PL-Decider: Finistische Rekonstruktion der Aussagenlogik

PD Dr. Timm Lampert

[Dies ist weitgehend ein Auszug aus einem unveröffentlichten und unabgeschlossenem Manuskript zur finistischen Grundlegung der Logik und Mathematik.]

Ziel dieses Abschnittes ist die finitistische Rekonstruktion der Aussagenlogik PL.Es geht also um die algorithmische Analyse bekannter wff von PL, in denen atomare Formeln durch P,Q,R,S,T bzw. P1,P2,P3, ... dargesetellt werden, die in nicht-atomaren wffs durch Junktoren wie ¬, ∧, ∨, →, ↔ verknüpft werden. Es wird mir im Folgenden zunächst darum gehen, die Analogie zwischen finitistischer Rekonstruktion von ℚ und PL vor Augen zu führen. ℚ ist die algorithmische Analyse von Gleichungen mit rationalen Zahlen, vgl. hierzu die Vorlesung "Arithmetik" und "QDeciderinto". Sie führt ebenso zu einem Algorithmus, der Gleichungen rationaler Zahlen auf rein finitistischer Grundlage entscheidet. Dieser wird im Folgenden schon vorausgesetzt.

ℚ und PL sind autonome Systeme - keines ist Teil des anderen; es gibt keine Überschneidungen; keines ist auf das andere reduzibel. Das eine basiert auf arithmetischen Operationen, die an Zahlen (arithmetischen Formen) ausgeführt werden. Das andere auf logischen Operationen, die an Satzformen (logischen Formen) ausgeführt werden. PL betrifft nach diesem Verständnis die Analyse logischer Operationen und der Formen, auf die sie sich beziehen, während ℚ die arithmetischer Operationen und der Formen, auf die sich diese beziehen, betrifft. Erst im Anschluss an die finitistische Rekonstruktion von PL werde ich dann darauf eingehen, wie ℚ und PL miteinander kombiniert werden können, indem Δ0-Sätze an die Stelle atomarer Formeln treten.

1. Aussagenlogik im Paradigma der mathematischen Logik

Es sei nur kurz an das logisch-axiomatische Beweisverständnis im Rahmen von PL erinnert. Insofern nur PL in den Blick genommen werden soll, kann weder von der Sprache L_A noch von axiomatischen Systemen wie Q und PA, die L_A und FOL voraussetzen, ausgegangen werden. Es entfällt damit auch der ganze Apparat, mathematische und metamathematische Sätze und Prädikate auszudrücken. Vom Blickpunkt der mathematischen Logik ist PL damit - ganz anders etwa als aus dem Blickpunkt einer philosophische Analyse von Argumenten - auf Grund der geringen Ausdruckskraft (gemessen an dem, was es in diesem Ansatz heißt, einen Satz auszudrücken) mehr oder weniger irrelevant. Beweise in PL betreffen deshalb auch nur den Beweis der *logischen* Wahrheit, was gemäß vorausgesetzter extensionalen Semantik Wahrheit in allen Interpretationen bedeutet. Es geht damit gar nicht um den Beweis der Wahrheit von bestimmten Sätzen, sondern allenfalls um den Beweis davon, dass alle Sätze, die eine aussagenlogischen Formel instanziieren, wahr sind ganz unabhängig von der Wahrheit oder Falscheit der Sätze, die die atomaren Formeln instanziieren. Logische Wahrheit von Sätzen wird über Allgemeingültigkeit von Aussagefunktionen definiert, in denen nur die aussagenlogischen Konstanten konstant sind. Sie wird damit als materiale Eigenschaft, nicht als formale Eigenschaft von Sätzen verstanden (zur Erinnerung: materiale Eigenschaften werden im Unterschied zu formalen durch Aussagefunktionen dargestellt und sind auf deren Interpretation angewiesen). Die Intuition, dass logische Wahrheit eine Form von "notwendiger", "formaler" oder "analytischer" Wahrheit ist, kann damit nicht eingefangen werden, da der Begriff der logischen Wahrheit über die Extension aller Wahrheitswertkombinationen atomarer Sätze definiert wird. Die (außersymbolische, materiale) Wahrheit oder Falschheit atomarer Sätze ist primär. Dass Allgemeingültigkeit - Wahrheit relativ zu allen Interpretationen - kein Kriterium für notwendige Wahrheit ist, erkennt man, wenn man Aussagenfunktionen betrachtet, in denen nicht-logische Termini konstant gehalten werden, z.B. "x ist amerikanische Präsidentin von Amerika". Diese Aussagefunktion ist in allen Interpretationen falsch und doch nicht notwendigerweise falsch. Solange nur auf Wahrheit oder Falschheit von Aussefunktionen rekurriert wird, können nur materiale Eigenschaft oder materiale Begriffe definiert werden. Ein konsequenter extensionaler Ansatz beansprucht auch nichts anderes. Im Gegenteil, er sieht einen Vorteil darin, von obskuren modalen Begrifflichkeiten oder opaquen begriffliche Intuitionen, mit denen der Begriff der logischen Wahrheit überladen sein mag, abzusehen.

Die Korrektheit und Vollständigkeit aussagenlogischer Kalküle wird im logisch-axiomatischen Beweisverständnis allein an der logischen Wahrheit gemessen: Ein Kalkül ist korrekt und vollständig gdw. er genau die Formeln ableitet, die gemäß aller Interpretationen wahr sind. Freges Kalkül ist ein Beispiel für ein axiomatischen, aussagenlogischen Kalkül, der nur allgemeingültige Formeln ausgehend von 6 Axiomenschemata (3 davon wären ausreichend) unter Verwendung einer Substitutionsregel sowie MPP als Theoreme beweist. Vollständigkeit ist hier anders als etwa im Falle atomarer Δ0-Sätze in Q nicht "negation completeness", denn es gibt aussagenlogische Formeln, die nicht logisch wahr sind und deren Negation auch nicht logisch wahr ist (dies ist sogar die Regel). Ein korrekter und vollständiger Kalkül - wie etwa Freges Kalkül oder ein Sequenzenkalkül oder ein Kalkül des natürlichen Schließens - liefert damit auch noch kein Entscheidungsverfahren für logische Wahrheit, denn wenn weder eine Formel noch ihre Negation als Theorem ableitbar ist, terminiert die Ableitung in beiden Fällen nicht.

Als Entscheidungsverfahren fungiert vielmehr die Wahrheitswerttabellenmethode. Diese gehört im logisch-axiomatischen Ansatz aber nicht zur Syntax, sondern zur extensionalen Semantik. Es werden in dieser einfach alle endlich vielen möglichen Interpretationen durchgegangen und die modelltheoretischen Definitionen angewendet. Da wffs und Interpretationen endlich viele sind, kann auf diese Weise berechnet und entschieden werden, ob einer wff in allen Interpretationen der Wahrheitswert W zugeordnet wird. Im

Rahmen des logisch-axiomatischen Ansatzes werden die Wahrheitswerttabellen weder als Ausdruck einer möglichen Welten-Semantik (oder anderen intensionalen Semantik) verstanden noch als ein etwaiger Ausdruck einer syntaktischen Umformung (etwa analog zu kanonischen disjunktiven Normalformen). Es gibt zwar auch rein syntaktische Entscheidungsverfahren wie etwa die Umfomrung in disjunktive Normalformen (DNF) oder das Tableau-Verfahren, aber diese sind nicht der typische Ausdruck des logisch-axiomatischen Beweisverständnisses. Oft werden diese Verfahren als Nachahmung oder Optimierung der Wahrheitstabellen-methode verstanden und nicht als Ausdruck eines anderen Beweisverständnisses. Sollte letzteres doch der Fall sein, nähert man sich hiermit einem finitistischen Verständnis an; - dies wird später anhand der Analogie finitistischer logischer Beweise mit Umformungen in optimierte disjunktive Normalformen (in PL wie auch FOL) deutlich werden.

Zumindest heuristisch ist es nicht irrelevant, dass die Wahrheitswerttabellenmethode als Paradigma eines Entscheidungsverfahrens in PL dient. Denn es ist klar, dass die Methode, alle Interpretationen auszuwerten, nicht als Leitbild für ein Entscheidungsverfahren für FOL dienen kann, da mit Einführung von Variablen, über die quantifiziert wird, der Raum der Interpretationen nicht mehr endlich ist. Selbst wenn man den Gegenstandsbereich auf einen abzählbar unendlichen einschränkt, erhält man unter Voraussetzung von Cantors Theorem sogar überabzählbar viele mögliche Interpretationen von Prädikaten. Ich werde demgegenüber im Rahmen der Ausführung zu FOL zeigen, wie ein finitistisches Beweisverständnis im Allgemeinen und das für PL-Beweise im Besonderen ein ganz anderes Leitbild für die Entwicklung von Beweis- und Entscheidungsverfahren in FOL bereitstellt.

Dass die Wahrheitswerttabellenmethode ein Entscheidungsverfahren ist, wird unter Voraussetzung von Churchs These durch Übersetzung der Wahrheitsfunktionen in primitiv rekursive, also numerische, Funktionen bewiesen. Auch hier ist der Maßstab ein extensionaler; die Übersetzung beruht auf materialer Äquivalenz (Extensionsgleichheit,nicht Synonymie oder formaler Äquivalenz). Prioritiät hat stets der Bezug zu Extensionen und damit die Semantik, an der die Syntax und die Entscheidbarkeit gemessen werden.

2. Finitische Aussagenlogik

Ich rekurriere im Folgenden auf Wittgensteins ab-Notation, die dieser in den Jahren 1912-14 entwickelt hat, vgl. hierzu [habil]. Es ist diese Notation und nicht die auch von Wittgenstein verwendeten Wahrheitswerttabellen ("WF-Schemata"), die Wittgenstein auch auf FOL überträgt und von der er in einem Brief an Russell vom November 1913 (CL, S. 53) behauptet, dass sie - in heutiger Terminologie ausgedrückt - ein Entscheidungsverfahren für FOL bereitstelle. Wittgensteins Logikkonzeption im TLP stützt sich, zumindest was FOL angeht, allerdings nicht mehr auf die ab-Notation, sondern auf seinen N-Operator (vgl. TLP 6). Dies setzt voraus, dass sich quantifizierte Aussagen als Wahrheitsfunktionen von (endlichen oder unendlichen) Mengen von Elementarsätzen analysieren lassen, ohne dass ein Verfahren angegeben wird, wie diese Mengen jeweils zu konstruieren sind (vgl. TLP 5.52ff). Diese Auffassung nannte Wittgenstein später den "biggest mistake" des TLP und ein Entscheidungsverfahren für FOL gewinnt man jedenfalls nicht auf diesem Wege. Wittgenstein kehrt m.E. später (vgl. insbesondere "The voices of Wittgenstein", S.162-170) zur Auffassung der Irreduzibilität von FOL-Sätzen zurück, die auch schon seiner ab-Notation zu Grunde liegt. Die Auffassung, dass FOL-Tautologie zu sein, eine formale Eigenschaft ist, die sich durch bloßes Umformen von FOLwffs identifizieren läßt, vertritt er allerdings vor und im TLP und hat er nach meinem Verständnis nie revidiert. Die ab-Notation war Wittgensteins Ansatz, diese Auffassung umzusetzen; ebenso wie seine Ω-Notation in der Arithmetik hat er sie aber nie ausbuchstabiert. Die Tatsache, dass Wittgensteins Verständnis logischer Beweise im Allgemeinen und seine Vision der ab-Notation im Besonderen die Entscheid-barkeit von FOL impliziert, ist ein Hauptgrund ihrer Ablehnung (vgl. Potter, Landini, aber auch schon Ramsy, Anscombe, Sund-holm). Diese Ablehnung beruht aber letztlich auf der Voraussetzung des logisch-axiomatischen Beweisverständnis und damit dem Paradigma der mathematischen Logik. Die Ausarbeitung von Wittgensteins zugegebenermaßen nur programmatisch formulierten Andeutungen soll dieses Urteil revidieren.

