Kapitel VI:
Beweissysteme mit Logiken höherer Stufe

(Michael Kohlhase)

Ein ursprüngliches Ziel der Künstlichen Intelligenz ist es, Deduktionssysteme zu bauen, die bei der Suche nach Beweisen in der Mathematik ähnlich leistungsfähig und vielseitig einsetzbar sind wie der Mensch. Die bisher in diesem Buch vorgestellten Systeme basieren auf der Prädikatenlogik erster Stufe, in der man nur einen Teil der Mathematik natürlich formalisieren kann. Um aber dieses große Ziel verwirklichen zu können, benötigt man ein logisches System, das die ganze Mathematik umfaßt. Wir werden deshalb im ersten Teil dieses Kapitels etwas ausführlicher Formalisierungen der Logik höherer Stufe diskutieren und darlegen, warum die Typtheorie eine besonders geeignete Basis für das automatische Theorembeweisen in der Mathematik ist. Anschließend werden wir auf die Unifikation in der Typtheorie eingehen und uns mit Huets Kalkül der Constrained Resolution als einem Beispiel für die Mechanisierung der Typtheorie genauer auseinandersetzen.

Wegen der Komplexität des Themas benötigen wir in diesem Kapitel einen höheren Aufwand an Formalismus als das in den vorhergehenden Teilen des Buchs notwendig war. Trotzdem sollen auch in diesem Kapitel die Ideen im Vordergrund stehen. Wir empfehlen deswegen, beim ersten Lesen alle Formeln, die nicht sofort einsichtig sind, zu überspringen und sich weiter auf die Entwicklung der Ideen zu konzentrieren. Eine weitere Technik für das Lesen von Formeln in der Typtheorie besteht darin, die Typen der Symbole außer acht zu lassen, da sie weniger zum Verständnis als vielmehr zur Wohldefiniertheit der Formeln beitragen. Die Erfahrung zeigt, daß sich nach einer Weile ein gewisser Gewöhnungseffekt einstellt und die Formeln dadurch leichter lesbar werden. Wir hoffen, daß mit diesen Vorsichtsmaßnahmen der Formalismus in diesem Kapitel nicht zu abschreckend ist.

1 Einführung in die Typtheorie

In der Mathematik gibt es *einfache Objekte*, wie Zahlen oder Punkte der euklidischen Ebene, und es gibt *kompliziertere Objekte*, wie Mengen, Funktionen oder Vektorräume. Die Prädikatenlogik erster Stufe (PL1) beschränkt sich darauf, gerade die einfachen Objekte zu beschreiben, indem sie ausschließlich Variablen für diese zuläßt (siehe auch Kapitel II, Abschnitt 2.4). Da sich alle komplizierten Objekte der Mathematik auf Mengen zurückführen lassen, haben Logiker wie Georg Cantor versucht, Mengen zu formalisieren. Cantor definierte Mengen als „Zusammenfassungen von bestimmten und wohlunterschiedenen Objekten unserer Anschauung oder unseres Denkens, die in Gottes Angesicht eine Einheit bilden“. Diese Definition setzt die Existenz eines zweistelligen Prädikats $\in$ voraus, das zu jedem Objekt A und zu jeder Menge M angibt, ob A ein Element von M ist oder nicht. Schränkt man dieses Prädikat auf eine konkrete Menge M ein, so erhält man gerade die definierende Eigenschaft $\in M$ dieser Menge. Dieser naive Ansatz führt allerdings recht schnell zu Problemen, so ist z.B. „die Menge M aller Mengen, die sich nicht selbst enthalten“ kein sinnvolles mathematisches Objekt, denn M wäre

ein Element von M genau dann, wenn $M \notin M$. Dieses Phänomen (Russells Paradoxon) zeigt, daß das Prädikat $\in$ nicht wohldefiniert ist und deshalb für die Formalisierung von Mengen in der Prädikatenlogik noch mehr Arbeit investiert werden muß.

Zum einen kann man die Wohldefiniertheit des Prädikats $\in$ erreichen, indem man die Modelle der Prädikatenlogik durch Axiomensysteme so weit einschränkt, daß sie keine Paradoxa mehr enthalten. Zum anderen kann man den mengentheoretischen Ansatz aufgeben und die Logik direkt um Konstrukte für komplizierte Objekte erweitern. Bei beiden Ansätzen hat eine Stufentheorie der Mathematik zum Erfolg geführt. Diese Theorie beruht auf der Idee, die Unterscheidung der Objekte in einfache und komplizierte zu verfeinern und allen Objekten – insbesondere auch den Mengen – eine sogenannte *Stufe* zuzuordnen.

Die *axiomatische Mengentheorie* schränkt die Bildung von Mengen dahingehend ein, daß eine Menge nur Elemente enhalten darf, die eine kleinere Stufe haben als die Menge selbst. Die wichtigsten Beispiele für diesen Ansatz sind das Axiomensystem von Zermelo und Fraenkel und das von Gödel, Bernays und von Neumann (siehe [Ebb77]). Da sich mit diesen die ganze bisher bekannte Mathematik formalisieren läßt, sind sie heute weitgehend als Grundlage der Mathematik anerkannt. Mit diesen Axiomensystemen ließe sich PL1 prinzipiell als Basis von Deduktionssystemen für die Mathematik verwenden, aber die Darstellung mathematischer Objekte ist in der mengentheoretischen Kodierung sehr umständlich. Zum Beispiel ist die Kodierung von Funktionen als „linkstotale, rechtseindeutige Relationen“ wenig intuitiv. Dieser Zugang ist also für die Mechanisierung der Mathematik wenig geeignet, deshalb wollen wir ihn nicht weiter behandeln.

Die Idee bei der Erweiterung der Prädikatenlogik besteht darin, einerseits Konstrukte für komplizierte Objekte zur Verfügung zu stellen, andererseits durch die Einführung von sogenannten *Typen* ein syntaktisches Mittel zu schaffen, das die Konsistenz des System sicherstellt. Da verschiedene Konstrukte aufgenommen werden können, gibt es verschiedene Erweiterungen. Um diese Systeme klassifizieren zu können, wurde der Begriff der Stufe eines logischen Systems eingeführt. Es hat sich eingebürgert, den Objekten der Mathematik eine *Ordnung* zu geben. Einfache Objekte (*Individuen* und *Wahrheitswerte*) haben die Ordnung 0 und *Funktionen*, deren Argumente und Werte maximal die Ordnung n haben, die Ordnung n+1. Damit können wir nun den Begriff der *Stufe* eines logischen Systems definieren. Eine Logik ist $(2n\text{-}1)$-ter Stufe, wenn die Ordnung der vorkommenden Variablen und Konstanten maximal n ist und nicht über Variablen der Ordnung n quantifiziert wird. Ist nur die erste Teilbedingung erfüllt, so ist die Logik $(2n)$-ter Stufe. PL1 ist gerade ein Beispiel für eine Logik erster Stufe. Man kann zeigen, daß das Teilsystem der Formeln, die keine Quantoren enthalten, isomorph zu der Aussagenlogik ist. Diese ist deswegen ein Beispiel einer Logik nullter Stufe (PL0). Es erscheint natürlich, das System von PL1 schrittweise um Variablen höherer Stufe und das Konzept der Quantifikation darüber zu erweitern. Dadurch erhält man Systeme von Logiken beliebig hoher Stufe. Anhand des so gewonnenen Systems PLΩ versuchen wir anschließend die *Typtheorie* zu erklären und zu motivieren.

1.1 Typen

Als eine Möglichkeit, Selbstbezüge wie in Russels Paradoxon auszuschließen, hat B. Russell in [Rus08] ein System von Typen für Formeln und Terme eingeführt, das Alonzo Church dann in [Chu40] zu der Form vereinfacht hat, die wir hier vorstellen werden. Diese Typen orientieren sich an der Semantik der Objekte, die sie beschreiben. Aus den sogenannten *Basistypen* o (omikron) der Wahrheitswerte und ι (iota) der Individuen werden induktiv alle anderen Typen gewonnen: Sind $\beta_1,\ldots,\beta_n$ und α Typen, so ist $(\alpha[\beta_1,\ldots,\beta_n])$ der Typ der *n*-stelligen *Funktionen*, deren Argumente die Typen $\beta_1,\ldots,\beta_n$ und deren Werte den Typ α haben. Wir nennen alle Typen, die nicht Basistypen sind, *Funktionstypen*. Zum Beispiel erhalten in der PL1 3-stellige Funktionssymbole den Typ (ι[ιιι]) und 2-stellige Prädikatensymbole den Typ (o[ιι]). Formeln und Terme in PL1 beschreiben Wahrheitswerte beziehungsweise Individuen und erhalten deswegen die Typen o und ι. Alle Formeln und Terme müssen in Russels getypter Logik *wohlgetypt* sein, das heißt, ein Term $f(t^1,\ldots,t^n)$ wird nur dann als syntaktisch korrekt (und sinnvoll) angesehen, wenn *f* vom Typ $(\alpha[\beta_1,\ldots,\beta_n])$ ist und die t^i die Typen β_i haben.

Um die Wirkungsweise der Typen zu veranschaulichen, betrachten wir noch einmal Russells Paradoxon. In einer Prädikatenlogik zweiter Stufe könnte man die paradoxe Menge *M* durch die Formel

$$\forall Q \;\; M(Q) \Leftrightarrow [\text{Menge}(Q) \wedge \neg Q(Q)]$$

definieren. In dieser Formalisierung müßte *Q* sowohl den Typ $(o[\alpha])$ als auch den Typ α haben, damit der Ausdruck *Q*(*Q*) wohlgetypt ist, denn das Symbol *Q* wird sowohl als Prädikats- als auch als Individuenvariable verwendet. Es hat sich herausgestellt, daß durch solche (und ähnliche) Typsysteme alle bekannten Arten von Paradoxa verhindert werden können. Aus diesem Grund haben sich getypte Systeme höherer Stufe bei der Beschreibung der Mathematik durchgesetzt.

In einer Logik höherer Stufe reicht es aus, explizit nur einstellige Funktionen zu betrachten. Sind zum Beispiel die Individuen die natürlichen Zahlen, so können wir die zweistellige Additionsfunktion *plus*, die zwei natürliche Zahlen auf ihre Summe abbildet, auch als eine Funktion ansehen, die einer Zahl *n* diejenige einstellige Funktion *plus-n* zuordnet, die zu jeder Zahl *n* addiert. Die Additionsfunktion kann also einerseits durch eine Funktionskonstante *plus* vom Typ (ι[ι,ι]), andererseits auch durch eine Funktionskonstante vom Typ (ι[(ι[ι])]) dargestellt werden. Im zweiten Fall würde zum Beispiel "3+4" durch *plus*(3)(4) dargestellt, zu *plus-3*(4) und dann in einem weiteren Schritt zu der Zahl 7 ausgewertet werden. Wir werden uns im folgenden auf einstellige Funktionen beschränken und den Typ $(\alpha[\beta])$ kurz als $(\alpha\beta)$ schreiben. Typen sind also von der Form o, ι oder $(\alpha\beta)$, wobei α und β wieder Typen sind. Der Trick, mehrstellige Funktionen durch einstellige Funktionen höherer Stufe darzustellen, ist in der Logik von vielen Forschern unabhängig entdeckt worden, und wird im Deutschen *Schönfinkeln* nach dem Logiker Moses Schönfinkel (englisch *currying* nach Haskell.J.B. Curry) genannt. Vereinbart man in Typen die Konvention der Linksklammerung, so ist $(\alpha(\beta_1(\beta_2\ldots\beta_n)\ldots))$ gerade $(\alpha\beta_1\beta_2\ldots\beta_n)$ und entspricht damit in der Darstellung gerade wieder dem ursprünglichen $(\alpha[\beta_1,\ldots,\beta_n])$.

1.2 Syntax von PLΩ

Um die im vorherigen Abschnitt vorgestellten Ideen zu präzisieren, führen wir jetzt ein logisches System PLΩ ein, das für alle natürlichen Zahlen n die Logik n-ter Stufe enthält. Die primitiven Symbole von PLΩ sind die Klammern [und], die logischen Symbole $\neg$, $\forall$, $\exists$, $\vee$, $\wedge$, $\Rightarrow$, $\Leftrightarrow$, eine abzählbare Liste von Variablen zu jedem Typ und eine Liste von getypten Konstantensymbolen. Die wohlgeformten Ausdrücke vom Typ α werden definiert durch:

1. Jedes Variablen- und jedes Konstantensymbol vom Typ α ist ein Term vom Typ α.
2. Sind $\mathbf{A}_o$ und $\mathbf{B}_o$ Terme vom Typ o und X_α eine Variable vom Typ α, so sind $\neg\mathbf{A}_o$, $\forall X_\alpha\mathbf{A}_o$, $\exists X_\alpha\mathbf{A}_o$, $\mathbf{A}_o\vee\mathbf{B}_o$, $\mathbf{A}_o\wedge\mathbf{B}_o$,,$\mathbf{A}_o\Rightarrow\mathbf{B}_o$ und $\mathbf{A}_o\Leftrightarrow\mathbf{B}_o$ Terme vom Typ o.
3. Ist $\mathbf{A}_{\alpha\beta}$ ein Term vom Typ $(\alpha\beta)$ und B_β ein Term vom Typ β, so ist $[\mathbf{A}_{\alpha\beta}\mathbf{B}_\beta]$ ein Term vom Typ α.

Offensichtlich ist es durch die Typisierung der Ausdrücke nicht mehr notwendig zwischen *Formeln* (Ausdrücke vom Typ o) und *Term*en (Typ≠o) oder zwischen *Prädikatssymbol*en (Typ=$(o\beta_1\beta_2\ldots\beta_n)$) und *Funktionssymbol*en (Typ=$(\alpha\beta_1\beta_2\ldots\beta_n)$, wobei $\alpha\neq o$) zu unterscheiden, da diese Information bereits aus dem Typ abgelesen werden kann. Wir werden deswegen im folgenden immer von Termen reden, wenn wir nichts über den Typ eines Ausdrucks aussagen wollen. Wir verwenden allerdings die Ausdrücke Prädikat, Funktion und Formel weiterhin in ihrer intuitiven Bedeutung, um Ausdrücke des jeweiligen Typs zu beschreiben. Wir schreiben den Typ eines Terms in den Index, außer wenn er irrelevant oder aus dem Kontext zu ersehen ist.

