Ein Arbeitsplatz zum rechnerunterstützten handschriftlichen Rechnen mit mathematischen Formeln

Dissertation
zur Erlangung des Grades
des Doktors der Naturwissenschaften
der Mathematisch-Naturwissenschaftlichen Fakultät
der Universität des Saarlandes
von

Reiner Marzinkewitsch

Saarbrücken
1990

Tag des Kolloquiums:	
Dekan:	Prof. Dr. H.-G. Zimmer
Berichterstatter:	Prof. Dr. G. Hotz
	Prof. Dr. W.-J. Paul

Inhaltsverzeichnis

Einleitung

Mit dem Aufkommen der Personalcomputer in den 70er Jahren begann eine neue Ära der Rechnernutzung. War der Zugriff auf Rechenleistung bis dahin auf relativ wenige Spezialisten beschränkt, konnten sich nun breitere Benutzerschichten ihren eigenen Rechner leisten.

Hand in Hand mit dieser Ausweitung des Benutzerkreises ging eine Ausweitung der Anwendungen. Die sinkenden Preise für Rechenleistung machten es möglich, Rechner nicht nur für einzelne Spezialaufgaben einzusetzen, sondern auch lästige Routinearbeiten auf ihn abzuwälzen: Terminplanung, Textverarbeitung, Buchführung. Die Rechnerbenutzung wurde Bestandteil des Arbeitsalltages, der Rechner ein Arbeitswerkzeug wie einige Jahrzehnte vorher das Telefon.

Neben den sinkenden Hardwarepreisen waren an dieser Karriere die Programme beteiligt, die den Einsatz der PC's erst lohnend werden ließen. Am Beispiel der Tabellenkalkulationsprogramme wollen wir drei wesentliche Eigenschaften benutzerfreundlicher Programme nennen:

- Die Benutzeroberfläche ist problemorientiert. Der Anwender braucht sein Problem nicht in eine andere, für den Rechner geeignetere Darstellung zu transformieren; die Transformation wird dem Rechner überlassen. Da der Benutzer mit seiner gewohnten Sehweise weiterarbeiten kann, verlangt ihm der Rechner keine überflüssige Leistung ab, die Aufmerksamkeit des Benutzers bleibt dem eigentlichen Problem voll erhalten.

- Die Benutzeroberfläche suggeriert die Bedienung. Weil die Problemdarstellung an der gewohnten Sehweise des Benutzers orientiert ist, findet dieser sich auf Anhieb mit der Bedienung zurecht. Erste Schritte lassen sich fast immer ohne einen Blick in das Handbuch machen.

- Die Benutzeroberfläche ist so flexibel gestaltet, daß sie sich — zumindest in Grenzen — auf die Bedürfnisse des Benutzers zuschneiden läßt

Die Bedeutung der puren Funktionalität tritt fast etwas hinter diese Äußerlichkeiten zurück: Tabellenkalkulationsprogramme können im wesentlichen Ausdrücke auswerten, Zahlenkolonnen verrechnen und Diagramme zeichnen. Also machen nicht ausgefeilte Funktionen, sondern eine dem Benutzer naheliegende Bedienoberfläche den Witz bei dieser Art von Programmen aus.

Zwei Aspekte der gewohnten Arbeitsweise lassen sich 'naturbedingt' in Rechnerprogrammen nur schwer berücksichtigen:

- Die zwangsläufig sequentielle Eingabe per Tastatur verbietet jegliche Zweidimensionalität der Eingabe. Zweidimensionale Objekte müssen deshalb durch eine künstliche lineare Ordnung sequentialisiert werden.

- Handschriftliches Arbeiten bereitet Rechnern wegen des hohen für die Schrifterkennung zu treibenden Aufwandes Schwierigkeiten.

Gerade das zweidimensionale handschriftliche Arbeiten ist aber in vielen Bereichen die naheliegende Bedienoberfläche. Dies darf man nicht auf den bloßen Aspekt der Bequemlichkeit verkürzen. Man 'denkt mit dem Kugelschreiber'. Die herkömmlichen Notationen sind wesentlich effizienter als die Eingabeformate der entsprechenden Rechenprogramme und durch den Gebrauch hat sich auch das Denken den gewohnten Notationen angepaßt.

Wir stellen also fest, daß Rechner sich schwertun, gerade eine der effizientesten Arbeitsweisen, das handschriftliche Arbeiten mit zweidimensionalen Diagrammen, Formeln und Skizzen, zu unterstützen.

0.1 Aufgabenstellung

Die vorliegende Arbeit schildert die wesentlichen Aspekte eines Projektes, dem die Aufgabe zugrundelag, ein Gesamtsystem von der Art einer 'Werkbank' für eine repräsentative Anwendung, das symbolische Arbeiten mit mathematischen Formeln, prototypisch zu realisieren.

Bei diesem Anwendungsgebiet macht sich das Fehlen einer natürlichen Benutzeroberfläche besonders schmerzlich bemerkbar. Die Programmpakete zur symbolischen Formelmanipulation verlangen eine linearisierte Schreibweise der zu bearbeitenden Formeln. Gerade mathematische Notationen leben von ihrer Zweidimensionalität. Man verdeutliche sich hierzu die Schreibweisen von Brüchen, Integralen, Matrizen etc.

Weil man diese Schreibweisen aufgeben muß, lassen die existierenden Computeralgebrasysteme einen Großteil an Attraktivität vermissen. Zwar bieten Computeralgebrasysteme bereits heute weitreichende Unterstützung beim Rechnen:

- Auswerten einfacher Ausdrücke

- Symbolische Integration

- Symbolische Differentiation

- Lösen von Gleichungen über Polynomen

- Polynomarithmetik, Faktorisieren von Polynomen

- Rechnen mit Matrizen

- Bestimmung von Eigenwerten, der Determinante und der Spur einer Matrix

- Darüber hinaus sind Computeralgebrasysteme in der Regel benutzerspezifisch erweiterbar.

Das ungewohnte und unübersichtliche Eingabeformat der Computeralgabrasysteme führt jedoch dazu, daß sie als Werkzeuge wenig genutzt werden. Man beobachtet deshalb eine Diskrepanz zwischen der weitreichenden Unterstützung beim Rechnen, die durch Algebrasysteme geboten wird und der schlechten Akzeptanz seitens der Benutzer.

Deshalb folgt das hier beschriebene Projekt der Philosophie, den Gesichtspunkt der Ergonomie durchgängig vorrangig zu berücksichtigen. Neben der naheliegenden und effizienten Bedienung erscheint uns Echtzeitverhalten im Frage– und Antwortrhythmus besonders wichtig zu sein, um dem Benutzer die sofortige Kontrolle seiner Aktivitäten zu ermöglichen ("WYSIWYG"–Prinzip).

Die Betonung des Echtzeitverhaltens zwingt zu einem pragmatischen Vorgehen bei der Lösung bekanntermaßen schwieriger Teilaufgaben, wie der Zeichenerkennung oder der Formelerkennung: Hier sind wir bereit, Erkennungsfehler in Kauf zu nehmen, solange die Fehlerhäufigkeit tolerierbar bleibt (Erkennungsrate $> 90\%$).

Wir halten diese Konzession für vertretbar, da der Benutzer die Ausgaben der Programme kontrollieren und korrigieren kann und Interpretationsaufgaben wie Zeichenerkennung und Formelerkennung immer fehlerbehaftet sein werden, so daß die Kontrolle dem Benutzer in keinem Fall erspart werden kann. Solange die Fehlerfrequenz nicht störend wird, scheint der gewählte Ansatz deshalb akzeptabel.

Das Gesamtsystem soll nicht nur die handschriftliche *Eingabe* von Formeln erlauben. Möglichst viele Operationen sollen mit dem Stift auf möglichst naheliegende Weise durchführbar sein. Zusätzliche Aktionen (Knopfdrücken etc.) sollen möglichst vermieden werden.

Innerhalb des Projektes entwickelte sich die Zeichenerkennung mit neuronalen Methoden zu einem Teilbereich mit einiger Eigendynamik. In der einschlägigen Literatur wurden bisher keine Erfahrungen mit der Erkennung von Handschrift durch neuronale Netze innerhalb einer konkreten Anwendung publiziert. Die erfolgreiche Verwendung neuronaler Methoden zu Echtzeiterkennung von Blockschrift verleiht dem Projekt deshalb einen besonderen Akzent.

0.2 Stand der Forschung

Der Wunsch, Rechner handschriftlich zu bedienen, ist nicht neu. Zahlreiche Publikationen sind zu diesem Thema erschienen. Es existieren bereits Personalcomputer, die die Eingabe von Zeichen mit einem Stift erlauben [**Lin**].

Ende der sechziger Jahre wurden im Zusammenhang mit dem Aufkommen der ersten Computeralgebrasysteme Versuche unternommen, den Benutzern Schnittstellen zur handschriftlichen Eingabe zur Verfügung zu stellen.

Aus verschiedenen Gründen führten diese Ansätze nicht zu dem gewünschten Ziel. Wir erläutern dies am Beispiel der wohl bekanntesten Entwicklung in dieser Richtung, Andersons 'Sophisticated Scratchpad' [**And**], das aus einer Kombination von Blockschrifterkennung [**Gro**], zweidimensionaler Syntaxanalyse, Algebrasystem und einer Verwaltung für bereits erhaltene Ergebnisse bestand.

An diesem System läßt sich gut sehen, daß die Schwierigkeit beim Entwurf einer Oberfläche zur handschriftlichen Eingabe in ein Algebrasystem nicht nur im Erreichen der bloßen Funktionalität (handschriftliche Eingabe, Erkennung, Berechnung, Ausgabe), sondern im Entwurf eines Gesamtsystemes liegt, das sich effizient und naheliegend bedienen läßt.

Gerade die Gewährung der beim Arbeiten gewohnten Freiheit stellt den Entwickler der Oberfläche vor große Schwierigkeiten: Mehrdeutigkeiten müssen aufgelöst und Ungenauigkeiten der Schreibweise verarbeitet werden. Andererseits läßt sich die Akzeptanz einer Schnittstelle nur erreichen, wenn diese Freiheit gewährt wird.

Der von Anderson verwendete Blockschrifterkennungsalgorithmus erforderte ein Training des Benutzers, da die entscheidungsorientierte Erkennungsstufe nicht flexibel an die Schrift des Benutzers anzupassen war. Die Syntaxanalyse erfolgte top-down und verlangte, daß das signifikanteste Symbol einer Formel oder Teilformel am weitesten links steht. Dies führt bei flüssigem Schreiben häufig zu Fehlinterpretationen der eingegebenen Formeln (siehe hierzu Abschnitt 2.4). Die Unterstützung des Arbeitsablaufes erfolgte nur rudimentär. In seinem Wesen als bloße Kombination von Mustererkennung, Strukturanalyse und Auswertung angelegt, unterstützte das System von Anderson das handschriftliche Arbeiten nicht ausreichend, um Akzeptanz zu erzielen.

Die Aufgabe, den Benutzern von Softwarepaketen die gewohnte handschriftliche Eingabe zu ermöglichen, ist also bislang immer noch nicht gelöst.

0.3 Weitere Anwendungen

Das Konzept der handschriftlichen Eingabe erscheint auch bei anderen Anwendungen vielversprechend. Dies ist gerade in Bereichen der Fall, in denen Anwender eine eingebürgerte handschriftliche Notation, die nur schwer in ein eindimensionales Format konvertierbar ist, auf dem Rechner benutzen wollen.

Insbesondere chemische Strukturformeln und technische Diagramme (Schaltpläne, Ablaufdiagramme etc.) scheinen ebenfalls lohnende Anwendungsgebiete für das rechnerunterstützte, handschriftliche Arbeiten zu sein. Zum Beispiel könnten Anfragen an Datenbanken, in denen chemische Verbindungen gespeichert werden, durch eine handgeschriebene Strukturformel statt mit Hilfe der heute üblichen Standardnomenklaturen erfolgen.

Die Problematik der Analyse dieser Strukturen ist jedoch anders geartet als bei der Analyse mathematischer Formeln. Die Segmentierung zusammenhängender Diagramme ist wesentlich schwieriger als die Segmentierung der Blockschriftzeichen, aus denen mathematische Formeln bestehen. (Das Segmentierungsproblem ist bei handgezeichneten Diagrammen bis heute nicht gelöst.) Andererseits sind Diagramme nur schwach strukturiert. In der Regel ist die Berechnung des Zusammenhangsgraphen völlig ausreichend, um die Bedeutung von technischen Diagrammen zu erschließen. Mathematische Formeln können im Gegensatz dazu sehr stark strukturiert sein.

