Untersuchung und Implementation der Reduktionskalküle RPC_n

Studienarbeit

Carsten Sinz
Kapellenweg 35
72070 Tübingen

20. Juni 1997

Inhaltsverzeichnis

1 Einführung

Im automatischen Beweisen haben unterschiedliche Verfahren wie Resolution, Sequenzen- oder Modus-Ponens-Kalküle große Bekanntheit und zum Teil beachtliche Erfolge erzielt. Ein grundsätzliches Problem ist allerdings bei allen automatischen Beweistechniken zu erkennen: die Größe des Suchraumes nimmt sehr viel schneller zu als die Größe der untersuchten Formeln. Dadurch ist es kaum möglich, realistische Beweise, wie sie etwa von Mathematikern betrachtet werden, automatisch führen zu lassen. Selbst in der Aussagenlogik ist dieses Phänomen bekannt, in der Prädikatenlogik erster Stufe verschärfen sich die Probleme noch.

Eine grobe Einteilung der bestehenden Kalküle läßt sich anhand des Vorhandenseins oder Nichtvorhandenseins einer bestimmten Ableitungsregel, der Schnittregel

$$\frac{A \longrightarrow C \qquad C \longrightarrow B}{A \longrightarrow B}$$

vornehmen. Kalküle ohne Schnittregel werden direkte, die mit Schnittregel indirekte genannt.

Indirekte Kalküle leiden darunter, daß die Auswahl der Schnittformel C praktisch keinen Einschränkungen unterliegt; sie sind daher im automatischen Beweisen eher von theoretischem Interesse.

In der Prädikatenlogik erster Stufe tritt aber auch bei "direkten" Beweisen ein ähnliches Problem auf (siehe z.B. [Gor94], Theorem 1.3.7): Sowohl im Sequenzen- wie auch im Modus-Ponens-Kalkül kann man keine obere Schranke für die für den Beweis benötigten Variablen angeben.

Gordeews RPC_n-Kalküle ([Gor94]) versuchen daher über eine Einschränkung der Variablenanzahl Verbesserungen zu errreichen. Darüberhinaus kommen sie ohne Schnittregel aus, sind aber, im Gegensatz zum üblichen Sequenzenkalkül, nicht schwächer als dasselbe Kalkül mit Schnittregel bei endlicher Variablenanzahl.

Die Verwandschaft zu Tarskis Relationenalgebra [TG87] macht diese Kalküle noch interessanter.

2 Grundlegende Definitionen

Der RPC$_n$-Formalismus wird im folgenden für eine beliebige feste Ordinalzahl $n \leq \omega$ erklärt.

Die Sprache $\mathcal{L}_n$ des RPC$_n$-Formalismus besteht aus Symbolen für

Individuenvariablen: $v_0, \ldots, v_n$

Individuenkonstanten: $c_0, \ldots, c_q$

logische Konstanten: $\top, \bot, \neg, \vee, \wedge, \exists, \forall$

Prädikatvariablen: $V_0, \ldots, V_p$

Klammern: (und).

Die Menge dieser Symbole, das Alphabet, wird Σ_n genannt.

Elemente der Menge $\mathcal{V}_n = \{v_0, \ldots, v_n\}$[1] der Individuenvariablen werden mit $x, y, z, \ldots$ bezeichnet. Dabei werden die metalogischen Konstanten $x, y, z, \ldots$ als Abkürzungen für $v_0, v_1, v_2, \ldots$ verstanden. Dies gilt auch für die nachfolgenden metalogischen Konstanten. Individuenkonstanten sind Elemente der endlichen und eventuell leeren Menge $\mathcal{C} = \{c_1, \ldots, c_q\}$. Die Individuenkonstanten seien verschieden von den Individuenvariablen, d.h. es gelte $\mathcal{V}_n \cap \mathcal{C} = \emptyset$. $\mathcal{T}_n = \mathcal{V}_n \cup \mathcal{C}$ wird als Menge der Terme bezeichnet. Terme werden im folgenden mit $a, b, c, \ldots$ abgekürzt. Prädikatvariablen V_i, die in der endlichen Menge $\mathcal{P} = \{V_0, \ldots, V_p\}$ zusammengefaßt und mit $X, Y, Z, \ldots$ gekennzeichnet werden, sind mit einer Stelligkeit $\alpha(V_i) \in \mathbb{N}$ versehen. Logische Konstanten der Sprache $\mathcal{L}_n$ sind $\top, \bot, \neg, \vee, \wedge, \exists$ und $\forall$. Diese stehen für Verum, Falsum, Negation, Disjunktion, Konjunktion, existentielle und universelle Quantifizierung. Als Hilfssymbole werden die Klammern (und) verwendet.

Über der soeben definierten Sprache werden Formeln (Bezeichnung: $A, B, C, \ldots$) als zusammengesetzte Ausdrücke wie folgt definiert:

Definition 2.1 (Literal, Formel) *Zeichenketten der Form $Xt_1 \ldots t_k$ und $\neg Xt_1 \ldots t_k$ mit $X \in \mathcal{P}, \alpha(X) = k$ und $t_i \in \mathcal{T}_n$ werden Literale genannt.*[2] *Diese bilden zusammen mit den Symbolen $\top$ und $\bot$ die atomaren Formeln.*

Ein Ausdruck F ist eine Formel von $\mathcal{L}_n$, wenn er zu jeder Menge Ω gehört, so daß

1. *alle atomaren Formeln in Ω sind,*
2. *mit $A, B \in \Omega$ auch $(A \vee B), (A \wedge B) \in \Omega$ und*
3. *$\exists xA, \forall xA \in \Omega$ falls $A \in \Omega$ und $x \in \mathcal{V}_n$.*

Die Menge aller Formeln von $\mathcal{L}_n$ wird mit $\mathcal{F}[\mathcal{L}_n]$, $\mathcal{F}_n$ oder nur mit $\mathcal{F}$ bezeichnet, falls das entsprechende n aus dem Kontext zu entnehmen ist. Um Klammern zu sparen wird $A_0 \vee \ldots \vee A_i$ als Abkürzung für $(\ldots(A_0 \vee A_1) \vee \ldots \vee A_i)$ verwendet. Entsprechend wird

[1] Dies soll den Fall $\mathcal{V}_\omega = \{v_i | i \in \omega\}$ einschließen.

[2] Zur Bezeichnung von Literalen wird im folgenden meist die Variable L verwendet.

auch für $\wedge$ vorgegangen. Darüberhinaus sollen Quantoren eine höhere Priorität als die binären Junktoren haben und $\wedge$ stärker binden als $\vee$. Klammern werden im folgenden möglichst sparsam verwendet werden; zusätzliche (auch überflüssige) Klammern sollen trotzdem erlaubt sein, wenn dadurch die Struktur einer Formel klarer wird.

Freie und gebundene Variablen werden wie üblich definiert. Der Vollständigkeit halber sei nichtsdestoweniger die formale Definition angegeben. Dabei steht $\bar{X}$ für X oder $\neg X$, $\otimes$ für $\vee$ oder $\wedge$ und $\mathcal{Q}$ für $\exists$ oder $\forall$. Diese Abkürzungen werden auch im weiteren ohne spezielle Kennzeichnung in diesem Sinne verwendet.

Definition 2.2 (freie und gebundene Variablen) *Die Menge $Fr(A)$ der freien Variablen einer Formel A ist rekursiv definiert durch*

$$\mathrm{Fr}(\top) = \mathrm{Fr}(\bot) = \emptyset$$

$$\mathrm{Fr}(\bar{X}t_1 \ldots t_k) = \{x \in \mathcal{V}_n \mid x = t_i \ \textit{für ein} \ i \leq k\}$$

$$\mathrm{Fr}(A \otimes B) = \mathrm{Fr}(A) \cup \mathrm{Fr}(B)$$

$$\mathrm{Fr}(\mathcal{Q}xA) = \mathrm{Fr}(A) \setminus \{x\}$$

Die Menge $Bd(A)$ der gebundenen Variablen einer Formel A ist rekursiv definiert durch

$$\mathrm{Bd}(\top) = \mathrm{Bd}(\bot) = \emptyset,$$

$$\mathrm{Bd}(\bar{X}t_1 \ldots t_k) = \emptyset$$

$$\mathrm{Bd}(A \otimes B) = \mathrm{Bd}(A) \cup \mathrm{Bd}(B)$$

$$\mathrm{Bd}(\mathcal{Q}xA) = \mathrm{Bd}(A) \cup \{x\}$$

Die Menge $\mathrm{Var}(A)$ *der Variablen einer Formel A ist definiert als* $\mathrm{Var}(A) = \mathrm{Fr}(A) \cup \mathrm{Bd}(A)$.

Auf Formeln werden im weiteren verschiedene einfache Operationen angegeben. Diese umfassen die Substitution von Termen für Variablen, die gebundene Umbenennung und die für die Reduktionskalküle charakteristische Variablenelimination.

Definition 2.3 (Substitution mit gebundener Umbenennung) *Die Substitution $A[x/t]$ einer Variablen x durch einen Term t in einer Formel A ist rekursiv definiert durch*

$$\top[x/t] = \top, \ \ \bot[x/t] = \bot$$

$$\bar{X}t_1 \ldots t_k[x/t] = \bar{X}s_1 \ldots s_k, \ \textit{wobei} \ s_i = \begin{cases} t, & \textit{falls} \quad t_i = x \\ t_i, & \textit{falls} \quad t_i \neq x \end{cases}$$

$$(A \otimes B)[x/t] = A[x/t] \otimes B[x/t]$$

$$(\mathcal{Q}yA)[x/t] = \begin{cases} \mathcal{Q}yA & \textit{falls} \quad x \notin \mathrm{Fr}(\mathcal{Q}yA) \\ \mathcal{Q}yA[x/t] & \textit{falls} \quad x \in \mathrm{Fr}(\mathcal{Q}yA) \wedge t \neq y \\ \mathcal{Q}xA[x \rightleftharpoons y] & \textit{falls} \quad x \in \mathrm{Fr}(\mathcal{Q}yA) \wedge t = y \end{cases}$$

Dabei entsteht $A[x \rightleftharpoons y]$ aus A dadurch, daß man jedes Vorkommen (ob frei oder gebunden) von x in A durch y ersetzt und umgekehrt.

Diese Definition der Substitution hat den Vorteil, daß keine Variablen benötigt werden, die nicht schon zuvor in der Formel vorkamen. Wie wir noch sehen werden, ist diese Eigenschaft für die RPC-Kalküle bei fester, endlicher Variablenanzahl essentiell.

Definition 2.4 (gebundene Umbenennung) *Für jede Abbildung $\beta : \mathcal{V}_n \to \mathcal{V}_n$ wird die Relation $\overset{\beta}{\sim} \subseteq \mathcal{F} \times \mathcal{F}$ definiert als die kleinste Relation, die die folgenden Bedingungen erfüllt:*

1. $\top \overset{\beta}{\sim} \top, \ \bot \overset{\beta}{\sim} \bot$
2. $\bar{X}t_0 \dots t_k \overset{\beta}{\sim} \bar{X}s_0 \dots s_k \quad falls\ s_i = \begin{cases} t_i & falls \quad t_i \in \mathcal{C} \\ \beta(t_i) & falls \quad t_i \in \mathcal{V}_n \end{cases}$
   und $\beta|_{\mathrm{Var}(\bar{X}t_0 \dots t_k)} : \mathrm{Var}(\bar{X}t_0 \dots t_k) \to \mathrm{Var}(\bar{X}s_0 \dots s_k)$ *eine Bijektion ist.*
3. $A \otimes B \overset{\beta}{\sim} C \otimes D \quad falls\ A \overset{\beta}{\sim} C$ *und* $B \overset{\beta}{\sim} D$
4. $\mathcal{Q}xA \overset{\beta}{\sim} \mathcal{Q}yB \quad falls\ A \overset{\beta'}{\sim} B$, *wobei* $\beta'(z) = \begin{cases} y & falls \quad z = x \\ \beta(z) & falls \quad z \neq x \end{cases}$

Anstatt $\overset{id}{\sim}$ wird auch $\sim$ geschrieben und diese Relation als gebundene Umbenennung bezeichnet.
Für $\beta : \mathcal{V}_n \to \mathcal{V}_n$ wird $\beta = \{x_0 \mapsto y_0, \dots, x_i \mapsto y_i\}$ als alternative Schreibweise für die Abbildung $\beta(x) = \begin{cases} y_i & falls \quad x = x_i \\ x & sonst \end{cases}$ *verwendet.*

Definition 2.5 (Variablenelimination) *Das Eliminieren einer Variablen x aus einer Formel F, $F[\div x]$, wird rekursiv definiert durch:*

$\top[\div x] = \top, \ \bot[\div x] = \bot$

$\bar{X}t_1 \dots t_k[\div x] = \begin{cases} \bar{X}t_1 \dots t_k & falls \quad x \notin \mathrm{Var}(\bar{X}t_1 \dots t_k) \\ \bot & sonst \end{cases}$

$(A \otimes B)[\div x] = A[\div x] \otimes B[\div x]$

$(\mathcal{Q}yA)[\div x] = \begin{cases} \mathcal{Q}yA & falls \quad x = y \\ \mathcal{Q}yA[\div x] & sonst \end{cases}$

Darüberhinaus sei $A[x \div y] = \begin{cases} A & falls \quad x = y \\ A[\div y] & sonst \end{cases}$

Später werden wir auch den Begriff des Wortes und der Konkatenation benötigen. Daher nun die folgende Definition.

