Originalveröffentlichung in: Fisseni, Bernhard/Schmitz, Hans-Christian/Schröder, Bernhard/Wagner, Petra (Hrsg.): Sprachtechnologie, mobile Kommunikation und linguistische Ressourcen. Beiträge zur GLDV-Tagung 2005 in Bonn. - Frankfurt am Main [u.a.]: Lang, 2005. S. 604-618. (Sprache, Sprechen und Computer. Computer Studies in Language and Speech 8)

PROOFML – eine Annotationssprache für natürlichsprachliche mathematische Beweise

Bernhard Fisseni

Mathematische Texte sind natürlichsprachlich (ggf. mit formelsprachliche Anteilen), nicht formalsprachlich. PROOFML ist ein Dateiformat, das erlaubt, sie so zu annotieren, daß der natürlichsprachlichen eine logische Struktur zugeordnet wird.

Mathematical texts are usually not fully formalised but are written in natural language (possibly interspersed with formulae). PROOFML is an XML application that permits to annotate mathematical proofs so that a logical structure is assigned to the linguistic structure.

1. Problemstellung

Mathematische Texte sind natürlichsprachlich, nicht formalsprachlich. Dennoch gilt als ein Kriterium für die Annahme ihrer Korrektheit, daß sie in einem (zumindest korrekten) logischen Kalkül komplett formalisierbar sind. Formalisierte Beweise können prinzipiell mit einem Rechner verarbeitet und an allgemein akzeptierten Regeln der Beweisführung verifiziert werden.[207] Derart formalisierte Beweise sind allerdings für den Leser schwer verständlich, weil sie zu detailreich sind. Hingegen ist eine formale Verifikation unvorbereiteter Texte auch wegen der sprachlichen Komplexität nicht einfachhin möglich; es gibt bisher nur experimentelle Systeme mit beschränkter Reichweite (vgl. Bauer 1999, Zinn 2004); deren Repräsentation des Beweises entspricht allerdings eher der statischen Struktur eines formalisierten Beweises und wird nicht mehr an die sprachliche Struktur rückgekoppelt.

PROOFML ist eine Annotationssprache für natürlichsprachliche mathematische Beweise, die der natürlichsprachlichen Struktur durch die Annotation eine formal-logische Interpretation zuordnet und somit (1) den Anspruch der Formalisierbarkeit explizit macht,[208] (2) es aber auch erlaubt, formalisierte und nichtformalisierte Beweise direkt zu vergleichen, da die natürlichsprachliche Struktur des Beweises erhalten bleibt. Diese Aufgaben soll PROOFML im Forschungsprojekt *Beweisformen* an der Universität Bonn erfüllen; dort soll es auch als Datei-

[207] An der für die Beweisführung nötigen Kreativität mangelt es Computern bisher noch.
[208] PROOFML baut auf den Grundlagen von Koepke & Schröder 2003 auf.

Publikationsserver des Instituts für Deutsche Sprache
URN: http://nbn-resolving.de/urn:nbn:de:bsz:mh39-82509

format eines Beweis-Validierers für natürlichsprachliche Texte dienen. Außerdem wäre es möglich, PROOFML im Bereich des *E-Learning* in „Tutoren" einzusetzen, die Lernende ans mathematische Beweisen heranführen.

PROOFML wurde hauptsächlich an einem kleinen Korpus von Beweisen entwickelt, das durch Material aus der Sekundärliteratur ergänzt wurde. Die Beweise des Beispiel-Korpus stammen einerseits aus den *Elementen* von Euklid, andererseits aus der *Einführung in die mathematische Logik* (Ebbinghaus et al. 1996). Sie decken mit Geometrie und Logik zwei sehr unterschiedliche und grundlegende Bereiche der Mathematik ab und sind auch stilistisch sehr verschieden; bei Euklid erscheinen z.B. kaum Formeln, und Namen tragen weniger Bedeutung. Alle Beispiele stammen zwar aus einführenden Werken[209] und präsentieren daher Beweise recht detailliert; doch finden sich bereits in diesen Texten für Neulinge *Lücken*, also für Menschen verständliche Vereinfachungen, die aber in einem formalisierten Beweis ergänzt werden müssen.

Nach einer Beschreibung der linguistischen und technischen Grundlagen von PROOFML werden für die Annotation relevante Eigenschaften mathematischer Texte beschrieben und die Grundsätze der Annotation in PROOFML erläutert.

2. (Computer-)Linguistische Grundlagen von PROOFML

Texttechnologie. Die Entwicklung von PROOFML war eine texttechnologische Arbeit; PROOFML orientiert sich an und integriert sich mit offenen Standards (z.B. MATHML und OMDOC), nicht zuletzt um redundante Entwicklungen zu vermeiden.