Die *Ordnung* der Typen ι und o ist 0, die des Typs $(\alpha\beta_1\beta_2\ldots\beta_n)$ ist das Maximum der Ordnungen von α und der β_i um 1 erhöht, die Ordnung eines Konstanten- oder Variablensymbols ist die seines Typs, die Ordnung eines Terms ist die maximale Ordnung der vorkommenden Variablen und Konstanten. Für $n\in\mathbb{N}$ bilden die Teilsysteme PL[2n] der Terme der Ordnung $\leq n$ in PLΩ und die Teilsysteme PL[2n-1] von PL[2n], wo in keinem Term über Variablen der Ordnung $\geq n$ quantifiziert wird, gerade die oben geforderten Systeme für eine Logik (2n)-ter beziehungsweise (2n-1)-ter Stufe.

1.3 Semantik von PLΩ

Für die Semantik von PLΩ muß man durch das Vorhandensein von Funktionen und Wahrheitswerten im System ein komplexeres *Universum* betrachten als dies in PL1 notwendig ist. Wie in der Semantik von PL1 führen wir zu den Basistypen o und ι das Universum $\mathbb{U}_o$:={T,F} der Wahrheitswerte und den nichtleeren *Individuenbereich* $\mathbb{U}_\iota$ ein. Zusätzlich gibt es *Funktionsuniversen* $\mathbb{U}_{\alpha\beta} = (\mathbb{U}_\beta\rightarrow\mathbb{U}_\alpha)$ für die Funktionstypen $(\alpha\beta)$, das heißt, $\mathbb{U}_{\alpha\beta}$ ist die Menge der Funktionen von $\mathbb{U}_\beta$ nach $\mathbb{U}_\alpha$. Die *Interpretationen* sind dann Abbildungen, die Konstanten-Symbolen Objekte im Universum des zugehörigen Typs zuordnen. Ein *Modell* für PLΩ ist ein Universum $\mathbb{U}$:={$\mathbb{U}_\alpha \mid \alpha$ ist Typ} zusammen mit einer Interpretation $\mathbb{I}$ nach $\mathbb{U}$. Wie in der Tarskisemantik für PL1 läßt sich jedem Term $\mathbf{A}_\alpha$ in PLΩ, der keine freien Variablen

enthält, ein *Wert* $\mathbb{I}(\mathbf{A}_\alpha) \in \mathbb{U}_\alpha$ unter der Interpretation $\mathbb{I}$ zuordnen. Insbesondere läßt sich jedem *Satz* $\mathbf{A}_o$ (also jeder Formel ohne freie Variablen) ein Wahrheitswert aus $\mathbb{U}_o$ zuordnen. Ein Satz $\mathbf{A}_o$ wird durch ein Modell $(\mathbb{U},\mathbb{I})$ *erfüllt*, falls $\mathbb{I}(\mathbf{A}_o) = T \in \mathbb{U}_o$ und *falsifiziert*, falls $\mathbb{I}(\mathbf{A}_o) = F \in \mathbb{U}_o$ .Ist $(\mathbb{U}, \mathbb{I})$ ein Modell für PLΩ, so nennt man eine Funktion φ, die allen Variablen X_α einen Wert in $\mathbb{U}_\alpha$ zuordnet, *Variablenbelegung*. Der *Wert* $\mathbb{I}_\varphi(\mathbf{A}_\alpha)$ eines Terms $\mathbf{A}_\alpha$ mit freien Variablen hängt dann von der Variablenbelegung φ ab, denn alle freien Variablen X_α erhalten die Werte $\varphi(X_\alpha)$.

Diese Modelle nennt man *Standardmodelle*, denn sie kommen der intuitiven Vorstellung von der Welt der Mathematik am nächsten. Kurt Gödel hat allerdings in seinem berühmten *Unvollständigkeitssatz* gezeigt, daß alle logischen Systeme, die die *Arithmetik* formalisieren können, *unvollständig* sein müssen bezüglich der Semantik der Standardmodelle. Die Arithmetik ist das System der natürlichen Zahlen, zusammen mit Addition und Multiplikation und läßt sich in PLΩ formalisieren (siehe unten). Deswegen kann es keinen korrekten und vollständigen Kalkül für PLΩ geben.

Wenn wir die Semantik der Standardmodelle verallgemeinern und für die Funktionsuniversen nur noch die Bedingung $\mathbb{U}_{\alpha\beta} \subseteq (\mathbb{U}_\beta \to \mathbb{U}_\alpha)$ fordern, erhalten wir die Semantik der *verallgemeinerten Modelle* oder *Henkinmodelle*. Dies liefert eine viel reichere Klasse von Modellen, deswegen gibt es nicht so viele allgemeingültigen Formeln (das sind die Formeln, die in allen Modellen gelten) wie in der Standardsemantik. Leon Henkin hat gezeigt, daß man in diesem Fall vollständige und korrekte Kalküle finden kann (siehe [Hen50]). Die Semantik der verallgemeinerten Modelle ist zwar nicht die intuitive Semantik der Mathematik, aber da jedes Standardmodell auch ein verallgemeinertes Modell ist, sind auch alle in der *verallgemeinerten Semantik* allgemeingültigen Sätze in der *Standardsemantik* allgemeingültig. Insbesondere sind auch alle in den oben erwähnten Kalkülen ableitbaren Sätze allgemeingültig in der Standardsemantik.

Nach dem Satz von Lindström ist allerdings die Prädikatenlogik erster Stufe in gewisser Weise die ausdrucksmächtigste Logik, die vollständige Kalküle erlaubt. Tatsächlich kann man (siehe [Ker91]) das System PLΩ und dessen verallgemeinerte Modelle so in die Prädikatenlogik erster Stufe und deren Tarskimodelle kodieren, daß die Folgerungsrelation erhalten bleibt. In diesem Sinn ist, wenn man verallgemeinerte Modelle betrachtet, die Typtheorie nur eine geschickte Kodierung von komplizierteren Objekten in PL1 und nicht ein wirklich ausdrucksmächtigeres System. Von einem philosophischen Standpunkt aus unterscheidet sich also der eingangs erwähnte mengentheoretische Ansatz wenig vom Ansatz der Typtheorie.

1.4 Existenzaxiome

Um sicherzustellen, daß in allgemeinen Modellen die Funktionsuniversen immer genügend Funktionen enthalten, benötigt man in PLΩ die sogenannten *Existenzaxiome* oder *Komprehensionsaxiome*

Für jeden Term $\mathbf{A}_\alpha$, in dem die Variable $F_{\alpha\beta}$ nicht frei vorkommt, und für jede Variable V_β soll folgendes gelten:

$$\exists F_{\alpha\beta}\ \forall V_\beta\ [[F_{\alpha\beta}V_\beta] = \mathbf{A}_\alpha].$$

Die Existenzaxiome besagen, daß es für jeden Term $\mathbf{A}_\alpha$ und jede Variable V_β eine Funktion $f \in \mathbb{U}_{\alpha\beta}$ gibt, so daß $f(u) = \mathbb{I}_\varphi(\mathbf{A}_\alpha)$, falls $\varphi(V_\beta)=u$. Man kann den Term $\mathbf{A}_\alpha$ als eine Zuordnungsvorschrift $V_\beta \rightarrow \mathbf{A}_\alpha$ ansehen, die einer Variablen V_β den Term $\mathbf{A}_\alpha(V_\beta)$ zuordnet, das heißt, jedem Term $\mathbf{B}_\beta$ wird durch $\mathbf{A}_\alpha$ der Term $\sigma(\mathbf{A}_\alpha)$ zugeordnet, wobei $\sigma=\{V_\beta \leftarrow \mathbf{B}_\beta\}$

Obwohl wir für die Existenzaxiome nun die (noch nicht eingeführte) Gleichheit benötigt haben, ist dies keine wirkliche Einschränkung, denn in PLΩ kann man das Gleichheitsprädikat über das *Leibnizsche Prinzip* „Zwei Dinge sind gleich, wenn sie bezüglich aller ihrer Eigenschaften übereinstimmen“ definieren. Sind $\mathbf{A}_\alpha$ und $\mathbf{B}_\alpha$ Terme des gleichen Typs α, so soll $[\mathbf{A}_\alpha=\mathbf{B}_\alpha]$ eine Abkürzung sein für den Term

$$[\forall P_{o\alpha}[P_{o\alpha}\mathbf{A}_\alpha \Rightarrow P_{o\alpha}\mathbf{B}_\alpha]].$$

Diese Formel besagt, daß für alle Prädikate (Eigenschaften) $P_{o\alpha}$ gilt: Ist P auf $\mathbf{A}_\alpha$ erfüllt, so auch auf $\mathbf{B}_\alpha$. Man beachte, daß es ausreicht die Implikation $P_{o\alpha}\mathbf{A}_\alpha \Rightarrow P_{o\alpha}\mathbf{B}_\alpha$ zu fordern, denn für ein Prädikat $P_{o\alpha}$ ist auch $\neg P_{o\alpha}$ ein Prädikat, und über alle solchen wird in der Formel oben quantifiziert. Als Alternative dazu, die Gleichheit aus Allquantor und Implikation zu definieren, kann man die Gleichheit als primitiv ansehen und die Gleichheitszeichen "$=_{o\alpha\alpha}$" als einzige logische Konstanten einführen, aus denen dann alle anderen definiert werden können (siehe [And86]).

1.5 λ-Kalkül

Alonzo Church hat vorgeschlagen, die Funktion, deren Existenz durch das obige Axiom garantiert wird, $[\lambda X_\beta\ \mathbf{A}_\alpha]$ zu nennen [Chu40]. Die syntaktische Notation $[\lambda X_\beta\ \mathbf{A}_\alpha]$ macht die Abhängigkeit des Terms $\mathbf{A}_\alpha$ von der Variablen X_β explizit. Der Term $[\lambda X_\beta\ \mathbf{A}_\alpha]$ steht also für die Funktion, die einem Argument $\mathbf{B}_\beta$ den Wert $\sigma(\mathbf{A}_\alpha)$ zuordnet, wobei σ die Substitution $\{X_\beta \leftarrow \mathbf{B}_\beta\}$ ist. Kommt die Variable X_β nicht frei in $\mathbf{A}_\alpha$ vor, so entspricht $[\lambda X_\beta \mathbf{A}_\alpha]$ gerade der konstanten Funktion, die auf alle Eingaben den Term $\mathbf{A}_\alpha$ liefert. Das Zeichen λ nennt man den *Abstraktionsoperator*, weil die Funktion $[\lambda X_\beta\ \mathbf{A}_\alpha]$ im Gegensatz zu der Zuordnungsvorschrift $\mathbf{A}_\alpha$ nicht von der Einsetzung für X_β abhängt.

Durch die λ-Abstraktion erhält man eine neue Art, Variablen zu binden (wie durch den Allquantor ∀ oder den Existenzquantor ∃, nur mit einer anderen Bedeutung). In der Typtheorie kann eine allquantifizierte Formel $\forall X_\alpha \mathbf{A}_o$ als eine Abkürzung der Formel $\Pi_{ooo\alpha}[\lambda X_\alpha \mathbf{A}_o]$ aufgefaßt werden. Das Prädikat $\Pi_{ooo\alpha}$ prüft, ob die Menge, auf die es angewendet wird, die Menge aller Objekte vom Typ α ist. Die Formel $\forall X_\alpha \mathbf{A}_o$ gilt tatsächlich genau dann, wenn das Prädikat $[\lambda X_\alpha \mathbf{A}_o]$ die konstante Funktion ist, die jedem Argument den Wahrheitswert T zuordnet. Der Existenzquantor läßt sich wie in der Prädikatenlogik aus Allquantor und Negation darstellen. Es ist also in der Typtheorie nicht nötig, außer der λ-Abstraktion noch andere Bindungsmechanismen für Variablen zu betrachten.

Logische Systeme, die die syntaktische Möglichkeit der λ-Abstraktion bieten, heißen *λ-Kalküle*. Churchs Formalisierung der Logik höherer Stufe ist ein *getypter λ-Kalkül* und ist auch als

einfache Typtheorie bekannt. Die Regeln für die Syntax lauten:

1. Jedes Variablen- und jedes Konstantensymbol vom Typ α ist ein Term vom Typ α.
2. Sind $\mathbf{A}_o$ und $\mathbf{B}_o$ Terme vom Typ o und X_α eine Variable vom Typ α, so sind $\neg\mathbf{A}_o$, $\forall X_\alpha \mathbf{A}_o$, $\exists X_\alpha \mathbf{A}_o$, $\mathbf{A}_o \vee \mathbf{B}_o$, $\mathbf{A}_o \wedge \mathbf{B}_o$, $\mathbf{A}_o \Rightarrow \mathbf{B}_o$ und $\mathbf{A}_o \Leftrightarrow \mathbf{B}_o$ Terme vom Typ o.
3. Ist $\mathbf{A}_{\alpha\beta}$ ein Term vom Typ $(\alpha\beta)$ und $\mathbf{B}_\beta$ ein Term vom Typ β, so ist $[\mathbf{A}_{\alpha\beta}\mathbf{B}_\beta]$ ein Term vom Typ α.
4. Ist $\mathbf{A}_\alpha$ ein Term vom Typ α,und X_β eine Variable vom Typ β, so ist $[\lambda X_\beta\ \mathbf{A}_\alpha]$ ein Term vom Typ $(\alpha\beta)$.

Die einfache Typtheorie ist also eine Erweiterung der Prädikatenlogik erster Stufe um Typen und λ-Abstraktionen. Insbesondere sehen alle Formeln und Terme erster Stufe in der Typtheorie genauso aus wie in PL1, wenn man davon absieht, daß $f(t^1, ..., t^n)$ jetzt als $[f\ t^1\ ...\ t^n]$ geschrieben wird.