Definition 2.6 (Wort, Konkatenation) *Sei M eine beliebige nichtleere Menge. Ein n-Tupel $(m_1, \ldots, m_n) \in M^n$ heißt Wort über M. Für $n = 0$ ist dies λ, das leere Wort. Ist w das Wort $(m_1, \ldots, m_n)$, so ist n die Länge von w (in Zeichen: $|w| = n$) und für $1 \leq i \leq n$ ist w_i das Zeichen m_i, d.h. die i-te Komponente. Das Wort (m) wird mit dem Zeichen m identifiziert. $M^* = \bigcup_{n \in \mathbb{N}} M^n$ bezeichnet die Menge aller Worte über M.*

Für zwei Worte $u = (a_1, \ldots, a_m)$ und $v = (b_1, \ldots, b_n)$ ist $u \circ v = (a_1, \ldots, a_m, b_1, \ldots, b_n)$ die Konkatenation von u und v. Anstatt $u \circ v$ kann man auch einfach uv schreiben.

3 Die RPC$_n$-Reduktionskalküle

Um eine genauere Untersuchung der Formeln im Rahmen der Termersetzung zu ermöglichen ist es angebracht, Formeln nicht nur als Zeichenreihen zu verstehen, sondern auch als Elemente einer Termalgebra. Diese (frei generierte) Termalgebra entsteht durch Interpretation der logischen Konstanten und Literale als Konstruktorfunktionen mit der entsprechenden Stelligkeit (siehe z.B. [Gal86]). Genauer handelt es sich bei der betrachteten Termalgebra um eine mehrsortige Algebra (Sorten sind Formeln und Terme), was in dem hier betrachteten Zusammenhang allerdings keine Rolle spielt, da nur die Formelkonstruktoren relevant sind. Diese sind im einzelnen:

$C_\vee, C_\wedge : \mathcal{F} \times \mathcal{F} \longrightarrow \mathcal{F}$ sind die Disjunktions- bzw. Konjunktionskonstruktoren mit $C_\vee(A, B) = A \vee B$ und $C_\wedge(A, B) = A \wedge B$.

$C_{\exists i}, C_{\forall i} : \mathcal{F} \longrightarrow \mathcal{F}$ sind die verschiedenen existentiellen und universellen Quantorenkonstruktoren, wobei v_i die gebundene Variable ist; dabei ist $C_{\exists i}(A) = \exists v_i A$ und $C_{\forall i}(A) = \forall v_i A$.

$C_A :\longrightarrow \mathcal{F}$ bezeichne für eine beliebige atomare Formeln A den entsprechenden Atomformelkonstruktor, d.h. $C_A = A$ für atomares A.

Die Konstruktorfunktionen für Konjunktion und Disjunktion sollen darüberhinaus für beliebige Stelligkeiten $s \geq 2$ definiert sein und werden dabei als AC-Operatoren verstanden (assoziativ und kommutativ). Dies ist möglich, da sich in der booleschen Algebra für beliebige Formeln A, B, C zeigen läßt:

$$(A \otimes B) \otimes C \equiv A \otimes (B \otimes C) \quad (1)$$

$$A \otimes B \equiv B \otimes A \quad (2)$$

Wenn die Anwendbarkeit einer Termersetzungsregel überprüft werden soll, seien die Junktoren immer als AC-Operatoren gedacht. Die Darstellung mittels AC-Operatoren entspricht bis auf die fehlende Idempotenzregel $A \otimes A \longrightarrow A$ der Mengenformulierung des RPC$_n$-Formalismus in Anmerkung 8 von [Gor94].

Mittels der soeben beschriebenen Termalgebra lassen sich weitere wichtige Begriffe formal definieren.

Definition 3.1 (Position, Subterm, Markierung, Subterm-Ersetzung) *Sei p ein Wort über $\mathbb{N}$, $A \in \mathcal{F}$. Dann sind Positionen p, Subterme $A|_p$ und Markierungen $A(p)$ an einer Position p definiert durch:*

- *$p = \lambda$ ist eine Position in A. Die Markierung $A(p)$ von A an der Position $p = \lambda$ ist $A(p) = C_X$, wobei $A = C_X(A_0, \ldots, A_k)$ und C_X ein k-stelliger Formelkonstruktor ist. Der Subterm $A|_\lambda$ von A an der Position $p = \lambda$ ist $A|_\lambda = A$.*
- *$p = ip'$ mit $i \in \mathbb{N}$ und $p' \in \mathbb{N}^*$ ist eine Position in A, falls $A = C_X(A_0, \ldots, A_k)$, C_X ein k-stelliger Formelkonstruktor, $0 \leq i \leq k$ und p' eine Position in A_i ist. Die Markierung $A(p)$ ist $A(p) = A_i(p')$, der Subterm $A|_p$ von A an der Position p ist $A|_p = A_i|_{p'}$.*

Die Ersetzung eines Subterms in A an Position p durch B wird mit $A[p \leftarrow B]$ bezeichnet, die Menge aller Positionen in einer Formel F mit Pos(F)*. Darüberhinaus soll unter $F[p \leftarrow A/B]$ die Ersetzung von Subterm A an Position p durch B verstanden werden und, falls die Position p aus dem Kontext hervorgeht, ist auch $F[A/B]$ als verkürzte Schreibweise erlaubt.*

Eine Position p ist eine Subposition von q, $p \leq q$, falls q ein Präfix von p ist. Falls S ein Subterm von A ist, so schreibt man auch $A \trianglerighteq S$.

Es ist nun möglich das Termersetzungssystem RPC$_n$ anzugeben. Die Formulierung weicht etwas von der in [Gor94] ab.

Definition 3.2 (RPC$_n$ **Termersetzungssystem)** RPC$_n$ *besteht aus folgenden Termersetzungsregeln:*

$$
\begin{array}{lrcl}
(R1) & A \vee \top & \longrightarrow & \top \\
(R2) & L \vee \neg L & \longrightarrow & \top \\
(R3) & A \wedge \top & \longrightarrow & A \\
(R4) & A \vee (B \wedge C) & \longrightarrow & (A \vee B) \wedge (A \vee C) \\
(R5) & \exists x A & \longrightarrow & \exists x A \vee A[x/a] \\
(R6) & \forall x A \vee B & \longrightarrow & \forall x A \vee B \vee A[\div x] \vee \forall y(\forall y B \vee A[x \div y][x/y]) \\
(R6') & \forall x A & \longrightarrow & \forall x A \vee A[\div x] \vee \forall y A[x \div y][x/y]
\end{array}
$$

Dabei ist $A, B, C \in \mathcal{F}_n$, $x, y \in \mathcal{V}_n$, $a \in \mathcal{T}_n$ und L ein beliebiges Literal.

Die Termersetzungsregeln sind so zu interpretieren, daß in jeder Regel für jede metalogische Konstante jede beliebige Formel (bzw. Literal, Variable, Term) einzusetzen ist. Das Termersetzungssystem besteht also aus einer unendlichen Menge von Ersetzungsregeln, deren Formeln in der Sprache $\mathcal{L}_n$ liegen.

Alternativ kann man das Regelsystem auch als Strukturbeschreibung einer mehrsortigen Termalgbra verstehen. Die drei Sorten sind Formeln, Variablen und Terme. Unter diesen Voraussetzungen sind dann A, B und C Formelvariablen, x ist eine Termvariable, die eine Variable aus $\mathcal{V}_n$ beschreibt; ferner sind a und y Parameter, wobei a von der Sorte Term und y von der Sorte Variable ist. Die Ersetzungen sind dann als Axiome zu lesen, wobei $\longrightarrow$ dann als Gleichheit zu interpretieren ist.

3.1 Ableitungsrelation und Beweisbarkeit

Nun soll für die RPC$_n$-Kalküle der Begriff der Ableitbarkeit definiert werden. Dazu wird der Ableitungsbegriff des Termersetzungssystems verwendet. Im folgenden soll dabei n als fest vorgegeben angenommen werden.

Definition 3.3 (Ableitungsrelation) *Die Relation* $\longrightarrow_{\mathrm{RPC}} \subseteq \mathcal{F} \times \mathcal{F}$ *enthält genau die Formelpaare* (A_l, A_r)*, für die es eine Regel* $G \longrightarrow H$ *im obigen Termersetzungssystem und eine Position* p *in* A_l *gibt, so daß* $A_l|_p =_{AC} G$ *und* $A_r = A_l[p \leftarrow H]$*.*

Dabei ist $A =_{AC} B$*, falls sich* A *durch Anwendung der Gleichungen (1) und (2) für Assoziativität und Kommutativität in* B *überführen läßt.*

Der Subterm $A_l|p$ *wird Redex genannt.*

Die Relation $\xrightarrow{k}_{\mathrm{RPC}}$ *ist das* k*-fache Produkt von* $\longrightarrow_{\mathrm{RPC}}$*, d.h.* $A \xrightarrow{k}_{\mathrm{RPC}} B$ *genau dann, wenn* A *in genau* k *Reduktionsschritten zu* B *abgeleitet werden kann.* $\xrightarrow{*}_{\mathrm{RPC}}$ *ist der reflexiv-transitive Abschluß von* $\longrightarrow_{\mathrm{RPC}}$*, d.h.* $\xrightarrow{*}_{\mathrm{RPC}} = \bigcup_{k \in \mathbb{N}} \xrightarrow{k}_{\mathrm{RPC}}$.

Als nächstes wollen wir nun den Begriff der Beweisbarkeit im RPC-Kalkül definieren.

Definition 3.4 (Beweisbegriff) $A \in \mathcal{F}_n$ *ist beweisbar in* RPC$_n$*,* RPC$_n \vdash A$*, genau dann, wenn* $A \xrightarrow{*}_{\mathrm{RPC}} \top$*.*

Gordeew zeigt in [Gor94] die Korrektheit und relative Vollständigkeit des RPC$_n$-Formalismus (Theorem 3.1). Relative Vollständigkeit bedeutet dabei, daß dieser Formalismus gleichmächtig zu MPC$_n$, EPC$_n$ und SPC$_n$ hinsichtlich Beweisbarkeit ist. MPC$_n$, EPC$_n$ und SPC$_n$ bezeichnen dabei entsprechend das Modus-Ponens-Prädikatenkalkül, die "Equational Logic" und das Sequenzenkalkül mit Schnittregel, jeweils mit höchstens n Individuenvariablen.

Insbesondere erhält man damit für den Fall $n = \omega$ die Vollständigkeit von RPC$_\omega$ für die Prädikatenlogik erster Stufe.

Darüberhinaus hat Gordeew in [Gor96] einen Satz über Variablen-Kompaktheit bewiesen, der besagt, daß auch schon eine endliche Anzahl Variablen im RPC-Kalkül zur Vollständigkeit ausreicht: Zu jeder gültigen Formel der Prädikatenlogik erster Stufe, die höchstens m Variablen und höchstens m-stellige Prädikate enthält, gibt es einen Beweis im RPC$_{m+2}$-Kalkül.

Wir wollen diesen Abschnitt mit einem Beispielbeweis abschließen.

Beispiel: Wir suchen einen Beweis für die Formel $F = \exists y Pxy \vee \exists x \forall y \neg Pxy$.