Semantische Repräsentation. Da PROOFML die Bedeutung mathematischer Texte repräsentieren soll, greift es auf linguistisch motivierte Theorien der semantischen Repräsentation zurück, um Dynamik und Kompositionalität der natürlichen Sprache wiederzugeben. Dies zeigt sich u.a. darin, daß Nominalphrasen als generalisierte Quantoren annotiert werden und dabei dynamische Existenzquantoren verwandt werden (vgl. 5.2), aber auch in der grundsätzlichen Strukturierung der Annotation in Propositionen.

Formale Pragmatik und Kontext. Wie in vielen Bereichen der Linguistik zeigt sich auch bei der Definition von PROOFML, daß es an vielen Stellen wichtig ist, den Einfluß des Kontextes, insbesondere des gemeinsamen Hintergrund-Wissens der Kommunikanten zu berücksichtigen. Zwar geht eine Formalisierung

[209] Allerdings darf man Euklids Werk einen deutlich größeren Einfluß zuschreiben, der einen Grund für die Auswahl der *Elemente* darstellte. Für die Verwendung der englischen Übersetzung sprach vor allem ihre Texttreue, der Wunsch, PROOFML auch an fremdsprachigen Texten zu entwickeln, und die Bedeutung des Englischen als Wissenschaftssprache.

(1) 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>
```

Abb. 1: Zwei Dreiecke

des Hintergrundwissens über den Rahmen der Arbeit hinaus, es sind aber Anbindungen an entsprechende Repräsentations-Modelle vorgesehen (vgl. 4.).

3. Technische Grundlagen von PROOFML

OPENMATH ist eine XML-Anwendung, in der Hintergrundwissen semiformal in einer maschinenlesbaren Form aufbereitet werden kann. OMDOC erweitert OPENMATH um Komponenten zur Beschreibung von Layout und Strukturierung von Texten sowie formalisierter Beweise. PROOFML ist eine inoffizielle Erweiterung von OMDOC, die Möglichkeiten zur semantischen und logischen Annotation natürlichsprachlicher Texte hinzufügt. PROOFML erweitert die Beschreibung der logischen Struktur formaler Beweisen von OMDOC (vgl. OMDoc : Kap. 19) um eine Annotation für die sprachliche Dimension.

4. Hintergrundwissen

Für das Verständnis eines Beweises sind sowohl Hintergrundwissen über die mathematischen Zusammenhänge als auch linguistisches und sprachliches Wissen notwendig, die freilich nicht immer klar zu trennen sind.

Zumindest die mathematischen Zusammenhänge repräsentieren mathematische Ontologien und *Proof Assistants* (vgl. Siekman et al. 1998, Mizar) sowie die typenlogischen Grammatiken bei Ranta (u.a. Ranta 1999), welche z.B. Existenzbeweise für im Text genannte Entitäten verlangen; im Bereich der XML-Anwendungen sind OMDOC-, OPENMATH- oder MATHML-Dokumente zu nennen. Für PROOFML müssen vor allem folgende Informationen im Hintergrundwissen vorhanden sein: (a) Definitionen der mathematischen Kategorien und Relationen. (b) Informationen über die Bedeutung, die in der Struktur von

Kennzeichnungen liegt (s. 5.4). (c) Informationen, die Entitäten verbinden: mereologische Beziehungen, Bridging-Relationen etc. (d) Informationen über Stelligkeit und Argumentstruktur der Relationen zwischen Entitäten oder Propositionen. (e) (ggf. domänenspezifische) Schluß- und Beweisregeln. Systeme, die mathematische Texte verstehend verarbeiten wollen, (Bauer 1999, Zinn 2004) bringen die linguistische und die mathematische Seite der Kontext-Repräsentation zusammen. Sie enthalten neben mathematischem Hintergrundwissen zumindest eine Grammatik – die wir hier außer Acht lassen können – und ein Lexikon. Für PROOFML ist vor allem wichtig, daß das Lexikon den semantischen Kategorien mathematische Kategorien und den Wort-Valenzen Argumentstrukturen mathematischer Relationen zuordnet. Von den *'Requirements for a theory of discourse representation'* für mathematische Beweise, die Zinn (2000: Kap. 2.3) formuliert, sind für unsere Annotation wichtig: (a) die Forderung nach einer strukturierten Datenstruktur (hier: XML-Dokument); (b) das Vorsehen von Variablen (in PROOFML: IDs, vgl. v.a. 5.2) und (c) die Behandlung von Namen (s. 5.4).

5. Annotation auf phrasaler Ebene

Dieses Kapitel beschreibt, wie die logische und die sprachlich-syntaktische Struktur von Phrasen und Sätzen in mathematischen Beweisen annotiert werden kann. In Bezug auf die logische Struktur orientiert PROOFML sich an den Beschreibungs-Modellen der formalen Semantik, die seit Frege meist auf der Funktor-Argument-Struktur beruhen. Syntaktische Konstituenz wird in PROOFML Oberflächen-orientiert durch Verschachtelung von XML-Elementen annotiert, um die Annotation nicht zu theorielastig zu machen. Syntaktische Dependenz spielt keine eigene Rolle, denn es wird nur die – oft gleiche – semantische Funktor-Argument-Struktur annotiert.

5.1. Die grundlegenden XML-Elemente

Natürlichsprachliche Ausdrücke stehen im allgemeinen in einem `<expression>`-Element; die Attribute dieses Elements werden im folgenden erläutert, hier sei nur kurz angedeutet, daß ein Attribut `id` („die ID") einen Ausdruck identifiziert und später mittels eines `refid`-Attributs darauf Bezug genommen werden kann. Außer `<expression>` gibt es noch weitere generelle Strukturierungs-Elemente. Das Element `<keyword>` kann Funktionswörter enthalten, die anzeigen, daß die durch das Elternelement annotierte Phrase zu einer bestimmten Kategorie gehört. Zum Beispiel kann in einem `<arg>`-Element eine Präposition bei einem Präpositionalobjekt ein `<keyword>` sein (vgl. Abb. 3). `<keyword>` kann auch in

(2) AB is equal to BC

