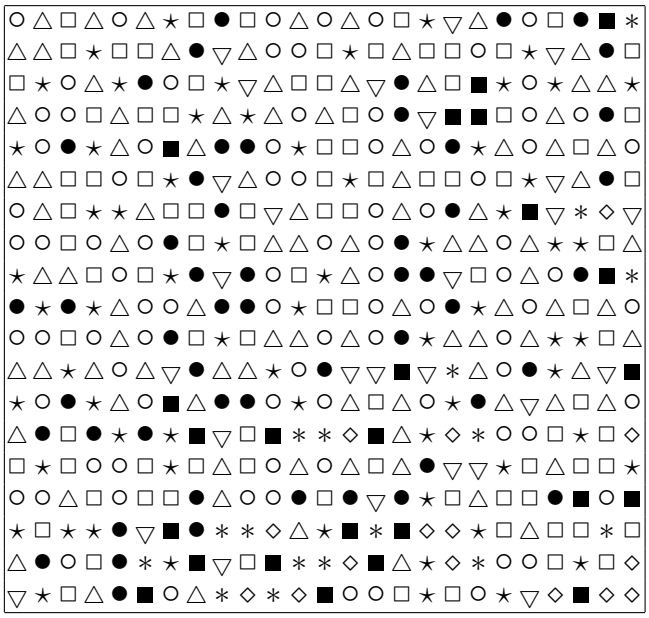
Morphogrammatik



Eine Einführung in die Theorie der logischen Form

Thomas Mahler

Abstract

Gotthard Günthers works on polycontextural logics emphasized the importance of morphogrammatics as a general pre-logical theory of the architecture of logic-systems. In spite of their theoretical significance morphogrammatics have not been explicated as a formal theory.
The present work locates morphogrammatics in the intersection of kenogrammatics and polycontextural logic and explains it against the background of these theories. Kenogrammatics are formally introduced as an underlying general theory of semiotic processes. Kenogrammatic concepts, structures and operations are then used to develop the mathematical theory of morphogrammatics. The following analytical part classifies and analyses combinatorical properties of morphogrammatic structures and operations. Part IV is dedicated to applications of morphogrammatics to logic and computer science. It gives an outline of the formal foundation of logical system (Günthers 'place-value logic' and polycontextural logic) by morphogrammatics. In the last chapter an implementation of the 'proemial-relation' is designed, which is suggested as an extension of functional programming and as an implementation technique for computational reflection and process communication.

Keywords: artificial intelligence, antinomies, autopoieses, circular systems, computational reflection, cybernetics, formal languages, foundations of mathematics, functional programming, kenogrammatics, logic, morphogrammatics, number theory, parallel processing, polycontextural logic, proemiality, process communication, selfreferential systems, semiotics, simultaneity.

Danksagung

Dieses Buch ist das Resultat meiner Mitarbeit am Forschungsprojekt 'Theorie komplexer biologischer Systeme' an der Ruhr Universität Bochum unter der Leitung von Dr. R. Kaehr. Ich danke allen Mitarbeitern des Projektes für zahlreiche Diskussionen und Anregungen.
Besonderen Dank schulde ich Rudolf Kaehr, der den Entwurf der Arbeit und die gesamte Entstehung des Manuskriptes engagiert betreute. Ich verdanke ihm in sehr vielen Detailfragen wichtige Hinweise, Hilfe und Kritik, ohne die das Buch in dieser Form nicht hätte entstehen können.
Bedanken möchte ich mich außerdem bei Johannes Joemann (Wissenschaftsladen Dortmund) und Georg Reichwein (Universität Lissabon, Portugal) für die fruchtbare Zusammenarbeit am Informatiklehrstuhl für künstliche Intelligenz an der Uni Dortmund, die wichtige Vorarbeiten für dieses Buch insbesondere auf den Gebieten 'Funktionale Programmierung', 'Meta-level Architekturen' und der 'Reflektionalen Programmierung' hervorbrachte.
Diese Arbeit wurde gefördert von der Volkswagen-Stiftung.

Thomas Mahler, Dortmund,
im Juli 1993

Übersicht

Inhaltsverzeichnis

Abbildungsverzeichnis

Teil I

Motivation

Kapitel 1

Einführung

Ich fliege von hier nach einem Weltteil, wo die Menschen nur unbestimmte oder wo sie gar keine Nachricht von der Möglichkeit des Fliegens haben. Ich sage ihnen, ich sei soeben von ... zu ihnen geflogen. Sie fragen mich, ob ich mich irren könnte. — Sie haben offenbar eine falsche Vorstellung davon, wie die Sache vor sich geht. (Wenn ich in eine Kiste gepackt würde, wäre es möglich, daß ich mich über die Art des Transportes irrte.) Sage ich ihnen einfach, ich könnte mich nicht irren, so wird sie das vielleicht nicht überzeugen; wohl aber, wenn ich ihnen den Vorgang beschreibe. Sie werden darin die Möglichkeit eines Irrtums *gewiß nicht in Frage ziehen. Dabei könnten sie aber — auch wenn sie mir trauen — glauben, ich habe geträumt oder ein Zauber habe mir das eingebildet.*

L. Wittgenstein

Das Thema der Arbeit

Die von Gotthard Günther entworfene Polykontexturale Logik (PKL) postuliert eine über den strukturellen Bereich klassischer formaler Systeme hinausreichende Formkonzeption, die es ermöglichen soll, komplexe, dialektische und selbstreferentielle Systeme nicht-reduktionistisch abzubilden.

Günthers Ansatz geht von der These aus, daß mit der transzendentalen Dialektik des deutschen Idealismus eine neuartige Konzeption der logischen Form entdeckt wurde, die jenseits der aristotelischen Formkonzeption stehe, aber ebenso wie diese einer philosophischen *und* mathematischen Analyse zugänglich sei [Gue33], [Gue78].

In seinen umfangreichen Arbeiten ([Gue80], Bd. 1–3) entwirft er eine selbstreferentielle Architektur zur Abbildung seiner *transklassischen* Formkonzeption. Die grundlegende Idee zur Realisierung einer solchen Architektur in der PKL ist es, diese als einen Mechanismus der Vermittlung distribuierter Logiken in einem komplexen Systemverbund darzustellen. Selbstreferentialität soll im Gesamtkomplex der logisch unabhängigen, jedoch außerlogisch verkoppelten, formalen Systeme — *Kontexturen*

— nicht-reduktionistisch und antinomienfrei abgebildet werden [Kae78].

Die Konzeption der PKL benötigt zur formalen Beschreibung der außerlogischen Verteilung und Vermittlung logischer Kontexturen eine spezielle prälogische Theorie, die Günther unter dem Namen *Morphogrammatik*[1] einführt ([Gue80b], [Gue80b1]). Die Morphogrammatik ist eine Theorie der Umformung und Verknüpfung sub-logischer Operationen, die Günther als Tiefenstruktur des klassischen Aussagenkalküls nachwies. Sie beschreibt allgemein die prälogische Architektur logischer Systeme. Günther verwendete sie insbesondere zur Fundierung seiner logischen Konzeptionen der Stellenwertlogik und der PKL. „*Die Morphogrammatik [beschreibt] eine Strukturschicht, in der die Differenz zwischen Subjektivität und Objektivität erst etabliert wird und deshalb dort noch nicht vorausgesetzt werden kann.*“[2] Dieser prälogische Charakter der Morphogrammatik ermöglicht die formal widerspruchsfreie Abbildung der gegen die Axiomatik der Logiken verstoßenden Vermittlung mehrerer Logiken in einem polykontexturalen Verbund.
Trotz ihrer fundamentalen Bedeutung für die Konzeption der PKL wurde die Morphogrammatik bisher nur fragmentarisch als formale Theorie ausgearbeitet. Dieses Buch rekonstruiert die bisherigen Forschungsarbeiten zur Morphogrammatik und leistet die formale Ausarbeitung und Ergänzung des bruchstückhaften historischen Materials vor dem Hintergrund der Polykontexturalen Logik.

Anlaß

Eine erste Fassung dieses Buches erschien als Arbeitsbericht des Forschungsprojektes „Theorie komplexer biologischer System“ der Volkswagenstiftung [Mah93]. Das Ziel dieses Projektes war zum einen die Formalisierung der „Theorie Autopoietischer Systeme“ (TAS) [Mat85], [Mat87] mit den formalen Methoden der Polykontexturalen Logik. Zum anderen sollte die PKL mit der TAS als semantischem Modell inhaltlich interpretiert werden.
Im Rahmen des Forschungsprojektes erwies sich die umfassende Rekonstruktion und Aufarbeitung der Morphogrammatik vor allem als notwendig, da sie wichtige Konzepte der PKL beschreibt. Ein weiterer Grund war, daß die frühen grundlegenden Arbeiten zur PKL und zur TAS zeit- und raumgleich am Biological Computer Laboratory (BCL) der University of Illinois entstanden. Wichtige Arbeiten zur Morphogrammatik, die bislang kaum erschlossen und berücksichtigt wurden, entstanden im unmittelbaren Kontext der BCL-Arbeiten. Die Rekonstruktion dieser Forschungen versprach daher vertiefende Einblicke in die PKL und die TAS zu gewähren und den Hintergrund ihrer gemeinsamen Enstehung zu erhellen. Da in der aktuellen interdisziplinären Diskussion der „Second Order Cybernetics“ (SOC) und Systemtheorie die Theorie Autopoietischer Systeme eine breite Akzeptanz als methodisches Paradigma einer Reihe von Einzeldisziplinen (u.a. Biochemie, Neurobiologie, Organische Chemie, Geschichte, Soziologie, Psychologie, Kommunikationswissenschaften, Ökonomie, Ökologie, Physik, Systemtheorie) gefunden und eine Zahl von Diskursen eröffnet hat, erschien es wünschenswert, die Arbeit in überarbeiteter und erweiterter Form einer größeren Öffentlichkeit zu präsentieren.
Im Umfeld der Rezeption und Diskussion der Arbeiten zur TAS und SOC hat eine umfangreiche Auseinandersetzung mit den am BCL durchgeführten Arbeiten [BCL76]

[1] Gr. morphe: Form
[2] [Gue80], Bd.1, S. 216.

begonnen. Am BCL entstanden in der Zeit von 1956 bis 1974 unter anderen die Second Order Cybernetics (Heinz von Foerster, Lars Löfgren, Gordon Pask), die Theorie Autopoietischer Systeme (Humberto Maturana, Francisco Varela) und die Polykontexturale Logik (Gotthard Günther). Desweiteren wurden am BCL bedeutende Pionierarbeiten zur Kybernetik, Systemtheorie, Selbstorganisationstheorie und des Konnektionismus geleistet. Die frühen Arbeiten Maturanas zur TAS sowie die wichtigsten Arbeiten Günthers und anderer zur PKL entstammen der gemeinsamen Arbeit am BCL. Die in dieser Arbeit geleistete Rekonstruktion und Aufarbeitung einiger schwer zugänglicher Arbeiten ([Na64], [Sch67a], [Sch67b], [Sch67c], [And65]) stellt einen Beitrag zu der aktuellen Rezeption der BCL-Arbeiten dar.

Der Grund des zunehmenden Interesses an der Second Order Cybernetics, der Theorie Autopoietischer Systeme und der Polykontexturalen Logik dürfte in der diesen Theorien gemeinsamen Kritik der formalen Methoden der Mathematik und Logik und dem Versuch der Erweiterung dieser klassischen Methoden zu finden sein.
Diese Kritik geht schon auf die Auseinandersetzung Hegels und Fichtes mit den Verfahren der Mathematik und Logik zurück. Die Relevanz dieser philosophischen Untersuchungen besteht darin, daß sie die im 20. Jahrhundert in zunehmendem Ausmaß zutage tretenden Probleme der naturwissenschaftlichen Disziplinen vorwegnehmend auf die Struktur der von den Naturwissenschaften verwendeten formalen Methoden zurückführen.
So rührt beispielsweise die von der SOC thematisierte Beobachter-Problematik daher, daß die Wechselwirkung eines Beobachter mit der von ihm beobachteten Welt eine fundamental selbstreferentielle nicht ohne Komplexitätsverlust reduzierbare Ganzheit darstellt, weil der Beobachter stets auch Teil seiner Beobachtungswelt ist[3]. Die Probleme bei der formalen Beschreibung dieser Situation beruhen nach der angeführten Kritik auf der Unfähigkeit klassischer formaler Methoden, selbstreferentielle Systeme widerspruchsfrei abzubilden.
Selbstreferentielle Strukturen sind in der Mathematik und Logik als Paradoxien, Antinomien und als *circuli vitiosi* bekannt, die sich in *klassischen* Kalkülen[4] als syntaktisch wohlgeformte Ausdrücke erzeugen lassen, denen jedoch im Rahmen der Fixpunktsemantiken keine Werte aus einer semantischen Domäne zugeordnet werden können. Aufgrund dieser Nichtevaluierbarkeit ist die klassische Mathematik stets bemüht gewesen, selbstreferentielle Strukturen aus ihrem Beschreibungsbereich schon syntaktisch auszuschließen[5], wie es sich etwa die Theorie der logischen Typen[6] zur expliziten Aufgabe gemacht hat.
Die formalen Arbeiten der Second Order Cybernetics setzen sich mit diesen grundlegenden Limitationen der klassischen Mathematik und Logik auseinander und versuchen Konzepte zu entwickeln, die auch die formale Beschreibbarkeit und Evaluierbarkeit fundamental selbstreferentieller Strukturen ermöglichen.
Die Mängel der klassischen formalen Methoden und die Notwendigkeit ihrer Erweiterung ist in zunehmendem Maße ins Bewußtsein der interdisziplinären Diskurse gedrungen, womit sich auch das wachsende Interesse an den Ansätzen der SOC, der TAS und der PKL erklärt.
Damit ist außer den transzendental-dialektischen und grammatologischen Kritiken der klassischen Logik (Hegel, Husserl, Heidegger, Günther, Derrida) auch die Grund-

[3] Diese These wird in Kapitel 2 ausführlich diskutiert.
[4] [Ass65]
[5] [Foe75]
[6] [Whi25], [Wit87]

lagenkrise der Mathematik und Logik sowie der durch die BCL-Arbeiten eingeleitete Paradigmawechsel der kybernetischen Systemtheorie als allgemeiner Hintergrund dieser Arbeit angesprochen.

Absicht und Ziel

In der Rezeption der Arbeiten Günthers und anderer an einer Polykontexturalen Logik im engeren Umfeld der BCL–Arbeiten haben bislang die konzeptionellen und philosophischen Ansätze überwogen. Dies ist zum einem auch darauf zurückzuführen, daß in den vorhandenen Arbeiten die Formalisierungen nur ansatzweise oder bruchstückhaft entwickelt werden. Zum anderen sind aber einige wichtige mathematische Arbeiten ([Na64], [Sch67a], [Sch67b]) nur schwer zugänglich und bislang kaum beachtet worden. Ziel dieser Arbeit ist es, das vorhandene fragmentarische und verstreute Material in einem einheitlichen formalen Aufbau zu systematisieren und darzustellen. Die historischen Formalisierungsansätze sollen rekonstruiert, formal ausgearbeitet erweitert und wo nötig ergänzt werden. Die Implementierung dieser Formalisierung soll die Morphogrammatik der experimentellen Exploration zugänglich machen.
Die Absicht der Arbeit ist es, diese Ausarbeitung in die aktuelle Diskussion und Rekonstruktion der BCL–Arbeiten einzubringen und eine formale Auseinandersetzung mit der PKL–Konzeption anzuregen, die aus den oben angeführten Gründen bisher nur in geringem Umfang stattgefunden hat.
Hierbei wird ein der Güntherschen Theorie immanenter und affirmativer Standpunkt eingenommen, um deren Konzeption und Intention so weit formal auszuarbeiten, daß sie im Rahmen der modernen mathematisch-logischen Grundlagenforschung einer kritischen Überprüfung unterzogen werden kann. Erst nach einer solchen — bis heute nur unvollständigen — Explikation kann entschieden werden, ob sich die transklassische Formkonzeption der PKL mathematisch und logisch realisieren läßt und welche Bedeutung sie für die philosophische und mathematisch-logische Grundlagenforschung tatsächlich hat.

Ein weiteres Anliegen dieses Buches ist es, anhand des zentralen Motivs der Formalisierung selbstreferentieller Systeme Verbindungen der PKL, Morphogrammatik und Kenogrammatik zur Grundlagenforschung der Mathematik, Logik und Informatik aufzuzeigen. Besonders sollen aktuelle Probleme der KI-Forschung, (Meta-level Architekturen, Reflektive Programmierung, Prozeßkommunikation, KI-Programmiersprachen) in Zusammenhang mit grundlegenden Konzepten der Morphogrammatik gesetzt werden, um weitere Forschungen auf diesen Gebieten anzuregen
Die Arbeit wendet sich dabei an einen Leserkreis, der mit den Diskursen der Second Order Cybernetics und der Theorie Autopoietischer Systeme vertraut ist und grundlegende mathematische Kenntnisse besitzt. Insbesondere dürfte die Arbeit für Logiker, Mathematiker und Informatiker von Interesse sein, die mit den Fragestellungen der Grundlagenkrise der Mathematik und alternativen Logikkonzeptionen vertraut sind.

Begrenzung

Innerhalb der PKL-Konzeption läßt sich die Morphogrammatik als Bindeglied zwischen der *Kenogrammatik*[7] und der Polykontexturalen Logik einordnen. Die Ausar-

[7] Gr. kenos: leer.

beitung der Morphogrammatik wird hier deshalb in Anknüpfung an diese Theorien vorgenommen.
Für die Zwecke dieser Ausarbeitung wird dabei zunächst ein hierarchischer Fundierungszusammenhang Kenogrammatik — Morphogrammatik — PKL angenommen. Eine solche Hierarchie wird als Konstruktionsstrategie gewählt, um zu einer klassisch induktiven Formalisierung zu gelangen. Unter der Annahme dieser hierarchischen Abfolge wird die Morphogrammatik durch die Kenogrammatik fundiert. Daher muß als erstes die Kenogrammatik, in dem Umfang, wie es für den definitorischen Aufbau der Morphogrammatik nötig ist, formal eingeführt werden. In dieser Hierarchie fußt dann die PKL wiederum auf der Morphogrammatik und benutzt deren Konzepte. Diese Arbeit leistet nun jedoch keine vollständige Ausarbeitung der PKL, sondern beschränkt sich darauf, die Architektur des Gesamtsystems Kenogrammatik – Morphogrammatik – PKL aus der Sicht der aktuellen Forschung zu skizzieren. Besonderer Wert wird darauf gelegt, zu zeigen wie die erarbeiteten Konzepte, Strukturen und Operationen der Kenogrammatik und Morphogrammatik im Gesamtaufbau der PKL lokalisiert sind. Diese Skizzierung wird deutlich machen, daß die Architektur der Polykontexturalen Logik fundamental selbstreferentiell strukturiert ist, daß also die Annahme eines hierarchischen, induktiven Aufbaus verworfen und in einer heterarchischen und *proemiellen* (d.h. sich simultan und wechselseitig fundierenden) Gesamtstruktur von Kenogrammatik, Morphogrammatik und PKL aufgehoben werden muß. Für die simultane wechselseitige Fundierung verschiedener logischer Ebenen wird in der Güntherschen PKL-Konzeption der Begriff der *Proemialrelation*[8] eingeführt. Aufgrund der fundamentalen Bedeutung der Proemialrelation für die Gesamtkonzeption der PKL, muß der Vollständigkeit halber auch eine formale Charakterisierung der Proemialrelation entwickelt werden.
Die Modellierung der Proemialrelation geht über den Rahmen der historischen Rekonstruktion hinaus und entwickelt mit dem Proemialkombinator PR ein formales Instrumentarium, das eine Anknüpfung der Morphogrammatik an aktuelle Grundlagenforschungen der Mathematik und Informatik herstellt. Die angrenzenden Fragestellungen können hier nur gestreift werden, wo es für das unmittelbare Verständnis dieser Arbeit notwendig ist. Für eine tiefergehende Auseinandersetzung mit der Arbeit wird jedoch eine gewisse Vertrautheit mit diesem Themenkreis vorausgesetzt.
Die Fortentwicklung der Morphogrammatik hat zu einer Reihe theoretischer und formaler Arbeiten geführt, die nicht in unmittelbarem Zusammenhang mit der Applikation in logischen Systemen stehen[9]. Die hier geleistete Darstellung beschränkt sich auf diejenigen Gebiete der Morphogrammatik, die sich auf die Formalisierung logischer (insbesondere polykontexturaler) Systeme beziehen. Diese Einschränkung geschieht im Hinblick auf das Primat der Formalisierung der polykontexturalen Logik.

Aufbau und Methode

Die Arbeit ist in vier Teile gegliedert:

I Motivation
II Darstellung
III Analyse
IV Anknüpfungen

[8] Gr. prooimion: Vorspiel.
[9] Vergleiche z. B. [Kro86], [Tho85], [Nie88].

Im einführenden ersten Teil wird vor dem Hintergrund des Grundlagenproblems der Formalisierung komplexer und selbstreferentieller Systeme zur Konzeption der PKL hingeführt. Als Motivation und erste Orientierung über die Arbeit wird die grundlegende Architektur der PKL und ihre Fundierung in der Morphogrammatik und Kenogrammatik umrissen.
Bewußt wird hier auf eine explizite Darstellung der philosophischen Argumentation zur Begründung der Güntherschen Theorien verzichtet, die sowohl bei Günther als auch in der Rezeption seiner Arbeiten umfassend dargestellt wurde ([Gue78], [Dit79], [Cas93]). Dieser Verzicht geschieht zugunsten einer rein formalen Auseinandersetzung mit seiner Arbeit, die bis heute nur in geringem Umfang geleistet wurde. Gerade vor dem Hintergrund des Güntherschen Motivs der „*Formalisierung der transzendental-dialektischen Logik*“[10] scheint ein solches rein formallogisches Vorgehen angemessen: „*Günther tritt in seinen Werken mit dem Anspruch an, getrennte Reiche der Philosophie und Wissenschaft füreinander fruchtbar zu machen. Er publiziert 1933 ein Buch unter dem Titel ‘Grundzüge einer neuen Theorie des Denkens in Hegels Logik’*[11]*, dessen Titel die Richtung seiner Forschungen weist: Es geht ihm darum, die Kritik der (spekulativen) Philosophie an den formalen Verfahren z. B. der Mathematik und Logik ernstzunehmen und dennoch nicht auf Exaktheit und Formalisierbarkeit zu verzichten. Er hält die Hegelsche Kritik an den Trennungen von Subjekt und Objekt, Form und Inhalt, Sein und Nichts usw. für ebenso berechtigt, wie er um den Erfolg der Wissenschaften, die auf diesen Trennungen basieren — wissenschaftstheoretisch fundiert durch den Logischen Positivismus und Kritischen Rationalismus — als bedeutende abendländische Errungenschaften weiß.* “[12]

Der zweite Teil ist mit der formalen Darstellung der Kenogrammatik und Morphogrammatik befaßt. Da die mathematische Systematik der Morphogrammatik erheblich von der ihrer historischen Genese abweicht, wird in dieser Darstellung eine rein definitorische Einführung der Morphogrammatik gewählt, die die mathematischen Konzepte, die den historischen Fragmenten zur Morphogrammatik entnommen wurden, zusammen mit notwendigen Erweiterungen und Korrekturen in eine durchgehende und einheitliche Form bringt. So wird in dieser Arbeit beispielsweise die Kenogrammatik als die im Sinne einer formalen Systematik der Morphogrammatik vorgeordnete Theorie eingeführt, obwohl sie von Günther erst nach seiner Entdeckung der Morphogrammatik thematisiert wurde.
Die hier geleistete Rekonstruktion und Systematisierung geht von den Arbeiten Günthers aus ([Gue78], [Gue80]), stützt sich jedoch ebenso auf eine Reihe begleitender und nachfolgender Ausarbeitungen und Kritiken seiner Theorien. Aus der Phase der Arbeit Günthers am Biological Computer Laboratory (BCL) an der University of Illinois, werden hier besonders die Arbeiten Von Foersters ([Foe70], [Foe75], [Foe76]), die abbildungstheoretischen Untersuchungen Schadachs ([Sch67a], [Sch67b], [Sch67c]) und die kombinatorischen Analysen Nas ([Na64]) herangezogen. Aus den Arbeiten Kaehrs ([Kae74], [Kae78], [Kae82], [Kae92], [Kae93]) wurden zahlreiche Ansätze und Methoden zur Formalisierung von Kenogrammatik, Morphogrammatik und Polykontexturaler Logik rekonstruiert, ausgeführt und erweitert. Die Arbeiten [Nie88] und [Hou88], die sich auf [Kro86] beziehen, werden in den Formalisierungen der Kenogrammatik und Kenoarithmetik einbezogen. Die vorliegende Arbeit setzt sich desweiteren an mehreren Stellen mit Heises ‘Analyse der Morphogrammatik’ [Hei91] auseinander.

[10] [Gue80b1].
[11] [Gue33].
[12] [Hei91].

Die in der Darstellung entwickelten Formalismen sollen in eine Computerimplementierung überführt werden, um die Morphogrammatik zu operationalisieren und einem experimentellem Zugang zu öffnen. Als Implementierungssprache wurde ML (Meta Language) gewählt, das sich aus mehreren Gründen sehr zur Implementierung der hier entwickelten Formalismen eignet. ML wurde ursprünglich als Spezifikations- und Metasprache für automatische Beweissysteme entwickelt[13]. Aufgrund seiner fortschrittlichen Konzeption, seiner exakten, sicheren, übersichtlichen und effizienten Struktur wird ML inzwischen in vielen Bereichen der Informatik angewandt und neben anderen funktionalen Sprachen (Miranda, Scheme, FP, Haskell) an einer wachsenden Zahl von Universitäten als erste Programmiersprache gelehrt. Zu den fortschrittlichen Konzepten von ML gehören das strikte Typechecking bei Typpolymorphie, Pattern Matching, ein hochentwickeltes Modulkonzept, separate und inkrementelle Compilierung, Interaktivität und Portabilität. ML stellt bestimmte Grunddatentypen (insbesondere Listen) zur Verfügung, die zur Implementierung morphogrammatischer Objekte (Kenogrammkomplexionen, Morphogrammketten, Morphogrammatrizen) benötigt werden und ermöglicht desweiteren die Definition infiniter Datenstrukturen (mittels des Konzeptes der 'Lazy Lists'), die zur finiten Repräsentation und Manipulation unendlicher Mengen[14] notwendig sind. Von der Möglichkeit, in ML neben allen Konzepten rein funktionaler Sprachen auch imperative Konstrukte (wie Zuweisungen und verzeigerte Strukturen) zu benutzen, wird bei der Implementierung einer Graphreduktionsmaschine zur Modellierung der Proemialrelation in Kapitel 10 Gebrauch gemacht.

In ML lassen sich zu mathematischen Definitionen äquivalente, d.h. lediglich syntaktisch verschiedene Implementierungen erstellen, die einerseits als Programm ausführbar sind, andererseits aber auch gut lesbare Definitionen und Spezifikationen darstellen. Die von ML erreichte unmittelbare Nähe der Implementierung zum mathematischen Formalismus ist sehr hilfreich bei der explorativen Entwicklung formaler Systeme und ermöglicht in dieser Arbeit eine hochgradige Durchdringung von Darstellung und Implementierung der morphogrammatischen Theorie. Spezifikationen, Algorithmen und Beispielberechnungen können unmittelbar in den Text miteinbezogen werden, ohne dessen Form und Ablauf zu stören. Diese Durchdringung dient der Stringenz des Textes, führt zu einer größeren Tranzparenz der abstrakten Formalismen und soll dem Leser einen intensiven und kompakten (nach Möglichkeit auch rechnerbegleiteten) Kompetenzerwerb hinsichtlich der morphogrammatischen Theorie ermöglichen. Die in dieser Arbeit entwickelten Implementierungen sind nicht als effiziente Programme gedacht, die bestimmte fixierte Probleme mit einem minimalen Resourcenaufwand lösen, sondern als formale Definitionen und Notationen in einer formalen Metasprache (eben ML), die in einem begrenzten Rahmen[15] auch zu direkten Berechnungen benutzt werden können. Für bestimmte Definitionen (etwa der Definition einer unendlichen Menge mittels der 'Lazy–List'–Konzeption, lassen sich prinzipiell keine endlichen Berechnungen angeben. Für andere Probleme, die aufgrund der zugrundeliegenden kombinatorischen Eigenschaften der betrachteten Objekte und Operationen exponentiell wachsenden Speicher– und Rechenaufwand implizieren, las-

[13] Das in [Bas93] dargestellte automatisierte PKL–Beweissystem ist ebenfalls in ML implementiert.

[14] Etwa der Menge aller Kenogrammsequenzen, vgl. Kapitel 4.

[15] Die Begrenzungen einer solchen Implementierung ergeben sich nicht etwa aus der mangelnden Effizienz des ML–Systems oder aus der Verwendung rekursiver Algorithmen, sondern aus den prinzipiellen kombinatorischen Eigenschaften morphogrammatischer Objekte und Operationen, die im Verhältnis zur Größe der Eingabedaten exponentiell wachsenden Speicher– und Rechenaufwand benötigen.

sen sich durch Optimierungen (etwa der Linearisierung rekursiver Funktionen oder der Reimplementierung in maschinennäheren Sprachen) nur in begrenztem Umfang Effizienzverbesserungen durchführen.
Zur Einarbeitung in die Programmiersprache ML sei das Buch '*ML for the Working Programmer*' [Pau91], das auch Bezugsadressen für kostenlose ML–Implementierungen enthält, besonders empfohlen.

Aufgrund der kombinatorischen Probleme einer faktischen Realisierung morphogrammatischer Objekte und Operationen bildet die von den konkreten Strukturen abstrahierende Klassifikation und Analyse der morphogrammatik den Schwerpunkt des dritten Teils der Arbeit.
Es werden verschiedene Klassifikationen des Operandensystems der Morphogrammatik eingeführt. Morphogrammatische Operationen werden bezüglich ihres Verhaltens hinsichtlich dieser Klassifikationen analysiert. Bei diesen Analysen werden kombinatorische und graphentheoretische Verfahren eingesetzt.

Im vierten Teil werden Anknüpfungspunkte der Morphogrammatik an die Gesamtarchitektur der PKL sowie aktuellen Grundlagenforschungen der Informatik dargestellt. Diese Ausführungen schließen die Einordnung der hier geleisteten formalen Darstellung und Implementierung ab und eröffnen Ausblicke für mögliche zukünftige Untersuchungen auf diesem Gebiet.

Ablauf der Arbeit

Der Teil **I Motivation** enthält neben dieser Einführung noch das Kapitel **2 Logische Form**. Dort wird als hinführendes Motiv zur PKL das Problem der Formalisierung selbstreferentieller Systeme entwickelt. Die PKL wird als ein möglicher Lösungsansatz vorgeschlagen. Der Zusammenhang von PKL, Morphogrammatik und Kenogrammatik wird umrissen, sowie orientierende Verweise zu den jeweils vertiefenden Kapiteln des Buches genannt.

Der Teil **II Darstellung** enthält die Kapitel **3 Kenogrammatik**, **4 Kenoarithmetik** und **5 Morphogrammatik**.
In Kapitel 3 wird die Kenogrammatik zunächst anhand der Gegenüberstellung von Semiotik und Kenogrammatik inhaltlich motiviert. Nach einer Reflektion auf die Verwendung klassischer formaler Methoden zur Formalisierung der transklassischen Konzepte der PKL wird dann ein formaler Aufbau der Kenogrammatik erarbeitet.
Die im vorhergehenden Kapitel entwickelte Unterteilung der Kenogrammatik in *Proto-, Deutero-* und *Tritostruktur* wird in Kapitel 4 arithmetisch interpretiert und zur Einführung von Proto-, Deutero- und Tritoarithmetik benutzt. Dieses Kapitel stellt in erster Linie eine arithmetische Interpretation der Thematik des vorhergehenden Kapitels vor und kann beim ersten Durchgang der Arbeit ausgelassen werden.
In Kapitel 5 wird dann die Morphogrammatik im engeren Sinne eingeführt. Nach einer kurzen inhaltlichen Motivation wird zunächst der Günthersche Ansatz der Wertabstraktion der Operatoren des Aussagenkalküls zu den *Basismorphogrammen* nachvollzogen, um dann die Einführung der Basismorphogramme auch formal in der Kenogrammatik zu fundieren. Im Hauptteil des Kapitels werden grundlegende morphogrammatische Strukturen, Morphogrammketten, Morphogrammatrizen sowie die Operationen Komposition, Dekomposition und Reflektor definiert.

Der so definitorisch eingeführte Raum morphogrammatischer Strukturen und Operationen wird in Teil **III Analyse** untersucht. In Kapitel **6 Klassifikation der Morphogrammatik** werden verschiedene Klassifikationen der Basismorphogramme und der Reflektoren eingeführt. In den beiden folgenden Kapiteln wird das Verhalten morphogrammatischer Operationen bezüglich dieser Klassifikationen untersucht. Die besonderen kombinatorischen Eigenschaften (kenogrammatische und morphogrammatische *Polysemie*) morphogrammatischer Operationen führen zu exponentiell anwachsenden Lösungsmengen, die konkrete Berechnungen schon für 4×4–Matrizen vor enorme Zeit- und Speicherprobleme stellt. Daher erweisen sich die hier vorgestellten abstraktiven Verfahren als nützliche Hilfmittel zur Untersuchng der Morphogrammatik.
In Kapitel **7 Graphentheoretische Analyse Reflektionaler Operationen** wird eine Analyse der Reflektoren durchgeführt.
In Kapitel **8 Analyse der Komposition** werden die kombinatorischen Eigenschaften der morphogrammatischen Komposition analysiert.
Teil III rekonstruiert und erweitert historische Ansätze zur Klassifikation und Analyse der Morphogrammatik. Da diese Analysen für das unmittelbare Verständnis zunächst nicht notwendig sind, kann dieser Teil der Arbeit beim ersten Lesen übersprungen werden.

Im vierten Teil des Buches, **Anknüpfungen**, werden Verbindungen der Morphogrammatik zum Gesamtsystem der PKL–Konzeption sowie zu aktuellen Grundlagenforschungen der Informatik aufgezeigt.
In Kapitel **9 Einbettung von Logiken in der Morphogrammatik** wird aus Gründen der historischen Vollständigkeit und des tiefergehenden Verständnis der PKL zunächst die Fundierung der Güntherschen *Stellenwertlogik* — einem Vorläufer der PKL — in der Morphogrammatik gezeigt. Dann wird in Anlehnung an die aktuellen Untersuchungen Kaehrs [Kae92] ein Konstruktionsschema der Polykontexturalen Logik skizziert, das den proemiellen Fundierungszusammenhang von Kenogrammatik, Morphogrammatik und PKL näher bestimmt.
In Kapitel **10 Dynamische Modellierung der Proemialrelation** wird ein operationales Modell der Proemialrelation erarbeitet. Da die Proemialrelation eines der fundamentalen Konzepte der gesamten Polykontexturalitätstheorie ist, wäre diese Arbeit ohne eine formale Darstellung der Proemialrelation unvollständig.
Die Entwicklung des hier entworfenen Modells basiert auf einer speziellen Implementierungstechnik funktionaler Programmiersprachen, der parallelisierten Graphreduktion. In diesem Kapitel werden mögliche Anknüpfungspunkte für zukünftige Forschungsarbeiten erörtert.
Der Aufbau der Arbeit ist in Abbildung (1.1) schematisch dargestellt. Für jedes Kapitel sind jeweils die wichtigsten erörterten Themen und Begriffe aufgeführt. Die dünnen Linien stellen die wichtigsten Querverbindungen zwischen den Begriffen und den vertiefenden Kapiteln dar. Die Einführung der wichtigsten in der Arbeit verwendeten Begriffe und die Motivation für den weiteren Ablauf erfolgt nun in Kapitel 2.

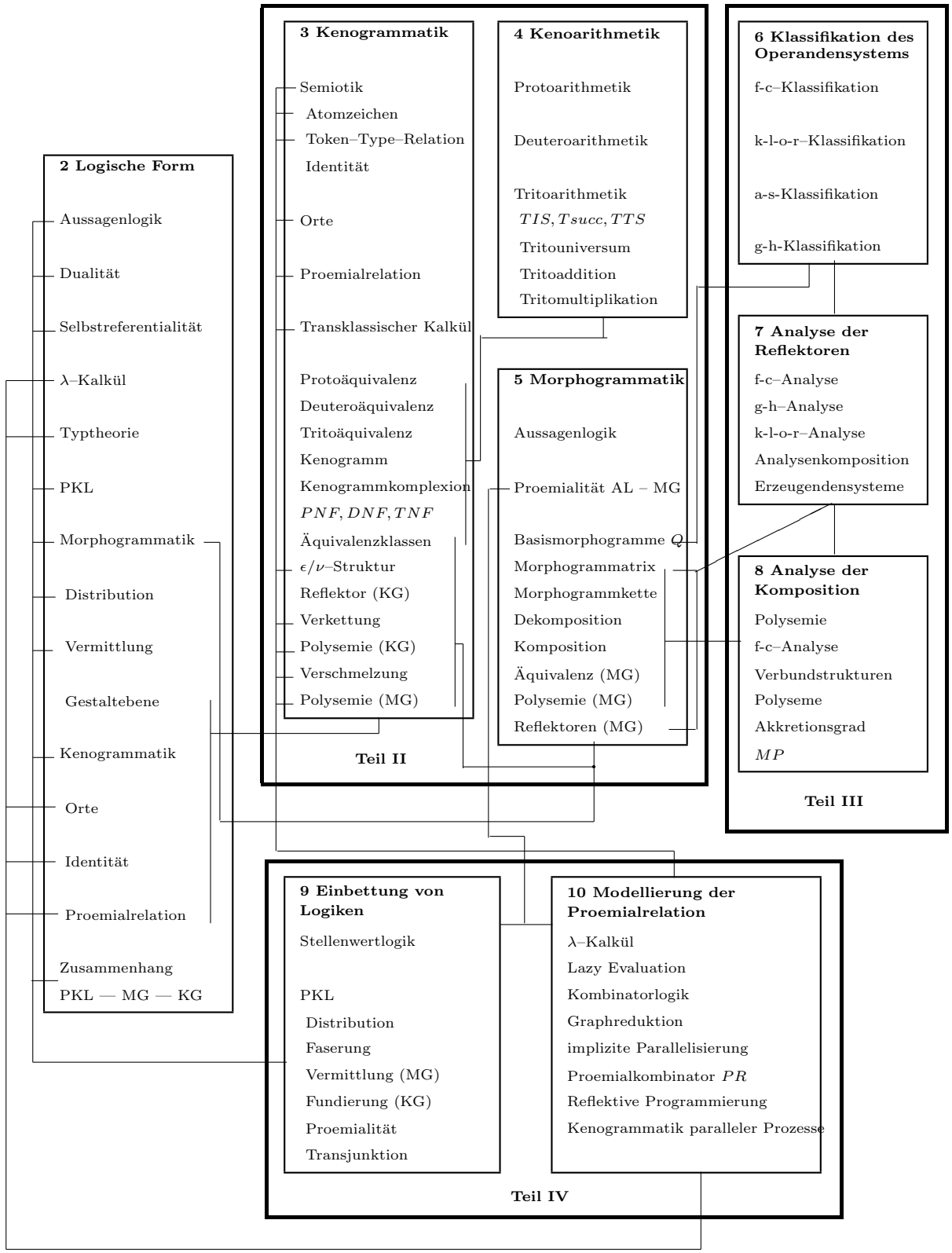


Abbildung 1.1: Aufbau der Arbeit

Kapitel 2

Logische Form

Man glaubt wieder und wieder der Natur nachzufahren,
und fährt nur der Form nach, durch die wir sie betrachten.
L. Wittgenstein

2.1 Die logische Form wissenschaftlicher Beschreibungen

Die auf G. Günther zurückgehenden Arbeiten zur Entwicklung einer *Polykontexturalen Logik* (PKL) sind zum überwiegenden Teil philosophisch motiviert und schließen insbesondere an Hegels *Wissenschaft der Logik* an[1].
Die vorliegende Arbeit verzichtet darauf, die philosophische Fragestellung und Argumentation Günthers wiederzugeben, da sie es sich zur zentralen Aufgabe gemacht hat, die Grundlagen der Polykontexturalen Logik formal darzustellen und zu implementieren. Relevant sind für eine formale Beschreibung lediglich die logisch-strukturellen Aspekte, die *logische Form*. Die einführende Motivation der Polykontexturalen Logik geschieht aus diesem Grund hier ausschließlich vor dem Hintergrund eines strukturellen Kernproblems der abendländischen Denktradition. Diese Problematik beruht auf der dualistischen Formkonzeption der klassischen Logik — Anfang aller Rationalität — und den sich zwangsläufig aus ihr ergebenden, widersprüchlichen und selbstbezüglichen sprachlichen und formalen Konstruktionen.
Der Prozeß wissenschaftlicher Erkenntnis beginnt mit der Beobachtung und dem Anfertigen von *Beschreibungen* der zu untersuchenden Phänomene. Aus diesen Beschreibungen werden Theorien und Modelle abgeleitet, deren Gültigkeit wiederum mit wissenschaftlichen Methoden untersucht wird[2]. Zur Evaluierung wissenschaftlicher Beschreibungen wird eine Methode des richtigen Folgerns — die Logik — benötigt.
Zur Bedeutung der aristotelischen Logik für die Wissenschaft bemerkt etwa Heisenberg: „*Um eine feste Grundlage für das wissenschaftliche Denken zu schaffen, hat es Aristoteles in seiner* Logik *unternommen, die Formen der Sprache zu analysieren, die formale Struktur von Schlüssen und Ableitungen unabhängig von ihrem Inhalt zu untersuchen. In dieser Weise hat er einen Grad von Abstraktion und Genauigkeit erreicht, der bis dahin in der griechischen Philosophie unbekannt war, und er hat dadurch in höchstem Maße zur Klärung beigetragen, zur Aufrichtung einer gewissen*

[1] [Heg86], [Gue33], [Gue78].
[2] [Mat85], S. 34f.

Ordnung in unserer Methode des Denkens. Er hat tatsächlich die Grundlage für die wissenschaftliche Sprache geschaffen.“ [Hei59]. Die Frage nach dem Fundament der Wissenschaft erscheint also als die Frage nach der logischen Struktur von Beschreibungen. In der Philosophie sind die Erkenntnistheorie und die Sprachphilosophie mit dieser Thematik befaßt.

2.1.1 Die Form der Dualität

Im Folgenden sollen anhand der exemplarisch ausgewählten[3] sprachkritischen Analyse der Metalinguistik grundlegende Probleme der dualistischen Formkonzeption der aristotelischen Logik aufgezeigt werden.
Die hauptsächlich an Sapir[4] und Whorf[5] anschließende *Metalinguistik* untersucht allgemein „*die Beziehungen einer Sprache und der gesamten übrigen Kultur der Gesellschaft, die diese Sprache spricht. Die Metalinguistik hat demgemäß auch Beziehungen zu allen übrigen Wissenschaften und der Philosophie. Diese alle müssen sich einerseits einer Sprache bedienen und so von dieser mitformen lassen [...]* .“[6]
Die Ergebnisse der metalinguistischen Forschung haben verdeutlicht, daß die Grammatik — das *linguistische System* — einer Sprache nicht nur eine reproduktives, neutrales Instrument zum Ausdruck von Gedanken ist, sondern vielmehr selbst die Gedanken formt. Die Formulierung von Gedanken ist kein unabhängiger (bewußt oder rational reflektierter) Vorgang, sondern er ist beeinflußt von der jeweiligen Grammatik. Er ist daher für verschiedene Grammatiken mehr oder weniger verschieden. „*Wir gliedern die Natur an Linien auf, die durch unsere Muttersprachen vorgegeben sind. Die Kategorien und Typen, die wir aus der phänomenalen Welt herausheben, finden wir nicht einfach in ihr [...]; ganz im Gegenteil präsentiert sich die Welt in einem kaleidoskopartigen Strom von Eindrücken, der durch unseren Geist organisiert werden muß — das aber heißt weitgehend: von dem linguistischen System in unserem Geist.*“[7]
Die linguistischen Systeme der indoeuropäischen Sprachen zeichnen sich im Gegensatz zu anderen Sprachen (etwa der chinesischen oder den indianischen Sprachen) durch eine Trennung von Substantiven und Verben aus. „*Die Griechen und ganz besonders Aristoteles haben den Gegensatz ‘Substantiv:Verb’ noch hervorgehoben und daraus ein Gesetz der Vernunft gemacht. Seitdem hat er [der Gegensatz] in der Logik viele verschiedene Fassungen gefunden: Subjekt und Prädikat, Täter und Tätigkeit, Dinge und Relationen zwischen ihnen, Objekte und ihre Attribute, Quantitäten und Operationen.*“[8]
Diese Substantiv:Verb Unterscheidung innerhalb der Grammatik kann nach Whorf nicht als Beleg für eine tatsächliche Aufspaltung der ‘Realität’ in äquivalente dichotome Strukturen gewertet werden, da Sprachen existieren, in denen solche Unterscheidungen unbekannt sind, die den Menschen des jeweiligen Kulturkreises dennoch eine adäquate Kommunikation über die Welt ermöglichen[9]. Die Struktur der indoeu-

[3] Im 20. Jahrhundert hat es eine große Anzahl vergleichbarer sprachkritischer und relativistischer Analysen gegeben. Vgl. [Wit87], [Qui75], [Kor26].
[4] [Sap61].
[5] [Who63].
[6] [Who63], (S. 140 f.).
[7] [Who63], S. 12. Dieser hier entwickelte Gedanke ist als ‘linguistisches Relativitätsprinzip’ oder auch als ‘Sapir–Whorf–Hypothese’ bekannt geworden und deckt sich weitgehend mit den erkenntnis- und sprachtheoretischen Ergebnissen des Konstruktivismus [Sch87], [Mat85] (vgl. [Hei59], [Geb88]).
[8] [Who63], S. 42.
[9] Die Entstehung von ‘Hochkulturen’ ist nicht auf den Bereich der indogermanischen Sprachen

ropäischen Sprachen erscheint somit als *kulturelle* Besonderheit, nicht als 'naturgegebene' 'Abbildung' der 'Realität'.

Die dichotome Struktur der Grammatiken indoeuropäischer Sprachen, deckt sich, wie Whorf bemerkt, mit der dualistischen Form der klassischen Logik, die die grammatikalische Dichotomie auf ihre knappste Gestalt, die Spaltung von Subjekt und Objekt, verkürzt und sie zur Grundlage der abendländischen Rationalität erhebt. Da die Form der Logik aus der Struktur indoeuropäischer Grammatiken abgeleitet ist, kann sie keine universale Gültigkeit beanspruchen, sondern bleibt wie die Grammatik in den Bereich der kulturellen Bedingtheit verwiesen[10].

Die dichotome Struktur der Logik hat in der Geschichte der Philosophie stets Ausdruck in dualistischen Ontologien gefunden. Der Cartesianische Dualismus von *res extensa* und *res cogitans* bildete mit seinem Postulat einer nicht hintergehbaren oder überbrückbaren Spaltung des Beobachters von der Welt die methodische Grundlage der neuzeitlichen empirischen Wissenschaft[11].

Wissenschaftliche Beschreibung ist somit in dreierlei Hinsicht der dualistischen Formkonzeption unterworfen: zum einen bedient sie sich indoeuropäischer Sprachen (aus deren Kulturraum sie stammt) und ist an die grammatische Substantiv:Prädikat Spaltung gebunden; zum zweiten geschieht ihre rationale Evaluierung mit Hilfe der klassischen zweiwertigen, dualistischen Logik; zum dritten muß sie der postulierten Beobachter:Welt Dichotomie genügen, um mit dem Fundament der wissenschaftlichen Rationalität übereinzustimmen.

Beschreibungen 'toter Gegenstände' (d.h. Gegenstände die in keiner erkennbaren Wechselwirkung mit ihrer Umgebung stehen) können ohne Schwierigkeiten im naturwissenschaftlichen Sprachrahmen abgefaßt werden, da sie stets die oben genannten dualistischen Strukturen reflektieren können. „*Die von den älteren Logikern bei der Behandlung dieser Frage [nach dem Verhältnis von Sprache und Realität] benützten Beispiele sind meist schlecht ausgewählt. Tische, Stühle und Äpfel auf Tischen werden als Belegstücke bevorzugt, um die objektartige Natur der Wirklichkeit und ihre genaue Entsprechung mit der Logik aufzuweisen. Die Artefakte des Menschen und die landwirtschaftlichen Produkte, die er von den lebenden Pflanzen trennt, haben einen einzigartigen Grad von Isolierung. Es ist daher ohne weiteres zu erwarten, daß die Sprachen gesonderte Terme für sie enthalten. Die wirklich interessante Frage ist nicht, was verschiedene Sprachen mit solchen künstlich isolierten Objekten tun, sondern: Was tun sie mit der fließenden Natur in ihrer Bewegung, Farbigkeit und wechselnden Form — mit Wolken, Ufern und dem Flug der Vögel? Denn so wie wir das Antlitz der Natur aufgliedern, so wird unsere Physik des Kosmos sein. [...] Die Aufgliederung der Natur [ist] ein Aspekt der Grammatik.* "[12]

---

beschränkt und kann daher ebenfalls nicht als Beleg für eine kulturgeschichtliche Prävalenz dieser Sprachen gewertet werden.

[10] Im direkten Anschluß an die oben zitierte Würdigung der Leistungen Aristoteles um die formale Analyse der Sprache (d.h. der Grammatik) bemerkt Heisenberg kritisch: „*Andererseits bringt die logische Analyse der Sprache auch die Gefahr einer zu großen Vereinfachung mit sich. In der Logik wird die Aufmerksamkeit auf spezielle sprachliche Strukturen gerichtet, auf unzweideutige Verknüpfungen und Folgerungen, auf einfache Muster des Schließens; alle anderen sprachlichen Strukturen werden vernachlässigt.*" a.a.O.

[11] [Hei59], S. 58ff.

[12] [Who63], S. 40f.

2.1.2 Die Form der Reflexion

Tatsächlich werfen die grammatikalischen Strukturen der Sprache erst dann Probleme auf, wenn 'Gegenstände' untersucht werden, deren Beschreibungen den grammatikalisch-logischen Dichotomien wissenschaftlicher Sprache widersprechen. Ein einfaches Beispiel einer solchen Konstruktion stellt etwa die folgende Beschreibung des Lebensvorgangs einer Pflanze dar: *Die Pflanze entsteht aus dem Samenkorn und das Samenkorn geht aus der Pflanze hervor.* Jede der Teilaussagen weist in sich eine grammatikalisch und logisch wohlgeformte Struktur auf. Da die dichotomen Teilaussagen jedoch aufeinander Bezug nehmen, ist die Gesamtaussage zirkulär oder *selbstreferentiell* strukturiert, wodurch sie im logischen Sinne bedeutungslos ist.

Im Falle dieses trivialen Beispiels kann argumentiert werden, daß die zirkuläre Beschreibung durch eine angemessene Beschreibung ersetzt werden müsse, die Pflanze und Samenkorn nur als Phänomene *eines* tieferliegenden Vorgangs enthält und die Zirkularität *linearisiert*, indem sie sie auf einen eindimensionalen Vorgang reduziert, der leicht formal abzubilden ist. Diese reduktionistische Methode wird in der Naturwissenschaft in einer Vielzahl von Bereichen erfolgreich angewendet und bedient sich zur formalen Darstellung beispielsweise mathematischer Differentialgleichungssysteme. Die Nützlichkeit und der Erfolg eines solchen Vorgehens soll hier nicht bestritten werden, jedoch soll die Aufmerksamkeit auf die ungelösten grundlegenden Probleme der Wissenschaft, Logik und Mathematik im Umgang mit komplexen, nicht linearisierbaren Strukturen gelenkt werden. Solche Probleme ergeben sich in Situationen, die nur durch *fundamental selbstreferentielle* Beschreibungen erfasst werden können und sich ohne Komplexitätsverlust nicht linearisieren (d.h auch: widerspruchsfrei abbilden) lassen.

Als Belege für die Existenz und Relevanz solcher Probleme seien hier zunächst zwei grundlegende Fragestellungen der Physik und der Biologie exemplarisch für eine Vielzahl analoger Probleme angeführt.

1. Heisenberg stellt in einem Vergleich zwischen der Philosophie Descartes und den Implikationen der Quantenmechanik fest: „*[Die cartesianische] Spaltung [in res extensa und res cogitans] in der Naturwissenschaft [war] für einige Jahrhunderte außerordentlich erfolgreich. Die Newtonsche Mechanik und alle anderen Teile der klassischen Physik, die nach ihrem Vorbild aufgebaut waren, beruhten auf der Annahme, daß man die Welt beschreiben kann, ohne über Gott oder uns selbst zu sprechen. Diese Möglichkeit galt beinahe als eine notwendige Voraussetzung für alle Naturwissenschaft.*

   Aber eben an dieser Stelle hat sich die Lage durch die moderne Quantentheorie von Grund auf geändert; daher können wir jetzt zu einem Vergleich der Philosophie des Descartes und unserer gegenwärtigen Situation in der modernen Physik übergehen. Es ist schon in früheren Abschnitten ausgeführt worden, daß wir in der Kopenhagener Deutung der Quantentheorie die Natur beschreiben können, ohne uns selbst als Einzelwesen in diese Beschreibung einzubeziehen, daß wir aber nicht von der Tatsache absehen können, daß die Naturwissenschaft vom Menschen gebildet ist. Die Naturwissenschaft beschreibt und erklärt die Natur nicht einfach so, wie sie an sich *ist. Sie ist vielmehr ein Teil des Wechselspiels zwischen der Natur und uns selbst. Sie beschreibt die Natur, die unserer Fragestellung und unseren Methoden ausgesetzt ist. An diese Möglichkeit konnte Descartes noch nicht denken, aber dadurch wird eine scharfe Trennung zwischen*

der Welt und dem Ich unmöglich.“[13]

Noch deutlicher wird die zirkuläre Struktur der von Heisenberg formulierten ‘unscharfen Trennung zwischen der Welt und dem Ich’ durch Spencer-Browns Analyse der Erkenntnissituation des Physikers hervorgehoben: „*Let us then consider, for a moment, the world as described by the physicist. It consists of a number of fundamental particles which, if shot through their own space, appear as waves, and are thus of the same laminated structure as pearls or onions, and other wave forms called electromagnetic which it is convenient [...] to consider as travelling through space with a standard velocity. All these appear to be bound by certain natural laws which indicate the form of their relationship. Now the physicist himself, who describes all this, is, in his own account, himself constructed of it. He is in short, made of a conglomeration of the very particulars he describes, no more, no less, bound together by and obeying such general laws as he himself has managed to find and to record.*“[14]

2. Auf eine entsprechende fundamentale Zirkularität biologischer Erkenntnis weist von Foersters Ausspruch hin: „*The laws of nature are written by man, the laws of biology must write themselves.*[15]“ Hiermit ist die im logischen Sinne paradoxe Situation angesprochen, daß ein wissenschaftlicher Beobachter, der das ‘Leben’ erforscht, ebenfalls ein Lebewesen ist, somit streng genommen das Leben *sich selbst* wissenschaftlich beschreibt.

   Diese Zirkularität findet sich folgerichtig auch in von Foersters allgemeiner Charakterisierung eines Lebewesens: „*Ein lebender Organismus ist eine selbstständige, autonome, organisatorisch geschlossene Wesenheit; und ein lebender Organismus ist selbst Teil, Teilhaber und Teilnehmer seiner Beobachtungswelt.*“

Die seit Descartes als Grundlage der Naturwissenschaft geltende Trennung des Beobachters von der Welt, sowie die von der aristotelischen Logik implizierte grammatikalisch-logische Struktur von Beschreibungen werden in den angeführten Beschreibungen der naturwissenschaftlichen Erkenntnissituation fundamental unterlaufen. Schrödinger[16] faßt dieses Dilemma in einer prägnanten Formel zusammen, die das diesem Kapitel vorangestellte Zitat Wittgensteins paraphrasiert: „*Der Grund, warum unser fühlendes, wahrnehmendes und denkendes Ich nirgendwo in unserem wissenschaftlichen Weltbild anzutreffen ist, kann einfach in sieben Worten angegeben werden: weil es nämlich selbst dieses Weltbild ist. Es ist identisch mit dem Ganzen und kann deshalb in demselben nicht als Teil enthalten sein.*“ Von Weizsäcker setzt diese Problematik in Beziehung zu Heidegger: „*Ein Kernstück von* Sein und Zeit *ist die Kritik der Cartesischen Ontologie. Dort wird gezeigt, daß Descartes nach dem Sein selbst nicht fragt. Deshalb ist es möglich, daß für ihn spezielle Bestimmungen [des Seins] zu definierenden Merkmalen seiner zwei ‘Substanzen’ werden. Die substantielle Trennung von res cogitans und res extensa nun ist die methodische Voraussetzung der gesamten klassischen Naturwissenschaft. Der sogenannte ‘naturwissenschaftliche’*

[13] [Hei59], S.60 f.

[14] [Bro72], S. 104 f.

[15] „*The laws of nature are written by man. The laws of biology must write themselves. In order to refute this theorem it is temting to invoke Gödel's proof of the Entscheidungsproblem in systems that attempt to speak of themselves. But Lars Löfgren and Gotthard Günther have shown that self–explanation and self–reference are concepts that are untouched by Gödel's arguments.*“ (v. Foerster, H. In: Journal of Cybernetics). Die Analyse in 1. zeigt gerade, daß auch die ‘Gesetze der Natur sich selbst schreiben müssen’.

[16] Schrödinger, E.: *Mind and Matter.* (S. 52), Cambridge, 1959.

Begriff methodischer Sauberkeit verlangt das absolute Vermeiden von 'Grenzüberschreitungen' zwischen diesen beiden Bereichen. Es scheint mir charakteristisch für die positive Wissenschaft unserer Zeit, daß die innere Logik ihrer eigenen Probleme sie zur Sprengung dieses Dammes zwingt. Dies wird evident in allen psychophysischen Problemen, wie etwa der Erforschung der Wahrnehmung, der Bewegung organischer Körper, des Ausdrucks. Es zeigt sich aber ebenso in der Problematik des 'Beobachters' in der Atomphysik. "[17]

Die allgemeine Bewußtwerdung dieser grundlegenden erkenntnistheoretischen Problematik hat zu zahlreichen philosophischen, erkenntnis- und wissenschaftstheoretischen Diskursen insbesondere im Rahmen des Konstruktivismus und der 'Second Order Cybernetics' geführt. Auf diese Diskussionen wird nur verwiesen[18], da hier allein die logische Form von Beschreibungen, nicht jedoch ihre inhaltlichen Implikationen erörtert werden sollen. Für die Zwecke dieser Untersuchung genügt es festzuhalten, daß es wissenschaftlich relevante sprachliche Konstruktionen gibt, die fundamental (d.h nicht ohne Komplexitätsverlust linearisierbar) *selbstreferentiell* strukturiert sind.

Gemeinsam ist solchen selbstreferentiellen Beschreibungen, daß sie ihre Gegenstände aus verschiedenen Perspektiven, in unterschiedlichen Funktionalitäten charakterisieren, wobei die einzelnen Teilbeschreibungen zueinander komplementär (widersprüchlich) sind und simultan gelten. Deutlich wird dies etwa bei der Beschreibung des Verhältnisses Beobachter:Welt. Der Beobachter ist *Subjekt* seiner Beobachtung der Welt, da er aber selbst auch 'Teil, Teilhaber und Teilnehmer seiner Beobachtungswelt' ist, ist er *simultan* auch *Objekt* seiner Beobachtung. Die von der klassischen Logik postulierte lineare Ordnung (im Sinne einer logischen Hierarchie) der Beobachtungsrelation wird durch diese Konstellation unterlaufen[19].

Eine formale Notation solcher Beschreibungen in mathematischen Kalkülen würde aufgrund der Zirkularität zwangsläufig zu logischen Widersprüchen in Form unendlicher Regresse führen. Zur Vermeidung logischer Widersprüche ist es daher eine Kernforderung der empirischen Naturwissenschaft, daß die Eigenschaften des Beobachters nicht in den Bereich seiner Beschreibungen gelangen dürfen[20]. Die zirkuläre Gestalt solcher Beschreibungen ist jedoch, wie erörtert, kein willkürlicher Verstoß gegen die Gesetze der Logik, sondern ergibt sich als zwangsläufige begriffliche Struktur zur Erfassung von Phänomenen, die sich nicht auf das reduzierte Schema der logischen Subjekt:Objekt Dichotomie abbilden lassen.

Selbstreferentielle Beschreibungen sind aufgrund ihrer logischen Struktur nicht im mathematisch naturwissenschaftlichen Rahmen evaluierbar (und daher im strengen Sinne

[17] [Wei63], S.244 f.

[18] [Sch87], [Mat85], [Mat87], [Cas92], [Kae92].

[19] Vgl. hierzu etwa die folgende Charakterisierung von Beschreibungen in der Quantentheorie: „*Die Dialektik von Ganzem und Teil ist in der Quantenwelt grundsätzlich verschieden von den in den klassischen Naturwissenschaften üblichen Beschreibungen. [...] Nach den Vorstellungen der modernen Quantenphysik ist die materielle Realität eine Ganzheit, und zwar eine Ganzheit, die nicht aus Teilen besteht.* Dabei heißt ganz, was nur durch eine Vielheit von komplementären Beschreibungen erfaßt werden kann." Primas, H.:*Umdenken in der Naturwissenschaft.* In: GAIA, Vol.1, 1992, S. 10f.

[20] „*The properties of the observer shall not enter into the descriptions of his observations.*" [Foe75] Um einen physikalischen Vorgang zu beobachten, muß der Beobachter ihn in irgendeiner Weise beeinflußen, da Sinnesorgane und Meßinstrumente nur auf energetische Veränderungen reagieren. Meßergebnisse sagen also nichts über den Vorgang *an sich* aus, sondern nur über sein Verhalten bezüglich des experimentellen Einwirkens auf ihn. Der Beobachter ist daher unauflöslich mit den Objekten seiner Beobachtung verkoppelt. Im Gegensatz zur klassischen Mechanik, wo die energetische Wechselwirkung im Verhältnis zu den beobachteten Energien vernachlässigt werden kann, ist dies in der Quantenmechanik nicht mehr möglich. Daher gilt das klassische Objektivitätsparadigma dort nicht mehr.

der 'hard science' nicht gültig). Da sie aber sprachlich korrekte, sich aus der grammatikalischen Struktur indoeuropäischer Sprachen ergebende Beschreibungen 'realer Phänomene' sind, werden sie in wissenschaftlichen Bereichen, die nicht der mathematischen Logik verpflichtet sind, als sinnvolle Konstruktionen zugelassen. Die begrifflichen Methoden der Dialektik, des hermeneutischen Zirkels und des Chiasmus etwa, die auf zirkulären sprachlichen Konstruktionen beruhen, sind in der Geschichte der Philosophie stets von großer Bedeutung gewesen und finden heute in den Disziplinen der 'soft-science' ihre Anwendung.
Die aus der 'Second Order Cybernetics' hervorgehende *Theorie autopoietischer Systeme* benutzt beispielsweise eine solche selbstreferentielle begriffliche Struktur zur Beschreibung von Lebewesen. So wird „*die autopoietische Organisation [...] als eine Einheit definiert durch ein Netzwerk der Produktion von Bestandteilen, die 1. rekursiv an demselben Netzwerk der Produktion von Bestandteilen mitwirken, das auch diese Bestandteile produziert, und die 2. das Netzwerk der Produktion als eine Einheit in dem Raum verwirklichen, in dem die Bestandteile sich befinden.*“[21] In dieser Definition eines autopoietischen Systems erscheint zunächst das 'Netzwerk' als Operator der Produktion von Bestandteilen. Aber das Operieren dieser Bestandteile erst realisiert das Netzwerk. Diese 'Definition' ist ganz offensichtlich selbstreferentiell konstruiert und unter formalen Gesichtspunkten sinnlos. Als rein beschreibende Theorie hat sich die Theorie autopoietischer Systeme jedoch in so verschiedenen Disziplinen wie Soziologie, Psychologie, Biologie und Ökologie als fruchtbares Paradigma erwiesen[22].

2.2 Selbstreferentialität in formalen Systemen

2.2.1 Dualistische Form und Selbstreferenz

Aus dem Vorhergehenden ist deutlich geworden, daß selbstreferentiell und zirkulär strukturierte sprachliche Konstruktionen erzeugbar sind, die sich in ihrem sprachlichen Kontext als sinnvoll erweisen, jedoch formal-logisch einen *circulus vitiosus* darstellen.
Auch in der Logik und Mathematik sind selbstreferentielle Strukturen notierbar, jedoch nicht evaluierbar, so daß sie als Paradoxien, Antinomien und Zirkularitäten von der formalen Betrachtung ausgeschlossen werden. Die historisch bekannteste Paradoxie ist sicherlich die Aussage des Kreters Epimenides: „Alle Kreter sind Lügner“. Unter der Annahme, diese Aussage sei *wahr*, muß auch Epimenides ein Lügner, seine Aussage daher *falsch* sein und vice versa. Das Paradoxon läßt sich daher zu der Aussage A verkürzen: „Die Aussage A ist falsch“. Die kürzeste logische Formalisierung dieser Paradoxie hat dann die folgende Gestalt:

$$A = \neg A.^{23}$$

Dieser Aussage kann keiner der Wahrheitswerte wahr oder falsch zugeordnet werden, da jede solche Setzung gleichzeitig ihre Negation impliziert.
Moderne Fassungen dieser grundlegenden Paradoxie sind aus allen Gebieten der Mathematik bekannt und lassen sich rein syntaktisch leicht generieren. Eines der bekanntesten ist etwa das *Russellsche Paradoxon* der Menge w aller Mengen, die sich nicht

[21] [Mat85], S. 158.
[22] [Hel90], [Sch87], [Mat87]
[23] Dies entspricht der *re-entry* Form $f = \overline{f|}$ [Bro72].

selbst enthalten. Unter der Annahme, daß $w \notin w$ folgt zwangsläufig $w \in w$. Wird jedoch angenommen, daß $w \in w$, folgt $w \notin w$, also:

$$w \in w \Longleftrightarrow w \notin w,$$

was ein offensichtlicher Widerspruch ist. In ihrer *Principia Mathematica* [Whi25] stellen Whitehead und Russell zu den oben erwähnten Paradoxien fest: „*In all the above contradictions (which are merely selections from an infinite number) there is a common characteristic which we may describe as* self-reference *or reflexiveness. The remark of Epimenides must include itself in its own scope. If* all *classes, provided they are not members of themselves, are members of w this must also apply to w; and similarly for the analogous relational contradictions*“[24].
Die Einführung ihrer Theorie der logischen Typen dient dem Zweck, solche selbstreferentiellen syntaktischen Konstruktionen, die zu mathematischen Widersprüchen führen, als nicht wohlgeformt auszuschliessen und so die Widerspruchsfreiheit der mathematischen Theorie zu bewahren. So kann etwa im Falle des Ausdrucks $A = \neg A$ argumentiert werden, daß A keine Aussage ist: Eine logische Aussage ist entweder wahr oder falsch, sie ist genau eines von beiden (Satz der Identität), sie kann nicht zugleich beide Werte annehmen (Satz vom verbotenen Widerspruch) und sie kann auch keinen anderen Wert annehmen (Satz vom ausgeschlossenen Dritten). Da A weder wahr noch falsch sein kann, erfüllt sie diese grundlegende Anforderung an eine logische Aussage nicht, und ist daher *keine* Aussage[25].
Das von der Typtheorie formulierte Verbot der Selbstreferenz mathematischer Ausdrücke hat enorme Konsequenzen für die Ausdrucksstärke formaler Systeme. So ist etwa der *untypisierte* λ-Kalkül *berechnungsuniversell* im Sinne der Turingberechenbarkeit, da in ihm Fixpunktoperatoren wie der Y-Kombinator notierbar sind. Die Möglichkeit der Selbstapplikation, die Anwendung einer Funktion auf sich selbst, ist notwendige Voraussetzung eines Fixpunktoperators. Der Y-Kombinator wird beispielsweise definiert als:

$$Y\ f = f\ (Y\ f).$$

Diese selbstreferentielle[26] Definition des Y-Kombinators ermöglicht einerseits die Notierung regulär rekursiver Funktionen, gewährleistet somit die Berechnunsuniversalität, erlaubt jedoch gleichzeitig auch das Auftreten nicht-terminierender Berechnungen[27]. Wird durch eine Typisierung des λ-Kalküls jedoch die Selbstapplikation im Sinne des Y-Kombinators unterbunden, ist der resultierende Kalkül im allgemeinen nicht mehr berechnungsuniversell [Bar80].
Die *begrenzte* Selbstreferentialität der rekursiven Funktionen, die nach einer endlichen Anzahl von Selbstapplikationen ihren Rekursionsanfang erreichen und terminie-

[24] [Whi25], S. 37.

[25] Wittgenstein schließt im *Tractatus* selbstbezügliche Aussagen durch die Definition des ‘Satzzeichens’ aus: „*Kein Satz kann etwas über sich selbst aussagen, weil das Satzzeichen nicht in sich selbst enthalten sein kann, (das ist die ganze ‘Theory of Types’).*“ [Wit87], S.28, 3.332.

[26] Im Gegensatz zur *rekursiven* Fakultätsfunktion:

$$\texttt{fak}\ (n) = \begin{cases} 1 & \text{falls } n = 0 \\ n * \texttt{fak}\ (n-1) & \text{sonst.} \end{cases}$$

weist diese Notation des Y-Kombinators keinen Rekursionsanfang auf, weshalb sie hier (unbegrenzt) *selbstreferentiell* im Gegensatz zu rekursiv (begrenzt selbstreferentiell) genannt wird.

[27] „*Y called* ‘the *paradoxical combinator’ may be used to construct logico-mathematical objects of a more or less paradoxical nature. For instance Y may be used to construct Russell’s paradox [. . .]*“ [Foe75].

ren, zeigt, daß Selbstreferenz nicht unbedingt 'bösartig' sein muß, sondern gerade die Mächtigkeit der klassischen formalen Sprachen ausmacht.
Lediglich die *unbegrenzte* Selbstreferentialität ist 'bösartig' und sprengt den Rahmen der klassischen Kalküle, da sie, wie etwa der Ausdruck $A = \neg A$, eine unendliche Anzahl von Selbstapplikationen hervorruft. Die Bemühungen, Selbstreferentialität in formalen System rein syntaktisch auszuschließen (wie in der Typtheorie), sind natürlich darauf gerichtet, das Auftreten unbegrenzter Selbstbezüglichkeit zu vermeiden. Sie schließen jedoch auch die syntaktisch nicht immer unterscheidbare, begrenzte Selbstreferenz aus und beschneiden so zwangsläufig, wie die angeführte Typisierung des λ-Kalküls, die Ausdrucksstärke der formalen Systeme. Alle anderen Versuche, begrenzte und unbegrenzte Selbstreferentialität zu unterscheiden und nur die begrenzten Fälle zuzulassen, entsprechen der Bestimmung der Terminierung von Berechnungen und scheitern daher am Halteproblem.

Die besondere Bedeutung selbstreferentieller Strukturen in der wissenschaftlichen Beschreibung biologischer, sozialer, ökologischer u.a. Phänomene und in der mathematischen Grundlagenforschung wirft das Problem ihrer *widerspruchsfreien Formalisierung* auf. „*Die Grenzen meiner [formalen] Sprache bedeuten die Grenzen meiner Welt*“[28], stellt Wittgenstein im Tractatus fest: solange Selbstreferentialität von antinomienfreier Formalisierung ausgeschlossen bleibt, ist eine im strengen Sinne wissenschaftliche Erfassung und operationale Modellierung all der Phänomene nicht möglich, deren Beschreibungen eine 'fundamentale Zirkularität' oder Selbstreferentialität aufweisen.

2.2.2 Die Reflexionsform der Polykontexturalen Logik

Kritische Auseinandersetzungen mit den Grundlagen der klasssischen Mathematik, mit dem Ziel den Raum des formal Beschreibbaren zu erweitern, haben zur Entwicklung einer Vielzahl alternativer Logikkonzeptionen geführt. Einige der bekanntesten Ansätze sind etwa:

- Mehrwertige Logiken[29].
- Fuzzy Logik.
- Quanten Logiken[30].
- Modale Logiken.
- Temporale Logik.
- Kombinatorische Logik[31].
- Dialog Logik[32].
- Calculus of indication[33].
- Calculus for self-reference[34].

[28] [Wit87], S. 89, 5.6.
[29] [Luk70]
[30] [Wei73]
[31] [Cur69]
[32] [Lor78]
[33] [Bro72]
[34] [Var75]

Eine Analyse all dieser Erweiterungsansätze kann in dieser Einführung selbstverständlich nicht gegeben werden, hierzu sei auf die Untersuchungen Kaehrs[35] verwiesen , in denen nachgewiesen wird, daß — aus der Perspektive der Güntherschen Transklassik betrachtet — alle bislang bekannten Ansätze konservative Erweiterungen der klassischen Logik darstellen, insofern sie die wesentliche *dualistische Formkonzeption* nicht aufheben, sondern lediglich den Raum der Ausdrucksmöglichkeit um bestimmte ontologische Zustände, Wahrscheinlichkeiten, Modalitäten, zeitliche Orientierungen oder andere Eigenschaften erweitern: „*Bei den genannten Erweiterungsversuchen — wie auch bei allen anderen mir bekannten 'Alternativlogiken' — handelt es sich im Prinzip darum, innerhalb des formalen Systems Parametrisierungen von Systemvariablen vorzunehmen und Systemkonstanten zu variablisieren. Im Nachhinein läßt sich sagen, daß [mit der Einführung o.g. Kalküle] keine wesentlich neuen Erkenntnisse erzielt wurden — außer einer Fülle von praktischen Methoden und Applikationen. Es ist daher nicht verwunderlich, daß von rein logischer Seite sowohl die Fuzzy-Logik wie auch die mehrwertige Logik in ihren Ansprüchen, neue Logiken zu sein, stark kritisiert wurden. [. . .] Die [o.g.] klassischen Kalküle haben eine transzendente Subjektivitätskonzeption zu ihrer Voraussetzung. Solange die Logik nur die Aufgabe hat, die allgemeinen Gesetze der objektiven, d.h. von jeder Subjektivität befreiten Welt, zu beschreiben, ist diese Konzeption optimal. Sie entspricht dem klassischen Paradigma:* The properties of the observer shall not enter into the description.“[36]

Die Polykontexturale Logik Günthers stellt sich explizit der Aufgabe, die Annahme einer in die Welt eingebundenen, *immanenten Subjektivität* in einer formalen Logik abzubilden. Ihr Anspruch ist es, die im klassischen Sinne widersprüchlichen Situationen, die eine Einbeziehung des Beobachters in seinen Beobachtungsbereich oder anders geartete Selbstreferentialitäten beinhalten, widerspruchsfrei zu modellieren.

Bei dieser Abbildung selbstreferentieller Strukturen geht die Polykontexturale Logik von der oben charakterisierten Struktur selbstreferentieller Beschreibungen aus, daß jeder Teilaspekt eine in sich logisch widerspruchsfreie Situation repräsentiert, die der jeweiligen Beobachtungsperspektive entspricht. Im Gegensatz zur klassischen Logik, die die widersprüchlichen Teile zu einem *monokontexturalen* Gesamtsystem zusammenfassen muß, was zu den beschriebenen logischen Widersprüchen führt, bildet die Polykontexturale Logik selbstreferentielle Strukturen ab, indem sie die Teilaspekte über mehrere logische Orte[37] — Kontexturen — verteilt, an denen jeweils eine klassische Logik gilt. Diese *verteilten* Logiken sind nun durch eine *außerlogische Vermittlung* miteinander verkoppelt. Da sowohl die Kontexturen in sich, als auch ihre Vermittlung widerspruchsfrei beschreibbar sind, lassen sich innerhalb der Polykontexturale Logik zirkuläre, selbstreferentielle und antinomische Strukturen widerspruchsfrei modellieren[38].

Im Gegensatz zu den oben angeführten klassischen Erweiterungen der Logik, die den monokontexturalen Formalismus durch *interne* (d.h. die Monokontexturalität bewahrende) 'Parametrisierungen und Variablisierungen' modifizieren, besteht die Erweiterungsstrategie der Polykontexturalen Logik Günthers darin, den monokontexturell

[35] [Kae80], [Kae92].

[36] [Kae80], S. 41.

[37] Die Idee des logischen Ortes bei Günther unterscheidet sich grundlegend von Wittgensteins Konzeption. Bei Wittgenstein bestimmt der logische Satz einen Ort im logischen Raum ([Wit87], S.31, 3.4). Der logische Raum bezeichnet dabei den von der monokontexturalen klassischen Logik aufgespannten Sprachrahmen. Der logische Ort Günthers meint hingegen einen *Ort im Sein*, an dem sich eine Logik realisieren kann (vgl. [Kae92], Abschnitt 7).

[38] In [Pfa88] demonstriert Pfalzgraf beispielsweise die Distribution einer logischen Antinomie über drei Kontexturen (S. 51f.).

begrenzten Rahmen in einer Vielzahl von distribuierten und vermittelten Kontexturen aufzuheben. Die Polykontexturale Logik beansprucht explizit, den strukturellen Rahmen der klasssischen Logik zu überschreiten und eine neuartige — *transklassische* — Formkonzeption, die 'Reflexionsform' zu realisieren, die die widerspruchsfreie Abbildung selbstreferentieller Strukturen ermöglicht. Aufgrund der grundlegenden Bedeutung, die die Existenz einer solchen formalen Theorie für die Grenzen formaler Beschreibbarkeit (und allgemein der Sprache) hätte[39], erscheint eine gründliche formalistische Auseinandersetzung mit der Polykontexturale Logik und den ihr zugrundeliegenden Konzeptionen — wie sie im weiteren Verlauf dieser Studie unternommen wird — als notwendig.
Im folgenden Abschnitt wird die Polykontexturale Logik näher bestimmt und ihr Zusammenhang mit der Morphogrammatik und Kenogrammatik erarbeitet.

2.3 Polykontexturale Logik, Morphogrammatik und Kenogrammatik

2.3.1 Modellierung selbstreferentieller Strukturen in der Polykontexturalen Logik

Die von G. Günther konzipierte *Polykontexturale Logik* (PKL) ging aus seinen umfangreichen Untersuchungen zur Formalisierung der Dialektik Hegels und der Reflexionstheorie Fichtes[40] und seinen Arbeiten zur Formalisierung selbstreferentieller Systeme innerhalb der 'Second Order Cybernetics'[41] hervor.
Ausgehend von einer an die Philosophie des deutschen Idealismus anschließende Kritik der klassischen Logik, ihrer zugrundeliegenden Ontologie und der von ihr implizierten dualistischen logischen Form[42] entwirft Günther eine polykontexturale Ontologie[43]. Die Ontologie der klassischen Logik reduziert die phänomenologisch beobachtbare Vielzahl der Subjekte auf eine einzige universale Subjektivität, die strikt von der objektiven Welt isoliert ist. Im Gegensatz dazu nimmt Günthers polykontexturale Ontologie „*die Immanenz der Subjektivität in der Welt*"[44], sowie „*die Irreduzibilität von Ich–Subjektivität und Du–Subjektivität aufeinander in einem universalen Subjekt*"[45] an.
Diese grundlegende Ontologie der Subjektivität im Kosmos will Günther in seiner Polykontexturalen Logik formalisieren, deren Anspruch er präzisiert: „*Jedes Einzel-*

[39] Wittgenstein bemerkt zur Ontologie der klassischen Logik:
„*Die Logik erfüllt die Welt, die Grenzen der Welt sind auch ihre Grenzen.*
Wir können also in der Logik nicht sagen: Das und das gibt es in der Welt, jenes nicht.
Das würde nämlich scheinbar voraussetzen, daß wir gewisse Möglichkeiten ausschließen und dies kann nicht der Fall sein, da sonst die Logik über die Grenzen der Welt hinaus müßte: wenn sie nämlich diese Grenzen auch von der anderen Seite betrachten könnte." ([Wit87], S. 89f., 5.61.)
Diese Aussage deckt sich mit Günthers Analyse der *monokontexturalen Abgeschlossenheit* der klassischen Logik. Er billigt dieser Ontologie jedoch keine universale Gültigkeit zu, sondern relativiert sie in seiner *polykontexturalen* Logik durch eine Vielzahl von Kontexturen. In jeder dieser Kontexturen ist eine klassische Logik realisiert, wobei jede Kontextur im *Jenseits* der anderen liegt. Die Grenze der Logik erscheint so in Günthers Konzeption nicht als etwas dem Kalkül externes sondern als in ihm abgebildet.
[40] [Gue33], [Gue78], [Gue80b1].
[41] [Gue80], [Gue80b].
[42] [Gue33], [Gue78].
[43] So etwa in seiner Arbeit '*Cybernetic Ontology and Transjunctional Operations*' [Gue80b].
[44] [Gue80], Bd.3, S. 87.
[45] A.a.O.

subjekt begreift die Welt mit der selben Logik, aber es begreift sie von einer anderen Stelle im Sein. Die Folge davon ist: insofern, als alle Subjekte die gleiche Logik benutzen, sind ihre Resultate gleich, insofern aber, als die Anwendung von unterschiedlichen Stellen her geschieht, sind ihre Resultate verschieden. [...] Ein logischer Formalismus [hat] nicht einfach zwischen Subjekt und Objekt zu unterscheiden, er muß vielmehr die Distribution der Subjektivität in eine Vielzahl von Ichzentren in Betracht ziehen. Das aber bedeutet, daß das zweiwertige Verhältnis sich in einer Vielzahl von ontologischen Stellen abspielt, die nicht miteinander zur Deckung gebracht werden können.“[46]
Die PKL erkennt also die zweiwertige Logik als Formulierung der ontologischen Stellung eines einzelnen Subjektes zu *seiner* objektiven Welt an. Die Monokontexturalität der klassischen Logik wird nun nicht durch *intrakontexturale* Modifikationen, sondern durch die *extrakontexturale* Ergänzung einer Vielzahl weiterer Kontexturen erweitert. Die PKL bildet komplexe Systeme ab, indem sie deren Subsysteme über mehrere logische Kontexturen verteilt. Innerhalb dieser lokalen Subsysteme, *intrakontextural*, gilt jeweils eine klassische Logik. Die verteilten Logiken sind durch bestimmte ausserlogische Operationen miteinander verkoppelt, wodurch die Interaktion der das Gesamtsystem konstituierenden Subsysteme modelliert wird.

Dieser Kerngedanke der PKL läßt sich auf die weiter oben charakterisierte Form selbstreferentieller Beschreibungen und Strukturen anwenden. Eine selbstreferentielle Beschreibung besteht aus eine Vielzahl simultan geltender, komplementärer Teilbeschreibungen. Jede Teilbeschreibung repräsentiert einen der jeweiligen Perspektive entsprechenden, widerspruchsfrei formalisierbaren Aspekt. Die PKL kann nun selbstrefentielle Beschreibungen abbilden, indem sie jeder Teilbeschreibung eine eigene logische Kontextur zuweist und das simultane Zusammenspiel der Komplementarität durch Vermittlung der Kontexturen erfasst.
Die oben angeführte Definition eines autopoietischen Systems ließe sich in der PKL beispielsweise als ein mindestens dreikontexturales System abbilden: Kontextur 1 enthält die formale Darstellung des ‘Netzwerkes das Bestandteile produziert’. Ein solches Produktionssystem läßt sich problemlos formalisieren. Kontextur 2 enthält die formale Beschreibung der ‘Bestandteile, die das Netzwerk hervorbringen’. Auch dieser Aspekt läßt sich für sich genommen widerspruchsfrei erfassen. Kontextur 3 beschreibt das komplementäre, simultane Zusammenwirken von Kontextur 1 und 2, welches das autopoietische System als Einheit definiert. Diese rudimentäre Formalisierung ließe sich um zusätzliche Kontexturen erweitern, die bestimmte Aspekte der Umgebung des Systems (z.B. den Beobachter!) und der Wechselwirkung des Systems mit diesen beschreiben. Auf diese Weise lassen sich autopoietische Systeme sowie deren komplexe Wechselwirkungen in einer polykontexturalen Modellierung erfassen.

Die PKL zeichnet sich also durch eine *Distribution* und *Vermittlung* verschiedener logischer Kontexturen aus, wobei innerhalb einer Kontextur — *intra-kontextural* — alle Regeln der klassischen Aussagenlogik ihre vollständige Gültigkeit besitzen. Durch die Vermittlung sind die einzelnen Kontexturen nicht im Sinne einer Typenhierarchie voneinander isoliert, sondern durch besondere *inter-kontexturale* Übergänge miteinander verkoppelt. Da sowohl die Kontexturen in sich, als auch ihre Vermittlung widerspruchsfrei beschreibbar ist, lassen sich somit zirkuläre und selbstreferentielle Strukturen innerhalb der PKL widerspruchsfrei modellieren.
Die innerhalb der monokontexturalen klassischen Logik ausschließlich geltende Form der Dualität ist in der PKL nur noch lokal (je Kontextur) gültig, nicht mehr jedoch

[46] [Gue80], Bd. 3, S. 87.

ausschließlich bzw. global (für das Gesamtsystem). Die strukturelle Erweiterung der dualistischen Form der klassischen Logik zur *Reflexionsform* der PKL ergibt sich aus dem *proemiellen* (d.h. simultanen und parallelen) Zusammenspiel der Distribution und Vermittlung von Kontexturen und ist auf keinen dieser Aspekte allein reduzierbar. Die Distribution regelt die Verteilung von Logiken über Kontexturen und läßt dabei den interkontexturalen Raum unberücksichtigt. Die Vermittlung beschreibt hingegen die interkontexturale Operativität der PKL, kann aber die intrakontexturalen, logischen Aspekte nicht erfassen, da sie von aller logischen Wertigkeit abstrahieren muß. Die Distribution bildet den Hintergrund der Vermittlung, die Vermittlung den Hintergrund der Distribution. Der formale Aufbau der PKL muß dieser fundamental selbstreferentiellen Architektur Rechnung tragen und das proemielle Zusammenspiel der Distribution und Vermittlung vollständig abbilden.
Im Verlauf der Darstellung wird deutlich werden, da die PKL nur im Rahmen ihrer Fundierung durch die Morphogrammatik und Kenogrammatik zu verstehen ist. Die systematische Einordnung der Morphogrammatik innerhalb dieses Fundierungszusammenhangs bestimmt den Platz und Stellenwert der in Teil II entwickelten Formalismen.

2.3.2 PKL und Morphogrammatik

Die Distribution der klassischen Logik über mehrere Kontexturen läßt sich im Rahmen der Faserbündeltheorie als Faserung logischer Kontexturen formalisieren, wobei je Kontextur jeweils eine zur typischen Logik L *strukturisomorphe* Logik L_i operiert[47]. Die Vermittlung der distribuierten Logiken muß die Identifizierung von logischen Werten verschiedener Logiken gewährleisten[48]. Eine solche Identifizierung verschiedener Wahrheitswerte widerspricht der Axiomatik der klassischen Logik und kann weder von der typischen Logik noch von ihren Derivaten L_i geleistet werden.
Zur Beschreibung der Vermittlung der distribuierten Logiken ist nach Günther eine spezielle ausserlogische Theorie notwendig, die die Operativität der Vermittlung in einer nicht auf die typische Logik L reduzierbaren Methode beschreibt. Eine solche Theorie entwirft Günther unter dem Namen *Morphogrammatik* ([Gue80b], [Gue80b1]). Die Morphogrammatik (MG) abstrahiert von der Wertigkeit[49] logischer Operatoren und notiert nur deren *prä-logischen Gestaltaspekt*. In der MG wird die Wahrheitswert-widersprüchlichkeit, die bei einer direkten (d.h. auf dem Identitätsprinzip beruhenden) Vermittlung aufträte, durch die Abstraktion von der Wertigkeit umgangen und die Vermittlung somit in einem prä-logischen Raum widerspruchsfrei abgebildet[50]. Mit der Morphogrammatik ist „*das Niveau eines tiefer [als die Logik] liegenden und allgemeineren Formalismus erreicht, weil aus ihm [der MG] auch das letzte entfernt worden ist, was sich auf den kontingent-objektiven Charakter der Welt bezieht, nämlich der faktische Eigenschaften designierende logische Wert.*“[51]
Die Morphogrammatik stellt einen Versuch dar, einen Standpunkt einzunehmen und formal zu beschreiben, der der phänomenologisch beobachteten Aufspaltung der individuellen Realität in Subjekt und Objekt (oder auch Sein und Nichtsein) vorangeht.

[47] [Pfa88], in Kapitel 9.2 wird diese Konstruktion näher beschrieben.
[48] Das ist notwendig, um z.B. ausdrücken zu können, daß die ‘Bestandteile’ der Kontextur 1 (s.o.) die *selben* sind wie die in Kontextur 2, jedoch in unterschiedlicher Funktionalität beschrieben werden.
[49] Die morphogrammatische Abstraktion lässt sich nicht nur auf Wertlogiken anwenden, sondern ist nach Kaehr ganz allgemein auf die Satz– und Regelstruktur von Logiken und anderen formalen Systemen übertragbar [Kae92].
[50] [Kae92].
[51] [Gue80] Bd.1, S. 216.

Die Anerkennung und Wahl einer solchen Position ist der klassischen abendländischen Tradition, die sich stets innerhalb des Spannungsfeldes der Dualität von On und Me-on bewegte, fremd und läßt analoge Bezüge nur zu östlichen philosophischen Traditionen erkennen[52]. Als Versuch einer Formalisierung dieser Perspektive lassen sich die Kenogrammatik und — als auf die Logik bezogene Theorie — die Morphogrammatik verstehen. „*Die Morphogrammatik [beschreibt] eine Strukturschicht, in der die Differenz zwischen Subjektivität und Objektivität erst etabliert wird und deshalb dort noch nicht vorausgesetzt werden kann.*“[53]Dieser prä-logische Charakter der Morphogrammatik ermöglicht die formal widerspruchsfreie Abbildung der gegen die Axiomatik der Logik verstoßenden Vermittlung mehrerer Logiken in einem polykontexturalen Verbund. Die Morphogrammatik abstrahiert von der Wertigkeit logischer Operatoren und notiert nur deren kenogrammatischen Gestaltaspekt. Von der Designation von Wahrheitswerten, die die Kernaufgabe der Aussagenlogik ist, wird durch diesen Schritt vollständig abgesehen. Durch die Wertabstraktion wird die Wahrheitswertwidersprüchlichkeit einer auf dem ontologisch-logisch-semiotischen Identitätsprinzip beruhenden Kontexturvermittlung umgangen.