Ein möglicher Beweis von F in RPC$_2$ ist dann beispielsweise der folgende:

$$\exists y Pxy \vee \underline{\exists x \forall y \neg Pxy}$$

$$\downarrow_{(R5,x)}$$

$$\underline{\exists y Pxy} \vee \overline{\exists x \forall y \neg Pxy \vee \underline{\forall y \neg Pxy}}$$

$$\downarrow_{(R6,y)}$$

$$\exists x \forall y \neg Pxy \vee \overline{\forall y \neg Pxy \vee \exists y Pxy \vee \forall y (\underline{\forall y \exists y Pxy} \vee \neg Pxy)}$$

$$\downarrow_{(R6')}$$

$$\exists x \forall y \neg Pxy \vee \ldots \vee \forall y (\overline{\forall y \exists y Pxy \vee \underline{\exists y Pxy}} \vee \neg Pxy)$$

$$\downarrow_{(R5,y)}$$

$$\exists x \forall y \neg Pxy \vee \ldots \vee \forall y(\forall y \exists y Pxy \vee \overline{\exists y Pxy \vee \underline{Pxy}} \vee \neg Pxy)$$

$$\downarrow_{(R2)}$$

$$\exists x \forall y \neg Pxy \vee \ldots \vee \forall y(\underline{\forall y \exists y Pxy \vee \exists y Pxy \vee \overline{\top}})$$

$$\downarrow_{(R1)}$$

$$\exists x \forall y \neg Pxy \vee \ldots \vee \underline{\forall y \overline{\top}}$$

$$\downarrow_{(R6')}$$

$$\underline{\exists x \forall y \neg Pxy \vee \ldots \vee \overline{\forall y \top \vee \top}}$$

$$\downarrow_{(R1)}$$

$$\overline{\top}$$

Dabei sind Redexe durch Unterstreichung markiert, die daraus entstandene Teilformel nach Regelanwendung ist durch Überstreichung gekennzeichnet.

4 Lineare Beweissuche

Das dem RPC-Formalismus zu Grunde liegende Termerstzungssystem läßt noch keine Rückschlüsse darauf ziehen, wie man einen möglichen Beweis findet. Natürlich ist ein möglicher Weg der Beweissuche, alle erlaubten Reduktionsmöglichkeiten bis zu einer festen Tiefe der Reihe nach auszuprobieren. Allerdings kann der Suchraum dabei sehr groß sein. Außer den verschiedenen Reduktionspositionen liefern die Parameter a und y der Regeln $(R5)$, $(R6)$ und $(R6')$ Variationsmöglichkeiten.

Ziel dieses und der folgenden Abschnitte ist es, ein Verfahren zu entwickeln, das für Formeln, die in RPC_n beweisbar sind, einen Beweis findet, ohne dabei durch eine vollständige Generierung des Beweisbaumes den Indeterminismus der Ableitungsrelation zu berücksichtigen. Unter Indeterminismus ist dabei zu verstehen, daß es keinen funktionalen Zusammenhang zwischen einer Formel und einer daraus abgeleiteten Formel gibt. Stattdessen kann eine Formel viele mögliche Nachfolger im nächsten Ableitungsschritt haben.

Der Begriff der zuverlässigen Reduktionsstrategie soll ein Verfahren liefern, das eine deterministische, aber dennoch korrekte und vollständige Beweisgenerierung ermöglicht. Lineare Beweissuche steht in diesem Zusammenhang für Suchverfahren, die ohne Backtracking auskommen.

4.1 Reduktionsstrategien

Reduktionsstrategien liefern einen Weg, zu einer gegebenen Formel eine feste Nachfolger-Formel auszuwählen. Dazu nun die folgenden Definitionen.

Definition 4.1 (Reduktionskette, Beweis) *Eine Formelfolge $(A_0, \ldots, A_k) \in \mathcal{F}^*$ mit $k \in \mathbb{N}$ ist eine Reduktionskette, falls $A_0 \longrightarrow_{\mathrm{RPC}} A_1 \longrightarrow_{\mathrm{RPC}} \ldots \longrightarrow_{\mathrm{RPC}} A_k$. Die letzte Formel einer Reduktionskette wird aktuelle Formel (*CF*) genannt. Die Funktion $\mathrm{CF} : \mathcal{F}^* \longrightarrow \mathcal{F} : (A_0, \ldots, A_k) \mapsto A_k$ liefert für beliebige Reduktionsketten die aktuelle Formel. Falls $A = A_0$ und $A_k = \top$, so heißt die Reduktionskette $(A_0, \ldots, A_k)$ Beweis für A.*

Definition 4.2 (Reduktionsstrategie) *Eine Reduktionsstrategie ist eine Funktion*

$$\mathrm{RS} : \mathcal{F}^* \longrightarrow \mathcal{F}^*$$

mit

$$\mathrm{RS}(A_0, \ldots, A_k) = \begin{cases} (A_0, \ldots, A_k, A_{k+1}) & \text{falls } A_k \text{ reduzibel} \\ undefiniert & \text{falls } A_k \text{ irreduzibel} \end{cases}$$

Dabei sind sowohl $(A_0, \ldots, A_k)$ als auch $(A_0, \ldots, A_k, A_{k+1})$ Reduktionsketten, womit insbesondere $A_k \longrightarrow_{\mathrm{RPC}} A_{k+1}$ gilt. RS *verlängert also eine Reduktionskette $(A_0, \ldots, A_k)$ um eine Formel. Dies entspricht der Verlängerung eines bestehenden Beweisversuchs um einen Beweisschritt.*

Essentiell ist der deterministische Aufbau der Reduktionskette im Vergleich zur Relation $\longrightarrow_{\mathrm{RPC}}$, die beliebig viele Nachfolger zu einer Formel zuläßt.

Für eine Formel F, eine Reduktionsstrategie RS *und $k \in \mathbb{N}$ wird eine Reduktionskette R der Form $R = (F_0, \ldots, F_k)$ mit $F = F_0$ und $F_{i+1} = \mathrm{CF}(\mathrm{RS}(F_0, \ldots, F_i))$ eine von* RS *(aus F) generierte Reduktionskette genannt. Falls $F_k = \top$, so wird R auch der von* RS *generierte Beweis von F genannt.*

Es ist nicht unbedingt notwendig zur Auswahl des nächsten Beweisschrittes den kompletten vorhergehenden Beweis zuzulassen. Ausreichend wäre auch die letzte Formel des Beweisversuchs. Allerdings ist die zusätzliche Information, die in der bestehenden Reduktionskette enthalten ist, oft sinnvoll zur Definition einer konkreten Beweisstrategie.

Andererseits kann es manchmal hilfreich sein, eine Beweisstrategie mittels zusätzlicher Informationen zu spezifizieren, die aus obiger Definition nicht sofort ersichtlich sind. Solche Informationen werden im folgenden an verschiedenen Stellen ohne expliziten Hinweis verwendet, lassen sich aber immer aus dem bestehenden Beweis extrahieren.

Definition 4.3 (zuverlässige Reduktionsstrategie) *Eine Reduktionsstrategie* RS *heißt zuverlässig, falls für alle* $F \in \mathcal{F}_n$ *gilt:*[3]

$$\mathrm{RPC}_n \vdash F \;\Rightarrow\; \exists k \in \mathbb{N} : \mathrm{CF}(\mathrm{RS}^k(F)) = \top.$$

Eine zuverlässige Reduktionsstrategie findet also für jede Formel F *mit* $\mathrm{RPC}_n \vdash F$ *einen Beweis.*

4.2 Symbolvorkommen

Zur Definition der folgenden Reduktionsstrategien wird es erforderlich sein, bestimmte Symbol-, Konstruktor- bzw. Subformelvorkommen auszuzeichen. Diese Auszeichnung wird zweckmäßigerweise anhand der Position dieser Objekte vorgenommen. Eine vereinfachte Schreibweise ist dabei hilfreich. So sollen für ein Symbol $S \in \Sigma_n$, einen Konstruktor C, wie in 3 definiert, oder eine Subformel A einer Formel F, sowie für eine Position $p \in \mathrm{Pos}(F)$ die Bezeichnungen S_p, $C_{S,p}$ und A_p das Vorkommen des entsprechenden Objekts an Position p in F ausdrücken.

4.3 Symbolmarkierungen

Symbolmarkierungen[4] dienen der Kennzeichnung verschiedener Reduktionsmöglichkeiten in der aktuellen Formel eines Beweisversuchs. Ein markiertes Symbol (genauer: Symbolvorkommen) zeigt dabei an, daß an diesem Symbol (bzw. an einem Subterm, der dieses Symbol enthält) eine Ableitungsmöglichkeit vorhanden ist, und seit welchem Beweisschritt diese besteht. Quantorensymbole sind darüberhinaus mit einem Term versehen, der die Reduktionsmöglichkeit mit der entsprechenden Eigenvariable angibt.

Markierungen werden, da sie auch das erste Auftreten einer Reduktionsmöglichkeit anzeigen sollen, für Reduktionsketten definiert. Dazu wird jeder Formel A_k einer Reduktionskette $(A_0, \ldots, A_l)$ eine Funktion $M_k : \mathrm{Pos}(A_k) \longrightarrow \mathbb{P}(\mathbb{N} \times (\mathcal{T}_n \cup \{\Box\}))$ zugeordnet.[5]

Eine Markierung $M_i(p)$ oder $M_i(S_p)$ eines Symbols S an Position p in Formel A_i besteht also aus einer Menge von Tupeln (k, t), wobei $k \in \mathbb{N}$ anzeigt, daß die entsprechende Reduktionsmöglichkeit seit Beweisschritt k besteht, also in Formel A_k zum ersten Mal auftrat. t zeigt im Falle eines markierten Quantors den Term an, der als Reduktionsparameter verwendet werden kann. Dieser Term entspricht dem a der Regel $(R5)$ bzw. dem y der

[3] $\mathrm{RS}^k(F)$ steht für die k-fache Anwendung der Funktion RS auf F.
[4] Alternativ könnte man auch von Konstruktormarkierungen sprechen.
[5] $\mathbb{P}(X)$ bezeichnet die Potenzmenge der Menge X.

Regeln ($R6$) und ($R6'$). Allen anderen Symbolen wird als zweite Komponente t der Markierung immer das "Blank" ($\Box$) zugeordnet. Die Bezeichnung $M_i(S_p)$ entspricht $M_i(p)$, wobei allerdings zusätzlich angenommen wird, daß sich an Position p Symbol S befindet.

Um die Definition der Symbolmarkierungen einfacher zu halten, wird eine Familie von Hilfsfunktionen M' zur Definition herangezogen. Diese Aufteilung in Markierung M_k und Hilfsfunktion M'_k wird auch durch deren intuitive Bedeutung unterstützt, denn man kann sich den Markierungsvorgang in zwei Phasen eingeteilt denken: In der ersten Phase, der Präreduktionsphase, werden alle Reduktionsmöglichkeiten in der Formel A_k festgestellt. In der auf einen Reduktionsschritt folgenden Postreduktionsphase werden Änderungen an den Markierungen, die durch die Reduktion erforderlich werden, berücksichtigt. Diese beiden Markierungsphasen mit zugeordneten Funktionsfamilien M und M' sollen nun näher erläutert werden.