Nach Wittgenstein sind ℚ (= Algorithmus zur Entscheidung von Gleichungen mit rationalen Zahlen) und PL gleichartige, autonome Kalküle (Beweissysteme), vgl. z.B. WWK, S. 218ff. Dies soll in diesem Abschnitt konkretisiert werden, indem ein Algorithmus zur Entscheidung von PLwffs ganz in Analagie zum ℚ-Algorithmus entwickelt wird. Im folgenden Abschnitt wird dann gezeigt, wie diese Algorithmen kombiniert werden. In keinem Fall ist einer die Grundlage des anderen.

2.1 Bipolarität

Während ℚ-Terme als Anweisungen zur Konstruktion von Ω-Repräsentanten arithmetischer Formen aufgefasst werden, sind PL-Terme Anweisungen zur Konstruktion von ab-Symbolen, die logische Formen repräsentieren. In ℚ mussten hierfür Ziffern (typischerweise in der Dezimalnotation) zunächst in Ausdrücke der Form $S^{\mu\,\overline{S0}}\,0$ umgeformt werden, bevor dann die arithmetischen Definitionen für +,-,×,÷,^ angewendet werden können. Der Umformung der Ziffern entspricht in PL die Umformung atomarer Formeln (z.B. P, Q, etc.) in bipolare ab-Ausdrücke, auf die dann anschließend die logischen Definitionen für ¬,∧,∨,→,↔ angewen-det werden können. Ein atomarer Ausdruck, z.B. P, ist als bipolarer Ausdruck zu schreiben, indem er mit einem a- und einem b-Pol versehen wird, z.B. a-P-b. Auf bipolare Ausdrücke können dann Definitionen logischer Junktoren in Form von ab-Operationen in Form von Vorschriften zur Zuordnung von a- und b- Polen zu a- und b-Polen angewendet werden. Bipolarität ist eine wesentliche,

formale Eigenschaft atomarer Ausdrücke, ihre Kennzeichnung durch a- und b-Pole eine zufällige Eigenschaft ihrer Darstellung in der ab-Notation.

Dieser allererste Schritt der Umformung in eine Notation, deren äussere Merkmale die Anwendung bestimmter Definitionen erlaubt und damit ermöglicht, aritmethische oder logische Eigenschaften über die äussere Form von Ausdrücken zu identifizieren, ist - so trivial er erscheinen mag - für das Verständnis der Beweiskonzeptionen enorm wichtig. Er erscheint als unbedeutend, solange man unter einem Beweis den Beweis der Wahrheit (ggf. auch der logischen Wahrheit im Sinne seiner Allgemeingültigkeit) eines Satzes versteht - in welcher Form dieser auch immer dargestellt sein mag; - zu beweisen ist ja nach diesem Verständnis die Extension eines Satzes (sein Wahrheitswert) und nicht wie im finitistischen Beweisverständnis, dass er eine bestimmte formale Eigenschaft besitzt, die durch den Beweis allererst zu identifizieren ist, indem durch Anwendung der Definitionen (Beweisregeln) diese formale Eigenschaft eineindeutig einer äusseren Eigenschaft der resultierenden Ausdrücke zugeordnet wird. Bipolarität ist die äussere Eigenschaft der ab-Symbole, die "logische Unentscheidbarkeit" repräsentiert, d.i. die Eigenschaft weder logisch wahr (tautologisch, logisch gültig) noch logisch falsch (kontradiktorisch, logisch ungültig) zu sein. Logische Wahrheit / Falschheit ist innerhalb von PL eine Folge der junktorenlogischen Verknüpfung atomarer Ausdrücke; - die formale Eigenschaft der Bipolarität kann erst in Folge der Anwendung von ab-Operationen und weiteren Definitionen zur Herstellung idealer ab-Symbole "verloren" gehen. Atomare Ausdrücke sind demzufolge *im Rahmen des Systems PL* (d.h. ohne Berücksichtigung weiterer Regel für etwaige spezifische atomare Ausdrücke wie z .B. ℚ-Gleichungen) bipolar und folglich mit einem a- und einem b-Pol zu versehen.

Wie die Ω-Notation soll durch die ab-Notation eine extensionale Deutung des Ausgangsformalismus (ℚ bzw. PL) ausgeschlossen bzw. überflüssig gemacht werden. Solange man zur Darstellung von (rationalen wie irrationalen) Zahlen die Dezimalnotation verwendet, ist nicht offensichtlich, dass Zahlen Formen sind, denn ihre Ausdrücke in der Dezimalnotation bringen ihre formalen Merkmale nicht zum Ausdruck. Schon deutlich zweckmässiger als die Dezimalnotation in Hinblick auf die Identifizierung von Eigenschaften von Zahlen im Sinne formaler Eigenschaften sind Brüche (im Falle irrationaler Zahlen reguläre oder irreguläre Kettenbrüche). Aber, wie wir in ℚ insbesondere in Bezug auf die Stellung der 0 gesehen haben, kann auch diese Notation nicht vollständig ersichtlich machen, dass die fraglichen arithmetischen Eigenschaften ausschließlich formale Eigenschaften sind, die gar nicht auf eine etwaigen extensionalen Bedeutung der Ausdrücke Bezug nehmen. Aus der Sicht eines finitistischen Beweisverständnisses sind es diese äusseren Eigenschaften der gängigen Zeichensprachen in Mathematik und Logik, die zu ihrer extensionalen Deutung veranlassen, wenn man sie nicht als Anweisungen zur Konstruktion von Ausdrücken versteht, deren äussere Form die fraglichen formalen Eigenschaften identifiziert. "2" scheint äusserlich wie ein Name, "SS0" dagegen hat schon eine Struktur, in der z.B. 0 und S unterschieden sind und die zu "0", "S0" und "SSS0" etc. eine interne, strukturelle Beziehung hat, die "2" im Unterschied zu "0", "1", "3" etc. nicht aufweist. $\frac{2}{2}$erscheint äusserlich kaum anders als $\frac{2}{0}$während in der Ω-Notation durch die fundamentale Unterscheidung von Operation (S im Falle von ℚ) und Anfangsbasis(0 im Falle von ℚ) sowie die Einschränkung der Möglichkeit der Umkehrung auf Operationen sofort ersichtlich ist, dass "durch 0 nicht geteilt werden kann". Das Verständnis arithmetischer Terme mit +,-,×,÷, ^ als Funktionsausdrücke, die Gegenstände bedeuten vs. ihres Verständnisses als Operationen sowie das Verständnis der Gleichheit als zweistellige propositionaler Funktion vs. seinem Verständnis als formaler Eigenschaft, sind die konsequente Folge des ersten Schrittes in der Darstellung von Ziffern. Die Frage, ob zwei ℚ-Terme identisch sind, fragt nach dem einen Verständnis danach, ob sie dieselbe Extension haben, während sie nach dem anderen nach einer Übersetzung in einen idealen kanonischen Repräsentanten fragt, - und damit in eine Notation, in der diese Frage sich nicht mehr stellt.

Die Rechengesetze und Eigenschaften der Zahlen haben nach finistischem Beweisverständnis rein formale Gründe, die in der äusseren Struktur einer angemessenen Notation ausgedrückt werden. Es wird keineswegs behauptet, dass es notwendig ist, dies zu tun. Die üblichen syntaktischen Regeln zur Dezimalrechnung und Bruchrechnung verleihen den Ausdrücken dieser Notation eben die Eigenschaften, die auch durch die Ω-Ausdrücke identifiziert werden. Es geht auch hier wieder in erster Linie nur um die Schärfung des Blickes für die jeweiligen Beweisverständnisse: Sieht man die syntaktischen Regeln nicht aus der Not einer nicht vollständig expliziten Notation geboren, dann wird man sich veranlasst sehen, diese Regeln an einer etwaigen Interpretation der Zeichen, die aus derselben Not geboren wird, zu messen. Man wird die Rechenregeln für +,-,×,÷, ^ in Bezug auf Brüche dann daran messen, ob sie die entsprechenden mengentheoretischen Funktionen abbilden, statt die Aufgabe darin zu sehen, diese Regeln durch Rekonstruktion in eine Übersetzung in eine vollständig transparente Notation (d.i. eine Notation, die die fraglichen arithmetischen Eigenschaften anhand äusseren Merkmale eineindeutig identifiziert) zu rekonstruieren bzw. zu explizieren. Man sieht: Es sind die allerersten, scheinbar trivialen und aus Sicht eines logisch-axiomatischen Beweisverständnis völlig bedeutungslosen Anfänge der Darstellung der fraglichen Ausdrücke, die aus Sicht eines finitistischen Beweisverständnisses bereits den alles entscheidenden Unterschied ausmachen.

Die Bedeutung dieses ersten Schrittes in der Darstellung ist noch viel augenfälliger in der Logik. Die Darstellung atomarer Formeln in Form unstrukturierter Ausdrücke wie "P", "Q", "P1" etc. verleitet dazu, sie bzw. ihre Instanzen wie Namen von Gegenständen zu deuten (was z.B. Frege explizit tat). Diese Deutung verlangt dann die Annahme von Wahrheitswerten als etwaiger Gegenstände der Referenz. Dies wiederum bedingt die Interpretation logischer Formeln als Wahrheitswertfunktionen. Man muss sich nicht so

explizit auf eine entsprechende Semantik und Ontologie festlegen und man kann auch eine andere Semantik und Ontolgie vertreten, z.B. in einer intensionalen Semantik, nach der atomare Ausdrücke Sachverhalte darstellen oder atomare Tatsachen behaupten. Einer Semantik vollständig entledigen wird man sich aber nur können, wenn man die Darstellung atomarer Ausdrücke in der herkömmlichen Notation als Ausdruck einer missverständlichen Notation versteht, die die logischen Eigenschaften der Symbole nicht durch ihre äussere Form ausdrückt. Eine weniger missverständliche Notation drückt formale Eigenschaften durch die äussere Struktur ihrer Ausdrücke aus. Eben dies leistet die ab-Notation. Um auch hier wieder hervorzuheben, dass es nur um strukturelle Merkmale der Zeichen geht und nicht um ihre etwaige Bedeutungen, werden bewusst die arbiträren Zeichen “a” und “b” zur Darstellung der Satzpole verwendet und nicht “W” und “F”, die als Wahrheitswerte oder Satzseigenschaften gedeutet werden könnten.