Die Existenzaxiome in Churchs Schreibweise sehen nun wie folgt aus:

$$\forall V_\beta\ [[[\lambda V\ \mathbf{A}_\alpha]V] = \mathbf{A}_\alpha].$$

Ist in einem Term $\mathbf{B}_\beta$ keine Variable frei, die in $\mathbf{A}_\alpha$ gebunden ist, so können wir $\mathbf{B}_\beta$ für alle freien Vorkommen der Variablen V_β einsetzen. Dabei ist zu beachten, daß die Variablen im ersten Vorkommen von $\mathbf{A}_\alpha$ durch die λ-Abstraktion gebunden sind. Deswegen werden nur die Vorkommen V_β im zweiten Vorkommen von $\mathbf{A}_\alpha$ ersetzt. Wir erhalten also die Gleichung

$$[[\lambda V_\beta\ \mathbf{A}_\alpha]\mathbf{B}_\beta] = \sigma(\mathbf{A}_\alpha), \qquad (\lambda\text{-Axiom})$$

wobei σ die Substitution ist, die V_β durch $\mathbf{B}_\beta$ ersetzt. Diese Gleichung wird oft das *λ-Axiom* genannt, weil es das Verhalten von Funktionen beschreibt, die durch λ-Abstraktion gebildet wurden.

Durch die Verwendung von λ-Kalkülen ist es also möglich, die Existenzaxiome durch die Theorie der *λ-Gleichheit* zu ersetzen, die – wie wir sehen werden – besser mechanisiert werden kann.

Wie in PL1 ist auch in der Typtheorie der Name von gebundenen Variablen irrelevant. Das gilt auch für die Bindung von Variablen mit dem Abstraktionsoperator λ. Die Terme $[\lambda X_\beta\ \mathbf{A}_\alpha]$ und $[\lambda Y_\beta\ \rho(\mathbf{A}_\alpha)]$ sind also als gleich anzusehen, wenn keine Y_β frei in $\mathbf{A}_\alpha$ vorkommen und ρ die Substitution ist, die alle freien Variablen X_β in $\mathbf{A}_\alpha$ zu Y_β umbenennt. Terme, die durch eine Reihe von Umbenennungen der gebundenen Variablen auseinander hervorgehen, nennt man *alphabetische Varianten*. Um diese Tatsache für den Kalkül zu repräsentieren, führt man das folgende Axiom über die *alphabetische Umbenennung* oder kurz *α-Axiom* ein.

$$[\lambda X_\beta\ \mathbf{A}_\alpha] = [\lambda Y_\beta\ \sigma(\mathbf{A}_\alpha)]. \qquad (\alpha\text{-Axiom})$$

In der Mathematik geht man davon aus, daß zwei Funktionen genau dann gleich sind, wenn sie auf allen möglichen Argumenten übereinstimmen. Dieses Phänomen wird die Extensionalität der Gleichheit genannt. Es kommt also nicht auf die *Form* einer Funktion an, sondern nur auf ihre *Wirkung* auf ihrem Definitionsbereich. Im Gegensatz zur Mathematik kann es zum Beispiel

in der Informatik sinnvoll sein, *nichtextensionale* Gleichheit zu betrachten. Wenn man beispielsweise Programme beschreiben will, sollte man zwischen verschiedenen korrekten Sortierprogrammen unterscheiden können, obwohl sie als Funktionen gesehen alle die gleiche Wirkung haben (nämlich zu sortieren). Das folgende Schema formalisiert gerade die Extensionalität der Gleichheit und heißt deswegen *Extensionalitäts*a*xiom*

$$\forall F_{\alpha\beta}\ \forall G_{\alpha\beta}\ [\forall X_{\beta}\ [FX = GX]] \Rightarrow [F = G].$$

In vielen Kalkülen ist es ausreichend, statt dieses komplizierteren Axioms eine schwächere Form, das sogenannte *η-Axiom*

$$\forall F_{\alpha\beta}\ [\lambda X_{\beta}[FX]] = F_{\alpha\beta} \qquad (\eta\text{-Axiom})$$

zu benutzen. Die Extensionalität ist so natürlich und zentral, daß sie meist in irgendeiner Form mit in die Axiome der Typtheorie aufgenommen wird.

Die Standardmodelle für die Typtheorie sind ähnlich definiert wie die für PLΩ, allerdings müssen wir die Bildung von *Werten* auf λ-Abstraktionen erweitern. Wir definieren $\mathbb{I}_{\varphi}([\lambda X_{\beta \supseteq}\mathbf{A}_{\alpha}])$ als diejenige Funktion in $\mathbb{U}_{\alpha\beta}$, die einem Objekt $g \in \mathbb{U}_{\beta}$ den Wert $\mathbb{I}_{\psi}(\mathbf{A}_{\alpha}) \in \mathbb{U}_{\alpha}$ zuordnet. Dabei ist ψ die Variablenbelegung, die X_{β} auf $g \in \mathbb{U}_{\beta}$ abbildet und sich sonst wie φ verhält. Auf diese Weise kann man jedem wohlgetypten Term einen Wert zuordnen.

Definieren wir die verallgemeinerten Modelle der Typtheorie wie oben, so erhalten wir aus den Existenzaxiomen die zusätzliche Forderung, daß es für alle wohlgetypten λ-Terme $\mathbf{A}_{\alpha}$ einen Wert $\mathbb{I}(\mathbf{A}_{\alpha}) \in \mathbb{U}_{\alpha}$ geben muß. Diese Forderung ist aber nicht so stark, daß sie den Unterschied zwischen Standard- und verallgemeinerten Modellen aufhebt, denn sie stellt nur die Existenz der (relativ kleinen) Klasse der λ-definierbaren Funktionen in den Funktionsuniversen sicher.

Die Semantik für die Typtheorie ist so gebaut, daß sowohl in der Standard- als auch in der verallgemeinerten Semantik die α-, λ- und η-Axiome allgemeingültig sind.

1.6 Normalformen

In diesem Abschnitt greifen wir auf Methoden und Ergebnisse aus dem Kapitel über Termersetzungssysteme zurück. Wir sind dabei besonders an der Mechanisierung der λη-Gleichheit interessiert. Für die automatische Behandlung der Gleichheit hat es sich als sinnvoll herausgestellt, die Anwendung von Gleichungen (beim Ersetzen von Untertermen) wenn möglich nur in eine Richtung zuzulassen (Richten von Gleichungen zu Regeln). Unter bestimmten Bedingungen kann man ein Gleichungssystem so richten, daß auf jeden Term die Gleichungen nur endlich oft angewandt werden können (*Termination*). Die Terme werden also durch die Anwendung von Gleichungen in irgendeinem Sinne kleiner, deswegen heißen solche Gleichungsanwendungen *Reduktionsschritte* und die in Gegenrichtung *Expansionsschritte*. Die Terme, auf die keine Reduktionsschritte mehr angewandt werden können, heißen *Normalformen*. Gleichungstheorien, in denen die Normalformen für Terme eindeutig sind, lassen sich besonders gut automatisch behandeln, denn in diesem Fall sind zwei Terme genau dann gleich in der Theorie, wenn ihre (eindeutigen) Normalformen syntaktisch gleich sind.

Richtet man das λ-Axiom und das η-Axiom zu den Regeln

1. $[[\lambda V_\beta\ \mathbf{A}_\alpha]\mathbf{B}_\beta] \longrightarrow_\lambda \{V_\beta \rightarrow \mathbf{B}_\beta\}\mathbf{A}_\alpha$, falls keine freie Variable von $\mathbf{B}_\beta$ in $\mathbf{A}_\alpha$ gebunden auftritt, (λ-Reduktion)

2. $[\lambda X_\beta[\mathbf{A}_{\alpha\beta}X_\beta]] \longrightarrow_\eta \mathbf{A}_{\alpha\beta}$, falls X_β nicht in $\mathbf{A}_{\alpha\beta}$ vorkommt, (η-Reduktion)

so erhält man nach dem berühmten *Church-Rosser-Theorem* ein kanonisches Reduktionssystem. Das heißt, jede Kette von *λη-Reduktionen* eines Terms $\mathbf{A}_\alpha$ terminiert und überführt $\mathbf{A}_\alpha$ in die eindeutige *λη-Normalform*. Zwei Formeln in der Typtheorie sind also genau dann gleich, wenn ihre λη-Normalformen gleich sind. Eine Formel **A** in λη-Normalform hat die folgende allgemeine Form:

$$\mathbf{A} = [\lambda X^1 \ldots X^n\ [\mathbf{K}\ \mathbf{B}^1 \ldots \mathbf{B}^k]],$$

wobei **K** eine Konstante oder Variable ist und die Subterme $\mathbf{B}^i$ auch in λη-Normalform sind. Der Teil $\lambda X^1 \ldots X^n$ in **A** heißt der *Binder*, der Teil $[\mathbf{K}\ \mathbf{B}^1 \ldots \mathbf{B}^k]$ der *Rumpf* und **K** das *Kopfsymbol* von **A**. Ist **K** eine freie Variable, so heißt **A** *flexibel*, sonst *starr*. **A** wird *j-Projektionsterm* genannt, falls **K** die gebundene Variable X^j ist. Für das Rechnen in der Typtheorie ist oft auch die sogenannte *η-Langform* eines Terms wichtig. Für diese Normalform wird ein Term in λ-Normalform so lange η-expandiert, bis der Rumpf einen Basistyp hat. So ist zum Beispiel die η-Langform von $=_{o\alpha\alpha}$ gerade $[\lambda X_\alpha Y_\alpha\ [=XY]]$. Diese Normalform hat den Vorteil, daß man immer am Binder sehen kann, wieviel Argumente eine Funktion noch erwartet. Außerdem haben wir in der η-Langform im Gegensatz zu der λη-Normalform das Phänomen, daß λ-Reduktionen Terme in η-Langform in ebensolche überführt. Diese Beobachtung erlaubt es, die η-Gleichheit implizit behandeln, indem wir uns bei der Betrachtung der Typtheorie auf die Teilsprache der Terme in η-Langform beschränken (für eine ausführliche Diskussion siehe [Hue76] und [GS89]).

Leider ist es nicht möglich auch das α-Axiom zu richten, denn jede Umbenennung ρ der gebundenen Variablen kann durch die inverse Umbenennung ρ^{-1} wieder rückgängig gemacht werden. Für welche Richtung des α-Axioms wir uns auch entscheiden, es wird immer unendlich lange Ketten von Anwendungen dieses Axioms geben. Deswegen wird die Theorie der α-Gleichheit meist implizit behandelt, indem man alphabetische Varianten als syntaktisch gleich ansieht. Allerdings kann es bei der Anwendung einer Substitution $\sigma=\{\ldots, X \leftarrow \mathbf{B}, \ldots\}$ auf einen Term $\mathbf{A}_\alpha$ passieren, daß Variablen, die in **B** frei vorkommen, in $\sigma(\mathbf{A}_\alpha)$ gebunden werden, wir sagen, sie werden *gefangen*. Da dieses Phänomen zu unkorrekten Schlüssen führt, müssen bei der λ-Reduktion die gebundenen Variablen systematisch so umbenannt werden, daß keine freien Variablen gefangen werden. Es gibt sogar Formalisierungen der Typtheorie, die dieses Vorgehen explizit machen und ganz auf Namen für gebundene Variablen verzichten (siehe [dBr72]).

Wir betrachten im folgenden die Typtheorie immer unter der αλη-Gleichheit.

1.7 Beispiele

Wir wollen die Ausdrucksstärke der Typtheorie am Beispiel der *einfachen Mengentheorie* zei-

gen, indem wir Mengen als Prädikate darstellen. Sei $M_{o\alpha}$ eine Menge, dann kann man die Aussage „j_α ist ein Element von $M_{o\alpha}$“ gerade durch die Formel $[M_{o\alpha} j_\alpha]$ darstellen. Das Element-Prädikat $\in$ erwartet zwei Argumente, ein Element (Typ α) und eine Menge (Typ($o\alpha$)), es hat also den Typ $o(o\alpha)\alpha$. Wir können es daher als die Formel

$$\in_{o(o\alpha)\alpha} := [\lambda J_\alpha \, \lambda M_{o\alpha} \, [MJ]]$$

definieren, wobei wir uns das Prädikat $\in_{o(o\alpha)\alpha}$ als eine Funktion vorstellen, die als Eingabe ein Objekt a_α und eine Menge $N_{o\alpha}$ bekommt und als Ausgabe $[N_{o\alpha} a_\alpha]$ liefert. Die Berechnung erledigt dabei die λ-Reduktion, denn $[\in_{o(o\alpha)\alpha} a_\alpha N_{o\alpha}] \longrightarrow_\lambda [N_{o\alpha} a_\alpha]$. Ähnlich kann man die Teilmengen-Beziehung definieren:

$$\subseteq_{o(o\alpha)(o\alpha)} := [\lambda M_{o\alpha} \, \lambda N_{o\alpha} \, [\forall F_\alpha [MF \Rightarrow NF]]].$$

Wie oben ist das Teilmengenprädikat $\subseteq_{o(o\alpha)(o\alpha)}$ eine Funktion, die bei Eingabe von zwei konkreten Mengen $m_{o\alpha}$ und $n_{o\alpha}$ prüft, ob $\forall F_\alpha [m_{o\alpha} F_\alpha \Rightarrow n_{o\alpha} F_\alpha]$ gilt, das heißt, ob jedes Element von $m_{o\alpha}$ auch eines von $n_{o\alpha}$ ist. Die Operatoren für die Vereinigung und den Schnitt von Mengen lassen sich durch die Konjunktion und Disjunktion definieren:

$$\cap_{o\alpha(o\alpha)(o\alpha)} := [\lambda M_{o\alpha} \, \lambda N_{o\alpha} \, [\lambda F_\alpha [MF \wedge NF]]],$$
$$\cup_{o\alpha(o\alpha)(o\alpha)} := [\lambda M_{o\alpha} \, \lambda N_{o\alpha} \, [\lambda F_\alpha [MF \vee NF]]].$$

Wir sehen, daß es eine Stärke der Typtheorie ist, wichtige mathematische Funktionen und Konzepte in Form von λ-Ausdrücken darzustellen. Mit Hilfe von solchen Abkürzungen können mathematische Sachverhalte in der Typtheorie direkt und natürlich formalisiert werden.