Während der Entwicklung des Prototypen zum handschriftlichen, rechnerunterstützten Rechnen wurden auch Versuche durchgeführt **[Wei]**, die unterschiedliche Verfahren zur Segmentierung großer, von Hand erstellter Objekte untersuchten. Diese Versuche führten bisher nicht zu einem befriedigenden Ergebnis und werden deshalb in dieser Arbeit nicht geschildert.

0.4 Aufbau des Projektes

Aus den bei der Lösung der Gesamtaufgabe anfallenden Teilaufgaben

- Schnittstellen
- Zeichenerkennung
- Syntaktische Analyse
- Benutzerführung, Ergonomie

ergab sich die Gliederung des Projektes in fünf Schwerpunkte durch Aufteilung der Zeichenerkennung in Blockschrifterkennung und Schreibschrifterkennung.

Die für die Lösung der Teilaufgaben nötigen Softwaremodule wurden jeweils als eigenständige Programme entwickelt, so daß eine parallele Entwicklung der einzelnen Softwarekomponenten möglich war. Durch die Verwendung von Interprozeßkommunikation (auf der Basis des in BSD-Unix integrierten TCP/IP-Socket-Konzeptes) zur Verbindung der einzelnen Prozesse zu einem Gesamtsystem ergaben sich gut überschaubare Schnittstellen.

Die Hardwareausstattung bestand zunächst nur aus einer Workstation auf Motorola 68020-Basis, die eine Integerleistung von 4 MIPS und eine (eher mäßige) Floatingpointleistung von 0.4 MFLOPS erbringt und unter der BSD-Variante von UNIX betrieben wird. Diese Hardwareplattform ist der Aufgabe angemessen. Es handelt sich um keine Hochleistungshardware, was besonders an der Tatsache deutlich wird, daß sich das Preis/Leistungsverhälnis von Workstations (gemessen an der Rechenleistung) in der bisher etwas mehr als zweijährigen Projektlaufzeit um mehr als den Faktor zehn verbessert hat.

Zu der genannten Workstation gesellte sich später ein PC-AT, der aber nur als Teil des Handschrift-Ein/Ausgabegerätes fungierte.

Einige Algorithmen zur Schreibschrifterkennung wurden probeweise auf ein aus sechs T800 bestehendes Transputernetzwerk portiert, das unter Kontrolle der Workstation betrieben wurde.

Besondere Schwierigkeiten bereitete die Wahl eines für die Handschrift geeigneten Eingabemediums. Wir dachten zunächst an einen Lichtgriffel. Diese Lösung wurde jedoch wegen der zu geringen Auflösung und des wegen der statischen Aufladung der Bildschirme unangenehmen Arbeitens verworfen.

Die Eingabe mit Hilfe eines Digitalisiertabletts — kombiniert mit der Ausgabe auf einem Monitor — erwies sich wegen der fehlenden Kontrolle bei der Positionierung des Stiftes als zu ungewohnt und schwer zu bedienen.

Der Ausweg bestand schließlich im Eigenbau der Ein/Ausgabeeinheit: Eine Flüssigkristallanzeige wurde auf ein Digitalisiertablett montiert.

Das Digitalisiertablett ertastet die Koordinaten des Stiftes durch die Flüssigkristallanzeige hindurch. An der gemessenen Position wird ein Punkt auf der

Abbildung 1: Aufbau der Ein/Ausgabeeinheit

Flüssigkristallanzeige gezeichnet, so daß die gezeichnete Figur allmählich unter dem Stift erscheint. Weil das Digitalisiertablett die Stiftkoordinaten mit Hilfe des Halleffektes mißt, die Flüssigkristallanzeige die Kristalle hingegen mit einem elektrischen Feld ausrichtet, beeinträchtigen die beiden Geräte sich gegenseitig nicht.

Schwierigkeiten entstanden lediglich durch die nicht ausreichende mechanische Genauigkeit beim Selbstbau des Gerätes. Dadurch stimmt die auf der Flüssigkristallanzeige angezeigte Position nicht immer exakt mit der tatsächlichen Position überein.

Ein Kunstgriff war notwendig, um das Verhalten des Schreibgerätes dem herkömmlichen Schreiben auf Papier anzugleichen. Das Digitalisiertablett ertastete die Stiftposition bereits, wenn sich der Stift ca. 5mm über der Oberfläche der Flüssigkristallanzeige befand. Dieser — beim Schreiben irritierende — Effekt wurde durch die Montage einer Distanzplatte aus Glas beseitigt. Das Digitalisiertablett wird dadurch ständig im Grenzbereich betrieben, was aber keine störenden Auswirkungen auf die Genauigkeit der gemessenen Koordinaten bewirkt.

Bei zu starker Neigung des Stiftes (Aufsetzwinkel kleiner als ca. 50^0) kann das Digitalisiertablett die Stiftkoordinaten nicht mehr messen. Der Koordinatenstrom bricht dann vorzeitig ab und die Zeichen werden von der Segmentierungsstufe zerrissen, was zu schweren Erkennungsfehlern führt.

In der Zwischenzeit sind Geräte der beschriebenen Art auch im Handel erhältlich, bei denen die genannten Schwierigkeiten wohl nicht mehr auftreten.

0.5 Aufbau der Arbeit

Die Gliederung dieser Arbeit folgt im wesentlichen der Gliederung der Projektaufgabe in die drei Hauptaufgaben *Zeichenerkennung, syntaktische Analyse* und *Benutzerunterstützung*.

Obwohl es sich um eine Arbeit handelt, die der experimentellen Informatik zuzuordnen ist (und deshalb keine Beweise enthält) habe ich mich bemüht, keine detaillierte 'Bauanleitung' anzufertigen, sondern nur die wesentlichen Seiten des Projektes zu beleuchten und die ausschlaggebenden Entwurfsentscheidungen zu motivieren. Da das Projekt über die Entwicklung des existierenden Prototypsystems hinausgehen wird, handelt es sich um eine Momentaufnahme des Projektes.

Weiterführende Informationen werden in den Diplomarbeiten vermittelt, die im Rahmen des Projektes angefertigt werden **[Kap]** **[Koel]** **[Rit]** und **[Wei]**.

0.6 Danksagungen

Mein Dank gilt zunächst Prof. Dr. G. Hotz. Er hatte die Idee zu diesem Projekt und finanzierte es aus Mitteln des ihm verliehenen Leibniz-Preises.

Prof. Hotz begleitete das Projekt mit Rat und Interesse, gewährte mir aber andererseits Freiheit bei der Leitung des Projektes. Ich danke ihm vor allen Dingen für die sicher nicht alltäglichen Arbeitsbedingungen und für die Aufgabe, die wegen ihrer Vielseitigkeit besonders reizvoll war (die Aufgaben reichten von 'Implementierungsaufgaben' mit dem Lötkolben auf Hardwareebene über Systemprogrammierung und Softwareentwicklung, bis zur Untersuchung theoretischer Fragestellung und der Konzeptionierung).

Von den (ca. 10) Studenten, die im Rahmen des Projektes mitarbeiteten, gilt mein besonderer Dank Randolph Kappes, Jürgen Kölsch, Uwe Ringling, Stephan Ritter und Frank Weigel. Sie trugen die Hauptlast der Implementierungsarbeiten und waren ein engagiertes, kritisches Diskussionsforum. Ich gestehe gerne, daß nicht nur sie von mir lernten ...

Der Spaß an der Zusammenarbeit, der auch in hektischen Situationen nie ganz verlorenging, hatte seine Ursache nicht nur in dem interessanten Thema des Projektes.

An meine Kollegen ergeht zunächst ein pauschaler Dank für Hilfe jeder Art.

Besonders herzlich möchte ich meinem Freund Andreas Nikolaus danken, der die Arbeit unter nicht einfachen Bedingungen korrekturgelesen hat und diese Arbeit im erweiterten Sinne als eine Art Testleser versah.

Elmar Schömer danke ich für die Geduld bei zahlreichen Diskussionen.

Kapitel 1

Mustererkennung und Vorverarbeitung

Das Aufspüren von Ähnlichkeiten und das Klassifizieren von Mustern sind zentrale Aufgaben der Informationsverarbeitung. In vielen Bereichen der Technik sind Mustererkennungsaufgaben zu lösen, wenn Arbeitsabläufe automatisiert werden sollen: Schriftverstehen, Klassifikation von Spektren in Massen- und NMR-Spektroskopie, Objekterkennung, Qualitätskontrolle, Navigation, Überwachung und Spracherkennung sind Beispiele solcher Aufgaben [**Dev**], [**Fer**]. Auch bei Untersuchungen menschlicher Intelligenz ist Mustererkennung ein wesentlicher Aspekt. [**Dre**] vertreten die Auffassung, daß Menschen Situationen alleine durch den Vergleich mit ähnlichen, bereits erlebten Situationen beurteilen. Sie halten die Fähigkeit des Menschen, Ähnlichkeiten zu erkennen und sich an ähnliche Situationen zu erinnern, für ausreichend, sich auch in neuen Situationen zu orientieren.

Es ist also nicht erstaunlich, daß Mustererkennungsaufgaben wesentliches Interesse gewidmet wurde. Hieraus resultieren zahlreiche unterschiedliche Ansätze, diese Aufgabe zu bewältigen.

Erstaunlich ist die Diskrepanz zwischen der Leichtigkeit, mit der Menschen auch gestörte Muster mit hoher Güte klassifizieren und der mangelnden Robustheit der entwickelten Erkennungsalgorithmen. Deshalb gilt der Erforschung der Mechanismen, die den Lebewesen ihre Überlegenheit in dieser Hinsicht verschaffen, seit Jahrzehnten rege Aufmerksamkeit.

In den letzten Jahren entwickelte sich die Erforschung konnektionistischer Methoden bzw. die Entwicklung sogenannter neuronaler Netze zu einem Forschungsschwerpunkt der Informatik. Ziel der Forschung ist hier die Entwicklung von Modellnervensystemen, die den natürlichen Vorbildern nachempfunden sind.

Zahlreiche Neuronen- und Synapsenmodelle[1] wurden entwickelt und anhand verschiedenster Lernparadigmen trainiert.

Obwohl die Dimensionen der auf Rechnern simulierten Modelle im Vergleich zu ihren natürlichen Vorbildern geradezu rührend klein anmuten, führt ihr Ein-

[1]Eine Einführung in die Begriffe und die prinzipielle Funktionsweise natürlicher Neuronen gibt [**Mea**].

satz zu ermutigenden Ergebnissen. Dies ist angesichts der mangelnden Kenntnisse über die Struktur von Nervensystemen nicht selbstverständlich. In der Organisation steckt der Witz der Nervensysteme von Lebewesen. [**Mea**] stellt fest, daß gerade der hohe Integrationsgrad des Nervensytems die analytische Erforschung unmöglich macht.

Dies deutet auch auf die Schwierigkeit der Konstruktionsaufgabe hin. Es hat also den Anschein, als ob gerade die Eigenschaft, die biologischen neuronalen Systemen die Überlegenheit über künstliche symbolverarbeitende Systeme verleiht, besonders schwer nachzuempfinden wäre.

Trotzdem werden künstliche neuronale Netze mit Gewinn in der Mustererkennung eingesetzt. Die mangelnden Kenntnisse über die Organisation der Nervensysteme versucht man dabei durch unterschiedliche Lernstrategien auszugleichen.

Ein sehr bekanntes Beispiel[2] für ein künstliches neuronales Netz zur Mustererkennung ist das *Neokognitron* [**Fuk**]. Es beruht auf physiologischen Erkenntnissen, die in den sechziger Jahren aus Versuchen an Katzen und Affen gewonnen wurden [**HW**] und ist in der Lage, auch Muster, die von den gelernten Instanzen in beschränktem Ausmaß abweichen, zu klassifizieren.

Interessant an diesem Ansatz ist die stufenweise Extraktion von Merkmalen aus dem zu klassifizierendem Bild. Das Neokognitron besteht aus in Schichten angeordneten Neuronen. Von Schicht zu Schicht sprechen einzelne Neuronen auf immer komplexere Muster an. Gleichzeitig werden die Neuronen immer unempfindlicher für affine Transformationen der Muster.

Neben der erheblichen aber (s.o.) wohl unumgänglichen Größe des Netzwerkes liegt der Nachteil des Neokognitrons in dem nur teilweise automatisierten Lernverfahren. Sollen neue Prototyp-Muster gelernt werden, müssen im schlimmsten Fall alle Schichten des Neokognitrons vom Benutzer gesteuert — d.h. manuell— trainiert werden.