Es soll damit gar nicht geleugnet oder ausser Acht gelassen werden, dass die Logik es mit Sätzen und damit auch mit Wahrheit und Falschheit zu tun hat. Dies heisst aber nicht, dass sie sich auf Wahrheitswerte von Sätzen oder auf mögliche Welten beziehen muss. Vielmehr hat sie es mit den Eigenschaften zu tun, die ein Ausdruck überhaupt haben muss, um wahr oder falsch sein zu können. Vor aller Interpretation - extensional oder intensional - müssen Ausdrücke eine bestimmte Form haben, um überhaupt auf die eine oder andere Weise interpretiert werden zu können. Bipolarität ist die formale Eigenschaft, die ein Ausdruck haben muss, um ihn als einen Satz im Sinne eines Ausdruckes, der wahr oder falsch sein kann, interpretieren zu können.
Bipolarität ist eine formale Eigenschaft, eine Eigenschaft von Satzformen (logischen Formen), nicht etwa zu verwechseln mit der Bivalenz von Sätzen, d.i. der Eigenschaft von Sätzen auf Grund einer Interpretation entweder wahr oder falsch zu sein (nichts Drittes). Das Bivalenzprinzip ist ein semantisches Prinzip und im Rahmen eines finitistischen Verständnisses von PL überflüssig.

In einer Notation, in der Bipolarität kein äußeres formales Merkmal ist, hat ein Ausdruck diese Eigenschaft auf Grund der syntaktischen Regeln, die für ihn gelten (vgl. zur Äquivalenz von ab-Notation und PL-Syntax die Ausführungen unten zur Korrektheit von PL): Aus P folgt z.B. nicht ¬P, vielmehr folgt P ∨ ¬ P aus allem und aus P ∧ ¬ P folgt alles, P und ¬¬P sind äquivalent; P und ¬P haben, sozusagen, entgegengesetzte syntaktische Funktionen. Es ist im Rahmen eines logischen Systems mit diesen Regeln unmöglich, P und ¬P denselben Sinn oder dieselbe Bedeutung zu geben. In der ab-Notation wird diese schon anhand der äußeren Form ausgedrückt: a-P-b hat eine andere Struktur als b-P-a. Dies kann man sich für eine etwaige Interpretation zu Nutze machen, indem man a-P-b das (ideale) ab-Symbol für P und ¬¬P und b-P-a das (ideale) ab-Symbol für ¬P sein lässt. Die Tatsache, dass einmal “a” links und von “P” steht und einmal nicht, kann dann verwendet werden, um einen Gegensatz auszudrücken. Man kann dann z.B. festlegen: Dass “a” links von “P” steht, drückt aus, dass eine Instanz von “P” wahr ist gdw. P der Fall ist und die Tatsache, dass “a” nicht links von “P” steht, drückt aus, dass eine Instanz von “P” falsch ist gdw. P nicht der Fall ist. Und umgekehrt: Dass “a” nicht links von “P” steht, drückt aus, dass eine Instanz von “P” nicht wahr ist gdw. P nicht der Fall ist und die Tatsache, dass “a” rechts von “P” steht, drückt aus, dass eine Instanz von “P” wahr ist gdw. P der Fall ist. Das Bestehen und Nicht-Bestehen einer Eigenschaft der Struktur des Symbols, wird verwendet, um die Wahrheitsbedingungen bzw. Möglichkeiten der Wahrheit und Falschheit eines Satzes festzulegen. Auch hier wird eine modale Eigenschaft - in diesem Fall, die Möglichkeit eines Satzes, wahr oder falsch sein zu können - durch eine formale Eigenschaft - in diesem Falle die der Bipolarität - expliziert. Die Trivialität dieser Beispiele täuscht hier nur allzu schnell über den fundamentalen Unterschied zu einem auf Semantik basierenden Verständnis der Logik hinweg. Ich werde in den folgenden Abschnitten vor Augen führen, wie durch einen Algorithmus der Übersetzung von PLwffs in ideale ab-Symbole sowohl ein logisch-axiomatisches Beweisverständnis wie auch eine Semantik überflüssig werden, da die logischen Eigenschaften - die Eigenschaften, die es ermöglichen, Ausdrücke als Sätze mit bestimmten Wahrheitsbedingungen zu interpretieren - eindeutig anhand der äusseren Eigenschaften von idealen ab-Symbolen identifiziert werden können.

Es wird hierbei natürlich auch nicht geleugnet, dass sowohl die Gestalt der jeweiligen Zeichen als die spezifischen strukturellen Eigenschaften willkürlich sind: statt a-P-b wäre genausogut a-P oder $\overset{a}{\underset{b}{P}}$ o.ä. möglich und statt “a” und “b” hätten beliebige andere

Zeichen verwendet werden können; ich werde in dem in *Mathematica* implementierten Algorithmus zudem weitere, für die *Mathematica*-Programmierung zweckdienliche notationelle Varianten verwenden. Aber eines ist nicht willkürlich: Damit ein Symbol überhaupt wahr oder falsch sein kann bzw. den Unterschied von Wahreit und Falschheit ausdrücken kann, muss es Eigenschaften einer bestimmten Mannigfaltigkeit haben; - es muss in irgendeiner Weise Bipolarität sowie die Grenzfälle der Aufhebung dieser Eigenschaft in Folge der spezifischen Verknüpfung atomarer Ausdrücke ausdrücken können. Diese Eigenschaften können ihm durch syntaktische Regeln, die die Beziehung zu anderen Symbolen in einer intransparenten Notation regeln, verliehen werden oder es kann sie auf Grund der äusseren Eigenschaften einer transparenten Notation besitzen, auf die dann in einer Interpretation bzw. Festlegung von Wahrheitsbedingungen rekurriert wird. Etwaige Interpretationen setzen nach diesem Verständnis nicht an dem allererst noch zu analysierenden, missverständlichen aussagenlogischen Ausdruck an, in dem etwa noch die Junktoren als zu interpretierende Ausdrücke vorkommen, sondern erst am idealen Symbol als dem Resultat der algorithmischen Analyse der aussagenlogischen Formeln. Interpretation idealer Symbole ist aber auch in diesem Fall keine Zuordnung zu etwaigen Extensionen, sondern nur die mechanische Übersetzung oder Paraphrase in Form der Explikation von Möglichkeiten der Wahrheit und Falschheit von Instanzen aussagenlogischer Formeln mit den Mitteln standardisierter umgangsprachlicher Ausdrücke: Die

Sprache bleibt bei alledem unhintergehbar; sprachliche Ausdrücke werden unter keinen Umstände etwaigen Extensionen zugeordnet. Explikationskraft haben etwaige Paraphrasen idealer Symbole allein dadurch, dass “dasselbe” noch einmal auf eine andere, transparente Weise ausgedrückt wird, indem - anders als in den zu explizierenden Ausdrücken können - fragliche formale Eigenschaften anhand der äußeren Form der Explikate identifiziert werden können.

In jedem Fall müssen die logischen Ausdrücke vor jeder Interpretation formalen Eigenschaften auf die eine oder andere Weise besitzen und die Logik hat es nur mit diesen Eigenschaften zu tun. Sie untersucht logische Formen - Satzformen - unabhängig und vor jeder Bezugnahme auf Extensionen oder Intensionen. Nach diesem Verständnis besteht immer eine Priorität der Syntax gegenüber der Semantik und es ist widersinnig erstere durch letztere messen zu wollen, indem etwa die Korrektheit oder Vollständigkeit von Kalkülen (Beweissystemen) oder die Entscheidbarkeit bestimmter logischer Eigenschaften an der Übereinstimmung mit bestimmten Interpretationen beurteilt werden. Allenfalls kann man die Syntax (bzw. unterschiedliche Kalküle einer Formelsprache) und Semantik (bzw. unterschiedliche Ansätze, die jeweilige Formelsprache zu interpretieren) einander gegenüberstellen - sollten sie in irgendwelchen Ergebnissen nicht übereinstimmen, kann nach finistischem Beweisverständnis nicht auf die Inkorrektheit des Beweissystems geschlossen werden. Es könnte höchstens geschlossen werden, dass ein Beweissystem bestimmte Interpretationen nicht zulässt, die man unter bestimmten semantischen Voraussetzungen trifft bzw. treffen möchte. Im Rahmen eines finitistischen Ansatzes kann einer Semantik von PL nur Rechnung getragen werden, wenn diese selbst wieder als ein autonomes System von Regeln zur Manipulation von PL-wffs rekonstruiert werden kann, etwa indem man Wahrheitswerttabellen als Umformungen von PL-wffs rekonstruiert. Statt ein Beweissystem von PL hieran zu messen, kann dann gefragt werden, ob eine so rekonstruierte Semantik und ein PL-Beweissystem sich aufeinander abbilden lassen (vgl. hierzu Lampert, “Klassische Logik”, Kap. 6 zur Rekonstruktion des Korrektheits- und Vollständigkeitsbeweises des Gentzen-Lemmon-Kalküls).

Nach diesen allgemeinen Bemerkungen zum Verständnis von PL, sei nun in den folgenden Abschnitten ausgeführt, wie PL finitistisch durch Übersetzung von PLwffs in ideale ab-Symbole ganz analog zu Q zu rekonstruieren ist. Dies geschieht durch Beschreibung eines *Mathematica*-Programmes, das PL-wffs in deren ideale ab-Symbole umformt.

2.2. Primitive Regeln

PL-wffs entsprechen strenggenommen Q-Terme, nicht Q-Gleichungen. Das Pendant zur Evaluierung von Q-Gleichungen würde darin bestehen, zu evaluieren, ob zwei PL-wffs auseinander folgen. Dementsprechend müsste eine Formel der Form A -||- B in ein Algorithmus eingegeben werden, wobei dann diese Frage daran entschieden wird, ob A und B demselben idealen ab-Symbol zugeordnet werden. -||- ist dann wie = eine formale Eigenschaft (bzw. Relation). Natürlich könnte ebensogut auch entschieden werden, ob eine Formel B aus einer Formel A folgt oder ob eine Formel A tautologisch / kontradiktorisch ist. All dies sind formale Eigenschaften, die durch Umformung von PL-wffs in ab-Symbole entschieden werden; diese Umformung “berechnet die logischen Eigenschaften der PL-wffs”. Ich werde mich im Folgenden hierauf beschränken - das Programm gibt also ideale ab-Symbole aus, denen man ihre logischen Eigenschaften direkt ablesen kann. Sie sind die idealen Repräsentanten logischer Formen (Satzformen). Es gleicht damit der Umformung von Q-Termen in ideale kanonische Ω-Repräsentanten, indem PL-wffs in ideale kanonische ab-Symbole umgeformt werden.