4.3.1 Präreduktionsphase

Gegeben sei eine Reduktionskette $R = (F_0, \ldots, F_l)$. Für jedes k mit $0 \leq k \leq l$ wird dann die Markierungsfunktion M_k zur Angabe der Reduktionsmöglichkeiten in Formel F_k wie folgt definiert:

$$
\begin{aligned}
M_k(\top_p) &= \begin{cases} \{(k,\Box)\} & \text{falls } M'_k(\top_p) = \emptyset \text{ und } (F_k \trianglerighteq A \vee \top_p \text{ oder } F_k \trianglerighteq A \wedge \top_p) \\ M'_k(\top_p) & \text{sonst} \end{cases} \\
M_k(L_p) &= \begin{cases} \{(k,\Box)\} & \text{falls } M'_k(L_p) = \emptyset \text{ und } F_k \trianglerighteq L_p \vee \neg L \\ M'_k(L_p) & \text{sonst} \end{cases} \\
M_k(\wedge_p) &= \begin{cases} \{(k,\Box)\} & \text{falls } M'_k(\wedge_p) = \emptyset \text{ und } F_k \trianglerighteq A \vee (B \wedge_p C) \\ M'_k(\wedge_p) & \text{sonst} \end{cases} \\
M_k(\exists_p) &= \left\{(k,t) \;\middle|\; t \in \mathcal{C} \cup \mathrm{Var}(F_k) \cup \{v_{neu}\} \;\wedge\; \forall k' < k : (k',t) \notin M'_k(\exists_p)\right\} \;\cup\; M'_k(\exists_p) \\
M_k(\forall_p) &= \left\{(k,t) \;\middle|\; t \in \mathrm{Var}(F_k) \cup \{v_{neu}\} \;\wedge\; \forall k' < k : (k',t) \notin M'_k(\forall_p)\right\} \;\cup\; M'_k(\forall_p) \\
M_k(S_p) &= \emptyset \text{ für alle anderen Positionen}
\end{aligned}
$$

Dabei bezeichnet v_{neu} eine neue Variable, falls es eine solche gibt. D.h. $v_{neu} \in \mathcal{V}_n \setminus \mathrm{Var}(F_k)$, falls diese Menge nicht leer ist; ansonsten ist in obiger Definition $\mathrm{Var}(F_k) \cup \{v_{neu}\} = \mathcal{V}_n$.

4.3.2 Postreduktionsphase

Auch hier wird eine Reduktionskette $R = (F_0, \ldots, F_l)$ als gegeben angenommen und für jedes k mit $0 \leq k \leq l$ die entsprechende Hilfsfunktion $M'_k : \mathrm{Pos}(F_k) \longrightarrow \mathbb{P}(\mathbb{N} \times (\mathcal{T}_n \cup \{\Box\}))$ definiert. Die Funktionsfamilie M' dient der Anpassung der Markierungen, die ein Reduktionsschritt erforderlich macht.

$$M'_0(p) = \emptyset \text{ für alle } p \in \mathrm{Pos}(F_0)$$

Für $k > 0$ wird M'_k durch Fallunterscheidung nach der Regel (Ri), die von F_{k-1} zu F_k führt, definiert:

($R1$): $F_k = F_{k-1}[A \vee T \,/\, T_p]$

$$M'_k(q) = \begin{cases} \emptyset & \text{falls } q = p \\ M_{k-1}(q) & \text{sonst} \end{cases}$$

($R2$): $F_k = F_{k-1}[L \vee \neg L \,/\, T_p]$

$$M'_k(q) = \begin{cases} \emptyset & \text{falls } q = p \\ M_{k-1}(q) & \text{sonst} \end{cases}$$

($R3$): $F_k = F_{k-1}[A_o \wedge T \,/\, A_p]$

$$M'_k(q) = \begin{cases} M_{k-1}(q') & \text{falls } q < p, \text{ wobei } q = p \circ s \text{ und } q' = o \circ s^6 \\ M_{k-1}(o) & \text{falls } q = p \text{ und } A_o \neq \top \\ \emptyset & \text{falls } q = p \text{ und } A_o = \top \\ M_{k-1}(q) & \text{sonst} \end{cases}$$

($R4$): $F_k = F_{k-1}[A_{o_1} \vee (B_{o_2} \wedge C_{o_3}) \,/\, (A_{p_1} \vee_{p_2} B_{p_3}) \wedge_{p_4} (A_{p_5} \vee_{p_6} C_{p_7})]$

$$M'_k(q) = \begin{cases} M_{k-1}(q') & \text{falls } q \leq p_1 \text{ wobei } q = p_1 \circ s \text{ und } q' = o_1 \circ s \\ M_{k-1}(q') & \text{falls } q \leq p_3 \text{ wobei } q = p_3 \circ s \text{ und } q' = o_2 \circ s \\ M_{k-1}(q') & \text{falls } q \leq p_5 \text{ wobei } q = p_5 \circ s \text{ und } q' = o_1 \circ s \\ M_{k-1}(q') & \text{falls } q \leq p_7 \text{ wobei } q = p_7 \circ s \text{ und } q' = o_3 \circ s \\ \emptyset & \text{falls } q = p_2, q = p_4 \text{ oder } q = p_6 \\ M_{k-1}(q) & \text{sonst} \end{cases}$$

($R5$): $F_k = F_{k-1}[\exists_{o_1} x A_{o_2} \,/\, \exists_{p_1} x A_{p_2} \vee_{p_3} A[x/a]_{p_4}]$

$$M'_k(q) = \begin{cases} M_{k-1}(o_1) \setminus \{(k', a) \mid k' < k\} & \text{falls } q = p_1 \\ M_{k-1}(q') & \text{falls } q \leq p_2 \text{ wobei } q = p_2 \circ s \text{ und } q' = o_2 \circ s \\ \emptyset & \text{falls } q = p_3 \text{ oder } q \leq p_4 \\ M_{k-1}(q) & \text{sonst} \end{cases}$$

($R6$): $F_k = F_{k-1}[\forall_{o_1} x A_{o_2} \vee B_{o_3} \,/\, \forall_{p_1} x A_{p_2} \vee_{p_3} B_{p_4} \vee_{p_5} A[\div y]_{p_6} \vee_{p_7} \forall_{p_8} y(\forall y B \vee A[x \div y][x/y])]$

$$M'_k(q) = \begin{cases} M_{k-1}(o_1) \setminus \{(k', y) \mid k' < k\} & \text{falls } q = p_1 \\ M_{k-1}(q') & \text{falls } q \leq p_2 \text{ wobei } q = p_2 \circ s \text{ und } q' = o_2 \circ s \\ M_{k-1}(q') & \text{falls } q \leq p_4 \text{ wobei } q = p_4 \circ s \text{ und } q' = o_3 \circ s \\ \emptyset & \text{falls } q = p_3, q = p_5, q \leq p_6, q = p_7 \text{ oder } q \leq p_8 \\ M_{k-1}(q) & \text{sonst} \end{cases}$$

($R6'$): $F_k = F_{k-1}[\forall_{o_1} x A_{o_2} \,/\, \forall_{p_1} x A_{p_2} \vee_{p_3} A[\div y]_{p_4} \vee_{p_5} \forall_{p_6} y A[x \div y][x/y]]$

$$M'_k(q) = \begin{cases} M_{k-1}(o_1) \setminus \{(k', y) \mid k' < k\} & \text{falls } q = p_1 \\ M_{k-1}(q') & \text{falls } q \leq p_2 \text{ wobei } q = p_2 \circ s \text{ und } q' = o_2 \circ s \\ \emptyset & \text{falls } q = p_3, q \leq p_4, q = p_5 \text{ oder } q \leq p_6 \\ M_{k-1}(q) & \text{sonst} \end{cases}$$

4.4 MD-Reduktionsstrategien

Definition 4.4 (p-Reduktion, (p,t)-Reduktion, maximale p-Reduktion) *Sei F eine Formel und $p \in \mathrm{Pos}(F)$. Dann wird $F \longrightarrow_{\mathrm{RPC}} F'$ eine p-Reduktion genannt, falls der Redex Q in F eine der folgenden Formen annimmt:*

$$A \vee T_p, \quad L_p \vee \neg L, \quad A \wedge T_p, \quad A \vee (B \wedge_p C),$$
$$\exists_p x A, \quad \forall_p x A \vee B, \quad \forall_p x A$$

Eine p-Reduktion ist eine (p,t)-Reduktion, falls $t = \Box$ für eine Reduktion mittels einer der RPC_n-Regeln $(R1)$, $(R2)$, $(R3)$ und $(R4)$, $t = a$ für eine Reduktion mittels Regel $(R5)$ und Substitutionsterm a in $A[x/a]$, sowie $t = y$ für eine Reduktion mittels einer der Regeln $(R6)$ bzw. $(R6')$, wobei y die verwendete Eigenvariable ist.

Eine p-Reduktion in einer Formel F heißt maximal, falls es keine weitere p-Reduktion in F mit Redex $Q' \triangleright Q$ gibt.

Beispiel: In den folgenden Beispielen ist der Redex Q durch Unterstreichen markiert.

1. $\forall x(Qx \wedge \exists y(\underline{Pxy \vee \neg Pxy})) \xrightarrow{(R2)}_{\mathrm{RPC}} \forall x(Qx \wedge \exists y \top)$ ist eine maximale $(1,2,1)$-Reduktion.
2. $Px \vee (\underline{Qx \vee (Rx \wedge \bot)}) \xrightarrow{(R4)}_{\mathrm{RPC}} Px \vee ((Qx \vee Rx) \wedge (Qx \vee \bot))$ ist eine $(2,2)$- oder $((2,2),\Box)$-Reduktion, die nicht maximal ist.
3. $\underline{\exists x Px} \xrightarrow{(R5)}_{\mathrm{RPC}} \exists x Px \vee Py$ ist eine maximale λ- oder (λ, y)-Reduktion.

Definition 4.5 (MD-Reduktionsstrategie) *Sei* RS *eine Reduktionsstrategie. Falls für jede von* RS *generierte Reduktionskette $R = (F_0, \ldots, F_l)$ jede Reduktion $F_k \longrightarrow_{\mathrm{RPC}} F_{k+1}$ für $0 \leq k < l$ eine maximale p-Reduktion ist, so ist* RS *eine* MD*-Reduktionsstrategie (MD: maximal disjunction).*

Man überprüft leicht, daß für eine von einer MD-Reduktionsstrategie generierte Reduktionskette $R = (F_0, \ldots, F_l)$ und $k < l$ jede Reduktion $F_k \longrightarrow_{\mathrm{RPC}} F_{k+1}$ eine p-Reduktion mit $M_k(p) \neq \emptyset$ ist. Außerdem gilt das folgende Lemma:

Lemma 4.6 *Sei $R = (F_0, \ldots, F_k)$ eine von einer* MD*-Reduktionsstrategie generierte Reduktionskette. Dann gilt:*

$$F_k \text{ irreduzibel} \iff \bigcup_{p \in \mathrm{Pos}(F_k)} M_k(p) = \emptyset$$

Beweis: "⇒": Zuerst soll der Fall $F_k = \top$ betrachtet werden. In diesem Fall ist F_k offensichtlich irreduzibel. Für $k = 0$ ist sofort ersichtlich, daß $M_0(\top_\lambda) = \emptyset$. Ist $k > 0$, so muß F_{k-1} die Form $A \vee \top$, $L \vee \neg L$ oder $\top \wedge \top$ angenommen haben, und die Reduktion zur Formel F_k wurde entsprechend mit $(R1)$, $(R2)$ oder $(R3)$ erreicht. In allen Fällen findet man anhand der Definition von M', daß $M'_k(\top_\lambda) = \emptyset$ und damit $M_k(\top_\lambda) = \emptyset$. Insgesamt gilt also für $F_k = \top$, daß $\bigcup_{p \in \mathrm{Pos}(F_k)} M_k(p) = M_k(\top_\lambda) = \emptyset$.

Betrachten wir nun ein beliebiges irreduzibles $F_k \neq \top$. Dieses F_k ist dann quantorenfrei, enthält kein Verum $(\top)$ und ist in konjunktiver Normalform.[7] Die einzigen Symbole

[7] Eine Formel in konjunktiver Normalform kann in diesem Zusammenhang auch das Symbol $\bot$ enthalten.

in F_k, die nicht-leere Markierungen besitzen können, sind daher (positive) Literale und das Konjunktionssymbol $\wedge$. Man beachte dazu, daß von der Funktion M' keine Symbole markiert werden, die zuvor eine leere Markierung besaßen. Markierte Literale können in F_k jedoch nicht auftreten, da MD-Reduktionsstrategien Disjunktionen aus positivem und negativem Literal nie auseinandertrennen. Ein markiertes Konjunktionssymbol könnte in F_k nur auftreten, wenn ein Subformelvorkommen $A \vee (B \wedge C)$ zu $B \wedge C$ reduziert werden könnte, was in RPC_n ebenfalls unmöglich ist. Also gilt die Behauptung auch in diesem Fall.