Durch Operationen auf der Gestaltebene des Logischen konstruiert Günther komplexe morphogrammatische Systeme, denen er wiederum — in Form der Stellenwertlogik oder der PKL — logische Interpretationen zuweist.

Die auf ihrer morphogrammatischen Konstruktion beruhende Mehrwertigkeit der PKL unterscheidet sich fundamental von den konservativ erweiterten mehrwertigen Logiken, da die zusätzlichen Werte nicht bestimmte zusätzliche *intra-kontexturale* Unterscheidungen, sondern die *inter-kontexturale* Operativität der polykontexturalen Systemkonzeption designieren.

Die Konstruktion der Polykontexturalen Logik wird durch die Morphogrammatik fundiert. Für eine vollständige Formalisierung der PKL, die bis jetzt nicht geleistet wurde, ist daher auch eine formale Ausarbeitung der Morphogrammatik notwendig. Eine solche Ausarbeitung der MG im formalen Sprachrahmen der klassischen Mathematik soll in dieser Arbeit geleistet werden. Die Morphogrammatik ist eine Umformungstheorie sublogischer Strukturen, die sich allgemein auf formale Systeme anwenden lässt und es ermöglicht, klassifikatorische Probleme dieser Systeme zu lösen (vgl. etwa Kapitel 9.1). Aufgrund der Allgemeinheit ihrer Konzeptionen, Strukturen und Operationen ist die Morphogrammatik, unabhängig von ihrer fundierenden Rolle für die PKL, ein eigenständiger Kalkül mit interessanten inhaltlichen Bezügen zur Semiotik, Arithmetik, Kombinatorik, Logik und Informatik. Eine ausführliche Darstellung dieser Aspekte der Morphogrammatik erscheint daher ebenfalls angebracht. Die Bezüge der Morphogrammatik zur Semiotik und Arithmetik werden in den Kapiteln 3 und 4 untersucht. Die Darstellung und Ausarbeitung der Morphogrammatik im engeren Sinne geschieht in Kapitel 5. In Teil III werden Struktur und Dynamik morphogrammatischer Operationen mit Hilfe graphentheoretischer und kombinatorischer Methoden klassifiziert und analysiert. Die Einbettung logischer Systeme in die Morphogrammatik wird in Teil IV behandelt. Auf die Konstruktion der PKL aus klassischen Logiken sowie das Verhältnis von PKL, Morphogrammatik und Kenogrammatik wird dabei gesondert in Kapitel 9.2 eingegangen.

[52]Vgl. etwa [Var92].
[53][Gue80], Bd.1, S. 216.

2.3.3 Kenogrammatik als Theorie der Zeichenprozesse

Die im Vorhergehenden rudimentär skizzierte Architektur der PKL als eine System der Distribution und Vermittlung logischer Kontexturen wirft Fragen nach der formalen Konstruktion eines solchen Mechanismus auf.

1. Die klassische Logik ist monokontextural geschlossen, d.h. es existiert nur ein logischer Ort, der eben genau von der klassischen Logik ausgefüllt wird, Ort und Logik fallen zusammen, so daß die *Orthaftigkeit*[54] der Logik nicht thematisiert werden kann. In der klassischen Logik gibt es keinen Ort der Logik, da es nur *eine* nicht hintergehbare Logik gibt, die universell (d.h für jeden Beoabachter und überall) gilt, wodurch, ontologisch interpretiert, die Orthaftigkeit des Beobachters verdeckt wird. Die Konstruktion der PKL setzt in ihrer Distribution von Kontexturen das Vorhandensein logischer Orte voraus. Klassische Kalküle sind monokontextural geschlossen, da sie auf der Logik fundieren. Die Orthaftigkeit eines formalen Systems — insbesondere die Verteilung formaler Systeme über eine Vielzahl logischer Orte — ist deshalb mittels klassischer Kalküle nicht (oder nur näherungsweise) abzubilden.

2. Wie bereits gezeigt, muß die morphogrammatische Vermittlung der Kontexturen von der Wertigkeit logischer Operationen, und demzufolge vom *semiotischen Identitätsprinzip* abstrahieren, um widerspruchsfrei Wahrheitswertkollisionen der logischen Ebene notieren zu können. Da der klassische Kalkülbegriff an das logische und semiotische Identitätsprinzip gebunden ist, ergibt sich hier ebenfalls eine grundlegende Darstellungsschranke, die klassische Kalküle als zur Formalisierung der PKL ungeeignet erscheinen läßt.

3. Die strukturelle Erweiterung der dualistischen Formkonzeption zur '*Reflexionsform*' ergibt sich, wie in der obigen Skizze der PKL umrissen, erst aus dem Zusammenspiel der *Distribution* und *Vermittlung* logischer Kontexturen, ist jedoch auf keinen dieser Teilsaspekte allein reduzierbar. Die Distribution beschreibt die Verteilung von Logiken über logische Orte und läßt den *interkontexturalen* Raum zwischen den Orten unthematisiert. Die Vermittlung beschreibt gerade die interkontexturale Operativität der PKL, kann aber die *intrakontexturalen*, logischen Aspekte nicht erfassen (da sie von aller logischen Wertigkeit abstrahiert). Distribution gibt den Hintergrund der Vermittlung ab, Vermittlung den Hintergrund der Distribution[55]. Wie in Abschnitt 2.2 analysiert, kann eine solche fundamental zirkuläre Architektur nicht im Sprachrahmen klassischer formaler Systeme abgebildet werden.

Diese drei grundlegenden Anforderungen an die Formalisierung der Polykontexturalen Logik zeigen deutlich, daß sie sich mit Hilfe klassischer Methoden nicht formalisieren läßt. Die Grenzen des im klassischen Sinne Formalisierbaren werden durch die Logik und Semiotik bestimmt. Hält man jedoch wie Günther an dem Ansatz fest, den „*Begriff der logischen Form [...] zu generalisieren und in einem transklassischen Sinn, der der Problematik der Transzendentallogik entspräche, [formal] zu erweitern,*“[56] erweist es sich als notwendig, eine von den Restriktionen der Semiotik und Logik unabhängigen, allgemeinen Theorie formaler Prozesse einzuführen.

[54] D.h. die Eigenschaft einen bestimmten Ort im Sein einzunehmen.

[55] Die PKL, die zur Formalisierung selbstreferentieller Strukturen benötigt wird , erweist sich nach dieser Überlegung gerade selbst als fundamental selbstreferentiell strukturiert [Kae90].

[56] [Gue80b1], S. 74.

„*Eine solche tiefere und wirklichkeitsfreiere Formalstruktur des Logischen [...], die zwar von Werten [allg. formalen Prozessen] besetzt werden kann, die aber nicht mit ihnen identisch ist,*“[57] konzipiert Günther unter dem Namen *Kenogrammatik*[58]. Die Definition der Kenogramme als Leerstellen, an denen semiotische Prozesse eingeschrieben werden können, ordnet der Semiotik die Kenogrammatik vor, da sie die Notierung semiotischer Prozesse erst ermöglichen, d.h. ihnen einen Ort der Realisierung geben. Diese Theorie bietet somit in Form der Kenogramme und Kenogrammkomplexionen ein Instrumentarium zur formalen Thematisierung der Orthaftigkeit von Zeichenprozessen (ad 1).
Die Kenogrammatik stellt einen Bereich des Formalen dar, der allen klassischen Logik- und Formkonzeptionen vorangeht, sie ist „*eine Strukturtheorie, die noch nicht durch die Differenz von Form und Materie belastet ist*“[59]. In ihr ist insbesondere das logische und semiotische Identitätsprinzip nicht gültig (ad 2).
Das zirkuläre, wechselseitige Fundierungsverhältnis der Distribution und Vermittlung logischer Kontexturen (und aller entsprechenden selbstreferentiellen Strukturen) wird von Günther eingehend analysiert und als *proemielles* Verhältnis charakterisiert[60]. Nach Günther ist diese Proemialität in der Kenogrammatik abbildbar (ad 3). Da die Lösung der drei oben skizzierten grundlegenden Formalisierungsprobleme der PKL:

1. die Notierung der Orthaftigkeit formaler Prozesse,
2. die Aufhebung des logisch-semiotischen Identitätsprinzips und
3. die Modellierung der Proemialität

von der Güntherschen Kenogrammatik beansprucht wird, erscheinen Morphogrammatik und PKL als in ihr notierbar.
Die Strukturformen logischer Operationen, die Morphogramme, bilden das Operandensystem komplexer Operationen in der Morphogrammatik (vgl. Kapitel 5). Die Kenogrammatik beschreibt nicht diese Operativität der *äußeren* Gesamtgestalt der Morphogramme in komplexeren Gebilden, sondern thematisiert ihre *interne* Formstruktur als ein System von Gleichheiten und Verschiedenheiten. Die Kenogrammatik ist die allgemeine Theorie solcher Leerstellengebilde, wie sie etwa Morphogramme darstellen. In Günthers Konzeption bildet die Kenogrammatik den notationalen Rahmen der Morphogrammatik und der Polykontexturalen Logik. Dieses Fundierungsverhältnis entspricht dem Verhältnis von Semiotik und Kalkülbegriff[61] in der klassischen Mathematik. Die Kenogrammatik stellt gewissermaßen eine ‘transklassische Semiotik’ für einen durch die PKL repräsentierten transklassischen Kalkülbegriff dar.
Im zweiten Teil der Arbeit wird nun zunächst die formale Einführung der Kenogrammatik geleistet, soweit dies im Rahmen klassischer Methoden möglich ist (Kapitel 3). In Kapitel 4 werden kenogrammatische Strukturen und Operationen anschließend einer arithmetischen Interpretation und Analyse unterzogen, die sich besonders mit der klassischen Arithmetik auseinandersetzt. Fußend auf den Konzepten und Methoden der Kenogrammatik wird dann in Kapitel 5 ein definitorischer Aufbau der Morphogrammatik präsentiert. Aufgrund der besonderen Bedeutung, die der Proemialrelation in der Konzeption der PKL zukommt, ist ihrer operationalen Modellierung ein eigenes Kapitel vorbehalten (Kapitel 10).

[57] [Gue80b1], S. 90.
[58] Gr. kenos : leer. [Gue80], [Gue80c].
[59] [Gue80], Bd.3, S. 115.
[60] [Gue80a], [Gue80a1].
[61] [Ass65]

Teil II

Darstellung

Kapitel 3

Kenogrammatik

Je mehr Äußerung, desto stiller
Je stiller, desto mehr Äußerung.
F. Hölderlin

3.1 Intention der Kenogrammatik

Günthers Bemühungen um eine Generalisierung des Begriffes der logischen Form und seiner Erweiterung im Sinne eines transklassischen Formbegriffs motivierten ihn zur Einführung einer allgemeinen Strukturtheorie des Logischen, der Morphogrammatik. Die Strukturformen logischer Operationen, die Morphogramme, bilden das Operandensystem komplexer morphogrammatischer Operationen. Innerhalb der morphogrammatischen Theorie wird hierbei die Operativität der *äußeren* Gesamtgestalt der Morphogramme in komplexen Strukturen untersucht. Günther wies die *morphogrammatische Unvollständigkeit*[1] der klassischen Aussagenlogik nach und stieß damit auf die Notwendigkeit, „*den kenogrammatischen Unterbau der Logik prinzipiell zu erweitern,*“[2] um die anvisierte Generalisierung der Logik vollziehen zu können. Die Kenogrammatik ist die allgemeine Theorie von *Leerstellenkomplexionen* (wie etwa Morphogramme). Sie beschreibt die *interne* Formstruktur solcher Komplexionen als ein System von Gleichheiten und Verschiedenheiten.
In Günthers Konzeption bildet die Kenogrammatik mit „*dem Aufbau einer universalen kenogrammatischen Struktur der Logik*“[3] den notationalen Rahmen der Morphogrammatik und der Polykontexturalen Logik. Dieses Fundierungsverhältnis entspricht dem Verhältnis von Semiotik und Kalkülbegriff[4] in der klassischen Mathematik. Die Kenogrammatik stellt gewissermaßen eine ‘transklassische Semiotik’ als Fundierung eines transklassischen Kalkülbegriffs dar.
Bevor im Abschnitt 3.3 eine formale Einführung der Kenogrammatik entwickelt wird, soll in den beiden folgenden Abschnitten die Kenogrammatik zunächst vor dem Hintergrund des angedeuteten Fundierungsverhältnisses motiviert und näher bestimmt werden. Im weiteren Verlauf dieses Abschnitts 3.1 werden die im vorherigen Kapitel postulierten Eigenschaften einer kenogrammatischen Theorie:

[1] [Gue80b1], S. 94.
[2] [Gue80c], S. 110.
[3] [Gue80c], S. 110.
[4] [Ass65].

1. die Notierung der Orthaftigkeit formaler Prozesse,

2. die Aufhebung des logisch-semiotischen Identitätsprinzips und

3. die Abbildung der Proemialität

in einem kontrastierenden Vergleich zur klassischen Semiotik deutlicher bestimmt.
Im nächsten Abschnitt 3.2 wird anhand der Diskussion des klassischen Kalkülbegriffs die Frage nach der Formalisierbarkeit der Kenogrammatik aufgeworfen. In Analogie und Abgrenzung zur klassischen Theorie wird dann Günthers Konzept des transklassischen Kalküls erörtert. Erst nach dieser grundlegenden Reflexion auf den Gebrauch klassischer Mathematik zur Formalisierung transklassischer Konzepte wird ein definitorischer Aufbau der Kenogrammatik entwickelt. Diese Vorüberlegungen sind notwendig, um möglichen Mißverständnisse zum Status und Stellenwert einer klassischen Formalisierung der Güntherschen Theorien zu begegnen.

3.1.1 Kenogrammatik und Semiotik

3.1.1.1 Das klassische Zeichenkonzept

Alle klassischen Kalküle und Formalismen sind dem klassischen Kalkülbegriff untergeordnet, der wiederum auf der Zeichen- und Zeichenreihenkonzeption der *Semiotik*[5] basiert.
Die *Ausdrücke* (oder Terme) eines solchen Kalküls sind *Zeichenreihen*, die durch *lineares Aneinanderreihen* von jeweils endlich vielen *Grundzeichen* entstehen, die einem Alphabet von *wohlunterschiedenen* (d.h. mit sich selbst identischen und von allen anderen verschiedenen) Atomzeichen entstammen.
Der Begriff der *Gleichheit* von Zeichenreihen abstrahiert von der physikalischen Realisierung und benutzt die idealisierte Zeichenreihen*gestalt* als Gleichheitskriterium: Zwei Zeichenreihen sind gleich, wenn sie die gleiche Gestalt haben. Die Zeichenreihengestalt oder kurz Zeichengestalt meint also nicht nur eine individuelle Realisation einer Zeichenreihe, sondern stets die gesamte Klasse aller Zeichenreihen, die mit dieser Zeichenreihe *gleichgestaltet* sind (aber physikalisch durchaus unterschiedlich realisiert sein können). Dieses Verhältnis zwischen den konkreten Zeichenreihenvorkommnissen (*Token*) und der idealisierten Zeichenreihengestalt (*Type*) ist als *Token–Type–Relation* bekannt. Das Abstraktum aller gleichen Zeichenreihen–Tokens ist der Zeichenreihen–Type. Zwei Token sind genau dann gleich, wenn sie dem selben Type angehören.
Das lineare Aneinandereihen von Zeichengestalten geschieht in der Zeichentheorie durch eine im algebraischen Sinne assoziative Verkettungsoperation, die jedem Paar z_1, z_2 von Zeichengestalten eine eindeutig bestimmte Zeichengestalt $z_3 = z_1 z_2$ zuordnet[6].
Der semiotischen Zeichenreihentheorie wird nun die Konzeption von *Kenogramm* und *Kenogrammkomplexion* gegenübergestellt, um die spezifische Perspektive der Güntherschen Kenogrammatik zu verdeutlichen.

[5] Semiotik wird hier als *allgemeine Zeichentheorie* verstanden, wie sie im Rahmen der Grundlagenuntersuchungen der mathematischen Logik aus der Gruppentheorie abgeleitet wird. [Ass65], S. 174 ff.

[6] Die Menge aller Zeichengestalten bildet in Bezug auf die Verkettungsoperation eine *freie Halbgruppe mit Nullelement*, wobei die Menge aller Atomgestalten ein *freies Erzeugendensystem* für diese Halbgruppe ist. [Ass65], S. 172ff.

3.1.1.2 Kenogramme und Kenogrammkomplexionen

Kenogrammkomplexionen sind aus einzelnen Kenogrammen aufgebaut, die Bildung von Komplexionen unterscheidet sich jedoch fundamental vom linearen Aufbau der Zeichenreihen, der auf dem atomistischen Grundzeichenkonzept und der eindeutigen und assoziativen Verkettungsoperation beruht.
Die Atomzeichengestalten sind als selbstidentische definiert, so daß in der Semiotik bereits die Gleichheit oder Verschiedenheit von isolierten Atomzeichengestalten bestimmt werden kann. Die Kenogrammatik verzichtet bewußt auf die Konzeption des Atomzeichen–Types. Verschiedene Realisationen von einzelnen, isolierten Kenogrammen können daher nicht unterschieden werden: $\circ \overset{t}{\equiv} \triangle \overset{t}{\equiv} \square \overset{t}{\equiv} \ldots$[7]. Da einzelne Kenogramme nicht unterscheidbar sind, besitzt die Kenogrammatik kein Alphabet von Atomzeichengestalten.
Während in der Zeichentheorie die Gleichheit von Zeichenreihen gerade induktiv über die Gleichheit der Atomzeichen definiert ist, existiert in der Kenogrammatik eine Möglichkeit der Unterscheidung von Gleichheit und Verschiedenheit erst auf der Ebene der Kenogrammkomplexionen. Der kenogrammatische Gleichheitsbegriff basiert nicht auf den selbstidentischen (d.h. von anderen unterschiedenen) Atomzeichen, sondern auf den strukturellen Verhältnissen von Identität und Differerenz innerhalb eines Ganzen (der Kenogrammkomplexion). Vergleiche von einzelnen Kenogrammen sind nur innerhalb von Kenogrammkomplexionen möglich, nicht jedoch zwischen Kenogrammen aus unterschiedlichen Komplexionen[8]. Da für Kenogrammkomplexionen kein Type existiert, wird die Gleichheit von Kenogrammkomplexionen ohne die Token–Type–Relation der Semiotik definiert.
Kenogrammkomplexionen sind nicht durch semiotische Zeichen fixiert (Zeichen *für* Struktur). Sie stellen selbstdifferenzierende strukturelle Ordnungen dar und sind als solche *Realisierung von Struktur*. Relationalität ist in der Kenogrammatik ohne Rückgriff auf die Form der Selbstidentität und des Atomismus der Semiotik formulierbar [Dit82].
Die semiotische Verkettungsoperation verbindet zwei Zeichenreihengestalten z_1 und z_2 stets eindeutig zu $z_1 z_2$, ohne daß der innere Aufbau von z_1 oder z_2 berücksichtigt würde; die selbstidentischen Atomzeichen, die z_1 und z_2 konstituieren, bleiben gegenüber der Verkettungsoperation und allen anderen Zeichenprozessen invariant. Im Gegensatz dazu muß bei der Verkettung von Kenogrammkomplexionen berücksichtigt werden, daß unterschiedliche Kenogramme nur *innerhalb* einer Komplexion von einander unterschieden sind, keinesfalls jedoch als isolierbare, selbstidentische Zeichen interpretiert werden können.
An eine semiotische Zeichenreihe z_1 kann ein einzelnes Atomzeichen a ohne Berücksichtigung der internen Struktur von z_1 angehängt werden: $z_1 a$. Die Verkettung einer Kenogrammkomplexion K_1 mit einem einzelnen Kenogramm k kann hingegen nur unter Rückbezug auf die innere Struktur von K_1 geschehen, da k kein wohlunterschiedenes Atomzeichen ist. An der Stelle, die das Einzelkenogramm k in der verketteten Komplexion einnimmt, kann eines der bereits in K_1 enthaltenen Kenogramme wiederholt werden oder aber ein neues Kenogramm eingeführt werden. Diese *Rückbezüglichkeit* bei der Verkettung läßt eine Kenogrammkomplexion als *Ganzheit* oder

[7] Zur Definition der kenogrammatischen Äquivalenzen vgl. 3.3.1.

[8] Matzka interpretiert diese Eigenschaft als ein *lokales Type–Konzept*, da Kenogramme nur lokal innerhalb von Komplexionen, nicht jedoch global wie semiotische Zeichen vergleichbar sind. (Matzka, R.: *Semiotische Abstraktionen bei Gotthard Günther und George Spencer–Brown*. In: Acta Analytica, (in Prep.), 1993)

Gestalt entstehen, die nicht auf eine lineare Verkettung reduzierbar ist. Die Komplexion ist nicht über Atomzeichen definiert, sondern stellt ein zwischen den Plätzen (*Orten*) der einzelnen Kenogramme aufgespanntes Relationsgebilde dar.
Da nicht Zeichen- sondern Strukturidentität das entscheidende Merkmal der Komplexionen ist, sind sie beliebig notierbar. Enthalten beispielsweise zwei zu verkettende Kenogrammkomplexionen K_1 und K_2 jeweils das Kenogramm ○, so sind für die Verkettung der beiden Komplexionen das Kenogramm ○ in K_1 und das Kenogramm ○ in K_2 *nicht* im Sinne der Zeichentheorie als zwei Vorkommnisse der gleichen Atomzeichengestalt miteinander zu identifizieren. Da ○ eine willkürlich gewählte Notation für eine Leerstelle ist, ist es möglich, die beiden durch ○ notierten Leerstellen von K_1 und K_2 innerhalb der verketteten Komplexion durch unterschiedliche Kenogrammsymbole zu notieren. Aus diesem Grund können durch die Verkettungsoperation mehrere verschiedene Komplexionen entstehen. Die kenogrammatische Verkettung ist also keine eindeutige Operation, sondern vielmehr eine mehr-eindeutige nicht-assoziative Relation (Zur Definition der kenogrammatischen Verkettung vgl. (3.3.5.2))..

3.1.1.3 Der Ort des Zeichenprozesses

Der semiotische Begriff des Atomzeichen abstrahiert von der (physikalischen) Realisierung eines Zeichensystems[9], so daß der ontologische Status von Zeichen, daß sie nämlich einen *Ort im Sein* einnehemen, nicht thematisiert werden kann. In der Semiotik gibt es mithin keinen Begriff des Ortes von Zeichensystemen, sondern nur einen — nichtthematisierten — Universalort der Semiosis. Die Kenogrammme der Kenogrammatik sind als Leerstellen (als Orte) intendiert, an denen semiotische Zeichenprozesse eingeschrieben werden können. In der Kenogrammatik existiert also eine fundamentale Differenz zwischen Ort und Zeichen (und nicht wie in der Semiotik eine Ineins-setzung). Somit ist in der Kenogrammatik die Orthaftigkeit von Zeichenprozessen notierbar.
Die Kenogrammatik geht historisch und konstruktiv aus der Semiotik hervor, kenogrammatische Strukturen werden zunächst als Abstraktionen semiotischer Zeichenreihen definiert (*Kenosis*). Da die semiotischen Gesetzmäßigkeiten für die kenogrammatischen Strukturen aber nicht mehr gelten, können sie nicht als abgeleitete semiotische Konstrukte betrachtet werden. Vielmehr erweisen sich Zeichen vom erweiterten Standpunkt der Kenogrammatik als Reduktionen oder Kristallisationen von Kenogrammen. Die Semiotik kann Zeichen nur als aus einem schon gegebenen Alphabet stammend voraussetzen, den semiotischen Zeichen ist aber die Semiose, der Prozeß der Zeichengenerierung selbst vorgeordnet. Die Kenogrammatik, insofern sie den Prozeß der Semiose notierbar macht, muß also der Semiotik systematisch vorgeordnet werden, da sie diese überhaupt ermöglicht.
Die im Vorhergehenden genannten Verknüpfungseigenschaften der Kenogramme und Kenogrammkomplexionen lassen sich mittels klassischer Formalismen beschreiben, wie dies in 3.3 geschieht. Eine solche Formalisierung wird jedoch dem Anspruch der Kenogrammatik nicht gerecht, nicht nur eine spezielle algebraische Struktur (z.B. eine Wortalgebra) darzustellen, sondern darüber hinaus als 'transklassische Semiotik' die Fundierung eines transklassischen Kalkülbegriffs. Diese Interpretation der Kenogrammatik als eine allgemeine Theorie der Zeichenprozesse hebt sie von klassischen Algebren ab. Ein Vergleich zwischen der klassischen Semiotik und der Kenogrammatik müßte als Verwechselung der Beschreibungsebenen und nicht als angemessene

[9] Dies ist der Inhalt der Token–Type–Relation

Abgrenzung eingeordnet werden, falls sie nicht wie hier vor dem Hintergrund dieses transklassischen Anspruchs geschähe.

3.1.2 Die Proemialrelation

Im Vorhergehenden wurden die Konzepte von Kenogramm und Kenogrammkomplexion gegenüber semiotischen Zeichen- und Zeichenreihenkonzeptionen abgegrenzt. Neben dieser den statischen und strukturellen Aufbau der Kenogrammatik betreffenden Analyse erscheint auch die Bestimmung ihrer dynamischen und operationalen Beschaffenheit notwendig, um die Kenogrammatik angemessen zu charakterisieren. Die von Günther eingeführte *Proemialrelation „gehört zur Ebene der kenogrammatischen Strukturen*“[Gue80a] und beschreibt die in der Kenogrammatik mögliche Notierung von 'Relationalität' und Operativität, die aller dualistischer Aufspaltung in Subjekt-Objekt, Form-Inhalt usw. vorangeht[10].
„Es gibt einen deutlichen Unterschied zwischen der symmetrischen Umtauschrelation, wie sie zum Beispiel die Negationstafel in der zweiwertigen Logik darstellt, und dem Umtausch von Relator und Relatum. In der klassischen Symmetrierelation wechseln die beiden Relata lediglich ihre Plätze. Formal ausgedrückt:

$$R(x, y) \tag{3.1}$$

wird zu

$$R(y, x) \tag{3.2}$$

Hierbei ändert sich materiell überhaupt nichts. Wenn dagegen der Relator die Stelle eines Relatums einnimmt, dann ist der Umtausch nicht wechselseitig. Der Relator kann zum Relatum werden, doch nicht in der Relation, für die er zuvor die Beziehung einrichtete, sondern nur relativ zu einem Verhältnis bzw. Relator höherer Ordnung. Umgekehrt kann das Relatum zum Relator werden, jedoch nicht in Bezug auf das Verhältnis, in dem es als relationales Glied — als Relatum — aufgetreten ist, sondern nur in Bezug auf Relata niedrigerer Ordnung. Wenn

$$R_{i+1}(x_i, y_i)$$

gegeben ist und das Relatum x oder y zum Relator wird, dann erhalten wir

$$R_i(x_{i-1}, y_{i-1})$$

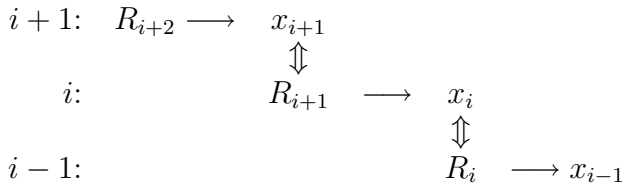
wobei $R_i = x_i$ oder y_i ist. Wenn dagegen der Relator zu einem Relatum wird, dann erhalten wir

$$R_{i+2}(x_{i+1}, y_{i+1})$$

wobei $R_{i+1} = x_{i+1}$ oder y_{i+1} ist. Der Index i bezeichnet höhere oder niedrigere logische Ordnung. Wir nennen diese Verbindung zwischen Relator und Relatum das Proemialverhältnis, *da es der symmetrischen Umtauschrelation und der Ordnungsrelation vorangeht und — wie wir sehen werden — ihre gemeinsame Grundlage bildet.*“([Gue80a1], p.33)
Die Ordnungsrelation bestimmt also immer das hierarchische Verhältnis des Relators R_{i+1} zum Relatum x_i innerhalb einer logischen Stufe i. Die Umtauschrelation bezieht sich auf den Wechsel zwischen dem Relator R_i einer Relation der Stufe $i-1$ und dem Relatum x_i der logischen Stufe i. Dieser Zusammenhang ist im folgenden Diagramm

[10] gr. prooimion: Vorspiel

(3.1) abgebildet. Die Proemialrelation stellt sich somit als ein ineinandergreifender Mechanismus von Umtausch und Ordnung dar und kann in zweifacher Weise interpretiert werden. Zum Einen läßt sich Proemialität als ein Umtausch deuten, der auf Ordnung basiert. Die Ordnung innerhalb einer logischen Stufe i ist jedoch dadurch begründet, daß ein Relator R_i der Stufe $i-1$ durch die Umtauschrelation zum Relatum x_i der Stufe i wird und daß gleichfalls der Relator R_{i+1} durch Umtausch des Relatums x_{i+1} auf die Stufe i versetzt wird. Somit ist zum Anderen Proemialität auch als Ordnung zu verstehen, die auf Umtausch gründet.



R_i: Relator R_i der Stufe i
x_i: Relatum x_i der Stufe i
$\longrightarrow$: Ordnungsrelation zwischen einem Relator R_i und einem Relatum x_{i-1}
$\Updownarrow$: Umtauschrelation zwischen einem Relator R_i und einem Relatum x_i

Abbildung 3.1:
Die Proemialrelation

„Weder die Umtauschrelation noch die Ordnungsrelation wären uns begreiflich, wenn unsere Subjektivität nicht in der Lage wäre, zwischen einem Relator überhaupt und einem einzelnen Relatum ein Verhältnis herzustellen. Auf diese Weise stellt das Proemialverhältnis eine tiefere Fundierung der Logik bereit, als ein abstraktes Potential, aus dem die klassischen Relationen des symmetrischen Umtauschs und der proportionalen Ordnung hervorgehen.
Dies ist so, weil das Proemialverhältnis jede Relation als solche konstituiert. Es definiert den Unterschied zwischen Relation und Einheit oder — was das gleiche ist — zwischen der Unterscheidung und dem was unterschieden ist — was wiederum das gleiche ist — wie der Unterschied zwischen Subjekt und Objekt.“([Gue80a1], S. 33)
„Der von der Proemialrelation bewirkte Umtausch ist einer zwischen höherer und niedrigerer relationaler Ordnung. Wir können beispielsweise ein Atom als Relation zwischen mehreren Elementarpartikeln betrachten, wobei letztere dann den Part der Relata einnehmen. Wir können jedoch auch sagen, daß das Atom ein Relatum in einer komplexeren Ordnung darstellt, die wir als Molekül bezeichnen. Folglich ist ein Atom beides: ein Relator relativ zu den Elementarpartikeln, jedoch kann es diese Eigenschaft mit der eines Relatums vertauschen, wenn wir es innerhalb der umfassenderen Relation (Relator) eines Moleküls betrachten.“([Gue80a1], S. 34)
Die Proemialität läßt sich somit als eine Begriffsbildung verstehen, die es ermöglicht, ein bestimmtes Objekt auf mehrere logische Stufen (Bezugssysteme) verteilt in verschiedenen Funktionalitäten zu erfassen. Der fundamentale Unterschied zwischen der Proemialrelation wie sie von Günther intendiert ist und klassischen Konzepten logischer Stufung in Objekt- und Metaebenen ist die *Simultaneität* der Ebenen innerhalb der Proemialrelation[11], die sich der klassischen Darstellung entzieht. So ist ja das

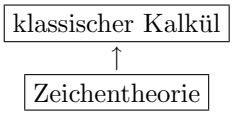
[11] Eine umfassende kategorientheoretische Analyse der Proemialrelation als *Fundierungsrelation*

von Günther angeführte Beispiel des Atoms so zu verstehen, daß das Atom *simultan* Relatum (bzgl. des Moleküls) und Relator (bzgl. seiner Partikel) ist, und durch dieses Zusammenwirken erst konstituiert wird.
Für das Verständnis der folgenden Darstellung ist diese informelle Charakterisierung der Proemialrelation zunächst ausreichend. Da die Proemialrelation jedoch eines der fundamentalen Konzepte des gesamten Güntherschen Theoriekomplexes darstellt[12], wäre diese Arbeit ohne eine formale Charakterisierung der Proemialrelation unvollständig. Aus diesem Grund wird in Kapitel 10 eine operationale Modellierung der Proemialrelation entwickelt.

3.2 Zur Formalisierbarkeit der Kenogrammatik

3.2.1 Fundierung des klassischen Kalküls in der Semiotik

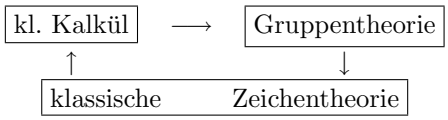
Für ein Verständnis des in dieser Arbeit gewählten Formalisierungsansatzes erscheint es zunächst als notwendig, das in der Einführung angesprochene Verhältnis von Semiotik und Kalkültheorie näher auszuführen. Ein klassischer Kalkül basiert fundamental auf dem semiotischen Zeichenreihenkonzept, ohne das kein mathematischer Ausdruck 'schreibbar' wäre. Dies wird durch das folgende Schema 3.2 veranschaulicht:



x ⟶ y: y ist in x notierbar

Abbildung 3.2: Semiotik und klassischer Kalkülbegriff

Innerhalb des so in Bezug auf die Notierbarkeit begründeten Kalkülkonzeptes lassen sich nun alle klassischen mathematischen Theorien formulieren. So z.B die Gruppentheorie der Algebra. Wie Asser [Ass65] ausführt, läßt sich die Semiotik ihrerseits in ihrer vollständigen Funktionalität innerhalb der Gruppentheorie als freie Halbgruppe mit Nullelement darstellen. Wir gelangen also zu Schema 3.3, das deutlich die zirkuläre Struktur zeigt:



x ⟶ y: y ist in x notierbar

Abbildung 3.3: Die Zirkularität von Semiotik und Kalkülbegriff

von Objekt– und Subjektkategorien findet sich in [Dit90], S. 91–108.

[12]So wurde bereits in (2.3) die Grundstruktur der PKL als ein proemielles Zusammenspiel der Distribution und Vermittlung formaler Systeme beschrieben. Vgl. auch Kapitel 9.2

Analoge Zirkularitäten lassen sich in allen Versuchen, die natürlichen Zahlen definitorisch einzuführen, nachweisen und haben in der mathematischen Grundlagenforschung das Problem der Begründbarkeit der klassischen Theorie aufgeworfen. „*Das Ideal des Rationalismus, den Zahlbegriff [mithin die gesamte kl. Mathematik] zu begründen [...], scheitert [...] aus drei Gründen (sog. 'Münchhausen–Trilemma'):*

1. *Wir geraten in einen Zirkel, der gar nichts aussagt.*

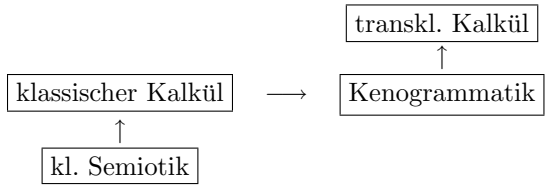
2. *Versucht man die Begründung mit Hilfe einer Metasprache (wie in der formalistischen Mathematik), um diesem Zirkel zu entgehen, so hat man die Rechtfertigungsfrage auf immer höheren Metastufen zu stellen, was zu einem* unendlichen Regreß *führen würde.*

3. *Bricht man dieses Verfahren an einer Stelle ab, so beruft man sich auf eine willkürliche* dogmatische Entscheidung, *die natürlich auch keine zwingende Begründung darstellt.*

Hier tritt noch ein weiteres Problem auf: In der Grundlagenkrise kam nicht nur die Sicherheit der Fundamente der Mathematik ins Wanken, sondern auch die Sicherheit der logischen Herleitung von Erkenntnissen aus den Fundamenten. Kann die Logik selbst *begründet werden? Da man bei einer solchen Begründung die logischen Regeln benutzen müsste, scheitert sie ebenso nach dem 'Münchhausen-Trilemma'! [...] Sie kann jedoch mit rationaler Argumentation auch nicht verworfen werden, weil das rationale Argumentieren durch die Regeln des schematischen Operierens [der Logik] geradezu definiert ist.*“[13]

3.2.2 Fundierung des transklassischen Kalküls in der Kenogrammatik

Anspruch der Kenogrammatik ist es, den Gesetzen der klassischen Semiotik entbunden, einen *trans*klassischen Kalkülbegriff begründen zu können, der ebenfalls nicht mehr an die klassischen Konzepte gebunden ist, diese jedoch vollständig umfasst.



x ⟶ y: y ist in x notierbar

Abbildung 3.4: Kenogrammatik und transklassischer Kalkül

Aus diesem Schema wird ersichtlich, daß die Kenogrammatik als eine 'transklassische Semiotik' verstanden werden kann, insofern sie als notationelle Fundierung eines transklassischen Kalkülkonzeptes dient. Gleichzeitig wird hier aber auch das Problem

[13] [Bac85], p. 228f.

ersichtlich, daß die Kenogrammatik zunächst mit den Methoden der klassischen Semiotik und des auf ihr gründenden klassischen Kalkülkonzeptes formalisiert werden muß. Denn ein anderer als der klassischer Kalkülbegriff existiert eben nicht, bzw. soll ein solcher gerade erst in der Kenogrammatik fundiert werden. Die solchermaßen formulierte Kenogrammatik bleibt selbstverständlich an die Eigenschaften der Semiotik gebunden. Alle bisherigen Formalisierungsansätze und gleichfalls die hier vorgestellten Formalismen und Implementierungen repräsentieren diesen unvollständigen Entwicklungsstand der Kenogrammatik. Dieser bisher formalisierte Bereich ist also keinesfalls mit der Kenogrammatik identisch und wird im Folgenden mit KG_1 bezeichnet, da er eben nur die erste Stufe einer Formalisierung der Kenogrammatik darstellt.
Nun kann KG_1 einerseits isoliert als klassischer Kalkül betrachtet werden, der der Semiotik und dem klassischen Kalkülbegriff unterworfen ist[14]. Andererseits kann KG_1 aber auch als erstes Glied einer Kette von Formalisierungsstufen angesehen werden, wie dies im folgenden Schema 3.5 veranschaulicht wird.

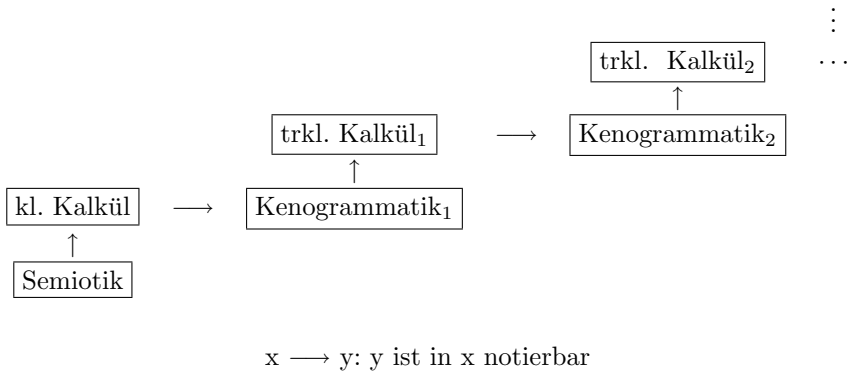


Abbildung 3.5: Formalisierungsstufen der Kenogrammatik

An dieser Stelle stellt sich die Frage, ob diese Folge einen unendlichen Regreß von Formalisierungsstufen impliziert, der finit niemals die anvisierte Kenogrammatik realisieren kann, oder ob für diese Folge ein endlicher Fixpunkt existiert, für den die angestrebte Loslösung von der klassischen Semiotik und Logik faktisch gegeben ist.
Die bisher unbewiesene These der Kenogrammatik lautet, daß ein solcher Fixpunkt existiert und daß die Kenogrammatik finit realisierbar ist. Eine Argumentation, die sich etwa auf v. Foersters [Foe76] Interpretation der Eigenwerttheorie oder auf die Konzeptionen der metazirkulären Interpreter und des Bootstrappingverfahrens der Informatik stützt [Abe87], kann dieser These zumindestens einige Evidenz verleihen. Die Argumentation beruht darauf, daß im Falle eines solchen 'Bootstrapprozesses' das ursprüngliche Ausgangsobjekt — hier die klassische Theorie — von den abgeleiteten Stufen nicht mehr direkt referiert wird. Dies bedeutet eine Loslösung vom 'Induktionsanfang', so daß die abgeleiteten Stufen in sich abgeschlossen sind und ihren 'Anfang' — der ja nun kein Anfang mehr ist — verabschiedet, vergessen, verdrängt haben

[14]So versucht etwa Heise [Hei91] die Morphogrammatik in der elementaren Mengenlehre „aufzuheben". Hierbei betrachtet er aber gerade nur MG_1, die allererste einer Folge von Formalisierungsstufen der von Günther anvisierten allgemeinen Morphogrammatik. Selbst eine korrekte mengentheoretische Einbettung von MG_1 hätte keine Aussagekraft bezüglich der allgemeinen MG.

([Hom90], [Foe76]). Folgen wir diesem Argument, so ergibt sich folgende Situation (Abb. 3.6):

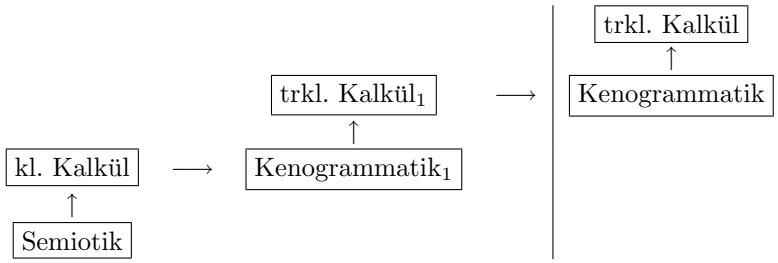


Abbildung 3.6: Fixpunkt der KG-Formalisierung

Der allgemeine transklassische Kalkül wird hier innerhalb der allgemeinen Kenogrammatik formuliert. Konstruktionshistorisch basiert die Kenogrammatik auf dem transklassischen Kalkül$_1$ (der wiederum auf die klassische Semiotik zurückgeht). Der senkrechte Strich markiert nun die Abschlußgrenze der Fixpunktbildung, so daß der Bereich rechts des Strichs nicht mehr durch die Semiotik fundiert wird, denn der transklassische Kalkül$_1$, der die Kenogrammatik fundiert, kann durch den allgemeinen transklassischen Kalkül ersetzt werden. In (10.4.3) wird diese Argumentation näher ausgeführt.
Das Münchhausen-Trilemma wird durch diese Argumentation dahingehend aufgelöst, daß zur Vermeidung rein 'kurzgeschlossener' Zirkularität, diese auf mehrere Hierarchiebenen simultan verteilt ist. Die Fixpunktbildung verhindert einen unendlichen Regreß, jedoch ohne eine willkürliche Setzung, allein aus den der Kenogrammatik immanenten Eigenschaften heraus. Die Kenogrammatik kann also losgelöst vom Semiotischen als Theorie innerhalb des allgemeinen transklassischen Kalkülkonzeptes formuliert werden. Wir gelangen so zu Schema 3.7.

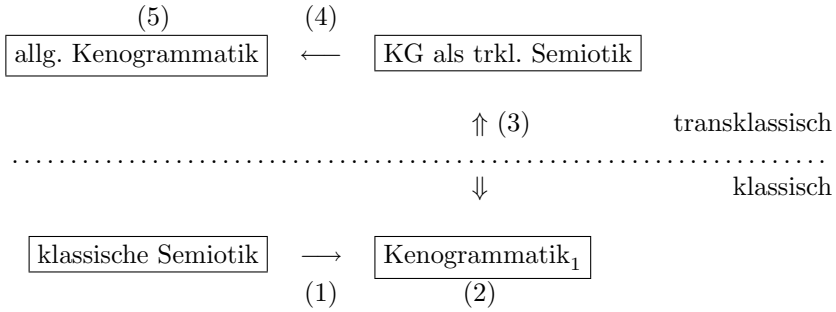


Abbildung 3.7: Kenogrammatik und Semiotik

Die Semiotik fundiert das klassische Kalkülkonzept (1). Im Sprachrahmen des Kalküls wird die Kenogrammatik als klassischer Kalkül KG$_1$ formuliert (2). Die in der KG$_1$ formalisierten Konzepte werden durch den proemiellen Umtausch (3) als transklassische Semiotik gesetzt. Der Rollentausch bringt KG$_1$ vom Status des *fundierten* Kalküls in die Position der *fundierenden* Semiotik. Diese so als *transklassische Semiotik* gesetzte

Kenogrammatik fundiert nun ihrerseits das Konzept des *transklassischen Kalküls* (4). Im erweiterten Sprachrahmen des transklassischen Kalküls wird nun die allgemeine Kenogrammatik als transklassischer Kalkül notiert (5). Dieses Schema veranschaulicht den von Günther anvisierten Konstruktionsprozeß eines transklassischen Formalismus, der eine Loslösung von der klassischen Semiotik erreicht.

3.3 Formale Grundlagen der Kenogrammatik

Kenogrammatische Eigenschaften werden zunächst in 3.3.1 nach einer von Schadach [Sch67a] vorgestellten Klassifikation von Morphismen zwischen endliche Mengen eingeführt. Diese Methode betont in erster Linie, daß kenogrammatische Strukturen gerade nicht als 'Zeichenketten', sondern als 'Relationsstrukturen' verstanden werden .
Als vereinfachte Notation dieser Relationen wird dann in 3.3.2 die aus der Literatur bekannteste (und historisch älteste) Darstellung kenogrammatischer Strukturen als *Kenogrammsequenzen* gewählt. Unter diesem Aspekt sind viele operationale Eigenschaften der Strukturen in einer 'zeichenketten-ähnlichen' Notation einfach zu veranschaulichen. Diese Darstellung birgt allerdings auch die Gefahr in sich, kenogrammatische Strukturen als 'Zeichenketten' mißzudeuten. Aus diesem Grund wird in dem hier gewählten Aufbau der Kenogrammatik besonders darauf geachtet, mögliche Mißverständnisse dieses Ansatzes auszuschließen. In 3.3.3 werden Äquivalenzklassen von Kenogrammsequenzen untersucht.
Als dritte Methode der Notierung kenogrammatischer Strukturen wird in 3.3.4 die ϵ/ν-Darstellung eingeführt, die die strukturellen Verhältnisse von Gleichheit und Verschiedenheit innerhalb der Strukturen explizit beschreibt.
Mittels des so aufgebauten Beschreibungsapparates werden dann in Abschnitt 3.3.5 elementare kenogrammatische Umformungs– und Verknüpfungsoperationen definiert, die auch für die Einführung der Kenoarithmetik (Kapitel 4) und der Morphogrammatik (Kapitel 5) benötigt werden.

Jeder mathematische Text bedient sich zur Definition mathematischer Objekte einer bestimmten Metasprache, für die einige allgemein akzeptierte Konventionen gelten. So wird eine rekursive Funktion im allgemeinen etwa wie folgt definiert:

Definition 3.1 (Fakultät) *Die Fakultät einer natürlichen Zahl n, $\texttt{fak}(n)$, ist gegeben als:*

$$\texttt{fak}(n) = \begin{cases} 1 & \text{falls } n = 0 \\ n * \texttt{fak}(n-1) & \text{sonst.} \end{cases}$$

In der für diese Arbeit gewählten Programmiersprache ML kann diese Funktion folgendermaßen implementiert werden:

```
fun fak(0) = 1
   |fak(n) = n*fak(n-1);
```

Abgesehen von den syntaktischen Unterschieden ist diese Implementierung offensichtlich äquivalent zur obigen Definition. Aufgrund dieser Äquivalenz wurde in einigen einfachen Definitionen eine Notierung in der Metasprache ML gewählt. Die Definition der Fakultät geschähe also folgendermaßen:

Definition 3.2 (Fakultät) *Die Fakultät einer natürlichen Zahl n, $\texttt{fak}(n)$, ist gegeben als:*

```
fun fak(0) = 1
   |fak(n) = n*fak(n-1);
```

Diese Abweichung von der Konvention dient der Straffung des Textes, da so Definition und Implementierung gemeinsam präsentiert werden können und erspart dem Leser die Mühe, den gleichen Sachverhalt in zwei verschiedenen Notationen nacheinander lesen zu müssen.

3.3.1 Kenogrammatische Klassifikation von Morphismen

Wertsequenzen wie etwa die Wahrheitswertverläufe der klassischen Aussagenlogik lassen sich als Abbildungen von Plätzen (innerhalb der Sequenzen) auf Werte verstehen. Der Wertverlauf der Konjunktion $\wedge$, 1222 läßt sich etwa als Morphismus $\mu_\wedge$ zwischen der Menge der Plätze, $A = \{a_1, a_2, a_3, a_4\}$ und der Menge der Werte, $B = \{1, 2\}$ notieren:

$\mu_\wedge : A$	$\longrightarrow$	B
$\mu_\wedge(a_1)$	$=$	1
$\mu_\wedge(a_2)$	$=$	2
$\mu_\wedge(a_3)$	$=$	2
$\mu_\wedge(a_4)$	$=$	2

$\mu_\wedge : A$	$\longrightarrow$	B
a_1	$\longrightarrow$	1
a_2	$\searrow$	
a_3	$\longrightarrow$	2.
a_4	$\nearrow$	

Günthers Abstraktion der Wahrheitswertfunktionen des Aussagenkalküls in Proto-Deutero- und Tritostrukturen wird nun für beliebige Abbildungen eingeführt[15].
Seien allgemein A und B endliche, nichtleere Mengen,

$$A = \{a_1, \ldots, a_n\} \text{ und } B = \{b_1, \ldots, b_m\}.$$

Bezeichne B^A die Menge aller Abbildungen μ von A nach B:

$$B^A = \{\mu | \mu : A \longrightarrow B\},$$

dann ist die Kardinalität von B^A gegeben durch:

$$|B^A| = |B|^{|A|} = m^n.$$

Definition 3.3 (Protoäquivalenz) *Zwei Abbildungen μ_1, μ_2 aus B^A sind protoäquivalent, $\mu_1 \overset{\text{p}}{\equiv} \mu_2$, genau dann wenn $|A/\text{Kern } \mu_1| = |A/\text{Kern } \mu_2|$.*

Wobei $|A/\text{Kern } \mu|$ die Kardinalität der Quotientenmenge $A/\text{Kern } \mu$ von A nach dem Kern von μ ist. Kern μ ist eine Äquivalenzrelation in A und damit Untermenge der Produktmenge $A \times A$. Für alle $a_i, a_j \in A$ gilt: $(a_i, a_j) \in \text{Kern } \mu$ gdw. $\mu(a_i) = \mu(a_j)$:

$$\text{Kern } \mu = \{a_i, a_j) \in A \times A | \mu(a_i) = \mu(a_j)\}$$

Die Äquivalenzklasse eines Elementes a_i von A relativ zu Kern μ wird mit $[a_i]\text{Kern } \mu$ bezeichnet.

$$[a_i]_{\text{Kern } \mu} = \{a \in A | \mu(a) = \mu(a_i)\}$$

[15] [Sch67a]

Beispiel: Betrachten wir wieder $\mu_\wedge$ aus dem ersten Beispiel, dann ist:

$$\begin{aligned}[a_1]_{\text{Kern } \mu_\wedge} &= \{a_1\}\\ [a_2]_{\text{Kern } \mu_\wedge} &= \{a_2, a_3, a_4\}\\ [a_3]_{\text{Kern } \mu_\wedge} &= \{a_2, a_3, a_4\}\\ [a_4]_{\text{Kern } \mu_\wedge} &= \{a_2, a_3, a_4\}\end{aligned}$$

Die Quotientenmenge $A/\text{Kern } \mu$ ist die Menge aller Äquivalenzklassen $[a_i]_{\text{Kern } \mu}$.

Beispiel: Im obigem Beispiel existieren nur zwei solcher Äquivalenzklassen:

$$\begin{aligned}A/\text{Kern } \mu_\wedge &= \{\{a_1\}, \{a_2, a_3, a_4\}\}\\ |A/\text{Kern } \mu_\wedge| &= 2\end{aligned}$$

$|A/\text{Kern } \mu|$ bezeichnet die Anzahl der Äquivalenzklassen in A relativ zu Kern μ. Zwei Morphismen sind somit genau dann protoäquivalent, wenn sie die gleiche Anzahl von Äquivalenzklassen $[a_i]_{\text{Kern } \mu}$ aufweisen. Die Protoäquivalenz ist eine Äquivalenzrelation und zerlegt deshalb A in paarweise disjunkte, nichtleere Mengen.

Beispiel: Sind zwei Morphismen μ_1, μ_2 durch die folgenden Diagramme bestimmt:

$\mu_1 : A$	$\longrightarrow$	B
a_1	$\searrow$	b_1
a_2	$\longrightarrow$	b_2
a_3	$\nearrow$	b_3
a_4	$\longrightarrow$	b_4

$\mu_2 : A$	$\longrightarrow$	B
a_1	$\longrightarrow$	b_1
a_2	$\nearrow$	b_2
a_3	$\longrightarrow$	b_3
a_4	$\nearrow$	b_4.

Dann ist:

$$\begin{aligned}[a_1]_{\text{Kern } \mu_1} &= \{a_1, a_2, a_3\}\\ [a_4]_{\text{Kern } \mu_1} &= \{a_4\}\end{aligned}$$

und

$$\begin{aligned}[a_1]_{\text{Kern } \mu_2} &= \{a_1, a_2\}\\ [a_3]_{\text{Kern } \mu_2} &= \{a_3, a_4\}.\end{aligned}$$

Also ist $A/\text{Kern } \mu_1 = \{\{a_1, a_2, a_3\}, \{a_4\}\}$ und $A/\text{Kern } \mu_2 = \{\{a_1, a_2\}, \{a_3, a_4\}\}$. Da nun $|A/\text{Kern } \mu_1| = |A/\text{Kern } \mu_2|$, gilt:$\mu_1 \overset{\text{p}}{\equiv} \mu_2$.

Definition 3.4 (Deuteroäquivalenz) *Zwei Morphismen $\mu_1, \mu_2 \in B^A$ heißen deuteroäquivalent, $\mu_1 \overset{\text{d}}{\equiv} \mu_2$, gdw. $A/\text{Kern } \mu_1 \cong A/\text{Kern } \mu_2$.*

Wobei der Isomorphismus $\cong$ definiert ist durch: $A/\text{Kern } \mu_1 \cong A/\text{Kern } \mu_2$ gdw. es eine Bijektion $\varphi : A/\text{Kern } \mu_1 \longrightarrow A/\text{Kern } \mu_2$ gibt, mit $|\varphi([a_i]_{\text{Kern } \mu_1})| = |([a_i]_{\text{Kern } \mu_2})|$ für alle $a_i \in A$. $[a_i]_{\text{Kern } \mu}$ ist hierbei die Äquivalenzklasse von a_i relativ zu Kern μ:

$$[a_i]_{\text{Kern } \mu} = \{a \in A | (a_i, a) \in \text{Kern } \mu\}.$$

Die Deuteroäquivalenzrelation ist ebenfalls reflexiv, symmetrisch und transitiv.

Beispiel: μ_1 und μ_2 sind gegeben als:

$\mu_1 : A$	$\longrightarrow$	B
a_1	$\searrow$	b_1
a_2	$\longrightarrow$	b_2
a_3	$\nearrow$	b_3
a_4	$\longrightarrow$	b_4

$\mu_2 : A$	$\longrightarrow$	B
a_1	$\longrightarrow$	b_1
a_2	$\searrow$	b_2
a_3	$\longrightarrow$	b_3
a_4	$\nearrow$	b_4.

Dann ist:

$$\begin{aligned} M_1 &= A/\text{Kern } \mu_1 = \{\{a_1, a_2, a_3\}, \{a_4\}\} \text{ und} \\ M_2 &= A/\text{Kern } \mu_2 = \{\{a_1\}, \{a_2, a_3, a_4\}\}. \end{aligned}$$

Wählen wir nun passend $\varphi : M_1 \longrightarrow M_2$ mit:

$$\begin{aligned} \varphi(a_1) &= a_4 \\ \varphi(a_2) &= a_2 \\ \varphi(a_3) &= a_3 \\ \varphi(a_4) &= a_1, \end{aligned}$$

so ist $\varphi(M_1) = \{\{a_4, a_2, a_3\}, \{a_1\}\} = M_2$. Deswegen ist auch $|\varphi(M_1)| = |M_2|$ und es gilt: $\mu_1 \overset{\text{d}}{\equiv} \mu_2$.

Definition 3.5 (Tritoäquivalenz) *Zwei Morphismen $\mu_1, \mu_2 \in B^A$ heißen tritoäquivalent, $\mu_1 \overset{\text{t}}{\equiv} \mu_2$, gdw. $A/\text{Kern } \mu_1 = A/\text{Kern } \mu_2$.*

Für alle $a_i \in A$ gilt also: $[a_i]_{\text{Kern } \mu_1} = [a_i]_{\text{Kern } \mu_2}$. Die Tritoäquivalenzrelation ist ebenfalls reflexiv, symmetrisch und transitiv.

Beispiel: μ_1 und μ_2 sind gegeben durch:

$\mu_1 : A$	$\longrightarrow$	B
a_1	$\searrow$	b_1
a_2	$\searrow$	b_2
a_3	$\longrightarrow$	b_3
a_4	$\longrightarrow$	b_4

$\mu_2 : A$	$\longrightarrow$	B
a_1	$\longrightarrow$	b_1
a_2	$\longrightarrow$	b_2
a_3	$\nearrow$	b_3
a_4	$\nearrow$	b_4.

Dann ist:

$$\begin{aligned} M_1 &= A/\text{Kern } \mu_1 = \{\{a_1\}, \{a_2, a_3\}, \{a_4\}\} \text{ und} \\ M_2 &= A/\text{Kern } \mu_2 = \{\{a_1\}, \{a_2, a_3\}, \{a_4\}\}. \end{aligned}$$

Da $M_1 = M_2$, gilt $\mu_1 \overset{\text{t}}{\equiv} \mu_2$.

Satz 3.1 *Seien μ_1 und μ_2 Abbildungen aus B^A. Dann gilt:*

$$\mu_1 \overset{\text{t}}{\equiv} \mu_2 \quad \Longrightarrow \quad \mu_1 \overset{\text{d}}{\equiv} \mu_2 \quad \Longrightarrow \quad \mu_1 \overset{\text{p}}{\equiv} \mu_2$$

Beweis: Der Satz ergibt sich anschaulich aus den Definitionen für die Trito-, Deutero- und Protoäquivalenz. ∎

3.3.1.1 Vollständigkeit der Klassifikation

Die von Günther nur inhaltlich, nicht aber formal begründete Klassifikation in Proto-, Deutero- und Tritostruktur wirft zwei Fragen auf:

1. Welches ist das formale Kriterium der kenogrammatischen Klassifikation von Morphismen?
2. Ist die Klassifikation vollständig oder lassen sich weitere Äquivalenzrelationen finden, die dem Klassifikationskriterium genügen?

Diese beiden Fragen sollen mit dem Ausführungen dieses Abschnittes beantwortet werden.
Die Proto-, Deutero- und Tritoäquivalenz von Abbildungen $\mu_i : A \longrightarrow B$ greifen in ihren Definitionen jeweils auf die Quotientenmenge $A/\text{Kern}\ \mu_i$ zurück. Diese Menge erfaßt die Zerlegung von A in disjunkte, nichtleere Mengen von Äquivalenzklassen, nicht jedoch die jeweiligen Bildwerte $b_j \in B$ dieser Äquivalenzklassen. Die drei kenogrammatischen Äquivalenzen abstrahieren also vollkommen von der Belegungsmenge B. Der durch diese Relationen definierte Bereich der Kenogrammatik unterscheidet Abbildungen allein nach ihrer Quotientenstruktur von Gleichheit und Verschiedenheit, die unabhängig von der tatsächlichen Belegung ist und durch $A/\text{Kern}\ \mu_i$ angegeben wird.

Definition 3.6 (Kenogrammatisches Äquivalenzkriterium) *Zur Definition kenogrammatischer Äquivalenzrelationen zwischen zwei Abbildungen $\mu_1, \mu_2 \in B^A$ dürfen nur Eigenschaften der Quotientenmengen $A/\text{Kern}\ \mu_1$ und $A/\text{Kern}\ \mu_2$ benutzt werden.*

Anhand des so eingeführten Kriteriums läßt sich unter der Annahme nicht linkstotaler Abbildungen noch eine weitere Abstraktionsstufe einführen, die nur noch die Kardinalität der Urbildbereiche als Äquivalenzbedingung verwendet. Die Quotientenmenge $A/\text{Kern}\ \mu_i$ enthält $|A/\text{Kern}\ \mu_i|$ Äquivalenzklassen:

$$A/\text{Kern}\ \mu_i = \{\ddot{\text{A}}_1, \ldots, \ddot{\text{A}}_{|A/\text{Kern}\ \mu_i|}\}$$

Definition 3.7 *Die Größe einer Abbildung $\mu : A \longrightarrow B$ ist gegeben durch:*

$$\texttt{size}(\mu) = \sum_{k=1}^{|A/\text{Kern}\ \mu|} |\ddot{A}_k| \quad , \ \ddot{A}_k \in A/\text{Kern}\ \mu.$$

Hiermit läßt sich definieren:

Definition 3.8 (Cardäquivalenz) *Zwei Abbildungen $\mu_1, \mu_2 \in B^A$ heißen cardäquivalent, $\mu_1 \equiv_{\texttt{card}} \mu_2$, gdw. $\texttt{size}(\mu_1) = \texttt{size}(\mu_2)$.*

Offensichtlich erfüllt diese Relation das kenogrammatische Äquivalenzkriterium, nur Eigenschaften von $A/\text{Kern}\ \mu_i$ zu betrachten, und kann also als kenogrammatische Äquivalenzrelation angesehen werden. Gleichzeitig entspricht diese Relation aber auch der Gleichheitsrelation der Theorie der natürlichen Zahlen, die Objekte rein quantitativ anhand ihrer Kardinalität vergleicht. Die klassische Arithmetik der natürlichen Zahlen erweist sich somit als reduziertester Spezialfall kenogrammatischer Strukturbeschreibung. Die von Günther anvisierte Kenogrammatik thematisiert die qualitative

Gestalthaftigkeit relationaler Objekte, die durch die Quotientenstruktur formal erfaßt wird, wohingegen die Cardäquivalenz von dieser Gestalthaftigkeit wieder vollkommen abstrahiert und in den Bereich der klassischen Arithmetik führt. Aus diesem Grund wird die Cardäquivalenz und die auf ihr beruhende Arithmetik nicht zum Bereich der intendierten Kenogrammatik gezählt, obwohl sie systematisch durchaus in den Rahmen der Kenogrammatik fällt.
Als Kritik und Erweiterung der Ad-hoc Klassifikation Günthers entwickelte Schadach [Sch67b] ein System von insgesamt acht Äquvalenzrelationen zur Klassifikation von $B^A = \{\mu | \mu : A \longrightarrow B\}$. Diese acht Relationen werden aus fünf Äquivalenzbedingungen $(\alpha, \ldots, \epsilon)$ und deren möglichen unabhängigen Konjunktionen gebildet:

$$(\alpha) \quad \mu_1 \equiv_\alpha \mu_2 \quad \Longleftrightarrow \quad A/\text{Kern } \mu_1 = A/\text{Kern } \mu_2.$$

$$(\beta) \quad \mu_1 \equiv_\beta \mu_2 \quad \Longleftrightarrow \quad A/\text{Kern } \mu_1 \cong A/\text{Kern } \mu_2.$$

$$(\gamma) \quad \mu_1 \equiv_\gamma \mu_2 \quad \Longleftrightarrow \quad |A/\text{Kern } \mu_1| = |A/\text{Kern } \mu_2|.$$

$$(\delta) \quad \mu_1 \equiv_\delta \mu_2 \quad \Longleftrightarrow \quad |\mu_1^{-1}(b_j)| = |\mu_2^{-1}(b_j)| \quad \text{für alle } b_j \in B.$$

$$(\epsilon) \quad \mu_1 \equiv_\epsilon \mu_2 \quad \Longleftrightarrow \quad \mu_1(A) = \mu_2(A).$$

Zusammen mit der Identitätsrelation (i):

$$(i) \quad \mu_1 \equiv_i \mu_2 \quad \Longleftrightarrow \quad \mu_1(a_j) = \mu_2(a_j) \quad \text{für alle } a_j \in A$$

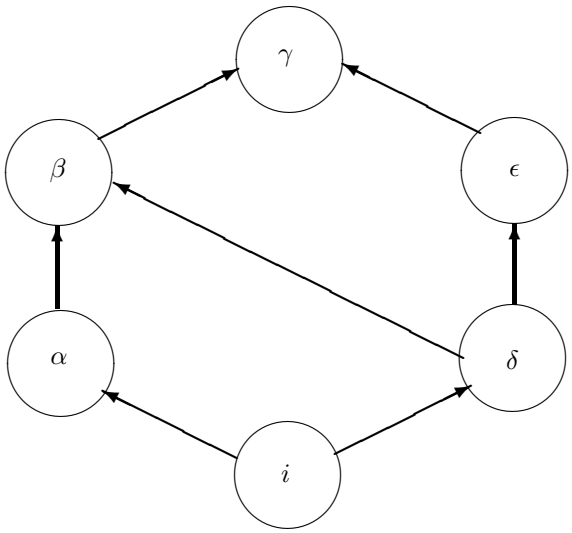


Abbildung 3.8: Das Verhältnis der sechs Charakteristika nach Schadach. Die Pfeile $\longrightarrow$ symbolisieren die Implizierungsrelation.

bilden diese fünf Äquivalenzkriterien die in Abbildung 3.8 dargestellte Struktur. Die Pfeile $\longrightarrow$ repräsentieren in diesem Schema die Implikationsbeziehung zwischen den

einzelnen Relationen. Aufgrund dieser Abhängigkeiten lassen sich nur bestimmte unabhängige Konjunktionen dieser Kriterien bilden. Da beispielsweise $(i) \longrightarrow (\alpha)$, ist eine Konjunktion $(i \wedge \alpha)$ nicht unabhängig, da sie einfach mit (i) koinzidiert. Die möglichen unabhängigen Konjunktionen sind nach Schadach: $(\alpha \wedge \delta), (\alpha \wedge \epsilon)$ und $(\beta \wedge \epsilon)$. Die fünf Kriterien (α), (β), (γ), (δ), (ϵ), die unabhängigen Konjunktionen $(\alpha \wedge \delta)$, $(\alpha \wedge \epsilon)$, $(\beta \wedge \epsilon)$, die Identität (i) sowie die Cardäquivalenz (c):

$$(c) \quad \mu_1 \equiv_c \mu_2 \quad \Longleftrightarrow \quad \mathtt{size}(\mu_1) = \mathtt{size}(\mu_2)$$

bilden zusammen eine Struktur von zehn Äquivalenzrelationen über B^A, deren wechselseitigen Implikationen in Abbildung 3.9 dargestellt sind.
Die Bedingung (α) repräsentiert die Tritoäquivalenz, (β) die Deutero- und (γ) die Protoäquivalenz. Offensichtlich genügen $(\alpha), (\beta), (\gamma)$ dem kenogrammatischen Äquivalenzkriterium, allein Eigenschaften der Quotientenmengen $A/\text{Kern}\ \mu_i$ zu vergleichen. Bedingung (δ) überprüft, ob jeder Bildwert $b_j \in B$ von der gleichen Anzahl von Plätzen aus A referiert wird. Bedingung (ϵ) vergleicht, ob zwei Abbildungen den gleichen Bildbereich besitzen. Die Bedingungen (δ) und (ϵ) beziehen sich nicht, wie vom kenogrammatischen Äquivalenzkriterium gefordert, auf $A/\text{Kern}\ \mu_i$ sondern auf Eigenschaften, die unabhängig von der 'Gestalt' der Abbildungen sind und eine Wohlunterschiedenheit der Bildmenge B im Sinne der atomistischen Identitätsbeziehung (i) der klassischen Semiotik implizieren. Aus diesem Grund müssen fünf der acht von Schadach eingeführten Relationen, $(\alpha \wedge \delta), (\alpha \wedge \epsilon), (\delta), (\beta \wedge \epsilon), (\epsilon)$, als nicht genuin kenogrammatisch abgewiesen werden. Gültig bleiben allein $(\alpha), (\beta), (\gamma)$, also das System der Proto-, Deutero- und Tritoäquivalenz.
Schadach kann die Vollständigkeit seiner Klassifikation beweisen, es existieren somit keine weiteren unabhängigen Äquivalenzrelationen über B^A. Da nur $(\alpha), (\beta), (\gamma)$ (und (c)) das kenogrammatische Äquivalenzkriterium erfüllen, sind innerhalb der Kenogrammatik keine weiteren unabhängigen Äquivalenzrelationen erzeugbar. Die Klassifikation Günthers von Proto-, Deutero- und Tritoßtruktur ist in diesem Sinn vollständig.

3.3.2 Kenogrammkomplexionen

Mit den im vorhergehenden Abschnitt eingeführten Äquivalenzrelationen lassen sich nun beliebige, aus semiotischen Zeichenprozessen hervorgegangene Konstrukte vergleichen.

> **Beispiel:** Die Zeichenketten (*Strings*) $z_1 =$ "*bbcd*" und $z_2 =$ "*kkln*" sind offensichtlich semiotisch ungleich: $z_1 \not\equiv_{sem} z_2$. Werden diese Strings nun als Abbildungen von einer Menge von Plätzen (der Stringpositionen) A in die Menge der verwendeten Zeichen B interpretiert, dann ergibt sich für z_1 die Abbildung μ_1 und für z_2 entsprechend μ_2:
>
> | $\mu_1 : A$ | $\longrightarrow$ | B_1 |
> |---|---|---|
> | a_1 | $\searrow$ | 'a' |
> | a_2 | $\longrightarrow$ | 'b' |
> | a_3 | $\longrightarrow$ | 'c' |
> | a_4 | $\longrightarrow$ | 'd' |
>
> | $\mu_2 : A$ | $\longrightarrow$ | B_2 |
> |---|---|---|
> | a_1 | $\longrightarrow$ | 'k' |
> | a_2 | $\nearrow$ | 'l' |
> | a_3 | $\nearrow$ | 'm' |
> | a_4 | $\longrightarrow$ | 'n'. |
>
> Es ist hierbei:
>
> $$\begin{aligned} A/\text{Kern}\ \mu_1 &= \{\{a_1, a_2\}, \{a_3\}, \{a_4\}\} = \\ A/\text{Kern}\ \mu_2 &= \{\{a_1, a_2\}, \{a_3\}, \{a_4\}\}, \end{aligned}$$

also gilt $\mu_1 \overset{t}{\equiv} \mu_2$.

Für die semiotische Gleichheitsrelation ist relevant, ob jeder der Plätze von z_1 durch dasselbe Zeichen wie der entsprechende Platz von z_2 belegt ist, ob also $\mu_1(a_i) = \mu_2(a_i)$ für alle $a_i \in A$. Die Tritoäquivalenz (und ebenso Deutero- und Protoäquivalenz) abstrahiert vollkommen von der semiotischen Gleichheit, das heißt von der tatsächlichen Belegung eines Platzes und untersucht lediglich die Struktur der Platzbelegungen untereinander. Da die Strukturen von Gleichheit und Verschiedenheit von z_1 und z_2 gleich ist, sind sie trito-, deutero- und protoäquivalent.

3.3.2.1 Kenogramme

Die für die Trito-, Deutero- und Protoäquivalenz relevante Belegungsstruktur wird für beliebige Zeichengebilde $\mu : A \longrightarrow B$ durch die Quotientenmenge $A/\text{Kern}\ \mu$ angegeben. Um nun bei Berechnungen nicht auf die unübersichtliche Notierung von $A/\text{Kern}\ \mu$ angewiesen zu sein, werden als Notationsvereinfachung Normalformen als Standardrepräsentanten für Trito-, Deutero- und Protoäquivalenzklassen definiert.

Definition 3.9 (Kenogrammsymbole) K *ist eine abzählbar unendliche Menge von Standardsymbolen.*

Diese so definierte Menge von Standardsymbolen erlaubt eine von allen möglichen Belegungsmengen B unabhängige Notation. Aus Darstellungsgründen wird zusätzlich vereinbart, daß **K** die Menge {○, △, □, ⋆, ●, ▽, ■, ∗, ◇} enthält.

Definition 3.10 (Lexikographische Ordnung) *Auf* **K** *existiert eine (totale) lexikographische Ordnung* $<$. *Es wird vereinbart, daß* ○< △< □< ⋆< ●< ▽< ■< ∗< ◇.

Für die Implementierung in ML wurde die zu **K** isomorphe Menge der Natürlichen Zahlen zur Darstellung der Standardsysmbole gewählt:

```
type keno = int;
```

Die so vereinbarten Symbole sind *als Zeichen für* Kenogramme und nicht *als* Kenogramme zu verstehen. Ausdrücklich wird noch einmal darauf hingewiesen, daß **K** ausschließlich zur Vereinheitlichung und Vereinfachung der Notation eingeführt wurde, daß also kein a priori gegebenes 'Kenogrammalphabet' existiert und daß ebenfalls keines der Kenogrammsymbole als isolierbares und interpretierbares semiotisches Zeichen zu verstehen ist.

3.3.2.2 Die Tritonormalform TNF

Mit Hilfe der so vereinbarten Notation lassen sich jetzt Normalformen als Standardrepräsentanten der Trito-, Deutero- und Protoäquivalenzklassen definieren.

Definition 3.11 (Tritonormalform) *Die Tritonormalform eines Morphismus* $\mu : A \longrightarrow B$ *ist die lexikographisch erste zu* μ *tritoäquivalente Abbildung* $TNF(\mu) : A \longrightarrow$ **K**.

Beispiel: Sei $\mu_0 : \{a_1, a_2, a_3, a_4\} \longrightarrow \{1, 2\}$ gegeben durch:

$\mu_0 : A$	$\longrightarrow$	B
a_1	$\searrow$	
a_2	$\longrightarrow$	1
a_3	$\longrightarrow$	2
a_4	$\nearrow$	

Dann sind $\mu_1 \overset{t}{\equiv} \mu_2 \overset{t}{\equiv} \mu_3 \overset{t}{\equiv} \mu_0$:

$\mu_1 : A$	$\longrightarrow$	$\mathbf{K}$
a_1	$\longrightarrow$	○
a_2	$\nearrow$	△
a_3	$\nearrow\nearrow$	□
a_4		⋆

$\mu_2 : A$	$\longrightarrow$	$\mathbf{K}$
a_1	$\longrightarrow$	○
a_2	$\nearrow$	△
a_3	$\longrightarrow$	□
a_4	$\nearrow$	⋆

$\mu_3 : A$	$\longrightarrow$	$\mathbf{K}$
a_1	$\searrow$	○
a_2	$\longrightarrow$	△
a_3	$\searrow$	□
a_4	$\longrightarrow$	⋆.

Da μ_1 die nach der definierten lexikographischen Ordnung von $\mathbf{K}$ erste dieser Abbildungen ist, gilt: $TNF(\mu_0) = \mu_1$.

3.3.2.3 Kenogrammsequenzen

Als Verallgemeinerung aller möglichen diskreten Strukturen von Kenogrammkomplexionen wie Bäumen, Ringen, n-dimensionale Matrizen oder Graphen werden im Folgenden *lineare Listen* benutzt, wie sie in der Informatik zur Repräsentation beliebiger Datenstrukturen verwandt werden. Da stets ein-eindeutige Abbildungen zwischen linearen Listen und diskreten Strukturen existieren, wird hierdurch die Allgemeinheit nicht eingeschränkt.

Definition 3.12 (Kenogrammsequenz) *Die Notierung einer Abbildung* $TNF(\mu) : A \longrightarrow \mathbf{K}$ *als lineare Liste heißt Kenogrammsequenz.*

Beispiel: $\mu_0, \mu_1, \mu_2, \mu_3$ aus dem obigen Beispiel werden als lineare Listen notiert:

μ_0 = [1,1,2,2]	keine Kseq., da $\mu_0 : A \longrightarrow \{1, 2\}$
μ_1 = [○,○,△,△]	Kseq., da $\mu_1 = TNF(\mu_0) : A \longrightarrow \mathbf{K}$
μ_2 = [○,○,□,□]	keine Kseq., da nicht in TNF
μ_3 = [△,△,⋆,⋆]	keine Kseq., da nicht in TNF

In der Implementierung von Kenogrammsequenzen wird die TNF-Bedingung aus Vereinfachungsgründen nicht explizit berücksichtigt, der Typ `kseq` wird also einfach als:

```
type kseq = keno list;
```

vereinbart. So läßt sich beispielsweise `a` definieren als[16]:

```
- val a = [2,2,1,1] : kseq;
> val a = [2,2,1,1] : kseq
```

Die im strengen Sinne korrekte TNF-sequenz muß explizit berechnet werden mit der Funktion `tnf`[17]:

[16] `-` ist der Eingabeprompt des ML-Systems. Das Ergebnis einer Berechnung erscheint nach dem Ausgabeprompt `>`. Hinter dem Doppelpunkt `:` wird der durch den Typecheck-mechanismus ermittelte Typ des Ergebnisausdruckes angeben.

[17] Allgemein verwendete Funktionen, wie etwa `pos` und `nfirst` sind aus Gründen der Übersichtlichkeit in A.1.1 definiert.

```
fun tnf ks =
  let
    fun firstocc item list =
      let
        fun place1 item [] n = raise Place
           |place1 item (x::xs) n = if item=x then n
                                    else place1 item xs n+1;
      in
        place1 item list 1
      end;

    fun tnf1 [] res n k = res
       |tnf1 (hd::tl) res 1 k = tnf1 tl [1] 2 2
       |tnf1 (hd::tl) res n k =
          if member (pos n ks) (nfirst (n-1) ks)
          then tnf1 tl
                   (res@[pos (firstocc (pos n ks) ks) res])(n+1) k
          else tnf1 tl
                   (res@[k]) (n+1) (k+1);
  in
    tnf1 ks [] 1 1
  end;
```

Beispiel:

```
- tnf a;
> [1,1,2,2] : kseq
```

3.3.2.4 Die Deuteronormalform DNF

Mit der Einführung der TNF und der Kenogrammsequenz ist eine übersichtliche und eindeutige Notation für kenogrammatische Strukturen entwickelt. Deutero- und Protonormalformen lassen sich jetzt einfach definieren.

Definition 3.13 (Deuteronormalform) *Die Deuteronormalform einer Abbildung $\mu : A \longrightarrow B$ ist die lexikographisch erste zu μ deuteroäquivalente Abbildung, $DNF(\mu) : A \longrightarrow \mathbf{K}$*

Implementiert wird die DNF durch die ML-Funktion `dnf`:

```
fun dnf ks =
  let
    fun count x []= 0
       |count x (y::ys)= (if x=y then 1 else 0)+count x ys;
  in
    flat (map (fn k=> nlistof (count k (tnf ks)) k)
              (rd (tnf ks)))
  end;
```

Beispiel:

```
- dnf [7,3,5,2,7,3];
> [1,1,2,2,3,4] : kseq
- dnf [1,2,1,2,4,5];
> [1,1,2,2,3,4] : kseq
```

3.3.2.5 Die Protonormalform PNF

Definition 3.14 (Protonormalform) *Die Protonormalform einer Abbildung* $\mu : A \longrightarrow B$ *ist die lexikographisch erste zu* μ *protoäquivalente Abbildung,* $PNF(\mu) : A \longrightarrow \mathbf{K}$

Implementiert wird die Protonormalform durch die Funktion `pnf` :

```
fun pnf ks = (nlistof (length ks - length(rd ks)) 1)@tnf(rd ks);
```

Beispiel:

```
- pnf [1,2,3,2,1,3];
> [1,1,1,1,2,3] : kseq
- pnf [4,5,5];
> [1,1,2] : kseq
```

3.3.3 Äquivalenzklassen

Zwei Kenogrammkomplexionen a, b sind nach den Definitionen von TNF, DNF, PNF äquivalent, wenn sie die gleiche Normalform aufweisen. Diese Äquivalenzen lassen sich somit implementieren als:

```
fun teq a b = (tnf a = tnf b);

fun deq a b = (dnf a = dnf b);

fun peq a b = (pnf a = pnf b);
```

Die Normalformen TNF, DNF, PNF dienen als Standardrepräsentanten der jeweiligen Äquivalenzklassen von Kenogrammkomplexionen. Für Kenogrammsequenzen der Länge n bestimmt sich nach Schadach [Sch67a] die Anzahl der Äquivalenzklassen, bzw. Normalformen wie folgt.
Die Anzahl der PNF's ist

$$Pcard(n) = n$$

und wird mittels der ML-Funktion `Pcard` implementiert:
`fun Pcard n = n;`
Die Anzahl der DNF's beträgt:

$$Dcard(n) = \sum_{k=1}^{n} P(n,k).$$

wobei $P(n,k)$ die Anzahl der möglichen Partitionen ist.

```
fun sum from to f=
        if (from > to) then 0
        else (f from) + sum  ( from + 1) to f;

fun P (n,1) = 1
   |P (n,k) =
```

```
     if k>n then 0
     else if k=n then 1
     else P(n-1,k-1) + P(n-k,k);

fun Dcard n = sum 1 n (fn k => P(n,k));
```

Die Anzahl der TNF's ergibt sich als:

$$Tcard(n) = \sum_{k=1}^{n} S(n,k)$$

wobei $S(n,k)$ die Stirlingzahl der zweiten Art ist [And65].

```
fun S (n,1) = 1
   |S (n,k) =
     if k>n then 0
     else if k=n then 1
     else S(n-1,k-1) + k*S(n-1,k);

fun Tcard n = sum 1 n (fn k => S(n,k));
```

Die Menge der `Pcard n` PNF's der Länge n wird berechnet durch:

```
fun Pcontexture n =
   map (fn k => (nlistof (n-k) 1)@(nlist k))
       (nlist n);
```

Beispiel:

```
- Pcontexture 4;
> [[1,1,1,1],[1,1,1,2],[1,1,2,3],[1,2,3,4]] : kseq list
```

Die Menge der `Dcard n` DNF's der Länge n wird durch die Funktion `Dcontexture n` berechnet:

```
fun allperms []=[]
   |allperms [x]=[[x]]
   |allperms [x,y]=[[x,y],[y,x]]
   |allperms l=
     let
      fun combine a l=
         map (fn x => a::x) l;
      fun remov x []=[]
         |remov x (y::ys)= if (x=y) then ys
                           else y::remov x ys;
     in
      flat ( map (fn a => combine a (allperms (remov a l)))
                 l)
     end;

fun combine a l=
         map (fn x => a::x) l;
```

```
fun allsums n 1=[[n]]
   |allsums n k=
     if (n=k) then [nlistof n 1]
     else
       flat(map (fn e => combine e (allsums (n-e) (k-1)))
                (nlist (n-k+1)));

fun allpartitions n k=
  let
    fun Exists f [] = false
       |Exists f (hd::tl)=
          if (f hd) then true
          else Exists f tl;
    fun remdups [] = []
       |remdups (hd::tl)=
          if Exists (fn x => (member x tl)) (allperms hd)
            then remdups tl
            else hd::(remdups tl);
  in
   remdups (allsums n k)
  end;

fun PDconcrete ks =
   map (fn p => flat (map (fn k => nlistof (pos k p) k)
                          (nlist (length (rd ks)))))
       (allpartitions (length ks) (length (rd ks)));

fun Dcontexture n =
   flat(map PDconcrete (Pcontexture n));
```

Beispiel:

```
- Dcontexture 4;
> [[1,1,1,1],[1,1,1,2],[1,1,2,2],
   [1,1,2,3],[1,2,3,4]] : kseq list
```

Die Menge der `Tcard n` TNF's der Länge n wird durch die Funktion `Tcontexture n` berechnet:

```
fun DTconcrete ks =
   rd(map (fn i => tnf i)
          (allperms ks));

fun Tcontexture n=
   flat(map DTconcrete (Dcontexture n));
```

Beispiel:

```
- Tcontexture 4;
> [[1,1,1,1],[1,1,1,2],[1,1,2,1],[1,2,1,1],[1,2,2,2],
   [1,1,2,2],[1,2,1,2],[1,2,2,1],[1,1,2,3],[1,2,1,3],
   [1,2,3,1],[1,2,2,3],[1,2,3,2],[1,2,3,3],[1,2,3,4]] : kseq list
```

Die drei Beispielberechnungen lassen sich zu folgender Tabelle 3.10 zusammenstellen, die die Kenogrammsequenzen der Länge 4 anhand der Proto-, Deutero- und Tritostruktur klassifiziert.

3.3.4 Die ϵ/ν-Darstellung

Kenogrammsequenzen wurden als Notationsvereinfachung für die Quotientenmenge $A/\text{Kern}\ \mu$ einer Abbildung $\mu : A \longrightarrow B$ eingeführt. Kenogrammsequenzen übersetzen die Strukturbeschreibung der Quotientenmenge in eine geordnete Sequenz von 'gefärbten' Leerstellen, den Kenogrammen. Diese Übersetzung ist ein-eindeutig und funktional vollständig, sie betont allerdings den Aspekt der *isolierten* Leerstellen, die mit semiotischen Zeichen belegt werden können (worauf ja gerade die Gefahr der Verwechselung mit semiotischen Zeichenketten beruht).
Als eine andere Veranschaulichungsweise der Quotientenstruktur läßt sich die ϵ/ν-*Darstellung* verwenden. Die ϵ/ν-Schreibweise betont stärker den Aspekt der Struktur von Gleichheit und Verschiedenheit *zwischen* den Leerstellen[18], der *Tritostruktur*[19]. Auch sie ist isomorph zur Quotientenmenge und damit auch zu der entsprechenden Kenogrammsequenz.

Definition 3.15 (ϵ/ν-Tripel) *Die Funktion $\delta((i,j),z)$, mit $z = [b_1,\ldots,b_n]$, $b_k \in B$, ordne jedem Paar von Positionen (i,j) das Tripel (i,j,ϵ) zu falls $b_i = b_j$, sonst (i,j,ν).*

Implementiert wird δ durch die folgende ML-Funktion:

```
datatype EN =E|N;

fun delta (i,j) z=
   if (pos i z) = (pos j z)
     then (i,j,E)
     else (i,j,N);
```

um alle möglichen Tripel zu erzeugen, muß jede Position j mit ihren $j-1$ Vorgängerpositionen verglichen werden. Es existieren also:

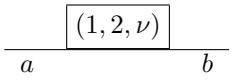
$$\sum_{j=1}^{n}(j-1) = \frac{n(n-1)}{2} \tag{3.3}$$

solcher Tripel.

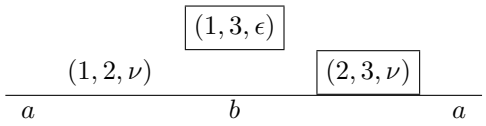
> **Beispiel:** Sei $z_1 = [a,b,a,c]$, die Länge der Sequenz ist $n = 4$. Nach Gleichung (3.3) wird die Tritostruktur von z_1 durch $\frac{4(4-1)}{2} = 6$ ϵ/ν-Tripel beschrieben:
>
> $$(1,2,\nu),(1,3,\epsilon),(2,3,\nu),(1,4,\nu),(2,4,\nu),(3,4,\nu)$$
>
> Zur graphischen Veranschaulichung wird die Konstruktion dieser Vergleichsstruktur hier schrittweise nachvollzogen. Für die erste Stelle, $j = 1$ und $b_1 = a$ existieren keine Vorgänger, es können also auch keine ϵ/ν-Tripel gebildet werden:

[18] Die Bezeichnung ϵ/ν leitet sich aus der unterscheidung ϵqual/νon equal ab.
[19] [Kro86], [Nie88], [Hou88], [Kae74].