Wie der Q-Algorithmus ist ein PL-Beweis im engen Sinn (d.i. ohne Anwendung der Definition einer formalen Eigenschaft wie -||-) eine systematische Anwendung primitiver Regeln zum Zwecke der Umformung in einen idealen Repräsentanten. Diese Regeln zerfallen wieder in 2 Sorten: die Regeln der Übersetzung in die ab-Notation und die ab-Regeln. Während die Regeln zur Übersetzung in die Ω-Notation mit der Übersetzung von Ziffern natürlicher Zahlen beginnt, setzten die Regeln der Übersetzung von PL-wffs in die ab-Notation bei der Übersetzung atomarer PL-wffs in die entsprechenden bipolare Ausdrücke an. Hierbei sei abgewichen von Wittgensteins Darstellung einer atomaren Formel wie P durch a-P-b, um einen einheitlichen Algorithmus zu erhalten, der insbesondere auch von Wittgensteins umständlichen graphischen Diagrammen absieht. Stattdessen werde ich - auch einem Vorschlag Wittgensteins folgend - Satzformen in Form von Polgruppen schreiben, wobei von vorneherein a- und b-Polgruppen unterschieden werden. Diese Gruppen enthalten selbst wieder einzelne Pole, z.B. a-P bzw. b-P, wobei ich auch hier die einfache Schreibweise, in der die Pole immer links stehen, vorziehe. Die atomare Formel P ist dann entsprechend durch die Polgruppen a-{a-P} und b-{b-P} darzustellen. Im *Mathematica*-Programm werde ich Polgruppen als Listen mit a bzw. b als Kopf darstellen und alle a- und b-Polgruppen zusammengenommen in einer Liste aufführen. P wird dementsprechend durch {a[{a[P]}], b[{b[P]}]} dargestellt. Diese Notation werde ich im Folgenden zu Grunde legen. Wählt man $\mathscr{A}$ als Variable atomarer Formeln ergibt sich dementsprechend:

Def. $\mathscr{A}$: $\mathscr{A}$ = {a[{a[$\mathscr{A}$]}], b[{b[$\mathscr{A}$]}]}

Man kann den aus der Anwendung von Def. $\mathscr{A}$ resultierenden ab-Ausdruck einer atomaren Formel gebrauchen, um die Wahrheitsbedingungen einer atomaren Formel festzulegen, indem man etwa die a-Polgruppen als Ausdruck der Bedingungen der Wahrheit und die b-Polgruppen als Ausdruck der Bedingungen der Falschheit liest: "Eine Instanz von P ist wahr gdw. P wahr ist und falsch gdw.

P falsch ist" ist dann nichts anderes als eine mechanische Paraphrase des entsprechenden ab-Ausdruckes. Diese ist bereits ein ab-Symbol, d.i. ein idealer kanonischer Repräsentant, einer atomaren Formeln. ab-Symbole explizieren die formalen Eigenschaften, die nötig sind, um einem Satz als Ausdruck von Wahrheitsbedingungen interpretieren zu können.
Die allgemeine primitive Regel zur Übersetzung atomarer Ausdrücke lautet demzufolge:

```
ptoapb[atom_] :=
  Module[{term},
   term = {a[{a[atom]}], b[{b[atom]}]};
   term];
```

Im Folgenden werde ich die restlichen primitiven und zusammengesetzten Regeln des PL-Deciders angeben; ich werde dabei nicht ihren "logischen" Gehalt erklären. Dies geschieht dann in Abschnitt "2.4 Korrektheit"; falls der Leser also bekannte Algorithmen in den Regeln der ab-Notation wiedererkennen will, nehme er diesen Abschnitt mit zur Hilfe.
Analog zu den arithmetischen Definition von +,-,×,÷, ^ als Übersetzungsregeln in die Ω-Notation unter Voraussetzung der Übersetzung natürlicher Zahlen in Ω-Ausdrücke der Form $\left\{S^{\{\mu,\overline{\{S,0\}}\}}, 0\right\}$, können nun die Definitionen der Junktoren als Übersetzungsregeln in die ab-Notation unter Voraussetzung von Def. $\mathscr{S}$ angegeben werden. Diese werden als ab-Operationen definiert, d.i. als Operationen die a- und b-Polgruppen bzw. Paare von a- und b-Polgruppen wieder a- bzw. b-Polgruppen zuordnen. In der *Mathematica*-Syntax ausgedrückt, erhält man stets wieder Polgruppen, deren Kopf a ist und Polgruppen, deren Kopf b ist. Bei einer Anwendung einer dyadischen ab-Operation werden jeweils die 4 Cartesischen Produkte der 4 Polgruppen gebildet und dem a- bzw. b-Pol zugeordnet. Prinzipiell gibt es zwei monadische und 14 dyadische Operationen (nicht 16, da stets sowohl der a- als auch der b-Pol zuzuordnen ist). Ich beschränke mich auf die Operationen, die die Junktoren ¬,∧,∨,→,↔ definieren. Um diese Definition in Übereinstimmung mit ihrer herkömmlichen Anwendung und Definition anzugeben, setzte ich voraus, dass der a-Pol als Ausdruck der Bedingung der Wahrheit eines Satzes und der b-Pol als Ausdruck seiner Falschheit interpretiert werden soll. Ich führe damit einen Unterschied zwischen "a" und "b" ein, der im Prinzip willkürlich ist, aber irgendeine Festlegung muss getroffen werden, um Übereinstimmung mit der herkömmlichen Verwendungen (der "intendierten Interpretation") zu erhalten. (Besonders deutlich wird dies für den Fall symmetrischer ab-Ausdrücke).

Def.¬: ¬[{a[PG1],b[PG2]}] = {a[PG2],b[PG1]}
Def.∧: ∧[{a[PG1],b[PG2]},{a[PG3],b[PG4]}] = {a[PG1×PG3],b[PG1×PG4,PG2×PG3,PG2×PG4]]}
Def.∨: ∨[{a[PG1],b[PG2]},{a[PG3],b[PG4]}] = {a[PG1×PG3,PG1×PG4,PG2×PG3],b[,PG2×PG4]]}
Def.→: →[{a[PG1],b[PG2]},{a[PG3],b[PG4]}] = {a[PG1×PG3,PG2×PG3,PG2×PG4],b[PG1×PG4]]}
Def.↔: ↔[{a[PG1],b[PG2]},{a[PG3],b[PG4]}] = {a[PG1×PG3,PG2×PG4],b[PG2×PG3,PG1×PG4]]}

Durch den Bezug auf Polgruppen (PG) im Definiens und Definiendum können die Junktoren wieder als reine explizite Umformungsregeln ohne Fallunterscheidungen definiert werden. Diese Art Definition unterscheidet sich damit von ihrer Interpretation als Wahrheitsfunktionen. Besonders deutlich wird dieser Unterschied im Fall von P ∧ P: Im Rahmen der ab-Notation muss P erstmal jeweils ersetzt werden: ∧[{a[a[P]],b[b[P]]}, {a[a[P]],b[b[P]]}]. Hierauf kann dann Def.∧ angewendet werden: {a[a[P],a[P]}], b[{a[P],b[P]},{b[P],a[P]},{b[P],b[P]}]}. Die a-Pogruppe enthält hier den Pol a[P] mehrfach; die b-Polgruppe enthält zwei Polgruppen mit den entgegengesetzen Polen a[P] und b[P]. Hierfür gibt es jeweils kein Pendant in der Interpretation von Wahrheitsfunktionen. Diese geht von Fallunterscheidungen aus und kennt hierbei nicht den Fall, in dem ein und derselben atomaren Formel sowohl der Wahrheitswert W als auch F zugeordnet wird; im Rahmen von Interpretationen macht das keinen Sinn bzw. widerspricht dem Bivalenzprinzip. Im Rahmen der ab-Notation wird aber gar nicht auf Interpretationen Bezug genommen und es werden auch nicht unterschiedliche Fälle unterschieden. Vielmehr kombiniert die Anwendung der Definitionen der dyadischen Junktoren stets 4 Polgruppen neu; die Regeln der ab-Notation erzwingen, dass im Definiendum stets [{a[PG1],b[PG2]},{a[PG3],b[PG4]}] vorkommt und im Definiens stets erneut a- und b-Polgruppen. Es gibt aus diesem Grund auch keine etwaige "Tautologie-" oder "Kontraditionsoperation" (während es eine derartige Wahrheitsfunktion, die alle Wahrheitswertkombinationen W bzw. F zuordnet, gibt). Tautologie bzw. Kontradiktion zu sein, ist eine formale Eigenschaft, keine Operation. Diese formale Eigenschaft kann in Folge der Anwendung sämtlicher primitiven Regeln der ab-Notation an den ab-Symbolen direkt abgelesen werden; - die Anwendung junktorenlogischer Regeln sind hierfür notwendig, aber nicht hinreichend.

Die Definitionen der Junktoren macht deutlich, dass sich ihre Bedeutung darin erschöpft, Anweisungen zur Konstruktion logischer Formen zu sein. In den ab-Symbolen kommen sie gar nicht mehr vor. Sie sind damit auch gar keine Zeichen die bei einer Interpretation der idealen Symbole zu interpretieren wären. Ihre Bedeutung erschöpft sich vollständig in ihrer "algorithmischen Bedeutung", d.i. ihrer Bedeutung im Rahmen des Umformungsprozesses aussagenlogischer Formeln in ab-Symbole. Auf diese Weise wird eine Interpretation der Junktoren im Sinne von Wahrheitsfunktionen

ausgeschlossen.

Wiederholte Anwendung der genannten Definition erlaubt ganz analog zur iterativen Anwendung der arithmetischen Definitionen zum Zwecke der Übersetzung in die Ω-Notation die vollständige Übersetzung von PLwffs in die ab-Notation. In der Implementierung der Junktorendefinitionen wird davon ausgegangen, dass "not" statt ¬, "and" statt ∧, "or" statt ∨, "implies" statt → und "iff" statt ↔ verwendet wird, um die Anwendung vorimplementierter Regeln auszuschalten. Dies ist wieder analog zur Verwendung von "plus", "minus", "times", "div", "power" im Falle von Q-wffs. Analog wird auch wieder ein entsprechender Hilfsbefehl für die Identifikation von Termen, die die Junktoren nicht enthalten, verwendet:

```
freePLQ[term_] := FreeQ[term, and] && FreeQ[term, or] &&
   FreeQ[term, implies] && FreeQ[term, iff] && FreeQ[term, not];
```

```
? freePLQ
```

freePLQ[term] identifies terms that do not contain the logical operators not,and,or,implies,iff.