"$\Leftarrow$": Sei $\bigcup_{p \in \mathrm{Pos}(F_k)} M_k(p) = \emptyset$. Angenommen F_k sei reduzibel. Dann gibt es ein $p' \in \mathrm{Pos}(F_k)$ und ein $F' \in \mathcal{F}_n$, so daß $F_k \longrightarrow_{\mathrm{RPC}} F'$ eine p'-Reduktion ist. Für dieses p' gilt (unabhängig von der Definition von M'), daß $M_k(p') \neq \emptyset$, also auch $\bigcup_{p \in \mathrm{Pos}(F_k)} M_k(p) \neq \emptyset$. Dies ist ein Widerspruch zur Voraussetzung, also muß F_k irreduzibel sein. □

Satz 4.7 *Sei* RS *eine* MD*-Reduktionsstrategie. Falls zu jeder von* RS *generierten Reduktionskette* $R = (F_0, \ldots, F_l)$ *ein* $k \in \mathbb{N}$ *existiert, so daß* $RS^k(R)$ *definiert ist und für* $RS^k(R) = (F_0, \ldots, F_l, \ldots, F_{l+k})$ *gilt:*

$$\forall l' \leq l \;\; \forall t \in \mathcal{T}_n \cup \{\Box\} \;\; : \;\; (l', t) \;\notin \bigcup_{p \in \mathrm{Pos}(F_{l+k})} M_{l+k}(p),$$

so ist RS *eine zuverlässige Reduktionsstrategie.*

Intuitiv besagt dieser Satz, daß eine Reduktionsstrategie, in der alle alten Reduktionsmöglichkeiten irgendwann einmal zu einer Reduktion verwendet werden, zuverlässig ist.

Ein Beweis des vorhergehenden Satzes ist leider bisher nicht gelungen. Die Idee hinter diesem Satz und Indizien für dessen Gültigkeit sollen trotzdem kurz angegeben werden. Die Termination des booleschen Teils des RPC_n-Kalküls kann leicht gezeigt werden; bleiben also noch die Quantorenregeln. Diese sind auch dafür verantwortlich, daß das Kalkül nicht terminiert. Allerdings haben sie die Eigenschaft, daß bestehende Teilformeln erhalten bleiben in dem Sinn, daß neue Teilformeln nur bestehende Disjunktionen verlängern. Eine alte Reduktionsmöglichkeit geht daher nicht verloren und kann auch später noch ausgeführt werden. Allerdings kann eine ungeschickte Anwendung des Distributivgesetzes unter Umständen die Beweislänge verdoppeln, falls in $(R4)$ eine Ableitung in A zu einem kürzestmöglichen Beweis führen würde. Nach Anwendung von $(R4)$ ist A doppelt vorhanden, wodurch nötige Beweisschritte in A eventuell zweimal vorgenommen werden müssen. Auch ohne Beweis wollen wir uns nun speziellen Varianten von Reduktionsstrategien zuwenden.

Satz 4.7 liefert zwar ein Kriterium, um die Zuverlässigkeit einer gegebenen Reduktionsstrategie zu überprüfen, sagt aber nichts über die Existenz einer solchen aus. In den folgenden Abschnitten soll daher gezeigt werden, daß es zuverlässige Reduktionsstrategien gibt.

4.5 Die ORP-Reduktionsstrategie

Eine ORP-Reduktionsstrategie (oldest reduction possibility) wählt den nächsten Reduktionsschritt aus der Menge der am längsten bestehenden Reduktionsmöglichkeiten aus. Diese Strategie, von der gezeigt werden soll, daß sie zuverlässig ist, wird nun definiert.

Definition 4.8 (ORP**-Reduktions-Strategie)** *Eine* ORP*-Reduktions-Strategie* ORP : $\mathcal{F}^* \longrightarrow \mathcal{F}^*$ *ist eine* MD*-Reduktionsstrategie mit zugeordneter Symbolmarkierungsfamilie* M*, wobei in*

$$\mathrm{ORP}(F_0, \ldots, F_l) = \begin{cases} (F_0, \ldots, F_l, F_{l+1}) & \textit{falls } F_l \textit{ reduzibel} \\ \textit{undefiniert} & \textit{falls } F_l \textit{ irreduzibel} \end{cases}$$

F_{l+1} *dadurch bestimmt ist, daß* $F_l \longrightarrow_{\mathrm{RPC}} F_{l+1}$ *eine* (p,t)*-Reduktion mit* $(k_{\min}, t) \in M_l(p)$ *ist; dabei ist*

$$k_{\min} = \min\Big\{ k \;\Big|\; \exists t' \in \mathcal{T}_n \cup \{\Box\} : (k,t') \in \bigcup_{p' \in \mathrm{Pos}(F_l)} M_l(p') \Big\}.$$

Lemma 4.9 *Sei* RS *eine* ORP*-Reduktionsstrategie,* $R = (F_0, \ldots, F_l)$ *eine beliebige von* RS *generierte Reduktionskette,* F_l *reduzibel und*

$$k_{\min} = \min\Big\{ k \;\Big|\; \exists t \in \mathcal{T}_n \cup \{\Box\} : (k,t) \in \bigcup_{p' \in \mathrm{Pos}(F_l)} M_l(p') \Big\}.$$

Dann gibt es ein $k \in \mathbb{N}$*, so daß* $RS^k(R)$ *definiert ist und für* $RS^k(R) = (F_0, \ldots, F_l, \ldots, F_{l+k})$ *gilt:*

$$\forall k' \leq k_{\min} \;\; \forall t \in \mathcal{T}_n \cup \{\Box\} : (k',t) \notin \bigcup_{p \in \mathrm{Pos}(F_{l+k})} M_{l+k}(p) \Big\}.$$

Anmerkung: Das Minimum $k_{\min}$ ist wohldefiniert, da nach Lemma 4.6 aus der Reduzibilität von F_l folgt, daß $\bigcup_{p \in \mathrm{Pos}(F_l)} M_l(p) \neq \emptyset$.

Beweis: Sei RS eine ORP-Reduktionsstrategie, $R = (F_0, \ldots, F_l)$ eine von dieser Strategie generierte Reduktionskette, F_l reduzibel und $k_{\min}$ definiert wie im Lemma. Wir betrachten nun erweiterte, von RS generierte Reduktionsketten $R' = (F_0, \ldots, F_l, \ldots)$ und definieren für $i \geq l$ eine Familie von Gewichtungsfunktionen $w_i : \mathcal{F}_n \longrightarrow \mathbb{N}$ rekursiv durch:

$$\begin{aligned}
w_i(C_{\top,p}) &= \begin{cases} 2 & \text{falls } M_i(p) = \{(k_{\min}, \Box)\} \\ 1 & \text{sonst} \end{cases} \\
w_i(C_{\bot,p}) &= 1 \\
w_i(C_{L,p}) &= \begin{cases} 2 & \text{falls } M_i(p) = \{(k_{\min}, \Box)\} \\ 1 & \text{sonst} \end{cases} \\
w_i(C_{\neg L,p}) &= 1 \\
w_i(C_{\vee,p}(F_0, \ldots, F_r)) &= \prod_{0 \leq j \leq r} w_i(F_j) \\
w_i(C_{\wedge,p}(F_0, \ldots, F_r)) &= \begin{cases} 2 \cdot \sum_{0 \leq j \leq r} w_j(F_j) & \text{falls } M_i(p) = \{(k_{\min}, \Box)\} \\ \sum_{0 \leq j \leq r} w_i(F_j) \;-\; r & \text{sonst} \end{cases} \\
w_i(C_{\exists v,p}(F)) &= \Big|\Big\{(k,t) \in M_i(p) \mid k = k_{\min}\Big\}\Big| \;+\; w_i(F) \\
w_i(C_{\forall v,p}(F)) &= \Big|\Big\{(k,t) \in M_i(p) \mid k = k_{\min}\Big\}\Big| \;+\; w_i(F)
\end{aligned}$$

Jedes w_i weist der entsprechenden Formel F_i der erweiterten Reduktionskette ihr Gewicht $w_i(F_i)$ zu. Damit zwei Formeln, die modulo Assoziativität und Kommutativität gleich

sind, dasselbe Gewicht erhalten, seien in allen F_i die Konjunktionen flach dargestellt, d.h. Subterme von F_i der Form $C_\wedge(S_0,\ldots,S_{j-1},C_\wedge(T_0,\ldots,T_s),S_{j+1},\ldots,S_r)$ werden als $C_\wedge(S_0,\ldots,S_{j-1},T_0,\ldots,T_s,S_{j+1},\ldots,S_r)$ verstanden.

Man überprüft leicht, daß alle Funktionen w_i die folgenden Eingenschaften erfüllen:

1. $w_i(F_i) \geq 1$
2. $w_i(F_i) = 1 \iff \forall t \in \mathcal{T}_n \cup \{\Box\} : (k_{\min}, t) \notin \bigcup_{p\in \mathrm{Pos}(F_i)} M_i(p)$
   $\iff \forall k' \leq k_{\min}\ \forall t \in \mathcal{T}_n \cup \{\Box\} : (k', t) \notin \bigcup_{p\in \mathrm{Pos}(F_i)} M_i(p)$

Die letzte Äquivalenz ergibt sich aus den Definitionen von M, M' und $k_{\min}$.

Bleibt also zu zeigen, daß $w_i(F_i) > w_{i+1}(F_{i+1})$ falls $w_i(F_i) > 1$. Dazu nehmen wir an, daß $F_{i+1} = F_i[q \leftarrow G/H]$. Zuerst wollen wir zeigen, daß $w_i(G_q) > w_{i+1}(H_q)$. Symbole S_p, für die es ein t gibt, so daß $(k_{\min}, t) \in M_i(p)$, werden im folgenden durch Unterstreichen hervorgehoben. Fallunterscheidung nach der Regel (R_i), deren Instanz $G \longrightarrow H$ ist, liefert:

(R1): $w_i(A \vee \underline{\top}) = w_i(A) \cdot w_i(\underline{\top}) = 2 \cdot w_i(A) > 1 = w_{i+1}(\top)$

(R2): $w_i(\underline{L} \vee \neg L) = w_i(\underline{L}) \cdot w_i(\neg L) = 2 > 1 = w_{i+1}(\top)$

(R3): $w_i(A \wedge \underline{\top}) = w_i(A) + w_i(\underline{\top}) - 1 = w_i(A) + 1 > w_i(A) = w_{i+1}(A)$ und $w_i(A \underline{\wedge} \underline{\top}) = 2 \cdot (w_i(A) + w_i(\underline{\top})) = 2 \cdot w_i(A) + 4 > w_i(A) = w_{i+1}(A)$

(R4): $w_i(A \vee (B \underline{\wedge} C)) = 2 \cdot w_i(A) \cdot (w_i(B) + w_i(C)) = 2 \cdot w_i(A)w_i(B) + 2 \cdot w_i(A)w_i(C) > w_i(A)w_i(B) + w_i(A)w_i(C) - 1 = w_{i+1}(A)w_{i+1}(B) + w_{i+1}(A)w_{i+1}(C) - 1 = w_{i+1}((A \vee B) \wedge (A \vee C))$

(R5): $w_i(\underline{\exists} x A) = K + w_i(A) > K - 1 + w_i(A) = K - 1 + w_{i+1}(A) = w_{i+1}(\exists x A \vee A[x/a])$

(R6): $w_i(\underline{\forall} x A \vee B) = (K + w_i(A)) \cdot w_i(B) > (K - 1 + w_i(A)) \cdot w_i(B) = (K - 1 + w_{i+1}(A)) \cdot w_{i+1}(B) = w_{i+1}(\forall x A \vee B \vee A[\div x] \vee \forall y(\forall y B \vee A[x \div y][x/y]))$

(R6'): $w_i(\underline{\forall} x A) = K + w_i(A) > K - 1 + w_i(A) = K - 1 + w_{i+1}(A) = w_{i+1}(\forall x A \vee A[\div x] \vee \forall y A[x \div y][x/y])$

In den Fällen $(R5)$, $(R6)$ und $(R6')$ ist dabei $K = \left|\left\{(k,t) \in M_i(p) \mid k = k_{\min}\right\}\right|$.

