TECHNISCHE
UNIVERSITÄT
MÜNCHEN

INSTITUT FÜR INFORMATIK

Sonderforschungsbereich 342:
Methoden und Werkzeuge für die Nutzung
paralleler Rechnerarchitekturen

Einsatz eines automatischen Theorembeweisers in einer taktikgesteuerten Beweisumgebung zur Lösung eines Beispiels aus der Hardware-Verifikation – Fallstudie –

Andreas Wolf, Andreas Kmoch

TUM-I9730
SFB-Bericht Nr. 342/20/97 A
Mai 97

TUM–INFO–05-I9730-250/1.–FI

Anforderungen an: Prof. Dr. A. Bode
Sprecher SFB 342
Institut für Informatik
Technische Universität München
D-80290 München, Germany

Druck: Fakultät für Informatik der
Technischen Universität München

Einsatz eines automatischen Theorembeweisers in einer taktikgesteuerten Beweisumgebung zur Lösung eines Beispiels aus der Hardware-Verifikation

– Fallstudie –

1. Mai 1997

Andreas Wolf
Institut für Informatik
Technische Universität München
80290 München
Germany
Phone: +49-89-28927924 Fax: +49-89-28927902
Email: wolfa@informatik.tu-muenchen.de

Andreas Kmoch
GMD FIRST
Rudower Chaussee 5
12489 Berlin
Germany
Phone: +49-30-6392-1861 Fax: +49-30-6392-1805
Email: kmoch@first.gmd.de

Zusammenfassung

Anhand einer Fallstudie aus dem Bereich der Verifikation eines Mikroprozessors wird der Einsatz des automatischen Theorembeweisers SETHEO in einer interaktiven taktikgesteuerten Beweisumgebung demonstriert. Dazu wird nach Vorstellung der theoretischen Grundlagen beschrieben, wie dieser Einsatz erfolgt. Es werden die verwendeten Taktiken erläutert und die maschinell generierten Beweise angegeben.

Inhaltsverzeichnis

Einleitung

Beim Entwurf von Hardware und Programmen wird häufig eine anfängliche grobe Spezifikation der erwünschten Funktionalität stufenweise immer weiter verfeinert, bis schließlich ein in der verwendeten Programmiersprache formulierbares Programm oder ein Konstruktionsplan für einen Schaltkreis entsteht. Aufgabe der Hardware- oder Software-Verifikation ist es nun, die Korrektheit der Übergänge zwischen den einzelnen Spezifikationsstufen mittels formaler Methoden zu beweisen.

Die dabei auftretenden Beweisverpflichtungen sollen in diesem Bericht an einem Beispiel untersucht werden, welches von Harman und Tucker stammt [7]. Dabei wird eine Implementation einer Additionsfolge mit maximal 2 Additionen betrachtet. Die Maschine des höheren Abstraktionslevels verfügt über die Addition natürlicher Zahlen, während die Maschine des niedrigeren Abstraktionslevels nur inkrementieren und dekrementiern kann. Im ersten Abschnitt werden die für den Beweis der Aufgabe notwendigen theoretischen Grundlagen dargelegt, die Beweisaufgabe exakt formuliert und die zur Lösung der während des Beweises der Aufgabe entstehenden Probleme verwendeten Systeme Ilf [1] und Setheo [8] vorgestellt. Im zweiten Abschnitt werden die zur Lösung der Aufgabe erforderlichen Taktiken entwickelt. Der dritte Abschnitt beschäftigt sich mit Problemen der Theorieauswahl. Im vierten Abschnitt wird die Lösung der Beispielaufgabe beschrieben. Nach Schlußfolgerungen im fünften Abschnitt findet man im Anhang ein Beispiel eines maschinellen Beweises einschließlich seiner Darstellung in einer unter Benutzung weiterer Funktionalitäten von Ilf [3] durchgeführten maschinellen Übersetzung in natürliche Sprache.

1 Theoretische Grundlagen

In diesem Abschnitt werden Erläuterungen zum mathematischen Beweisen und zur algebraischen Beschreibung von Objekten gegeben. Außerdem wird der theoretische Hintergrund zur Beweisaufgabe besprochen.

1.1 Mathematische Grundlagen

Die mathematische Grundlage dieses Berichts bildet die Prädikatenlogik erster Stufe. Prädikatenlogik und Aussagenlogik werden wie in [16] beschrieben vorausgesetzt. Wir verwenden in diesem Bericht den Prädikatenkalkül erster Stufe ohne freie Variablen. Weiterhin werden einige Grundbegriffe aus der Modelltheorie und der Mengenlehre, wie Struktur, (freie) Worthalbgruppe, geordnetes Paar, Kreuzprodukt, rekursive Funktion, rekursive Relation vorausgesetzt. Auch hierbei wird weitgehend der Darstellung in [16] gefolgt.

Definition 1 .
Seien I, J und K Indexmengen. Sei A eine Menge und $f_i (i \in I)$ Funktionen auf A, $S_j (j \in J)$ Relationen auf A und $a_k (k \in K)$ Konstanten in A, so heißt die Zusammenstellung $\mathcal{A} = \langle A;\ f_{i \in I};\ S_{j \in J};\ a_{k \in K} \rangle$ eine Struktur.

Für viele Anwendungen einer Struktur ist die Menge A von komplexer Gestalt. Ebenso sind es auch die Funktionen, Relationen und Konstanten. Für die praktische Handhabung ist es sinnvoll, die Funktionen, Relationen und Konstanten auf Teilmengen beziehungsweise Trägermengen von A zu definieren. Dafür werden mehrsortige Strukturen eingeführt. Als *Sorte* wird eine Klasse von Objekten (Termen) bezeichnet. Die Sorte jedes Objektes ist eindeutig bestimmt. Die Mengen der Objekte gleicher Sorte sind Äquivalenzklassen auf A. Die dazugehörige Äquivalenzrelation ist die *Typgleichheit*. Dadurch wird es notwendig, die Funktionen, Relationen und Konstanten mit ihren Sorten zu deklarieren. Die Sortenidentifikation ist möglich als $x \in A_s$ (*x ist Element der Menge A_s*) oder als $x : s$ (*x ist von der Sorte s*). Oftmals werden die Menge von Elementen einer Sorte und diese Sorte gleich bezeichnet. Der Übergang zu mehrsortigen Strukturen wird entsprechend der Darstellung in [17] im folgenden beschrieben.

Definition 2 .
Sei S eine (nichtleere) Menge von Sorten.

1. Eine Menge A heißt S-sortig, wenn $A = (A_s | s \in S)$.

2. Sie ist endlich, wenn die disjunkte Vereinigung über alle $A_s (s \in S)$ endlich ist.

3. Seien A und B zwei S-sortige Mengen, dann ist A Teilmenge von B ($A \subseteq B$), wenn $A_s \subseteq B_s$ für alle $s \in S$.

Definition 3 .
Sei $(W(S), *)$ die Worthalbgruppe über S.

1. $A^s := A_s$ und $A^{w*s} := A^w \times A^s$ für $s \in S$ und $w \in W(S)$.
2. Eine Funktion f auf A mit Deklaration $[w \to s]$ ist definiert als Funktion mit Definitionsbereich A^w und Wertebereich A^s.
3. Eine Relation S auf A mit Deklaration $[w]$ ist definiert als Relation auf A^w.
4. Eine Konstante a in A mit Deklaration $[s]$ ist definiert als Konstante in A^s.

Die Worthalbgruppe über S ist frei, das heißt sie enthält keine nichttrivialen Gleichungen. Mit Σ wird die Menge der Deklarationen über A bezeichnet. Die Gleichheit wird jeweils für jede Sorte vorausgesetzt. Die einzelnen Gleichheitsrelationen werden in ihrer Schreibweise identifiziert. Im weiteren werden die (mehrsortige) Menge A und die Sortenmenge S nicht explizit angegeben, sondern nur ihre Trägermengen $A_s (s \in S)$.

Definition 4 .
Seien $A = (A_s | s \in S)$ eine mehrsortige Menge und $f_i (i \in I)$ Funktionen mit Deklarationen σ_i auf A, $S_j (j \in J)$ Relationen mit Deklarationen σ_j auf A und $a_k (k \in K)$ Konstanten mit Deklarationen σ_k in A, wobei die Deklarationen für Funktionen, Relationen und Konstanten $\sigma_{i \in I}, \sigma_{j \in J}, \sigma_{k \in K} \in \Sigma$ jeweils dem Muster der vorhergehenden Definition entsprechen. Dann heißt die Zusammenstellung $\mathcal{A} = \langle A_{s \in S};\ f_{i \in I}[\sigma_i];\ S_{j \in J}[\sigma_j];\ a_{k \in K}[\sigma_k] \rangle$ eine mehrsortige Struktur.

Bemerkung 1 .
Sei $\mathcal{A}_1 = \langle A_{s \in S_1};\ f_{i \in I_1}[\sigma_i];\ S_{j \in J_1}[\sigma_j];\ a_{k \in K_1}[\sigma_k] \rangle$ eine mehrsortige Struktur und gelte gleiches auch für $\mathcal{A}_2 = \langle A_{s \in S_2};\ f_{i \in I_2}[\sigma_i];\ S_{j \in J_2}[\sigma_j];\ a_{k \in K_2}[\sigma_k] \rangle$. Gelten $S_1 \subseteq S_2$, $I_1 \subseteq I_2$, $J_1 \subseteq J_2$ und $K_1 \subseteq K_2$, so kann die Struktur $\mathcal{A}_2$ auch in der folgenden vereinfachten Form angegeben werden:

$$\mathcal{A}_2 = \langle \mathcal{A}_1;\ A_{s \in (S_2 \setminus S_1)};\ f_{i \in (I_2 \setminus I_1)}[\sigma_i];\ S_{j \in (J_2 \setminus J_1)}[\sigma_j];\ a_{k \in (K_2 \setminus K_1)}[\sigma_k] \rangle$$

Seien im folgenden noch einige Anmerkungen zum Kreuzprodukt und seiner Darstellung gegeben. Das Kreuzprodukt oder auch kartesische Produkt der Mengen $A_1, \ldots, A_n$ ist die Menge aller geordneten n-Tupel $(a_1, \ldots, a_n)$ mit $a_i \in A_i$ für alle $i = 1, \ldots, n$. Die Gleichheit auf Kreuzprodukten wird auf die komponentenweise Gleichheit zurückgeführt. Jedem Kreuzprodukt wird eine eigene Sorte zugeordnet.

Die bis jetzt gemachten Aussagen zum Kreuzprodukt sind mengentheoretischer Natur. Für die praktische Handhabung von n-Tupeln in der Prädikatenlogik ist es allerdings notwendig, jedem Kreuzprodukt eine Kompositionsfunktion für die Bildung und Darstellung der geordneten n-Tupel zuzuordnen. Diese bildet die $a_1, \ldots, a_n$ auf das n-Tupel $(a_1, \ldots, a_n)$ ab. Da die Gleichheit auf dem Kreuzprodukt äquivalent zur Gleichheit bezüglich der einzelnen Komponenten ist, muß die Kompositionsfunktion injektiv sein. Gleichzeitig werden zu jeder n-stelligen Kompositionsfunktion ihre n Projektionsfunktionen betrachtet.

Auf die sich daraus ergebenen Möglichkeiten der Beweisreduzierung durch eine Zerlegung in die einzelnen Projektionen wird im Abschnitt 3 eingegangen.

1.2 Algebraische Modellierung und Korrektheit

Die in diesem Bericht behandelte Aufgabe wurde aus *Algebraic Models and the Correctness of Microprocessors in Correct Hardware Design and Verification Methods* von Harman und Tucker [7] entnommen. In dem Artikel werden Maschinen auf verschiedenen Abstraktionsstufen betrachtet und algebraisch modelliert. Als solche Maschinen können u. a. Mikroprozessoren, Programmiersprachen und Computerprogramme aufgefaßt werden. Ziel ist es, Kriterien zu finden, um die Korrektheit von Maschinen verschiedener Stufen nachzuweisen. Korrektheit wird in dem Sinne gefordert, daß bei gleicher Eingabe gleiche Ergebnisse (unabhängig vom Berechnungsweg) erzielt werden. Da aufgrund der unterschiedlichen Abstraktionsstufen im allgemeinen unterschiedlich viele Rechenschritte ausgeführt werden müssen, ist eine Zeitrelativierung, das sogenannte *Retiming*, notwendig.

Unter einer Maschine wird ein (abstrakter) Automat mit einer Zustandsmenge und einer Überführungsfunktion verstanden. Die Überführungsfunktion operiert von der Zustandsmenge in die Zustandsmenge. Es werden in diesem Bericht nur deterministische Automaten betrachtet, die eine eindeutige Überführungsfunktion besitzen. In der Automatentheorie werden häufig neben der Zustandsmenge noch ein Eingabe- und ein Ausgabealphabet sowie eine Ausgabefunktion betrachtet. Bei den Betrachtungen in dieser Arbeit spielt die Ein- und Ausgabe keine Rolle. Es wird angenommen, daß keine Ausgabe und nur zu Beginn der Arbeit eine Eingabe erfolgt. Diese Eingabe fließt gleich in den Anfangszustand mit ein. Sie wird nicht explizit betrachtet.

Da Maschinen sehr komplex aufgebaut seien können, ist es ratsam, sie durch mehrsortige Strukturen darzustellen. Außerdem ist es üblich, die Überführungsfunktion durch Komposition der *partiellen* Überführungsfunktionen bezüglich der einzelnen Trägermengen zu definieren. Dieses Vorgehen wird sich später bei der Theorieauswahl als nützlich erweisen.

Seien $\mathcal{A}$ und $\mathcal{B}$ die algebraischen Modelle zweier Maschinen mit den Zustandsmengen $A = A_1 \times \cdots \times A_n$ und $B = B_1 \times \cdots \times B_m$. Die Trägermengen A_i können von unterschiedlicher Sorte sein. Analoges gilt für die Mengen B_j. Die iterativen Abbildungen auf den Kreuzprodukten A und B, die Einschritt-Überführungsfunktionen, seien f und g. Diese werden durch die Funktionen $f_{i=1,\ldots,n} : A \to A_i$ und $g_{j=1,\ldots,m} : B \to B_j$ wie folgt definiert. Die Funktionen ak und bk sind die jeweiligen Kompositionsfunktionen. ap_i und bp_i sind die zugehörigen Projektionsfunktionen.

$$f(a) := ak(f_1(a), \ldots, f_n(a))$$

und

$$g(b) := bk(g_1(b), \ldots, g_m(b))$$

für alle $a \in A$ und $b \in B$. Die Funktionen $f_1, \ldots, f_n$ und $g_1, \ldots, g_m$ werden als die partiellen Einschritt-Überführungsfunktionen bezeichnet.

Damit werden die Maschinen durch folgende Strukturen beschrieben:

$$\mathcal{A} = \langle A, A_1, \ldots, A_n; f[A \to A], f_{j=1,\ldots,n}[A \to A_i], ak[A_1 \times \cdots \times A_n \to A], ap_{i=1,\ldots,n}[A \to A_i] \rangle$$

$$\mathcal{B} = \langle B, B_1, \ldots, B_m; g[B \to B], g_{j=1,\ldots,m}[B \to B_j], bk[B_1 \times \cdots \times B_m \to B], bp_{j=1,\ldots,m}[B \to B_j] \rangle$$

Von Interesse ist auch die Arbeit einer Maschine über einen längeren Zeitraum. Die Arbeit erfolgt in Takten. Damit wird die Zeit durch eine diskrete Zeitskala charakterisiert. Folglich ist die Analyse des Zustandes der Maschine nach mehreren Arbeitsschritten bei gegebenem Ausgangszustand erforderlich. Dazu werden die Strukturen $\mathcal{A}$ beziehungsweise $\mathcal{B}$ zu $\mathcal{A}^*$ beziehungsweise $\mathcal{B}^*$ erweitert, indem diskrete Zeitskalen T beziehungsweise S eingeführt werden.

$$\mathcal{A}^* = \langle \mathcal{A};\ T;\ F[A \times T \to A],\ F_{j=1,\ldots,n}[A \times T \to A_j],\ +1[T \to T];\ 0[T] \rangle$$

$$\mathcal{B}^* = \langle \mathcal{B};\ S;\ G[B \times S \to B],\ G_{i=1,\ldots,m}[B \times S \to B_i],\ +1[S \to S];\ 0[S] \rangle$$

Die mehrfache Deklaration der Operatoren +1 und 0 auf den verschiedenen Mengen T und S wird dadurch gerechtfertigt, daß die beiden diskreten Zeitskalen isomorph zu den natürlichen Zahlen sind. Sie werden auch nur in diesem Sinne benutzt. Die Operatoren auf ihnen werden als auf den natürlichen Zahlen deklariert betrachtet und nicht unterschieden. Der Operator +1 kann als Suffix-Operator angesehen werden und ermöglicht die rekursive Definition über den Zeitskalen. In der praktischen Anwendung wird die kanonische Addition der natürlichen Zahlen zugrunde gelegt, wobei die zweite Argumentstelle als mit 1 fixiert betrachtet wird.

Die *Mehrschritt-Überführungsfunktionen* F und G werden wie folgt rekursiv definiert:

$$F(a,0) := a,\ F(a,t+1) := f(F(a,t))$$

$$G(b,0) := b,\ G(b,s+1) := g(G(b,s))$$

für alle $a \in A$, $b \in B$, $t \in T$ und $s \in S$.

Zusätzlich werden für die Maschine $\mathcal{B}$ die partiellen Mehrschritt-Überführungsfunktionen $G_{i=1,\ldots,m}$ wie folgt rekursiv definiert:

$$G_i(b,0) := bp_i(b),\ G_i(b,s+1) := g_i(bk(G_1(b,s),\ldots,G_m(b,s)))$$

für alle $i = 1,\ldots,m$, $b \in B$ und $s \in S$.

Die partiellen Mehrschritt-Überführungsfunktionen werden hier nur für die Maschine $\mathcal{B}$ angegeben. Die Definition für $\mathcal{A}$ geschieht analog. Es gilt dann der folgende Satz.

Satz 1 .
Für alle $b \in B$ und für alle $s \in S$ gilt $G(b,s) = bk(G_1(b,s),\ldots,G_m(b,s))$.

Beweis: Induktion über s.

Induktionsanfang: Für alle $b \in B : G(b,0) = bk(G_1(b,0),\ldots,G_m(b,0))$. Da $G(b,0) = b$ und $G_i(b,0) = bp_i(b)$ für alle $i = 1,\ldots,m$ ist, ist obige Formel äquivalent zur Projektionseigenschaft $b = bk(bp_1(b),\ldots,bp_m(b))$.

Induktionsschritt: Gelte für beliebiges $s \in S$ sowie für alle $b \in B$ die Gleichung $G(b,s)$=$bk(G_1(b,s),\ldots,G_m(b,s)$. Nach Definition und Induktionsvoraussetzung gilt $G_i(b,s+1) = g_i(bk(G_1(b,s),\ldots,G_m(b,s))) = g_i(G(b,s))$. Aus $G(b,s+1) = g(G(b,s))$=$bk(g_1(G(b,s)),\ldots,g_m(G(b,s)))$ folgt dann die Induktionsbehauptung $G(b,s+1)$=$bk(G_1(b,s+1),\ldots,G_m(b,s+1))$. □

Im nun folgenden geht es um die Beziehung zwischen beiden Strukturen. Beide Maschinen sind von unterschiedlichem Abstraktionsniveau. Ohne Beeinträchtigung der Allgemeingültigkeit nehmen wir an, daß $\mathcal{A}$ von höherem Abstraktionsniveau als $\mathcal{B}$ ist. Damit ist die Menge B von komplizierterer Gestalt als A. Deshalb müssen im allgemeinen für jeden Befehl in $\mathcal{A}$ mehrere Befehle in $\mathcal{B}$ ausgeführt werden.

Zunächst werden äquivalente Zustände charakterisiert. Das geschieht durch die beiden Abbildungen $\phi : A \to B$ und $\psi : B \to A$. Den Zuständen aus A werden äquivalente Zustände in B zugeordnet und umgekehrt. Mit der Annahme des größeren Abstraktionsniveaus für $\mathcal{A}$ muß die Abbildung ϕ nicht eindeutig sein. Im allgemeinen wird jeder Zustand in B durch zusätzliche Daten charakterisiert. Diese können beim Start des Systems beliebig sein. Die zusätzlichen Daten entfallen bei Anwendung der Projektion ψ. Für die Komposition beider Abbildungen $\psi \circ \phi$ gilt: $\forall a \in A : \psi(\phi(a)) = a$.