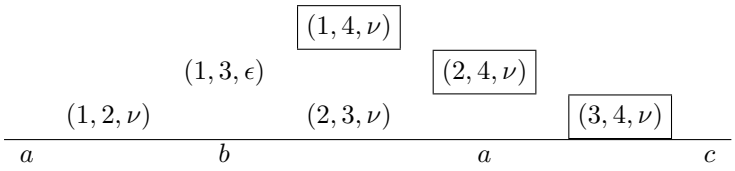
Für die zweite Position, $j = 2, b_2 = b$ kann nur ein Vergleich durchgeführt werden:



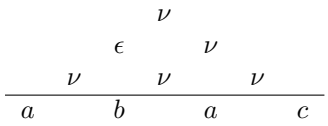
Die dritte Position, $j = 3, b_3 = a$ muß mit den ersten beiden verglichen werden:



Die vierte Position, $j = 4, b_4 = c$ muß mit ihren drei Vorgängern verglichen werden:



Offensichtlich sind die Positionen der Tripel in dieser Darstellung eindeutig bestimmt, so daß auf die Positionsangabe (i, j) verzichtet werden kann:



Die Struktur dieser ϵ/ν-Tripel einer n-stelligen Sequenz kann somit als Liste von n Listen mit jeweils $(j-1)$ Tripeln dargestellt werden:

$$[[], [(1, 2, \epsilon/\nu)], [(1, 3, \epsilon/\nu), (2, 3, \epsilon/\nu)], \ldots, [(1, n, \epsilon/\nu), \ldots, (n-1, n, \epsilon/\nu)]]$$

Definition 3.16 (ϵ/ν-Struktur) *Die ϵ/ν-Struktur einer n-stelligen Sequenz $z = [b_1, \ldots, b_j, \ldots, b_n]$ ist die Liste der n Listen von jeweils $(j-1)$ ϵ/ν-Tripel von z:*

$$[[], [(1, 2, \epsilon/\nu)], \ldots, [(1, j, \epsilon/\nu), \ldots, (j-1, j, \epsilon/\nu)], \ldots, [(1, n, \epsilon/\nu), \ldots, (n-1, n, \epsilon/\nu)]]$$

Die ϵ/ν-Struktur einer Sequenz wird durch die ML-Funktion `ENstructure` berechnet:

```
type enstruc = (int*int*EN) list list;

(* pairstructure n erzeugt die Struktur der m"oglichen Paare
   f"ur eine Sequenz der L"ange n *)
fun pairstructure n =
```

```
   map (fn j => map (fn i => (i,j))
                    (fromto 1 (j-1)))
       (fromto 1 n);

fun ENstructure z =
   map (fn trl => map (fn pair => delta (pair,z))
                      trl)
       (pairstructure (length z));
```

Beispiel:

```
- ENstructure ["a","b","a","d"];
> [[],
   [(1,2,N)],
   [(1,3,E),(2,3,N)],
   [(1,4,N),(2,4,N),(3,4,N)]] : enstruc
```

Da die ϵ/ν-Struktur die Tritostruktur einer Sequenz repräsentiert, kann die Tritoäquivalenz zweier Sequenzen a, b auch mittels der ϵ/ν-Struktur definiert werden:

```
fun teq a b = (ENstructure a) = (ENstructure b);
```

Die ϵ/ν-Struktur läßt sich eineindeutig auf die entsprechende Kenogrammsequenz abbilden:

```
exception Entoks;
fun ENtoKS enstruc =
  let
   fun entoks1 [] ks = ks
      |entoks1 ((f,s,en)::tl) ks =
        let
          val fir = pos f ks;
          val sec = if (length ks< s) then [] else pos s ks;
        in
         (if (en=E andalso sec=[])
          then entoks1 tl (ks@[fir])
          else if (en=E andalso member (hd fir) sec)
            then entoks1 tl (replace sec ks fir)
          else if (en=E andalso not(member (hd fir) sec))
            then raise Entoks
          else if (en=N andalso sec=[])
            then entoks1 tl (ks@[remove (hd fir)
                         (nlist ((kmax ks)+1:int))])
          else if (en=N andalso fir=sec)
            then raise Entoks
          else if (en=N andalso member (hd fir) sec)
            then entoks1 tl (replace sec ks
                                     (remove (hd fir) sec))
          else entoks1 tl ks)
        end;
  in
    (flat (entoks1 (flat enstruc) [[1]]))
  end;
```

Beispiel:

```
- ENtoKS [[],[(1,2,N)],[(1,3,E),(2,3,N)],
          [(1,4,N),(2,4,N),(3,4,N)]];
> [1,2,1,3] : kseq
```

3.3.5 Kenogrammatische Operationen

Im Vorhergehenden wurden verschiedene Darstellungsmöglichkeiten kenogrammatischer Strukturen eingeführt. Damit ist für die Zwecke dieser Arbeit der strukturelle Aufbau der Kenogrammkomplexionen hinreichend erläutert. In diesem Abschnitt wird nun näher auf den operativen Aspekt der KG eingegangen.
Auf der Listendarstellung der Kenogrammkomplexionen, den Kenogrammsequenzen lassen sich eine Vielzahl von Operationen definieren, die von der TNF-Bedingung abstrahieren und wie die allgemeinen listenverarbeitenden Funktionen von LISP oder ML arbeiten. Solche Operationen sind in der Informatik erschöpfend behandelt worden und werden hier nicht weiter berücksichtigt[20]. Auf allen Strukturebenen der KG lassen sich beliebige n-äre Operationen einführen.
Die *Protostruktur* spiegelt lediglich die Anzahl verschiedener Kenogrammsymbole einer `kseq` wider, Operationen der Protoebene lassen sich mithin als Kardinalzahlarithmetik verstehen.
Die *Deuterostruktur* berücksichtigt zusätzlich die Partitionierung der n Plätze der `kseq` über die verwendeten Kenogrammsymbole. Operationen auf der Deuteroebene lassen sich mit Hilfe der Kombinatorik von Partitionen bestimmen [Jeg73].
Die *Tritostruktur* reflektiert auch die Positionen der verschiedenen Kenogrammsymbole innerhalb einer `kseq`. Operationen der Tritoebene lassen sich mittels der Kombinatorik von Permutationen und Auswahlen unter Berücksichtigung der Reihenfolge bestimmen. Auf die arithmetischen Eigenschaften der Kenogrammatik wird in Kapitel 4 ausführlich eingegangen.

3.3.5.1 Kenogrammatischer Reflektor

In 3.3.2 wurden die unären Normalformoperationen `tnf, dnf, pnf` definiert. In 3.3.3 wurden die unären Funktionen `PDconcrete` und `DTconcrete` implementiert. Als weiteres Beispiel einer unären Operation wird der Reflektor `kref` definiert, der besonders in der Morphogrammatik (vgl. Kapitel 5) von Bedeutung ist.

```
fun kref ks = tnf(rev ks);
```

Wobei `rev` die ML-Funktion zur Umkehrung von Listen ist. Da nur die Tritoebene den genuin kenogrammatischen Formaspekt voll widerspiegelt, lassen sich Deutero- und Protooperationen als DNF und PNF der Tritooperationen trivial ableiten. Proto- und Deuteroreflektor ergeben sich also als:

```
fun Dref ks = dnf(ref ks);
fun Pref ks = pnf(ref ks);
```

Beispiel:

[20] [Wir86], [Abe87].

```
- val a = [1,2,3,1] : kseq;
> val a = [1,2,3,1] : kseq
- kref a;
> [1,2,3,1] : kseq
- Dref a;
> [1,1,2,3] : kseq
- Pref a;
> [1,1,2,3] : kseq
```

3.3.5.2 Kenogrammatische Verkettung

In dem einführenden Vergleich zwischen Kenogrammatik und Semiotik wurde auf die besonderen Eigenschaften kenogrammatischer Operationen hingewiesen. Mithilfe der folgenden Definition der kenogrammatischen Verkettung @[21] lassen sich die angeführten Besonderheiten formal explizieren.
Die kenogrammtische Verkettung zweier Kenogrammsequenzen a, b muß berücksichtigen, daß gleiche Kenogrammsymbole der beiden Sequenzen nicht notwendig zu identifizieren sind. Sei $a = \circ\circ\triangle$ und $b = \circ$. In der verketteten Sequenz $a@b$ kann sich daher das Kenogrammsymbol $\circ$ aus b von den beiden Kenogrammen $\circ\circ$ aus a unterscheiden; entsprechend darf es aber auch mit $\triangle$ aus a identifiziert werden. Außerdem ist es möglich, daß es sich von beiden unterscheidet, was die Einführung eines neuen Kenogrammsymbols aus $\mathbf{K}$, $\square$ impliziert. Daraus ergibt sich die Mehrdeutigkeit der Verkettungsoperation @:

$$\circ\circ\triangle \ @ \ \circ = \{\circ\circ\triangle\circ,\ \circ\circ\triangle\triangle,\ \circ\circ\triangle\square\}$$

Die Verkettung geschieht aber nicht willkürlich, sondern berücksichtigt die Tritostruktur der einzelnen Kenogrammsequenzen:

$$\begin{aligned}\circ\circ\triangle \ @ \ \circ\triangle \quad = \quad \{&\circ\circ\triangle\circ\triangle,\ \circ\circ\triangle\triangle\circ,\ \circ\circ\triangle\circ\square,\ \circ\circ\triangle\triangle\square,\\ &\circ\circ\triangle\square\circ,\ \circ\circ\triangle\square\triangle,\ \circ\circ\triangle\square\star\}\end{aligned}$$

Die Tritostruktur der Ausgangskenogrammsequenzen wird bewahrt. Die kenogrammatische Verkettung @ erzeugt so eine Klasse von Kenogrammsequenzen, die von einander unterschieden sind, jedoch aus den gleichen Komponenten aufgebaut sind[22].

Definition 3.17 (Kenogrammatische Verkettung @)
Die Kenogrammatische Verkettung zweier Kenogrammsequenzen a und b, a@b, ist gegeben durch die Funktion `kconcat a b`*:*

```
fun AG ks = length (rd ks);

fun EE (n,k) =
  let
   fun combinec item list= map (fn x=> item::x) list;
   fun max  (x : int) y= if x>y then x else y;
   fun mkfg from to 0 = [[]]
      |mkfg from to step=
         flat(map (fn i => combinec i (mkfg (i+1) to (step-1)))
```

[21] Die kenogrammatische Verkettung @ ist nicht zu verwechseln mit der ML-Funktion `@` zur Verkettung von Listen.

[22] Diese Kenogrammsequenzen mit unterschiedlicher Trito- jedoch gleicher Komponentenstruktur bilden eine *morphogrammatische Äquivalenzklasse*, siehe 5.4.2

```
            (fromto from (max from to)));
  in
   mkfg  1 (n+1) k
  end;

fun mappat pat template=
   map (fn x => pos x template) pat;

fun mkpats a b =
  let
   fun max (x:int) y= if x>=y then x
               else y;
   fun free n ([] : int list) = []
      |free n (hd::tl) =
          if hd<=n then hd::(free n tl)
          else [];
   fun possperms [] ag=[]
      |possperms [x] ag = [[x]]
      |possperms rest ag=
         flat(map (fn k=> combine k (possperms (remove k rest)
                                               (max k (ag))))
                  (free (ag+1) rest ));
  in
   flat
     (map (fn e => possperms e (AG a))
          (EE (AG a,AG b)))
  end;

fun combinea item list=
   map (fn x=> item@x) list;

fun kconcat ks1 ks2=
   combinea ks1 (map (fn pat => mappat ks2 pat)
                     (mkpats ks1 ks2));
```

Dieser für alle höheren kenogrammatischen Operationen grundlegende Algorithmus der kenogrammatischen Verkettung soll im Folgenden näher erläutert werden. Bei der Verkettung der Sequenzen a und b bleibt a in allen Resultierenden unverändert erhalten. Auch die Struktur von Gleichheit und Verschiedenheit (ϵ/ν-Struktur) von b bleibt konstant, wobei jedoch die Belegung der $AG(b)$ verschiedenen Kenogramme durch verschiedene Auswahlen und Permutationen von Kenogrammsymbolen variiert wird.

Definition 3.18 (Akkretionsgrad) *Die Anzahl verschiedener Kenogrammsymbole in einer Kenogrammsequenz ks, wird durch den Akkretionsgrad $AG(ks)$ bezeichnet. Es gilt $1 \leq AG(ks) \leq |ks|$, wobei $|ks|$ die Länge der Kenogrammsequenz ist.*

Die Menge $E(AG(a), AG(b))$ aller *möglichen* (d.h. der TNF-Bedingung der Bildung von Kenogrammsequenzen genügenden) *Permutationsklassen* wird durch die Funktion `EE(AG a, AG b)` berechnet. E ist Teilmenge der Menge *aller* Permutationsklassen einer kombinatorischen Auswahl ohne Berücksichtigung der Reihenfolge und ohne Wiederholung, vom Umfang $AG(b)$ aus einer $AG(a) + AG(b)$-elementigen Menge,

$P(AG(a), AG(b))$, für die gilt:

$$|P(AG(a), AG(b))| = \binom{AG(a) + AG(b)}{AG(b)}.$$

Aufgrund der TNF-Bedingung darf E nur diejenigen Elemente $e_i = [x_1, \ldots, x_j, \ldots, x_{AG(b)}] \in P$ enthalten, für die entweder gilt:

1. $x_j \leq AG(a) + 1$ oder
2. $(x_j > AG(a) + 1) \Longrightarrow (x_j - \max(x_1, \ldots, x_{j-1}) \leq 1)$.

Beispiel: Sei $AG(a) = 2$ und $AG(b) = 4$, dann ist:

$$|P(2,4)| = \binom{6}{2} = 15$$

und

$$\begin{aligned} P(2,4) \quad = \quad & \{[1,2,3,4],[1,2,3,5],[1,2,3,6],[1,2,4,5],[1,2,4,6], \\ & [1,2,5,6],[1,3,4,5],[1,3,4,6],[1,3,5,6],[1,4,5,6], \\ & [2,3,4,5],[2,3,4,6],[2,3,5,6],[2,4,5,6],[3,4,5,6]\}. \end{aligned}$$

Keines der Elemente von P erfüllt Bedingung 1 und nur $[1,2,3,4]$, $[1,2,3,5]$, $[2,3,4,5]$ und $[3,4,5,6]$ erfüllen Bedingung 2. Daher ist:

$$E(2,4) = \{[1,2,3,4],[1,2,3,5],[2,3,4,5],[3,4,5,6]\}.$$

Die allgemeine Konstruktionsvorschrift für die Menge $E(AG(a), AG(b)$ läßt sich aus diesem Beispiel ableiten: An der ersten Stelle x_1 eines $e_i = [x_1, \ldots, x_j, \ldots, x_{AG(b)}]$ kann nur eines der Kenogrammsymbole $1, \ldots, AG(a) + 1$ stehen. An jeder weiteren Stelle x_j können jeweils die Kenogrammsymbole $k_{j-1} + 1, \ldots, 1 + \max(AG(a), k_{j-1})$ plaziert werden, wobei k_{j-1} eines der möglichen Kenogrammsymbole für die x_{j-1} Stelle des e_i ist.

Beispiel:

```
- EE(3,4);
> [[1,2,3,4],[1,2,4,5],[1,3,4,5],[1,4,5,6],
   [2,3,4,5],[2,4,5,6],[3,4,5,6],[4,5,6,7]] : int list list
```

Als Konstruktionsschema ergibt sich folgende Struktur:

x_1	1				2		3	4
x_2	2		3	4	3	4	4	5
x_3	3	4	4	5	4	5	5	6
x_4	4	5	5	6	5	6	6	7
	e_1	e_2	e_3	e_4	e_5	e_6	e_7	e_8

Für die Permutationsklassen $e_i = [x_1, \ldots, x_j, \ldots, x_{AG(b)}] \in E$ müssen nun alle *passenden* Permutationen gebildet werden. Bei dieser Erzeugung gilt es wiederum, die TNF-Form der Kenogrammsequenzen einzuhalten, so daß nur die Kenogrammsymbole $x_j \leq 1 + AG(b)$ *frei* permutiert werden können, die anderen sind *gebunden* (d.h. ihre Reihenfolge ist dadurch bestimmt, daß $x_j \leq x_k$ für $j \leq k$ gelten muß). Diese auf den Bereich der freien Kenogrammsymbole (durch die Funktion `(free (1+AG a)` e_i`)` berechnet) beschränkte Permutation führt die Funktion `mkpats a b` aus.

Beispiel:

```
- mkpats [1,2] [1,2,3,4];
> val it =
   [[1,2,3,4],[1,3,2,4],[1,3,4,2],[2,1,3,4],
    [2,3,1,4],[2,3,4,1],[3,1,2,4],[3,1,4,2],
    [3,2,1,4],[3,2,4,1],[3,4,1,2],[3,4,2,1],
    [1,3,4,5],[3,1,4,5],[3,4,1,5],[3,4,5,1],
    [2,3,4,5],[3,2,4,5],[3,4,2,5],[3,4,5,2],
    [3,4,5,6]] : int list list

- length it;
> val it = 21 : int
```

Die Permutationen geben nun alle möglichen Muster an, nach denen $AG(b)$ verschiedene Kenogramme mit Symbolen passend belegt werden können. Diese Muster berücksichtigen nur den Akkretionsgrad $AG(b)$, nicht jedoch mögliche Wiederholungen von Kenogrammen innerhalb von b. Die Funktion `mappat b pat` bildet nun die Permutationsmuster auf die tatsächliche ϵ/ν-Struktur von b ab. Die auf diese Weise erzeugten Teilsequenzen stellen alle möglichen Belegungen für b innerhalb der Verkettung $a@b$ dar, so daß vor jede dieser Teilsequenzen nur noch die unveränderte Sequenz a gehängt zu werden braucht, um die Menge aller möglichen Verknüpfungsprodukte zu erzeugen.

Beispiel:

```
- kconcat [1,2] [1,2,3,1,4];
> val it = [[1,2,1,2,3,1,4],[1,2,1,3,2,1,4],[1,2,1,3,4,1,2],
            [1,2,2,1,3,2,4],[1,2,2,3,1,2,4],[1,2,2,3,4,2,1],
            [1,2,3,1,2,3,4],[1,2,3,1,4,3,2],[1,2,3,2,1,3,4],
            [1,2,3,2,4,3,1],[1,2,3,4,1,3,2],[1,2,3,4,2,3,1],
            [1,2,1,3,4,1,5],[1,2,3,1,4,3,5],[1,2,3,4,1,3,5],
            [1,2,3,4,5,3,1],[1,2,2,3,4,2,5],[1,2,3,2,4,3,5],
            [1,2,3,4,2,3,5],[1,2,3,4,5,3,2],
            [1,2,3,4,5,3,6]] : int list list
- length it;
> val it = 21 : int;
```

Die Deutero- und Protoform der Tritoverkettung lassen sich wieder einfach ableiten als:

```
fun Dconcat a b = dnf(kconcat a b);
fun Pconcat a b = pnf(kconcat a b);
```

Beispiel:

```
- kconcat [1,1] [1,1];
> [[1,1,1,1],[1,1,2,2]] : kseq list
- kconcat [1,1] [1,2];
> [[1,1,1,2],[1,1,2,1],[1,1,2,3]] : kseq list
- kconcat [1,2] [1,1];
> [[1,2,1,1],[1,2,2,2],[1,2,3,3]] : kseq list
- kconcat [1,2] [1,2];
> [[1,2,1,2],[1,2,2,1],
   [1,2,1,3],[1,2,2,3],[1,2,3,1],[1,2,3,2],
   [1,2,3,4]] : kseq list
```

3.3.5.2.1 Kenogrammatische Polysemie Die Verkettungsrelation @ ist für die Kenogrammatik von fundamentaler Bedeutung, da sie die Grundlage aller höheren Operationen bildet. Aus diesem Grund soll hier die Anzahl verschiedener Kenogrammsequenzen bestimmt werden, die die Relation $a@b$ erfüllen. Diese Anzahl wird kenogrammatische Polysemie genannt.

Satz 3.2 (Kenogrammatische Polysemie) *Die Anzahl verschiedener Kenogrammsequenzen, die die Verkettungsrelation $a@b$ erfüllen, $N_@(a,b)$, ist gegeben durch:*

$$N_@(a,b) = \sum_{i=1}^{|E(AG(a),AG(b))|} \frac{AG(b)!}{(1+gn(e_i))!}.$$

Hierbei ist E die Menge der verschiedenen Permutationsklassen von $a@b$, $|E|$ die Kardinalität dieser Menge und e_i das i-te Element aus E. $AG(ks)$ bezeichnet die Anzahl verschiedener Kenogrammsymbole in ks. $gn(e_i)$ gibt die Anzahl von Elementen in der Permutationsklasse e_i an, die größer als $1 + AG(a)$ sind.

Beweis: Bei der Verkettung zweier Kenogrammsequenzen a, b zu $a@b$ bleibt die Sequenz a in allen Resultierenden unverändert erhalten. Auch die Struktur von Gleichheit und Verschiedenheit (ϵ/ν-Struktur) von b bleibt konstant, lediglich die Belegung der $AG(b)$ Kenogramme durch verschiedene Permutationsklassen von Kenogrammsymbolen, $e_i \in E$ und deren möglichen Permutationen ermöglichen Mehrdeutigkeiten. Jede Permutationsklasse e_i enthält $AG(b)$ verschiedene Kenogrammsymbole, die aus der Menge der ersten $AG(a) + AG(b)$ Kenogramme aus $\mathbf{K}$ ausgewählt sind. Die Menge $E(AG(a), AG(b))$ enthält alle von einander unabhängigen (d.h. nicht durch Permutationen auf einander abbildbaren) Permutationsklassen e_i.
Die Kardinalität von E, $|E|$ wird durch Aufaddieren aller Zweige der rekursiven Konstruktion von E berechnet:

$$\begin{aligned} |E(n,k)| &= h(1, n+k, k) \text{ mit} \\ h(i,n,k) &= \begin{cases} 1 \quad \text{für } k = 0 \\ \sum_{j=i}^{n} h(j+1, 1+\max(n,j), k-1) \quad \text{sonst} \end{cases} \end{aligned}$$

Für jede der $|E(AG(a), AG(b)|$ Permutationsklassen e_i existieren $|e_i|! = AG(b)!$ verschiedene Permutationen. Die Bildungsbedingungen von Kenogrammsequenzen besagen, daß auf jedem Platz j einer Kenogrammsequenz entweder eines der Kenogramme

$k_1, \ldots, k_{j-1}$ der Plätze $1, \ldots, j-1$ wiederholt, oder aber das $1+\max(k_1, \ldots, k_{j-1})$-te Kenogrammsymbol aus **K** eingeführt wird. Bei der Verkettung zweier Kenogrammsequenzen, $a@b$, muß also berücksichtigt werden, daß nur die Kenogrammsymbole $1, \ldots, AG(a)+1$ beliebig über b permutiert werden können, daß jedoch die Kenogrammsymbole $AG(a)+2, \ldots, AG(a)+AG(b)$ nur dann auftreten dürfen, falls schon *alle* niedrigeren Kenogrammsymbole benutzt werden. Die Anzahl der von einer Permutationsklasse e_i benutzten Kenogramme, die größer als $AG(a)+1$ sind, wird durch $gn(e_i)$ angeben. Da diese $gn(e_i)$ Kenogramme wegen ihrer festgelegten Reihenfolge nicht permutiert werden können, sind für jedes e_i nur:

$$\frac{AG(b)!}{(1+gn(e_i))!}$$

Permutationen möglich. Die Gesamtanzahl möglicher Permutationen von Kenogrammen über b und somit die Anzahl aller möglichen $a@b$-Strukturen ist also:

$$N_@(a,b) = \sum_{i=1}^{|E(AG(a),AG(b))|} \frac{AG(b)!}{(1+gn(e_i))!} \quad \blacksquare$$

Die Kardinalität von E, $|E(n,k)|$ wird durch die Funktion `Ekard (n,k)` berechnet:

```
fun Ekard (n,k) =
  let
   fun max (x: int) y = if x>y then x else y;
   fun Xi from to 0 = 1
      |Xi from to step=
         sum from to (fn i => Xi (i+1) (max to (i+1)) (step-1));
  in
   Xi 1 (n+1) k  end;
```

Die Anzahl der Kenogrammmsequenzen, die die Verkettungsrelation $a@b$ erfüllen, $N_@(a,b)$, die kenogrammatische Polysemie, wird durch die Funktion `NN(a,b)` bestimmt:

```
fun NN (a,b) =
    let
      val E = EE ((AG a),(AG b));
      fun e i =pos i E;
      fun gn [] = 0
         |gn (x::xs) = if (x>((AG a)+1)) then 1+(gn xs)
                       else gn xs;
    in
      sum 1 (Ekard((AG a),(AG b)))
         (fn i => (fak (length (e i))) div (fak(1+gn(e i))) )
    end;
```

Beispiel:

```
- NN ([1,2],[1,2,3,1,4]);
> val it = 21 : int
```

```
- NN ([1,2,3], [1,2,3,1,4]);
> 73 : int

- length(kconcat [1,2,3] [1,2,3,1,4]);
> 73 : int;
```

Die von `NN` berechnete Anzahl deckt sich also mit der Anzahl der durch `kconcat` konstruktiv erzeugten *Polyseme*.

3.3.5.3 Indizierte Verkettung

Die Mehrdeutigkeit der kenogrammatischen Verkettung @ läßt sich durch die ϵ/ν-Schreibweise anschaulich erklären. Sei beispielsweise $a = \circ\triangle$ und $b = \circ\circ$, dann ist in ϵ/ν-Notation:

$$a = \begin{array}{cc} \nu & \\ \hline \circ & \triangle \end{array} \qquad \text{und} \qquad b = \begin{array}{cc} \epsilon & \\ \hline \circ & \circ \end{array}$$

Werden a und b zu $a@b$ verkettet, so entstehen Kenogrammsequenzen der Länge 4, deren Komponenten jeweils die gleiche Struktur wie a und b aufweisen:

$$\begin{array}{cccc} \nu & & \epsilon & \\ \hline \circ & \triangle & \_ & \_ \end{array}$$

Aus diesen Informationen über die Länge der resultierenden Sequenzen und der internen Struktur der Teilsequenzen erhalten wir also nur eine partielle Bestimmung der resultierenden ϵ/ν-Strukturen. In diesem Fall sind erst zwei der $\frac{4(4-1)}{2} = 6$ Positionen bestimmt:

$$\begin{array}{cccccc} & & ?_4 & & & \\ & ?_2 & & ?_3 & & \\ \nu & & ?_1 & & \epsilon & \\ \hline \circ & \triangle & & \_ & \_ & \end{array}$$

Die Unbestimmtheiten oder *Freiheiten*[23] $?_i$ innerhalb der ϵ/ν-Struktur müssen nun mit konkreten ϵ oder ν Werten belegt werden, um eine bestimmte Kenogrammsequenz zu designieren. Diese Belegung kann allerdings nicht willkürlich geschehen, sondern muß der auf der Transitivität von δ beruhenden Konsistenzbedingung genügen:

$$\forall i,j,k \quad \delta(i,j) = ((i,j),\epsilon) \iff \begin{array}{l} \delta(i,k) = ((i,k),x) \wedge \\ \delta(j,k) = ((j,k),x) \end{array} \tag{3.4}$$

Da im obigen Beispiel $\delta(3,4) = ((3,4),\epsilon)$, ist es wegen der Konsistenzbedingung nicht möglich $\delta(2,4)$ mit $((2,4),\epsilon)$ und gleichzeitig $\delta(2,3)$ mit $((2,3),\nu)$ zu belegen. Die resultierenden ϵ/ν-Strukturen von $a@b$ ergeben sich also zu:

$$\begin{array}{cccccccc} & & \epsilon & & & & \nu & \\ & \epsilon & & \nu & & \nu & & \epsilon \\ \nu & & \nu & & \epsilon & & \nu & & \epsilon & & \epsilon \\ \hline \circ & \triangle & \circ & & \circ & & \circ & \triangle & \triangle & & \triangle \end{array}$$

$$\begin{array}{cccc} & \nu & & \\ \nu & & \nu & \\ \nu & & \epsilon & \\ \hline \circ & \triangle & \square & \square \end{array}$$

[23] [Hou88], [Nie88].

Die Verkettung @ ist mehrdeutig und erzeugt eine Klasse von möglichen Kenogrammsequenzen, die die Verkettungsrelation erfüllen. Jede dieser einzelnen Kenogrammsequenzen ist aber durch eine jeweilige ϵ/ν-Struktur eindeutig bestimmt. Benutzt man die ϵ/ν-Struktur als Indexierung, gelangt man zu einer alternativen Interpretation der kenogrammatischen Verkettung: Die resultierende Kenogrammsequenz ist dann jeweils eineindeutig bestimmt, jedoch gibt es eine *Vielzahl* von kenogrammatischen Verkettungsoperationen, die sich jeweils aus @ und einer — als Index dienenden — bestimmten Belegung der ϵ/ν-Freiheiten zusammensetzen. Für a und b kann dies so notiert werden:

$$\begin{aligned} a@_1 b &= \text{○△○○} \\ a@_2 b &= \text{○△△△} \\ a@_3 b &= \text{○△□□}. \end{aligned}$$

Wobei:

$$\begin{aligned} @_1 &= @_{[(1,1,\epsilon)]} \\ @_2 &= @_{[(1,1,\nu),(2,1,\epsilon)]} \\ @_3 &= @_{[(1,1,\nu),(2,1,\nu)]} \end{aligned}$$ [24]

Diese indizierte Benutzung der Verkettungsoperation wird durch die ML-Funktion `kligate` implementiert:

```
fun collfits a [] rule = []
   |collfits a (b::bs) (rule as (x,y,en))=
       if ((en=E) andalso ((pos x a)=(pos (y) b)))
       then
         b::collfits a bs rule
       else if ((en=N) andalso ((pos x a)<>(pos (y) b)))
       then
         b::collfits a bs rule
       else collfits a bs rule;

fun mapvermat a bs []=bs
   |mapvermat a [] enstruc = []
   |mapvermat a bs (rule::rules)=
       mapvermat a (collfits a bs rule) rules;

fun kligate a b enstruc =
   combinea a
            (mapvermat a
                       (map (fn pat => mappat b pat)
                            (mkpats a b))
                       enstruc);
```

Beispiel:

```
- val a = [1,2] : kseq;
> val a = [1,2] : kseq
- val b = [1,1] : kseq;
> val b = [1,1] : kseq
```

[24] Diese Indexierung ist wie folgt zu lesen: für $@_1$ besteht zwischen dem ersten Kenogramm aus a und dem ersten Kenogramm aus b eine ϵ–Beziehung usw.

```
- kligate a b [(1,1,E)];
> [[1,2,1,1]] : kseq list

- kligate a b [(1,1,N),(2,1,E)];
> [[1,2,2,2]] : kseq list

- kligate a b [(1,1,N),(2,1,N)];
> [[1,2,3,3]] : kseq list
```

Außer dieser *totalen* Indizierung, die eindeutige Ergebnisse bestimmt, läßt sich auch eine *partielle* Indizierung angeben, die nicht eindeutig ist, sondern jeweils nur bestimmte Teilmengen der allgemeinen Verkettungsrelation @ repräsentiert.

Beispiel:

```
- kligate a b [(1,3,N)];
> [[1,2,2,2],[1,2,3,3]] : kseq list
```

Diese partiell indizierte Verkettung wird in der Morphogrammatik (vgl 5.4.3) bei der *Komposition* von Morphogrammen zu Morphogrammatrizen benutzt. Die sich bei dieser Operation ergebende Mehrdeutigkeit, die *morphogrammatische Polysemie*, wird in Kapitel 8 analysiert.

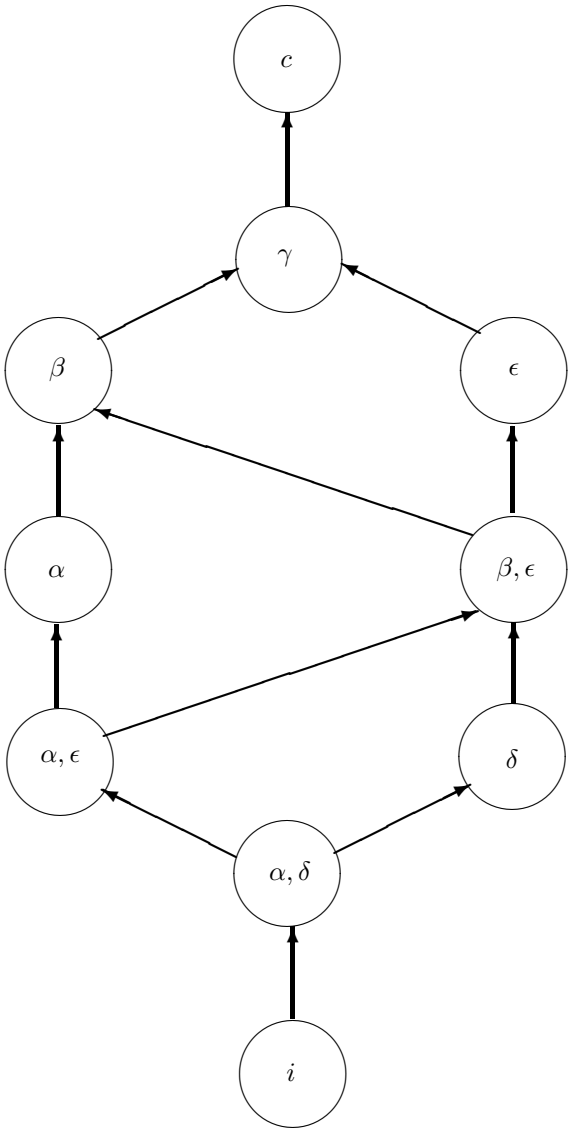


Abbildung 3.9: Das Verhältnis der zehn Äquivalenzrelationen über B^A. Die Pfeile $\longrightarrow$ symbolisieren die Implizierungsrelation.

Proto-	○			○						○					○
struktur	○			○						○					△
	○			○						△					□
`Pcontexture 4;`	○			△						□					★
`PDconcrete`	⇓														
Deutero-	○		○			○				○					○
struktur	○		○			○				○					△
	○		○			△				△					□
`Dcontexture 4;`	○		△			△				□					★
`DTconcrete`	⇓														
Trito-	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○
Struktur	○	○	○	△	△	○	△	△	○	△	△	△	△	△	△
	○	○	△	○	△	△	○	△	△	○	△	□	□	□	□
`Tcontexture 4`	○	△	○	○	△	△	△	○	□	□	□	○	△	□	★

Abbildung 3.10:
Klassifikation der Kenogrammsequenzen der Länge 4

Kapitel 4

Kenoarithmetik

Im vorhergehenden Kapitel wurde die Kenogrammatik zunächst durch einen kontrastierenden Vergleich der Kenogramm(komplexions)-Konzeption mit dem Zeichen(reihen)-Konzept von der klassischen Semiotik abgegrenzt. Im zweiten Teil des Kapitels wurden die kenogrammatischen Konzepte in einem definitorischen Aufbau formal eingeführt, wobei die Darstellung von Kenogrammkomplexionen als Kenogrammsequenzen in Analogie zur klassischen Zeichenreihen- oder Stringnotation geschah, was wiederum der abgrenzenden Bestimmung der Kenogrammatik zu semiotischen Konzepten diente.
In diesem Kapitel soll nun eine andere Darstellungsperspektive gewählt werden, indem die Kenogrammatik in Beziehung zur Arithmetik der natürlichen Zahlen gesetzt wird. Diese Untersuchung der arithmetischen Operativität der Kenogrammatik führt jede der drei kenogrammatischen Strukturebenen in einer der jeweiligen Arithmetik angemessenen Notation ein. Die für jede Ebene geltende Arithmetik wird formal beschrieben und einige der jeweils geltenden Gesetzmäßigkeiten analysiert. Für die Proto- und Deuteroebene lassen sich vereinfachte Partitionsnotierungen angeben, wodurch Proto- und Deuteroarithmetik in direkter Anlehnung an die klassische Arithmetik beschrieben werden können. Da in der Tritoebene der eigentliche kenogrammatische Formaspekt der Strukturen in seiner vollen Ausprägung auftritt, lassen sich hier keine quantitativen, von der Gestalt abstrahierenden Notationen angeben.
Für die Tritoebene wird eine Arithmetik eingeführt, die sich in größtmöglicher Nähe zu klassischen semiotischen und arithmetischen Konzeptionen befindet. Dieser Ansatz führt zu mehrdeutigen arithmetischen Operationen, die auch als Vielheit von indizierten, eindeutigen Operationen interpretiert werden können.

4.1 Protoarithmetik

4.1.1 Struktur der Protozahlen

Die im letzten Kapitel definierte Funktion `Pcontexture(n)` erzeugt die Menge aller (Normalformen von) Protoäquivalenzklassen der Länge n. Jedem $n \in N$ ist so eindeutig eine *Protokontextur* mit $Pcard(n) = n$ PNF-sequenzen zugeordnet. Der Aufbau dieser Protokontexturen ist in Schema 4.1 veranschaulicht.
Auf jeder Stufe n existieren jeweils $Pcard(n)$ Strukturen, die nur durch die Anzahl der enthaltenen Kenogrammsymbole unterschieden sind. Jede dieser Strukturen ist

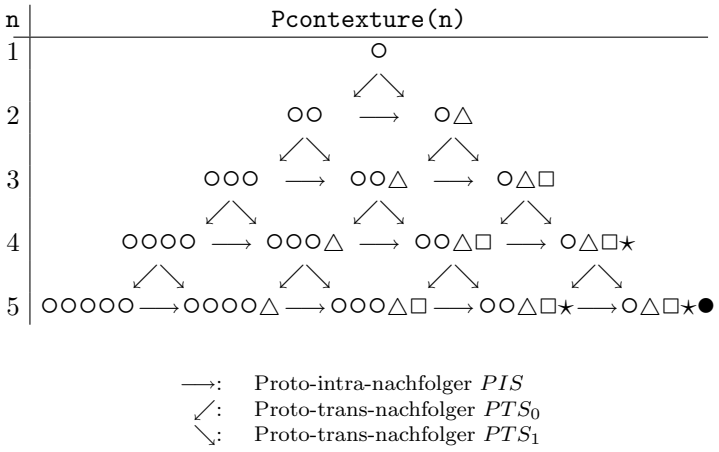


Abbildung 4.1: Die ersten fünf Protokontexturen

eindeutig durch ein Tupel (n, k) — im folgenden *Protozahl* genannt — bestimmt, wobei n die Länge der Sequenz und k die Anzahl von Kenogrammen ist. Diese Tupeldarstellung der Protokontexturen ist Abbildung 4.2 dargestellt. Die eingeführten Pfeile symbolisieren die arithmetischen Beziehungen innerhalb der *Protostruktur*.

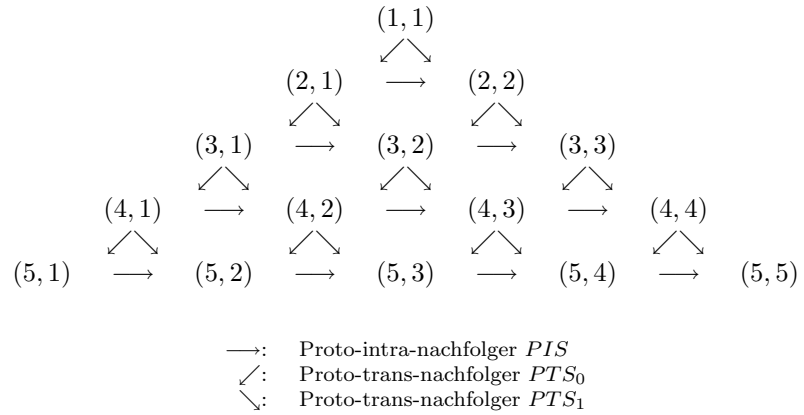


Abbildung 4.2: Die ersten fünf Protokontexturen in Tupeldarstellung

Definition 4.1 (Protozahl) *Eine Protozahl (n, k) ist die Klasse aller Partitionen von n in k Teile, wobei $n, k \in N$ und $n \geq k$.*

Definition 4.2 (Proto-gleichheit) *Zwei Protozahlen (n, k) und (m, j) sind genau dann gleich, wenn $n = m$ und $k = j$.*

Definition 4.3 (Proto-intra-nachfolger PIS) *der Proto-intra-nachfolger PIS einer Protozahl (n, k) ist definiert durch:*

```
exception Pis;
fun PIS (n,k) = if k=n then raise Pis
                else (n,k+1);
```

Definition 4.4 (Proto-trans-nachfolger PTS_0) *Der Proto-trans-nachfolger PTS_0 einer Protozahl (n,k), $\swarrow$, ist definiert durch:*

```
fun PTS0 (n,k) = (n+1,k);
```

Definition 4.5 (Proto-trans-nachfolger PTS_1) *Der Proto-trans-nachfolger PTS_1 einer Protozahl (n,k), $\searrow$, ist definiert durch:*

```
fun PTS1 (n,k) = (n+1,k+1);
```

4.1.2 Arithmetische Operationen und Eigenschaften

Die Protozahlen und die über ihnen definierten Nachfolgeoperationen spannen einen arithmetischen Raum auf, die *Protoarithmetik*. Zur näheren Veranschaulichung dieser Protoarithmetik sollen hier die von Schadach [Sch67c] im Rahmen seiner Untersuchungen zur Theorie der Protozahlen erzielten Ergebnisse zusammengefaßt werden.

Definition 4.6 (Protoaddition) *Die Addition zweier Protozahlen (n,k) und (m,j) erzeugt eine Menge von zwei möglichen Lösungen:*

$$(n,k) +_p (m,j) = \{(n+m, k+j-1), (n+m, k+j)\}.$$

Beispiel:

$$(3,2) +_p (4,3) = \{(7,4),(7,5)\}$$

oder in PNF-Notation:

$$\circ\circ\triangle +_p \circ\circ\triangle\square = \{\circ\circ\circ\circ\triangle\square\star,\ \circ\circ\circ\triangle\square\star\bullet\}$$

Die Protoaddition ist abgeschlossen in dem Sinne, daß die Summe zweier Protozahlen stets wieder zwei Protozahlen ergibt.

Satz 4.1 (Distributivität) *Die Protoaddition ist distributiv bezüglich Mengen von Protozahlen.*

Beweis:

$$\begin{aligned}(n,k) +_p \{(p,q),(r,s)\} &= \{(n,k) +_p (p,q), (n,k) +_p (r,s)\}\\ \{(p,q),(r,s)\} +_p (n,k) &= \{(p+q) +_p (n,k), (r,s) +_p (n,k)\}.\end{aligned}$$

Satz 4.2 (Kommutativität) *Die Addition von Protozahlen ist kommutativ.*

Beweis:

$$\begin{aligned}(n,k) +_p (m,j) &= \{(n+m, k+j), (n+m, k+j-1)\}\\ &= \{(m+n, j+k), (m+n, j+k-1)\}\\ &= (m,j) +_p (n,k)\ \blacksquare\end{aligned}$$

Satz 4.3 (Assoziativität) *Die Addition von Protozahlen ist assoziativ.*

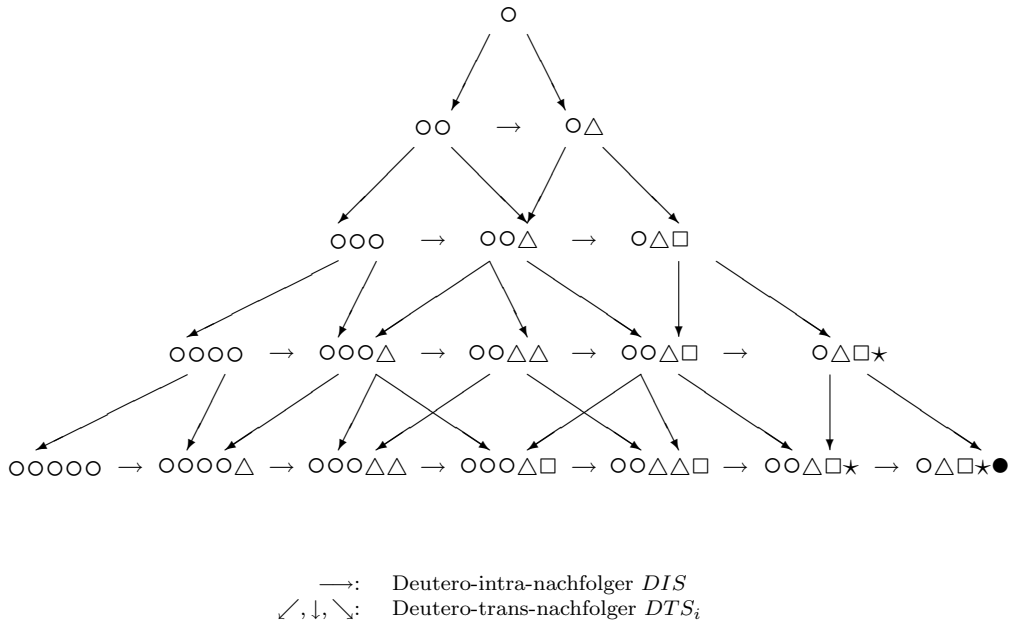
Beweis:

$$
\begin{aligned}
&(n,k) +_p [(p,q) +_p (r,s)] \\
&= (n,k) +_p \{(p+r, q+s), (p+r, q+s-1)\} \\
&= \{(n,k) +_p (p+r, q+s), (n,k) +_p (p+r, q+s-1)\} \\
&= \{\{(n+p+r, k+q+s), (n+p+r, k+q+s-1)\}, \\
&\quad \{(n+p+r, k+q+s-1), (n+p+r, k+q+s-2)\}\} \\
&= \{(n+p, k+q) +_p (r,s), (n+p, k+q-1) +_p (r,s)\} \\
&= \{(n+p, k+q), (n+p, k+q-1)\} +_p (r,s) \\
&= [(n,k) +_p (p,q)] +_p (r,s) \ \blacksquare
\end{aligned}
$$

4.2 Deuteroarithmetik

4.2.1 Die Struktur der Deuterozahlen

Die Funktion `Dcontexture(n)` berechnet die Menge aller Normalformen von Deuteroäquivalenzklassen der Länge n. Jedem $n \in N$ ist so eine *Deuterokontextur* mit $Dcard(n) = \sum_{k=1}^{n} P(n,k)$ DNF-sequenzen zugeordnet. Die ersten fünf Deuterokontexturen weise den folgenden Aufbau auf (Abbildung 4.3):



⟶: Deutero-intra-nachfolger DIS
↙, ↓, ↘: Deutero-trans-nachfolger DTS_i

Abbildung 4.3: Die ersten fünf Deuterokontexturen

Die Deuterostrukturen repräsentieren die Partitionierung der n Positionen einer Sequenz über k Kenogrammsymbolen. Der obige Aufbau läßt sich demzufolge auch in einer Partitionsnotation darstellen (Abbildung 4.4).

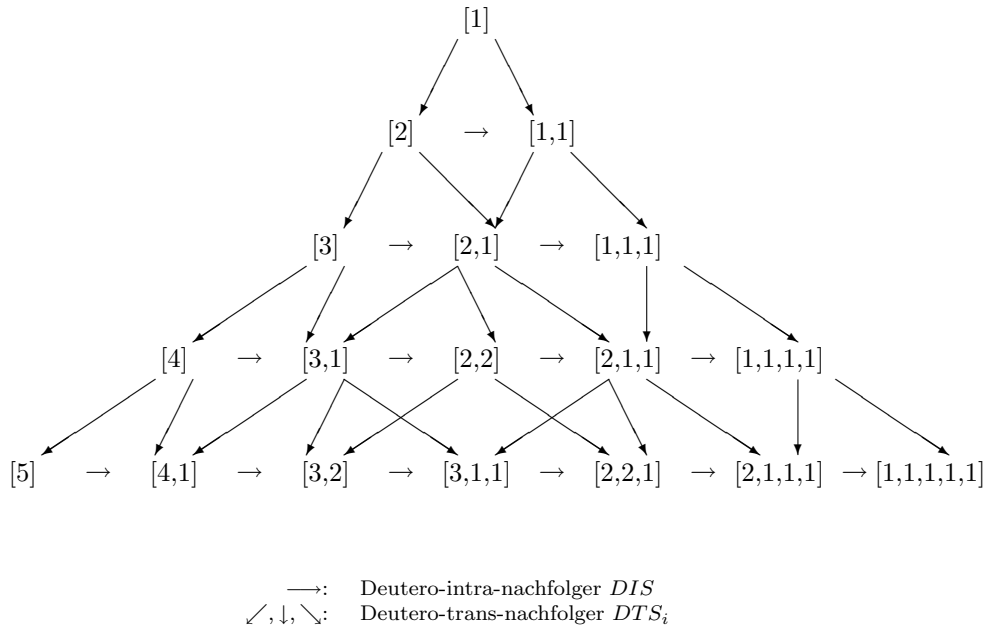


Abbildung 4.4: Die ersten fünf Deuterokontexturen als Partitionen

Definition 4.7 (Deuterozahl) *Eine Deuterozahl D der Kardinalität n ist eine Partition $\Pi(n)$ von n in* max *Teile mit jeweils P_i ($1 \leq i \leq$ max) Elementen.*

$$D = \Pi(n) = [p_1, p_2, \ldots, p_{\max}]$$

Definition 4.8 (Deutero-gleichheit) *Zwei Deuterozahlen D und E sind gleich, gdw. ihre Partitionen gleich sind:*

$$D = E \Longleftrightarrow \Pi(|D|) = \Pi(|E|)$$

Die in den beiden obigen Diagrammen benutzten Pfeile symbolisieren die arithmetischen Nachfolgebeziehungen zwischen den Deuterozahlen:

Definition 4.9 (Deutero-intra-nachfolger DIS) *Der Deutero-intra-nachfolger DIS einer Deuterozahl D in Partitionsnotierung ist durch die Funktion* `DIS D` *bestimmt:*

```
exception Dis;
fun DIS D =
  if (forall D (fn x => x=1)) then raise Dis
  else
   let
     val m = sum 1 (length D) (fn i => (pos i D))
     val i = ((firstocc 1 D) - 1) handle Place => (length D)
     val pi = (pos i D)-1
     val u = m - (sum 1 (i-1) (fn k => (pos k D)))
```

```
    val j = u div pi
    val rest = u mod pi
    val news = map (fn x => pi) (fromto 1 (j-1)) ;
  in
    (nfirst (i-1) D) @ [pi] @ news @ (if rest=0 then []
                                      else [rest])
  end;

- DIS [2,2,1];
> val it = [2,1,1,1] : int list
```

Die Deutero-trans-nachfolgeoperationen ist mehrdeutig. Nach Niegel [Nie88] existieren für jede Deuterozahl $D = [p_1, \ldots, p_{\max}]$ insgesamt:

$$n_{DTS}(D) = 2 + \sum_{i=1}^{\max-1} \texttt{sign}(p_i - p_{i+1})$$

verschiedene Trans-nachfolger, wobei:

```
fun sign n = if n>0 then 1
             else if n=0 then 0
             else ~1;
```

Definition 4.10 (Deutero-trans-nachfolger DTS) *Die Menge der $n_{DTS}(D)$ Deutero-trans-nachfolger der Deuterostruktur D, $\{DTS_1(D), \ldots, DTS_{n_{DTS}(D)}(D)\}$ ist bestimmt durch* `DTS D`*:*

```
fun DTS D =
    [((pos 1 D)+1)::(tl D)] @
    (remnils (map (fn i => if sign((pos i D) - (pos (i+1) D)) = 1
                           then replacepos (i+1) D ((pos (i+1) D)+1)
                           else [])
                  (fromto 1 ((length D)-1)))) @
    [D@[1]]
```

Beispiel:

D	$n_{DTS}(D)$	$DTS_1, \ldots, DTS_{n_{DTS}(D)}$
$[3,1]$	3	$[[4,1],[3,2],[3,1,1]]$
$[3,2,1]$	4	$[[4,2,1],[3,3,1],[3,2,2],[3,2,1,1]]$

4.2.2 Arithmetische Operationen

Auch für die Deuterozahlen läßt sich eine Additionsoperation angeben:

Definition 4.11 (Deuteroaddition) *Die Summe zweier Deuterozahlen $D = [p_1, \ldots, p_{\max_D}]$ und $E = [q_1, \ldots, q_{\max_E}]$ ist gegeben als die Menge:*

$$\begin{aligned}
&[p_1, \ldots, p_{\max_D}] +_d [q_1, \ldots, q_{\max_E}] \\
&\quad = \Big\{ [p_1, \ldots, p_{\max_D}, q_1, \ldots, q_{\max_E}], \\
&\qquad [p_1, \ldots, p_{i-1}, p_i - 1, p_{i+1}, \ldots, p_{\max_D}, \\
&\qquad\ q_1, \ldots, q_{j-1}, q_j - 1, q_{j+1}, \ldots, q_{\max_E}, 1] \Big\},
\end{aligned}$$

wobei $1 \leq i \leq \max_D$ *und* $1 \leq j \leq \max_E$.

4.3 Tritoarithmetik

4.3.1 Die Struktur der Tritozahlen

Der genuin kenogrammatische Gestaltaspekt erscheint erst auf der Tritoebene in seiner vollen Ausprägung. Da sich die Deutero- und Protoarithmetik als Reduktionen der Tritoarithmetik verstehen lassen, wird diese hier eingehender betrachtet.
Die Funktion `Tcontexture(n)` berechnet alle möglichen TNF-sequenzen der Länge n. Jedem $n \in N$ ist auf diese Weise eine aus $Tcard(n) = \sum_{k=1}^{n} S(n,k)$ TNF's bestehende *Tritokontextur* zugeordnet. Der Aufbau der ersten vier Kontexturen hat folgende Gestalt (Abbildung 4.3.1).

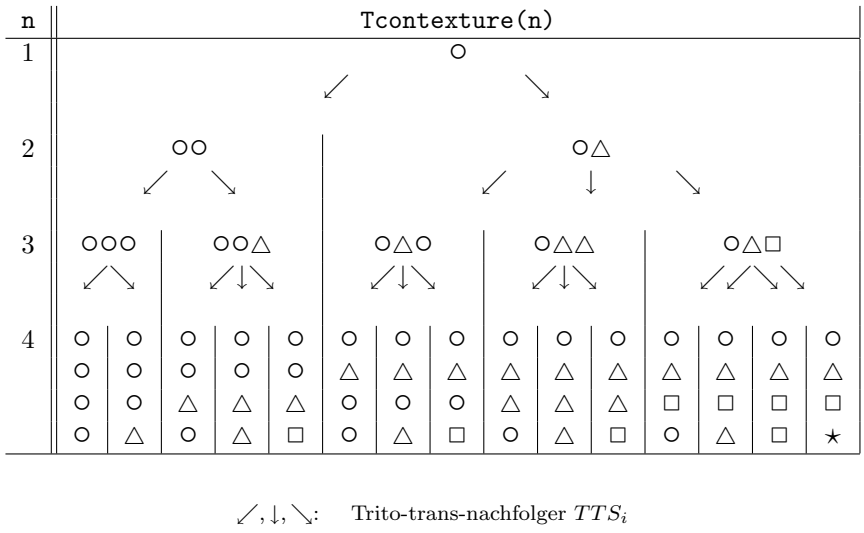


Abbildung 4.5: Die ersten vier Tritokontexturen

Definition 4.12 (Trito-intra-nachfolger TIS) *Der Trito-intra-nachfolger einer TNF-sequenz ts, $TIS(ts)$, ist bestimmt durch die Funktion* `TIS ts`*:*

```
exception Tis;
fun TIS ts=
  let
   val n=length ts;
   fun lastrep []=[]
      |lastrep seq=
        let
         fun member x []=false
            |member x (y::ys)= (x=y) orelse member x ys;
         val (last::rest) = rev seq;
```

```
        in
         if (member last rest) then seq
         else lastrep (rev rest)
        end;
  in
   if (pos n ts)=n then raise Tis
   else if (last ts) <= (max (nfirst (n-1) ts))
          then (nfirst (n-1) ts) @ [1+(last ts)]
   else let
          val first = rev(lastrep (nfirst (n-1) ts));
        in
          (rev (tl first))@[1+(hd first)]@(nlistof (n-(length first)) 1)
        end
  end;
```

Der Trito-intra-nachfolger ist nicht definiert für Tritosequenzen die den vollen Akkretionsgrad $AG(ts) = n$ der Tritokontextur `Tcontexture(n)` aufweisen, da diese innerhalb der Kontextur keinen weiteren Nachfolger mehr besitzen (und die obere *Kontexturgrenze* darstellen). Die Funktion `TIS` prüft dies durch den `if ...then` Ausdruck `if (pos n ts)=n then raise Tis` und ruft beim Erreichen der Kontexturgrenze die Exception (Fehlerroutine) `Tis` auf.
Eine Funktion `Tsucc`, die die Vereinigungsmenge aller Tritokontexturen sukkzessive durchläuft, muß beim Erreichen der oberen Kontexturgrenze einer Kontextur `Tcontexture(n)` zum ersten Element `(nlistof n+1 1)` der nächsthöheren Kontextur `Tcontexture(n+1)` springen und diese dann wiederum mit dem `TIS` durchlaufen. Die Funktion `Tsucc` kann also durch *handling* des Exceptionmechanismus `raise Tis` der Funktion `TIS` definiert werden:

```
fun Tsucc(ts) = TIS(ts)
        handle Tis =>(nlistof (n+1) 1);

- Tsucc [1,2,3,4,4];
> val it = [1,2,3,4,5] : int list
- Tsucc it;
> val it = [1,1,1,1,1,1] : int list
```

Definition 4.13 (Trito-universum) *Das Trito-universum TU ist die abzählbar unendliche Menge aller Tritosequenzen. TU enthält* [1]*, die erste Tritosequenz, und alle ihre* `Tsucc`*–Nachfolger.*

Eine solche unendliche Datenstruktur läßt sich in ML mit Hilfe des Konzepts der 'Lazy-Lists' formulieren.

```
datatype 'a seq = Nil
                | Cons of 'a * (unit -> 'a seq);
```

Diese Deklaration definiert den Datentyp `seq` der Lazy Liste, deren jeweils erstes Element, `head`, aktual vorliegt, deren `tail` jedoch als Funktion notiert ist und nur bei Bedarf schrittweise tiefer evaluiert wird:

```
fun head (Cons (x,_))=x;
fun tail (Cons(_,xf))=xf();
```

Die Menge aller Tritosequenzen ab einer bestimmten Sequenz `ts` wird notiert als:

```
fun from ts = Cons(ts,fn () => from (Tsucc ts));
```

Die Menge aller Tritosequenzen, das Trito–universum TU kann dann als

```
val TU = from [1];
```

dargestellt werden. Die Funktion `nfirstq(n,seq)` berechnet die ersten `n` Elemente der Lazy Liste `seq` und verkettet sie zu einer linearen Liste:

```
fun nfirstq (0, xq)=[]
   |nfirstq (n, Nil)=[]
   |nfirstq (n, Cons(x,xf))= x::(nfirstq (n-1, xf ()));
```

Beispiel: Die Berechnung:

```
- nfirstq(23,TU);
> val it = [[1],[1,1],[1,2],[1,1,1],[1,1,2],
            [1,2,1],[1,2,2],[1,2,3],[1,1,1,1],
            [1,1,1,2],[1,1,2,1],[1,1,2,2],[1,1,2,3],
            [1,2,1,1],[1,2,1,2],[1,2,1,3],[1,2,2,1],
            [1,2,2,2],[1,2,2,3],[1,2,3,1],[1,2,3,2],
            [1,2,3,3],[1,2,3,4]] : int list list
```

erzeugt die ersten 24 Tritosequenzen aus TU, die auch in Abbildung 4.3.1 aufgeführt sind.

Die Nachfolgeoperation zwischen zwei Kontexturen ist wie der DTS mehrdeutig. Wie man leicht sieht, existieren für jede Kenogrammsequenz ts:

$$n_{TTS}(ts) = AG(ts) + 1$$

verschiedene Nachfolger, wobei $AG(ts)$ den *Akkretionsgrad* der Kenogrammsequenz, d.h. die Anzahl der von ihr verwendeten verschiedenen Kenogrammsymbole angibt.

Definition 4.14 (Trito-trans-nachfolger TTS) *Die Menge der $n_{TTS}(ts)$ Transnachfolger einer Kenogrammsequenz ts wird durch die Funktion* `TTS(ts)` *bestimmt:*

```
fun TTS ts =   map (fn i => ts@[i])
                   (fromto 1 ((AG ts)+1));
```

Beispiel:

ts	$AG(ts)$	$\{TTS_1, \ldots, TTS_{n_{TTS}(ts)}\}$
$[1,2,1]$	2	$[[1,2,1,1],[1,2,1,2],[1,2,1,3]]$
$[1,2,3]$	3	$[[1,2,3,1],[1,2,3,2],[1,2,3,3],[1,2,3,4]]$

Die Funktion $TTS(ts)$ entspricht der im vorigen Kapitel entwickelten Verkettung von Kenogrammsequenzen mit dem Einselement ○ :

```
- kconcat [1,2,3] [1];
> [[1,2,3,1], [1,2,3,2], [1,2,3,3], [1,2,3,4]] : kseq list
```

4.3.2 Arithmetische Operationen

4.3.2.1 Kenogrammatische Addition

Die hier angedeutete Rückführung der kenoarithmetischen Operationen auf entsprechende kenogrammatische Verkettungsoperationen geschieht analog zur konstruktiven Begründung der Nachfolgeoperation der natürlichen Zahlen durch die semiotischen Verkettungsoperation Eine natürliche Zahl n wird klassisch als n-malige Hintereinanderreihung (im Sinne einer semiotischen Verkettung) des Zeichens | verstanden:

$$n = \underbrace{|||\ldots|}_{n \text{ mal}}.$$

Der Nachfolger der Zahl n ist dann gerade das Ergebnis der Verkettung der Zeichenkette n mit einem einzelnen Zeichen |:

$$n| = \underbrace{|||\ldots|}_{n \text{ mal}}| = \underbrace{|||\ldots||}_{n+1 \text{ mal}}.$$

Ausgehend von dieser Überlegung liegt es nun nahe, eine kenoaritmetische Addition mittels der kenogrammatischen Verkettung zu definieren.

Definition 4.15 (Trito-arithmetische Addition) *Die Menge aller möglichen Summen zweier Kenogrammsequenzen a, b wird bestimmt durch die Funktion $kadd(a, b)$:*

```
fun kadd (a,b) = kconcat a b;
```

Die Mehrdeutigkeit dieser allgemeinen Addition läßt sich in Anlehnung an das in 3.3.5 eingeführte Indizierungsschema als eine Vielzahl unterschiedlicher, indizierter Additionen $kadd_i$ deuten. Für jede so bestimmte Additionsoperation $kadd_i$ gilt nun:

$$|Kadd_i(ks_1, ks_2)| = |ks_1| + |ks_2|.$$

Die klassische Arithmetik der natürlichen Zahlen erweist sich somit als eine Reduktion auf den rein quantitativen Aspekt kenogrammatischer Arithmetik. Diese Reduktion wird durch die in 3.3.1.1 definierte Cardäquivalenzrelation definiert.

4.3.2.2 Kenogrammatische Multiplikation

Die auf den ersten Blick vielleicht trivial erscheinende Interpretation der kenogrammatischen Verkettung als kenoarithmetische Addition führt jedoch zu überraschenden Ergebnissen, wenn wir versuchen in einem weiteren Schritt eine kenoarithmetische Multiplikation einzuführen.
In der klassischen Arithmetik wird die Multiplikation zweier natürlicher Zahlen m und n als n-malige Addition der Zahl m verstanden:

$$m \times n = \underbrace{m + m + \cdots + m}_{n-\text{mal}}.$$

Aufgrund der Kommutativität gilt:

$$m \times n = n \times m = \underbrace{n + n + \cdots + n}_{m-\text{mal}}.$$

Das Einselement der Addition ist das Neutralelement der Multiplikation, so daß wir für die kenoarithmetische Multiplikation *kmul* folgern:

$$kmul(ks_1, [1]) = ks_1$$

und

$$kmul([1], ks_2) = ks_2.$$

Besteht nun die zweite Kenogrammsequenz aus mehr als einer Stelle, muß auch die in der klassischen Arithmetik unberücksichtigte *Gestalthaftigkeit* der zu multiplizierenden Objekte berücksichtigt werden.
Im einfachsten Fall besteht ks_2 lediglich aus dem Kenogramm ○, das $|ks_2|$ mal wiederholt wird: $ks_2 = \underbrace{[1, \ldots, 1]}_{|ks_2| \text{ mal}}$. Dann ist:

$$kmul(ks_1, ks_2) = \underbrace{ks_1 ks_1 \ldots ks_1}_{|ks_2| \text{ mal}}$$

Wobei die Hintereinanderschreibung $ks_1 ks_1 \ldots$ die kenogrammatische Verkettung symbolisiert. Sofort zeigt sich, daß *kmul* nicht kommutativ ist:

$$\begin{aligned} kmul([1,2],[1,1,1]) &= [1,2,1,2,1,2] \neq \\ kmul([1,1,1],[1,2]) &= [1,1,1,2,2,2] \end{aligned}$$

Die Eigenschaften der kenoarithmetischen Multiplikation für den Fall zweier beliebiger Sequenzen lassen sich am besten anhand eines Beispiels erläutern. Sei $ks_1 = [1,2,2]$ und $ks_2 = [1,2,1]$, dann ist offensichtlichtlich die Länge der resultierenden Sequenz(en):

$$\underbrace{|ks_1| + |ks_1| + \cdots + |ks_1|}_{|ks_2|=3 \text{ mal}} = 9.$$

Da ○ das neutrale Element der Multiplikation ist, wird an den ersten drei Stellen der insgesamt neun resultierenden die erste Sequenz ks_1 identisch wiederholt:

[○,△,△, _,_,_, _,_,_].

Würde nun das zweite Kenogrammsymbol von ks_2, △, als isolierbare, rein quantitative arithmetische Einheit interpretiert, müßten auch die Plätze 4-6 mit einer identischen Kopie von ks_1 belegt werden:

[○,△,△, ○,△,△, _,_,_].

Da aber Kenogramme innerhalb einer Kenogrammsequenz nicht als isolierbare Atomzeichen betrachtet werden können, sondern stets einen ausdifferenzierten Formaspekt der Gesamtstruktur wiederspiegeln, muß dies auch von der kenoarithmetischen Multiplikation berücksichtigt werden. Die Belegung der Plätze 4-6 muß also sowohl die interne Struktur von ks_1 als auch die Gestalt von ks_2 abbilden, in der der erste Platz mit ○, der zweite hingegen mit einem anderen Kenogrammsymbol △ belegt ist. Wir erhalten folgende Möglichkeiten:

[○,△,△, △,○,○, _,_,_]
[○,△,△, △,□,□, _,_,_]
[○,△,△, □,★,★, _,_,_].

Nicht erlaubt sind hingegen alle Kombinationen, die an der vierten Stelle ein ○ oder auf den Plätzen 5 und 6 ein △ aufweisen, weil dies der Struktur von ks_2 widerspräche.. Da der dritte Platz von ks_2 nun wieder von einem ○ belegt ist, müssen die Stellen 7-9 mit den gleichen Kenogrammen wie die ersten drei Plätze belegt werden:

[○,△,△, △,○,○, ○,△,△]
[○,△,△, △,□,□, ○,△,△]
[○,△,△, □,★,★, ○,△,△].

Wir können also festhalten:

$kmul([1,2,2],[1,2,1]) = \{[1,2,2,2,1,1,1,2,2],[1,2,2,2,3,3,1,2,2],[1,2,2,3,4,4,1,2,2]\}$

Definition 4.16 (Trito-arithmetische Multiplikation) *Die Menge aller möglichen Produkte zweier Kenogrammsequenzen a, b wird bestimmt durch die Funktion* `kmul a b`*:*

```
fun kmul [] b = [[]]
   |kmul a [] = [[]]
   |kmul a [1] = [a]
   |kmul [1] b = [b]
   |kmul a b =
     let
        fun makeEN a k [] = []
           |makeEN a k kyet=
             flat(map
                  (fn mem => map
                              (fn p => (((firstocc mem b)-1)*(length a)+p,
                                       p,
                                       if (k=mem) then E
                                       else N))

                              (nlist(length a)))

                   (rd kyet));

        fun kmul1 a nil used res = res
           |kmul1 a (hd::tl) used res =
              kmul1 a tl (hd::used)
                 (flat(map (fn x => kligate x a
                                       (makeEN a hd used))
                           res));
     in
        kmul1 a b [] [[]]
     end;


- kmul [1,2] [1,2];
> val it = [[1,2,2,1],[1,2,3,1],[1,2,2,3],[1,2,3,4]] : int list list

- kmul [1,2,2] [1,2,3,1];
> val it =
        [[1,2,2,2,1,1,3,4,4,1,2,2],[1,2,2,3,1,1,2,3,3,1,2,2],
         [1,2,2,3,1,1,2,4,4,1,2,2],[1,2,2,3,1,1,4,3,3,1,2,2],
```

```
[1,2,2,3,1,1,4,5,5,1,2,2],[1,2,2,2,3,3,3,1,1,1,2,2],
[1,2,2,2,3,3,4,1,1,1,2,2],[1,2,2,2,3,3,3,4,4,1,2,2],
[1,2,2,2,3,3,4,5,5,1,2,2],[1,2,2,3,4,4,2,1,1,1,2,2],
[1,2,2,3,4,4,4,1,1,1,2,2],[1,2,2,3,4,4,5,1,1,1,2,2],
[1,2,2,3,4,4,2,3,3,1,2,2],[1,2,2,3,4,4,2,5,5,1,2,2],
[1,2,2,3,4,4,4,3,3,1,2,2],[1,2,2,3,4,4,5,3,3,1,2,2],
[1,2,2,3,4,4,4,5,5,1,2,2],[1,2,2,3,4,4,5,6,6,1,2,2]]
: int list list
```

Der Ablauf dieser Operation am besten anhand einer Matrixdarstellung veranschaulicht werden. Für die Multiplikation (`kmul [1,2,2] [1,2,3,1]`) ergibt sich folgendes Schema:

`kmul`	1	2	3	1
1				
2				
2				

Die erste und letzte Spalte ergeben sich als einfache Wiederholungen von [1,2,2]:

`kmul`	1	2	3	1
1	1			1
2	2			2
2	2			2

Die zweite Spalte muß eine Wiederholung von [1,2,2] enthalten, die die gleichen Kenogrammsymbole in anderer Verteilung oder neue Kenogrammsymbole benutzt:

`kmul`	1	2	3	1
1	1	2		1
2	2	1		2
2	2	1		2

`kmul`	1	2	3	1
1	1	3		1
2	2	1		2
2	2	1		2

`kmul`	1	2	3	1
1	1	3		1
2	2	4		2
2	2	4		2

`kmul`	1	2	3	1
1	1	2		1
2	2	3		2
2	2	3		2

Da auch an der dritten Position von [1,2,3,1] ein vorher noch nicht benutztes Kenogrammsymbol auftaucht, muß die dritte Spaltung eine neue Verteilung der schon benutzen Kenogramme oder aber die Hinzufügung weiterer Symbole enthalten, z.B.:

`kmul`	1	2	3	1
1	1	3	2	1
2	2	1	3	2
2	2	1	3	2

`kmul`	1	2	3	1
1	1	3	2	1
2	2	1	4	2
2	2	1	4	2

`kmul`	1	2	3	1
1	1	3	4	1
2	2	1	5	2
2	2	1	5	2

`kmul`	1	2	3	1
1	1	3	4	1
2	2	1	3	2
2	2	1	3	2

Die Funktion `makeEN` erzeugt eine ϵ/ν-Indizierung für die Verkettung der einzelnen Wiederholungen von [1,2,2]. Diese Indizierung bildet anhand der oben beschriebenen Gleichheits- und Verschiedenheitsstruktur die Gestalt von [1,2,3,1] auf die Spalten der Matrix ab.

Die Mehrdeutigkeit dieser allgemeinen Multiplikation läßt sich wiederum dahingehend interpretieren, daß eine Vielzahl von jeweils eindeutigen kenogrammatischen Multiplikationen $kmul_i$ existieren, die nach dem oben angegebenen Verfahren indiziert sind. Für alle Multiplikationen $kmul_i$ gilt dann:

$$|kmul_i(a, b)| = |a| \times |b|.$$

Auch für die kenogrammatische Multiplikation erweist sich die klassische Multiplikation als Reduktion auf den quantitativen Aspekt der Kenoarithmetik.

Kapitel 5

Morphogrammatik

Es schlug einer,
ein Lehrer,
mit dem Stock auf den Tisch:

Zu sterben, das ist Grammatik!

Ich lachte.

Nimm den Leib
wörtlich, das Wort
leiblich.

Ich lachte.

Ich starb.

E. Meister

5.1 Einführung

Günthers Entwurf einer transklassischen Logik ist dadurch gekennzeichnet, daß er die zweiwertige Logik als Formalisierung der Erkenntnissituation eines einzelnen Subjektes in der (für dieses Subjekt) objektiven Welt anerkennt, und dieses Verhältnis im Gegensatz zu klassischen Erweiterungen unangetastet läßt. Die Polykontexturale Logik ergänzt vielmehr die monokontexturale klassische Logik um eine beliebige Vielzahl weiterer logischer Kontexturen und beschreibt deren komplexe Wechselwirkungen. Wie bereits in Kapitel 2.3 erläutert, läßt sich die Vermittlung und Wechselwirkung der distribuierten Kontexturen nicht innerhalb eines klassischen (d.h. an die Axiomatik der Identität, des verbotenen Widerspruchs und des Tertium non datur gebundenen) Formalismusabbilden[1]. Günther führt deshalb eine prä–logische Theorie logischer Operativität ein, die es unternimmt, einen Standpunkt einzunehmen und formal zu beschreiben, der der phänomenologisch beobachteten Aufspaltung der individuellen Realität in Subjekt und Objekt vorangeht. Als Versuch einer Formalisierung dieser Perspektive lassen sich die Kenogrammatik und — als auf die Logik bezogene Theorie — die Morphogrammatik verstehen.
„*Die Morphogrammatik [beschreibt] eine Strukturschicht, in der die Differenz zwischen Subjektivität und Objektivität erst etabliert wird und deshalb dort noch nicht*

[1] vgl. Kapitel 9.2.

vorausgesetzt werden kann.“[2] Dieser prä–logische Charakter der Morphogrammatik ermöglicht die formal widerspruchsfreie Abbildung der gegen die Axiomatik der Logiken verstoßenden Vermittlung mehrerer Logiken in einem polykontexturalen Verbund. Die Morphogrammatik abstrahiert von der Wertigkeit logischer Operatoren und notiert nur deren kenogrammatischen TNF–Gestaltaspekt. Von der Designation von Wahrheitswerten, die die Kernaufgabe der Aussagenlogik ist, wird durch diesen Schritt vollkommen abgesehen. Durch diese Abstraktion wird die Wahrheitswertwidersprüchlichkeit einer auf dem ontologisch–logisch–semiotischen Identitätsprinzip beruhenden Kontexturvermittlung umgangen.
Durch Operationen auf dieser Gestaltebene des Logischen, die wir im vorigen Kapitel als Kenogrammatik einführten, konstruiert Günther komplexe morphogrammatische Systeme, denen er wiederum — in Form der Stellenwertlogik oder der PKL — logische Interpretationen zuweist.
Die auf ihrer morphogrammatischen Konstruktion beruhende Mehrwertigkeit der PKL unterscheidet sich fundamental von den konservativ erweiterten mehrwertigen Logiken, da die zusätzlichen Werte nicht bestimmte zusätzliche *intra-kontexturale* Unterscheidungen, sondern die *inter-kontexturale* Operativität der polykontexturalen Systemkonzeption designieren.
In diesem Kapitel soll zunächst Günthers Herleitung der Morphogrammatik (MG) aus dem klassischen Aussagenkalkül rekonstruiert werden, um die Anlehnung der Morphogrammatik an aussagenlogische Systeme zu dokumentieren. Um die formale Fundierung der MG durch die Kenogrammatik zu gewährleisten, wird dann die Herleitung aus der KG skizziert. Im Hauptteil des Kapitels werden morphogrammatische Strukturen (5.3) und Operationen (5.4) eingeführt, die die Grundlage für Günthers Konstruktion von Stellenwertlogik und PKL bilden.

5.2 Einführung der Basismorphogramme

5.2.1 Dekonstruktion der Aussagenlogik

Grundlegende Objekte der Morphogrammatik sind die *Morphogramme*. Günthers Herleitung der Morphogramme beginnt mit der Wertabstraktion der klassischen aussagenlogischen Operatoren.
Die Aussagenlogik ist prinzipiell zweiwertig, jede Aussage ist entweder *wahr* oder *falsch*: sie ist genau eines der beiden (Satz der Identität), sie kann nicht zugleich wahr und falsch sein (Satz vom verbotenen Widerspruch) und kann keinen anderen Wert annehmen (Satz vom ausgeschlossenen Dritten). Der Wahrheitswert zusammengesetzer Aussagen kann ohne jede inhaltliche Betrachtung allein aus den Wahrheitswerten der Teilaussagen und deren logischer Verknüpfung abgeleitet werden. Ausgangspunkt der Aussagenlogik ist die Definition von Wahrheitswertfunktionen (auch: Funktoren, Junktoren, Konnektoren, Operationen, Operatoren u.ä.) die alle möglichen Kombinationen von Wahrheitswerten auf die Menge {w,f}[3] abbildet. Neben der einstelligen Negation $\neg$ gibt es sechzehn binäre Operatoren, die jede der vier kombinatorisch möglichen Wahrheitswertkombinationen zweier Aussagen p, q jeweils auf einen der beiden Wahrheitswerte w oder f abbildet. Der aus dieser Abbildung resultierende Verlauf von vier Wahrheitswerten definiert und repräsentiert die Operation. Die aussagenlogischen Operatoren werden üblicherweise als *Wahrheitstafel* dargestellt. Zum

[2] [Gue80], Bd.1, S. 216.
[3] Oder auch: $\{T, F\}, \{0, 1\}, \{1, 2\}$.

Beispiel sind Konjunktion $\wedge$ und Disjunktion $\vee$ folgendermaßen definiert:

p	q	$p \wedge q$	$p \vee q$
w	w	w	w
w	f	f	w
f	w	f	w
f	f	f	f

Da für zwei Aussagen, die jeweils wahr oder falsch sind, $2^2 = 4$ Wahrheitswertkombinationen auftreten und jede dieser Kombinationen auf einen der beiden Wahrheitswerte abgebildet werden kann, ergeben sich $4^2 = 16$ verschiedene binäre Funktionen $\circ_1 \ldots \circ_{16}$ (Abbildung 5.1).

p	q	$\circ_1$	$\circ_2$	$\circ_3$	$\circ_4$	$\circ_5$	$\circ_6$	$\circ_7$	$\circ_8$	$\circ_9$	$\circ_{10}$	$\circ_{11}$	$\circ_{12}$	$\circ_{13}$	$\circ_{14}$	$\circ_{15}$	$\circ_{16}$
w	w	w	f	w	f	w	f	w	f	f	w	w	f	w	f	w	f
w	f	w	f	w	f	w	f	f	w	w	f	f	w	w	f	f	w
f	w	w	f	w	f	f	w	w	f	w	f	f	w	w	f	f	w
f	f	w	f	f	w	w	f	w	f	w	f	f	w	f	w	w	f

Abbildung 5.1:
Die Wahrheitswertfunktionen der Aussagenlogik

Um sinnvolle Aussagen über die *Gestalt* — die Anordnung der Wahrheitswerte zueinander innerhalb des Wertverlaufs — der Operationen treffen zu können, wird eine standardisierte Kombinationsfolge für die Wahrheitswerte der Variablen p und q per Konvention festgelegt:

p	q
w	w
w	f
f	w
f	f

Diese Konvention erlaubt es, auf die sechzehn Operatoren allein durch den 'Operatorenrumpf' Bezug zu nehmen (Abbildung 5.2).

$\circ_1$	$\circ_2$	$\circ_3$	$\circ_4$	$\circ_5$	$\circ_6$	$\circ_7$	$\circ_8$	$\circ_9$	$\circ_{10}$	$\circ_{11}$	$\circ_{12}$	$\circ_{13}$	$\circ_{14}$	$\circ_{15}$	$\circ_{16}$
w	f	w	f	w	f	w	f	f	w	w	f	w	f	w	f
w	f	w	f	w	f	f	w	w	f	f	w	w	f	f	w
w	f	w	f	f	w	w	f	w	f	f	w	w	f	f	w
w	f	f	w	w	f	w	f	w	f	f	w	f	w	w	f

Abbildung 5.2:
Der Operatorenrumpf der Aussagenlogik

Offensichtlich bestehen diese 16 Operationen aus acht 'Negationspaaren' ($\circ_1$, $\circ_2$) ,($\circ_3$, $\circ_4$), ..., ($\circ_{15}$, $\circ_{16}$), die jeweils die Negation voneinander sind. Diese Eigenschaft eröffnet anschaulich die Möglichkeit zu einer Wertabstraktion. Die beiden Operationen eines Negationspaares sind aussagenlogisch und zeichentheoretisch wohlunterschieden.

Wird jedoch nur die Struktur von Identität und Differenz im Belegungsmuster der jeweils vier Stellen betrachtet, sind sie (trito-)äquivalent. Für die sechzehn aussagenlogischen Funktionen läßt sich also folgende Tritoabstraktion vornehmen (Abbildung 5.3):

Operator	$\circ_1$	$\circ_2$	$\circ_3$	$\circ_4$	$\circ_5$	$\circ_6$	$\circ_7$	$\circ_8$	$\circ_9$	$\circ_{10}$	$\circ_{11}$	$\circ_{12}$	$\circ_{13}$	$\circ_{14}$	$\circ_{15}$	$\circ_{16}$
	1	2	1	2	1	2	1	2	2	1	1	2	1	2	1	2
Wahr-	1	2	1	2	1	2	2	1	1	2	2	1	1	2	2	1
heitswert-	1	2	1	2	2	1	1	2	1	2	2	1	1	2	2	1
verlauf	1	2	2	1	1	2	1	2	1	2	2	1	2	1	1	2
TNF	⇓		⇓		⇓		⇓		⇓		⇓		⇓		⇓	
	○		○		○		○		○		○		○		○	
Morpho-	○		○		○		△		△		○		△		△	
gramm	○		○		△		○		△		△		○		△	
	○		△		○		○		△		△		△		○	

Abbildung 5.3:
Die acht klassischen Morphogramme

Diese acht Strukturen nennt Günther 'klassische' Morphogramme: „*In order to stress the logical significance of these patterns, and to point out that they, and not their actual occupancies, represent invariants in any logic we shall give them a special name. These patterns will be called 'morphograms', since each of them represents an individual structure or 'Gestalt'*"[4]. Die acht Morphogramme repräsentieren den von logischer Wertigkeit abstrahierten Gestaltaspekt der Aussagenlogik. Die von Günther analysierte Schwäche der Aussagenlogik, lediglich rein objektive Ausssagen formulieren zu können (und alle auf Subjektivität oder auf das Subjekt/Objekt Kontinuum bezogenen Aussagen auszuschließen), drückt sich nach Günther in der 'morphogrammatischen Unvollständigkeit' der Aussagenlogik aus. Betrachtet man nämlich Sequenzen von vier Plätzen allein vom Gesichtspunkt der morphogrammatischen Gestalthaftigkeit, sind nicht nur acht sondern fünfzehn verschiedene Morphogramme kombinatorisch erzeugbar (Abbildung 5.4, vgl. Abbildung 3.10 auf Seite 68):

○	○	○	○	○	○	○	○	○	○	○	○	○	○	○
○	○	○	○	○	△	△	△	△	△	△	△	△	△	△
○	○	△	△	△	○	○	○	△	△	△	□	□	□	□
○	△	○	△	□	○	△	□	○	△	□	○	△	□	⋆

Abbildung 5.4:
Die 15 Basismorphogramme

Die 'transklassisch' erweiterte morphogrammatischen Basis, für die es auf der Ebene der klassischen Aussagenlogik keinerlei Interpretationsmöglichkeit gibt, versucht Günther in seinen Arbeiten vollständig in einen logischen Formalismus (die PKL) zu integrieren (vgl. Kapitel 9).

[4] [Gue80b], p. 348.

5.2.2 Morphogramme als Umkehrungen logischer Funktionen

In einer gemeinsam mit G. Günther erstellten Arbeit schlägt v. Foerster [Foe70] eine Definition der Morphogramme als Umkehrrelationen logischer Funktionen vor, die hier aus Gründen der historischen Vollständigkeit und als Hinführung zur Fundierung der MG in der Kenogrammatik kurz skizziert werden soll.
Die Umkehrrelation einer Abbildung $f : A \longrightarrow B$ wird mit $f^{-1} : B \longrightarrow A$ bezeichnet. Diese Relationen ist im Allgemeinen keine Abbildung, da sie nicht notwendigerweise eindeutig ist. f^{-1} ist genau dann eine (bijektive) Abbildung, wenn f bijektiv ist, wobei $f^{-1} \circ f = 1_A$ und $f \circ f^{-1} = 1_B$ (1_M bezeichnet die Identitätsabbildung über der Menge M). Diese Definition der Umkehrabbildung wendet v. Foerster an, um zu zeigen „*that Günther's kenograms are nothing else but the original dependent variables becoming independent after inversion.*“ Zunächst führt er eine mathematische Notation logischer Funktionen ein. $X_n = \{x_1, \ldots, x_n\}$ bezeichne die unabhängige Variable in einer logischen Funktion:

$$Y = \mathrm{F}(X_n),$$

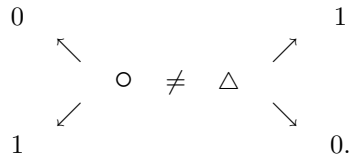
wobei X_n durch ein n-Tupel unabhängiger, elementarer Aussagevariablen x_i ($i \in \{1, \ldots, n\}$) repräsentiert ist. Der Wertebereich dieser Variablen x_i hängt von der Wertigkeit der betrachteten Logik ab, in einer m-wertigen Logik kann jede Variable x_i genau m verschiedene Werte annehmen. Für die Gesamtvariable X_n existieren demzufolge m^n verschiedene Wertbelegungen, die Menge $WB(X_n)$. Für jede dieser Belegungen ist eine logische Funktion definiert, die ihr einen bestimmten Wert $y_i \in \{0, \ldots, m-1\}$ zuordnet. Ein m-wertiges logisches System mit n unabhängigen Aussagevariablen besitzt folglich genau m^{m^n} verschiedene Funktionen. Die klassische aussagenlogische Funktion $Y = p \wedge q$ und ihre Negation $Y = \neg(p \wedge q)$ (oder kurz $Y = p \overline{\wedge} q$) werden in dieser Notation wie folgt dargestellt:

X_2		$\mathrm{F}(X_2)$	$\mathrm{F}(X_2)$	M
x_1	x_2	$x_1 \wedge x_2$	$x_1 \overline{\wedge} x_2$	
0	0	0	1	○
0	1	1	0	△
1	0	1	0	△
1	1	1	0	△

Die in der letzten Spalte angegebene, beiden Funktionen entsprechende morphogrammatische Struktur M bezieht sich offensichtlich nicht auf eine bestimmte Wertbelegung der resultierenden Funktionswerte $\mathrm{F}(X_2)$, da ○ und △ sowohl mit 0 als auch mit 1 (jedoch nicht gleichzeitig) belegt werden können. Die Wertevertauschung der Funktion $\wedge$ durch die Negation $\neg$ beeinflußt also weder die morphogrammatische Struktur der Funktion noch die Werte der unabhängigen Variablen X_2. Innerhalb des Morphogramms M kann ein einzelnes Kenogramm durch beliebige logische Werte belegt werden und ist lediglich fest an eine bestimmten Teilmenge der Belegungen von X_n gebunden. Im obigen Beispiel ist etwa:

$$\begin{array}{lcl} ○ & \leftrightarrow & \{(0,0)\} \text{ und} \\ △ & \leftrightarrow & \{(0,1),(1,0),(1,1)\} \end{array}$$

Kenogramme in M können mit unterschiedlichen logischen Werten, ungleiche Kenogramme jedoch nicht mit gleichen Werten belegt werden:



Daher ist es legitim, Kenogramme als 'elementare Variablen' y_i der abhängigen Variable $Y = \{y_1, \ldots, y_r\}$ zu betrachten, wobei stets gilt:

$$y_i \neq y_j \text{ für } i \neq j, \quad \text{mit} \quad i, j \in \{1, \ldots, r\}.$$

$r \leq m$ gibt hier die Anzahl verschiedener Werte innerhalb von Y an.
Die Menge X_{n_i} aller n-Tupel t (Wertbelegungen für $X_n = (x_1, \ldots, x_n)$), die durch das gleiche Kenogramm k_i bezeichnet werden, ist definiert als:

$$X_{n_i} = \{t \in WB(X_n) | \mathrm{F}(t) = y_i\}.$$

Diese Beziehung kann auch notiert werden als:

$$X_{n_i} = \mathrm{F}^{-1}(y_i).$$

Im Falle des Morphogramms ○△△△, das die logische Funktionen $\wedge$ und $\overline{\wedge}$ repräsentiert, ergibt sich:

$$\begin{array}{rcll} \{(0,0)\} & = & \mathrm{F}^{-1}(○) & \\ \{(0,1),(1,0),(1,1)\} & = & \mathrm{F}^{-1}(△), & \text{zusammengefaßt:} \\ [\{(0,0)\};\{(0,1),(1,0),(1,1)\}] & = & \mathrm{F}^{-1}\ (○;△) & \end{array}$$

Diese Darstellung repräsentiert die beiden Umkehrrelationen der Funktionen $\wedge$ und $\overline{\wedge}$:

$$\begin{array}{rcl} [\{(0,0)\};\{(0,1),(1,0),(1,1)\}] & = & \mathrm{F}^{-1}(0;1) = (\wedge)^{-1} \\ [\{(0,0)\};\{(0,1),(1,0),(1,1)\}] & = & \mathrm{F}^{-1}(1;0) = (\overline{\wedge})^{-1}. \end{array}$$

Aus dem Beispiel kann die allgemeine Definition eines Morphogramms M einer logischen Funktion $Y = \mathrm{F}(X_n)$ abgeleitet werden:

$$M(\mathrm{F}(X_n)) = [X_{n_1}; X_{n_2}; \ldots; X_{n_r}] = \mathrm{F}^{-1}(y_1; y_2; \ldots; y_r)$$

mit $1 \leq r \leq m$.
Aus dieser Darstellung ergibt sich deutlich die Negationsinvarianz der Morphogramme, so daß v. Foerster mit seiner Untersuchung zu dem Schluß gelangt,: „*a morphogram represents the logical structure of a particular [logical] function plus all its negations.*“ Seine Definition der Teilmenge X_{n_i} von X_n bezüglich F entspricht der in 3.3.1 eingeführten Äquivalenzklassenbildung. Die Zusammenfassung dieser X_{n_i} in der Umkehrfunktion F^{-1} entspricht folglich der Quotientenstruktur $A/\mathrm{Kern\ F}$ (d.h. der Menge aller Äquivalenzklasssen $[a_i]_{\mathrm{Kern\ F}}$), die zur Definition der kenogrammatischen Äquivalenzrelationen und der Kenogrammkomplexionen benutzt wurde. Hiermit ist deutlich geworden, daß die Definition der Morphogramme auf dem Konzept der Kenogrammkomplexionen fußt. Die Herleitung der Basismorphogramme aus der Kenogrammatik wird im nächsten Abschnitt behandelt.