Da $w_i(F) > w_i(F[A/B])$ falls $w_i(A) > w_i(B)$, und da die Funktion M_{i+1} nur neue Markierungen mit erster Komponente $i+1$ erzeugt, erhält man aus $w_i(G_q) > w_{i+1}(H_q)$ sofort $w_i(F_i) > w_{i+1}(F_{i+1})$ falls $w_i(F_i) > 1$. Es gibt daher ein $k \in \mathbb{N}$, für das $w_{l+k}(F_{l+k}) = 1$ gilt. Zusammen mit der zweiten Eigenschaft von w_i ergibt sich die Behauptung. □

Satz 4.10 *Jede* ORP*-Reduktionsstrategie ist zuverlässig.*

Beweis: Sei RS eine ORP-Reduktionsstrategie und $R = (F_0,\ldots,F_l)$ eine beliebige von RS generierte Reduktionskette. Wir definieren nun der kürzeren Schreibweise wegen für $s \in \mathbb{N}$:

$$K(s) = \bigcup_{p\in \mathrm{Pos}(F_s)} M_s(p) \qquad \text{und} \qquad k_{\min}(s) = \min\left\{ k \;\middle|\; \exists t \in \mathcal{T}_n \cup \{\Box\} : (k,t) \in K(s)\right\}.$$

Mittels Lemma 4.9 läßt sich sukzessive eine Folge $(k_1,\ldots,k_r) \in \mathbb{N}^r, r \in \mathbb{N}$ mit assoziierten Reduktionsketten $(R_1,\ldots,R_r)$ konstruieren, so daß:

1. $R_i = (F_0, \ldots, F_{l+\sum_{j=1}^{i} k_j})$ für $0 \leq i \leq r$, also insbesondere $R_0 = R$,
2. $\forall k' \leq k_{\min}(l + \sum_{j=1}^{i-1} k_j) \;\; \forall t \in \mathcal{T}_n \cup \{\Box\} : (k', t) \notin K(l + \sum_{j=1}^{i} k_j)$ für $1 \leq i \leq r$ und
3. $K(l + \sum_{j=1}^{r} k_j) = \emptyset$ oder $k_{\min}(l + \sum_{j=1}^{r-1} k_j) > l$.

Es gibt immer ein $r \in \mathbb{N}$ mit der letztgenannten Eigenschaft, denn es gilt

$$k_{\min}(l + \sum_{j=1}^{i} k_j) > k_{\min}(l + \sum_{j=1}^{i-1} k_j) \;\text{ für }\; 1 \leq i \leq r.$$

Anzumerken ist noch, daß falls $F_{l+\sum_{j=1}^{i} k_j}$ irreduzibel ist, nach Lemma 4.6 $K(l+\sum_{j=1}^{i} k_j) = \emptyset$ gilt. Also ist die Voraussetzung von Lemma 4.9 (Reduzibilitat) bei jeder Verlängerung der Folge erfüllt.

Wir setzen nun $k = \sum_{j=1}^{r} k_j$ und unterscheiden die beiden Fälle, die für das letzte Folgenglied k_r bzw. R_r gelten. Ist $K(l + k) = \bigcup_{p \in \mathrm{Pos}(F_{l+k})} M_{l+k}(p) = \emptyset$, so liefert Satz 4.7 mit dem soeben definierten k die Behauptung. Ansonsten gilt die zweite Abbruchbedingung, d.h. es ist $k_{\min}(l + \sum_{j=1}^{r-1} k_j) > l$. Damit erhält man

$$\forall k' \leq l \;\; \forall t \in \mathcal{T}_n \cup \{\Box\} : (k', t) \notin K(l + k)$$

als Spezalisierung der zweiten Eigenschaft der Folge für $i = r$. Satz 4.7 liefert nun auch hier die Behauptung. □

Korollar 4.11 *Es gibt für jedes $n \leq \omega$ eine zuverlässige Reduktionsstrategie für* RPC_n.

Beweis: Die ORP-Reduktionsstrategie ist fur jedes $n \in \mathbb{N}$ und für $n = \omega$ definiert. Man beachte dazu, daß die Markierungsfunktionen M und M' auch für $n = \omega$ definiert sind. Nach Satz 4.10 ist ORP unabhängig von n eine zuverlässige Reduktionsstrategie, □

4.6 Verallgemeinerung der ORP-Reduktionsstrategie

Wie zuletzt gezeigt wurde, sind alle ORP-Reduktionsstrategien zuverlässig. Allerdings lassen diese keinen großen Spielraum in der Auswahl des nächsten Beweisschrittes. Deshalb soll nun eine allgemeinere Klasse von Reduktionsstrategien auf der Grundlage der ORP-Reduktionsstrategien entwickelt werden, bei denen die Zuverlässigkeit natürlich erhalten bleiben soll.

Ohne explizite Formulierung kann folgende Strategie durch leichte Änderung der ORP-Strategie verwendet werden: Anstatt in jedem Beweisschritt die älteste Reduktionsmöglichkeit auszuwählen, wird nur in jedem n-ten Schritt die älteste ausgewählt. Die dazwischenliegenden Schritte können beliebig vorgenommen werden. Auch mit dieser Strategie verschwinden nach und nach alte Reduktionsmöglichkeiten, wie durch eine Modifikation des letzten Beweises gezeigt werden kann.

Dadurch kann man eine beliebige Reduktionsstrategie RS mit der ORP-Strategie mischen. In jedem n-ten Beweisschritt wird die ORP-Strategie verwendet, in allen anderen die Strategie RS. Damit kann jede beliebige Reduktionsstrategie RS zu einer zuverlässigen gemacht werden.

Die nun folgende, effizientere LP-Reduktionsstrategie macht Gebrauch von dieser Tatsache. Wir brauchen uns also im folgenden keine Gedanken über Zuverlässigkeit machen, da durch Mischen mit der ORP-Strategie dies immer erreicht werden kann.

4.7 Die LP-Reduktionsstrategie

Die Idee der Literalpaar-Reduktionsstrategie (LP) besteht in dem Versuch, die Auswahl einzelner Reduktionssschritte gezielt vorzunehmen. Dazu wird ein Literal $L = P(t_0, \ldots, t_k)$ und ein zu L komplementäres Literal $L' = \neg P(s_0, \ldots, s_k)$ ausgewählt. Dieses versucht die Strategie dann durch gezielte Reduktionen zum "verschwinden" zu bringen. Dabei werden sowohl die Reduktionspositionen als auch die Reduktionsparameter (a und y) so gewählt, daß die Literale sich "aufeinander zu bewegen". Dieses informelle Konzept soll nun genauer untersucht werden.

Definition 4.12 (Literalpaar) *Sei F eine Formel und $p, p' \in \mathrm{Pos}(F)$, wobei $F(p) = L = Q(t_0, \ldots, t_k)$, $F(p') = L' = \neg Q(s_0, \ldots, s_k)$ für ein Prädikatensymbol Q und Terme $t_0, \ldots, t_k, s_0, \ldots, s_k$. Außerdem seien die beiden komplementären Literale L und L' durch ein Disjunktionssymbol verbunden, d.h. $F(r) = \vee$, wobei r der größte gemeinsame Präfix von p und p' ist. Dann ist (L, L') ein Literalpaar von F an der Position (p, p').*

Wir wollen nun für ein Literalpaar (L, L') die Teilformel beschreiben, die nur diese beiden Literale als atomare Formeln enthält.

Definition 4.13 (Literalpaar-Subformel) *Sei (L, L') ein Literalpaar einer Formel F. Die Literalpaar-Subformel von (L, L') entsteht aus F durch Verkürzung von Konjunktionen und Disjunktionen.*

1. *$(A_0 \wedge \ldots \wedge A_k)$ wird zu A_i, falls $L \trianglelefteq A_i$ oder $L' \trianglelefteq A_i$.*
2. *$(A_0 \vee \ldots \vee A_k)$ wird zu $A_i \vee A_j$, falls $L \trianglelefteq A_i$ und $L' \trianglelefteq A_j$.*
3. *$(A_0 \vee \ldots \vee A_k)$ wird zu A_i, falls $L \trianglelefteq A_i$ oder $L' \trianglelefteq A_i$ und der vorhergehende Fall trifft nicht zu.*

Beispiel: Sei

$$F = \forall x(\exists y(Py \vee (Qx \wedge R)) \vee \neg Px).$$

Dann ist $(L, L') = (Py, \neg Px)$ ein Literalpaar von F. Die Literalpaar-Subformel von (L, L') ist dann

$$F_S = \forall x(\exists y Py \vee \neg Px).$$

Literalpaar-Subformeln haben die in Abbildung 1 dargestellte Form.

Der Pfad von λ zum Disjunktionssymbol der Literalpaar-Subformel wird *Wurzelpfad*, der von dem Disjunktionssymbol zum positiven Literal *P-Pfad*, und der zum negativen Literal *N-Pfad* genannt.

Als nächstes wollen wir ein Kriterium entwickeln, unter dem ein Literalpaar-Vorkommen zum verschwinden gebracht werden kann. Dazu wird anhand der Literalpaar-Subformel

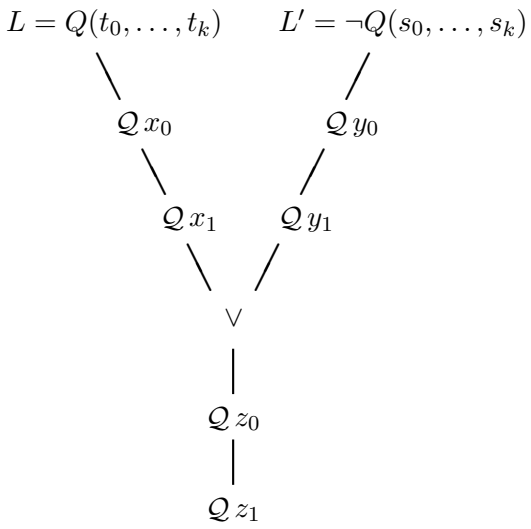

Abbildung 1: Schematische Darstellung der Literalpaar-Subformel

ein Gleichungssystem generiert, das genau dann lösbar ist, wenn das Literalpaar eliminiert werden kann. Die einzelnen Schritte von einer Literalpaar-Subformel zu dem ihr zugeordneten Gleichungssystem soll nun beschrieben werden. Grob gliedert sich das Verfahren in vier Schritte. Wir gehen von einer Literalpaar-Subformel F_S aus.

1. Generierung des universellen Abschlusses von F_S.
2. Umbenennung von gebundenen Variablen in F_S, so daß keine Variable von zwei Quantoren gebunden wird.
3. Eliminieren der $\forall$-Quantoren durch Skolemisierung: Dieser Schritt generiert Funktionssymbole f_i.
4. Berechnung des Gleichungssystem aus den Literaltermen: Für jedes Termpaar t_i, s_i bilde Gleichung $t_i = s_i$, wobei Variablen in t_i und s_i unterschieden werden.

Jeder dieser Schritte wird nur zur Generierung des Gleichungssystem benötigt. Die entstehende Formel in Schritt 3 muß daher nicht im RPC-Formalismus darstellbar sein (Funktionssymbole, neue Variable). Die Schritte 3 und 4 sollen nun näher beleuchtet werden.

In Schritt 3 wird für jede $\forall$-quantifizierte Variable x ein neues Funktionssymbol f_x eingeführt. Die Stelligkeit von f_x hängt von den weiter aussen liegenden $\exists$-Quantoren ab. So wird in der Formel $\exists x_0 \ldots x_i \forall y \ldots$ ein $(i+1)$-stelliges Funktionssymbol für y generiert, das von den $\exists$-quantifizierten Variablen abhängt. Also wird y in den Literalen ersetzt durch $f_y(x_0, \ldots, x_i)$. Das Verfahren ist dual zu dem bei Resolution verwendeten Skolemisierungsverfahren. In Schritt 4 werden die durch Schritt 3 modifizierten Literalterme paarweise gleichgesetzt, also $t'_i = s'_i$, wobei t'_i und s'_i die modifizierten Terme darstellen. Variablen im positiven und negativen Literal müssen dabei unterschieden werden. Daher werden Variablen im negativen Literal durch einen Balken gekennzeichnet: $\bar{x}$.

Ein Beispiel soll den gesamten Prozeß verdeutlichen.

Beispiel: Die Literalpaar-Subformel sei $F_S = \exists x(\forall y Pxy \vee \exists z \neg Pyz)$. Dann sehen die vier Schritte wie folgt aus:

1. Universeller Abschluß: $\forall y \exists x(\forall y Pxy \vee \exists z \neg Pyz)$
2. Umbenennen: $\forall y \exists x(\forall u Pxu \vee \exists z \neg Pyz)$
3. Skolemisierung: $\exists x(Pxf_u(x) \vee \exists z \neg Pf_y z)$
4. Gleichungssystem:

$$\begin{aligned} x &= f_y \\ f_u(x) &= \bar{z} \end{aligned}$$

Das Auflösen der hier auftretenden Gleichungssysteme geschieht mit den bekannten Methoden. Wir wollen hier nur definieren, wie ein System in gelöster Form aussieht.