```
<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. 2: Gleichheit

einem Element vorkommen, das die logische Struktur beschreibt (s. 6.). Man sollte verhindern, daß ein Textknoten als Geschwister von Elementen auftritt, indem man ein <word>-Element setzt, das ggf. auch mehrere Wörter aufnehmen kann (s.a. Abb. 3). Das Element `<ling>` enthält Wörter, die für die formale Semantik keinen wichtigen Beitrag leisten (aber oft für den Fluß des Textes relevant sind), z.B. „Füllwörter" und Gradpartikeln wie *auch*, *aber*.

Annotation von Funktor-Argument-Strukturen. Ein `<expression>`-Element kann Kinder vom Typ `<arg>` haben, die entweder ein semantisches Argument eines Funktors enthalten oder mit dem Attribut `refid` auf ein solches Element verweisen. Die Argument-Struktur ist im Hintergrundwissen festgelegt; mehrere Argumente werden ggf. durch ihre Position identifiziert. Im Beispiel (s. Abb. 4) sind die beiden Argumente zu *haben* die besitzenden Dreiecke (mit `name="T1"`), und die besessenen Seiten (mit `name="S1"`). In Sprachen, die weniger feste Wortstellung haben als etwa das Englische, würde eine morphologische Analyse sicherlich die Zuweisung der Argumente zu ihren Rollen vereinfachen.

5.2. Nominalphrasen

An der Annotation von Nominalphrasen lassen sich alle wichtigen Prinzipien der Annotation von Phrasen in ProofML zeigen. Mathematisch gesprochen: Die Annotation anderer Konstituenten erfolgt *analog*. Wir nehmen für Nominalphrasen im allgemeinen eine Denotation vom Typ „Individuum" (hier vielleicht passender: „Objekt") an.[210] Wir gehen davon aus, daß die Denotation von Variablen, Namen, Pronomina und Nominalphrasen gleichartig ist.

Einführung und Fortführung von Diskursreferenten. Einem Objekt wird, wenn es das erste Mal im Kontext eines Beweises erscheint, eine ID zugeordnet,

[210] In PROOFML ist bisher keine Intensionalität vorgesehen; dies wäre nur notwendig, wenn z.B. über unmögliche Objekte gesprochen würde. Im Korpus kamen solche Fälle nicht vor.

(3) 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.

```
<expression type="object" form="NP">
  <coreference><arg name="S1"/><arg name="S2"/></coreference>
  <word>the equal sides</word>
</expression>
```

Abb. 3: Split Antecedents

welche es für den gesamten Text eindeutig identifiziert. Dies ist ähnlich zu verstehen wie die Einführung einer existenzquantifizierten Variablen in der Dynamischen Prädikatenlogik (Groenendijk & Stokhof 1991), die ihren Geltungsbereich in der Proposition hat, in der sie das erste Mal auftritt. Solche dynamischen Existenzquantoren zeichnen sich durch zwei Eigenschaften aus (vgl. Abb. 5): ihr Geltungsbereich ist bei Aneinanderreihungen von Sätzen (modelliert als Konjunktionen) nach hinten quasi offen; außerdem können Implikationen über Existenz-Aussagen in All-Aussagen umgewandelt werden (vgl. Geachs Sätze von Bauern und Eseln; viele mathematische Beweise beginnen wie in (4.a) schließen wie in (4.b)).

