätze, Definitionen, Axiome und Postulate anwendbar sind, oder zumindest, was es bedeutet, wenn sie angewendet werden.

Wir benötigen allerdings auch Wissen, daß über dasjenige hinausgeht, was Euklid definiert. Ein Beispiel sind grundlegende Schlußprinzipien wie der *modus ponens*, der ja im Prinzip jeder Anwendung eines vorher geführten Beweises oder eines Postulates, Axioms oder einer Definition zugrundeliegt, und den Beweis durch Fallunterscheidung (obwohl Euklid nur einen Fall durchspielt, s.u.). Diese „Lücken" in Euklids Beweisen und Definitionen sind so selbstverständlich, daß sie uns bei der Betrachtung des Beweises kaum auffallen.

Ähnliches gilt für die Namensgebung, auf die wir im Hauptteil der Arbeit ausführlich eingegangen sind (Kap. 8.5 auf Seite 67): Man muß die Namen der Entitäten deuten können, um die Bezeichnungen der Seiten zu verstehen und zu wissen, daß es sich nicht einfach um Seiten handelt, die zum genannten Dreieck *ABC* gehören. Auch die Einführung des Punktes *D* wird nicht explizit erwähnt, sondern ist nur aus der Benennung der Strecke *DB* zu erschließen, die auf *AB* abgetragen wird, ebenso die Tatsache, daß *B* der andere Endpunkt der Strecke ist.

Darüberhinaus muß der Leser verstehen, daß wenn für eine Seite bewiesen ist, daß sie nicht größer als die andere ist, ein analoger Beweis für den umgekehrten Fall geführt werden kann. Denn daraus, daß *AB* nicht größer als *AC* ist, folgt nicht direkt, daß *AB* nicht ungleich *AC* ist — es könnte ja noch kleiner sein.

Euklid setzt auch nicht genau auseinander, weshalb das Dreieck *ABC* denn kleiner sein müßte als *DCB*. '[...]the less equals the greater.' — daß dies aus dem vorangehenden folgt, versteht man daher nur, wenn man — mit „gesundem Menschenverstand", daher der Name der Schlußregel in der Annotation — annimmt, daß ein Dreieck, daß zum ursprünglichen zwei gleiche und mindestens eine kürzere Seite hat, kleiner sein müßte als dieses (vgl. Heaths *Euclid in Greek* (1920:171), Anm. 15). An dieser Stelle sieht man, weshalb grundsätzlich Skizzen geometrischen Beweisen beigegeben werden.

[7] Vgl. auch Joyce' Anmerkung zu I.4 bezüglich der Kongruenz von Dreiecken.

B.2. Euklid: *Elemente*, Satz I.4

Es gilt entsprechend das zum vorherigen Beweis Gesagte.

B.2.1. Der Text

If two triangles have two sides equal to two sides respectively, and have the angles contained by the equal straight lines equal, then they also have the base equal to the base, the triangle equals the triangle, and the remaining angles equal the remaining angles respectively, namely those opposite the equal sides.

Let *ABC* and *DEF* be two triangles having the two sides *AB* and *AC* equal to the two sides *DE* and *DF* respectively, namely *AB* equal to *DE* and *AC* equal to *DF*, and the angle *BAC* equal to the angle *EDF*.

I say that the base *BC* also equals the base *EF*, the triangle *ABC* equals the triangle *DEF*, and the remaining angles equal the remaining angles respectively, namely those opposite the equal sides, that is, the angle *ABC* equals the angle *DEF*, and the angle *ACB* equals the angle *DFE*.

If the triangle *ABC* is superposed on the triangle *DEF*, and if the point *A* is placed on the point *D* and the straight line *AB* on *DE*, then the point *B* also coincides with *E*, because *AB* equals *DE*.

Again, *AB* coinciding with *DE*, the straight line *AC* also coincides with *DF*, because the angle *BAC* equals the angle *EDF*. Hence the point *C* also coincides with the point *F*, because *AC* again equals *DF*.

But *B* also coincides with *E*, hence the base *BC* coincides with the base *EF* and equals it. [C.N.4][8]

Thus the whole triangle *ABC* coincides with the whole triangle *DEF* and equals it. [C.N.4]

And the remaining angles also coincide with the remaining angles and equal them, the angle *ABC* equals the angle *DEF*, and the angle *ACB* equals the angle *DFE*.

Therefore *if two triangles have two sides equal to two sides respectively, and have the angles contained by the equal straight lines equal, then they also have the base equal to the base, the*

[8] Things which coincide with one another equal one another.

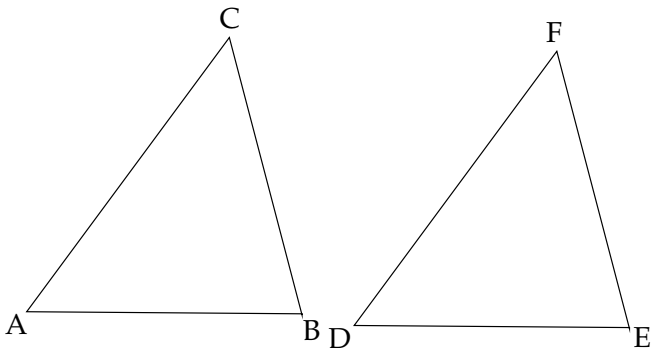


Abb. B.2: **Skizze zu *Elemente* I.4**

triangle equals the triangle, and the remaining angles equal the remaining angles respectively, namely those opposite the equal sides.