Definition 4.14 *Ein Gleichungssystem* $\{s_0 = t_0, \ldots, s_k = t_k\}$ *ist in gelöster Form, falls*

1. s_i *ist eine Variable für alle* i *und*
2. $s_i \neq s_j$ *für alle* i, j *mit* $i \neq j$ *und*
3. s_i *tritt nicht in* t_j *auf für alle* i, j.

Wenn für ein Literalpaar ein lösbares Gleichungssystem generiert werden konnte, so interessiert natürlich, welche RPC-Reduktionen zur Elimination dieses Literalpaars angewandt werden müssen. Diese Schritte werden in drei Phasen eingeteilt:

1. **Vorbereitende Reduktion auf dem Wurzelpfad:** Löst die Fälle auf, in denen Gleichungen einer der beiden folgenden Formen in der Lösung auftreten:
   (a) $\{x = t(\bar{x})\}$; die komplementäre Variable $\bar{x}$ tritt in der rechten Seite der Gleichung für x auf.
   (b) $\{x = t_1, \bar{x} = t_2\}$ mit $t_1 \neq t_2$; Variable und komplementäre Variable treten in Gleichungen mit verschiedenen rechten Regelseiten auf.
2. **Reduktionen auf P- bzw. N-Pfad:** Bewegt Literale aufeinander zu durch:
   (a) Anwendung des Distributivgesetzes ($R4$)
   (b) Überspringen von $\exists$-Quantoren ($R5$)
   (c) Überspringen von $\forall$-Quantoren ($R6$ mit komplementärem Literal in B bzw. $R6'$)
3. **Abschließende Reduktionen auf dem Wurzelpfad:** Anpassen der freien Variablen durch Anwendung von ($R5$).

Wir betrachten die drei Punkte nun genauer:

1. Die beiden Fälle können nur dann auftreten, wenn in der Literalpaar-Subformel ein $\exists$-Quantor auf dem Wurzelpfad vorkommt. Anwendung der Regel ($R5$) auf den am weitest außen liegenden $\exists$-Quantor liefert dann:

$$\exists x A[L, L'] \longrightarrow_{\mathrm{RPC}} \exists x A[\underline{L}, L'] \vee A^*[L^*, \underline{L'^*}]$$

   Das neue Literalpaar ist durch Unterstreichung hervorgehoben, durch * werden mögliche Variablenumbenennungen angedeutet. Das dabei entstandene Gleichungssystem für das Literalpaar (L, L'^*) bleibt lösbar, enthält aber keinen der beiden oben angegebenen Fälle mehr.

2. In dieser Phase werden die Literale aufeinander zu bewegt. Dazu werden verschiedene Reduktionsmuster verwendet. Alle Reduktionen finden entweder auf dem N- oder auf dem P-Pfad statt und werden von außen nach innen hin vorgenommen . Ob auf dem N- oder P-Pfad die nächste Reduktion stattfindet hängt von Prioritätsregeln ab. $\forall$-Quantoren werden dabei bevorzugt behandelt. Eines der verwendeten Reduktionsmuster, das bei $\forall$-Quantoren angewandt wird, sieht wie folgt aus:

$$\begin{aligned} \forall x A[L] \vee B[L'] &\xrightarrow[\mathrm{RPC}]{(R6)} \ldots \vee \forall y(\forall y B[L'] \vee A[L][x \div y][x/y]) \\ &\xrightarrow[\mathrm{RPC}]{(R6)} \ldots \vee \forall y(\ldots \vee B[L'] \vee A[L][x \div y][x/y]) \end{aligned}$$

   In der letzten Formel sind die Literale L und L' nicht mehr durch den $\forall$-Quantor getrennt. y muß dabei so gewählt werden, daß $y \notin \mathrm{Fr}(L')$ und darüberhinaus $y \notin \mathrm{Fr}(L)$ oder $y = x$. Weitere gebundene Variablen auf dem N- oder P-Pfad erlauben noch weitere Möglichkeiten zur Wahl von y.

   Am Ende der zweiten Phase sind die Literale direkt disjunktiv verknüpft, also $L \vee L'$. Eventuell unterschiedliche Terme in L und L' werden in der abschließenden dritten Phase behandelt.

3. Von innen nach außen werden nun durch passende ($R5$)-Anwendungen die Terme angepaßt.

Jeder dieser Schritte ist möglich, die Lösbarkeit des ursprünglichen Gleichungssystems vorausgesetzt. Die Lösbarkeit bleibt darüberhinaus für die transformierten Literalpaarvorkommen in jedem Schritt jeder Phase als Invariante erhalten.

Die LP-Reduktionsstrategie ermöglicht (zumindest empirisch) deutlich kürzere Beweise als die ORP-Reduktionsstrategie. Die Zuverlässigkeit von LP muß nicht sichergestellt werden, sofern man in regelmäßigen Abständen ORP-Reduktionsschritte einschiebt.

5 Implementation

Die im Rahmen dieser Studienarbeit entstandene Implementation eines RPC-Beweisers verwendet die funktionale Programmiersprache Haskell [PH96]. Haskell entstand 1987 aus einem Versuch, die bis dahin bestehenden nicht strikten, rein funktionalen Programmiersprachen zu vereinheitlichen. Die derzeit aktuelle Version 1.4 wird von einer Reihe relativ effizienter Compiler unterstützt. Daneben existiert mit Hugs [Jon97] ein Interpreter für die Version 1.3 von Haskell. Der Quellcode des RPC-Beweisers kann nach wenigen kleinen Änderungen auch von Hugs interpretiert werden.

Bei dem implementierten RPC-Beweiser handelt es sich um keinen Proof-Checker, wie er in diesem Bereich des automatischen Beweisens relativ häufig zu finden ist. Stattdessen arbeitet der RPC-Beweiser größtenteils vollautomatisch. Nach dem Einlesen der zu beweisenden Formel und deren eventueller Vorverarbeitung kann eine Beweisstrategie, die Anzahl der für den Beweis zu verwendenden Variablen und die maximale Beweislänge festgelegt werden. Dann versucht der Beweiser mit der angegebenen Strategie einen Beweis innerhalb der maximalen Schrittzahl zu finden.

Die Implementation enthält sowohl die ORP- als auch die LP-Reduktionsstrategie, jeweils mit etlichen Variationsmöglichkeiten. Für die praktische Anwendung hat sich die ORP-Strategie als eher unhandlich erwiesen. Zur Sicherstellung der Zuverlässigkeit der LP-Reduktionsstrategie leistet sie jedoch wertvolle Dienste.

Im weiteren wird zuerst das Eingabeformat des (interaktiven) Beweisers vorgestellt, danach wird auf besondere Implementationstechniken zur Beschleunigung eingegangen. Abschließend erlaubt ein Programmlauf des Beweisers einen ersten Einblick in Benutzerführung und Funktionalität.

5.1 Eingabeformat der Formeln

Das Format der Eingabeformeln wird durch die EBNF-Grammatik in Abbildung 2 beschrieben.

Dabei steht, wie üblich, $(\ )^*$ für beliebig oft wiederholte, $(\ )^+$ für mindestens einmal wiederholte, | für alternative und [] für optionale Strukturen.

Durch die Zuordnung von Nonterminalen zu den verschiedenen Konnektiven erhält man eine Abstufung der Präzedenz, die mit den üblichen Konventionen zur Einsparung von Klammern übereinstimmen. So binden beispielsweise Quantoren und Negationen stärker als Konjunktionen, die wiederum stärker binden als Disjunktionen.

Abweichend von der bekannten Notation sind Implikation, Disjunktion und Konjunktion aber rechtsassoziativ. Dadurch wird `P -> Q -> R` implizit äquivalent zu `P -> (Q -> R)` verstanden.

Die nachfolgenden Beispiele zeigen die Umsetzung einiger weiterer Formeln.

Beispiel:

1. Die Formel $\forall x \forall y \exists z(xPy \vee y \neg Pz)$ wird dargestellt durch

   ```
   (forall x,y)(exists z)(xPy \/ y~Pz).
   ```

2. Die Formel $\exists z(xQz \wedge \exists u(zRu \wedge uSy))$ kann beispielsweise ausgedrückt werden durch

Formula	→	*Implication* (“`<=>`” *Formula*)*
Implication	→	*Disjunction* (“`->`” *Implication*)*
Disjunction	→	*Conjunction* (“`\/`” *Disjunction*)*
Conjunction	→	*Quantified* (“`/\`” *Conjunction*)*
Quantified	→	“`(`” (“`exists`” | “`forall`”) *VariableList* “`)`” *Quantified*
		| “`~`” *Quantified*
		| *Primitive*
Primitive	→	*Atom* | “`(`” *Formula* “`)`”
Atom	→	“`true`” | “`false`”
		| *Predicate* “`(`” *TermList* “`)`”
		| *Predicate Terms*
		| *Term* [“`~`”] *Predicate Term*
TermList	→	*Term* (“`,`” *Term*)*
Terms	→	(*Term*)*
VariableList	→	*Variable* (“`,`” *Variable*)*
Term	→	*Variable* | *Constant*
Variable	→	(“`i`” ... “`z`”)+
Constant	→	(“`a`” ... “`h`” | “`0`” ... “`9`”)+
Predicate	→	(“`A`” ... “`Z`” | “`!`” | “`#`” | “`=`” | “`<`” | “`>`” | “`[`” | “`]`” | “`&`” | “`%`” | “`+`” | “`*`” | “`-`” | “`\`” | “`'`”)+

Abbildung 2: Syntax der Formeln des RPC-Beweisers

```
(exists z)(Q(x,z) /\ (exists u)(zRu /\ S u y)).
```

Dabei wurde jede der drei möglichen Schreibweisen für Literale verwendet.

5.2 Spezielle Verfahren zur Verkürzung von Beweisen

In dem RPC-Beweiser sind neben den beiden Strategien ORP und LP weitere spezielle Verfahren eingebaut, die die Beweissuche teilweise beschleunigen können. Diese werden jetzt kurz erläutert.

1. Priorität für verkürzende Regeln (d.h. $(R1)$, $(R2)$, $(R3)$): Formeln werden immer in $\{R1, R2, R3\}$-Normalform gehalten
2. Quantoren, die keine Variablen binden, werden entfernt, also:

$$\exists x A \longrightarrow A \text{ falls } x \notin \mathrm{Fr}(A)$$

$$\forall x A \longrightarrow A \text{ falls } x \notin \mathrm{Fr}(A)$$

3. Generieren von Teilbeweisen: $F = A \wedge B$ wird zerlegt in zwei Beweise für $F_1 = A$ und $F_2 = B$.
4. Zusätzliche $\forall$-Regel, falls genügend Variablen zur Verfügung stehen:

$$\forall x A \vee B \longrightarrow \forall y(A[x/y] \vee B) \text{ falls } y \notin \mathrm{Fr}(A) \cup \mathrm{Fr}(B)$$

5. Löschen von Literalen, die nur pos. oder neg. Vorkommen haben (Ersetzen durch $\perp$).
6. $F \vee \tilde{F}$ bzw. $F \wedge \tilde{F}$ werden durch F ersetzt, wenn F gebundene Umbenennung von F ist.
7. Vorverarbeitung der zu beweisenden Formel durch:
   (a) Anwenden des Distributivgesetzes zum Generieren von Teilbeweisen.
   (b) Minimierung der Quantorenbereiche:

$$\exists x(A \vee B) \longrightarrow \exists x A \vee \exists x B$$

$$\exists x(A \wedge B) \longrightarrow A \wedge \exists x B \text{ falls } x \notin A$$

$$\forall x(A \wedge B) \longrightarrow \forall x A \wedge \forall x B$$

$$\forall x(A \vee B) \longrightarrow A \vee \forall x B \text{ falls } x \notin A$$

   (c) Partielle Skolemiserung: $\forall$-quantifizierte Variablen, die nicht im Bereich eines $\exists$-Quantors liegen, werden durch neue Konstanten ersetzt und der $\forall$-Quantor entfernt (Tautologie-Eigenschaft bleibt erhalten).
   (d) Freie Variablen werden durch neue Konstanten ersetzt.