(4) a. Sei A ein Punkt[...]
b. Also gilt für alle Punkte, daß [...]

Koppelung ans Lexikon. Wortformen erhalten im PROOFML-Korpus Typen nach dem Muster `type="`*klasse*`:`*typ*`"`. Dies soll auf den Lexikoneintrag verweisen, zumindest auf die Valenz der annotierten Wortform. Für Phrasen gliedern wir den `type` gröber: `"(named-)proposition"`, `"object"` etc. bedeuten jeweils, daß die entsprechende Phrase als Denotation eine Proposition oder ein Objekt hat. Eine genauere Einteilung scheint schwierig, da kaum zu klären wäre, wie speziell etwa ein „Objekt" zu annotieren wäre: als Dreieck, als Vieleck oder als geometrische Figur.

Numeri und Konjunktion. Den Numerus einer NP gibt das Attribut `nr` an, nämlich `"sg"` (Singular, der Standardfall) und `"pl"` (Plural). Das Attribut `pl-type` gibt den Typ des Plurals an, sofern die NP im Plural steht, `"distributive"` oder `"collective"` (Default) für distributive oder kollektive Plurale. Das Attribut `form` gibt Beziehungen zwischen Elementen wieder; `form="conj"` etwa bezeichnet die Konjunktion, wie sie in der natürlichen Sprache mit „und" vor-

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

Abb. 4: Funktor-Argument-Struktur mit Verweisen auf Argumente

genommen 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.6.1.

NPs als generalisierte Quantoren. Phrasen, die der Form nach als generalisierte Quantoren (vgl. z.B. Barwise & Cooper 1981), annotiert werden sollten, werden in ein `<quant>`-Element geschachtelt und, soweit möglich, aufgeteilt in Determinierer (`<det>`) und Restriktor (`<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 eingeschachtelt ist. Das Attribut `type` bei Quantoren gibt an, ob es sich um einen (dynamischen) Existenz- (`type="exist"`, Default), All- (`type="all"`) oder einen anderen Typ generalisierter Quantoren (`type="gen"`) handelt. Einem `<scope>`-Element kann ein Quantor mit einem `<scope-of>`-Element zugewiesen werden, dessen `refid`-Attribut auf die ID der Variable verweist, die durch den Quantor gebunden ist (analog für Negationen `<neg>`). Ein Name, der eine von einem Quantor gebundene Variable bezeichnet (er ist üblicherweise dem Restriktor beigeordnet), verweist mit einem `refid`-Attribut auf sie. Bei Kardinalzahl-Quantoren kann es geschehen, daß mehrere Namen angegeben werden (vgl. Abb. 1). In diesem Fall ist es dennoch 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. Quantoren der Art „einige", „zwei", ... behandeln wir als Existenzquantoren.

Definite Nominalphrasen. Definite Nominalphrasen werden als `<expression type="object">` annotiert. Sie enthalten meist ein `<det>`-Element sowie eine Annotation eventueller näherer Bestimmungen (Substantive, Adjektive, Präpositionalphrasen, Namen etc.). Nominalphrasen, die mit anderen Nominalphrasen koreferent sind, zeigen dies mit dem Attribut `refid` an. Gelegentlich werden bei

Unabhängig davon, ob im folgenden x in α oder β frei vorkommt, gilt:

$$(\exists x.\alpha) \wedge \beta \Leftrightarrow (\exists x.\alpha \wedge \beta)$$
$$(\exists x.\alpha) \rightarrow \beta \Leftrightarrow (\exists x.\alpha \rightarrow \beta)$$
$$(\exists x.\alpha \rightarrow \beta) \Leftrightarrow (\forall x.\alpha \rightarrow \beta)$$

Abb. 5: Regeln der DPL nach Groenendijk & Stokhof 1991: 65

der Einführung benannter Objekte auch definite NPs verwandt (Bauer 1999: 33).[211]. In diesem Fall wird wie bei indefiniten NPs annotiert.

5.3. Koreferenz

Koreferenz ist in der Annotation vor allem auf der Ebene der Nominalphrasen (einschließlich der Pronomina) zu erfassen, und zwar indem in den 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. Für den Fall, daß sich Anaphern auf zwei getrennt stehende Antezendentia (*'Split antecedents'*) beziehen, führen wir das Element `<coreference>` ein, dessen `<arg>`-Kinder auf die Antezendentia verweisen (vgl. Abb. 3). Man kann die Angabe der Koreferenz bei `<name>` und `<expression>` mit dem Attribut `refid` als eine Abkürzung für ein `<coreference>`-Element mit nur einem `<arg>`-Kind sehen, die in der Mehrzahl der Fälle ausreicht.

