Fachbereich 6.2 - Informatik
Naturwissenschaftlich-Technische Fakultät I
Universität des Saarlandes

Interaktiver Beweisassistent für höherstufige Logik

Implementierung in SML

von
Christian Hümbert

Bachelorarbeit

in Informatik

Angefertigt am
Lehrstuhl Programmiersysteme
Informatik

unter der Leitung von
Professor Dr. Gert Smolka
betreut durch
Professor Dr. Gert Smolka

Juni 2005

Verfasser: Christian Hümbert

Erstgutachter: Prof. Dr. Gert Smolka

Zweitgutachter: Prof. Dr. Jörg H. Siekmann

Eidesstattliche Erklärung

Hiermit erkläre ich an Eides Statt, dass ich die vorliegende Arbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet habe.

Saarbrücken, den 29. Juni 2005

Christian Hümbert

Danksagung

Mein besonderer Dank gilt Herrn Prof. Dr. Gert Smolka, der es mir ermöglichte an diesem interessanten Thema zu arbeiten, für die ausgezeichnete Betreuung dieser Arbeit und die sehr hilfreichen Gespräche und Anregungen.

Inhaltsverzeichnis

Kapitel 1

Einleitung

Mathematische Beweise zu führen, ist eine nichttriviale Angelegenheit. Prinzipiell hätte man gerne Programme, die einem Arbeit abnehmen, indem sie selbständig Beweise führen.

Zunächst muss man zwischen drei Arten von Beweissystemen unterscheiden.
Es gibt zum Einen die sogenannten „Proof Checker“, die einen vorgegebenen Beweis Zeile für Zeile auf Korrektheit prüfen können. Diese Systeme sind wenig konstruktiv, wenngleich hilfreich.

Eine andere Klasse von Beweissystemen sind die „Automated Theorem Prover“(ATP). Diese Systeme versuchen für ein gegebenes Problem, selbständig und somit automatisch einen Beweis zu finden. Leider haben solche Systeme eine obere Schranke, was die Komplexität der zugrunde liegenden Logik angeht. Da Prädikatenlogik erster Stufe unentscheidbar ist, kann ein ATP System hier bereits scheitern. Aussagenlogische Probleme sind hingegen entscheidbar und können mit ATP's gelöst werden.

Um eine Systemunterstützung für Probleme, die zum Beispiel in Prädikatenlogik höherer Stufe formuliert sind, zu erhalten, geht man einen Kompromiss ein und benutzt sogenannte „Beweisassistenten“. Der Benutzer muss interaktiv in die Beweisführung des Systems eingreifen und die richtigen Entscheidungen treffen. Triviale Teilbeweise, zum Beispiel aussagenlogische Behauptungen, kann das System selbständig führen.

Ziel dieser Arbeit ist es, einen einfachen Beweisassistenten für höherstufige Logik zu entwickeln, der auf den in der Vorlesung ICL [1] eingeführten Techniken basiert und insbesondere in der Lehre eingesetzt werden kann.

Struktur der Arbeit

Kapitel 2: Wir wollen formal zeigen, was wir unter Prädikatenlogik höherer Stufe verstehen.

Kapitel 3: Dieses Kapitel soll veranschaulichen, was Deduktionskalküle sind und erklären, welche Rolle sie in Beweisassistenzsystemen spielen.

Kapitel 4: Nach den beiden vorbereitenden Kapiteln, wollen wir in Kapitel 4 unser System genauer analysieren.

Kapitel 5: In diesem Kapitel finden wir eine Fallstudie Induktionsbeweise über natürliche Zahlen.

Kapitel 6: Anschließend wollen wir unser System hier in den Kontext von bekannten Beweisassistenzsystemen einordnen.

Kapitel 7: Die Arbeit schließt mit einem Fazit und gibt einen Ausblick auf eventuelle Erweiterungen unseres Systems.

Anhang A: Ein Tutorial, mit dessen Hilfe der Benutzer den Umgang mit unserem Beweissystem erlernen kann.

Anhang B: Konstruktion eigener Deduktionsregeln über Formeln.

Kapitel 2

Höherstufige Logik

Im vorherigen Kapitel wurde erläutert, was wir allgemein unter einem mathematischen Assistenzsystem verstehen. Um zu verstehen, wie diese Systeme arbeiten, müssen wir zunächst klären, was wir unter höherstufiger (Prädikaten) Logik (im Folgenden HOL) verstehen. Wir werden hierfür den von Church entwickelten einfach getypten Lambda-Kalkül [1940] verwenden. Indem wir eine bestimmte Signatur und eine Menge an Axiomen vorgeben, erhalten wir schließlich HOL als eine Instanz des einfach getypten Lambda-Kalküls (im Folgenden λ-Kalkül).

Die in diesem Kapitel definierten Begriffe basieren auf Notationen, die in der Vorlesung ICL eingeführt wurden [1].

2.1 Syntax des λ-Kalkül

Die abstrakte Syntax wird formal durch die *Signatur* festgelegt. Die Signatur ist ein Tripel *(TC, VC, ty)*, wobei es sich bei

- TC um eine Menge von Typkonstanten,
- VC um eine Menge von Wertkonstanten,
- ty um eine Funktion, die Wertkonstanten Typen zuordnet,

handelt.
Die Menge Ty der Typen ist wie folgt definiert:

$$b \in TC$$
$$t \in Ty = b \mid t_1 \to t_2$$

Folgende Ausdrücke können mit den Konstanten einer Signatur gebildet werden:

t	$\in$	Ty	Typ
c	$\in$	VC	Wertkonstante
x	$\in$	Var $= \mathbb{N}$	Variable
M,N	$\in$	PT $= c \mid x \mid MN \mid \lambda x : t.M$	Präterm

Bei den oben definierten Ausdrücken handelt es sich zunächst um Präterme. Das sind Ausdrücke, die mit den Konstanten der Signatur gebildet werden können, wobei die Einschränkungen durch das Typsystem ignoriert werden.

Im Folgenden wollen wir jedoch ausschließlich wohlgetypte Präterme betrachten. Das sind Terme, die zusammen mit der Funktion $ty : VC \to Ty$ und einer Typumgebung $\Gamma : Var \to Ty$, die Variablen Typen zuordnet, gebildet werden können, ohne die Typrestrektion zu verletzen.

2.2 Semantik des λ-Kalkül

Um einen Term, der aus Konstanten und Variablen einer Signatur gebildet wurde, auswerten zu können, müssen wir Interpretationen für die Konstanten und Variablen angeben. Der Term beschreibt dann einen Wert, der in der durch den Typ des Terms beschriebenen Menge enthalten ist.

Unter einer *Interpretation* einer Signatur verstehen wir eine Funktion D, die jeder Wert- und Typkonstanten ein Objekt zuordnet, man spricht von Denotationspflicht.

Wir sind bis jetzt also in der Lage, allen Konstanten der Signatur Werte zuzuweisen. Nun benötigen wir noch Funktionen, die Werte für freie Variablen vorgeben. Wir nennen diese Funktionen *Zustände*.

Somit können wir nun Terme gemäß eines Zustandes auswerten.

2.3 Gleichungen

Eine *Gleichung* ist ein Paar (M,N) aus zwei Termen, die dengleichen Typ haben. Wir schreiben M=N, um zu sagen, dass (M,N) eine Gleichung ist. Eine Gleichung heißt *gültig* in einer Interpretation D (kurz: $D \models M = N$), wenn die Terme M und N in D für alle Zustände denselben Wert liefern.

Eine Gleichung M=N heißt *universell gültig* (kurz: $\models M = N$), wenn M=N in allen Interpretationen gültig ist.

Geben wir eine Basismenge A gültiger Gleichungen vor, so sagen wir, dass eine neue Gleichung E aus A semantisch folgt ($A \models E$), wenn E in jeder Interpretation gültig ist, in der alle Gleichungen aus A gültig sind.

Damit sind wir in der Lage, auf semantischer Ebene neue Gleichungen aus bereits bekannten abzuleiten.

2.4 HOL als Instanz des λ-Kalkül

Wir haben nun alle Vorbereitungen getroffen, um HOL formal beschreiben zu können.

$$HOL = (Sig_{HOL}, D_{HOL}, LAx)$$

wobei
$Sig_{HOL} = (TC_{HOL}, VC_{HOL}, ty_{HOL})$

$$\begin{array}{lcll} TC_{HOL} & = & \{B, X_1, X_2, \ldots\} & (\text{B, } X_i \text{ Basistypen}) \\ VC_{HOL} & = & \{\bot, \top, \vee, \wedge, \neg, \forall_\alpha, \exists_\alpha, \ldots\} & (\textit{logische} \text{ Konstanten}) \\ ty_{HOL} & = & \{(\bot, B), (\top, B), (\neg, B \to B), & \\ & & (\vee, B \to B \to B), (\wedge, B \to B \to B), & \\ & & (\forall_\alpha, (\alpha \to B) \to B), (\exists_\alpha, (\alpha \to B) \to B), \ldots\} & \\ & & \text{für alle } \alpha \in Ty_{HOL} & \end{array}$$

D_{HOL} soll eine Standardinterpretation sein, die die boolschen Operatoren gemäß der zweiwertigen Boolschen Algebra und die Quantoren wie folgt interpretiert:

$$\begin{array}{lcl} D_{HOL}(\bot) & = & 0 \\ D_{HOL}(\top) & = & 1 \\ D_{HOL}(\vee) & = & \lambda x, y : B.max(x, y) \\ D_{HOL}(\wedge) & = & \lambda x, y : B.min(x, y) \\ D_{HOL}(\neg) & = & \lambda x : B.1 - x \\ D_{HOL}(\forall_\alpha) & = & \lambda f : \alpha \to B.f = \lambda x : \alpha.1 \\ D_{HOL}(\exists_\alpha) & = & \lambda f : \alpha \to B.f \neq \lambda x : \alpha.0 \end{array}$$

Zusätzlich fordern wir die Gültigkeit aller logischen Axiome in LAx.
$LAx = QAx \cup BAx$. Seien a,b,c,f,u und x Variablen.

$$\begin{array}{ll} \text{QAx}_\alpha = \{ \forall_\alpha f = \forall_\alpha f \wedge fx, & \text{BAx} = \{ a \wedge b = b \wedge a, \\ \quad \forall_\alpha x.(fx \vee u) = \forall_\alpha f \vee u\} & \quad a \wedge (b \vee c) = (a \wedge b) \vee (a \wedge c), \\ & \quad a \wedge \neg a = \bot, \\ \text{QAx} = \bigcup_{\alpha \in Ty} QAx_\alpha & \quad a \wedge \top = a, \ \ldots\} \end{array}$$

Definitionen
Um Schreibarbeit zu sparen, lassen wir die Typangaben für die Quantoren weg. Außerdem führen wir die folgenden abkürzenden Definitionen ein:

- $\forall x.M :=^{def} \forall(\lambda x.M)$
- $\exists f :=^{def} \neg\forall x.\neg fx$
- $x \Rightarrow y :=^{def} \neg x \vee y$

Bemerkungen
In der Menge der Boolschen Axiome BAx sind zusätzlich zu den aufgeführten Gleichungen, die dualen Gleichungen enthalten, die man durch Vertauschen von $\wedge$ und $\vee$ bzw. $\top$ und $\bot$ erhält.

Der Existenzquantor taucht in der Menge QAx nicht auf, er kann, wie oben gezeigt, definiert werden.

Mit den oben definierten Konstanten und den Techniken, die uns der λ-Kalkül zur Verfügung stellt, sind wir nun in der Lage, neue Terme und somit neue Gleichungen in HOL anzugeben und Aussagen über deren Gültigkeit zu treffen.

Ein Term, der den Typ B hat, wird im Folgenden als *Formel* bezeichnet. Desweiteren sagen wir, dass eine Formel F in einer Interpretation gültig ist, wenn $D \models F = \top$. Entsprechend ist eine Formel F universell gültig, wenn gilt $\models F = \top$.

Im Rahmen von mathematischen Assistenzsystemen möchte man die Gültigkeit von Formeln zeigen. Dies wäre bei komplexeren Formeln auf semantischer Ebene umständlich. Deshalb entwickelt man sogenannte Deduktionssysteme, die auf rein syntaktischer Ebene und mit den üblichen Beweistechniken arbeiten. In Kapitel 3 werden wir zwei entsprechende Deduktionssysteme vorstellen.

Kapitel 3

Deduktionssysteme

3.1 Einführung

In Kaptiel 2.4 haben wir die Syntax und Semantik von HOL angegeben. Mit Hilfe der dort vorgestellten logischen Axiome wären wir in der Lage, neue Gleichungen aus alten auf semantischer Ebene abzuleiten. Ein *Deduktionskalkül* stellt hingegen ein syntaktisches Regelsystem zur Verfügung, bestehend aus Schlussregeln, die es erlauben, neue gültige Aussagen aus bereits bekannten gültigen Aussagen abzuleiten, ohne deren Semantik zu verstehen.
Eine entsprechende Schlussregel hat folgende Form:

$$\frac{A_1 \quad ... \quad A_n}{A}$$

Sie erlaubt es, die Aussage A aus den Aussagen $A_1 ... A_n$ in einem Umformungsschritt abzuleiten. Wird eine Aussage B in mehreren Schritten aus den Aussagen $B_1 ... B_n$ durch Anwendung von Schlussregeln abgeleitet, so schreibt man $B_1 ... B_n \vdash B$.

Sei Γ eine Menge von universell gültigen Aussagen.
Von einem deduktiven Kalkül verlangt man, dass nur solche Ableitungen möglich sind, die im Rahmen der Semantik erlaubt sind. Man spricht dann von der *Korrektheit* des Kalküls.

$$\Gamma \vdash A \qquad \Longrightarrow \qquad \Gamma \models A$$

Umgekehrt nennt man einen Kalkül *vollständig*, wenn jede universell gültige Aussage durch endlich viele Anwendungen der Schlussregeln aus Γ hergeleitet werden kann.

$$\Gamma \models A \qquad \Longrightarrow \qquad \Gamma \vdash A$$

3.2 Gleichheitsregeln

Wir geben nun für den λ-Kalkül ein Deduktionssystem an, das es erlaubt, neue Gleichungen aus einer vorgegebenen Gleichungsmenge abzuleiten [1].

$$\frac{}{M = M}\ \text{Reflexivität} \qquad \frac{N = M}{M = N}\ \text{Symmetrie} \qquad \frac{M = N \quad N = M'}{M = M'}\ \text{Transitivität}$$

$$\frac{N = N' \quad M/N \approx M'/N'}{M = M'}\ \text{Ersetzung}$$

$$\frac{M = M' \quad \text{x und N haben denselben Typ}}{M[x := N] = M'[x := N]}\ \text{Einsetzung}$$

$$\frac{}{(\lambda x.M)N = M[x := N]}\ \text{Beta} \qquad \frac{x \notin FV(M)}{(\lambda x.Mx) = M}\ \text{Eta}$$

$M/N \approx M'/N'$ bedeutet, dass man M' aus M erhält, indem man ein Auftreten von N in M durch N' ersetzt.
$x \notin FV(M)$ bedeutet, dass die Variable x nicht als freie Variable innerhalb des Terms M vorkommen darf. Freie Variablen sind solche Variablen, die nicht durch eine Lambdaabstraktion gebunden sind.

Die Gleichheitsregeln stellen ein korrektes Deduktionssystem dar. Der Korrektheitsbeweis wurde in [1] geführt. Die Vollständigkeit des Systems kann für den Fall einer leeren Ausgangsmenge an Gleichungen ebenfalls gezeigt werden [8]. Es gilt: $\varnothing \models E \Longrightarrow \varnothing \vdash E$

3.3 Sequenten-Kalkül

Mit Hilfe der Gleichheitsregeln können wir nun Gleichheitsbeweise führen. Wollen wir die Gültigkeit einer Formel F zeigen, so können wir dies über die Gleichheit F = $\top$ mit unseren Regeln zeigen. Meist jedoch sind die Formeln so komplex, dass die Beweise auf der Ebene der Gleichheitsregeln groß und unübersichtlich werden.

Wir wollen deshalb in diesem Abschnitt ein Formeldeduktionssystem angeben, bei dem jede Regel aus den Gleichheitsregeln und den logischen Axiomen (LAx)

abgeleitet ist.
In diesem Ableitungsprozess bedienen wir uns des sogenannten *Bootstrapping*. Wir leiten zunächst wenige Hilfsregeln ab, mit deren Hilfe wir unter anderem die Einführungsregeln herleiten. Beim Ableiten der Eliminierungsregeln, können wir dann bereits auf die Einführungsregeln zurückgreifen. Das heißt, unser Deduktionssystem wird mit jeder abgeleiteten Regel mächtiger.

Seien $A_1, ..., A_n, B, \triangle, \Phi$ Formeln und M,N Terme desselben Typs.
Eine Schlussregel $\frac{A_1 ... A_n}{B}$ heißt *abgeleitet*, wenn gilt

$$(\forall i : LAx \vdash A_i = \top) \Rightarrow (LAx \vdash B = \top)$$

Hilfsregeln

$$\frac{(M = N) \in LAx}{LAx \vdash (M = N)}\ (LAx) \qquad \frac{LAx \vdash M = \top \quad (M = N) \in LAx}{LAx \vdash N = \top}\ (LAx(M = N))$$

Die Regel (LAx) ist offensichtlich. Die Regel (LAx(M=N)) wollen wir mit unseren Gleichheitsregeln beweisen. Wir sparen uns im Folgenden das Präfix $LAx \vdash$, der Übersichtlichkeit halber und sagen, LAx sei unsere vorgegebene Menge von Gleichungen.

$$\dfrac{\dfrac{\dfrac{}{M = N}\ (\text{LAx})}{N = M}\ (\text{Sym}) \qquad M = \top}{N = \top}\ (\text{Trans})$$

Beim Beweis einer konkreten Regel, wie zum Beispiel $LAx(A \vee B = B \vee A)$ benötigen wir zusätzlich die Einsetzungsregel.

$$\frac{\triangle \wedge \neg A \Rightarrow \Phi}{\triangle \Rightarrow A \vee \Phi}\ (move2L) \qquad \frac{\triangle \Rightarrow \Phi \vee \neg A}{\triangle \wedge A \Rightarrow \Phi}\ (move2R)$$

Wir wollen die Regel (move2L) ableiten. Die Ableitung der Regel (move2R) erfolgt analog.