Danach werden die beiden Zeitskalen T und S mittels eines *Retimings* in Beziehung gesetzt. Dabei können die Uhren T und S unterschiedliche Geschwindigkeiten aufweisen. Auch müssen nicht alle Zeitzyklen gleich lang sein, das heißt die Uhren können irregulär sein. Ein Retiming $\lambda : S \to T$ ist eine surjektive, monotone Abbildung. Die dazugehörende *Immersion* $\bar{\lambda} : T \to S$ ist wie folgt definiert: $\bar{\lambda}(t) := (\mu s \in S)[\lambda(s) = t]$.

Sei A eine Zustandsmenge, so daß die Anzahl der Zyklen, die die Maschine $\mathcal{B}$ durchlaufen muß, um auf das äquivalente Ergebnis der Maschine $\mathcal{A}$ nach einem Zyklus zu kommen, vom Startzustand abhängig ist. Dann ist $\lambda : A \times S \to T$ ein statusabhängiges Retiming, wenn es für alle $a \in A$ surjektiv und monoton bezüglich s ist. Das bedeutet, daß aus $s_1 > s_2$ für alle $s_1, s_2 \in S$ und für alle $a \in A$ die Aussage $\lambda(a, s_1) \geq \lambda(a, s_2)$ folgt. Analog zum nicht statusabhängigen Fall wird die Immersion $\bar{\lambda} : A \times T \to S$ definiert, als $\bar{\lambda}(a,t) := (\mu s \in S)[\lambda(a,s) = t]$ für alle $a \in A$.

Daraus ergibt sich folgende Struktur $\mathcal{M}$ für diese mathematische Problemstellung:

$$\mathcal{M} = \langle \mathcal{A}^*,\ \mathcal{B}^*;\ \phi[A \to B],\ \psi[B \to A],\ \lambda[A \times S \to T],\ \bar{\lambda}[A \times T \to S] \rangle$$

Mittels der so eingeführten Funktionen kann die Korrektheit im eingangs beschriebenen Sinne mathematisch folgendermaßen beschrieben werden.

Korrektheitsbedingung: Die Funktion G (auf B) simuliert die Funktion F (auf A) bgzl. ϕ und ψ, wenn ein (statusabhängiges) Retiming mit $\bar{\lambda} : A \times T \to S$ existiert, so daß das folgende Diagramm (Abbildung 1) für alle $a \in A$ und $t \in T$ kommutiert.

Äquivalent dazu ist die folgende Formel:

$$\forall a \in A\ \forall t \in T : F(a,t) = \psi(G(\phi(a), \bar{\lambda}(a,t))).$$

Da die Retimingfunktion λ, die der obigen Korrektheitsbedingung genügt, im allgemeinen nur kompliziert dargestellt werden kann, ist ihre Existenz allgemein nur schwer konstruktiv nachweisbar. Wir geben nun ein mögliches Kriterium an

Abbildung 1:

$$\begin{tikzcd}
A \times T \arrow[r, "F"] \arrow[d, "(\phi, \bar{\lambda})"] & A \\
B \times S \arrow[r, "G"] & B \arrow[u, "\psi"']
\end{tikzcd}$$

ein solches λ an. Dieses sollte gleichzeitig praktisch anwendbar für die Verifikation sein. Der folgende Satz gibt ein solches Kriterium, welches dadurch charakterisiert ist, daß jedem Zustand der Menge A eine positive Anzahl von Arbeitsschritten bezüglich Maschine $\mathcal{B}$ zugeordnet wird.

Satz 2 .
Sei $l : A \to I\!N^+$ eine beliebige Funktion. Durch eine solche Funktion l werden die Funktionen $\bar{\lambda} : A \times T \to S$ und $\lambda : A \times S \to T$ auf die folgende Weise definiert:

1. $\forall x \in A : \bar{\lambda}(x, 0) := 0$
2. $\forall x \in A\ \forall t \in T : \bar{\lambda}(x, t+1) := \bar{\lambda}(x, t) + l(F(x, t))$
3. $\forall x \in A\ \forall s \in S : \lambda(x, s) := (\mu t \in T)[\bar{\lambda}(x, t+1) > s]$

Dann ist λ ein Retiming und es gilt: $\bar{\lambda}(x, t) = (\mu s \in S)[\lambda(x, s) = t]\ \forall x \in A\ \forall t \in T$.

Beweis: Zunächst wird gezeigt, daß die Funktion λ total ist. Da die Funktion l auf die positiven natürlichen Zahlen abbildet, ist die Funktion $\bar{\lambda}$ nach (2) streng monoton wachsend bezüglich t. Somit ist in (3) λ stets definiert.

Als nächstes wird gezeigt, daß λ surjektiv ist. Sei $t \in T$ beliebig. Nach (1) und (2) existiert $\bar{\lambda}(x, t)$ für alle $t \in T$ und ist streng monoton in t. Sei $s := \bar{\lambda}(x, t)$. Daraus folgt nach (3), daß $\lambda(x, s) = (\mu u \in T)[\bar{\lambda}(x, u+1) > \bar{\lambda}(x, t)]$. Aufgrund der strengen Monotonie von $\bar{\lambda}$ ist $(\mu u \in T)[\bar{\lambda}(x, u+1) > \bar{\lambda}(x, t)] = t$ und damit $\lambda(x, s) = t$.

Weiterhin ist λ monoton in t. Seien s_1 und s_2 aus S mit $s_1 > s_2$. Seien $t_1 := (\mu t \in T)[\bar{\lambda}(x, t+1) > s_1]$ und $t_2 := (\mu t \in T)[\bar{\lambda}(x, t+1) > s_2]$. Somit ist $\bar{\lambda}(x, t_1+1) > s_1$

und folglich auch $\bar{\lambda}(x, t_1+1) > s_2$. Dies bedeutet, daß $t_2 \leq t_1$. Da $\lambda(x, s_1) = t_1$ und $\lambda(x, s_2) = t_2$, gilt die Monotonie $s_1 > s_2 \to \lambda(x, s_1) \geq \lambda(x, s_2)$.

Noch zu zeigen ist die Gleichheit $\bar{\lambda}(x,t) = (\mu s \in S)[\lambda(x,s) = t]$ für alle $t \in T$. Sei $t \in T$ beliebig, aber fest. Dann ist, wie oben bereits bewiesen, $\lambda(x, \bar{\lambda}(x,t)) = t$; also $(\mu s \in S)[\lambda(x,s) = t] \leq \bar{\lambda}(x,t)$. Wenn $t = 0$, dann gilt nach (1), daß $\bar{\lambda}(x,t) = 0$, und somit trivialer Weise $(\mu s \in S)[\lambda(x,s) = t] = \bar{\lambda}(x,t)$. Wenn $t > 0$, dann gilt aufgrund der strengen Monotonie von $\bar{\lambda}$, daß $\bar{\lambda}(x,t) > 0$. Dann ist $\bar{\lambda}(x,t) > \bar{\lambda}(x,t) - 1$ (≥ 0). Folglich gilt $(\mu u \in T)[\bar{\lambda}(x, u+1) > \bar{\lambda}(x,t) - 1] \leq t - 1$ $(< t)$. Nach (3) bedeutet dies $\lambda(x, \bar{\lambda}(x,t) - 1) < t$ und da λ monoton ist, daß $s = \bar{\lambda}(x,t)$ minimal für $\lambda(x,s) = t$. Damit ist gezeigt, daß $(\mu s \in S)[\lambda(x,s) = t] = \bar{\lambda}(x,t)$. □

Bemerkung 2 .
Sind die Funktionen l und F rekursiv, so ist es nach dem Rekursionsschema auch die Funktion $\bar{\lambda}$. Da $+$ und $>$ rekursiv sind, folgt nach der Anwendung des μ-Operators, daß auch λ rekursiv ist. Damit folgt aus der Rekursivität der Überführungsfunktionen $f_1, \ldots, f_n$, daß das Retiming rekursiv ist, da F mittels Überlagerung und Rekursionsschema aus $f_1, \ldots, f_n$ gewonnen wurde. Zur Definition rekursiver Funktionen siehe auch [11].

1.3 Die Beweisaufgabe

In diesem Abschnitt wird ein Beispiel zweier Maschinen betrachtet, deren Korrektheit im erläuterten Sinne nachgewiesen werden soll. Der Zustandsraum beider Maschinen besteht jeweils aus 3 Speichern, in denen natürliche Zahlen enthalten sind. Im folgenden wird der Zustandsraum als geordnetes Tripel natürlicher Zahlen betrachtet. Bei Maschine $\mathcal{A}$ wird der Speicherinhalt des dritten Speichers (sofern grösser Null) zum Inhalt des ersten addiert und der dritte Speicher auf Null gesetzt. Danach wird der Speicherinhalt des zweiten Speichers (sofern grösser Null) zum Inhalt des ersten addiert und der zweite Speicher auf Null gesetzt. Ist sowohl der Inhalt des zweiten als auch des dritten Speichers gleich Null, wird der Zustand der Maschine nicht geändert. Bei Maschine $\mathcal{B}$ wird der Speicherinhalt des dritten Speichers (sofern grösser Null) dekrementiert und der Inhalt des ersten Speichers inkrementiert. Dieses wird so oft wiederholt bis der Inhalt des dritten Speichers gleich Null ist. Dann wird der Speicherinhalt des zweiten Speichers (sofern grösser Null) dekrementiert und der Inhalt des ersten Speichers inkrementiert. Auch dieses wird wiederholt bis der Inhalt des zweiten Speichers gleich Null ist. Ebenso ändert sich der Zustand der Maschine nicht, wenn sowohl der Inhalt des zweiten als auch des dritten Speichers gleich Null ist.

Da beide Maschinen die gleiche Zustandsmenge $I\!N^3$ besitzen, vereinfachen sich die Beziehungen zwischen beiden Maschinen entsprechend. Die Abbildungen ϕ und ψ entsprechen der Identität auf $I\!N^3$. Sie werden nicht explizit angegeben.

Auch die Kompositions- und Projektionsfunktionen werden nicht unterschieden. Die Beschreibung des Problems erfolgt in der Struktur:

$$\mathcal{M} = \langle \mathcal{A}^*,\ \mathcal{B}^*;\ l[I\!N^3 \to I\!N],\ \bar{\lambda}[I\!N^3 \times I\!N \to I\!N] \rangle.$$

Zustandsgruppen. Bei der Abarbeitung werden vier Gruppen von Zuständen unterschieden:

$Z_1 = \{(x_1, x_2, x_3) \in I\!N^3 \mid x_2 > 0 \wedge x_3 > 0\}$
$Z_2 = \{(x_1, x_2, x_3) \in I\!N^3 \mid x_2 = 0 \wedge x_3 > 0\}$
$Z_3 = \{(x_1, x_2, x_3) \in I\!N^3 \mid x_2 > 0 \wedge x_3 = 0\}$
$Z_4 = \{(x_1, x_2, x_3) \in I\!N^3 \mid x_2 = 0 \wedge x_3 = 0\}.$

Diese Fallunterscheidung ist vollständig über $I\!N^3$ und wird der Definition der partiellen Überführungsfunktionen zugrunde gelegt.

Überführungsfunktionen. Die Mehrschritt-Überführungsfunktionen F und G sind rekursiv, wie oben angegeben, definiert.

$\forall x \in I\!N^3 : F(x, 0) = x$
$ax(F, ra, rek\_def, Rekursionsanfang)$
$\forall x \in I\!N^3\ \forall t \in I\!N : F(x, t+1) = f(F(x, t))$
$ax(F, rs, rek\_def, Rekursionsschritt)$
$\forall x \in I\!N^3 : G(x, 0) = x$
$ax(G, ra, rek\_def, Rekursionsanfang)$
$\forall x \in I\!N^3\ \forall t \in I\!N : G(x, t+1) = g(G(x, t))$
$ax(G, rs, rek\_def, Rekursionsschritt)$
$\forall x \in I\!N^3 : G_{i=1,2,3,}(x, 0) := p_i(x)$
$ax(G_{i=1,2,3}, ra, rek\_def, Rekursionsanfang)$
$\forall x \in I\!N^3\ \forall t \in I\!N : G_{i=1,2,3}(x, t+1) := g_i(k(G_1(x, t), G_2(x, t), G_3(x, t)))$
$ax(G_{i=1,2,3}, rs, rek\_def, Rekursionsschritt)$

Die Strukturen $\mathcal{A}^*$ und $\mathcal{B}^*$ haben folgende Gestalt, wobei die Zeitskalen und ihre Operatoren durch die Struktur $\mathcal{N}$ dargestellt werden:

$$\mathcal{A}^* = \langle \mathcal{A};\ F[I\!N^3 \times I\!N \to I\!N^3] \rangle$$

$$\mathcal{B}^* = \langle \mathcal{B};\ G[I\!N^3 \times I\!N \to I\!N^3],\ G_{i=1,2,3}[I\!N^3 \times I\!N \to I\!N] \rangle.$$

Die Überführungsfunktionen f und g sind mittels der Kompositionsfunktion $k : I\!N \times I\!N \times I\!N \to I\!N^3$ und den partiellen Überführungsfunktionen definiert:

$\forall x \in I\!N^3 : f(x) = k(f_1(x), f_2(x), f_3(x))$
$ax(f, k, def, Komposition)$

$\forall x \in \mathbb{N}^3 : g(x) = k(g_1(x), g_2(x), g_3(x))$
$ax(g, k, def, Komposition).$

Die partiellen Überführungsfunktionen haben folgende Gestalt:

$\forall x \in \mathbb{N}^3 : p_3(x) > 0 \rightarrow f_1(x) = p_1(x) + p_3(x)$ $\{Z_1, Z_2\}$
$ax(f_1, ?, def\_fu, Definition\ 1)$
$\forall x \in \mathbb{N}^3 : p_3(x) = 0 \wedge p_2(x) > 0 \rightarrow f_1(x) = p_1(x) + p_2(x)$ $\{Z_3\}$
$ax(f_1, ?, def\_fu, Definition\ 2)$
$\forall x \in \mathbb{N}^3 : p_3(x) = 0 \wedge p_2(x) = 0 \rightarrow f_1(x) = p_1(x)$ $\{Z_4\}$
$ax(f_1, ?, def\_fu, Definition\ 3)$

$\forall x \in \mathbb{N}^3 : p_3(x) > 0 \rightarrow f_2(x) = p_2(x)$ $\{Z_1, Z_2\}$
$ax(f_2, ?, def\_fu, Definition\ 1)$
$\forall x \in \mathbb{N}^3 : p_3(x) = 0 \wedge p_2(x) > 0 \rightarrow f_2(x) = 0$ $\{Z_3\}$
$ax(f_2, ?, def\_fu, Definition\ 2)$
$\forall x \in \mathbb{N}^3 : p_3(x) = 0 \wedge p_2(x) = 0 \rightarrow f_2(x) = p_2(x)$ $\{Z_4\}$
$ax(f_2, ?, def\_fu, Definition\ 3)$

$\forall x \in \mathbb{N}^3 : p_3(x) > 0 \rightarrow f_3(x) = 0$ $\{Z_1, Z_2\}$
$ax(f_3, ?, def\_fu, Definition\ 1)$
$\forall x \in \mathbb{N}^3 : p_3(x) = 0 \rightarrow f_3(x) = p_3(x)$ $\{Z_3, Z_4\}$
$ax(f_3, ?, def\_fu, Definition\ 2)$

und

$\forall x \in \mathbb{N}^3 : p_3(x) \rightarrow g_1(x) = inc(p_1(x)$ $\{Z_1, Z_2\}$
$ax(g_1, ?, def\_fu, Definition\ 1)$
$\forall x \in \mathbb{N}^3 : p_3(x) = 0 \wedge p_2(x) > 0 \rightarrow g_1(x) = inc(p_1(x)$ $\{Z_3\}$
$ax(g_1, ?, def\_fu, Definition\ 2)$
$\forall x \in \mathbb{N}^3 : p_3(x) = 0 \wedge p_2(x) = 0 \rightarrow g_1(x) = p_1(x)$ $\{Z_4\}$
$ax(g_1, ?, def\_fu, Definition\ 3)$

$\forall x \in \mathbb{N}^3 : p_3(x) > 0 \rightarrow g_2(x) = p_2(x)$ $\{Z_1, Z_2\}$
$ax(g_2, ?, def\_fu, Definition\ 1)$
$\forall x \in \mathbb{N}^3 : p_3(x) = 0 \wedge p_2(x) > 0 \rightarrow g_2(x) = dec(p_2(x))$ $\{Z_3\}$
$ax(g_2, ?, def\_fu, Definition\ 2)$
$\forall x \in \mathbb{N}^3 : p_3(x) = 0 \wedge p_2(x) = 0 \rightarrow g_2(x) = p_2(x)$ $\{Z_4\}$
$ax(g_2, ?, def\_fu, Definition\ 3)$

$\forall x \in \mathbb{N}^3 : p_3(x) > 0 \rightarrow g_3(x) = dec(p_3(x))$ $\{Z_1, Z_2\}$
$ax(g_3, ?, def\_fu, Definition\ 1)$
$\forall x \in \mathbb{N}^3 : p_3(x) = 0 \rightarrow g_3(x) = p_3(x)$ $\{Z_3, Z_4\}$
$ax(g_3, ?, def\_fu, Definition\ 2).$

Bei den auf Fallunterscheidung basierenden Definitionen wurden am Ende jeder Zeile jeweils die Mengen der zugehörigen Zustandsgruppen angegeben. Jede dieser partiellen Überführungsfunktionen könnte auch als ein Axiom dargestellt werden,

Abbildung 2: Maschine $\mathcal{A}$

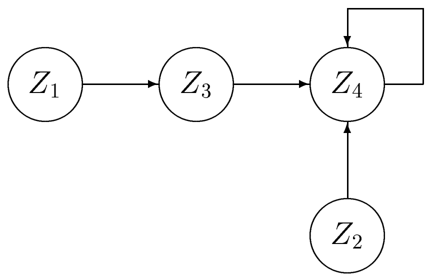
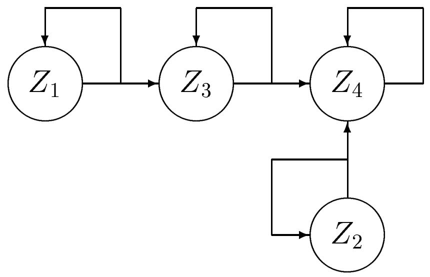


Abbildung 3: Maschine $\mathcal{B}$



in dem die einzelnen Fälle konjunktiv verbunden werden. Die beiden folgenden Graphen (Abbildungen 2 und 3) zeigen die möglichen Übergänge zwischen den einzelnen Zustandsgruppen.

Die beiden Maschinen werden durch die zwei folgenden mehrsortige Strukturen beschrieben.

$$\mathcal{A} = \langle \mathcal{N}^3;\ f[I\!N^3 \to I\!N^3],\ f_{i=1,2,3}[I\!N^3 \to I\!N] \rangle$$

und

$$\mathcal{B} = \langle \mathcal{N}^3;\ g[I\!N^3 \to I\!N^3],\ g_{i=1,2,3}[I\!N^3 \to I\!N] \rangle$$

Die Strukturen des dreifachen Kreuzproduktes der natürlichen Zahlen und der natürlichen Zahlen selbst ist wie folgt definiert.

$$\mathcal{N}^3 = \langle \mathcal{N};\ I\!N^3;\ k[I\!N \times I\!N \times I\!N \to I\!N^3],\ p_{i=1,2,3}[I\!N^3 \to I\!N] \rangle$$

$$\mathcal{N} = \langle I\!N;\ +,\ -,\ inc,\ dec;\ >,\ <,\ \geq,\ \leq;\ 0,\ 1 \rangle$$

Komposition und Projektion. Wir geben nun die Funktionen und Relationen der Struktur $\mathcal{N}$ sowie die Kompositions- und Projektionsfunktionen an. Der Zusammenhang zwischen der Komposition und der Projektion wird durch folgende vier Axiome beschrieben:

$\forall x_1 \in I\!N\ \forall x_2 \in I\!N\ \forall x_3 \in I\!N : p_1(k(x_1, x_2, x_3)) = x_1$
$ax(pr, p_1, def, Projektion\ 1)$
$\forall x_1 \in I\!N\ \forall x_2 \in I\!N\ \forall x_3 \in I\!N : p_2(k(x_1, x_2, x_3)) = x_2$
$ax(pr, p_2, def, Projektion\ 2)$
$\forall x_1 \in I\!N\ \forall x_2 \in I\!N\ \forall x_3 \in I\!N : p_3(k(x_1, x_2, x_3)) = x_3$
$ax(pr, p_3, def, Projektion\ 3)$
$\forall x \in I\!N^3 : k(p_1(x), p_2(x), p_3(x)) = x$
$ax(pr, k, def, Komposition)$.