Wir wollen als weiteres Beispiel die *natürlichen Zahlen* über die *Peano-Axiome* formalisieren. Dazu verwenden wir das Prädikat $n_{o\iota}$ für die Menge $\mathbb{N}$ der natürlichen Zahlen, die Konstante 0_ι für die Zahl 0 und die Funktionskonstante $s_{\iota\iota}$ für die *Nachfolgerfunktion*. Die Axiome lauten dann

1. $n_{o\iota} 0_\iota$ (0 ist eine natürliche Zahl),
2. $\forall X_\iota \, [nX \Rightarrow [n[sX]]]$,
   (der Nachfolger jeder natürlichen Zahl ist eine natürliche Zahl),
3. $\neg \exists X_\iota \, n \, X \wedge [sX = 0]$ (0 hat keinen Vorgänger),
4. $\forall X_\iota \, \forall Y_\iota \, [sX = sY] \Rightarrow [X = Y]$ (die Nachfolgerfunktion ist injektiv)
5. $\forall P_{o\iota} \, [P0 \wedge [\forall X_\iota \, [PX \Rightarrow P[sX]]] \Rightarrow [\forall Y_\iota \, [nY \Rightarrow [PY]]]$
   (Induktionsaxiom: Alle Eigenschaften P, die für 0 gelten, und die mit jeder natürlichen Zahl auch für ihren Nachfolger gelten, gelten für alle natürlichen Zahlen.)

Auch so komplizierte Aussagen wie den Satz von Cantor über die Überabzählbarkeit der Menge $\mathbb{F}(\mathbb{N})$ der Folgen mit Gliedern in $\mathbb{N}$ kann man relativ natürlich formalisieren. Er besagt, daß es keine surjektive Abbildung von $\mathbb{N}$ in die Menge $\mathbb{F}(\mathbb{N})$ gibt. Die Menge der Folgen ist aber bekanntlich die Menge der Abbildungen von $\mathbb{N}$ auf sich, also können wir den Satz von Cantor folgendermaßen in der Typtheorie formalisieren:

$$\neg \exists F_{\iota\iota\iota} \, [\forall G_{\iota\iota} \, [\forall K_\iota \, [nK \Rightarrow nGK]] \Rightarrow \exists \, J_\iota \, [nJ \wedge [FJ = G]]].$$

Die Abbildung F, deren Nichtexistenz postuliert wird, bildet natürliche Zahlen (Typ ι) auf Funktionen (Typ $(\iota\iota)$) ab, sie muß also den Typ $(\iota(\iota\iota))$ haben. Die Teilformel ab dem ersten Allquantor sagt gerade aus, daß die Abbildung $F_{\iota(\iota\iota)}$ surjektiv ist. Sie besagt nämlich, daß es für jede Funktion $G_{\iota\iota}$, die natürliche Zahlen in natürliche Zahlen überführt, eine natürliche Zahl J_ι gibt, deren Bild unter $F_{\iota(\iota\iota)}$ gerade diese Funktion $G_{\iota\iota}$ ist.

An dieser Stelle können wir nun auf einen interessanten Unterschied zwischen Standardmodellen und verallgemeinerten Modellen hinweisen. Eine Konsequenz aus der Existenz von *vollständigen* Kalkülen für die Typtheorie ist die Gültigkeit eines *Abzählbarkeitssatzes*, der besagt, daß jede erfüllbare Menge Φ von Formeln ein Modell hat, mit nur abzählbar vielen Elementen. Cantors Satz sagt aber gerade aus, daß $\mathbb{F}(\mathbb{N})$ überabzählbar ist. Das sogenannte *Skolemsche Paradoxon* besteht darin, daß es auch für diesen Satz (und damit für die Menge der Folgen) ein *abzählbares*, verallgemeinertes Modell gibt. Weil Cantors Satz allgemeingültig ist – wir werden ihn unten aus den Peano-Axiomen beweisen – muß er in allen Modellen gelten. Skolems Paradoxon löst sich auf, wenn wir die genaue Form unserer Formalisierung des Satzes betrachten. Die obige Formel sagt nämlich nur in *Standardmodellen* etwas über die Kardinalität der Menge $\mathbb{F}(\mathbb{N})$ aus. Die abzählbaren, verallgemeinerten Modelle für Cantors Satz können deswegen keine Standardmodelle sein. In den *Nichtstandardmodellen* sagt der Satz nur aus, daß das Funktionsuniversum $\mathbb{U}_{\iota(\iota\iota)} \subseteq (\mathbb{U}_\iota \to \mathbb{U}_{\iota\iota})$ keine surjektiven Funktionen enthält.

1.8 Varianten und Erweiterungen der einfachen Typtheorie

Der getypte λ-Kalkül, den wir bisher beschrieben haben, hat einen relativ einfachen Typmechanismus. Es hat jedoch schon bald Erweiterungen dieses Kalküls für spezielle Zwecke gegeben. So kann man zum Beispiel die Menge der Basistypen vergrößern und dadurch das Universum in Klassen einteilen. Ferner werden ordnungssortierte Typsysteme für die Deduktion untersucht, um dadurch, wie bei den Systemen erster Stufe, einen Teil der Mengentheorie in den Kalkül abzusenken und so zu effizienteren Beweissystemen zu kommen.

Die für die Deduktionssysteme vielleicht wichtigsten Erweiterungen sind die Systeme abhängiger Typen, die die Darstellung der eigenen Beweistheorie in sich selbst erlauben. Dies wird durch die Idee ermöglicht, die zu beweisenden Formeln in die Typen zu kodieren. Dieser Kodierungstrick liefert einen Isomorphismus zwischen einer speziellen Klasse *abhängiger Typen* und den Formeln der Logik, er wird deswegen als „*Propositionen-als-Typen-Isomorphismus*“ oder nach den Erfindern „*Curry-Howard-Isomorphismus*“ bezeichnet. Wir wollen hier nur die Ideen vorstellen und uns nicht mit Formalismen beschäftigen.

Man betrachtet also ein Typsystem, in dem Typen wieder von Formeln und Termen abhängen können. So kann man in einem solchem System den Typ $Sohn\{Pip\}$ der Söhne von Pippin dem Kurzen (Pip) betrachten, und damit beispielsweise ausdrücken, daß Karl der Große ($KARL$) ein Element der Menge der Söhne von Pippin dem Kurzen ist ($KARL$: $Sohn\{Pip\}$).

Das „Propositionen-als-Typen“-Prinzip besteht nun darin, Typen der Form $pf\{\mathbf{F}_o\}$ zu betrachten, und sie als Menge der Beweise (pf steht für proof) für $\mathbf{F}_o$ aufzufassen. Eine Formel $\mathbf{F}_o$ ist also genau dann allgemeingültig, wenn der Typ $pf\{\mathbf{F}_o\}$ nicht leer ist. Wir sagen, der Typ

$pf\{\mathbf{F}_o\}$ ist *bewohnt*. Der Beweisprozeß ist dann die Konstruktion eines Terms **P** (dem sogenannten Zeugen) vom Typ $pf\{\mathbf{F}_o\}$. Jeder Zeuge für die Bewohntheit von $pf\{\mathbf{F}_o\}$ repräsentiert also einen Beweis für $\mathbf{F}_o$. Zur Konstruktion von **P** verwendet man typischerweise Regeln des natürlichen Schließens, wie Gentzens Kalküle **NJ**, **NK** oder Varianten davon. Diese Kalküle sind für PL1 in Kapitel II.3 erklärt, wir werden sie im folgenden in den Formalismus der abhängigen Typen kodieren und als Beispiel für eine Beweistheorie benutzen. Für jedes Axiom und jede Regel wählt man eine Konstante. Das Axiom des "tertium non datur" stellt man dann als eine Konstante

$$TND_{\mathbf{A}} : pf\{\mathbf{A}_o \vee \neg \mathbf{A}_o\}$$

dar. Der Wert des Terms $TND_{\mathbf{A}}$ ist also ein Element der Beweise für die Formel $\mathbf{A}_o \vee \neg \mathbf{A}_o$ und repräsentiert den Beweis, der aus dem einmaligen Zitieren des Axioms besteht. Für Schlußregeln ist der Typ der zugehörigen Konstante ein Funktionstyp, in den dann die Wirkung der Regel kodiert wird. So verbindet zum Beispiel die Regel der "Und-Einführung" einen Beweis für die Formel $\mathbf{A}_o$ mit einem Beweis für die Formel $\mathbf{B}_o$ zu einem Beweis für $\mathbf{A}_o \wedge \mathbf{B}_o$; deswegen können wir diese Regel in eine Konstante

$$UE_{\mathbf{AB}} : (pf\{\mathbf{A}_o \wedge \mathbf{B}_o\}\ pf\{\mathbf{A}_o\}\ pf\{\mathbf{B}_o\})$$

kodieren. Dabei besagt der Funktionstyp $(pf\{\mathbf{A}_o \wedge \mathbf{B}_o\}\ pf\{\mathbf{A}_o\}\ pf\{\mathbf{B}_o\})$, daß $UE_{\mathbf{AB}}$ eine Funktion ist, die einem Beweis für die Formel $\mathbf{A}_o$ (Typ $pf\{\mathbf{A}_o\}$) und einem Beweis für die Formel $\mathbf{B}_o$ (Typ $pf\{\mathbf{B}_o\}$) einen Beweis für $\mathbf{A}_o \wedge \mathbf{B}_o$ (Typ $pf\{\mathbf{A}_o \wedge \mathbf{B}_o\}$) zuordnet. Sind beispielsweise $\mathbf{P} : pf\{\mathbf{A}_o\}$ und $\mathbf{Q} : pf\{\mathbf{B}_o\}$ Beweise für $\mathbf{A}_o$ und $\mathbf{B}_o$, so ist der Term $[UE_{\mathbf{AB}}\mathbf{PQ}] : pf\{\mathbf{A}_o \wedge \mathbf{B}_o\}$ ein Beweis für $\mathbf{A}_o \wedge \mathbf{B}_o$.

Ein Nachteil dieser Kodierung besteht darin, daß man für jede Formel $\mathbf{A}_o$ ein Axiom $TND_{\mathbf{A}}$ und zu jedem Paar von Formeln $\mathbf{A}_o$, $\mathbf{B}_o$ eine Regel $UE_{\mathbf{AB}}$ hat. Eine solche Fülle von Regeln ist aber in einem praktischen System nicht handhabbar. Eine Abhilfe ist daher, die λ-Abstraktion und λ-Reduktion im Typ zu erlauben und die Axiome und Regeln so von den konkreten Formeln zu abstrahieren. Das Axiomenschema des *tertium non datur* kann damit als eine Konstante

$$TND : (\lambda X_o\ pf\{X_o \vee \neg X_o\})$$

dargestellt werden. Diese Konstante wird nun durch die Anwendung auf eine konkrete Formel $\mathbf{A}_o$ durch λ-Reduktion instanziiert. Der Term $[TND\ \mathbf{A}_o]$ hat nach λ-Reduktion im Typ den gewünschten Typ $pf\{\mathbf{A}_o \vee \neg \mathbf{A}_o\}$. Das Regelschema der *Und-Einführung* können wir nun mit der Abstraktion im Typ in eine Konstante

$$UE : \lambda X_o\, \lambda Y_o\, (pf\{X_o \wedge Y_o\}\ pf\{X_o\}\ pf\{Y_o\})$$

kodieren und erhalten damit $[UE\ \mathbf{A}_o\ \mathbf{B}_o] \longrightarrow_\lambda UE_{\mathbf{AB}}$.

Wir wollen einen etwas komplizierteren Beweis als Beispiel betrachten, um die Möglichkeiten der Kodierung von Beweisen darzustellen. Dazu brauchen wir die *Und-Beseitigungs*-Regeln *UBR* (rechts), *UBL* (links)

$$UBR : \lambda X_o\, \lambda Y_o\, (pf\{Y_o\}\ pf\{X_o \wedge Y_o\}),$$
$$UBL : \lambda X_o\, \lambda Y_o\, (pf\{X_o\}\ pf\{X_o \wedge Y_o\})$$

und die *Folgt-Einführungs*-Regel *FE*. Diese besagt gerade, daß „Ist die Formel $\mathbf{B}_o$ unter der Annahme $\mathbf{A}_o$ beweisbar, so ist auch die Formel $\mathbf{A}_o \Rightarrow \mathbf{B}_o$ beweisbar“. Wir können den Beweis von $\mathbf{B}_o$ unter der Annahme $\mathbf{A}_o$ als eine Funktion ansehen, die bei Eingabe eines beliebigen Beweises für $\mathbf{A}_o$ einen Beweis für $\mathbf{B}_o$ liefert (durch Aneinanderhängen der Beweise). Das heißt, die Regel *FE* transformiert eine Funktion vom Typ $(pf\{\mathbf{B}_o\}pf\{\mathbf{A}_o\})$ in ein Element von $pf\{\mathbf{A}_o \Rightarrow \mathbf{B}_o\}$. Diese Überlegungen ergeben die folgende Kodierung

$$FE : \lambda X_o\, \lambda Y_o\, (pf\{X_o \Rightarrow Y_o\}\ (pf\{X_o\}\ pf\{Y_o\})).$$

Damit läßt sich dann der Beweis für die Kommutativität des Junktors $\wedge$ in Gentzens Baumdarstellung

$$\dfrac{\dfrac{\dfrac{[\mathbf{A}_o \wedge \mathbf{B}_o]}{\mathbf{B}_o}\,UBR \qquad \dfrac{[\mathbf{A}_o \wedge \mathbf{B}_o]}{\mathbf{A}_o}\,UBL}{\mathbf{B}_o \wedge \mathbf{A}_o}\,UE}{\mathbf{A}_o \wedge \mathbf{B}_o \Rightarrow \mathbf{B}_o \wedge \mathbf{A}_o}\,FE$$

in den folgenden Term in der Typtheorie mit abhängigen Typen kodieren:

$$\mathbf{P} := FE\ [\mathbf{A}_o \wedge \mathbf{B}_o][\mathbf{B}_o \wedge \mathbf{A}_o]\ \mathbf{Q}.$$