5.2.3 Herleitung aus der Kenogrammatik

Im Vorhergehenden wurden die Basismorphogramme anhand Günthers inhaltlicher Kritik der Aussagenlogik aus der Wertabstraktion des Aussagenkalküls und der kombinatorischen Ergänzung der morphogrammatischen Unvollständigkeit hergeleitet. Als weitere Veranschaulichung wurde v. Foersters Darstellung der Morphogramme als Umkehrrelationen logischer Funktionen vorgestellt.
Aufgrund der Proemialität von logischer Wertebene und kenogrammatischer Ebene besteht die Möglichkeit, die Morphogrammatik vom Standpunkt der Wertebene oder vom Standpunkt der Kenogrammatik aus darzustellen. Von der logischen Wertebene aus werden Morphogramme als bestimmte Äquivalenzklassen über Wertstrukturen logischer Operatoren definiert. In diesem Sinne stellen die Kenogramme, wie v. Foerster zeigt, 'elementare Variablen' y_i der abhängigen Variable $Y = \{y_1, \ldots, y_r\}$ dar, d.h. Repräsentanten von Werten. In dieser Perspektive erscheint die Morphogrammatik als Reduktion bzw. Abstraktion der logischen Wertebene.
Vom Standpunkt der Kenogrammatik wird die Morphogrammatik rein kenogrammatisch fundiert. In der Kenogrammatik stellen Morphogramme bestimmte Kenogrammkomplexionen, d.h. Differenzenstrukturen dar, wobei die einzelnen Kenogramme, (die nicht im Sinne der Semiotik als individuell unterscheidbare Atomzeichen isolierbar sind), keine Werte, Zeichen oder Variablen sein können. Die logischen Werte erscheinen als 'Konkretionen' oder 'Kristallisationen' der Kenogrammkomplexionen[5]. Im folgenden wird diese formale Fundierung der Morphogrammatik durch die Kenogrammatik mit der Einführung der Basismorphogramme als Kenogrammkomplexionen geleistet

Definition 5.1 (Basismorphogramme) *Ein Basismorphogramm mg ist eine Kenogrammsequenz der Länge 4,* $|mg| = 4$.

Es existieren daher:

$$Tcard(4) = \sum_{k=1}^{4} S(4,k) = 15$$

verschiedene Basismorphogramme. Diese 15 Kenogrammsequenzen der Länge 4 werden durch `Tcontexture(4)` berechnet (vgl. 3.3.3).

Definition 5.2 (Morphogrammatisches Operandensystem Q)
Das Operandensystem der Morphogrammatik ist die Menge $Q =$ `Tcontexture(4)`, *mit* $|Q| = 15$.

```
val Q = Tcontexture(4);
```

Ein einzelnes Morphogramm mg_i, mit $1 \leq i \leq 15$ wird durch die Funktion `mg(i)` berechnet:

```
exception Mg;
fun mg i = if i<1 orelse i>15 then raise Mg
           else pos i Q;
```

Dieser Zusammenhang ist in Abbildung 5.5 dargestellt.

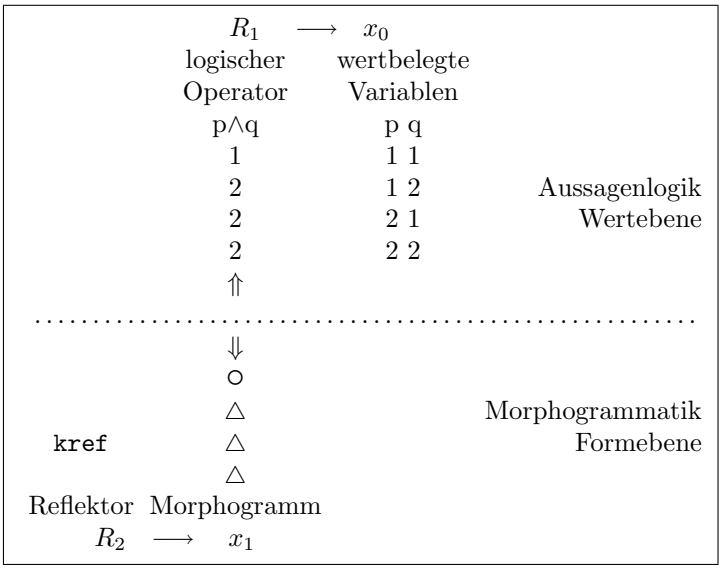
[5] [Kae74], S. 108f.

Q	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○
	○	○	○	○	○	△	△	△	△	△	△	△	△	△	△
	○	○	△	△	△	○	○	○	△	△	△	□	□	□	□
	○	△	○	△	□	○	△	□	○	△	□	○	△	□	★
mg_i	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

Abbildung 5.5:
Numerierung der Basismorphogramme

5.2.3.1 Zur Proemialität von Morphogrammatik und Aussagenlogik

Als Morphogrammatik wird nun jenes Teilgebiet der Kenogrammatik verstanden, das nur die Operationen auf den Basismorphogrammen untersucht. Da die wertabstrahierten Morphogramme gegenüber Negationsoperationen auf der aussagenlogischen Ebene invariant sind, kann auf ihnen nicht innerhalb der Logik, sondern nur in der MG operiert werden. Die durch morphogrammatische Operationen manipulierten Strukturen können nun, vermittelt durch eine Wertbelegung WB, wieder als aussagenlogische Operatoren innerhalb der Logik fungieren. So führt etwa die Wertbelsegung $WB(○)$=1, $WB(△)$=2 des Morphogramms mg_{10} = ○△△△ zu 1222, dem aussagenlogischen Operator der Konjunktion, siehe Abbildung (5.6).



⟶: R_i operiert auf x_{i-1}
⇓: Wertabstraktion
⇑: Wertbelegung

Abbildung 5.6:
Das proemielle Verhältnis von Aussagenlogik und MG

Innerhalb der Aussagenlogik fungiert R_1 als logischer *Operator* auf 2-Tupeln von Wahrheitswerten, x_0. Gleichzeitig ist die wertabstrahierte Kenogrammstruktur von R_1 innerhalb der MG lediglich das Resultat (und möglicher *Operand*) der kenogrammatischen Operation R_2. Das Verhältnis von Morphogrammatik und Aussagenlogik erweist sich nach diesen Überlegungen als ein weiteres Beispiel für die kenogrammatische Proemialrelation. Aufgrund der morphogrammatischen Unvollständigkeit des klassischen Aussagenkalküls, läßt sich jedoch noch keine vollständige Spiegelung der MG auf die Aussagenlogik durchführen, so daß die proemielle Simultaneität von logischer und morphogrammatischer Ebene hier noch nicht vollständig realisiert sein kann. Im Kapitel 9 wird die Einbettung der Stellenwertlogik und der PKL in die MG beschrieben, für die vollständige Entsprechungen morphogrammatischer und logischer Ebene besteht.

5.3 Morphogrammatische Komplexionen

Die im letzten Abschnitt definierten Basismorphogramme bilden das *Operandensystem* der Morphogrammatik, Q, das im nächsten Kapitel analysiert und klassifiziert wird. Alle komplexeren morphogrammatischen Strukturen werden aus den Basismorphogrammen aufgebaut. Mit Hilfe dieser morphogrammatischen Komplexionen fundiert Günther seine stellenwertlogischen und polykontexturalen Logikkonzeptionen. Die Kernidee dieser Konzeptionen besteht darin, die Basismorphogramme als 2×2-Matrizen aufzufassen, aus denen beliebige quadratische $n \times n$-Matrizen gebildet werden, welche wiederum zur Definition polykontexturaler logischer Operatoren benutzt werden.

Definition 5.3 (Wertmatrix) *Eine quadratische Matrix W^n:*

$$W^n = \begin{pmatrix} w_{11} & \cdots & w_{1n} \\ \vdots & \vdots & \vdots \\ w_{n1} & \cdots & w_{nn} \end{pmatrix}$$

deren $n \times n$ Matrixplätze w_{ij} mit beliebigen Werten aus einer Menge B belegt sind, heißt Wertmatrix.

Definition 5.4 (Linearform) *Die Abbildung einer Wertmatrix W^n auf eine $4\binom{n}{2}$-stellige KenogrammsequenzL(W):*

$$L(W^n) = TNF([\underbrace{w_{11}, w_{12}, w_{21}, w_{22}}_{\text{Stellen } 1\ldots4}, \ldots, \underbrace{w_{ii}\ ,\ \ w_{ij}\ ,\ \ w_{ji}\ ,\ \ w_{jj}}_{4(k(i,j)-1)+1\ldots4(k(i,j)-1)+4}, \ldots, \underbrace{w_{11}, w_{1n}, w_{n1}, w_{nn}}_{4\binom{n}{2}-3\ldots4\binom{n}{2}}])$$

heißt Linearform von W^n. Hierbei ist $k(i,j) = \binom{j}{2} - i + 1$, mit $i < j$, und $1 \leq k(i,j) \leq \binom{n}{2}$.

Implementiert wird L durch die ML-Funktion `LL`:

```
type 'a mat = 'a list list;

fun maufn m n=
  let
   fun combine it list= map (fn x=> it::x) list;
```

```
   fun maufn1 n []=[]
      |maufn1 0 l =[]
      |maufn1 1 l = map (fn x=> [x]) l
      |maufn1 p l =
         flat(map (fn x => combine x (maufn1 (p-1) (remove x l)))
                  l);
   fun remdups []=[]
      |remdups ([x,y]::tl)=
         if x=y then remdups tl
         else if member [y,x] tl then [x,y]::(remdups(remove [y,x] tl))
         else [x,y]::(remdups tl);
  in
   remdups(maufn1 n (nlist m))
  end;

fun matpos i j c=
   pos j (pos i c);

fun sort l=
  let
     exception Assoc;
     fun assoc n []= raise Assoc
        |assoc n ((k,pair)::tl)=
           if n=k then (k,pair)
           else assoc n tl;
  in
   map (fn k=> assoc k l)
       (nlist (length l))
  end;

fun k (i,j)=((j*(j-1)) div 2)-i+1;

fun subsystems n=
  sort(map (fn [i,j] => (k(i,j),[i,j]))
            (maufn n 2));

fun LL (w: 'a mat)=
  let
   val n = length w;
  in
   tnf(flat (map (fn (k,[i,j])=> [matpos i i w,matpos i j w,
                                  matpos j i w,matpos j j w])
            (subsystems n)))
  end;
```

Beispiel:

```
- LL [["w", "f", "ff"],
      ["f", "f", "ff"],
      ["ff","ff","ff"]];
> [1,2,2,2,2,3,3,3,1,3,3,3] : kseq
```

Die Umkehrfunktion L^{-1} von L bildet eine Linearform $L(W)$ auf die entsprechende Matrix W ab, $L^{-1}(L(W)) = W$. Implementiert wird L^{-1} durch die ML–Funktion

L_1:

```
fun L_1 kseq =
  let
   exception Subsystems;
   val n = 0.5+sqrt(0.25+real((length kseq) div 2));
   val n = if real(floor n) = n then floor n
           else raise Subsystems;
   val subs = subsystems n;
   val mat = nlistof n (nlistof n 0);
   fun set (i,j,x,mat)=
     let
      fun replacepos n list w=
       let
        exception Replace;
        fun replacepos1 n [] x m= raise Replace
           |replacepos1 n (hd::tl) x m=
              if n=m then x::tl
              else hd::(replacepos1 n tl x (m+1));
      in
       replacepos1 n list w 1
      end;
     in
      replacepos i mat (replacepos j (pos i mat) x)
     end;
   fun kstomm [] subs mat=mat
      |kstomm (ii::ij::ji::jj::restks)
              ((k,[i,j])::restpairs)
              mat=
         kstomm restks
                restpairs
                (set (i,i,ii,
                (set (i,j,ij,
                (set (j,i,ji,
                (set (j,j,jj, mat))))))))
  in
   kstomm kseq subs mat
  end;
```

Definition 5.5 (Tritonormalform TNF_{MM})
Die Tritonormalform $TNF_{MM}(W^n)$ ist die Abbildung einer $n \times n$ Matrix W^n auf ihren Tritorepräsentanten $L^{-1}(L(W^n))$:

```
fun TNFMM w = L_1(LL(w));
```

Beispiel:

```
- TNFMM [["w", "f", "ff"],
         ["f", "f", "ff"],
         ["ff","ff","ff"]];
> [[1,2,3],[2,2,3],[3,3,3]] : keno mat
```

Definition 5.6 (Morphogrammatrix) *Eine Wertmatrix W^n heißt Morphogrammatrix, gdw. sie in TNF ist: $W^n = TNF_{MM}(W^n)$.*

Um Morphogrammatrizen von Wertmatrizen zu unterscheiden, wird im Folgenden eine $n \times n$-Morphogrammatrix stets mit Q^n bezeichnet.

Beispiel: Die Matrix

$$W = \begin{pmatrix} 1 & 4 & 3 \\ 1 & 3 & 2 \\ 1 & 3 & 4 \end{pmatrix}$$

ist keine Morphogrammatrix, da:

$$TNF_{MM}(W) = \begin{pmatrix} 1 & 2 & 3 \\ 1 & 3 & 4 \\ 1 & 3 & 2 \end{pmatrix}$$

Die fünfzehn Basismorphogramme lassen sich entsprechend als Q^2-Matrizen darstellen:

$$mg_1 = \begin{pmatrix} 1 & 1 \\ 1 & 1 \end{pmatrix} \quad mg_2 = \begin{pmatrix} 1 & 1 \\ 1 & 2 \end{pmatrix} \quad mg_3 = \begin{pmatrix} 1 & 1 \\ 2 & 1 \end{pmatrix} \quad mg_4 = \begin{pmatrix} 1 & 1 \\ 2 & 2 \end{pmatrix} \quad mg_5 = \begin{pmatrix} 1 & 1 \\ 2 & 3 \end{pmatrix}$$

$$mg_6 = \begin{pmatrix} 1 & 2 \\ 1 & 1 \end{pmatrix} \quad mg_7 = \begin{pmatrix} 1 & 2 \\ 1 & 2 \end{pmatrix} \quad mg_8 = \begin{pmatrix} 1 & 2 \\ 1 & 3 \end{pmatrix} \quad mg_9 = \begin{pmatrix} 1 & 2 \\ 2 & 1 \end{pmatrix} \quad mg_{10} = \begin{pmatrix} 1 & 2 \\ 2 & 2 \end{pmatrix}$$

$$mg_{11} = \begin{pmatrix} 1 & 2 \\ 2 & 3 \end{pmatrix} \quad mg_{12} = \begin{pmatrix} 1 & 2 \\ 3 & 1 \end{pmatrix} \quad mg_{13} = \begin{pmatrix} 1 & 2 \\ 3 & 2 \end{pmatrix} \quad mg_{14} = \begin{pmatrix} 1 & 2 \\ 3 & 3 \end{pmatrix} \quad mg_{15} = \begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix}.$$

Definition 5.7 (Morphogrammkette) *Eine lineare Liste von* $\binom{n}{2}$ *Basismorphogrammen* $[M_1, \ldots, M_{\binom{n}{2}}]$ *heißt Morphogrammkette.*

5.4 Morphogrammatische Operationen

In einer Morphogrammatrix stehen die Morphogramme der einzelnen Subsysteme in direktem Zusammenhang innerhalb *einer* Kenogrammkomplexion, so daß semiotische Gleichheit eines Kenogrammsymbols des Subbsystems, $k_i \in S_i$ mit einem Kenogrammsymbol eines anderen Subsystems, $k_j \in S_j$: $k_i \equiv_{\text{sem}} k_j$ auch eine kenogrammatische ϵ-Beziehung zwischen k_i und k_j impliziert. In einer Morphogrammkette hingegen sind die einzelnen Morphogramme voneinander isoliert, so daß die semiotische Gleichheit $k_i \equiv_{\text{sem}} k_j$ hier keine ϵ-Beziehung zwischen den Kenogrammsymbolen impliziert. Die Asymmetrie zwischen Morphogrammketten und Morphogrammmatrizen führt bei Abbildungen zwischen diesen Strukturen zu interessanten Ein-mehrdeutigkeiten. In diesem Abschnitt werden als Prototypen solcher Abbildungen die Komposition und Dekomposition eingeführt.

5.4.1 Die Dekomposition von Morphogrammatrizen

Definition 5.8 (Zerlegung) *Die Zerlegung* $Z(Q^n)$ *einerMorphogrammatrix* Q^n *mit* $n \times n$ *Stellen,*

$$Q^n = \begin{pmatrix} q_{11} & \cdots & q_{1n} \\ \vdots & \vdots & \vdots \\ q_{n1} & \cdots & q_{nn} \end{pmatrix},$$

bildet Q^n auf eine Zerlegungsfolge $Z(Q^n) = [M_1, \ldots, M_{\binom{n}{2}}]$ ab. Wobei

$$M_{k(i,j)} = \begin{pmatrix} q_{ii} & q_{ij} \\ q_{ji} & q_{jj} \end{pmatrix},$$

mit $i < j$ und $k(i,j) = \binom{j}{2} - i + 1$.

Definition 5.9 (Dekomposition) *Die Zerlegung einer Morphogrammatrix Q^n, $Z(Q^n) = [M_1, \ldots, M_{\binom{n}{2}}]$ mit anschließender TNF-bildung der einzelnen M_k heißt Dekomposition, $Dek(Q^n) = [TNF(M_1), \ldots, TNF(M_{\binom{n}{2}})]$. $Dek(Q^n)$ ist eine Morphogrammkette.*

Beispiel: Es soll eine 4×4 Matrix Q_1 dekomponiert werden:

$$Q_1 = \begin{pmatrix} 1 & 2 & 3 & 4 \\ 2 & 2 & 3 & 4 \\ 3 & 3 & 3 & 4 \\ 4 & 4 & 4 & 4 \end{pmatrix}.$$

Da $n = 4$, sind $\binom{4}{2} = 6$ Paarbildungen i, j möglich:

$$(1,2), (1,3), (1,4), (2,3), (2,4), (3,4).$$

Die Werte $k(i,j)$ für die sechs Tupel sind:

$$\begin{aligned} k(1,2) &= \binom{2}{2} - 1 + 1 &&= 1 \\ k(1,3) &= \binom{3}{2} - 1 + 1 &&= 3 \\ k(1,4) &= \binom{4}{2} - 1 + 1 &&= 6 \\ k(2,3) &= \binom{3}{2} - 2 + 1 &&= 2 \\ k(2,4) &= \binom{4}{2} - 2 + 1 &&= 5 \\ k(3,4) &= \binom{4}{2} - 3 + 1 &&= 4. \end{aligned}$$

Diese Tupel entsprechen den sechs Subsystemen $S_1 : (1,2), S_2 : (2,3), S_3 : (1,3), S_4 : (3,4), S_5 : (2,4)$ und $S_6 : (1,4)$ (jeweils durch Einrahmung markiert) :

$S_1 : (1,2)$	1	2	3	4
1	**1**	**2**	3	4
2	**2**	**2**	3	4
3	3	3	3	4
4	4	4	4	4

$S_2 : (2,3)$	1	2	3	4
1	1	2	3	4
2	2	**2**	**3**	4
3	3	**3**	**3**	4
4	4	4	4	4

$S_3:(1,3)$	1	2	3	4
1	[1]	2	[3]	4
2	2	2	3	4
3	[3]	3	[3]	4
4	4	4	4	4

$S_4:(3,4)$	1	2	3	4
1	1	2	3	4
2	2	2	3	4
3	3	3	[3]	[4]
4	4	4	[4]	[4]

$S_5:(2,4)$	1	2	3	4
1	1	2	3	4
2	2	[2]	3	[4]
3	3	3	3	4
4	4	[4]	4	[4]

$S_6:(1,4)$	1	2	3	4
1	[1]	2	3	[4]
2	2	2	3	4
3	3	3	3	4
4	[4]	4	4	[4]

Die aus sechs Morphogrammen bestehende Dekompositionsfolge $Dek(Q_1) = [M_1, \ldots, M_6]$ wird aus der Kette der Subsystemmorphogramme gebildet:

$$Dek(Q_1) = [M_{k(1,2)}, M_{k(2,3)}, M_{k(1,3)}, M_{k(3,4)}, M_{k(2,4)}, M_{k(1,4)}].$$

Wobei

$$\begin{aligned}
M_{k(1,2)} &= TNF_{MM}(\begin{pmatrix} 1 & 2 \\ 2 & 2 \end{pmatrix}) = \begin{pmatrix} 1 & 2 \\ 2 & 2 \end{pmatrix} \\
M_{k(2,3)} &= TNF_{MM}(\begin{pmatrix} 2 & 3 \\ 3 & 3 \end{pmatrix}) = \begin{pmatrix} 1 & 2 \\ 2 & 2 \end{pmatrix} \\
M_{k(1,3)} &= TNF_{MM}(\begin{pmatrix} 1 & 3 \\ 3 & 3 \end{pmatrix}) = \begin{pmatrix} 1 & 2 \\ 2 & 2 \end{pmatrix} \\
M_{k(3,4)} &= TNF_{MM}(\begin{pmatrix} 3 & 4 \\ 4 & 4 \end{pmatrix}) = \begin{pmatrix} 1 & 2 \\ 2 & 2 \end{pmatrix} \\
M_{k(2,4)} &= TNF_{MM}(\begin{pmatrix} 2 & 4 \\ 4 & 4 \end{pmatrix}) = \begin{pmatrix} 1 & 2 \\ 2 & 2 \end{pmatrix} \\
M_{k(1,4)} &= TNF_{MM}(\begin{pmatrix} 1 & 4 \\ 4 & 4 \end{pmatrix}) = \begin{pmatrix} 1 & 2 \\ 2 & 2 \end{pmatrix} = mg_{10}.
\end{aligned}$$

Also ist $Dek(Q_1) = [mg_{10}, mg_{10}, mg_{10}, mg_{10}, mg_{10}, mg_{10}]$.

Implementiert wird die Dekomposition von Morphogrammatrizen durch die Funktion `decompose(mm)`:

```
type mg = kseq;
type mgchain = mg list;

fun makemg i j c=
   tnf [matpos i i c,
        matpos i j c,
        matpos j i c,
        matpos j j c];

fun decompose (mm : mg mat)=
  let
```

```
 val n = length mm;
in   (map (fn (k,[i,j])=> makemg i j c)
      (subsystems n) : mgchain)
end;
```

5.4.2 Morphogrammatische Äquivalenz

Bei der Betrachtung der folgenden fünf Morphogrammatrizen $Q_1 \ldots Q_5$:

$$Q_1 = \begin{pmatrix} 1 & 2 & 2 \\ 2 & 1 & 2 \\ 2 & 2 & 1 \end{pmatrix} \qquad Q_2 = \begin{pmatrix} 1 & 2 & 3 \\ 2 & 1 & 2 \\ 3 & 2 & 1 \end{pmatrix}$$

$$Q_3 = \begin{pmatrix} 1 & 2 & 2 \\ 2 & 1 & 3 \\ 2 & 3 & 1 \end{pmatrix} \qquad Q_4 = \begin{pmatrix} 1 & 2 & 3 \\ 2 & 1 & 3 \\ 3 & 3 & 1 \end{pmatrix} \qquad Q_5 = \begin{pmatrix} 1 & 2 & 4 \\ 2 & 1 & 3 \\ 4 & 3 & 1 \end{pmatrix}$$

und ihren Linearformen:

$$\begin{aligned} L(Q_1) &= [1,2,2,1,1,2,2,1,1,2,2,1] \\ L(Q_2) &= [1,2,2,1,1,2,2,1,1,3,3,1] \\ L(Q_3) &= [1,2,2,1,1,3,3,1,1,2,2,1] \\ L(Q_4) &= [1,2,2,1,1,3,3,1,1,3,3,1] \\ L(Q_5) &= [1,2,2,1,1,3,3,1,1,4,4,1] \end{aligned}$$

fällt eine interessante Eigenschaft auf. Q_1 benutzt nur 2 verschiedene Kenogrammsymbole, Q_2, Q_3, Q_4 jeweils drei (sie gehören also zur gleichen Protoäquivalenzklasse) und Q_5 vier Kenogramme. Q_2, Q_3, Q_4 weisen jeweils die Partitionsstruktur $[3,4,2]$ auf, sie gehören somit auch der gleichen Deuteroklasse an. Alle fünf Strukturen sind jedoch Repräsentanten verschiedener Tritoäquivalenzklassen, da sie verschiedene Linearformen haben. Überraschend ist nun, daß alle fünf Matrizen Q_i, $1 \leq i \leq 5$, die gleiche Dekompositionsstruktur aufweisen:

$$Dek(Q_i) = [\begin{pmatrix} 1 & 2 \\ 2 & 1 \end{pmatrix}, \begin{pmatrix} 1 & 2 \\ 2 & 1 \end{pmatrix}, \begin{pmatrix} 1 & 2 \\ 2 & 1 \end{pmatrix}].$$

Definition 5.10 (Morphogrammatische Äquivalenz) *Zwei Morphogrammatrizen Q_1, Q_2 sind morphogrammatisch äquivalent, $Q_1 \equiv_{\mathrm{mg}} Q_2$, gdw. $Dek(Q_1) = Dek(Q_2)$.*

Definition 5.11 (Morphogrammatische Äquivalenzklasse)
Die Menge $MA(K) = \{Q_i | Dek(Q_i) = K, 1 \leq i \leq MP\}$ heißt morphogrammatische Äquivalenzklasse der Morphogrammkette K.

Definition 5.12 (Morphogrammatische Polysemie) *Die Anzahl der Elemente in $MA(K)$, $|MA(K)| = MP$ heißt* Polysemie *der Klasse $MA(K)$. Die Elemente von $MA(K)$ heißen* Polyseme *der Klasse.*

Diese Anzahl MP wird in Kapitel 8 für beliebige morphogrammatische Äquivalenzklassen bestimmt.

5.4.3 Die Komposition von Morphogrammketten

5.4.3.1 Die Vermittlungsbedingungen

Günthers Kompositionsverfahren komponiert jeweils $\binom{n}{2}$ Morphogramme zu einer $n \times n$ Matrix. Eine $n \times n$ Matrix enthält n^2 Kenogramme, die $\binom{n}{2}$ Morphogramme zusammen jedoch $4\binom{n}{2} = 2n^2 - 2n$ Kenogramme. Da für $n > 2$:

$$(2n^2 - 2n) - n^2 = n^2 - 2n > 0,$$

müssen sich bei der Komposition von Morphogrammen zu Morphogrammatrizen stets $n^2 - 2n$ Doppelbelegungen von Matrizenplätzen durch Kenogramme aus verschiedenen Morphogrammen (*Subsystemen*)ergeben. Günther löst diese Überdetermination der Matrizen durch *Vermittlungsbedingungen* zwischen den Subsystemen, die jeweils die Identifizierung bestimmter Kenogramme der Basismorphogramme innerhalb der Matrix angeben. Günthers Kompositionsverfahren schränkt die Auswirkungen der Vermittlungsbedingungen (VB) auf die Hauptdiagonalstellen der Subsystemmorphogramme, die auf den n Plätze der Hauptdiagonalen der $n \times n$-Matrix liegen, ein.

Beispiel: Im Falle einer 3×3 Matrix:

	1	2	3
1	○	○	○
2	○	○	○
3	○	○	○

können $\binom{3}{2} = 3$ Subsysteme $S_1 : (1,2), S_2 : (2,3), S_3 : (1,3)$ gebildet werden (durch ausgefüllte Kreise • markiert):

S_1	1	2	3
1	•	•	○
2	•	•	○
3	○	○	○

S_2	1	2	3
1	○	○	○
2	○	•	•
3	○	•	•

S_3	1	2	3
1	•	○	•
2	○	○	○
3	•	○	•

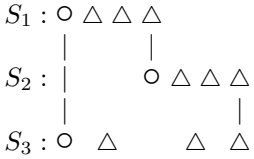
Überschneidungen zwischen den einzelnen Subsystemen finden nur auf den Hauptdiagonalplätzen statt:

$$\begin{aligned} S_1 \cap S_3 &= (1,1) \\ S_1 \cap S_2 &= (2,2) \\ S_2 \cap S_3 &= (3,3). \end{aligned}$$

Diese Überschneidungen werden durch die Vermittlungsbedingungen (im folgenden Diagramm durch senkrechte Striche | dargestellt) zwischen den S_i geregelt:

$$\begin{array}{lccc} S_1: & (1 & 2) & \\ & | & | & \\ S_2: & | & (2 & 3) \\ & | & & | \\ S_3: & (1 & & 3) \end{array}$$

Sollen die drei Subsysteme jeweils durch das Morphogramm mg_{10} =○△△△ belegt werden, ergibt sich die folgende Situation:



Um S_1, S_2, S_3 zu einer Matrix zusammenzufassen, müssen die Strukturen der Subsystem-Morphogramme beibehalten und gleichzeitig die Vermittlungsbedingungen erfüllt werden. Die Matrizen:

$[mg_9 mg_{10} mg_{10}]$	1	2	3
1	○	△	△
2	△	○	△
3	△	△	△

$[mg_{10} mg_1 mg_{10}]$	1	2	3
1	○	△	△
2	△	△	△
3	△	△	△

erfüllen zwar die Vermittlungsbedingungen zwischen S_1 und S_2, jedoch unter Verlust der mg_{10} Struktur in S_1 oder S_2. Sinnvoll ist also nur die Matrix:

$[mg_{10} mg_{10} mg_{10}]$	1	2	3
1	○	△	□
2	△	△	□
3	□	□	□

5.4.3.2 Komposition

Definition 5.13 (Komposition) *Die Komposition einer Morphogrammkette K,Kom(K) erzeugt die morphogrammatische Äquivalenzklasse MA(K).*

Beispiel: Die fünf Morphogrammatrizen $Q_1, \ldots, Q_5$ aus Abschnitt (5.4.2), Seite 97 sind paarweise morphogrammatisch äquivalent, da:

$$\begin{aligned} Dek(Q_1) = \quad Dek(Q_2) \quad &= Dek(Q_3) = \\ Dek(Q_4) = \quad Dek(Q_5) \quad &= [mg_9, mg_9, mg_9]. \end{aligned}$$

Da $Dek(Q_i) = [mg_9, mg_9, mg_9]$ für $1 \leq i \leq 5$, sind die fünf Matrizen Elemente von $MA([mg_9, mg_9, mg_9])$.

Die Komposition der Morphogrammkette $K_1 = [mg_9, mg_9, mg_9]$, $Kom(K)$, erzeugt gerade die Menge dieser fünf Matrizen:

$$Kom(K_1) = \{Q_1, Q_2, Q_3, Q_4, Q_5\}.$$

Wobei $|MA(K_1)| = 5$.

Die Komposition beliebiger Matrizen wird durch die ML-Funktion `Kom` implementiert:

```
fun makeEN (k,ik,jk) subsystems=
   flat(map (fn (l,[il,jl])=>
               if il=ik then [((l-1)*4+1,1,E)]
          else if il=jk then [((l-1)*4+1,4,E)]
          else if jl=jk then [((l-1)*4+4,4,E)]
          else if jl=ik then [((l-1)*4+4,1,E)]
          else [])
            (nfirst (k-1) subsystems));

fun kligs x ys ENS=
   flat (map (fn y=> kligate y x ENS)
             ys);

fun compose1 [] (subs:(int*int list) list) res subsystems=res
   |compose1 (hd::tl) ((k,[ik,jk])::subs) res subsystems=
     (compose1 tl
               subs
               (kligs hd
                      res
                      (makeEN (k,ik,jk) subsystems))
              subsystems);
fun Kom mk =
  let
   exception Compose;
   val n = 0.5+sqrt(0.25+real(2*(length mk)));
   val n = if real(floor n) = n then floor n
           else raise Compose;
   val subsystems =subsystems n;
  in
   map L_1
       (compose1 mk subsystems [[]] subsystems)
  end;
```

Dieser Algorithmus basiert auf folgender Grundidee. Da eine Q^n-Matrix stets aus $\binom{n}{2}$ Morphogrammen komponiert ist, ergibt sich für eine Morphogrammkette K der Länge l eine $n \times n$-Matrix, wobei $n = \frac{1}{2} + \sqrt{\frac{1}{4} + 2l}$. Die $\binom{n}{2}$ Subsysteme (i, j) werden anhand ihrer Position in der Morphogrammkette, $k(i, j)$, sortiert und an die Variable `subsystems` gebunden.
Für jedes Morphogramm M_k, das in die Matrix eingefügt werden soll, muß überprüft werden, ob sich in der Matrix durch M_k zu belegende Positionen mit Positionen der $k-1$ ersten Morphogramme überschneiden.

> **Beispiel:** Im Verlauf der Komposition einer sechsstelligen Morphogrammkette $K = [M_1, \ldots, M_6]$ zu einer 4×4 Matrix $Kom(K)$, sei das Subsystem $S_5 : (2, 4)$ in die Matrix zu integrieren.

$S_5 : (2,4)$	1	2	3	4
1				
2		○		○
3				
4		○		○

S_5 muß auf Überschneidungen mit den ersten vier Subsystemen untersucht werden.

$S_1 : (1,2)$	1	2	3	4
1	•	•		
2	•	•		○
3				
4		○		○

Da sich die Subsysteme S_1 und S_5 in der Position $(2,2)$ überschneiden, müssen sie aufgrund der Vermittlungsbedingungen innerhalb der Matrix an dieser Stelle das gleiche Kenogrammsymbol aufweisen.

$S_2 : (2,3)$	1	2	3	4
1				
2		•	•	○
3		•	•	
4		○		○

Auch S_2 überschneidet sich mit S_5 in $(2,2)$.

$S_3 : (1,3)$	1	2	3	4
1	•		•	
2		○		○
3	•		•	
4		○		○

Da sich S_3 und S_5 innerhalb der Matrix nicht überlagern, können sie unabhängig voneinander belegt werden.

$S_4 : (3,4)$	1	2	3	4
1				
2		○		○
3			•	•
4		○	•	•

S_4 und S_5 decken sich in der Position $(4,4)$, was wiederum für den Kompositionsvorgang als eine Identifizierung (ϵ) der beiden Subsysteme an dieser Stelle markiert werden muß.

Die Funktion `makeEN` nimmt für jedes Subsystem diese Markierung vor, indem sie eine ϵ/ν-Indizierung für die kenogrammatische Verschmelzungsoperation `kligate` erzeugt, mittels der die Funktion `klig` nur solche Kenogrammstrukturen erzeugt, die der einschränkenden ϵ/ν-Indizierung genügen.

5.4.3.3 Kenogrammatische Fundierung

Die Komposition von Morphogrammketten ist somit definitorisch auf die indizierte Verkettung `kligate` zurückgeführt. Die Fundierung morphogrammatischer Operationen in der Kenogrammatik unterscheidet sich deutlich von Na's Ansatz (vgl. Kapitel 6 und 8), die Basismorphogramme als grundlegende Einheiten zu betrachten und die Polysemie aus den kombinatorischen Eigenschaften der Basismorphogramme abzuleiten. Obwohl Na's Analyse korrekt ist und zu den gleichen kombinatorischen Ergebnissen wie die hier präsentierte Methode führt, erweist sich eine generelle Fundierung in der Kenogrammatik als allgemeineres und mächtigeres Instrumentarium. So ist Na zwar in der Lage, die Anzahl von Matrizen einer morphogrammatischen Äquivalenzklassezu bestimmen, kann jedoch kein Verfahren zur faktischen Konstruktion dieser Matrizen angeben.

5.4.4 Reflektoren

Wie bereits festgestellt, sind morphogrammatische Strukturen invariant gegenüber logischen Negationen, da sie die Struktur von Operatoren der logischen Ebene unabhängig von einer durch Negationen permutierbaren Wertbelegung notieren. Günther entwarf die Reflektoren als morphogrammatische Operationen, um an morphogrammatischen Strukturen direkte Manipulationen vornehmen zu können, die auch logisch interpretierbare Ergebnisse liefern.

5.4.4.1 Einfache Reflektoren

Der kenogrammatische Reflektor `fun kref ks = tnf(rev ks);` bewirkt die Spiegelung einer Kenogrammsequenz. Die Anwendung des Reflekors auf die fünfzehn Basismorphogramme ist in der folgenden Tabelle (5.7) dargestellt:

i	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
mg_i	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○
	○	○	○	○	○	△	△	△	△	△	△	△	△	△	△
	○	○	△	△	△	○	○	○	△	△	△	□	□	□	□
	○	△	○	△	□	○	△	□	○	△	□	○	△	□	⋆
`kref(mg i)`	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○
	○	△	△	○	△	○	△	△	△	○	△	△	△	○	△
	○	△	○	△	□	△	○	□	△	○	△	□	○	△	□
	○	△	○	△	□	○	△	△	○	△	□	○	□	□	⋆
mg_i	1	10	6	4	14	3	7	13	9	2	11	12	8	5	15

Abbildung 5.7: Anwendung des Reflektors auf die Basismorphogramme

So werden beispielsweise die binären logischen Funktionen $p \vee q$ und $\neg(p \vee q)$ beide durch das negationsinvariante Morphogramm mg_2 repräsentiert.

p	q	$p \vee q$	$\neg(p \vee q)$	$TNF:\ mg_2$
1	1	1	2	○
1	2	1	2	○
2	1	1	2	○
2	2	2	1	△

Wird in diesen dualen Formeln jedoch nicht die ganze Funktion negiert, sondern nur jeweils die atomaren Teilformeln p und q, so ergibt sich folgende Situation:

p	q	$\neg p \vee \neg q$	$\neg(\neg p \vee \neg q)$	$TNF:\ mg_{10}$
1	1	2	1	○
1	2	1	2	△
2	1	1	2	△
2	2	1	2	△

Diese logischen Operationen der Negierung der Teilformeln p und q bewirkt also eine dem Reflektor `kref` entsprechende Spiegelung der morphogrammatischen Struktur der Ausgangsoperation. Der Reflektor kann demzufolge „*in einem gewissen Sinne als Verallgemeinerung der Negation aufgefasst werden.*“[6]

Die Funktionsweise dieser Spiegelungsoperation soll nun auf die komplexen Verbundsysteme der Morphogrammatrizen übertragen werden. Da eine $n \times n$ Morphogrammatrix $\binom{n}{2}$ Subsysteme enthält, existieren auch $\binom{n}{2}$ *einfache Reflektoren*, die jeweils ein Subsystemmorphogramm spiegeln.

Definition 5.14 (Einfacher Reflektor) *Der einfache Reflektor $r_i(Q^n)$ setzt die Reversion* $\texttt{rev}(M_i)$ *des i-ten Subsystemmorphogramms M_i von Q^n anstelle von M_i in die Matrix Q^n ein. Wobei $1 \leq i \leq \binom{n}{2}$.*

Für 3×3 Matrizen Q^3 existieren $\binom{3}{2} = 3$ einfache Reflektoren, die jeweils in einem der drei Subsysteme $S_1 : (1,2), S_2 : (2,3), S_3 : (1,3)$ operieren. Der Reflektor $r_1(Q^3)$ spiegelt S_1 usw. (das jeweils betroffene Subsystem ist durch ausgefüllte Kreise markiert):

$r_1(Q^3)$	1	2	3
1	●	●	○
2	●	●	○
3	○	○	○

$r_2(Q^3)$	1	2	3
1	○	○	○
2	○	●	●
3	○	●	●

$r_3(Q^3)$	1	2	3
1	●	○	●
2	○	○	○
3	●	○	●

Für 4×4 Matrizen Q^4 existieren $\binom{4}{2} = 6$ einfache Reflektoren, die in den Subsystemen $S_1 : (1,2), S_2 : (2,3), S_3 : (1,3), S_4 : (3,4), S_5 : (2,4), S_6 : (1,4)$ operieren:

$S_1 : (1,2)$	1	2	3	4
1	●	●	○	○
2	●	●	○	○
3	○	○	○	○
4	○	○	○	○

$S_2 : (2,3)$	1	2	3	4
1	○	○	○	○
2	○	●	●	○
3	○	●	●	○
4	○	○	○	○

$S_3 : (1,3)$	1	2	3	4
1	●	○	●	○
2	○	○	○	○
3	●	○	●	○
4	○	○	○	○

[6] [Gue80b1], S. 97.

$S_4:(3,4)$	1	2	3	4
1	○	○	○	○
2	○	○	○	○
3	○	○	●	●
4	○	○	●	●

$S_5:(2,4)$	1	2	3	4
1	○	○	○	○
2	○	●	○	●
3	○	○	○	○
4	○	●	○	●

$S_6:(1,4)$	1	2	3	4
1	●	○	○	●
2	○	○	○	○
3	○	○	○	○
4	●	○	○	●

5.4.4.2 Komplexe Reflektoren

Für eine n-elementige Menge M enthält die Potenzmenge $P(M)$ (die Menge aller Teilmengen von M) genau 2^n Elemente. Eine $n \times n$ Morphogrammatrix besitzt $\binom{n}{2}$ Subsysteme, auf denen jeweils ein einfacher Reflektor r_i operiert. Die Menge aller möglichen Kombinationen von einfachen Reflektoren, R^n enthält deshalb $2^{\binom{n}{2}}$ Elemente (einschließlich der durch die leere Menge $\{\}$ repräsentierte Nulloperation). Da jedoch $\{\}$ keinen eigentlichen Reflektor, sondern die Identitätsabbildung repräsentiert, existieren für eine $n \times n$ Matrix genau:

$$|R^n| = 2^{\binom{n}{2}} - 1 = \sum_{k=1}^{n} \binom{n}{k} \tag{5.1}$$

verschiedene Reflektoren. Da in R^n $\binom{n}{2}$ einfache Reflektoren enthalten sind, ist die Anzahl der *komplexen Reflektoren*, die auf mehreren Subsystemen gleichzeitig operieren:

$$\begin{aligned} |R^n| - \binom{n}{2} &= \left[\sum_{k=1}^{n} \binom{n}{k}\right] - \binom{n}{2} \\ &= \binom{n}{1} + \binom{n}{2} + \left[\sum_{k=3}^{n} \binom{n}{k}\right] - \binom{n}{2} \\ &= \binom{n}{1} + \left[\sum_{k=3}^{n} \binom{n}{k}\right] \\ &= n + \sum_{k=3}^{n} \binom{n}{k} \qquad \text{für} \quad n > 2 \end{aligned} \tag{5.2}$$

Für 3×3 Matrizen existieren $2^{\binom{3}{2}} - 1 = 7$ Reflektoren:

$$R^3 = \{r_1, r_2, r_3, r_{1.2}, r_{1.3}, r_{2.3}, r_{1.2.3}\}.$$

Die restlichen vier komplexen Reflektoren für Q^3 sind also:

$r_{1.2}(Q^3)$	1	2	3
1	●	●	○
2	●	●	●
3	○	●	●

$r_{1.3}(Q^3)$	1	2	3
1	●	●	●
2	●	●	○
3	●	○	●

$r_{2.3}(Q^3)$	1	2	3
1	●	○	●
2	○	●	●
3	●	●	●

$r_{1.2.3}(Q^3)$	1	2	3
1	•	•	•
2	•	•	•
3	•	•	•

Für Q^4 Matrizen gibt es entsprechend $2^{\binom{4}{2}} - 1 = 63$ Reflektoren:

$$R^4 = \{r_1, r_2, r_3, r_4, r_5, r_6, r_{1.2}, \ldots, r_{1.2.3.4.5.6}\}$$

Günther definiert die komplexen Reflektoren *nicht* als kommutative Verknüpfung der einfachen Reflektoren. Der Reflektor $r_{1.2}$ ist daher nicht als eine Komposition $r_1 \circ r_2$ oder $r_2 \circ r_1$ zu beschreiben[7].

Definition 5.15 (komplexer Reflektor) *Der komplexe Reflektor $r_I(Q^n)$ spiegelt die i-ten Subsystemmorphogramme M_i von Q^n. Dabei ist $i \in I$ und $I \subseteq P(1, \ldots, n)$. Das Spiegelungsverfahren ist durch* `r I Qn` *bestimmt:*

```
fun disjuncts subs n=
  let
   fun intersected sk sl =
      let
        val subsystems = subsystems n;
        val [ik,jk] = assoc sk subsystems;
        val [il,jl] = assoc sl subsystems;
      in
       (il=ik) orelse (il=jk) orelse (jk=jl) orelse (jl=ik)
      end;
   fun allintersectedwith si =
      remzeros(map (fn sj => if (intersected si sj) then sj
                             else 0)
                   (subs));

   fun allconnectedto sis =
      let
        val step = rd(flat(map allintersectedwith sis));
      in
        if (seteq sis step) then sis
        else allconnectedto step
      end;

   fun alldisjunctions [] =[]
      |alldisjunctions (hd::tl) =
        let
         val thisdisj = allconnectedto [hd];
         val rest = difference (hd::tl) thisdisj;
        in
         if rest=[] then [thisdisj]
         else thisdisj::(alldisjunctions rest)
        end;
  in
   alldisjunctions subs
  end;
```

[7] Vgl. auch [Hei91] S.87.

```
fun hauptdiag disj n =
  let
   val positions = rd (flat (map (fn s =>  (assoc s (subsystems n)))
                                 disj));
   val hdiag = map (fn pos => if (member pos positions) then pos
                              else 0)
                   (nlist n);
   fun split [] os = (os,[])
      |split (0::rest) os = split rest (0::os)
      |split x os = (os,x);


   val (leados,rest) = split hdiag [];
   val (followos,revdisj) = split (rev rest) [];
   val revdiag = leados@revdisj@followos
  in
   map (fn p => if (pos p hdiag)= 0 andalso
                    (pos p revdiag)= 0 then 0
                else p)
       (nlist n)
  end;

fun alltouchedby disj alldisj n =
  let
   fun remdups [] res = res
      |remdups (hd::tl) res =
        if (member hd res) then remdups tl res
        else remdups tl (hd::res);
   val touched= map k
                    (remdups (allpairs (remzeros (hauptdiag disj n))
                                       (remzeros (hauptdiag disj n)))
                             []);
  in
   rd(flat(remnils(map (fn disi => if (exists (fn s => member s touched)
                                              disj)
                                   then disi
                                   else [])
                       alldisj)))
  end;


fun allRBs [] n = []
   |allRBs (hd::tl) n =
     let
        val thisRB = hd@(alltouchedby hd tl n);
        val rest = difference (flat (hd::tl)) thisRB
     in
        if rest=[] then [thisRB]
        else thisRB::(allRBs (disjuncts rest n) n)
     end;
```

```
fun reflectRB RB mat=
 let
  val n = length mat;
  fun poke [] m = m
     |poke (si::tl) m =
        let
         val [i,j] = assoc si (subsystems n);
        in
         poke tl
         (set (i,i,pos i (pos i mat),
         set (i,j,pos j (pos i mat),
         set (j,i,pos i (pos j mat),
         set (j,j,pos j (pos j mat),m)))))
        end;

  fun split [] os = (os,[])
     |split (0::rest) os = split rest (0::os)
     |split x os = (os,x);

  val rhomat = poke RB (nlistof n (nlistof n 0));

  val (leados,rest) = split (flat rhomat) [];
  val (followos,revrho) = split (rev rest) [];
  val flatmat = flat mat;
  val flatrhomat = (leados @ revrho @ followos);
exception Nthcdr;
fun nthcdr n [] = raise Nthcdr
   |nthcdr 0 liste = liste
   |nthcdr n (hd::tl) = nthcdr (n-1) tl;

fun elements n m liste=
  nfirst (m-n+1) (nthcdr n liste);


fun mkmat n liste =
   map (fn z => elements (z*n) (z*n+n-1) liste)
       (fromto 0 (n-1));
 in
  mkmat n (map (fn i => if (pos i flatrhomat)=0 then (pos i flatmat)
                        else (pos i flatrhomat))
               (nlist (n*n)))
 end;


fun reflectall [] mat = mat
   |reflectall (RB::rest) mat =
           reflectall rest (reflectRB RB mat);

fun r I mat=
   reflectall (allRBs (disjuncts I (length mat)) (length mat)) mat;
```

Das diesem Algorithmus zugrunde liegende Verfahren wurde [Kae74] (S. 33 ff.) entnommen und soll kurz erläutert werden. Ein komplexer Reflektor zerfällt in einen

oder mehrere disjunkte Wirkungsbereiche ρ, die unabhängig voneinander gespiegelt werden können.

Beispiel: Der Reflektor $r_{1.4}(Q^4)$ zerfällt in zwei disjunkte Spiegelungen r_1 und r_4, da sich S_1 und S_4 nicht überschneiden. D.h. kein Punkt von S_1 gehört gleichzeitig auch zu S_4 und umgekehrt:

$r_{1.4}(Q^4)$	1	2	3	4
1	●	●	○	○
2	●	●	○	○
3	○	○	●	●
4	○	○	●	●

Die Funktion `disjuncts I n` erzeugt die Menge aller disjunkten Wirkungsbereiche eines Reflektors $r_{\mathtt{I}}(Q^{\mathtt{n}})$. Die Bestimmung unabhängiger Subsysteme verläuft analog zur Bestimmung unabhängigerSubsysteme bei der Komposition von Morphogrammketten (vgl. 5.4.3).
Aufgrund der nicht-kommutativen Verknüpfungseigenschaften der Subsystemspiegelungen können sich Spiegelungen der disjunkten Wirkungsbereiche auch auf Subsysteme auswirken, die nicht dem jeweiligen Wirkungsbereich angehören.

Beispiel: Der Reflektor $r_{1.6.8}(Q^5)$ zerfällt in zwei Wirkungbereiche $r_{1.6}(Q^5)$ und $r_8(Q^5)$, da sich S_8 weder mit S_1 noch mit S_6 überschneidet:

$r_{1.6.8}(Q^5)$	1	2	3	4	5
1	●	●	○	●	○
2	●	●	○	○	○
3	○	○	●	○	●
4	●	○	○	●	○
5	○	○	●	○	●

Die Spiegelung des Wirkungbereichs $r_{1.6}(Q^5)$ spiegelt die Position $(2,2)$ auf die Position $(3,3)$ der Matrix Q^5, die zum Wirkungsbereich des Reflektors $r_8(Q^5)$ gehört. Die Spiegelung der beiden Wirkungsbereiche $r_{1.6}(Q^5)$ und $r_8(Q^5)$ ist deshalb nicht unabhängig. D.h. insbesondere $r_{1.6}(r_8(Q^5)) \neq r_8(r_{1.6}(Q^5))$.

Wären beliebige Reihenfolgen der Wirkungsbereichspiegelungen zugelassen, würde es aufgrund der Nicht-Kommutativität zu verschiedenen Gesamtspiegelungen kommen. Um eindeutige Spiegelungsoperationen zu erhalten, müßte daher eine willkürliche Reihenfolge der Wirkungsbereichspiegelungen festgelegt werden, was der Idee der Zerlegung eines Gesamtreflektors in unabhängige Teilspiegelungen widerspricht.
Um ohne eine solche festgelegte Reihenfolge dennoch zu eindeutigen Reflektoren zu gelangen, werden in [Kae74] alle Wirkungsbereiche, die sich *möglicherweise* überlappen, zu einem *Reflektionsbereich RB* zusammengefasst. Möglich sind solche Überlappungen der Wirkungsbereiche immer dann, wenn Hauptdiagonalstellen eines Wirkungsbereiches zwischen den Hauptdiagonalstellen eines anderen Wirkungsbereiches liegen. Im obigen Beispiel liegt etwa die Position $(3,3)$ von $r_8(Q^5)$ auf der Hauptdiagonalen zwischen $(1,1)$ und $(4,4)$ von $r_{1.6}(Q^5)$ und kann daher von der Spiegelung dieses zweiten Bereiches betroffen sein.

Diese Methode behandelt jedoch beispielsweise den Reflektor $r_{6.8}(Q^5)$, als wenn er nicht aus zwei disjunkten Spiegelungen bestünde.

$r_{6.8}(Q^5)$	1	2	3	4	5
1	●	○	○	●	○
2	○	○	○	○	○
3	○	○	●	○	●
4	●	○	○	●	○
5	○	○	●	○	●

Da $(3,3)$ aus $r_8(Q^5)$ zwischen $(1,1)$ und $(4,4)$ aus $r_6(Q^5)$ liegt, wäre eine Überlappung der beiden Bereiche möglich, wenn z.B. S_1 auch zu diesem zweiten Bereich gehören würde. Die vorgeschlagene Methode muß daher die beiden unabhängigen Spiegelungen zu einem Reflektionsbereich zusammenfassen. Der Vorteil dieser ungenauen Methode besteht darin, daß sie mit relativ geringem Aufwand alle möglichen Überlappungen feststellen kann und zu eindeutigen Gesamtspiegelungen führt.
Hier wird im Gegensatz zu [Kae74] eine exakte Methode zur Bestimmung unabhängiger Reflektionsbereiche gewählt. Ausgehend von den disjunkten Wirkungsbereichen werden alle disjunkten Reflektionsbereiche durch die Funktion `allRBs` bestimmt. So wird z.B. der Reflektor $r_{6.8}(Q^5)$ in zwei disjunkte Bereiche zerlegt, jedoch der Reflektor $r_{1.6.8}(Q^5)$ als ein zusammenhängender Reflektionsbereich erkannt.
Die Verknüpfung der Spiegelungen dieser unabhängigen Reflektionsbereiche geschieht durch `(reflectall (allRBs (disjuncts I n)) Qn)`, wobei `reflectdisj RB Qn` die Spiegelung eines einzelnen Reflektionsbereiches RB bewirkt. Diese Spiegelung wird am Beispiel der Matrix:

$$Q^5 = \begin{matrix} \mathbf{1} & \mathbf{2} & 3 & \mathbf{4} & 5 \\ \mathbf{6} & \mathbf{7} & 8 & 9 & 10 \\ 11 & 12 & \mathbf{13} & 14 & \mathbf{15} \\ \mathbf{16} & 17 & 18 & \mathbf{19} & 20 \\ 21 & 22 & \mathbf{23} & 24 & \mathbf{25} \end{matrix}$$

mit $RB = \{S_1, S_6, S_8\}$ illustriert. In `rhomat` wird eine Kopie von Q^n angelegt, wobei alle Positionen, die nicht zu RB gehören, durch Nullen überschrieben werden. Diese Matrix wird durch zeilenweise Aneinanderreihung in eine Linearform überführt `(flat rhomat)`:

1 2 0 4 0 6 7 0 0 0 0 0 13 0 15 16 0 0 19 0 0 0 23 0 25

Innerhalb dieser Linearform wird RB (hier die gesamte Liste) reversiert:

25 0 23 0 0 0 19 0 0 16 15 0 13 0 0 0 0 0 7 6 0 4 0 2 1

Jeder Position, an der eine Null steht, wird der entsprechende Wert der Ausgangsmatrix zugewiesen:

Diese Liste wird dann durch `mkmat` wieder in eine Matrixdarstellung überführt:

$$\begin{matrix} 25 & 2 & 23 & 4 & 5 \\ 6 & 19 & 8 & 9 & 16 \\ 15 & 12 & 13 & 14 & 15 \\ 16 & 17 & 18 & 7 & 6 \\ 21 & 4 & 23 & 2 & 1 \end{matrix}$$

In dem verwendeten Beispiel wurden die Positionen 3 durch 23, 10 durch 16, 11 durch 15, 20 durch 6, 22 durch 4 und 24 durch 2 ersetzt. Die 25 ursprünglichen Belegungen wurden durch die Spiegelung $r_{1.6.8}$ auf 19 Belegungen reduziert. Allgemein wirken die Reflektoren also *reduktiv*.

Die Menge der $2^n - 1$ Reflektoren für Q^n-Matrizen, R^n, wird durch die Funktion `RG n` berechnet:

```
fun pot [] = [[]]
   |pot [x] = [[],[x]]
   |pot (hd::tl) =
     let
        val half = pot tl
     in
        half@(map (fn l => hd::l)
                  half)
     end;

fun RG n =
  map (fn I => r I)
      (remnils (pot (rev (nlist (choose n 2)))));
```

Mit dieser Definition der Reflektoren für Matrizen beliebiger Größe liegt die elementare Morphogrammatik, wie sie in den Arbeiten [Gue80b] und [Gue80b1] konzipiert wurde, in einer allgemein formulierten Darstellung vor.
In Teil III der Arbeit werden nun morphogrammatische Strukturen und Operationen analysiert. Insbesondere werden in Kapitel 7 die Eigenschaften der Reflektoroperationen einer eingehenden Analyse unterzogen.

Teil III

Analyse

Kapitel 6

Klassifikation der Morphogrammatik

In Teil II dieser Arbeit wurde die Günthersche Theorie der Morphogrammatik definitorisch als Teilgebiet der Kenogrammatik eingeführt. Grundsätzlich lassen sich daher die Eigenschaften morphogrammatischer Strukturen und Operationen aus den sie fundierenden kenogrammatischen Entsprechungen herleiten. Auf dieses Fundierungsverhältnis und seine formale Stärke wurde bereits in 5.4.3 hingewiesen. Unter dem Gesichtspunkt der definitorischen Einführung und der operationalen Implementierung ist die Morphogrammatik durch den in Teil II vorgelegten Aufbau abgeschlossen. Da die Morphogrammatik jedoch nur eine exakt begrenzte Teilmenge der kenogrammatischen Strukturen, Q, die Menge der 15 Basismorphogramme, als Operandensystem benutzt, liegt es nahe, zur formalen Beschreibung und Analyse der Morphogrammatik nicht den gesamten komplexen Apparat der Kenogrammatik zu verwenden, sondern sich auf die Analyse der Struktur und der kombinatorischen Eigenschaften von Q zu beschränken. Ein solches Vorgehen dient der Vereinfachung der Analyse, ersetzt jedoch nicht die grundsätzliche *Fundierung* der Morphogrammatik in der Kenogrammatik.
In Teil III sollen die wichtigsten Ergebnisse zur Klassifikation, Analyse und Theoriebildung über morphogrammatischen Operationen vorgestellt werden.
In diesem Kapitel werden zunächst verschiedene Klassifikationen des Operandensystems Q in disjunkte Teilmengen eingeführt, die der Analyse von morphogrammatischen Kompositions- und Reflektionsoperationen dienen. Im zweiten Abschnitt wird Kaehrs [Kae74] Klassifikation reflektionaler Operationen dargestellt.
In Kapitel 7 wird eine an [Kae78] angelehnte graphentheoretische Analyse morphogrammatischer Reflektions– und Umformungsoperationen skizziert. In Kapitel 8 wird dann Na's [Na64] kombinatorische Analyse derKomposition von Morphogrammketten unter besonderer Berücksichtigung der*Polysemie* rekonstruiert.

6.1 Klassifikation des Operandensystems Q

6.1.1 Die f-c-Klassifikation

Bei der in 5.4.3 definierten Komposition einer Morphogrammkette $MK = [M_1, \ldots, M_{\binom{n}{2}}]$ zu einer Morphogrammatrix W^n, müssen sich wegen der Vermittlungsbedingungen (VB) gewisse Paare von Subsystemmorphogrammen (M_i, M_j) in

einzelnen Stellen ihrer Hauptdiagonalen überschneiden. Ob sich eine Morphogrammkette komponieren läßt, ist deshalb allein von den Hauptdiagonalstrukturen der einzelnen Morphogramme M_i abhängig.

> **Beispiel:** Bei der Komposition einer $MK = [M_1, M_2, M_3]$ existiert nur dann eine Matrix W^3, falls entweder genau ein oder genau drei Morphogramme gleiche Hauptdiagonalkenogramme aufweisen. Für $MK_1 = [[1,1,1,1],[1,1,1,1],[1,1,1,2]]$ existiert keine Komposition, da die Position $(3,3)$ der Matrix gleichzeitig mit den Kenogrammsymbolen ○ und △ belegt werden müßte:

MK_1	1	2	3
1	1	1	1
2	1	1	1
3	1	1	1,2

Um allgemeine Aussagen über die Komponierbarkeit von Morphogrammketten treffen zu können, wird die Menge der Morphogramme, Q, in zwei disjunkteKlassen Q_c und Q_f zerlegt.

Definition 6.1 (Core-Klasse Q_c) *In der Klasse $Q_c \subset Q$ befinden sich alle Basismorphogramme, deren Diagonalelemente mit dem gleichen Kenogramm belegt sind:*

$$Q_c = \{mg \in Q | mg = \begin{pmatrix} w_{11} & w_{12} \\ w_{21} & w_{22} \end{pmatrix} \text{ und } w_{11} = w_{22}\}.$$

$$Q_c = \{\begin{pmatrix} 1 & 1 \\ 1 & 1 \end{pmatrix}, \begin{pmatrix} 1 & 1 \\ 2 & 1 \end{pmatrix}, \begin{pmatrix} 1 & 2 \\ 1 & 1 \end{pmatrix}, \begin{pmatrix} 1 & 2 \\ 2 & 1 \end{pmatrix}, \begin{pmatrix} 1 & 2 \\ 3 & 1 \end{pmatrix}\}.$$

Definition 6.2 (Frame-Klasse Q_f) *In der Klasse $Q_f \subset Q$ befinden sich alle Basismorphogramme, deren Diagonalelemente mit verschiedenen Kenogrammen belegt sind:*

$$Q_f = \{mg \in Q | mg = \begin{pmatrix} w_{11} & w_{12} \\ w_{21} & w_{22} \end{pmatrix} \text{ und } w_{11} \neq w_{22}\}.$$

$$\begin{aligned} Q_f \quad = \quad \{&\begin{pmatrix} 1 & 1 \\ 1 & 2 \end{pmatrix}, \begin{pmatrix} 1 & 1 \\ 2 & 2 \end{pmatrix}, \begin{pmatrix} 1 & 1 \\ 2 & 3 \end{pmatrix}, \begin{pmatrix} 1 & 2 \\ 1 & 2 \end{pmatrix}, \begin{pmatrix} 1 & 2 \\ 1 & 3 \end{pmatrix}, \\ &\begin{pmatrix} 1 & 2 \\ 2 & 2 \end{pmatrix}, \begin{pmatrix} 1 & 2 \\ 2 & 3 \end{pmatrix}, \begin{pmatrix} 1 & 2 \\ 3 & 2 \end{pmatrix}, \begin{pmatrix} 1 & 2 \\ 3 & 3 \end{pmatrix}, \begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix}, \}. \end{aligned}$$

Offensichtlich gilt:

$$Q = Q_c \cup Q_f \text{ und } Q_c \cap Q_f = \{\}.$$

Eine $n \times n$ Matrix W^n weist genau n Diagonalstellen $d_1, \ldots, d_n$ auf:

$$W^n = \begin{pmatrix} d_1 & - & \cdots & - \\ - & d_2 & \cdots & - \\ \vdots & \vdots & \ddots & \vdots \\ - & - & \cdots & d_n \end{pmatrix}.$$

Die Sequenz $[d_1, \ldots, d_n]$ kann nun genau $Tcard(n)$ verschiedene Kenogrammbelegungen aufweisen, die durch `Tcontexture(n)` berechnet werden. Die $\binom{n}{2}$ Subsysteme der Matrix W^n werden durch `subsystems(n)` bestimmt. Das $k(i,j)$-te Subsystem $S_{k(i,j)} : (i,j)$ ist nun vom c-Typ (d.h. Element von Q_c), wenn $d_i = d_j$. Falls $d_i \neq d_j$, ist $S_{k(i,j)} \in Q_f$ (f-Typ). Die f-c-Struktur einer Hauptdiagonalen hd ist die Liste der f oder c Typen ihrer Subsysteme, berechnet wird sie durch die Funktion `FCstructure(hd)`:

```
datatype fc = F|C;

fun FCstructure hd =
  map (fn (k,[i,j]) => if (pos i hd)=(pos j hd) then C
            else F)
    (subsystems (length hd));
```

Beispiel:

```
- FCstructure [1,2,1,1];
> [F,F,C,C,F,C] : fc list
```

Die Menge aller möglichen f-c-Strukturen für eine Matrix W^n wird durch die Funktion `allFCs(n)` berechnet:

```
fun allFCs n =
  map (fn ks => FCstructure ks)
    Tcontexture n;
```

Beispiel:

```
- allFCs(3);
> [[C,C,C],[C,F,F],[F,F,C],
   [F,C,F],[F,F,F]] : fc list list
```

Die oben versuchte Komposition von $[mg_1, mg_1, mg_2]$ ist vom Typ $[c, c, f]$ und wie zu erwarten nicht in der Liste aller möglichen W^3 f-c-Strukturen enthalten.

Der f-c-Typ eines Morphogramms wird durch die Funktion `FCtype(mg)` bestimmt:

```
fun FCtype mg = if (pos 1 mg)=(pos 4 mg) then C
        else F;
```

Die f-c-Struktur einer Morphogrammkette MK wird durch die Funktion `FCtypes(MK)` bestimmt:

```
fun FCtypes MK =
  map (fn mg => FCtype mg)
    MK;


fun exmm MK =
  if (member (FCtypes MK)
```

```
      (allFCs floor(0.5+sqrt(0.25+
                    real(2*(length MK))))))
  then true
 else false;
```

Die Funktion `exmm(MK)` ergibt den Wert `true`, falls sich MK zu einer Matrix komponieren läßt, sonst false.

Beispiel:

```
- exmm [mg 1,mg 1,mg 2];
> false : bool
```

6.1.2 Weitere Klassifikationen

6.1.2.1 Die k-l-o-r–Klassifikation

Kaehr schlägt in [Kae74] und [Kae78] weitere Klassifikationen von Q vor, die vor allem für die Analyse der Reflektoroperationen von Bedeutung sind (vgl. Kapitel 7). In der k-l-o-r–Klassifikation wird das f-c-Klassifikationsprinzip der Gleichheit der Diagonalelemente zusätzlich auf die Nebendiagonalplätze der Morphogramme angewandt. Für Q_c entstehen so die Unterklassen Q_r und Q_o, für Q_f die Klassen Q_l und Q_k.

Definition 6.3 (Klasse Q_k) *In der Klasse $Q_k \subset Q_f \subset Q$ befinden sich diejenigen Morphogramme aus Q_f, deren Nebendiagonalstellen mit verschiedenen Kenogrammen belegt sind:*

$$Q_k = \{mg \in Q_f | mg = \begin{pmatrix} w_{11} & w_{12} \\ w_{21} & w_{22} \end{pmatrix} \text{ und } w_{12} \neq w_{21}\}.$$

Definition 6.4 (Klasse Q_l) *In der Klasse $Q_l \subset Q_f \subset Q$ befinden sich diejenigen Morphogramme aus Q_f, deren Nebendiagonalstellen mit dem gleichen Kenogramm belegt sind:*

$$Q_l = \{mg \in Q_f | mg = \begin{pmatrix} w_{11} & w_{12} \\ w_{21} & w_{22} \end{pmatrix} \text{ und } w_{12} = w_{21}\}.$$

Definition 6.5 (Klasse Q_o) *In der Klasse $Q_o \subset Q_c \subset Q$ befinden sich diejenigen Morphogramme aus Q_c, deren Nebendiagonalstellen mit dem gleichen Kenogramm belegt sind:*

$$Q_o = \{mg \in Q_c | mg = \begin{pmatrix} w_{11} & w_{12} \\ w_{21} & w_{22} \end{pmatrix} \text{ und } w_{12} = w_{21}\}.$$

Definition 6.6 (Klasse Q_r) *In der Klasse $Q_r \subset Q_c \subset Q$ befinden sich diejenigen Morphogramme aus Q_c, deren Nebendiagonalstellen mit verschiedenen Kenogrammen belegt sind:*

$$Q_r = \{mg \in Q_c | mg = \begin{pmatrix} w_{11} & w_{12} \\ w_{21} & w_{22} \end{pmatrix} \text{ und } w_{12} \neq w_{21}\}.$$

Offensichtlich gilt:

$$\begin{aligned} Q_c &= Q_o \cup Q_r \text{ und } Q_o \cap Q_r = \{\} \\ Q_f &= Q_l \cup Q_k \text{ und } Q_l \cap Q_k = \{\} \end{aligned}$$

Der k-l-o-r–Typ eines Morphogramms wird durch die Funktion `klortype mg`, der Typ einer Morphogrammkette wird durch `KLORtypes mk` bestimmt:

```
datatype klor =K|L|O|R;

fun klortype mg=
  if (pos 1 mg) = (pos 4 mg)
    then if (pos 2 mg)=(pos 3 mg)
           then O
           else R
    else if (pos 2 mg)=(pos 3 mg)
           then L
           else K;

fun KLORtypes mk = map klortype mk;
```

6.1.2.2 Die a-s-Klassifikation

Morphogramme besitzen zwei Haupt- und zwei Nebendiagonalelemente. Die a-s-Klassifikation zerlegt nun Q in zwei disjunkte Klassen Q_a, Q_s die bestimmte Kombinationen von Haupt- und Nebendiagonaleigenschaften aufweisen.

Definition 6.7 (Klasse Q_a) *In der Klasse $Q_a \subset Q$ befinden sich diejenigen Morphogramme aus Q, die entweder gleiche Diagonal- und ungleiche Nebendiagonalkenogramme oder ungleiche Diagonal- und gleiche Nebendiagonalkenogramme besitzen:*

$$\begin{aligned} Q_a = \{mg \in Q | mg = &\begin{pmatrix} w_{11} & w_{12} \\ w_{21} & w_{22} \end{pmatrix} \\ &\text{und entweder } (w_{11} = w_{22} \wedge w_{12} \neq w_{21}) \text{ oder } (w_{11} \neq w_{22} \wedge w_{12} = w_{21})\}. \end{aligned}$$

Definition 6.8 (Klasse Q_s) *In der Klasse $Q_s \subset Q$ befinden sich diejenigen Morphogramme aus Q, die entweder sowohl gleiche Diagonal- als auch gleiche Nebendiagonalkenogramme oder sowohl ungleiche Diagonal- als auch ungleiche Nebendiagonalkenogramme besitzen:*

$$\begin{aligned} Q_s = \{mg \in Q | mg = &\begin{pmatrix} w_{11} & w_{12} \\ w_{21} & w_{22} \end{pmatrix} \\ &\text{und entweder } (w_{11} = w_{22} \wedge w_{12} = w_{21}) \text{ oder } (w_{11} \neq w_{22} \wedge w_{12} \neq w_{21})\}. \end{aligned}$$

Offensichtlich ist:

$$Q = Q_a \cup Q_s \text{ und } Q_a \cap Q_s = \{\}.$$

Zwischen den Klassen Q_o, Q_l, Q_r, Q_k und Q_a und Q_s bestehen die folgenden Beziehungen:

$$\begin{aligned} Q_s = Q_o \cup Q_k \quad , \quad Q_o \cap Q_k = \{\} \\ Q_a = Q_r \cup Q_l \quad , \quad Q_r \cap Q_l = \{\}. \end{aligned}$$

6.1.2.3 Die g-h-Klassifikation

Als weitere Unterteilung von Q schlägt Kaehr die g-h-Klassifikation vor.

Definition 6.9 (Klasse Q_g) *Die Klasse $Q_g \subset Q$ ist gegeben durch:*

$$Q_g = Q_r \cup Q_k \quad \text{mit} \quad Q_r \cap Q_k = \{\}.$$

Definition 6.10 (Klasse Q_h) *Die Klasse $Q_h \subset Q$ ist gegeben durch:*

$$Q_h = Q_o \cup Q_l \quad \text{mit} \quad Q_o \cap Q_l = \{\}.$$

Der g-h-Typ eines Morphogramms wird durch die Funktion `ghtype mg`, der Type einer Morphogrammkette durch `GHtypes mk` berechnet:

```
datatype gh=G|H;
fun ghtype mg=
   if (pos 2 mg)=(pos 3 mg) then H
   else G;

fun GHtypes mk = map ghtype mk;
```

6.1.3 Zusammenfassung der Klassifikationen

Die Klassifikationen von Q in die Systeme f-c, a-s und g-h decken alle kombinatorisch möglichen Paarbildungen aus der Menge $\{k, l, o, r\}$ ab. Diehier definierten Klassifikationen lassen sich zu der folgenden Übersicht6.1 zusammenstellen , die die wichtigsten Ergebnisse dieses Abschnitts tabellarisch aufführt.

Klasse Q_i	Zusammensetzung aus disjunkten Unterklassen	Struktur der Morphogramme in Q_i	Morphogramme in Q_i
Q_c	$Q_r \cup Q_o$	$w_{11} = w_{22}$	○○ ○○ ○△ ○△ ○△ ○○, △○, ○○, △○, □○
Q_f	$Q_l \cup Q_k$	$w_{11} \neq w_{22}$	○○ ○○ ○○ ○△ ○△ ○△, △△, △□, ○△, ○□, ○△ ○△ ○△ ○△ ○△ △△, △□, □△, □□, □★
Q_a	$Q_r \cup Q_l$	$(w_{11} = w_{22} \wedge w_{12} \neq w_{21}) \vee$ $(w_{11} \neq w_{22} \wedge w_{12} = w_{21})$	○○ ○○ ○△ ○△ ○△ ○△, △○, ○○, △△, △□, ○△ □○
Q_s	$Q_o \cup Q_k$	$(w_{11} = w_{22} \wedge w_{12} = w_{21}) \vee$ $(w_{11} \neq w_{22} \wedge w_{12} \neq w_{21})$	○○ ○○ ○○ ○△ ○△ ○○, △△, △□, ○△, ○□, ○△ ○△ ○△ ○△ △○, □○, □□, □★
Q_g	$Q_r \cup Q_k$	$w_{12} \neq w_{21}$	○○ ○○ ○○ ○△ ○△ △○, △△, △□, ○○, ○△, ○△ ○△ ○△ ○△ ○△ ○□, □○, □△, □□, □★
Q_h	$Q_o \cup Q_l$	$w_{12} = w_{21}$	○○ ○○ ○△ ○△ ○△ ○○, ○△, △○, △△, △□
Q_k	—	$(w_{11} \neq w_{22} \wedge w_{12} \neq w_{21})$	○○ ○○ ○△ ○△ ○△ △△, △□, ○△, ○□, □△, ○△ ○△ □□, □★
Q_l	—	$(w_{11} \neq w_{22} \wedge w_{12} = w_{21})$	○○ ○△ ○△ ○△, △△, △□
Q_o	—	$(w_{11} = w_{22} \wedge w_{12} = w_{21})$	○○ ○△ ○○, △○
Q_r	—	$(w_{11} = w_{22} \wedge w_{12} \neq w_{21})$	○○ ○△ ○△ △○, ○○, □○

Abbildung 6.1: Klassifikation von Q

6.1.4 Die Klassifikation nach Na

Bei der Analyse der morphogrammatischen Komposition *kom* in Kapitel 8 wird eine auf Na zurückgehende Klassifikation von Q benutzt [Na64]. Diese Klassifikation ist in Abbildung 6.2 veranschaulicht.

Klasse	α	β			γ	ξ				ρ					ϕ
	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○
Morpho-	○	○	△	△	△	○	○	△	△	○	□	△	□	□	□
gramm	○	△	○	△	□	○	△	○	△	□	○	□	△	□	⋆
	○	○	○	○	○	△	△	△	△	△	△	△	△	△	△
Nummer mg_i	1	3	6	9	12	2	4	7	10	5	8	13	14	11	15
f-c–Klass.	Q_c					Q_f									

Abbildung 6.2: Die Klassifikation von Q nach Na

Deutlich erkennbar ist, daß Q_c die Klassen α, β, γ und Q_f die Klassen ξ, ρ, ϕ umfasst.

6.2 Klassifikation reflektionaler Operationen

In [Kae78] schlägt Kaehr eine allgemeine Klassifikation unärer morphogrammatischer Operationen vor. Als spezifisches Beispiel solcher Operationen werden hierbei die Reflektoren (vgl. 5.4.4) gewählt. Die Analyse gilt jedoch allgemein für beliebige unäre Operationen. Kaehr klassifiziert Umformungen nach ihrem *Modus*, ihrem *Typ* und ihrer *Intensität*.

6.2.1 Umformungsmodi

Definition 6.11 (Direkte Umformung) *Eine reflektionale Umformung r_I heißt direkte Umformung gdw., genau diejenigen Subsysteme S_i von der Umformung betroffen sind, die zum Wirkungsbereich ρ von r_I gehören.*

Beispiel:

$$r_{1.3}([2, 10, 10]) = [11, 10, 2]$$

Dabei repräsentiert $[2, 10, 10]$ die Morphogrammkette $[mg_2, mg_{10}, mg_{10}]$. Da hier der Wirkungsbereich die Systeme S_1 und S_3 enthält, $\rho = \{S_1, S_3\}$, und die Umformung nur diese Subsysteme betrifft (das Morphogramm mg_{10} in S_2 bleibt unverändert), liegt eine direkte Umformung vor.

Definition 6.12 (Indirekte Umformung) *Eine reflektionale Umformung r_I heißt indirekte Umformung gdw., nur Subsysteme S_i von der Umformung betroffen sind, die nicht zum Wirkungsbereich ρ von r_I gehören.*

Beispiel:

$$r_1([11, 2, 2]) = [11, 10, 2]$$

Hier ist $\rho = \{S_1\}$, verändert werden von der Umformung jedoch nur die Subsysteme S_2 und S_3.

Definition 6.13 (Gemischte Umformung) *Eine reflektionale Umformung r_I heißt gemischte Umformung, wenn mindestens zwei Subsysteme S_i, S_j umgeformt werden, wobei $S_i \in \rho$ und $S_j \notin \rho$, mit $i, j \in I$.*

Beispiel:

$$r_1([10, 2, 10]) = [2, 11, 10]$$

Betroffen von der Umformung sind hier die Subsysteme S_1 und S_2, wobei $S_1 \in \rho$ und $S_2 \notin \rho$.

Definition 6.14 (Eigentliche Umformung) *Eine reflektionale Umformung r_I heißt eigentliche Umformung gdw., $r_I(Q^n) \neq Q^n$.*

Definition 6.15 (Uneigentliche Umformung) *Eine reflektionale Umformung r_I heißt uneigentliche Umformung gdw., $r_I(Q^n) = Q^n$.*

Beispiel:

$$r_1([9, 10, 2]) = [9, 10, 2]$$

mg_9 (○△△○) ist bezüglich des Reflektors symmetrisch: `kref [1,2,2,1] = [1,2,2,1]`. Aufgrund dieser Spiegelsymmetrie wird $[9, 10, 2]$ durch r_1 auf sich selbst abgebildet, es liegt somit eine uneigentliche Umformung vor.

Definition 6.16 (Reversible Umformung) *Eine reflektionale Umformung r_I heißt reversible Umformung gdw., $r_I(r_I(Q^n)) = Q^n$.*

Definition 6.17 (Irreversible Umformung) *Eine reflektionale Umformung r_I heißt irreversible Umformung gdw., $r_I(r_I(Q^n)) \neq Q^n$.*

6.2.2 Umformungstypen

Es werden drei Typen von Umformungen unterschieden: *Verformung*, *Verschiebung* und *Verkehrung*.

Definition 6.18 (Verformung) *Eine reflektionale Umformung $r_I(Q^n)$ heißt Verformung, wenn durch sie mindestens ein Subsystemmorphogramm M_i des Subsystems S_i, $i \in I$ seinen Klassifikationstyp τ_i^j verändert:*

$$\tau_i^j(r_I(Q^n)) \neq \tau_i^j(Q^n).$$

Dabei ist τ_i^j der Klassifikationstyp des Morphogramms M_i, mit $j \in \{\alpha, \beta, \gamma, \ldots\}$. Wobei $\alpha = \{k, l, o, r\}, \beta = \{c, f\}, \gamma = \{g, h\}$.

Beispiel:

$$r_{1.3}([2, 5, 2]) = [11, 10, 10]$$

$$\begin{aligned} \tau^\alpha([2, 5, 2]) &= [l, k, l] \\ \tau^\alpha([11, 10, 10]) &= [l, l, l] \end{aligned}$$

Der k-l-o-r–Klassifikationstyp $[l, k, l]$ wird durch $r_{1.3}$ zu $[l, l, l]$ umgeformt, so daß:

$$\tau_2^\alpha([l, k, l]) = k \qquad \neq \qquad \tau_2^\alpha([l, l, l]) = l$$

Definition 6.19 (Verschiebung) *Eine reflektionale Umformung $r_I(Q^n)$ heißt Verschiebung, wenn sie die Morphogrammkette $Dek(Q^n)$ permutiert:*

$$Dek(r_I(Q^n)) = \pi(Dek(Q^n)).$$

Beispiel:

$$r_3([11, 10, 2]) = [2, 11, 10]$$

Offensischtlich ist $[2, 11, 10]$ eine Permutation von $[11, 10, 2]$.

Definition 6.20 (Verkehrung) *Eine reflektionale Umformung $r_I(Q^n)$ heißt Verkehrung, wenn sie den Klassifikationstyp $\tau^j(Q^n)$ dualisiert:*

$$\tau^j(r_I(Q^n)) = \tau^{j'}(Q^n)$$

.
Dabei ist $j \in \{\alpha, \beta, \gamma, \ldots\}$, mit $\alpha = \{k, l, o, r\}$, $\beta = \{g, h\}$ und $\gamma = Q^1$. Es gelten folgende duale Beziehungen:

$\tau^{klor'}$	
k	l
o	r

$\tau^{gh'}$	
g	h

$\tau^{Q'}$	
1	*1*
2	*10*
3	*6*
4	*4*
5	*14*
7	*7*
8	*13*
9	*9*
11	*11*
12	*12*
15	*15*

Beispiel:

$$r_{1.3}([2, 5, 2]) = [11, 10, 10]$$

$$\begin{aligned} \tau^{\alpha}([2, 5, 2]) &= [l, k, l] \\ \tau^{\alpha}([11, 10, 10]) &= [l, l, l] \end{aligned}$$

Da l dual zu k ist, stellt die Typveränderung des zweiten Morphogramms eine Dualisierung bezüglich der k-l-o-r–Klassifikation dar.

6.2.3 Umformungsintensitäten

Kaehr unterscheidet die folgenden Intensitätsaspekte reflektionaler Umformungen: den *Umformungsgrad*, die *Umformungsart* und die *Umformungskompliziertheit*.

[1] Vgl. Tabelle (5.7) auf Seite 102.

Definition 6.21 (Umformungsgrad) *Eine reflektionale Umformung $r_I(Q^n)$, die m Subsysteme umformt, ist vom Umformungsgrad* $\mathrm{grad}(r_I(Q^n))$*, mit*

$$\mathrm{grad}(r_I(Q^n)) = \frac{m}{\binom{n}{2}}$$

Wird bei einer Umformung nur ein Teil der Subsysteme umgeformt ($\mathrm{grad}(r_I(Q^n)) < 1$), so wird sie *partielle* Umformung genannt. Werden alle Subsystem umgeformt ($\mathrm{grad}(r_I(Q^n)) = 1$), so heißt sie *totale* Umformung.

Definition 6.22 (Typkonstante Umformung)
Eine reflektionale Umformung $r_I(Q^n)$ heißt typkonstante Umformung gdw., wenn der Klassifikationstyp $\tau^j(Q^n)$ durch die Umformung nicht verändert wird:

$$\tau^j(r_I(Q^n)) = \tau^j(Q^n).$$

Definition 6.23 (Typvariable Umformung) *Eine reflektionale Umformung $r_I(Q^n)$ heißt typvariable Umformung gdw., wenn der Klassifikationstyp $\tau^j(Q^n)$ durch die Umformung verändert wird:*

$$\tau^j(r_I(Q^n)) \neq \tau^j(Q^n).$$

Definition 6.24 (Umformungskompliziertheit) *Die Kompliziertheit einer reflektionalen Umformung U,* $\mathrm{komp}(U)$ *bezeichnet die Anzahl der in U vorkommenden Reflektoren, wobei ein aus m disjunkten Reflektionsbereichen RB zusammengesetzter Reflektor als m einzelne Reflektoren gezählt wird.*

Beispiel:

$$\mathrm{komp}(r_1(r_{1.2}(r_{1.6.8}(Q^5)))) = 1 + 1 + 2 = 4,$$

da $r_{1.6.8}$ in zwei disjunkte Reflektionsbereiche zerfällt (vgl. 5.4.4).

Kapitel 7

Graphentheoretische Analyse Reflektionaler Operationen

Im vorherigen Kapitel wurden verschiedene Klassifikationen des morphogrammatischen Operandensystems Q eingeführt. In diesem Kapitel soll die Verknüpfungsstruktur unärer morphogrammatischer Operationen (insbesondere der Reflektoren) bezüglich der entwickelten Klassifikationen analysiert werden. Aufgrund der exponentiellen Komplexität kenogrammatischer und morphogrammatischer Operationen, die schon für 4×4 Morphogrammatrizen zu erheblichen Speicherplatz– und Rechenzeitproblemen führen,[1] erscheint außer der konstruktiven Definition morphogrammatischer Operationen (vgl. Teil II) ein analytischer Zugang zur Morphogrammatik ebenfalls angebracht, um Aussagen über faktisch nicht mehr realisierbare Objekte und Operationen treffen zu können. Die hier entwickelten Analysen rekonstruieren und verallgemeinern die Arbeiten Kaehrs [Kae74] und [Kae78].

In Abschnitt 5.4.4 wurde darauf hingewiesen, daß die komplexen Reflektoren aus R^n nicht als kommutative Verknüpfungen der $\binom{n}{2}$ einfachen Subsystemreflektoren, sondern als nichtreduzierbar komplex strukturierte Operationen definiert sind. Verknüpfungen dieser Reflektoren können daher nicht auf kommutative Elementaroperationen reduziert werden. Um dennoch die Verknüpfungsstruktur d er Reflektoren beschreiben zu können, werden hier Analysen dieser Verknüpfungen bezüglich verschiedener Klassifikationsstufen von Q (f-c–, g-h–, k-l-o-r–Klassifikation) durchgeführt. Aufgrund der kombinatorischen Komplexität ist es erst nach einer solchen Analyse sinnvoll möglich, die konkrete (d.h nichtabstrahierte) Verknüpfungsstruktur zu untersuchen.

Aus Gründen der Übersichtlichkeit werden nur 3×3–Matrizen explizit untersucht, wohingegen die Formalisierung und Implementierung für Objekte beliebiger Größe erfolgt. Allgemein sollen in den hier durchgeführten Untersuchungen das Verhalten morphogrammatischer Komplexionen bezüglich unärer Umformungoperationen analysiert werden.

[1] Die Komposition `Kom [mg 15,mg 15,mg 15,mg 15,mg 15,mg 15]` erzeugt beispielsweise 65 766 991 verschiedene 4×4 Matrizen, [Na64], S. 125, (vgl. Kapitel 8).

7.1 Die f-c–Analyse

Die f-c–Klassifikation von Q, der Menge der fünfzehn Basismorphogramme, berücksichtigt nur die Hauptdiagonalstellen der Morphogramme. Die Funktion `(Cmg M)` testet ob das Morphogramm `M` vom f-c–Typ c ist, ebenso testet `(Fmg M)` ob `M` vom Typ f ist:

```
fun Cmg [w11,_,_,w22] = (w11=w22);
fun Fmg [w11,_,_,w22] = (w11<>w22);
```

Die Menge Q_c enthält alle Basismorphogramme vom Typ c, entsprechend enthält Q_f die restlichen f-Typ Morphogramme aus Q:

```
fun allof typ []=[]
   |allof typ(hd::tl)=
     if (typ hd) then hd::(allof typ tl)
     else allof typ tl;

val Qc= allof Cmg Q;
val Qf= allof Fmg Q;
```

Aus den Basismorphogrammen können komplexe Q^n $n \times n$–Matrizen gebildet werden. Jede Q^n Matrix ist aus $\binom{n}{2}$ Basismorphogrammen zusammengesetzt. Aufgrund der Vermittlungsbedingungen der Subsysteme sind für eine Q^n Matrix nur bestimmte f-c–Strukturen möglich (vgl. 6.1.1), die durch die Funktion `allFCs(n)` ermittelt werden. Für Q^3 ist dies:

```
- allFCs(3);
> [[C,C,C],[C,F,F],[F,F,C],[F,C,F],[F,F,F]] : fc list list
```

Die Klasse aller möglichen f-c–Strukturen, Q^n_{fc} ist nach der Definition der Funktion `allFCs` (Seite 115) isomorph zur Tritokontextur `tcontexture n` und enthält demgemäß $Tcard(n) = \sum_{k=1}^{n} S(n,k)$ verschiedene Strukturen:

$$|Q^n_{fc}| = \sum_{k=1}^{n} S(n,k) \tag{7.1}$$

Beispiel:

$$|Q^3_{fc}| = \sum_{k=1}^{3} S(3,k) = 5$$

Die Menge aller zu einer bestimmten f-c–Klasse $[\mathtt{fc}_1, \ldots, \mathtt{fc}_{\binom{n}{2}}]$ gehörenden Morphogrammketten, $Q^n_{fc_1 \ldots fc_{\binom{n}{2}}}$ ist durch die Funktion `allMKs_of` $[\mathtt{fc}_1, \ldots, \mathtt{fc}_{\binom{n}{2}}]$ bestimmt:

```
fun allMKs_of typ=
  let
   fun comb []= [[]]
      |comb [l]= map (fn x => [x]) l
      |comb (hd::tl)=
```

```
          flat(map (fn mg => combine mg (comb tl))
                   hd);
  in
   comb (map (fn fc => if fc=C then Qc
                       else Qf)
             typ)
  end;
```

Die Klasse aller möglichen Morphogrammketten die zu 3×3 Matrizen komponierbar sind, Q^3_{MK}, ist demnach gegeben als:

$$Q^3_{MK} = \texttt{allMKs}(Q_{\text{ccc}}) \cup \texttt{allMKs}(Q_{\text{cff}}) \cup \texttt{allMKs}(Q_{\text{ffc}}) \cup \texttt{allMKs}(Q_{\text{fcf}}) \cup \texttt{allMKs}(Q_{\text{fff}}).$$

Allgemein wird Q^n_{MK} durch die Funktion `QMK(n)` berechnet:

```
fun QMK n=  map (fn fctyp => allMKs_of fctyp)
                (allFCs n);
```

Da Q_f zehn und Q_c fünf Morphogramme enthält, ist $|Q^n_{MK}|$, die Anzahl komponierbarer Morphogrammketten:

$$|Q^n_{MK}| = \sum_{k=1}^{|Q^n_{fc}|} \left(10^{\text{Fcard}(Q^n_{fc_k})} \times 5^{\text{Ccard}(Q^n_{fc_k})}\right) \tag{7.2}$$

Hierbei ist Fcard(`fctyp`) die Anzahl von f–Morphogrammen in `fctyp` und entsprechend Ccard(`fctyp`) die Anzahl der c–Morphogramme, $|Q_c| = 5$ und $|Q_f| = 10$.