Bridging. Unter *Bridging* verstehen wir hier alle Fälle, in denen Diskursreferenten verfügbar sind, die nicht explizit eingeführt wurden.[212] Wir annotieren Bridging ähnlich wie Existenzbeweise in den Editionen der *Elemente*: dort werden die Sätze, Definitionen oder Postulate des Textes angegeben, die eine Konstruktion erlauben, im Beispiel (5) zunächst das erste Postulat (αἴτ 1), dann Beweis I.1 (prop. I) aus den *Elementen*. Die Benennungskonventionen für Dreiecke erlauben, daß wir über jeden beliebigen Punkt und jede beliebige Seite des Dreiecks namentlich sprechen; dies muß also im Hintergrundwissen definiert sein; im Beispiel als Regel `"naming:triangle"`. Die Annotation für Bridging (`<bridged-from>`) gibt im wesentlichen das Wissen an, das auch einem menschlichen Leser vorliegen muß (Abb. 6): eine Bridging-Regel (`"naming: triangle"`) und ggf. Parameter (hier: die ID des Dreiecks und den Punkt *A*).

---

[211] Dies ist nicht weiter verwunderlich, wenn man die Variablen-Namen als Eigennamen sieht.

[212] Ansätze zur semantischen Modellierung dieser Phänomene finden wir z.B. in der *Diskursrepräsentations-Theorie* (vgl. van Eijck & Kamp 1997 : Kap. 9).

(5) Ἐπεζεύχθω γὰρ ἀπὸ τοῦ A σημείου ἐπὶ τὸ B σημεῖον εὐθεῖα ἡ AB, καὶ συνεστάτω ἐπ' αὐτης τρίγωνον ἰσόπλευρον τὸ ΔAB, [...]
ducatur enim a puncto A ad B punctum recta AB [αἴτ 1], et in ea construatur triangulus equilaterus ΔΔAB [prop. I], [...]
(Elemente, Beweis I.2)

Sätze als Diskursreferenten. Es ist in mathematischen Texten üblich, auf Sätze zu verweisen, wenn diese als Rechtfertigung für einen Schluß dienen; diese erhalten dann üblicherweise einen Namen und werden annotiert als `<expression type="named-proposition">`, die wiederum einen Namen und ein Element vom Typ `<expression type="proposition">` enthält.

Formeln. Diese Arbeit beschäftigt sich zwar vornehmlich mit dem natürlichsprachlichen Anteil der mathematischen Beweise, jedoch ist dieser oft so eng mit darin vorkommenden Formeln verwoben, daß sie kaum zu trennen sind. So können in Formeln Referenten eingeführt werden, auf die später Bezug genommen wird; es muß also ggf. eine ID vergeben werden. Formeln im fortlaufenden Text stehen in einem `<formula>`-Element, das MATHML-Präsentations-Markup enthalten kann.[213] Außerdem ein umschließendes `<expression>`-Element anzeigen, ob die Formel ein Objekt bezeichnet oder für einen Satz steht (s. 6.1).

5.4. Namen

Für mathematische Texte ist es charakteristisch, mathematische Entitäten so zu benennen, daß Ambiguitäten reduziert und komprimiert Informationen übermittelt werden. Namen identifizieren üblicherweise eine Entität eindeutig (vgl. (6)), aber gelegentlich treten Ambiguitäten auf. Bei Euklid können etwa Namen aus drei Buchstaben Dreiecke, Winkel und Kreise bezeichnen, (s. (7), eigentlich handelt es sich um ein Tripel von Referenzpunkten). In ProofML werden Namen (`<name>`) daher durch das Attribut `form` in einfache Namen (`"simple"` wie in (6)) und zusammengesetzte (`"compound"`, wie in (7)) unterschieden. Zusammengesetzte Namen können weiter aufgelöst werden in einzelne Namen.[214] Oft kann man aus der Form von Namen auf die Art von Objekt schließen, das sie bezeichnen (vgl. (8)); dies kann über das Attribut `class` erfaßt werden.

---

[213] Hier sind Formeln sehr einfach annotiert, damit sie nicht vom eigentlichen Thema der Arbeit ablenken.