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{x = \neg\neg x}\ (\text{LAx(DNeg)})}{A = \neg\neg A}\ (\text{Ein})}{(\neg\triangle \vee A) \vee \Phi = (\neg\triangle \vee \neg\neg A) \vee \Phi}\ (\text{Ers}) \qquad \dfrac{\dfrac{\dfrac{}{\neg x \vee \neg y = \neg(x \wedge y)}\ (\text{LAx})}{\neg\triangle \vee \neg\neg A = \neg(\triangle \wedge \neg A)}\ (\text{2xEin})}{(\neg\triangle \vee \neg\neg A) \vee \Phi = \neg(\triangle \wedge \neg A) \vee \Phi}\ (\text{Ers}) \qquad \dfrac{(\triangle \wedge \neg A) \Rightarrow \Phi = \top}{\neg(\triangle \wedge \neg A) \vee \Phi = \top}\ (\text{Def} \Rightarrow)}{(\neg\triangle \vee \neg\neg A) \vee \Phi = \top}\ (\text{Trans})}{(\neg\triangle \vee A) \vee \Phi = \top}\ (\text{Trans})}{\neg\triangle \vee A \vee \Phi = \top}\ (\text{LAx(Assoz)})}{\triangle \Rightarrow A \vee \Phi = \top}\ (\text{LAx(Def} \Rightarrow))$$

$$\frac{A \wedge \triangle \Rightarrow \Phi}{\triangle \wedge A \Rightarrow \Phi}\ (cShiftL) \qquad \frac{\triangle \Rightarrow \Phi \vee A}{\triangle \Rightarrow A \vee \Phi}\ (cShiftR)$$

Ableitung (cShiftR) :
(cShiftL) lässt sich analog ableiten

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{x \vee y = y \vee x}\ (\text{LAx})}{A \vee \Phi = \Phi \vee A}\ (\text{2xEin})}{\neg\triangle \vee A \vee \Phi = \neg\triangle \vee \Phi \vee A}\ (\text{Ers}) \qquad \dfrac{\triangle \Rightarrow \Phi \vee A = \top}{\neg\triangle \vee \Phi \vee A = \top}\ (\text{LAx(Def }\Rightarrow))}{\dfrac{\neg\triangle \vee A \vee \Phi}{\triangle \Rightarrow A \vee \Phi}\ (\text{LAx(Def }\Rightarrow))}\ (\text{Trans})$$

Einführungsregeln (Introduction-Rules)

(1)
$$\frac{\triangle \wedge A \Rightarrow \bot}{\triangle \Rightarrow \neg A}\ (\neg I)$$

Ableitung:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\triangle \wedge A \Rightarrow \bot = \top}{\triangle \Rightarrow \neg A \vee \bot = \top}\ (\text{move2L})}{\neg\triangle \vee \neg A \vee \bot = \top}\ (\text{LAx(Def }\Rightarrow))}{(\neg\triangle \vee \neg A) \vee \bot = \top}\ (\text{LAx(Asso)})}{\neg\triangle \vee \neg A = \top}\ (\text{LAx(Iden)})}{\triangle \Rightarrow \neg A = \top}\ (\text{LAx(Def }\Rightarrow))$$

(2)
$$\frac{\triangle \Rightarrow A}{\triangle \Rightarrow A \vee B}\ (\vee I_L)$$

Ableitung:

$$\dfrac{\dfrac{\dfrac{\dfrac{\triangle \Rightarrow A = \top}{\neg\triangle \vee A = \top}\ (\text{LAx(Def }\Rightarrow))}{(\neg\triangle \vee A) \vee B = \top \vee B}\ (\text{Ers}) \qquad \dfrac{\dfrac{\dfrac{}{x \vee \top = \top}\ (\text{LAx})}{B \vee \top = \top}\ (\text{Ein})}{\top \vee B = \top}\ (\text{LAx(Komm)})}{(\neg\triangle \vee A) \vee B = \top}\ (\text{Trans})}{\dfrac{\neg\triangle \vee A \vee B = \top}{\triangle \Rightarrow A \vee B = \top}\ (\text{LAx(Asso)})}\ (\text{LAx(Def }\Rightarrow))$$

(3)
$$\frac{\triangle \Rightarrow B}{\triangle \Rightarrow A \vee B}\ (\vee I_R)$$

Ableitung:

$$\dfrac{\dfrac{\triangle \Rightarrow B = \top}{\triangle \Rightarrow B \vee A = \top}\ (\vee I_L)}{\triangle \Rightarrow A \vee B = \top}\ (\text{cShiftR})$$

(4)
$$\dfrac{\triangle \Rightarrow A \qquad \triangle \Rightarrow B}{\triangle \Rightarrow A \wedge B}\ (\wedge I)$$

Ableitung:

$$\dfrac{\dfrac{\dfrac{\triangle \Rightarrow A = \top}{(\neg\triangle \vee A) = \top}\ (\text{LAx (Def }\Rightarrow))}{(\neg\triangle \vee A) \wedge (\neg\triangle \vee B) = \top \wedge (\neg\triangle \vee B)}\ (\text{Ers}) \qquad \dfrac{\dfrac{\dfrac{\triangle \Rightarrow B = \top}{(\neg\triangle \vee B) = \top}\ (\text{LAx (Def }\Rightarrow))}{\top \wedge (\neg\triangle \vee B) = \top \wedge \top}\ (\text{Ers}) \qquad \dfrac{\dfrac{}{x \wedge \top = x}\ (\text{LAx})}{\top \wedge \top = \top}\ (\text{Ein})}{\top \wedge (\neg\triangle \vee B) = \top}\ (\text{Trans})}{(\neg\triangle \vee A) \wedge (\neg\triangle \vee B) = \top}\ (\text{Trans})$$
$$\dfrac{\dfrac{(\neg\triangle \vee A) \wedge (\neg\triangle \vee B) = \top}{\neg\triangle \vee A \wedge B = \top}\ (\text{LAx Distrib})}{\triangle \Rightarrow A \wedge B = \top}\ (\text{LAx (Def }\Rightarrow))$$

(5)
$$\dfrac{\triangle \wedge A \Rightarrow B}{\triangle \Rightarrow A \Rightarrow B}\ (\Rightarrow I)$$

Ableitung:

$$\dfrac{\dfrac{\dfrac{\dfrac{}{x \Rightarrow y = \neg x \vee y}\ (\text{LAx})}{A \Rightarrow B = \neg A \vee B}\ (\text{2xEin})}{\neg\triangle \vee (A \Rightarrow B) = \neg\triangle \vee \neg A \vee B}\ (\text{Ers}) \qquad \dfrac{\dfrac{\triangle \wedge A \Rightarrow B = \top}{\triangle \Rightarrow \neg A \vee B = \top}\ (\text{move2L})}{\neg\triangle \vee \neg A \vee B = \top}\ (\text{LAx (Def }\Rightarrow))}{\neg\triangle \vee (A \Rightarrow B) = \top}\ (\text{Trans})$$
$$\dfrac{\neg\triangle \vee (A \Rightarrow B) = \top}{\triangle \Rightarrow A \Rightarrow B = \top}\ (\text{LAx (Def }\Rightarrow))$$

Eliminierungsregeln (Elimination-Rules)

(1)
$$\dfrac{\triangle \wedge A \Rightarrow \Phi}{\triangle \wedge (A \wedge B) \Rightarrow \Phi}\ (\wedge E_L)$$

Ableitung:

$$\dfrac{\dfrac{\dfrac{\dfrac{}{\neg(x \wedge y) = \neg x \vee y}\ (\text{LAx})}{\neg(A \wedge B) = \neg A \vee \neg B}\ (\text{2xEin})}{\neg(\triangle \wedge \neg\Phi) \vee \neg(A \wedge B) = \neg(\triangle \wedge \neg\Phi) \vee \neg A \vee \neg B}\ (\text{Ers}) \qquad \dfrac{\dfrac{\dfrac{\triangle \wedge A \Rightarrow \Phi = \top}{\triangle \Rightarrow \neg A \vee \Phi = \top}\ (\text{move2L })}{\triangle \wedge \neg\Phi \Rightarrow \neg A = \top}\ (\text{move2R})}{\triangle \wedge \neg\Phi \Rightarrow \neg A \vee \neg B = \top}\ (\vee I_L)}{\neg(\triangle \wedge \neg\Phi) \vee \neg A \vee \neg B = \top}\ (\text{LAx(Def }\Rightarrow))$$
$$\dfrac{\dfrac{\dfrac{\dfrac{\neg(\triangle \wedge \neg\Phi) \vee \neg(A \wedge B) = \top}{\triangle \wedge \neg\Phi \Rightarrow \neg(A \wedge B) = \top}\ (\text{LAx (Def }\Rightarrow))}{\triangle \Rightarrow \Phi \vee \neg(A \wedge B) = \top}\ (\text{move2L})}{\triangle \wedge (A \wedge B) \Rightarrow \Phi = \top}\ (\text{move2R})}{}\ (\text{Trans})$$

(2)
$$\frac{\triangle \wedge B \Rightarrow \Phi}{\triangle \wedge (A \wedge B) \Rightarrow \Phi}\ (\wedge E_R)$$

Ableitung:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\triangle \wedge B \Rightarrow \Phi = \top}{\triangle \wedge (B \wedge A) \Rightarrow \Phi = \top}\ (\wedge E_L)}{(B \wedge A) \wedge \triangle \Rightarrow \Phi = \top}\ (\text{cShiftL})}{(B \wedge A) \Rightarrow \neg\triangle \vee \Phi = \top}\ (\text{move2L})}{(B \wedge A) \Rightarrow \Phi \vee \neg\triangle = \top}\ (\text{cShiftR})}{(A \wedge B) \Rightarrow \Phi \vee \neg\triangle = \top}\ (\text{cShiftL})}{(A \wedge B) \wedge \triangle \Rightarrow \Phi = \top}\ (\text{move2R})}{\triangle \wedge (A \wedge B) \Rightarrow \Phi = \top}\ (\text{cShiftL})$$

(3)
$$\frac{\triangle \wedge A \Rightarrow \Phi \qquad \triangle \wedge B \Rightarrow \Phi}{\triangle \wedge (A \vee B) \Rightarrow \Phi}\ (\vee E)$$

Ableitung:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\neg(x \vee y) = \neg x \wedge \neg y}\ (\text{LAx})}{\neg(A \vee B) = \neg A \wedge \neg B}\ (\text{2xEin})}{\neg(\triangle \wedge \neg\Phi) \vee \neg(A \vee B) = \neg(\triangle \wedge \neg\Phi) \vee \neg A \wedge \neg B}\ (\text{Ers}) \qquad \dfrac{\dfrac{\dfrac{\triangle \wedge A \Rightarrow \Phi = \top}{\triangle \Rightarrow \neg A \vee \Phi = \top}\ (\text{move2L})}{\triangle \wedge \neg\Phi \Rightarrow \neg A = \top}\ (\text{move2R}) \qquad \dfrac{\triangle \wedge B \Rightarrow \Phi = \top}{\triangle \Rightarrow \neg B \vee \Phi = \top}}{\triangle \wedge \neg\Phi \Rightarrow \neg B = \top}}{\triangle \wedge \neg\Phi \Rightarrow \neg A \wedge \neg B = \top}\ (\wedge I)}{\neg(\triangle \wedge \neg\Phi) \vee \neg A \wedge \neg B = \top}\ (\text{LAx(Def }\Rightarrow))}{\neg(\triangle \wedge \neg\Phi) \vee \neg(A \vee B) = \top}\ (\text{Trans})}{\triangle \wedge \neg\Phi \Rightarrow \neg(A \vee B) = \top}\ (\text{LAx(Def }\Rightarrow))}{\triangle \Rightarrow \Phi \vee \neg(A \vee B) = \top}\ (\text{move2L})}{\triangle \wedge (A \vee B) \Rightarrow \Phi = \top}\ (\text{move2R})$$

(4)
$$\frac{\triangle \Rightarrow A \vee \Phi \qquad \triangle \wedge B \Rightarrow \Phi}{\triangle \wedge (A \Rightarrow B) \Rightarrow \Phi}\ (\Rightarrow E)$$

Ableitung:

$$\dfrac{\dfrac{\dfrac{\dfrac{}{x \Rightarrow y = \neg x \vee y}\ (\text{LAx})}{A \Rightarrow B = \neg A \vee B}\ (\text{2xEin})}{\triangle \wedge (A \Rightarrow B) \Rightarrow \Phi = \triangle \wedge (\neg A \vee B) \Rightarrow \Phi}\ (\text{Ers}) \qquad \dfrac{\dfrac{\dfrac{\triangle \Rightarrow A \vee \Phi}{\triangle \Rightarrow \Phi \vee A}\ (\text{cShiftR})}{\triangle \wedge \neg A \Rightarrow \Phi = \top}\ (\text{move2R}) \qquad \triangle \wedge B \Rightarrow \Phi = \top}{\triangle \wedge (\neg A \vee B) \Rightarrow \Phi = \top}\ (\vee E)}{\triangle \wedge (A \Rightarrow B) \Rightarrow \Phi = \top}\ (\text{Trans})$$

Quantorregeln

(1)
$$\frac{\triangle \Rightarrow A \qquad \text{x nicht frei in } \triangle}{\triangle \Rightarrow \forall x.A}\ (\forall I)$$

Ableitung:

$$\dfrac{\dfrac{\dfrac{\dfrac{\triangle \Rightarrow A = \top}{(\neg\triangle \vee A) = \top}\ (\text{LAx(Def }\Rightarrow))}{(A \vee \neg\triangle) = \top}\ (\text{LAx Komm})}{\forall x.(A \vee \neg\triangle) = \forall x.\top}\ (\text{Ers}) \qquad \dfrac{\dfrac{\overline{\forall x.u = u}\ (\text{LAx})}{\forall x.\top = \top}\ (\text{Ein})}{\dfrac{\dfrac{\dfrac{\forall x.(A \vee \neg\triangle) = \top}{(\forall x.A) \vee \neg\triangle = \top}\ (\text{LAx } \forall\vee)}{\neg\triangle \vee \forall x.A = \top}\ (\text{LAx Komm})}{\triangle \Rightarrow \forall x.A = \top}\ (\text{LAx(Def }\Rightarrow))}\ (\text{Trans})$$

(2)
$$\frac{\triangle \wedge A[x := M] \Rightarrow \Phi}{\triangle \wedge (\forall x.A) \Rightarrow \Phi}\ (\forall E)$$

Ableitung:

$$\dfrac{\dfrac{\dfrac{\overline{\forall f = \forall f \wedge fx}\ (\text{LAx})}{(\forall x.A) = (\forall x.A) \wedge A[x := M]}\ (\text{2xEin})}{\triangle \wedge (\forall x.A) \Rightarrow \Phi = \triangle \wedge ((\forall x.A) \wedge A[x := M]) \Rightarrow \Phi}\ (\text{Ers}) \qquad \dfrac{\triangle \wedge A[x := M] \Rightarrow \Phi = \top}{\triangle \wedge ((\forall x.A) \wedge A[x := M]) \Rightarrow \Phi = \top}\ (\wedge E_R)}{\triangle \wedge (\forall x.A) \Rightarrow \Phi = \top}\ (\text{Trans})$$

(3)
$$\frac{\triangle \Rightarrow A[x := M]}{\triangle \Rightarrow \exists x.A}\ (\exists I)$$

Ableitung:

$$\dfrac{\dfrac{\dfrac{\overline{\neg(\exists f) = \forall x.\neg fx}\ (\text{LAx})}{\neg(\exists x.A) = \forall x.\neg A}\ (\text{Ein})}{\triangle \wedge \neg(\exists x.A) \Rightarrow \bot = \triangle \wedge (\forall x.\neg A) \Rightarrow \bot}\ (\text{Ers}) \qquad \dfrac{\dfrac{\triangle \Rightarrow A[x := M] = \top}{\triangle \wedge (\neg A)[x := M] \Rightarrow \bot = \top}\ (\text{move2R})}{\triangle \wedge (\forall x.\neg A) \Rightarrow \bot = \top}\ (\forall E)}{\dfrac{\triangle \wedge \neg(\exists x.A) \Rightarrow \bot = \top}{\triangle \Rightarrow \exists x.A = \top}\ (\text{move2L})}\ (\text{Trans})$$

(4)
$$\frac{\triangle \wedge A \Rightarrow \Phi \qquad \text{x nicht frei in } \triangle, \Phi}{\triangle \wedge (\exists x.A) \Rightarrow \Phi}\ (\exists E)$$

Ableitung:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\neg(\exists f) = \forall x.\neg fx}\,(\text{LAx})}{\neg(\exists x.A) = \forall x.\neg A}\,(\text{Ein})}{\triangle \wedge \neg\Phi \Rightarrow \neg(\exists x.A) = \triangle \wedge \neg\Phi \Rightarrow \forall x.\neg A}\,(\text{Ers}) \quad \dfrac{\dfrac{\dfrac{\triangle \wedge A \Rightarrow \Phi = \top}{\triangle \Rightarrow \neg A \vee \Phi = \top}\,(\text{move2L})}{\triangle \wedge \neg\Phi \Rightarrow \neg A = \top}\,(\text{move2R})}{\triangle \wedge \neg\Phi \Rightarrow \forall x.\neg A = \top}\,(\forall I)}{\triangle \wedge \neg\Phi \Rightarrow \neg(\exists x.A) = \top}\,(\text{Trans})}{\triangle \Rightarrow \Phi \vee \neg(\exists x.A) = \top}\,(\text{move2L})}{\triangle \wedge (\exists x.A) \Rightarrow \Phi = \top}\,(\text{move2R})$$

Zusätzliche Regeln

(1) $$\dfrac{\triangle \Rightarrow A \quad \triangle \wedge A \Rightarrow \Phi}{\triangle \Rightarrow \Phi}\,(Cut)$$

Ableitung:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{x \wedge \top = x}\,(\text{LAx})}{\triangle \wedge \top = \triangle}\,(\text{Ein})}{\triangle = \triangle \wedge \top}\,(\text{Sym})}{\triangle \Rightarrow \Phi = \triangle \wedge \top \Rightarrow \Phi}\,(\text{Ers}) \quad \dfrac{\dfrac{\dfrac{\dfrac{}{x \vee \neg x = \top}\,(\text{LAx})}{A \vee \neg A = \top}\,(\text{Ein})}{\top = A \vee \neg A}\,(\text{Sym})}{\triangle \wedge \top \Rightarrow \Phi = \triangle \wedge (A \vee \neg A) \Rightarrow \Phi}\,(\text{Ers}) \quad \dfrac{\triangle \wedge A \Rightarrow \Phi \quad \dfrac{\dfrac{\triangle \Rightarrow A}{\triangle \Rightarrow \Phi \vee A}\,(\vee I_R)}{\triangle \wedge \neg A \Rightarrow \Phi}\,(\text{m2R})}{\triangle \wedge (A \vee \neg A) \Rightarrow \Phi = \top}\,(\vee E)}{\triangle \wedge \top \Rightarrow \Phi = \top}\,(\text{Trans})}{\triangle \Rightarrow \Phi = \top}\,(\text{Trans})$$

(2) $$\dfrac{\varnothing \vdash A = B \quad A}{B}\,(Lam)$$

(3) $$\dfrac{BAx \vdash A \Rightarrow B = \top}{A \Rightarrow B}\,(Bool)$$

Es handelt sich bei den oben aufgeführten Regeln um den Sequenten-Kalkül, eine Variante des Kalkül des natürlichen Schließens (ND-Kalkül), ein deduktiver Kalkül, der von dem deutschen Logiker Gerhard Gentzen in seiner Dissertation entwickelt wurde. Gentzen hatte das Hauptanliegen, die in der Mathematik gebräuchlichen Schlussweisen in seinem Kalkül nachzubilden.