Das Training eines neuronalen Netzes dient der Anpassung des Netzes an ein gegebenes Erkennungsproblem. Der Benutzer teilt dem Netz seine Vorstellung über die Einteilung der Menge aller Muster in Klassen mit Hilfe einer 'geeignet' gewählten Lernstichprobe mit. Die angestrebte, ideale Einteilung wird das *Musteruniversum* in disjunkte Klassen zerlegen. In unserer Anwendung wird jedem Muster eine Bedeutung in Form eines Buchstabens des Alphabetes zugeordnet (oder ggf. die Bedeutung 'nicht klassifizierbar').

Die zum Training verwendeten Prototypen enthalten neben dem Muster auch die Information, welcher Klasse das Muster angehört. In der Prototypmenge können inkonsistente Klassifizierungen auftreten d.h. ein Muster hat mehrere Bedeutungen. Dann muß der Lernalgorithmus eine Strategie zur Konfliktlösung vorsehen, da während des Betriebes des Netzes im Erkennungsmodus eine eindeutige Entscheidung gefällt werden soll. Strategien zur Konfliktlösung können auf die Berücksichtigung der relativen Häufigkeiten der Klassen in der Lernstichprobe (Maximum Likelihood) oder des Musterkontextes (z.B. Viterbi-Algorithmus) zurückgreifen.

[2]Eine — allerdings optimistisch anmutende — Übersicht über 'handelsübliche' neuronale Netze findet man in [**HN**].

Auch der Prototyp unseres Systems arbeitet mit einem neuronalen System. Neben dem ohne Zweifel vorhandenen, von der Aktualität dieses Ansatzes ausgehenden Reiz, gab es für den Einsatz eines neuronalen Netzes noch weitere Gründe, die sich zum Teil aus unserer Aufgabenstellung ergeben (siehe Abschnitt 1.2).

1.1 Mustererkennungsaufgabe

Der Schwierigkeitsgrad der Erkennungsaufgabe ergibt sich aus der Art und der Anzahl der zu unterscheidenden Musterklassen, der geforderten Erkennungsgüte und Erkennungsgeschwindigkeit, sowie der Art der Eingabedaten.

- Unser vorliegendes Problem erfordert die Echtzeitklassifizierung handschriftlich erstellter Symbole in ca. 50 - 80 Klassen (Die Erkennungsstufe des Prototyps verarbeitet 50 Klassen). Wir unterscheiden die Verwendung von Blockschrift und gebundener Schreibschrift.

  - Blockschrift kann auf einfache Weise in einzelne Zeichen zerlegt werden, die in der Regel einer von weniger als hundert Klassen angehören.
  - Die Erkennung von Schreibschrift gestaltet sich wesentlich schwieriger. Um zu einer handhabbaren Anzahl von Symbolklassen zu gelangen, werden Schreibschriftsymbole (Worte oder Silben) zunächst in Segmente zerlegt. Erst diese Segmente werden klassifiziert. Der höhere Schwierigkeitsgrad gegenüber der Blockschrifterkennung ergibt sich zum einen aus der sehr aufwendigen Segmentierungsaufgabe. Zum anderen variiert die Schreibweise der verwendeten Symbole stärker als bei Blockschriftzeichen, was auch zu einer größeren Anzahl von Symbolklassen führen kann.

  In unserem Projekt wurden Erkennungsalgorithmen zum Verarbeiten von Blockschrift und gebundener Schreibschrift implementiert und erprobt.

  Beim Schreiben von Formeln bietet die Verwendung von gebundener Schrift dem Benutzer keinen wesentlichen Vorteil. Zusätzlich ist die Erkennung gebundener Handschrift zeitaufwendig und störanfällig. Um befriedigende Erkennungsergebnisse zu erzielen, muß der Benutzer sehr sauber schreiben. Möglicherweise gewinnt die Behandlung gebundener Schrift aber an Bedeutung, wenn das System zu einer integrierten Arbeitsumgebung zum Verarbeiten von Formeln und Texten erweitert werden soll. Dies verdeutlicht, warum der Schwerpunkt der Untersuchungen dabei ganz eindeutig auf dem Gebiet der Blockschrifterkennung lag.

- Die Erkennungsgüte der Mustererkennung sollte größer als 90% sein, d.h. maximal jedes 10. Zeichen sollte wegen Fehlklassifikation korrigiert werden müssen.

- Wir legen größten Wert auf Echtzeitverarbeitung, damit das System flüssiges Arbeiten ermöglicht und die Kontrolle und evtl. Korrektur der Klassifikation durch den Benutzer möglich ist. Selbst bei schneller Schreibweise soll die Klassifikation nur mit der Verzögerung einer geringen, konstanten Anzahl von Zeichen erfolgen.

- Damit der Benutzer die von ihm benutzten Zeichen und deren Schreibweise selbst festlegen kann, soll das Mustererkennungssystem lernfähig sein. Der Benutzer versorgt das System mit Hilfe einer Lernstichprobe — d.h. einer Menge von Paaren bestehend aus Symbol und dessen Bedeutung — mit Information über die von ihm benötigten Zeichen und seine Handschrift. Zu unterschiedlichen Klassen gehörende Zeichen sollten dabei unterscheidbar sein, da der Prototyp die Symbole zunächst ohne Berücksichtigung des Kontextes erkennt.

  Wir setzen die Art und Anzahl der Klassen als a priori bekannt und während der Erkennung als invariant voraus. D.h. Lernen und Erkennen erfolgen in getrennten Arbeitsschritten. Aus Rücksicht auf die Handlichkeit des Gesamtsystems setzt die Instruierung des Erkennungsalgorithmus durch den Benutzer keinerlei Kenntnisse über den Erkennungsvorgang voraus. Manuelle oder halbautomatische Lernverfahren (wie z.B. das vom o.g. Neokognitron verwendete) kommen für unsere Anwendung also nicht in Frage.

  Die Eingabe der Lernstichprobe findet auf einer Benutzeroberfläche statt, die der beim Rechnen verwendeten fast völlig gleicht. Als Lernstichprobe werden alle verwendeten Zeichen einige Male in einer festgelegten Reihenfolge auf die Ein- /Ausgabeeinheit geschrieben. Der eigentliche Lernvorgang findet im Batchbetrieb z.B. über Nacht statt.

  Eine Besonderheit unserer Erkennungsaufgabe besteht darin, daß das System dem Benutzer beim Schreiben von Symbolen und Formeln 'über die Schulter schaut', d.h. Information über die zeitliche Entwicklung der vom Benutzer erstellten Objekte erhält. Diese zusätzliche Information kann sowohl die Mustererkennung als auch die Formelstrukturierung vereinfachen.

- Die Eingabe liegt in Form eines Stroms von Koordinaten vor, der von einem Digitalisiertablett geliefert wird und die auf der Ein-/Ausgabeeinheit geschriebenen Symbole beschreibt.

- Gerade in mathematischen Formeln werden Symbole unterschiedlicher Größe verwendet, die ohne Rücksicht auf Schreiblinien auf der Arbeitsfläche plaziert werden. Diese Freiheitsgrade erschweren die Mustererkennungsaufgabe, da die Abmessungen und in einigen Fällen auch die Proportionen der Zeichen stark variieren (z.B. beim Wurzelzeichen).

Es hat sich beim Entwurf von Mustererkennungsverfahren eingebürgert, der eigentlichen Klassifikation eine evtl. mehrstufige Vorverarbeitung vorzuschalten (siehe z.B. **[Har]** **[HN]** **[RMS]** **[Lee]**), die die Klassifikation der Symbole erleichtert und dadurch für eine höhere Erkennungsgüte sorgt.

Dieses *Preprocessing* ist auf das jeweilige Erkennungsproblem zugeschnitten. Deshalb werden wir in den folgenden beiden Abschnitten jeweils Preprocessing und Erkennungsverfahren gemeinsam besprechen.

1.2 Zwei konnektionistische Verfahren zur Blockschrifterkennung

Folgende Gründe sprechen für den Einsatz eines neuronalen Netzes in der Mustererkennungsstufe unseres Systemes:

1. Konnektionismus entstand in der Absicht, die Methoden, die dem Menschen seine im Vergleich mit Rechnern beeindruckenden Fähigkeiten bei der Mustererkennung verleihen, zu kopieren: Besonders interessant ist für uns der Aspekt der Robustheit der Mustererkennung.

2. Neuronale Netze können mit geeigneten Lernstrategien auf unterschiedliche Schreibweisen und Symbole adaptiert werden.

3. Die Lernstrategie nimmt dem Systementwickler die Aufgabe ab, die Unterscheidungskriterien zum Separieren unterschiedlicher Klassen zu formulieren. Diese brauchen also nicht manuell gefunden zu werden. Eine 'Algorithmisierung' des Problemes ist also nicht nötig. Die Programmierung beschränkt sich auf die Auswahl von Prototypen, die gute Repräsentanten der jeweiligen Musterklasse sind.

4. Die Implementierung neuronaler Netze gestaltet sich vergleichsweise einfach, da die eigentlichen Lern- und Erkennungsverfahren wenig Programmcode beanspruchen.

Der letzte Punkt wird leider durch das Fehlen exakter Analysen über Lernfähigkeit und Lernverhalten der einzelnen Netzwerkmodelle – von denen es eine Vielzahl gibt [**Kem**] – relativiert. Der genaue Aufbau des Netzes (Netzmodell, Netzstruktur, Eingabeformat, Ausgabecodierung und in Ausnahmefällen einige "magische Parameter") muß deshalb durch systematische exprimentelle Untersuchungen bestimmt werden [**Rit**].

Die in der Literatur veröffentlichten Beispiele von Mustererkennungsalgorithmen auf konnektionistischer Grundlage leiden meist unter Mängeln, die sie als Referenz beim Entwurf eines neuronalen Netzes — zumindest zu unserem Zweck — ungeeignet werden lassen. Zu diesen Mängeln gehören eine zu geringe Anzahl von zu unterscheidenden Klassen, Fehlen aussagekräftiger Versuche über die Erkennungsgüte, fehlende Angaben über die Verarbeitungsgeschwindigkeit oder eine unrealistische Beurteilung der Erkennungssicherheit bei unterschiedlichen Schreibweisen.

1.2.1 Eingabeformat, Segmentierung und Normalisierung

Um die Vorteile der Robustheit und der einfachen Implementierung auszunutzen, entscheiden wir uns für eine *analogische* Darstellung der Eingabedaten. Die Aufgabe, geeignete Merkmale aus den Mustern zu extrahieren, bleibt dem Netz überlassen. Die Muster werden also im wesentlichen bildhaft, d.h. als Bitmaps (die Größe beträgt 16×24 oder 12×18 Pixel), in das Netz eingelesen, das Preprocessing wurde auf wenige, schnell durchführbare Arbeitsschritte beschränkt.

Zunächst wird der von der Eingabeeinheit stammende Koordinatenstrom *segmentiert* d.h. in einzelne Muster zerlegt. Die Polygonzüge, die der Benutzer zeichnet, ohne den Stift abzuheben, bestehen aus Geradenstücken, die je zwei Koordinatenpunkte verbinden. Mehrere zeitlich aufeinanderfolgend entstandene Polygonzüge können zum selben Muster gehören und müssen vor der Mustererkennung zu einem Segment zusammengefaßt werden.

Zwei verschiedene Polygonzüge gehören genau dann zum selben Segment, wenn sie sich schneiden. Deshalb werden die Polygonzüge einem zweistufig arbeitenden Test auf Schnitt unterworfen. Zuerst werden die umfassenden Rechtecke der Polygonzüge auf Schnitt untersucht. Wird Schnitt festgestellt, werden die zu den sich schneidenden Polygonzügen gehörenden Geradensegmente auf Schnitt untersucht. Nur wenn sich tatsächlich Geradensegmente zweier Polygonzüge schneiden, werden die Polygonzüge zum gleichen Segment hinzugefügt.

Ein Segment gilt als abgeschlossen, wenn keiner von drei zeitlich aufeinanderfolgenden Polygonzügen sich mit ihm schneidet. Auf diese Weise können sowohl unzusammenhängend geschriebene Segmente (z.B. der Buchstabe H) erfaßt als auch ineinandergeschachtelte unterschiedliche Segmente (z.B. Wurzelzeichen und Radikant) auseinandergehalten werden. Bei der Untersuchung auf Schnitt wird eine Toleranz von einigen Pixeln gewährt, um durch Ungenauigkeiten beim Schreiben auseinandergerissene Segmente als zusammengehörend zu erkennen.