[214] Der Übersichtlichkeit halber nimmt dieser Text diese Auflösung allerdings selten explizit vor.

```
<expression type="object:triangle" id="ABC">
  das Dreieck ABC
</expression>

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

Abb. 6: Bridging (vereinfachte Annotation)

(6) a. der Punkt A b. die Strecke C
(7) a. die Strecke AB b. das Dreieck ABC c. der Kreis ABC d. der Winkel ABC.
(8) Ziehe DC.

Bedeutung des grafischer Merkmale. Häufig trägt bei einem Namen die Auswahl einer bestimmten Schriftart oder eines Alphabets und sogar der Groß- und Kleinschreibung eine bestimmte Bedeutung: in der Logik stehen griechische Buchstaben oft für Ausdrücke der Objektsprache, kleine für einzelne Propositionen und große für Propositionen-Mengen; kleine lateinische Buchstaben für Individuenvariablen, große für Prädikatenvariablen etc. Die mathematische Formelsprache verwendet auch diakritische Zeichen, und zwar auf dreierlei Weise: (a) als *Symbole für Funktionen*, z.B. liefert die Ableitungsfunktion („′“) im Ausdruck f' die erste Ableitung von f; (b) für *Funktionsargumente*, etwa Laufindizes bei Summen ($\sum x_i$) und schließlich (c) zur Erweiterung des Namensraums, etwa wenn mehrere Mittelpunkte M_1, M_2, M_3 heißen. In PROOFML ist vorgesehen, das Layout von Formeln durch MATHML und Unicode zu erfassen; die jeweiligen Konventionen bei der Verwendung von Zeichensätzen und Diakritika sollten im Hintergrundwissen definiert sein, so daß sie bei der maschinellen Verarbeitung von Namen genutzt werden können.

5.5. Ellipsen

In den Texten des Korpus, das wir für die Erstellung von PROOFML verwandt haben, kam nur eine Ellipse vor, nämlich (3) (s. Abb. 3). Es ist offensichtlich, daß man bei *those* das Wort *angles* ergänzen muß. In diesen Fällen wird durch ein leeres `<expression>`-Element der Lexikon-Eintrag des Wortes nachgetragen, das „fehlt“ (ähnlich bei Lücken, vgl. 6.3), ein `refid`-Attribut kann auf die ID der Elemente verweisen, anhand derer man ergänzt (also hier etwa „angles“).

6. Annotation auf logischer Ebene

Dieser Abschnitt behandelt die Annotation der logischen Verhältnisse innerhalb mathematischer Beweise und erweitert die Fähigkeiten von OMDoc für formalisierte Beweise um Annotationen für natürlichsprachliche Texte.

6.1. Propositionen

Einteilung in Propositionen. Die logische und die sprachliche Form entsprechen einander nicht immer vollständig, so daß wir Sätze gelegentlich in Propositionen einteilen müssen, die der sprachlichen Struktur widersprechen. Beispielsweise können Nominalisierungen oder Nebensätze Informationen enthalten, die eine eigene Rolle in der logischen Struktur des Textes enthalten: etwa Prämissen oder Folgerungen. PROOFML folgt der sprachlichen Struktur des Beweises soweit, wie möglich. Insofern ist als eine Proposition (`<expression type="proposition">`) maximal ein Satz zu annotieren, der durch ein Satzendezeichen beendet wird. Eine minimale Proposition im natürlichsprachlichen Text besteht aus einem (logischen) Prädikat oder Quantor und seinen Argumenten. Formeln im laufenden Text können dabei grundsätzlich den Wert von Propositionen haben; z.B. kann a (als einfacher Variablenname) keine Proposition sein, wohl aber $a = 42$. Kon- und Disjunktion können einerseits Propositionen, andererseits Entitäten verbinden; wir verwenden für die Konjunktion `form="conj"` in beiden Fällen, da dies intuitiv am einfachsten ist und keine Probleme für die Semantik der Annotation entstehen. Bei