Beispiel:

$$\begin{aligned}
|Q^3_{MK}| &= \sum_{k=1}^{|Q^3_{fc}|} \left(10^{\text{Fcard}(Q^3_{fc_k})} \times 5^{\text{Ccard}(Q^3_{fc_k})}\right) \quad : \\
\text{ccc}: &\quad 5^3 = 125 \\
\text{cff}: &\quad 5 \times 10^2 = 500 \\
\text{ffc}: &\quad 5 \times 10^2 = 500 \\
\text{fcf}: &\quad 5 \times 10^2 = 500 \\
\text{fff}: &\quad \underline{10^3 = 1000} \\
&= 2625 \\
|Q^3_{MM}| &= \sum_{k=1}^{3^2} S(3^2, k) \quad = 21\ 147
\end{aligned}$$

7.1.1 Empirische Analyse

Um das Verhalten der f-c–Klassen von Morphogrammketten bezüglich der Reflektoroperationen zu untersuchen wird zunächst die Funktion `mapsto` definiert:

```
fun mapsto typ reflist =
 let
  fun applyrefs k (reflist : ('a -> 'a) list)=
```

```
   let
     fun apprefs mm [] = []
        |apprefs mm (r::rs) = (r mm)::(apprefs mm rs)
   in
    flat(map (fn mm => apprefs mm reflist)
             k)
   end;
 in
  rd(map (fn mm=> FCtypes (decompose mm))
         (applyrefs (flat (map (fn mk=> Kom mk)
                               (allMKs_of typ)))
                    reflist))
 end;
```

Dabei ist `typ` eine beliebige f-c–Struktur $[\mathtt{fc}_1, \ldots, \mathtt{fc}_{\binom{n}{2}}] \in \mathtt{allFCs(n)}$, `reflist` eine beliebig lange Liste von unären Matrixoperationen, hier insbesondere der Güntherschen Reflektoren $R^3 = \{\mathtt{r}[1], \mathtt{r}[2], \mathtt{r}[3], \mathtt{r}[1,2], \mathtt{r}[1,3], \mathtt{r}[2,3], \mathtt{r}[1,2,3]\}$.

Beispiel:

```
- mapsto [F,F,C] [r1,r2,r3] FCtype;
> [[F,C,F][C,F,F][F,F,C]] : fc list list
```

Die Funktion `mapsto` berechnet also die f-c-Klassen, auf die die Ausgangsklasse `typ` durch Anwendung der Operationen in `reflist` abgebildet werden. Auf dieser einfachen Abbildungsklassifikation aufbauend läßt sich nun die f-c–Analyse eines Q^n Systems wie folgt darstellen:

```
fun analyze MKtypes reflist mgtype =
   map (fn strukt => mapsto strukt reflist mgtype)
       MKtypes;

val R3= RG 3;

- val FCanalyse_emp = analyze (allFCs 3) R3 FCtype;
> val FCanalyse_emp =
  [[[C,C,C],[C,C,C],[C,C,C],[C,C,C],[C,C,C],[C,C,C],[C,C,C]],
   [[C,F,F],[F,F,C],[F,C,F],[F,C,F],[F,C,F],[F,C,F],[F,C,F]],
   [[F,C,F],[C,F,F],[F,F,C],[F,F,C],[F,F,C],[F,F,C],[F,F,C]],
   [[F,F,C],[F,C,F],[C,F,F],[C,F,F],[C,F,F],[C,F,F],[C,F,F]],
   [[F,F,F],[F,F,F],[F,F,F],[F,F,F],[F,F,F],[F,F,F],[F,F,F]]]
  : fc list list list list
```

Das Ergebnis dieser Analyse ist übersichtlich in der folgenden Tabelle 7.1 zusammengestellt:
Diese Abbildung läßt sich auch als Graph (Abbildung 7.2) veranschaulichen. Die Zahlen $1, \ldots, 7$ an den Pfeilen sowie über oder unter den Kästchen symbolisieren in dieser Darstellung die sieben Reflektoren aus R^3. Die Pfeile notieren den Übergang von einer Klasse in eine andere durch Anwendung der jeweiligen Reflektoren. Die Zahlen direkt an den Kästchen notieren Reflektoren, die eine bestimmte Klasse auf sich selbst abbilden. Aus dieser Darstellung ist sofort ersichtlich, daß Q^3_{fc} bezüglich der Güntherschen Reflektoren R^3 in drei disjunkte Unterstrukturen zerfällt und daß Q^3_{ccc} und Q^3_{fff} bezüglich R^3 abgeschlossen sind.

	1 r1	2 r2	3 r3	4 r12	5 r13	6 r23	7 r123
ccc	ccc	ccc	ccc	ccc	ccc	ccc	ccc
cff	cff	ffc	fcf	fcf	fcf	fcf	fcf
ffc	fcf	cff	ffc	ffc	ffc	ffc	ffc
fcf	ffc	fcf	cff	cff	cff	cff	cff
fff	fff	fff	fff	fff	fff	fff	fff

Abbildung 7.1: Die f-c–Analyse von Q^3

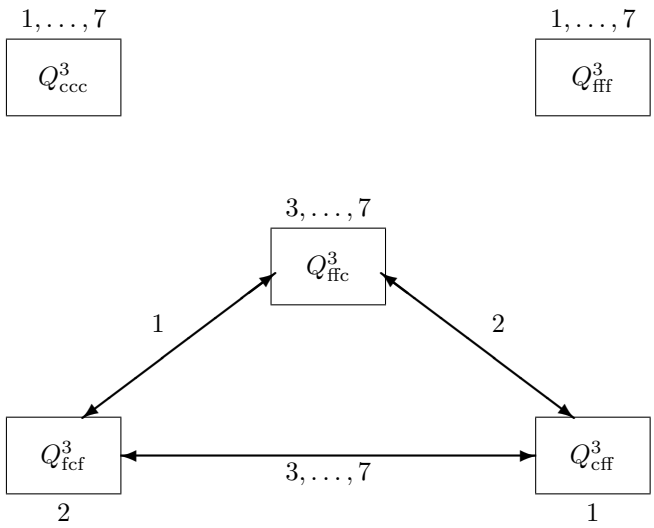


Abbildung 7.2: Graph der f-c–Analyse von Q^3

Die bis hierhin entwickelte empirische Analyse ist mit extremen Speicherplatz– und Rechenzeitaufwand verbunden und läßt sich schon für Q^4_{fc} nicht mehr faktisch realisieren:

Ein Q^n System enthält

$$|Q^n_{MM}| = \sum_{k=1}^{n^2} S(n^2, k) \tag{7.3}$$

verschiedene Morphogrammatrizen.

Q^3_{MM} enthält $\sum_{k=1}^{3^2} S(3^2,k) = 21\ 147$ verschiedene Matrizen[2]. Eine Tabelle, die das Verhalten dieser Matrizen bezüglich der 7 Reflektoren darstellt, muß $21\ 147 \times 7 = 148\ 029$ Einträge aufweisen. Die Berechnung dieser 148 029 Reflektoranwendungen benötigte für die empirische Analyse von Q^3_{fc} ca. 76 Minuten Rechenzeit auf einem NeXT-Cube Rechner.
Q^4_{MM} enthält $\sum_{k=1}^{4^2} S(4^2,k) \approx 1,048 \times 10^{10}$ verschiedene Matrizen und besitzt 65 Reflektoren. Eine Matrix, die das Verhalten all dieser Matrizen bezüglich der Reflektoren angeben soll, besäße ca. $6,812 \times 10^{11}$ Einträge, deren Berechnung auf dem NeXT-Cube mehr als **650** Jahre Rechenzeit benötigte. Diese aus dem kombinatorischen Verhalten morphogrammatischer Strukturen resultierenden astronomischen Summen, die schon für $n > 3$ eine faktische Berechnung verunmöglichen, verdeutlichen die Notwendigkeit abstraktiver Verfahren zur Handhabung morphogrammatischer Objekte und Operationen. Aus diesem Grund schlägt Kaehr[3] eine *abstraktive* Analyse vor, die im folgenden Abschnitt rekonstruiert wird.

7.1.2 Abstraktive Analyse

Der Hauptaufwand der empirischen Analyse bestand darin, daß für jede f-c–Struktur (d.h. jede Hauptdiagonalklasse) alle Morphogrammketten und für jede dieser Ketten alle möglichen polysemen Morphogrammatrizen erzeugt wurden, auf die dann erst die Reflektoroperationen angewandt wurden.
Die f-c–Analyse gibt nur Auskunft über das Verhalten der Hauptdiagonalelemente bezüglich der Reflektoranwendungen. Dieses Verhalten ist jedoch unabhängig von den konkreten Morphogrammketten einer bestimmten f-c–Klasse und insbesondere unabhängig von den Matrixpolysemen der einzelnen Ketten.
Kaehr schlägt daher eine abstraktive f-c–Analyse vor, die das Hauptdiagonalverhalten der Reflektoren nicht empirisch aus den Matrixoperationen ermittelt, sondern direkt aus den Reflektordefinitionen extrahiert. Diese Exktraktion des Hauptdiagonal– oder f-c–Verhaltens eines Reflektors `r` bezüglich eines bestimmten f-c–Typs `fctyp` wird durch die Funktion `fcref r fctyp` geleistet. Dabei wird zunächst die einfachste zu $Q^n_{\texttt{fctyp}}$ gehörende Morphogrammkette gebildet, diese wird komponiert und das einfachste Polysem der komponierten Matrizen als Repräsentant für $Q^n_{\texttt{fctyp}}$ an die Variable `mat` gebunden. Auf diese Matrix wird der Reflektor `r` angewandt, die resultierende Matrix wird dekomponiert und der f-c–Typ dieser Morphogrammkette ermittelt:

```
fun fcref r fctyp=
 let
  val mat= hd(Kom(map (fn fc =>
                                  if fc=C then [1,1,1,1]
                                          else [1,1,1,2])
                      fctyp));
  in
   FCtypes(decompose (r mat))
  end;
```

[2] Die verwendete Formel zur Anzahlbestimmung erzeugbarer Matrizen geht davon aus, daß alle n^2 möglichen Kenogramme benutzt werden, (dies entspricht der Anzahl $\overline{\mu}$ in Formel (23) auf S. 111 in [Gue80b1]). Werden jedoch höchstens n–wertige Matrizen zugelassen, sind nur $\mu(n) = \sum_{k=1}^{n} S(n^2,k)$ Matrizen erzeugbar, (Formel (20) a.a.O). Für ein dreiwertiges System ergibt dies $\mu(3) = 3281$ Matrizen. Die Untersuchungen Kaehrs ([Kae74], [Kae78]) gehen im Gegensatz zu der vorliegenden Arbeit von höchstens dreiwertigen Q^3-Matrizen aus.

[3] [Kae78], S.95.

Die abtraktive f-c–Analyse kann also dargestellt werden als:

```
fun fcanalyse types reflist=
  map (fn fctyp =>
        map(fn r => fcref r fctyp)
           reflist)
      types;

- val FCanalyse_abs =(fcanalyse (allFCs 3) R3);
> val it =
  [[[C,C,C],[C,C,C],[C,C,C],[C,C,C],[C,C,C],[C,C,C],[C,C,C]],
   [[C,F,F],[F,F,C],[F,C,F],[F,C,F],[F,C,F],[F,C,F],[F,C,F]],
   [[F,C,F],[C,F,F],[F,F,C],[F,F,C],[F,F,C],[F,F,C],[F,F,C]],
   [[F,F,C],[F,C,F],[C,F,F],[C,F,F],[C,F,F],[C,F,F],[C,F,F]],
   [[F,F,F],[F,F,F],[F,F,F],[F,F,F],[F,F,F],[F,F,F],[F,F,F]]]
  : fc list list list
```

Diese Analyse verläuft folgendermaßen: die Hauptiagonaltypen der Länge n werden durch `allFCs(n)` erzeugt (vgl. 6.1.1). Für jeden dieser f-c–Typen `fctyp` wird nun für jeden Reflektor `r` $\in$ `reflist` das f-c–Verhalten `(fcref r fctyp)` ermittelt.
Die abstraktive Analyse benötigt nur einen Bruchteil des Resourcenaufwandes[4] der empirischen Analyse, gelangt jedoch wie zu erwarten zum gleichen Resultat:

```
- FCanalyse_emp = FCanalyse_abs;
> true : bool
```

Aufgrund dieser Gleichheit wird hier auf eine gesonderte tabellarische Darstellung der abstraktiven f-c–Analyse von Q^3_{fc} verzichtet, da sie identisch mit der in Tabelle 7.1 aufgeführten empirischen Analyse ist.

7.2 Die g-h–Analyse

Die g-h–Klassifikation von Q bezieht sich auf die Nebendiagonalstellen d er Morphogramme. Morphogrammketten sind unabhängig von ihren Nebendiagonalstellen und daher von ihrer g-h–Struktur zu Q^n Matrizen komponierbar. Im Gegensatz zur f-c–Klassifikation sind also alle $2^{\binom{n}{2}}$ Kombinationen von g– und h–Morphogrammen möglich. Die Anzahl von g-h–Typen für Q^n_{gh} wird mit $|Q^n_{gh}|$ bezeichnet:

$$|Q^n_{gh}| = 2^{\binom{n}{2}} \tag{7.4}$$

```
fun allGHs (n)=
 let
  fun comb []= [[]]
     |comb [l]= map (fn x => [x]) l
     |comb (hd::tl)=
       flat(map (fn mg => combine mg (comb tl))
                hd);
 in
```

[4] Rechenzeit für $Q^3_{fc} < 5$ sec.

```
  comb (nlistof (choose n 2) [G,H])
 end;

- allGHs 3;
> val it = [[G,G,G],[G,G,H],[G,H,G],[G,H,H],
            [H,G,G],[H,G,H],[H,H,G],[H,H,H]] : gh list list
```

Die Klasse aller zu einem bestimmten g-h–Typen $[gh_1, \ldots, gh_{\binom{n}{2}}]$ gehörenden Morphogrammketten wird mit $Q^n_{gh_1 \ldots gh_{\binom{n}{2}}}$ bezeichnet. Die Klasse aller g-h–Morphogrammketten von Q^3, Q^3_{gh}, ist beispielsweise gegeben als:

$$Q^3_{gh} = Q_{\mathrm{ggg}} \cup Q_{\mathrm{ggh}} \cup Q_{\mathrm{ghg}} \cup Q_{\mathrm{ghh}} \cup Q_{\mathrm{hgg}} \cup Q_{\mathrm{hgh}} \cup Q_{\mathrm{hhg}} \cup Q_{\mathrm{hhh}}.$$

Die Funktion `Gmg M` testet, ob das Morphogramm `M` vom g-h–Typ g ist, ebenso testet `(Hmg M)` ob `M` vom Typ h ist:

```
fun Gmg [_,w12,w21,_]= w12<>w21;
fun Hmg [_,w12,w21,_]= w12=w21;
```

Die Menge Q_g enthält alle Basismorphogramme vom Typ g, Q_h enthält entsprechend die h-Typ Morphogramme aus Q:

```
val Qg = allof Gmg Q;
val Qh = allof Hmg Q;
```

Zur Durchführung einer abstraktiven g-h–Analyse wird zunächst eine Abbildung `ghref r ghtyp` definiert, die das Nebendiagonalverhalten eines Reflektors `r` bezüglich des g-h–Typs `ghtyp` ermittelt, indem sie den aus der Reflektoranwendung resultierenden g-h–Typ bestimmt:

```
fun ghref r ghtyp=
 let
  val mat= hd(Kom(map (fn gh =>
                    if gh=G then [1,1,2,1]
                    else [1,1,1,1])
                  ghtyp));
  in
   GHtypes(decompose (r mat))
  end;
```

In Analogie zur f-c–Analyse kann die g-h–Analyse implementiert werden als:

```
fun ghanalyse types reflist=
  map (fn ghtyp =>
        map(fn r => ghref r ghtyp)
           reflist)
      types;
```

Die g-h–Analyse von Q^3_{gh} ist dann gegeben als:

```
- val GHanalyse =ghanalyse (allGHs 3) R3;
> val it =
```

```
[[[G,G,G],[G,G,G],[G,G,G],[G,G,G],[G,G,G],[G,G,G],[G,G,G]],
 [[G,G,H],[G,G,H],[G,G,H],[G,G,H],[G,G,H],[G,G,H],[G,G,H]],
 [[G,H,G],[G,H,G],[G,H,G],[H,G,G],[G,G,G],[H,H,G],[H,G,G]],
 [[G,H,H],[G,H,H],[G,H,H],[H,G,H],[G,G,H],[H,H,H],[H,G,H]],
 [[H,G,G],[H,G,G],[H,G,G],[G,H,G],[H,H,G],[G,G,G],[G,H,G]],
 [[H,G,H],[H,G,H],[H,G,H],[G,H,H],[H,H,H],[G,G,H],[G,H,H]],
 [[H,H,G],[H,H,G],[H,H,G],[H,H,G],[H,H,G],[H,H,G],[H,H,G]],
 [[H,H,H],[H,H,H],[H,H,H],[H,H,H],[H,H,H],[H,H,H],[H,H,H]]]
 : gh list list list
```

Das Ergebnis dieser Analyse ist übersichtlich in der folgenden Tabelle 7.3 zusammengestellt:

	1 r1	2 r2	3 r3	4 r12	5 r13	6 r23	7 r123
ggg	ggg	ggg	ggg	ggg	ggg	ggg	ggg
hhg	hhg	hhg	hhg	hhg	hhg	hhg	hhg
ghg	ghg	ghg	ghg	hgg	ggg	hhg	hgg
hgg	hgg	hgg	hgg	ghg	hhg	ggg	ghg
hhh	hhh	hhh	hhh	hhh	hhh	hhh	hhh
ggh	ggh	ggh	ggh	ggh	ggh	ggh	ggh
hgh	hgh	hgh	hgh	ghh	hhh	ggh	ghh
ghh	ghh	ghh	ghh	hgh	ggh	hhh	hgh

Abbildung 7.3: Die abstraktive g-h–Analyse von Q^3

Die Zweiteilung der Tabelle hebt deutlich die Dualität von g– und h–Typen hervor, die die Struktur Q^3_{gh} in zwei disjunkte und duale Teilstrukturen zerlegt. Der Graph von Q^3_{gh} (Abbildung 7.4) veranschaulicht diese Struktur. Die Güntherschen Reflektoren R^3 zerlegen also Q^3_{gh} in zwei zueinander duale Teilsysteme. hgg, ghg und ghh, hgh sind die Quellen, ggg, hhg und hhh, ggh sind die Senken des Graphen.

7.3 Die k-l-o-r–Analyse

7.3.1 Abstraktive Analyse

In diesem Abschnitt soll zunächst in Anlehnung an die c-f– und g-h–Analysen die abstraktive Analyse für die k-l-o-r–Klassifikation durchgeführt werden. Im nächsten Abschnitt soll die Struktur der k-l-o-r–Analyse von Q^3_{klor} als graphentheoretische Verknüpfung hergeleitet werden.
Die Funktion `(Kmg M)` testet, ob das Morphogramm `M` vom k-l-o-r–Typ k ist, entsprechend führen die Funktionen `Lmg, Omg, Rmg` diesen Test für die Typen l, o und r durch:

```
fun Kmg [w11,w12,w21,w22]= (w11<>w22) andalso (w12<>w21);
fun Lmg [w11,w12,w21,w22]= (w11<>w22) andalso (w12=w21);
fun Omg [w11,w12,w21,w22]= (w11=w22) andalso (w12=w21);
fun Rmg [w11,w12,w21,w22]= (w11=w22) andalso (w12<>w21);
```

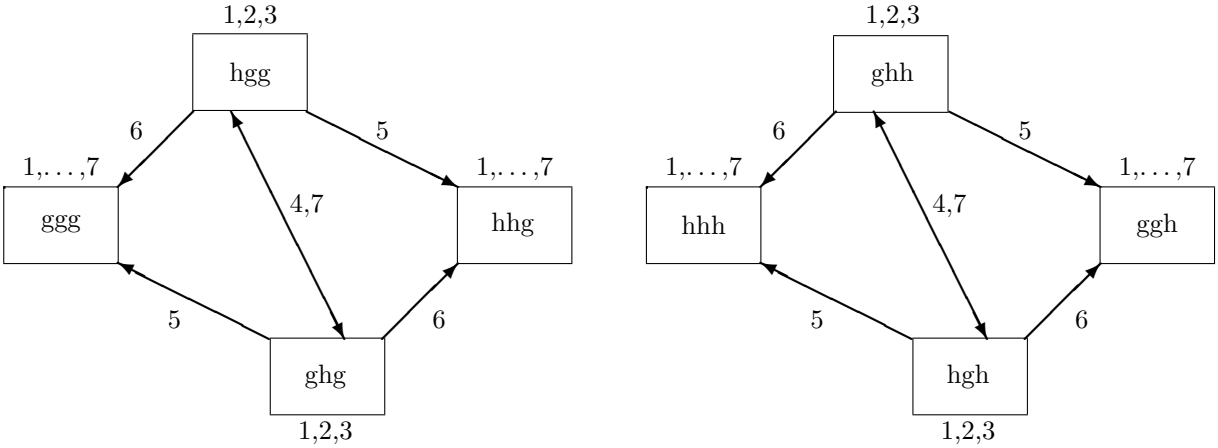


Abbildung 7.4: Der Graph der g-h–Analyse von Q^3

Die Mengen Q_k, Q_l, Q_o, Q_r enthalten jeweils alle Basismorphogramme der Typen k,l,o und r:

```
val Qk = allof Kmg Q;
val Ql = allof Lmg Q;
val Qo = allof Omg Q;
val Qr = allof Rmg Q;
```

Da $Q_c = Q_o \cup Q_r$ und $Q_f = Q_l \cup Q_k$, sind nicht alle erzeugbaren Kombinationen von k-l-o-r–Morphogrammen zu Morphogrammatrizen komponierbar. Die Menge aller komponierbaren k-l-o-r–Strukturen von Q^n, Q^n_{klor}, wird durch die Funktion `allKLORs(n)` berechnet:

```
fun allKLORs(n) =
  flat(map comb
           (map (fn fcstr => map (fn fc =>if fc=F then [L,K]
                                          else [O,R])
                                 fcstr)
                (allFCs n) ));
```

```
- allKLORs 3;
> val it =
  [[O,O,O],[O,O,R],[O,R,O],[O,R,R],[R,O,O],[R,O,R],[R,R,O],[R,R,R],
   [O,L,L],[O,L,K],[O,K,L],[O,K,K],[R,L,L],[R,L,K],[R,K,L],[R,K,K],
   [L,L,O],[L,L,R],[L,K,O],[L,K,R],[K,L,O],[K,L,R],[K,K,O],[K,K,R],
   [L,O,L],[L,O,K],[L,R,L],[L,R,K],[K,O,L],[K,O,K],[K,R,L],[K,R,K],
   [L,L,L],[L,L,K],[L,K,L],[L,K,K],[K,L,L],[K,L,K],[K,K,L],[K,K,K]]
  : klor list list
```

Zur Durchführung der abstraktiven Analyse wird zunächst die Funktion `klorref r klortyp` implementiert, die das Abbildungsverhalten des Reflektors `r` bezüglich des Typs `klortyp` ermittelt:

```
fun klorref r klortyp=
 let
  val mat= hd(Kom(map (fn klor =>
                     if klor=K then [1,1,2,2]
                else if klor=L then [1,1,1,2]
                else if klor=O then [1,1,1,1]
                else                [1,1,2,1])
                   klortyp));
  in
   KLORtypes(decompose (r mat))
  end;
```

Die k-l-o-r–Analyse kann dann in Analogie zu den vorhergehenden Analysen folgendermaßen implementiert werden:

```
fun kloranalyse types reflist=
  map (fn klortyp =>
        map(fn r => klorref r klortyp)
           reflist)
      types;
```

Beispiel: Das Ergebnis der k-l-o-r–Analyse von Q^3, `kloranalyse (allklors 3) R3;` ist in der Tabelle 7.5 übersichtlich zusammengestellt. Die fünf durch horizontale Linien abgeteilten Blöcke von jeweils acht k-l-o-r–Typen entsprechen den fünf f-c–Typen ccc, cff, ffc, fcf und fff.

7.3.2 Die graphentheoretische Komposition der k-l-o-r–Analyse

Aus der Klassifikationstafel 6.1 auf Seite 119 ist ersichtlich, daß die vier Klassen Q_k, Q_l, Q_o, Q_r als Schnittmengenbildungen der f-c– und der g-h–Klassen dargestellt werden können:

$$\begin{aligned} Q_k &= Q_f \cap Q_g \\ Q_l &= Q_f \cap Q_h \\ Q_o &= Q_c \cap Q_h \\ Q_r &= Q_c \cap Q_g \end{aligned}$$

Aufgrund dieser Beziehungen kann auch die Q^n_{klor}–Klassifikation mittels der Schnittmengenprodukte $Q^n_{fc} \bigcap Q^n_{gh}$ definiert werden.

Beispiel:

$$\begin{aligned} Q^3_{klor} &= Q^3_{fc} \bigcap Q^3_{gh} \\ Q^3_{fc} &= \{\text{ccc}, \text{fff}, \text{cff}, \text{fcf}, \text{ffc}\} \\ Q^3_{gh} &= \{\text{ggg}, \text{hgg}, \text{hhg}, \text{ghg}, \text{hhh}, \text{ghh}, \text{ggh}, \text{hgh}\} \end{aligned}$$

Die Schnittmengenbildung von Q^3_{fc} und Q^3_{gh} ist in der folgenden Tabelle abgebildet:

$\bigcap$	ggg	ggh	ghg	ghh	hgg	hgh	hhg	hhh
ccc	rrr	rro	ror	roo	orr	oro	oor	ooo
cff	rkk	rkl	rlk	rll	okk	okl	olk	oll
ffc	kkr	kko	klr	klo	lkr	lko	llr	llo
fcf	krk	krl	kok	kol	lrk	lrl	lok	lol
fff	kkk	kkl	klk	kll	lkk	lkl	llk	lll

Die Anzahl der k-l-o-r–Typen $|Q^n_{klor}|$ für Q^n beträgt, wie dieser Schnittmengendarstellung zu entnehmen ist, genau:

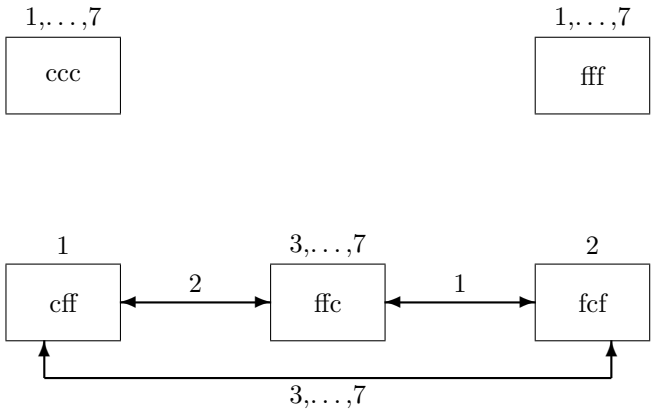
$$|Q^n_{klor}| \quad = \quad |Q^n_{gh}| \times |Q^n_{fc}| \quad = \quad 2^{\binom{n}{2}} \times \sum_{k=1}^{n} S(n,k). \tag{7.5}$$

Beispiel:

$$|Q^3_{klor}| = 2^{\binom{3}{2}} \times \sum_{k=1}^{3} S(3,k) \quad = \quad 40$$

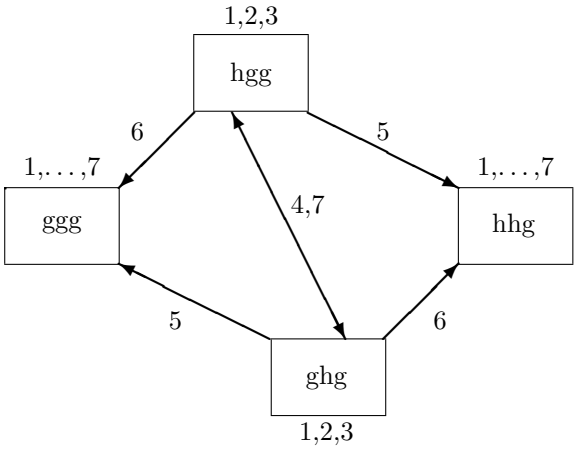
Der Graph der Q^n_{klor} Analyse kann entsprechend als Verknüpfung der Graphen von Q^n_{fc} und Q^n_{gh} dargestellt werden[5]. Die Nützlichkeit eines solchen Verknüpfungsverfahrens besteht darin, daß komplexe Strukturen, wie beispielsweise (Q^n_{klor}, R^3) als Verknüpfung einfacher Strukturen definiert und dargestellt werden können.

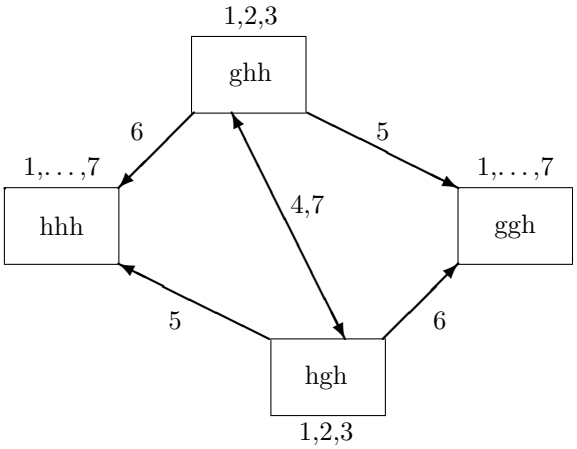
Beispiel: Der Graph von Q^3_{klor} soll aus den Graphen von Q^3_{fc} und Q^3_{gh} konstruiert werden. Q^3_{fc} zerfällt in drei disjunkte Teilsysteme Q^3_{ccc}, Q^3_{fff} und $Q^3_{\text{cff}} \cup Q^3_{\text{ffc}} \cup Q^3_{\text{fcf}}$:



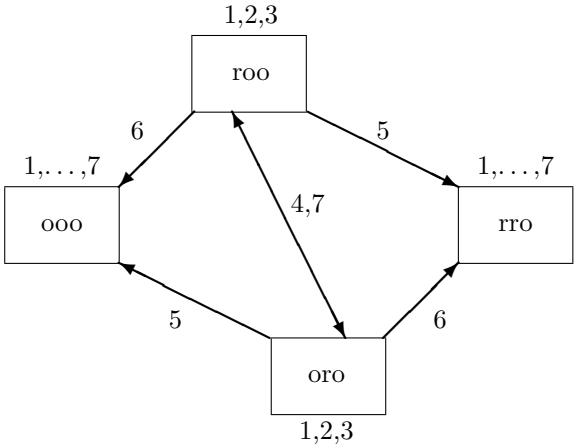
Q^3_{gh} zerfällt in zwei duale Teilsysteme:

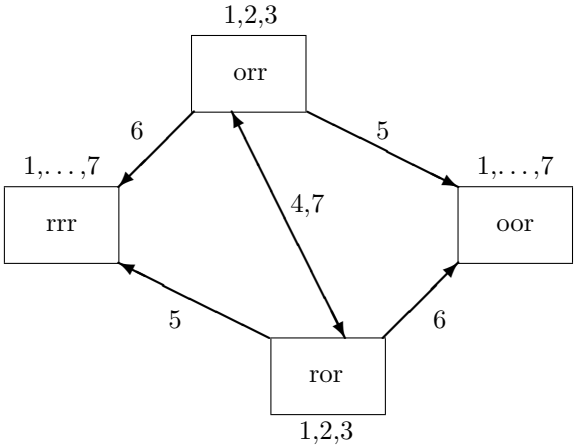
[5] [Kae74], [Kae78].



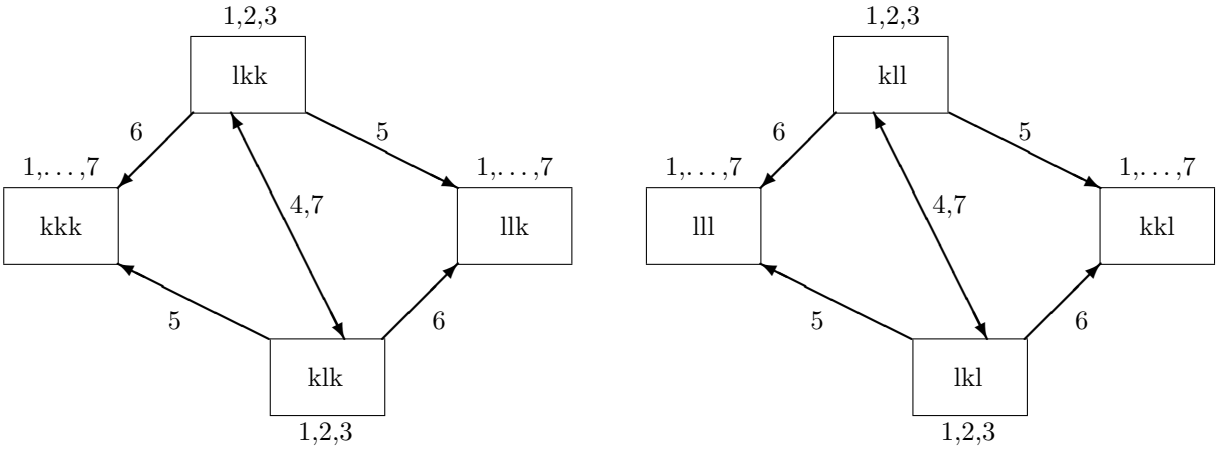


Der erste Block von acht k-l-o-r–Typen in der Tabelle 7.5 auf Seite 145 entspricht der Schnittmengenbildung von Q^3_{ccc} und Q^3_{gh}: $Q^3_{or} = Q^3_{\text{ccc}} \bigcap Q^3_{gh}$. Der Graph dieses Teilsystems von Q^3_{klor} weist sowohl die Eigenschaften von Q^3_{ccc} (Abgeschlossenheit bezüglich der sieben Reflektoren) als auch von Q^3_{gh} (Struktur des Graphen und Dualität von g und h) auf:





Die Schnittmengenbildung $Q^3_{kl} = Q^3_{\text{fff}} \bigcap Q^3_{gh}$ entspricht dem letzten Block von acht k-l-o-r–Typen in Tabelle 7.5. Der Graph dieses Teilsystems von Q^3_{klor} weist sowohl die Eigenschaften von Q^3_{fff} (Abgeschlossenheit bezüglich der sieben Reflektoren) als auch von Q^3_{gh} (Struktur des Graphen und Dualität von g und h) auf:



Die drei mittleren Achterblöcke der Tabelle 7.5 entsprechen dem k-l-o-r–Teilsystem $(Q^3_{\text{cff}} \cup Q^3_{\text{ffc}} \cup Q^3_{\text{fcf}}) \bigcap Q^3_{gh}$. Diese Schnittmengenbildung bewahrt ebenfalls die Struktur von $(Q^3_{\text{cff}} \cup Q^3_{\text{ffc}} \cup Q^3_{\text{fcf}})$ und Q^3_{gh} (Abbildung 7.6). Auf eine formale Definition der Verknüpfungsregeln kann hier aus Platzgründen nicht eingegangen werden.

7.4 Analyse höherwertiger Q^n-Systeme

Die f-c–Analyse für Q^3 ergab, daß die fünf f-c–Typen `allFCs(3)` bezüglich der sieben Reflektoren `RG(3)` in $Dcard(3) = 3$ disjunkte Bereiche zerfallen. Wie im Vorhergehenden gezeigt, ermöglicht es die Disjunktheit der drei Bereiche, die aus der Schnittmengenbildung von f-c– und g-h–Klassifikation abgeleitete k-l-o-r–Klassifikation von Q^3 ebenfalls in drei disjunkte Bereiche zu zerlegen.
Die f-c–Typen [f,f,f] und [c,c,c] werden durch die Reflektoren jeweils auf sich selbst abgebildet. Die Typen [f,c,f], [c,f,f] und [f,f,c] werden durch die Reflektoren entweder auf sich selbst oder auf einen der beiden anderen Typen abgebildet (vgl. Abbildung 7.2 auf Seite 129). Allgemein ausgedrückt, werden die f-c-Strukturen von Q^3 durch die Reflektoren permutiert.
Dies gilt jedoch nicht mehr für höherwertige Q^n-Systeme mit $n > 3$. So existieren für $Q^n, n > 3$, nicht mehr nur Reflektoren, die bezüglich der f-c–Strukturen *permutativ* wirken, sondern auch *reduktive* Reflektoren.

Beispiel:

$$r_{1.6} \begin{pmatrix} \mathbf{1} & 2 & 3 & 4 \\ 5 & \mathbf{6} & 7 & 8 \\ 9 & 10 & \mathbf{11} & 12 \\ 13 & 14 & 15 & \mathbf{16} \end{pmatrix} = \begin{pmatrix} \mathbf{16} & 2 & 3 & 13 \\ 5 & \mathbf{6} & 7 & 8 \\ 9 & 10 & \mathbf{6} & 5 \\ 4 & 14 & 2 & \mathbf{1} \end{pmatrix}$$

Hierbei wird die dritte Position der Hauptdiagonalen (11) durch die zweite Hauptdiagonalstelle (6) überschrieben. Dieses Verhalten wird als Reduktionstyp I bezeichnet.

$$r_{4.6}\begin{pmatrix} \mathbf{1} & 2 & 3 & 4 \\ 5 & \mathbf{6} & 7 & 8 \\ 9 & 10 & \mathbf{11} & 12 \\ 13 & 14 & 15 & \mathbf{16} \end{pmatrix} = \begin{pmatrix} \mathbf{16} & 15 & 3 & 13 \\ 12 & \mathbf{11} & 7 & 8 \\ 9 & 10 & \mathbf{11} & 12 \\ 4 & 14 & 15 & \mathbf{1} \end{pmatrix}$$

Bei dieser Spiegelung wird die zweite Position der Hauptdiagonalen (6) durch die dritte Position (11) überschrieben. Dieses Verhalten wird als Reduktionstyp II bezeichnet.

Aufgrund dieser Reduktivität zerfällt nun der Graph der f-c–Analyse für Q^4 nicht in $Dcard(4) = 5$ disjunkte Teilsysteme, sondern bildet eine zusammenhängende Struktur (Abbildung 7.7 auf Seite 147).
In dieser Abbildung sind die 15 f-c–Typen von Q^4 durch die sie repräsentierenden Hauptdiagonalen der Länge 4, den Basismorphogrammen $(1, \ldots, 15)$ abkürzend notiert. Die Ziffern I und II an den gerichteten Pfeilen geben an, ob die Umformung durch einen Reflektor vom Reduktionstyp I oder II stattfindet. Die ungerichteten Kanten symbolisieren die permutativen Reflektoren, die nicht einzeln numeriert werden, da hier einzig die Struktur und die Reduktionsmöglichkeiten von (Q^n_{fc}, R^n) interessieren.
Aus dem Graphen ist leicht ersichtlich, daß die permutativen Reflektoren stets die Deuterostruktur der Hauptdiagonalen bewahren. Jede Deuteroklasse ist durch einen gestrichelten Kasten und die zugehörige Partitionsstruktur gekennzeichnet. Die Reflektoren vom Reduktionstyp I und II bilden jedoch f-c-Typen einer Deuteroklasse auf Typen anderer Deuteroklassen ab. Deshalb ist der Graph insgesamt zusammenhängend. Aus der Hauptdiagonalen 15 (○△□⋆) lassen sich durch Reflektoranwendung alle anderen Typen ableiten, er ist die Quelle[6] des Graphen. Der Typ 1 (○○○○) läßt sich aus allen anderen Typen durch Reflektoranwendung ableiten, er ist die Senke des Graphen.
Die für (Q^3, R^3) beobachtete Zerlegbarkeit des Gesamtsystems in disjunkte Substrukturen gilt somit nicht allgemein für $n > 3$. Analysen höherwertiger Systeme (Q^n, R^n)-Systeme können daher nicht in disjunkte Teilanalysen aufgegliedert werden.

7.5 Erzeugendensysteme morphogrammatischer Q^n Systeme

Die in den vorhergehenden Abschnitten durchgeführten abstraktiven Analysen bilden die Struktur der reflektionalen Operationen eines Q^n Systems bezüglich bestimmter Klassifikationen ab.
Da eine Analyse morphogrammatischer Operationen auf der Ebene der konkreten Matrixpolyseme von Q^n im allgemeinen[7] an der Komplexität dieser Systeme scheitert, erweist sich die abstraktive Analyse anhand bestimmter Klassifikationen als wirksame Komplexitätsreduktion der durchzuführenden Analysen. Anhand der Analyse des Verhaltens der Reflektoren R^n bezüglich der Morphogrammkettenstruktur Q^n_{MK} von Q^n soll hier eine weitere Analysetechnik eingeführt werden, die Bestimmung von Erzeugendensystemen.

[6] Die Begriffe Quelle und Senke des Graphen werden im nächsten Abschnitt definiert.
[7] Ab $n > 3$.

Diese Morphogrammkettenanalyse von Q^n (oder bestimmten Teilsystemen von Q^n) gibt Auskunft über die gegenseitige Definierbarkeit und Darstellbarkeit der Morhogrammketten. Entsprechend zur klassischen Aussagenlogik lassen sich Fragen der Definierbarkeit und der funktionalen Vollständigkeit beantworten. Da schon Q^3 2625 Morphogrammketten enthält, ist eine tabellarische Aufstellung nicht mehr übersichtlich möglich. Aus diesem Grund erweisen sich graphentheoretische Analysen zur Lösung der angesprochenen Fragen als notwendig.
Das Verhaltens der einzelnen Morphogrammketten aus Q^n_{MK} bezüglich der Reflektoren R^3 läßt sich wie folgt bestimmen:

```
fun mknums mk=
 let
  fun mgnr m=
   let
    exception Findplace
    val ns = fromto 1 15
    fun findplace m [] = raise Findplace
       |findplace m (n::ns)= if (m=mg n) then n
                             else findplace m ns;
   in
    findplace m ns
   end;
 in
  map mgnr mk
 end;

fun MKanalyse MKs=
  map (fn MKi =>
        flat(map(fn r => (rd(map(fn MKmat=>
                                    mknums(decompose(r MKmat)))
                             (Kom MKi))))
                 R3))
       MKs;
```

Die Funktion `mknums mk` wandelt zur besseren Lesbarkeit eine Morphogrammkette `mk` in eine Liste der Nummern der in `mk` enthaltenen Morphogramme um (vgl. Abbildung 5.5 auf Seite 90). Die eigentliche Analyse verläuft folgendermaßen: für jede Morphogrammkette `MKi` in `MKs` werden alle erzeugbaren Morphogrammmatrizen `Kom MKi` komponiert. Auf jede dieser Matrizen `MKmat` werden alle Reflektoren `r` aus R^3 angewandt. Die resultierenden Matrizen werden zu Morphogrammketten dekomponiert, deren Morphogrammnummern ermittelt werden.
Zum Auffinden eines Erzeugendensystems von Q^n muß die *minimale Quellenmenge* des Verknüpfungsgraphen bestimmt werden. Hierzu muß eine Menge von Morphogrammketten aus Q^n oder einem Teilsystem gefunden werden, aus denen alle übrigen Ketten durch wiederholte Anwendung der Reflektoren R^3 erzeugt werden können. Die Menge, die die wenigsten Elemente enthält, ist die minimale Quellenmenge. Der *Quellenbereich* einer Morphogrammkette enthält alle durch iterative Reflektoranwendung aus ihr erzeugbaren Ketten. Hiermit läßt sich eine Quelle wie folgt definieren:

Definition 7.1 (Quelle) *Eine Morphogrammkette $mk \in Q^n$ ist eine Quelle des Analysegraphen $G(Q^n_{MK})$, gdw. mk entweder aus keiner oder nur aus Morphogrammketten des Quellenbereiches von mk durch wiederholte Reflektoranwendung erzeugt werden kann.*

Sind die Quellen–Morphogrammketten bekannt, läßt sich aus ihnen das gesamte Q^n System (oder das jeweilige Teilsystem) erzeugen. Die Quellenmorphogramme bilden dann zusammen mit der Komposition `Kom` und den Reflektoren R^3 das *Erzeugendensystem* von Q^n.
Mittels der Funktion `makegraph` wird die `MKanalyse` eines Q^n Systems in eine Graphendarstellung überführt:

```
fun makegraph qs []=[]
   |makegraph [] ms =[]
   |makegraph (q::qs) (m::ms)=
        (q,m)::makegraph qs ms;
```

Die Bestimmung einer minimalen Quellenmenge eines Graphen `Graph` erfolgt durch die Funktion `Minquell Graph`[8]:

```
fun Minquell Graph =
  let
   fun remnils []=[]
      |remnils (hd::tl) = if hd=[] then remnils tl
                          else hd::(remnils tl);
   fun step [] Q = Q
      |step ((name,cons)::Graph) Q =
        step Graph (remnils
                   (map (fn q=> if (member q cons)
                                  andalso (q<>name) then []
                                else q)
                        (name::Q)));
  in
   step Graph []
  end;
```

Der Algorithmus zur Bestimmung der minimalen Quellenmenge funktioniert folgendermaßen. Der Graph `Graph` ist als eine Liste von Knoten `(name,cons)` repräsentiert. `name` enthält die Notierung einer bestimmten Morphogrammkette (z.B. `[2,10,11]`), `cons` ist die Liste aller Morphogrammketten, die von `name` aus durch Reflektoranwendungen erreicht werden können. In der anfangs leeren Liste `Q` werden von der Funktion `step` die Quellenmorphogrammketten gesammelt. Falls der Graph bis zum Ende durchlaufen ist, wird `Q` als Ergebnis zurückgegeben. Sonst wird der erste Konoten `(name,cons)` von `Graph` untersucht. Dabei wird `name` versuchsweise der Menge `Q` hinzugefügt. Jedes der Quellenmorphogramme `q` wird dann aus `name::Q` entfernt, falls es von `name` aus erreicht werden kann (`member q cons`) und nicht gleichzeitig `name` selber ist (zyklische Strukturen würden sonst nicht korrekt abgearbeitet). Der restliche Graph `Graph` wird mit der so veränderten Quellenliste sukzessive weiterdurchlaufen.

[8] [Kae74], S.56f.

Beispiel: Gesucht wird ein minimales Erzeugendensystem für Q^3_{lll}. Dieses Teilsystem von Q^3 wurde gewählt, da Q_l nur drei Morphogramme enthält und Q^3_{lll} eine Senke von Q^3_{kl} (d.h. bezüglich der Reflektoren R^3 abgeschlossen) ist[9]. Q_l enthält die Morphogramme ○○○△, ○△△△ und ○△△□:

$$Q_l = \{mg_2, mg_{10}, mg_{11}\}.$$

Q^3_{lll} enthält alle $3^3 = 27$ möglichen Kombinationen der Länge 3 von Q_l:

```
- val Qlll= comb [Ql,Ql,Ql];
> val it =
 [[[1,1,1,2],[1,1,1,2],[1,1,1,2]],[[1,1,1,2],[1,1,1,2],[1,2,2,2]],
  [[1,1,1,2],[1,1,1,2],[1,2,2,3]],[[1,1,1,2],[1,2,2,2],[1,1,1,2]],
  [[1,1,1,2],[1,2,2,2],[1,2,2,2]],[[1,1,1,2],[1,2,2,2],[1,2,2,3]],
  [[1,1,1,2],[1,2,2,3],[1,1,1,2]],[[1,1,1,2],[1,2,2,3],[1,2,2,2]],
  [[1,1,1,2],[1,2,2,3],[1,2,2,3]],[[1,2,2,2],[1,1,1,2],[1,1,1,2]],
  [[1,2,2,2],[1,1,1,2],[1,2,2,2]],[[1,2,2,2],[1,1,1,2],[1,2,2,3]],
  [[1,2,2,2],[1,2,2,2],[1,1,1,2]],[[1,2,2,2],[1,2,2,2],[1,2,2,2]],
  [[1,2,2,2],[1,2,2,2],[1,2,2,3]],[[1,2,2,2],[1,2,2,3],[1,1,1,2]],
  [[1,2,2,2],[1,2,2,3],[1,2,2,2]],[[1,2,2,2],[1,2,2,3],[1,2,2,3]],
  [[1,2,2,3],[1,1,1,2],[1,1,1,2]],[[1,2,2,3],[1,1,1,2],[1,2,2,2]],
  [[1,2,2,3],[1,1,1,2],[1,2,2,3]],[[1,2,2,3],[1,2,2,2],[1,1,1,2]],
  [[1,2,2,3],[1,2,2,2],[1,2,2,2]],[[1,2,2,3],[1,2,2,2],[1,2,2,3]],
  [[1,2,2,3],[1,2,2,3],[1,1,1,2]],[[1,2,2,3],[1,2,2,3],[1,2,2,2]],
  [[1,2,2,3],[1,2,2,3],[1,2,2,3]]] : int list list list
```

Der Graph der Morphogrammkettenanalyse von Q_{lll}, `Glll`, wird folgendermaßen berechnet:

```
- val Glll = makegraph (map mknums Qlll) (MKanalyse Qlll);
> val Glll =
[([2,2,2],[[10,11,11],[2,10,2],[11,2,10],[10,10,10],[11,10,10],
  [10,2,10],[10,10,10]]),
 ([2,2,10],[[10,11,10],[2,10,11],[11,2,2],[10,10,2],[11,10,2],
  [10,2,2],[10,10,2]]),
 ([2,2,11],[[10,11,2],[10,11,11],[2,10,10],[2,10,11],
  [11,2,11],[10,10,11],[11,10,11],[10,2,11],[10,10,11]]),
 ([2,10,2],[[10,10,11],[2,2,2],[11,11,10],[2,10,10],[11,10,10],
  [2,11,10],[2,10,10]]),
 ([2,10,10],[[10,10,10],[2,2,11],[11,11,2],[2,10,2],[11,10,2],
  [2,11,2],[2,10,2]]),
 ([2,10,11],[[10,10,2],[10,10,11],[2,2,10],[2,2,11],[11,11,11],
  [2,10,11],[11,10,11],[2,11,11],[2,10,11]]),
 ([2,11,2],[[10,2,11],[10,11,11],[2,11,2],[11,10,10],[11,11,10],
  [11,10,10],[11,10,10],[11,10,10],[11,11,10],[11,10,10]]),
 ([2,11,10],[[10,2,10],[10,11,10],[2,11,11],[11,10,2],[11,11,2],
  [11,10,2],[11,10,2],[11,10,2],[11,11,2],[11,10,2]]),
 ([2,11,11],[[10,2,2],[10,2,11],[10,11,2],[10,11,11],[2,11,10],
  [2,11,11],[11,10,11],[11,11,11],[11,10,11],[11,10,11],
  [11,10,11],[11,11,11],...]),
 ([10,2,2],[[2,11,11],[11,10,2],[10,2,10],[10,2,10],[10,2,10],
  [10,2,10],[10,2,10]]),
```

[9] Vgl. Abbildung 7.5 auf Seite 145.

```
 ([10,2,10],[[2,11,10],[11,10,11],[10,2,2],[10,2,2],[10,2,2],
  [10,2,2],[10,2,2]]),
 ([10,2,11],[[2,11,2],[2,11,11],[11,10,10],[11,10,11],[10,2,11],
  [10,2,11],[10,2,11],[10,2,11],[10,2,11]]),...]
: (int list * int list list) list
```

Die minimale Quellenmenge von `Glll` wird berechnet als:

```
- val MinQlll = Minquell Glll;
> val MinQlll = [[11,11,11]] : int list list
```

Die Klasse der 27 Morphogrammketten in Q^3_{lll} läßt sich also mit dem Erzeugendensystem $(\{mg_{11}\}, \texttt{Kom}, R^3)$ darstellen. In [Kae78][10] wird eine Quellenanalyse des Q_{lll} Systems durchgeführt, die nur Polyseme bis zum Akkretionsgrad 3 berücksichtigt. Diese Untersuchung ergibt $\{[10,10,10],[11,11,11]\}$ als minimale Quellenmenge. Die Einschränkung bezüglich des Akkretionsgrades der Matrixpolyseme ist die Ursache dieses abweichenden Ergebnisses.

Die Bestimmung einer minimalen Quellenmenge des gesamten Q^3 Systems verläuft wie folgt. Q^3 zerfällt in drei disjunkte Substrukturen, Q^3_{ccc}, Q^3_{fff} und $Q^3_{\text{fc}} = Q^3_{\text{cff}} \cup Q^3_{\text{ffc}} \cup Q^3_{\text{fcf}}$. Für jede dieser Substrukturen muß daher eine gesonderte Quellenanalyse durchgeführt werden. Zunächst werden die Mengen aller möglichen Morphogrammketten in den disjunkten Substrukturen von Q^3 erzeugt. `Qccc` enthält alle Ketten aus Q^3_{ccc}, `Qfff` enthält alle Ketten aus Q^3_{fff} und `Qfc` alle Ketten aus Q^3_{fc}:

```
val Qccc = allMKs_of [C,C,C];
val Qfff = allMKs_of [F,F,F];
val Qfc = flat(map allMKs_of
                  [[C,F,F],[F,F,C],[F,C,F]]);
```

Als nächstes werden die Graphen der Morphogrammkettenanalysen dieser Strukturen erzeugt:

```
val Gccc = makegraph (map mknums Qccc) (MKanalyse Qccc);
val Gfff = makegraph (map mknums Qfff) (MKanalyse Qfff);
val Gfc  = makegraph (map mknums Qfc)  (MKanalyse Qfc);
```

Die Quellenanalysen dieser Teilgraphen von Q^3 ergeben folgende Resultate.

```
- Minquell Gccc;
> val it =
 [[12,12,12],[12,12,6],[12,12,9],[12,12,1],[12,6,12],[12,6,6],
  [12,6,9],[12,6,1],[12,9,12],[12,9,6],[12,9,9],[12,9,1],
  [12,1,12],[12,1,6],[12,1,9],[12,1,1],[6,6,12],[6,6,6],
  [6,6,9],[6,6,1],[6,9,12],[6,9,6],[6,9,9],[6,9,1],
  [6,1,12],[6,1,6],[6,1,9],[6,1,1],[9,1,12],[9,1,6],
  [9,1,9],[9,1,1]] : int list list
```

[10] [Kae78], S. 99.

Die Struktur der 125 Morphogrammketten[11] von Q^3_{ccc} läßt sich aus diesen 32 Morphogrammketten der minimalen Quellenmenge durch wiederholte Anwendung der Reflektoren erzeugen. Die Ketten sind aus insgesamt nur 4 verschiedenen Morphogrammen komponiert, Q^3_{ccc} kann demnach mit folgendem Erzeugensystem dargestellt werden:

$$Q^3_{\mathrm{ccc}} = \Big(\{mg_1, mg_6, mg_9, mg_{12}\}, \mathtt{Kom}, R^3\Big)$$

```
- Minquell Gfff;
> val it = [[15,15,15],[15,15,11],[15,11,15],[15,11,11]]
          : int list list
```

Die 1000 Morphogrammketten aus Q^3_{fff} lässen sich demnach aus nur 4 Morphogrammketten erzeugen, die aus nur 2 Morphogrammen aufgebaut sind. Das Erzeugendensystem von Q^3_{fff} ist also:

$$Q^3_{\mathrm{fff}} = \Big(\{mg_{11}, mg_{15}\}, \mathtt{Kom}, R^3\Big)$$

```
- val minq = Minquell Gfc;
val it = [[15,11,15],[15,11,14],[15,11,13],[15,11,12],
          [15,11,10],[15,11,9],[15,11,8],[15,11,5],[15,11,7],
          [15,11,2],[15,7,15],[15,7,14],...]
       : int list list

- length it;
> val it = 556 : int

- rd(flat(minq));
> val it = [1,2,4,5,6,7,8,9,10,11,12,13,14,15] : int list
```

Die Menge der 1500 Morphogrammketten in Q^3_{fc} läßt sich aus 556 Morphogrammketten erzeugen, die aus 14 verschiedenen Morphogrammen aufgebaut sind. Das Erzeugendensystem von Q^3_{fc} ist demnach:

$$Q^3_{\mathrm{fc}} = \Big(Q_{/mg_3}, \mathtt{Kom}, R^3\Big)$$

Nach diesen Analysen der unären Reflektoroperationen wird im nächsten Kapitel die Polysemie der morphogrammatischen Komposition untersucht.

[11] Im **Beispiel** auf Seite 127 werden die Anzahlen von Ketten in den Klassen von Q^3 berechnet.

k-l-o-r– Typ	1 r1	2 r2	3 r3	4 r12	5 r13	6 r23	7 r123
ooo	ooo	ooo	ooo	ooo	ooo	ooo	ooo
oor	oor	oor	oor	oor	oor	oor	oor
oro	oro	oro	oro	roo	ooo	rro	roo
orr	orr	orr	orr	ror	oor	rrr	ror
roo	roo	roo	roo	oro	rro	ooo	oro
ror	ror	ror	ror	orr	rrr	oor	orr
rro	rro	rro	rro	rro	rro	rro	rro
rrr	rrr	rrr	rrr	rrr	rrr	rrr	rrr
oll	oll	llo	lol	lol	lol	lol	lol
olk	olk	llr	lok	lok	lok	lok	lok
okl	okl	lko	lrl	kol	lol	krl	kol
okk	okk	lkr	lrk	kok	lok	krk	kok
rll	rll	klo	kol	lrl	krl	lol	lrl
rlk	rlk	klr	kok	lrk	krk	lok	lrk
rkl	rkl	kko	krl	krl	krl	krl	krl
rkk	rkk	kkr	krk	krk	krk	krk	krk
llo	lol	oll	llo	llo	llo	llo	llo
llr	lok	olk	llr	llr	llr	llr	llr
lko	lrl	okl	lko	klo	llo	kko	klo
lkr	lrk	okk	lkr	klr	llr	kkr	klr
klo	kol	rll	klo	lko	kko	llo	lko
klr	kok	rlk	klr	lkr	kkr	llr	lkr
kko	krl	rkl	kko	kko	kko	kko	kko
kkr	krk	rkk	kkr	kkr	kkr	kkr	kkr
lol	llo	lol	oll	oll	oll	oll	oll
lok	llr	lok	olk	olk	olk	olk	olk
lrl	lko	lrl	okl	rll	oll	rkl	rll
lrk	lkr	lrk	okk	rlk	olk	rkk	rlk
kol	klo	kol	rll	okl	rkl	oll	okl
kok	klr	kok	rlk	okk	rkk	olk	okk
krl	kko	krl	rkl	rkl	rkl	rkl	rkl
krk	kkr	krk	rkk	rkk	rkk	rkk	rkk
lll	lll	lll	lll	lll	lll	lll	lll
llk	llk	llk	llk	llk	llk	llk	llk
lkl	lkl	lkl	lkl	kll	lll	kkl	kll
lkk	lkk	lkk	lkk	klk	llk	kkk	klk
kll	kll	kll	kll	lkl	kkl	lll	lkl
klk	klk	klk	klk	lkk	kkk	llk	lkk
kkl	kkl	kkl	kkl	kkl	kkl	kkl	kkl
kkk	kkk	kkk	kkk	kkk	kkk	kkk	kkk

Abbildung 7.5: Die abstraktive k-l-o-r–Analyse von Q^3

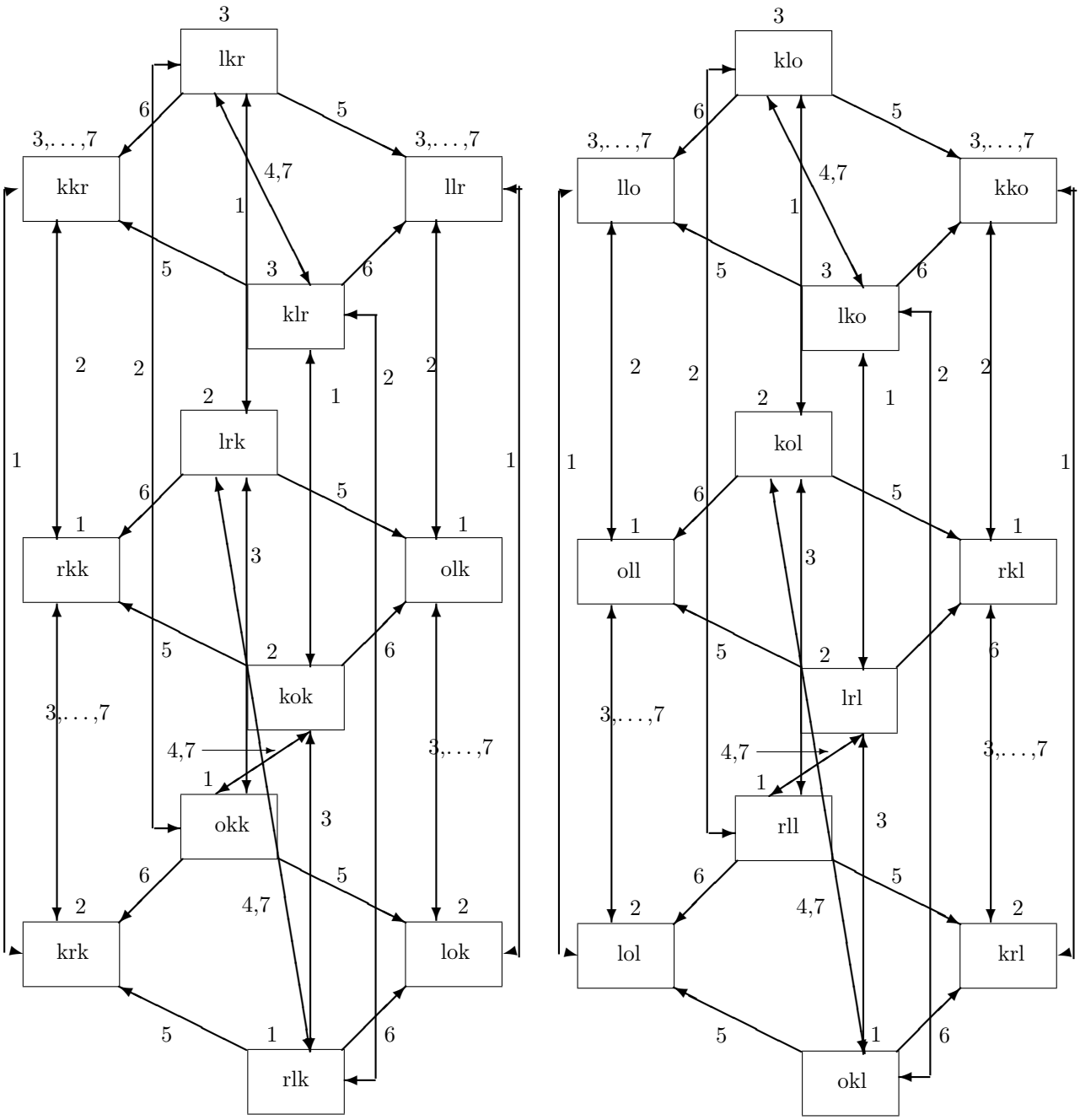


Abbildung 7.6: Die Komposition des k-l-o-r–Graphen aus Q^3_{fc} und Q^3_{gh}

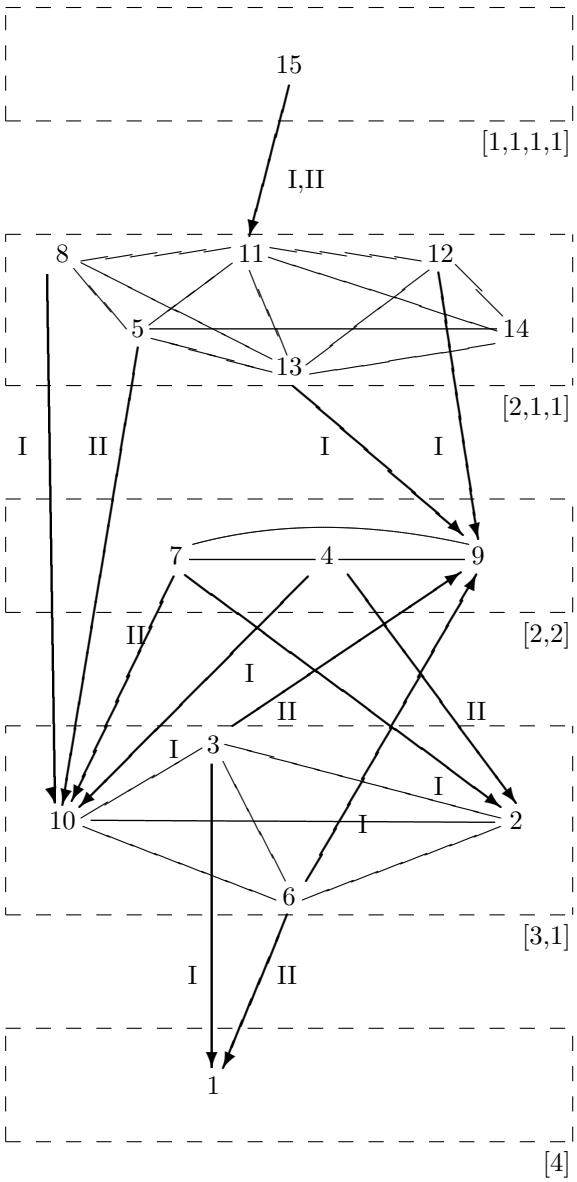


Abbildung 7.7: Die f-c–Analyse von Q^4

Kapitel 8

Kombinatorische Analyse der morphogrammatischen Komposition

8.1 Das kombinatorische Problem der Polysemie

Die in Kapitel 5 definierte Komposition von Morphogrammen zu Morphogrammatrizen ist keine eindeutige Operation, sondern erzeugt eine ganze Klasse von morphogrammatisch äquivalenten aber kenogrammatisch verschiedenen Matrizen. Für diese Mehrdeutigkeit wurde der Begriff der *Polysemie* eingeführt.
Im vierten Kapitel ihrer Arbeit *'On structural analysis of many valued logic'*, die in Zusammenarbeit mit G. Günther und H.v. Foerster erstellt wurde [Na64], stellt H.S.H. Na eine umfassende kombinatorische Analyse der Komposition morphogrammatischer Strukturen vor. Allgemeiner Ausgangspunkt ihrer Überlegungen sind aussagenlogische Systeme $L(n, s)$ mit n Variablen und s Werten, die durch ihre zugrundeliegenden morphogrammatischen Strukturen repräsentiert werden. Jeweils $\binom{m}{s}$ solcher $L(n, s)$ Strukturen können zu einer $L(n, m)$ *Verbundstruktur* komponiert werden. Die klassische zweiwertige Aussagenlogik mit zwei Aussagevariablen stellt ein $L(2, 2)$ System dar, aus dessen Wertfunktionen mittels der morphogrammatischen Komposition $\mathtt{Kom}[\mathtt{M}_1, \ldots, \mathtt{M}_{\binom{m}{2}}]$ höherwertige $L(2, m)$ Verbundstrukturen erzeugt werden.
Um Fragen nach der Komponierbarkeit bestimmter Strukturen und der Anzahl von Polysemen bestimmter Kompositionen zu beantworten, führt Na zunächst eine allgemeine strukturelle Klassifikation morphogrammatischer Strukturen ein, um dann mittels dieser Klassifikation die Komposition der Strukturen zu analysieren.

8.2 Die Klassifikation morphogrammatischer Strukturen

Na analysiert aussagenlogische Systeme $L(n, s)$ mit n Variablen und s Werten, sowie deren zugrundeliegenden morphogrammatischen Strukturen. „Every $L(n, m)$-system, according to the theory of decomposibility, represents a set of morphogrammatic structures. In the case of [the] $L(2, 2)$-system, the structures are the fifteen morphograms

...“ [Na64], p.72. Eine m-wertige Verbundstruktur aus $L(n,m)$ ist aus $\binom{m}{s}$ $L(n,s)$ Strukturen zusammengesetzt. Wie in der f-c-Klassifizierung in Kapitel 6 gezeigt wurde, ist für die Möglichkeit der Komposition morphogrammatischer Strukturen nur die Gestalt der Hauptdiagonalen relevant. Die Hauptdiagonale einer morphogrammatischen Struktur nennt Na *Rahmen* ('frame') alle übrigen Elemente bilden den *Kern* ('core').

Beispiel:

	1	2	3	
1	**1**	2	3	- core
2	2	**2**	3	
3	3	3	**3**	- **frame**

Wie aus dieser n-dimensionalen Darstellung ersichtlich, besteht die Hauptdiagonale (der frame) einer $L(n,m)$-Struktur aus m Stellen. Der core enthält die restlichen $m^n - m$ Elemente.

8.2.1 Rahmen (frames) eines $L(n,m)$ Systems

Na zeigt (Theorem 4.1) daß es in einem System $L(n,m)$ genau

$$NF(m) = \sum_{k=1}^{m} S(m,k) \tag{8.1}$$

verschiedene frames gibt.
Beweis: Die Stirlingzahl der 2. Art $S(m,k)$ bestimmt die Anzahl von Möglichkeiten, m Stellen mit k verschiedenen Objekten zu belegen. Die Kardinalität der Menge m-elementiger frames, die genau k verschiedene Kenogramme benutzen, ist also durch $S(m,k)$ bestimmt. Da frames der Länge m mindestens ein Kenogramm, höchstens jedoch m verschiedene benutzen, ergibt sich folglich als Gesamtanzahl

$$NF(m) = \sum_{k=1}^{m} S(m,k) \quad \blacksquare$$

Beispiel: Für $L(2,2)$, ergibt sich:

$$\begin{aligned} NF(2) &= \sum_{k=1}^{2} S(2,k) \\ &= S(2,1) + S(2,2) \\ &= 1 + 1 \\ &= \mathbf{2} \end{aligned}$$

Da $L(2,2)$ durch die 15 Basismorphogramme repräsentiert wird, auf deren Hauptdiagonale jeweils 2 Kenogramme liegen, gibt es genau $NF(2) = 2$ Strukturmöglichkeiten: ○○ und ○△. Für $L(2,3)$ gibt es $NF(3)$ verschiedene Gestalten der Hauptdiagonale:

$$NF(3) = \sum_{k=1}^{3} S(3,k)$$

$$\begin{aligned} &= S(3,1) + S(3,2) + S(3,3) \\ &= 1 + 3 + 1 \\ &= \mathbf{5}. \end{aligned}$$

Als Kenogrammsequenzen dargestellt: ○○○, ○○△, ○△○, ○△△, ○△□. Die Anzahl $NF(m)$ ist gleich $Tcard(m)$, der Anzahl verschiedener Kenogrammsequenzen der Länge m.

8.2.2 Rahmengruppen (framegroups) eines $L(n,m)$ Systems

Die $NF(m)$ verschiedenen Rahmen des $L(n,m)$ Systems teilt Na nun in Gruppen auf. Zwei Rahmen mit der gleichen Anzahl k verschiedener Kenogramme und der gleichen Partitionierung der m Stellen über diesen Kenogrammen gehören zur gleichen *Rahmengruppe* 'framegroup'. Die Rahmen ○○△, ○△○ und ○△△ aus dem vorhergehenden Beispiel gehören zur gleichen Rahmengruppe, da sie zwei verschiedene Kenogramme benutzen, wovon eins einmal, das jeweils andere zweimal auftritt. Eine Rahmengruppe ist also bestimmt durch eine Partition von m Stellen über k Kenogrammen. Folglich gibt es genau (Theorem 4.2):

$$GF(m) = \sum_{k=1}^{m} P(m,k) \tag{8.2}$$

verschiedene Rahmengruppen für jedes m-wertige System. $P(m,k)$ steht hier für die Anzahl möglicher Partitionen von m Elementen über k Klassen.

Beispiel:

$$\begin{aligned} GF(2) &= \sum_{k=1}^{2} P(2,k) \\ &= P(2,1) + P(2,2) \\ &= 1 + 1 \\ &= \mathbf{2} \end{aligned}$$

Dies ist die Anzahl der möglichen Kenogrammstrukturen: [○○, ○△].

$$\begin{aligned} GF(3) &= \sum_{k=1}^{3} P(3,k) \\ &= P(3,1) + P(3,2) + P(3,3) \\ &= 1 + 1 + 1 \\ &= \mathbf{3} \end{aligned}$$

Die möglichen Kenogrammstrukturen sind: [○○○, ○○△, ○△□].

Anhand der Kenogrammstruktur von $GF(3)$ wird ersichtlich, daß Na's Einteilung der Rahmen in Gruppen der Deutero-Abstraktion von Wertsequenzen entspricht (vgl. Kapitel 3).

8.2.3 Der Grad einer Gruppe

Die Anzahl der Kenogramme, die von den Rahmen einer Rahmengruppe benutzt werden, nennt Na den *Grad*, ('rank'), der Gruppe. Der Grad einer bestimmten Gruppe G_J wird mit k_J bezeichnet.
Eine Rahmengruppe vom Grad k enthält:

$$g\{m_k\} = \frac{m!}{\prod_{i=1}^{k} m_i!} \tag{8.3}$$

Elemente, wobei $m_1, m_2, \cdots, m_k$ die Verteilung der k Kenogramme darstellt und $m_1 + m_2 + \cdots + m_k = m$ (Theorem 4.3).
Beweis: Für m Stellen gibt es $m!$ Permutationen. Wenn m_i Stellen mit dem gleichen Kenogramm belegt sind, erzeugen Permutationen unter diesen m_i Stellen *keine unterschiedlichen* Rahmen. Da die m_i Wiederholungen des gleichen Kenogramms $m_i!$ deutero-äquivalente Rahmen erzeugen, gibt es genau:

$$\frac{m!}{m_1! \times m_2! \times \cdots \times m_k!} = g\{m_k\}$$

verschiedene Rahmen, die zu der Gruppe vom Grad k mit der Verteilung $m_1, m_2, \cdots, m_k$ gehören. ∎

> **Beispiel:** Wieviele verschiedene frames enthält die Rahmengruppe mit genau zwei verschiedenen Kenogrammen (Grad $k = 2$) in einem dreiwertigem System $L(2, 3)$ ($m = 3$)?
>
> $$g\{3_2\} = \frac{3!}{\prod_{i=1}^{2} m_i}! = \frac{6}{m_1! m_2!}$$
>
> in diesem Fall ist die einzig mögliche Verteilung für die m_i: $m_1 = 1, m_2 = 2$ und $m_1 + m_2 = 3$, also ist
>
> $$g\{3_2\} = \frac{6}{1!2!} = \mathbf{3}$$
>
> Dies ist die Anzahl der (Trito-)frames [○○△, ○△○, ○△△] in der (Deutero-)Gruppe ○○△.

8.2.4 Unterteilung der Rahmengruppen in Klassen

Die Elemente dieser Gruppen unterteilt Na nun weiter in verschiedene Klassen entsprechend der Anzahl verschiedener Kenogramme, die von der gesamten morphogrammatischen Struktur (sowohl Frame- als auch Coreelemente) benutzt werden. Die Anzahl der Kenogramme r die von den Strukturen einer solchen *strukturellen Klasse* ('structural class') benutzt werden, nennt sie den *Grad der Klasse* ('rank of the class'). Sie bezeichnet wie folgt: C_{J_r} ist eine strukturelle Klasse der Gruppe G_J mit dem Grad r.
Bezeichne k_J den Grad der Gruppe G_J (k_J verschiedene Kenogramme in den Rahmen der Gruppe J), dann gilt (Theorem 4.4):

$$k_J \leq r \leq r_J \tag{8.4}$$

wobei

$$r_J = (m^n - m) + k_J \tag{8.5}$$

die maximale Anzahl unterschiedlicher Kenogramme ist, die von den Strukturen aus G_J benutzt werden können.

Beweis: Jede Struktur des Systems $L(n, m)$ hat m^n Stellen. Von den m Rahmenstellen werden genau k_J verschiedene Kenogramme benutzt. Die restlichen $m^n - m$ Stellen können höchstens $m^n - m$ verschiedene Kenogramme belegen. Also ist

$$r_J = (m^n - m) + k_J$$

und $r \leq r_J$. Nur wenn alle Stellen des Kerns mit den gleichen Kenogrammen wie der Rahmen belegt werden, kann $r = k_J$ werden. Also gilt

$$k_J \leq r \leq r_J \quad \blacksquare$$

Da Klassen nach der Anzahl verwendeter Kenogramme unterschieden werden und von den Strukturen einer Gruppe G_J höchstens aber r_J verschiedene Kenogramme benutzt werden, gibt es (Theorem 4.5):

$$d_J = r_J - k_J + 1 \tag{8.6}$$

verschiedene Klassen in G_J

> **Beispiel:** Wieviele strukturelle Klassen gibt es in der Gruppe G_2 von $L(2, 2)$, deren Rahmenstellen zwei verschiedene Kenogramme ($\circ\triangle$) benutzen?
>
> $$\begin{aligned} k_2 &= 2 \\ r_2 &= 2^2 - 2 + 2 \\ &= \mathbf{4} \\ d_2 &= 4 - 2 + 1 \\ &= \mathbf{3} \end{aligned}$$
>
> nämlich jeweils eine Klasse mit 2, 3 und 4 verschiedenen Kenogrammen (C_{22}, C_{23}, C_{24} in Tabelle 8.1).

Die Anzahl der verschiedenen Strukturen, die zu einer Klasse C_{Jr} gehören, bezeichnet Na mit p_{Jr}, gibt allerdings keine Formel zur Bestimmung dieser Zahl an. Zum Abschluß des klassifikatorischen Teils gibt Na die vollständige Klassifikation von $L(2, 2)$ an, die sie später für die Analyse der Komposition von $L(2, 2)$ Strukturen zu $L(2, 3)$ und $L(2, 4)$ Verbundstrukturen benützt. Die Ergebnisse dieser Klassifikation sind in Tabelle 8.1 zusammengestellt.

Die morphogrammatischen Strukturen des $L(2, 2)$-Systems entsprechen der Menge der Kenogrammsequenzen der Länge $2^2 = 4$, das heißt der Menge der 15 Basismorphogramme, Q. Wie bereits in Kapitel 6 festgestellt, entspricht G_1 der Teilmenge Q_c und G_2 der Teilmenge Q_f von Q.

Gruppe G_J	G_1					G_2									
k_J (Anzahl der Kenos im frame)	1 ○○					2 ○△									
r_J (max. Anzahl benutzter Kenos)	3 ○△□					4 ○△□★									
$d_J = \|G_J\|$	3					3									
Klasse C_{Jr}	C_{11}	C_{12}			C_{13}	C_{22}				C_{23}					C_{24}
Bezeichnung	α	β			γ	ξ				ρ					ϕ
$P_{Jr} = \|C_{Jr}\|$	1	3			1	4				5					1
morpho-	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○
grammatische	○	○	△	△	△	○	○	△	△	○	□	△	□	□	□
Struktur	○	△	○	△	□	○	△	○	△	□	○	□	△	□	★
	○	○	○	○	○	△	△	△	△	△	△	△	△	△	△
Bezeichnung	I	B	C	E	T_E	D	P	Q	K	T_D	T_Q	T_K	T_P	T	U

Abbildung 8.1: Die Klassifikation von $L(2,2)$ nach [Na64], S. 78.

8.3 Die Komposition von Verbundstrukturen

In der folgenden Diskussion der Komposition morphogrammatischer Strukturen beschränkt sich Na auf die Betrachtung von Fällen, bei denen die einzelnen Strukturen nur an ihren Rahmenstellen (Hauptdiagonalelementen) miteinander verkoppelt werden. So wird von ihre Analyse zwar die Komposition von $L(2,2)$ zu $L(2,3)$ und $L(2,4)$ Strukturen erfasst, nicht jedoch die Komposition von $L(2,3)$ zu $L(2,4)$ Strukturen.

8.3.1 Zulässige Kombinationen von Einheiten

Um eine $L(n,m)$ Struktur zu erzeugen, werden $\binom{m}{s}$ $L(n,s)$ Strukturen benötigt. Allerdings können nicht alle beliebigen Kombinationen solcher $L(n,s)$ Einheiten eine *Verbundstruktur* ('compound structure') bilden (d.h. in der in Kapitel 5 entwickelten Terminologie: nicht alle mglichen Morphogrammketten knnen zu Morphogrammatrizen komponiert werden). Dies wird am Beispiel der Bildung eines $L(2,3)$ Verbundes aus drei $L(2,2)$ Einheiten (Basismorphogrammen) gezeigt. Werden die Morphogramme $I =$ ○○○○, $D =$ ○○○△ und $P =$ ○○△△ benutzt, so ist es möglich einen $L(2,3)$ Verbund zu erzeugen:

Stelle	I	D	P	Verbund
1 1	○		○	○
1 2	○			○
1 3			○	○
2 1	○			○
2 2	○	○		○
2 3		○		○
3 1			△	△
3 2		○		○
3 3		△	△	△

oder abkürzend:

$[I,D,P]$	1	2	3
1	○	○	○
2	○	○	○
3	△	○	△

Wie in der in Kapitel 6 eingeführten f-c–Klassifikation gezeigt wurde, kann in diesem Verbund $[I, D, P]$ das Morphogramm P nicht durch $E =$○△△○ ersetzt werden, da für die Morphogramme $[I : \mathrm{c}]$, $[D : \mathrm{f}]$, $[E : \mathrm{c}]$ nur f-c-Verteilungen möglich sind, für die keine Kompositionen existieren: {ccf, cfc, fcc}. Analog zu den Ergebnissen aus Kapitel 6 stellt Na fest[1]:„*Thus we may conclude: wether a set of $L(n, s)$ units may form an $L(n, m)$ structure or not depends solely on the fact wether their frames are compatible or not, namely, wether their frames can be linked without conflict or not.*“

8.3.2 Familien von Kombinationen

Ob $\binom{m}{s}$ $L(n, s)$ Einheiten zu einer $L(n.m)$ Struktur komponiert werden können, hängt also allein davon ab, ob ihre Rahmen ohne Konflikte zu einem $L(n, m)$ Rahmen zusammengefügt werden können. Es ist allerdings möglich, daß unterschiedliche zulässige Kombinationen von $L(n, s)$ Einheiten den gleichen $L(n, m)$ Rahmen bilden.

> **Beispiel:** die Komposition der $L(2, 2)$ Morphogramme I, D und D ergibt den gleichen $L(2, 3)$ Rahmen wie die Komposition von I, K und K:

$[I, D, D]$	1	2	3
1	○	○	○
2	○	○	○
3	○	○	△

$[I, K, K]$	1	2	3
1	○	○	△
2	○	○	△
3	△	△	△

Rahmen		
○		
	○	
		△

Frau Na wirft an dieser Stelle die Frage auf: „*Thus the next question is: given an $L(n, m)$-frame, how many admissible combinations are there?*“ [Na64], p.83. Die folgenden Überlegungen sind der Beantwortung dieser Frage gewidmet.
Es wird zunächst der Fall der Komposition von $\binom{m}{s}$ $L(n, s)$ Einheiten zu einer $L(n, m)$ Verbundstruktur betrachtet. Die Anzahl verschiedener Rahmengruppen des $L(n, s)$ Systems ist durch Gleichung (8.2) als $GF(s)$ bestimmt. Für eine bestimmte zulässige Kombination $\mathbf{C}$ von $L(n, s)$ Einheiten wird die Anzahl Einheiten aus der Rahmengruppe j von $L(n, s)$ mit λ_j bezeichnet. Da $\binom{m}{s}$ Einheiten zusammengefügt werden, muß gelten:

$$\binom{m}{s} = \sum_{j=1}^{GF(s)} \lambda_j \tag{8.7}$$

> **Beispiel:** für die Komposition einer $L(2, 3)$ Struktur aus den $L(2, 2)$ Strukturen I, K, K gilt:

$$I \in G_1, \qquad \lambda_1 = 1$$

$$K \in G_2, \qquad \lambda_2 = 2$$

$$GF(2) = 2$$

$$\sum_{j=1}^{GF(2)} \lambda_j = 1 + 2 = \binom{3}{2} = \mathbf{3}$$

[1] [Na64], S.82.

Der Grad der $L(n,s)$ Gruppe j, k_j, gibt die Anzahl der von den Einheiten der Gruppe j benutzten Kenogramme an. Jede dieser $L(n,s)$ Einheiten besitzt s Rahmenstellen. Die s Rahmenstellen sind bei den Einheiten der Gruppe G_j über k_j Kenogramme verteilt, was wie folgt notiert wird:

$$s_1, s_2, \ldots, s_{k_j}$$

wobei gilt:

$$s_1 + s_2 + \cdots + s_{k_j} = s$$

Beispiel: in der Gruppe G_2 in $L(2,2)$ ist $k_2 = 2$ und $s = 2$, die erste Rahmenstelle benutzt das Kenogramm ○ und die zweite Rahmenstelle das Kenogramm △, also $s_1 = 1$ und $s_2 = 1$, deshalb:

$$s_1 + s_2 = 1 + 1 = 2 = s$$

Für die Kombination **C** sind die λ_j Einheiten aus der Grupe G_j über die d_j Klassen von G_j verteilt. Da sich die d_j Klassen nicht in der Anzahl und der Verteilung der benutzten Kenogramme der Rahmenstellen unterscheiden, sondern lediglich in der Gesamtanzahl, r, benutzter Kenograme der Strukturen differieren, ist jede Verteilung von λ_j über d_j für eine Komposition zulässig. Bezeichne $D(\lambda_j, d_j)$ die Anzahl aller möglichen Verteilungen von λ_j über d_j, dann gilt:

$$D(\lambda_j, d_j) = \binom{\lambda_j + d_j - 1}{\lambda_j} \tag{8.8}$$

Beispiel: die Kombination I, K, K aus dem vorherigen Beispiel benutzt eine Einheit aus der Gruppe G_1 und zwei Einheiten aus G_2. Für eine solche Kombination gilt allgemein:

$$\begin{aligned} \lambda_1 &= 1 \\ \lambda_2 &= 2 \\ d_1 &= 3 \\ d_2 &= 3 \end{aligned}$$

also:

$$\begin{aligned} D(1,3) &= \binom{1+3-1}{1} = \binom{3}{1} = \mathbf{3} \\ D(2,3) &= \binom{2+3-1}{2} = \binom{4}{2} = \mathbf{6} \end{aligned}$$

Da es $N = GF(s)$ $L(n,s)$ Rahmengruppen gibt, sind über den $d_1 + d_2 + \cdots + d_N$ Klassen von $L(n,s)$

$$D(\lambda_1, d_1) \times D(\lambda_2, d_2) \times \cdots \times D(\lambda_N, d_n)$$

verschiedene Kombinationen von den $\binom{m}{s}$ $L(n,s)$ Einheiten möglich. Jede dieser Verteilungen der $(\lambda_1 + \lambda_2 + \cdots + \lambda_N) = \binom{m}{s}$ Einheiten über den $(d_1 + d_2 + \cdots + d_N)$ Klassen von $L(n,s)$ bildet eine eigene *Familie* von $L(n,m)$ Strukturen, sodaß es also:

$$\varphi(m,s) = D(\lambda_1, d_1) \times D(\lambda_2, d_2) \times \cdots \times D(\lambda_N, d_N) \tag{8.9}$$

solcher Familien gibt.
Um diese Familien zu bezeichnen, soll für eine bestimmte zulässige Kombination n_{jr_i} die Anzahl benutzter Einheiten der Klasse C_{jr} aus der Gruppe G_j angeben. Wobei $1 \leq i \leq d_j$ und r_i der Grad der i-ten Klasse aus G_j ist. Die λ_j Einheiten aus G_j verteilen sich dann wie folgt über die d_j Klassen von G_j:

$$\lambda_j = n_{jr_1} + n_{jr_2} + \cdots + n_{jr_{d_j}}$$

Beispiel: Betrachtet wird die Kombination I, B, T_E. Da $I \in C_{11}$, $B \in C_{12}$ und $T_E \in C_{13}$, ist hier $\lambda_1 = 3$ und $\lambda_2 = 0$. Es gilt:

$$r_1 = 1, r_2 = 2, r_3 = 3 \qquad \text{für } G_1,$$

$$r_1 = 2, r_2 = 3, r_3 = 4 \qquad \text{für } G_2$$

und

$$n_{11} = 1, n_{12} = 1, n_{13} = 1, n_{22} = 0, n_{23} = 0, n_{24} = 0$$

also:

$$\lambda_1 = 1 + 1 + 1 = 3$$
$$\lambda_2 = 0 + 0 + 0 = 0$$

Da die Klasse C_{jr} p_{jr} Einheiten enthält und verschiedene $L(n, s)$ Einheiten zu unterschiedlichen $L(n, m)$ Verbundstrukturen komponiert werden können, enthält jede dieser Familien σ unterschiedliche $L(n, m)$ Kombinationen[2] (Theorem 4.8):

$$\sigma = \prod_{j=1}^{GF(s)} \left\{ \frac{\lambda_j!}{\prod_{i=1}^{d_j} n_{jr_i}!} \times \prod_{i=1}^{d_j} (p_{jr_i})^{n_{jr_i}} \right\} \tag{8.10}$$

Beweis: Die λ_j Einheiten sind über die d_j Klassen mit den Anzahlen $n_{jr_1}, n_{jr_2}, \ldots, n_{jr_{d_j}}$ verteilt. Da die Reihenfolge in einer Kombination von $L(n, s)$ Einheiten nicht berücksichtigt werden muß, erzeugen Permutationen innerhalb dieser n_{jr_i} Einheiten aus C_{jr_i} keine verschiedenen Kombinationen. Also gibt es nur:

$$\frac{\lambda_j!}{\prod_{i=1}^{d_j} n_{jr_i}!}$$

mögliche Permutationen, um unterscheidbare Kombinationen zu erzeugen. Für jede so gegebene Permutation existieren für jede der n_{jr_i} Einheiten genau p_{jr_i} mögliche Auswahlen aus der Klasse C_{jr_i}. Es gibt also:

$$(p_{jr_i})^{n_{jr_i}}$$

[2]Da Na von einem feststehenden $L(n, m)$ Rahmen ausgeht, muß σ noch mit der Anzahl von $g\{m_{k_J}\}$ (Trito-)Rahmen der Rahmengruppe G_J multipliziert werden, um die tatsächliche Anzahl von möglichen Verbundstrukturen zu erhalten.

mögliche Auswahlen für alle d_j Klassen. Folglich sind:

$$\frac{\lambda_j!}{\prod_{i=1}^{d_j} n_{jr_i}!} \times \prod_{i=1}^{d_j} (p_{jr_i})^{n_{jr_i}}$$

unterschiedliche Kombinationen von den λ_j Einheiten aus G_j erzeugbar, falls alle anderen Einheiten aus den restlichen $GF(s)-1$ Gruppen unverändert bleiben. Werden auch für diese Einheiten Veränderungen zugelassen, ergibt sich:

$$\begin{aligned}\sigma &= \left\{ \frac{\lambda_1!}{\prod_{i=1}^{d_1} n_{1i}!} \times \prod_{i=1}^{d_1} (p_{1i})^{n_{1i}} \right\} \times \cdots \times \left\{ \frac{\lambda_N!}{\prod_{i=1}^{d_N} n_{Ni}!} \times \prod_{i=1}^{d_N} (p_{Ni})^{n_{Ni}} \right\} \\ &= \prod_{j=1}^{N} \left\{ \frac{\lambda_j!}{\prod_{i=1}^{d_j} n_{jr_i}!} \times \prod_{i=1}^{d_j} (p_{jr_i})^{n_{jr_i}} \right\}\end{aligned}$$

wobei $N = GF(s)$. ∎

8.3.3 Exemplarische Berechnung

Die bis hierhin entwickelten Ergebnisse reichen aus, um die oben zitierte Fragestellung nach der Anzahl von Kombinationen von $L(n, s)$ Einheiten, die zu einem gegebenen $L(n, m)$ Rahmen komponiert werden können, zu beantworten. Dies soll exemplarisch vorgeführt werden.
Betrachtet wird das Problem, einen $L(2, 3)$ Verbund mit dem Rahmen ○○○ aus $L(2, 2)$ Morphogrammen zu generieren. Da der Rahmen für diesen speziellen Fall nur aus identischen Kenogrammen besteht($k = 1$), läßt er sich nicht aus Einheiten aus G_2 zusammensetzen ($k_2 = 2$). Also gilt:

$$d_1 = 3, d_2 = 3$$

$$\lambda_1 = 3, \lambda_2 = 0$$

da $\lambda_2 = 0$, folgt:

$$n_{2r} = 0 \text{ für alle } r \in \{k_j, \ldots, r_j\}$$

Nach Gleichung (8.8) ergibt sich:

$$D(\lambda_1, d_1) = \binom{3+3-1}{3} = \mathbf{10}$$

und:

$$D(\lambda_2, d_2) = \binom{0+3+1}{0} = \mathbf{1}$$

zusammen mit Gleichung (8.9) ergibt sich hieraus:

$$\varphi(3, 2) = 10 \times 1 = \mathbf{10}. \tag{8.11}$$

Das bedeutet, daß die $L(2,2)$ Morphogramme zehn Familien von $L(2,3)$ Strukturen bilden können, deren Rahmengestalt ○○○ ist. Diese Familien und die entsprechenden Verteilungen der n_{jr_i} sind in der Tabelle 8.2 angegeben. Die Anzahl σ der $L(2,3)$ Verbundstrukturen für jede Familie, berechnet nach (8.10), ist in der letzten Spalte aufgeführt.

n_{11}	n_{12}	n_{13}	Familie	σ
3	0	0	α^3	1
2	1	0	$\alpha^2\beta$	9
2	0	1	$\alpha^2\gamma$	3
1	2	0	$\alpha\beta^2$	27
1	1	1	$\alpha\beta\gamma$	18
1	0	2	$\alpha\gamma^2$	3
0	3	0	β^3	27
0	2	1	$\beta^2\gamma$	27
0	1	2	$\beta\gamma^2$	9
0	0	3	γ^3	1

Abbildung 8.2: Die Familien der ○○○ $L(2,3)$ Verbundstrukturen.