Retiming. In den theoretischen Betrachtungen wurde die Existenz eines (statusabhängigen) Retimings λ auf die Existenz einer Funktion $l : A \to I\!N^+$ zurückgeführt. Im folgenden wird nun ein solches $l : I\!N^3 \to I\!N^+$ für die Beispielaufgabe konkret angegeben:

$\forall x \in I\!N^3 : p_3(x) > 0 \to l(x) = p_3(x)$ $\{Z_1, Z_2\}$
$ax(l, ?, def\_fu, Definition\ 1)$
$\forall x \in I\!N^3 : p_3(x) = 0 \wedge p_2(x) > 0 \to l(x) = p_2(x)$ $\{Z_3\}$
$ax(l, ?, def\_fu, Definition\ 2)$
$\forall x \in I\!N^3 : p_3(x) = 0 \wedge p_2(x) = 0 \to l(x) = 1$ $\{Z_4\}$
$ax(l, ?, def\_fu, Definition\ 3)$.

Auch hierbei spielen die vier Zustandsgruppen wieder eine wesentliche Rolle bei der Unterscheidung. Das Retiming λ und seine Immersion $\bar{\lambda}$ sind damit eindeutig festgelegt. Für die Funktion $\bar{\lambda} : I\!N^3 \times I\!N \to I\!N$ werden dabei folgende Axiome mit in die Theorie aufgenommen:

$\forall x \in I\!N^3 : \bar{\lambda}(x, 0) = 0$
$ax(\bar{\lambda}, ra, rek\_def, Rekursionsanfang)$
$\forall x \in I\!N^3\ \forall t \in I\!N : \bar{\lambda}(x, t+1) = \bar{\lambda}(x, t) + l(F(x, t))$
$ax(\bar{\lambda}, rs, rek\_def, Rekursionsschritt)$.

Natürliche Zahlen. Zum Abschluß der Beschreibung dieses Beispiels werden noch die für die Aufgabe notwendigen Funktionen und Relationen auf den natürlichen Zahlen charkterisiert:

- Addition als kommutatives Monoid.

  $\forall x \in I\!N : x + 0 = x$
  $ax(+, ?, eig, neutrales\ Element)$

$\forall x \in I\!N \ \forall y \in I\!N : x + y = y + x$
$ax(+, ?, eig, kommutativ)$
$\forall x \in I\!N \ \forall y \in I\!N \ \forall z \in I\!N : x + (y + z) = (x + y) + z$
$ax(+, ?, eig, assoziativ)$

- Subtraktion.

  $\forall x \in I\!N : x - 0 = x$
  $ax(-, ?, eig, rechtsneutrales\ Element)$
  $\forall x \in I\!N \ x - x = 0$
  $ax(-, ?, eig, gleich)$
  $\forall x \in I\!N \ \forall y \in I\!N : x > y \rightarrow x - y > 0$
  $ax(-, ?, eig, größer)$
  $\forall x \in I\!N \ \forall y \in I\!N \ \forall z \in I\!N : (x - y) - z = x - (y + z)$
  $ax(-, ?, eig, nacheinander)$

- Die Funktion `inc` und die partielle Funktion `dec`.

  $\forall x \in I\!N : inc(x) = x + 1$
  $ax(inc, ?, def, Definition)$
  $\forall x \in I\!N : x > 0 \rightarrow dec(x) = x - 1$
  $ax(dec, ?, part\_def, Definition)$

- $>$-Relation als irreflexive Ordnung mit kleinstem Element, Beziehungen zu weiteren Relationen.

  $1 > 0$
  $ax(gr, >, def, Fakt(1 > 0))$
  $\forall x \in I\!N : x = 0 \lor x > 0$
  $ax(gr, >, eig, kleinstes\ Element)$
  $\forall x \in I\!N : \neg(x > x)$
  $ax(gr, >, eig, irreflexiv)$
  $\forall x \in I\!N \ \forall y \in I\!N \ \forall z \in I\!N : x > y \land y > z \rightarrow x > z$
  $ax(gr, >, eig, transitiv)$
  $\forall x \in I\!N \ \forall y \in I\!N : x > y \lor y > x \lor x = y$
  $ax(gr, >, eig, konnex)$

  $\forall x \in I\!N \ \forall y \in I\!N : x < y \rightarrow y > x$
  $ax(gr, <, def, Definition\ 1)$
  $\forall x \in I\!N \ \forall y \in I\!N : x < y \leftarrow y > x$
  $ax(gr, <, def, Definition\ 2)$
  $\forall x \in I\!N \ \forall y \in I\!N : x \geq y \rightarrow x > y \lor x = y$
  $ax(gr, \geq, def, Definition\ 1)$
  $\forall x \in I\!N \ \forall y \in I\!N : x \geq y \leftarrow x > y$
  $ax(gr, \geq, def, Definition\ 2)$
  $\forall x \in I\!N \ \forall y \in I\!N : x \geq y \leftarrow x = y$

$ax(gr, \geq, def, Definition\ 3)$
$\forall x \in I\!N\ \forall y \in I\!N : x \leq y \rightarrow y \geq x$
$ax(gr, \leq, def, Definition\ 1)$
$\forall x \in I\!N\ \forall y \in I\!N : x \leq y \leftarrow y \geq x$
$ax(gr, \leq, def, Definition\ 2)$

Weiterhin steht die PROLOG-Arithmetik für Zahlen zur Verfügung.

Lemmata. Aus diesen Axiomen lassen sich einige Lemmata beweisen, die mehrfach bei der Beweisführung benutzt werden. Auch diese Lemmata wurden automatisch mit Hilfe von Setheo bewiesen. Sie wurden ebenfalls in die Theorie aufgenommen:

$\forall x \in I\!N^3\ \forall t \in I\!N : G(x,t) = k(G_1(x,t), G_2(x,t), G_3(x,t)).$
$ax(G, k, lem, Komposition)$
$\forall x \in I\!N\ \forall y \in I\!N : x + 1 \leq y \rightarrow x < y$
$ax(gr, ?, lem, Lemma\ 1)$
$\forall x \in I\!N\ \forall y \in I\!N : x < y \rightarrow x \leq y$
$ax(gr, ?, lem, Lemma\ 2)$
$\forall x \in I\!N : x \leq x$
$ax(gr, ?, lem, Lemma\ 3).$

Aufgabe. In der Beispielaufgabe sind die Zustandsmengen gleich. Die Abbildungen ϕ und ψ entsprechen der Identität auf $I\!N^3$. Die zu verifizierende Aufgabe hat damit die folgende Form:

$$\forall x \in I\!N^3\ \forall t \in I\!N : F(x,t) = G(x, \bar{\lambda}(x,t)).$$

Der Beweis dieser Aufgabe ist nun unser weiteres Ziel. Dabei werden die im folgenden beschriebenen Systeme und Hilfsmittel angewandt. Die zu beweisende Formel wird auf einfachere Teilprobleme zurückgeführt. Hierbei werden die neu entstehendenen Teilprobleme und die Rechtfertigung für diesen Schritt angegeben. Dieser Prozeß wird rekursiv wiederholt, solange bis ein Teilproblem durch den automatischen Theorembeweiser SETHEO lösbar ist.

1.4 ILF

Das System ILF (Integrate Logical Functions) [1] integriert interaktive und automatische Theorembeweiser sowie gebietsspezifische Deduktionssysteme. Es stellt eine einheitliche grafische Oberfläche, eine Taktiksprache und eine natürlichsprachliche Beweisausgabe zur Verfügung. ILF verfügt über einen eigenen Klausel-Normalform-Algorithmus. Dieser baut bei allgemeinen prädikatenlogischen Ausdrücken die Quantoren ab und ersetzt die Variablen durch lokale Konstanten beziehungsweise durch Skolemfunktionen. Anschließend werden konjunktive Normalformen gebildet, und gegebenenfalls Konjunktionen getrennt.

Für die Beweisführung stehen verschiedene interaktive Deduktionssysteme, wie zum Beispiel `proofpad` ([1], Seite 44 ff) zur Verfügung. Damit können Taktiken auf die Beweisaufgabe(n) angewandt werden und der Beweis strukturiert werden. Solche Taktiken sind unter anderem direkte und indirekte Beweisführung, Induktion und Fallunterscheidung, und Einfügen von Lemmata. Dadurch werden die ursprünglichen Beweisaufgaben auf einfachere Aufgaben zurückgeführt. Gleichzeitg ist eine Konfiguration des oder der verwendeten automatischen Beweisers mit einer bestimmten Theorie möglich. Diese Theorieauswahl kann auch automatisch beziehungsweise halbautomatisch durchgeführt werden. Ein auf syntaktischer Formelanalyse beruhender Ansatz wird im Abschnitt 4 ausführlich betrachtet.

Ziel der natürlichsprachlichen Beweisdarstellung ist es, Beweise auszugeben, die ohne ILF-spezifisches Wissen verstanden werden können. Dazu werden die Beweise aufbereitet und anschliessend in einen LATEX-Script übertragen. Dieses kann sowohl für Beweise automatischer Theorembeweiser, als auch für Beweise des `proofpad` getan werden. Bei letzteren können automatisch generierte Teilbeweise eingebunden werden. Dieser Prozeß läuft in 4 Stufen ab, wobei die 1. Stufe nur für automatisch generierte Beweise notwendig ist. Die Stufen 3 und 4 können variert und wiederholt werden.

Die 1. Stufe (syntaktische Beweisunifikation) erstellt ein einheitliches Darstellungsformat des Beweises. Der Beweis kann dann als gerichteter azyklischer Graph aufgefaßt werden. In der 2. Stufe (semantische Beweisunifikation) wird dieser Beweis dann in Blokkalkül [2] transformiert. Die 3. Stufe (strukturelle Beweistransformation) versucht charakteristische Strukturen im Beweis zu finden und entsprechend umzuformen. Dazu gehören Lemmagenerierung, Weglassung von „Trivialitäten", Löschung von Duplikaten u. weiteres. In der 4. Stufe (Generierung der Beweisdarstellung) werden die logischen Konstruktionen durch natürlichsprachliche Formulierungen ersetzt. Ausserdem können nutzerdefinierte Darstellungen von Formeln erzeugt werden. Auch ist hierbei eine typengerechte Darstellung möglich.

1.5 SETHEO

SETHEO [8] ist ein auf Modellelimination [9, 15] basierender automatischer Theorembeweiser, der Beweise in Tableauform generiert. Bedingt durch den verwendeten Kalkül, kann er nur Klauseltheorien verarbeiten. Da ILF über einen Klausel-Normalform-Algorithmus verfügt, können beliebige Axiome des Prädikatenkalküls erster Ordnung verarbeitet werden. Das gleiche gilt für das Beweisziel, das heißt jede aus der Theorie ableitbare prädikatenlogische Aussage kann von SETHEO bewiesen werden. Das Beweisziel wird negiert in die Datenbasis aufgenommen. Mit dieser Datenbasis wird versucht, einen Widerspruch zu finden, was äquivalent dazu ist, daß die zu beweisende Aussage ableitbar ist. Der mathematische Hintergrund dieser Sachverhalte wird ausführlich in ([16], Kapitel 2) beschrieben.

SETHEO wird bei der hier behandelten Beweisaufgabe wettbewerbsparallel eingesetzt, das heißt, jede zu widerlegende Formelmenge wird an mehrere mit unterschiedlichen Parametersätzen versehene Beweiser übergeben. Der zuerst erfolgreiche Beweiser stoppt über eine Message in der Parallelen Virtuellen Maschine (PVM) [4] die anderen Beweiserinstanzen und informiert ILF. Es zeigte sich bei den Messungen, daß es zur Lösung der in dieser Studie generierten Beweisaufgaben keine optimale Parametrisierung des Beweisers gibt. So lösen bestimmte Parametrisierungen einige Aufgaben sehr schnell, andere Aufgaben werden innerhalb der Zeitschranken nicht gelöst. Erst bei Verwendung eines parallelen Systems ist man überhaupt in der Lage, ohne manuellen Eingriff in die Beweiserparametrisierung alle auftretenden Beweisverpflichtungen in zumutbarer Zeit zu lösen. Zur Gewinnung einer vollständigen Übersicht der Beweiszeiten wurden in dieser Studie die übrigen Beweiser nicht terminiert. Beim Einsatz von SETHEO erfolgt zunächst eine Compilation der zu widerlegenden Formelmenge. Anschließend wird der eigentliche Beweiser (sam = Setheo Abstrakte Maschine) aufgerufen.

SETHEO kann mit unterschiedlichen Parametereinstellungen eingesetzt werden. Zunächst ist festzulegen, welches Verfahren für die iterative Durchsuchung des Suchbaumes verwendet wird. Unsere Konfigurationen verwenden iterative Tiefensuche (dr) und gewichtete iterative Tiefensuche (wdr). Bei letzterer wird das starre Konzept der Suche bis genau in eine bestimmte Tiefe durch die Zuteilung von Gewichten auf die Formeln im Beweistableau gelockert. Desweiteren wurden in dieser Studie suchraumumordnende Techniken verwendet, die dynamisch während der Beweissuche die Liste der noch zu lösenden Teilziele unter verschiedenen Gesichtspunkten umordnet (dynsgreord $< n >$). Außerdem wurden suchraumeinschränkende Constraints wie etwa Regularität benutzt (cons). SETHEO bietet einen ganzen Zoo an verschiedenen Einstellungsoptionen, die von Fall zu Fall entsprechend den Anforderungen der aktuellen Anwendung kombiniert werden können. Für die Zwecke dieser Studie waren die hier verwendeten Einstellungen erfolgreich.

In dieser Studie wurde SETHEO mit 6 verschiedenen Konfigurationen eingesetzt.

- -cons -dr

- -cons -dr -dynsgreord 2
- -cons -dr -dynsgreord 5
- -cons -wdr
- -cons -wdr -dynsgreord 2
- -cons -wdr -dynsgreord 5

Die Abarbeitungszeiten der einzelnen Beweiservarianten sind im Abschnitt 5 aufgeführt.

2 Taktiken

Bei mathematischen Beweisen haben sich im Laufe der Zeit Beweisstrategien und Beweistaktiken herauskristallisiert, die das Auffinden und die Übersichtlichkeit von Beweisen verbessern. Dazu gehören u.a. Induktion, Fallunterscheidung, indirekte Beweisführung, gleichzeitige Addition und Subtraktion von geeigneten Termen usw.

Im Mittelpunkt dieses Abschnittes stehen Auswahl, Anwendung und Analyse von Beweistaktiken beim maschinellen Beweisen im Rahmen von `proofpad`. Bei der Auswahl geht es um Kriterien für die Anwendung einzelner Taktiken. Diese Kriterien werden im allgemeinen nicht notwendig sein. Es ist sinnvoll, Taktikvorschläge (Heuristiken) anzubieten, denen der Nutzer folgen, die er jedoch auch verwerfen kann. Die Analyse soll Vorteile der gezielten Anwendung von Beweistaktiken bei Suchaufwand und Beweisdarstellung aufzeigen.

2.1 Induktion

Eine der wichtigsten Beweistaktiken im Bereich der natürlichen Zahlen ist die Methode der vollständigen Induktion. Das in diesem Abschnitt verwendete Induktionsschema wird im folgenden Satz 3 angegeben. Dabei können durch zwei Beweisaufgaben – Induktionsanfang und Induktionsschritt – Aussagen über natürliche Zahlen verifiziert werden.

Satz 3 .
Sei $H(n)$ ein Ausdruck (des Prädikatenkalküls erster Ordnung) mit der (einzigen) freien Variable n, die vom Typ $\mathbb{N}$ ist. Gelten die Aussagen $H(0)$ (Induktionsanfang) und $\forall k \in \mathbb{N} : H(k) \rightarrow H(k+1)$ (Induktionsschritt), so ist auch die Aussage $\forall n \in \mathbb{N} : H(n)$ gültig.

Bemerkung 3 .
Die Ausdrücke $H(k)$ und $H(k+1)$ werden auch entsprechend als Induktionsvoraussetzung und Induktionsbehauptung bezeichnet. Die Variable n wird Induktionsvariable genannt.

Der Beweis von Satz 3 kann indirekt über das Fundierungsaxiom geführt werden ([16],S.163). Die Bedeutung für die Arithmetik (natürlicher Zahlen) ergibt sich aus dem mengentheoretischen Aufbau der natürlichen Zahlen, der genau diesem Schema folgt. Diese Methode läßt sich variieren zum Beispiel als Übergang von $n = k$ und $n = k+1$ nach $n = k+2$, wobei der Induktionsanfang für $n = 0$ und $n = 1$ bewiesen werden muß) und auch auf andere Aufbauprozesse (wie zum Beispiel den Formelaufbau) übertragen. Eine umfangreiche Beschreibung der Induktion ist in [12] gegeben. Dieses Beweismittel steht in engem Zusammenhang mit der Behandlung von Funktionen, die mittels Rekursionsschemata definiert sind. Diese werden unter anderem bei der Behandlung diskreter Zeitsysteme genutzt, sofern die Zeitskala mit den natürlichen Zahlen identifiziert wird.

Ein wichtiger Spezialfall ist die Induktion über Aussagen der Form

$$\forall n \in I\!N : \bigwedge_{i \in I} H_i(n)$$

mit einer endlichen Indexmenge I. Sei $H(n) = H_1(n) \wedge H_2(n)$. So ergibt sich für den Induktionsanfang die Beweisaufgabe $H_1(0) \wedge H_2(0)$. Der Beweis dieser Aufgabe folgt aus der Gültigkeit der beiden Teilaussagen $H_1(0)$ und $H_2(0)$, die voneinander unabhängig bewiesen werden können. Der Induktionsschritt

$$\forall k \in I\!N : H_1(k) \wedge H_2(k) \rightarrow H_1(k+1) \wedge H_2(k+1)$$

läßt sich auf die obige Weise nicht in zwei unabhängige Teilaufgaben zerlegen, da die Induktionsvoraussetzung global ist. Es existiert eine Zerlegung in folgende zwei Beweisaufgaben.

$$H_1(k) \wedge H_2(k) \rightarrow H_1(k+1)$$

$$H_1(k) \wedge H_2(k) \rightarrow H_2(K+1)$$

Aus diesen beiden voneinander unabhängig zu beweisenden Aufgaben folgt die Gültigkeit des Induktionsschrittes. Dieser Spezialfall (und die Verallgemeinerung auf endlich viele Konjunktionsglieder) wird auch als simultane Induktion bezeichnet. Dabei ergeben sich $2k$ Beweisaufgaben, wobei k die Anzahl der Konjunktionsglieder ist.

2.2 Fallunterscheidung

Das Beweisen mittels Fallunterscheidung ist eine Taktik, die es erlaubt, Beweise zu strukturieren und damit übersichtlicher zu gestalten. Dabei wird für eine Aussage jeweils eine zusätzliche Annahme gemacht. Zusätzlich muß die Alternative über alle Annahmen bewiesen werden.

Satz 4 .
Seien $Q, P_1, \ldots, P_k$ $(k \in I\!N, k > 1)$ beliebige Aussagen des Prädikatenkalküls erster Ordnung. Gelten die Aussagen $P_1 \rightarrow Q, \ldots, P_k \rightarrow Q$ und $P_1 \vee \ldots \vee P_k$, so gilt auch Q. □

Damit wird die Aussage Q k-mal, jeweils unter einer der Annahmen P_i bewiesen. Daraus ergeben sich $k+1$ neue Beweisaufgaben bei Anwendung dieser Fallunterscheidung. Für $k = 1$ entspricht obiger Satz der Abtrennungsregel. Auf die sich ergebenden Implikationen $P_i \to Q$ können weitere Taktiken, wie später beschrieben, angewandt werden.