Ist $\mathbf{Q}$ vom Typ $(pf\{\mathbf{B}_o \wedge \mathbf{A}_o\}\ pf\{\mathbf{A}_o \wedge \mathbf{B}_o\})$, so ist $\mathbf{P}$ wohlgetypt, vom gewünschten Typ $pf\{\mathbf{A}_o \wedge \mathbf{B}_o \Rightarrow \mathbf{B}_o \wedge \mathbf{A}_o\}$ und repräsentiert einen Beweis für die Formel $\mathbf{A}_o \wedge \mathbf{B}_o \Rightarrow \mathbf{B}_o \wedge \mathbf{A}_o$. Ist nun X eine Variable vom Typ $pf\{\mathbf{A}_o \wedge \mathbf{B}_o\}$, so ist

$$[UE\ \mathbf{B}_o\ \mathbf{A}_o\ [UBR\ \mathbf{A}_o\ \mathbf{B}_o\ X]\ [UBL\ \mathbf{A}_o\ \mathbf{B}_o\ X]]$$

vom Typ $pf\{\mathbf{B}_o \wedge \mathbf{A}_o\}$ und stellt eine Zuordnungsvorschrift dar, die aus jedem Beweis X für $\mathbf{A}_o \wedge \mathbf{B}_o$ durch Einsetzen einen Beweis für die Formel $\mathbf{B}_o \wedge \mathbf{A}_o$ macht. Durch λ-Abstraktion erhalten wir den Term

$$\mathbf{Q} := [\lambda X : pf\{\mathbf{A}_o \wedge \mathbf{B}_o\}\ [UE\ \mathbf{B}_o\ \mathbf{A}_o\ [UBR\ \mathbf{A}_o\ \mathbf{B}_o\ X]\ [UBL\ \mathbf{A}_o\ \mathbf{B}_o\ X]]]$$

vom Typ $(pf\{\mathbf{B}_o \wedge \mathbf{A}_o\}\ pf\{\mathbf{A}_o \wedge \mathbf{B}_o\})$. Die Kodierung des Beweises ist also komplett. Dabei wird die Baumstruktur des Gentzen-Beweises genau in die Baumstruktur des Terms $\mathbf{P}$ überführt. Durch die Kodierung in Terme sind Beweise in der Typtheorie mit abhängigen Typen Objekte der *Logik* und nicht mehr der *Meta-Logik* und stehen damit auch wieder deduktiven Methoden offen.

Beispiele für Systeme mit abhängigen Typen sind LCF [GMW79] und Martin-Löfs Typtheorie [Mar84].

2 Beweisverfahren in der Typtheorie

Die meisten automatischen Beweiser verwenden als zentrale Operation die *Unifikation*, deswegen stellen wir die Unifikation für die Typtheorie vor, bevor wir auf Beweisverfahren eingehen.

2.1 Unifikation in der Typtheorie

Da die α-, λ- und η-Axiome fest in die Typtheorie eingebaut sind, muß die Unifikation die Unifikation in der $\alpha\lambda\eta$-Theorie sein. Wir verwenden die Begriffe, Methoden und Ergebnisse aus dem Kapitel über Unifikationstheorie.

Um die speziellen Probleme der Unifikation in der Typtheorie darzustellen, betrachten wir als einfaches Beispiel das *Unifikationsproblem* $\langle [X_{\alpha\beta\alpha} a_\alpha b_\beta] = a_\alpha \rangle$, wobei $X_{\alpha\beta\alpha}$ eine Variable und a_α beziehungsweise b_β Konstanten sind. Die Lösung eines solchen besteht aus einer Substitution $\{X_{\alpha\beta\alpha} \leftarrow \mathbf{A}_{\alpha\beta\alpha}\}$, so daß $[\mathbf{A}_{\alpha\beta\alpha} a_\alpha b_\beta]$ in der $\alpha\lambda\eta$-Theorie gleich a_α ist. Die Substitutionen

$$\sigma_I := \{X_{\alpha\beta\alpha} \leftarrow [\lambda Z_\alpha\ \lambda Y_\beta\ a_\alpha]\}$$

und

$$\sigma_P := \{X_{\alpha\beta\alpha} \leftarrow [\lambda Z_\alpha\ \lambda Y_\beta\ Y_\beta]\}$$

sind allgemeinste Unifikatoren. Diese Substitutionen stellen zwei grundsätzlich verschiedene Möglichkeiten dar, $[Xab]$ in der $\alpha\lambda\eta$-Theorie zu a_α zu machen. Der Unifikator σ_I *imitiert* den Term a_α, indem er die Variable $X_{\alpha\beta\alpha}$ an einen Term $\mathbf{A}_{\alpha\beta\alpha} = [\lambda Z_\alpha\ \lambda Y_\beta\ a_\alpha]$ bindet, dessen *Kopfsymbol* gerade die Konstante a_α ist. Offensichtlich ist das Kopfsymbol von $\sigma_I([Xab]) \longrightarrow_\lambda a_\alpha$ nach der λ-Reduktion immer noch die Konstante a_α. Der Unifikator σ_P *projiziert* den Subterm a_α in $X_{\alpha\beta\alpha} a_\alpha b_\beta$ eine Ebene nach oben, das heißt, nach der λ-Reduktion ist dieser Subterm das *Kopfsymbol* von $\sigma_P([Xab])$.

Das Beispiel oben zeigt auch, daß die Unifikation in der Typtheorie im Gegensatz zur Robinson-Unifikation nicht unitär ist. Schon die Unifikation in der Logik zweiter Stufe ist *unentscheidbar* (siehe [Gol81]) und *vom Typ* 0 (siehe [Hue75]). Es gibt also keinen Algorithmus, der entscheiden kann, ob ein beliebig gegebenes Termpaar unifizierbar ist oder nicht. Weiterhin gibt es Termpaare, für die es keine Menge minimaler Unifikatoren gibt.

Wir stellen im folgenden Gérard Huets Unifikationsalgorithmus für die Typtheorie als System von *Transformationsregeln* dar. Eine *Termpaarmenge*

$$\Gamma := \{\langle \mathbf{A}^1 = \mathbf{B}^1, \ldots, \mathbf{A}^n = \mathbf{B}^n \rangle\}$$

heißt *Unifikationsproblem*, wenn die $\mathbf{A}^i$ und $\mathbf{B}^i$ jeweils den gleichen Typ haben. Transformationsregeln überführen Unifikationsprobleme in Unifikationsprobleme. Aus einem System von Transformationsregeln erhält man einen *Unifikationsalgorithmus*, wenn man den Suchraum, der durch dieses Regelsystem aufgespannt wird, systematisch durchsucht. Dabei werden – ausgehend von einem *initialen* Unifikationsproblem Γ – die Transformationsregeln so lange angewandt bis keine Regel mehr anwendbar ist. Der Algorithmus ist *korrekt*, falls jeder Unifikator für ein transformiertes Unifikationsproblem Δ auch ein Unifikator für das ursprüngliche Problem Γ war. Die Unifikation ist also ein Prozeß des Vereinfachens von Unifikationsproblemen.

Ein Unifikationsproblem heißt *gelöst*, wenn alle Termpaare von der Form $X^i = \mathbf{A}^i$ sind, so daß die Variable X^i nicht in $\mathbf{A}^i$ vorkommt. Ist Γ ein Unifikationsproblem in gelöster Form, dann ist

$\sigma_\Gamma := \{X^i \leftarrow \mathbf{A}^i\}$ ein allgemeinster Unifikator für Γ.

Die folgenden Transformationsregeln geben Huets Unifikationsalgorithmus für die Typtheorie mit η-Gleichheit wieder. Dieser Algorithmus wird ausführlich in [GS89] diskutiert.

(**T**) $\langle \mathbf{A}_\alpha = \mathbf{A}_\alpha \rangle \;\&\; \Gamma \longrightarrow \Gamma.$ (Tautologie)

(**D**) $\langle [\lambda X^1 \ldots X^k\, [\mathbf{K}\; \mathbf{U}^1 \ldots \mathbf{U}^n]] = [\lambda X^1 \ldots X^k\, [\mathbf{K}\; \mathbf{V}^1 \ldots \mathbf{V}^n]] \rangle \;\&\; \Gamma \longrightarrow$

$\langle [\lambda X^1 \ldots X^k\, \mathbf{U}^1] = [\lambda X^1 \ldots X^k\, \mathbf{V}^1],\ \ldots,\ [\lambda X^1 \ldots X^k\, \mathbf{U}^n] = [\lambda X^1 \ldots X^k\, \mathbf{V}^n] \rangle \;\&\; \Gamma,$

falls **K** eine Konstante oder Variable ist. (Dekomposition)

(**B**) $\langle [\lambda X^1 \ldots X^k\, [Y\, X^j \ldots X^k]] = [\lambda X^1 \ldots X^k\, \mathbf{A}_\alpha] \rangle \;\&\; \Gamma \longrightarrow \langle Y = [\lambda X^j \ldots X^k\, \mathbf{A}_\alpha] \rangle \;\&\; \sigma(\Gamma),$

falls X^i und Y Variablen der Typen β_i und $(\alpha\beta_j \ldots \beta_k)$ sind und Y eine Variable, die noch frei in Γ, aber nicht frei in $\mathbf{A}_\alpha$ vorkommt und $\sigma := \{Y \leftarrow [\lambda X^j \ldots X^k\, \mathbf{A}_\alpha]\}$. (Bindung)

(**O**) $\langle \mathbf{A}_\alpha = \mathbf{B}_\alpha \rangle \;\&\; \Gamma \longrightarrow \langle \mathbf{B}_\alpha = \mathbf{A}_\alpha \rangle \;\&\; \Gamma$, falls $\mathbf{B}_\alpha$ flexibel und $\mathbf{A}_\alpha$ starr. (Orientierung)

(λ) $\Gamma \longrightarrow \Delta$, falls Δ aus Γ durch λ-Reduktion hervorgeht (λ-Reduktion)

(**I**) $\langle [\lambda X^1 \ldots X^k\, [Y_\alpha\, \mathbf{U}^1 \ldots \mathbf{U}^n]] = [\lambda X^1 \ldots X^k\, [\mathbf{K}\; \mathbf{V}^1 \ldots \mathbf{V}^m]] \rangle \;\&\; \Gamma \longrightarrow$

$\langle Y_\alpha = \mathbb{G}_\alpha(\mathbf{K}),\ [\lambda X^1 \ldots X^k\, [Y\, \mathbf{U}^1 \ldots \mathbf{U}^n]] = [\lambda X^1 \ldots X^k\, [\mathbf{K}\; \mathbf{V}^1 \ldots \mathbf{V}^m]] \rangle \;\&\; \Gamma,$

falls Y_α eine Variable, **K** eine Konstante oder freie Variable und $\mathbb{G}_\alpha(\mathbf{K})$ der allgemeinste Term vom Typ α mit Kopfsymbol **K** ist. (Imitation)

(**P^j**) $\langle [\lambda X^1 \ldots X^k\, [Y_\alpha\, \mathbf{U}^1 \ldots \mathbf{U}^n]] = [\lambda X^1 \ldots X^k\, [\mathbf{K}\; \mathbf{V}^1 \ldots \mathbf{V}^m]] \rangle \;\&\; \Gamma \longrightarrow$

$\langle Y_\alpha = \mathbb{G}_\alpha(j),\ [\lambda X^1 \ldots X^k\, [Y\, \mathbf{U}^1 \ldots \mathbf{U}^n]] = [\lambda X^1 \ldots X^k\, [\mathbf{K}\; \mathbf{V}^1 \ldots \mathbf{V}^m]] \rangle \;\&\; \Gamma,$

falls Y_α eine Variable, das Kopfsymbol von $\mathbf{U}^j$ gerade **K** ist und $\mathbb{G}_\alpha(j)$ der allgemeinste j-Projektionsterm vom Typ α ist. (j-Projektion)

(**E**) $\langle [\lambda X^1 \ldots X^k\, [Y_\alpha\, \mathbf{U}^1 \ldots \mathbf{U}^n]] = [\lambda X^1 \ldots X^k\, [Z_\beta \mathbf{V}^1 \ldots \mathbf{V}^m]] \rangle \;\&\; \Gamma \longrightarrow$

$\langle Y_\alpha = \mathbb{G}_\alpha,\ [\lambda X^1 \ldots X^k\, [Y_\alpha \mathbf{U}^1 \ldots \mathbf{U}^n]] = [\lambda X^1 \ldots X^k\, [Z_\beta \mathbf{V}^1 \ldots \mathbf{V}^m]] \rangle \;\&\; \Gamma,$

falls Y_α und Z_β freie Variablen sind, und $\mathbb{G}_\alpha$ gerade irgendein $\mathbb{G}_\alpha(\mathbf{K})$ oder irgendein $\mathbb{G}_\alpha(j)$ wie in der Imitation oder Projektion ist. (Explosion)

Man kann zeigen, daß der Algorithmus, der durch diese Transformationsregeln induziert wird, ein *vollständiger* und *korrekter* Unifikationsalgorithmus ist, das heißt, der Algorithmus findet zu jedem Unifikator σ für Γ einen Unifikator θ, der allgemeiner ist als σ. Huets Unifikationsalgorithmus realisiert also ein *Semi-Entscheidungsverfahren* für das Unifikationsproblem. Ein solcher Algorithmus ist auch das Beste, was wir nach den oben erwähnten theoretischen Resultaten (Unentscheidbarkeit und Typ 0) erwarten können. Dieser Unifikationsalgorithmus bleibt vollständig, wenn die Regeln (**B**) und (λ) bevorzugt angewendet werden. Bei dieser Strategie werden diese Regeln nach jeder Regelanwendung so lange auf die neu erzeug-

ten Paare angewandt, bis alle isolierten Variablen gebunden und alle Terme in λ-Normalform sind.