Die Größe der Segmente wird normalisiert, indem jedes Segment mit dem jeweils kleinsten Maßstab skaliert wird, der entweder die Höhe oder die Breite des Segmentes einen vorgegebenen Maximalwert erreichen läßt. Das Zeichen wird also bündig in eine (in unserem Fall die linke untere) Ecke eines vorgegebenen Rechtecks eingespannt und gleichmäßig gedehnt oder gestaucht, bis es in einer Dimension an den Rechteckrändern anstößt. Dadurch bleiben die Proportionen des Zeichens erhalten, was für die Unterscheidbarkeit wesentlich ist. Ein möglicher Fehler beim Entwurf von Erkennungssystemen liegt in der Wahl zu grob arbeitender Preprocessingalgorithmen, die zu nicht mehr behebbaren Erkennungsfehlern führen.

Abbildung 1.1: Normalisierte Buchstabensegmente

Für die beiden folgenden Abschnitte legen wir einige Notationen fest, die häufig wiederkehren.

Es bezeichne

- $T = \{t^1, \ldots, t^l\} \subset A^n$ eine Lernstichprobe vom Umfang l. Die Menge A ist dabei entweder $\mathbf{R}_0^+$ oder $\{-1, 1\}$. Gelegentlich bietet es sich an, T als Matrix zu schreiben, deren j-te Spalte aus t^j besteht. Die Matrix T ist i.a. nicht quadratisch.
- $p^j \in A^n$ ein zu erkennendes Muster.
- k die Anzahl der unterschiedlichen Musterklassen.
- $q^s \in A^n$ den globalen Zustand des neuronalen Netzes mit n Neuronen zum Zeitpunkt s und $q_i^s \in A$ entsprechend den Zustand des Neurons i zum Zeitpunkt s.

- $C \in \mathbf{R}^{n \times n}$ die Matrix der Synapsenstärken, die mit einem Lernalgorithmus in Abhängigkeit von den zu lernenden Mustern eingestellt werden. Die Synapse $C_{i,j}$ führt von Neuron i zu Neuron j. Synapsen mit $C_{i,j} < 0$ heißen inhibitorisch, Synapsen mit $C_{i,j} > 0$ exzitatorisch.

Alle Vektoren v seien Spaltenvektoren, Zeilenvektoren werden transponiert als v^{T} notiert.

Das aus den Neuronen und Synapsen $C_{i,j}$ bestehende neuronale Netz speichert sein 'Wissen' in den Synapsenstärken. Synapsen stellen Verbindungen dar, über die die Neuronen aufeinander einwirken (s. u.). Wird das neuronale Netz in einen globalen Zustand q^l versetzt, so wählt jedes Neuron j unter dem Einfluß der über die Synapsen $C_{i,j}$ auf es einwirkenden Neuronen i seinen neuen Zustand aus. Nach einigen Schritten erreicht das Netz gegebenenfalls einen *stabilen* Zustand, der keine Veränderung mehr erfordert. Dieser Zustand gilt als Ergebnis der Berechnung des Netzes bei Eingabe von q.

1.2.2 Hopfield-Modell

Hierbei handelt es sich um den wohl bekanntesten Vertreter von Modellen neuronaler Netze. Die Neuronen können Zustände aus $\{-1, 1\}$ annehmen. Die Eingabe ordnet allen n Neuronen des Netzes einen Anfangszustand zu. Mit einer a priori nicht bekannten Anzahl von Iterationsschritten ändern sich die Zustände der einzelnen Neuronen in Abhängigkeit von der Synapsenmatrix C und den Zuständen der anderen Neuronen. Dabei legt jedes Neuron die Berechnungsregel

$$q_i^{s+1} = [\theta(C \cdot q^s, q^s)]_i$$

zugrunde. θ bewirkt das komponentenweise Anwenden von Schwellen θ_i auf die Komponenten des Vektors $C \cdot q^s$. Es ist also

$$[\theta(C \cdot q^s, q^s)]_i = \begin{cases} 1 & \text{falls } (C \cdot q^s)_i > \theta_i \\ q_i^s & \text{falls } (C \cdot q^s)_i = \theta_i \\ -1 & \text{sonst} \end{cases}$$

Ein globaler Netzzustand q^s heißt stabil, falls $q^{s+1} = q^s$.

Durch geschicktes Einstellen der Synapsenstärken versucht man, die Muster der Lernstichprobe zu stabilen Netzzuständen zu machen. Hopfieldnetze werden dazu mit dem *Cooperschen* Algorithmus trainiert: Aus den zu lernenden Prototypen berechnet sich die Synapsenmatrix zu

$$C = \frac{1}{n} \sum_{t^i, t^j \in T} t^i \cdot {t^j}^{\mathrm{T}} = \frac{1}{n} \cdot T \cdot T^{\mathrm{T}},$$

d.h. die Synapsenmatrix wird im wesentlichen von der Autokorrelationsmatrix $T \cdot T^{\mathrm{T}}$ der zu lernenden Muster bestimmt.

Dies entspricht der *Hebbschen Lernregel*, wonach Synapsen zwischen Neuronen, die häufig den gleichen Zustand annehmen, zu verstärken und Synapsen zwischen Neuronen, die häufig entgegengesetzte Zustände annehmen, abzuschwächen sind.

Ein Hopfieldnetz zur Mustererkennung wird außer den zur Codierung des Musters benötigten Neuronen noch über zusätzliche Neuronen verfügen, die die Zugehörigkeit des Musters zu einer der k Klassen angeben (siehe hierzu Abb. 1.4). Die Muster der Lernstichprobe werden um die Information ihrer Zugehörigkeit zu einer Musterklasse erweitert und gelernt. Soll ein unbekanntes Muster klassifiziert werden, wird es zunächst in das Netz eingelesen, wobei die Neuronen, die die Klassenzugehörigkeit codieren, willkürliche Belegungen erhalten.

In einer nicht bekannten Anzahl von Schritten relaxiert das Netz in einen stabilen Zustand. Läßt sich nun eine Klassenzugehörigkeit an den Neuronen ablesen, ist die Erkennung geglückt, andernfalls wird das Muster zurückgewiesen.

Das Hopfield-Modell setzt sich der Kritik aus, daß die verwendete Lernregel im allgemeinen nicht die Stabilität der Prototypen sicherstellt. **[RGD]** stellen deshalb eine auf dem Konzept der *Moore-Penrose-Pseudoinversen*[3] beruhende Lernregel vor, bei der Prototypen sicher gelernt werden.

Die Stabilität der Prototypen t^l drückt sich in der Bedingung

$$\mathrm{sgn}(\left[C \cdot t^l - \theta\right]_i) = \begin{cases} \mathrm{sgn}(t_i^l) & \text{oder} \\ 0 & \end{cases}$$

aus. Wählt man A^l als Diagonalmatrix freier Schlupfvariablen $A^l_{i,j} > 0$, so führt dies zu folgender Gleichung:

$$C \cdot t^l - \theta = A^l \cdot t^l$$

Da diese Gleichung — mit geeignetem A^l — für alle t^l gilt, ist zur Festlegung der Synapsenmatrix C insgesamt die Gleichung

$$C \cdot T = \left[A^l \cdot t^l + \theta\right] \tag{1.1}$$

zu lösen. Der Ausdruck auf der rechten Seite der Gleichung beschreibt dabei die Matrix, deren l-te Spalte $A^l \cdot t^l + \theta$ ist. Hat T maximalen Rang, so lautet die Lösung für C

$$C = \left[A^l \cdot t^l + \theta\right] \cdot T^+.$$

$T^+ = (T^{\mathrm{T}} \cdot T)^{-1} \cdot T^{\mathrm{T}}$ ist die Pseudoinverse von T, die sich aus den Prototypen t^l mit dem *Grevilleschen* Algorithmus **[Koh2]** iterativ berechnen läßt

Sind die Prototypen nicht nur linear unabhängig, sondern sogar orthogonal, so liefert Gleichung (1.1) bei der Wahl $A^l = I$ für alle l und $\theta = 0$

$$\begin{aligned} C = T \cdot T^+ &= T \cdot (T^{\mathrm{T}} \cdot T)^{-1} \cdot T^{\mathrm{T}} \\ &= T \cdot (n \cdot I)^{-1} \cdot T^{\mathrm{T}} \\ &= \frac{1}{n} T \cdot T^{\mathrm{T}}. \end{aligned}$$

[3] Ist $TT^{\mathrm{T}} = V \cdot diag(\lambda_i) \cdot V^{\mathrm{T}}$ die Hauptachsentransformation der positiv definiten Matrix $T \cdot T^{\mathrm{T}}$, dann ist $T = U \cdot diag(\sqrt{\lambda_i}) \cdot V^{\mathrm{T}}$ die *Singulärwertzerlegung* von T ($U = diag(1/\sqrt{\lambda_i})$, $U \cdot U^{\mathrm{T}} = I$). Die Pseudoinverse T^+ von T ist durch $T^+ = V \cdot diag(\Sigma_i) \cdot U^{\mathrm{T}}$ definiert, mit $\Sigma_i = \frac{1}{\sqrt{\lambda_i}}$ falls $\lambda_i \neq 0$ und $\Sigma_i = 0$ sonst definiert **[Hua]**. Besitzt T maximalen Rang, dann gilt $T^+ = (T^{\mathrm{T}} \cdot T)^{-1} \cdot T^{\mathrm{T}}$

Für den Spezialfall orthonormaler Prototypen sind die beiden Lernverfahren also identisch. Die von **[RGD]** getroffenen Aussagen lassen sich dann auf das Coopersche Verfahren übertragen.

Das auf der Pseudoinversen beruhende Verfahren bietet folgende Vorteile:

- Es wird eine hinreichende Bedingung für die Lernbarkeit von Prototypen formuliert.
- Lernbare Prototypen werden sicher gelernt.
- Falls die Multiplikation mit der Pseudoinversen wegen linearer Abhängigkeit der Prototypen keine exakte Lösung liefert, erhält man eine Näherungslösung der Gleichung 1.1 mit kleinstem Defekt (gemessen in der euklidschen Matrizenmetrik $\| \quad \|_E = \sqrt{\mathrm{spur}(A^{\mathrm{T}} \cdot A)}$).
- Das Lernen läßt sich inkrementell durchführen, da die Berechnung der Pseudoinversen nach Greville inkrementell erfolgt.

In einer realen Entwurfssituation kann die Wahl der Schwellen θ durch die technischen Rahmenbedingungen eingeschränkt sein. Die Schlupfmatrizen A^l sind frei wählbar.

Ist $A^l = I$ für alle l und $\theta = 0$ eine zulässige Wahl, so können alle möglichen Kombinationen von Vektoren gelernt werden (die Lernbarkeitsbedingung fällt also weg). Die Gleichung

$$C \cdot T = T$$

wird nämlich für beliebige Matrizen T von $C = T \cdot T^+$ gelöst.

Weil TT^+ die Projektion von $\mathbf{R}^n$ auf Bild(T) ist, erfüllen in diesem Fall alle Vektoren $t \in$ Bild(T) die Stabilitätsbedingung $C \cdot t = t$. Alle Vektoren in dem von den Vektoren t^l aufgespannten Vektorraum sind also stabil.

Im Rahmen einer Mustererkennungsaufgabe ist dies unerwünscht, weil sich Mischformen der gelernten Muster als stabile Zustände einstellen. Deshalb sollte die Wahl $\theta = 0$ und $A^l = I$ für alle l vermieden werden.

1.2.3 Versuche mit Hopfieldnetzen

Die Versuche mit dem Hopfield-Modell wurden durchgeführt, um die Eignung dieses Modells, das wohl das am besten untersuchte Modell neuronaler Netze sein dürfte, für unsere Schrifterkennungsaufgabe zu untersuchen.

Alle in den Versuchen verwendeten Netze benutzen das gleiche Musterformat. Ein Muster besteht aus 16×24 binären Pixeln. Jedes Pixel gibt die Anfangsbelegung eines Neurons des Netzes an. Zusätzlich zu den für die Codierung der Muster benötigten Neuronen gibt es noch für jede verwendete Musterklasse ein weiteres Neuron. Dieses Neuron wird aktiviert, wenn das gerade im Netz gespeicherte Muster der entsprechenden Musterklasse angehört.

Im ersten Versuch wurden vier unterschiedliche Prototypen mit dem Lernalgorithmus von **[RGD]** gelernt. Das Netz erkannte anschließend alle Protoypen wieder, auch wenn diese zu 30% verrauscht wurden(Abb. 1).

Das Hopfield-Modell eignet sich also sehr gut zum Ausfiltern von Rauschen, wenn das Aussehen des Musters nicht verändert wurde. Diese Leistung wird

Abbildung 1.2: Gelernte Prototypen und korrekte Klassifikation

aber auch von herkömmlichen linearen Filtermethode erbracht, die additives weißes Rauschen aus Bildern recht gut wieder entfernen können ([**RK**] Kap.7).