In der nächsten Abbildung 8.3werden die neun Verbundstrukturen der Familie $\alpha^2\beta$ und die drei der Familie $\alpha^2\gamma$ beispielhaft angeführt.

Familie	$L(2,3)$ Strukturen
$\alpha^2\beta$	$[I,I,B],[I,B,I],[B,I,I]$, $[I,I,C],[I,C,I],[C,I,I]$, $[I,I,E],[I,E,I],[E,I,I]$
$\alpha^2\gamma$	$[I,I,T_E],[I,T_E,I],[T_E,I,I]$

Abbildung 8.3: Familien und Verbundstrukturen

8.4 Komponierte Strukturen und Polysemie

8.4.1 Verbundstrukturen und Polyseme (M-functions)

In Kapitel 3 wurde gezeigt, daß die Verkettung von Kenogrammsequenzen und folglich auch die Komposition von Morphogrammketten zu Matrizen eine nicht eindeutige Operation ist. Die Verbundstruktur (Morphogrammkette) $[E,E,E]$ etwa repräsentiert fünf kenogrammatisch verschiedene $L(2,3)$ Strukturen (Morphogrammatrizen):

$[EEE]_1$	1	2	3
1	○	△	△
2	△	○	△
3	△	△	○

$[EEE]_2$	1	2	3
1	○	△	□
2	△	○	△
3	□	△	○

$[EEE]_3$	1	2	3
1	○	△	△
2	△	○	□
3	△	□	○

$[EEE]_4$	1	2	3
1	○	△	□
2	△	○	□
3	□	□	○

$[EEE]_5$	1	2	3
1	○	△	★
2	△	○	□
3	★	□	○

Diese fünf Strukturen sind jeweils aus drei Morphogrammen vom Typ E aufgebaut, jedoch kenogrammatisch verschieden: $[EEE]_1$ benutzt zwei verschiedene Kenogramme, $[EEE]_2$, $[EEE]_3$ und $[EEE]_4$ jeweils drei, allerdings in unterschiedlicher Verteilung, $[EEE]_5$ benutzt vier Kenogramme. Diese kenogrammatisch verschiedenen, aber komponentenweise (morphogrammatisch) äquivalenten Strukturen nennt Na 'M-functions', für die in dieser Arbeit der Begriff *Polyseme* eingeführt wurde. Im Weiteren werden die beiden Begriffe synonym verwenden. (Vgl. die Definition der morphogrammatischen Äquivalenz(klasse) und der morphogrammatischen Polysemie in 5.4.2).
Weisen verschiedene M-functions die gleiche Komponententsruktur auf, spricht Na davon, daß sie die gleiche 'synthetic structure' besitzen und zur gleichen 'category' von M-functions gehören. Die aus $L(n,s)$ Einheiten komponierten $L(n,m)$ Verbundstrukturen die in der bisherigen Diskussion betrachtet wurden, sind synthetische Strukturen und repräsentieren eine ganze *Kategorie* (morphogrammatische Äquivalenzklasse) von M-functions. „Compound structures, discussed in the previous sections are thus synthetic structures and not M-function“ [Na64], p.92. Dieser Unterschied zwischen den synthetischen Strukturen und den M-functions drückt sich in der Notation wie folgt aus:

$$[M_1, M_2, \ldots, M_{\binom{m}{s}}]$$

bezeichnet eine $L(n,m)$ Verbundstruktur (synthetische Struktur), die sich aus $\binom{m}{s}$ $L(n,s)$ morphogrammatischen Strukturen (M_i) zusammensetzt. Auf Grund der Polysemie repräsentiert eine solche Verbundstruktur eine ganze Kategorie von MP M-functions:

$$[M_1 M_2 \ldots M_{\binom{m}{s}}]_1, \ldots, [M_1 M_2 \ldots M_{\binom{m}{s}}]_{MP}$$

Mithilfe der folgenden Überlegungen soll die Anzahl von M-functions (MP) einer gegebenen Kategorie bestimmt werden.

8.4.2 Der kenogrammatische Akkretionsgrad von M-functions

Um sich dieser Fragestellung zu nähern führt Na zuerst den Begriff des *Akretionsgrades* einer M-function ein. Die fünf M-functions der Kategorie $[E,E,E]$ aus dem obigen Beispiel benutzen entweder zwei, drei oder vier verschiedene Kenogramme. Die Anzahl benutzter Kenogramme einer gegebenen M-function nennt Na 'rank of the M-function', was hier in Übereinstimmung mit Abschnitt 3.3.5als Akkretionsgrad übersetzt ist. Diese Anzahl benutzter Kenogramme wird mit $\overset{*}{m}$ bezeichnet. Da eine M-function eines $L(n,m)$ System aus mindestens einem, höchstens jedoch m^n unterschiedlichen Kenogrammen aufgebaut ist, gilt für $L(n,m)$:

$$1 \leq \overset{*}{m} \leq m^n$$

Beispiel: Für $L(2,3)$ kann $\overset{*}{m}$ Werte zwischen 1 und 3^2 annehmen.

$[III]_1$	1	2	3
1	○	○	○
2	○	○	○
3	○	○	○

$\overset{*}{m}= 1$

$[UUU]_{MP}$	1	2	3
1	○	△	∗
2	□	⋆	●
3	◇	▽	■

$\overset{*}{m}= 9$

Allerdings gilt nicht für jede Kategorie, daß $\overset{*}{m}$ alle Werte zwischen 1 und m^n annehmen kann. Für $[E, E, E]$ ist beispielsweise $\min\{\overset{*}{m}\} = 2$ und $\max\{\overset{*}{m}\} = 4$. $\min\{\overset{*}{m}\}$ wird mit $\mathbf{R}$ und $\max\{\overset{*}{m}\}$ mit $\mathbf{M}$ bezeichnet. Also gilt:

$$1 \leq \mathbf{R} \leq \overset{*}{m} \leq \mathbf{M} \leq m^n.$$

Na stellt fest, daß für jede Kategorie gilt:

$$\mathbf{R} = \max\{k_J, r_{\max}\} \tag{8.12}$$

Wobei $r_{\max}$ das Maximum aller r_i für alle $n_{jr_i} \neq 0$ ist. k_J gibt die Anzahl der Kenogramme in der Rahmengruppe G_J von $L(n, m)$ an. n_{jr_i} bestimmt die Anzahl von Einheiten M_l aus der Klasse C_{jr_i} von $L(n, s)$, die die Kategorie benutzt.

Beispiel: Die Kategorie $[E, E, E]$ repräsentiert die folgende Verteilung:

$$\lambda_1 = 3 \qquad \lambda_2 = 0$$

$$n_{11} = 0 \qquad n_{22} = 0$$

$$n_{12} = 3 \qquad n_{23} = 0$$

$$n_{13} = 0 \qquad n_{24} = 0$$

$[E, E, E]$ gehört zu der $L(2, 3)$ Rahmengruppe ○○○, daher ist $k_J = 1$. Alle Einheiten der Verbundstruktur kommen aus C_{12} und enthalten jeweils zwei verschiedene Kenogramme, also ist:

$$r_{\max} = 2$$

und

$$\mathbf{R} = \max\{k_J, r_{\max}\} = 2$$

Der maximalen Akkretionsgrad $\mathbf{M}$ ist bestimmt durch:

$$\mathbf{M} = k_J + \sum_{j=1}^{GF(s)} \left\{ \sum_{i=1}^{d_j} (r_i - k_j) \times n_{jr_i} \right\} \tag{8.13}$$

Beweis: Um für eine gegebene Verbundstruktur M-functions eines höheren Akkretionsgrades als $\mathbf{R}$ zu erzeugen, ist allein die Anzahl von Kenogrammen in den Corestellen der $L(n, s)$ Einheiten zu berücksichtigen, da die Anzahl k_J benutzter Kenogrammme des $L(n, m)$ Rahmens durch die Anzahlen k_j der $L(n, s)$ Rahmen genau bestimmt und nicht variabel ist. Darum kann jede der njr_i Einheiten aus der Klasse Cjr_i der $L(n, s)$ Gruppe G_j höchstens:

$$r_i - k_j$$

weitere Kenogramme für die Corestellen einführen, da der Rahmen bereits k_j verschiedene Kenogramme enthält. Die n_{jr_i} Einheiten aus Cjr_i können zusammen also höchstens:

$$(r_i - k_j) \times n_{jr_i}$$

weitere Kenogramme zur Verfügung stellen. Da für jede der $GF(s)$ Gruppen d_j solcher Klassen existieren, können die $\binom{m}{s}$ $L(n,s)$ Einheiten einer $L(n,m)$ Verbundstruktur M-functions mit höchstens:

$$\mathbf{M} = k_J + \sum_{j=1}^{GF(s)} \left\{ \sum_{i=1}^{d_j} (r_i - k_j) \times n_{jr_i} \right\}$$

verschiedenen Kenogrammen erzeugen. ∎

Beispiel: Für die Verbundstruktur $[E,E,E]$ gilt, wie oben festgestellt:

$$k_J = 1$$

für $L(2,2)$ gilt allgemein:

$$k_1 = 1 \qquad k_2 = 2$$

dann ergibt sich:

$$\begin{aligned}
\mathbf{M} &= k_J + \sum_{j=1}^{GF(s)} \left\{ \sum_{i=1}^{d_j} (r_i - k_j) \times n_{jr_i} \right\} \\
&= 1 + \left[\sum_{i=1}^{3} (r_i - 1) n_{1r_i} \right] + \left[\sum_{i=1}^{3} (r_i - 2) n_{2r_i} \right] \\
&= 1 + [(1-1)0 + (2-1)3 + (3-1)0] + [0+0+0] \\
&= 1 + 3 + 0 \\
&= \mathbf{4}
\end{aligned}$$

Die Berechnungen $\mathbf{R} = 2$ und $\mathbf{M} = 4$ decken sich mit der am Beispiel der M-functions von $[E,E,E]$ gewonnenen Beobachtung, daß diese M-functions mindestens zwei ($[EEE]_1$), höchstens jedoch vier verschiedene ($[EEE]_5$) Kenogramme benutzen.

8.4.3 Die Anzahl der M-functions einer Kategorie

Nach diesen Vorüberlegungen zur Bestimmung des Akkretionsgrades wendet sich Na nun direkt der oben erwähnten Frage nach der Anzahl von M-functions oder Polysemen einer gegebenen Kategorie zu. Diese gesuchte Anzahl nennt sie MP. MP hängt nur vom Grad der komponierten $L(n,s)$ Einheiten und deren möglichen Permutationen ab. Deswegen nimmt MP für alle Verbundstrukturen, deren Einheiten aus den gleichen Klassen stammen, d.h. für alle Kategorien einer Familie, den gleichen Wert an.

8.4.3.1 MP der Kategorien der Familie A_0

Zunächst werden nur Fälle betrachtet, bei denen alle komponierten Einheiten aus nur einer $L(n,s)$ Gruppe G_j stammen. Die zehn Familien aus Tabelle 8.2 sind Beispiele für solche Kompositionen. Für einen solchen Fall gilt:

$$\lambda_j = \binom{m}{s}$$

und

$$\sum_{i=1}^{d_j} n_{jr_i} = \lambda_j.$$

Sei nun A_0 eine Familie von Verbundstrukturen, die alle durch die gleiche Verteilung der n_{jr_i} charakterisiert ist:

$$n_{jr_i} = \begin{cases} \lambda_j & \text{für} \quad r_i = k_j \\ 0 & \text{für} \quad r_i \neq k_j. \end{cases}$$

Wobei k_j der Grad der Gruppe ist. Aus den Gleichungen (8.12) und (8.13) ergibt sich:

$$\mathbf{R}_0 = k_j = \mathbf{M}_0.$$

Also können die Verbundstrukturen der Familie A_0 nur M-functions erzeugen, für die:

$$\overset{*}{m} = k_j.$$

Da es für diesen Fall für jede solche Kategorie nur eine M-function gibt, ist:

$$MP_0 = 1. \tag{8.14}$$

Beispiel: Für die Komposition von drei $L(2,2)$ Strukturen zu einem $L(2,3)$ Verbund sind die beiden einzigen Familien, die der Verteilung von A_0 genügen: α^3 und ξ^3. Da α nur eine Struktur I enthält, ist $[I,I,I]$ der einzig mögliche α^3 Verbund. Da $MP_0 = 1$, gibt es für diesen Verbund nur eine einzige M-function $[III]_1$:

$[III]_1$	1	2	3
1	○	○	○
2	○	○	○
3	○	○	○

.

Als Beispiel für einen ξ^3 Verbund sei hier $[P,K,Q]$ angeführt. Da auch für diesen Verbund die Verteilung A_0 erfüllt ist (also $MP_0 = 1$), gibt es ebenfalls nur eine M-function $[PKQ]_1$:

$[PKQ]_1$	1	2	3
1	○	○	□
2	△	△	△
3	○	△	□

.

8.4.3.2 MP der Familie A_1

Betrachtet werden als nächstes alle Kategorien der Familie A_1, die durch die folgende Verteilung der njr betimmt ist:

$$n_{jr_i} = \begin{cases} \lambda_j - 1 & \text{für} \quad r_i = k_j \\ 1 & \text{für} \quad r_i = k_j + \Delta r \\ 0 & \text{für} \quad r_i \neq k_j \ \text{ oder } \ r_i \neq k_j + \Delta r, \end{cases}$$

mit

$$1 \leq \Delta r \leq d_j - 1.$$

Bezeichne $\mathrm{a}(\lambda_j, 0; \overset{*}{m})$ die Anzahl von M-functions vom Grad $\overset{*}{m}$, die jede Kategorie der Familie A_0 besitzt. Dann ist $\mathrm{a}(\lambda_j - 1, 1; \overset{*}{m})$ die Anzahl von M-functions vom Grad $\overset{*}{m}$ aller Kategorien der Familie A_1. Die induktive Beziehung zwischen $\mathrm{a}(\lambda_j, 0; \overset{*}{m})$ und $\mathrm{a}(\lambda_j - 1, 1; \overset{*}{m})$ ist dann bestimmt durch (Theorem 4.10):

$$\begin{aligned} \mathrm{a}(\lambda_j - 1, 1; \overset{*}{m}) \quad = \quad & \left[\binom{\overset{*}{m} - k_j}{\Delta r} \Delta r! \right] \times \mathrm{a}(\lambda_j, 0; \overset{*}{m}) \\ & + \left[\sum_{q=1}^{\Delta r} \binom{\overset{*}{m} - k_j - q}{\Delta r - q} \times \frac{\Delta r!}{q!} \times \mathrm{a}(\lambda_j, 0; \overset{*}{m} - q) \right] \end{aligned} \tag{8.15}$$

Beweis: Der erste Summand gibt die Anzahl von möglichen M-functions hergeleitet aus M-functions mit der Verteilung $\{\lambda_j, 0\}$ und dem Grad $\overset{*}{m}$ an. Der zweite Summand bezeichnet die Anzahl von M-functions hergeleitet aus M-functions mit der gleichen Verteilung $\{\lambda_j, 0\}$ aber von niedrigerem Grad $(\overset{*}{m} - q)$.
Eine $\{\lambda_j - 1, 1\}$ M-function vom Grad $\overset{*}{m}$ kann aus einer $\{\lambda_j, 0\}$ M-function gleichen Grades erzeugt werden, indem eine der $L(n, s)$ Einheiten vom Grad k_j durch eine Einheit vom Grad r erzetzt wird, wobei:

$$r = k_j + \Delta r \qquad (1 \leq \Delta r \leq d_j - 1).$$

Es werden also Δr weitere Kenogrammsymbole gebraucht. Da die Gesamtanzahl der Kenogramme, $\overset{*}{m}$, nicht verändert werden soll, gibt es:

$$\overset{*}{m} - k_j$$

mögliche Kenogramme. Da die Einheit vom Grad r Δr zusätzliche Kenogramme aus der Menge der $\overset{*}{m} - k_j$ möglichen Kenogramme benutzt, gibt es:

$$\binom{\overset{*}{m} - k_j}{\Delta r}$$

mögliche Auswahlen. Für jede bestimmte Auswahl können die Δr Kenogramme die Einheit vom Grad r auf:

$$\Delta r!$$

verschiedene Arten belegen.

Aus jeder $\{\lambda_j, 0\}$ M-function vom Grad $\overset{*}{m}$ lassen sich demzufolge:

$$\left[\binom{\overset{*}{m}}{\Delta r} -k_j \Delta r!\right]$$

$\{\lambda_j - 1, 1\}$ M-functions generieren. Es gibt $\mathrm{a}(\lambda_j, 0; \overset{*}{m})$ M-functions vom Grad $\overset{*}{m}$ mit der Verteilung $\{\lambda_j, 0\}$ und deswegen:

$$\left[\binom{\overset{*}{m}}{\Delta r} -k_j \Delta r!\right] \times \mathrm{a}(\lambda_j, 0; \overset{*}{m})$$

verschiedene $\{\lambda_j - 1, 1\}$ M-functions, die auf diese Weise generiert werden können. Eine $\{\lambda_j - 1, 1\}$ M-function vom Grad $\overset{*}{m}$ kann aber auch aus $\{\lambda_j, 0\}$ M-functions von einem niedrigerem Grad $\overset{*}{m}_1$ abgeleitet werden. Für die zusätzlichen Δr Kenogramme können dann andere als die bereits vorhandenen $\overset{*}{m}_1$ Kenogramme ausgewählt werden. Sei q diese Anzahl neuer Kenogramme, mit $q \leq \Delta r$, dann brauchen nur:

$$\Delta r - q$$

Kenogramme aus der Menge der $(\overset{*}{m} -k_j)$ Kenogramme, die von den $\lambda_j - 1$ restlichen Einheiten benutzt werden, ausgewählt zu werden. Also gibt es:

$$\binom{\overset{*}{m}_1}{\Delta r - q} -k_j$$

mögliche Auswahlen. Da die q neuen Kenogramme nicht in den anderen Einheiten vorkommen, produzieren Permutationen unter ihnen keine kenogrammatisch verschiedenen M-functions. Deswegen erzeugt die Menge von Δr Kenogrammen nur:

$$\frac{\Delta r!}{q!}$$

kenogrammatisch unterschiedliche Belegungen. Jede der $\mathrm{a}(\lambda_j, 0; \overset{*}{m}_1)$ M-functions generiert also:

$$\binom{\overset{*}{m}_1}{\Delta r - q} -k_j \times \frac{\Delta r!}{q!}$$

$\{\lambda_j - 1, 1\}$ M-functions. Demzurfolge können auf diese Weise insgesamt:

$$\binom{\overset{*}{m}_1}{\Delta r - q} -k_j \times \frac{\Delta r!}{q!} \times \mathrm{a}(\lambda_j, 0; \overset{*}{m}_1)$$

M-functions mit einer $\{\lambda_j - 1, 1\}$ Verteilung erzeugt werden. Da die erzeugten M-functions vom Grad $\overset{*}{m}$ sind, ist:

$$\overset{*}{m}_1 + q = \overset{*}{m}$$

und

$$\overset{*}{m}_1 = \overset{*}{m} - q \qquad (1 \leq q \leq \Delta r).$$

Hiermit ergibt sich:

$$\begin{aligned} \mathrm{a}(\lambda_j - 1, 1; \overset{*}{m}) &= \left[\binom{\overset{*}{m} - k_j}{\Delta r} \Delta r!\right] \times \mathrm{a}(\lambda_j, 0; \overset{*}{m}) \\ &\quad + \left[\sum_{q=1}^{\Delta r} \binom{\overset{*}{m} - k_j - q}{\Delta r - q} \times \frac{\Delta r!}{q!} \times \mathrm{a}(\lambda_j, 0; \overset{*}{m} - q)\right] \end{aligned}$$

■

Die Gesamtanzahl von M-functions, die eine Kategorie vom Typ A_1 enthalten kann, ist dann:

$$MP_1 = \sum_{\overset{*}{m}=\mathbf{R}_1}^{\mathbf{M}_1} \mathrm{a}(\lambda_j - 1, 1; \overset{*}{m}),$$

wobei $\mathbf{R}_1$ und $\mathbf{M}_1$ die untere und obere Grenze für $\overset{*}{m}$ angeben.

8.4.3.3 MP der Familie A''

Allgemein können, bei gegebenem Δr und $\mathrm{a}(\ldots)$ für eine Familie A, die Werte von $\mathrm{a}(\ldots)$ für eine abgeleitete Familie A' bestimmt werden, wenn A die folgende Verteilung aufweist:

$$n_{jr} = \begin{cases} \lambda_j - \nu & \text{für} \quad r = k_j \\ \nu & \text{für} \quad r = k_j + \Delta r \\ 0 & \text{für alle anderen Werte } r. \end{cases}$$

Die abgeleitete Familie A' weist dann die folgende Verteilung von Einheiten auf:

$$n_{jr} = \begin{cases} \lambda_j - \nu - 1 & \text{für} \quad r = k_j \\ \nu + 1 & \text{für} \quad r = k_j + \Delta r \\ 0 & \text{für alle anderen Werte } r. \end{cases}$$

Die induktive Beziehung zwischen den beiden Funktionen lautet:

$$\begin{aligned} &\mathrm{a}(\overbrace{\lambda_j - \nu - 1, 0, \ldots, 0, \nu + 1, 0, \ldots, 0}^{d_j \text{ Klassen in } G_j}; \overset{*}{m}) = \\ &\quad \left[\binom{\overset{*}{m} - k_j}{\Delta r} \Delta r!\right] \times \mathrm{a}(\lambda_j - \nu, 0, \ldots, 0, \nu, 0, \ldots, 0; \overset{*}{m}) \\ &\quad + \left[\sum_{q=1}^{\Delta r} \binom{\overset{*}{m} - k_j - q}{\Delta r - q} \times \frac{\Delta r!}{q!} \times \mathrm{a}(\lambda_j - \nu, 0, \ldots, 0, \nu, 0, \ldots, 0; \overset{*}{m} - q)\right] \end{aligned} \tag{8.16}$$

Sei A'' eine Familie mit der folgenden Verteilung der λ_j Einheiten über den d_j Klassen:

$$n_{jr_1}, n_{jr_2}, \ldots, n_{jr_{d_j}},$$

wobei:

$$\begin{aligned} r_1 &= k_j + (\Delta r)_1 = k_j \qquad \text{mit } (\Delta r)_1 = 0 \\ r_2 &= k_j + (\Delta r)_2 \\ &\vdots \\ r_{d_j} &= k_j + (\Delta r)_{d_j} \end{aligned}$$

und:

$$n_{jr_1} + n_{jr_2} + \ldots + n_{jr_{d_j}} = \lambda_j.$$

Dann ergeben sich die entsprechenden Funktionen a(...) für A'' wie folgt: Um $\mathrm{a}(\lambda_j - n_{jr_2}, n_{jr_2}, 0, \ldots, 0; \overset{*}{m})$ aus $\mathrm{a}(\lambda_j, 0, \ldots, 0; \overset{*}{m})$ zu erhalten, muß Gleichung (8.16) n_{jr_2}-mal angewendet werden, wobei dann $\Delta r = (\Delta r)_2$ gesetzt ist. Um nun:

$$\mathrm{a}(\lambda_j - n_{jr_2} - n_{jr_3}, n_{jr_2}, n_{jr_3}, 0, \ldots, 0; \overset{*}{m})$$

aus $\mathrm{a}(\lambda_j - n_{jr_2}, n_{jr_2}, 0, \ldots, 0; \overset{*}{m})$ zu erzeugen, muß Gleichung (8.16) n_{jr_3}-mal angewandt werden. Dieser Vorgang wird jetzt entsprechend für $(\Delta r)_4, \ldots, (\Delta r)_{d_j}$ wiederholt, um schließlich:

$$\mathrm{a}(n_{jr_1}, n_{jr_2}, \ldots, n_{jr_{d_j}}; \overset{*}{m})$$

zu erhalten.
Die Gesamtanzahl von M-functions einer Kategorie der Familie A'' ist dann:

$$MP = \sum_{\overset{*}{m}=\mathbf{R}}^{\mathbf{M}} \mathrm{a}(n_{jr_1}, n_{jr_2}, \ldots, n_{jr_{d_j}}; \overset{*}{m}) \tag{8.17}$$

8.4.3.4 Exemplarische Berechnung

Um diese Ergebnisse zu illustrieren, sollen sie in einer einfachen Berechnung demonstriert werden. Betrachtet wird hierzu die Gruppe G_1 von $L(2,2)$ mit den Klassen $C_{11} = \alpha, C_{12} = \beta, C_{13} = \gamma$. G_1 enthält also:

$$d_1 = 3$$

Klassen. Da $\binom{3}{2} = 3$ $L(2,2)$ Einheiten zu einem $L(2,3)$ Verbund komponiert werden und alle verwendeten Einheiten aus G_1 stammen, ist:

$$\lambda_1 = 3.$$

In den Berechnungen zu Tabelle 8.2 wurde in Gleichung (8.11) berechnet:

$$\varphi(3,2) = 10.$$

Diese zehn Familien sind in der Tabelle 2 aufgeführt. Die Werte $\mathrm{a}(n_{jr_1}, n_{jr_2}, \ldots, n_{jr_{d_j}}; \overset{*}{m})$ für die Kategorien dieser Familien lassen sich dann folgendermassen bestimmen (unter Beschränkung auf die Familien α^3, $\alpha^2\beta$ und $\alpha\beta\gamma$):
Für die Familie α^3 ist nach (8.14):

$$\mathrm{a}(3,0,0;1) = 1$$

und

$$\mathrm{a}(3,0,0;\overset{*}{m}) = 0 \qquad \text{für} \quad \overset{*}{m} \neq 1,$$

also

$$MP = \sum_{\overset{*}{m}=\mathbf{R}}^{\mathbf{M}} \mathrm{a}(3,0,0;\overset{*}{m}) = \mathbf{1}$$

Die Verteilung für die Familie $\alpha^2\beta$ ist:

$$\begin{aligned} n_{1r_1} &= 2 \qquad \text{für } r_1 = k_1 = 1 \\ n_{1r_2} &= 1 \qquad \text{für } r_2 = k_1 + 1 \\ n_{1r_3} &= 0 \qquad \text{für } r_3 = k_1 + 2. \end{aligned}$$

Nach den Gleichungen (8.12) und (8.13) ist:

$$\mathbf{R} = 2$$

und

$$\mathbf{M} = 2$$

also muß gelten:

$$\mathrm{a}(2,1,0;\overset{*}{m}) = 0 \qquad \text{für} \quad \overset{*}{m} \neq 2$$

und

$$\begin{aligned} \mathrm{a}(2,1,0;2) &= \left[\binom{2-1}{1}1!\right]\mathrm{a}(3,0,0;2) \\ &\quad + \sum_{q=1}^{1}\binom{2-1-q}{1-q}\frac{1!}{q!}\mathrm{a}(3,0,0;2-q) \\ &= \binom{1}{1}1! \times 0 + \binom{0}{0}\frac{1!}{1!}\mathrm{a}(3,0,0;1) \\ &= 0+1 \\ &= \mathbf{1}. \end{aligned}$$

Wobei $\Delta r = 1$. Also ist:

$$\begin{aligned} MP &= \sum_{\overset{*}{m}=\mathbf{R}}^{\mathbf{M}} \mathrm{a}(2,1,0;\overset{*}{m}) \\ &= \mathbf{1}. \end{aligned}$$

Für die Familie $\alpha\beta\gamma$ ist die Verteilung der λ_1 Einheiten über den d_j Klassen:

$$\begin{aligned} n_{1r_1} &= 1 \qquad \text{für } r_1 = k_1 = 1 \\ n_{1r_2} &= 1 \qquad \text{für } r_2 = k_1 + 1 \\ n_{1r_3} &= 1 \qquad \text{für } r_3 = k_1 + 2. \end{aligned}$$

Die Werte $\mathrm{a}(1,1,1;\overset{*}{m})$ werden aus $\mathrm{a}(2,1,0;\overset{*}{m})$ hergeleitet, wobei eine der beiden α Einheiten (vom Grad $r = 1$) durch eine γ Einheit vom Grad $r = 3$ ersetzt wird. Also ist:

$$\Delta r = 2.$$

Somit ist:

$$\mathbf{R} = 3$$

und

$$\mathbf{M} = 4.$$

Also:

$$3 \leq \overset{*}{m} \leq 4.$$

Deswegen ist:

$$\mathrm{a}(1,1,1;\overset{*}{m}) = 0 \qquad \text{für} \quad \overset{*}{m} < 3 \text{ und } \overset{*}{m} > 4$$

und

$$\begin{aligned}
\mathrm{a}(1,1,1;3) &= \binom{3-1}{2} 2!\mathrm{a}(2,1,0;3) \\
&\quad + \sum_{q=1}^{2} \binom{3-1-q}{2-q} \frac{2!}{q!} \mathrm{a}(2,1,0;3-q) \\
&= \binom{2}{2} 2! \times 0 \\
&\quad + \binom{1}{1} \frac{2!}{1!} \mathrm{a}(2,1,0;2) \\
&\quad + \binom{0}{0} \frac{2!}{2!} \mathrm{a}(2,1,0;1) \\
&= 0 + (2 \times 1) + (1 \times 0) \\
&= \mathbf{2}
\end{aligned}$$

sowie:

$$\begin{aligned}
\mathrm{a}(1,1,1;4) &= \binom{4-1}{2} 2!\mathrm{a}(2,1,0;4) \\
&\quad + \sum_{q=1}^{2} \binom{4-1-q}{2-q} \frac{2!}{q!} \mathrm{a}(2,1,0;4-q) \\
&= \binom{3}{2} 2! \times 0 \\
&\quad + \binom{2}{1} \frac{2!}{1!} \mathrm{a}(2,1,0;3) \\
&\quad + \binom{1}{1} \frac{2!}{2!} \mathrm{a}(2,1,0;2) \\
&= 0 + (2 \times 2 \times 0) + (1 \times 1) \\
&= \mathbf{1}.
\end{aligned}$$

Also ist:

$$\begin{aligned}
MP &= \sum_{\overset{*}{m}=\mathbf{R}}^{\mathbf{M}} \mathrm{a}(1,1,1;\overset{*}{m}) \\
&= 2 + 1 \\
&= \mathbf{3}.
\end{aligned}$$

Die Struktur $[I, E, T_E]$ ist ein Beispiel für einen Verbund vom Typ $\alpha\beta\gamma$. Für $[I, E, T_E]$ existieren $MP = 3$ verschiedene M-functions $[IET_E]_1$, $[IET_E]_2$ und $[IET_E]_3$:

$[IET_E]_1$	1	2	3
1	○	○	△
2	○	○	△
3	□	△	○

$[IET_E]_2$	1	2	3
1	○	○	□
2	○	○	△
3	△	△	○

$[IET_E]_c$	1	2	3
1	○	○	□
2	○	○	△
3	⋆	△	○

Entsprechend unseren Berechnungsergebnissen bestehen die beiden ersten M-functions aus drei verschiedenen Kenogrammen (○, △, □) und die dritte aus vier Kenogrammen (○, △, □, ⋆).

8.4.3.5 MP der Kategorien der allgemeinen Familie A

In diesem Beispiel wurden die Werte für $\mathrm{a}(1,1,1;\overset{*}{m})$ aus $\mathrm{a}(2,1,0;\overset{*}{m})$ hergeleitet und nicht direkt aus $\mathrm{a}(3,0,0;\overset{*}{m})$. Der Grund hierfür liegt in der Beschaffenheit der Rekursionsgleichung (8.16). Es ist also festzustellen, daß Gleichung (8.16) eine Familie A' nur dann aus der Familie A herleiten kann, wenn beide Familien den folgenden Anforderungen genügen:

(a) Alle Werte n_{jr_i} außer zweien (n_{jr_a}, n_{jr_b}) müssen identisch sein.

(b) Sind für die Familie A diese beiden Werte n_{jr_a} und n_{jr_b} und für A' n'_{jr_a} und n'_{jr_b}, dann muß gelten:

$$n'_{jr_a} = n_{jr_a} - 1$$

und

$$n'_{jr_b} = n_{jr_b} + 1.$$

(c) $r_a = k_j$ und $r_b = k_j + \Delta r$.

Im weiteren Verlauf wird gezeigt , daß trotz dieser Einschränkung für alle möglichen Familien A' Herleitungen nach Gleichung (8.16) existieren.
In den bisherigen Überlegungen wurde davon ausgegangen, daß alle zu komponierenden Einheiten aus nur einer Gruppe G_j stammen. Na erweitert jetzt ihre Überlegungen um Fälle, bei denen Einheiten aus zwei Gruppen komponiert werden. Da $L(2,2)$ zwei Rahmengruppen enthält führt dies ihre Arbeit, zumindest hinsichtlich $L(2,2)$, zu einem Abschluß.
Eine Familie A_{00} enthalte Einheiten aus zwei Gruppen G_1 und G_2 mit der folgenden Verteilung:

$$n_{1r} = \begin{cases} \lambda_1 & \text{für} \quad r = k_1 \\ 0 & \text{für alle anderen Werte } r, \end{cases}$$

$$n_{2r} = \begin{cases} \lambda_2 & \text{für} \quad r = k_2 \\ 0 & \text{für alle anderen Werte } r, \end{cases}$$

dann gilt für jede Familie vom Typ A_{11} mit der Verteilung:

$$n_{2r} = \begin{cases} \lambda_2 & \text{für} \quad r = k_2 \\ 0 & \text{für alle anderen Werte } r, \end{cases}$$

$$n_{1r} = \begin{cases} n_{1r_1} & \text{für} \quad r = r_1 \\ n_{1r_2} & \text{für} \quad r = r_2 \\ \vdots & \\ n_{1r_{d_1}} & \text{für} \quad r = r_{d_j} \end{cases}$$

weiterhin Gleichung (8.16) (Theorem 4.11).
Beweis: A_{01} repräsentiere Familien mit folgender Verteilung:

$$n_{1r} = \begin{cases} \lambda_1 - 1 & \text{für } r = k_1 \\ 1 & \text{für } r = k_1 + 1 \\ 0 & \text{für alle anderen Werte } r, \end{cases}$$

$$n_{2r} = \begin{cases} \lambda_2 & \text{für } r = k_2 \\ 0 & \text{für alle anderen Werte } r. \end{cases}$$

A_{01} unterscheidet sich von A' nur durch die zusätzlichen λ_2 Einheiten aus G_2 vom Grad k_2. Die Anwesenheit dieser zusätzlichen Einheiten betrifft aber nur die Werte $\mathbf{M}_{01}$ und $\mathbf{R}_{01}$. Da A_{00} und A_{01} den drei Anforderungen (a), (b), (c) genügen, gilt für sie Gleichung (8.16). Durch wiederholte Anwendung dieser Gleichung mit verschiedenen Werten für Δr können die Werte $\mathrm{a}(\ldots;\overset{*}{m})$ für die Kategorien der Familien A_{11} aus den Werten für A_{00} abgeleitet werden. ∎

Es ist möglich $\mathrm{a}(\ldots;\overset{*}{m})$ für Kategorien einer Familie A mit der Verteilung $n_{1r_1}, n_{1r_2}, \ldots, n_{1r_{d_1}}, n_{2r_1}, n_{2r_2}, \ldots, n_{2r_{d_2}}$ aus den Werten $\mathrm{a}(\ldots;\overset{*}{m})$ für A_{00} durch wiederholte Anwendung von Gleichung (8.16) zu berechnen. Der Beweis hierfür verläuft analog zum vorhergehenden.

Die Werte $\mathrm{a}(\ldots;\overset{*}{m})$ für Kategorien mit beliebiger Verteilung der n_{jr} lassen sich aus den Werten $\mathrm{a}(\ldots;\overset{*}{m})$ für Kategorien mit der Verteilung:

$$n_{jr_i} = \begin{cases} \lambda_j & \text{für } r_i = k_j \\ 0 & \text{für } r_i \neq k_j \end{cases} \qquad \text{mit } 1 \leq j \leq GF(s)$$

durch mehrfache Anwendung der Formel (8.16) berechnen.
Die Gesamtanzahl von M-functions, die eine gegebene Kategorie repräsentiert, ist also durch:

$$MP = \sum_{\overset{*}{m}=\mathbf{R}}^{\mathbf{M}} \mathrm{a}(n_{1r_1}, n_{1r_2}, \ldots, n_{1r_{d_1}}, n_{2r_1}, n_{2r_2}, \ldots, n_{2r_{d_2}}; \overset{*}{m}) \tag{8.18}$$

gegeben.

8.5 Fazit

Die gewonnen Ergebnisse stellt Na in einer abschließenden Tabelle zusammen, von der hier nur der Teil für die Komposition von $L(2,3)$ Strukturen unter Korrektur von Berechnungsfehlern wiedergegeben ist.
Für jede der $GF(n)$ Rahmengruppen ist die Anzahl von Rahmen, die sie enthält $(g\{m_k\})$, sowie die Anzahl φ von Familien je Rahmen in dieser Gruppe angegeben. In der nächsten Spalte sind alle diese Familien aufgeführt. Für jede dieser Familien ist die Anzahl σ von Verbundstrukturen, die sie enthält, berechnet[3]. Für jede dieser Kategorien sind die Werte $\mathbf{R}$ und $\mathbf{M}$ sowie die Anzahl von M-functions MP angegeben.

[3]Die tatsächliche Anzahl von Verbundstrukturen ergibt sich aus $\sigma \times g\{m_{k_J}\}$. Vgl. die Anmerkung auf Seite 157.

G_i	$g\{3_i\}$	φ	Familie	σ	MP	**R**	**M**
			α^3	1	1	1	1
			$\alpha^2\beta$	9	1	2	2
			$\alpha^2\gamma$	3	1	3	3
			$\alpha\beta^2$	27	1+1 = 2	2	3
			$\alpha\beta\gamma$	18	2+1 = 3	3	4
G_1	1	10	$\alpha\gamma^2$	3	2+4+1 = 7	3	5
	○○○		β^3	27	1+3+1 = 5	2	4
			$\beta^2\gamma$	27	4+5+1 = 10	3	5
			$\beta\gamma^2$	9	4+14+8+1 = 27	3	6
			γ^3	1	4+32+38+12+1 = 87	3	7
			$\alpha\xi^2$	16	1	2	2
			$\alpha\xi\rho$	40	1	3	3
			$\alpha\xi\phi$	8	1	4	4
			$\alpha\rho^2$	25	1+1 = 2	3	4
			$\alpha\rho\phi$	10	2+1 = 3	4	5
			$\alpha\phi^2$	1	2+4+1 = 7	4	6
			$\beta\xi^2$	48	1+1 = 2	2	3
			$\beta\xi\rho$	120	2+1 = 3	3	4
			$\beta\xi\phi$	24	3+1 = 4	4	5
G_2	3	18	$\beta\rho^2$	75	2+4+1 = 7	3	5
	○○△		$\beta\rho\phi$	30	6+6+1 = 13	4	6
	○△○		$\beta\phi^2$	3	6+18+9+1 = 34	4	7
	○△△		$\gamma\xi^2$	16	2+1 = 3	3	4
			$\gamma\xi\rho$	40	2+4+1 = 7	3	5
			$\gamma\xi\phi$	8	6+6+1 = 13	4	6
			$\gamma\rho^2$	25	2+10+7+1 = 20	3	6
			$\gamma\rho\phi$	10	12+24+10+1 = 47	4	7
			$\gamma\phi^2$	1	12+60+54+14+1 = 141	4	8
			ξ^3	64	1	3	3
			$\xi^2\rho$	240	1+1 = 2	3	4
			$\xi^2\phi$	48	2+1 = 3	4	5
			$\xi\rho^2$	300	1+3+1 = 5	3	5
			$\xi\rho\phi$	120	4+5+1 = 10	4	6
g_3	1	10	$\xi\phi^2$	120	4+14+8+1 = 27	4	7
	○△□		ρ^3	125	1+7+6+1 = 15	3	6
			$\rho^2\phi$	75	8+19+9+1 = 37	4	7
			$\rho\phi^2$	15	8+46+46+13+1 = 114	4	8
			ϕ^3	1	8+100+184+98+18+1 = 409	4	9

Abbildung 8.4: Analyse der $L(2,3)$ Komposition nach [Na64], S. 112f.

Nas Analyse der $L(n,m)$ Systeme wird abschliessend kurz zusammengefasst: Ein $L(n,m)$ System ist die Menge aller:

$$\sum_{i=1}^{n^m} S(n^m, i)$$

TNF–Kenogrammsequenzen der Länge n^m (M-functions). Ein solches System wird anhand der Rahmenstruktur der M-functions in $GF(m)$ Rahmengruppen aufgeteilt. Die M-functions in diesen Gruppen werden dann entsprechend der Verteilung der λ_j $L(n,s)$ Einheiten über den d_j Klassen von G_j in φ Familien eingeteilt. Jede dieser Familien enthält σ Kategorien (Verbundstrukturen). Jede Kategorie enthält MP M-functions, von denen a$(\ldots;\overset{*}{m})$ vom Grad $\overset{*}{m}$ sind, mit $\mathbf{R} \leq \overset{*}{m} \leq \mathbf{M}$.

8.6 Implementierung der Algorithmen

Abschließend werden nun die entwickelten Formeln und Algorithmen implementiert. Die Funktion

```
fun sum from to f=
   if (from > to) then 0
   else (f from) + sum  ( from + 1) to f;
```

berechnet die Summenformel $\sum_{k=from}^{to} f(k)$. Die Summe $\sum_{k=0}^{100} k^2$ ergibt sich beispielsweise als:

```
-sum 0 100 (fn k => k*k);
>338350 : int
```

Die Funktion:

```
fun S (n,1) = 1
   |S (n,k) =
     if k>n then 0
     else if k=n then 1
     else S(n-1,k-1) + k*S(n-1,k);
```

berechnet die Stirlingzahl der zweiten Art nach [And65]. Beispielsweise ergibt:

```
-S(10,5);
>42525 : int
```

Die Anzahl der Rahmen eines $L(n,m)$ Systems wird nach Gleichung (8.1) angegeben durch

$$NF(m) = \sum_{k=1}^{m} S(m,k).$$

Als ML-Funktion notiert:

```
fun NF m = sum 1 m (fn k => S(m,k));
```

Für ein $L(2,4)$ System ergibt sich die Anzahl der Rahmen als:

```
-NF 4;
>15 : int
```

Die Anzahl der Partitionen $P(n,k)$ von n Objekten über k Objektklassen wird nach [Sch67a] berechnet durch:

```
fun P (n,1) = 1
   |P (n,k) =
     if k>n then 0
     else if k=n then 1
     else P(n-1,k-1) + P(n-k,k);
```

Beispielsweise ergibt:

```
- P(10,5);
> 7 : int
```

Die Anzahl von Rahmengruppen eines $L(n,m)$ Systems berechnet sich nach Gleichung (8.2) als:

$$GF(m) = \sum_{k=1}^{m} P(m,k).$$

Als ML-Funktion notiert:

```
fun GF m = sum 1 m (fn k => P(m,k));
```

Für ein $L(2,4)$ System ergibt sich die Anzahl der Rahmengruppen als:

```
- GF 4;
> 5 : int
```

Die Fakultätsfunktion $n!$ ist durch folgende ML-Funktion bestimmt:

```
fun fak 0=1
   |fak n= n * fak (n-1);

- fak 10;
> 3628800 : int
```

Der Binomialkoeffizient $\binom{n}{k}$ wird berechnet durch die ML-Funktion:

```
fun choose n k=
   (fak n) div ((fak k)* fak (n-k));

- choose 4 2;
> 6 : int
```

Die Potenzfunktion m^n wird angegeben durch die ML-Funktion:

```
fun powers m n=
   if n=0 then 1
   else if n=1 then m
   else m*(powers m (n-1));

- powers 3 5;
> 243 : int
```

Die durch Gleichung (8.10) bestimmte Funktion σ wird durch die folgende ML-Funktion implementiert:

```
fun sigma n11 n12 n13 n22 n23 n24=
  let
   val l1=n11+n12+n13;
   val l2=n22+n23+n24;
  in
    ((fak l1) div ((fak n11)*(fak n12)*(fak n13)))
   * (powers 3 n12) *
    ((fak l2) div ((fak n22)*(fak n23)*(fak n24)))
   * (powers 4 n22) * (powers 5 n23)
  end;
```

Die sechs Variablen `n11, n12, n13, n22, n23, n24` repräsentieren die Verteilung der $\lambda_1 + \lambda_2$ Einheiten über den sechs $L(2,2)$ Klassen $\alpha, \beta, \gamma, \xi, \rho, \phi$. Um also beispielsweise $\sigma(\beta\xi\rho)$ zu berechnen, wird eingegeben:

```
- sigma 0 1 0 1 1 0;
```

Das Ergebnis lautet:

```
> 120 : int
```

Die Hilfsfunktion `max` berechnet das Maximum einer Liste von Zahlen:

```
fun max []=0
   |max [x]=x
   |max (x::xs)=
     let
      val restmax = max xs;
     in
      if (x > restmax) then x else restmax
     end;
```

Der Grad einer $L(2,2)$ Gruppe G_j ist durch k_j bestimmt:

```
fun k n= n;
```

Der minimale Akkretionsgrad **R** einer Komposition von Morphogrammen ist durch Gleichung (8.12) bestimmt. Implementiert wird **R** durch:

```
fun R n11 n12 n13 n22 n23 n24=
  let
   val kJ = if (n11+n12+n13)=3 then 1
            else if (n22+n23+n24)=3 then 3
            else 2;
  in
   max (kJ::(map (fn (n,j,r) => if (n=0) then 0
                                else (k j)+r)
                  [(n11,1,0),(n12,1,1),(n13,1,2),
                   (n22,2,0),(n23,2,1),(n24,2,2)]))
  end;
```

Der maximale Akkretionsgrad **M** einer Komposition ist durch Gleichung (8.13) bestimmt. Implementiert wird er durch:

```
fun M n11 n12 n13 n22 n23 n24=
  let
   val kJ = if (n11+n12+n13)=3 then 1
            else if (n22+n23+n24)=3 then 3
            else 2;
  in
   kJ + n12 + (2*n13) + n23 + (2*n24)
  end;
```

Auch die Funktionen `R` und `M` benötigen wieder die Angabe der Verteilung der Einheiten über den sechs Gruppen:

```
- R 0 1 0 1 1 0;
> 3 : int
- M 0 1 0 1 1 0;
> 4 : int
```

Nach der Bedingung (c) ist $\Delta r = r_b - k_j$. Dieser Zusammenhang wird berechnet durch:

```
fun dr n2 n3=
  if n3<>0 then 2
  else if n2<>0 then 1
  else 0;
```

Die Werte von `a`$(\ldots)$ für eine Familie vom Typ A' sind durch Gleichung (8.16) bestimmt. Berechnet wird diese Funktion durch:

```
exception A
fun a1 (n1,n2,n3,m,j) =
   if (j=1) andalso
      ((m<(R n1 n2 n3 0 0 0)) orelse (m>(M n1 n2 n3 0 0 0)))
      then 0
   else if j=2 andalso
           ((m<(R 0 0 0 n1 n2 n3)) orelse (m>(M 0 0 0 n1 n2 n3)))
      then 0
   else if n1=n1+n2+n3 then 1
   else if n3<>0 then
    sum 0 (dr n2 n3)
        (fn q => (choose (m-(k j)-q) ((dr n2 n3)-q))
                *((fak(dr n2 n3)) div (fak q))
                *a1(n1+1,n2,n3-1,m-q,j))
   else
     sum 0 (dr n2 n3)
        (fn q => (choose (m-(k j)-q) ((dr n2 n3)-q))
                *((fak(dr n2 n3)) div (fak q))
                *a1(n1+1,n2-1,n3,m-q,j));
```

Die folgende ML-Funktion:

```
exception B;
fun a (n11,n12,n13,n22,n23,n24,m) =
 let
   val J = if (n11+n12+n13)=3 then 1
           else if (n22+n23+n24)=3 then 3
```

```
           else if (n11+n12+n13+n22+n23+n24)=3 then 2
           else raise A;
in
  if (m<(R n11 n12 n13 n22 n23 n24))
      orelse (m>(M n11 n12 n13 n22 n23 n24))
     then 0
  else if J=1 then a1(n11,n12,n13,m,1)
  else if J=3 then a1(n22,n23,n24,m,2)
  else if n11=1 andalso n22=2 then 1
  else if (n12>0 orelse n13>0) then
    sum 0 (dr n12 n13)
        (fn q => (choose (m-(k 1)-q) ((dr n12 n13)-q))
                *((fak(dr n12 n13)) div (fak q))
                *a(n11+1,if n12=0 then 0 else n12-1,
                         if n13=0 then 0 else n13-1,
                   n22,n23,n24,m-q))
  else if (n24>0) then
    sum 0 (dr n23 n24)
        (fn q => (choose (m-(k 2)-q) ((dr n23 n24)-q))
                *((fak(dr n22 n24)) div (fak q))
                *a(n11,n12,n13,n22+1,n23,if n24=0 then 0
                                         else n24-1,m-q))
  else if (n23>0) then
    sum 0 (dr n23 n24)
        (fn q => (choose (m-(k 2)-q) ((dr n23 n24)-q))
                *((fak(dr n22 n24)) div (fak q))
                *a(n11,n12,n13,n22+1,if n23=0 then 0
                                     else n23-1,n24,m-q))
  else raise B
 end;
```

berechnet die Werte für a(...) der Familien vom allgemeinen Ty p A. Der Wert a(...;6) für die Familie $\gamma\phi^2$ wird du rch die folgende Applikation berechnet:

```
- a(0,0,1,0,0,2,6);
> 54 : int
```

Die Hilfsfunktion:

```
exception Fromto;
fun fromto n m =
  if n>m+1 then raise Fromto
  else if n=m+1 then nil
  else n::fromto (n+1) m;
```

erzeugt eine Liste der ganzen Zahlen von n bis m.
Die Funktion MP für $L(2,3)$ ist nach Gleichung (8.18) gegeben als

$$MP = \sum_{\overset{*}{m}=\mathbf{R}}^{\mathbf{M}} \mathtt{a}(n_{11}, n_{12}, n_{13}, n_{22}, n_{23}, n_{24}; \overset{*}{m}).$$

Implementiert wird MP durch:

```
fun MP n11 n12 n13 n22 n23 n24=
```

```
let
 val alist =
      map (fn m => a(n11,n12,n13,n22,n23,n24,m))
          (fromto (R n11 n12 n13 n22 n23 n24)
                  (M n11 n12 n13 n22 n23 n24))
in
 (map (fn x => (print x ; print " ")) alist;
 sum (R n11 n12 n13 n22 n23 n24) (M n11 n12 n13 n22 n23 n24)
     (fn m => a(n11,n12,n13,n22,n23,n24,m)))
end;
```

Die Eingabe:

```
- MP 0 0 0 0 0 3;
```

berechnet MP für die Familie ϕ^3:

```
8 100 184 98 18 1
> 409 : int

- length(Kom [mg 15,mg 15,mg 15]);
> val it = 409 : int;
```

Dies entspricht den Ergebnissen aus Tabelle 8.4 und deckt sich mit der Anzahl der konstruktiv erzeugten Morphogrammatrizen für die Komposition `Kom` $[\phi, \phi, \phi]$.
Nach diesen analytischen Untersuchungen zur operationalen Struktur und Dynamik der Morphogrammatik sollen nun im letzten Teil der Arbeit die Einbettungen verschiedener Logiken in die Morphogrammatik studiert werden. Besonderes Gewicht soll hierbei auf die Ableitung der polykontexturalen Logik aus der Morphogrammatik gelegt werden.

Teil IV

Anknüpfungen

Kapitel 9

Einbettung von Logiken in der Morphogrammatik

Müsset im Naturbetrachten
Immer eins wie alles achten:
Nichts ist drinnen, nichts ist draußen;
Denn was innen, das ist außen.
So ergreifet ohne Säumnis
Heilig öffentlich Geheimnis.

Freuet euch des wahren Scheins,
Euch des ernsten Spieles:
Kein Lebendiges ist ein Eins,
Immer ists ein Vieles.

J.W. v. Goethe

9.1 Stellenwertlogik in der Morphogrammatik

9.1.1 Einführung

In seinen Arbeiten „*Die aristotelische Logik des Seins und die nicht-aristotelische Logik der Reflexion*“ [Gue80] und „*Das Problem einer Formalisierung der transzendentaldialektischen Logik*“ [Gue80] entwirft Günther ausgehend von einer an Hegel anschließenden philosophischen Kritik der klassischen Logik die Konzeption einer *Stellenwertlogik* und deren Einbettung in die Morphogrammatik. Die Stellenwertlogik ist der Vorläufer der Polykontexturalen Logik und unterscheidet sich von dieser hautsächlich durch ihre *globale* Betrachtung der Verbundstrukturen, die in der PKL zu Gunsten einer deutlicheren Abgeschlossenheit der lokalen Subsysteme aufgegeben wurde. Anhand der Stellenwertlogik kann der Ansatz der Polykontexturalen Logik und ihr formaler Aufbau einführend illustriert werden. Die argumentative Untermauerung und der formale Aufbau der Stellenwertlogik soll hier jedoch nur angedeutet werden (zu einer vollständigen Darstellung sei auf [Gue80] verwiesen), da im Rahmen dieser Arbeit lediglich die *Einbettung* der Stellenwertlogik (SWL) und der Polykontexturalen Logik in die Morphogrammatik von Interesse ist.
Zentrales Anliegen der SWL ist die Formalisierung von Bewußtseinsstrukturen, die

innerhalb der klassischen Aussagenlogik nicht abgebildet werden können. Die klassische AL ist durch ihre zweiwertige Axiomatik monokontextural in sich abgeschlossen, d.h. sie stellt die Formulierung der Erkenntnissituation eines einzigen Subjektes dar, das einer vollkommen homogenen und subjektfreien, rein objektiven Welt gegenübersteht. Da sich diese in der Aussagenlogik formalisierte Ontologie nicht mit der empirisch beobachtbaren Vielzahl von erkennenden und handelnden Subjekten deckt, fordert Günther, daß „*ein logischer Formalismus nicht einfach zwischen Subjekt und Objekt zu unterscheiden hat, er muß vielmehr die Distribution der Subjektivität in eine Vielzahl von Ichzentren in Betracht ziehen. Das aber bedeutet, daß das zweiwertige Verhältnis sich in einer Vielzahl von ontologischen Stellen abspielt, die nicht miteinander zur Deckung gebracht werden können. [...]*
Jedes Einzelsubjekt begreift die Welt mit derselben Logik, aber es begreift sie von einer anderen Stelle im Sein. Die Folge davon ist: insofern, als alle Subjekte die gleiche Logik benutzen, sind ihre Resultate gleich, insofern aber, als die Anwendung von unterschiedlichen Stellen her geschieht, sind ihre Resultate verschieden. Dieses Zusammenspiel von Gleichheit und Verschiedenheit in logischen Operationen wird durch die Stellenwert-Theorie der mehrwertigen Logik beschrieben. Die zusätzlichen [logischen] Werte sind hier überhaupt nicht mehr Werte im klassischen Sinn, ... sie repräsentieren vielmehr die unterschiedlichen ontologischen Stellen, an denen zweiwertige Bewußtseinsoperationen auftreten können.“[1]

9.1.2 Der Stellenwertverbund logischer Subsysteme

Im Folgenden soll umrissen werden, wie ein solches Stellenwertsystem von zweiwertigen Logiken durch die Morphogrammatik ermöglicht wird. Zur Vereinfachung der Darstellung beziehen wir uns dabei auf das einfachste Stellenwertsystem von $\binom{3}{2} = 3$ zweiwertigen Aussagenlogiken, $L(2,3)$. Jedes der Subsysteme S_1, S_2, S_3 realisiert nun eine von den jeweils anderen Systemen unabhängige zweiwertige Aussagenlogik mit den Wertepaaren $S_{k(i,j)} : (i,j)$:

$$S_1 : (1,2), S_2 : (2,3), S_3 : (1,3).$$

Die geordnete Liste der numerierten Subsystemwertpaare (i,j) wird allgemein von der Funktion `subsystems(n)` berechnet.

Beispiel: In allen drei Systemen soll eine klassische Konjunktion $\wedge$ operieren:

$S_1:$ p	q	$p \wedge_1 q$
1	1	1
1	2	2
2	1	2
2	2	2

$S_2:$ p	q	$p \wedge_2 q$
2	2	2
2	3	3
3	2	3
3	3	3

$S_3:$ p	q	$p \wedge_3 q$
1	1	1
1	3	3
3	1	3
3	3	3

[1] [Gue80] Bd.3, p.87.

Die unabhängigen $\binom{n}{2}$ Subsysteme sollen nun in den Verbundzusammenhang der SWL integriert werden. Günther realisiert dies, indem er die Aussagenvariablen der einzelnen Subsysteme zusammenfasst und eine *globale* Wertbelegung der Variablen mit n und nicht mit jeweils zwei Werten definiert. Dies führt zu einem globalen n^2-stelligen Stellenwertjunktor J :

Beispiel:

Variablen		S_1	S_2	S_3	Stellenwertjunktor
p	q	$\wedge_1$	$\wedge_2$	$\wedge_3$	$J = \wedge_1 \wedge_2 \wedge_3$
1	1	1	$\overline{VB_1}$	1	1
1	2	2			2
1	3			3	3
2	1	2			2
2	2	2	$\overline{VB_2}$ 2		2
2	3		3		3
3	1			3	3
3	2		3		3
3	3		3 $\overline{VB_3}$	3	3

$J = \wedge_1 \wedge_2 \wedge_3$ in Matrixdarstellung:

J	1	2	3
1	1	2	3
2	2	2	3
3	3	3	3

Offensichtlich läßt sich die Konstruktion eines solchen globalen Stellenwertjunktors nur durchführen, wenn die Subsystemjunktoren die Vermittlungsbedingungen VB_i erfüllen, daß die jeweils erste und vierte Position der Subsystemjunktoren mit den korrespondierenden Werten der anderen Junktoren 'gleich' sind.

Beispiel: Die drei Junktoren j_1, j_2, j_3:

j_1	1	2
1	1	2
2	2	2

j_2	2	3
2	2	3
3	3	3

j_3	1	3
1	1	3
3	3	1

lassen sich nicht zu einem globalen Stellenwertjunktor zusammenfassen:

Variablen		S_1	S_2	S_3	Stellenwertjunktor
p	q	j_1	j_2	j_3	$J = j_1 j_2 j_3$
1	1	1	$\overline{VB_1}$	1	1
1	2	2			2
1	3			3	3
2	1	2			2
2	2	2	$\overline{VB_2}$ 2		2
2	3		3		3
3	1			3	3
3	2		3		3
3	3		$\boxed{3\ \overline{VB_3}\ 1}$		$\boxed{1,3}$

Für die letzte Position des Junktors läßt sich keine eindeutige Belegung angeben, da an dieser Stelle die Vermittlungsbedingung VB_3 nicht erfüllbar ist.

Die zur Erfüllung der Vermittlungsbedingungen notwendige Gleichheitsbeziehung läßt sich jedoch nicht über die semiotische Identität definieren, da ja jedes Subsystem ein vom lokalen *Tertium non datur* abgeschlossenes System ist, in dem keinerlei Beziehungen zu externen Werten hergestellt werden können. Diese Gleichheitsrelation kann also nicht subsystemimmanent bestimmt werden.
Die Wertgleichheit an den Verknüpfungsstellen der Subsysteme läßt sich von einem globalen, alle Werte n umfassenden Standpunkt rein syntaktisch entscheiden. Streng genommen kann aber auch diese Methode nicht zur Definition der Vermittlungsbedingungen benutzt werden, da sie die intendierte Lokalität und Abgeschlossenheit der Subsysteme anullieren würde.
Das Definitionsproblem der Vermittlungsbedingungen läßt sich nach Günther nur mittels eine morphogrammatischen Verknüpfungstheorie logischer Systeme zufriedenstellend lösen, die unabhängig von der Wertstruktur der Subsysteme ist. Die in Kapitel 6 eingeführte f-c-Klassifikation der Basismorphogramme (d.h. der Tritonormalformen aussagenlogischer Operatoren) beantwortet die Frage der Komponierbarkeit beliebiger $n \times n$ Morphogrammatrizen aus $\binom{n}{2}$-stelligen Morphogrammketten.
Ob also zu den lokalen Junktoren $j_1, \ldots, j_{\binom{n}{2}}$ ein globaler Stellenwertjunktor J existiert, läßt sich durch die Applikation $\texttt{exmm}([\texttt{tnf j}_1, \ldots, \texttt{tnf j}_{\binom{n}{2}}])$; berechnen. Falls diese Berechnung den Wert `false : bool` liefert, sind die Vermittlungsbedingungen nicht erfüllt und es läßt sich kein Globaljunktor generieren. Lautet das Ergebnis jedoch `true : bool`, läßt sich mit der Applikation $\texttt{Kom}([\texttt{tnf j}_1, \ldots, \texttt{tnf j}_{\binom{n}{2}}])$; die Menge der möglichen Kenogrammstrukturen des Gesamtjunktors berechnen.

9.1.3 Zur Dekomponierbarkeit der Stellenwertjunktoren

Die in 5.4 entwickelten Dekomposition von Morphogrammatrizen in Subsystemmorphogramme, $dek(Q^n)$, arbeitet auf beliebigen Matrizen.
Da die Junktoren der Stellenwertlogik als Komposition lokaler logischer Funktionen definiert ist, sind nur solche Funktionen als Stellenwertjunktoren zulässig, die sich entsprechend der Axiomatik der lokalen Logiken dekomponieren lassen.

Beispiel: Der globale Stellenwertjunktor J:

J	1	2	3
1	1	2	3
2	1	1	2
3	1	1	1

soll in die Subsystemjunktoren j_1, j_2 und j_3 zerlegt werden:

j_1	1	2
1	1	2
2	1	1

j_2	2	3
2	1	2
3	1	1

j_3	1	3
1	1	3
3	1	1

Hier fällt auf, daß j_2 keine gültige logische Funktion des Systems $S_2(2, 3)$, weil der Funktionswert 1 das Tertium Non Datur (TND) dieses Systems

verletzt. Die beiden Junktoren j_2 und j_3 hingegen sind gültige logische Funktionen, da hier die Funktionswerte dem jeweiligen Definitionsbereich entstammen. Da j_2 keine gültige logische Funktion ist, kann J kein aus lokalen Junktoren komponierter Stellenwertjunktor sein.

Ein weiteres Dekompositionsproblem zeigt sich bei der Betrachtung logischer Funktionen mit mehr als zwei Variablen (n-äre Funktionen).

Beispiel: Es wird ein Junktor $J(p, q, r)$ betrachtet:

Variablen			S_1	S_2	S_3
p	q	r	$(1,2)$	$(2,3)$	$(1,3)$
1	1	1	x		x
1	1	2	x		
1	1	3			x
1	2	1	x		
1	2	2	x		
1	2	3		?	
1	3	1	x		x
1	3	2		?	
1	3	3	x		x
	⋮				
2	1	3		?	
	⋮				
2	3	1		?	
	⋮				
3	1	2		?	
	⋮				
3	2	1		?	
	⋮				
3	3	3		x	x

Diejenigen Stellen des Gesamtjunktors J, an denen die Aussagevariablen p, q und r insgesamt nur einen oder zwei verschiedene Werte aufweisen, lassen sich problemlos den jeweils zweiwertigen Subsystemen zuordnen. Eine solche Zurodnung ist jedoch nicht möglich für die Stellen, an denen die Variablen drei verschiedene Werte haben. Eine logische Funktion mit einer Wertkombination wie $p = 1, q = 2$ und $r = 3$ läßt sich keinem der zweiwertigen Kalküle $S_1(1,2), S_2(2,3)$ oder $S_3(1,3)$ zuordnen, da für jeden das Gesetz vom ausgeschlossenen Dritten (TND) verletzt wird.

Wie Na zeigt [Na64], existieren für Junktoren mit $n > 2$ Variablen und $m > 2$ Werten stets Wahrheitswertkombinationen (Positionen), die sich nicht den Subsystemjunktoren zuordnen lassen.
Zur Lösung dieses Problems schlägt Kaehr vor, einen Stellenwertjunktor nicht über allen kombinatorisch möglichen m^n Positionen zu definieren, sondern nur über den Positionen, die lediglich zwei verschiedene Eingabewerte über den n Variablen verteilen. Durch eine solche Einschränkung läßt sich das Prinzip der Zerlegbarkeit der

globalen Junktoren in Subjunktoren bewahren, ohne die Funktionalität der Subsysteme zu beschneiden.
Die beiden angeführten Problematiken der Stellenwertlogik stellen sich jedoch erst in einer logischen Interpretation, wo die Eigenschaften der Wertemenge und die semiotische Unterscheidbarkeit der Werte thematisiert werden.
Wird jedoch wie in der Morphogrammatik die Gestalt und Struktur logischer Funktionen untersucht und nicht eine spezifische Wertbelegung, stellen sich die obigen Probleme nicht. Wie gezeigt, lassen sich beliebige n-äre Morphogrammatrizen in Subsystemmorphogramme dekomponieren.

9.1.4 Morphogrammatische Klassifikation der SWL-Junktoren

Da in $L(2,3)$ die beiden Aussagevariablen drei verschiedene Werte annehmen können und der Gesamtjunktor J insgesamt neun Stellen aufweist, die jeweils mit einem der drei Werte belegt werden können, existieren für das $L(2,3)$ Stellenwertsystem $3^9 = 19683$ verschiedene globale Junktoren J_i. Die Anzahl verschiedener 3×3 Morphogrammatrizen die mit mindestens einem, höchstens jedoch mit drei verschiedenen Kenogrammsymbolen belegt sind, beträgt:

$$\sum_{i=1}^{3} S(3^2, i) = 1 + 255 + 3025 = 3281$$

Für die Belegung von k Kenogrammen aus einer Menge von n Werten sind genau:

$$P(n,k) = \frac{n!}{(n-k)!}$$

verschiedene *geordnete Auswahlen* möglich[2]. Diese Anzahl entspricht einer lexikographisch geordneten Wertbelegung der Matrixkenogramme und allen möglichen Negationen der Matrix.
Die Summe $N_J(3)$ aller Möglichkeiten, verschiedene Morphogrammatrizen zu bilden, diese mit Werten zu belegen und alle möglichen Negationen anzuwenden beträgt also:

$$\begin{aligned} N_J(3) &= \sum_{i=1}^{3} S(3^2, i) \times P(3^2, i) \\ &= 1 \times 3 + 255 \times 6 + 3025 \times 6 \\ &= 19683. \end{aligned}$$

Allgemein ist:

$$N_J(n) = \sum_{i=1}^{n} S(n^2, i) \times P(n^2, i) = n^{(n^2)}.$$

Alle kombinatorisch möglichen globalen Stellenwertjunktoren J lassen sich also eineindeutig auf ein Tupel $(TNF_{MM}J, neg_i)$ abbilden. Die Identifikation und Klassifikation stellenwertlogischer Operationen kann somit vollständig mittels morphogrammatischer Klassifikationen geleistet werden.

[2] [Fla69], p.48.

9.2 Polykontexturale Logik in der Morphogrammatik

9.2.1 Polykontexturale Logik als Distribution und Vermittlung formaler Systeme

Eine allgemeine Einführung in die Polykontexturale Logik wurde bereits in Kapitel 2 gegeben. An dieser Stelle geschieht deshalb eine Motivierung der PKL nur in dem Umfang wie sie für die weiteren Überlegungen notwendig ist.

Die PKL bildet komplexe Systeme ab, indem sie deren Subsysteme über mehrere logische Orte verteilt. Innerhalb dieser lokalen Subsysteme, *intrakontextural* , gilt jeweils eine klassische Logik. Die verteilten Logiken sind durch bestimmte ausserlogische Operationen miteinander verkoppelt, wodurch die Interaktion der das Gesamtsystem konstituierenden Subsysteme modelliert wird.

Grundlegendes Motiv der PKL ist also eine *Distribution* und *Vermittlung* logischer Kontexturen. Durch die Vermittlung sind die einzelnen Kontexturen nicht im Sinne einer Tyenphierarchie voneinander isoliert, sondern durch *inter-kontexturale* Übergänge miteinander verkoppelt. Die strukturelle Erweiterung der dualistischen Form der klassischen Logik zur *Reflexionsform* der PKL ergibt sich aus dem *proemiellen* (d.h. simultanen und parallelen) Zusammenspiel der Distribution und Vermittlung von Kontexturen und ist auf keinen dieser Aspekte allein reduzierbar. Die Distribution regelt die Verteilung von Logiken über Kontexturen und läßt dabei den interkontexturalen Raum unberücksichtigt. Die Vermittlung beschreibt hingegen die interkontexturale Operativität der PKL, kann aber die intrakontexturalen, logischen Aspekte nicht erfassen, da sie von aller logischen Wertigkeit abstrahieren muß. Die Distribution bildet den Hintergrund der Vermittlung, die Vermittlung den Hintergrund der Distribution. Der formale Aufbau der PKL muß dieser fundamental selbstreferentiellen Architektur Rechnung tragen und das proemielle Zusammenspiel der Distribution und Vermittlung vollständig abbilden. Im weiteren Verlauf der Darstellung wird die Konstruktion der PKL, die über beliebigen formalen Systemen aufgebaut werden kann[3], exemplarisch für den klassischen Aussagenkalkül durchgeführt. Zunächst wird mit der Distribution der Logik begonnen, die dann um die Vermittlung der distribuierten Systeme zur komplexen polykontexturalen Logik erweitert wird.

Diese knappe Skizze des Konstruktionsvorgangs der Polykontexturalen Logik nach [Kae92] soll hier nicht zu einer vollständigen Ausarbeitung der PKL ausgeweitet werden, was den Rahmen dieser Studie sprengen würde. Ziel dieser Darstellung ist es vielmehr, ein induktives Konstruktionsschema der PKL zu entwickeln, das der vom systematischen Aufbau dieser Arbeit suggerierten hierarchischen Abfolge 'Kenogrammatik – Morphogrammatik – Polykontexturale Logik' entspricht. Dabei soll einerseits dieser lineare Aufbau als Konstruktionsstrategie gerechtfertigt werden, zum anderen soll gezeigt werden, daß ein solches lineares und hierarchisches Modell aufgrund der fundamentalen Selbstreferentialität der PKL-Architektur zugunsten einer proemiellen und heterarchischen Verkoppelung der drei Strukturebenen Kenogrammatik, Morphogrammatik und PKL aufgegeben werden muß.

[3] [Kae92]

9.2.2 Distribution der Aussagenlogik

Die Distribution der klassischen Logik (allgemein: eines formalen Systems) über verschiedene Orte läßt sich als Faserung logischer Orte (Kontexturen) formalisieren, wobei je Kontextur jeweils eine zur typischen Logik L *strukturisomorphe* Logik L_i operiert. Die folgende Darstellung orientiert sich an der Arbeit von Pfalzgraf zur Formalisierung der PKL[4].

„*Diese Grundmerkmale [der Distribution und Vermittlung] von 'lokal-global-Interaktion' gegebener Strukturen und von Übergängen zwischen einzelnen beteiligten Räumen der Gesamtstruktur sind typisch für Faserungen von Räumen, wobei die Fasern jeweils mittels einer gegebenen 'typischen Faser' modelliert werden. Wesentlich für die Gesamtstruktur sind die 'Fasernübergänge'. Hinsichtlich einer PKL wäre dann von einer Faserung logischer Orte zu sprechen, wobei jede Faser (Ort) ein klassischer logischer Raum ist.*“[5]

Unter einer logischen Faserung wird ein Faserraum $\mathcal{F}$ verstanden, wobei die typische Faser ein klassischer logischer Raum L im Sinne des logischen Raums Wittgensteins ist (d.h. der Raum aller Sätze des klassischen Aussagenkalküls), mit dem die einzelnen Fasern (die Kontexturen) modelliert werden. Anstelle des klassischen Aussagenkalküls kann jedoch auch jedes andere formale System als typische Faser verwendet werden.

Für die PKL ist der *Basisraum* der Faserung eine endliche Menge $B = \{1, \ldots, i, \ldots, n\}$. L_i bezeichnet die Faser über i, d.h. die distribuierte Logik im Subsystem S_i.

Der Strukturisomorphismus $\varphi_i : L \longrightarrow L_i$ bildet die typische Logik auf die lokalen Subsysteme ab und garantiert die Einhaltung der logischen Axiomatik im Subsystem S_i. Der Morphismus $\varphi_{ji} = \varphi_j \circ \varphi_i^{-1} : L_i \longrightarrow L \longrightarrow L_j$ bezeichnet den Fasernübergang von Subsystem S_i nach S_j. $\mathcal{L}^n = L_1 \underline{\parallel} L_2 \underline{\parallel} \ldots \underline{\parallel} L_n$ bezeichnet das Gesamtsystem der distribuierten Subsysteme.

$W_i = \{T_i, F_i\}$ mit $1 \leq i \leq n$ bezeichnet die lokale Wahrheitswertmenge im System L_i. Die globale Wertemenge, W^n, entsteht durch die Zusammensetzung aller W_i: $W^n = W_1 \underline{\parallel} \ldots \underline{\parallel} W_n = \{T_1, F_1, \ldots, T_n, F_n\}$

Die grob umrissene Faserdarstellung der Distribution logischer Systeme wird im folgenden Schema 9.1 veranschaulicht.

Die Fasernübergänge zwischen den einzelnen Kontexturen L_i und L_j, $\varphi_{ji} = \varphi_j \circ \varphi_i^{-1} : L_i \longrightarrow L \longrightarrow L_j$ sind strukturbewahrende Abbildungen (Isomorphismen) die stets unter Rückgriff auf die typische Logik L stattfinden. Eine direkte ausserlogische Vermittlung im Sinne der Güntherschen Konzeption der Vermittlungsbedingungen kann mit diesem einfachen Faserungskonzept nicht modelliert werden.

9.2.3 Vermittlung der Kontexturen

9.2.3.1 Vermittlung als Quotientenstruktur

Die in der Faserung nicht zur Darstellung gelangenden Vermittlungsbedingungen (VB) der Güntherschen Konzeption morphogrammatischer Verbundkontexturen (vgl. 5.4.3) bewirken auf der logischen Ebene eine Identifizierung bestimmter lokaler Wahrheitswerte über die Subsystemgrenzen hinweg. Im Falle eines $\mathcal{L}^{(3)}$-Systems führen die Vermittlungsbedingungen (durch senkrechte Striche | dargestellt) zwischen den W_i zur folgenden Situation:

[4] [Pfa88].
[5] [Pfa88], S. 1.

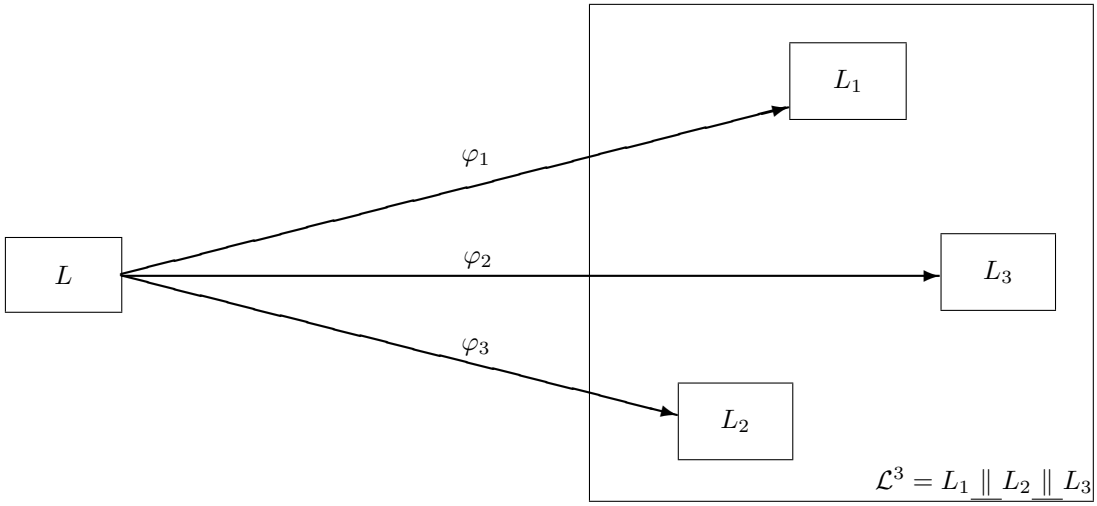


Abbildung 9.1: Distribution der klassischen Logik über mehrere Kontexturen als Faserraum

$$\begin{array}{lccc} W_1 = & \{T_1, & F_1\} & \\ & | & | & \\ W_2 = & | & \{T_2, & F_2\} \\ & | & & | \\ W_3 = & \{T_3, & & F_3\}. \end{array}$$

Die Vermittlung kann also als eine Äquivalenzrelation $\equiv_{VB}$ über der globalen Wertemenge W^n interpretiert werden, mit:

$$\begin{array}{lll} T_1 & \equiv_{VB} & T_3 \\ F_1 & \equiv_{VB} & T_2 \\ F_2 & \equiv_{VB} & F_3. \end{array}$$

Pfalzgraf schlägt vor, die von $\equiv_{VB}$ bestimmten Vermittlungsbedingungen durch die Bildung einer Quotientenmenge $W^n/\equiv_{VB}$ abzubilden.
Hierzu geht er wie folgt vor. Die globale Wertemenge $W^{(m)} = \{1, \ldots, m\}$ leitet er aus $W_1 \underline{\|} \ldots \underline{\|} W_n$ mittels der Identifizierungsrelation $\equiv_{VB}$ als die Quotientenmenge $W^{(m)} = W^n/\equiv_{VB}$ ab, wobei $n = \binom{m}{2}$. Im Falle der dreikontexturalen PKL $\mathcal{L}^3$ führt die kanonische Restklassenabbildung $W^n \longrightarrow W^{(m)}$ zu drei Äquivalenzklassen, welche die globale Wertemenge $W^{(3)} = \{[T_1], [F_1], [F_2]\}$ bilden. Günther bezeichnet diese Äquivalenzklassen als die globale Wertemenge $G = \{1, 2, 3\}$.

9.2.3.2 Vermittlung in der Morphogrammatik

Diese Methode weist deutlich auf die morphogrammatische Fundierung der Vermittlung der logischen Subsysteme hin. Die Quotientenmenge $W^n/\equiv_{VB}$ ist das Ergebnis einer durch die kanonische Abbildung $W^n \longrightarrow W^{(m)}$ vollzogenen Abstraktion von den

lokalen Wahrheitswertstrukturen auf die globale Wertemenge. Eine Abstraktion von der Wertstruktur einer lokalen Logik L_i kann aber weder innerhalb von L_i noch innerhalb der typischen Faser L geschehen, da hiermit die logische Axiomatik der Zweiwertigkeit ausser Kraft gesetzt wäre. In der Morphogrammatik der typischen Logik L wird die Abstraktion von der Wertstruktur logischer Operationen $\circ_j$ $(1 \leq j \leq 16)$ durch die kanonische Tritonormalformabbildung $TNF : \circ \longrightarrow \circ/\text{Kern} \circ$ (vgl. 3.3) geleistet. Dieser Abstraktionsvorgang liefert für die Operatoren der Aussagenlogik die in Abbildung 5.3 auf Seite 86 dargestellten Abstraktionen. Diese TNF-Abstraktion von der typischen Logik L auf die typische Morphogrammatik M ist in Diagramm 9.2 schematisch dargestellt.

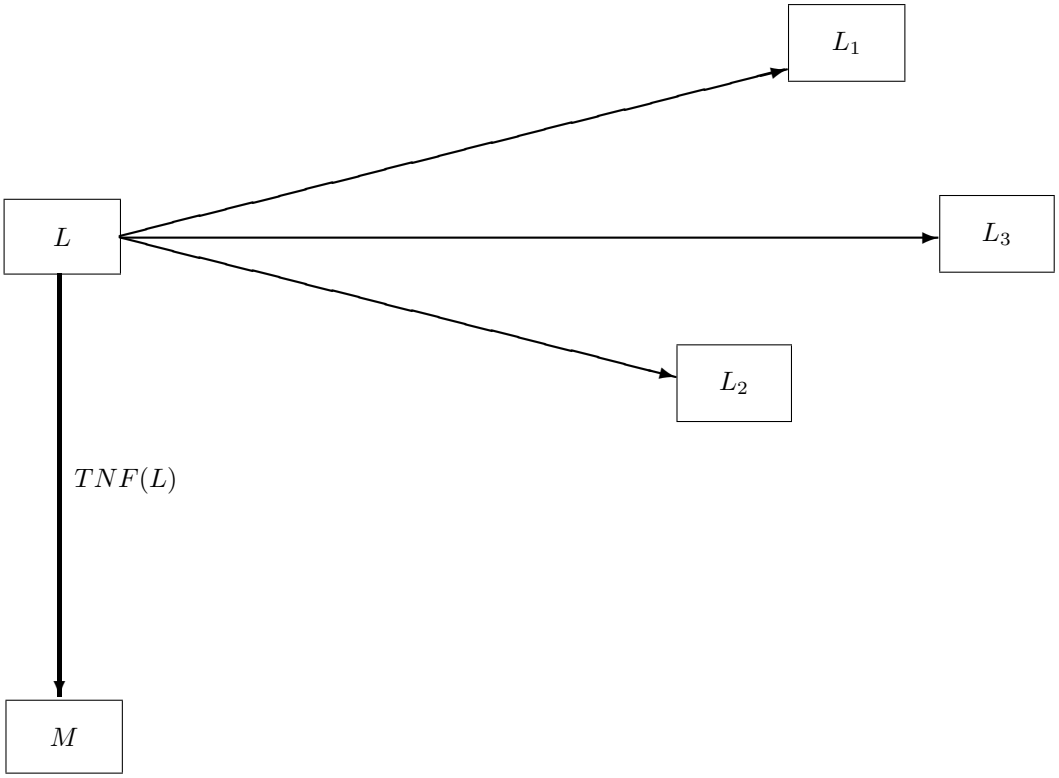


Abbildung 9.2: Morphogrammatische Abstraktion der typischen Logik L

Die in Kapitel 5 entwickelte Morphogrammatik beschreibt nun die Erzeugung komplexer Verbundstrukturen aus den Basismorphogrammen (TNF-Abstraktionen logischer Operatoren). Hierbei werden die Vermittlungsbedingungen zwischen den Subsystemen als Indizierung der Komposition von Morphogrammketten, $\texttt{Kom}[\texttt{M}_1, \ldots, \texttt{M}_n]$, angegeben (vgl. 5.3 und 5.4.3). Da die Morphogrammatik vollständig von der logischen Wertigkeit abstrahiert, kann sie die logisch kontradiktorische Vermittlung (d.h. Identifizierung) bestimmter lokaler Wahrheitswerte durch die Vermittlungsrelation $\equiv_{VB}$ antinomienfrei modellieren. Diese Erzeugung vermittelter Verbundstrukturen ist in Abbildung

9.3 dargestellt.

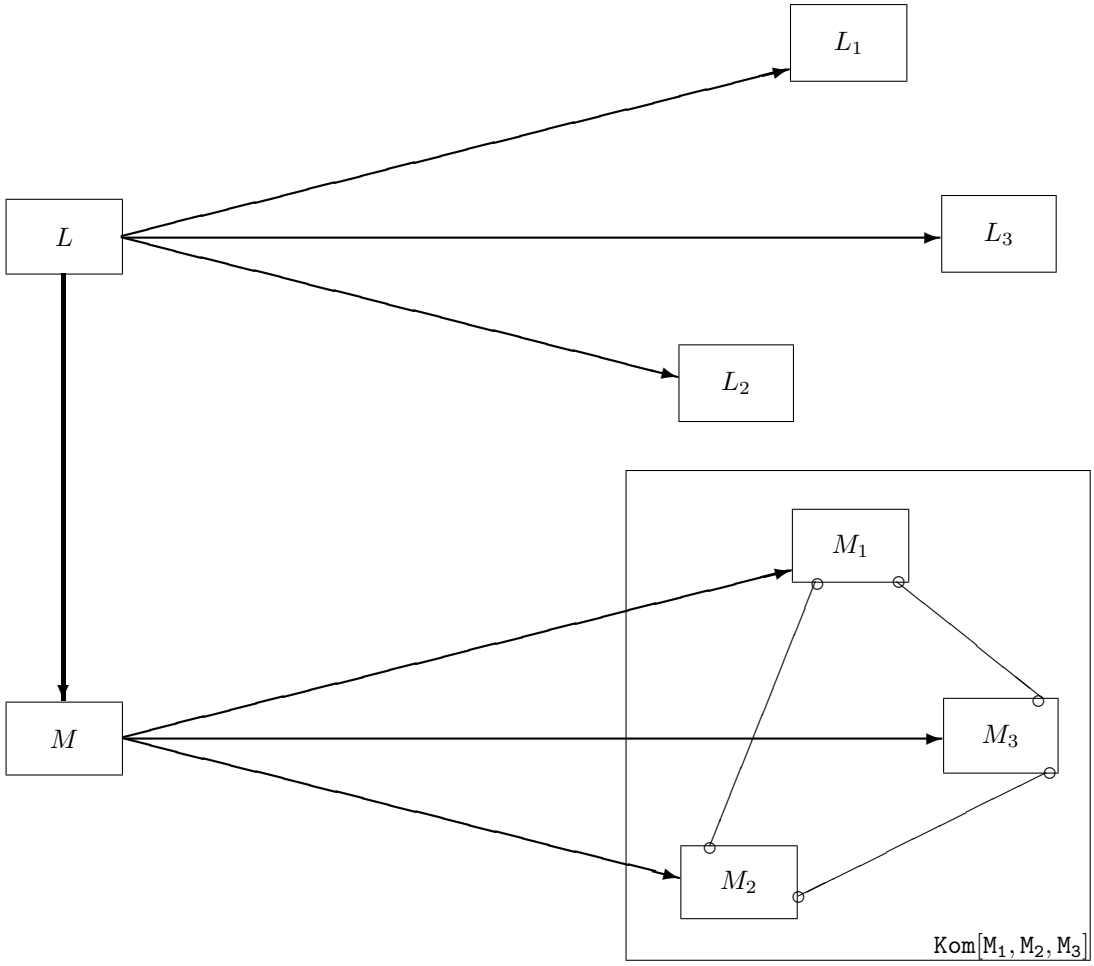


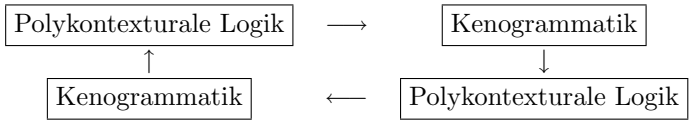
Abbildung 9.3: Morphogrammatische Komposition von Verbundstrukturen. ○—○ stellt dabei die Vermittlung der Subsysteme gemäß $\equiv_{VB}$ dar.

Die komponierten morphogrammatischen Komplexionen können mit logischen Operationen und Wertstrukturen (d.h mit lokalen logischen Systemen) belegt werden, die durch die morphogrammatische Vermittlung miteinander verkoppelt sind. Die Distribution dieser lokalen Logiken geschieht dabei, wie im vorherigen Abschnitt erläutert, über die Faserung der typischen Logik L. Dies führt zu Abbildung 9.4, in der die Wertstrukturbelegung der morphogrammatischen Struktur durch WB symbolisiert ist.