In einem Unifikationsproblem müssen die Binder $\lambda X^1 \ldots X^k$ der beiden Terme jeden Paares die gleiche Länge und die gleichen Typen der gebundenen Variablen haben, sonst ist das Paar nicht unifizierbar. Damit gibt es also immer alphabetische Varianten des Paars, in denen die Binder gleich sind. Bis auf die Bindungsregel (**B**) erhalten alle Transformationsregeln die Binder der Paare und verändern nur die Rümpfe, denn die Binder haben in diesen Regeln nur die Funktion zu unterscheiden, welche Variablen im Rumpf der Terme gebunden und welche Variablen frei vorkommen. Betrachten wir die Dekompositionsregel (**D**) ohne die Binder, so erhalten wir die Dekompositionsregel der Robinson-Unifikation: bei gleichen Kopfsymbolen sind Terme genau dann unifizierbar, wenn die Subterme unifizierbar sind. Die Bindungsregel (**B**) arbeitet auf Paaren, die (bis auf η-Gleichheit und gebundene Variablen) von der Form $\langle Y_\alpha = \mathbf{A}_\alpha \rangle$ sind, wobei Y_α eine Variable ist, die nicht frei in $\mathbf{A}_\alpha$ vorkommt. Wie in der Robinson-Unifikation wird durch die Bindungsregel der Teilunifikator $\{Y_\alpha \leftarrow \mathbf{A}_\alpha\}$ für das Paar $\langle Y_\alpha = \mathbf{A}_\alpha \rangle$ auf das restliche Unifikationsproblem angewandt. Die Regeln (**T**), (**D**), (**B**) und (**O**) entsprechen also gerade den Regeln für die Robinson-Unifikation in PL1.

Die Regeln (**I**) und ($\mathbf{P^j}$) erzeugen gerade Teilunifikatoren σ_I und σ_P, wie oben am Beispiel beschrieben, und fügen sie dem Unifikationsproblem hinzu. Ist **K** eine Variable oder Konstante vom Typ $(\beta\gamma^1 \ldots \gamma^m)$, dann ist der *allgemeinste Term* $\mathbb{G}_\alpha(\mathbf{K})$ *vom Typ* $\alpha = (\beta\delta^1 \ldots \delta^n)$ *mit Kopfsymbol* **K** von der Form

$$\mathbb{G}_\alpha(\mathbf{K}) = [\lambda Z^1 \ldots Z^n\ [\mathbf{K}\ [H^1 Z^1 \ldots Z^n] \ldots [H^m Z^1 \ldots Z^n]],$$

wobei die H^i und die Z^j neue Variablen der Typen $(\gamma^i\delta^1 \ldots \delta^n)$ beziehungsweise δ^j sind. Ist **K** die gebundene Variable Z^j, dann nennen wir den obigen Term den allgemeinsten *j-Projektionsterm* $\mathbb{G}_\alpha(j)$. Man kann die Konstruktion dieses Terms durch die folgenden Überlegungen plausibel machen. Der Term $\mathbb{G}_\alpha$ soll den Typ $\alpha = (\beta\delta^1 \ldots \delta^n)$ haben, deswegen muß der Binder von der Form $\lambda Z^1 \ldots Z^n$ sein. Der Rumpf soll der allgemeinste Term mit Kopfsymbol **K** sein. Da **K** den Typ $(\beta\gamma^1 \ldots \gamma^m)$ hat, muß der Rumpf von der Form $[\mathbf{K}\mathbf{E}^1 \ldots \mathbf{E}^m]$ sein, wobei die $\mathbf{E}^i$ allgemeinste Terme vom Typ γ^i sind, die von allen X^j abhängen können. Also müssen die $\mathbf{E}^i$ Terme der Form $[H^1 Z^1 \ldots Z^n]$ sein. Für einen gegebenen Typ α und ein Kopfsymbol **K** oder einen Projektionsindex *j* sind diese Terme (bis auf die Wahl der Namen für die neuen Variablen H^i) eindeutig. Offensichtlich tritt hier ein erster *Indeterminismus* in der Suche nach Unifikatoren auf, denn die Regeln (**I**) und ($\mathbf{P^j}$) können (auch für verschiedene *j*) gleichzeitig anwendbar sein.

Regel (**E**) ist für *flex-flex*-Paare zuständig, das heißt, für Paare aus flexiblen Termen. Da in diesem Fall beide Terme eine freie Variable als Kopfsymbol haben, darf man sich nicht auf the Imitation des Kopfsymbols beziehungsweise Projektion auf einen Subterm wie in den Regeln (**I**) und ($\mathbf{P^j}$) beschränken, sondern muß Einsetzungen für beliebige Kopfsymbole zulassen. Die freien Variablen am Kopf müssen also an alle passenden, allgemeinsten Terme $\mathbb{G}_\alpha(j)$ beziehungsweise $\mathbb{G}_\alpha(\mathbf{K})$ gebunden werden können. Die Regel (**E**) ergibt eine *Explosion* des Suchraums für Unifikatoren. Ist $\alpha = (\beta\delta^1 \ldots \delta^n)$, so gibt es für jedes $j \leq n$ eine Möglichkeit zur Anwendung von ($\mathbf{P^j}$) und für jedes Konstantensymbol **K** vom Typ $(\beta\gamma^1 \ldots \gamma^m)$ eine Möglichkeit

zur Anwendung der Transformationsregel (**I**). Gibt es zum Beispiel unendlich viele Konstantensymbole, so kann die Regel (**E**) sogar unendlich verzweigend sein. Man kann jedoch nicht auf eine Regel (**E**) in irgendeiner Form verzichten, ohne die Vollständigkeit des Algorithmus zu verlieren (siehe [Hue76]).

Ist auf ein Unifikationsproblem Δ, das durch eine Reihe von Anwendungen der obigen Transformationsregeln aus einem Unifikationsproblem Γ entstanden ist, keine Regel mehr anwendbar, so ist es entweder in gelöster Form (und damit ist σ_Γ ein Unifikator für Γ) oder es enthält Termpaare der Form $\langle [\lambda X^1 ... X^k\ [a\ \mathbf{U}^1 ... \mathbf{U}^n]] = [\lambda X^1 ... X^k\ [b\ \mathbf{V}^1 ... \mathbf{V}^n]]\rangle$, wobei a und b voneinander verschiedene Konstanten sind (*clash*) oder Termpaare der Form $\langle X_\alpha = \mathbf{A}_\alpha\rangle$, wobei X_α in $\mathbf{A}_\alpha$ vorkommt (*occurs check*). In beiden Fällen sind weder Δ noch Γ unifizierbar.

Aus dem obigen System von Transformationsregeln erhalten wir ein System für *einseitige Unifikation* (englisch *matching*) in der Typtheorie, indem wir die Orientierungsregel (**O**) streichen und in der Bindungsregel (**B**) die Anwendung der Substitution σ auf die linken Seiten der Termpaare in Γ beschränken. Das einseitige Unifikationsproblem ist für Logiken erster und zweiter Stufe entscheidbar (siehe [Hue76]). Für Logiken dritter und höherer Stufe ist die Frage nach der Entscheidbarkeit noch offen.

Wir wollen die Transformationen an einem kleinen Beispiel erläutern. Ist F eine Variable und sind g und a Konstanten, so führt die folgende Folge von Anwendungen von Transformationsregeln zu einem Unifikationsproblem in gelöster Form:

$$
\begin{aligned}
\langle F[ga] = g[Fa]\rangle &\longrightarrow_{(\mathrm{I})} \langle F = [\lambda X\ g[HX]]\rangle, \langle F[ga] = g[Fa]\rangle \\
&\longrightarrow_{(\mathrm{B})} \langle F = [\lambda X\ g[HX]]\rangle, \langle [\lambda X\ g[HX]][ga] = g[[\lambda X\ g[HX]]a]\rangle \\
&\longrightarrow_{(\lambda)} \langle F = [\lambda X\ g[HX]]\rangle, \langle g[H[ga]] = g[g[Ha]]\rangle \\
&\longrightarrow_{(\mathbf{D})} \langle F = [\lambda X\ g[HX]]\rangle, \langle H[ga] = g[Ha]\rangle \\
&\longrightarrow_{(\mathbf{P1})} \langle H = [\lambda Z\ Z]\rangle, \langle F = [\lambda X\ g[HX]]\rangle, \langle H[ga] = g[Ha]\rangle \\
&\longrightarrow_{(\mathbf{B})} \langle H = [\lambda Z\ Z]\rangle, \langle F = [\lambda X\ g[[\lambda Z\ Z]X]]\rangle, \langle [\lambda Z\ Z][ga] = g[[\lambda Z\ Z]a]\rangle \\
&\longrightarrow_{(\lambda)} \langle H = [\lambda Z\ Z]\rangle, \langle F = [\lambda X\ gX]\rangle, \langle ga = ga\rangle \\
&\longrightarrow_{(\mathbf{T})} \langle H = [\lambda Z\ Z]\rangle, \langle F = [\lambda X\ gX]\rangle
\end{aligned}
$$

Die Substitution $\{F \leftarrow [\lambda X\ gX]\}$ ist also ein Unifikator der Terme $F[ga]$ und $g[Fa]$. Als ersten Schritt in dieser Ableitung hätten wir auch gleich die Regel ($\mathbf{P}^1$) anwenden, und uns so die ersten vier Schritte sparen können. Dies zeigt, daß bei der Suche nach Unifikatoren ein echter Indeterminismus auftreten kann.

2.2 Prä-Unifikation

Gérard Huet hat gesehen, daß es für einen Resolutionskalkül ausreicht, alle Termpaare bis auf die flex-flex-Paare zu unifizieren (siehe unten) und die problematischen flex-flex-Paare als schon unifiziert zu betrachten. Wir wollen zwei Terme π-*gleich* nennen, wenn sie den gleichen Typ haben und beide flexibel sind. Die Unifikations in der Typtheorie unter der $\alpha\beta\eta\pi$-Gleichheitstheorie wird *Prä-Unifikation* genannt, Lösungen von Prä-Unifikationsproblemen nennen wir *Prä-Unifikatoren*.

Wir nennen ein Unifikationsproblem Γ *prä-gelöst*, wenn alle Paare in Γ in gelöster Form oder flex-flex-Paare sind. Γ ist also eine Vereinigung $\Sigma \cup \Phi$, wobei Σ das Teilsystem der gelösten und Φ das der flex-flex-Paare ist. Offensichtlich ist der allgemeinste Unifikator σ_Σ für die Teilmenge Σ der gelösten Paare ein *Prä-Unifikator* für Γ. Sei

$$\Pi := \langle [\lambda X^1 \ldots X^k\, [Y\, \mathbf{U}^1 \ldots \mathbf{U}^n]] = [\lambda X^1 \ldots X^k\, [Z\, \mathbf{V}^1 \ldots \mathbf{V}^m]] \rangle$$

ein flex-flex-Paar aus Φ, wobei Y eine Variable vom Typ $(\alpha\beta^1 \ldots \beta^n)$ ist und Z eine Variable vom Typ $(\alpha\gamma^1 \ldots \gamma^m)$. Ist F_α eine neue Variable vom Typ α, so ist

$$\sigma_\Pi := \{Y \leftarrow [\lambda\, V_{\beta 1} \ldots V_{\beta n}\, F_\alpha],\, Z \leftarrow [\lambda\, V_{\gamma 1} \ldots V_{\gamma m}\, F_\alpha]\}$$

ein Unifikator für Π, denn $\sigma_\Pi(\Pi) =_\lambda \langle [\lambda X^1 \ldots X^k\, F_\alpha] = [\lambda X^1 \ldots X^k\, F_\alpha] \rangle$ ist ein tautologisches Paar. Auf diese Weise lassen sich Prä-Unifikatoren für Γ immer zu Unifikatoren für Γ erweitern. Also ist ein Unifikationsproblem *unifizierbar* genau dann, wenn es *prä-unifizierbar* ist. Man beachte, daß deswegen die Prä-Unifikation in der Logik höherer Stufe ebenso unentscheidbar ist wie die Unifikation.

Man erhält einen Prä-Unifikationsalgorithmus, wenn man in dem obigen Algorithmus die Transformationsregel (**E**) wegläßt, die Dekompositionsregel (**D**) nur anwendet, wenn **K** eine Konstante oder die gebundene Variable X^j ist, und die Regeln (**I**) beziehungsweise ($\mathbf{P}^j$) nur anwendet, wenn **K** eine Konstante ist. Dieser Algorithmus ist vollständig in dem Sinne, daß der Algorithmus zu jedem Prä-Unifikator σ für Γ einen allgemeineren Prä-Unifikator θ für Γ findet. Obwohl Huets Prä-Unifikationsalgorithmus immer noch nur ein Semi-Entscheidungsverfahren ist, ist er in der Praxis brauchbar. Er hat nämlich einen endlich verzweigenden Suchraum, da man auf die Explosionsregel (**E**) verzichten kann, die den Hauptteil der Komplexität in den Unifikationsalgorithmus gebracht hat.

2.3 Beweisprüfer

Beweisprüfer, auch *Beweiseditoren* genannt, sind Deduktionssysteme, die es dem Benutzer ermöglichen, *interaktiv* und kontrolliert Beweise zu entwickeln. Sie setzen meist auf einer Typtheorie (oft mit abhängigen Typen wegen der Darstellbarkeit der Beweistheorie) und einem System von Schlußregeln auf. Dabei umfassen die Schlußregeln meist eines von Gentzens Systemen zum natürlichen Schließen und sind, wie die Wahl der Logik, stark vom gewünschten Anwendungsfeld abhängig.