Da eine Beweisaufgabe durch Erzeugung von $k+1$ neuen Beweisaufgaben gelöst wurde, steht die Frage nach Notwendigkeit und Effektivität von Fallunterscheidungen. Die vier folgenden Beispiele untersuchen das Zeitverhalten und zeigen den möglichen Vorteil der Taktik für Fallunterscheidungen. Dabei wurden sowohl Fallunterscheidungen mit zwei als auch mit drei Fällen behandelt. Die Untersuchungen erfolgten mit SETHEO auf einer SPARC-10. Die benötigten Abarbeitungszeiten sind in der folgenden Tabelle 1 angegeben (Zeiten in Sekunden). Die Beweistiefe bezeichnet die Tiefe des gefundenen Beweistableaus, die Inferenzenzahl die Anzahl der im Beweis enthaltenen Inferenzen.

Beispiel 1 :
Struktur: $P = \{a, b\};\ f : P \to P,\ g : P \to P$
Aufgabe: $\forall x \in P : f(x) = g(x)$
Axiome:
$\forall x \in P : x = a \to f(x) = b$
$\forall x \in P : x = b \to f(x) = a$
$\forall x \in P : x = a \to g(x) = b$
$\forall x \in P : x = b \to g(x) = a$
$\forall x \in P : x = a \lor x = b$

Beispiel 2 :
Struktur: $T = \{a, b, c\};\ f : T \to T,\ g : T \to T$
Aufgabe: $\forall x \in T : f(x) = g(x)$
Axiome:
$\forall x \in T : x = a \to f(x) = b$
$\forall x \in T : x = b \to f(x) = c$
$\forall x \in T : x = c \to f(x) = a$
$\forall x \in T : x = a \to g(x) = b$
$\forall x \in T : x = b \to g(x) = c$
$\forall x \in T : x = c \to g(x) = a$
$\forall x \in T : x = a \lor x = b \lor x = c$

Beispiel 3 :
Struktur: $P = \{a, b\};\ f : P \to P$
Aufgabe: $\forall x \in P : f(f(x)) = x$
Axiome:

$\forall x \in P : x = a \rightarrow f(x) = b$
$\forall x \in P : x = b \rightarrow f(x) = a$
$\forall x \in P : x = a \vee x = b$

Beispiel 4 :
Struktur: $T = \{a, b, c\}$; $f : T \rightarrow T$
Aufgabe: $\forall x \in T : f(f(f(x))) = x$
Axiome:
$\forall x \in T : x = a \rightarrow f(x) = b$
$\forall x \in T : x = b \rightarrow f(x) = c$
$\forall x \in T : x = c \rightarrow f(x) = a$
$\forall x \in T : x = a \vee x = b \vee x = c$

Vergleicht man die Laufzeiten, so bringen die Fallunterscheidungen in den Beispielen 1 bis 4 Zeitgewinne. Das ist aber aber nicht immer notwendig, um die Aufgabe innerhalb eines CPU-Limits von 2 Minuten, wie es ILF vorgibt, zu lösen. Dagegen sind die Fallunterscheidungen in den Beispielen 2 und 4 notwendig, um eine Lösung innerhalb des Zeitlimits zu finden. Eine geringfügige Veränderung des CPU-Limits hätte in diesen beiden Beispielen auch zu keiner Lösung geführt, da die Suchzeiten im Stundenbereich liegen. Derartige Zeiten sind in interaktiven Systemen nicht akzeptabel.

Wird die Fallunterscheidung für den Beweis benötigt und auch im automatisch generierten Beweis des Ursprungsproblems implizit durchgeführt, dann wird durch die manuelle Vorwegnahme dieses Schrittes die Beweistiefe der verbleibenden Teilaufgaben verringert, im besten Fall halbiert. Das läßt sich jedoch nicht allgemein feststellen, wie das folgende Beispiel 5 veranschaulicht.

Beispiel 5 :
Struktur: $T = \{a, b\}$; $q : T$, $p_1 : T$; $p_{2,3,4,5}$
Aufgabe: $\forall x \in T : q(x)$
Axiome:
$\forall x \in T : p1(x) \wedge p_2 \rightarrow q(x)$
$p_1(a)$
$p_1(a)$
$\forall x \in T : x = a \vee x = b$
$p_3 \rightarrow p_2$
$p_4 \rightarrow p_3$
$p_5 \rightarrow p_4$
p_5

Sowohl ohne Fallunterscheidung als auch mit der Fallunterscheidung $x = a \vee x = b$ werden die Aufgaben jeweils in der Tiefe 5 gelöst. Bei Anwendung der Fallunterscheidung vergrößert sich die Theorie durch Hinzunahme der jeweiligen Annahme.

Tabelle 1: Fallunterscheidungen

Tiefe/Inf./Zeit	1		2		3		4		5	
ohne Falluntersch.	7	49.43	8	10015.48	7	30.33	9	7049.15	5	0.03
mit Falluntersch.										
Alternative	2	<0.01	2	<0.01	2	<0.01	2	<0.01	2	0.02
Fall 1	4	0.10	4	0.10	4	0.05	5	0.37	5	0.17
Fall 2	4	0.08	4	0.10	4	0.05	5	0.35	5	0.17
Fall 3	-	-	4	0.10	-	-	5	0.40	-	-

Damit vergrößert sich auch der Suchraum. Es ist folglich möglich, daß ein Beweis erst später gefunden wird, als es ohne Fallunterscheidung der Fall gewesen wäre. Die Anwendung der Fallunterscheidung kann die Beweistiefe des gefundenen Beweises nicht vergrößern, da der alte Beweis ohne Fallunterscheidung ebenfalls im Suchraum verbleibt. Das folgt unmittelbar daraus, daß die Beweisaufgabe gleich bleibt und sich die Theorie nur durch Hinzunahme der Annahme vergrößert. Also wird der Beweis ohne Anwendung der Fallunterscheidung in der gleichen Tiefe wieder gefunden. Durch Suchraumvergrösserung aufgrund der Theorieerweiterung und Erzeugung von $n+1$ neuen Beweisaufgaben bei Anwendung einer n-fachen Alternative kann sich der kumulierte Suchaufwand erheblich steigern.

Werden Funktionen mittels Fallunterscheidung definiert, ist es im allgemeinen notwendig und auch sinnvoll, diese Fallunterscheidung auch in Beweisen, in denen diese Definition genutzt wird, anzuwenden. In der Beispielaufgabe trifft dies auf die Funktionen $f_{i=1,2,3}$ und $g_{i=1,2,3}$ zu, die in Abhängigkeit von dem Wert x durch Fallunterscheidung definiert worden sind. Die letzte Aussage charakterisiert bereits ein Anwendungskriterium für die Fallunterscheidung. Wird in der Formelmenge der Beweisaufgabe eine Funktion ermittelt, die durch Fallunterscheidung definiert wurde, so wird diese Alternative vorgeschlagen. Ein weiteres Kriterium könnte die Suche nach Alternativen in der Theorie sein. Weiterhin kann durch Fallunterscheidungen, in denen einzelne Fälle schnell widerlegt werden können, der Beweisumfang reduziert werden.

2.3 Projektion

Ziel dieses Abschnittes ist die Zerlegung von Gleichheitsaussagen auf Kreuzprodukten auf die einzelnen Faktoren. Dieses Konzept wird im weiteren verallgemeinert und auch auf Funktionen angewandt. Grundlage hierfür ist die Paramodulation.

Maschinen werden als komplexe Strukturen interpretiert. Gleichzeitig werden die Übergänge zwischen zwei aufeinander folgenden Zuständen komponentenweise definiert. Die Einschritt-Überführungsfunktionen werden dann als Kompositionen über diesen Komponenten erklärt. Das dient der ganzheitlichen Behandlung und Charakterisierung der Maschinen. Für den Nachweis der Äquivalenz zweier Maschinen wird der Kompositionsprozeß mit der folgenden Projektionsregel wieder rückgängig gemacht.

Satz 5 .
Seien $A = A_1 \times \cdots \times A_n$ und $a, b \in A$, wobei $a = (a_1, \ldots, a_n)$ und $b = (b_1, \ldots, b_n)$. Dann gilt $a = b$ genau dann, wenn $a_1 = b_1, \ldots, a_n = b_n$.

Der Satz ist die Verallgemeinerung der Charakterisierung geordneter Paare ([16], Satz 5.4).

Ausgehend von dieser mengentheoretischen Betrachtung gehen wir nun über zur prädikatenlogischen Darstellung. Sei A wie im vorangegangenen Satz, $k : A_1 \times \cdots \times A_n \to A$ die Kompositionsfunktion und $p_{i=1,\ldots,n} : A \to A_{i=1,\ldots,n}$ die Projektionsfunktionen. Dann werden a und b wie folgt definiert: $a := k(a_1, \ldots, a_n), b := k(b_1, \ldots, b_n)$. Für die Projektionen gilt dann: $p_i(a) = a_i$ und $p_i(b) = b_i$ für alle $i = 1, \ldots, n$. Aufgrund der Injektivität der Kompositionsfunktion gilt dann auch die obige Projektionsregel, das heißt für die Gleichheit der Werte der Kompositionsfunktion (oder der geordneten n-Tupel) genügt es, die Gleichheit für alle Parameter der Kompositionsfunktion (oder der Komponenten der geordneten n-Tupel) zu zeigen. Diese letzte Aussage, daß aus der Gleichheit der Parameter die Gleichheit für die Funktionswerte folgt, entspricht der Paramodulation für Funktionen bezüglich aller Parameter. Die Umkehrung dieser Aussage, daß aus der Gleichheit der Funktionswerte auch die Gleichheit der Argumente folgt, gilt nur bei injektiven Funktionen, also auch bei den Kompositionsfunktionen.

Mit Hilfe der Paramodulation kann diese Idee der Beweisreduzierung auf Gleichungen der Form $f(\bar{x}) = f(\bar{y})$ verallgemeinert werden, wobei f eine beliebige Funktion ist. Der Nachweis der Gleichheit der Argumente $\bar{x} = \bar{y}$ ist bei injektiven Funktionen immer ein möglicher Beweisweg. Bei nicht-injektiven Funktionen ist dieser Weg auch möglich, er muß jedoch nicht zum Erfolg führen.

Bei SETHEO kann die Anwendung dieser Taktik zur Tiefenreduzierung führen. Die Weglassung von Paramodulationsaxiomen führt zu einer Reduzierung des Suchraumes. Werden Paramodulationsaxiome weggelassen, so lassen sich nicht mehr alle Aussagen durch SETHEO beweisen. In einem solchen Falle dient die Projektionstaktik nicht nur der Reduzierung des Beweisumfanges, sondern ist sogar notwendig.

2.4 Beweisansätze

Wie bereits erwähnt, kann es keine allgemeine Strategie geben, nach der Beweise gefunden werden können. Vielfach werden deshalb verschiedene Ansätze parallel betrachtet. Intuitiv werden die Zwischenergebnisse der einzelnen Ansätze bewertet und erfolgversprechende Ansätze weiterverfolgt.

Mit dem `proofpad` kann diese Taktik durch wiederholte Einfügung (`ins`) der zu beweisenden Formel realisiert werden. Anstelle der zu beweisenden Formel kann auch eine hinreichende Bedingung eingefügt werden. Da die Beweisverwaltung im `proofpad` durch eine lineare Abhängigkeit gekennzeichnet ist und die einzelnen Beweisansätze natürlich von einander unabhängig seien müssen, werden die einzelnen Ansätze mit `lock` nach außen gesperrt. Diese sind alle gleichberechtigt, eine Bewertung mittels Heuristiken existiert nicht. Die Aufgabe ist bewiesen, sobald ein Ansatz erfolgreich war. Dieser Ansatz – oder einer, wenn mehrere gleichzeitig erfolgreich waren – wird ausgewählt.

Grundsätzlich können für alle Aufgaben mehrere Ansätze vorgenommen werden. Zunächst werden Ansätze betrachtet, bei denen durch logische Umformungen verschiedene Beweiswege realisiert werden können. Dazu seien zwei Beispiele diskutiert.

Werden Formeln betrachtet, auf die die Induktionstaktik angewandt werden kann, kann die Induktion über mehrere Variable durchgeführt werden. Es wird für jede dieser Variablen ein Ansatz generiert. Zusätzlich kann die Aufgabe auch ohne Anwendung der Induktion gelöst werden. Da Abhängigkeiten zwischen den Variablen bestehen können, ist es möglich, daß verschiedene Ansätze nicht erfolgreich sind.

Ein zweites Beispiel ist das Beweisen von Implikationen. SETHEO kann eine Implikation durch Widerlegung der Prämisse oder durch Beweis der Konlusion bei zur Datenbasis hinzugenommener Prämisse beweisen. Im Falle der Widerlegung der Prämisse muß die Konklusion eventuell nicht in die Formelmenge aufgenommen werden. Das kann insbesondere bei der mehrfachen Anwendung von Fallunterscheidungen auftreten, wenn sich bereits die einzelnen Annahmen widersprechen. Weiterhin kann der Umfang der genutzten Annahmen auch die Theorieauswahl und damit den gesamten Beweisumfang beeinflussen.

Ein abschließendes Beispiel ist die Möglichkeit der unterschiedlichen Konfiguration. Dabei geht es darum, durch eine geeignete Axiomendarstellung andere Axiome aus der Theorie entfernen zu können. Das ist möglich bei Paramodulationsaxiomen und als Spezialfall bei der Transitivität der Gleichheit. Weiterhin steht die Frage nach eventueller Hinzunahme von Fakten (Literalen). Darüber hinaus können verschiedene Ansätze mit unterschiedlicher Theorieauswahl betrachtet werden.

Die Beweisauswahl bei nur einem erfolgreichen Ansatz ist eindeutig:

1. die nicht erfolgreichen Ansätze werden mit `delete` gelöscht und
2. der erfolgreiche Ansatz wird mit `unlock` sichtbar gemacht.

Mit Hilfe dieses Ansatzes wird dann die Formel, für die mehrere Ansätze generiert wurden, bewiesen. Da dies im allgemeinen nur eine triviale aussagenlogische Umformung ist, könnte hierfür auch ein Regelsystem genutzt werden. Für diese triviale Aufgabe müßte kein spezieller Beweiser gestartet werden. Welche Regeln in solch ein Regelsystem aufgenommen werden, wird aus der Situation, in der die verschiedenen Ansätze gemacht werden, abgeleitet.

Für die Beweisauswahl bei mehreren erfolgreichen Ansätzen gibt es keine Regel, welcher Ansatz ausgewählt werden sollte. Nach Auswahl eines erfolgreichen Ansatzes wird analog der Beweisauswahl bei einem erfolgreichen Ansatz verfahren.

2.5 Verwendete Strategien

Im folgenden wird die Aufgabe aus Abschnitt 2 im Rahmen der Möglichkeiten von ILF bearbeitet. Die vorangegangen Taktiken dieses Kapitels werden, wenn möglich, angewandt. Außerdem wird bereits auf die Theorieauswahl eingegangen. Diese Betrachtungen werden in Abschnitt 4 noch vertieft. Die Grundlage für die Bearbeitung bildet das interaktives Tool `proofpad`, welches auf einzelne Beweisaufgaben angewandt wird und entsprechende Taktikvorschläge unterbreitet. Außerdem prüft es, ob bei mehreren Beweisansätzen wenigstens einer erfolgreich war. Im Erfolgsfall wird ein solcher Beweis entsprechend ausgewählt. Zusätzlich kann die Theorieauswahl angesteuert werden. Nach jeder Veränderung der Beweisstruktur wird diese im TREEVIEWER neu angezeigt. Das Tool arbeitet nach dem folgenden Schema.

1. Sind an wenigstens einer Stelle im Gesamtbeweis mehrere Beweisansätze für eine Aufgabe gemacht worden, so wird überprüft, ob einer der Ansätze bereits erfolgreich war. Im Erfolgsfall wird entsprechend ausgewählt.

   Auf diese Weise werden alle entsprechenden Stellen behandelt. Die Information, daß für eine Aufgabe mehrere Ansätze gemacht wurden, wird separat gespeichert. Anschließend wird zu 2. übergegangen.

2. Abfrage nach der weiteren Vorgehensweise. Dabei sind folgende Eingaben möglich:

   (a) – der Steuerung

   (b) Übergang zur Theorieauswahl (→ 5.)
   (c) Weiterbearbeitung der aktuellen Position (→ 3.)
   (d) Bearbeitung einer anderen unbewiesenen Position (→3.)

3. Analyse der zu bearbeitenden Formel nach folgenden Kriterien:

   (a) Sind mehrere Beweisansätze möglich?
   (b) Kann eine Induktion angewandt werden?
   (c) Ist eine Zerlegung in einzelne Projektionen möglich?
   (d) Kann eine Fallunterscheidung angewandt werden?

   Alle diese so ermittelten Vorschläge werden dem Nutzer angeboten. Außerdem wird die aktuelle Position, die Formel, sowie alle Annahmen (`assumption`) angezeigt. Anschließend wird zu 4. übergegangen.

4. Eingabe einer Anweisung. Dabei muß den angebotenen Taktikvorschlägen nicht gefolgt werden. Nach der erfolgten Eingabe wird die veränderte Beweisstruktur sowohl im ILF-Fenster, als auch im TREEVIEWER angezeigt. Anschließend wird zu 2. zurückgesprungen.

5. Theorieauswahl für alle noch nicht behandelten Beweisaufgaben. Zuerst wird der Umfang der Theorieauswahl abgefragt. Neben der zu beweisenden Formel können auch alle für die zu beweisende Formel nutzbaren Formeln oder eine Teilmenge davon analysiert werden. Es kann auf eine Standardtheorie zurückgegriffen werden oder eine individuelle Theorieangabe erfolgen.

Für das Lösen der Beispielaufgabe wird zunächst die Induktion auf den Zeitparameter angewandt. Der Induktionsanfang ist dann sofort ohne weitere Zerlegung durch SETHEO beweisbar. Die vollständige Lösung der Aufgabe findet mal im Abschnitt 4

3 Theorieauswahl

In diesem Abschnitt wird untersucht, welche Axiome für eine Beweisaufgabe tatsächlich benötigt werden. Ist es möglich, bereits vorher darüber Aussagen zu treffen, welche Axiome für eine Beweisaufgabe nicht von Bedeutung sind, so sollten diese dem Beweiser nicht übergeben werden. Diese Theorieeinschränkung ist rein technischer Natur, aber für den effektiven Einsatz von automatischen Theorembeweisern unerläßlich, da sie eine Suchraumeinschränkung mit relativ einfachen Mitteln ermöglicht. Bei der Theorieeinschränkung spielen mathematische und intuitive Heuristiken, die nicht einem mathematischen Verfahren gleichgesetzt werden können, eine Rolle. Ein solches Verfahren existiert nicht, da manche mathematischen Probleme erst dadurch gelöst werden, daß die Problemstellung oder ein Teil davon in ein anderes mathematisches Teilgebiet transformiert wird.

Zunächst beschäftigen wir uns mit der Einteilung der Axiome in *Gruppen* und damit, wie diese Gruppen untereinander in Beziehung stehen und ausgewählt werden. Außerdem wird die Frage nach der Erweiterung der Theorie durch Aufnahme von bewiesenen Lemmata diskutiert. Danach geht es um die Frage, wie die zu beweisende Formel analysiert werden sollte, um eine für eine schnelle Beweisfindung *gute* Theorieauswahl zu erhalten. Abschließend wird auf die Darstellung der Axiome und deren Bedeutung für den Beweisumfang eingegangen. Dabei wird auf den Beweiser Setheo Bezug genommen.

3.1 Axiomengruppierung

Wenn nur bestimmte Axiome in eine Theorie für einen Beweiser aufgenommen werden sollen, müssen die Axiome unterscheidbar sein. Diese Unterscheidung ist durch den Inhalt (die Formel) der Axiome gegeben Ein solches Unterscheidungsmerkmal ist praktisch meist schlecht anwendbar. Soll ein Axiom ausgewählt werden, müßte immer die – oftmals umfangreiche – Formel angegeben werden. Günstiger ist es, jedem Axiom einen *Namen* zu geben.

Im System Ilf werden alle Axiome durch zwei Prolog-Terme dargestellt. Der erste Term stellt den Inhalt des Axioms dar und der zweite den Namen. Mit den Kommandos `use`, `use_also` und `use_always` kann die Theorie für eine Beweisaufgabe geändert werden. Als Parameter für diese Kommandos wird eine Liste übergeben, die Axiomennamen, Theorienamen und weitere Handles enthalten kann. Kommen in den Listenelementen Variablen vor, so werden alle Terme, die damit matchen, ausgewählt. Wird keine Theorieauswahl vorgenommen, so wird eine vom Nutzer vorgegebene Standardtheorie verwendet.