Unter diesen Voraussetzungen lautet die Implementierung der Junktorenregeln:

```
notDef[not[term_]] :=
  Module[{nterm},
   nterm = Sort[Replace[term, {a → b, b → a}, 2, Heads -> True]];
   nterm];

andDef[and[conj1_, conj2_]] :=
  Module[{apgs1, apgs2, bpgs1, bpgs2, napgs, nbpgs1, nbpgs2, nbpgs3, nbpgs, nterm},
   apgs1 = Apply[List, conj1[[1]]];
   bpgs1 = Apply[List, conj1[[2]]];
   apgs2 = Apply[List, conj2[[1]]];
   bpgs2 = Apply[List, conj2[[2]]];
   napgs = Tuples[{apgs1, apgs2}];
   napgs = Map[Flatten, napgs, 2];
   napgs = Apply[a, napgs];
   nbpgs1 = Tuples[{apgs1, bpgs2}];
   nbpgs1 = Map[Flatten, nbpgs1, 2];
   nbpgs2 = Tuples[{bpgs1, apgs2}];
   nbpgs2 = Map[Flatten, nbpgs2, 2];
   nbpgs3 = Tuples[{bpgs1, bpgs2}];
   nbpgs3 = Map[Flatten, nbpgs3, 2];
   nbpgs = Join[nbpgs1, nbpgs2, nbpgs3];
   nbpgs = Apply[b, nbpgs];
   nterm = {napgs, nbpgs};
   nterm];
```

```
orDef[or[disj1_, disj2_]] :=
  Module[{apgs1, apgs2, bpgs1, bpgs2, napgs1, napgs2, napgs3, napgs, nbpgs, nterm},
   apgs1 = Apply[List, disj1[[1]]];
   bpgs1 = Apply[List, disj1[[2]]];
   apgs2 = Apply[List, disj2[[1]]];
   bpgs2 = Apply[List, disj2[[2]]];
   napgs1 = Tuples[{apgs1, apgs2}];
   napgs1 = Map[Flatten, napgs1, 2];
   napgs2 = Tuples[{apgs1, bpgs2}];
   napgs2 = Map[Flatten, napgs2, 2];
   napgs3 = Tuples[{bpgs1, apgs2}];
   napgs3 = Map[Flatten, napgs3, 2];
   napgs = Join[napgs1, napgs2, napgs3];
   napgs = Apply[a, napgs];
   nbpgs = Tuples[{bpgs1, bpgs2}];
   nbpgs = Map[Flatten, nbpgs, 2];
   nbpgs = Apply[b, nbpgs];
   nterm = {napgs, nbpgs};
   nterm];
impliesDef[implies[ant_, con_]] :=
  Module[{apgs1, apgs2, bpgs1, bpgs2, napgs1, napgs2, napgs3, napgs, nbpgs, nterm},
   apgs1 = Apply[List, ant[[1]]];
   bpgs1 = Apply[List, ant[[2]]];
   apgs2 = Apply[List, con[[1]]];
   bpgs2 = Apply[List, con[[2]]];
   napgs1 = Tuples[{apgs1, apgs2}];
   napgs1 = Map[Flatten, napgs1, 2];
   napgs2 = Tuples[{bpgs1, bpgs2}];
   napgs2 = Map[Flatten, napgs2, 2];
   napgs3 = Tuples[{bpgs1, apgs2}];
   napgs3 = Map[Flatten, napgs3, 2];
   napgs = Join[napgs1, napgs2, napgs3];
   napgs = Apply[a, napgs];
   nbpgs = Tuples[{apgs1, bpgs2}];
   nbpgs = Map[Flatten, nbpgs, 2];
   nbpgs = Apply[b, nbpgs];
   nterm = {napgs, nbpgs};
   nterm];
```

```
iffDef[iff[ant_, con_]] :=
  Module[
   {apgs1, apgs2, bpgs1, bpgs2, napgs1, napgs2, napgs, nbpgs1, nbpgs2, nbpgs, nterm},
   apgs1 = Apply[List, ant[[1]]];
   bpgs1 = Apply[List, ant[[2]]];
   apgs2 = Apply[List, con[[1]]];
   bpgs2 = Apply[List, con[[2]]];
   napgs1 = Tuples[{apgs1, apgs2}];
   napgs1 = Map[Flatten, napgs1, 2];
   napgs2 = Tuples[{bpgs1, bpgs2}];
   napgs2 = Map[Flatten, napgs2, 2];
   napgs = Join[napgs1, napgs2];
   napgs = Apply[a, napgs];
   nbpgs1 = Tuples[{bpgs1, apgs2}];
   nbpgs1 = Map[Flatten, nbpgs1, 2];
   nbpgs2 = Tuples[{apgs1, bpgs2}];
   nbpgs2 = Map[Flatten, nbpgs2, 2];
   nbpgs = Join[nbpgs1, nbpgs2];
   nbpgs = Apply[b, nbpgs];
   nterm = {napgs, nbpgs};
   nterm];
```

Zusätzlich zu diesen primitiven Regeln der Übersetzung in die ab-Notation werden noch - wie im Falle der Umformung von Ω-Ausdrücken in Q - primitive Regeln benötigt, um die resultierenden Ausdrücke innerhalb der ab-Notation in ideale kanonische Repräsentanten umzuformen. Diese Regeln bezeichne ich als ab-Regeln, da sie innerhalb der ab-Notation operieren. Ich werde die einzelnen Regeln auch als "S"-Regeln ("S" für "Simplify") kennzeichnen, denn sie erlauben Polgruppen zu vereinfachen. Im Rahmen des Finitismus sollen durch diese Regeln Eigenschaften von ab-Ausdrücken eliminiert werden, auf die eine etwaige Interpretation nicht zurückgreifen muss; gemäß Wittgensteins Terminologie sind dies keine "symbolisierenden Eigenschaften"; - sie gehören damit auch nicht zum idealen ab-Symbol, sondern nur zu ab-Zeichen auf dem Wege der Konstruktion des idealen ab-Symbols.
In der Formulierung der S-Regeln setze ich voraus, dass die Reihenfolge der Elemente einer Liste beliebig ist.
Ich verwende einen langen Unterstrich ___, um eine beliebige Anzahl weiterer Elemente einer Liste (Pole im Falle einer Liste von Polen, Polgruppen im Falle einer Liste von Polgruppen) anzudeuten. Diese Anzahl kann auch 0 sein. Dagegen verwende ich einen kürzeren Unterstrich __, um eine beliebige Anzahl weiterer Elemente anzudeuten, die allerdings nicht 0 sein darf. Es ist immer bedeutungslos, an welcher Stelle diese weiteren Elemente stehen, da die Reihenfolge der Elemente der Listen in keinem Fall eine Eigenschaft von ab-Ausdrücken ist (sie ist keine "symbolisierende Eigenschaft", d.i. keine formale Eigenschaft von ab-Ausdrücken, die für eine etwaige Interpretation eine Bedeutung spielen könnte).
Die Regel S11 erlaubt Polgruppen mit entgegengesetzten Polen a[𝒜]/ b[𝒜] durch {b} zu ersetzen.

$$\mathbf{S11:}\ \{\mathbf{a}[\mathscr{A}],\ b[\mathscr{A}],\ \_\_\_\} = \{b\}$$

Intuitiv zugänglicher wäre es im Hinblick auf eine Interpretation der ab-Ausdrücke statt {b} den Ausdruck {FALSE} zu wählen. Aber es soll durch die Verwendung von {b} deutlich gemacht werden, dass keine Interpretation vorausgesetzt wird. Viel wesentlicher ist, dass die Polgruppe keine atomare Formel mehr enthält. Sie enthält damit nicht mehr die Eigenschaft, dass "b" links von einer atomaren Formel steht: eine derartige etwaige Eigenschaft steht für eine spätere Interpretation nicht mehr zur Verfügung. Damit kann diese Polgruppe auch nicht mehr so interpretiert werden, dass die Wahrheit oder Falschheit eines Satzes von der Wahrheit oder Falschheit etwaiger Instanzen atomarer Ausdrücke in der Polgruppe abhängt: atomare Ausdrücke kommen ja gar nicht mehr vor! Die formale Eigenschaft ist vielmehr schlicht das Vorkommen von "b". Die naheliegende Interpretation von einer etwaigen a-Polgruppe a[{b}] ist z.B., dass auf Grund der junktorenlogischen Verknüpfung atomarer Ausdrücke in der Ausgangsformel keine möglichen Bedingungen der Wahrheit für Instanzen der Ausgangsformel angegeben werden können. b[{a}] könnte demgegenüber so interpretiert werden, dass die Falschheit von Instanzen der Ausgangsformel bedingslos gegeben ist; sie hängt nicht von dem spezifischen Wahrheitswert irgendeines atomaren Satzes ab. Dementsprechend ist {a[{b}],b[{a}]} das ab-Symbol für Kontradiktionen. Ein derartiger Ausdruck ist nicht mehr bipolar, da "a" bzw. "b" nicht mehr links oder rechts von atomaren Formeln vorkommen. Die a- bzw. b-Polgruppen sind, wenn man so will, "Pseudo"-Polgruppen (ähnlich wie die {0} in der Ω-Notation ein Pseudo-Zähler ist).