„Die Formalisierung des logischen Schließens, wie sie insbesondere durch Frege, Russell und Hilbert entwickelt worden ist, entfernt sich ziemlich weit von der Art des Schliessens, wie sie in Wirklichkeit bei mathematischen Beweisen geübt wird. Dafür werden beträchtliche formale Vorteile erzielt. Ich wollte nun zunächst einmal einen Formalismus aufstellen, der dem wirklichen Schließen möglichst nahe kommt. So ergab sich ein Kalkül des natürlichen Schließens. “ (Gentzen: Untersuchungen über das logische Schließen [3])

Alle in diesem Abschnitt abgeleiteten Einführungs- und Eliminierungsregeln, sowie die zusätzlichen Regeln und einige der Hilfsregeln sind in unserem Beweisassistenten implementiert. Dabei werden die Regeln meist rückwärts angewandt. Wir führen die Formel zurück auf ihre atomaren Bestandteile, die axiomatisch oder trivial gelten. Man spricht von der *Backward-Proof-Technik.*

Bemerkungen
Wir vereinbaren, dass wir nur *implikative* Formeln ableiten können, also Formeln der Form $A \Rightarrow B$. Ist eine Formel nicht in implikativer Form dargestellt, so schreiben wir sie in eine solche um.
Beispiel: $\forall x.x \vee \neg x \quad \rightsquigarrow \quad \top \Rightarrow \forall x.x \vee \neg x$

Die Regel (Lam) wurde mit in das Regelsystem aufgenommen, um Formeln ebenfalls β/η-reduzieren zu können. Da die Gültigkeit von aussagenlogischen Formeln entscheidbar ist, liefert uns die Regel (Bool) einen automatischen Gültigkeitstest.

Die Redundanz der Regel (Cut) wurde in Gentzens Hauptsatz bewiesen [3].
Allerdings scheint gerade die Cut-Regel interessant für interaktive Beweissysteme, da sie erlaubt, an einem beliebigen Punkt innerhalb des Beweises, ein Lemma einzufügen, dieses zu beweisen und anschließend zu benutzen.

Kapitel 4

Beweis Assistent NED

4.1 Logische Struktur des Systems

In diesem Abschnitt wollen wir die logische Struktur unseres Beweisassistenten NED (Natural and Equational Deduction) analysieren.

Abbildung 4.1 veranschaulicht den Aufbau des Systems. Die Grundlage unseres Beweissystems bildet der einfach getypte Lambda-Kalkül. Ebenfalls fest integriert ist die Signatur von HOL. Damit sind wir bereits in der Lage, HOL-Ausdrücke zu bilden.

Desweiteren stehen uns zwei Beweismodi zur Verfügung. Möchte der Benutzer Gleichheitsbeweise führen, so ist dies im **Rewriting-Modus** mit Hilfe der Gleichheitsregeln aus Kapitel 3 möglich. Beweise im Sequenten-Kalkül lassen sich im **ND-Modus** führen. Das System erkennt automatisch bei der Eingabe des Theorems und den Kommandos zur Lösung desselben, welcher Modus aktiviert werden soll. Im Gegensatz zum Gleichheitskalkül im Rewriting-Modus erweitert der Sequenten-Kalkül im ND-Modus implizit die Semantik unseres Systems, da die Regeln auf LAx basieren.

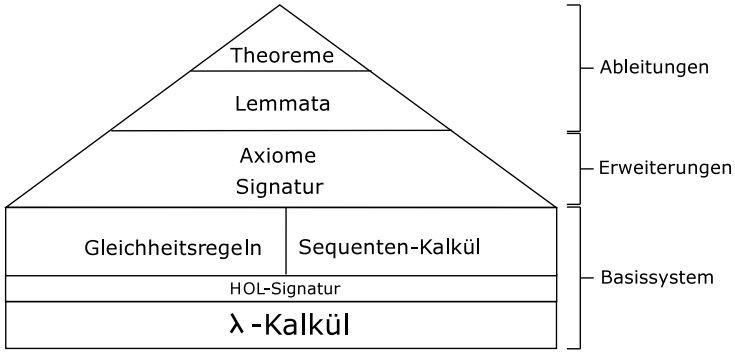

Abbildung 4.1: Systemstruktur NED

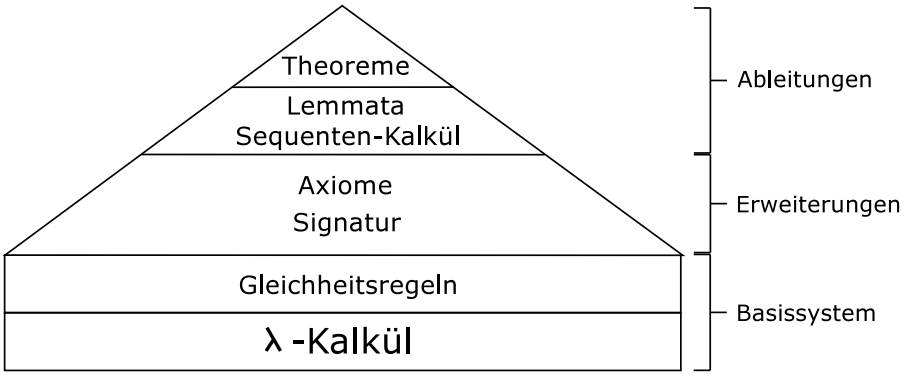

Abbildung 4.2: Alternative Systemstruktur

Benötigt der Benutzer darüberhinaus weitere Typen, Konstanten oder Variablen, so kann er diese deklarieren und damit die Signatur erweitern. Indem der Benutzer eigene Axiome hinzufügt, erweitert er damit auch die Semantik des Systems.

Ein alternativer Ansatzpunkt ist in Abbildung 4.2 dargestellt. Hier geben wir lediglich den Lambda-Kalkül und die Gleichheitsregeln vor, verzichten dabei ganz auf ein fest integriertes HOL-System. HOL könnte dann als Modul implementiert sein und bei Bedarf nachgeladen werden. Möchte der Benutzer die Gültigkeit von Formeln zeigen, so bietet es sich an, ein Formeldeduktionssystem (zum Beipiel den Sequenten-Kalkül) in Form eines Moduls, das mit Hilfe der Gleichheitsregeln aus den logischen Axiomen abgeleitet werden kann, zur Verfügung zu stellen.

Dieses Modell würde dem Benutzer einen maximalen Grad an Flexibilität zur Verfügung stellen, wäre aber aufwendiger in der Implementierung.

Zusammenfassend erkennt man, dass man mit dem Modell, das wir für NED gewählt haben, nur ein kleines Maß an Flexibilität verliert, lediglich die Signatur von HOL ist fest in das System integriert. Der Benutzer entscheidet selbst, ob er mit dem Sequenten-Kalkül arbeiten möchte oder ein eigenes Deduktionssystem in Form von Lemmata spezifiziert.

In Anhang B zeigen wir an zwei Beispielen, wie man mit NED Deduktionsregeln über Formeln konstruiert, diese Regeln im Gleichheitskalkül beweist und sie innerhalb eines Beweises geschickt anwendet.

4.1.1 Rewriting-Modus

In Abbildung 4.3 sind alle im Rewriting-Modus verfügbaren Regeln mit den entsprechenden NED-Kommandos angegeben. Da wir Beweise nach der *Backward-Proof-Technik* führen, entspricht der Eintrag in der Spalte $Goal^T$ gerade der Konklusion und der Eintrag in der Spalte $Goal^{T+1}$ der Prämisse der entsprechenden Regel.

Neben der einfachen Beta- und Etareduktion gibt es außerdem die Möglichkeit, die Beta- und Etanormalform eines Terms berechnen zu lassen. Dies geschieht über das Kommando **lam**.

4.1.2 ND-Modus

In Abbildung 4.4 sind alle im ND-Modus verfügbaren Regeln mit den entsprechenden NED-Kommandos angegeben. Diese Tabelle wird in Abschnitt 4.2 noch erweitert.

Mit dem Kommando **simplify** lassen sich zusätzlich aussagenlogische Probleme automatisiert vereinfachen.

Regel	**Goal**^{T}	**Kommando**	**Goal**$^{T+1}$
Reflexivität	M = M	**ref**	Ø (PROOF DONE)
Symmetrie	M = N	**sym**	N = M
Transitivität	M = M'	**trans** N	M = N (1/2) N = M' (2/2)
Einsetzungsregel	M[x] = M'[x]	**subst** x **=** N	M[x:=N] = M'[x:=N]
Ersetzungsregel	*axiom A:= (N = N') (Vor)*		
(Rewriting)	M[N] = M'	**rew** (Pos) **with** A **l**	M[N'] = M'
mit Axiom als Vor.	M[N'] = M'	**rew** (Pos) **with** A **r**	M[N] = M'
mit Lemma als Vor.	*lemma B:= (N = N') (Vor)*		
	M[N] = M'	**rew** (Pos) **with** B **l**	M[N'] = M' (1/2) N = N' (2/2)
ohne Voraussetzung	M[N] = M'	**rew** (Pos) **with** (N=N')	M[N'] = M' (1/2) N = N' (2/2)
Beta	(λ x.M[x])N	**rew** (Pos) **with beta**	M[x:=N]
Eta	(λ x.M x)	**rew** (Pos) **with eta**	M

Abbildung 4.3: Gleichheitsregeln in NED

Regel	**Goal**^{T}	**Kommando**	**Goal**$^{T+1}$
Einführungsregeln			
($\neg$I)	$\triangle \Longrightarrow \neg A$	**intro**	$A \wedge \triangle \Longrightarrow \bot$
($\wedge$I)	$\triangle \Longrightarrow A \wedge B$	**intro**	$\triangle \Longrightarrow A$ (1/2)
			$\triangle \Longrightarrow B$ (1/2)
($\Rightarrow$I)	$\triangle \Longrightarrow A \Rightarrow B$	**intro**	$A \wedge \triangle \Longrightarrow B$
($\vee I$)	$\triangle \Longrightarrow A_1 \vee \ldots \vee A_n$	**remove r** i	$\triangle \Longrightarrow A_1 \vee \ldots \vee A_{i-1} \vee A_{i+1} \vee \ldots \vee A_n$
Quantoren			
($\forall I$)	$\triangle \Longrightarrow \forall x.A$	**intro**	$\triangle \Longrightarrow A$
($\exists I$)	$\triangle \Longrightarrow \exists x.A$	**intro**	$\triangle \Longrightarrow A[x :=?i]$ (?i Gamma Variable i)
		intro more	$\triangle \Longrightarrow A[x :=?i] \vee \exists x.A$
Eliminierungsregeln			
($\vee$E)	$(A \vee B) \wedge \triangle \Longrightarrow \Phi$	**elim**	$A \wedge \triangle \Longrightarrow \Phi$ (1/2)
			$B \wedge \triangle \Longrightarrow \Phi$ (2/2)
($\Rightarrow$E)	$(A \Rightarrow B) \wedge \triangle \Longrightarrow \Phi$	**elim**	$B \wedge \triangle \Longrightarrow \Phi$ (1/2)
			$\neg\Phi \wedge \triangle \Longrightarrow A$ (2/2)
($\Rightarrow$E)	$(A \Rightarrow B) \wedge A \wedge \triangle \Longrightarrow \Phi$	**apply** 2 1	$B \wedge (A \Rightarrow B) \wedge A \wedge \triangle \Longrightarrow \Phi$
($\wedge$E)	$A_1 \wedge \ldots \wedge A_n \Longrightarrow \Phi$	**remove l** i	$A_1 \wedge \ldots \wedge A_{i-1} \wedge A_{i+1} \wedge \ldots \wedge A_n \Longrightarrow \Phi$
Quantoren			
($\forall E$)	$(\forall x.A) \wedge \triangle \Longrightarrow \Phi$	**elim**	$A[x :=?i] \wedge \triangle \Longrightarrow \Phi$
		elim more	$A[x :=?i] \wedge (\forall x.A) \wedge \triangle \Longrightarrow \Phi$
($\exists E$)	$(\exists x.A) \wedge \triangle \Longrightarrow \Phi$	**elim**	$A \wedge \triangle \Longrightarrow \Phi$
Zusätzliche Regeln			
(Cut)	$\triangle \Longrightarrow \Phi$	**cut** A	$\triangle \Longrightarrow A$ (1/2)
			$A \wedge \triangle \Longrightarrow \Phi$ (2/2)
(Instance)	A[?i]	**instance** i = N	A[?i := N]
(move2L)	$\triangle \Longrightarrow A \vee \Phi$	**move r**	$\neg A \wedge \triangle \Longrightarrow \Phi$
(move2R)	$A \wedge \triangle \Longrightarrow \Phi$	**move l**	$\triangle \Longrightarrow \neg A \vee \Phi$
(Cyclic Shift links)	$A_1 \wedge \ldots \wedge A_n \Longrightarrow \Phi$	**shift l u** i	$A_{i+1} \wedge \ldots \wedge A_n \wedge A_1 \wedge \ldots \wedge A_i \Longrightarrow \Phi$
		shift l d i	$A_{n-i+1} \wedge \ldots \wedge A_n \wedge A_1 \wedge \ldots \wedge A_{n-i} \Longrightarrow \Phi$
(Cyclic Shift rechts)	$\triangle \Longrightarrow A_1 \vee \ldots \vee A_n$	**shift r u** i	$\triangle \Longrightarrow A_{i+1} \vee \ldots \vee A_n \vee A_1 \vee \ldots \vee A_i$
		shift r d i	$\triangle \Longrightarrow A_{n-i+1} \vee \ldots \vee A_n \vee A_1 \vee \ldots \vee A_{n-i}$

Abbildung 4.4: Sequenten-Kalkül in NED

4.2 Gleichheit in NED

In NED begegnen uns zwei Arten von Gleichheit. Die erste wollen wir **externe** Gleichheit nennen. Sie ist unabhängig von einer speziellen Logik (wie HOL) und Hauptbestandteil der Gleichheitsregeln. Definieren wir neue Axiome/Lemmata der Form $A{=}B$ (A,B Terme desselben Typs), so verwenden wir hiefür die externe Gleichheit. Mit Hilfe der angegebenen Axiome oder Lemmata und unserer Gleichheitsregeln können wir dann Gleichheitsbeweise führen, die lediglich auf externer Ebene ablaufen.

Beispiel eines typischen Gleichheitsbeweises:

type X; const A:X; const B:X; const C:X; (* Deklarationen *)
axiom Ax1 := A = B ; (* 1. Voraussetzung *)
axiom Ax2 := B = C ; (* 2. Voraussetzung *)

goal A = C ; (* Zeige A = C *)
Ersetze A durch B gemäß Ax1
B = C
Ersetze B durch C gemäß Ax2
C = C
Reflexivität □

Sei X eine beliebige Menge und seien $a, b \in X$.
Wollen wir zwei Objekte a,b in unserer Sprache HOL miteinander vergleichen, so benötigen wir ein neues Gleichheitsprädikat.
Wir führen hierfür die logische Konstante

$$\doteq_X : X \to X \to \mathbb{B}$$

ein und nennen diese **interne** Gleichheit (auch Identität genannt).

Es handelt sich also um eine Funktion, die zwei Objekte vom Typ X als Eingabe nimmt und $\top$ liefert, sofern a und b „logisch gleich“ sind, $\bot$ sonst. Es stellt sich die Frage, wie man überprüft, ob zwei Elemente beliebigen Typs logisch gleich sind. Hier machte Leibniz folgende Feststellung: „Zwei Objekte sind genau dann gleich, wenn für sie dieselben Eigenschaften gelten.“
Dies führt uns zur *Leibnizschen Charakterisierung* der Gleichheit:

$$(a \doteq_X b) = \forall p \in X \to \mathbb{B} : pa \Rightarrow pb$$

Interessanterweise definieren wir hier die interne Gleichheit mit Hilfe der externen. Im Gegensatz zur externen, erweitert die interne Gleichheit die Signatur unseres Systems.