Entitäten ist zu unterscheiden zwischen kollektiver und distributiver Konjunktion (vgl. auch 5.2). Disjunktionen zwischen Entitäten sind im Korpus und der Sekundärliteratur nicht aufgetreten; es scheint aber, daß eine Ergänzung zu zwei (oder mehr) Propositionen (vgl. 6.3) sinvoll wäre. Logische Verknüpfungen zwischen Propositionen werden über das `form`-Attribut eines umschließenden `<expression>`-Elements angegeben. Definiert sind Kon- und Disjunktion (`"conj"`, `"disj"`), Implikation (`"implication"` *wenn, dann* und `"rimplication"` *gilt, wenn*) und Äquivalenz (`"equivalence"`).

Die übliche Übersetzung eines Satzes mit einer definiten NP in (dynamische) Prädikatenlogik wie in (9) zeigt zwei Propositionen, die durch eine Konjunktion verbunden sind.[215] Da in keinem Fall unseres Korpus die Eigenschaft alleine als Prämisse oder Annahme für einen Schluß auftritt, sind in unseren Beispielen in solchen Appositionen immer „Punkt" und andere Eigenschaften als `<expression type="property:point">` etc. ohne explizite Konjunktion annotiert. Im Falle einer indefiniten NP ist die Interpretation eines solchen Satzes in mathematischen Beweisen – wie sie uns vorgelegen haben – generisch, wie in (10). In

[215] Den Existenzquantor verwenden wir, weil wir davon ausgehen, daß *A* als Diskursreferent eingeführt wird. Die Definitheit erklärt sich aus dem Gebrauch eines Namens.

solchen Sätzen wird die NP als generalisierter Quantor annotiert, dabei ist die Eigenschaft der Restriktor. Für Adektivphrasen verfährt man sinngemäß ebenso wie für beigeordnete Eigenschaften. Relativsätze, Partizipialkonstruktionen u.ä. werden genauso annotiert wie Hauptsätze als eigenständige Propositionen; wir gehen davon aus, daß sie mit ihren Hauptsätzen durch eine (logische) Konjunktion oder eine Implikation verbunden sind. 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) Der Punkt A liege auf g $\exists A : [(punkt' A) \wedge ((lieg' g) A)]$

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

6.2. Schlußregeln

Wir nehmen die Beweis-Annotation von OMDoc als Grundlage unserer Annotation; wir übernehmen also auch die Annotation für Schlußregeln von OMDoc (OMDoc, Kap. 19). Daher besitzt PROOFML Elemente für Hypothesen (`<hypothesis>`), Behauptungen (`<assertion>`), Beweise (`<proof>`), Schlußregeln (`<method>`), Beweisschritte (`<derive>`), den letzten Schluß eines Beweises (`<conclude>`) und für Kommentare zum Beweisgeschehen, die keinen direkten Beitrag zum logischen Verlauf des Beweises leisten, aber oft einem menschlichen Leser das Verständnis erleichtern (`<metacomment>`, vgl. (11)); 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.

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

Erweiterung von OMDoc-Prämissen und Schlußregeln. Schlußregeln können im mathematischen Text natürlichsprachlich ausgedrückt sein (s. Abb. 7). PROOFML sieht daher das Element `<nl-method>` (eine Erweiterung des `<method>`-Elements von OMDoc) vor, das eine Referenz auf die Schlußregel (wie `<method>`) erhält, aber auch den Text, der sie identifiziert. Eventuelle Parameter der Regel können mit `<arg>`-Elementen angegeben werden. Von OMDoc übernehmen wir die Unterscheidung zwischen `<CMP>`-Elementen, die natürlichsprachliche Texte (aber auch `<formula>`-Elemente) enthalten, und `<FMP>`-Elementen, welche formalisierte Darstellungen enthalten. In PROOFML sind `<FMP>`-Elemente v.a. notwendig, wenn Lücken (vgl. 6.3) gefüllt werden. Bereits