Demgegenüber weist der zweite Versuch die mangelnde Robustheit des Netzes gegenüber Deformationen, wie sie beim Schreiben vorkommen, nach. Das Netz aus Versuch 2 sollte 20 unterschiedliche, von den Prototypen abweichende Buchstaben erkennen. Dies gelang jedoch nur bei 11 Buchstaben.

Als Vergleich wurde ein einfacher Algorithmus herangezogen, der die Buchstaben mit Hilfe der Hammingdistanz klassifiziert und bei 16 Buchstaben das richtige Ergebnis lieferte. Die mit dem Vergleichsalgorithmus korrekt klassifizierten Buchstaben waren eine Obermenge der von dem Hopfieldnetz korrekt erkannten Buchstaben. Es zeigte sich, daß sich neben den gelernten Prototypen und deren Linearkombinationen noch weitere stabile Netzzustände eingestellt hatten, was natürlich unerwünscht ist.

Abbildung 1.3: Fehlklassifikation, Hammingdistanz zwischen gelerntem und zu erkennendem Buchstaben, korrekte Klassifikation

Abhilfe schafft hier natürlich die Vergößerung der Lernstichprobe. Ein Versuch mit einer Lernstichprobe, die je vier Prototypen jeder Klasse enthielt, erreichte eine Erkennungsquote von 100%.

Die Vergrößerung der Lernstichprobe erfordert aber eine entsprechende Vergrößerung des Netzes, die bei realistischen Versuchsaufbauten nicht akzeptabel ist[4].

Versuchsweise wurden sowohl die gelernten als auch die zu erkennenden Muster punktweise verbreitert, um so die Hammingdistanz zwischen leicht unterschiedlich geschriebenen Buchstaben einer Klasse zu verringern. Dieses einfache Preprocessing verbesserte das Ergebnis von Versuch 1 auf 17 korrekt erkannte Buchstaben.

Abbildung 1.4: verbreiterte Prototypen, gelungene Klassifikation, Fehlklassifikation (als stabiler Zustand stellte sich eine Linearkombination zweier Prototypen ein.

Abschließend wurde ein Versuch mit einer realistischen Anzahl von unterschiedlichen Klassen durchgeführt. Je zwei Exemplare von 24 unterschiedlichen Buchstaben wurden gelernt. Eine zweite Stichprobe vom gleichen Umfang diente der Überprüfung der Erkennungsgüte. Siebzehn Buchstaben wurden korrekt erkannt, in einem Fall wurde ein Buchstabe falsch klassifiziert, in allen anderen Fällen wurde der Buchstabe zurückgewiesen, d.h. das Netz berechnete einen nicht gelernten stabilen Zustand als Endzustand der Klassifizierung.

[4][**Hop**] setzt die Kapazität des Hopfield-Modells empirisch mit 15% der Neuronenanzahl an. Um 10 Exemplare aus 50 verschiedenen Klassen zu lernen, sind also ca. $\frac{1}{3} \cdot 10^4$ Neuronen und folglich $\frac{1}{9} \cdot 10^8$ Synapsen nötig. Schon in unserem Versuchsaufbau mit ca. $2 \cdot 10^4$ Synapsen betrug die Erkennungszeit ca. 2 Sekunden.

Die Klassifizierung auf der Grundlage der Hammingdistanz erwies sich mit 25 korrekt erkannten Buchstaben auch in diesem Fall überlegen.

Die durchgeführten Versuche sprechen also aus zwei Gründen gegen den Einsatz des Hopfield-Modells als Mustererkennungsalgorithmus in unserem System zum handschriftlichen Arbeiten:

- Im Gegensatz zur Robustheit der Netze gegenüber Verrauschen steht ihre Empfindlichkeit gegenüber Deformationen der Muster. Selbst die Klassifizierung mit Hilfe der Hammingdistanz, die ebenfalls gegenüber Deformationen der Muster empfindlich ist, erweist sich als bzgl. der Erkennungsgüte überlegen. Bei einer Erweiterung der Versuche auf realistische Größenordnungen ist eher noch mit einer Verschlechterung der Erkennungsgüte zu rechnen.
- Abhilfe scheint nur die Vergrößerung der Lernstichprobe zu bieten. Die dann zu erwartende Größenordnung des Netzes läßt diese Lösung aber unattraktiv werden.

1.2.4 Rumelhart-Modell

Das Rumelhart-Modell besteht aus in mehreren Schichten angeordneten Neuronen, die Zustände aus $\mathbf{R}_0^+$ annehmen können. In Abweichung vom Hopfield-Modell berechnen die Neuronen des Rumelhartnetzes ihren Zustand nicht mit Hilfe einer Schwellenfunktion, sondern mit einer differenzierbaren Transferfunktion σ, die der Schwellenfunktion θ nachgebildet ist (σ heißt wegen ihres s-förmigen Graphen *Sigmoidfunktion*).

$$q_i^{s+1} = \sigma\left(\sum_j C_{ji} \cdot q_i^s\right)$$

Nur Neuronen aufeinanderfolgender Schichten sind durch Synapsen verbunden. Die Synapsen verlaufen dabei ausschließlich von Schicht i nach Schicht $i+1$. Es existieren keine Synapsen, die innerhalb einer Schicht verlaufen oder einige Schichten überspringen.

Die Eingabe ordnet nur den n Neuronen der untersten Schicht eine Belegung zu, die Ausgabe des Netzes wird an den m Neuronen der obersten Schicht abgelesen. Die restlichen *versteckten* Schichten werden vom Netz zum Rechnen benutzt. Das Rumelhart-Modell eignet sich also besonders zum Lernen funktionaler Abhängigkeiten.

Weil der Verbindungsgraph des Netzes gerichtet und azyklisch ist, wird die Ausgabe in einer festen Anzahl von Schritten berechnet.

[**RCW**] verallgemeinern eine Lernregel für zweistufige Netze zur *Backpropagation-Lernregel*, die Netze beliebiger Tiefe trainieren kann. Ausgehend von einer zufälligen Anfangsbelegung der Synapsenmatrix führt die Backpropagation-Regel für jedes zu lernende Muster einen zweistufigen Lernschritt durch. (Wir notieren die Muster t^l und p^l als Paare $(e^l, a^l) \in {\mathbf{R}_0^+}^n \times {\mathbf{R}_0^+}^m$, wobei e^l die Eingabe in die unterste Netzschicht und a^l die Ausgabe der obersten Schicht darstellt.)

1) Das Netz berechnet mit Hilfe der aktuellen Synapsenmatrix aus der Eingabe e^l eine Ausgabe $\overline{a}^l$, die i.a. von a^l abweichen wird. Als Maß für die Abweichung dient die Summe der Fehlerquadrate

$$E_l(C) = \frac{1}{2}\sum_{j=1}^{m}(a_j^l - \overline{a}_j^l)^2.$$

2) Beginnend mit der obersten Schicht wird die Synapsenmatrix mit dem Ziel, den gemessenen Fehler zu minimieren, korrigiert.

   Dazu muß für jede Synapse C_{ij} die partielle Ableitung

$$\frac{\partial E_l(C)}{\partial C_{ij}}$$

   bestimmt werden. Mit Hilfe der Kettenregel wird diese Ableitung in

$$\frac{\partial E_l(C)}{\partial C_{ij}} = \frac{\partial E_l(C)}{\partial(\sum_r C_{rj}\cdot q_r^l)} \cdot \frac{\partial(\sum_r C_{rj}\cdot q_r^l)}{\partial C_{ij}} = \frac{\partial E_l(C)}{\partial(\sum_r C_{rj}\cdot q_r^l)} \cdot q_i^l =: \delta_j \cdot q_i^l$$

   zerlegt. δ_j gibt also die Abhängigkeit der Fehlerfunktion von der an Neuron j anliegenden Eingabe $\sum_r C_{rj}\cdot q_r^l$ an.

Die Werte δ_j lassen sich rekursiv berechnen. Für Synapsen der obersten Synapsenschicht ergibt sich naheliegend

$$\begin{aligned}
\delta_j &= \frac{1}{2}\cdot\frac{\partial \sum_{\substack{\text{Ausgabe-}\\ \text{neuronen}}} (a_j^l - \overline{a}_j^l)^2}{\partial \sum C_{ij} q_i^l} \\
&= \frac{1}{2}\cdot\frac{\partial \sum \left(a_j - \sigma(\sum C_{ij} q_i^l)\right)^2}{\partial \sum C_{ij} q_i^l} \\
&= \left(a_j - \sigma\left(\sum C_{ij} q_i^l\right)\right)\cdot \sigma'\left(\sum C_{ij} q_i^l\right)
\end{aligned}$$

σ' ist dabei die Ableitung der Funktion σ.

Für Synapsen versteckter Schichten ist diese Berechnung nicht anwendbar. Dort ergibt sich δ_j aus den Werten der nachfolgenden Schicht, was durch eine ähnliche Rechnung wie oben gezeigt werden kann.

$$\delta_j = \sum_k (\delta_k \cdot C_{jk}) \cdot \sigma'\left(\sum_r C_{rj} q_r^l\right)$$

Die Berechnung der Änderungen schreitet also rückwärts im Netz voran.

[**RCW**] weisen nach, daß die Änderungen der Synapsen dem Gradienten der Funktion $E_p(C)$ folgt. Die Backpropagation-Regel ist also ein Gradientenverfahren. Sie ist heuristisch, da die Schrittweite der Änderung willkürlich mit einem magischen Parameter festgelegt wird.

Der zweistufige Lernschritt wird der Reihe nach für alle zu lernenden Paare $p^l = (e^l, a^l)$ durchgeführt, bis die Fehlerfunktion unter eine festgelegte Schranke gedrückt, oder eine maximale Schrittzahl erreicht wurde.

1.3 Eigenschaften des Rumelhart-Modells

Da der Verlauf der Fehlerfunktion völlig unbekannt ist, können keine Aussagen über die Konvergenzgeschwindigkeit oder die Wahrscheinlichkeit, das globale Minimum zu erreichen, gemacht werden.

In Versuchen weisen [**RCW**] jedoch nach, daß als schwierig zu lernen geltende Funktionen sich mit dem Rumelhart-Modell lernen lassen. Dazu gehören Funktionen $f : \mathbf{R} \longrightarrow \mathbf{R}$, deren Einschränkung auf $\{0, 1\}$ die Quersummenbildung (modulo 2) beschreibt.

Obwohl die Leistungsfähigkeit des Modells auch in anderen Versuchen nachgewiesen wurde [**RMS**][**Lee**], liegen nur wenige theoretische Untersuchungen über die Leistungsfähigkeit des Modells vor.

[**IM**] beweisen, daß sich jede stetige Funktion f in n reellen Variablen durch ein dreistufiges Netzwerk berechnen läßt, falls statt der Übergangsfunktion σ eine auf $\mathbf{R}$ absolut integrierbare Funktion verwendet wird und der Urbildbereich von f kompakt ist. Dieser Darstellungssatz beruht auf der Fourieranalyse von f und führt auf ein Netzwerk, dessen mittlere Schicht potentiell überabzählbar viele Neuronen besitzt.

[**Fun**] übertrug diesen Satz auf die im Rumelhart-Modell verwendete Transferfunktion σ. Die Darstellung ist dann jedoch nicht mehr exakt, aber beliebig genau.

Leider sind die zitierten Sätze nur von theoretischem Interesse, da keine Abschätzungen des Fehlers bei Beschränkung auf endliche Netzwerkdimensionen durchgeführt wurden.

1.3.1 Musterformat

Da die Neuronen eines Rumelhartnetzes reellwertige Zustände annehmen, werden wir Muster verwenden, die reellwertige Intensitäten aufweisen. Diese Eigenschaft können wir verwenden, um die Muster mit Informationen über den zeitlichen Ablauf ihrer Entstehung zu versehen. So können die Pixel, die an Kreuzungspunkten liegen, besonders markiert werden, oder in jedem Pixel die Verweildauer beim Schreibvorgang notiert werden. In den durchgeführten Versuchen bestanden die Muster in der Regel aus 16×24 reellwertigen Pixeln, in einigen Fällen wurde das kleinere Format 12×18 verwendet.

Abbildung 1.5: Beispiel für einen Satz von Prototypbuchstaben im Format 16×24

Durch Aneinanderhängen der Musterzeilen wurden die Muster in einen Vektor umgeformt, der als Eingabe in die unterste Schicht des Rumelhartnetzes diente. Die Art der Umformung, mit der die Mustermatrix in einen Vektor verwandelt wird, ist jedoch unerheblich, solange die Eingabeschicht mit der darüberliegenden Schicht totalvernetzt ist, d.h. jedes Neuron der Eingabeschicht mit jedem Neuron der zweituntersten Schicht des Netzes durch eine Synapse verbunden ist.