Da in NED der Sequenten-Kalkül fest in das System integriert ist, spendieren wir zusätzlich noch folgende Erweiterungsregeln:

$$\frac{\triangle \wedge A \Rightarrow B \quad \triangle \wedge B \Rightarrow A}{\triangle \Rightarrow A \doteq_{\mathbb{B}} B}\ (I_{=_{\mathbb{B}}})$$

Wenn A,B $\in \mathbb{B}$ und A $\Leftrightarrow$ B, dann gilt A = B (*Boolsche Extensionalität*)

$$\frac{\triangle \Rightarrow \forall p \in X \to \mathbb{B}.pA \Rightarrow pB}{\triangle \Rightarrow A \doteq_X B}\ (I_{=_X})$$

(*Leibniz Gleichheit*)

$$\frac{\triangle \Rightarrow \forall x \in X.fx \doteq_Y gx}{\triangle \Rightarrow f \doteq_{X \to Y} g}\ (I_{=_{X \to Y}})$$

Wenn die beiden Funktionen f,g $\in X \to Y$ für alle Argumente gleich sind, dann sind die Funktionen gleich.(*Funktionale Extensionalität*)

Man muss beachten, dass es sich bei den obigen Regeln um drei verschiedene Arten von Regeln handelt. Während die Boolsche Extensionalität als einfaches Lemma deklariert und abgeleitet werden kann, handelt es sich bei der Leibniz Gleichheitsregel um eine definierende Regel für die interne Gleichheit. Bei der Funktionalen Extensionalität handelt es sich um eine axiomatische Regel, die man oft in mathematischen Beweissystemen zur Verfügung stellt. In [5] werden Formeln angegeben, deren Gültigkeit nur durch Hinzunahme der Funktionalen Extensionalität gezeigt werden können.

Wie in 4.1 diskutiert, könnte man auch auf die feste Integration der Erweiterungsregeln verzichten und diese als Lemma, Definition oder Axiom zur Verfügung stellen. Durch die Integration hat der Benutzer aber den Vorteil, dass er weniger Beweiskommandos benötigt.

Beispiel 4.2.1

	$\forall g : X \to Y.\exists f : X \to Y.g \doteq f$	
$\dashv$	$\exists f : X \to Y.g \doteq f$	(*intro*)
$\dashv$	$g \doteq g$	(*intro and instance* $f := g$)
$\dashv$	$\forall x : X.gx \doteq gx$	(*intro*)
$\dashv$	$gx \doteq gx$	(*intro*)
$\dashv$	$\forall p : Y \to \mathbb{B}.p(gx) \Rightarrow p(gx)$	(*intro*)
$\dashv$	$p(gx) \Rightarrow p(gx)$	(*intro*)
$\dashv$	$\top$	(*simplify*)

Substitutionsregel für interne Gleichheit

Wissen wir, dass zwei Terme N, N' logisch gleich sind, also N $\doteq$ N' gilt, so wäre es nützlich, eine Substitutionsregel zu haben (ähnlich der Substitutionsregel für die externe Gleichheit), die es erlaubt, Vorkommen von N/N' innerhalb eines weiteren Terms M durch N'/N zu ersetzen.
Substitutions Regel für interne Gleichheit:

$$\frac{\triangle \Rightarrow N \doteq N' \qquad \triangle \Rightarrow M[x := N]}{\triangle \Rightarrow M[x := N']}\ (IntErsL)$$

$$\frac{\triangle \Rightarrow N \doteq N' \qquad \triangle \Rightarrow M[x := N']}{\triangle \Rightarrow M[x := N]}\ (IntErsR)$$

Ableitung (verkürzt) (IntErsL)

$\triangle \Rightarrow M[x := N']$	$=$	$\triangle \Rightarrow (M[x := N'] \wedge \top)$	BAx
$\dashv$	$=$	$\triangle \Rightarrow (M[x := N'] \wedge M[x := N])$	BAx, Vor.
$\dashv$	$=$	$\triangle \Rightarrow ((\neg M[x := N] \vee M[x := N']) \wedge M[x := N])$	BAx
$\dashv$	$=$	$\triangle \Rightarrow ((M[x := N] \Rightarrow M[x := N']) \wedge \top)$	Def $\Rightarrow$, Vor.
$\dashv$	$=$	$\triangle \Rightarrow (((\lambda x.M[x])N \Rightarrow (\lambda x.M[x])N') \wedge (N \doteq N'))$	β, BAx, Vor.
$\dashv$	$=$	$\triangle \Rightarrow (((\lambda x.M[x])N \Rightarrow (\lambda x.M[x])N') \wedge (\forall p.pN \Rightarrow pN'))$	Def. $\doteq$
$\dashv$	$=$	$\triangle \Rightarrow (\forall p.pN \Rightarrow pN')$	LAx
$\dashv$	$=$	$\triangle \Rightarrow (N \doteq N')$	Def. $\doteq$
$\dashv$	$=$	$\top$	Vor.

Ableitung (verkürzt) (IntErsR)

$\triangle \Rightarrow M[x := N]$	$=$	$\triangle \Rightarrow ((M[x := N'] \Rightarrow M[x := N]) \wedge \top)$	analog IntErsL
$\dashv$	$=$	$\triangle \Rightarrow ((\top \Rightarrow (M[x := N'] \Rightarrow M[x := N])) \wedge \top)$	BAx
$\dashv$	$=$	$\triangle \Rightarrow ((FN \Rightarrow FN') \wedge (N \doteq N'))$	β, BAx, Vor.
$\dashv$	$=$	$\triangle \Rightarrow ((FN \Rightarrow FN') \wedge (\forall p.pN \Rightarrow pN'))$	Def. $\doteq$
$\dashv$	$=$	$\triangle \Rightarrow (\forall p.pN \Rightarrow pN')$	LAx
$\dashv$	$=$	$\triangle \Rightarrow (N \doteq N')$	Def. $\doteq$
$\dashv$	$=$	$\top$	Vor.
		mit $F = (\lambda x.M[x] \Rightarrow M[x := N])$	

□

Im Gegensatz zur Ersetzungsregel für die externe Gleichheit erlaubt die Substitutionsregel für interne Gleichheit lediglich kaperfreies Ersetzen von Termen. Wir können aber zwei weitere Regeln angeben, die genau dann Kapern von

freien Variablen erlauben, wenn diese zuvor allquantifiziert wurden. Wir geben nur eine der beiden Regeln an. Die symmetrische Regel kann analog hergeleitet werden.

Notation
Seien P,Q Terme, mit FV(P) = $\{x_1, \ldots, x_n\}$.

- $\lambda FV(P).Q :=^{def} \lambda x_1 \ldots \lambda x_n.Q$
- $\forall FV(P).Q :=^{def} \forall x_1 \ldots \forall x_n.Q$
- $(\lambda FV(P).Q)(FV(P)) :=^{def} (\lambda x_1 \ldots \lambda x_n.Q)x_1 \ldots x_n$

Substitutions Regel für interne Gleichheit (mit Kapern)

$$\frac{\triangle \Rightarrow \forall FV(T).N \doteq N' \qquad \triangle \Rightarrow M[(\lambda FV(T).N)]}{\triangle \Rightarrow M[(\lambda FV(T).N')]}\ (KapIntErsL)$$

mit $T = N \doteq N'$

Ableitung (nach G. Smolka):

$$\dfrac{\dfrac{\dfrac{\triangle \Rightarrow \forall FV(T).N \doteq N'}{\triangle \Rightarrow \forall FV(T).(\lambda FV(T).N)(FV(T)) \doteq (\lambda FV(T).N')(FV(T))}\ (\beta)}{\triangle \Rightarrow (\lambda FV(T).N) \doteq (\lambda FV(T).N')}\ (Fun.Ext.) \qquad \triangle \Rightarrow M[x := (\lambda FV(T).N)]}{\triangle \Rightarrow M[x := (\lambda FV(T).N')]}\ (IntErsL)$$

Im letzten Ableitungsschritt dürfen wir die Substitutionsregel für interne Gleichheit *ohne* Kapern verwenden, da es sich bei $(\lambda FV(T).N)$ und $(\lambda FV(T).N')$ jeweils um geschlossene Terme handelt, also Terme bei denen alle auftretenden Variablen gebunden sind.

Beispiel 4.2.2
Seien g,h : $X \to \mathbb{B}$ und $f : (X \to \mathbb{B}) \to \mathbb{B}$ beliebige Funktionen.
Nehmen wir an, dass gilt:

(1) $\forall x : X.gx \doteq hx$ (2) $f(\lambda x : X.gx)$

Mit Hilfe der Regel (KapIntErsL) folgt, dass auch $f(\lambda x : X.hx)$ gültig ist.

Alle in diesem Abschnitt angegebenen Regeln stehen dem Benutzer auch in unserem Beweisassistenten NED zur Verfügung. Die folgende Tabelle erweitert die Tabelle aus Abbildung 4.4.

Regel	**Goal**^{T}	**Kommando**	**Goal**$^{T+1}$
(Boolsche Extensionalität)	$\triangle \Longrightarrow A \doteq_B B$ $(A \doteq_B B) \wedge \triangle \Longrightarrow \Phi$	**intro** **elim**	$\triangle \Longrightarrow (A \Rightarrow B) \wedge (B \Rightarrow A)$ $(A \Rightarrow B) \wedge (B \Rightarrow A) \wedge \triangle \Longrightarrow \Phi$
(Leibniz Gleichheit)	$\triangle \Longrightarrow A \doteq_X B$ $(A \doteq_X B) \wedge \triangle \Longrightarrow \Phi$	**intro** **elim**	$\triangle \Longrightarrow \forall p : X \to B.pA \Rightarrow pB$ $(\forall p : X \to B.pA \Rightarrow pB) \wedge \triangle \Longrightarrow \Phi$
(Funktionale Extensionaliät)	$\triangle \Longrightarrow f \doteq_{X \to Y} g$ $(f \doteq_{X \to Y} g) \wedge \triangle \Longrightarrow \Phi$	**intro** **elim**	$\triangle \Longrightarrow \forall x : X.fx \doteq_Y gx$ $(\forall x : X.fx \doteq_Y gx) \wedge \triangle \Longrightarrow \Phi$
(IntErsL) (IntErsR)	$N \doteq N' \wedge \triangle \Longrightarrow M[N]$ $N \doteq N' \wedge \triangle \Longrightarrow M[N']$	**rew** (Pos) **with** 1 **l** **rew** (Pos) **with** 1 **r**	$N \doteq N' \wedge \triangle \Longrightarrow M[N']$ $N \doteq N' \wedge \triangle \Longrightarrow M[N]$
(KapIntErsL)	$(\forall x_1 \ldots x_n.N \doteq N') \wedge \triangle$ $\Longrightarrow M[(\lambda x_1 \ldots x_n.N)]$ mit $FV(N, N') = \{x_1, \ldots, x_n\}$	**rew** (Pos) **with** 1 **l**	$(\forall x_1 \ldots x_n.N \doteq N') \wedge \triangle$ $\Longrightarrow M[(\lambda x_1 \ldots x_n.N')]$
(KapIntErsR)	$(\forall x_1 \ldots x_n.N \doteq N') \wedge \triangle$ $\Longrightarrow M[(\lambda x_1 \ldots x_n.N')]$ mit $FV(N, N') = \{x_1, \ldots, x_n\}$	**rew** (Pos) **with** 1 **r**	$(\forall x_1 \ldots x_n.N \doteq N') \wedge \triangle$ $\Longrightarrow M[(\lambda x_1 \ldots x_n.N)]$

Abbildung 4.5: Sequenten-Kalkül in NED (Erweiterung durch Identitätsregeln)

4.3 Gamma - Variablen

Die in diesem Abschnitt folgenden Untersuchungen basieren auf Techniken, die in [4] eingeführt werden.
Wir wollen die Ableitungsregel (∃ I) genauer betrachten.

$$\frac{\triangle \Rightarrow A[x := M]}{\triangle \Rightarrow \exists x.A} \ (\exists I)$$

Diese Regel erlaubt es, den Existenzquantor rechts vom Sequenten (⇒) dann zu eliminieren, wenn wir einen Term angeben können, der denselben Typ wie die gebundene Variable x hat. Es werden alle Auftreten von x in A durch M ersetzt. Die Regel (∀ E) verhält sich analog.

In vielen Fällen entscheidet die Wahl eines geeigneten Terms M über den Erfolg des Beweises. Zum Beweis der folgenden Formel jedoch, ist die Wahl des Terms M nicht entscheidend:

	$\forall x \in \mathbb{B}.\exists y \in \mathbb{B}.(x \vee \neg x) \vee y$	
$\dashv$	$\exists y \in \mathbb{B}.(x \vee \neg x) \vee y$	(∀ I)
$\dashv$	$(x \vee \neg x) \vee \mathbf{M}$	(∃ I)
$\dashv$	$\top$	(Bool)

Die Wahl des Terms M ist deshalb unwichtig, da die Formel $(x \vee \neg x) \vee M$ eine Tautologie ist, also eine universell gültige Formel.

In NED werden die Existenzquantoren rechts und die Allquantoren links vom Sequenten wie folgt eliminiert. Es werden sogenannte **Gamma-Variablen** für die entsprechende gebundene Variable eingesetzt. Diese werden von dem Zeitpunkt der Eliminierung an als freie Variablen betrachtet und können danach instanziiert werden.
Mit $GA = \{\Gamma i | i \in \mathbb{N}\}$ bezeichnen wir die Menge aller Gammavariablen.

Beispiel 4.3.1

	$\forall x \in \mathbb{B}.\exists y \in \mathbb{B}.x = y$	
$\dashv$	$\exists y \in \mathbb{B}.x = y$	(∀ I)
$\dashv$	$x = \Gamma 0$	(∃ I)
$\dashv$	$x = x$	(Instance 0 = x)
$\dashv$	$\top$	(Reflexivität)

Probleme mit Gamma Variablen

Vertauschen wir im letzten Beispiel die beiden Quantoren, so erhalten wir die Formel $\exists x \in \mathbb{B}.\forall y \in \mathbb{B}.x = y$. Diese Formel ist nicht gültig, dennoch könnte man sie mit Hilfe der Gamma Variablen zu beweisen versuchen.

Beispiel 4.3.2

$$\begin{array}{lll} & \exists x \in \mathbb{B}.\forall y \in \mathbb{B}.x = y & \\ \dashv & \forall y \in \mathbb{B}.\Gamma 0 = y & (\exists \text{ I}) \\ \dashv & \Gamma 0 = y & (\forall \text{ I}) \\ \dashv & y = y & \end{array}$$

Die letzte Ableitung ist nicht erlaubt, da die freie Variable y zum Zeitpunkt der Existenzquantoreliminierung noch nicht bekannt war. Würde man diese Instanziierungen erlauben, wäre das Deduktionssystem bezüglich der Semantik inkorrekt.
Um diese Instanziierungen zu vermeiden, führen wir eine Relation R ein. In der Literatur heißt diese Relation *Variable-Condition*.

$$R \subseteq FV \times FV \qquad \text{, mit FV = Var} \cup \text{GA}$$

Ein Tupel $(x, y) \in R$ soll so interpretiert werden, dass die freie Variable x „älter“ ist als die freie Variable y. Die Relation wird wie folgt erweitert:

1. Durch Allquantoreliminierung rechts und Existenzquantoreliminierung links

$$\frac{\triangle \Rightarrow A \qquad \text{x nicht frei in } \triangle}{\triangle \Rightarrow \forall x.A}\ (\forall I) \qquad R := R \cup (FV(\triangle, A) \times \{x\})$$

$$\frac{\triangle \wedge A \Rightarrow \Phi \qquad \text{x nicht frei in } \triangle, \Phi}{\triangle \wedge (\exists x.A) \Rightarrow \Phi}\ (\exists E) \qquad R := R \cup (FV(\triangle, \Phi, A) \times \{x\})$$

2. Durch Instanziierung von Gammavariablen

   Sei $\sigma : GA \rightharpoonup PT$ eine Instanziierung. Dann erhalten wir für
   $R := R \cup \{(z, x) | x \in Dom\ \sigma \wedge z \in FV(\sigma(x))\}$

Nun können wir die erlaubten σ-Instanziierungen von den nicht erlaubten unterscheiden.

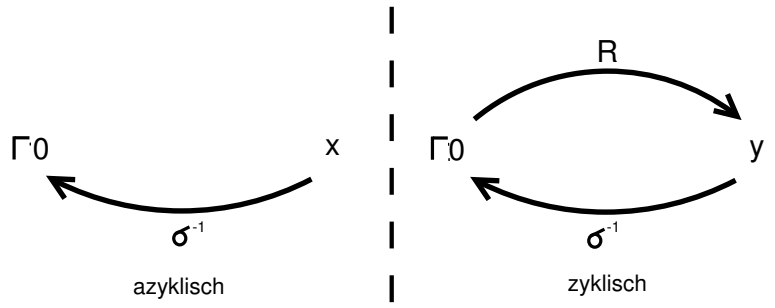

Abbildung 4.6: Relationsgraphen für Beispiel 4.3.1 und 4.3.2

Sei R eine Variable Condition und σ eine Instanziierung. σ ist eine *erlaubte* Instanziierung, wenn die Relation R wohlfundiert ist.

Da wir in unserem System NED nur endliche Relationen R und endliche Instanziierungen σ betrachten werden, können wir den Wohlfundiertheitstest durch einen effizienten Azyklustest auf dem entsprechenden Graphen ersetzen.

Abbildung 4.6 zeigt die entsprechenden Graphen zu den Beispielen 4.3.1 und 4.3.2 . Man erkennt sofort, dass im zweiten Beispiel die Instanziierung von $\Gamma 0 \longmapsto y$ wegen des entstehenden Zyklus in der Variable-Condition nicht erlaubt ist.