Die innerhalb der Morphogrammatik realisierte Vermittlung der Subsysteme wird durch die von Pfalzgraf angegebene Quotientenstruktur modelliert.

9.2.4 Kenogrammatische Fundierung

Das im Vorhergehenden entwickelte Fundierungsschema der Polykontexturalen Logik beschreibt das proemielle Verhältnis von Distribution und Vermittlung logischer Systeme, deren formale Erfassung eine notwendige Voraussetzung der Formalisierung der PKL ist.
In dem bis hierhin entwickelten Schema ist jedoch ein weiterer zentraler Aspekt polykontexturaler Operativität im Sinne der Güntherschen Theorie noch nicht formulierbar. Da die Tritonormalformbildung aussagenlogischer Operationen lediglich die Menge der acht *klassischen* Morphogramme bildet, und die Komposition und Vermittlung von Morphogrammketten die Struktur der Subsystemmorphogramme bewahrt, sind innerhalb des dargestellten Prozesses *transklassische* Morphogramme — und somit *Transjunktionen* im Bereich der PKL — noch nicht erzeugbar. Die Morphogramme bilden das Operandensystem, Q, komplexer Operationen. Hinsichtlich dieser Operationen bilden Morphogramme elementare Einheiten, deren interne Struktur stets bewahrt wird. Die interne ϵ/ν-Struktur von Gleichheiten und Differenzen wird erst in der Kenogrammatik untersucht und zum Gegenstand kenogrammatischer Operationen. Die Kenogrammatik erfasst die ϵ/ν-Struktur K der Morphogramme. Kenogrammatische Operationen erzeugen und manipulieren beliebige Komplexionen und können somit auch transklassische Morphogramme erzeugen und transjunktive Operationen auf der Kenogrammkomplexion der Verbundstruktur ausführen (vgl. Kapitel 3 und 5.4.4). Diese transklassische Operativität innerhalb der Verbundstruktur ist in der folgenden Abbildung durch die Doppelpfeile ($\longleftrightarrow$) zwischen den lokalen Strukturen symbolisiert.
Die transjunktiven Eigenschaften werden von der kenogrammatisch fundierten Morphogrammatik übernommen und damit auch in die Wertstruktur des belegenden Polykontexturalen Systems vererbt. Dort wirken sie sich als Transjunktionen aus. Mit der Einführung der kenogrammatischen Fundierung ist das Konstruktionsschema der PKL vervollständigt (Abbildung 9.5).
Mit dieser knappen Skizze des Konstruktionsvorganges der Polykontexturalen Logik ist gleichzeitig eine Lokalisierung der Morphogrammatik im Schnittpunkt von Kenogrammatik und Polykontexturaler Logik erreicht.
Die Morphogrammatik beschreibt die Vermittlung komponierter Verbundstrukturen, die durch die Belegung mit einer PKL-Wertstruktur logisch interpretiert wird. Das Notationssystem der Morphogrammatik wird durch die Kenogrammatik realisiert, deren transklassischen Strukturen sich auf der logischen Ebene als Transjunktionen auswirken. Diese Darstellung rechtfertigt somit auch die vom Aufbau dieser Arbeit suggerierte, hierarchische Abfolge 'Kenogrammatik — Morphogrammatik — Polykontexturale Logik' als eine Konstruktionsmethode zur Formalisierung der PKL.
In 3.2 wurde darauf hingewiesen, daß eine vollständige Formalisierung der Kenogrammatik nur innerhalb eines transklassischen Kalküls möglich ist (vgl. Abbildung 3.6 auf Seite 40). Einerseits wird die Kenogrammatik zur Fundierung transklassischer Kalküle — wie der PKL — benötigt, andererseits ist eine vollständige Formalisierung der Kenogrammatik erst in einem transklassischen Kalkül möglich. Die Zirkularität dieser Konstruktion ist in dem folgenden Schema veranschaulicht (die Pfeile stehen hier für die Fundierungsrelation):



Diese simultane und wechselseitige Fundierung von Kenogrammatik und PKl, die Kaehr als Proemialverhältnis charakterisiert, macht die fundamentale Selbstreferenz der PKL-Architektur aus.
Hier zeigt sich, daß der hierarchische Aufbau Kenogrammatik — Morphogrammatik — PKL zwar als eine Konstruktionstrategie nützlich ist, jedoch der eigentlichen selbstreferentiellen Architektur der PKL nicht gerecht wird. Denn diese Konstruktion erfasst zwar die Fundierung der PKL durch Morphogrammatik und Kenogrammatik, kann jedoch nicht gleichzeitig die Fundierung der Kenogrammatik in der transklassischen PKL thematisieren, da ein solcher zirkulärer Aufbau gegen das klassische Induktionsprinzip verstößt.
Aufgrund der fundierenden Rolle der Proemialrelation für den Aufbau des gesamten Systems der Polykontexturalen Logik wird im nächsten Kapitel ein Modellierungsansatz für die Proemialrelation vorgestellt. Ein solches Modell kann zur Exploration der noch offenen Fragen der Polykontexturalitätstheorie verwendet werden.

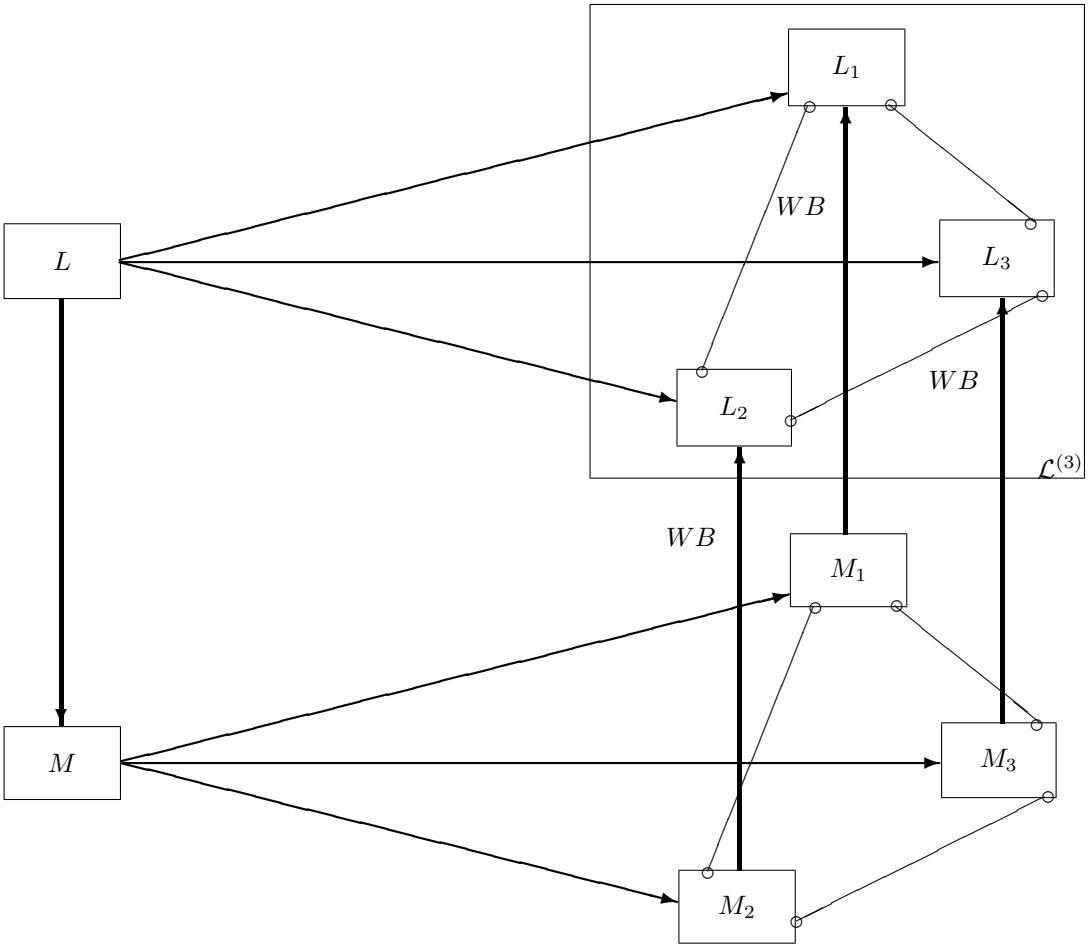


Abbildung 9.4: Die morphogrammatisch fundierte Distribution und Vermittlung klassischer Logiken in $\mathcal{L}^{(3)}$

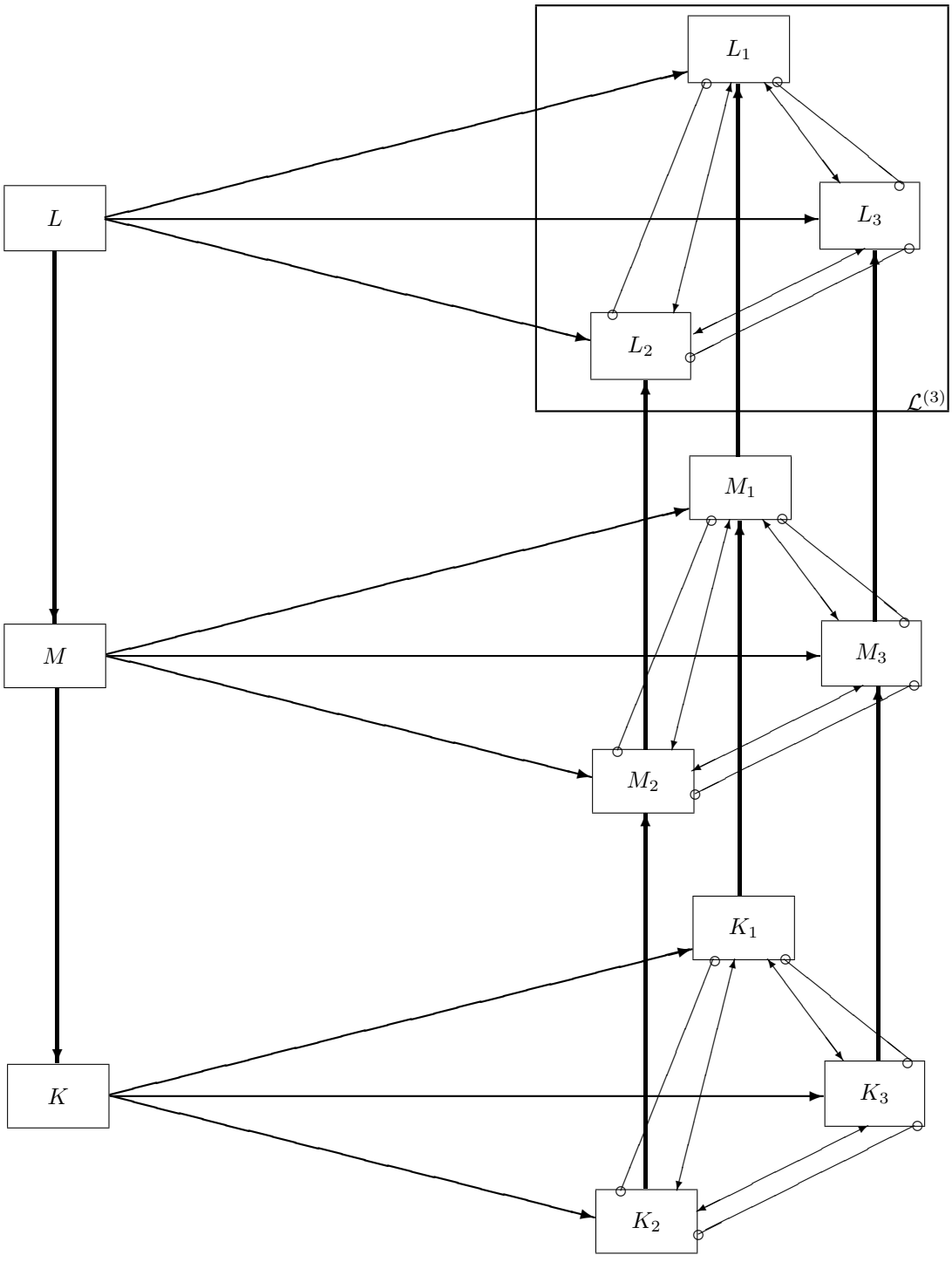


Abbildung 9.5: Das kenogrammatisch fundierte Konstruktionsschema der Polykontexturalen Logik $\mathcal{L}^{(3)}$. Vollständige Integration der Vermittlungsbedingungen und der Transjunktivität.

Kapitel 10

Modellierung der Proemialrelation

Das Tao, das genannt werden kann, ist nicht das wahre Tao.
Lao Tse

10.1 Einführung

Die Formalisierung der Kenogrammatik und Morphogrammatik in Teil II dieser Arbeit folgte in Argumentation und formalem Aufbau den aus der Literatur geläufigen Formalisierungsansätzen. Alle bisherigen Formalisierungen der Güntherschen Theorien erfolgten weitgehend in Anlehnung und Abgrenzung an klassische Formalisierungen der Semiotik mittels arithmetischer und algebraischer Methoden. Im Rahmen solcher Darstellungen lassen sich statische Eigenschaften kenogrammatischer Objekte und Operationen, wie etwa die Verkettungsregeln und die ϵ/ν-Strukturen abbilden.

Die von Günther ebenfalls in der Kenogrammatik lokalisierte Proemialrelation läßt sich jedoch, wie in 3.1.2 gezeigt, nicht innerhalb einer algebraischen Darstellung abbilden. In diesem Kapitel soll eine Modellierung der dynamischen Aspekte der Kenogrammatik und insbesondere der von der Proemialrelation beschriebenen kenogrammatischen Relationalität formal erarbeitet werden.

Im Gegensatz zur Charakterisierung der statischen Eigenschaften der Kenogrammatik, die in Analogie zu klassischen Darstellungen der Semiotik durchgeführt wurde, geschieht die Formalisierung der kenogrammatischen Dynamik in Analogie zur klassischen Berechenbarkeitstheorie. Als berechnungsuniverseller Kalkül wird dabei der λ-Kalkül zugrundegelegt.

Die Implementierung des λ-Kalküls wird mittels einer parallelisierten Graphreduktionsmaschine vorgenommen. Repräsentationen der λ-Ausdrücke werden von einer solchen abstrakten Maschine entsprechend der *normal-order-reduction* zu ihren Normalformen reduziert.

Durch eine Modifikation und Erweiterung der abstrakten Maschine wird zunächst die Proemialrelation operational modelliert. Die Bedeutung dieser Erweiterung für die Semantik des λ-Kalküls wird diskutiert und zum Begriff der ϵ/ν-Struktur paralleler Berechnungen erweitert. Verschiedene Anwendungsmöglichkeiten dieser Modellierungstechnik werden skizziert.

10.2 Der λ-Kalkül

Turing-Maschinen, rekursive Funktionen und Register Maschinen sind formale Modelle der Berechenbarkeit. Der von Alonzo Church [Chu51] entwickelte λ-Kalkül ist eine der frühesten Theorien der berechenbaren Funktionen und bildet heute die formale Grundlage der meisten funktionalen Programmiersprachen und ihrer Implementierungen.
Im Gegensatz zur naiven Mengenlehre, die in einer eher statischen Sichtweise Funktionen als spezielle Teilmengen kartesischer Produkte auffaßt, interpretiert der λ-Kalkül Funktionen als Berechnungsvorschriften und betont so die dynamischen Aspekte einer von den Argumenten der Funktion ausgehenden Berechnung der Funktionswerte.
Der λ-Kalkül ist eine einfache Funktionentheorie. Die einzigen Objekte des λ-Kalküls sind Funktionen. Auch die Argumente von Funktionen sind daher notwendigerweise wieder Funktionen. Insbesondere dürfen Funktionen auf sich selbst angewandt werden, was in der Mengenlehre nicht möglich ist.

10.2.1 λ-Terme

λ-Terme werden aus Variablen und anderen λ-Termen rekursiv aufgebaut.

Definition 10.1 (λ-Term) *Sei $\mathcal{V}$ eine abzählbare Menge von Variablensymbolen. Dann gilt:*

1. *Jede Variable $x \in \mathcal{V}$ ist ein λ-Term.*
2. *Wenn t ein λ-Term und $x \in \mathcal{V}$ ist, dann ist auch $(\lambda x.t)$ ein λ-Term (Abstraktion).*
3. *Sind t und u λ-Terme, dann ist auch $(t\ u)$ ein λ-Term (Applikation).*

Eine Funktionsabstraktion $(\lambda x.t)$ ist das Modell einer *anonymen Funktion*, x ist die *gebundene Variable* und t der *Rumpf* der Funktion. Jedes Vorkommen von x in t ist durch die Abstraktion *gebunden*. Entsprechend ist das Vorkommen einer Variablen y *frei*, wenn es nicht innerhalb des Rumpfes u einer Abstraktion $(\lambda y.u)$ geschieht. In dem Term $(\lambda z.(\lambda x.(y\ x)))$ ist beispielsweise x gebunden und y frei.
Eine Funktionsapplikation $(t\ u)$ modelliert die Anwendung der Funktion t auf ein Argument u.
λ-Terme werden in der folgenden Implementierung als rekursiver Datentyp `term` repräsentiert:

```
datatype term = Free  of string
              | Bound of string
              | Abs   of string*term
              | Apply of term*term;
```

Der reine λ-Kalkül enthält keine Konstanten. Zahlen und andere in Programmiersprachen gebräuchliche Konstanten können durch bestimmte λ-Terme modelliert werden. Obwohl verzichtbar werden jedoch im folgenden aus Gründen der Effizienz und Übersichtlichkeit auch Konstanten als Terme zugelassen. Der Ausdruck (+ 1 2) ist dann ein λ-Term, weil er eine Applikation der konstanten Operation + auf die Konstanten 1 und 2 notiert. Diese Erweiterung führt zu folgender ML Definition der λ-Terme:

```
datatype term = Free  of string
              | Bound of string
              | Int of int
              | Op of string
              | Abs   of string*term
              | Apply of term*term;
```

In der weiteren Darstellung werden die gebräuchlichen Konventionen zur Darstellung von λ-Termen benutzt: $x, y, t, \ldots$ sind Variablensymbole. Klammern werden nach Möglichkeit fortgelassen. So wird:

$$(\lambda x.(\lambda y.(\ldots(\lambda z.(E))\ldots)))$$

als $\lambda xyz.E$ geschrieben. Der Ausdruck:

$$(\ldots((E_1E_2)E_3)\ldots E_n)$$

wird abkürzend notiert als $E_1E_2E_3\ldots E_n$.

10.2.2 λ-Konversionen

λ-Konversionen sind Regeln zur symbolischen Transformation, nach denen λ-Terme unter Bewahrung ihrer intuitiven Bedeutung umgeformt werden können.
Die α-Konversion benennt die gebundene Variable einer Abstraktion um:

$$(\lambda x.t) \Longrightarrow_\alpha (\lambda y.t_{[y/x]}).$$

Die Abstraktion über x wird in eine Abstraktion über y umgeformt und jedes Vorkommen von x in t durch y ersetzt. Zwei λ-Terme sind äquivalent, wenn sie durch α-Konversionen in identische Terme umgeformt werden können.
Für das Verständnis funktionaler Programmiersprachen am wichtigsten ist die β-Konversion, die eine Funktionsapplikation umformt, indem sie die gebundene Variable durch das Funktionsargument substituiert:

$$((\lambda x.t)u) \Longrightarrow_\beta t_{[u/x]}.$$

Die β-Konversion entspricht der Substitutionsregel für lokale Variablen in Programmiersprachen.

10.2.3 λ-Reduktion und Normalformen

Ein Reduktionsschritt $t \Longrightarrow u$ formt den λ-Term t in den Term u durch Anwendung einer β-Konversion auf einen beliebigen Unterterm von t um. Wenn ein λ-Term keine weiteren Reduktionen mehr zuläßt, ist er in *Normalform*. Einen Term zu *normalisieren* heißt also, so lange β-Konversionen auf ihn anzuwenden, bis eine Normalform erreicht ist.
Das **erste Church–Rosser Theorem** [Bar80] besagt, daß die Normalformal eines λ-Terms, falls sie existiert, eindeutig ist. Mit anderen Worten: verschiedene Folgen von Reduktionen, die von einem bestimmten λ-Term ausgehen, müssen äquivalente Normalformen erreichen.
Viele λ-Terme besitzen keine Normalform. DD mit $D := (\lambda x.xx)$ ist ein Beispiel für einen solchen Term, da $DD \Longrightarrow_\beta DD$. Ein Term kann eine Normalform besitzen,

auch wenn bestimmte Reduktionsfolgen nicht terminieren. Typisches Beispiel ist etwa das Auftreten eines Subtermes u, der keine Normalform besitzt, der jedoch durch bestimmte Reduktionen eliminiert werden kann.

Beispiel: Die Reduktion

$$\underline{(\lambda x.a)(DD)} \Longrightarrow a$$

erreicht sofort eine Normalform und eliminiert den Term (DD). Dies entspricht dem *Call-by-name* Verhalten von Prozeduren: das Argument wird *nicht* ausgewertet, sondern einfach in den Prozedurrumpf hineinsubstituiert. Wird das Argument hingegen ausgewertet, *bevor* es in den Rumpf eingesetzt wird, entspricht dies der *Call-by-value* Technik. Im obigen Beispiel würde diese Auswertungsstrategie keine Normalform erreichen:

$$(\lambda x.a)\underline{(DD)} \Longrightarrow (\lambda x.a)\underline{(DD)} \Longrightarrow \ldots.$$

Wird wie im ersten Fall immer der am weitesten links befindliche Redex reduziert, so wird immer eine Normalform erreicht, sofern eine solche existiert. Eine solche Reduktion ist dann in *Normalordnung* (normal-order) . Besitzt ein Term t eine Normalform u, dann gibt es eine Reduktion in Normalordnung von t nach u (**zweites Church–Rosser Theorem** [Bar80]). Die normal-order Reduktion findet daher jede existierende Normalform.

10.3 Implementierung funktionaler Sprachen

Der im vorhergehenden Abschnitt eingeführte λ-Kalkül ist berechnungsuniversell [Bar80] und kann als Termtransformationssystem (*Stringreduktion*) implementiert werden. Ein solches System ist jedoch extrem ineffizient und zur praktischen Anwendung als Programmiersprache kaum anwendbar [Pau91]. Tatsächliche Implementierungen funktionaler Sprachen bedienen sich häufig abstrakter Maschinen[1]. Der hier gewählte Implementierungsansatz bedient sich einer *Graphreduktionsmaschine*.
Die Auswahl dieser modernen Implementierungstechnik soll kurz begründet werden. Die Graphreduktion ermöglicht die Reduktion von Ausdrücken in Normalordnung (alle existierenden Normalformen werden gefunden). Im Gegensatz zur Call-by-name Evaluation der Termersetzungsmethode müssen Argumente von Funktionen jedoch nicht *textuell* in die Funktionsrümpfe kopiert werden (was zu erheblichen Raum und Zeitaufwand führt), sondern können als verzeigerte Objekte gehandhabt werden. Mehrere Instanzen einer Variablen verweisen so auf ein einziges physikalisches Objekt (*node-sharing*). Diese Technik reduziert durch die Verzeigerung den Speicheraufwand und, da ein bestimmter Knoten nur ein einziges Mal evaluiert werden muß, auch den Zeitaufwand einer Berechnung.
Die Graphreduktion eignet sich desweiteren zu einer einfachen nund anschaulichen Implementierung auf Mehrprozess(or)-Systemen. Was ein weiteres Hauptargument für ihre berlegenheit zu klassischen von-Neumann Architekturen ausmacht[2].
Eine Graphreduktion, die direkt auf λ-Ausdrücken operiert, erfordert erheblichen Kopieraufwand, selbst wenn voller Gebrauch von der Node-sharing Technik gemacht

[1] Abelson und Sussman [Abe87] behandeln die Implementierung von LISP auf einer abstrakten Registermaschine. Die wohl bekannteste abstrakte Maschine zur Implentierung funktionaler Sprachen ist die SECD-Maschine [Lan64], [Hen80], [Dav92].

[2] [Dav92], S.1 ff.

wird. Dieser hohe Resourcenaufwand ist auf das Vorhandensein von Variablen zurückzuführen, wobei jede Variablenbindung eine komplette Kopie des Funktionsrumpfes erforderlich macht.[3] Aus diesem Grund werden in der hier gewählten Implementierung λ-Terme zunächst in variablenfreie *Kombinatorausdrücke* übersetzt, für die effiziente Graphreduktionsmethoden existieren.
Die nun folgende Darstellung dieses Verfahrens orientiert sich an den Arbeiten Turners [Tur79a] und Davies [Dav92].

10.3.1 Übersetzung von λ-Ausdrücken

10.3.1.1 Kombinatoren

Funktionen wie etwa:

$$S\ x\ y\ z = (x\ z)(y\ z), \tag{10.1}$$
$$K\ x\ y = x, \tag{10.2}$$
$$I\ x = x \tag{10.3}$$

enthalten keine freien Variablen. Solche Funktionen werden *Kombinatoren* genannt. Der Kombinator S verteilt sein drittes Argument an die beiden ersten. K eliminiert sein zweites Argument und I bildet die Identitätsfunktion. Ausdrücke des λ-Kalküls können mittels einfacher Termtransformationen in Kombinatorausdrücke umgeformt werden, wobei alle Variablen der λ-Terme eliminiert werden. Diese Kombinatorausdrücke können von einer Kombinatormaschine, die die Definitionsgleichungen der Kombinatoren als Reduktionsvorschriften benutzt, evaluiert werden.

10.3.1.2 Variablen–Abstraktion

Im λ-Kalkül werden Funktionen notiert als:

$$\lambda x.t$$

Die *Abstraktion* der Variablen x im Ausdruck t wird notiert als:

$$[x]t.$$

Der resultierende Ausdruck enthält keine Vorkommen von x, besitzt jedoch die gleiche Normalform wie der ursprüngliche Ausdruck. Im Gegensatz zu einer Run-time Bindung eines Wertes an die Variable x, stellt $[x]$ eine Compile-time Transformation dar. Diese Abstraktion verläuft nach folgenden Regeln[4]:

$$[x]x = I, \tag{10.4}$$
$$[x]y = K\ y, \tag{10.5}$$
$$[x](e_1, e_2) = S\ ([x]e_1)([x]e_2). \tag{10.6}$$

Wobei y entweder eine von x verschiedene Variable oder aber eine Konstante ist. Implementiert wird $[x]$ von der ML-Funktion `abs x`, die auf λ-Termen operiert:

[3] [Dav92], S. 154.
[4] [Dav92], S.155.

```
exception Doublebind
fun abs x (Free y) = Apply(Op "K", (Free y))
   |abs x (Bound y) = if y=x then (Op "I") else Apply(Op "K",(Bound y))
   |abs x (Int y) = Apply(Op "K",(Int y))
   |abs x (Op y) = Apply("K",(Op y))
   |abs x (Abs(y,body)) = abs x (abs y body)
   |abs x (Apply(a,b)) =
         Apply(Apply(Op "S",abs x a),
                    (abs x b));
```

10.3.1.3 Compilierung

Der Datentyp `snode` implementiert eine binäre Kombinatorgraphendarstellung:

```
datatype snode = satom of value
               |scomb of comb
               |sapp of (snode*snode);
```

Die Übersetzung von λ-Termen nach Kombinatortermen geschieht nach den folgenden Regeln[5]:

$$c(x) = x, \tag{10.7}$$

$$c(t\ u) = c(t)\ c(u), \tag{10.8}$$

$$c(\lambda x.u) = c([x]u). \tag{10.9}$$

Implementiert wird c durch die ML–Funktion `c`:

```
exception Compile;
fun c (Free a) = scomb(DEF a)
   |c (Bound a) = raise Compile
   |c (Int a) = sapp(scomb K,satom(int a))
   |c (Op k) = sapp(scomb K,scomb(mkComb k))
   |c (Apply(a,b)) = sapp(c a,c b)
   |c (Abs(x,body)) = c (abs x body);

fun mkComb "I" = I
   |mkComb "K" = K
   |mkComb "S" = S
   |mkComb "B" = B
   |mkComb "C" = C
   |mkComb "Y" = Y
   |mkComb "CONS" = CONS
   |mkComb "HD" = HD
   |mkComb "TL" = TL
   |mkComb "+" = PLUS
   |mkComb "-" = MINUS
   |mkComb "*" = TIMES
   |mkComb "/" = DIV
   |mkComb "IF" = IF
```

[5] [Dil88], S.88.

```
   |mkComb "EQ" = EQ
   |mkComb "AND" = AND
   |mkComb "OR" = OR
   |mkComb "NOT" = NOT
   |mkComb "PR" = PR
   |mkComb str = DEF str;
```

Beispiel: Der folgende λ-Term:

$$(\lambda x\ y.ADD\ x\ y)\ 3\ 4$$

soll übersetzt werden.

Die im Anhang beschriebene Funktion `r` liest λ-Ausdrücke im Form von Strings ein und konstruiert `term`-Ausdrücke aus ihnen[6]:

```
> r "(%x y.ADD x y) 3 4";
-
> c it;
- val it =
```

Da selbst so kurze Definitionen zu großen Kombinatorausdrücken führen, schlägt Turner eine Optimierungsfunktion *opt* vor, die auf der Einführung zweier weiterer Kombinatoren B und C beruht:

$$B\ x\ y\ z \;=\; x(y\ z), \tag{10.10}$$
$$C\ x\ y\ z \;=\; x\ z\ y. \tag{10.11}$$

Die Optimierungsregeln:

$$opt(S\ (K\ e_1)(K\ e_2)) \;=\; K(e_1\ e_2), \tag{10.12}$$
$$opt(S\ (K\ e)\ I) \;=\; e, \tag{10.13}$$
$$opt(S\ (K\ e_1)\ e_2) \;=\; B\ e_1\ e_2, \tag{10.14}$$
$$opt(S\ e_1\ (K\ e_2)) \;=\; C\ e_1\ e_2, \tag{10.15}$$

werden von der ML-Funktion `ropt` solange auf die Resultate der Compilierung angewandt, bis alle möglichen Optimierungen `opt` durchgeführt wurden:

```
fun opt (sapp(sapp(scomb S,sapp(scomb K,e)),scomb I)) = (e : snode)
   |opt (sapp(sapp(scomb S,sapp(scomb K,e1)),sapp(scomb K,e2))) =
       sapp(scomb K,sapp(e1,e2))
   |opt (sapp(sapp(scomb S,sapp(scomb K,e1)),e2)) =
       sapp(sapp(scomb B,e1),e2)
   |opt (sapp(sapp(scomb S,e1),sapp(scomb K,e2))) =
       sapp(sapp(scomb C,e1),e2)
   |opt (x : snode) = x;

fun ropt x =
 let
  val y = opt x;
```

[6]Der Parsemechanismus, der sich hinter `r` verbirgt, wurde [Pau91], S. 315ff. entnommen

```
 in
   if y=x then x
   else ropt y
 end;

> ropt(it);
-
```

10.3.2 Reduktion von Kombinatorausdrücken

Um Kombinatorausdrücke auszuwerten, müssen die definierenden Gleichungen der Kombinatoren als Reduktionsregeln benutzt werden. Die Auswertung muß zudem in Normalordnung erfolgen, um das Auffinden aller existierender Normalformen zu gewährleisten. Außer den Kombinatoren S, K, I, B und C müssen auch noch arithmetische und andere grundlegende Operationen definiert werden. Die Reduktionsregeln für einige dieser Operationen lauten:

$$PLUS\ x\ y = (\texttt{eval}\ x) + (\texttt{eval}\ y) \tag{10.16}$$
$$MINUS\ x\ y = (\texttt{eval}\ x) - (\texttt{eval}\ y) \tag{10.17}$$
$$TIMES\ x\ y = (\texttt{eval}\ x) * (\texttt{eval}\ y) \tag{10.18}$$
$$DIV\ x\ y = (\texttt{eval}\ x)/(\texttt{eval}\ y) \tag{10.19}$$
$$IF\ \texttt{true}\ x\ y = x \tag{10.20}$$
$$IF\ \texttt{false}\ x\ y = y \tag{10.21}$$
$$IF\ exp\ x\ y = \texttt{if (eval}\ exp) = \texttt{true then}\ x\ \texttt{else}\ y \tag{10.22}$$
$$EQ\ x\ y = \texttt{if (eval}\ x) = (\texttt{eval}\ y)\ \texttt{then true else false} \tag{10.23}$$
$$CONS\ x\ y = x :: y \tag{10.24}$$
$$\vdots$$

Wobei `(eval node)` die Normalform von `node` bezeichnet. Die zugrundeliegende Datenstruktur der Kombinatormaschine sind binäre Bäume. Ein Knoten eines Baumes kann entweder atomar sein, oder aber eine Applikation (@) zweier Knoten darstellen. Der Kombinatorausdruck $S\ x\ y\ z$ wird durch folgende Datenstruktur repräsentiert:

```
sappl(sappl(sappl(scomb S, x),
            y),
      z);
```

Diese Struktur ist in Abbildung (10.1) als Graph notiert.
Die Reduktion dieses Ausdrucks anhand der Definition $S\ x\ y\ z = (x\ z)(y\ z)$ ist in Abbildung (10.2) dargestellt.
Um die Reduktion der Kombinatorgraphen in Normalordnung durchzuführen, kann folgende Technik angewandt werden. Ausgehend vom Wurzelknoten wird durch Herabsteigen zu den linken Nachfolgern der äußerste linke Kombinator ermittelt. Ist dieser Kombinator *gesättigt*, d.h. sind alle benötigten Argumente vorhanden, wird eine Reduktion dieses Kombinators (entsprechend der oben beschriebenen Methode) ausgeführt. Dieser Ablauf wird so lange wiederholt, bis keine weiteren Reduktionen mehr möglich sind, der Graph also in Normalform vorliegt. Der verbleibende Ausdruck ist das Resultat der Berechnung.

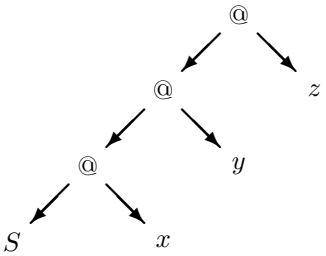


Abbildung 10.1: Der Ausdruck $S\ x\ y\ z$ als Kombinatorgraph

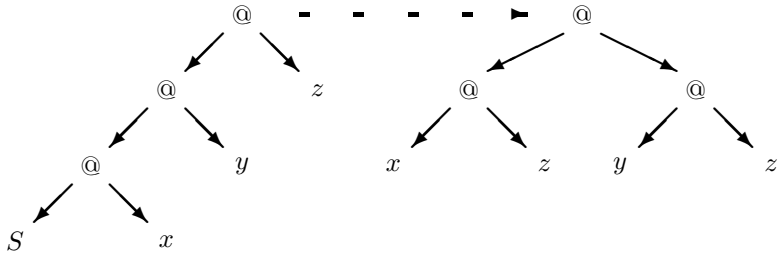


Abbildung 10.2: Die Auswertung von $S\ x\ y\ z$ als Graphreduktion

Beispiel: Dem ML-Ausdruck

```
let
   fun succ x = 1+x
in
   succ 2
end;
```

entspricht der λ-Ausdruck $(\lambda x.(+\ 1\ x))\ 2$, der zu $C\ I\ 2\ (PLUS\ 1)$ compiliert werden kann. Dieser Kombinatorausdruck wird von der Kombinatormaschine wie folgt reduziert (Abbildung 10.3):

10.3.3 Implementierung der Kombinator–Maschine

Im Vorhergehenden wurde angedeutet, daß die Transformationen der Kombinatorausdrücke von der Kombinatormaschine nicht auf ihrer abstrakten Termstruktur sondern auf einer verzeigerten Datenstruktur durchgeführt wird, um alle Effizienzvorteile der Graphreduktion auszuschöpfen.
So repräsentieren die breiten gestrichelten Pfeile in den beiden vorhergehenden Abbildungen nicht nur die Termumformung des Kombinatorausdrucks, sondern auch, daß der Wurzelknoten des jeweiligen Redexes durch den resultierenden Ausdruck *überschrieben* wird[7]. Im Fall der Reduktion von $S\ x\ y\ z$ bleibt also der oberste Knoten

[7]Bei den beiden Knoten rechts und links neben einem solchen gestrichelten Pfeil handelt es sich also um den selben physikalischen Knoten.

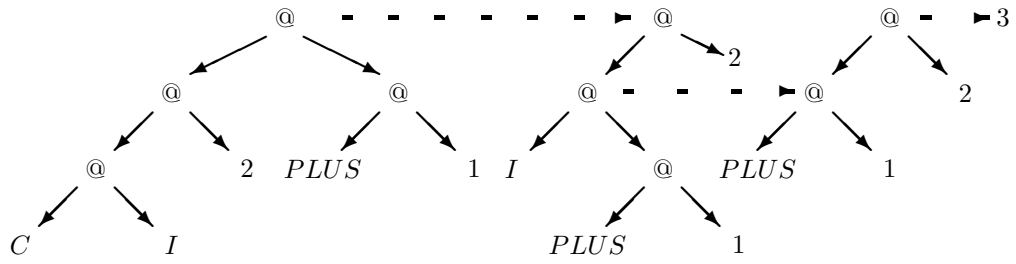


Abbildung 10.3: Die Graphreduktion von $C\ I\ 2\ (PLUS\ 1)$

erhalten und erhält neue Verweise auf die dynamisch generierten Repräsentationen von $(x\ z)$ und $(y\ z)$. x, y und z selbst brauchen jedoch nicht neu kopiert zu werden, da sie als verzeigerte Objekte vorliegen. Die beiden in einer Stringreduktion textuell verschiedenen Vorkommen von z sind somit physikalisch an einer Speicheraddresse lokalisiert, so daß z, falls es im im weiteren Verlauf der Auswertung benötigt wird, nur einmal berechnet wird.
Eine verzeigerte Repräsentierung der Kombinatorgraphen ist durch die Definition des Typs `node` gegeben:

```
datatype Emark = Eval|Busy|Ready;

datatype node = atom of (value * Emark ref * (node ref list) ref)
              |comb of (comb  * Emark ref * (node ref list) ref)
              |app of  ((node ref * node ref) * Emark ref *
                                          (node ref list) ref);
```

Die `(Emark ref)`-Verweise der Knoten enthalten spezielle Markierungen, die erst für die Einführung einer parallelisierten Evaluation benötigt werden und deren Erläuterung weiter unten geschieht.
Mittels der Funktion `alloc` werden abstrakte Termdarstellungen der Kombinatorausdrücke vom Typ `snode` in verzeigerte Objekte umgewandelt und so im Speicher alloziert:

```
fun alloc x =
 let
   fun allocate (satom val) = atom(val,ref Ready, ref [])
      |allocate (scomb com) = comb(com,ref Ready, ref [])
      |allocate (sapp(a,b)) = app((ref (allocate a),ref (allocate b)),
                                  ref Eval, ref [])
 in
   ref(allocate x)
 end;
```

Die oben beschriebene Reduktion in Normalordnung kann mit einem *left-ancestors-stack* (Stack der linken Nachfolge-Knoten) realisiert werden. Der Stack enthält zunächst nur einen Eintrag, einen Zeiger zum auszuwertenden Knoten. So lange, wie

der oberste Knoten eine Applikation `app((l,r),_,_)` ist, wird ein neuer Knoten `l` auf den Stack geschoben. Auf diese Weise wird eine Kette von Knoten zum linken *spine* des obersten Knotens erzeugt. Die Funktion `spine node` ermittelt den Spine des Knotens `node`, sowie den Stack der linken Nachfolger von `node`:

```
fun spine (ref(atom(a,m,q))) stack = (atom(a,m,q),stack)
   |spine (ref(comb(c,m,q))) stack = (comb(c,m,q),stack)
   |spine (node as (ref(app((l,r),_,_)))) stack =
         spine l (node::stack);
```

Der Spine des Knotens `node` enthält den äußersten anwendbaren Kombinator, der nun direkt auf seine auf dem Stack gesammelten Argumente angewendet werden kann. Diese Anwendung des Kombinators `k` auf die in `stack` gesammelten Argumente geschieht mittels der Funktion `apply (k,stack)`:

```
fun apply (I,(node as ref(app((_,ref x),_,_)))::_) =
        node := x
   |apply (K,ref(app((_,ref x),_,_))::node::_) =
        node := x
   |apply (S,(ref(app((_,x),_,_)))::(ref(app((_,y),_,_)))
              ::(node as (ref(app((_,z),_,_))))::_) =
        node := app((ref(app((x,z),ref Eval,ref [])),
                     ref(app((y,z),ref Eval,ref []))),
                    ref Eval,ref [])
   |apply (PLUS,ref(app((_,ref(atom(int x,_,_))),_,_))::(node as
               ref(app((_,ref(atom(int y,_,_))),_,_)))::_) =
        node := atom(int(x+y),ref Ready,ref [])
   |apply (PLUS,(stack as ref(app((_,x),_,_))::
                          ref(app((_,y),_,_))::_)) =
        node := atom(int((eval x)+(eval y)),ref Ready,ref [])
```

Die schrittweise Durchführung der Reduktion eines Knotens `node` geschieht mittels der Funktion `step node`:

```
fun step node =
   let
    val (c,stack) = (spine node []);
   in
    if is_atom c then ()
    else let val comb(k,_,_)= c
         in apply (k,stack)
         end
   end;
```

10.3.4 Parallelisierung

Die Lazy–Evaluation besitzt mehrere nützliche Eigenschaften. Da sie die Reduktion in Normalordnung nachbildet, terminiert sie immer, wenn eine Normalform existiert (im Gegensatz zur Eager–Evaluation). Sie erlaubt außerdem die Verwendung sehr großer, potentiell unendlicher Datenstrukturen, da sie immer nur diejenigen Teile der Strukturen berechnet, die von anderen Teilen der Gesamtberechnung benötigt

werden[8]. Das Ziel von parallelisierten Evaluationsmechanismen sollte es offenbar sein, die Semantik der Lazy Evaluation zu bewahren und ihre Effizienz durch Verwendung von Mehr-prozess(or) Techniken zu optimieren.
Im Gegensatz zu den *nicht strikten* Funktionen (wie etwa den Kombinatoren S, K, I, B, C und $CONS$, die ihre Argumente vor der Applikation *nicht* auswerten, erwarten *strikte* Operationen (wie etwa die arithmetischen und booleschen Operationen) stets vollständig evaluierte Argumente. In den Reduktionsregeln (10.16) bis (10.24) ist diese Evaluierung eines Argumentes x einer strikten Funktion durch (`eval` x) ausgedrückt. Die Evaluation des Terms $(+\ x\ y)$ muß also zunächst die Ausdrücke x und y auswerten und kann erst dann die resultierenden Werte addieren (vgl. Gleichung 10.16). Dies spiegelt sich in der Implementierung des Kombinators $PLUS$ in der Funktion `apply` direkt wider (Unterstreichung):

```
|apply (PLUS,(stack as ref(app((_,x),_,_))::
                        ref(app((_,y),_,_))::_)) =
     node := atom(int((eval x)+(eval y)),ref Ready,ref [])
```

x und y müssen also auch im Fall der Lazy Evaluation immer ausgewertet werden. Da die Argumente strikter Operationen immer evaluiert werden müssen, können durch *Striktheitsanalyse* diejenigen Teile von Berechnungen ermittelt werden, die unabhängig von der Auswertungsstrategie unbedingt berechnet werden müssen. Strikte Operationen bieten daher die Möglichkeit zur Parallelisierung der Lazy Evaluation[9].
Im Folgenden wird eine an [Dav92] orientierte Implementierung einer parallesierten Kombinatormaschine entwickelt, die die oben geschilderten Eigenschaften strikter Operationen ausnützt.
Die Reduktionsregeln der nicht-strikten Kombinatoren sind von der Parallelisierung nicht betroffen und operieren weiterhin wie oben definiert. Die Reduktion strikter Kombinatoren geschieht wie am Beispiel der Additionsoperation `PLUS` gezeigt:

```
|apply (PLUS,ref(app((_,ref(atom(int x,_,_))),_,_))::(node as
            ref(app((_,ref(atom(int y,_,_))),_,_))::_) =
     node := atom(int(x+y),ref Ready,ref [])
|apply (PLUS,(stack as ref(app((_,x),_,_))::
                       ref(app((_,y),_,_))::_)) =
     (subEval (last stack,x);
      subEval (last stack,y); ())
```

Falls die Argumente bereits ausgewertet vorliegen (im Falle der Addition als Integer Zahlen), kann direkt operiert werden (erster Teil der Definition), ansonsten müßen zunächst die Argumente `x` und `y` ausgewertet werden (zweiter Teil). Die Funktion `subEval` wertet die Argumente nicht selbst ausm, was wieder eine sequentielle Bearbeitung von `x` und `y` zur Folge hätte, sondern erzeugt neue Subprozesse, die dem Schedulingmechanismus übergeben werden:

```
fun subEval (root,node) =
 let
   val emark = get_mark node;
   val wq = get_q node;
 in
```

[8] Vgl. beispielsweise die Verwendung der Lazy Lists Konzeption zur Implementierung des Trito–universums TU in Kapitel 4.
[9] [Dav92], S. 201.ff.

```
  if (! emark = Ready) then ()
  else if (! emark = Busy) then
   (make_wait root;
    wq := root::(! wq))
  else
   (make_wait root;
    emark := Busy;
    wq := [root];
    newTask node)
 end;
```

Falls die Evaluationsmarkierung des Subknotens `node` auf `Ready` gesetzt ist, wurde der Knoten bereits ausgewertet und der `subEval` Prozess kann sofort beendet werden. Falls die Markierung `Busy` lautet, zeigt dies an, daß bereits ein anderer Prozeß an der Evaluierung des Knotens arbeitet. Es muß daher kein neuer Prozeß zur Evaluierung von `node` gestartet werden, jedoch muß der Wurzelknoten von `node`, `root`, so lange in den Wartemodus gesetzt werden, bis die Reduktion abgeschlossen ist. Dies geschieht durch die Funktion `(make_wait root)`:

```
fun make_wait node =
 let
  val (k,(w::_)) = spine node [];
  val ref(app((ref rator,rand),_,_)) = w;
 in
  w := app((ref(app((ref(comb(WAIT,ref Eval,ref [])),
                    ref rator),ref Eval,ref [])),
            rand),E,Q)
 end;
```

Diese Funktion ersetzt den Spine Kombinator von `root` durch den Kombinator $WAIT$. Die Reduktionsregeln für $WAIT$ lautet:

$$WAIT\ x \quad = \quad WAIT1\ x \tag{10.25}$$

$$WAIT1\ x \quad = \quad WAIT\ x \tag{10.26}$$

Diese beiden speziellen Kombinatoren erzeugen eine Endlosschleife, die den Evaluationsvorgang von `root` solange suspendiert, bis durch Terminierung der Subprozesse der $WAIT$ Kombinator wieder entfernt wird:

```
  fun make_unwait node =
   let
    val (_,w::_) = spine node [];
    val ref(app((_,ref k),_,_)) = w;
   in
    w := k
   end;
```

Damit der Subknoten `node` bei der Beendung seiner Reduktion auch den Prozess root, der ihn ja nicht gestartet hat, reaktivieren kann, muß `root` in die *Waitequeue* von `node` aufgenommen werden. Diese Warteschlange enthält alle Knoten, deren Evaluation auf

die Auswertung von `node` wartet. Ist die Evaluation von `node` abgeschlossen, werden alle diese Prozesse reaktiviert.
Falls die Markierung jedoch `Eval` lautet, muß `node` evaluiert werden. `node` wird in den `Busy` Status gesetzt, was anderen Prozessen signalisiert, daß `node` gerade bearbeitet wird. Durch die Zuweisung `wq := [root]` wird die Warteschlange von `node` initialisiert, zunächst wartet nur der Knoten `root` auf die Auswertung von `node`. Dann wird durch `(newTask node)` ein neuer Prozeß, nämlich die Evaluation von node, gestartet.

```
fun newTask task =
   Tasks := (task::(! Tasks));
```

Die globale Variable `Tasks` enthält einen Zeiger auf eine Liste aller zu evaluierender Knoten und bildet den *Task-pool* der Kombinatormaschine.
Der Schedulingmechanismus eines parallelverarbeitenden Systems ist für die optimale Verteilung der Tasks auf die physikalischen Prozessoren verantwortlich. Da es hier lediglich um die *Modellierung* einer parallelen Architektur geht, wird ein rudimentärer stochastischer Scheduler benutzt, der das Verhalten eines Mehrprozessorsystems simuliert:

```
val Seed = ref 4.34793;

fun Scheduler () =
 let
  fun rnd () =
   let
    val x = (! Seed)* 1475.37697;
    val res = x-real(floor x);
   in
    (Seed := res; res)
   end;
  fun intrand n = floor(rnd()*(real n));
 in
  if (!Tasks) = [] then ()
  else
     (evalstep (nth(!Tasks,intrand (length (!Tasks))));
      Scheduler())
 end;
```

Der Scheduler wählt zufällig einen Knoten aus dem Task-pool `!Tasks` aus und führt für ihn einen Evaluationsschritt aus. Diesen Vorgang wiederholt er so lange, bis alle Knoten vollständig evaluiert sind und wieder aus dem Task-pool entfernt wurden. Ein einzelner Evaluationsschritt wird durch die Funktion `evalstep node` ausgeführt:

```
fun evalstep (node as ref(atom(_,_,_))) = remTask node
   |evalstep (node as ref(comb(_,_,_))) = remTask node
   |evalstep (node as ref(app((ref rator,ref rand),Emark,WQ))) =
     let
      val old = copy node
     in
      (step node;
       if ((equal old node) orelse
```

```
          (!Emark = Ready) orelse
          (not (is_app (!node))))
       then (wakeup WQ;
             remTask node;
             Emark := Ready)
       else ())
    end;
```

Falls `node` atomar, d.h vom Typ `value` oder `comb` ist, evaluiert der Knoten zu sich selbst und die Evaluierung kann beendet werden (`remTask node`).

```
fun remTask task =
   Tasks := (remove task (! Tasks));
```

Andernfalls wird ein Evaluationsschritt `(step node)` ausgeführt. Danach wird überprüft, ob die Normalform bereits gefunden wurde. Dies geschieht durch den Vergleich des Zustandes vor dem `step` mit dem neuen. Falls sie gleich sind, ist eine Normalform gefunden und die Reduktion somit abgeschlossen. Die auf das Ergebnis wartenden Prozesse können mit `(wakeup WQ)` wieder reaktiviert werden:

```
fun wakeup waitQ =
  (map (fn task => (make_unwait task))
       (! waitQ);
   waitQ := []);
```

Dann wird `node` aus dem Task-pool entfernt und die Evaluationsmarkierung auf `Ready` gesetzt. Falls noch keine Normalform gefunden wurde, ist der Evaluationsschritt direkt beendet. Anschließend kann der Scheduler mit seiner Bearbeitung des Task-pools fortfahren.

Beispiel: Das Verhalten der parallelisierten Kombinatormaschine soll an der Auswertung des Ausdrucks $x = (ADD(ADD\ 1\ 2)(ADD\ 3\ 4))$ veranschaulicht werden. Der Ausdruck wird zunächst übersetzt und alloziert:

```
- val x = alloc(ropt(c(r "ADD (ADD 1 2)(ADD 3 4)")));
```

Der Task-pool `Tasks` ist vor der Berechnung leer, durch `(newTask x)` wird x als erster Prozess in den Taskpool gesetzt:

`Tasks:` $\boxed{ADD(ADD\ 1\ 2)(ADD\ 3\ 4)}$

Dann wird mit `Scheduler ()` der Scheduler gestartet. Da x der einzige Prozess ist, wird er für die Durchführung eines `evalstep` ausgewählt, was durch die Unterstreichung markiert ist:

$$\boxed{\underline{ADD(ADD\ 1\ 2)(ADD\ 3\ 4)}}$$

Die Auswertung des äußersten ADD Kombinators erzeugt zwei neue Prozesse $(ADD\ 1\ 2)$ und $(ADD\ 3\ 4)$ und setzt die äußere Berechnung so lange in den Wartemodus, bis die Subprozesse terminieren (Die parallelen Prozesse in `Tasks` sind jeweils durch vertikale Doppelstriche $\|$ von einander getrennt):

$$\underline{WAIT(WAIT(ADD(ADD\ 1\ 2)(ADD\ 3\ 4)))} \parallel ADD\ 1\ 2 \parallel ADD34$$

Wählt der Scheduler nun zufällig den ersten Knoten zur Auswertung aus, so wird $WAIT$ durch $WAIT1$ ersetzt. Erneute Reduktionen dieses Knotens würden $WAIT1$ wiederum durch $WAIT$ ersetzen. Dieser Wechsel zwischen den beiden Kombinatoren wird bei jedem Reduktionsschritt wiederholt. Der Evaluationsprozess des Knoten bleibt daher so lange in einer Warteschleife, bis der Wartekombinator durch Reaktivierung des Prozesses wieder entfernt wird. Wird jedoch der zweite Knoten ausgewählt:

$$WAIT1(WAIT(ADD(ADD\ 1\ 2)(ADD\ 3\ 4))) \parallel \underline{ADD\ 1\ 2} \parallel ADD34$$

so stellt die Funktion `Apply` fest, daß die Argumente der Addition bereits atomar (d.h. vollständig ausgewertet) sind und kann eine direkte Addition ausführen. Durch die Verzeigerung des Graphen führt diese Reduktion dazu, daß auch an der Stelle '$(ADD\ 1\ 2)$' des ersten Knotens nun das Ergebnis '3' steht. Da durch die einmalige Anwendung von `evalstep` bereits eine Normalform gefunden wurde, kann der zweite Knoten nun aus dem Taskpool entfernt werden und der wartetende Prozess reaktiviert werden. Da der erste Prozess nu noch auf einen Prozess wartet, wird er nur noch von einem $WAIT$ blockiert. Im Pool befinden sich jetzt nur noch zwei auszuwertende Knoten:

$$WAIT1(ADD\ 3\ (ADD\ 3\ 4)) \parallel \underline{ADD\ 3\ 4}$$

Wird nun als nächstes der verbleibende zweite Knoten ausgewertet, wird wie zuvor sofort eine Normalform, nämlich $3+4 = 7$ gefunden. Der Prozess wird entfernt, der wartende Prozess reaktiviert. Somit verbleibt nur noch ein Knoten im Pool:

$$\underline{ADD\ 3\ 7}$$

Hier kann ebenfalls mit einem Berechnungsschritt das Ergebnis $3+7 = 10$ bestimmt werden. Dann wird auch dieser letzte Knoten aus dem Taskpool entfernt. Daraufhin beendet der Scheduler seine Arbeit und der Knoten `x` enthält das Berechnungsergebnis.

```
> x;
- val it = value(int 10,ref Ready,ref []) : node
```

Der hier verwendete Scheduler simuliert die parallele Auswertung verschiedener Knoten durch die wiederholte Ausführung eines einzelnen Reduktionsschrittes auf einen zufällig aus dem Taskpool ausgewählten Knoten. Dieses *quasi–parallele* Auswertungsschema entspricht der Arbeitsweise eines Einprozessor Multitasking Betriebssystems. Ein solches System stellt abwechselnd jedem Prozess eine bestimmte Prozessorzeit zur Verfügung. Das Hin- und Herschalten zwischen den einzelnen Prozessen erfolgt dabei

in so schneller Folge, daß für den Benutzer der Eindruck einer simultanen Bearbeitung der Prozesse entsteht. Würde anstelle des Schedulers eine Mehrprozessormaschine zur Bearbeitung der Prozesse im Taskpool verwendet, könnten diese tatsächlich simultan (d.h. zeitgleich) ausgeführt werden.
Das im vorhergehenden Beispiel demonstrierte Auswertungsschema kann zu der Funktion `eval` zusammengefasst werden. Nachdem ein λ-Term compiliert und alloziert wurde, kann er von `eval` zu seiner Normalform reduziert werden:

```
fun eval node =
 (newTask node;
  Scheduler();
  node);

val ENV = ref [] : (string * node ref) list ref;

fun define name value =
   ENV := (name, alloc(ropt(c(r value)))):(!ENV);

fun run exp = eval(alloc(ropt(c(r exp))));

- define "fac" "%n.IF (EQ n 0) 1 (MUL n (fac (SUB n 1)))";
> () : unit
- run "fac 7";
> val it = value(int 5040,ref Ready,ref []) : node
```

Hiermit ist die Implementierung der parallelisierten Graphreduktionsmaschine abgeschlossen. Im den folgenden Abschnitten wird die operationale Semantik dieser abstrakten Maschine modifiziert und erweitert, um den Rahmen der klassisch funktionalen Sprachen um kenogrammatische Programmierkonstrukte zu erweitern.

10.4 Modellierung der Proemialrelation

Die Einführung einer parallelisierten Lazy-Evaluation funktionaler Programmiersprachen hatte es zur Aufgabe, die Vorteile einer Multi-prozess-Architektur mit denen der Lazy-Evaluation zu vereinen. Oberstes Gebot war hierbei die Bewahrung der Normalorder-Reduktion und der Semantik der funktionalen Programme.
Aus diesem Grund wurden keine Sprachkonstrukte für eine explizite Handhabung der parallelen Architektur eingeführt, sondern die Parallelisierung nur implizit für die Auswertung strikter Operationen, unter strenger Beachtung der Synchronisation der beteiligten Prozesse eingeführt. Daher muß ein Prozeß, der neue Unterprozesse startet, so lange im Wartemodus verbleiben, bis alle Subprozesse terminieren und ihn reaktivieren. Ein solcher Parallelisierungsmechanismus erfüllt zwar die an ihn gestellten Anforderungen (Einhaltung der Lazy-Evaluation Semantik), deckt jedoch nur einen sehr begrenzten Bereich der parallelen Prozesse ab, die auf einer wie oben modellierten Architektur möglich wären.
Im Folgenden soll ein weiteres Parallelitätskonzept entwickelt werden, daß eine Modellierung der kenogrammatischen Proemialrelation darstellt. In 3.1.2 wurde die Proemialrelation als das simultane Wechselverhältnis von Ordnungs- und Umtauschbeziehungen zwischen Objekten verschiedener logischer Stufen charakterisiert. Aus den

Arbeiten zur Kenogrammatik sind keine algebraischen Charakterisierungen der Proemialrelation bekannt. Aufgrund der speziellen Eigenschaften der Proemialrelation müßte eine algebraische Dartellung der Proemialrelation selbstreferentiell, innerhalb klassischer Formalismen also paradoxal und antinomisch strukturiert sein. Aufgrund der Mängel einer solchen Darstellung wird hier versucht, eine *operationale* Modellierung der Proemialrelation zu entwickeln[10]. Hierzu wird die operationale Semantik der abstrakten Kombinatormaschine um eine proemiellen Kombinator PR erweitert.
Die in 3.1.2 eingeführte Proemialrelation beschreibt das Zusammenspiel einer Ordnungsrelation, $\longrightarrow$, und einer Umtauschrelation, $\Updownarrow$, zwischen je zwei Operatoren und Operanden.

$$PR\Big(R_{i+1}, R_i, x_i, x_{i-1}\Big) :$$

$$\begin{array}{lcl} R_{i+1} & \longrightarrow & x_i \\ & & \Updownarrow \\ & & R_i \quad \longrightarrow \quad x_{i-1}. \end{array}$$

Kaehr unterscheidet die *offene* und *geschlossene* Form der Proemialrelation[11]. Die geschlossene Proemialrelation ist zyklisch:

$$\begin{array}{lcl} R_{i+1} & \longrightarrow & x_i \\ \Updownarrow & & \Updownarrow \\ x_{i-1} & \longleftarrow & R_i \end{array}$$

Es gilt also $PR(PR^i) = PR^i$. Für die offene Proemialrelation gilt hingegen: $PR(PR^i) = PR^{(i+1)}$. Sie hat die folgende Gestalt:

$$\begin{array}{lllll} i+1\text{:} & R_{i+2} \longrightarrow & x_{i+1} & & \\ & & \Updownarrow & & \\ i\text{:} & & R_{i+1} \quad \longrightarrow & x_i & \\ & & & \Updownarrow & \\ i-1\text{:} & & & R_i & \longrightarrow x_{i-1} \end{array}$$

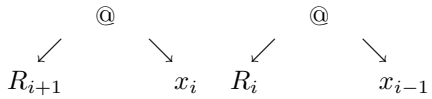
PR wurde bisher nur informell beschrieben als eine Relation, die ein gegebenes Objekt als *simultan* auf mehrere logische Ebenen verteilt bestimmt. Diese Eigenschaft erinnert an die Möglichkeit des λ-Kalküls, *identische* λ-Terme sowohl als Operatoren als auch als Operanden zu verwenden. Die simultane Verteilung des *selben* λ-Terms über mehrere logische Stufen läßt sich jedoch im λ-Kalkül nicht modellieren.
Der λ-Term $(f\ x)(x\ f)$ verwendet f und x innerhalb eines Ausdrucks sowohl als Operator als auch als Operand. Innerhalb des λ-Kalküls selbst kann, wie in allen semiotisch fundierten Kalkülen, zwar die *Identität* von Termen, nicht jedoch deren *Selbigkeit* als einmalig realisierte Objekte ausgedrückt werden. Die semiotische Gleichheit des Zeichen 'f' in '$(f\ x)$' mit dem 'f' in '$(x\ f)$' besagt nichts darüber, ob diese beiden identischen Terme innerhalb einer Reduktion des λ-Terms auch als physikalisch gleich, d.h als das *selbe* Objekt behandelt werden. Der Grund hierfür liegt in der Token–Type–Relation der Semiotik, die den λ-Kalkül fundiert. Die Token–Type–Relation subsumiert alle physikalisch verschiedenen, jedoch *gleichgestalteten* Token unter einem Type. Da mit diesem Schritt von der physikalischen Realisierung der

[10] Die Problematik der Formalisation paradoxaler und selbstreferentieller Systeme wurde in Kapitel 2 erörtert.
[11] [Kae78], S. 5 f.

Token abstrahiert wird, kann die physikalische Identität oder Differenz verschiedener Zeichenreihenvorkommen nicht aus den Termen eines semiotisch fundierten Kalküls abgeleitet werden. Ein Kalkül operiert nur mit Types und da die Typegleichheit (oder auch Gestaltgleichheit $\equiv_{sem}$) nicht auch Tokengleichheit (d.h. physikalische oder Zeigergleicheit $\equiv_z$) impliziert, kann er gar keinen Begriff der Selbigkeit verwenden[12].
Aus der syntaktischen Struktur von λ-Termen kann desweiteren auch nicht abgeleitet werden, nach welchem Reduktionsverfahren sie auszuwerten sind. Aus dem obigen Ausdruck geht nicht hervor, ob — und in welcher Reihenfolge — $(f\ x)$ und $(x\ f)$ sequentiell, oder simultan ausgewertet werden sollen. Die Frage der Selbigkeit von Repräsentationen von λ-Termen bleibt ebenso wie die Wahl der Auswertungsstrategie allein Thema der Implementierungstechniken des λ-Kalküls, da sie die (semiotisch fundierte) Semantik der Terme nicht betrifft.
In der Proemialrelation ist nun aber gerade die im λ-Kalkül (stellvertretend für alle klasssischen formalen Systeme) nicht abbildbare simultane Verteilung des selben Objektes über mehrere Bezugssysteme gemeint. Das hier vorgeschlagene Modell der Proemialrelation versucht nun, gerade dieses Verhalten operational abzubilden. Dazu geht es von Günthers Konzeption der Kenogramme als Leerstellen, an denen semiotische Prozesse eingeschrieben werden können, aus. Die Selbigkeit eines Terms (die auf semiotischer Ebene nicht definiert werden kann) wird dann durch die Selbigkeit des Kenogramms, in das er eingeschrieben wird, bestimmt. Dieser in einem und dem selben Kenogramm realisierte Term kann nun simultan innerhalb verschiedener semiotischer Prozesse sowohl als Operator als auch als Operand dienen.

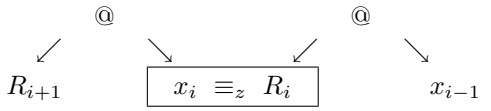
10.4.1 Implementierung des Proemialkombinators PR

Von dieser grundlegenden Idee der innerhalb der Kenogrammatik notierten Selbigkeit semiotischer Prozesse ausgehend soll nun die operationale Semantik des Proemialkombinators $PR\big(R_{i+1}, R_i, x_i, x_{i-1}\big)$ mittels der oben entwickelten Kombinatormaschine bestimmt werden.
Dieses Modell nutzt die Homogenität von Programmen und Daten der Graphrepräsentation der Kombinatormaschine aus. So kann ein bestimmter Knoten z, der als physikalisches Objekt an einer bestimmten Adresse im Arbeitsspeicher realisiert ist, innerhalb verschiedener Applikationsknoten sowohl als Operator als auch als Operand dienen. Aufgrund der parallelen Architektur der Kombinatormaschine kann dieser Rollenumtausch (Operator $\Longleftrightarrow$ Operand) des selben Knotens z simultan ausgeführt werden.
R_i, R_{i+1}, x_i und x_{i-1} können beliebige Knoten des Kombinatorgraphen sein. Die Ordnungsrelation der Proemialrelation, $\longrightarrow$, ist hier die Applikation `app(rator,rand)`, die stets eine eindeutige Unterscheidung von Operator und Operand gewährleistet:



[12] Quine formuliert diese der klassischen Semiotik zugrundeliegende Konzeption wie folgt: „*Um was für Dinge handelt es sich denn, genaugenommen, bei diesen Ausdrücken [den Termen eines klassischen formalen Systems]? Es sind Typen, keine Vorkommnisse. Jeder Ausdruck ist also, so könnte man annehmen, die Menge aller seiner Vorkommnisse. Das heißt, jeder Ausdruck ist eine Menge von Inschriften, die sich an verschiedenen Orten des Raum-Zeit-Kontinuums befinden, die aber Aufgrund ähnlichen Aussehens zusammengefasst werden.*“ [Qui75].

Zwei solche Applikationsknoten können an beliebigen Stellen innerhalb eines Kombinatorgraphen vorkommen. Innerhalb eines solchen Knotens ist durch die Applikationsstruktur stets determiniert, ob ein Subknoten als Operator oder als Operand dient. Durch die Verzeigerung des Kombinatorgraphen ist es nun möglich, daß x_i und R_i das *selbe* physikalische Objekt z bezeichnen. Innerhalb von $\texttt{app}(R_{i+1}, x_i)$ fungiert z dann als Operand und in $\texttt{app}(R_i, x_{i-1})$ als Operator:



Die Zeigergleichheit von x_i und R_i, $x_i \equiv_z R_i$, bewirkt also, daß z in verschiedenen Applikationen (Ordnungsverhältnissen) als Operator und als Operand fungiert. Dieser Positionswechsel innerhalb der Ordnungsrelation `app(rator,rand)` dient als Modell für die Umtauschrelation der Proemialrelation, $\Leftrightarrow$. Somit erfüllen R_i, R_{i+1}, x_i und x_{i-1}, mit $R_i \equiv_z x_i$, das Schema der Proemialrelation:

$$\begin{array}{ccc} R_{i+1} & \longrightarrow & x_i \\ & & \Updownarrow \\ & & R_i \quad \longrightarrow \quad x_{i-1}. \end{array}$$

Die in der informellen Spezifikation der Proemialrelation geforderte Simultaneität der logischen Ebenen, innerhalb derer z an verschiedenen Stellen der Ordnungsrelation `app(rand,rator)` steht, wird nun dadurch modelliert, daß die Applikationen $\texttt{app}(R_{i+1}, x_i)$ und $\texttt{app}(R_i, x_{i-1})$ simultan ausgewertet werden. Das physikalische Objekt z dient dann simultan (im Sinne der jeweiligen verwendeten parallelen Architektur) innerhalb verschiedener Applikationen sowohl als Operator als auch als Operand. Diese theoretische Konzeption führt zu der folgenden Implementierung des Proemialkombinators PR:

```
  |apply (PR, (stack as ( ref(app((_,R1),_,_))::
                         ref(app((_,R2),_,_))::
                         ref(app((_,x1),_,_))::(node as
                        (ref(app((_,x2),_,_))))::_))) =
       let
        val first =  ref(app((R1,x1),ref Eval,ref []));
        val second = ref(app((R2,x2),ref Eval,ref []));
       in
        (node := app((ref(app((ref(comb(CONS,ref Ready,ref [])),
                              first),ref Ready,ref [])),
                     second),ref Ready,ref [];
         parEval (last stack, first);
         parEval (last stack, second))
       end


fun parEval (root,node) =
 let
  val emark = get_mark node;
  val wq = get_q node;
 in
```

```
if (! emark = Ready) then ()
else if (! emark = Busy) then
 (make_wait root;
  wq := root::(! wq))
else
 (emark := Busy;
  newTask node)
end;
```

Die Reduktion des Kombinators `PR` erwartet vier Argumente, `R1, R2, x1` und `x2`. Aus diesen werden zwei Applikationen `first` = (`app(R1,x1)`) und `second` = (`app(R2,x2)`) konstruiert. Diese beiden Knoten werden zunächst zu einem Consobjekt `CONS(first,second)` zusammengefügt, dadurch bleiben etwaige Ergebnisse der Applikationen `first` und `second` erhalten und können später inspiziert werden (für die eigentliche Semantik von `PR` ist dies jedoch nicht notwendig). Anschließend werden durch `(parEval first)` und `(parEval second)` die zwei Applikationen als parallele Prozesse dem Scheduling-mechanismus übergeben.

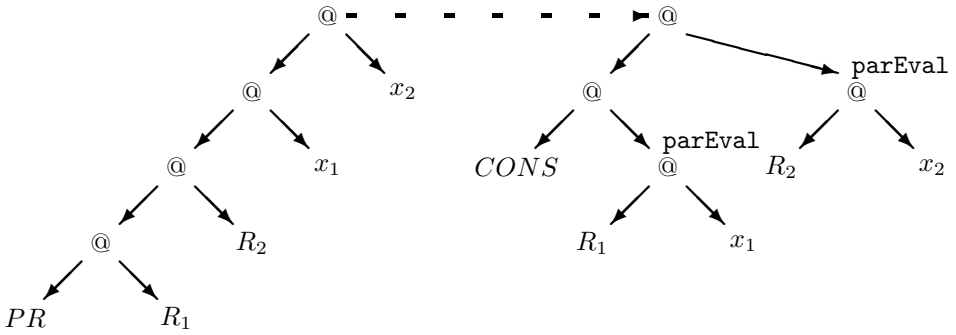


Abbildung 10.4: Die Graphreduktion von $PR(R_1, R_2, x_1, x_2)$

Dieses Reduktionsschema ist in Abbildung (10.4) dargestellt. Die Markierung `parEval` an den Knoten $@(R_1, x_1)$ und $@(R_2, x_2)$ sollen hierbei andeuten, daß die Reduktion weder strikt noch nicht-strikt verläuft: Die Knoten werden nicht ausgewertet, bevor sie zu einem $CONS$-Objekt zusammengefügt werden. Im Gegensatz aber zur Reduktion nicht-strikter Operationen werden sie durch die Funktion `parEval` der parallelen Auswertung durch den Scheduler übergeben.
Die Funktion `parEval` ähnelt der Funktion `subEval`, im Gegensatz zu dieser werden jedoch die beteiligten Prozesse nicht synchronisiert. Die Synchronisation mittels `(make_wait root; wq := [root])` kann entfallen, da `first` und `second` nicht Subprozesse eines übergeordneten strikten Operators sind, der auf ihre Berechnungsergebnisse wartet, sondern autonome, lediglich proemiell verknüpfte Prozesse sind.
Sind nun R_2 und x_1 als das *selbe* physikalische Objekt z realisiert, so ergibt sich offensichtlich folgendes Reduktionsschema (Abbildung 10.5).
Bezüglich R_1 ist z hier Operand, bezüglich x_2 Operator. Da die beiden Applikationen $@(R_1, z)$ und $@(z, x_2)$ parallel ausgertet werden, erfüllt diese Reduktion das simultane Umtauschverhältnis der Proemialrelation.

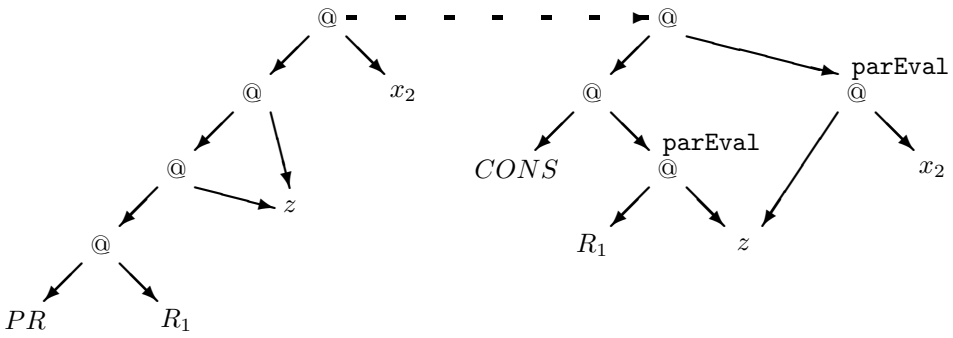


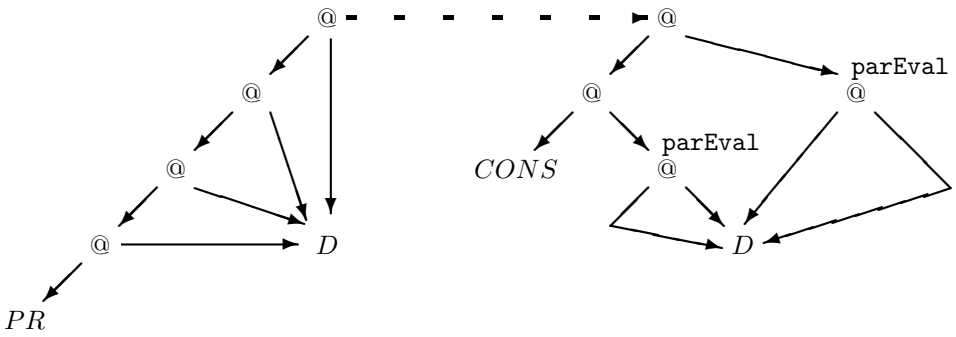
Abbildung 10.5: Die Graphreduktion von $PR(R_1, z, z, x_2)$

Beispiel: Als Beispielberechnung soll hier der selbstreproduzierende λ-Term DD mit $D = (\lambda x.xx)$ proemiell verteilt werden:

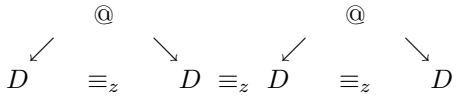
```
- define "D" "%x.xx";
> () : unit

- run "(%x.PR x x x x) D";
```

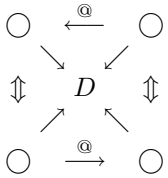
Das Reduktionsschema ist also:



Für diese offensichtlich nichtterminierende Berechnung ist die Struktur des Kombinatorgraphen wie folgt ($\equiv_z$ bezeichnet hierbei die Zeigergleichheit von Objekten):



Ein einziges physikalisches Objekt ist somit über die vier Stellen der Proemialrelation verteilt:



Die Kreise ◯ symbolisieren hier die vier Stellen der Proemialrelation, die jeweils einen Verweis auf das Objekt D enthalten. Die Verweise sind jeweils durch einfache Pfeile notiert. Die Pfeile $\overset{@}{\longleftarrow}$ bezeichnen die Ordnungsrelation der Proemialrelation, die hier durch die Trennung von Operator und Operand *innerhalb* der Applikation @ realisiert ist. Die Umtauschrelation der Proemialrelation, ⇳, steht für den Funktionswechsel von Operator zu Operand (und umgekehrt) der ◯-Objekte *zwischen* den verschiedenen Applikationen @.

Allgemein gilt für $\texttt{PR}(f, g, x, y)$: falls $x \equiv_z g$, dann sind die beiden Berechnungen $(f\ x)$ und $(g\ y)$ (offen) proemiell verkoppelt. Falls außerdem $y \equiv_z f$ liegt die geschlossene Proemialrelation vor. Ist wie im obigen Beispiel $f \equiv_z g \equiv_z x \equiv_z y$, so ist die Struktur *autoproemiell geschlossen*.

10.4.2 Anwendungen des Proemialkombinators PR

10.4.2.1 Meta-level Architekturen und Reflektionale Programmierung

Die Definition des Kombinators erweitert den Sprachrahmen funktionaler Programmiersprachen um die Modellierung proemieller Konstruktionen. Um die Nützlichkeit einer solchen Erweiterung zu demonstrieren, wird hier kurz ein mögliches Anwendungsgebiet gestreift.
Unter den Schlagworten *Computational Reflection* (CR) und *Meta-level-Architectures* versammeln sich in der aktuellen Grundlagenforschung der Informatik Bemühungen, den klassischen Berechnungsbegriff, wie er etwa im λ-Kalkül formuliert wird, zu erweitern. Insbesondere geht es darum, Berechnungssysteme zu entwickeln, die über ihre Berechnungen 'reflektieren', d.h. wiederum Berechnungen anstellen können.
Nach Maes[13] zeichnet sich eine reflektive Programmiersprache dadurch aus, daß sie Methoden zur Handhabung reflektiver Berechnungen explizit bereitstellt. Konkret heißt dies, daß:

1. Der Interpreter einer solchen Sprache es jedem auszuwertenden Programm ermöglichen muß, auf die Datenstrukturen, die das Programm selbst (oder bestimmte Aspekte des Programms) repräsentieren, zuzugreifen. Ein solches Programm hat dann die Möglichkeit diese Daten, d.h. seine eigene Repräsentation zu manipulieren (Meta–berechnung).

2. Der Interpreter außerdem gewährleisten muß, daß eine kausale Verbindung (*causal connection*) zwischen diesen Daten und den Aspekten des Programmes, die sie repräsentieren, besteht. Die Modifikation, die ein Reflektives Programm an seiner Repräsentation vornimmt, modifizieren also auch den Zustand und wei-

[13] [Mae88], S.22 ff.

teren Ablauf des Programmes (der Objektberechnung). In diesem Sinne werden die Metaberechnungen in der Objektberechnung reflektiert[14].

In einem solchen System können Repräsentationen von Berechnungsvorschriften einerseits auf der Ebene der Objektberechnungen *als Programm* evaluiert werden, oder zum anderen — z.B. im Falle eines Fehlers — auf einer Meta-berechnungsebene *als Daten* einer Metaberechnung dienen, die beispielsweise den Fehler korrigieren soll. Diese Struktur ist im nachfolgenden Schema (10.6) dargestellt.

$$\begin{array}{lllll} \text{Metaberechnung:} & PRG_2 & \longrightarrow & DAT_1 & \\ & & & \Updownarrow & \\ \text{Objektberechnung:} & & & PRG_1 & \longrightarrow \quad DAT_0 \end{array}$$

$\longrightarrow$: Programm PRG_i arbeitet auf Daten DAT_{i-1}.
$\Updownarrow$: PRG_i wird zu DAT_i und umgekehrt.

Abbildung 10.6: Logische Ebenen einer reflektiven Berechnung

Der Umtausch des Operators PRG_i der Objektberechnung zum Operanden DAT_i der Metaberechnung kann von der Umtauschrelation der Proemialrelation $\Updownarrow$ beschrieben werden. Die Unterscheidung von Programm (Operator) und Datum (Operand) innerhalb einer Berechnungsebene entspricht der Ordnungsrelation der Proemialrelation. Das Strukturschema eines reflektiven Berechnungssystems entspricht daher exakt dem der Proemialrelation.
Da der Proemialkombinator $PR\ x_i$ und R_i als ein einziges physikalisches Objet handhabt, wirken sich alle Modifikationen von x_i direkt auch auf R_i aus[15]. Die Zeigergleichheit $x_i \equiv_z R_i$ gewährleistet somit die kausale Verknüpfung von Meta– und Objektebene. Der Proemialkombinator PR ist daher zur Modellierung von reflektiven Systemen im Sinne von Maes Definition geeignet.
In den bislang bekannten reflektiven Systemen (z.B. 3LISP [Smi82]) sind Meta– und Objektebenen nicht als simultane Prozesse realisiert, sondern treten nur rein sequentiell in Aktion. Daher beeinflussen Metaberechnungen PRG_{i+1}, die die Repräsentation des Objektprogramms als Datum DAT_i manipulieren,die Objektberechnung $(PRG_i \longrightarrow DAT_{i-1})$ nicht gleichzeitig. Sie werden erst dann wirksam , wenn die Metaberechnung terminiert und wieder die Objektebene aktiviert, indem sie dem Interpreter die veränderte Objektberechnungsvorschrift übergibt. Zu einem bestimmten Zeitpunkt der Gesamtberechnung wird also entweder auf einer Metaebene oder auf der

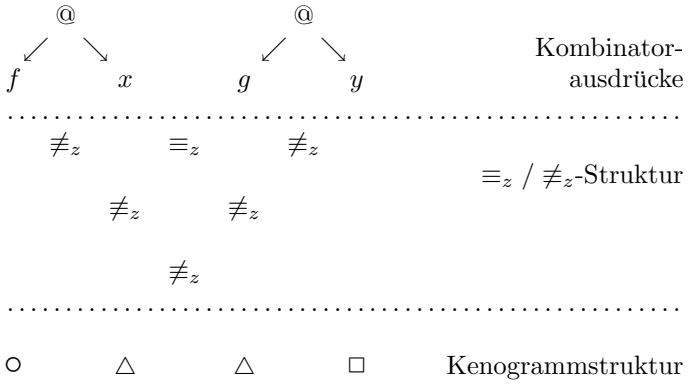
[14] In vielen Punkten entspricht das Konzept der computational reflection dem Konzept der Meta–level–Architekturen[Har91]. Eine Meta–level–Architektur ermöglicht Metaberechnungen, d.h. das Schlußfolgern (Agieren) eines Berechnungssystems über (auf) ein zweites Berechnungssystem. Eine Meta–Level–Architektur erlaubt jedoch nicht notwendigerweise auch reflektive Berechnungen. Dies ist zum Beispiel nicht der Fall, wenn die Metaberechnung nur statischen Zugriff auf das Objektsystem hat, wenn also keine kausale Verknüpfung zwischen den beiden existiert.

[15] In rein funktionalen Sprachen sind Reduktionen, die keine semantische Veränderung bewirken, die einzig möglichen Modifikationen. Sind jedoch — wie die Zuweisung `:=` in ML — auch imperative Konstrukte zugelassen, lassen sich Modifikationnen erzeugen, die auch die Semantik einer Variablen verändern. Um semantische Modifikationen der Objektberechnung zu erzeugen, wird daher streng genommen noch ein Zuweisungsoperator $SET\ x\ value$ benötigt, der der Variablen x den Wert $value$ zuweist (vgl. Anhang).

Objektebene evaluiert. Ob eine Berechnungsvorschrift also als Programm (Operator) oder als Datum (Operand) dient, ist stets streng disjunkt determiniert.
Der Proemialkombinator PR und auf ihm basierende Programmierkonstrukte ermöglichen im Gegensatz zu diesen Modellen gerade die *simultane Verkoppelung* von Objekt und Metaberechnungen. Damit bietet PR ein parallelisiertes Modellierungskonzept für reflektive Systeme.
Es kann im Rahmen dieser Arbeit nicht ermittelt werden, inwieweit die hier entwickelte Konstruktion zur Lösung grundlegender und praktischer Probleme der Computational Reflection Forschung beitragen kann. Da jedoch die kenogrammatische Proemialrelation die selben strukturellen Probleme thematisiert wie die Computational Reflection, und darüber hinaus ein Konzept zur Modellierung der kausalen Verkoppelung paralleler Prozesse bietet, scheinen weitere fruchtbare Forschungen in dieser Richtung möglich[16].

10.4.2.2 Verallgemeinerung des Parallelitätsbegriffes

Die Definition des Proemialkombinators PR im vorherigen Abschnitt beruhte auf der physikalischen Verkoppelung paralleler Berechnungen. Dieser Modellierungsansatz soll hier zu einer kenogrammatisch fundierten Notation paralleler Prozesse ausgeweitet werden. Hierzu wird zunächst die Struktur paralleler Prozesse in dem verwendeten Modell untersucht.
Eine Berechnung $PR\ f\ g\ x\ y$ weist die Form der offenen Proemialrelation auf, wenn $g \equiv_z x$ und $f \not\equiv_z y$. Ist $g \equiv_z x$ und $f \equiv_z y$, handelt es sich um die geschlossene Form. Im Falle der autoproemiell geschlossenen Proemialrelation gilt $f \equiv_z g \equiv_z x \equiv_z y$.
Welche Art der Proemialrelation bei der parallelen Auswertung zweier Kombinatorausdrücke $(f\ x)$ und $(g\ y)$ vorliegt, kann nicht aus der Termstruktur sondern nur aus der Struktur von Zeigergleichheit ($\equiv_z$) und Zeigerverschiedenheit ($\not\equiv_z$) von f, x, g und y ermittelt werden.



Die $\equiv_z$ / $\not\equiv_z$–Struktur ist strukturisomorph zu der ϵ/ν–Struktur[17] von Kenogrammkomplexionen[18]. Aufgrund dieser Isomorphie kann die obige $\equiv_z$ / $\not\equiv_z$–Struktur von f, x, g, y als Kenogrammsequenz ○△△□ dargestellt werden. Da hier

[16] Bestimmte Probleme, wie z.B. das 'Interfacing' von Prozessen (z.B. innerhalb von Betriebssystemen und Benutzeroberflächen) lassen sich in einem sequentiellen CR–System (wie 3LISP) nicht zufriedenstellend lösen. Die vom Kombinator PR gewährleistete simultane Ausführung der beteiligten Prozesse, läßt ihn zur Implementierung solcher Systeme geeignet erscheinen.

[17] Vgl. 3.3.4.

[18] Eine strukturisomorphe Abbildung zwischen $\equiv_z$ / $\not\equiv_z$–Struktur und ϵ/ν–Struktur kann etwa durch den Isomorphismus $\mathcal{I} : (\equiv_z, \not\equiv_z) \longrightarrow (\epsilon, \nu)$ mit $\mathcal{I}(\equiv_z) = \epsilon$ und $\mathcal{I}(\not\equiv_z) = \nu$ definiert werden.

$x \equiv_z g$ und $f \not\equiv_z y$, entspricht ○△△□ der offenen Proemialrelation. Die Kenogrammsequenz ○△△○ bezeichnet die geschlossene Proemialrelation, da hier $x \equiv_z g$ und $f \equiv_z y$. Die Kenogrammsequenz ○○○○ kennzeichnet eine Situation, in der alle vier Argumente der Proemialrelation auf das selbe Objekt verweisen, diese somit autoproemiell geschlossen ist.
Auch die Sequenzen ○○○△ und ○△△△ bezeichnen proemielle Verhältnisse, da für sie ebenfalls $x \equiv_z g$ gilt. Die restlichen zehn der insgesamt fünfzehn Kenogrammsequenzen der Länge 4 bezeichnen jedoch Verhältnisse von Zeigergleichheit und Verschiedenheit, die nicht der Proemialrelation genügen (da $x \not\equiv_z g$ liegt kein Umtauschverhältnis vor).
Da die proemiellen Verhältnisse nur einen Teilbereich der kombinatorisch möglichen Kenogrammsequenzen abdecken, liegt es nahe, die Proemialrelation als Spezialfall eines Parallelitätsbegriffes zu verstehen, der alle möglichen $\equiv_z$ / $\not\equiv_z$–Strukturen zuläßt. Aufgrund der Isomorphie zwischen $\equiv_z$ / $\not\equiv_z$– und ϵ/ν–Struktur kann die Struktur der parallelen Berechnungen die durch den PR-Kombinator ermöglicht werden, von der Kenogrammatik beschrieben werden. Physikalische Verkoppelung und Wechselwirkung paralleler Prozesse kann dann allgemein mittels kenogrammatischer Operationen formal dargestellt werden.
Soll beispielsweise den beiden parallelen Berechnungen $(f\ x)$ und $(g\ y)$ mit der (proemiell geschlossenen) Struktur ○△△○ eine weitere Berechnung $(h\ z)$ mit der Strukur ○△ hinzugefügt werden, sind für die resultierende Gesamtberechnung $(f\ x)\|(g\ y)\|(h\ z)$ verschiedene ϵ/ν–Strukturen möglich:

f	x	g	y	h	z
ν	ϵ	ν	$?_1$	ν	
ν	ν	$?_2$	$?_5$		
ν	$?_3$	$?_6$			
$?_4$	$?_7$				
$?_8$					

Die entstehenden acht Freiheiten $?_i$ können, unter Berücksichtigung der Konsistenzbedingung (Gleichung 3.4, Seite 64) mit ϵ oder ν belegt werden. Die möglichen Kenogrammsequenzen sind dann:

$$\begin{aligned} ○△△○ \ @\ ○△ \ = \ \{&○△△○○△,\ ○△△○○□,\\ &○△△○△○,\ ○△△○△□,\\ &○△△○□○,\ ○△△○□△,\\ &○△△○□\star\}. \end{aligned}$$

Die indizierte Verkettung $@_i$ (vgl. 3.3.5) bestimmt einzelne Elemente dieser Menge. Sie kann also dazu verwendet werden, genau spezifizierte Verkoppelungen paralleler Prozesse zu bestimmen.