1.3.2 Versuche mit dem Rumelhart-Modell

Die im Rahmen des Projektes durchgeführten Versuche dienten zunächst dem Test der Implementierung und der systematischen Einstellung der Netzparameter. Dazu wurden die in [**RCW**] beschriebenen Experimente wiederholt und im wesentlichen nachvollzogen. Weitere Versuche waren auf die Belange der Buchstabenerkennung zugeschnitten. Hier wurden die Netzwerkdimensionen, geeignete Darstellungen der Buchstaben und Verfahren zur Vorverarbeitung der Buchstaben bestimmt.

Die in [**Rit**] geschilderten Versuche zur Erprobung unterschiedlicher Netzwerkdimensionen lassen sich in der Faustformel zusammenfassen, daß dreischichtige Netze, bei denen die Anzahl der Neuronen der Ausgabeschicht und der versteckten Schicht der Anzahl der zu unterscheidenden Musterklassen entspricht, zur Mustererkennung besonders geeignet sind.

Diese Faustformel sollte durch einen Modellversuch weitergehend gestützt werden. Eine [**Sch**] entnommene Rechnung zur MaximumLikelihoodKlassifikation von Zeichen wurde in einen Versuchsaufbau abgewandelt.

Es wurden Punkte, die jeweils einer von drei voneinander unabhängigen (zweidimensionalen) Normalverteilungen angehörten, durch Zufallszahlengeneratoren gewürfelt. Ein neuronales Netz lernte die Punkte und die Zugehörigkeit der Punkte zu einer der drei Verteilungen. Anschließend wurde überprüft, ob das Netz die Maximum–Likelihood–Grenze herausfinden konnte (d.h. ob das Abstiegsverfahren die entsprechende Einstellung der Synapsenstärken finden konnte und ob die darzustellende Funktion durch die vorgegebene Netzwerkstruktur überhaupt interpoliert werden konnte).

Weil die Versuche sehr rechenaufwendig sind, wurden nur Versuche mit zwei oder drei Klassen durchgeführt. Wir schildern hier nur den Versuch mit drei Klassen.

Abbildung 1.6: Eine der drei Klassen: Die Einteilung der Punkte in Klassen ist nicht disjunkt, so daß eine Maximum-Likelihood-Entscheidung getroffen werden muß.

Abbildung 1.7: Alle drei Klassen: In den Randbereichen muß die Entscheidung wegen der geringen Anzahl von Punkten unsicher bleiben.

Abbildung 1.8: Der Versuch mit zwei versteckten Neuronen (Bei einem versteckten Neuron kann die Verteilung überhaupt nicht gelernt werden, bei zwei versteckten Neuronen ist die Maximum-Likelihood-Entscheidung bereits erkennbar.)

Es wurden Punkte aus einem Quadrat der Kantenlänge 1 gewürfelt. Um einen Netzaufbau zu erhalten, der den Netzen zur Schrifterkennung weitgehend entsprochen hätte, wäre es naheliegend gewesen, das Quadrat mit einem Gitter zu überziehen und jeden Gitterpunkt (oder jedes Teilquadrat) einem Neuron der

Abbildung 1.9: Die Versuche mit drei und zehn versteckten Neuronen: Das Netz vollzieht die Maximum-Likelihood-Grenzen nach. Das Netz mit zehn versteckten Neuronen ist nicht wesentlich besser als das mit drei versteckten Neuronen.

Eingabeschicht zuzuordnen. Der dann anfallende hohe Rechenaufwand verbot aber diesen Versuchsaufbau. Deshalb mußte auf einen einfacheren Versuch ausgewichen werden, bei dem die Eingabeschicht nur aus zwei Neuronen bestand, von denen jedes eine Koordinate codierte.

Die Größe der Ausgabeschicht entsprach der Anzahl der zu unterscheidenden Klassen, die Größe der versteckten Schicht wurde variiert.

Wie man den entsprechenden Abbildungen entnehmen kann, scheint die Dimensionierung der versteckten Schicht in der Größe der Ausgabeschicht tatsächlich ein Optimum im Verhältnis von Aufwand und Erkennungsgüte zu liefern. Das wesentlich vergrößerte Netz mit zehn versteckten Neuronen bot keine Verbesserung. Die Festlegung der Maximum–Likelihood–Grenze durch das Netz ist in Gebieten, in denen sich nur wenige Punkte der Verteilungen befinden, naturgemäß unsicher.

In den durchgeführten Versuchen zeigte sich, daß Musterklassen, die sich aus sehr unterschiedlichen Zeichen zusammensetzten, in der normalerweise ausreichenden Anzahl von Lernschritten nicht gelernt wurden. Dieses Verhalten liegt nicht in der Struktur des Rumelhart-Modells — d.h. in der Art der darstellbaren Funktionen — begründet, sondern in dem Backpropagation-Lernalgorithmus. Wir demonstrieren an einem einfachen Beispiel, daß der Lernalgorithmus in Sackgassen geraten kann, wenn sehr unterschiedliche Muster zu einer Klasse gehören:

Die Menge aller möglichen (Eingabe-) Muster, $\mathbf{R}^n$, enthalte die zwei Klassen $\mathbf{A}$ und $\mathbf{B}$. $\mathbf{A}$ sei aus den disjunkten Komponenten A' und A'' zuammengesetzt. Die beiden Komponenten von $\mathbf{A}$ entsprechen zwei unterschiedlichen Schreibweisen der Muster in $\mathbf{A}$.

Nachdem das Netz die Schreibweise A' und die Klasse B gelernt hat, sieht die Struktur des Netzes evtl. wie in Abb. 1.10 aus.

Abbildung 1.10: Mögliche Struktur des Netzes beim Lernen zweier Klassen A und B, von denen eine sehr unterschiedliche Muster A' und A" beinhaltet

Die Bäume über A' und B sollen die Verbindungen zwischen Neuron A und A' bzw. Neuron B und B darstellen. Wir untersuchen zwei unterschiedliche Fälle:

1) Es bestehen auch Verbindungen zwischen A'' und dem Neuron A. Bei Präsentation von A'' werden diese Verbindungen verstärkt, so daß ein Verbindungsbaum zwischen A'' und Neuron A entsteht. Gleichzeitig werden die von A' zu A führenden Verbindungen aber nicht abgeschwächt, da die Synapsenänderungen gemäß

$$\Delta C_{ij} \sim \delta_j \cdot q_i^l$$

berechnet werden und alle Neuronen des Baumes über A' inaktiv sind oder auch zum Baum über A'' gehören.

Nachdem das Netz A' und A'' gelernt hat, wird es also auch alle Kombinationen von A' und A'' als Muster von A anerkennen, was unerwünscht sein kann.

2) Es gibt keine Verbindungen zwischen A'' und dem Neuron A. In diesem Fall ist die Schreibweise A'' nicht als zu **A** gehörend lernbar, da keine Fehlerinformation, die zu einer Synapsenverstärkung führen könnte, zu den von A'' ausgehenden Synapsen propagiert wird.

Die Diskussion erklärt die Schwierigkeiten, unterschiedliche Schreibweisen eines Buchstaben zu lernen. Trotz allem sind Rumelhartnetze in der Lage, unzusammenhängende Klassen zu lernen, wie das Beispiel der oben erwähnten Quersummenfunktionen zeigt. Dies ist möglich, weil die Netze üblicherweise mit *kleinen* Anfangsbelegungen der Synapsen zu lernen beginnen, eine Verminderung der Synapsenwerte also meist unnötig ist. Das Aufreißen von Synapsen dürfte sehr unwahrscheinlich sein, da mit Gleitkommaarithmetik gerechnet wird. Jedoch verzögern betragsmäßig kleine Synapsenwerte in diesem Fall die Entwicklung des Netzes.

Wir fordern aus den oben angeführten Gründen und wegen der entsprechenden Versuchsergebnisse, daß unterschiedliche Schreibweisen eines Buchstabens in getrennten Klassen erfaßt werden. Außerdem sollten die Zustände der Neuronen symmetrisch um 0 liegen, im einfachsten Fall sollte ein aktiviertes Neuron den Zustand 1 und ein desaktiviertes Neuron den Zustand -1 annehmen. Nach dieser Änderung können störende Synapsen auch wieder abgebaut werden.

Um ein 'Auseinanderreißen' des Netzes zu verhindern, sollte vermieden werden, daß Synapsen den Wert 0 annehmen können. Dies erreicht man gegebenenfalls durch Aufrunden.

Die vorgeschlagenen Änderungen beeinträchtigen die theoretische Herleitung des Abstiegsverfahrens nicht. Es ist erstaunlich, daß diese Modifikationen in der Literatur nicht bereits vorgeschlagen wurden.

1.4 Vorverarbeitung für das Rumelhart-Verfahren

Durch eine geeignete Aufbereitung der Muster vor der eigentlichen Klassifizierung kann die Erkennungsgüte evtl. gesteigert werden. Man wird versuchen, die Unterschiede zwischen Mustern, die der gleichen Klasse angehören, zu verringern, gleichzeitig aber die Unterscheidbarkeit der Klassen zu gewährleisten.

Bereits bei der Mustererkennung mit Hilfe der Hopfieldnetze hatten wir eine Steigerung der Erkennungsgüte mit dem sehr einfachen Vorverarbeitungsverfahren 'Verbreiterung jedes Bildpunktes' erreichen können. Da Rumelhartnetze Muster verarbeiten, deren Pixel reellwertige Intensitäten annehmen können, gibt es mehr Freiheitsgrade bei der Auswahl der Vorverarbeitung als im Fall der Hopfieldnetze.

Wir wollen die Wahl des verwendeten Preprocessing im folgenden begründen. Dazu führen wir einige einfache Begriffe der digitalen Bildverarbeitung ein und legen die häufig vorkommenden Notationen fest.

Die zu erkennenden Muster sind Bildmatrizen $P \in \mathbf{R}^{n \times m}$. Die Bildmatrizen lassen sich als Bildvektoren $\vec{p}$ auffassen, indem die Bildmatrizen zeilenweise aufgeschrieben werden.

Eine Abbildung $\mathcal{F}$, die Bilder in Matrix- oder Vektorform auf Bilder abbildet, nennen wir *Filter*:

$$\mathcal{F} : \mathbf{R}^{n \cdot m} \longrightarrow \mathbf{R}^{n \cdot m}$$

Bis auf die Ausnahme der translationsinvarianten Filter sind die von uns verwendeten Filter *linear*, d.h.

$$\mathcal{F}(\vec{p}) = F \cdot \vec{p}$$

wobei $F \in \mathbf{R}^{n \cdot m \times n \cdot m}$ ist. Da wir negativen Bildintensitäten in unserem Versuchsaufbau keine vernünftige physikalische Bedeutung zuordnen können, werden die Einträge der Filtermatrizen nur nichtnegativ sein. Wir verwenden außerdem nur *separierbare* Filter, d.h. Filter, die auf der Matrixform des Bildes berechnet werden können:

$$\mathcal{F}(P) = F_1 \cdot P \cdot F_2^{\mathrm{T}}$$

Dabei sind F_1 und $F_2 \in \mathbf{R}^{n \times m}$ beliebige Matrizen. Separierbare Matrizen stellen insofern eine Einschränkung der auf der Vektorform der Bilder arbeitenden Filter dar, als die Filtermatrix F dann eine sehr spezielle Form aufweist: F ergibt sich als direktes Produkt von F_1 und F_2.

Mit $\mathrm{Shift}(k,l)$ bezeichnen wir die zyklische Verschiebung einer Matrix um (k,l):

$$(\mathrm{Shift}(k,l)\,(A))_{i,j} = (A)_{(i-k) \bmod n, (j-l) \bmod m}$$

Die Übertragung der Verschiebung auf Bilder in Vektorschreibweise erfolgt in Einklang mit dieser Definition.

Ein Filter $\mathcal{F}$ heißt *ortsunabhängig*, wenn

$$\mathcal{F}\left(\mathrm{Shift}(k,l)(\vec{p})\right) = \mathrm{Shift}(k,l)\left(\mathcal{F}(\vec{p})\right)$$

gilt.

Die Matrizen ortsunabhängiger Filter sind in Toeplitzform [**Hua**], d.h. die Einträge der Matrix sind nur von ihrem Abstand zur Hauptdiagonale abhängig:

$$F_{i,j} = F_{k,l} \quad \text{für} \quad |i-j| = |k-l|$$

Diese Eigenschaft gilt analog für die Filter, die auf der Vektorschreibweise der Bilder operieren.