Für die Bearbeitung der Beispielaufgabe wurde folgende Struktur für die Axiomennamen gewählt: `ax(Operator,Spezifikation,Status,Name).` Der Operator sollte das für den Beweis wesentliche Funktions- oder Relationssymbol des

Axioms sein. Manchmal will oder kann man ein wesentliches Symbol nicht festlegen. So läßt sich zum Beispiel für das Axiom der Monotonie der Addition bezüglich der Relation $>$ nicht von vornherein eindeutig bestimmen, ob $+$ oder $>$ wesentlicher ist. Eine mögliche Lösung ist, als Operator auch Listen zuzulassen. Mit Hilfe der Spezifikation läßt sich die Axiomenauswahl bezüglich eines Operators noch verfeinern. Beispiele sind Rekursionsanfang und Rekursionsschritt sowie Projektionen mehrstelliger Funktionen. Der Status enthält eine Zusatzinformation, wie zum Beispiel Definition oder Lemma für die Verbesserung der Übersichtlichkeit. Für die Auswahl der Axiome der Addition kann dann das Kommando `use(ax(+,_,_,_))` verwendet werden. Wird von der rekursiv definierten Funktion F (dargestellt durch ff, Prolog interpretiert Großbuchstaben als Variable) nur das Axiom für den Rekursionsanfang benötigt, so ist dies mit dem Kommando `use(ax(ff,ra,_,_))` möglich.

Axiomennamen können auch nach inhaltlichen Gründen strukturiert werden. So können die Projektions- und Kompositonsaxiome zusammengefaßt werden, beispielsweise unter den Axiomennamen `ax(projektion,_,_,_)`, wobei _ für beliebige Terme steht. Eine solche semantische Zusammenfassung kann nur vom Nutzer realisiert werden, während für den ersten Ansatz auch eine automatische Axiomennamenbildung denkbar ist.

Axiome können zu Theorien zusammengefaßt sein, wie zum Beispiel die Axiome, die die Addition etwa in $\mathbb{N}$ beschreiben, zur Theorie der Addition. Dazu muß diese Theorie einen Namen erhalten, beispielsweise `thplus`, und es muß erklärt werden, welche Axiome diese Theorie enthält.

Beide Möglichkeiten der Axiomengruppierung können parallel genutzt werden. Letztere ist vor allem dann geeignet, wenn die Theorie (für einen Operator) im Laufe des Gesamtbeweises nicht mehr verändert wird, also keine Lemmata hinzukommen.

3.2 Formelanalyse

Für die Theorieauswahl gilt das Prinzip, daß

1. die verwendete Theorie so stark wie möglich eingeschränkt werden soll und andererseits
2. alle für den Beweis benötigten Axiome in die Datenbasis aufgenommen werden.

Dieses Auswahlverfahren wird als Vorbereitung für den automatischen Beweiser angesehen. Es sollte folglich entsprechend schnell sein. Das setzt voraus, daß die

Axiome entsprechend strukturiert ausgewählt werden können. Mit Hilfe des oben beschriebenen Ansatzes ist dies auf einfache Weise möglich.

Zur Analyse wird die zu beweisende Formel nach syntaktischen Regeln analysiert und die vorhanden Operatoren ermittelt. Eine weitergehende Unterscheidung wird in der Beispielaufgabe nur bei rekursiv definierten Funktionen beziehungsweise bei Projektionen gemacht: Unterscheidung in Rekursionsanfang und Rekursionsschritt, sofern das entsprechende Argument das zuläßt, beziehungsweise Ermittlung der Projektionskomponente.

Bei einigen Operatoren werden weitere zugehörige Axiome ausgewählt. Das ist hauptsächlich bei Operatoren notwendig, die auf Basis anderer Operatoren definiert wurden. Beispielsweise sollten bei den Operatoren `inc` beziehungsweise `dec` auch die Axiome für die Addition beziehungsweise für die Subtraktion hinzugenommen werden. Dieses ist ein semantisches Auswahlverfahren. Diese Zuordnungen werden innerhalb der hier beschriebenen Theorieauswahl nicht automatisch vom System durchgeführt, sondern beruhen auf den vom Nutzer eingegebenen Abhängigkeiten. Denkbar ist aber auch hier eine automatische Ermittlung von Zusammenhängen, bei denen Verfahren der Theorieauswahl, Strukturierung der Axiomennamen und semantischer Theorieaufbau gleichzeitig betrachtet werden.

Wie Beispiele zeigen, ist es oft nicht ausreichend, nur die zu beweisende Formel zu analysieren. Deswegen ist es möglich, weitere Annahmen und Voraussetzungen auf die gleiche Art und Weise zu untersuchen, und die so gewonnenen Axiome zusätzlich in die Datenbasis aufzunehmen. Im Rahmen des Strategie-Tools kann der Nutzer entscheiden, in welchem Umfang die Analyse erfolgen soll. Die zusätzliche Analyse von Annahmen beziehungsweise Voraussetzungen wird benötigt, wenn daraus andere für den Beweis notwendige Aussagen abgeleitet werden sollen. Dieser Ansatz beschränkt sich auf rein syntaktische Untersuchungen, semantische Aspekte spielen – mit Ausnahme der vom Nutzer fixierten Abhängigkeiten – keine Rolle.

3.3 Axiomendarstellung

In diesem Abschnitt wird die Aufmerksamkeit auf die Darstellung der Axiome gelenkt. Aussagen können logisch äquivalent seien. Welche von zwei logisch äquivalenten Aussagen in die Theorie aufgenommen wird, ist semantisch unbedeutsam. Das gleiche gilt auch für die Aufnahme beider Aussagen in die Theorie. Für einen automatischen Beweiser kann der Unterschied erheblich sein.

Definition 5 .
Seien ϕ_1 und ϕ_2 zwei Aussagen des Prädikatenkalküls erster Ordnung. ϕ_1 und ϕ_2 heißen (logisch) äquivalent ($\phi_1 \equiv \phi_2$), wenn gilt, daß $\emptyset \models \phi_1 \leftrightarrow \phi_2$. Dann wird die Aussage $\phi_1 \leftrightarrow \phi_2$ auch allgemeingültig genannt.

Aus dieser Definition folgt unter Benutzung des Ableitungssatzes unmittelbar $\{\phi_1\} \models \phi_2$ und $\{\phi_2\} \models \phi_1$.

Satz 6 .
Gelte $\phi_1 \equiv \phi_2$ und sei T eine Theorie des Prädikatenkalküls erster Ordnung. Wenn $T_1 = T \cup \{\phi_1\}$, $T_2 = T \cup \{\phi_2\}$ und $T_3 = T \cup \{\phi_1, \phi_2\}$ sind, so gelten die folgenden beiden Aussagen:

1. $Ded(T_1) = Ded(T_2)$ *und*
2. $Ded(T_1) = Ded(T_3)$.

Beweis: Aus $\{\phi_1\} \subseteq \{\emptyset \cup \{\phi_1\} \subseteq T_1$ und $\{\phi_1\} \models \phi_2$ folgt $T_1 \models \phi_2$. Aufgrund der Hülleneigenschaften der Deduktion gilt $Ded(T_1) \supseteq Ded(T) \supseteq T$. Da also aus $Ded(T_1)$ sowohl ϕ_2 als auch T folgen, gilt $Ded(T_1) \supseteq T_2 = T \cup \{\phi_2\}$. Nochmalige Anwendung der Hülleneigenschaften ergibt, daß $Ded(T_1) = Ded(Ded(T_1)) \supseteq Ded(T_2)$.

Analog wird $Ded(T_2) \supseteq Ded(T_1)$ gezeigt, woraus dann schließlich (1) folgt.

Da $T_1 \subseteq T_3$ gilt, folgt daß $Ded(T_1) \subseteq Ded(T_3)$. Wie oben läßt sich zeigen, daß $Ded(T_1) \supseteq T_3$. Daraus folgt mit Hilfe der Hülleneigenschaften, daß $Ded(T_1) = = Ded(Ded(T_1)) \supseteq Ded(T_3)$. Damit wurde auch (2) bewiesen. □

Analog zur Aussage (2) des Satzes 6 gilt natürlich auch $Ded(T_2) = Ded(T_3)$. Wenn die konkrete Darstellung eines Axioms semantisch gleichwertig ist, so ergibt sich die Frage, ob und warum das Verhalten der Beweiser von der Form abhängig ist.

Da SETHEO ein tableaubasierter Modelleliminationsbeweiser ist und sich der Suchumfang im allgemeinen exponentiell mit der Beweistiefe entwickelt, soll untersucht werden, ob durch eine bestimmte Form der Axiomendarstellung die Beweistiefe reduziert werden kann. Um einerseits dem automatischen Beweiser gerecht zu werden und um andererseits vom Nutzer nicht zu verlangen, die Theorie selbst geeignet umzuformen, sollte diese Prozedur vom `proofpad` übernommen werden. Eventuell könnten Optionen angegeben werden, um diese Prozedur zu beeinflussen.

Im folgenden wird aufgezeigt, welche Axiome in welcher Weise für SETHEO umgeformt werden können, um eventuell die Beweistiefe zu reduzieren. Auch wird auf die Möglichkeit des Weglassens von Substitutionsaxiomen eingegangen. Außerdem wird die Möglichkeit des Weglassens der Transitivität der Gleichheit untersucht, da diese wesentlich die Suche beeinflußt.

Untersucht wird jetzt die Axiomenklasse mit den bedingten Gleichungen der Art

$$p(\bar{x}) \rightarrow f(\bar{x}) = c(\bar{x}). \tag{1}$$

Sei ohne Beeinträchtigung der Allgemeingültigkeit f eine n-stellige Funktion mit den Parametern $g_1(\bar{x}), \ldots, g_n(\bar{x})$. Die Bedingung $p(\bar{x})$ sei eine Konjunktion von Literalen. Die Konjunktion kann auch leer sein, d.h. aus der obigen bedingten Gleichung wird dann eine reine Gleichung. Es werden dann $n+1$ neue Variablen $v_1, \ldots, v_n$ und w eingeführt und das Axiom dann wie folgt *paramoduliert* dargestellt:

$$v_1 = g_1(\bar{x}) \wedge \ldots \wedge v_n = g_n(\bar{x}) \wedge w = c(\bar{x}) \wedge p(\bar{x}) \rightarrow f(v_1, \ldots, v_n) = w. \qquad (2)$$

Diese Variablenersetzung kann bei allen Parametern beziehungsweise beim Funktionswert entfallen, wenn diese bereits eine Variable sind. Allerdings nur, wenn diese Variable erstmals als Parameter beziehungsweise Funktionswert auftritt. Bei jedem weiteren Auftreten dieser Variable als Parameter beziehungsweise Funktionswert wird sie an diesen Stellen durch eine neue Variable ersetzt. Dieses so neu gewonnene Axiom (2) wird an Stelle des Ausgangsaxioms (1) in die Theorie aufgenommen. Ist die Bedingung $p(\bar{x})$ leer, so kann zusätzlich das Ausgangsaxiom in die Theorie aufgenommen werden. Das kann dann möglicherweise die Beweistiefe um 1 reduzieren, es kann aber auch in einigen Fällen zu einem erheblichen Anwachsen des Suchumfanges führen.

Nachstehend werden die Beweistiefen von SETHEO bei der ursprünglichen (1) und der umgeformten (2) Axiomendarstellung untersucht. Dabei wird der Beweis einer Gleichung analysiert. Neben der ausgewählten Theorie können zusätzliche Paramodulationsaxiome in die Datenbasis mit aufgenommen werden. Das wird mit Hilfe der Einstellung des `ilfstate setheo_split` gesteuert. Ist dieser auf `off` gesetzt, werden keine, bei `on` wird für jeden Parameter ein Paramodulationsaxiom und bei `one_axiom` ein Paramodulationsaxiom für alle Parameter simultan erzeugt. Das bezieht sich jeweils auf jedes Funktions- und Relationssymbol, welches in der Datenbasis vorkommt.

Betrachtet wird die Gleichung $f(t_1, \ldots, t_n) = t$, die mit Hilfe des oben angegebenen Axioms (1) beziehungsweise des Axioms (2) bewiesen werden soll. Es müssen für alle $i = 1, \ldots, n$ $t_i = g_i(\bar{x})$ und $t = c(\bar{x})$ gezeigt werden. Zunächst werden einige Bezeichnungen eingeführt:

- PM_e = ursprüngliche Axiomendarstellung (1) und `ilfstate setheo_split on`.
- PM_a = ursprüngliche Axiomendarstellung (1) und `ilfstate setheo_split one_axiom`.
- PM_i = umgeformte Axiomendarstellung (2) und `ilfstate setheo_split off`.

m_e sei die Anzahl der zu paramodulierenden Elemente (Parameter und Funktionswert) von f, für die gilt, daß $t_i \not\equiv g_i(\bar{x})$ beziehungsweise $t \not\equiv c(\bar{x})$. Sei weiterhin

$$o_e := (\mu l \in I\!N)[2^l \geq m_e + 1].$$

Dann gilt, daß $0 \leq m_e \leq n + 1$. Weiter läßt sich abschätzen, daß $o_e \geq 0$. Sei $I = \{1, \ldots, n\}$ und $I^* := \{i \in I : t_i \not\equiv g_i(\bar{x})\}$. Dann ist $I^* \subseteq I$. Wenn k gleich der Kardinalität von I^* ist, so gilt $m_e \leq k + 1 \leq m_e + 1$.

Für alle $i = 1, \ldots, n$ seien d_i die minimalen Beweistiefen von SETHEO für die Aussagen $t_i = g_i(\bar{x})$. Analog seien d_0 und d_p die Beweistiefen für die Aussagen $t = c(\bar{x})$ und $p(\bar{x})$. Die Beweisaufgabe unter den Konfigurationen PM_e und PM_a kann im allgemeinen nicht in einem Schritt auf diese einzelnen Teilaufgaben zurückgeführt werden. Dazu muß die Aufgabe mittels des Transitivitätsaxioms der Gleichheit aufgespalten werden. Erst an diesen Stellen, wir nennen sie *Ansatzpunkte*, ist dann die Paramodulation bezüglich der Parameter und die Anwendung von (1) möglich. Dagegen wird bei der Konfiguration PM_i diese Reduzierung auf die Teilaufgaben durch die umgeformte Axiomendarstellung (2) in genau einem Schritt erreicht. Mit $d_{pme}, d_{pma}, d_{pmi}$ werden die Beweistiefen für die Gleichung $f(t_1, \ldots, t_n) = t$ bezüglich PM_e, PM_a, PM_i bezeichnet.

Definition 6 .
Sei A eine endliche Menge natürlicher Zahlen und $min(A)$ wie üblich definiert. Dann existiert ein $a_0 \in A$, so daß $min(A) = a_0$. Dann sei $min^\star(A) := min(A - \{a_0\})$.

Es gilt dann, daß $min(A) \leq min^\star(A)$ für alle Mengen A natürlicher Zahlen ist. Mit dieser Symbolik gelten dann folgende Beziehungen.

Satz 7 .
Abschätzungen der Beweistiefe bezüglich PM_e:

1. *Wenn $m_e = 0$, so ist $d_{pme} = 1 + d_p$.*
2. (a) *Wenn $m_e > 0$ und $t \equiv c(\bar{x})$, so ist $d_{pme} \geq 1 + max\{1 + d_{i \in I^*}, 1 + d_p\}$.*
   (b) *Wenn $m_e > 0$ und $t \equiv c(\bar{x})$, so ist $d_{pme} \geq o_e + min^\star\{1 + d_{i \in I^*}, 1 + d_p\}$.*
3. *Wenn $m_e > 0$, $t \not\equiv c(\bar{x})$ und $I^* = \emptyset$, so ist $d_{pme} = 1 + max\{1 + d_p, d_0\}$.*
4. (a) *Wenn $m_e > 0$, $t \not\equiv c(\bar{x})$ und $I^* \neq \emptyset$, so ist $d_{pme} \geq 1 + max\{1 + d_{i \in I^*}, 2 + d_p, d_0\}$.*
   (b) *Wenn $m_e > 0$, $t \not\equiv c(\bar{x})$ und $I^* \neq \emptyset$, so ist $d_{pme} \geq o_e + min^\star\{1 + d_{i \in I^*}, 1 + d_p, d_0\}$.*

Beweis: Wenn $m_e = 0$ so ist keine Paramodulation notwendig und es wird nur das Axiom (1) angewandt. Daraus folgt *1*.

Um die $m_e + 1$ Ansatzpunkte mit Hilfe der Transitivität der Gleichheit zu erzeugen, ist wenigstens die Tiefe o_e notwendig, da sich die Anzahl der unbewiesenen Literale durch die Transitivität jeweils verdoppelt. Wegen der Verzweigung sind wenigstens zwei Ansatzpunkte in dieser Tiefe, woraus dann *2(b)* folgt. Der erste Ansatzpunkt hat wenigsten die Tiefe 1, da $m_e > 1$. Daraus folgt *2(a)*.

Da $m_e + 1 = 2$ ist, muß die Transitivität genau einmal angewandt werden, woraus unmittelbar *3* folgt.

4(a) und *4(b)* ergeben sich mit analoger Argumentation wie *2(a)* und *2(b)*. □

Satz 8 .
Abschätzungen der Beweistiefe bezüglich PM_a:

1. *Wenn $m_e = 0$, so ist $d_{pma} = 1 + d_p$.*
2. *Wenn $m_e > 0$ und $t \equiv c(\bar{x})$, so ist $d_{pma} = 1 + max\{1 + d_{i \in I}, 1 + d_p\}$.*
3. *Wenn $m_e > 0$, $t \not\equiv c(\bar{x})$ und $I^* = \emptyset$, so ist $d_{pma} = 1 + max\{1 + d_p, d_0\}$.*
4. (a) *Wenn $m_e > 0$, $t \not\equiv c(\bar{x})$ und $I^* \neq \emptyset$, so ist $1 + max\{1 + max\{d_{i \in I}\}, 2 + d_p, d_0\} \leq d_{pma} \leq 2 + max\{1 + max\{d_{i \in I}\}, 1 + d_p, d_0\}$.*
   (b) *Wenn $m_e > 0$, $t \not\equiv c(\bar{x})$ und $I^* \neq \emptyset$, so ist $d_{pma} = min\{max\{2 + max\{d_{i \in I}\}, 3 + d_p, 2 + d_0\}, max\{1 + max\{d_{i \in I}\}, 3 + d_p, 2 + d_0\}\}$.*

Beweis: Der Beweis kann genauso wie der Beweis des vorhergehenden Satzes 7 geführt werden, wobei sich die Anzahl der Ansatzpunkte von $m_e + 1$ auf $m_e - (k - 1) + 1$ reduziert, falls $k > 0$. Andernfalls bleibt sie gleich. o_e wird entsprechend der Fallunterscheidung ersetzt. Wenn $k > 0$, so wird nicht mehr $max\{1 + d_{i \in I^*}\}$ gebildet, sondern aufgrund der allgemeineren Paramodulation $1 + max\{d_{i \in I}\}$, wobei $d_i = 1$ für alle $i \in I - I^*$. Die identischen Parameter werden mit durch das Reflexivitätsaxiom der Gleichheit bewiesen. □

Satz 9 .
Die Beweistiefe bezüglich PM_i beträgt $d_{pmi} = 1 + max\{d_{i \in I} d_p, d_0\}$.

Beweis: Die Behauptung folgt unmittelbar aus der Darstellung des Axioms (2).□

Satz 10 .
Vergleich der Beweistiefen bezüglich PM_e, PM_a und PM_i:

1. *$d_{pme} \geq d_{pma}$.*

2. *Wenn $d_p > 0$, so ist $d_{pma} \geq d_{pmi}$.*

3. *Wenn $d_p = 0$ und das Axiom $f(g_1(\bar{x}), ldots, g_n(\bar{x})) = c(\bar{x})$ in dieser Form mit in die Theorie aufgenommen wird , so gilt $d_{pma} \geq d_{pmi}$.*

Beweis: Zunächst gilt, daß die Fallunterscheidung in *1*, *2*, *3* und *4* aus den vorangegangenen Sätzen 7 und 8 vollständig ist.