Wir wollen ein weiteres Beispiel betrachten und in unserem Beweisassistenten NED die Gültigkeit folgender Formel zeigen:

Beispiel 4.3.3
Sei f eine Funktion vom Typ $X \to \mathbb{B}$.

goal $\exists x : X.fx \Rightarrow (\forall x : X.fx)$

$\dashv$	$\top \Longrightarrow f(\Gamma 0) \Rightarrow (\forall x : X.fx)$	(intro)	$(\exists I)$
$\dashv$	$f(\Gamma 0) \Longrightarrow \forall x : X.fx$	(intro)	$(\Rightarrow I)$
$\dashv$	$f(\Gamma 0) \Longrightarrow fx$	(intro)	$(\forall I)$
$\dashv$	*ERROR*	(instance 0 = x)	(Substitution)

Wieso scheitert unser Beweisversuch trotz der Gültigkeit der Formel?
Schauen wir uns die entsprechende Variable-Condition R an. ($R = \{(\Gamma 0, x), (x, \Gamma 0)\}$)
Da R nicht wohlfundiert ist, folgt, dass die Instanziierung $\Gamma 0 \mapsto x$ nicht erlaubt war. NED reagiert also korrekt, indem es die Instanziierung nicht erlaubt.

Dennoch sollte es möglich sein, die Gültigkeit dieser Formel in NED zu zeigen. Indem wir den Existenzquantor zweimal eliminieren, gelingt es uns, den Beweis zu führen.

goal $\exists x : X.fx \Rightarrow (\forall x : X.fx)$

$\dashv$	$\top \Longrightarrow (f(\Gamma 0) \Rightarrow (\forall x : X.fx))$ $\vee(\exists x : X.fx \Rightarrow (\forall x : X.fx)$	(intro **more**)	$(\exists I)$
$\dashv$	$f(\Gamma 0) \Longrightarrow (\forall x : X.fx))$ $\vee(\exists x : X.fx \Rightarrow (\forall x : X.fx))$	(intro)	$(\Rightarrow I)$
$\dashv$	$f(\Gamma 0) \Longrightarrow fx \vee (\exists x : X.fx \Rightarrow (\forall x : X.fx))$	(intro)	$(\forall I)$
$\dashv$	$f(\Gamma 0) \Longrightarrow (\exists x : X.fx \Rightarrow (\forall x : X.fx)) \vee fx$	(shift r u)	(Cyclic Shift)
$\dashv$	$f(\Gamma 0) \Longrightarrow (f(\Gamma 1) \Rightarrow (\forall x : X.fx)) \vee fx$	(intro)	$(\exists I)$
$\dashv$	$f(\Gamma 0) \wedge f(\Gamma 1) \Longrightarrow (\forall x : X.fx) \vee fx$	(intro)	$(\Rightarrow I)$
$\dashv$	$f(\Gamma 0) \wedge fx \Longrightarrow (\forall x : X.fx) \vee fx$	(instance 1 = x)	(Subst.)
$\dashv$	$\top$	(simplify)	(Bool)

□

Der Trick besteht darin, die existenzquantifizierte Ausgangsformel bei der ersten Eliminierung ($\Gamma 0$) durch das Kommando **intro more** nicht zu verwerfen, sondern disjunktiv verknüpft beizubehalten. Das analoge Kommando **elim more** bewirkt, dass eine Formel bei einer Allquantoreliminierung auf der linken Seite des Sequenten konjunktiv verknüpft beibehalten wird.

Dadurch können wir zu einem späteren Zeitpunkt den Existenzquantor erneut eliminieren und erhalten die zweite Gammavariable $\Gamma 1$. Nun ist unsere Variable-Condition R = $\{(\Gamma 0, x), (x, \Gamma 1)\}$, nachdem wir die Instanziierung $\Gamma 1 \mapsto x$ vorgenommen haben, wohlfundiert und unsere Instanziierung erlaubt.

Ein alternativer Ansatz, die Gültigkeit solcher Formeln zu zeigen, besteht darin, das Konzept der Gamma- und Deltavariablen zu erweitern und sogenannte *liberalisierte* Deltavariablen einzuführen. Bei diesem Ansatz wird versucht, die Variable-Condition möglichst minimal zu halten, und nur solche Abhängigkeiten von Variablen zu beachten, die logisch zu rechtfertigen sind. [4]

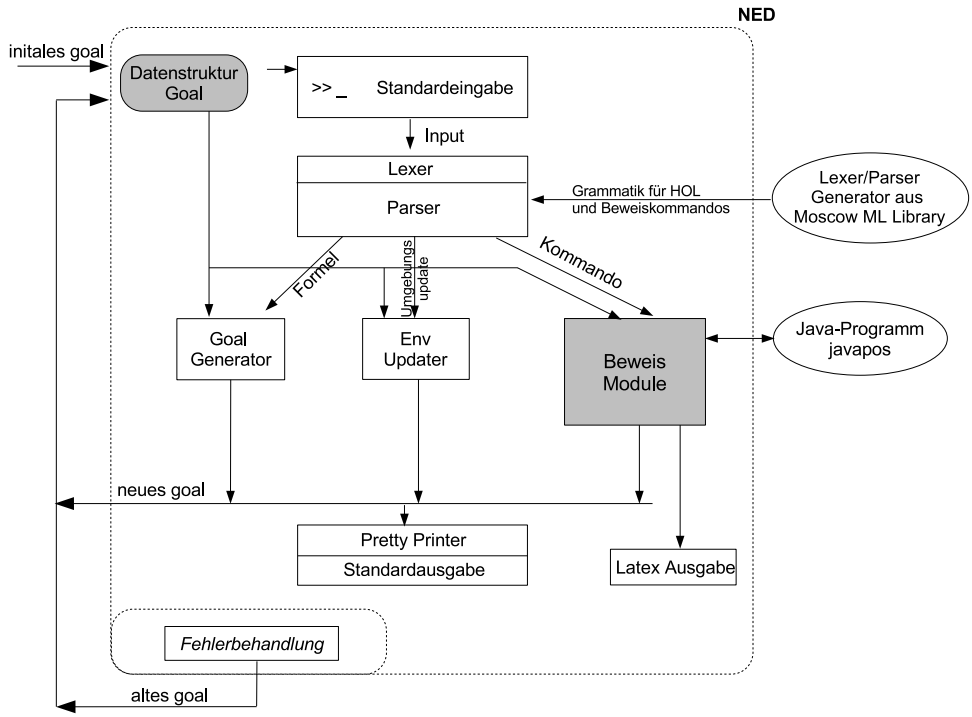

Abbildung 4.7: Architektur NED

4.4 Architektur von NED

Abbildung 4.7 zeigt die Architektur des Beweisassistenten NED. Das System besteht im Wesentlichen aus der zentralen Datenstruktur Goal, einer Hauptschleife und verschiedenen Beweismodulen, auf die wir im Einzelnen noch eingehen werden.
Die Hauptdatenstruktur ist ein Quadrupel bestehend aus

- einer Liste von Prätermen (Datenstruktur Pter)
- einer Liste von Umgebungen (Datenstruktur Env)
- Gammavariablen (Datenstruktur Gamma)
- Definitionen (Datenstruktur Definition)

In der Datenstruktur Env merken wir uns die Menge der Typkonstanten (TC), freien Variablen (Var) und Konstanten (VC) mit den entsprechenden Funktionen $ty : VC \rightarrow Ty$ und der Typumgebung $\Gamma : Var \rightarrow Ty$.

Die Datenstruktur Gamma ist eine Graphdatenstruktur, in der wir uns Abhängigkeiten von Variablen merken. (siehe Abschnitt 4.3)
Bei der Datenstruktur Definition handelt es sich um eine Hashtabelle, in der Platzhaltern Terme zugeordnet werden.

Um die Funktionsweise des Systems zu verstehen, schauen wir uns Beispielabläufe des Programms sowohl im ND-Modus als auch im Rewriting-Modus an.

Beweisvorgang im ND-Modus

Wir wollen zeigen, dass es einen „für Autofahrer schlechten Ampelzustand“ gibt. Dazu starten wir NED.

Das Programm fordert uns mit

> _

zur Eingabe auf. Im Hintergrund wurde eine initiale *goal* Instanz, bestehend aus einer leeren Liste von Prätermen, einer initialen Umgebung, einem leeren Gamma-Graphen und einer leeren Definitionen-Hashtabelle angelegt.
Die initiale Umgebung kennt nur die Typkonstante *bool*. Die logischen Konstanten sind sowieso fest in das System integriert.

Die Eingabe

> **type Ampel_Zustand;**

wird vom *Parser* als Umgebungsupdate erkannt und entsprechend an den *Env Updater* weitergeleitet. Dieser verändert die Umgebung der aktuellen *goal* Instanz, indem die neue Typkonstante *Ampel_Zustand* der Menge TC hinzugefügt wird. Das neue goal wird gesetzt. Damit ist der erste Schleifendurchlauf beendet, und der Benutzer kann erneut eine Eingabe tätigen.

> **const gruen :Ampel_Zustand;**
> **const gelb :Ampel_Zustand;**
> **const rot :Ampel_Zustand;**

Diese drei Eingaben werden ebenfalls zunächst vom *Parser* verarbeitet und an den *Env Updater* weitergeleitet. Dieses Mal wird die Menge der Wertkonstanten (VC) um die Elemente *gruen, gelb, rot* erweitert. Außerdem wird durch die Typangaben die Funktion ty entsprechend erweitert.

Bei der Eingabe von

> **def schlecht := λ z: Ampel_Zustand. z = rot;**

wird zunächst der Lambdaausdruck hinter dem Zuweisungsoperator geparst. In NED sind Ausdrücke intern in De Bruijnscher Darstellung gespeichert. Damit der Variablenname *z*, den der Benutzer sinnvoll gewählt hat, nicht verloren geht, wird dieser ebenfalls im *Env Updater* in der Umgebung gespeichert. Außerdem wird ein Eintrag in der Hashtabelle unter dem Namen *schlecht* gemacht.

Kommen wir zur Theoremeingabe

> **goal ∃ z:Ampel_Zustand. schlecht z;**

Wir behaupten also, dass es einen Zustand gibt, der *schlecht* ist. Wir haben zuvor das Prädikat *schlecht* definiert. Es wird auch hier der Ausdruck geparst und die gebundene Variable *z* der Umgebung hinzugefügt. Nachdem der *Parser* einen enstprechenden Präterm erzeugt hat, wird dieser im *Goal Generator* der Prätermliste hinzugefügt und somit ein neues *goal* erzeugt.
Dieses neue *goal* wird dann über den *Pretty Printer* ausgegeben, der für die übersichtliche Ausgabe insbesondere von größeren Formeln zuständig ist.

Goal 1/1
∃ z. schlecht z

Die Eingabe

> **intro;**

wird vom Parser als Beweiskommando erkannt und an das Beweismodul der Introduction-Regeln weitergeleitet. Gemäß der Regel ∃R wird nun der Existenzquantor eliminiert und die Gamma Variable Γ0 (im System ? 0) im Gamma-Graphen eingeführt. Ausgabe:

Goal 1/1
schlecht ? 0

Mit

> **instance 0 = rot;**

ersetzen wir Γ0 gleich wieder durch den Ausdruck *rot*, den der Parser zunächst auswerten musste. Außerdem wird in diesem Schritt überprüft, ob die Instanziierung erlaubt ist. Unter anderem muss der Instanziierungsausdruck denselben Typ wie die Gamma Variable haben, was durch den eingebauten Typchecker überprüft wird. Weitere Einschränkungen wurden in Abschnitt 4.3 erläutert. Ausgabe:

Goal 1/1
schlecht rot

Nun befindet sich an oberster Termposition eine Applikation, bei der unsere Definition *schlecht* verwendet wird. Mit dem Kommando

> **intro;**

wird der Eintrag unter *schlecht* in der Hashtabelle nachgeschlagen und der Bezeichner automatisch ersetzt.Ausgabe:

Goal 1/1
(λ z. z = rot) rot

Es folgt eine β-Reduktion mit dem Kommando

> **lam;**

die auch in einem eigenen Beweismodul implementiert ist. Ausgabe:

Goal 1/1
rot = rot

Mit dem Reflexivitätskommando

> **ref;**

was im Gleichheitsbeweismodul implementiert ist, meldet uns das System:

PROOF DONE

Bemerkungen
Insgesamt haben elf Schleifendurchläufe stattgefunden. Die Prätermliste und die Liste der Umgebungen waren maximal mit einem Element gefüllt. Es werden dennoch innerhalb der Goal Datenstruktur Listen verwendet, da es in größeren Beweisen zu einer Aufspaltung in Subgoals kommt, die eigenständige Umgebungen benötigen.

Alternativ zur direkten Definition des Prädikats *schlecht*, könnten wir auch mit

> **const schlecht : Ampel_Zustand → bool;**
> **axiom schlecht_ax := schlecht = (λ z: Ampel_Zustand. z = rot);**

zunächst die Wertkonstante *schlecht* deklarieren und anschließend axiomatisch über die externe Gleichheit definieren, dass *schlecht* einer Funktion entspricht, die Ampelzustände als Argumente nimmt und true liefert, falls der betreffende Zustand rot ist.
Wenn wir nun innerhalb des Beweises an den Punkt

Goal 1/1
schlecht rot

gelangen, kann das System nicht mehr automatisch den Lambdaausdruck für *schlecht* einsetzen. Vielmehr muss der Benutzer mit Hilfe des Ersetzungskommandos

> **rew (1) with schlecht_ax l;**

die Konstante *schlecht* ersetzen. Dies gelingt nur, wenn die entsprechenden Terme nach dem First-Order Unifikationsalgorithmus von Robinson unifizierbar sind. Dieser Algorithmus ist im Ersetzungsmodul implementiert.

Beweisvorgang im Rewriting-Modus

Beginnen wir erneut mit einer initialen *goal* Instanz, wie zuvor beschrieben. Die Eingabe

> **type X;**

wird vom *Parser* als Umgebungsupdate erkannt. Der *Env Updater* fügt die neue Typkonstante X der Menge TC hinzu.

> **const EX : (X → bool) → bool;**
> **const ALL : (X → bool) → bool;**

Wir erweitern die Menge der Wertkonstanten (VC) um den Existenz- und Allquantor (Im Folgenden verwenden wir $\forall$ statt *ALL* und $\exists$ statt *EX* der Übersichtlichkeit halber). Die in unserem System eingebauten Quantorkonstanten können nur mit einer impliziten Funktion verwendet werden, also $\forall x.fx$ (Abkürzung für $\forall(\lambda x.fx)$). Dadurch, dass wir die Quantoren zusätzlich explizit deklarieren, können wir zum Beispiel auch die Formeln $\forall(\lambda x.fx)$ oder $\forall g$ angeben.

Es folgen mit

> **var x : X;**
> **var u : bool;**
> **var v : bool;**
> **var f : X → bool;**

vier Variablendeklarationen. Der *Env Updater* erweitert die Menge *Var* und die Funktion Γ entsprechend. Der Typ *bool* kann verwendet werden, ohne zuvor deklariert werden zu müssen, da er fest eingebaut ist.

Bei der Eingabe von

> **axiom IMP_DEF := u ⇒ v = ¬ u ∨ v;**
> **axiom EX_DEF := ∃ f = ¬∀ (λ x. f x);**
> **axiom DNEG := ¬¬ u = u ;**
> **axiom ALL_OR := ∀ f ∨ u = ∀ (λ x. f x ∨ u);**

werden zunächst die Formeln hinter dem Zuweisungsoperator geparst und analog zu Definitionen in der Hashtabelle *Definition* unter dem entsprechenden Namen gespeichert. Wir verwenden *Axiome*, um die Gültigkeit von Formeln in unserem Kontext anzugeben. Analog können wir *Lemmata* (Schlüsselwort **lemma**) angeben, die bei Verwendung als neues Subgoal bewiesen werden müssen.

Kommen wir zur Theoremeingabe

> **goal** $\neg\exists$ (λ **x. f x**) = $\forall$ (λ **x.** $\neg$ **f x**);

Der *Parser* erzeugt einen Präterm. Dieser wird im *Goal Generator* der Prätermliste hinzugefügt, welche somit um ein neues *goal* erweitert wird.
Im Rewriting-Modus wird der *Pretty Printer* für die Ausgabe nicht verwendet, da wir nach Möglichkeit die Formeln in einer einzigen Zeile ausgeben wollen.

Goal 1/1
$\neg\exists$ (λ **x. f x**) = $\forall$ (λ **x.** $\neg$ **f x**)

Mit dem ersten Beweiskommando

> **rew (1,1) with EX_DEF l;**

teilen wir dem System mit, dass wir den Subterm $\exists$ (λ x. f x) an Position (1,1) in unserer Formel mit der linken Seite des Axioms *EX_DEF* matchen und entsprechend ersetzen wollen. Dies geschieht im Ersetzungsmodul. Wie man die Position von Subtermen berechnet wird in Abschnitt A.2 genauer erklärt. Ausgabe:

Goal 1/1
$\neg\neg\forall$ (λ **x.** $\neg$(λ **x. f x**) **x**) = $\forall$ (λ **x.** $\neg$ **f x**)

Die β-Normalform dieser Formel berechnet das Kommando

> **lam;**

Alternativ kann auch eine einzige β-Reduktion des Subterms vorgenommen werden. Das Kommando hierfür lautet **rew (1,1,1,2,1,1) with beta**. Analoges gilt auch für η-Reduktionen. Ausgabe:

Goal 1/1
$\neg\neg\forall$ (λ **x.** $\neg$ **f x**) = $\forall$ (λ **x.** $\neg$ **f x**)

Nun ersetzen wir mit

> **rew (1) with DNEG l;**

die linke Seite unserer Gleichung unter Zuhilfename des Axioms *DNEG*. Ausgabe:

Goal 1/1
$\forall\ (\lambda\ \mathbf{x.}\ \neg\ \mathbf{f\ x}) = \forall\ (\lambda\ \mathbf{x.}\ \neg\ \mathbf{f\ x})$

Mit dem Reflexivitätskommando

>**ref;**

schließen wir den Beweis ab.
Damit wir das gerade bewiesene Theorem wiederverwenden können, speichern wir es mit

>**save FIRST;**

unter dem Namen *FIRST* in unserer Hashtabelle.