Die Filtermatrix ist bei separierbaren, ortsunabhängigen Filtern noch stärker strukturiert als im Fall der nur separierbaren Filter. Benötigt man zur Beschreibung eines linearen Filters im allgemeinen Fall $(n \cdot m)^2$ Zahlen, so reichen bei separierbaren Filtern $2 \cdot n \cdot m$ und bei separierbaren, ortsunabhängigen Filtern sogar $n \cdot m$ Zahlen, um das Filter zu beschreiben.

Lineare, ortsunabhängige Filter können auf einfache Weise selbst als Bilder $\in \mathbf{R}^{n\times m}$ statt als Abbildungen $\in \mathbf{R}^{n\times m} \to \mathbf{R}^{n\times m}$ aufgefaßt werden. Dazu dient der Begriff der *Impulsantwort* IA

$$\mathrm{IA}(\mathcal{F}) := F_1 \cdot \delta(1,1) \cdot F_2^{\mathrm{T}},$$

wobei

$$\delta(1,1)_{i,j} = \begin{cases} 1 & \text{für } i=1 \text{ und } j=1 \\ 0 & \text{sonst} \end{cases}$$

der *Einheitsimpuls* ist. Die Impulsantwort ist die gefilterte Version des Einheitsimpulses. Natürlich ist $\mathrm{IA}(\mathcal{F})$ eine Bildmatrix.

Die Wirkung eines linearen, ortsunabhängigen Filters auf ein Bild läßt sich mit Hilfe der Impulsantwort des Filters erhalten, indem man jeden Bildpunkt einzeln filtert und das Ergebnis der Filterung als Summe der so erhaltenen Bilder berechnet. Um die einzelnen Bildpunkte zu filtern, bedient man sich der Impulsantwort.

Für beliebige Muster p gilt

$$\begin{aligned}
(\mathcal{F}(P))_{k,l} &= \left(\sum_{i,j} (P)_{i,j} \cdot \mathcal{F}\left(\operatorname{shift}(i,j)\left(\delta(1,1)\right)\right)\right)_{k,l} \\
&= \left(\sum_{i,j} (P)_{i,j} \cdot \operatorname{shift}(i,j)\left(\mathcal{F}\left(\delta(1,1)\right)\right)\right)_{k,l} \\
&= \left(\sum_{i,j} (P)_{i,j} \cdot \operatorname{shift}(i,j)\left(\mathrm{IA}(\mathcal{F})\right)\right)_{k,l} \\
&= \sum_{i,j} (P)_{i,j} \cdot \operatorname{shift}(i,j)\left(\mathrm{IA}(\mathcal{F})\right)_{k,l} \\
&= \sum_{i,j} (P)_{i,j} \cdot \left(\mathrm{IA}\left(\mathcal{F}\right)\right)_{(k-i)\bmod n,(l-j)\bmod m} \\
&=: \mathrm{IA}(\mathcal{F}) * P
\end{aligned}$$

Diese Operation wird (zyklische) Faltung von $\mathrm{IA}(\mathcal{F})$ und P genannt. Sie beschreibt die Filterung von P durch $\mathcal{F}$ als Überlagerung der getrennten Filterung aller Bildpunkte von P.

Impulsantwort und Faltung lassen sich auch für Bilder in Vektorschreibweise einführen.

Die in diesem Abschnitt gegebene Charakterisierung von Filtern können wir auch für Bildstörungen verwenden. Außer einer rein subjektiven Antipathie seitens des Bildverarbeitenden zeichnet Störungen nichts gegenüber den Filtern aus: Störungen sind die unerwünschten Filter.

1.4.1 Modellierung einer Störung

Wir versuchen, die beim Schreiben durch Ungenauigkeiten entstehenden Variationen durch einen Störungsprozeß wiederzugeben. Dazu stellen wir uns vor, daß für jede Musterklasse stets ein unveränderter idealer Buchstabe B eingegeben wird. Ein Störungsprozeß S erzeugt aus diesem eine gestörte Version $\overline{B}$. Beschränken wir uns auf lineare, ortsunabhängige Störungen, so können wir die Störung durch ihre Impulsantwort $\mathbf{IA}(S)$ charakterisieren und erhalten $\overline{B}$ durch Faltung von $\mathbf{IA}(S)$ und B.

Da es nicht nur eine gestörte Version von B, sondern für jede Variation der Schreibweise eine eigene Version $\overline{B}$ geben muß, können wir S und $\overline{B}$ nicht als einfache Matrizen auffassen. Vielmehr haben wir es mit einer Familie $\mathbf{S} = \{S_i, i \in I\}$ von Störungen zu tun, aus der eine Störung zufällig mit einer gewissen Wahrscheinlichkeit $p(i)$ ausgewählt wird, die aus B durch Faltung ein gestörtes Bild $\overline{B}$ erzeugt. Aus B entsteht also entsprechend eine Familie gestörter Bilder $\overline{\mathbf{B}} = \{\overline{B}_i, i \in I\}$ [5].

Mit $\mathbf{E}(\mathbf{S}) = \sum_i p(i) \cdot S_i$ bezeichnen wir den Erwartungswert der Störung bei vorgegebener Verteilung p. $\mathbf{E}(\mathbf{S})$ ist natürlich wieder eine zweidimensionale Größe und gibt die mittlere Impulsantwort der Störung wieder.

Wir werden Filter entwerfen, die die Störung möglichst gut beheben und hoffen, durch dieses Preprocessing die Variationen der Muster einer Klasse zu verringern. Als Störung nehmen wir vereinfachend die Verschiebung des gesamten Musters um einen geringen, zufälligen Betrag an. Dies ist natürlich eine Idealisierung. Das Normalisierungsverfahren (Abschnitt 1.2.1) wird nämlich in der Regel dafür sorgen, daß die Punkte des Buchstabens am linken unteren Bildrand fast immer die gleiche Lage einnehmen werden. Zum rechten oberen Bildrand hin wird die Lage der Musterpunkte dann immer unbestimmter.

Wie wir später sehen werden, können wir diese Besonderheit der tatsächlichen Störung aber im nachhinein berücksichtigen, indem wir das mit unserer Modellstörung gefundene Filter etwas modifizieren.

1.4.2 Auswahl des Filters

Bei der Auswahl des Preprocessing-Filters stellen die folgenden zwei Zielvorgaben sich widersprechende Extrempositionen dar:

- **Hammingdistanz** Man sieht leicht ein, daß die durch Rumelhart-Netze realisierten Funktionen stetig bezüglich der Metrik $d(x, y) = \sum_{i=1}^{n} |x_i - y_i|$ sind: In einem Rumelhart-Netz werden nur Funktionen zur Berechnung der Ausgabe verwendet, die stetig bzgl. der euklidschen Metrik sind. Also ist auch jede durch ein Rumelhart-Netz berechnete Funktion stetig bzgl. der euklidschen Metrik. Da die euklidsche Metrik sich homöomorph in die obige Metrik überführen läßt, sind die Funktionen auch bzgl. dieser Metrik stetig.

  Die Berechnung dieser Metrik entspricht der bekannten *Hammingdistanz* für binäre Muster, d.h. $A = \{0, 1\}$. Diese Bezeichnung werden wir der Einfachheit halber auch für die allgemeine Form verwenden. Wir ziehen die Hammingdistanz der euklidschen Metrik wegen der geringeren Berechnungskomplexität vor.

[5] Im Hintergrund steht das Konzept des statistischen Prozesses, das wir aber nicht exakt einführen.

Im Rahmen unserer Mustererkennungsaufgabe können wir die Stetigkeit der Netze bzgl. der Hammingdistanz (und natürlich auch bzgl. der euklidschen Metrik) so auffassen, daß zwei Muster umso wahrscheinlicher der gleichen Klasse zugeordnet werden, je geringer ihre Distanz ist.

Um die Annahme zu erhärten, daß die Klassifizierung durch Rumelhartnetze sich im wesentlichen an der Hammingdistanz orientiert, wurde in einem Versuch die mit einem Rumelhartnetz erzielte Erkennungsgüte mit der Erkennungsgüte verglichen, die das auf der Hammingdistanz beruhende Klassifizierungsverfahren erzielt. Dazu wurden je 60 Exemplare aus 50 Klassen von einem Rumelhart-Netz (3 Schichten mit 432, 50 und 50 Neuronen) gelernt.

Die zu erkennenden Buchstaben entstammten einer zweiten Stichprobe, die aus 5 Exemplaren pro Klasse bestand.

Von dieser Stichprobe wurden 97% durch das Rumelhartnetz korrekt klassifiziert. Der nach der Hammingdistanz entscheidende Klassifikator erzielte die gleiche Erkennungsgüte. Die Klassifikation mit der Hammingdistanz erforderte jedoch ein Vielfaches (Faktor einige hundert) der Zeit, die das Rumelhartnetz benötigte.

Der Versuch ist eine deutliche Bestätigung der obigen Vermutung. Er wurde durch zwei weitere Versuche mit 2 bzw. 20 Prototypen weiter untermauert.

Das vorrangige Ziel des zu entwerfenden Filters soll also die Verringerung der Hammingdistanz zwischen gegeneinander verschobenen identischen Mustern sein. Auch bei Verschiebungen um geringe Beträge wird die Hammingdistanz i.a. so groß sein wie die Summe aller Pixel der beiden Muster.

Die Hammingdistanz zwischen gegeneinander verschobenen identischen Mustern läßt sich am wirkungsvollstem mit einem *translationsinvarianten* Filter vermindern. Solch ein Filter liefert für Muster, die sich nur durch ihre Lage unterscheiden, das gleiche Ergebnis. Man sieht allerdings leicht ein, daß translationsinvariante, *lineare* Filter bestenfalls die Anzahl der Musterpunkte zählen können. Strukturinformationen des Musters würden bei einer solchen Filterung völlig verloren gehen.

Deshalb müssen wir beim Entwurf des Filters noch eine weitere Forderung beachten.

- **Auflösung** Das Filter soll auch feine Strukturen noch auflösen können.

  Das Auflösungsvermögen eines Filters wird mit Linienmustern gemessen [**RK**]. Ein Muster, das aus im Abstand λ angeordneten äquidistanten, parallelen Linien besteht, wird gefiltert. Läßt sich die Anzahl der Linien nach der Filterung noch korrekt feststellen —es trat also keine Auslöschung und keine Pseudoauflösung auf— so besitzt das Filter eine Auflösung, die größer als $1/\lambda$ ist.

Die oben für den diskreten Fall eingeführten Begriffe der Bildverarbeitung lassen sich auch für den kontinuierlichen Fall erklären [**RK**]. Mit Hilfe der kontinuierlichen Version läßt sich zeigen, daß die Forderung nach beliebig hoher Auflösung auf ein Filter führt, dessen Impulsantwort mindestens so schnell abfällt wie

$$\mathrm{IA}(F)(x) = \frac{1}{2} \cdot e^{-r}$$

gehorcht, wobei r den Abstand des betrachteten Punktes zum Ursprung bezeichnet. Die Verringerung der Hammingdistanz zwischen Punkten, die die Entfernung r voneinander haben, nimmt dann aber exponentiell mit r ab (siehe Abb. 1.11). Dies ist im Sinne von Punkt 1 unerwünscht. Wir müssen also einen Weg suchen, der einen Kompromiß zwischen den beiden Forderungen darstellt.

Abbildung 1.11: Schon bei Bildpunkten, die nur wenige Pixel gegeneinander verschoben sind, ergibt sich bei der Filterung mit dem Filter $1/2 \cdot e^{-r}$ keine wesentliche Verringerung der Hammingdistanz mehr (schraffierte Fläche).

In der Signalverarbeitungstheorie wurden Methoden entwickelt, die bei bekannten Störungen eine 'möglichst gute' Restaurierung des Bildes gewährleisten. Ziel einer solchen *Optimalfilterung* ist es, die mittlere quadratische Abweichung

$$\mathbf{E}\left(|B - \mathcal{F}(S(B))|^2\right)$$

zu minimieren. Die üblichen Verfahren wurden für Situationen entwickelt, in denen das Signal B durch eine feste Störung S und zusätzliches additives Rauschen beeinträchtig wird. Da wir kein Rauschen annehmen, vereinfacht sich die Theorie. Andererseits müssen wir eine ganze Familie von Störungen berücksichtigen.

Die mittlere Impulsantwort des Störprozesses

$$\mathbf{E}\left(\mathrm{IA}(\mathbf{S})\right)(x,y) = \sum_{i,j} pr(i,j) \cdot \mathrm{IA}(S_{i,j})(x,y)$$

gibt die Wahrscheinlichkeit an, mit der jeder Punkt des Bildes um die Strecke x verschoben wird.