Regel S12 erlaubt {b}-Polgruppen in einer Liste von Polgruppen zu eliminieren, solange hieraus keine leere Liste resultiert

(deshalb der kürzere Unterstrich).

$$\mathbf{S12:}\ \{\{b\}, \_\_\} = \{\_\_\}$$

Aus dieser Regel folgt, dass eine Explikation der Bedingungen der Wahrheit oder Falschheit eines Satzes nicht auf einen Ausdruck der Form {b} rekurrieren kann.
Regel S5 führt eine Polgruppe {a} ein:

$$\mathbf{S5:}\ \{\{a[\neg]\}, \{b[\neg]\}, \_\_\_\} = \{\{a\}\}$$

S5 erlaubt eine Liste von Polgruppen durch {{a}} zu ersetzen, wenn eine PG genau einen Pol der Form a-¬ enthält und eine andere Polgruppe nur den entgegengesetzten Pol enthält.
S01 und S02 beziehen sich wieder nur auf eine Polgruppe:

$$\mathbf{S01:}\ \{b, \_\_\_\} = \{b\}$$
$$\mathbf{S02:}\ \{a, \_\_\} = \{\_\_\}$$

In S02 muss zusätzlich zu "a" mindestens ein weiterer Pol vorkommen, damit das Resultat nicht die leere Liste ist (deshalb der kürzere Unterstrich). Kommt nur "b" als ein Pol in einer Polgruppe vor, dann "symbolisieren" die restlichen Pole nicht mehr, d.h. sie können zur Explikation der Bedingungen von Wahrheit oder Falschheit von Sätzen nicht mehr herangezogen werden. Das Vorkommen von "a" als Pol neben anderen ist wiederum keine Eigenschaft von ab-Ausdrücken, die zum ab-Symbol gehören: der Pol kann für eine Explikation der Wahrheitsbedingungen von Sätzen übergangen werden.

$$\mathbf{S2:}\ \{P1, P1, \_\_\_\} = \{P1, \_\_\_\}$$

Duplikate an Polen können in Polgruppen gestrichen werden.

$$\mathbf{S3:}\ \{\{a[\neg], P1\_\_\}, \{b[\neg], P1\_\_\}\} = \{\{P1\_\_\}\}$$

P1__ steht hier für eine Aufzählung weiterer Pole in der Polgruppe, wobei die Aufzählung mindestens ein Pol enthalten muss. S3 erlaubt zwei Polgruppen die identisch sind bis auf genau einen entgegengesetzten Pol durch ihre Schnittmenge zu ersetzen.

$$\mathbf{S4:}\ \{\{P1\_\_\}, \{P1\_\_, \_\_\_\}, \_\_\_\} = \{\{P1\}, \_\_\_\}$$

P1 steht hier wieder für eine Aufzählung von Polen in der Polgruppe, wobei die Aufzählung mindestens ein Pol enthalten muss. S4 erlaubt eine Polgruppe PG2 in einer Liste von Polgruppen zu streichen, wenn es eine weitere Polgruppe PG1 in der Liste gibt, die eine echte oder unechte Teilmenge von PG2 ist.

Alle diese S-Regeln sind wieder einfache, explizite Ersetzungsregeln. Sie sind jeweils Regeln im Sinne von Operationen. Zusammengenommen mit Def. ¬ und den Definitionen der Junktoren bilden sie die PL-Operationen. Für den PL-Algorithmus werden keine weiteren primitiven Regeln benötigt. In der Implementierung der S-Regeln werden diese Regeln auf die jeweiligen Ausdrücke maximal angewendet:

```
S01[pg_] :=
  Module[{npg},
   If[Not[FreeQ[pg, b, 1]], npg = {b}, npg = pg];
   npg];

S02[pg_] :=
  Module[{npg},
   If[Not[FreeQ[pg, a, 1]] && Length[pg] > 1, npg = DeleteCases[pg, TRUE], npg = pg];
   npg];
```

```
S11[pg_] :=
  Module[{apoles, bpoles, test, npg},
   apoles = Cases[pg, pole_ /; Head[pole] === a, 1];
   apoles = Map[#[[1]] &, apoles];
   bpoles = Cases[pg, pole_ /; Head[pole] === b, 1];
   bpoles = Map[#[[1]] &, bpoles];
   test = Intersection[apoles, bpoles];
   If[test =!= {}, npg = {b}, npg = pg];
   npg];

S12[pgs_] :=
  Module[{npgs},
   npgs = DeleteCases[pgs, pg_ /; Not[FreeQ[pg, b, 1]], 1];
   If[npgs === {}, npgs = {{b}}];
   npgs];
S2[pg_] :=
  Module[{npg},
   npg = DeleteDuplicates[pg];
   npg];

S3[{pg1_, pg2_}] :=
  Module[{c1, c2, npg},
   c1 = Complement[pg1, pg2];
   c2 = Complement[pg2, pg1];
   If[Length[c1] === 1 && Length[c2] === 1 && Head[c1[[1]]] =!= Head[c2[[1]]] &&
     c1[[1, 1]] === c2[[1, 1]], npg = {Intersection[pg1, pg2]}, npg = {pg1, pg2}];
   npg];
S4[pgs_] :=
    Module[{npgs, pg, i},
   npgs = Union[pgs];
   npgs = Sort[npgs, Length[#1] < Length[#2] &];
    i = 1;
    While[i <= Length[npgs],
    pg = npgs[[i]];
    npgs = DeleteCases[npgs, lists_List /;
       (lists =!= pg && Sort[Intersection[pg, lists]] == Sort[pg]), {1}];
    i = i + 1];
   Union[npgs];
 npgs];
S5[pgs_] :=
  Module[{singles, asingles, bsingles, test, npgs},
   singles = Cases[pgs, pg_ /; Length[pg] === 1, 1];
   If[singles =!= {} && Length[singles] > 1,
    asingles = Cases[singles, apoles_ /; Head[apoles] === a, {2}];
    asingles = Map[#[[1]] &, asingles];
    bsingles = Cases[singles, bpoles_ /; Head[bpoles] === b, {2}];
    bsingles = Map[#[[1]] &, bsingles];
    test = Intersection[asingles, bsingles], test = {}];
   If[test =!= {}, npgs = {{a}}, npgs = pgs];
   npgs];
```

2.3 Zusammengesetzte Regeln

Die zusammengesetzten Regeln im PL-Algorithmus regeln wiederum nichts anderes als die Abfolge der Anwendung der primitiven Regeln zum Zwecke der Umformung von PL-wffs in ab-Symbole als ihre idealen logischen Repräsentanten. Der oberste Befehl "decideId" beruht nach der anfänglichen Prüfung auf Wohlgeformtheit durch die Regel wffPL auf der "Analyse" der PL-wffs mittels Umformung in ab-Symbole beginnend mit der Übersetzung der atomaren Ausdrücke und anschließender Übersetzung der Junktoren. wffPL ist analog zu wff des Q-Deciders, termAnalysisPLCore zu termAnalysisCore, termAnalysisPL zu termAnalysis und decidePL zu decideId:

```
wffPL[input_] :=
  Module[{atoms, noatoms, illand, illor, illimplies, illiff, illnot},
   atoms = Cases[input,
     part_ /; Head[part] =!= and && Head[part] =!= or && Head[part] =!= implies &&
       Head[part] =!= iff && Head[part] =!= not, {0, Infinity}];
   noatoms = Cases[atoms, part_ /; Head[part] =!= Symbol, 1];
   If[noatoms =!= {}, Return["input contains inadmissible signs. "]];
   illand =
    Cases[input, part_ /; Head[part] === and && Length[part] =!= 2, {0, Infinity}];
   If[illand =!= {}, Return["input contains sign and without two arguments. "]];
   illor =
    Cases[input, part_ /; Head[part] === or && Length[part] =!= 2, {0, Infinity}];
   If[illor =!= {}, Return["input contains sign or without two arguments. "]];
   illimplies = Cases[input,
     part_ /; Head[part] === implies && Length[part] =!= 2, {0, Infinity}];
   If[illimplies =!= {}, Return[
     "input contains sign implies without two arguments. "]];
   illiff = Cases[input, part_ /; Head[part] === iff && Length[part] =!= 2,
     {0, Infinity}];
   If[illiff =!= {}, Return["input contains sign iff without two arguments. "]];
   illnot = Cases[input, part_ /;
      Head[part] === not && Length[part] =!= 1 && Length[part] =!= 2, {0, Infinity}];
   If[illnot =!= {}, Return[
     "input contains sign not without one or two arguments. "]];
   input];
```

wffPL ist neben freeQPL der einzige Hilfsbefehl.

```
termAnalysisPLCore[term_] :=
  Module[{nterm1, nterm2, nterm3, nterm4, nterm5, nterm6},
   nterm1 = term /. not[notterm_ /; freePLQ[notterm]] :> notDef[not[notterm]];
   nterm2 = nterm1 /. and[conj1_ /; freePLQ[conj1], conj2_ /; freePLQ[conj2]] :>
      Stotal[andDef[and[conj1, conj2]]];
   nterm3 = nterm2 /. or[disj1_ /; freePLQ[disj1], disj2_ /; freePLQ[disj2]] :>
      Stotal[orDef[or[disj1, disj2]]];
   nterm4 = nterm3 /. implies[ant_ /; freePLQ[ant], con_ /; freePLQ[con]] :>
      Stotal[impliesDef[implies[ant, con]]];
   nterm5 = nterm4 /. iff[ant_ /; freePLQ[ant], con_ /; freePLQ[con]] :>
      Stotal[iffDef[iff[ant, con]]];
   nterm5];

termAnalysisPL[term_] := FixedPoint[termAnalysisPLCore, term];

decidePL[input_] :=
  Module[{nterm},
   nterm = wffPL[input];
   If[StringQ[nterm], Print[nterm]; Return["input not well formed"]];
   nterm = input /. atom_ /; freePLQ[atom] :> ptoapb[atom];
   nterm = termAnalysisPL[nterm];
   nterm];
```

decidePL gibt einfach das Resultat der Analyse aus. Soll hierauf die Entscheidung beruhen, ob die input-Formel die formale Eigenschaft besitzt, eine Tautologie zu sein, ist folgender, nicht-implementierter Befehl hinzuzufügen:

```
If[nterm === {a[{a}], b[{b}]}, nterm = LOGICAL TRUE, nterm = NOT LOGICAL TRUE]
```

Das syntaktische Kriterium für “logische Wahrheit” ist die Umformung auf das ab-Symbol {a[{a}],b[{b}}.
termAnalysisPLCore rekurriert neben den Junktorendefinitionen nur noch auf die zusammengesetzte Regel Stotal zur eindeutigen Vereinfachung der ab-Ausdrücke. Diese Regel beruht auf den zwei weiteren zusammengesetzten Regeln “Merge” und “SCore” (damit sind dann auch schon alle Regeln des Programmes genannt):

```
Merge[pgs_] :=
  Module[{npgs, length, mpgs, lengthb, mpgs1, mpgs2, nmpgs, mpgpairs},
   npgs = Sort[pgs, Length[#1] < Length[#2] &];
   length = Length[Last[npgs]];
   While[length > 1, mpgs = Cases[npgs, pg_ /; Length[pg] === length, {1}];
    mpgs = Sort[mpgs, Length[Cases[#1, p_ /; Not[FreeQ[p, b]]]] <
        Length[Cases[#2, p_ /; Not[FreeQ[p, b]]]] &];
    If[mpgs =!= {}, lengthb = Length[Cases[Last[mpgs],
        pg_ /; Not[FreeQ[pg, b]], {1}]]];
    While[lengthb > 0,
     mpgs1 =
      Cases[mpgs, pg_ /; Length[Cases[pg, p_ /; Not[FreeQ[p, b]], 1]] === lengthb];
     mpgs2 = Cases[mpgs, pg_ /; Length[Cases[pg, p_ /; Not[FreeQ[p, b]], 1]] ===
         lengthb - 1];
     If[mpgs1 =!= {} && mpgs2 =!= {}, mpgpairs = Tuples[{mpgs1, mpgs2}];
      nmpgs = Map[Flatten, Map[S3, mpgpairs], 1];
      npgs = Union[npgs, nmpgs]];
     lengthb = lengthb - 1]; npgs = S4[npgs];
    length = length - 1];
   npgs];

SCore[pgs_] :=
  Module[{npgs01, npgs02, npgs11, npgs12, npgs2, npgs34, npgs5},
   npgs01 = Map[S01, pgs];
   npgs02 = Map[S02, npgs01];
   npgs11 = Map[S11, npgs02];
   npgs12 = S12[npgs11];
   npgs2 = Map[S2, npgs12];
   npgs34 = Merge[npgs2];
   npgs5 = S5[npgs34];
   npgs5];

Stotal[pgs_] :=
  Module[{apgs, napgs, bpgs, nbpgs, npgs},
   apgs = Apply[List, pgs[[1]]];
   napgs = SCore[apgs];
   napgs = Apply[a, napgs];
   bpgs = Apply[List, pgs[[2]]];
   nbpgs = SCore[bpgs];
   nbpgs = Apply[b, nbpgs];
   npgs = {napgs, nbpgs};
   npgs];
```

Stotal[pgs] ersetzt zuerst die a-Polgruppen des ab-Ausdruckes durch eine Liste. Diese Liste wird dann durch SCore vereinfacht; abschließend wird der Kopf “List” wieder durch “a” ersetzt, so dass eine vereinfachte Liste von a-Polgruppen resultiert. Dasselbe geschieht mit den b-Polgruppen. Abschließend wird dann wieder ein ab-Ausdruck, d.i. eine Liste von a- und b-Polgruppen zurückgegeben.