Wenn $m_e = 0$ (Fall *1*), so $d_{pme} = d_{pma}$. Im Fall *2* gilt $d_{pme} \geq d_{pma}$ wegen *2(a)* und weil $d_i = 1$ für alle $i \in I - I^*$ und $d_i \geq 1$ für alle $i \in I^*$. Im Fall *3* gilt trivialer Weise wieder $d_{pme} = d_{pma}$. Im Fall *4* gilt $d_{pma} = max\{1+1+max\{d_{i \in I^*}\}, 2+1+d_p, 2+d_0\}$ oder $d_{pma} = max\{2+1+max\{d_{i \in I^*}\}, 2+1+d_p, 1+d_0\}$, da die die beiden einzigen Aufspaltungen bezüglich dreier Ansatzpunkte sind. In beiden Fällen folgt, daß $d_{pme} \geq d_{pma}$, da d_{pme} wie folgt abgeschätzt werden kann. Weil $m_e \geq 2$, gilt $d_{pme} \geq max\{1 + 1 + d_{i0}, 2 + 1 + max\{d_{i \in I^* - \{i0\}}\}, 2 + 1 + d_p, 2 + d_0\}$ oder $d_{pma} \geq max\{2 + 1 + max\{d_{i \in I^*}\}, 2 + 1 + d_p, 1 + d_0\}$.

$d_{pma} \geq d_{pmi}$ folgt sofort mit der Fallunterscheidung für die Fälle *2*, *3* und *4*. Im Fall *2* gilt sogar $>$. Im Fall *1* gilt die Aussage $d_{pma} \geq d_{pmi}$ nur, wenn $d_p > 0$.

Die Bedingung $d_p > 0$ kann entfallen, sobald das oben angegebene Axiom (entspricht der ursprünglichen Axiomendarstellung (1)) mit in die Theorie aufgenommen wird. In diesem Falle ist auch $d_{pmi} = 1$. □

Bemerkung 4 .
Daß in den Konfigurationen PM_e und PM_a zur Beweistiefe der Ansatzpunkte noch Beweise der Tiefen $1 + d_i$ bzw $1 + d_p$ angehängt werden, folgt aus der Tatsache, daß an den Ansatzpunkten erst noch das Paramodulationsaxiom beziehungsweise das ursprüngliche Funktionsaxiom (1) angewandt werden muß. Für den Ansatzpunkt, an dem die Gleichheit der Funktionswerte von f nachgewiesen wird, ist dies nicht notwendig, da die Paramodulation hier bereits der Transitivität der Gleichheit entspricht. Bei drei oder mehr Ansatzpunkten kann maximal einer die Tiefe 1 haben, alle anderen haben eine größere Tiefe. Im Fall *4* liegt der Ansatzpunkt für die Anwendung des ursprünglichen Axioms (1) frühestens in der Tiefe zwei, da vorher sowohl Funktionsparameter als auch Funktionswert gleichgesetzt werden müssen. Das bedeutet, daß sowohl die linke Seite (mit wenigstens einem Split), als auch die rechte Seite der Axiomengleichung (mit genau einem Split) angeglichen werden müssen. Bei allen Betrachtungen wird jeweils die Transitivität der Gleichheit als $x = y \wedge y = z \rightarrow x = z$ angenommen, auch wenn diese bei der Konfiguration PM_a in der Form $x = y \wedge u = x \wedge v = y \rightarrow u = v$ denkbar wäre. In der Konfiguration PM_i ist die Einstellung des `ilfstate setheo_split` unbedeutend für die gemachten Untersuchungen.

Alle bis jetzt durchgeführten Betrachtungen bezogen sich auf ein umgeformtes Axiom (beziehungsweise eine umgeformte Funktionendefinition). Werden praktisch alle (bedingten) Gleichungen entsprechend umgeformt, so kann sich die Tiefenreduzierung folglich vergrößern. Hierbei muß beachtet werden, daß sich die Optimierung über alle Teile des Beweisbaumes gleichzeitig erstreckt, so daß die oben genannten Abschätzungen nicht einfach addiert werden können.

Da die Umformung der Axiome auf einer Einbeziehung der Paramodulation beruht, stellt sich die Frage, inwiefern die Paramodulationsaxiome noch gebraucht werden. Dabei sind zwei mögliche Fälle zu unterscheiden:

1. Nachweis der Gleichheit durch Berechnen der Funktionswerte und
2. Nachweis der Gleichheit durch Parametervergleich.

Im 1. Fall sind die Paramodulationsaxiome nicht mehr notwendig, da eines der Funktionsaxiome angewandt werden muß und die Paramodulation dabei realisiert werden kann. Im 2. Fall sind die Paramodulationsaxiome noch notwendig, allerdings ist die Aufgabenstellung untypisch. Um die Paramodulationsaxiome grundsätzlich einsparen zu können, ist eine entsprechende Taktik notwendig. Diese generiert für eine zu beweisende Aussage $f(p_1, \ldots, p_n) = f(q_1, \ldots, q_n)$ n neue Gleichungen $p_1 = q_1, \ldots, p_n = q_n$, wobei $p_{i=1,\ldots,n}$ und $q_{i=1,\ldots,n}$ beliebige Terme sind.

Die Transitivität der Gleichheit sollte nicht grundsätzlich ausgeschlossen werden, da im allgemeinen nicht alle Funktionen als bedingte Gleichung (1) beschrieben werden und so auch nicht in die paramodulierte Form (2) umgeformt werden können. Ein Beispiel hierfür ist die Addition, für die meist nur einige Eigenschaften, wie Kommutativität oder Assoziativität, anstelle einer vollständigen Definition angegeben werden.

Zum Abschluß sei noch ein Beispiel für die erfolgreiche Umwandlung der Axiome angegeben. Die Aufgabe ist eine reine Gleichungsumformungsaufgabe. Die Theorie wird in der ursprünglichen Darstellung angegeben. Die Umformung wird wie beschrieben durchgeführt.

Beispiel 6 :
Aufgabe: $f(g(h(g(d), b)), g(h(c, d)), g(b)) = g(a)$
Axiome:
$f(a, b, c) = d$
$g(a) = d, g(b) = c, g(c) = b, g(d) = a$
$h(a, b) = d, h(c, d) = c$

Dann werden die folgenden Konfigurationen betrachtet.

Tabelle 2: Vergleich Konfigurationen: Axiomendarstellung

Konfiguration	Tiefe	Zeit
K1	6	139.75 s
K2	5	0.30 s
K3	5	0.03 s
K4	4	0.05 s
K5	4	0.03 s

- **K1**: angegebene (ursprüngliche) Axiomendarstellung mit Paramodulationsaxiomen (einzeln) und mit Transitivität der Gleichheit.
- **K2**: umgeformte (neue) Axiomendarstellung ohne Paramodulationsaxiome, mit Transitivität der Gleichheit.
- **K3**: umgeformte (neue) Axiomendarstellung ohne Paramodulationsaxiome und ohne Transitivität der Gleichheit.
- **K4**: beide Axiomendarstellungen (neue Axiomendarstellung + Hinzunahme von Fakten) ohne Paramodulationsaxiome, mit Transitivität der Gleichheit.
- **K5**: beide Axiomendarstellungen (neue Axiomendarstellung + Hinzunahme von Fakten) ohne Paramodulationsaxiome und ohne Transitivität der Gleichheit.

In Tabelle 2 werden die benötigten Beweistiefen und Beweiszeiten SETHEO (SPARC-10, Zeiten in Sekunden) für diese Aufgabe bezüglich der einzelnen Konfigurationen dargestellt. Durch die neue Axiomendarstellung konnte nicht nur eine Tiefenstufe eingespart werden, sondern auch die benötigte Zeit wurde drastisch verkürzt. Die Hinzunahme der Fakten in den Konfiguration **K4** und **K5** verkürzt die Tiefe noch einmal um eine Stufe. Insgesamt waren die Konfigurationen **K2** bis **K5** um Größenordnungen schneller als die Konfiguration **K1**.

4 Lösung der Beispielaufgabe

In diesem Abschnitt wird die Lösung der Beispielaufgabe angegeben. Es werden die in dieser Arbeit beschriebenen Systeme und Hilfsmittel angewandt. Die zu beweisende Formel wird auf einfachere Teilprobleme zurückgeführt. Die dabei entstehenden Teilprobleme und die Rechtfertigung für diesen Schritt werden angegeben. Diese Problemzerlegung wird durchgeführt, bis das betreffende Teilproblem durch SETHEO lösbar ist.

Formeln, die Annahmen sind, (also nicht zu beweisen sind!), werden mit einem „A“ vor der Formelnummer gekennzeichnet.

Zunächst wird auf die Formel
$\forall x \in I\!N^3 \; \forall t \in I\!N : F(x,t) = G(x, \bar{\lambda}(x,t))$ [1,0]
Induktion über t angewendet ($\leadsto$ [1,1], [1,2]). Der Induktionsanfang
$\forall x \in I\!N^3 : F(x,0) = G(x, \bar{\lambda}(x,0))$ [1,2]
wird durch SETHEO im Beweis 3 gelöst. Beim Induktionsschritt [1,1] wird der Allquantor der Induktionsvariablen abbgebaut ($\leadsto$ [2,0], A[2,1]), und anschliessend Induktionsvoraussetzung und Induktionsbehauptung getrennt ($\leadsto$ [3,0], A[3,1]). An der Induktionsbehauptung [3,0] wird dann der $\forall$-Quantor bezüglich der (Zustands-)Variablen x eleminiert ($\leadsto$ [4,0], A[4,1]). Die Formel
$F(triple_1, nat_1 + 1) = G(x, \bar{\lambda}(triple_1, nat_1 + 1))$ [4,0]
wird durch SETHEO im Beweis 5 unter Zuhilfenahme von [4,2] und [4,3] gelöst. Auf die Formel
$\forall x \in I\!N^3 \; \forall t_1 \in I\!N \; \forall t_2 \in I\!N : G(x, t_1 + t_2) = G(G(x, t_1), t_2)$ [4,3]
wird Induktion (über der Variablen t_2) angewandt ($\leadsto$ [4,4], [4,5]). Der Induktionsanfang
$\forall x \in I\!N^3 \; \forall t_1 \in I\!N : G(x, t_1 + 0) = G(G(x, t_1), 0)$ [4,5]
wird durch SETHEO im Beweis 7 gelöst. Beim Induktionsschritt [4,4] wird der Allquantor der Induktionsvariablen abbgebaut ($\leadsto$ [5,0], A[5,1]). Aanschliessend werden Induktionsvoraussetzung und Induktionsbehauptung getrennt ($\leadsto$ [6,0], A[6,1]). Die Induktionsbehauptung
$\forall x \in I\!N^3 \; \forall t_1 \in I\!N : G(x, t_1 + nat_2 + 1) = G(G(x, t_1), nat_2 + 1)$ [6,0]
wird durch SETHEO im Beweis 8 gelöst.

Sei im folgenden $\alpha := F(triple_1, nat_1)$. Dann wird die Formel
$f(\alpha) = g(\alpha, l(\alpha))$ [4,2]
mittels Fallunterscheidung bewiesen ($\leadsto$ [4,6], [7,0], [7,1], [7,2]). Die Alternative
$p_3(\alpha) > 0 \vee (p_3(\alpha) = 0 \wedge p_2(\alpha) > 0) \vee (p_3(\alpha) = 0 \wedge p_2(\alpha) = 0)$ [4,6]
wird durch SETHEO im Beweis 11 gelöst. Im 1.Fall [7,2] werden Voraussetzung und Behauptung getrennt ($\leadsto$ [8,0], A[8,1]) und mit Hilfe ihrer 3 Projektion [8,2], [8,3] und [8,4] durch SETHEO im Beweis 18 gelöst. Die 3 Projektionen
$f_1(\alpha) = g_1(\alpha, l(\alpha))$ [8,2]
$f_2(\alpha) = g_2(\alpha, l(\alpha))$ [8,3]

$f_3(\alpha) = g_3(\alpha, l(\alpha))$ [8,4]
werden jeweils unter Zuhilfenahme der Formel [7,3] und der Fallannahme A[8,1] durch SETHEO im Beweis 13, 15 und 17 gelöst. Die eingefügte Formel [7,3] wird später bewiesen.

Der zweite Fall [7,1] wird ähnlich dem ersten behandelt. Zunächst werden Voraussetzung und Behauptung getrennt ($\leadsto$ [9,0], A[9,1]) und mit Hilfe der 3 Projektion [9,2], [9,3] und [9,4] durch SETHEO im Beweis 26 gelöst. Die 3 Projektionen [9,2], [9,3] und [9,4] werden wieder jeweils unter Zuhilfenahme der Formel [7,4] und der Fallannahme A[9,1] durch SETHEO im Beweis 21, 23 und 25 gelöst. Man beachte, daß die Formeln [8,2] und [9,2], [8,3] und [9,3], [8,4] und [9,4] jeweils gleich sind, die Fallannahmen A[8,1] und A[9,1] aber verschieden. Die eingefügte Formel [7,4] wird ebenfalls später bewiesen.

Im dritten Fall [7,0] werden Voraussetzung und Behauptung getrennt ($\leadsto$ [10,0], A[10,1]) und mit Hilfe der 3 Projektion [10,3], [10,4] und [10,5] durch SETHEO im Beweis 36 gelöst. Die 3 Projektionen [10,3], [10,4] und [10,5] werden jeweils unter Zuhilfenahme der Formel [10,2] und der Fallannahme A[10,1] durch SETHEO im Beweis 31, 33 und 35 gelöst. Auch hier sind wieder die Formeln [8,2] und [10,3], [8,3] und [10,4], [8,4] und [10,5] jeweils gleich, aber die Fallannahmen A[8,1] und A[10,1] verschieden. Die eingefügte Formel
$l(\alpha) = 0 + 1$ [10,2]
dient der Anwendung der Rekursionsaxiome für die $G_{i=1,2,3}$, und wird mit Hilfe der Fallannahme A[10,1] durch SETHEO im Beweis 29 gelöst.

Die eingefügte Formel
$\forall x \in I\!N^3\ \forall t \in I\!N : p_3(x) > 0 \wedge t \leq p_3(x) \rightarrow G_1(x,t) = p_1(x) + t \wedge G_2(x,t) = p_2(x) \wedge G_3(x,t) = p_3(x) - t$ [7,3]
wird mittels Induktion über t aufgelöst ($\leadsto$ [7,5], [7,6]). Der Induktionsanfang [7,6] wird durch SETHEO im Beweis 39 gelöst. Beim Induktionsschritt [7,5] wird zunächst der Allquantor der Induktionsvariablen elimiert ($\leadsto$ [11,0], A[11,1]), und dann in Induktionsvoraussetzung und Induktionsbehauptung aufgesplittet ($\leadsto$ [12,0], A[12,1]). Dann wird der Allquantor der (Zustands-)variablen elimiert ($\leadsto$ [13,0], A[13,1]), und anschliessend die Implikation in Voraussetzung und Behauptung aufgesplittet ($\leadsto$ [14,0], A[14,1]). Schließlich wird die Konjunktion rekursiv abgebaut ($\leadsto$ [14,2], [14,3] ($\leadsto$ [14,4], [14,5])). Die Formel
$G_1(triple_2, nat_3 + 1) = p_1(triple_2) + (nat_3 + 1)$ [14,2]
wird durch SETHEO im Beweis 46 mit Hilfe der Formeln [14,6] und [14,7] gelöst. Die eingefügte Formel
$inc(G_1(triple_2, nat_3)) = p_1(triple_2) + (nat_3 + 1)$ [14,7]
wird durch SETHEO im Beweis 44 mit Hilfe der Formeln [14,6] und den Annahmen A[12,1] und A[14,1] gelöst. Die Formel
$G_2(triple_2, nat_3 + 1) = p_2(triple_2)$ [14,4]
wird durch SETHEO im Beweis 48 mit Hilfe der Formel [14,6] und Annahmen

A[12,1] und A[14,1] gelöst. Die Formel
$G_3(triple_2, nat_3 + 1) = p_3(triple_2) - (nat_3 + 1)$ [14,5]
wird durch SETHEO im Beweis 54 mit Hilfe der Formeln [14,6] und [14,9] gelöst. Die eingefügte Formel
$dec(G_3(triple_2, nat_3)) = p_3(triple_2) - (nat_3 + 1)$ [14,9]
wird ihrerseits durch SETHEO im Beweis 53 mit Hilfe der Formeln [14,6] und [14,8] und den Annahmen A[12,1] und A[14,1] gelöst. Die eingefügte Formel
$(p_3(triple_2) - nat_3) - 1 = p_3(triple_2) - (nat_3 + 1), p_3(triple_2) - nat_3 > 0$ [14,8]
wird durch SETHEO im Beweis 50 und 51 mit Hilfe der Annahme A[14,1] gelöst. Die eingefügte Formel
$nat_3 \leq p_3(triple_2), G_3(triple_2, nat_3) > 0$ [14,6]
wird durch SETHEO im Beweis 41 und 42 mit Hilfe der Annahmen A[12,1] und A[14,1] gelöst.

Als letztes muß noch die eingefügte Formel
$\forall x \in I\!N^3 \; \forall t \in I\!N : p_3(x) = 0 \wedge p_2(x) > 0 \wedge t \leq p_2(x) \rightarrow G_1(x,t) = p_1(x) + t \wedge G_2(x,t) = p_2(x) - t \wedge G_3(x,t) = 0$ [7,4]
bewiesen werden. Diese wird mittels Induktion über t aufgelöst ($\leadsto$ [7,7], [7,8]). Der Induktionsanfang [7,8] wird durch SETHEO im Beweis 57 gelöst. Beim Induktionsschritt [7,7] wird zunächst der Allquantor der Induktionsvariablen elimiert ($\leadsto$ [15,0], A[15,1]), und dann in Induktionsvoraussetzung und Induktionsbehauptung aufgesplittet ($\leadsto$ [16,0], A[16,1]). Dann wird der Allquantor der (Zustands-)variablen elimiert ($\leadsto$ [17,0], A[17,1]), und anschliessend die Implikation in Voraussetzung und Behauptung aufgesplittet ($\leadsto$ [18,0], A[18,1]). Schließlich wird die Konjunktion rekursiv abgebaut ($\leadsto$ [18,2], [18,3] ($\leadsto$ [18,4], [18,5])). Die Formel
$G_1(triple_3, nat_4 + 1) = p_1(triple_3) + (nat_4 + 1)$ [18,2]
wird durch SETHEO im Beweis 65 mit Hilfe der Formeln [18,6] und [18,7] gelöst. Die eingefügte Formel
$inc(G_1(triple_3, nat_4)) = p_1(triple_3) + (nat_4 + 1)$ [18,7]
wird durch SETHEO im Beweis 63 mit Hilfe der Formel [18,6] und den Annahmen A[16,1] und A[18,1] gelöst. Die Formel
$G_2(triple_3, nat_4 + 1) = p_2(triple_3) - (nat_4 + 1)$ [18,4]
wird durch SETHEO im Beweis 72 mit Hilfe der Formeln [18,6] und [18,9] gelöst. Die eingefügte Formel
$dec(G_3(triple_2, nat_3)) = p_3(triple_2) - (nat_3 + 1)$ [18,9]
wird ihrerseits durch SETHEO im Beweis 70 mit Hilfe der Formeln [18,6] und [18,8] sowie den Annahmen A[16,1] und A[18,1] gelöst. Die eingefügte Formel
$(p_2(triple_3) - nat_4) - 1 = p_2(triple_3) - (nat_4 + 1), p_2(triple_3) - nat_4 > 0$ [18,8]
wird durch SETHEO im Beweis 67 und 68 mit Hilfe der Annahme A[18,1] gelöst. Die Formel
$G_3(triple_3, nat_4 + 1) = 0$ [18,5]
wird durch SETHEO im Beweis 73 mit Hilfe der Formel [18,6] gelöst. Die eingefügte Formel

$nat_4 \leq p_2(triple_3) \land G_3(triple_3, nat_4) = 0 \land G_2(triple_3, nat_4) > 0$ [18,6]
wird durch SETHEO im Beweis 59, 60 und 61 mit Hilfe der Annahmen A[16,1] und A[18,1] gelöst.

5 Schlußfolgerungen

Die folgende Tabelle zeigt die durch die einzelnen verschiedenen Beweiserstrategien erreichten Beweiserfolge und Beweiszeiten. Es ist zu erkennen, daß der parallele Einsatz mehrerer Beweiserparametrisierungen zu einer erheblichen Beweiszeitverkürzung führt. Das ist insbesondere beim Einsatz automatischer Beweiser in interaktiven Umgebungen unverzichtbar. Setzt man etwa voraus, daß die gesamte einem Beweiser zur Verfügung stehende Zeit, einschließlich der hier nicht mit erfaßten Compilationszeit, sich im interaktiven Betrieb in einer Größenordnung von 30 bis 60 Sekunden bewegt, werden viele Aufgaben überhaupt erst lösbar, indem ein paralleler Beweiser eingesetzt wird.