Wir wollen einen weiteren kleinen Beweis führen, in dem wir die Formel *FIRST* verwenden können.

> **goal** $\exists\ \mathbf{f} \Rightarrow \mathbf{u} = \forall\ (\lambda\ \mathbf{x.}\ \mathbf{f\ x} \Rightarrow \mathbf{u})$**;**

Zunächst verwenden wir die Definition der Implikation

> **rew (1) with IMP_DEF l;**

und erhalten folgende Ausgabe:

Goal 1/1
$\neg\exists\ \mathbf{f} \vee \mathbf{u} = \forall\ (\lambda\ \mathbf{x.}\ \mathbf{f\ x} \Rightarrow \mathbf{u})$

Nun würden wir gerne unsere Formel *FIRST* verwenden und die Applikation mit dem Existenzquantor durch die Allquantorapplikation ersetzen. In unserer Formel *FIRST* ist das Argument des Existenzquantor aber ein Lambdaausdruck. Damit sich der Subterm mit *FIRST* matchen lässt, müssen wir ihn η-expandieren. Mit folgendem Kommando

> **rew (1,1,1) with** $(\exists\ \mathbf{f} = \exists\ (\lambda\ \mathbf{x.}\ \mathbf{f\ x}))$ **;**

wird zunächst die Gleichung ($\exists$ f = $\exists$ (λ x. f x)) vom *Parser* verarbeitet. Anschließend wird versucht, den Subterm an Position (1,1,1) mit der linken Seite der Gleichung zu matchen und entsprechend zu ersetzen. Gelingt dies, wird im *Goal Generator* ein neues *Subgoal* erzeugt. Das heißt, die Gleichung muss ebenfalls bewiesen werden. Ausgabe:

Goal 1/2
$\neg\exists\ (\lambda\ \mathbf{x.}\ \mathbf{f\ x}) \vee \mathbf{u} = \forall\ (\lambda\ \mathbf{x.}\ \mathbf{f\ x} \Rightarrow \mathbf{u})$
Goal 2/2
$\exists\ \mathbf{f} = \exists\ (\lambda\ \mathbf{x.}\ \mathbf{f\ x})$

Jetzt dürfen wir die Formel *FIRST* verwenden.

> **rew (1,1) with FIRST l;**

Ist mehr als ein *Goal* aktiv, beziehen sich alle Beweiskommandos auf das erste *Goal*. Ausgabe:

Goal 1/2
$\forall\ (\lambda\ \mathbf{x}.\ \neg\ \mathbf{f}\ \mathbf{x})\ \vee\ \mathbf{u} = \forall\ (\lambda\ \mathbf{x}.\ \mathbf{f}\ \mathbf{x} \Rightarrow \mathbf{u})$
Goal 2/2
$\exists\ \mathbf{f} = \exists\ (\lambda\ \mathbf{x}.\ \mathbf{f}\ \mathbf{x})$

Mit den Kommandos

> **rew (1) with ALL_OR l;**
> **lam;**
> **rew (1,2,1) with IMP_DEF r;**
> **ref;**

beweisen wir das erste *Subgoal* vollständig. Es verbleibt das *Subgoal* der η-Expansion:

Goal 1/1
$\exists\ \mathbf{f} = \exists\ (\lambda\ \mathbf{x}.\ \mathbf{f}\ \mathbf{x})$

Mit dem Beweiskommando

> **lam;**

können wir die η-Normalform berechnen lassen. Ausgabe:

Goal 1/1
$\exists\ \mathbf{f} = \exists\ \mathbf{f}$

Das Reflexivitätskommando

>**ref;**

schließt erneut den Beweis ab.

Kapitel 5

Fallstudie: Induktionsbeweise

In diesem Kapitel beschäftigen wir uns beispielhaft mit Induktionsbeweisen. Wir werden die Kommutativität der Addition über natürliche Zahlen zeigen. Desweiteren wollen wir überprüfen, wie Induktionsbeweise mit dem Beweisassistenten NED realisiert werden können.

5.1 Induktionsaxiom

Eine der wichtigsten Beweismethoden für Aussagen über natürliche Zahlen ist die vollständige Induktion. Sie beruht auf dem sogenannten Induktionsaxiom

$$\forall f \in \mathbb{N} \to \mathbb{B}.f(0) \wedge (\forall x \in \mathbb{N}.f(x) \Rightarrow f(x+1)) \Rightarrow \forall x \in \mathbb{N}.f(x) \qquad (1)$$

Informell sagt dieses Axiom folgendes aus:
Instanziiert man $f \in \mathbb{N} \to \mathbb{B}$ mit der zu beweisenden Eigenschaft, so genügt es den Induktionsanfang f(0) und den Induktionsschritt $\forall x \in \mathbb{N}.f(x) \Rightarrow f(x+1)$ zu zeigen. Das Axiom liefert die Aussage, dass $f(x)$ für alle $x \in \mathbb{N}$ gilt.

5.2 Beipiel Kommutativität

5.2.1 Standard Induktionsbeweis

Seien $x, y \in \mathbb{N}$
Wir betrachten die Peano-Axiome:

$$0 + x = x \qquad (2)$$
$$(x+1) + y = (x+y) + 1 \qquad (3)$$

Lemma (5.2.1) $x + 0 = x$
Beweis (5.2.1)

Vollständige Induktion über x
Induktionsanfang: $x = 0$

$$x + 0 = 0 + 0 =^{(2)} 0 = x$$

Induktionsschritt: $x \to x + 1$

$$\begin{array}{lll} (x+1)+0 & =^{(3)} & (x+0)+1 \\ & =^{I.V.} & (x+1) \end{array}$$

□

Lemma (5.2.2) (Spezialfall Assoziativität)

$$(x + y) + 1 = x + (y + 1)$$

Beweis (5.2.2)
Vollständige Induktion über x
Induktionsanfang: $x = 0$

$$(x + y) + 1 = (0 + y) + 1 =^{(2)} y + 1 = (y + 1) =^{(2)} 0 + (y + 1)$$

Induktionsschritt: $x \to x + 1$

$$\begin{array}{lll} ((x+1)+y)+1 & =^{(3)} & ((x+y)+1)+1 \\ & =^{(I.V.)} & (x+(y+1))+1 \\ & =^{(3)} & (x+1)+(y+1) \end{array}$$

□

Satz (5.2.3) (Kommutativität der Addition über natürliche Zahlen)

$$x + y = y + x$$

Beweis (5.2.3)
Vollständige Induktion über x
Induktionsanfang: $x = 0$

$$x + y = 0 + y =^{(3)} y =^{(5.2.1)} y + 0 = y + x$$

Induktionsschritt: $x \to x + 1$

$$\begin{array}{lll} (x+1)+y & =^{(3)} & (x+y)+1 \\ & =^{(I.V.)} & (y+x)+1 \\ & =^{(5.2.2)} & y+(x+1) \end{array}$$

□

5.2.2 Induktionsbeweis in NED

Induktionsbeweise können im Beweissystem mit Hilfe des Induktionsaxioms formuliert werden. Zunächst jedoch müssen wir die natürlichen Zahlen mit Hilfe der Peano Axiome charakterisieren.

type NAT
const $0 : NAT$
const $+ : NAT \to NAT \to NAT$ (im Folgenden infix)
const $s : NAT \to NAT$ (Successor)

var $a : NAT$
var b : NAT
axiom $axiom1 := 0 + a = a$
axiom $axiom2 := s(a) + b = s(a + b)$

Das Induktionsaxiom wird wie folgt deklariert:
axiom $ind := \forall f \in \mathbb{N} \to \mathbb{B}.f(0) \wedge (\forall x \in \mathbb{N}.f(x) \Rightarrow f(x + 1)) \Rightarrow \forall x \in \mathbb{N}.f(x)$

Da wir für den Beweis der Kommutativität zwei Lemmata benötigen, müssen wir diese ebenfalls dem System bekannt machen.

lemma $lemma1 := a + 0 = a$
lemma $lemma2 := s(a + b) = a + s(b)$

Wir definieren die entscheidende Funktion f:
def $f := \lambda x \in NAT.\forall y : NAT.x + y = y + x$

Nun sind wir in der Lage, die Kommutativität der Addition über natürliche Zahlen zu beweisen:

goal $\forall x \in NAT.fx$

Wir beschränken uns im Folgenden auf den Beweis des Induktionsanfangs und Induktionsschrittes:

	$f(0)$	
$\dashv$	$(\lambda x.\forall y.x + y = y + x)0$	(Def.f)
$\dashv$	$\forall y.0 + y = y + 0$	(β)
$\dashv$	$0 + y = y + 0$	(Intro)
$\dashv$	$y = y + 0$	(rew (1) with axiom1 l)
$\dashv$	$y = y$	(rew (2) with lemma1 l)
$\dashv$	$\top$	(Bool)

Es folgt der Beweis des Induktionsschrittes:

	$\top$	$\Longrightarrow$	$\forall x.f(x) \Rightarrow f(s(x))$	
$\dashv$	$\top$	$\Longrightarrow$	$f(x) \Rightarrow f(s(x))$	(Intro)
$\dashv$	$f(x)$	$\Longrightarrow$	$f(s(x))$	(Intro)
$\dashv$	$(\lambda x'.\forall y.x' + y = y + x')x$	$\Longrightarrow$	$(\lambda x'.\forall y.x' + y = y + x')s(x)$	(Def. f)
$\dashv$	$\forall y.x + y = y + x$	$\Longrightarrow$	$\forall y.s(x) + y = y + s(x)$	(β)
$\dashv$	$\forall y'.x + y' = y' + x$	$\Longrightarrow$	$s(x) + y = y + s(x)$	(Intro)
$\dashv$	$\forall y'.x + y' = y' + x$	$\Longrightarrow$	$s(x + y) = y + s(x)$	(rew (2,1) with axiom2 l)
$\dashv$	$x+?1 =?1 + x$	$\Longrightarrow$	$s(x + y) = y + s(x)$	(Elim)
$\dashv$	$x + y = y + x$	$\Longrightarrow$	$s(x + y) = y + s(x)$	(Instance 1 = y)
$\dashv$	$x + y = y + x$	$\Longrightarrow$	$s(y + x) = y + s(x)$	(rew (2,1,2) with 1 l)
$\dashv$	$x + y = y + x$	$\Longrightarrow$	$y + s(x) = y + s(x)$	(rew (2,1) with lemma2 l)
$\dashv$	$\top$			(Ref)

□

Da Induktionsanfang und Induktionsschritt gültig sind, folgt laut Induktionsaxiom, $\forall x \in NAT.f(x)$
Analog können die beiden Lemmata bewiesen werden.

Dies schließt den Beweis für die Kommutativität der Addition zweier natürlicher Zahlen innerhalb der Beweisumgebung NED ab.

Kapitel 6

Weitere Assistenzsysteme

In diesem Kapitel wollen wir unseren Beweisassistenten NED in den Kontext von bekannten Assistenzsystemen einordnen. Wir schauen uns insbesondere das Prototype Verification System (**PVS**) [10] und den Beweisassistenten **Isabelle** [9] an.

In Kapitel 5 haben wir in NED gezeigt, dass die Summenfunktion für natürliche Zahlen kommutativ ist, indem wir unter anderem das Induktionsaxiom verwendet haben. Führen wir denselben Beweis in PVS, so benötigen wir hierfür genau ein Kommando *induct-and-simplify*.

Ausgabe PVS

```
komm :
⊢
{1} FORALL (m, n: nat): m + n = n + m

Rule? (induct-and-simplify ''n'')
By induction on n, and by repeatedly rewriting and simplifying,
Q.E.D.
```

Wenn wir uns die Spezifikation des Ausgangsproblems anschauen,

```
nat_bsp: THEORY
BEGIN
m, n: VAR nat
komm: THEOREM m + n = n + m
END nat_bsp
```

fällt auf, dass lediglich zwei freie Variablen *m,n* eines Typs *nat* und das zu beweisende Theorem *komm* spezifiziert werden. Im Vergleich zur NED Spezifikation, ist die PVS Version sehr klein. Dies führt uns zu einer weiteren Eigenschaft von

PVS.
PVS ist ähnlich wie die Mathematik in verschiedenen Theorien aufgebaut, die zum Teil in einer umfangreichen Bibliothek abgelegt sind, und bei Bedarf geladen werden können. So wurde in obigem Beispiel die Theorie über die natürlichen Zahlen inklusive des Induktionsaxioms zu Grunde gelegt. Die Möglichkeit, Theorien zu erstellen und diese in einer Indexstruktur abzulegen, gibt es in NED nicht.

Wie in NED, so findet man auch in PVS einen Basiskalkül, den Sequenten-Kalkül. Durch unübersichtlich viele Beweiskommandos können Beweise auf Kalkülebene geführt werden. PVS bietet auch die Möglichkeit, sogenannte Strategien (Taktiken) auszuführen, welche eine Sequenz von definierten Beweiskommandos ausführen, um den Beweisvorgang zu beschleunigen. Zum Beweis der Kommutativität der Addition wurden für den Benutzer unsichtbar zunächst das Induktionsaxiom korrekt instanziiert, und den Beweis mit Hilfe automatischer Induktionstechniken, wie zum Beispiel Rippling [6], geführt. Auf der Basis wurden Beweisregeln angewandt, wie wir sie im NED Beweis gesehen haben.

Isabelle, ein weiterer interaktiver Beweisassistent für höherstufige Logik, ist ähnlich wie das PVS-System aufgebaut. Isabelle verwendet Gentzens Kalkül des natürlichen Schließens und ist auch in Theorien strukturiert. Genau wie in PVS steht dem Benutzer eine große Theoremdatenbank zur Verfügung. Desweiteren sind auch in Isabelle automatische Beweis- und Optimierungsalgorithmen implementiert, die in NED nicht zur Verfügung stehen.
Aber auch Gleichheitsbeweise lassen sich in Isabelle führen, wie das folgende Beweisbeispiel zeigt.

Gleichheitsbeweis in Isabelle

```
theory Equality = Main:

consts
OR          :: ''bool ⇒ bool ⇒ bool''
AND         :: ''bool ⇒ bool ⇒ bool''
Q           :: ''(bool ⇒ bool) ⇒ bool''
```

Zunächst müssen wir einen Namen (*Equality*) für unsere Theorie angeben und Isabelle mitteilen, dass wir die eingebauten Haupttheorien erweitern möchten. Diese sind unter anderem HOL, die natürlichen Zahlen oder die Mengentheorie. Anschließend deklarieren wir die logischen Konstanten *OR, AND* und einen beliebigen Quantor *Q*. Es folgt

```
axioms
AND_def     : ''AND b False = False''
OR_def      : ''OR False b = b''
```

```
Q1          : ''Q f = AND (Q f) (f x)''
Q2          : ''OR (Q f) b = Q(λ x. OR (f x) b)''

lemma Quantor:
''Q (λ x. b) = b''
```

die Deklaration einiger logischer Axiome und die Spezifikation des zu zeigenden Lemmas. Es folgen die Beweiskommandos, mit denen sich das Lemma zeigen lässt:

```
apply (rule_tac P=''(λ a. Q(λx. a) = b'' and t=''b''
      and s=''OR False b'' in ssubst)
                (* Erstes Sublemma s=t *)
apply (subst OR_def)
apply simp

apply (rule_tac P=''λ a. a = b'' and t=''Q(λ x. OR False b)''
      and s=''OR (Q(λ x. False)) b'' in ssubst)
                (* Zweites Sublemma s=t *)
apply (subst Q2)
apply (simp)

apply (subst Q1)
apply (subst AND_def)
apply (subst OR_def)
by simp
end
```

Der Beweis besteht im Wesentlichen aus der Regel *subst* und den Taktiken *simp* und *ssubst*. Das Kommando *subst* implementiert eine Ersetzungsregel, die das *erste* Auftreten eines Subterms ersetzt. Die Taktik *ssubst* hingegen erlaubt es, eine Positionsangabe für den zu ersetzenden Subterm über einen speziellen Lambdaausdruck zu machen. Im Rahmen der Taktik *simp* wird unter anderem die Reflexivitätsregel angewandt.

Es fallen einige kleine Unterschiede zu unserem System NED auf:

- In Isabelle ist es nicht notwendig, freie Variablen (auch *schematische* Variablen in Isabelle genannt) zu deklarieren. Treten innerhalb einer Lemma-Spezifikation unbekannte und nicht gebundene Variablen auf, so werden sie automatisch mit dem passenden Typ als schematische Variable klassifiziert.

- Möchte man Substitutionen direkt mit Hilfe von Gleichheitsaxiomen (N=N') durchführen, so gelingt dies in Isabelle nur mit der Regel *subst*, die das erste Auftreten von N in der aktuellen Formel durch N' ersetzt. Um die entgegengesetzte Ersetzungsrichtung zu erhalten, müssen wir die Taktik

ssubst verwenden und die Position, sowie die Terme s und t explizit angeben. Es wird ein *Subgoal* (t=s) erzeugt, dass nun mit *subst* und *simp* gezeigt werden kann.

Bei der Entwicklung unseres Beweisassistenten NED haben wir uns stark an einem weiteren System, dem französischen Beweisassistenten **PhoX** [7], in Sachen Funktionalität, Benutzerfreundlichkeit und Erscheinungsbild orientiert. PhoX wird an der Université de Savoie eingesetzt, um Studenten der Mathematik ein Werkzeug zur Verfügung zu stellen, mit dem sie formale mathematische Beweise führen können.

Fassen wir die Beobachtungen zusammen. Wollen wir einfache Beweise führen, so genügt unser System NED. Dadurch, dass NED nur wenige Beweiskommandos zur Verfügung stellt, erleichtert dies den Umgang mit dem System und liefert dem Benutzer eine gewisse Übersichtlichkeit.