SCore wendet auf eine Liste von Polgruppen die S-Regeln an. Zunächst werden durch S01 in Polgruppen, die den Pol “b” enthalten, alle anderen Pole eliminiert. S02 eliminiert den Pol “a” in einzelnen Polgruppen. S11 ersetzt dann Polgruppen mit entgegengesetzten Polen durch “{b}” und S12 eliminiert derartige Polgruppen in der Liste von Polgruppen (soweit dies nicht zu einer leeren Liste führt). S2 eliminiert Duplikate von Polen in den Polgruppen. Die einzige etwas komplexere Regel ist die zusammengesetzte Regel “Merge”. Diese Regel wendet S3 an, um zwei Polgruppen, die identisch sind bis auf einen entgegengesetzten Pol, durch ihre Schnittmenge zu ersetzen. Infolgedessen kann S4 zu weiteren Vereinfachungen führen. Die genaue Anwendung von S3 und S4 dient dem Zweck auf Basis dieser zwei Regeln eine maximale und eindeutige Vereinfachung zu erziehlen. Wer den Quine-McCluskey Algorithmus kennt, erkennt in dieser Regel das Analogon zur Vereinfachung der Minterme in Primterme. SCore endet

mit der Anwendung von S5 um gegebenfalls eine Liste von Polgruppen, die relativ zu den genannten Regeln maximal vereinfacht sind, durch {a} zu ersetzen. Ich erspare dem Leser die Details von SCore, da es für das Verständnis des gesamten Algorithmus am Einfachsten ist, wenn man in ihm insgesamt ein Äquivalent zur Umformung von PL-wffs in sogenannte "reduced disjunctive normal forms" (RDNF) sieht. Dies werde ich im folgenden Abschnitt erklären.
Auf einen Terminierungsbeweis werde ich hingegen verzichten; erstens wäre er weitgehend analog zu Q, zweitens trivial und drittens wird durch den Korrektheitsbeweis offensichtlich, dass der Algorithmus weitgehend analog ist zu bekannten Algorithmen der Umformung von PL-wffs in RDNFs.

2.4 Korrektheit

Die "Korrektheit" des Verfahrens soll im Folgenden dargelegt werden, indem gezeigt wird, dass es durchweg auf bekannten aussagenlogischen Äquivalenzumformungen beruht, die zu demselben Resultaten führen wie Umformungen in RDNF (d.i. dem Resultat des 1. Schrittes des Quine-McCluskey Algorithmus). Da der Algorithmus ohne die Merge-Regel weitgehend der Umformung in kanonische disjunktive Normalformen (CDNF) gleicht und da diese wiederum auf Wahrheitswerttabellen abgebildet werden können, ließe sich sogar die Korrektheit des Verfahrens durch Vergleich mit der Semantik von PL nachweisen. Schließlich liessen sich in die Wahrheitswerttabellenmethode auch noch ein Pendant zur Merging-Regel einführen (man vergleiche etwa Baumkalküle), um vollständige Übereinstimmung zu erzielen. Hierauf wird aber verzichtet. Es sei auch noch einmal betont, dass nach finitistischer Auffassung weder die bekannten PL-Schlussregeln und Kalküle noch die Semantik von PL als Maßstab dienen. Vielmehr handelt es sich jeweils um autonome Regelsysteme, die sich ineinander übersetzen lassen und in diesem Sinne "äquivalent" sind, auch wenn das ihnen jeweils zu Grunde liegende Verständnis unterschiedlich ist und die daraus resultierenden Anwendungen unterschiedlich ausfallen können; - die rein logische "Praxis" ist insofern übereinstimmend als die logischen Formeln gleich klassifiziert werden: die Extension der "allgemeingültigen" Formeln (wahr in allen Interpretationen) ist z.B. nicht nur identisch mit der der Theoreme (in einem Kalkül ableitbaren Formeln), sondern auch mit der der Tautologien (Ausdrücke, die auf Grund ihrer formalen Eigenschaften nicht falsch sein können und deren Wahrheit von keiner Interpretation atomarer Ausddrücke abhängt = Ausdrücke, deren ab-Symbol {a[{a}],b[{b}} ist]).

Im Folgenden wird per Induktion bewiesen, dass das ab-Symbol, in das der Algorithmus eine PL-wff A umformt, äquivalent ist mit RDNF(A) ∨ RDNF(¬A), wobei die a-Polgruppen (a-PGs) äquivalent sind mit RDNF(A) und die b-PGs mit RDNF(¬A). Der Algorithmus formt eine PL-wff von innen nach aussen um, wobei jeder Teilausdruck in das entsprechende ab-Symbol umgeformt wird. Es wird folglich bewiesen, dass
Anwendung von Def. $\mathscr{A}$ äquivalent zur Umformung einer atomare Formel $\mathscr{A}$ in RDNF($\mathscr{A}$) ∨ RDNF(¬$\mathscr{A}$) ist (Verankerung) und dass unter der Voraussetzung, dass die Übersetzung von Teilausdrücken A in ab-Symbol äquivalent ist mit ihrer Umformung in RDNF(A) ∨ RDNF(¬A), dies auch für die Umformung der nächst komplexeren Teilausdrücke gilt. Hierbei werden folgende Übersetzungsregeln vorausgesetzt: a[$\mathscr{A}$] = $\mathscr{A}$, b[$\mathscr{A}$] = !$\mathscr{A}$; ein einzelner Pol "a" ist mit "TRUE" zu übersetzen, ein einzelner Pol "b" mit "FALSE"; Pole innerhalb von PGs werden konjunktiv verknüpft, PGs disjunktiv. DNF1 × DNF2 (= Cartesisches Produkt zweier DNFs) steht für die Umformung von DNF1 ∧ DNF2 in eine DNF.
Verankerung: Übersetzung des Definiens von Def. $\mathscr{A}$: $\mathscr{A}$ = {a[{a[$\mathscr{A}$]}], b[{b[$\mathscr{A}$]}]} lautet $\mathscr{A}$ ∨ ¬ $\mathscr{A}$. Dies ist RDNF($\mathscr{A}$) ∨ RDNF(¬$\mathscr{A}$).
Induktionsklausel für ¬-Teilausdrücke : Hier muss - anders als bei den anderen Junktorendefinitionen - gemäß termAnalysisCore nur Def.¬ angewendet werden. Def.¬ ist: ¬[{a[PG1],b[PG2]}] = {a[PG2],b[PG1]}. PG1 ist gemäß Übersetzungsregel und Induktionsvoraussetzung übersetzt RDNF1 und äquivalent mit der negierten PL-wff A und b[PG2] übersetzt RDNF2 und äquivalent mit PL-wff ¬A. Das Definiens von Def.¬ kehrt die Zuordnung der Polgruppen zu äußeren Polen um; damit wird RDNF2 nun der Negation ¬A zugeordnet, was laut Voraussetzung richtig ist, und RDNF1 der Negation der Negation von A, also ¬¬A, was gemäß DNB (Regel der doppelten Negationsbeseitigung) äquivalent mit A ist und somit wieder gemäß Voraussetzung richtig ist. Das Resultat sind demnach wieder RDNFs, wobei die eine mit dem übersetzten Teilausdruck und die andere mit der Negation des übersetzten Teilausdruckes äquivalent ist.
Induktionsklausel für ∧-Teilausdrücke: Es ist zu zeigen, dass Anwendung von S[∧Def.[conj1,conj2]] unter der Induktionsvoraussetzung zu einem ab-Symbol führt, dessen a-PGs äquivalent sind mit einer RDNF der entsprechenden PL-wff A (also der Konjunktion aus den RDNF, die conj1 und conj2 entsprechen) und dessen b-PGs äquivalent sind mit einer RDNF von ¬A.
Def.∧: ∧[{a[PG1],b[PG2]},{a[PG3],b[PG4]}] = {a[PG1×PG3],b[PG1×PG4,PG2×PG3,PG2×PG4]]}
Laut Voraussetzungen ist PG1 = RDNF1; PG2 = RDNF2, RDNF2 -||- ¬RDNF1, PG3 = RDNF3; PG4 = RDNF4, RDNF4 -||- ¬RDNF3. A = RDNF1 ∧ RDNF3. Es ist zu zeigen, dass
1) (RDNF1 ∨ RDNF2) ∧ (RDNF3 ∨ RDNF4) -||- (RDNF1 × RDNF3) ∨ (RDNF1 × RDNF4) ∨ (RDNF2 × RDNF3) ∨ (RDNF2 × RDNF4).
Dies folgt aus DIS1: (A ∨ B) ∧ C -||- (A ∧ C) ∨ (B ∧ C). RDNFA × RDNFB steht dabei für die Umformung von RDNFA ∧

RDNFB ebenfalls unter Anwendung von DIS1 in eine DNF.
2) RDNF1 ∧ RDNF3 -||- RDNF1 × RDNF3. Folgt aus Übersetzung von × und DIS1.
3) ¬(RDNF1 ∧ RDNF3) -||- (RDNF1 × RDNF4) ∨ (RDNF2 × RDNF3) ∨ (RDNF2 × RDNF4). Aus RDNF2 = ¬RDNF1 und RDNF4 = ¬RDNF3 folgt (RDNF1 × ¬RDNF3) ∨ (¬RDNF1 × RDNF3) ∨ (¬RDNF1 × ¬RDNF3), d.i. auf Grund der Übs. von × äquivalent mit (RDNF1 ∧ ¬RDNF3) ∨ (¬RDNF1 ∧ RDNF3) ∨ (¬RDNF1 ∧ ¬RDNF3). Diese Formel ist aus ¬(RDNF1 ∧ RDNF3) unter alleiniger Anwendung von Äquivalenzgesetzen abzuleiten:

Nr.	Formel	Regel
(1)	¬ (RDNF1 ⋀ RDNF3)	AE
(2)	¬ RDNF1 ∨ ¬ RDNF3	DMG ∧
(3)	(¬ RDNF1 ∧ (RDNF3 ∨ ¬ RDNF3)) ∨ (¬ RDNF3 ∧ (RDNF1 ∨ ¬ RDNF1))	2×TE
(4)	¬ RDNF1 ∧ RDNF3 ∨ ¬ RDNF1 ∧ ¬ RDNF3 ∨ ¬ RDNF3 ∧ RDNF1 ∨ ¬ RDNF3 ∧ ¬ RDNF1	DIS1
(5)	¬ RDNF1 ⋀ RDNF3 ⋁ ¬ RDNF1 ⋀ ¬ RDNF3 ⋁ ¬ RDNF3 ∧ RDNF1	∨ E
(6)	RDNF1 ∧ ¬ RDNF3 ∨ ¬ RDNF1 ∧ RDNF3 ∨ ¬ RDNF1 ∧ ¬ RDNF3	ASS, KOM

Folglich sind die a-PG äquivalent mit einer DNF, die äquivalent ist mit A und die b-PG äquivalent mit einer DNF, die äquivalent ist mit ¬A.