5.3 Beispielhafter Programmlauf

Wir wollen den Ablauf eines Beweises am Beispiel der Assoziativität der Relationskomposition vorstellen. D.h. wir wollen die Gültigkeit der folgenden Gleichung beweisen:

$$(P \odot Q) \odot R = P \odot (Q \odot R).$$

Übersetzung in Prädikatenlogik ergibt die Formel

$$\forall x \forall y \left(\exists z (\exists y (xPy \wedge yQz) \wedge zRy) \;\Leftrightarrow\; \exists z (xPz \wedge \exists x (zQx \wedge xRy)) \right).$$

Ein RPC-Beweiser-Programmlauf für diese Formel folgt nun.

```
------------------------------------------------------------------------------
----  Welcome to the RPC prover, version 0.1, (c) 1996-97 by Carsten Sinz.  ----
------------------------------------------------------------------------------

You have the following choice:
  (r) read formula
  (f) preprocess current formula
  (o) print / change proof options
  (c) show current formula
  (g) show generated subgoals
  (s) print formula statistics
  (p) start proof / next subgoal proof
  (q) quit
Selection (r/f/o/c/g/s/p/q)? r

Read input formula from file or standard input (f/s)? s
Enter formula:
(forall x)(forall y)((exists z)((exists y)(xPy /\ yQz) /\ zRy) <=>  (exists z
)(xPz /\ (exists x)(zQx /\ xRy)));

Input formula read successfully, converted to negation normal form,
simplified and added to subgoal list.

You have the following choice:
  (r) read formula
  (f) preprocess current formula
  (o) print / change proof options
  (c) show current formula
  (g) show generated subgoals
  (s) print formula statistics
  (p) start proof / next subgoal proof
  (q) quit
Selection (r/f/o/c/g/s/p/q)? f

Perform partial skolemization (y/n)? y
Subgoal generation (y/n)? y
  Partial, complete or recursive subgoal generation (p/c/r)? c
Generated 2 subgoal(s).

You have the following choice:
  (r) read formula
  (f) preprocess current formula
  (o) print / change proof options
  (c) show current formula
```

```
  (g) show generated subgoals
  (s) print formula statistics
  (p) start proof / next subgoal proof
  (q) quit
Selection (r/f/o/c/g/s/p/q)? g

1) origin: generation 0, step 0
   ~P(c_0,c_3) \/ ~Q(c_3,c_2) \/ ~R(c_2,c_1) \/ (exists x)(P(c_0,x) /\
   (exists y)(Q(x,y) /\ R(y,c_1)))
2) origin: generation 0, step 0
   ~Q(c_2,c_3) \/ ~R(c_3,c_1) \/ ~P(c_0,c_2) \/ (exists x)(R(x,c_1) /\
   (exists y)(P(c_0,y) /\ Q(y,x)))

You have the following choice:
  (r) read formula
  (f) preprocess current formula
  (o) print / change proof options
  (c) show current formula
  (g) show generated subgoals
  (s) print formula statistics
  (p) start proof / next subgoal proof
  (q) quit
Selection (r/f/o/c/g/s/p/q)? c

Simple output or layout (s/l)? l

\/
   ~P(c_0,c_3)
   ~Q(c_3,c_2)
   ~R(c_2,c_1)
   exists x
      /\
         P(c_0,x)
         exists y
            /\
               Q(x,y)
               R(y,c_1)

You have the following choice:
  (r) read formula
  (f) preprocess current formula
  (o) print / change proof options
  (c) show current formula
  (g) show generated subgoals
  (s) print formula statistics
  (p) start proof / next subgoal proof
  (q) quit
Selection (r/f/o/c/g/s/p/q)? p

Prove current formula or all subgoals (f/a)? a
Enter maximal number of variables: 4
Enter maximal number of proof steps: 100
Choose a major reduction strategy:
  (l) literal pair reduction strategy (incomplete!)
  (o) oldest reduction possibility strategy
  (m) mix both strategies
  (a) lit. pair. red. strat. with priority to AK3
Selection (l/o/m/a)? l
Initialization of literal pair reduction strategy:
```

```
Choose a literal pair selector:
  (d) minimal position distance
  (l) least recently used literals
  (i) interactive literal selection
  (s) super literal selector
  (f) first literal
Selection (d/l/i/s/f)? d
Choose a disjunction selector type:
  (i) minimal disjunction
  (l) matching literal selector
  (a) maximal disjunction
Selection (i/l/a)? i
Choose a term selector:
  (d) default term selector
  (i) interactive term selector
  (s) super term selector
Selection (d/i/s)? d
------------------------------------------------------------------------------
Starting RPC-proof with 4 variable(s).

started literal unification step #1
  selected literal pair: P(c_0,x), ~P(c_0,c_3)
    @ positions: [3.0.0], [0]
  literal unification equation system solved
  preprocessing
  main unification process EC
  postprocessing
reduction step #1 completed, 2 RPC-reduction(s), 2 in total.

started literal unification step #2
  selected literal pair: R(y,c_1), ~R(c_2,c_1)
    @ positions: [3.0.1], [1]
  literal unification equation system solved
  preprocessing
  main unification process EC
  postprocessing
reduction step #2 completed, 2 RPC-reduction(s), 4 in total.

***** SUBGOAL  P R O V E D *****

Statistics of subgoal proof:
  RPC-reductions performed:         4
  major reduction steps:            2
  CPU time used for subgoal proof:  20 ms

1 subgoal(s) left to prove.
------------------------------------------------------------------------------
Starting RPC-proof with 4 variable(s).

started literal unification step #1
  selected literal pair: R(x,c_1), ~R(c_3,c_1)
    @ positions: [3.0.0], [1]
  literal unification equation system solved
  preprocessing
  main unification process EC
  postprocessing
reduction step #1 completed, 2 RPC-reduction(s), 2 in total.

started literal unification step #2
```

```
  selected literal pair: Q(y,c_3), ~Q(c_2,c_3)
    @ positions: [3.0.1], [0]
  literal unification equation system solved
  preprocessing
  main unification process EC
  postprocessing
reduction step #2 completed, 2 RPC-reduction(s), 4 in total.

***** SUBGOAL  P R O V E D *****

Statistics of subgoal proof:
  RPC-reductions performed:         4
  major reduction steps:            2
  CPU time used for subgoal proof:  39 ms

0 subgoal(s) left to prove.

--------------------------------------------------------------------------------
*****  P R O O F  C O M P L E T E D  *****

Statistics of complete proof:
  RPC-reductions performed:         8
  major reduction steps:            4
  CPU time used for complete proof: 59 ms
--------------------------------------------------------------------------------

You have the following choice:
  (r) read formula
  (f) preprocess current formula
  (o) print / change proof options
  (c) show current formula
  (g) show generated subgoals
  (s) print formula statistics
  (p) start proof / next subgoal proof
  (q) quit
Selection (r/f/o/c/g/s/p/q)? q

Good bye!
```

6 Ergebnisse

Um die Leistungsfähigkeit des RPC-Beweisers festzustellen, wurden einige Experimente mit verschiedenen Formeln vorgenommen. Zum größten Teil stammen sie aus der Problemsammlung von Pelletier [Pel86].

Die folgenden Beispiele (Tabelle 1) umfassen einfache Probleme der Aussagenlogik, der monadischen Prädikatenlogik und der vollen Prädikatenlogik, allerdings ohne Funktionssymbole und Gleichheit. Pelletiers Probleme mit Gleichheit oder Funktionssymbolen wurden nicht in den RPC-Formalismus übersetzt.

Klassifikation	Nr.	Zeit (s)	Schritte	Kommentar
Aussagenlogik	12	4.55	48	
	17	0.11	10	
Monadische PL	21	0.12	18	
	24	0.17	16	
	26	0.38	26	
	27	0.12	15	
	29	0.18	16	
	33	0.06	4	
	34	10.66	392	$\approx$ 1600 Klauseln in KNF
Volle PL	39	0.01	6	Russel's Paradoxon
	40	50.12	122	
	41	28.00	69	
	42	1.43	22	zirkuläre Mengen
	43	1.17	63	
	44	0.16	11	
	46	2.44	39	
	47	-	-	steamroller

Tabelle 1: Experimentelle Ergebnisse bei Pelletiers Problemen

Alle Probleme, bis auf "Schuberts Steamroller" konnten vom RPC-Beweiser gelöst werden. Die Problemnummern beziehen sich auf Pelletiers Artikel, die Zeit wurde auf einem Pentium-100 unter Linux 2.0 gemessen. In den Zeiten ist nur die reine Zeit für den RPC-Beweis enthalten, nicht die zur Vorverarbeitung benötigte. Die Schrittzahl bezieht sich auf die RPC-Regelanwendungen unter Verwendung der LP-Reduktionsstrategie.

Als weiteres Beispiel aus der Relationenlogik diente Gordeews Formel F_3 (siehe [Gor96]). Diese Formel kann mit 5 Variablen bewiesen werden, nicht aber mit 4.

$$F_3 = (E \odot E^{\smile})^{-} + (E^{\smile} \odot E)^{-} + (E \odot E^{2-} \odot E) \cdot i + (E^2 \cdot E^{\smile 2})^2$$

wird in Prädikatenlogik zu

$$\forall z(x\neg Pz \vee y\neg Pz) \vee \forall z(z\neg Px \vee z\neg Py) \vee \exists y\exists z(zPx \wedge xPy \wedge \forall x(y\neg Px \vee x\neg Pz))$$

$$\vee \exists z\Big(\exists y(xPy \wedge yPz) \wedge \exists y(zPy \wedge yPx) \wedge \exists x(yPx \wedge xPz) \wedge \exists x(zPx \wedge xPy)\Big)$$

Diese Formel kann in 20 Schritten und 1.83 Sekunden bewiesen werden.

7 Zusammenfassung und Aussichten

In dieser Arbeit wurde ein Beweiser, basierend auf Gordeews RPC-Kalkül, entwickelt. Verschiedene Reduktionsstrategien wurden dazu entwickelt und in einem automatischen Beweissystem implementiert.

Die Ergebnisse des in Haskell programmierten Beweisers sind für einen ersten Prototypen recht erfolgversprechend. Besonders bei Formeln, die tief verschachtelte Quantoren enthalten, zeigt der RPC-Beweiser ein ansprechendes Verhalten. Im Vergleich zu anderen Verfahren, die Eingaben in Klauselform erwarten, kann diese Konvertierung hier entfallen. Auch daraus können sich Vorteile für den RPC-Beweiser ergeben.

Darüberhinaus ist es möglich, andere Logiken (beispielsweise Relationenlogik) durch Beschränkung der Variablenzahl zu simulieren. So konnte für Gordeews Formel F_3 ein Beweis mit 5 Variablen schnell gefunden werden, wohingegen ein Beweis mit 4 Variablen nicht gelang.[8]

Die Verwendung anderer als der hier verwendeten Strategien oder eine Implementation in einer effizienteren Programmiersprache lassen noch überzeugendere Ergebnisse erreichbar scheinen.

[8] Wie Gordeew gezeigt hat, ist er auch nicht möglich.

Literatur

[Gal86] Jean H. Gallier. *Logic for Computer Science, Foundations of Automatic Theorem Proving*. Harper & Row, New York, 1986.

[Gor94] Lew Gordeew. Cut free formalization of logic with finitely many variables, part I. In *CSL-94*, number 933 in LNCS, pages 136–150. Springer-Verlag, 1994.

[Gor96] Lew Gordeew. On variable compactness in predicate logic. In *LNCS*. Springer-Verlag, 1996.

[Jon97] M. P. Jones. *Hugs 1.3, The Haskell User's Gofer System: User Manual*. University of Nottingham, August 1997. (Part of the HUGS package).

[Pel86] F. J. Pelletier. Seventy-five problems for testing automatic theorem provers. In *Journal of Automated Reasoning*, volume 2, pages 191–216, 1986.

[PH96] J. Peterson and K. Hammond. Report on the programming language haskell 1.3. Technical Report YALEU/DCS/RR-1106, Yale University, Department of Computer Science, May 1996.

[TG87] A. Tarski and S. Givant. A formalization of set theory without variables. *AMS Coll. Publ.*, 41, 1987.