Der Automatisierungsgrad in PVS oder Isabelle ist essentiell, wenn es darum geht, kompliziertere Beweise zu führen, und dem Benutzer die trivialen Teilbeweise abzunehmen. Didaktisch gesehen, macht diese Art, Beweise zu führen, wenig Sinn. Wenn man als Benutzer einen Lerneffekt beim Benutzen des Systems erzielen möchte, dann kann dies nur erreicht werden, wenn man Beweise auf Kalkül-Ebene führt. Der Benutzer bekommt ein Gefühl für die zugrunde liegende Logik und lernt sehr schnell, worauf zum Beispiel die Induktion über die natürlichen Zahlen basiert.

Kapitel 7

Fazit

Im theoretischen Teil dieser Arbeit haben wir in Anlehnung an das Skript zur Vorlesung ICL [1] ausgehend von den logischen Axiomen und den Gleichheitsregeln das gesamte Formeldeduktionssystem des Sequenten-Kalküls abgeleitet. Gentzen schlug in seinen Untersuchungen über das natürliche Schließen eine andere Richtung ein. Er überlegte sich zunächst, welche die wahren Schlussweisen der Mathematik seien, hielt diese in den Regeln seines natürlichen Schlusssystems fest und zeigte dann, dass sein System äquivalent zu dem Hilbertschen Formalismus ist.[3]

Im praktischen Teil haben wir mit NED einen einfachen Beweisassistenten für HOL implementiert, der es erlaubt, mathematische Beweise sowie Gleichheitsbeweise auf formaler Ebene zu führen, die der Benutzer verstehen kann. Damit bekommt der Student, der dieses Werkzeug begleitend zur Vorlesung ICL einsetzt, neben der Theorie auch die Möglichkeit, das Gelernte in der Praxis einzusetzen und damit zu festigen.

Möchte man größere Projekte verifizieren oder komplexere Beweise führen, so sprengt dies den Rahmen von NED. In diesem Fall, sollte man auf leistungsfähigere Beweissysteme wie PVS, Isabelle oder ähnliche Systeme zurückgreifen.

Dennoch könnte NED um verschiedene Konstrukte erweitert werden. Eine mögliche Erweiterung könnte es sein, den Benutzer eigene Taktiken definieren zu lassen, Sequenzen von Kommandos, die oft benötigt werden. Verwendet man Term-Indexing Techniken, so könnte man sich ebenfalls kleine Theoremdatenbanken vorstellen, in denen der Benutzer seine Beweise abspeichern und wiederverwenden kann.

Auf dem Gebiet der mathematischen Assistenzsysteme wird sehr viel geforscht. Hard-und Softwareverifikation kann mittlerweile sehr erfolgreich damit praktiziert werden. Am Ende steht die Vision, dass auch Mathematiker diese Systeme benutzen, und dadurch eine Arbeitserleichterung erfahren.

Anhang A

Tutorial

A.1 Einführung in NED

Aufgabe dieses Tutorials ist es, dem Benutzer einen leichten Einstieg in die Beweisumgebung NED zu ermöglichen. Um dies zu erreichen, werden wir zunächst einen kleinen Überblick über die verwendete Software geben. Es folgt eine Einführung in die Oberfläche von Proof General. Um mit dem System arbeiten zu können, muss der Benutzer anschließend mit der konkreten Syntax von NED vertraut gemacht werden. Zuletzt werden wir in A.2.2 zwei Beispielbeweise führen, an denen schrittweise das System erklärt wird.

Informationen zur Software
Die Software *NED* ist im Web verfügbar [12]. Darüberhinaus wird empfohlen, mit dem Tool *Proof General* zu arbeiten [11]. Hierbei handelt es sich um eine Erweiterung des Texteditors (X)Emacs. Das Tool stellt ein Interface zwischen dem Editor und dem Beweisassistenten NED zur Verfügung und bietet neben der Möglichkeit durch ein Beweisskript zu navigieren unter anderem Unterstützung des *X-Symbol package*. (Der Benutzer sieht beispielsweise unmittelbar A $\vee$ B statt A + B im Editor.)

Nähere Informationen zur Installation der beiden Programme befinden sich auf den jeweiligen Homepages. Dieses Tutorial arbeitet mit dem Tool Proof General.

Oberfläche von Proof General
Betrachten wir zunächst Abbildung A.1 und beginnen im oberen Teilfenster. Es handelt sich hierbei um das *Script-Window*. Hier können im Vorfeld eines Beweises Typ- und Konstantendeklarationen vorgenommen werden. Desweiteren können Lemmata, Axiome oder Definitionen spezifiziert werden. Sind alle erforderlichen Deklarationen abgeschlossen, so wird das *Maingoal* eingegeben. Mithilfe weniger Kommandos kann nun im Anschluss das Goal bewiesen werden.

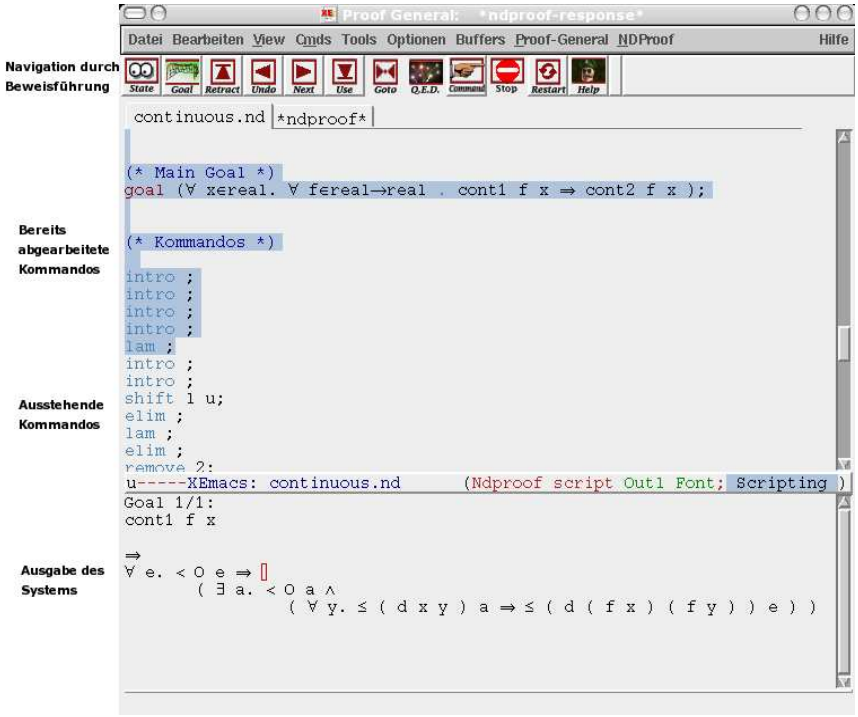

Abbildung A.1: NED Umgebung

Bei dem unteren Teilfenster handelt es sich um das *Response-Window*. Für jede Eingabe innerhalb des Script-Windows bekommt der Benutzer im Response-Window eine entsprechende Rückmeldung vom System. Auch Fehlermeldungen werden hier ausgegeben.

Die Navigatorleiste im oberen Bereich dient zur Navigation durch das Script-Window:

Use	überträgt das gesamte Skript zu NED und führt es aus
Next	übermittelt nur die nächste Zeile im Beweisscript
Goto	führt alle Deklarationen und Kommandos bis zu der Zeile im Skript aus, an der sich der Cursor befindet
Restart	startet den Beweis Assistenten NED neu
Stop	unterbricht die laufenden Berechnungen

Konkrete Syntax

Toplevel-Kommandos bzw. Kommandos der deklarativen Phase:

type *name*	Deklariert einen neuen Typ *name*
const *name* : *typ*	Deklariert neue Konstante *name* vom Typ *typ*
var *name* : *typ*	Deklariert neue Variable *name* vom Typ *typ*
def *name* := *formel*	Deklariert neue Definition *name* als Abkürzung für *formel*
lemma *name* := *formel*	Deklariert neues Lemma *name* (muss bei Verwendung bewiesen werden)
axiom *name* := *formel*	Deklariert neues Axiom *name* (muss bei Verwendung nicht bewiesen werden)
save *name*	Speichert den korrekt geführten Beweis unter dem Lemma *name*
goal *formel*	Startet den Beweis von *formel*
reset	Setzt das System zurück (vgl. Navigatorleiste)
undo	Macht den letzten Beweisschritt rückgängig (vgl. Navigatorleiste)
print	Gibt die aktuellen Goals / Subgoals im Response-Window aus
latex *name*	Exportiert den Beweis in ein LaTeX- File
exit	Beendet den Beweisassistenten NED

Gleichheitskommandos:

siehe Tabelle 4.3 in Abschnitt 4.1

ND-Beweis-Kommandos:

siehe Tabelle 4.4 in Abschnitt 4.1

Formel- und Typeingabe:

Formel	Eingabe
$\top$	$TRUE$
$\bot$	$FALSE$
$\neg A$	$\sim A$
$A \vee B$	$A + B$
$A \wedge B$	$A\&B$
$A \Rightarrow B$	$A => B$
$A \Leftrightarrow B$	$A <=> B$
$A \doteq B$	$A = B$
$\exists x \in t.x$	$/ \backslash\ x : t.x$
$\forall x \in t.x$	$\backslash / x : t.x$
$\lambda x \in t.x$	$\backslash x : t.x$

Typ	Eingabe
$\mathbb{B}$	bool
$t_1 \rightarrow t_2$	t1 -> t2

Anmerkung
Wir können auf die Typangabe einer gebundenen Variablen verzichten, wenn wir zuvor eine gleichnamige freie Variable mit entsprechendem Typ deklariert haben. Der Typ der gebundenen entspricht dann dem der gleichnamigen freien Variable.

A.2 Beispielbeweise

A.2.1 Beweis im Rewriting-Modus

In diesem Abschnitt wollen wir einen typischen Gleichheitsbeweis führen. Wir arbeiten nur mit den Gleichheitsregeln, die in Kapitel 3 aufgeführt sind.

Zunächst zeigen wir, wie man die Subterm Position eines Terms korrekt bestimmt. Hierfür gelte:

$B = \{\vee, \wedge, =, \Rightarrow, \Leftrightarrow\}$ - Die Menge der binären Operatoren
$U = \{\neg, \forall, \exists, \lambda\}$ - Die Menge der unären Operatoren

Folgende Pseudo-ML Funktion berechnet die gewünschte Position:

```
Position' ( For as B(n,m) ) p    =
       if (For desired subterm) then p else
       if (desired subterm contained in n) then Position' n (p,1)
            else Position' m (p,2)
Position' ( For as U(n) ) p      =
      if (For desired subterm) then p else Position' n (p,1)

Position TERM                    =  Position' TERM ( () )
```

Beispiel: Betrachte $\top \Rightarrow \forall x \in \mathbb{B}.\exists y \in \mathbb{B}.y \vee \underline{\neg(x \wedge \neg x)}$

Die unterstrichene Teilformel $\neg(x \wedge \neg x)$ hat die Position **(2,1,1,2)**. Das Java Tool *Javapos*(Kommando **pos**) unterstützt den Benutzer bei der Subtermbestimmung, indem es einen Baum des aktuellen Terms ausgibt. Der Benutzer muss den entsprechenden Subtermknoten im Baum finden, darauf klicken und bekommt die Subtermposition berechnet.

Alternativ kann man sich Labels in der Zeile unterhalb der Formel ausgeben lassen (Kommando **labels**). In unserem Beispiel würde das folgendermaßen aussehen:

$$\begin{array}{llllllllllll} \top & \Rightarrow & \forall x. & \exists y. & y & \vee & \neg & (x & \wedge & \neg & x) \\ B & A & C & D & F & E & G & I & H & J & K \end{array}$$

Der unterstrichene Subterm könnte dann ebenfalls mit **(E,2)** ausgewählt werden.

Gleichheitsbeweis
Beginnen wir mit der deklarativen Phase:

type T;

Standardmässig steht nur der Basistyp Bool zur Verfügung.
Da wir aber Aussagen über einen allgemeinen Typ treffen wollen,
deklarieren wir einen weiteren Basistyp T.

const + : T→T→T;
const * : T→T→T;
const O : T;

Die Signatur wird um die Konstanten *+,*,O* erweitert.

var x : T;
var y : T;
var z : T;

Die Variablenumgebung wird um die freien Variablen *x,y,z* erweitert.

axiom one := * x x = x;
axiom two := * x y = * y x;
axiom three := * x (f x) = O;
axiom four := * x O = x;
axiom five := * x (* y z) = * (* x y) z;
axiom six := + x (* y z) = * (+ x y) (+ x z);

Wir geben mit der obigen Deklaration an, dass die sechs Axiome gültig sind.

Nun wollen wir die folgenden zwei Behauptungen mit Hilfe unserer Gleichheitsregeln beweisen:

goal * x O = O; *(i)*
goal * x (+ x y) = x; *(ii)*

Beginnen wir mit *(i)*
Goal 1/1:
* x O = O

rew (1,2) with three r;

Goal 1/1:
* x (* x (f x)) = O

An Subterm Position (1,2) wird **O** unter Verwendung der Ersetzungsregel durch *** x (f x)** ersetzt.

rew (1) with five l;

Goal 1/1:
* (* x x) (f x) = O

Es wird automatisch in Axiom five für **z (f x)** eingesetzt (Einsetzungsregel) Danach wird der Subterm an Position (1) *** x (* x (f x))** unter Verwendung der Ersetzungsregel durch *** (* x x) (f x)** ersetzt.

Es folgen:
rew (1,1,2) with one l;
rew (1) with three l;

Goal 1/1:
O = O
ref;

Wir wenden die Reflexivitätsregel an und sind fertig mit dem Beweis.

PROOF DONE
save i;

save i speichert die gerade bewiesene Behauptung als Lemma mit der Bezeichung i. Für die nächste Behauptung verwenden wir i, um den Beweis abzukürzen.

Fahren wir fort mit *(ii)*

```
Goal 1/1:
  * x (+ x y) = x
```

rew (1,1,2) with four r;
rew (1) with six r;
rew (1,2) with two l;

```
Goal 1/1:
  + x (* y O) = x
```

Wir verwenden jetzt Lemma (i), was wir zuvor bewiesen haben, um den Beweis an dieser Stelle abzukürzen.

rew (1,2) with i l;

```
Goal 1/1:
  + x O = x
```

> In Lemma i wir automatisch für **x y** eingesetzt (Einsetzungsregel)
> Danach wird der Subterm an Position (1,2) **(* y O)** unter Verwendung der Ersetzungsregel durch **O** ersetzt

rew (1) with four l;

```
Goal 1/1:
  x = x
```
ref;
```
PROOF DONE
```
schließt den Beweis.

A.2.2 Beweis im ND-Modus

In diesem Abschnitt wollen wir in NED einen typischen mathematischen Beweis führen. Wir zeigen, dass zwei Definitionen für die Stetigkeit einer Funktion äquivalent sind.

Beginnen wir auch hier mit der *deklarativen Phase*:

type real;

Standardmässig steht nur der Basistyp Bool zur Verfügung.
Da hier eine Aussage über reele Zahlen getroffen wird,
muss der Typ *real* hinzugefügt werden.

const < : real→real→bool;
const ≤ : real→real→bool;
const d : real→real→real;
const O : real;

Die Signatur wird um die Konstanten *less, lesseq*
d(istance) und *O* erweitert.

axiom lemme1 := ∀ x∈real. ∀ y∈real. (< x y ⇒ ≤ x y);
axiom lemme2 := ∀ x. < O x ⇒ (∃ y. < O y ∧ (∀ z. ≤ z y ⇒ < z x));

Im Verlauf des Beweises werden zwei Lemmata benötigt, die
deklariert werden müssen.

def cont1 :=
λ f ∈ real → real.
λ x ∈ real.
∀ e ∈ real. < O e ⇒
(∃ a ∈ real. < O a ∧
(∀ y ∈ real. < (d x y) a ⇒ < (d (f x) (f y)) e));
def cont2 :=
λ f ∈ real → real.
λ x ∈ real.
∀ e ∈ real. < O e ⇒
(∃ a ∈ real. < O a ∧
(∀ y ∈ real. ≤ (d x y) a ⇒ ≤ (d (f x) (f y)) e));

Zwei Definitionen für die Stetigkeit einer Funktion an
einem Punkt x.

goal (∀ x∈real. ∀ f∈real→real . cont1 f x ⇒ cont2 f x);
Goal 1/1:
⊤
⟹
∀ x. ∀ f. cont1 f x ⇒ cont2 f x

Spezifikation des Maingoal. Wir wollen zeigen, dass für alle Funktionen f
und alle Werte x beide Definitionen äquivalent sind. Wir zeigen hier nur
die eine Richtung.