Name	dr, dyn2			dr, dyn5			dr			wdr, dyn2			wdr, dyn5			wdr		
	Tf	Inf	Zeit	Tf	Inf	Zeit	Tf	Inf	Zeit	Tf	Inf	Zeit	Tf	Inf	Zeit	Tf	Inf	Zeit
ilf_11	3	5	0.32	3	5	**0.27**	3	5	0.35	4	5	0.46	4	5	0.49	4	5	0.32
ilf_13	5	12	0.90	5	12	**0.65**	5	12	1.05	6	12	1.09	6	12	1.05	6	12	0.76
ilf_15	5	12	0.91	5	12	**0.69**	5	12	1.05	6	12	1.04	6	12	1.02	6	12	0.71
ilf_17	5	15	6.76	5	15	**4.31**	5	15	32.24	6	14	12.95	6	14	12.38	6	15	13.89
ilf_18	5	13	2.96	5	13	2.07	5	13	**1.42**	6	13	102.59	6	13	104.12	6	13	26.76
ilf_21	5	15	0.96	5	15	**0.68**	5	15	1.16	6	15	1.25	6	15	1.14	6	15	0.87
ilf_23	5	18	8.15	5	18	**5.38**	5	18	48.20	6	17	16.07	6	17	15.27	6	18	19.80
ilf_25	5	16	0.95	5	16	**0.68**	5	16	1.51	6	15	2.03	6	15	1.97	6	16	3.41
ilf_26	5	13	3.84	5	13	2.60	5	13	**1.53**	6	13	162.13	6	13	159.09	6	13	38.09
ilf_29	4	9	0.27	4	9	**0.21**	4	9	0.26	5	9	0.27	5	9	0.26	5	9	0.33
ilf_3	4	8	0.39	4	8	0.30	4	8	0.42	5	8	0.62	5	8	0.59	5	8	**0.26**
ilf_31	6	21	27.56	6	21	18.13	6	21	**17.45**	7	20	1884.54	7	20	1889.71	7		–
ilf_33	6	21	26.62	6	21	**17.79**	6	21	18.83	7	25	1998.04	7	25	2006.69	7		–
ilf_35	6	19	27.28	6	19	**18.08**	6	19	18.62	7	18	1965.88	7	18	1980.08	7		–
ilf_36	5	13	4.28	5	13	2.96	5	13	**1.49**	6	13	215.01	6	13	213.82	6	13	39.87
ilf_39	3	9	0.37	3	9	**0.26**	3	9	0.36	5	10	16.60	5	10	0.44	5	10	0.30
ilf_41	3	3	0.41	3	3	**0.30**	3	3	0.47	3	3	0.42	3	3	0.44	3	3	0.37
ilf_42	5	9	240.41	5	9	9.15	5	9	7.31	5	9	3.87	5	9	3.30	5	9	**2.11**
ilf_44	4	9	3.20	4	9	2.27	4	9	**0.43**	5	10	2.44	5	10	2.41	5	9	0.43
ilf_46	5		–	5		–	5	11	2.35	6	11	461.18	6	11	463.33	5	11	**2.18**
ilf_48	5	12	33.67	5	12	21.85	5	12	2.29	5	12	4.90	5	12	4.82	5	12	**2.14**
ilf_5	5	13	0.50	5	13	**0.33**	5	16	2.08	6	12	2.10	6	12	2.03	6	13	0.91
ilf_50	3	3	0.42	3	3	0.36	3	3	0.48	3	3	0.49	3	3	0.47	3	3	**0.32**
ilf_51	4	4	0.47	4	4	0.35	4	4	0.56	4	4	0.53	4	4	0.46	4	4	**0.34**
ilf_53	4	9	1.11	4	9	0.79	4	9	**0.63**	5	9	1.10	5	9	1.03	5	9	0.64
ilf_54	5		–	5		–	5	11	2.29	6	11	416.31	6	11	418.62	5	11	**2.21**
ilf_57	3	11	0.58	3	11	**0.30**	3	11	0.43	5	12	174.33	5	12	20.44	5	12	12.11
ilf_59	3	3	0.43	3	3	**0.33**	3	3	0.45	3	3	0.47	3	3	0.45	3	3	0.35
ilf_60	4	6	0.60	4	6	0.43	4	6	0.57	4	6	0.56	4	6	0.53	4	6	**0.38**
ilf_61	5	10	324.71	5	11	18.60	5	11	18.51	5	10	3.85	5	10	3.68	5	10	**2.42**
ilf_63	4	10	4.20	4	10	2.63	4	10	0.50	5	11	3.03	5	11	2.99	5	10	**0.44**
ilf_65	5		–	5		–	5	14	**2.43**	6		–	6		–	6	14	8.03
ilf_67	3	3	0.44	3	3	**0.34**	3	3	0.52	3	3	0.46	3	3	0.55	3	3	0.34
ilf_68	4	4	0.48	4	4	0.37	4	4	0.49	4	4	0.51	4	4	0.57	4	4	**0.35**
ilf_7	3	5	0.20	3	5	**0.14**	3	5	0.19	4	5	0.27	4	5	0.24	4	5	0.20
ilf_70	4	10	1.11	4	10	0.76	4	10	**0.46**	5	10	0.99	5	10	0.96	5	10	0.54
ilf_72	5		–	5		–	5	14	**2.39**	6		–	6		–	6	14	8.06
ilf_73	5		–	5	10	20.28	5	10	2.32	5	8	1.74	5	8	1.75	5	8	**0.53**
ilf_8	4	10	0.71	4	10	**0.45**	4	10	0.76	5	10	2.52	5	10	2.86	5	10	1.64

Prinzipiell wurde durch die in dieser Arbeit beschriebene Studie nachgewiesen, daß der Einsatz von automatischer Beweisertechnologie in einer interaktiven Beweisumgebung ein handhabbares Mittel zur Erfüllung von Verifikationsaufgaben ist.

Anhang: Ein Beispiel für einen maschinell gefundenen Teilbeweis

Im folgenden sei der Beweis für den oben beschriebenen Beweisereinsatz 33 protokolliert. Dieser Beweis wurde deshalb ausgewählt, weil er einer der komplexeren Beweise ist und an den automatischen Beweiser von den gelösten Aufgaben mit die höchsten Anforderungen stellte.

Das Beweisziel dieses Problems ist die Gleichheit der 2.Komponenten der Terme $f(\alpha)$ und $G(\alpha, l(\alpha))$ im (3.)Fall $\alpha \in Z_4$, d.h. die zweite und dritte Komponente von α ist jeweils Null. Aus der Tatsache $\alpha \in Z_4$ ergibt sich $l(\alpha) = 1$ und somit $G(\alpha, l(\alpha)) = g(\alpha)$. Die Gleichheit der 2.Komponenten der Terme $f(\alpha)$ und $g(\alpha)$ folgt dann aus den partiellen Überführungsfunktionen.

Die hohen Anforderungen für den automatischen Beweiser ergeben sich aus der Tiefe der Termstrukturen. Die unten angegebene Theorie ermöglicht es nur in kleinen Schritten, diese Strukturen abzubauen. Dieses geschieht durch vielfache Anwendung der split-Axiome sowie der Transitivität der Gleichheit.

Die .lop-Formulierung des Problems

```
#clausename ax(f2,?,def_fu,definition1)
eqq(to(nat,nat),f2(tr_nat,_984),p2(tr_nat,_984)) <-
        greater(nat,p3(tr_nat,_984),cc0).
#clausename ax(f2,?,def_fu,definition2)
eqq(to(nat,nat),f2(tr_nat,_984),cc0) <-
        eqq(to(nat,nat),p3(tr_nat,_984),cc0),
        greater(nat,p2(tr_nat,_984),cc0).
#clausename ax(f2,?,def_fu,definition3)
eqq(to(nat,nat),f2(tr_nat,_984),p2(tr_nat,_984)) <-
        eqq(to(nat,nat),p3(tr_nat,_984),cc0),
        eqq(to(nat,nat),p2(tr_nat,_984),cc0).
#clausename ax(g2,?,def_fu,definition1)
eqq(to(nat,nat),g2(tr_nat,_984),p2(tr_nat,_984)) <-
        greater(nat,p3(tr_nat,_984),cc0).
#clausename ax(g2,?,def_fu,definition2)
eqq(to(nat,nat),g2(tr_nat,_984),dec(nat,p2(tr_nat,_984))) <-
        eqq(to(nat,nat),p3(tr_nat,_984),cc0),
        greater(nat,p2(tr_nat,_984),cc0).
#clausename ax(g2,?,def_fu,definition3)
eqq(to(nat,nat),g2(tr_nat,_984),p2(tr_nat,_984)) <-
        eqq(to(nat,nat),p3(tr_nat,_984),cc0),
        eqq(to(nat,nat),p2(tr_nat,_984),cc0).
#clausename ax(gg,ra,rek_def,rekursionsanfang)
```

```
eqq(to(tr_nat,tr_nat),gg(tr_nat,_984,cc0),_984).
#clausename ax(gg,rs,rek_def,rekursionsschritt)
eqq(to(tr_nat,tr_nat),gg(tr_nat,_984,plus(nat,_987,cc1)), \
                g(tr_nat,gg(tr_nat,_984,_987))).
#clausename ax(gg2,ra,rek_def,rekursionsanfang)
eqq(to(nat,nat),gg2(tr_nat,_984,cc0),p2(tr_nat,_984)).
#clausename ax(gg2,rs,rek_def,rekursionsschritt)
eqq(to(nat,nat),gg2(tr_nat,_984,plus(nat,_987,cc1)), \
                g2(tr_nat,k(tr_nat,gg1(tr_nat,_984,_987), \
                gg2(tr_nat,_984,_987),gg3(tr_nat,_984,_987)))).
#clausename ax(gg,k,lem,komposition)
eqq(to(tr_nat,tr_nat),gg(tr_nat,_984,_985), \
                k(tr_nat,gg1(tr_nat,_984,_985), \
                gg2(tr_nat,_984,_985),gg3(tr_nat,_984,_985))).
#clausename A[3,1]
eqq(to(tr_nat,tr_nat),ff(tr_nat,_984,nat1(skolem)), \
                gg(tr_nat,_984,lambdaq(tr_nat,_984,nat1(skolem)))).
#clausename A[10,1] 1.Teil
eqq(to(nat,nat),p3(tr_nat,ff(tr_nat,tr_nat1(skolem), \
                nat1(skolem))),cc0).
#clausename A[10,1] 2.Teil
eqq(to(nat,nat),p2(tr_nat,ff(tr_nat,tr_nat1(skolem), \
                nat1(skolem))),cc0).
#clausename [10,2]
eqq(to(nat,nat),l(tr_nat,ff(tr_nat,tr_nat1(skolem),nat1(skolem))), \
                plus(nat,cc0,cc1)).
#clausename ref
eqq(to(_973,_973),_975,_975).
#clausename sym
eqq(to(_973,_974),_975,_976) <-
        eqq(to(_974,_973),_976,_975).
#clausename trans
eqq(to(_973,_974),_975,_976) <-
        eqq(to(_973,_981),_975,_983),
        eqq(to(_981,_974),_983,_976).
#clausename split(+(nat),1)
eqq(to(nat,nat),plus(nat,_981,_982),plus(nat,_985,_982)) <-
        eqq(to(nat,nat),_981,_985).
#clausename split(+(nat),2)
eqq(to(nat,nat),plus(nat,_981,_982),plus(nat,_981,_986)) <-
        eqq(to(nat,nat),_982,_986).
#clausename split(>(nat),1)
greater(nat,_977,_978) <-
        greater(nat,_982,_978),
        eqq(to(nat,nat),_977,_982).
```

```
#clausename split(>(nat),1)
greater(nat,_977,_978) <-
        greater(nat,_977,_983),
        eqq(to(nat,nat),_978,_983).
#clausename split(dec(nat),1)
eqq(to(nat,nat),dec(nat,_981),dec(nat,_984)) <-
        eqq(to(nat,nat),_981,_984).
#clausename split(f(tr_nat),1)
eqq(to(nat,nat),f2(tr_nat,_981),f2(tr_nat,_984)) <-
        eqq(to(tr_nat,tr_nat),_981,_984).
#clausename split(ff(tr_nat),1)
eqq(to(tr_nat,tr_nat),ff(tr_nat,_981,_982),ff(tr_nat,_985,_982)) <-
        eqq(to(tr_nat,tr_nat),_981,_985).
#clausename split(ff(tr_nat),2)
eqq(to(tr_nat,tr_nat),ff(tr_nat,_981,_982),ff(tr_nat,_981,_986)) <-
        eqq(to(nat,nat),_982,_986).
#clausename split(g(tr_nat),1)
eqq(to(tr_nat,tr_nat),g(tr_nat,_981),g(tr_nat,_984)) <-
        eqq(to(tr_nat,tr_nat),_981,_984).
#clausename split(g2(nat),1)
eqq(to(nat,nat),g2(tr_nat,_981),g2(tr_nat,_984)) <-
        eqq(to(tr_nat,tr_nat),_981,_984).
#clausename split(gg(tr_nat),1)
eqq(to(tr_nat,tr_nat),gg(tr_nat,_981,_982),gg(tr_nat,_985,_982)) <-
        eqq(to(tr_nat,tr_nat),_981,_985).
#clausename split(gg(tr_nat),2)
eqq(to(tr_nat,tr_nat),gg(tr_nat,_981,_982),gg(tr_nat,_981,_986)) <-
        eqq(to(nat,nat),_982,_986).
#clausename split(gg1(nat),1)
eqq(to(nat,nat),gg1(tr_nat,_981,_982),gg1(tr_nat,_985,_982)) <-
        eqq(to(tr_nat,tr_nat),_981,_985).
#clausename split(gg1(nat),2)
eqq(to(nat,nat),gg1(tr_nat,_981,_982),gg1(tr_nat,_981,_986)) <-
        eqq(to(nat,nat),_982,_986).
#clausename split(gg2(nat),1)
eqq(to(nat,nat),gg2(tr_nat,_981,_982),gg2(tr_nat,_985,_982)) <-
        eqq(to(tr_nat,tr_nat),_981,_985).
#clausename split(gg2(nat),2)
eqq(to(nat,nat),gg2(tr_nat,_981,_982),gg2(tr_nat,_981,_986)) <-
        eqq(to(nat,nat),_982,_986).
#clausename split(gg3(nat),1)
eqq(to(nat,nat),gg3(tr_nat,_981,_982),gg3(tr_nat,_985,_982)) <-
        eqq(to(tr_nat,tr_nat),_981,_985).
#clausename split(gg3(nat),2)
eqq(to(nat,nat),gg3(tr_nat,_981,_982),gg3(tr_nat,_981,_986)) <-
```

```
        eqq(to(nat,nat),_982,_986).
#clausename split(k(tr_nat),1)
eqq(to(tr_nat,tr_nat),k(tr_nat,_981,_982,_983), \
                k(tr_nat,_986,_982,_983)) <-
        eqq(to(nat,nat),_981,_986).
#clausename split(k(tr_nat),2)
eqq(to(tr_nat,tr_nat),k(tr_nat,_981,_982,_983), \
                k(tr_nat,_981,_987,_983)) <-
        eqq(to(nat,nat),_982,_987).
#clausename split(k(tr_nat),3)
eqq(to(tr_nat,tr_nat),k(tr_nat,_981,_982,_983), \
                k(tr_nat,_981,_982,_988)) <-
        eqq(to(nat,nat),_983,_988).
#clausename split(l(nat),1)
eqq(to(nat,nat),l(tr_nat,_981),l(tr_nat,_984)) <-
        eqq(to(tr_nat,tr_nat),_981,_984).
#clausename split(lambdaq(nat),1)
eqq(to(nat,nat),lambdaq(tr_nat,_981,_982),lambdaq(tr_nat,_985,_982)) <-
        eqq(to(tr_nat,tr_nat),_981,_985).
#clausename split(lambdaq(nat),2)
eqq(to(nat,nat),lambdaq(tr_nat,_981,_982),lambdaq(tr_nat,_981,_986)) <-
        eqq(to(nat,nat),_982,_986).
#clausename split(p2(nat),1)
eqq(to(nat,nat),p2(tr_nat,_981),p2(tr_nat,_984)) <-
        eqq(to(tr_nat,tr_nat),_981,_984).
#clausename split(p3(nat),1)
eqq(to(nat,nat),p3(tr_nat,_981),p3(tr_nat,_984)) <-
        eqq(to(tr_nat,tr_nat),_981,_984).
#clausename goal(33)
<- eqq(to(nat,nat),f2(tr_nat,ff(tr_nat,tr_nat1(skolem),nat1(skolem))),\
                gg2(tr_nat,ff(tr_nat,tr_nat1(skolem),nat1(skolem)), \
                l(tr_nat,ff(tr_nat,tr_nat1(skolem),nat1(skolem))))).
```

Das Beweisprotokoll

```
SAM V3.3 Copyright TU Munich (December 22, 1995)

Options : -dr -cons -dynsgreord 5 ilf_33

using antilemma-constraints
using regularity-constraints
using tautology-constraints
using subsumption-constraints

Start proving...

-d:    2  time     < 0.01 sec  inferences =       23  fails =       84
-d:    3  time     < 0.01 sec  inferences =      193  fails =      376
-d:    4  time =     0.11 sec  inferences =     2317  fails =     4442
-d:    5  time =    10.91 sec  inferences =   205908  fails =   367366
-d:    6  time =     6.58 sec  inferences =   124415  fails =   226970

                    ******** SUCCESS ************

Number of inferences in proof      :       21
        - E/R/F/L                  :       21/       0/       0/      0
Intermediate free variables        :        8
Intermediate inferences            :       23
Intermediate open subgoals         :        5
Generated antilemmata              :     2353
Number of unifications             :   332856
    - E/R/F/L                      :   256931/       0/   75925/      0
Number of generated constraints    :    27280
    - anl/reg/ts                   :     5407/   12245/    9628
Number of fails                    :   599238
    - unification                  :   296022
    - depth bound                  :   280406
    - constraints                  :    22810
        - anl/reg/ts               :     1594/    7658/   13558
Number of folding operations       :     2648
    - one level                    :        1
    - root                         :     2648
Instructions executed              :  1129102
Abstract machine time (seconds)    :    17.61
Overall time          (seconds)    :    17.79
```