Im Gegensatz zu *automatischen* Beweisern, bei denen der Benutzer dem System die Axiome und das zu beweisende Theorem vorgibt, das System startet und dann so lange wartet, bis das System den Beweis gefunden hat oder eines der Abbruchkriterien erfüllt ist, baut der Benutzer bei der Verwendung eines Beweisprüfers interaktiv durch Eingabe von Schlußregeln den Beweis auf. Diese Beweisschritte werden dann durch das System auf die Anwendbarkeit und Korrektheit *überprüft* (daher der Name Beweisprüfer). So ist sichergestellt, daß in dem System nur korrekte Beweise geführt werden können. Für die Überprüfung der Anwendbarkeit von Schlußregeln muß festgestellt werden, ob die jeweilige Formel eine Instanz der linken Seite der gewünschten Schlußregel ist, deswegen benutzen Beweisprüfer die einseitige Unifikation als die zentrale Inferenztechnik. Ein wichtiges Beispiel für einen Beweisprüfer ist das System

AUTOMATH (siehe [dBr80]), das für die Aufgabe des Prüfens von mathematischen Beweisen entwickelt wurde. In diesem System wurde ein wichtiges Buch aus der reinen Mathematik als formal korrekt bewiesen. Allerdings ist die Kodierung eines Beweises in AUTOMATH in der Regel um einen Faktor 10 - 20 länger als in der natürlichen mathematischen Sprache, so daß dieses System sich in der Praxis nicht durchsetzen konnte.

Die Vorgehensweise der interaktiven Beweisentwicklung wird normalerweise durch die Möglichkeit erleichtert, mehrere Schritte zu sogenannten *Taktiken* zusammenzufassen. Dafür wird eine Programmiersprache angeboten, mit der dann der Beweisprüfer programmiert werden kann. Solche Deduktionssysteme werden *taktische Theorembeweiser* genannt, denn es besteht die Hoffnung, daß über die Entwicklung von immer mächtigeren Taktiken ein gewisser Grad von Automation erreicht werden kann.

Das System Nuprl [Con86] ist ein Beispiel für einen taktischen Theorembeweiser. Dieses System beruht auf Martin-Löfs Typtheorie, einer Typtheorie mit abhängigen Typen und einem *intuitionistischen* Kalkül des natürlichen Schließens. Durch diese Wahl der zugrundeliegenden Logik sind alle Beweise sowohl konstruktiv als auch explizit in der Logik repräsentiert. Diese Tatsache erlaubt es, Beweise für existentielle Theoreme, also Sätze von der Form „Sei ..., dann gibt es für alle ... ein *X*, so daß ...“ als funktionale Programme anzusehen, die gerade ein solches *X* berechnen, deren Existenz in den Theoremen postuliert werden. Dabei ist bemerkenswert, daß verschiedene Beweise in der Regel auch verschiedene Programme ergeben. Nuprl unterstützt diese Beobachtung, indem es erlaubt, gefundene Beweise für existentielle Sätze im System direkt ablaufen zu lassen. So kann man zum Beispiel in Nuprl einen Sortieralgorithmus synthetisieren, indem man den Satz „Jede Liste *L* hat gleich viele Elemente wie die sortierte Liste *S* mit den gleichen Elementen“ beweist, denn der Beweis konstruiert eine Abbildung zwischen der Liste *L* und *S*.

2.4 Automatische Beweisverfahren

Wir werden im folgenden den Kalkül der *Constrained Resolution* von G. Huet (siehe [Hue72] und [And71]) als ein Beispiel für einen Widerlegungskalkül für das automatische Beweisen diskutieren.

Die Beweisverfahren in der Typtheorie sind Verallgemeinerungen der Beweisverfahren für PL1, deswegen wollen wir noch einmal kurz die gemeinsamen Grundregeln der wichtigsten Beweisprozeduren für PL1 wiederholen. In den klassischen Widerlegungskalkülen (Resolution, Matrixmethode, Tableaus usw.) haben wir vier Arten von Schlußregeln:

1. die *Junktorenregeln*, wie die Kommutativität und Distributivität von Konjunktion und Disjunktion, und die Negationsregeln,
2. die *Quantorenregeln*, wie zum Beispiel die Skolemisierungsregeln und die Vertauschungsregeln für Allquantor und Negation,
3. eine Form der *Schnittregel* (verallgemeinerter Modus Ponens)

4. und eine Form der *Substitutionsregel*, die die Semantik der freien beziehungsweise allquantifizierten Variablen beschreibt.

Die ersten beiden Klassen von Regeln werden zur Erstellung der Klauselnormalform (in der Resolution) oder der disjunktiven Normalform (in der Matrixmethode) gebraucht und kommen deswegen in der weiteren Beweissuche nicht mehr vor. Die Schnittregel findet sich im Resolutionsschritt beziehungsweise in komplementären Pfaden durch eine Matrix. Die Regeln in den ersten drei Klassen sind in Beweisen fast deterministisch anzuwenden, das heißt, wenn man eine Menge von Formeln gegeben hat, gibt es nur endlich viele Möglichkeiten, sie anzuwenden.

Die Substitutionsregel dagegen ist *unendlich verzweigend* – es gibt im allgemeinen unendlich viele Einsetzungen für eine Variable – deswegen wurde diese Regel im Resolutions- und Matrixkalkül auch durch die Einführung der Unifikation spezialisiert und damit handhabbar gemacht. Die *Resolutionsregel* ist eine Kombination aus Schnittregel und Substitutionsregel, die nur solche Instanziierungen (Anwendungen der Substitutionsregel) gestattet, die nötig sind, um die Schnittregel anzuwenden; der allgemeinste Unifikator repräsentiert dabei alle (weniger allgemeinen) Substitutionen (unendlich viele). Dies ist möglich, weil das Unifikationsproblem in PL1 entscheidbar ist und deswegen jede Resolvente gibt, die in beschränkter Zeit ausgerechnet werden kann.

In der Typtheorie ist das Vorgehen, die Schnittregel mit der Substitutionsregel in dieser Weise zu verknüpfen, nicht sinnvoll, denn das Unifikationsproblem ist nicht nur unentscheidbar, sondern auch vom Typ 0. Außerdem gibt es schon in der Logik zweiter Stufe Prädikatsvariablen, durch die mit der Substitution Junktoren und sogar Quantoren in den Beweis eingeführt werden können. Deswegen können die aussagenlogischen Regeln und Quantorenregeln nicht wie in PL1 in einen Präprozeß ausgelagert werden, sondern sie müssen in irgendeiner Form auch während der Beweissuche zur Verfügung stehen.

2.5 Huets Constrained Resolution

Gérard Huet behandelt die oben beschriebenen Probleme in seinem Kalkül der *Constrained Resolution*, indem er die sogenannten *Splitting*-Regeln einführt und den Unifikationsschritt aus dem Resolutionsschritt herausnimmt und verzögert, bis eine leere Klausel gefunden worden ist. Mit diesem Kalkül lassen sich alle Sätze ableiten, die auch mit Gentzens Kalkül **NK** ableitbar sind.

Wir stellen im folgenden eine Variante von Huets Constrained Resolution vor, die einen Semi-Entscheidungsverfahren für die Typtheorie liefert. Wir lösen das Problem mit der Unentscheidbarkeit des Unifikationsproblems und dem Unifikationstyp, indem wir die Schnittregel und die Unifikation trennen: Für zwei Klauseln mit *komplementären Literalen* **M** und **N** (gleiches Prädikatensymbol, aber verschiedene Vorzeichen) wird die *Resolvente* erzeugt, unabhängig davon, ob die Literale unifizierbar sind oder nicht. Das Unifikationsproblem $\langle \mathbf{M} = \mathbf{N} \rangle$ wird in eine *Nebenbedingung* (englisch *constraint*) zu der Klausel geschrieben. Diese Nebenbedingung merkt sich die Voraussetzung (**M** und **N** unifizierbar) dafür, daß der

Resolutionsschritt überhaupt ausgeführt werden durfte. Während der Beweissuche werden also immer mehr Nebenbedingungen aufgesammelt, die dann – gleichberechtigt zu der Suche nach leeren Klausen – vereinfacht werden. Der Beweis ist vollendet, sobald die *leere Klausel* mit *leerer Nebenbedingung* abgeleitet wurde. Die leere Klausel entspricht dem Widerspruch, die leere Nebenbedingung entspricht der Tatsache, daß alle Nebenbedingungen im Beweis unifizierbar – und damit alle Resolutionsschritte korrekt – waren. Auf diese Weise werden die beiden unentscheidbaren Suchprobleme parallel abgearbeitet, und so kann kein Teilproblem das andere unendlich lange verzögern. Würde man den Unifikationsschritt nicht vom Resolutionsschritt trennen, so würde das Verfahren auch bei widersprüchlicher Klauselmenge im allgemeinen nicht terminieren, denn es gibt auch in diesem Fall im allgemeinen komplementäre Literale, die nicht unifizierbar sind und auf denen die Unifikation nicht terminiert.

Der Kalkül arbeitet auf *Klauseln mit Nebenbedingung* von der Form $\mathbf{C} \parallel \Gamma$, wobei $\mathbf{C}$ eine Klausel (eine Menge von Literalen) und Γ ein Unifikationsproblem ist.

Die Schlußregeln des Kalküls der Constrained Resolution bestehen aus der *Resolutionsregel* R:

$$\begin{array}{ll} & \text{Klausel 1:} \quad \neg\mathbf{M}^1, ..., \mathbf{M}^m \parallel \Gamma \\ & \text{Klausel 2:} \quad \mathbf{N}^1, ..., \mathbf{N}^n \parallel \Delta \\ \text{R:} & \hrulefill \\ & \text{Resolvente:} \quad \mathbf{M}^2, ..., \mathbf{M}^m, \mathbf{N}^2, ..., \mathbf{N}^n \parallel \Gamma\&\Delta \;\&\; \langle \mathbf{M}^1 = \mathbf{N}^1 \rangle, \end{array}$$

der *Faktorregel* F:

$$\begin{array}{ll} & \text{Klausel 1:} \quad \mathbf{M}^1, \mathbf{M}^2, \mathbf{M}^3, ..., \mathbf{M}^m \parallel \Gamma \\ \text{F:} & \hrulefill \\ & \text{Faktor:} \quad \mathbf{M}^2, ..., \mathbf{M}^m \parallel \Gamma \;\&\; \langle \mathbf{M}^1 = \mathbf{M}^2 \rangle, \end{array}$$

der *Vereinfachungsregel*: U

$$\begin{array}{ll} & \mathbf{M}^1, ..., \mathbf{M}^m \parallel \Gamma \\ \text{U:} & \hrulefill \end{array}$$

$\mathbf{M}^1, ..., \mathbf{M}^m \parallel \Delta$, falls Δ aus Γ durch Anwendung einer der Transformationsregeln (**T**), (**D**), (**O**), (λ), (**I**) oder ($\mathbf{P}^j$) hervorgeht,

der *Bindungsregel* B:

$$\begin{array}{ll} & \mathbf{M}^1, ..., \mathbf{M}^m \parallel \Gamma \;\&\; \langle X = \mathbf{A} \rangle \\ \text{B:} & \hrulefill \end{array}$$

$\sigma(\mathbf{M}^1), ..., \sigma(\mathbf{M}^m) \parallel \sigma(\Gamma)$, wobei σ die Substitution $\{X \leftarrow \mathbf{A}\}$ist,

und den *Splittingregeln* für *positive Literale* S:

$$\begin{array}{ll} & \mathbf{M}^1, ..., \mathbf{M}^m \parallel \Gamma \\ \text{S:} & \hrulefill \\ & P_o, Q_o, \mathbf{M}^2, ..., \mathbf{M}^m \parallel \Gamma \;\&\; \langle \mathbf{M}^1 = P_o \vee Q_o \rangle, \end{array}$$

$$\text{S:}\quad \frac{\mathbf{M}^1, ..., \mathbf{M}^m \parallel \Gamma}{\neg P_o, \mathbf{M}^2, ..., \mathbf{M}^m \parallel \Gamma \;\&\; \langle \mathbf{M}^1 = \neg P_o \rangle},$$

$$\text{S:}\quad \frac{\mathbf{M}^1, ..., \mathbf{M}^m \parallel \Gamma}{R_{o\alpha} Z_\alpha, \mathbf{M}^2, ..., \mathbf{M}^m \parallel \Gamma \;\&\; \langle \mathbf{M}^1 = \forall X_\alpha\, R_{o\alpha} X_\alpha \rangle},$$

wobei die P_o, Q_o und Z_α neue Variablen sind.

Für negative Literale gibt es einen analogen Satz von *Splittingregeln* für *negative Literale*.

Die Splittingregeln bilden die Einsetzung eines einfachen Junktor- bzw. Quantorterms für die Kopfvariable eines flexiblen Literals mit nachfolgender Überführung in die Klauselnormalform nach. Die erste Splittingregel entspricht gerade der Anwendung der Substitution

$$\sigma := \{ Y \leftarrow [\lambda X^1 ... X^n\ [R X^1 ... X^n] \vee [S X^1 ... X^n]] \}$$

für die Kopfvariable von $\mathbf{M}^1$. Sind nämlich $\mathbf{M}^1 = [Y \mathbf{A}^1 ... \mathbf{A}^n]$ und $\mathbf{C} := \mathbf{M}^1, ..., \mathbf{M}^m \parallel \Gamma$, wobei P nicht mehr in $\mathbf{M}^1, ..., \mathbf{M}^m$ und Γ vorkommt, so ist

$$\sigma(\mathbf{C}) = [R \mathbf{A}^1 ... \mathbf{A}^n] \vee [S \mathbf{A}^1 ... \mathbf{A}^n], \mathbf{M}^2, ..., \mathbf{M}^m \parallel \Gamma$$

und nach Überführung in Klauselnormalform

$$\sigma(\mathbf{C}) = [R \mathbf{A}^1 ... \mathbf{A}^n], [S \mathbf{A}^1 ... \mathbf{A}^n], \mathbf{M}^2, ..., \mathbf{M}^m \parallel \Gamma.$$

Wir wollen nun diesen Vorgang durch die Anwendung einer Splittingregel simulieren:

1. $\mathbf{M}^1, ..., \mathbf{M}^m \parallel \Gamma$

2 = S(1). $P_o, Q_o, \mathbf{M}^2, ..., \mathbf{M}^m \parallel \Gamma \;\&\; \langle [Y \mathbf{A}^1 ... \mathbf{A}^n] = P_o \vee Q_o \rangle$

3 = I(2). $P_o, Q_o, \mathbf{M}^2, ..., \mathbf{M}^m \parallel \Gamma \;\&\; \langle [Y \mathbf{A}^1 ... \mathbf{A}^n] = P_o \vee Q_o,\ Y = [\lambda X^1 ... X^n\ [H^1 X^1 ... X^n] \vee [H^2 X^1 ... X^n]] \rangle$

4 = B(3). $P_o, Q_o, \mathbf{M}^2, ..., \mathbf{M}^m \parallel \Gamma \;\&\; \langle [H^1 \mathbf{A}^1 ... \mathbf{A}^n] \vee [H^2 \mathbf{A}^1 ... \mathbf{A}^n] = P_o \vee Q_o \rangle$,

5 = D(4). $P_o, Q_o, \mathbf{M}^2, ..., \mathbf{M}^m \parallel \Gamma \;\&\; \langle P_o = [H^1 \mathbf{A}^1 ... \mathbf{A}^n],\ Q_o = [H^2 \mathbf{A}^1 ... \mathbf{A}^n] \rangle$,

6 = B(5). $[H^1 \mathbf{A}^1 ... \mathbf{A}^n], [H^2 \mathbf{A}^1 ... \mathbf{A}^n], \mathbf{M}^2, ..., \mathbf{M}^m \parallel \Gamma$.