Es folgt die *Beweisphase*:

intro;
Goal 1/1:
$\top$
$\Longrightarrow$
$\forall$ f. cont1 f x $\Rightarrow$ cont2 f x

Es wurde der Allquantor eliminiert und der Parameter x
als neue freie Variable hinzugefügt. (vgl. $\forall$I)

intro;
Goal 1/1:
$\top$
$\Longrightarrow$
cont1 f x $\Rightarrow$ cont2 f x

Es wurde der Allquantor eliminiert und der Parameter f
als neue freie Variable hinzugefügt. (vgl. $\forall$I)

intro;
Goal 1/1:
cont1 f x
$\Longrightarrow$
cont2 f x

(cont1 f x) wurde gemäß Regel ($\Rightarrow$I) der Prämissenliste hinzugefügt

intro;
lam;
Goal 1/1:
cont1 f x
$\Longrightarrow$
$\forall$ e. $<$ O e $\Rightarrow$
($\exists$ a. $<$ O a $\wedge$
($\forall$ y. $\leq$ (d x y) a $\Rightarrow$ $\leq$ (d (f x) (f y)) e))

Das Kommando *intro* bewirkt, dass *cont2* durch seine Definition
substituiert wird. Mit dem Kommando *lam* wird die
β/η-Normalform berechnet.

intro; Erneute Allquantoreliminierung
intro;
Goal 1/1:

(1) $< O\ e$
(2) cont1 f x
$\Longrightarrow$
$\exists\ a.\ < O\ a \wedge (\forall\ y.\ \leq (d\ x\ y)\ a \Rightarrow \leq (d\ (f\ x)\ (f\ y))\ e)$

(< O e) wurde gemäß Regel (⇒I) der Prämissenliste hinzugefügt.

shift l u;
Goal 1/1:
(1) cont1 f x
(2) $< O\ e$
$\Longrightarrow$
$\exists\ a.\ < O\ a \wedge (\forall\ y.\ \leq (d\ x\ y)\ a \Rightarrow \leq (d\ (f\ x)\ (f\ y))\ e)$

Die Prämissen wurden um eine Position zyklisch nach oben geshiftet.

elim;
lam;
Goal 1/1:
(1) $\forall\ e0.\ < O\ e0 \Rightarrow$
$(\ \exists\ a0.\ < O\ a0 \wedge$
$(\ \forall\ y2.\ < (\ d\ x\ y2\ )\ a0 \Rightarrow$
$< (\ d\ (\ f\ x\ )\ (\ f\ y2\ )\ )\ e0\ )\ )$
(2) $< O\ e$
$\Longrightarrow$
$\exists\ a.\ < O\ a \wedge (\forall\ y.\ \leq (d\ x\ y)\ a \Rightarrow \leq (d\ (f\ x)\ (f\ y))\ e)$

Das Kommando *elim* bewirkt, dass *cont1* durch seine Definition substituiert wird. Mit dem Kommando *lam* wird die β/η-Normalform berechnet.

elim;
Goal 1/1:
(1) $< O\ \Gamma 0 \Rightarrow$
$(\exists\ a0.\ < O\ a0 \wedge$
$(\forall\ y2.\ < (d\ x\ y2)\ a0 \Rightarrow < (d\ (f\ x)\ (f\ y2))\ \Gamma 0))$
(2) $< O\ e$
$\Longrightarrow$
$\exists\ a.\ < O\ a \wedge (\forall\ y.\ \leq (d\ x\ y)\ a \Rightarrow \leq (d\ (f\ x)\ (f\ y))\ e)$

Der Allquantor der ersten Prämisse wird eliminiert. Es wird eine Gamma Variable Γ0 generiert, welche durch einen beliebigen Term ersetzt werden kann. Möchte man die Allquantorformel erhalten, um später weitere Gamma Variablen einzuführen, so benutzt man das Kommando **elim more**. (vgl. ∃E)

instance 0 = e;
Goal 1/1:
(1) $< \mathsf{O}\ \mathsf{e} \Rightarrow$
$(\exists\ \mathsf{a0}. < \mathsf{O}\ \mathsf{a0} \wedge$
$(\forall\ \mathsf{y2}. < (\mathsf{d}\ \mathsf{x}\ \mathsf{y2})\ \mathsf{a0} \Rightarrow < (\mathsf{d}\ (\mathsf{f}\ \mathsf{x})\ (\mathsf{f}\ \mathsf{y2}))\ \mathsf{e}))$
(2) $< \mathsf{O}\ \mathsf{e}$
$\Longrightarrow$
$\exists\ \mathsf{a}. < \mathsf{O}\ \mathsf{a} \wedge (\forall\ \mathsf{y}. \leq (\mathsf{d}\ \mathsf{x}\ \mathsf{y})\ \mathsf{a} \Rightarrow \leq (\mathsf{d}\ (\mathsf{f}\ \mathsf{x})\ (\mathsf{f}\ \mathsf{y}))\ \mathsf{e})$

Die Gammavariable $\Gamma 0$ wird mit dem Term e instanziiert

apply 2 1;
remove 2; Prämisse Nr. 2 wird entfernt
Goal 1/1:
(1)($\exists\ \mathsf{a0}. < \mathsf{O}\ \mathsf{a0} \wedge$
$(\forall\ \mathsf{y2}. < (\mathsf{d}\ \mathsf{x}\ \mathsf{y2})\ \mathsf{a0} \Rightarrow < (\mathsf{d}\ (\mathsf{f}\ \mathsf{x})\ (\mathsf{f}\ \mathsf{y2}))\ \mathsf{e}))$
(2) $< \mathsf{O}\ \mathsf{e}$
$\Longrightarrow$
$\exists\ \mathsf{a}. < \mathsf{O}\ \mathsf{a} \wedge (\forall\ \mathsf{y}. \leq (\mathsf{d}\ \mathsf{x}\ \mathsf{y})\ \mathsf{a} \Rightarrow \leq (\mathsf{d}\ (\mathsf{f}\ \mathsf{x})\ (\mathsf{f}\ \mathsf{y}))\ \mathsf{e})$

Die Regel Modus Ponens ($\Rightarrow$E) wurde angewandt. Die ursprüngliche Prämisse bleibt dabei erhalten, wurde jedoch durch *remove 2* entfernt.

elim;
elim;
Goal 1/1:
(1) $\forall\ \mathsf{y2}. < (\mathsf{d}\ \mathsf{x}\ \mathsf{y2})\ \mathsf{a0} \Rightarrow < (\mathsf{d}\ (\mathsf{f}\ \mathsf{x})\ (\mathsf{f}\ \mathsf{y2}))\ \mathsf{e})$
(2) $< \mathsf{O}\ \mathsf{a0}$
(3) $< \mathsf{O}\ \mathsf{e}$
$\Longrightarrow$
$\exists\ \mathsf{a}. < \mathsf{O}\ \mathsf{a} \wedge (\forall\ \mathsf{y}. \leq (\mathsf{d}\ \mathsf{x}\ \mathsf{y})\ \mathsf{a} \Rightarrow \leq (\mathsf{d}\ (\mathsf{f}\ \mathsf{x})\ (\mathsf{f}\ \mathsf{y}))\ \mathsf{e}\ )$

Das erste *elim* Kommando realisiert ($\exists$E).
Das zweite glättet die Prämissenliste.

apply lemme2;
Goal 1/1:
(1) $\forall\ \mathsf{x1}. < \mathsf{O}\ \mathsf{x1} \Rightarrow (\exists\ \mathsf{y1}. < \mathsf{O}\ \mathsf{y1} \wedge (\forall\ \mathsf{z}. \leq \mathsf{z}\ \mathsf{y1} \Rightarrow < \mathsf{z}\ \mathsf{x1}))$
(2) $\forall\ \mathsf{y2}. < (\mathsf{d}\ \mathsf{x}\ \mathsf{y2})\ \mathsf{a0} \Rightarrow < (\mathsf{d}\ (\mathsf{f}\ \mathsf{x})\ (\mathsf{f}\ \mathsf{y2}))\ \mathsf{e})$
(3) $< \mathsf{O}\ \mathsf{a0}$
(4) $< \mathsf{O}\ \mathsf{e}$
$\Longrightarrow$
$\exists\ \mathsf{a}. < \mathsf{O}\ \mathsf{a} \wedge (\forall\ \mathsf{y}. \leq (\mathsf{d}\ \mathsf{x}\ \mathsf{y})\ \mathsf{a} \Rightarrow \leq (\mathsf{d}\ (\mathsf{f}\ \mathsf{x})\ (\mathsf{f}\ \mathsf{y}))\ \mathsf{e}\ )$

Das Lemma *lemme2* wird der Prämissenliste hinzugefügt. Da es als Axiom und nicht als Lemma deklariert wurde, muss es im Anschluss an das Maingoal nicht bewiesen werden.

elim ;
instance 1 = a0;
apply 3 1;
remove 2;
elim ;
elim ;
intro ;
instance 2 = y1;
Goal 1/1:
...
$\Longrightarrow$
$< \mathsf{O}\ \mathsf{y1} \wedge (\forall\ \mathsf{y}.\ \leq (\mathsf{d}\ \mathsf{x}\ \mathsf{y})\ \mathsf{y1} \Rightarrow \leq (\mathsf{d}\ (\mathsf{f}\ \mathsf{x})\ (\mathsf{f}\ \mathsf{y}))\ \mathsf{e})$

intro;
Goal 1/2:
(2) $< \mathsf{O}\ \mathsf{y1}$
...
$\Longrightarrow$
$< \mathsf{O}\ \mathsf{y1}$

Goal 2/2:
...
$\Longrightarrow$
$\forall\ \mathsf{y}.\ \leq (\mathsf{d}\ \mathsf{x}\ \mathsf{y})\ \mathsf{y1} \Rightarrow \leq (\mathsf{d}\ (\mathsf{f}\ \mathsf{x})\ (\mathsf{f}\ \mathsf{y}))\ \mathsf{e}$

Das Goal wird in zwei Subgoals aufgesplittet (vgl. ($\wedge$I)).
Beide haben dieselben Prämissen. Alle Kommandos beziehen sich immer auf das erste Subgoal.

simplify;
Goal 1/1:
(1) $\forall$ z. $\leq$ z y1 $\Rightarrow$ $<$ z a0
(2) $<$ O y1
(3) $\forall$ y3. $<$ (d x y3) a0 $\Rightarrow$ $<$ (d (f x) (f y3)) e
(4) $<$ O a0
(5) $<$ O e
$\Longrightarrow$
$\forall$ y. $\leq$ (d x y) y1 $\Rightarrow$ $\leq$ (d (f x) (f y)) e

Das erste Subgoal wird durch die Regel Bool abgeschlossen. ($<$ O y1) erscheint sowohl in der Prämisse als auch in der Konklusion.

intro ;
intro ;
shift l d 3 ;
elim ;
instance 3 = y ;
shift l d 2 ;
elim ;
instance 4 = d x y ;
apply 6 1 ;
remove 2 ;
apply 1 3 ;
remove 4 ;
apply lemme1 ;
elim ;
instance 5 = d (f x) (f y) ;
elim ;
instance 6 = e ;
apply 2 1 ;

Goal 1/1:
(1) $\leq$ (d (f x) (f y)) e
...
$\Longrightarrow$
$\leq$ (d (f x) (f y)) e
simplify ;
PROOF DONE

Anhang B

Konstruktion von Deduktionsregeln

Wir geben in diesem Abschnitt ein kurzes Beweisskript an, in dem wir zwei eigene Schlussregeln für Formeln konstruiert haben. Diese beweisen wir mit Hilfe unserer Gleichheitsregeln und speichern sie als Lemmata ab. Das Skript findet man unter *\NED\proofs\NewND.ned*

```
(* NewND.ned *)
type X

var x:bool
var y:bool
var z:bool

var f:X→bool
var g:X→bool
var x':X

axiom Equal := x = (x = ⊤)


(* Boolsche Axiome *)


axiom Komm1 := x ∧ y = y ∧ x
axiom Komm2 := x ∨ y = y ∨ x

axiom Asso1 := (x ∧ y) ∧ z = x ∧ ( y ∧ z )
axiom Asso2 := (x ∨ y) ∨ z = x ∨ ( y ∨ z )

axiom Idem1 := x ∧ x = x
axiom Idem2 := x ∨ x = x

axiom Komp1 := x ∧ ¬x = ⊥
axiom Komp2 := x ∨ ¬x = ⊤

axiom Iden1 := ⊤ ∧ x = x
axiom Iden2 := ⊥ ∨ x = x
```

```
axiom Domi1 := ⊥ ∧ x = ⊥
axiom Domi2 := ⊤ ∨ x = ⊤

axiom Dist1 := x ∧ (y ∨ z) = x ∧ y ∨ x ∧ z
axiom Dist2 := x ∨ y ∧ z = (x ∨ y) ∧ (x ∨ z)

axiom Abso1 := x ∨ x ∧ z = x
axiom Abso2 := x ∧ (x ∨ z) = x

axiom DeMo1 := ¬(x ∧ y)= ¬x ∨ ¬y
axiom DeMo2 := ¬(x ∨ y)= ¬x ∧ ¬y

axiom Reso1 := (x ∧ y) ∨ (¬x ∧ z) = (x ∧ y) ∨ (¬x ∧ z) ∨ (y ∧ z)
axiom Reso2 := (x ∨ y) ∧ (¬x ∨ z) = (x ∨ y) ∧ (¬x ∨ z) ∧ (y ∨ z)

axiom Nega0 := ¬ ⊥ = ⊤
axiom Nega1 := ¬ ⊤ = ⊥

axiom Dneg := ¬¬x = x
axiom defImp := x ⇒ y = ¬x ∨ y


(* Quantoraxiome *)


axiom allI := ( ∀ a:X. f a) = (∀ a:X. f a) ∧ f x'
axiom exiI := ( ∃ a:X. f a) = (∃ a:X. f a) ∨ f x'

axiom allE := ( ∀ a:X. z) = z
axiom exiE := ( ∃ a:X. z) = z

axiom DeM1 := ¬(∀ a:X. f a) = ( ∃ a:X. ¬ (f a))
axiom DeM2 := ¬(∃ a:X. f a) = ( ∀ a:X. ¬ (f a))

axiom Allor := (∀ a:X. f a ∨ z) = (∀ a:X. f a) ∨ z
axiom Alland := (∀ a:X. f a ∧ z) = (∀ a:X. f a) ∧ z
axiom Alland' := (∀ a:X. f a ∧ g a) = (∀ a:X. f a) ∧ ( ∀a:X. g a)

axiom Exand := (∃ a:X. f a ∧ z) = (∃ a:X. f a) ∧ z
axiom Exor := (∃ a:X. f a ∨ z) = (∃ a:X. f a) ∨ z
axiom Exor' := (∃ a:X. f a ∨ g a) = (∃ a:X. f a) ∧ ( ∃ a:X. g a)


var A:bool
var C:bool
const O:X


(*** Konstruieren uns nun eine eigene ∀I Regel ***)


goal ( A ⇒ f x' ) ⇒ ( A ⇒ (∀a:X. f a) )


(* Benoetigen die Implikations-Introduction-Regel im ersten Schritt, um die
Praemisse später substituieren zu koennen! *)


intro
rew (1) with Equal l
```

```
rew (2) with defImp l
rew (2) with Komm2 l
rew (2) with Allor r
rew (2,1) with Komm2 l
rew (2,1) with defImp r
rew (2,1) with 1 l
rew (2) with allE l
rew (2) with Equal l
ref

(* Wir speichern die neue Deduktionsregel unter <allright> *)
save allright

(* Wir beweisen nun mit Hilfe der Deduktionsregel
<allright> die triviale Aussage (∀ k. ⊤) *)

goal ( ∀ k:X. (λ n:X. ⊤ ) k )

(* Wende nun die neue Inferenzregel an *)
(* (0) als Spezialfall-Position für die gesamte Formel *)
(* inklusive TRUE => ... *)
rew (0) with allright r
lam
rew with Equal l
ref

(*** Konstruieren uns nun eine eigene ∀E Regel ***)

goal (A ∧ f x' ⇒ C) ⇒ ( A ∧ (∀a:X. f a) ⇒ C )

(* Ebenfalls Implikations-Introduction zuerst *)
intro
rew (1) with Equal l
rew (2) with defImp l
rew (2,1) with DeMo1 l
rew (2,1,2,1) with allI l
rew (2,1,2) with DeMo1 l
rew (2,1,2) with Komm2 l
rew (2,1) with Asso2 r
rew (2,1) with Komm2 l
rew (2) with Asso2 l
rew (2,2,1) with DeMo1 r
rew (2,2) with defImp r
rew (2) with Komm2 l
rew (2,1) with 1 l
rew (2,1) with Equal l
```

```
ref
save allleft

(* Beispielbeweis
* Wollen zeigen : ¬ (∀n:X. ¬(n=0))
(Es gibt nämlich ein n=0, dass die Bedingung verletzt)
* Zeigen dies durch Widerspruch : (∀n:X. ¬(n=0)) ⇒ ⊥ *)

(* Legen mit Axiom instanz unsere spaetere Instanziierung fest *)
axiom instanz := x' = 0
goal ⊤ ∧ (∀ n:X. (λ n':X. ¬(n' = 0)) n ) ⇒ ⊥

rew with allleft r
rew (1,2,2) with instanz l
lam
rew (1) with Iden1 l
rew with defImp l
rew (1) with Dneg l
rew with Komm2 l
rew with Iden2 l
ref

exit
```

Literaturverzeichnis

[1] G. Smolka. Skript zur Vorlesung Introduction to computational logic (ICL) SS 2004, http://www.ps.uni-sb.de/courses/cl-ss04/script/index.html

[2] K.H. Bläsius und H.J. Bürckert. Deduction Systems, Oldenburg-Verlag, 1989

[3] G.Gentzen. Untersuchungen über das Logische Schließen. *Mathematische Zeitschrift*, 39:176-210, 405-431, 1935

[4] C.P. Wirth. Descente Infinie + Deduction, *Logic Journal of the IGPL*, 12:1-96, 2004

[5] C. Benzmüller und M.Kohlhase. Extensional Higher-Order Resolution. *Proceedings of the 15th International Conference on Automated Deduction (CADE)*, LNAI vol. 1421, pp. 56-71, Lindau, Germany, 1998. Springer

[6] A. A. Adams und L. A. Dennis. Rippling in PVS. *Proceedings of Design and Application of Strategies/Tactics in Higher Order Logics*, pp. 84 - 91, 2003

[7] R. David und C.Raffalli. An experiment concerning mathematical proofs on computers with French undergraduate students, *Journal of applied logic*, 2(2):219-239, 2004

[8] Mark Kaminski. Studies in Higher-Order Equational Logic. Bachelorarbeit, *http://www.ps.uni-sb.de/∼kaminski/bthesis/kaminski-bthesis.pdf*, 2005

[9] T. Nipkow, L.C.Paulson und M. Wenzel. Isabelle/HOL - A proof Assistant for Higher-Order Logic, Springer-Verlag, 2002. Lecture Notes in Computer Science No. 2283

[10] N. Shankar, S. Owre, J. M. Rushby und D. W. J. Stringer-Calvert. PVS Prover Guide, *http://pvs.csl.sri.com/doc/pvs-prover-guide.pdf*, 2001

[11] D. Aspinall with T.Kleymann. Adapting Proof General, *http://proofgeneral.inf.ed.ac.uk/*, 2004

[12] C. Hümbert. NED - Ein interaktiver Beweisassistent für höherstufige Logik, *http://www.ps.uni-sb.de/∼huembert/bt/download/downloads.htm*, 2005