> **Beispiel:** Ein Erzeuger/Verbraucher–System werde durch zwei Berechnungen `(produce list)` und `(consume list)` modelliert. `produce` fügt `list` jeweils neue Elemente hinzu, `consume` arbeitet `list` sequentiell ab. Damit die beiden parallelen Berechnungen wie gewünscht zusammenwirken, müssen die von `produce` neu erzeugten Elemente von `list` auch der Verbraucherberechnung `consume` verfügbar sein. Dies ist nur möglich,

wenn beide Prozesse auf dem selben Objekt operieren. Die Kenogrammstruktur von (PR `produce consume list list`) muß also ○△□△ lauten. Als indizierte Verkettung der Sequenzen ○△ und ○△, die die Struktur der Berechnungen `(produce list)` und `(consume list)` darstellen, ergibt sich daher:

$$○△ \ @_{[(1,1,\nu),(2,2,\epsilon)]} \ ○△ = ○△□△.$$

Da die Kombinatormaschine alle Vorkommen einer Variablen x in einer Abstraktion $\lambda x.\cdots x\cdots x\cdots$ als Instanzen eines einzigen Objektes behandelt (node–sharing), unterschiedliche Variablen jedoch unterschiedlichen Objekten zuordnet, läßt sich die benötigte ϵ/ν–Struktur des Erzeuger/Verbraucher–Systems realisieren durch:

```
- run "(%l. PR produce consume l l) list";
```

10.4.3 Grenzen des Modells

Das im Vorhergehenden entwickelte operationale Modell der Proemialrelation beschreibt die simultane Verwendung eines gegebenen Objektes innerhalb verschiedener Bezugssysteme und Funktionalitäten. Wie gezeigt, erfüllt das Modell die in 3.1.2 eingeführte Bestimmung der Proemialrelation als ein simultanes Wechselverhältnis von Ordnungs- und Umtauschbeziehungen zwischen Objekten verschiedener logischer Stufen. In diesem Abschnitt soll gezeigt werden, welche Aspekte der Güntherschen Konzeption der Proemialrelation durch dieses Modell nicht erfasst werden.

Die simultane Auswertung der proemiell verkoppelten Prozesse wird durch den Schedulermechanismus sichergestellt. Gegenüber den einzelnen Prozessen ist der Scheduler ein Metasystem, daß die scheinbare Vielzahl autonomer Prozesse zu einem globalen (allerdings nicht-deterministischen) System zusammenfasst. Unabhängig davon, ob ein bestimmtes Objekt innerhalb eines Prozesses als Operator und innerhalb eines anderen als Operand fungiert, ist es einfach ein Datenobjekt, das von einem Programm (dem Scheduler und den von ihm kontrollierten Prozessen) bearbeitet wird. Diese globale Analyse des Kontrollflusses zeigt deutlich, daß der aus der Sicht der Prozesse beobachtete simultane Funktionswechsel Operator:Operand nur eine sekundäre Interpretation des eigentlichen Vorganges ist.
Daß das vorgestellte Modell die simultane Umtauschbeziehung der Proemialrelation abbildet, gilt somit nur, wenn es aus der Perspektive der lokalen Prozesse interpretiert wird. Das Gesamtsystem ist jedoch ein klassisches formales System, weshalb es ja auch in einer klassischen Programmiersprache formuliert werden kann.

Eine weitere Einschränkung betrifft den Begriff des *Objekts* und die *Selbigkeit* von Objekten. Die Selbigkeit eines Objektes ist im vorgeschlagenen Modell über die physikalische Gleichheit bzw. Zeigergleichheit definiert. D.h. ein Objekt ist nur dann ein und dasselbe, wenn es tatsächlich an der selben physikalischen Addresse des Hauptspeichers liegt. Objekte, die die *gleiche* Termstruktur aufweisen aber an verschiedenen Speicheradressen liegen, sind zwar *identisch* (bzgl. der semiotisch fundierten Termgleichheit), jedoch nicht selbig. Diese Differenzierung von Gleichheit und Selbigkeit entspricht dem Unterschied der LISP-Funktionen `eq` und `equal`: `eq` testet Zeigergleichheit, `equal` Termgleichheit.

Die Modellierung des simultanen Umtauschs eines selben Objektes innerhalb verschiedener Bezugssysteme stützt sich auf diesen Selbigkeitsbegriff: Das vorgegebene selbige Objekt kann von verschiedenen Prozessen simultan sowohl als Operator als auch als Operand einer Applikation *interpretiert* werden. Wie bereits erwähnt erscheint also die Proemialität hier nur als abgeleitete *a posteriori* Interpretation eines *a priori* gegebenen Objektes.
Günthers Idee der Proemialrelation beabsichtigt jedoch gerade, eine der dualistischen Aufspaltung Operator:Operand vorangehende Begrifflichkeit zu entwickeln: „*Wir nennen diese Verbindung zwischen Relator [Operator] und Relatum [Operand] das* Proemialverhältnis, *da es der symmetrischen Umtauschrelation und der Ordnungsrelation vorangeht und [...] ihre gemeinsame Grundlage bildet. [...] Dies ist so, weil das Proemialverhältnis jede Relation als solche konstituiert. Es definiert den Unterschied zwischen Relation und Einheit oder — was das gleiche ist — [...] wie der Unterschied zwischen Subjekt und Objekt.*“[19]
Im Gegensatz zu dem obigen Modell, das die selbigen Objekte zur Definition des simultanen Umtauschs voraussetzen muß, kann in Günthers Ansatz der Objektbegriff (und insbesondere die Selbigkeit) nicht vorausgesetzt werden, da er, im Gegensatz zu klassischen Kalkülen, auf eine a priori Unterscheidung Subjekt:Objekt verzichtet. Die Proemialrelation soll vielmehr gerade erst die Unterscheidbarkeit von Subjekt- und Objektkategorien ermöglichen. Die Selbigkeit von Objekten ergibt sich dann als ein *abgeleitetes* Phänomen und nicht als eine fundierende Kategorie.
In dem hier entwickelten Modell tauchen die transklassischen Aspekte der Proemialrelation, wie gezeigt, lediglich aus der Perspektive einer bestimmten Interpretation des Proemialkombinators PR als emergentes Oberflächenphänomen auf, sind jedoch nicht bereits der Architektur der Kombinatormaschine inhärent. Es kann daher eingewandt werden, das hier keine transklassische Modellierung, sondern lediglich eine bestimmte Applikation und Interpretatin einer klassischen Formalisierung vorliegt. Diese Einschränkung ergibt sich zwangsläufig, da das Modell ja im Sprachrahmen klassischer formaler Systeme und Programmiersprachen formuliert wurde.
Diese Problematik wurde bereits in 3.2 angesprochen. Dort wurde argumentiert, daß eine Formalisierung der Kenogrammatik sich nur dann von der Fundierung durch die klassische Semiotik ablösen kann, wenn sie als ein mehrstufiger Bootstrapprozess im Sinne der Selbstimplementierung von Programmiersprachen durch metazirkuläre Interpreter angelegt ist [Abe87].

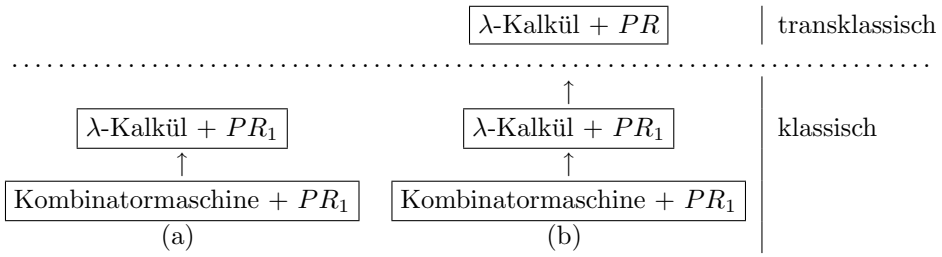


Abbildung 10.7: Bootstrapping der PR-Formalisierung

In Analogie zu dieser Argumentation läßt sich das hier implementierte Modell recht-

[19] [Gue80a1], S. 33.

fertigen. Im Rahmen einer Programmiersprache die den Proemialkombinator PR oder von ihm abgeleitete Konstrukte enthält (wie sie von dem hier entwickelten um PR erweiterten λ-Kalkül repräsentiert wird, vgl. Abbildung 10.7 (a)) ist es möglich, einen Interpreter eben dieser Sprache (d.h. einen metazirkulären Interpreter) zu programmieren. Diese Sprache würde dann den Proemialkombinator sowohl als abgeleitetes Programmierkonstrukt enthalten (Oberflächenphänomen), gleichzeitig wäre er jedoch auch Teil der zugrundeliegenden Sprache und somit in der Architektur des oberen Interpreters reflektiert. Die Modellierung der Proemialrelation wäre hier nicht mehr eine nachträgliche Interpretation vorgegebener klassischer Objekte, sondern könnte unmittelbar mittels der Konzepte von Ordnungs- und Umtauschrelation der Proemialrelation des unteren Interpreters beschrieben werden (b).

10.4.4 Ausblick

Der Kombinator PR ermöglicht zusammen mit kenogrammatischen Operationen die formale Beschreibung und Implementierung paralleler Prozesse unter genauer Spezifizierung der von mehreren Prozessen gleichzeitig verwendeten Objekten. Der Sprachrahmen funktionaler Programmiersprachen kann durch die Einführung dieser Konstrukte um die Komposition verkoppelter paralleler Prozesse erweitert werden. Aus dieser Erweiterung ergeben sich eine Fülle von Anwendungsmöglichkeiten, von denen hier nur die wichtigsten kurz angedeutet werden sollen.
Ein naheliegender Schritt wäre es, das vorgestellte Berechnungsmodell zu einer vollständigen Programmiersprache auszubauen oder in bestehende Systeme zu integrieren. Diese Programmiersprachen können dann zur Implementierung verkoppelter Parallelität (insbesondere der Proemialrelation) verwendet werden.
Mittels dieser um wenige Konstrukte erweiterten funktionalen Sprachen können dann formale Modelle von Prozeßkommunikation und –interaktion komplexer Systeme (z.B. Betriebssysteme) erstellt werden.
Ausser diesem klassischen Anwendungsfeld bieten sie aber auch die Möglichkeit, die Gültigkeit des hier entwickelten Modells der Proemialrelation anhand der Modellierung und Simulation proemieller, selbstreferentieller und verteilter Systeme zu verifizieren.
Sollte sich die Adäquatheit des Modells erweisen, müßte es möglich sein, transklassische Konzeptionen wie die Polykontexturale Logik oder die polykontexturale Arithmetik mittels dieser Programmiersprachen zu implementieren.
Mit diesem kurzen Überblick über mögliche anschließende Forschungsarbeiten wird die Darstellung der dynamischen Modellierung der Kenogrammatik beendet.

Anhang

Anhang A

Dokumentation der Implementierung

A.1 Implementierung der Kenogrammatik

A.1.1 Allgemeine Funktionen

```
(* Kapitel 4 Kenogrammatik *)
fun append l m = l @ m;

fun reduce f u nil = u
   |reduce f u (x::xs)=
        f x (reduce f u xs);

val flat = reduce append nil;

fun pair x y =(x,y);

fun allpairs xs ys=
     flat(map(fn x=> map (pair x) ys) xs);

exception Fromto;
fun fromto n m=
  if n>(m+1) then raise Fromto
  else if n=m+1 then nil
  else n::fromto (n+1) m;

val nlist = fromto 1;

fun nfirst n [] =raise Place
   |nfirst 0 _ = []
   |nfirst 1 (hd::tl)=[hd]
   |nfirst n (hd::tl)=hd::nfirst (n-1) tl;


fun member x [] = false
   |member x (hd::tl) = (x=hd) orelse member x tl;
```

```
fun combine a l=
         map (fn x => a::x) l;

fun remove x [] =[]
   |remove x (hd::tl) =
        if (x=hd) then remove x tl
        else hd::(remove x tl);

fun swap (x,y) = (y,x);

fun kmax ill = max (map max ill)
and max l = max1 0 l
and max1 n [] = n
   |max1 n (hd::tl)= if n>hd then max1 n tl
                     else max1 hd tl;

fun pos n l =nth(l,n-1);

fun replace item [] w = []
   |replace item (hd::tl) w =
        if (hd=item) then w::(replace item tl w)
        else hd::(replace item tl w);

fun rd []=[]
   |rd [x]=[x]
   |rd (x::xs) = x::rd(remove x xs);

fun nlistof 0 x = []
   |nlistof n x = x::nlistof (n-1) x;


fun nfirst n [] =raise Place
   |nfirst 1 (hd::tl)=[hd]
   |nfirst n (hd::tl)=hd::nfirst (n-1) tl;

fun fak 0=1
   |fak n= n * fak (n-1);

fun choose n k=
   (fak n) div ((fak k)* fak (n-k));

fun powers m n=
   if n=0 then 1
   else if n=1 then m
   else m*(powers m (n-1));
```

A.1.2 Normalformen

```
type keno = int;
type kseq = keno list;

fun tnf ks =
  let
    fun firstocc item list =
```

```
      let
        fun place1 item [] n = raise Place
           |place1 item (x::xs) n = if item=x then n
                                    else place1 item xs n+1;
      in
        place1 item list 1
      end;
    fun tnf1 [] res n k = res
       |tnf1 (hd::tl) res 1 k = tnf1 tl [1] 2 2
       |tnf1 (hd::tl) res n k =
          if member (pos n ks) (nfirst (n-1) ks)
          then tnf1 tl
                  (res@[pos (firstocc (pos n ks) ks) res])(n+1) k
          else tnf1 tl
                  (res@[k]) (n+1) (k+1);
  in
    tnf1 ks [] 1 1
  end;


  fun dnf ks =
  let
    fun count x []= 0
       |count x (y::ys)= (if x=y then 1 else 0)+count x ys;
  in
    flat (map (fn k=> nlistof (count k (tnf ks)) k)
              (rd (tnf ks)))
  end;


fun pnf ks = (nlistof (length ks - length(rd ks)) 1)@tnf(rd ks);
```

A.1.3 Äquivalenzklassen

```
fun teq a b = (tnf a = tnf b);

fun deq a b = (dnf a = dnf b);

fun peq a b = (pnf a = pnf b);


fun Pcard n = n;

fun sum from to f=
        if (from > to) then 0
        else (f from) + sum  ( from + 1) to f;

fun P (n,1) = 1
   |P (n,k) =
     if k>n then 0
     else if k=n then 1
     else P(n-1,k-1) + P(n-k,k);

fun Dcard n = sum 1 n (fn k => P(n,k));
```

```
fun S (n,1) = 1
   |S (n,k) =
     if k>n then 0
     else if k=n then 1
     else S(n-1,k-1) + k*S(n-1,k);

fun Tcard n = sum 1 n (fn k => S(n,k));


fun Pcontexture n =
   map (fn k => (nlistof (n-k) 1)@(nlist k))
       (nlist n);


fun allperms []=[]
   |allperms [x]=[[x]]
   |allperms [x,y]=[[x,y],[y,x]]
   |allperms l=
     let
      fun combine a l=
         map (fn x => a::x) l;
      fun remov x []=[]
         |remov x (y::ys)= if (x=y) then ys
                           else y::remov x ys;
     in
      flat ( map (fn a => combine a (allperms (remov a l)))
                 l)
     end;


fun allsums n 1=[[n]]
   |allsums n k=
     if (n=k) then [nlistof n 1]
     else
       flat(map (fn e => combine e (allsums (n-e) (k-1)))
                (nlist (n-k+1)));

fun allpartitions n k=
  let
    fun Exists f [] = false
       |Exists f (hd::tl)=
          if (f hd) then true
          else Exists f tl;
    fun remdups [] = []
       |remdups (hd::tl)=
          if Exists (fn x => (member x tl)) (allperms hd)
            then remdups tl
            else hd::(remdups tl);
  in
   remdups (allsums n k)
  end;
```

```
fun PDconcrete ks =
   map (fn p => flat (map (fn k => nlistof (pos k p) k)
                          (nlist (length (rd ks)))))
           (allpartitions (length ks) (length (rd ks)));

fun Dcontexture n =
   flat(map PDconcrete (Pcontexture n));


fun DTconcrete ks =
   rd(map (fn i => tnf i)
          (allperms ks));

fun Tcontexture n=
   flat(map DTconcrete (Dcontexture n));
```

A.1.4 ϵ/ν–Strukturen

```
datatype EN = E|N;

fun delta (i,j) z=
   if (pos i z) = (pos j z)
     then (i,j,E)
     else (i,j,N);


type enstruc = (int*int*EN) list list;

(* pairstructure n erzeugt die Struktur der m"oglichen Paare
   f"ur eine Sequenz der L"ange n *)
fun pairstructure n =
   map (fn j => map (fn i => (i,j))
                    (fromto 1 (j-1)))
       (fromto 1 n);

fun ENstructure z =
   map (fn trl => map (fn pair => delta pair z)
                      trl)
       (pairstructure (length z));


fun teq a b = (ENstructure a) = (ENstructure b);

exception Entoks;
fun ENtoKS enstruc =
  let
   fun entoks1 [] ks = ks
      |entoks1 ((f,s,en)::tl) ks =
        let
          val fir = pos f ks;
          val sec = if (length ks< s) then [] else pos s ks;
        in
         (if (en=E andalso sec=[])
```

```
          then entoks1 tl (ks@[fir])
          else if (en=E andalso member (hd fir) sec)
            then entoks1 tl (replace sec ks fir)
          else if (en=E andalso not(member (hd fir) sec))
            then raise Entoks
          else if (en=N andalso sec=[])
            then entoks1 tl (ks@[remove (hd fir)
                         (nlist ((kmax ks)+1:int))])
          else if (en=N andalso fir=sec)
            then raise Entoks
          else if (en=N andalso member (hd fir) sec)
            then entoks1 tl (replace sec ks
                                    (remove (hd fir) sec))
          else entoks1 tl ks)
        end;
  in
    (flat (entoks1 (flat enstruc) [[1]]))
  end;
```

A.1.5 Kenogrammatische Operationen

```
fun Tref ks = tnf(rev ks);
fun Dref ks = dnf(Tref ks);
fun Pref ks = pnf(Tref ks);


fun AG ks = length (rd ks);

fun EE (n,k) =
  let
   fun max  (x : int) y= if x>y then x else y;
   fun mkfg from to 0 = [[]]
      |mkfg from to step=
         flat(map (fn i => combine i (mkfg (i+1) to (step-1)))
             (fromto from (max from to)));
  in
   mkfg  1 (n+1) k
  end;

fun mappat pat template=
   map (fn x => pos x template) pat;

fun mkpats a b =
  let
   fun max (x:int) y= if x>=y then x
                else y;
   fun free n ([] : int list) = []
      |free n (hd::tl) =
          if hd<=n then hd::(free n tl)
          else [];
   fun possperms [] ag=[]
      |possperms [x] ag = [[x]]
      |possperms rest ag=
         flat(map (fn k=> combine k (possperms (remove k rest)
```

```
                                                  (max k (ag))))
                  (free (ag+1) rest ));
  in
   flat
     (map (fn e => possperms e (AG a))
          (EE (AG a,AG b)))
  end;

fun combinea item list=
   map (fn x=> item@x) list;

fun kconcat ks1 ks2=
   combinea ks1 (map (fn pat => mappat ks2 pat)
                     (mkpats ks1 ks2));


fun Dconcat a b = dnf(kconcat a b);
fun Pconcat a b = pnf(kconcat a b);


fun Ekard (n,k) =
  let
   fun max (x: int) y = if x>y then x else y;
   fun Xi from to 0 = 1
      |Xi from to step=
         sum from to (fn i => Xi (i+1) (max to (i+1)) (step-1));
  in
   Xi 1 (n+1) k
  end;

fun NN (a,b) =
  let
   val M = EE ((AG a),(AG b));
   fun e i =pos i M;
   fun gn [] = 0
      |gn (x::xs) = if (x>((AG a)+1)) then 1+(gn xs)
                    else gn xs;
  in
   sum 1 (Ekard(AG a,AG b))
       (fn i => (fak (length (e i))) div (fak(1+gn(e i))) )
  end;

fun collfits a [] rule = []
   |collfits a (b::bs) (rule as (x,y,en))=
       if ((en=E) andalso ((pos x a)=(pos (y) b)))
       then
         b::collfits a bs rule
       else if ((en=N) andalso ((pos x a)<>(pos (y) b)))
       then
         b::collfits a bs rule
       else collfits a bs rule;


(* andere Indizierung der Positionen
```

```
fun collfits a [] rule = []
   |collfits a (b::bs) (rule as (x,y,en))=
       if ((en=E) andalso ((pos x a)=(pos (y-(length a)) b)))
       then
         b::collfits a bs rule
       else if ((en=N) andalso ((pos x a)<>(pos (y-(length a)) b)))
       then
         b::collfits a bs rule
       else collfits a bs rule;
*)

fun mapvermat a bs []=bs
   |mapvermat a [] enstruc = []
   |mapvermat a bs (rule::rules)=
       mapvermat a (collfits a bs rule) rules;

fun kligate a b enstruc =
   combinea a
            (mapvermat a
                       (map (fn pat => mappat b pat)
                            (mkpats a b))
                       enstruc);
```

A.2 Implementierung der Kenoarithmetik

A.2.1 Protostruktur

```
(* Kapitel 5 Kenoarithmetik*)


exception Pis;
fun PIS (n,k) = if k=n then raise Pis
                else (n,k+1);
fun PTS0 (n,k) = (n+1,k);
fun PTS1 (n,k) = (n+1,k+1);
```

A.2.2 Deuterostruktur

```
fun firstocc item list =
              let
                fun place1 item [] n = raise Place
                   |place1 item (x::xs) n = if item=x then n
                                            else place1 item xs n+1;
              in
                place1 item list 1
              end;

fun nfirst n [] =raise Place
   |nfirst 0 _ = []
   |nfirst 1 (hd::tl)=[hd]
   |nfirst n (hd::tl)=hd::nfirst (n-1) tl;

fun forall [] p = true
   |forall (x::xs) p = if (p x)= false then false
```

```
                      else forall xs p;

exception Dis;
fun DIS D =
  if (forall D (fn x => x=1)) then raise Dis
  else
   let
     val m = sum 1 (length D) (fn i => (pos i D))
     val i = ((firstocc 1 D) - 1) handle Place => (length D)
     val pi = (pos i D)-1
     val u = m - (sum 1 (i-1) (fn k => (pos k D)))
     val j = u div pi
     val rest = u mod pi
     val news = map (fn x => pi) (fromto 1 (j-1)) ;
   in
     (nfirst (i-1) D) @ [pi] @ news @ (if rest=0 then []
                                       else [rest])
   end;



fun remnils []=[]
   |remnils (hd::tl) = if hd=[] then remnils tl
           else hd::(remnils tl);

exception Replace;
fun replacepos1 n [] x m= raise Replace
   |replacepos1 n (hd::tl) x m=
           if n=m then x::tl
           else hd::(replacepos1 n tl x (m+1));

fun DTS D =
   [((pos 1 D)+1)::(tl D)] @
   (remnils (map (fn i => if (pos i D) > (pos (i+1) D))
                          then replacepos (i+1) D ((pos (i+1) D)+1)
                          else [])
                 (fromto 1 ((length D)-1)))) @
   [D@[1]]

datatype 'a seq = Nil
                | Cons of 'a * (unit -> 'a seq);
```

A.2.3 Tritostruktur

```
exception Tis;
fun last l= hd(rev l);
fun TIS ts =
  let
   val n = length ts;
  in
   if (pos n ts) = n then raise Tis
   else if (last ts)<=(max (nfirst (n-1) ts))
     then (nfirst (n-1) ts) @ [1+(last ts)]
   else (nfirst (n-2) ts) @ [1+(pos (n-1) ts),1]
```

```
  end;

fun TTS ts =
   map (fn i => ts@[i])
       (fromto 1 ((AG ts)+1));



fun kmul [] b = [[]]
   |kmul a [] = [[]]
   |kmul a [1] = [a]
   |kmul [1] b = [b]
   |kmul a b =
     let
        fun makeEN a k [] = []
           |makeEN a k kyet=
             flat(map
                  (fn mem => map
                              (fn p => (((firstocc mem b)-1)*(length a)+p,
                                        p,
                                        if (k=mem) then E
                                        else N))

                              (nlist(length a)))

                   (rd kyet));

        fun kmul1 a nil used res = res
           |kmul1 a (hd::tl) used res =
              kmul1 a tl (hd::used)
                 (flat(map (fn x => kligate x a
                                        (makeEN a hd used))
                           res));
     in
        kmul1 a b [] [[]]
     end;
```

A.3 Implementierung der Morphogrammatik

A.3.1 Morphogrammatische Komplexionen

```
(* Kapitel 6 Morphogrammatik*)

val Q = Tcontexture(4);
exception Mg;
fun mg i = if i<1 orelse i>15 then raise Mg
           else pos i Q;

type 'a mat = 'a list list;

fun maufn m n=
  let
   fun combine it list= map (fn x=> it::x) list;
```

```
   fun maufn1 n []=[]
      |maufn1 0 l =[]
      |maufn1 1 l = map (fn x=> [x]) l
      |maufn1 p l =
         flat(map (fn x => combine x (maufn1 (p-1) (remove x l)))
                  l);
   fun remdups []=[]
      |remdups ([x,y]::tl)=
         if x=y then remdups tl
         else if member [y,x] tl
           then [x,y]::(remdups(remove [y,x] tl))
         else [x,y]::(remdups tl);
  in
   remdups(maufn1 n (nlist m))
  end;

fun matpos i j c=
   pos j (pos i c);

fun sort l=
  let
     exception Assoc;
     fun assoc n []= raise Assoc
        |assoc n ((k,pair)::tl)=
           if n=k then (k,pair)
           else assoc n tl;
  in
   map (fn k=> assoc k l)
       (nlist (length l))
  end;

fun k (i,j)=((j*(j-1)) div 2)-i+1;

fun subsystems n=
  sort(map (fn [i,j] => (k(i,j),[i,j]))
             (maufn n 2));

fun L (w: ''a mat)=
  let
   val n = length w;
  in
   tnf(flat (map (fn (k,[i,j])=> [matpos i i w,matpos i j w,
                                  matpos j i w,matpos j j w])
             (subsystems n)))
  end;

fun set (i,j,x,mat)=
  let
   fun replacepos n list w=
    let
     exception Replace;
     fun replacepos1 n [] x m= raise Replace
        |replacepos1 n (hd::tl) x m=
           if n=m then x::tl
```

```
          else hd::(replacepos1 n tl x (m+1));
    in
     replacepos1 n list w 1
    end;
  in
   replacepos i mat (replacepos j (pos i mat) x)
  end;

fun L_1 kseq =
  let
   exception Subsystems;
   val n = 0.5+sqrt(0.25+real((length kseq) div 2));
   val n = if real(floor n) = n then floor n
           else raise Subsystems;
   val subs = subsystems n;
   val mat = nlistof n (nlistof n 0);
   fun kstomm [] subs mat=mat
      |kstomm (ii::ij::ji::jj::restks)
              ((k,[i,j])::restpairs)
              mat=
         kstomm restks
                restpairs
                (set (i,i,ii,
                (set (i,j,ij,
                (set (j,i,ji,
                (set (j,j,jj, mat))))))))
  in
   kstomm kseq subs mat
  end;


fun TNFMM (w: ''a mat) = L_1(LL w);
```

A.3.2 Komposition und Dekomposition

```
type mg = kseq;
type mgchain = mg list;

fun makemg i j c=
   tnf [matpos i i c,matpos i j c,matpos j i c,matpos j j c];

fun decompose mm =
  let
   val n = length mm;
  in
   map (fn (k,[i,j])=> makemg i j mm)
       (subsystems n)
  end;


fun makeEN (k,ik,jk) subsystems=
   flat(map (fn (l,[il,jl])=>
               if il=ik then [((l-1)*4+1,1,E)]
```

```
          else if il=jk then [((l-1)*4+1,4,E)]
          else if jl=jk then [((l-1)*4+4,4,E)]
          else if jl=ik then [((l-1)*4+4,1,E)]
          else [])
            (nfirst (k-1) subsystems));

fun kligs x ys ENS=
   flat (map (fn y=> kligate y x ENS)
       ys);

fun compose1 [] (subs:(int*int list) list) res subsystems=res
   |compose1 (hd::tl) ((k,[ik,jk])::subs) res subsystems=
     (compose1 tl
               subs
               (kligs hd
                      res
                      (makeEN (k,ik,jk) subsystems))
               subsystems);

fun Kom mk =
  let
   exception Compose;
   val n = 0.5+sqrt(0.25+real(2*(length mk)));
   val n = if real(floor n) = n then floor n
           else raise Compose;
   val subsystems =subsystems n;
  in
   map (fn ks=> L_1 ks)
       (compose1 mk subsystems [[]] subsystems)
  end;
```

A.3.3 Reflektoren

```
fun mem xs x = member x xs;

fun difference s t =
  filter (not o mem t) s;

fun intersection s t=
   filter (mem s) t;

fun seteq a b =
   (difference a b)= [] andalso (difference b a) = [];

fun disjuncts subs n=
  let
   fun intersected sk sl =
      let
        val subsystems = subsystems n;
        val [ik,jk] = assoc sk subsystems;
        val [il,jl] = assoc sl subsystems;
      in
       (il=ik) orelse (il=jk) orelse (jk=jl) orelse (jl=ik)
      end;
```

```
   fun allintersectedwith si =
      remzeros(map (fn sj => if (intersected si sj) then sj
                             else 0)
                   (subs));

   fun allconnectedto sis =
      let
        val step = rd(flat(map allintersectedwith sis));
      in
        if (seteq sis step) then sis
        else allconnectedto step
      end;

   fun alldisjunctions [] =[]
      |alldisjunctions (hd::tl) =
        let
         val thisdisj = allconnectedto [hd];
         val rest = difference (hd::tl) thisdisj;
        in
         if rest=[] then [thisdisj]
         else thisdisj::(alldisjunctions rest)
        end;
  in
   alldisjunctions subs
  end;


fun reflectdisj disj mat=
 let
  val n = length mat;
  fun poke [] m = m
     |poke (si::tl) m =
        let
         val [i,j] = assoc si (subsystems n);
        in
   poke tl
         (set (i,i,pos i (pos i mat),
         set (i,j,pos j (pos i mat),
         set (j,i,pos i (pos j mat),
         set (j,j,pos j (pos j mat),m)))))
        end;

  fun split [] os = (os,[])
     |split (0::rest) os = split rest (0::os)
     |split x os = (os,x);

  val rhomat = poke disj (nlistof n (nlistof n 0));

  val (leados,rest) = split (flat rhomat) [];
  val (followos,revrho) = split (rev rest) [];
  val flatmat = flat mat;
  val flatrhomat = (leados @ revrho @ followos);
exception Nthcdr;
fun nthcdr n [] = raise Nthcdr
```

```
   |nthcdr 0 liste = liste
   |nthcdr n (hd::tl) = nthcdr (n-1) tl;

fun elements n m liste=
  nfirst (m-n+1) (nthcdr n liste);


fun mkmat n liste =
   map (fn z => elements (z*n) (z*n+n-1) liste)
       (fromto 0 (n-1));
 in
  mkmat n (map (fn i => if (pos i flatrhomat)=0 then (pos i flatmat)
                        else (pos i flatrhomat))
               (nlist (n*n)))
 end;


fun reflectall [] mat = mat
   |reflectall (disj::rest) mat =
      reflectall rest (reflectdisj disj mat);

fun r subslist mat=
   reflectall (disjuncts subslist (length mat)) mat;
```

A.4 Klassifikation von Q

```
(* Kapitel 7 Klassifikation des Operandensystems*)

datatype fc = F|C;

fun FCstructure hd =
  map (fn (k,[i,j]) => if (pos i hd)=(pos j hd) then C
            else F)
    (subsystems (length hd));


fun allFCs n =
  rd(map (fn ks => FCstructure ks)
    (Tcontexture n));


fun FCtype mg = if (pos 1 mg)=(pos 4 mg) then C
        else F;
fun FCtypes MK =
  map (fn mg => FCtype mg)
    MK;


fun exmm MK =
  if (member (FCtypes MK)
       (allFCs (floor(0.5+sqrt(0.25+
                       real(2*(length MK)))))))
   then true
```

```
  else false;
```

A.5 Graphentheoretische Analyse

```
(* Kapitel 8 Graphentheoretische Analyse*)

fun allof typ []=[]
   |allof typ(hd::tl)=
     if (typ hd) then hd::(allof typ tl)
     else allof typ tl;

fun Cmg [w11,_,_,w22] = (w11=w22);
fun Fmg [w11,_,_,w22] = (w11<>w22);


val Qc= allof Cmg Q;
val Qf= allof Fmg Q;

fun allMKs_of typ=
  let
     fun comb []= [[]]
        |comb [l]= map (fn x => [x]) l       (* Optimierung *)
        |comb (hd::tl)=
          flat(map (fn mg => combine mg (comb tl))
                   hd);
  in
   comb (map (fn fc => if fc=C then Qc
                       else Qf)
             typ)
  end;

fun Qfc n=
  map (fn fctyp => allMKs_of fctyp)
      (allFCs n);

fun apprefs mm [] = []
   |apprefs mm (r::rs) = (r mm)::(apprefs mm rs);

fun MKtype mk typfun=
   map (fn mg => typfun mg) mk;

fun mapsto typ reflist typfun=
 let
  fun applyrefs k (reflist : ('a -> 'a) list)=
   let
     fun apprefs mm [] = []
        |apprefs mm (r::rs) = (r mm)::(apprefs mm rs)
   in
    flat(map (fn mm => apprefs mm reflist)
             k)
   end;
  fun MKtype mk typfun=
   map (fn mg => typfun mg) mk;
```

```
 in
  rd(map (fn mm=> MKtype (decompose mm) typfun)
         (applyrefs (flat (map (fn mk=> Kom mk)
                               (allMKs_of typ)))
                    reflist))
 end;

val Rg=[r1,r2,r3,r12,r13,r23,r123];
val R_G=[r1,r2,r3,r12,r13,r23,r123];


fun analyze MKtypes reflist mgtype =
   map (fn strukt =>
         map (fn r => mapsto strukt [r] mgtype)
             reflist)
       MKtypes;

(* val FCanalyse = analyze (allFCs 3) Rg FCtype; *)

fun fcref r fctyp=
 let
  val mat= hd(Kom(map (fn fc =>
                                  if fc=C then [1,1,1,1]
                                  else [1,1,1,2])
                          fctyp));
  in
   FCtypes(decompose (r mat))
  end;

fun fcanalyse types reflist=
  map (fn fctyp =>
        map(fn r => fcref r fctyp)
           reflist)
      types;

val FCanalyse_abs =(fcanalyse (allFCs 3) R_G);


(* ********** G-H- Analyse ********** *)
  fun comb []= [[]]
     |comb [l]= map (fn x => [x]) l
     |comb (hd::tl)=
       flat(map (fn mg => combine mg (comb tl))
                hd);

fun allGHs (n)=
 let
  fun comb []= [[]]
     |comb [l]= map (fn x => [x]) l
     |comb (hd::tl)=
       flat(map (fn mg => combine mg (comb tl))
                hd);
 in
  comb (nlistof (choose n 2) [G,H])
```

```
 end;

fun Gmg [_,w12,w21,_]= w12<>w21;
fun Hmg [_,w12,w21,_]= w12=w21;

val Qg = allof Gmg Q;
val Qh = allof Hmg Q;


fun ghref r ghtyp=
 let
  val mat= hd(Kom(map (fn gh =>
                    if gh=G then [1,1,2,1]
                    else [1,1,1,1])
                  ghtyp));
  in
   GHtypes(decompose (r mat))
  end;

fun ghanalyse types reflist=
  map (fn ghtyp =>
        map(fn r => ghref r ghtyp)
           reflist)
      types;

val GHanalyse =ghanalyse (allGHs 3) R_G;


(* ************ KLOR-Analyse *********** *)
fun Kmg [w11,w12,w21,w22]= (w11<>w22) andalso (w12<>w21);
fun Lmg [w11,w12,w21,w22]= (w11<>w22) andalso (w12=w21);
fun Omg [w11,w12,w21,w22]= (w11=w22) andalso (w12=w21);
fun Rmg [w11,w12,w21,w22]= (w11=w22) andalso (w12<>w21);


val Qk = allof Kmg Q;
val Ql = allof Lmg Q;
val Qo = allof Omg Q;
val Qr = allof Rmg Q;

fun allKLORs(n) =
  flat(map comb
           (map (fn fcstr => map (fn fc =>if fc=F then [L,K]
                                          else [O,R])
                                 fcstr)
                (allFCs n) ));


fun klorref r klortyp=
 let
  val mat= hd(Kom(map (fn klor =>
                       if klor=K then [1,1,2,2]
                  else if klor=L then [1,1,1,2]
                  else if klor=O then [1,1,1,1]
```

```
               else                [1,1,2,1])
                  klortyp));
  in
   KLORtypes(decompose (r mat))
  end;

fun kloranalyse types reflist=
  map (fn klortyp =>
        map(fn r => klorref r klortyp)
           reflist)
      types;

val KLORanalyse =kloranalyse (allKLORs 3) R_G;


(* ************ Quellen und Senken ************* *)

val Qlll= comb [Ql,Ql,Ql];

val MMlll= flat(map (fn MKi => Kom(MKi))
               Qlll);

fun mgnr m=
 let
   exception Findplace
   val ns = fromto 1 15
   fun findplace m [] = raise Findplace
      |findplace m (n::ns)= if (m=mg n) then n
                            else findplace m ns;
 in
   findplace m ns
 end;

fun mknums mk=   map mgnr mk;

val MKanalyse =
map (fn MKi =>
        map(fn r => flat(rd(map(fn MKmat=> mknums(decompose(r MKmat)))
                            (Kom MKi))))
           R_G)
    Qlll;

fun makegraph qs []=[]
   |makegraph [] ms =[]
   |makegraph (q::qs) (m::ms)=
        (q,m)::makegraph qs ms;

val Glll = makegraph (map mknums Qlll) MKanalyse;


fun Minquell Graph =
  let
   fun remnils []=[]
      |remnils (hd::tl) = if hd=[] then remnils tl
```

```
                          else hd::(remnils tl);
   fun step [] Q = Q
      |step ((name,cons)::Graph) Q =
       step Graph (remnils
                   (map (fn q=> if (member q cons)
                                 andalso (q<>name) then []
                                else q)
                        (name::Q)));
  in
   step Graph []
  end;

val MinQlll = Minquell Glll;
```

A.6 Kombinatorische Analyse der Komposition

```
(*Kapitel 9 Kombinatorische Analyse der Polysemie*)

fun sum from to f=
   if (from > to) then 0
   else (f from) + sum  ( from + 1) to f;

fun S (n,1) = 1
   |S (n,k) =
     if k>n then 0
     else if k=n then 1
     else S(n-1,k-1) + k*S(n-1,k);


fun NF m = sum 1 m (fn k => S(m,k));


fun P (n,1) = 1
   |P (n,k) =
     if k>n then 0
     else if k=n then 1
     else P(n-1,k-1) + P(n-k,k);


fun GF m = sum 1 m (fn k => P(m,k));

fun fak 0=1
   |fak n= n * fak (n-1);

fun choose n k=
   (fak n) div ((fak k)* fak (n-k));

fun powers m n=
   if n=0 then 1
   else if n=1 then m
   else m*(powers m (n-1));
```

```
fun sigma n11 n12 n13 n22 n23 n24=
  let
   val l1=n11+n12+n13;
   val l2=n22+n23+n24;
  in
    ((fak l1) div ((fak n11)*(fak n12)*(fak n13)))
   * (powers 3 n12) *
    ((fak l2) div ((fak n22)*(fak n23)*(fak n24)))
   * (powers 4 n22) * (powers 5 n23)
  end;


fun max []=0
   |max [x]=x
   |max (x::xs)=
     let
      val restmax = max xs;
     in
      if (x > restmax) then x else restmax
     end;
fun k n= n;

fun R n11 n12 n13 n22 n23 n24=
  let
   val kJ = if (n11+n12+n13)=3 then 1
            else if (n22+n23+n24)=3 then 3
            else 2;
  in
   max (kJ::(map (fn (n,j,r) => if (n=0) then 0
                                else (k j)+r)
                  [(n11,1,0),(n12,1,1),(n13,1,2),
                   (n22,2,0),(n23,2,1),(n24,2,2)]))
  end;


fun M n11 n12 n13 n22 n23 n24=
  let
   val kJ = if (n11+n12+n13)=3 then 1
            else if (n22+n23+n24)=3 then 3
            else 2;
  in
   kJ + n12 + (2*n13) + n23 + (2*n24)
  end;


fun dr n2 n3=
  if n3<>0 then 2
  else if n2<>0 then 1
  else 0;


exception A
fun a1 (n1,n2,n3,m,j) =
   if (j=1) andalso
```

```
      ((m<(R n1 n2 n3 0 0 0)) orelse (m>(M n1 n2 n3 0 0 0)))
      then 0
   else if j=2 andalso
           ((m<(R 0 0 0 n1 n2 n3)) orelse (m>(M 0 0 0 n1 n2 n3)))
      then 0
   else if n1=n1+n2+n3 then 1
   else if n3<>0 then
    sum 0 (dr n2 n3)
         (fn q => (choose (m-(k j)-q) ((dr n2 n3)-q))
                 *((fak(dr n2 n3)) div (fak q))
                 *a1(n1+1,n2,n3-1,m-q,j))
   else
     sum 0 (dr n2 n3)
         (fn q => (choose (m-(k j)-q) ((dr n2 n3)-q))
                 *((fak(dr n2 n3)) div (fak q))
                 *a1(n1+1,n2-1,n3,m-q,j));

exception B;
fun a (n11,n12,n13,n22,n23,n24,m) =
 let
   val J = if (n11+n12+n13)=3 then 1
           else if (n22+n23+n24)=3 then 3
           else if (n11+n12+n13+n22+n23+n24)=3 then 2
           else raise A;
 in
   if (m<(R n11 n12 n13 n22 n23 n24))
       orelse (m>(M n11 n12 n13 n22 n23 n24))
      then 0
   else if J=1 then a1(n11,n12,n13,m,1)
   else if J=3 then a1(n22,n23,n24,m,2)
   else if n11=1 andalso n22=2 then 1
   else if (n12>0 orelse n13>0) then
     sum 0 (dr n12 n13)
         (fn q => (choose (m-(k 1)-q) ((dr n12 n13)-q))
                 *((fak(dr n12 n13)) div (fak q))
                 *a(n11+1,if n12=0 then 0 else n12-1,
                          if n13=0 then 0 else n13-1,
                    n22,n23,n24,m-q))
   else if (n24>0) then
     sum 0 (dr n23 n24)
         (fn q => (choose (m-(k 2)-q) ((dr n23 n24)-q))
                 *((fak(dr n22 n24)) div (fak q))
                 *a(n11,n12,n13,n22+1,n23,if n24=0 then 0
                                          else n24-1,m-q))
   else if (n23>0) then
     sum 0 (dr n23 n24)
         (fn q => (choose (m-(k 2)-q) ((dr n23 n24)-q))
                 *((fak(dr n22 n24)) div (fak q))
                 *a(n11,n12,n13,n22+1,if n23=0 then 0
                                      else n23-1,n24,m-q))
   else raise B
  end;
```

```
exception Fromto;
fun fromto n m =
  if n>m+1 then raise Fromto
  else if n=m+1 then nil
  else n::fromto (n+1) m;


fun MP n11 n12 n13 n22 n23 n24=
 let
  val alist =
       map (fn m => a(n11,n12,n13,n22,n23,n24,m))
           (fromto (R n11 n12 n13 n22 n23 n24)
                   (M n11 n12 n13 n22 n23 n24))
 in
  (map (fn x => (print x ; print " ")) alist;
  sum (R n11 n12 n13 n22 n23 n24) (M n11 n12 n13 n22 n23 n24)
      (fn m => a(n11,n12,n13,n22,n23,n24,m)))
 end;


(* Ende *)
```

A.7 Implementierung der Proemialrelation

A.7.1 Parsemechanismus

```
(**** ML Programs from the book

  ML for the Working Programmer
  by Lawrence C. Paulson, Computer Laboratory, University of Cambridge.
  (Cambridge University Press, 1991)

Copyright (C) 1991 by Cambridge University Press.
Permission to copy without fee is granted provided that this copyright
notice and the DISCLAIMER OF WARRANTY are included in any copy.

DISCLAIMER OF WARRANTY.  These programs are provided 'as is' without
warranty of any kind.  We make no warranties, express or implied, that the
programs are free of error, or are consistent with any particular standard
of merchantability, or that they will meet your requirements for any
particular application.  They should not be relied upon for solving a
problem whose incorrect solution could result in injury to a person or loss
of property.  If you do use the programs or functions in such a manner, it
is at your own risk.  The author and publisher disclaim all liability for
direct, incidental or consequential damages resulting from your use of
these programs or functions.
****)


(*** Basic library module.  From Chapter 9.  ***)

infix mem;
```

```
signature BASIC =
  sig
  exception Lookup
  exception Nth
  val minl : int list -> int
  val maxl : int list -> int
  val take : int * 'a list -> 'a list
  val drop : int * 'a list -> 'a list
  val nth : 'a list * int -> 'a
  val mem : ''a * ''a list -> bool
  val newmem : ''a * ''a list -> ''a list
  val lookup : (''a * 'b) list * ''a -> 'b
  val filter : ('a -> bool) -> 'a list -> 'a list
  val exists : ('a -> bool) -> 'a list -> bool
  val forall : ('a -> bool) -> 'a list -> bool
  val foldleft : ('a * 'b -> 'a) -> 'a * 'b list -> 'a
  val foldright : ('a * 'b -> 'b) -> 'a list * 'b -> 'b
  end;


functor BasicFUN() : BASIC =
  struct
  fun minl[m] : int = m
    | minl(m::n::ns) = if m<n  then  minl(m::ns)  else  minl(n::ns);

  fun maxl[m] : int = m
    | maxl(m::n::ns) = if m>n  then  maxl(m::ns)  else  maxl(n::ns);

  fun take (n, []) = []
    | take (n, x::xs) =  if n>0 then x::take(n-1,xs)
 else  [];

  fun drop (_, [])    = []
    | drop (n, x::xs) = if n>0 then drop (n-1, xs)
       else x::xs;

  exception Nth;
  fun nth (l,n) =    (*numbers the list elements [x0,x1,x2,...] *)
case drop(n,l) of [] => raise Nth
| x::_ => x;

  fun x mem []  =  false
    | x mem (y::l)  =  (x=y) orelse (x mem l);

  (*insertion into list if not already there*)
  fun newmem(x,xs) = if x mem xs then  xs   else  x::xs;

  exception Lookup;
  fun lookup ([], a) = raise Lookup
    | lookup ((x,y)::pairs, a) = if a=x then y else lookup(pairs, a);

  fun filter pred [] = []
    | filter pred (x::xs) =
if pred(x) then x :: filter pred xs
```

```
else  filter pred xs;

  fun exists pred []      = false
    | exists pred (x::xs) = (pred x)  orelse  exists pred xs;

  fun forall pred []      = true
    | forall pred (x::xs) = (pred x)  andalso  forall pred xs;

  fun foldleft f (e, [])    = e
    | foldleft f (e, x::xs) = foldleft f (f(e,x), xs);

  fun foldright f ([],    e) = e
    | foldright f (x::xs, e) = f(x, foldright f (xs,e));
end;


(*** Lexical Analysis -- Scanning.  From Chapter 9.  ***)

(*Formal parameter of LexicalFUN*)
signature KEYWORD =
  sig
  val alphas: string list
  and symbols: string list
  end;

(*Result signature of LexicalFUN*)
signature LEXICAL =
  sig
  datatype token = Id of string | Key of string
  val scan : string -> token list
  end;


(*All characters are covered except octal 0-41 (nul-space) and 177 (del),
  which are ignored. *)
functor LexicalFUN (structure Basic: BASIC
    and       Keyword: KEYWORD) : LEXICAL =
  struct
  local open Basic in
  datatype token = Key of string  |  Id of string;

  fun is_letter_or_digit c =
      "A"<=c andalso c<="Z" orelse
      "a"<=c andalso c<="z" orelse
      "0"<=c andalso c<="9";

  val specials = explode"!@#$%^&*()+-={}[]:\"|;'\\,./?`_~<>";

  (*scanning of an alphanumeric identifier or keyword*)
  fun alphanum (id, c::cs) =
if is_letter_or_digit c then  alphanum (id^c, cs)
else  (id, c::cs)
    | alphanum (id, []) = (id, []);
```

```
  fun tokenof a = if a mem Keyword.alphas  then  Key(a)  else  Id(a);

  (*scanning of a symbolic keyword*)
  fun symbolic (sy, c::cs) =
if sy mem Keyword.symbols orelse not (c mem specials)
        then  (sy, c::cs)
else  symbolic (sy^c, cs)
    | symbolic (sy, []) = (sy, []);

  fun scanning (toks, []) = rev toks    (*end of chars*)
    | scanning (toks, c::cs) =
if is_letter_or_digit c
then (*identifier or keyword*)
     let val (id, cs2) = alphanum(c, cs)
     in  scanning (tokenof id :: toks, cs2)
     end
else if c mem specials
then (*special symbol*)
     let val (sy, cs2) = symbolic(c, cs)
     in  scanning (Key sy :: toks, cs2)
     end
else (*spaces, line breaks, strange characters are ignored*)
     scanning (toks, cs);

  (*Scanning a list of characters into a list of tokens*)
  fun scan a = scanning([], explode a);
  end
  end;


(*** Parsing functionals.  From Chapter 9.  ***)

infix 5 --;
infix 3 >>;
infix 0 ||;

signature PARSE =
  sig
  exception SynError of string
  type token
  val reader: (token list -> 'a * 'b list) -> string -> 'a
  val -- : ('a -> 'b * 'c) * ('c -> 'd * 'e) -> 'a -> ('b * 'd) * 'e
  val >> : ('a -> 'b * 'c) * ('b -> 'd) -> 'a -> 'd * 'c
  val || : ('a -> 'b) * ('a -> 'b) -> 'a -> 'b
  val $  : string -> token list -> string * token list
  val empty : 'a -> 'b list * 'a
  val id : token list -> string * token list
  val infixes :
      (token list -> 'a * token list) * (string -> int) *
      (string -> 'a -> 'a -> 'a) -> token list -> 'a * token list
  val repeat : ('a -> 'b * 'a) -> 'a -> 'b list * 'a
  end;
```

```
functor ParseFUN (Lex: LEXICAL) : PARSE =
  struct
  type token = Lex.token;
  exception SynError of string;

  (*Phrase consisting of the keyword 'a' *)
  fun $a (Lex.Key b :: toks) =
        if a=b then (a,toks) else raise SynError a
    | $a _ = raise SynError "Symbol expected";

  (*Phrase consisting of an identifier*)
  fun id (Lex.Id a :: toks) = (a,toks)
    | id toks = raise SynError "Identifier expected";

  (*Application of f to the result of a phrase*)
  fun (ph>>f) toks =
      let val (x,toks2) = ph toks
      in  (f x, toks2)  end;

  (*Alternative phrases*)
  fun (ph1 || ph2) toks = ph1 toks   handle SynError _ => ph2 toks;

  (*Consecutive phrases*)
  fun (ph1 -- ph2) toks =
      let val (x,toks2) = ph1 toks
  val (y,toks3) = ph2 toks2
      in  ((x,y), toks3)  end;

  fun empty toks = ([],toks);

  (*Zero or more phrases*)
  fun repeat ph toks = (   ph -- repeat ph >> (op::)
                        || empty   ) toks;

  fun infixes (ph,prec_of,apply) =
    let fun over k toks = next k (ph toks)
        and next k (x, Lex.Key(a)::toks) =
              if prec_of a < k then (x, Lex.Key a :: toks)
              else next k ((over (prec_of a) >> apply a x) toks)
          | next k (x, toks) = (x, toks)
    in  over 0  end;

  fun reader ph a =   (*Scan and parse, checking that no tokens remain*)
 (case ph (Lex.scan a) of
      (x, []) => x
    | (_, _::_) => raise SynError "Extra characters in phrase");

  end;


(*** Pretty printing.  See Oppen (1980).  From Chapter 8.  ***)

signature PRETTY =
  sig
```

```
   type T
   val blo : int * T list -> T
   val str : string -> T
   val brk : int -> T
   val pr  : outstream * T * int -> unit
   end;


functor PrettyFUN () : PRETTY =
  struct
  datatype T =
      Block of T list * int * int  (*indentation, length*)
    | String of string
    | Break of int; (*length*)

  (*Add the lengths of the expressions until the next Break; if no Break then
    include "after", to account for text following this block. *)
  fun breakdist (Block(_,_,len)::sexps, after) = len + breakdist(sexps, after)
    | breakdist (String s :: sexps, after) = size s + breakdist (sexps, after)
    | breakdist (Break _ :: sexps, after) = 0
    | breakdist ([], after) = after;

  fun pr (os, sexp, margin) =
   let val space = ref margin

       fun blanks 0 = ()
         | blanks n = (output(os," ");  space := !space - 1;
                       blanks(n-1))

       fun newline () = (output(os,"\n");  space := margin)

       fun printing ([], _, _) = ()
 | printing (sexp::sexps, blockspace, after) =
  (case sexp of
       Block(bsexps,indent,len) =>
  printing(bsexps, !space-indent, breakdist(sexps,after))
     | String s => (output(os,s);   space := !space - size s)
     | Break len =>
 if len + breakdist(sexps,after) <= !space
 then blanks len
 else (newline();  blanks(margin-blockspace));
    printing (sexps, blockspace, after))
   in  printing([sexp], margin, 0);  newline()  end;

  fun length (Block(_,_,len)) = len
    | length (String s) = size s
    | length (Break len) = len;

  val str = String  and  brk = Break;

  fun blo (indent,sexps) =
    let fun sum([], k) = k
  | sum(sexp::sexps, k) = sum(sexps, length sexp + k)
    in  Block(sexps,indent, sum(sexps,0))  end;
```

```
  end;
```

A.7.2 λ-Term Parser

```
use "ParsePrint.ML";


(*** Lambda-terms ***)

signature LAMBDA =
  sig
  datatype term = Free  of string
                | Bound of string
                | Int of int
                | Op of string
                | Abs   of string*term
                | Apply of term*term
  val abstract: string -> term -> term
  val inst: (string * term) list -> term -> term
  end;


functor LambdaFUN(Basic: BASIC) : LAMBDA =
  struct
  local open Basic in
  datatype term = Free  of string
| Bound of string
| Int of int
| Op of string
| Abs   of string*term
| Apply of term*term;

  (*Convert occurrences of b to bound index i in a term*)
  fun abstract  b (Free a) = if a=b then  Bound(a)  else  Free a
    | abstract  b (Bound j) = Bound j
    | abstract  b (Abs(a,t)) = Abs(a, abstract b t)
    | abstract  b (Apply(t,u)) = Apply(abstract b t, abstract b u);



  (*Substitution for free variables*)
    fun member x [] = false
       |member x (y::ys) = (x=y) orelse (member x ys);

  fun isop x = member x ["I","K","S","B","C","Y","CONS","HD","TL","+",
                         "ADD","MUL","SUB","DIV",
"IF","EQ","AND","OR","NOT","PR","PAR","SET"];

  fun isnum str =
   let
    fun forall [] p = true
       |forall (x::xs) p = (p x) andalso (forall xs p);

   in
```

```
    forall (explode str)
           (fn ch => member ch ["0","1","2","3","4","5","6","7","8","9"])

   end;

  fun mkint "" num = num
     |mkint str num =
       let val dig =hd(explode str);
    val rest=implode(tl(explode str));
       in
mkint rest (10*num + (if dig="0" then 0
      else if dig="1" then 1
      else if dig="2" then 2
      else if dig="3" then 3
      else if dig="4" then 4
      else if dig="5" then 5
      else if dig="6" then 6
      else if dig="7" then 7
      else if dig="8" then 8
      else 9))
end;

  fun inst env (Free a) = if (isnum a) then Int(mkint a 0)
  else if (isop a) then Op(a)
  else (inst env (lookup(env,a))
                handle Lookup => Free a)
    | inst env (Bound i) = Bound i
    | inst env (Abs(a,t)) = Abs(a, inst env t)
    | inst env (Apply(t1,t2)) = Apply(inst env t1, inst env t2);
  end
  end;


(*** Parsing of lambda terms ***)
signature PARSELAM =
  sig
  type term
  val abslist: string list * term -> term
  val applylist: term * term list -> term
  val read: string -> term
  end;

functor ParseLamFUN (structure Parse: PARSE
     and       Lambda: LAMBDA) : PARSELAM =
  struct
  local open Parse  open Lambda  in
  type term = term;

  (*Abstraction over several free variables*)
  fun abslist([],    t) = t
    | abslist(b::bs, t) = Abs(b, abstract  b (abslist(bs, t)));

  (*Application of t to several terms*)
  fun applylist(t, []) = t
```

```
    | applylist(t, u::us) = applylist(Apply(t,u), us);

  fun makelambda ((((_,b),bs),_),t) = abslist(b::bs,t)

  (*term/atom distinction prevents left recursion; grammar is ambiguous*)
  fun term toks =
   (   $"%" -- id -- repeat id -- $"." -- term >> makelambda
     || atom -- repeat atom >> applylist
    ) toks
  and atom toks =
    (   id >> Free
     || $"(" -- term -- $")" >> (#2 o #1)
    ) toks;
  val read = reader term;
  end
  end;
```

```
(******** SHORT DEMONSTRATIONS ********)

structure Basic = BasicFUN();
structure LamKey =
    struct val alphas = []
           and symbols = ["(", ")", "'", "->"]
    end;
structure LamLex = LexicalFUN (structure Basic=Basic and Keyword=LamKey);
structure Parse = ParseFUN(LamLex);


structure Lambda = LambdaFUN(Basic);
structure ParseLam = ParseLamFUN (structure Parse=Parse and Lambda=Lambda);

open Basic;  open Lambda;


val stdenv = map  (fn (a,b) => (a, ParseLam.read b))
[    (*booleans*)
 ("true", "%x y.x"),           ("false",  "%x y.y"),
 ("if", "%p x y. p x y"),
     (*ordered pairs*)
 ("pair", "%x y f.f x y"),
 ("fst", "%p.p true"),         ("snd", "%p.p false"),
     (*natural numbers*)
 ("suc", "%n f x. n f (f x)"),
 ("iszero", "%n. n (%x.false) true"),
 ("0", "%f x. x"),             ("1", "suc 0"),
 ("2", "suc 1"),               ("3", "suc 2"),
 ("4", "suc 3"),               ("5", "suc 4"),
 ("6", "suc 5"),               ("7", "suc 6"),
 ("8", "suc 7"),               ("9", "suc 8"),
```

```
 ("add",  "%m n f x. m f (n f x)"),
 ("mult", "%m n f. m (n f)"),
 ("expt", "%m n f x. n m f x"),
 ("prefn", "%f p. pair (f (fst p)) (fst p)"),
 ("pre",  "%n f x. snd (n (prefn f) (pair x x))"),
 ("sub",  "%m n. n pre m"),
      (*lists*)
 ("nil",  "%z.z"),
 ("cons", "%x y. pair false (pair x y)"),
 ("null", "fst"),
 ("hd", "%z. fst(snd z)"),     ("tl", "%z. snd(snd z)"),
     (*recursion for call-by-name*)
 ("YN", "%f. (%x.f(x x))(%x.f(x x))"),
 ("fact", "YN (%g n. if (iszero n) 1 (mult n (g (pre n))))"),
 ("append", "YN (%g z w. if (null z) w (cons (hd z) (g (tl z) w)))"),
 ("inflist", "YN (%z. cons MORE z)"),
     (*recursion for call-by-value*)
 ("YV", "%f. (%x.f(%y.x x y)) (%x.f(%y.x x y))"),
 ("factV", "YV (%g n. (if (iszero n) (%y.1) (%y.mult n (g (pre n))))y)") ];


fun r a = inst stdenv (ParseLam.read a);
```

A.7.3 Compiler λ-Kalkül $\Rightarrow$ Kombinatorlogik

```
(* Compiler  Lambda -> Kombinatorlogik *)

datatype comb = I|K|S|B|C|Y|
                CONS|HD|TL|
                PLUS|MINUS|TIMES|DIV|
                IF|EQ|AND|OR|NOT|
                WAIT|WAIT1|COPY|
                PR|PAR|SET|
                DEF of string;

datatype value = int of int|
                 re of real|
                 bool of bool;

datatype snode = satom of value
                |scomb of comb
                |sapp of (snode*snode);


exception Doublebind
fun abs x (Free y) = Apply(Op "K",(Free y))
   |abs x (Bound y) = if x=y then Op("I") else Apply(Op "K",(Bound y))
   |abs x (Int y) = Apply(Op "K",Int y)
   |abs x (Op y) = Apply(Op "K",Op y)
   |abs x (Abs(y,body)) = abs x (abs y body)
   |abs x (Apply(a,b)) =
         Apply(Apply(Op "S",abs x a),
                    (abs x b));
```

```
fun mkComb "I" = I
   |mkComb "K" = K
   |mkComb "S" = S
   |mkComb "B" = B
   |mkComb "C" = C
   |mkComb "Y" = Y
   |mkComb "CONS" = CONS
   |mkComb "HD" = HD
   |mkComb "TL" = TL
   |mkComb "ADD" = PLUS
   |mkComb "SUB" = MINUS
   |mkComb "MUL" = TIMES
   |mkComb "DIV" = DIV
   |mkComb "IF" = IF
   |mkComb "EQ" = EQ
   |mkComb "AND" = AND
   |mkComb "OR" = OR
   |mkComb "NOT" = NOT
   |mkComb "PR" = PR
   |mkComb str = DEF str;
```

```
exception Compile;
fun c (Free a) = sapp(scomb(DEF a),satom(int 0))
   |c (Bound a) = raise Compile
   |c (Int a) = satom(int a)
   |c (Op k) = scomb(mkComb k)
   |c (Apply(a,b)) = sapp(c a,c b)
   |c (Abs(x,body)) = c (abs x body);
```

```
fun opt (sapp(sapp(scomb S,sapp(scomb K,e)),scomb I)) = (e : snode)
   |opt (sapp(sapp(scomb S,sapp(scomb K,e1)),sapp(scomb K,e2))) =
       sapp(scomb K,sapp(e1,e2))
   |opt (sapp(sapp(scomb S,sapp(scomb K,e1)),e2)) =
       sapp(sapp(scomb B,e1),e2)
   |opt (sapp(sapp(scomb S,e1),sapp(scomb K,e2))) =
       sapp(sapp(scomb C,e1),e2)
   |opt (sapp(e1,e2)) = sapp(opt e1,opt e2)
   |opt x = x;

fun ropt x =
 let
  val y = opt x;
 in
  if y=x then x
  else ropt y
```

```
 end;


 fun k2string I = "I"
   |k2string K = "K"
   |k2string S = "S"
   |k2string B = "B"
   |k2string C = "C"
   |k2string Y = "Y"
   |k2string CONS = "CONS"
   |k2string HD = "HD"
   |k2string TL = "TL"
   |k2string PLUS = "PLUS"
   |k2string MINUS = "MINUS"
   |k2string TIMES = "TIMES"
   |k2string DIV = "DIV"
   |k2string IF = "IF"
   |k2string EQ = "EQ"
   |k2string AND = "AND"
   |k2string OR = "OR"
   |k2string NOT = "NOT"
   |k2string WAIT = "WAIT"
   |k2string WAIT1 = "WAIT1"
   |k2string COPY = "COPY"
   |k2string PR = "PR"
   |k2string PAR = "SET"
   |k2string (DEF(name)) = name;

fun isapp (sapp _) = true
   |isapp _ = false;

fun show (satom (int n)) = makestring n
   |show (satom (re x)) =makestring x
   |show (satom (bool w)) = makestring w
   |show (scomb k) = k2string k
   |show (sapp (rator,rand)) = (show rator)^" "^(if (isapp rand)
then "("^(show rand)^")"
else (show rand));
```

A.7.4 Die parallelisierte Kombinatormaschine

```
(* Parallele Kombinatormaschine + Implementierung der PR*)

fun last l = hd(rev l);
fun remove x [] = []
   |remove x (hd::tl) =
if x = hd then remove x tl
        else hd::(remove x tl);

(* ***************** *)
datatype Emark = Eval|Busy|Ready;

datatype node = atom of (value * Emark ref * (node ref list) ref)
              |comb of (comb  * Emark ref * (node ref list) ref)
```

```
              |app of  ((node ref * node ref) * Emark ref *
                                        (node ref list) ref);

fun alloc snode =
 let
   fun allocate (satom(x)) = atom(x,ref Ready, ref [])
      |allocate (scomb(com)) = comb(com,ref Ready, ref [])
      |allocate (sapp(a,b)) = app((ref (allocate a),ref (allocate b)),
                                  ref Eval, ref [])
 in
   ref(allocate snode)
 end;

fun dealloc (ref (atom(x,_,_))) = satom(x)
   |dealloc (ref (comb(k,_,_))) = scomb(k)
   |dealloc (ref (app((a,b),_,_))) = sapp(dealloc a,dealloc b);



fun is_atom (atom(_)) = true
   |is_atom (comb(_)) = false
   |is_atom (app(_))  = false;

fun is_app (atom(_)) = false
   |is_app (comb(_)) = false
   |is_app (app(_))  = true;

fun copy (ref(atom(x,ref Emark,ref wq))) =
        ref(atom(x,ref Emark,ref wq))
   |copy (ref(comb(x,ref Emark,ref wq))) =
        ref(comb(x,ref Emark,ref wq))
   |copy (ref(app((rator,rand),ref Emark,ref wq))) =
        ref(app((copy rator,copy rand),ref Emark,ref wq));

fun equal (ref(atom(x,_,_))) (ref(atom(y,_,_))) = (x=y)
   |equal (ref(comb(x,_,_))) (ref(comb(y,_,_))) = (x=y)
   |equal (ref(app((xrator,xrand),_,_)))
          (ref(app((yrator,yrand),_,_))) =
        (equal xrator yrator) andalso (equal xrand yrand)
   |equal _ _ = false;


fun get_mark (ref(atom(_,m,_))) = m
   |get_mark (ref(comb(_,m,_))) = m
   |get_mark (ref( app(_,m,_))) = m;

fun get_q (ref(atom(_,_,q))) = q
   |get_q (ref(comb(_,_,q))) = q
   |get_q (ref( app(_,_,q))) = q;

fun set_q (ref(atom(_,_,q))) x = q:=x
   |set_q (ref(comb(_,_,q))) x = q:=x
   |set_q (ref( app(_,_,q))) x = q:=x;
```

```
val ENV = ref [] : (string * node ref) list ref;
fun define name value =
ENV := (name, alloc(ropt(c(r value)))::(!ENV);


exception Lookup;
fun lookup name =
let
 fun lookup1 name [] = raise Lookup
    |lookup1 name ((x,y)::rest) = if name=x then y
                                  else (lookup1 name rest)
in
 lookup1 name (!ENV)
end;

fun spine (ref(atom(a,m,q))) stack = (atom(a,m,q),stack)
   |spine (ref(comb(c,m,q))) stack = (comb(c,m,q),stack)
   |spine (node as (ref(app((l,r),_,_)))) stack = spine l (node::stack);

val Tasks = ref [] : node ref list ref;
val Taskcounter = ref 0;

fun newTask task =
   (Taskcounter := 1+(!Taskcounter);
    Tasks := (task::(! Tasks)));

fun remTask task =
   Tasks := (remove task (! Tasks));

fun make_wait node =
 let
  val (k,(w::_)) = spine node [];
  val ref(app((ref rator,rand),_,q)) = w;
 in
  w := app((ref(app((ref(comb(WAIT,ref Eval,ref [])),
                    ref rator),ref Eval,q)),
            rand),ref Eval,q)
 end;

  fun make_unwait node =
   let
    val (_,w::_) = spine node [];
    val ref(app((ref c,ref k),_,_)) = w;
    fun iswait (app(_,_,_)) = false
       |iswait (atom(_,_,_)) = false
       |iswait (comb(c,_,_)) = if (c=WAIT orelse c=WAIT1)
       then true else false;
   in
    if iswait(c)
      then w := k
      else ()
   end;

fun wakeup waitQ =
```

```
  (map (fn task => (make_unwait task))
       (! waitQ);
   waitQ := []);


fun subEval (root,node) =
 let
  val emark = get_mark node;
  val wq = get_q node;
 in
  if (! emark = Ready) then ()
  else if (! emark = Busy) then
   (make_wait root;
    wq := root::(! wq))
  else
   (make_wait root;
    emark := Busy;
    wq := root::(!wq);
    newTask node)
  end;

fun parEval (root,node) =
 let
  val emark = get_mark node;
  val wq = get_q node;
 in
  if (! emark = Ready) then ()
  else if (! emark = Busy) then
   (make_wait root;
    wq := root::(! wq))
  else
   (emark := Busy;
    newTask node)
  end;


fun apply (WAIT,(node as ref(app((_,x),m,q)))::_) =
        node := app((ref(comb(WAIT1,ref Ready,ref [])),
                     x),m,q)
   |apply (WAIT1,(node as ref(app((_,x),m,q)))::_) =
        node := app((ref(comb(WAIT,ref Ready,ref [])),
                     x),m,q)
   |apply (I,(node as ref(app((_,ref x),_,ref q)))::_) =
        (node := x; set_q node q)
   |apply (K,ref(app((_,ref x),_,ref q))::(node as ref(app(_,_,_)))::_) =
        (node := x; set_q node q)
   |apply (S,(ref(app((_,x),_,_)))::(ref(app((_,y),_,_)))
              ::(node as (ref(app((_,z),m,q))))::_) =
        node := app((ref(app((x,z),ref Eval,q)),
                     ref(app((y,z),ref Eval,q))),
                    ref Eval,q)
   |apply (B,(ref(app((_,x),_,_)))::(ref(app((_,y),_,_)))
              ::(node as (ref(app((_,z),m,q))))::_) =
node := app((x,ref (app((y,z),ref Eval,q))),ref Eval,q)
```

```
   |apply (C,(ref(app((_,x),_,_)))::(ref(app((_,y),_,_)))
              ::(node as (ref(app((_,z),m,q))))::_) =
node := app((ref(app((x,z),ref Eval,q)),y),ref Eval,q)

   |apply (Y,(node as ref(app((_,f),m,q)))::_) =
        node := app((f,node),ref Eval,q)
   |apply (DEF(name),(node as ref(app((_,_),_,_)))::_) =
        node := !(copy(lookup name))
   |apply (PLUS,ref(app((_,ref(atom(int x,_,_))),_,_))::(node as
                ref(app((_,ref(atom(int y,_,_))),_,q)))::_) =
        node := atom(int(x+y),ref Ready,q)
   |apply (PLUS,(stack as ref(app((_,x),_,_))::
                          ref(app((_,y),_,_))::_)) =
        (subEval (last stack,x);
         subEval (last stack,y); ())
   |apply (MINUS,ref(app((_,ref(atom(int x,_,_))),_,_))::(node as
                 ref(app((_,ref(atom(int y,_,_))),_,q)))::_) =
        node := atom(int(x-y),ref Ready,q)
   |apply (MINUS,(stack as ref(app((_,x),_,_))::
                          ref(app((_,y),_,_))::_)) =
        (subEval (last stack,x);
         subEval (last stack,y); ())
   |apply (TIMES,ref(app((_,ref(atom(int x,_,_))),_,_))::(node as
                 ref(app((_,ref(atom(int y,_,_))),_,q)))::_) =
        node := atom(int(x*y),ref Ready,q)
   |apply (TIMES,(stack as ref(app((_,x),_,_))::
                          ref(app((_,y),_,_))::_)) =
        (subEval (last stack,x);
         subEval (last stack,y); ())
   |apply (DIV,ref(app((_,ref(atom(int x,_,_))),_,_))::(node as
                ref(app((_,ref(atom(int y,_,_))),_,q)))::_) =
        node := atom(int(x div y),ref Ready,q)
   |apply (DIV,(stack as ref(app((_,x),_,_))::
                          ref(app((_,y),_,_))::_)) =
        (subEval (last stack,x);
         subEval (last stack,y); ())
   |apply (EQ,(stack as ref(app((_,x),_,_))::(node as
                          ref(app((_,y),_,q)))::_)) =
        if (!(get_mark x)) = Ready andalso
           (!(get_mark y)) = Ready
          then node := atom(bool(equal x y),ref Ready,q)
        else
          (subEval (last stack,x);
          subEval (last stack,y); ())
   |apply (IF,(ref(app((_,ref(atom(bool test,_,_))),_,_)))::
              (ref(app((_,x),_,_)))::(node as (ref(app((_,y),_,_))))::_) =
        if test then node := !x
        else node := !y
   |apply (IF,(stack as (ref(app((_,test),_,_))::
              ref(app((_,x),_,_))::(node as ref(app((_,y),_,q)))::_))) =
        subEval (last stack,test)
   |apply (CONS,_) = ()
   |apply (COPY,(node as ref(app((_,x),_,_)))::_) =
        node := !(copy x)
```

```
   |apply (PR, (stack as ( ref(app((_,f),_,_))::
                          ref(app((_,g),_,_))::
                          ref(app((_,xf),_,_))::(node as
                         (ref(app((_,xg),_,q))))::_))) =
        let
         val first =  ref(app((f,xf),ref Eval,ref []));
         val second = ref(app((g,xg),ref Eval,ref []));
        in
         (node := app((ref(app((ref(comb(CONS,ref Ready,ref [])),
                               first),ref Ready,ref [])),
                      second),ref Ready,q);
          parEval (last stack, first);
          parEval (last stack, second))
  end
   |apply _ = ();


fun step node =
   let
    val (c,stack) = (spine node []);
   in
    if is_atom c then ()
    else let val comb(k,_,_)= c
         in apply (k,stack)
         end
   end;

fun evalstep (node as ref(atom(_,Emark,WQ))) =
(wakeup WQ;
 remTask node;
 Emark := Ready)
   |evalstep (node as ref(comb(_,Emark,WQ))) =
(wakeup WQ;
 remTask node;
 Emark := Ready)
   |evalstep (node as ref(app((ref rator,ref rand),Emark,WQ))) =
     let
      val old = copy node
     in
      (step node;
       if ((equal old node) orelse
           (!Emark = Ready) orelse
           (not (is_app (!node)))) (* Evaluation beendet ? *)
        then (wakeup WQ;           (* Wartende Prozesse reaktivieren *)
              remTask node;        (* Prozess entfernen *)
              Emark := Ready)      (* Knoten ist vollstndig evaluiert *)
        else ())                   (* der Scheduler kann die Evaluation
                                      fortsetzen *)
     end;

val Seed = ref 4.34793435219781;
val Steps = ref 0;

fun Scheduler () =
```

```
 let
  fun rnd () =
   let
    val x = (! Seed)* 1475.37697;
    val res = x-real(floor x);
   in
    (Seed := res; res)
   end;
  fun intrand n = floor(rnd()*(real n));
 in
  if length(!Tasks) = 0 then ()
  else
     (evalstep (nth(!Tasks,intrand (length (!Tasks))));
      Steps := 1+(!Steps);
      Scheduler())
 end;


fun eval node =
 (Steps := 0; Taskcounter := 0;
  Tasks := [];
  newTask node;
  Scheduler() handle _ => ();
  (show(dealloc node),!Steps,!Taskcounter));

fun run str = eval(alloc(c(r str)));

fun s () =
 let
  fun rnd () =
   let
    val x = (! Seed)* 1475.37697;
    val res = x-real(floor x);
   in
    (Seed := res; res)
   end;
  fun intrand n = floor(rnd()*(real n));
 in
  if length(!Tasks) = 0 then ""
  else
     (evalstep (nth(!Tasks,intrand (length (!Tasks))));
      show(dealloc (last (!Tasks))))
 end;

fun rs str=
  (Tasks := [];
   newTask(alloc(ropt(c(r str))));
   s());
```

Literaturverzeichnis

[Abe87] Abelson, H., Sussman, G.J., Sussman, J.: *Structure and Interpretation of Computer Programs*. Cambridge, Massachusetts, The MIT Press, 1987.

[And65] Andrew, A.M. und Foerster, H.v.: *Table of Stirling numbers of the second kind*. BCL-Report No.6, Urbana, 1965.

[Ass65] Asser, G.: *Einführung in die mathematische Logik*. Teil I, Zürich u. Frankfurt a.M., Verlag Harri Deutsch, 1965.

[Bac85] Bachmann, H.: *Der Weg der mathematischen Grundlagenforschung*. Frankfurt a.M., Verlag Peter Lang, 1985.

[Bar80] Barendregt, H.P.: *The Lambda-calculus. Its Syntax and Semantics*. North-Holland, 1980.

[Bas93] Bashford, S., Kaehr, R.: *Implementierung eines Tableau-Beweisers für Polykontexturale Logiken*. Arbeitsbericht Nr. 4 des Forschungsprojektes „Theorie komplexer biologischer Systeme“, Ruhr Universität Bochum, 1993.

[BCL76] Wilson, K.L. (Hg.): *BCL Publications, The Collected Works of the Biological Computer Laboratory*. Blueprint Corp., Peoria, Illinois, USA, 1976.

[Blo85] Bloch, E.: *Subjekt — Objekt. Erläuterungen zu Hegel*. Frankfurt a.M., Suhrkamp Verlag, 1985.

[Bro72] Spencer-Brown, G.: *The Laws of Form*. New York, Julian Press, 1972. (Deutsche Übersetzung in Vorbereitung, Suhrkamp1993).

[Cas92] Castella, J.: *Kontextur — Différance — Kenogramm, Dekonstruktive Bemerkungen zur Symbol–Subsymbol–Debatte*. IKS-Berichte, Institut für Kybernetik und Systemtheorie, TU Dresden, 1992.

[Cas93] Castella, J.: *Polykontexturalitätstheorie und Philosophische Probleme der KI-Forschung*. Arbeitsbericht Nr. 3 des Forschungsprojektes „Theorie komplexer biologischer Systeme“, Ruhr Universität Bochum, 1993.

[Chu51] Church, A.: *The Calculi of Lambda–Conversion*. Princeton University Press, 1951.

[Cur69] Curry, H.B., Feys, R.: *Combinatory Logic*. Amsterdam, North-Holland, 1969.

[Dav92] Davies, A.J.T.: *An Introduction to Functional Programming Systems Using Haskell*. Cambridge Computer Science Texts 27, Cambridge University Press, 1992.

[Dil88] Diller, A.: *Compiling Functional Languages*. New York, John Wiley & Sons, 1988.

[Dit82] Ditterich, J.: *Logikwechsel und Theorie selbstreferentieller Systeme*. In: Hombach, D.(Hg.): *Zeta 01. Zukunft als Gegenwart*. (S.120-155), Berlin, Verlag Rotation, 1982.

[Dit79] Ditterich, J., Kaehr, R.: *Einübung in eine andere Lektüre. Diagramm einer Rekonstruktion der Güntherschen Theorie der Negativsprachen*. In: Philosophisches Jahrbuch, 86. Jg., 2. Halbband, Freiburg und München, S. 385-408, 1979.

[Dit90] Ditterich, J.: *Selbstreferentielle Modellierungen: Biologie — Kybernetik. Kategorientheoretische Untersuchungen zur second order cybernetics und ein polykontexturales Modell kognitiver Systeme*. Arbeitsbericht Nr. 2 des Forschungsprojektes „Theorie komplexer biologischer Systeme“, Ruhr Universität Bochum, 1993.

[Fla69] Flachsmeyer, J.: *Kombinatorik. Eine Einführung in die mengentheoretische Denkweise*. Berlin, DVW, 1969.

[Foe70] Von Foerster, H., Günther, G.: *The Logical Structure of Evolution and Emanation*. In: Annals of the New York Academy of Science, S. 874-891, 1970.

[Foe75] Von Foerster, H., Howe, R.H.: *Introductory Comments to Francisco Varela's Calculus for Self-reference*. In: Int. J. General Systems 1975, Vol.2 (S.1–3), 1975.

[Foe76] Von Foerster, H.: *Objects: Tokens for (Eigen)-behaviors*. In: ASC Cybernetics Forum VIII (3,4), S. 91-96, 1976.

[Fre86] Frege, G.: *Funktion, Begriff, Bedeutung. 5 logische Studien*. Göttingen, Vandenhoeck und Ruprecht, 1986.

[Geb88] Gebser, J.: *Ursprung und Gegenwart*. 3Bd.,München, dtv, 1988.

[Gue33] Günther, G.: *Grundzüge einer neuen Theorie des Denkens in Hegels Logik*. Hamburg, Felix Meiner Verlag, 1978.

[Gue67] Günther, G.: *Time, Timeless Logic and Self-referential Systems*. In: Annals of the New York Academy of Science 1967, New York, 1967.

[Gue78] Günther, G.: *Idee und Grundriß einer nicht-aristotelischen Logik. Die Idee und ihre philosophischen Voraussetzungen*. 2. Auflage, Hamburg, Verlag Felix Meiner, 1978.

[Gue80] Günther, G.: *Beiträge zur Grundlegung einer operationsfähigen Dialektik*. Bd. 1-3, Hamburg, Verlag Felix Meiner, 1976-1980.

[Gue80a] Günther, G.: *Cognition and Volition*. In: [Gue80] Bd.2.

[Gue80a1] Günther, G.: *Erkennen und Wollen.* Übersetzung von [Gue80a] durch Helletsberger, G. In: *Gotthard Günther und die Folgen.* Klagenfurt.

[Gue80b] Günther, G.: *Cybernetic Ontology and Transjunctional Operations.* In: [Gue80] Bd.1.

[Gue80b1] Günther, G.: *Das metaphysische Problem einer Formalisierung der transzendentaldialektischen Logik.* In: [Gue80] Bd.1.

[Gue80c] Günther, G.: *Logik, Zeit, Emanation und Evolution.* In: [Gue80] Bd.3.

[Har91] Van Harmelen, Frank: *Meta-level Inference Systems.* London, Pitman, 1991.

[Heg86] Hegel, G.W.F.: *Wissenschaft der Logik I und II.* Frankfurt a.M., Suhrkamp Verlag, 1986.

[Hei59] Heisenberg, W.: *Physik und Philosophie.* Frankfurt a. M., Ullstein Materialien, 1959.

[Hei91] Heise-Dinnebier, S.: *Analyse der Morphogrammatik von Gotthard Günther.* TU Berlin, Fachbereich Informatik, Dipl., 1991.

[Hel90] Hellingrath, B. et al. (Hg.): *Reader zur Ringvorlesung Radikaler Konstruktivismus.* Forschungsberichte des Fachbereichs Informatik der Universität Dortmund Nr. 288, (S. 15-35), 1990.

[Hen80] Henderson, P.: *Functional Programming: Application and Implementation.* New York, Prentice Hall, 1980.

[Hom90] Hombach, D.: *Die Drift der Erkenntnis. Zur Theorie selbstmodifizierter Systeme bei Gödel, Hegel und Freud.* München, Raben Verlag, 1990.

[Hou88] Houben, G. und Nitsch, F.: *Entwicklung einer Programmierumgebung zur Behandlung polykontexturaler Systeme.* Dipl. Arbeit, Universität d. Bundeswehr, FB Informatik, München, 1988.

[Jeg73] Jeger, M.: *Einführung in die Kombinatorik.* Bd. 1 u. 2, Stuttgart, Klett Verlag, 1973.

[Kae74] Kaehr, R., Seehusen, J., Thomas, G.: *Deskriptive Morphogrammatik.* FEoLL-GmbH, Paderborn, 1974.

[Kae78] Kaehr, R.: *Materialien zur Formalisierung der dialektischen Logik und der Morphogrammatik 1973-1975.* In: [Gue78], Anhang.

[Kae80] Kaehr, R.: *Neue Tendenzen in der KI-Forschung. Metakritische Untersuchungen über den Stellenwert der Logik in der neueren Künstlichen-Intelligenz-Forschung..* Stiftung Warentest, Berlin, 1980 (Wiederabdruck in: *Gotthard Günther und die Folgen.* Klagenfurter Beiträge, Heft 22 , 1988.

[Kae81] Kaehr, R.: *Das graphematische Problem einer Formalisierung der transklassischen Logik Gotthard Günthers.* In: Beyer, W.R.(Hg.): *Die Logik des Wissens und das Problem der Erziehung.* Hamburg, Verlag Felix Meiner, 1981.

[Kae82] Kaehr, R.: *Einschreiben in Zukunft.* In: Hombach, D.(Hg.): *Zeta 01. Zukunft als Gegenwart.* Berlin, Verlag Rotation, 1982.

[Kae90] Kaehr, R.: *Kalküle für Selbstreferentialität oder selbstreferentielle Kalküle.* In: [Hel90], 1990.

[Kae92] Kaehr, R.: *Disseminatorik: Zur Logik der 'Second Order Cybernetics'. Von den 'Laws of Form' zur Logik derReflexionsform.* In: *Kalkül und Form.* In Vorbereitung, Frankfurt a. M., Suhrkamp 1993.

[Kae93] Kaehr, R.: *Dekonstruktion der Tekno-Logik. Hinführungen zur Graphematik.* Arbeitsbericht Nr. 5 des Forschungsprojektes „Theorie komplexer biologischer Systeme", Ruhr Universität Bochum, 1993.

[Kor26] Korzybski, A.: *Time-Binding. The General Theory.* Englewood, New Jersey, Institute of General Semantics, 1926.

[Kro86] Kronthaler, E.: *Grundlegung einer Mathematik der Qualitäten.* Frankfurt a.M., Peter Lang Verlag, 1986.

[Lan64] Landin, P.J.: *The Mechanical Evaluation of Expressions.* The Computer Journal, Bd.6, S.308–320, 1964.

[Lor78] Lorenzen, P., Lorenz, K.: *Dialogische Logik.* Darmstadt, 1978.

[Luk70] Łukasiewicz, J.: *Selected Logical Papers.* Amsterdam, North-Holland, 1970.

[Mae88] Maes, P., Nardi, D.: *Meta-Level Architectures and Reflection.* Amsterdam, North–Holland, 1988.

[Mah92] Mahler, Th.: *Kombinatorische Analyse der Polysemie, Untersuchungen zu Morphogrammatik und Polykontexturaler Logik.* IKS-Berichte, Institut für Kybernetik und Systemtheorie, TU Dresden, 1992.

[Mah93] Mahler, Th.: *Morphogrammatik in Darstellung,Analyse, Implementierung und Applikation.* Arbeitsbericht Nr. 1 des Forschungsprojektes „Theorie komplexer biologischer Systeme", Ruhr Universität Bochum, 1993.

[Mat85] Maturana, H.: *Erkennen: Die Organisation und Verkörperung von Wirklichkeit.* Vieweg Verlag, 1985.

[Mat87] Maturana, H., Varela, F.: *Der Baum der Erkenntnis.* München, Scherz Verlag, 1987.

[Na64] Na, H.S.H., Foerster, H.v., Günther, G.: *On structural analysis of many valued logic.* Department of Electrical Engineering, University of Illinois, 1964.

[Nie88] Niegel, W.: *Introduction to Polycontexturality.* Ms, Universität d. Bundeswehr, FB Informatik, München, 1988.

[Pau91] Paulson, L.C.: *ML for the Working Programmer.* University of Cambridge, Computer Laboratory, Cambridge University Press, 1991.

[Pet71] Petrov, J.A.: *Logische Probleme der Realisierbarkeits- und Unendlichkeitsbegriffe*. Berlin, Akademie Verlag, 1971.

[Pfa88] Pfalzgraf, J.: *Zur Formalisierung polykontexturaler Logiksysteme*. ESG, München, 1988.

[Qui75] Quine, W.V.: *Ontologische Relativität und andere Schriften*. Stuttgart, Reclam, 1975.

[Sap61] Sapir, E.: *Die Sprache*. München, 1961.

[Sch67a] Schadach, D.J.: *A classification of mappings between finite sets and some applications*. BCL-Report No. 2.2, Urbana, 1967.

[Sch67b] Schadach, D.J.: *A system of Equivalence Relations and generalized Arithmetic*. BCL-Report No. 4.1, Urbana, 1967 .

[Sch67c] Schadach, D.J., Günther, G.: *Natural Numbers and Many-valued Systems*. University of Illinois, Department of Electrical Engineering, 1967 .

[Sch87] Schmidt, S.J.: *Der Diskurs des radikalen Konstruktivismus*. Frankfurt a.M., Suhrkamp Verlag, 1987.

[Smi82] Smith, B.C.: *Reflection and Semantics in a Procedural Language*. LCS Technical Report TR–272, MIT, Massachusets, 1982.

[Tho85] Thomas, G.G.: *Introduction to Kenogrammatics*. In: Proceedings of the 13^{th} Winterschool on Abstract Analysis, Section of Topology, Serie II numero 11-1985, Palermo, 1985.

[Tur79a] Turner, D.A.: *SASL Language Manual*. St. Andrews University Dep. of Computer Science, Report CS/79/3, 1979.

[Tur79b] Turner, D.A.: *A New Implementation Technique for Applicative Languages*. Software Practise and Experience Bd. 9, 1979.

[Var75] Varela, F.J.: *A Calculus for Self-reference*.In: Int. J. General System 1975, Vol.2 (S. 5–24), 1975.

[Var76] Varela, F.J.: *The Ground For A Closed Logic*. BCL-Report No. 3.5,Urbana, 1976.

[Var92] Varela, F.J., Thompson, E., Rosch, E.: *Der mittlere Weg der Erkenntnis*. München, Scherz Verlag, 1992.

[Wei63] Von Weizsäcker, C.F.: *Zum Weltbild der Physik*. Stuttgart, Hirzel Verlag, 1963.

[Wei73] Von Weizsäcker, C.F.: *Classical and Quantum Descriptions*. In: Mehra, J. (Hg.):*The Physicists Conception of Nature*. Dordrecht, Reidel, 1973.

[Wei77] Von Weizsäcker, C.F.: *Der Garten des Menschlichen. Beiträge zur geschichtlichen Anthropologie*. München, C. Hanser Verlag, 1977.

[Whi25] Whitehead, A.N., Russell, B.: *Principia Mathematica.* 2. Auflage, Cambridge, University Press, 1925.

[Who63] Whorf, B.L.: *Sprache — Denken — Wirklichkeit. Beiträge zur Metalinguistik und Sprachphilosophie.* Hamburg, Rowohlt Verlag, 1963.

[Wit87] Wittgenstein, L.: *Tractatus logico-philosophicus. Logisch-philosophische Abhandlung.* Frankfurt a. M., Suhrkamp Verlag, 1987.

[Wir86] Wirth, N.: *Algorithmen und Datenstrukturen.* Stuttgart, Teubner Verlag, 1986.

[Yes70] Yessenin-Volpin, A.S.: *The ultra-intuitionistic criticism and the anti-traditional program for foundations of mathematics.* In: *Intuitionism and proof theory.* (S. 3–45), Amsterdam, North-Holland, 1970.

Index

Seitenzahlen in *Schrägdruck* verweisen auf Definitionen.

D

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

U

V

W

Y

Z