Klausel 6 ist offensichtlich bis auf die Namen der neuen Variablen gleich $\sigma(\mathbf{C})$. Durch diesen Trick haben wir die aussagenlogischen Regeln und Quantorenregeln implizit in die Splittingregeln kodiert. Da sie nicht mehr explizit im Kalkül vorhanden sein müssen, können sie wie in der Resolution für PL1 in einen Präprozeß ausgelagert werden.

Das Beweissystem TPS [AMLP84] von der Gruppe um P. Andrews an der Carnegie Mellon University ist im Moment das einzige existierende, automatische Beweissystem für die Typtheorie. Es wird jedoch ein weiteres Sytem (Ω-MKRP) von der Gruppe um J. Siekmann an der Universität des Saarlandes entwickelt.

2.6 Beispiele

Als Beispiel für die Wirkungsweise der Resolution in der Typtheorie wollen wir aus den Peano-Axiomen beweisen, daß der Nachfolger $s n$ einer natürlichen Zahl n niemals gleich n ist. Wir geben die Klauselmenge für die Axiome an:

1. $n_{o\iota}0_\iota \parallel \emptyset$
2. $\neg n X_\iota \vee [n[sX]] \parallel \emptyset$
3. $\neg n X_\iota \vee [\neg[sX] = 0_\iota] \parallel \emptyset$
4. $\neg[sX = sY] \vee [X = Y] \parallel \emptyset$
5. $\neg P_{o\alpha}0 \vee [P[f_{\iota(o\iota)}P] \vee \neg n Y_\iota \vee PY \parallel \emptyset$
6. $\neg P_{o\alpha}0 \vee \neg[P[s[f_{\iota(o\iota)}P]] \vee \neg n Y_\iota \vee PY \parallel \emptyset$

und das negierte Theorem:

7. $n_{o\iota}k_\iota \parallel \emptyset$
8. $sk_\iota = k \parallel \emptyset$, wobei sind $f_{\iota(o\iota)}$ und k_ι Skolemkonstanten sind.

Wir beweisen nun

9=R(5,7). $\neg P_{o\alpha}0 \vee P[fP] \vee PY_\iota \parallel \langle Y = k\rangle$

10=B(9). $\neg P_{o\alpha}0 \vee P[fP] \vee Pk_\iota \parallel \emptyset$

11=S(10). $\neg P_{o\alpha}0 \vee P[fP] \vee \neg Q_o \parallel \langle \neg Q_o = Pk\rangle$

12=R(11,8). $\neg P_{o\alpha}0 \vee P[fP] \parallel \langle \neg Q_o = Pk\rangle, \langle Q_o = [sk_\iota = k]\rangle$

13=U(12). $\neg P_{o\alpha}0 \vee P[fP] \parallel \langle P = [\lambda X_\iota \neg[sX_\iota = X]\rangle$

14=B(13). $[s0 = 0] \vee P[fP] \parallel \langle P = [\lambda X_\iota \neg[sX_\iota = X]\rangle$

15=R(1,3). $\neg[s0 = 0] \parallel \emptyset$

16=R(14,15). $P[fP] \parallel \langle P = [\lambda X_\iota \neg[sX_\iota = X]]\rangle$

17=B(16). $\neg[s[fP] = fP] \parallel \langle P = [\lambda X_\iota \neg[sX_\iota = X]]\rangle$

und analog aus 6.

18. $[ss[fP] = s[fP]] \parallel \langle P = [\lambda X_\iota \neg[sX_\iota = X]]\rangle$

19=BR(4,18). $[s[fP] = [fP]] \parallel \langle P = [\lambda X_\iota \neg[sX_\iota = X]\rangle, \langle X = s[fP]\rangle, \langle Y = [fP]\rangle$

20=R(17,19). $\square \parallel \langle P = [\lambda X_\iota \neg[sX_\iota = X]\rangle, \langle X = s[fP]\rangle, \langle Y = [fP]\rangle$

21=B(20). $\square \parallel \emptyset$.

Mit diesem Lemma können wir nun auch den Satz von Cantor über die Kardinalität von $\mathbb{R}$ beweisen. Als eine kleine Vereinfachung zu der Formulierung oben nehmen wir die Menge $n_{o\iota}$

als die Menge aller Elemente vom Typ ι, das heißt, wir ersetzen $\pi_{o\iota}$ durch $[\lambda Z_\iota\ T_o]$,wobei T_o die Konstante ist, die immer zu $T \in U_o$ interpretiert wird. Durch diesen Trick erhalten wir (nach λ–Reduktion) folgende Formulierung des Satzes:

$$\neg\exists F_{\iota\iota\iota}\ \forall G_{\iota\iota}\ \exists\ J_\iota\ [FJ = G].$$

Die Klauselnormalform des negierten Theorems ist also

1. $f_{\iota\iota\iota}[j_{\iota(\iota\iota)}G_{\iota\iota}] = G_{\iota\iota} \parallel \emptyset,$
wobei $f_{\iota\iota\iota}$ und $j_{\iota(\iota\iota)}$ die Skolemkonstanten zu den Variablen $F_{\iota\iota\iota}$ und J_ι sind.

Wir benutzen im Beweis weiterhin das Axiom

2. $\neg F_{\iota\iota}=H_{\iota\iota} \vee F_{\iota\iota}N_\iota =H_{\iota\iota}N_\iota \parallel \emptyset$

zur Simulation einer Paramodulationsregel für die Gleichheit und das eben bewiesene Theorem

3. $\neg X_\iota = s_{\iota\iota}X_\iota \parallel \emptyset$

Der Beweis beruht nun im Wesentlichen auf der Unifikation:

4 = R(1,2). $FN = HN \parallel \langle F = f[jG]\ \rangle, \langle H = G\rangle$

5 = B(4). $f[jG]N = GN \parallel \emptyset$

6 = R(3,5). $\square \parallel \langle sf[jG]N = GN\ \rangle$

7 = I(6). $\square \parallel \langle G = \lambda Y\ s[H^1Y]\rangle, \langle sf[jG]N = GN\ \rangle$

8 = BD(7). $\square \parallel \langle G = \lambda Y\ s[H^1Y]\rangle, \langle f[jG]N = H^1N\rangle,$

9 = I(8). $\square \parallel \langle G = \lambda Y\ s[H^1Y]\rangle, \langle f[jG]N = H^1N\rangle, \langle H^1 = \lambda Y\ f[H^2Y][H^3Y]\rangle$

10 = BD(9). $\square \parallel \langle G = \lambda Y\ s[f[H^2Y][H^3Y]]\rangle, \langle H^1 = \lambda Y\ f[H^2Y][H^3Y]\rangle,$
$\langle jG = H^2N\rangle, \langle N = H^3N]\rangle$

11 = P^1P^1(10). $\square \parallel \langle G = \lambda Y\ s[f[H^2Y][H^3Y]]\rangle, \langle jG = H^2N\rangle, \langle N = H^3N]\rangle,$
$\langle H^2 = \lambda YY\rangle, \langle H^3 = \lambda YY\rangle$

12 = DDTO(11). $\square \parallel \langle G = \lambda Y\ s[fYY]\rangle, \langle N = jG\rangle$

13 = I(12). $\square \parallel \langle G = \lambda Y\ s[fYY]\rangle, \langle N = jG\rangle,\ \langle N = jH^4\rangle$

14 = BD(13). $\square \parallel \langle G = \lambda Y\ s[fYY]\rangle, \langle H^4 = G\rangle,\ \langle N = jH^4\rangle$

15 = BT(14). $\square \parallel \langle G = \lambda Y\ s[fYY]\rangle,\ \langle N = j[\lambda Y\ s[fYY]]\rangle$

16 = BB(15). $\square \parallel \emptyset.$

Wir wollen diese Ableitung im Kalkül der Constrained Resolution mit dem aus der Mathematik bekannten Beweis mittels Konstruktion einer Diagonalfolge vergleichen. Dieser beruht auf der folgenden Argumentation: Ist $f_{\iota\iota\iota}$ eine Abzählung der Folgen, so betrachten wir die Diagonalfolge $[\lambda Y\,[fYY]]$. Aus dieser Folge konstruiert man eine Folge, die sich von jeder Folge in der Aufzählung $f_{\iota\iota\iota}$ in mindestens einem Element (von der i-ten Folge im Element i)

unterscheidet, in dem man immer zum Nachfolger der Elemente der Diagonalfolge übergeht. Diese Folge G:=[$\lambda Y\, \mathcal{S}[fYY]$] ist also ein Gegenbeispiel zur Surjektivität der Abzählung $f_{\omega\omega}$. Es ist bemerkenswert, daß in diesem Beweis die Einsetzung [$\lambda Y\, \mathcal{S}[fYY]$] für G durch die Unifikation gefunden wird. Wie in diesem Beispiel ist die Unifikation oft der Teil des Kalküls, der die wesentlichen Ideen zu Beweisen beiträgt, deswegen ist es gerechtfertigt, die Unifikation gleichberechtigt zur Schnittregel abzuarbeiten.

Literatur

And86 P. B. Andrews: *An Introduction to Logic and Type Theory: To Truth through Proof,* Academic Press (1986)

And71 P. B. Andrews: *Resolution in Type Theory,* Journal of Symbolic Logic, No 36 (1971), 414-432

AMLP84 P. B. Andrews, D. A. Miller, E. Longini-Cohen, F. Pfenning: *Automating Higher Order Logic,* im Sammelband „Automated Theorem Proving, After 25 Years" (W.W. Bledsoe, D.W. Loveland, Hrsg..) Contemporary Mathematics Series, Vol. 29, American Mathematical Society (1984), 169-192.

Bar84 H. P. Barendregt: *The λ-Calculus, its Syntax and Semantics,* North Holland, (1984)

Chu40 A. Church: *A Formulation of the simple Theory of Types,* Journal of Symbolic Logic, No. 5 (1940), 56-68

Con86 R.L. Constable, et al. *Implementing Mathematics with the Nuprl Proof Development System,* Prentice-Hall, Englewood Cliffs, (1986)

dBr72 N. G.de Bruijn: *Lambda Calculus Notation with Nameless Dummies, a Tool for Automatic Formula Manipulation, with Application to the Church-Rosser Theorem,* Indag. Math. No 34 (1972), 381-392.

dBr80 N. G.de Bruijn: *A survey of the project AUTOMATH,* im Sammelband *To H.B. Curry - Essays on Colmbinatory Logic, Lambda Calculus and Formalism,* J. Seldin, J. R. Hindeley Hrsg., Academic Press, London (1980), 579-606

Ebb77 H.-D. Ebbinghaus: *Einführung in die Mengenlehre,* Wissenschaftliche Buchgesellschaft, Darmstadt (1977)

Gol81 W. D. Goldfarb: *The Undecidability of the Second Order Unification Problem,* Theoretical Computer Science, Vol 13 (1981), 225-230

GMW79 M. Gordon, R. Milner, C. Wadsworth: *Edinburgh LCF: A Mechanized Logic of Computation,* Lecture Notes in Computer Science 78, Springer Verlag (1979)

GS89 J. H. Gallier, W. Snyder: *Higher Order Unification Revisited: Complete Sets of Transformations,* Journal of Symbolic Computation, No. 8 (1989) 203-260

Hen50 L. Henkin: *Completeness in the Theory of Types,* Journal of Symbolic Logic, Vol 15 (1950), 81-91

HS86 J. R.Hindeley, J. P. Seldin: *Introduction to Combinators and λ-Calculus,* London

Mathematical Society Student Texts 1, Cambridge University Press, (1986)

Hue72 G. P. Huet: *Constrained Resolution: A Complete Method for Higher Order Logic,* Doktorarbeit, Case Western Reserve University, 1972

Hue73 G. P. Huet: *A Mechanisation of Type Theory,* Proceedings of the third IJCAI (1973), 139-146

Hue75 G. P. Huet: *A Unification Algorithm for the Typed λ-Calculus,* Theoretical Computer Science, Vol1 (1975), 27-57

Hue76 G. P. Huet: *Résolution d'Equations dans les Languages d'Ordre* 1,2,...ω, Thése d'Etat, Université de Paris VII (1976)

JP76 D. C. Jensen, T. Pietrzykowsky: *Mechanizing ω-order Type Theory through Unification,* Theoretical Computer Science, Vol 3 (1976), 123-171

Ker91 M. Kerber: *On the Representation of Mathematical Concepts and their Translation into First-Order Logic*, Doktorarbeit, Universität Kaiserslautern (1991)

Mar84 P. Martin-Löf: *Intuitionistic Type Theory,* Bibliopolis, Napoli (1984)

Rus08 B. Russell: *Mathematical Logic as based on the Theory of Types,* American Journal of Mathematics, Vol.1 (1908), 222-262

Sch77 K. Schütte: *Beweistheorie,* Springer Verlag, Berlin (1960)