Der maschinell generierte Beweisbaum

```
[
[~query__,[ 0 , ext__(0.1,45.1) ] ,[
  [ query__ ] ,
  [~eqq(to(nat,nat),f2(tr_nat,ff(tr_nat,tr_nat1(sk),nat1(sk))), \
              gg2(tr_nat,ff(tr_nat,tr_nat1(sk),nat1(sk)), \
              l(tr_nat,ff(tr_nat,tr_nat1(sk),nat1(sk))))),[ 1 , ext__(45.2,17.1) ] ,[
    [ eqq(to(nat,nat),f2(tr_nat,ff(tr_nat,tr_nat1(sk),nat1(sk))), \
              gg2(tr_nat,ff(tr_nat,tr_nat1(sk),nat1(sk)), \
              l(tr_nat,ff(tr_nat,tr_nat1(sk),nat1(sk))))) ] ,
    [~eqq(to(nat,nat),gg2(tr_nat,ff(tr_nat,tr_nat1(sk),nat1(sk)), \
              l(tr_nat,ff(tr_nat,tr_nat1(sk),nat1(sk)))), \
              f2(tr_nat,ff(tr_nat,tr_nat1(sk),nat1(sk)))),[ 2 , ext__(17.2,18.1) ] ,[
      [ eqq(to(nat,nat),gg2(tr_nat,ff(tr_nat,tr_nat1(sk),nat1(sk)), \
              l(tr_nat,ff(tr_nat,tr_nat1(sk),nat1(sk)))), \
              f2(tr_nat,ff(tr_nat,tr_nat1(sk),nat1(sk)))) ] ,
      [~eqq(to(nat,nat),gg2(tr_nat,ff(tr_nat,tr_nat1(sk),nat1(sk)), \
              l(tr_nat,ff(tr_nat,tr_nat1(sk),nat1(sk)))), \
              g2(tr_nat,gg(tr_nat,ff(tr_nat,tr_nat1(sk),nat1(sk)),cc0))), \
              [ 3 , ext__(18.2,18.1) ] ,[
        [ eqq(to(nat,nat),gg2(tr_nat,ff(tr_nat,tr_nat1(sk),nat1(sk)), \
              l(tr_nat,ff(tr_nat,tr_nat1(sk),nat1(sk)))), \
              g2(tr_nat,gg(tr_nat,ff(tr_nat,tr_nat1(sk),nat1(sk)),cc0))) ] ,
        [~eqq(to(nat,nat),gg2(tr_nat,ff(tr_nat,tr_nat1(sk),nat1(sk)), \
              l(tr_nat,ff(tr_nat,tr_nat1(sk),nat1(sk)))), \
              g2(tr_nat,k(tr_nat,gg1(tr_nat,ff(tr_nat,tr_nat1(sk),nat1(sk)),cc0), \
              gg2(tr_nat,ff(tr_nat,tr_nat1(sk),nat1(sk)),cc0), \
              gg3(tr_nat,ff(tr_nat,tr_nat1(sk),nat1(sk)),cc0)))),[ 4 , ext__(18.2,18.1) ] ,[
          [ eqq(to(nat,nat),gg2(tr_nat,ff(tr_nat,tr_nat1(sk),nat1(sk)), \
              l(tr_nat,ff(tr_nat,tr_nat1(sk),nat1(sk)))), \
              g2(tr_nat,k(tr_nat,gg1(tr_nat,ff(tr_nat,tr_nat1(sk),nat1(sk)),cc0), \
              gg2(tr_nat,ff(tr_nat,tr_nat1(sk),nat1(sk)),cc0), \
              gg3(tr_nat,ff(tr_nat,tr_nat1(sk),nat1(sk)),cc0)))) ] ,
          [~eqq(to(nat,nat),gg2(tr_nat,ff(tr_nat,tr_nat1(sk),nat1(sk)), \
              l(tr_nat,ff(tr_nat,tr_nat1(sk),nat1(sk)))), \
              gg2(tr_nat,ff(tr_nat,tr_nat1(sk),nat1(sk)),plus(nat,cc0,cc1))), \
              [ 5 , ext__(18.2,34.1) ] ,[
            [ eqq(to(nat,nat),gg2(tr_nat,ff(tr_nat,tr_nat1(sk),nat1(sk)), \
              l(tr_nat,ff(tr_nat,tr_nat1(sk),nat1(sk)))), \
              gg2(tr_nat,ff(tr_nat,tr_nat1(sk),nat1(sk)),plus(nat,cc0,cc1))) ] ,
            [~eqq(to(nat,nat),l(tr_nat,ff(tr_nat,tr_nat1(sk),nat1(sk))),plus(nat,cc0,cc1)), \
              [ 6 , ext__(34.2,15.1) ] ,[
              [ eqq(to(nat,nat),l(tr_nat,ff(tr_nat,tr_nat1(sk),nat1(sk))),plus(nat,cc0,cc1)) ]
            ] ]

          ] ],
          [~eqq(to(nat,nat),gg2(tr_nat,ff(tr_nat,tr_nat1(sk),nat1(sk)),plus(nat,cc0,cc1)), \
              g2(tr_nat,k(tr_nat,gg1(tr_nat,ff(tr_nat,tr_nat1(sk),nat1(sk)),cc0), \
              gg2(tr_nat,ff(tr_nat,tr_nat1(sk),nat1(sk)),cc0), \
              gg3(tr_nat,ff(tr_nat,tr_nat1(sk),nat1(sk)),cc0)))),[ 7 , ext__(18.3,10.1) ] ,[
            [ eqq(to(nat,nat),gg2(tr_nat,ff(tr_nat,tr_nat1(sk),nat1(sk)),plus(nat,cc0,cc1)), \
              g2(tr_nat,k(tr_nat,gg1(tr_nat,ff(tr_nat,tr_nat1(sk),nat1(sk)),cc0), \
              gg2(tr_nat,ff(tr_nat,tr_nat1(sk),nat1(sk)),cc0), \
              gg3(tr_nat,ff(tr_nat,tr_nat1(sk),nat1(sk)),cc0)))) ]
          ] ]

        ] ],
        [~eqq(to(nat,nat),g2(tr_nat,k(tr_nat,gg1(tr_nat,ff(tr_nat,tr_nat1(sk),nat1(sk)),cc0), \
              gg2(tr_nat,ff(tr_nat,tr_nat1(sk),nat1(sk)),cc0), \
              gg3(tr_nat,ff(tr_nat,tr_nat1(sk),nat1(sk)),cc0))), \
              g2(tr_nat,gg(tr_nat,ff(tr_nat,tr_nat1(sk),nat1(sk)),cc0))),[ 8,ext_(18.3,17.1)],[
          [ eqq(to(nat,nat),g2(tr_nat,k(tr_nat,gg1(tr_nat,ff(tr_nat,tr_nat1(sk),nat1(sk)),cc0), \
              gg2(tr_nat,ff(tr_nat,tr_nat1(sk),nat1(sk)),cc0), \
```

```
          gg3(tr_nat,ff(tr_nat,tr_nat1(sk),nat1(sk)),cc0))), \
          g2(tr_nat,gg(tr_nat,ff(tr_nat,tr_nat1(sk),nat1(sk)),cc0))) ] ,
    [~eqq(to(nat,nat),g2(tr_nat,gg(tr_nat,ff(tr_nat,tr_nat1(sk),nat1(sk)),cc0)), \
          g2(tr_nat,k(tr_nat,gg1(tr_nat,ff(tr_nat,tr_nat1(sk),nat1(sk)),cc0), \
          gg2(tr_nat,ff(tr_nat,tr_nat1(sk),nat1(sk)),cc0), \
          gg3(tr_nat,ff(tr_nat,tr_nat1(sk),nat1(sk)),cc0)))),[ 9 , ext__(17.2,28.1) ] ,[
      [ eqq(to(nat,nat),g2(tr_nat,gg(tr_nat,ff(tr_nat,tr_nat1(sk),nat1(sk)),cc0)), \
          g2(tr_nat,k(tr_nat,gg1(tr_nat,ff(tr_nat,tr_nat1(sk),nat1(sk)),cc0), \
          gg2(tr_nat,ff(tr_nat,tr_nat1(sk),nat1(sk)),cc0), \
          gg3(tr_nat,ff(tr_nat,tr_nat1(sk),nat1(sk)),cc0)))) ] ,
      [~eqq(to(tr_nat,tr_nat),gg(tr_nat,ff(tr_nat,tr_nat1(sk),nat1(sk)),cc0), \
          k(tr_nat,gg1(tr_nat,ff(tr_nat,tr_nat1(sk),nat1(sk)),cc0), \
          gg2(tr_nat,ff(tr_nat,tr_nat1(sk),nat1(sk)),cc0), \
          gg3(tr_nat,ff(tr_nat,tr_nat1(sk),nat1(sk)),cc0))),[ 10 , ext__(28.2,11.1) ] ,[
        [ eqq(to(tr_nat,tr_nat),gg(tr_nat,ff(tr_nat,tr_nat1(sk),nat1(sk)),cc0), \
          k(tr_nat,gg1(tr_nat,ff(tr_nat,tr_nat1(sk),nat1(sk)),cc0), \
          gg2(tr_nat,ff(tr_nat,tr_nat1(sk),nat1(sk)),cc0), \
          gg3(tr_nat,ff(tr_nat,tr_nat1(sk),nat1(sk)),cc0))) ]
      ] ]

    ] ]

  ] ]

] ],
[~eqq(to(nat,nat),g2(tr_nat,gg(tr_nat,ff(tr_nat,tr_nat1(sk),nat1(sk)),cc0)), \
          f2(tr_nat,ff(tr_nat,tr_nat1(sk),nat1(sk)))),[ 11 , ext__(18.3,18.1) ] ,[
  [ eqq(to(nat,nat),g2(tr_nat,gg(tr_nat,ff(tr_nat,tr_nat1(sk),nat1(sk)),cc0)), \
          f2(tr_nat,ff(tr_nat,tr_nat1(sk),nat1(sk)))) ] ,
  [~eqq(to(nat,nat),g2(tr_nat,gg(tr_nat,ff(tr_nat,tr_nat1(sk),nat1(sk)),cc0)), \
          p2(tr_nat,ff(tr_nat,tr_nat1(sk),nat1(sk)))),[ 12 , ext__(18.2,18.1) ] ,[
    [ eqq(to(nat,nat),g2(tr_nat,gg(tr_nat,ff(tr_nat,tr_nat1(sk),nat1(sk)),cc0)), \
          p2(tr_nat,ff(tr_nat,tr_nat1(sk),nat1(sk)))) ] ,
    [~eqq(to(nat,nat),g2(tr_nat,gg(tr_nat,ff(tr_nat,tr_nat1(sk),nat1(sk)),cc0)), \
          g2(tr_nat,ff(tr_nat,tr_nat1(sk),nat1(sk)))),[ 13 , ext__(18.2,28.1) ] ,[
      [ eqq(to(nat,nat),g2(tr_nat,gg(tr_nat,ff(tr_nat,tr_nat1(sk),nat1(sk)),cc0)), \
          g2(tr_nat,ff(tr_nat,tr_nat1(sk),nat1(sk)))) ] ,
      [~eqq(to(tr_nat,tr_nat),gg(tr_nat,ff(tr_nat,tr_nat1(sk),nat1(sk)),cc0), \
          ff(tr_nat,tr_nat1(sk),nat1(sk))),[ 14 , ext__(28.2,7.1) ] ,[
        [ eqq(to(tr_nat,tr_nat),gg(tr_nat,ff(tr_nat,tr_nat1(sk),nat1(sk)),cc0), \
          ff(tr_nat,tr_nat1(sk),nat1(sk))) ]
      ] ]

    ] ],
    [~eqq(to(nat,nat),g2(tr_nat,ff(tr_nat,tr_nat1(sk),nat1(sk))), \
          p2(tr_nat,ff(tr_nat,tr_nat1(sk),nat1(sk)))),[ 15 , ext__(18.3,6.1) ] ,[
      [ eqq(to(nat,nat),g2(tr_nat,ff(tr_nat,tr_nat1(sk),nat1(sk))), \
          p2(tr_nat,ff(tr_nat,tr_nat1(sk),nat1(sk)))) ] ,
      [~eqq(to(nat,nat),p3(tr_nat,ff(tr_nat,tr_nat1(sk),nat1(sk))),cc0), \
          [ 16 , ext__(6.2,13.1) ] ,[
        [ eqq(to(nat,nat),p3(tr_nat,ff(tr_nat,tr_nat1(sk),nat1(sk))),cc0) ]
      ] ],
      [~eqq(to(nat,nat),p2(tr_nat,ff(tr_nat,tr_nat1(sk),nat1(sk))),cc0), \
          [ 17 , ext__(6.3,14.1) ] ,[
        [ eqq(to(nat,nat),p2(tr_nat,ff(tr_nat,tr_nat1(sk),nat1(sk))),cc0) ]
      ] ]

    ] ]

  ] ],
  [~eqq(to(nat,nat),p2(tr_nat,ff(tr_nat,tr_nat1(sk),nat1(sk))), \
          f2(tr_nat,ff(tr_nat,tr_nat1(sk),nat1(sk)))),[ 18 , ext__(18.3,17.1) ] ,[
    [ eqq(to(nat,nat),p2(tr_nat,ff(tr_nat,tr_nat1(sk),nat1(sk))), \
          f2(tr_nat,ff(tr_nat,tr_nat1(sk),nat1(sk)))) ] ,
    [~eqq(to(nat,nat),f2(tr_nat,ff(tr_nat,tr_nat1(sk),nat1(sk))), \
```

```
            p2(tr_nat,ff(tr_nat,tr_nat1(sk),nat1(sk)))),[ 19 , ext__(17.2,3.1) ] ,[
          [ eqq(to(nat,nat),f2(tr_nat,ff(tr_nat,tr_nat1(sk),nat1(sk))), \
            p2(tr_nat,ff(tr_nat,tr_nat1(sk),nat1(sk)))) ] ,
          [~eqq(to(nat,nat),p3(tr_nat,ff(tr_nat,tr_nat1(sk),nat1(sk))),cc0), \
            [ 20 , ext__(3.2,13.1) ] ,[
           [ eqq(to(nat,nat),p3(tr_nat,ff(tr_nat,tr_nat1(sk),nat1(sk))),cc0) ]
          ] ],
          [~eqq(to(nat,nat),p2(tr_nat,ff(tr_nat,tr_nat1(sk),nat1(sk))),cc0), \
            [ 21 , ext__(3.3,14.1) ] ,[
           [ eqq(to(nat,nat),p2(tr_nat,ff(tr_nat,tr_nat1(sk),nat1(sk))),cc0) ]
          ] ]

         ] ]

        ] ]

       ] ]

      ] ]

     ] ]

    ] ]

    ].
```

Beweis in natürlichsprachiger Darstellung

A Proof from the ILF Server*

Setheo
Kmoch

April 8, 1997

Axiom 0.1 (*handle*[10, 2]) $l(F(triple_1, nat_1)) = 0 + 1$.

Axiom 0.2 ($split(gg2(nat), 2)$) $(A = B) \ \to \ (G_2(C, A) = G_2(C, B))$.

Axiom 0.3 (*recursionstep of* G_2)
$G_2(A, B + 1) = g_2(k(G_1(A, B), G_2(A, B), G_3(A, B)))$.

Axiom 0.4 (*transitivity*) $(A = B) \ \wedge \ (B = C) \ \to \ (A = C)$.

Axiom 0.5 (*lemma of composition of* G)
$G(A, B) = k(G_1(A, B), G_2(A, B), G_3(A, B))$.

Axiom 0.6 ($split(g2(nat), 1)$) $(A = B) \ \to \ (g_2(A) = g_2(B))$.

Axiom 0.7 (*symmetry*) $(A = B) \ \to \ (B = A)$.

Axiom 0.8 (*recursionstart of* G) $G(A, 0) = A$.

Axiom 0.9 (*assumption*[10, 1]) $p_3(F(triple_1, nat_1)) = 0$.

Axiom 0.10 (*assumption*[10, 1]) $p_2(F(triple_1, nat_1)) = 0$.

Axiom 0.11 (*definition* 3 *of* g_2) $(p_3(A) = 0) \ \wedge \ (p_2(A) = 0) \ \to \ (g_2(A) = p_2(A))$.

Axiom 0.12 (*definition* 3 *of* f_2) $(p_3(A) = 0) \ \wedge \ (p_2(A) = 0) \ \to \ (f_2(A) = p_2(A))$.

Theorem 0.1 $f_2(F(triple_1, nat_1)) = G_2(F(triple_1, nat_1), l(F(triple_1, nat_1)))$.

*This manuscript was generated by ILF. The development of ILF was supported by the Deutsche Forschungsgemeinschaft. For information on ILF contact gehne@mathematik.hu-berlin.de.

1

Proof[1]. We show directly that

$$f_2(F(triple_1, nat_1)) = G_2(F(triple_1, nat_1), l(F(triple_1, nat_1))). \tag{1}$$

Because of $split(gg2(nat), 2)$ and by $handle[10, 2]$

$$G_2(F(triple_1, nat_1), l(F(triple_1, nat_1))) = G_2(F(triple_1, nat_1), 0 + 1). \tag{2}$$

Because of *definition 3 of* f_2, $assumption[10, 1]$, and by $assumption[10, 1]$ $f_2(F(triple_1, nat_1)) = p_2(F(triple_1, nat_1))$. Hence by *symmetry*

$$p_2(F(triple_1, nat_1)) = f_2(F(triple_1, nat_1)). \tag{3}$$

Because of *definition 3 of* g_2, $assumption[10, 1]$, and by $assumption[10, 1]$ $g_2(F(triple_1, nat_1)) = p_2(F(triple_1, nat_1))$. Because of $split(g2(nat), 1)$ and by *recursionstart of G* $g_2(G(F(triple_1, nat_1), 0)) = g_2(F(triple_1, nat_1))$. Therefore by *transitivity* $g_2(G(F(triple_1, nat_1), 0)) = p_2(F(triple_1, nat_1))$. Hence by *transitivity* and by (3)

$$g_2(G(F(triple_1, nat_1), 0)) = f_2(F(triple_1, nat_1)). \tag{4}$$

Because of $split(g2(nat), 1)$ and by *lemma of composition of G* $g_2(G(F(triple_1, nat_1), 0)) = g_2(k(G_1(F(triple_1, nat_1), 0), G_2(F(triple_1, nat_1), 0), G_3(F(triple_1, nat_1), 0)))$. Hence by *symmetry* $g_2(k(G_1(F(triple_1, nat_1), 0), G_2(F(triple_1, nat_1), 0), G_3(F(triple_1, nat_1), 0))) = g_2(G(F(triple_1, nat_1), 0))$. By (2), by *transitivity*, and by *recursionstep of* G_2 $G_2(F(triple_1, nat_1), l(F(triple_1, nat_1))) = g_2(k(G_1(F(triple_1, nat_1), 0), G_2(F(triple_1, nat_1), 0), G_3(F(triple_1, nat_1), 0)))$. Therefore by *transitivity* $G_2(F(triple_1, nat_1), l(F(triple_1, nat_1))) = g_2(G(F(triple_1, nat_1), 0))$. Hence by *transitivity* and by (4) $G_2(F(triple_1, nat_1), l(F(triple_1, nat_1))) = f_2(F(triple_1, nat_1))$. Hence by *symmetry* $f_2(F(triple_1, nat_1)) = G_2(F(triple_1, nat_1), l(F(triple_1, nat_1)))$. Thus we have completed the proof of (1).

q.e.d.

[1] Setheo and Ilf

Literatur

[1] Dahn, B. I.; Gehne, J.; Honigmann, Th.; Walther, L.; Wolf, A.: Integrating Logical Functions with ILF; Preprint 94-10, Humboldt University Berlin, Mathematical Institute.

[2] Dahn, B. I.; Wolf, A.: A Calculus Supporting Structured Proofs; Journal for Information Processing and Cybernetics (EIK), (5–6): pp. 261–276, 1994.

[3] Dahn, B. I.; Wolf, A.: Natural Language Presentation and Combination of Automatically Generated Proofs; In: 1st Int. Workshop Frontiers of Combining Systems, Kluwer, 1996.

[4] Geist, A.; Beguelin, A.; Dongarra, J.; Jiang, W.; Manchek, R.; Sunderam, V.: PVM: Parallel Virtual Machine. A Users' Guide and Tutorial for Networked Parallel Computing; MIT Press, Cambridge, London, 1994.

[5] Gentzen, G.: Untersuchungen über das logische Schließen, Mathematische Zeitschrift 39, Berlin, 1935.

[6] Grabowski, J.; Jantke, K. P.; Thiele, H.: Grundlagen der Künstlichen Intelligenz. Akademie Verlag, Berlin, 1989.

[7] Harman, N. A.; Tucker, J. V.: Algebraic Models and the Correctness of Microprocessors in Correct Hardware Design and Verification Methods. In: Proceedings CHARME'93, Arles, Springer, 1993.

[8] Letz, R.; Schumann, J.; Bayerl, S.; Bibel, W.: SETHEO: A High-Performance Theorem Prover, In: Journal of Automated Reasoning, 8, 1992.

[9] Loveland, D. W.: Automated Theorem Proving: a Logical Basis; North-Holland, 1978.

[10] Meinel, Ch.: Effiziente Algorithmen. Sektion Mathematik der Humboldt-Universität zu Berlin, Seminarbericht Nr. 61, 1984.

[11] Shoenfield, J. R.: Mathematical Logic. Addison Wesley, 1973.

[12] Sominskij, I. S.; Golovina, L. I.; Jaglom, I. M.: Die vollständige Induktion. Deutscher Verlag der Wissenschaften, Berlin, 1986.

[13] Starke, P. H.: Abstrakte Automaten. Deutscher Verlag der Wissenschaften, Berlin, 1969.

[14] Starke, P. H.: Analyse von Petri-Netz-Modellen. Teubner, 1990.

[15] Stickel, M. E.: A Prolog Technology Theorem Prover, in: New generation computing 2 (1984).

[16] Tuschik, H. P.; Wolter, H.: Mathematische Logik – kurzgefaßt. BI Wissenschaftsverlag, 1994.

[17] W. Wechler: Universal Algebra for Computer Scientists. Springer, 1992.