Q.E.D.

B.3. Der Gödelsche Vollständigkeitssatz

B.3.1. Der Text

Wie bereits in der Einleitung zu diesem Kapitel ausgeführt, gewinnen wir aus 2.4 (für höchstens abzählbares S) bzw. 3.3 (für beliebiges S) die Vollständigkeit des Sequenzenkalküls:

4.1 Der Vollständigkeitssatz. *Für $\Phi \subset L^S$ und $\varphi \in L^S$ gilt:*

$$\textit{Wenn } \Phi \models \varphi, \textit{ so } \Phi \vdash_S \varphi.$$

4.1 ergibt zusammen mit dem Korrektheitssatz:

$$\text{Für } \Phi \subset L^S \text{ und } \varphi \in L^S\text{: } \Phi \models \varphi \text{ gdw } \Phi \vdash_S \varphi,$$

und nach 3.3 und IV.7.5 erhalten wir:

$$\text{Für } \Phi \subset L^S\text{: Erf } \Phi \text{ gdw } \mathrm{Wf}_S\, \Phi.$$

(Ebbinghaus et al. 1996 : 92)

B.3.2. Kommentar

Zunächst fällt auf, daß es sich hierbei nicht im herkömmlichen Sinne um einen Beweis handelt — jedenfalls nicht im üblichen Sinne. Obwohl der Satz einer der zentralen Sätze der mathematischen Logik ist, wird er durch einen Rückgriff auf zwei bereits bewiesene Sätze erschlossen, in einer Art Fallunterscheidung, die in einem gewissen Maße keine ist, da der zweite Fall den ersten miteinbegreift (endliche Mengen sind ein spezieller Fall beliebig großer Mengen).

Prinzipiell handelt es sich also um einen „didaktischen Satz“: er ist in der Literatur so bekannt, daß man ihn als eigenen Satz nennen sollte, aber sein Beweis erfolgt auf verschiedene Art und Weise für verschieden komplexe Fälle, so daß die Autoren ihn noch einmal unter seinem eigentlichen Namen einführen möchten.

Es schließt sich durch Anknüpfen an einen weiteren bereits geführten Beweis der „Beweis“ an, daß Folgerungs- ($\models$) in den Logiksprachen erster Stufe und Ableitbarkeitsbeziehung ($\vdash$) im Sequenzenkalkül äquivalent sind, ebenso die Erfüllbarkeit und die Widerspruchsfreiheit einer Aussagenmenge.

B.4. Ein Lemma

Das folgende Lemma stammt von Ebbinghaus et al. (1996 : 77). Wir geben zunächst seinen Text an; anschließend deuten wir das mathematische Hintergrundwissen, daß vorhanden sein muß, um den Satz beweisen zu können.

B.4.1. Der Text

7.2 Lemma. *Für eine Aussagenmenge* Φ *sind äquivalent:*

(a) $Wv\ \Phi$

(b) Für alle φ*:* $\Phi \vdash \varphi$.

Beweis. (a) ergibt sich unmittelbar aus (b). Sei umgekehrt Wv Φ, d.h. $\Phi \vdash \psi$, $\Phi \vdash \neg\psi$ für ein geeignetes ψ. Sei φ beliebig vorgegeben. Wir zeigen, daß $\Phi \vdash \varphi$.
Zunächst existieren Γ_1 und Γ_2 und Ableitungen

$$\begin{array}{ccc} \vdots & & \vdots \\ \Gamma_1\psi & \text{bzw.} & \Gamma_2\neg\psi \end{array}$$

Durch Zusammensetzen und Verlängern gelangen wir zur Ableitung

		$\vdots$		
$m.$		Γ_1	ψ	
		$\vdots$		
$n.$		Γ_2	$\neg\psi$	
$(n+1).$	Γ_1	Γ_2	ψ	(Ant) auf m.
$(n+2).$	Γ_1	Γ_2	$\neg\psi$	(Ant) auf n.
$(n+3).$	Γ_1	Γ_2	φ	(Wid) auf $(n+1).$, $(n+2).$.

Wir sehen, daß $\Phi \vdash \varphi$.

B.4.2. Kommentar

Dieser Text zeigt folgende Auffälligkeiten:

– Bezug auf eine Theorie

  - Sequenzenkalkül: Struktur und Semantik von Sequenzen, anwendbare Regeln
  - Definitionen (Wf, Wv)
  - Variablen-Namen: ϕ, ψ für Aussagen; Φ für Aussagenmengen, Γ für Aussagenmengen; S für Trägermengen; L^S für die Ausdrücke der Sprachen erster Stufe über der Trägermenge S.

– eine nochmalige Erklärung eines gegebenen Begriffs

– natürlichsprachliche Angabe von Schlußfolgerungsregeln: *Durch Zusammensetzen und Verlängern* klingt nach einer Folgerungs-<`method`>; allerdings wurde diese Methode mindestens einem Leser erst nach Betrachtung der Ableitungsformel klar. Daher haben wir die entsprechende Methode *'formula-manipulation'* genannt.

– ein Aufzählungsalphabet (Γ_i), das aber *gerade nicht* für eine regelrechte Aufzählung verwandt wird; es handelt sich eher um Pseudo-Diakritika (s. 76).