Im Folgenden ist zu zeigen, dass Anwendung der Regel Stotal auf das Resultat der Anwendung von ∧Def. äquivalent ist zur Umformung dieser beiden DNFs in jeweils eine RDNF. Die Vereinfachung der PGs geschieht jeweils durch SCore - dieser Regel entspricht der Umformung von DNFs in RDNFs. Die Regeln bis "Merge" entsprechen der Umformung von DNFs in KDNFs (kanonische disjunktive Normalformen). Die Minterme (Disjunkte) der KDNFs enthalten nicht "FALSE" neben anderen Konjunkten. Dies wird durch S01 sicher gestellt. Diese Regel übersetzt A ∧ FALSE -||- FALSE. Die Minterme enthalten auch nicht "TRUE", dies wird durch S02 sicher gestellt. Diese Regel übersetzt A ∧ TRUE -||- A. Die Minterme enthalten auch keine Widersprüche. Dies wird durch S11 sicher gestellt. Diese Regel übersetzt A ∧ ¬ A ∧ B -||- FALSE. FALSE ist kein Minterm zusätzlich zu anderen Mintermen. Dies stellt S12 sicher. Diese Regel übersetzt A ∨ FALSE -||- A. Schließlich enthalten Minterme auch keine Konjunkte mehrmals. Dies wird durch S2 sichergestellt. Diese Regel übersetzt A ∧ A -||- A.

Die Übersetzung von PL-wffs in ab-Ausdrücke mittels der Junktorendefinitionen führt zu Äquivalenten von DNFs, die komplexer sein können als KDNFs. Gegeben n unterschiedliche atomare Formeln (die aber ggf. mehrmals in einer Formel vorkommen können), enthält jeder Minterm (jedes Disjunkt) einer KDNF genau n Konjunkte und die Anzahl von Disjunkten von KDNF(A) plus Anzahl von Disjunkten von KDNF(B) ist 2^n (analog zu den Zeilen einer Wahrheitswertta - belle). Die Junktorendefinitionen im Algorithmus zur Übersetzung von PL-wffs in ab-Ausdrücke orientieren sich dagegen nicht an der Gestalt atomarer Formeln, sondern an ihrem Vorkommen. Nur so können sie als reine Operationen (und nicht als Funktionen) definiert werden, die ohne weitere Fallunterscheidungen einen Ausdruck erzeugen, der sowohl ein Äquivalent zu einer PL-wff A als auch zu ¬A enthält. Für den Algorithmus ist das natürlich umständlich, aber der Zweck des Algorithmus ist nicht die Effektivität, sondern allein die Illustration des finitistischen Beweisver - ständnis im Rahmen von PL und hier kommt es darauf an, die Schritte der Übersetzung stets allein auf Operationen (minimale syntaktische Umformungen) beruhen zu lassen und nicht auf etwaigen Funktionen. Im ersten Schritt wird deshalb in Folge der Anwendung der Junktorendefinitionen eine Äquivalent zu "maximalen" DNFs geschaffen, die dann im nächsten Schritt, - dem 1. Teil von SCore - in ein Äquivalent zu KDNFs vereinfacht werden.

Der zweite Schritt von SCore - die Anwendung der Regel "Merge" - übersetzt die Umformung von KDNF in RDNF. S3 übersetzt die Merging Rule: A ∧ B ∨ A ∧ ¬B -||- A. S4 übersetzt A ∧ B ∨ B -||- B. In der Regel "Merge" werden die PGs zunächst nach ihrer Länge sortiert und dann jeweils Polgruppen derselben Länge, beginnend mit der längsten, verschmolzen. Polgruppen derselben Länge werden nach der Anzahl der darin vorkommenden nicht-negierten Variablen sortiert. Dann wird auf benachbarte Klassen S3 maximal angewendet (d.h. so, dass eine PG mit allen PGs verschmolzen wird, mit der sie sich nur in einem entgegengesetzten Pol unterscheidet). Dieses Verfahren wird rekursiv auf alle PGs derselben Länge angewendet bis S3 jeweils nicht mehr anwendbar ist. Durch S4 werden jeweils nach einem Schritt, in dem S3 maximal auf alle PGs einer bestimmten Länge angewendet wurde, alle PGs gestrichen, die ver - schmolzen wurden. Das Verfahren endet, nachdem alle PGs der Länge 2 berücksichtigt wurden. Um auch noch PGs der Länge 1 zu berücksichtigen, ist schließlich im letzten, 3. Schritt von SCore noch S5 anzuwenden. S5 übersetzt A ∨ ¬A ∨

B -||- TRUE. Um zu eindeutigen RDNFs zu gelangen, ist auch noch diese Regel anzuwenden; - ich rechne sie noch zum 1. Schritt des Quine-McCluskey Algorithmus dazu. Anwendung von "Merge" und abschließende Anwendung von S5 ist nichts anderes als die Abbildung des 1. Schrittes des Quine-McCluskey Algorithmus auf PGs, d.i. der Umformung von KDNFs in RDNFs. Der 2. Schritt des Quine-McCluskey Algorithmus - die Minimierung der Anzahl minimaler Disjunkte - darf hingegen nicht abgebildet werden, denn er ist nicht mehr eindeutig, wie man sich an der RDNF P ∧ ¬Q ∨ ¬P ∧ Q ∨ P ∧ R ∨ Q ∧ R klar machen kann, die sowohl mit P ∧ ¬Q ∨ ¬P ∧ Q ∨ P ∧ R als auch mit P ∧ ¬Q ∨ ¬P ∧ Q ∨ P ∧ R äquivalent ist. RDNFs sind dagegen maximale Disjunktionen minimaler Disjunkte und eindeutige Repräsentanten einer Klasse äquivalenter PL-wffs. Sie sind das Pendant zu teilerfremden Brüchen in ℚ und die jeweiligen Äquivalente in der Ω- und der ab-Notation sind ideale Repräsentanten von Äquivalenzklassen der input-Ausdrücke.

Da SCore in Stotal sowohl auf die a-PGs als auch auf die b-PGs angewendet wird, ist Resultat von Stotal[∧Def.[conj1,-conj2]] ein ab-Symbol, dessen a-PGs äquivalent sind mit einer RDNF der entsprechenden PL-wff A und dessen b-PGs äquivalent sind mit einer RDNF von ¬A, was zu zeigen war.

Da für die restlichen Arten der Teilausdrücke analog argumentiert werden kann, wobei allenfalls noch Definitionen der Junktoren heranzuzuiehen sind, und da ohnehin alle anderen Junktoren durch ¬ und ∧ definierbar sind, kann hier auf die Beweise der restlichen Induktionsklauseln verzichtet werden. Der wesentliche Punkt ist ohnehin nur der, vor Augen zu führen, dass finitistische PL-Beweise äquivalent sind mit der herkömmlichen logischen Praxis; sämtliche primitiven Regeln beruhen auf bekannten Schlussregeln. Wie in ℚ soll nur eine bekannte Praxis rekonstruiert werden und dabei ein Ausdruck verliehen werden, der jegliche Art von Semantik oder axiomatischem Beweisverständnis überflüssig macht, indem Logik verstanden wird als die Kunst der Identifikation logischer Eigenschaften im Sinne von Eigenschaften von Satzformen vermittels der algorithmischen Übersetzung von wffs in ideale Repräsentanten durch Anwendung logischer Operationen. Nach finitistischem Verständnis ist diese Praxis autonom und an keinem externen Maßstab zu messen.

2.5 Printbefehle

Es sind damit alle im Programm verwendeten Befehle erläutert. Allerdings wurden durchweg die Printbefehle, die die Ausgabe der Beweise regeln, übergangen.

Um sich die Beweise ausgeben zu lassen, muss zunächst das package PLdecider.m eingelesen werden (vorher sind ggf. alle eingelesenen Befehle aus dem Speicher zu entfernen):

```
Remove["Global`*"]
```

```
<< "D:\\Logik\\PL-Decider.m";
```

Zusätzlich muss der parameter "printproof1" gleich 1 gesetzt werden:

```
printproof1 = 1
```

```
1
```

Wenn dieser Parameter eingelesen ist, wird die lückenlose Anwendung aller primitiven Regeln ausgegeben.
Hierbei wird nur das Resultat der Regeln auf die jeweiligen Teilausdrücke ausgegeben. Um den Beweis zu strukturieren, so dass bis zu einem gewissen Grad ersichtlich wird, auf welchen Teilausdruck die jeweilige primitive Regel angewendet wird, ist zusätzlich der Parameter "printproof2" einzulesen, wobei dieser einem Wert zwischen 1 und 5 gleichzusetzen ist, je nachdem, wie detailiert der Beweis gegliedert werden soll. Grad 5 liefert die detallierteste Gliederung, die an jeder Stelle kenntlich macht, auf welchen Teilausdruck die Regeln angewendet werden.

```
printproof2 = 5
```

```
5
```

```
quine = or[or[or[and[P, not[Q]], and[not[P], Q]], and[P, R]], and[Q, R]]
```

```
or[or[or[and[P, not[Q]], and[not[P], Q]], and[P, R]], and[Q, R]]
```

```
decidePL[quine]
```

```
{a[{a[Q], b[P]}, {a[Q], a[R]}, {a[P], b[Q]}, {a[P], a[R]}],
 b[{b[P], b[Q]}, {a[Q], a[P], b[R]}]}
```