```
<derive id="III-4.6-a">
  <nl-method  xref="Induktion">
    durch Induktion über den Aufbau der Terme
  </nl-method>
  ... <!-- Hier stünden Prämissen und Konklusion -->
</derive>
```

Abb. 7: Natürlichsprachlich angegebene Schlußregel (EFT96: 39)

bekannte Prämissen werden in natürlichsprachlichen Texten oft noch einmal genannt; daher führen wir zusätzlich zum OMDoc-Element `<premise>` – das nur einen Verweis auf eine Prämisse zuläßt – ein Element `<nl-premise>` ein, das auch Text und einen Verweis (wie `<premise>` mit `xref`) auf Propositionen oder Beweisschritte enthalten kann, deren Inhalt sie wiederholt.

Parametrisierte Regeln. 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 und so anzuzeigen, wo der Widerspruch besteht, und was man daraus folgert.

6.3. Lücken und Analogien

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

Wie bereits erwähnt, treten in mathematischen Beweisen gelegentlich Lücken auf, und zwar in zwei Arten: Analogien und „echte Lücken". Manche solcher Lücken sind für den eingeweihten Leser leicht zu füllen. Einen solchen Fall haben wir in unseren Beispielen (12). Echte Lücken können geschlossen werden, indem man eine Ableitung aus mehreren `<derive type="lacuna">`-Schritten einführt, die `<FMP>`-Elemente enthalten.

Analogien stellen eine wichtige Technik zur Abkürzung mathematischer Beweise dar, z.B. in einer Fallunterscheidung. Man zeigt einen Fall und gibt dann an, daß in den anderen Fällen „analog" zu verfahren ist. Üblicherweise besteht die Analogie darin, daß bestimmte Diskursreferenten oder Prädikate ausgetauscht werden. PROOFML stellt das Element `<sem-relation type="symmetry">` zur Verfügung, dessen Kinder den Austausch von Diskursreferenten durch die Angabe von Verweisen (über IDs) angeben. Ggf. sollte eine genauere Spezifikation allerdings implementationsbezogen gewählt werden.

6.4. Pragmatisch relevante, logisch irrelevante Relationen

Zwischen Sätzen können Beziehungen bestehen, die in der streng formal-logischen Struktur keine Bedeutung haben, aber in der Diskurs-Struktur eine bestimmte Rolle erfüllen, z.B. Wiederholungen, inhaltliche Parallelen etc.. PROOFML sieht generell eine Annotation für solche Relationen vor (`<relation>`), sie gehen aber über den eigentlichen Rahmen der Arbeit hinaus.

7. Schluß

PROOFML ist eine XML-Anwendung, die die formal-logische Struktur natürlich-sprachlich formulierter mathematischer Beweise soweit explizit macht, daß eine maschinelle Verarbeitung möglich sein sollte, wenn entsprechende Beschreibungen des sprachlichen und mathematischen Hintergrundwissens vorliegen. Es wurde gezeigt, wie die Struktur eines Beweises in Beweisschritte zergliedert und annotiert werden kann. Unter anderem wurde das Konzept der dynamischen Quantoren und die Behandlung von Formeln und Kennzeichnungen in PROOFML demonstriert.

Literatur

Abrahams, P.W. (1964). Application of LISP to Checking Mathematical Proofs. In Berkeley, E.C. & D.G. Bobrow (Hg.), The Programming Language LISP: Its Operations and Applications, S. 137–160. Cambridge, London. MIT Press.

Asperti, A., M. Kohlhase & C. Sacerdoti Coen (2003a). Prototype n. D2.b Document Type Descriptors: OMDoc Proofs. Mowgli deliverable, The MoWGLI Project.

Barwise, J. & R. Cooper (1981). Generalized quantiers and natural language. Linguistics and Philosophy, 4:159–219.

Baur, J. (1999). Syntax und Semantik mathematischer Texte – ein Prototyp. Diplomarbeit, Universität des Saarlandes.

Ebbinghaus, H.-D., J. Flum & W. Thomas (1996). Einführung in die mathematische Logik. 4. Aufl., korr. Nachdruck 1998. Heidelberg: Spektrum.

van Eijck, Jan und Hans Kamp (1997). Representing Discourse in Context. In Handbook of Logic and Linguistics, Kap. 3, S. 179–237. Elsevier.

[Elemente deutsch] Euclides. Die Elemente: Bücher I–XIII / von Euklid. (Oswalds Klassiker der exakten Wissenschaften 235). Hrsg. v. C. Thaer. Reprint, 3. Aufl. Thun, Frankfurt/M.: Deutsch, 1997.

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

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

[Elemente griechisch/lateinisch] Euclidis Elementa edidit et Latine interpretatus est I. L. Heiberg, Uol. 1, Libros I–IV continens. Leipzig: Teubner, 1883. http://perseus.mpiwg-berlin.mpg.de/cgi-bin/ptext?lookup=Euc.+1

Eisenreich, G. (1998). Die neuere Fachsprache der Mathematik seit Carl Friedrich Gauß. In Hoffmann, L., et al. (Hg.), Fachsprachen / Languages for Special Purposes: ein internationales Handbuch zur Fachsprachenforschung und Terminologiewissenschaft / An International Handbook of Special-Language and Terminology Research, 1. Halbband, Handbücher zur Sprach- und Kommunikationswissenschaft 14, Kap. 136, S. 1222–1230. Berlin, New York: de Gruyter.

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

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

Groenendijk, J. & M. Stokhof (1991). Dynamic Predicate Logic. Linguistics and Philosophy, 14(1):39–100.

Koepke, P. & B. 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, Sprachwissenschaft, Computerlinguistik, Neue Medien 5, hrsg. von N. Weber, S. 428–441. St. Augustin: Gardez!

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

Carlisle, D., P. Ion, R. Miner & N. 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

[OMDOC] Kohlhase, M. (to appear). OMDOC An open markup format for mathematical documents (Version 1.2). Techn. Ber., Computer Science, Carnegie Mellon University.

[OPENMATH] Caprotti, O., D.P. Carlisle & A.M. Cohen (2000). The Open Math Standard, Version 1.0. Techn. Ber., The Open Math Society.

Ranta, A. (1997). Structures grammaticales dans le français mathématique. Mathématiques, informatique et Sciences Humaines, 138:5–56; 139:5–36.

Siekmann, J., M. Kohlhase & E. Melis (1998). Omega: Ein mathematisches Assistenzsystem. Kognitionswissenschaft.

Zinn, C. (2000). Computing Presuppositions and Implicatures in Mathematical Discourse. In Bos, J. & M. Kohlhaase (Hg.), Workshop 'Inference in Computational Semantics' (ICoS-2). Schloß Dagstuhl.

Zinn, C. (2004). Understanding Mathematical Discourse (vorläufige Version 12. Juni 2003). Dissertation.