Nach der Störung kann man das natürlich auch als die Wahrscheinlichkeit auffassen, mit der sich das Urbild eines Punktes im Abstand x von der jetzigen Position des Punktes befand.

Um das Bild im Mittel gut zu restaurieren, werden wir jeden Punkt des gestörten Musters also unter Berücksichtigung der Wahrscheinlichkeitsverteilung wieder auf die Umgebung 'verteilen', aus der er stammt. Genauer gesagt werden wir das gestörte Bild mit der mittleren Impulsantwort des Störprozesses falten. Die folgende Rechnung sichert dieses Vorgehen ab:

Es ist bekannt (**[Pap2]**, Kapitel 3+8), daß die mittlere quadratische Abweichung

$$\mathbf{E}\left(|B - \mathcal{F}(\overline{B})|^2\right)$$

durch die Wahl des bedingten Erwartungswertes

$$\mathcal{F}(\overline{B}) = \mathbf{E}(B|\overline{B})$$

minimiert wird. Hier ist $\overline{B}$ ein beobachtetes, gestörtes Muster und B ein mögliches Urbild. Wir bezeichnen mit $S_{i,j}^{-1} = S_{-i,-j}$ die Umkehrabbildung einer Störung und mit $pr(i,j)$ die Wahrscheinlichkeit, mit der das Muster bei der Störung um den Vektor (i,j) verschoben wird, und erhalten

$$\begin{aligned}
\mathbf{E}\left(B|\overline{B}\right)(x,y) &= \sum_{i,j} pr(i,j) \cdot S_{i,j}^{-1}(\overline{B})(x,y) \\
&= \sum_{i,j} pr(i,j) \cdot \overline{B}(x+i,y+j).
\end{aligned}$$

Weil wir $pr(i,j)$ als symmetrische Verteilung ansetzen, ergibt dies[6]

$$\begin{aligned}
&= \sum_{i,j} pr(i,j) \cdot \overline{B}(x-i,y-j) \\
&= \left(pr * \overline{B}\right)(x,y)
\end{aligned}$$

Nun brauchen wir nur noch eine adäquate Verteilung $pr(i,j)$ auszuwählen, um das Preprocessing-Filter vollständig festzulegen.

Schwankungen physikalischer Größen, die sich durch Überlagerung einer Vielzahl einzelner Störungen mit jeweils geringer Auswirkung ergeben, lassen sich naheliegend als normalverteilt annehmen **[HF]**. Wir gehen davon aus, daß sich die Schwankungen beim Schreiben auf unserer Ein-/Ausgabeeinheit in der Regel in einem Bereich von ca. 3 Pixeln bewegen und wählen deshalb

$$\begin{gathered}
pr(i \bmod n, j \bmod m) = N(0;3)(i) \cdot N(0;3)(j), \\
\text{für } -\lfloor\frac{n}{2}\rfloor \leq i \leq \lceil\frac{n}{2}\rceil \\
\text{und } -\lfloor\frac{m}{2}\rfloor \leq j \leq \lceil\frac{m}{2}\rceil
\end{gathered}$$

wobei $N(0;3)$ eine Normalverteilung mit den Parametern $\mu = 0$ und $\sigma = 3$ ist. Pr ist also eine zweidimensionale Normalverteilung mit einer Standardabweichung von 3 Pixeln, deren Zentrum in der linken oberen Matrixecke liegt.

Die Filterung des gestörten Musters $\overline{B}$ mit pr läßt sich auch als

$$\begin{aligned}
(pr * \overline{B})_{k,l} &= \sum_{i,j=-\lfloor\frac{n}{2}\rfloor,-\lfloor\frac{m}{2}\rfloor}^{\lceil\frac{n}{2}\rceil,\lceil\frac{m}{2}\rceil} N\left((k-i) \bmod n\right) \cdot N\left((l-j) \bmod m\right) \\
&\qquad \cdot \overline{B}(i \bmod n, j \bmod m) \\
&= \left[N\left((k-i) \bmod n\right)\right] \cdot \overline{B} \cdot \left[N\left((l-j) \bmod m\right)\right]
\end{aligned}$$

berechnen, wobei die eckigen Klammern die Matrixschreibweise andeuten. $\overline{B}$ wird also mit zwei 'zyklischen' Bandmatrizen multipliziert. Bei unserer Anwendung ist das modulare, zyklische Rechnen nicht angezeigt, da wir den rechten und den linken Matrizenrand bzw. den oberen und den unteren Matrizenrand nicht identifizieren wollen. Wir gehen davon aus, daß die Verschiebung das Muster in der Regel nicht aus der Bildmatrix herausschieben wird und setzen

[6] Wir können hier ruhig die zyklische Faltung verwenden, wenn wir annehmen, daß keine Störung (geschwärzte) Musterpunkte aus der 16×24 Musterbitmap herausschiebt.

die Matrixelemente, deren Indizes in der obigen Rechnung nicht in der Menge $\{(i,j)|1 \leq i \leq n, 1 \leq j \leq m\}$ liegen, auf 0. Wir filtern $\overline{B}$ also durch Multiplikation mit den Matrizen $[N(0;3)(|k-i|)]$ und $[N(0;3)(|l-j|)]^{\mathrm{T}}$.

Da die Filterung ortsunabhängig ist, sind die Filtermatrizen in Toeplitzform (s.o.). Wir hatten bereits erwähnt, daß die Normalisierung der Muster eine *ortsabhängige* Filterung vorteilhaft erscheinen läßt, da die Lage eines Musterpunktes des gestörten Musters bei Musterpunkten, die in der oberen linken Ecke liegen, sicherer vorhersagbar ist, als bei Punkten, die in der rechten oberen Ecke liegen.

Dazu nehmen wir an, daß die Wahrscheinlichkeitsverteilung, die die Verschiebung der Punkte der linken unteren Ecke beschreibt, lediglich eine Standardabweichung von einem Pixel aufweist. Folglich ändern wir die Form der Filtermatrizen also zu

$$\left[N\left(0; 3 - \frac{2}{n-1} \cdot (i-1)\right)(i-k)\right] \qquad 1 \leq i,k \leq n$$

bzw.

$$\left[N\left(0; 3 - \frac{2}{m-1} \cdot (l-1)\right)(j-l)\right]^{\mathrm{T}} \qquad 1 \leq j,l \leq m$$

Der Filterung, die durch diese Matrizen beschrieben wird, läßt sich keine Impulsantwort mehr zuordnen.

1.4.3 Versuche zur Vorverarbeitung

Einige Versuche sollten die Steigerung der Erkennungsgüte, die der Einsatz des gerade beschriebenen Vorverarbeitungsverfahrens bewirkte, messen. Es wurden Versuche mit Prototypmengen durchgeführt, die jeweils von einem einzigen Schreiber stammten. Der durchschnittliche Zuwachs der Erkennungsgüte betrug ca. 10%.

Zum Vergleich wurde ein (nichtlineares) Verfahren herangezogen, das Translationsinvarianz herstellt. Das bekannteste translationsinvariante Filter ist sicher die Berechnung des Leistungsspektrums der Fouriertransformierten eines Musters, welches als

$$(\mathrm{LSF}(B))_{l,k} = \left|\sum_{i,j} B_{i,j} \cdot e^{\mathbf{i} \cdot \left(\frac{i \cdot k}{n} + \frac{j \cdot l}{m}\right)}\right|^2$$

definiert ist. Die Translationsinvarianz des Leistungsspektrums sieht man an folgender Rechnung ein:

$$\begin{aligned}
(\mathrm{LSF}\,(\mathrm{shift}(\Delta i, \Delta j)(B)))_{l,k} &= \left|\sum_{i,j} B_{(i-\Delta i)\mathrm{mod} n,(j-\Delta j)\mathrm{mod} m} \cdot e^{\mathbf{i} \cdot \left(\frac{i \cdot k}{n} + \frac{j \cdot l}{m}\right)}\right|^2 \\
&= \left|\sum_{i,j} B_{i,j} \cdot e^{\mathbf{i} \cdot \left(\frac{(i+\Delta i) \cdot k}{n} + \frac{(j+\Delta j) \cdot l}{m}\right)}\right|^2 \\
&= \left|e^{\mathbf{i} \cdot \left(\frac{\Delta i \cdot k}{n} + \frac{\Delta j \cdot l}{m}\right)}\right|^2 \cdot (\mathrm{LSF}(B))_{l,k} \\
&= (\mathrm{LSF}(B))_{l,k}
\end{aligned}$$

Der erste Faktor des Produktes auf der rechten Seite der Gleichung ist unabhängig von Δi und Δj immer gleich 1.

Die Versuche zeigten, daß die Filterung mit dem Leistungsspektrum geringfügig schlechtere Erkennungsquoten lieferte als die Filterung mit dem entwickelten Optimalfilter. Dies deutet darauf hin, daß die Herstellung von Translationsinvarianz auf Kosten der Phaseninformation eine zu grobe Filtermethode ist. Ein weiterer Nachteil des Leistungsspektrums liegt in dem hohen Rechenaufwand, der zur Durchführung der Filterung nötig ist. Obwohl das schnelle Verfahren nach Cooley-Tookey zur Berechnung der Transformierten verwendet wurde, wurde die Mustererkennung störend langsam. Diesem Nachteil wird durch die Ausführungen des nächsten Abschnittes noch mehr Bedeutung zugemessen.

1.4.4 Schnelle Berechnung linearer Filter bei der Mustererkennung mit Rumelhartnetzen

Die Wirkung eines linearen Filters F auf ein Muster $\vec{b}$ läßt sich durch die Matrixmultiplikation

$$F \cdot \vec{b}$$

berechnen.

Betrachtet man die Vorschrift, nach der die Neuronen der ersten versteckten Schicht eines Rumelhartnetzes ihren Zustand aus der Belegung der Eingabeschicht berechnen, so stellt man einen zweistufigen Ablauf fest:

- Zunächst wird ein Zwischenergebnis in Form eines Vektors $\vec{\tilde{a}}$ durch Multiplikation des Zustandsvektors der Eingabeschicht — der dem Mustervektor $\vec{b}$ entspricht — mit der Konnektivitätsmatrix K der untersten Synapsenschicht berechnet

$$\vec{\tilde{a}} = K \cdot \vec{b}.$$

- Dieses Zwischenergebnis wird komponentenweise durch die Transferfunktion σ auf den Zustandsvektor $\vec{a}$ der Neuronen der ersten versteckten Schicht abgebildet

$$\vec{a} = \sigma\left(\vec{\tilde{a}}\right).$$

Verarbeitet das Netz das gefilterte Muster $F \cdot \vec{b}$, so berechnet sich $\vec{a}$ zu

$$\vec{a} = \sigma\left(K \cdot F \cdot \vec{b}\right).$$

Die Filterung des Musters läßt sich also ohne Mehraufwand während der Verarbeitung des Musters durch das Rumelhartnetz durchführen, indem man statt der Konnektivitätsmatrix K die modifizierte Konnektivitätsmatrix

$$K' = K \cdot F$$

verwendet.

Wir gewinnen sogar noch einen weiteren, gewichtigeren Vorteil: Im Gegensatz zum gefilterten Muster $F \cdot \vec{b}$ wird das ungefilterte Muster $\vec{b}$ nur sehr wenige geschwärzte Pixel (Pixel $\vec{b}_i \neq 0$) besitzen (in der Regel weniger als $\frac{1}{4}$ aller Pixel), weil es nur aus wenigen Linienzügen besteht. Deshalb erzielt man einen bedeutenden Geschwindigkeitsgewinn, wenn bei der Berechnung des Produktes $K' \cdot \vec{b}$ nur die geschwärzten Pixel berücksichtigt werden.

Die Segmentierungs- und Normalisierungsstufe liefert der Mustererkennungsstufe dazu keine vollständigen Bitmaps mehr, sondern Listen der geschwärzten Pixel. Weil bei der Verarbeitung des Listenformates kein Mehraufwand gegenüber der Verarbeitung des Bitmapformates notwendig ist, beobachtet man tatsächlich in etwa eine Geschwindigkeitssteigerung um den Faktor 4. Diese Beschleunigung ist *nach* der Filterung nicht mehr zu erreichen.

Es ist bemerkenswert, daß wir ein Preprocessingverfahren erhalten haben, das laut der durchgeführten Messungen der Berechnung des Leistungsspektrums sogar etwas überlegen ist, gleichzeitig aber keinerlei Mehraufwand gegenüber der Klassifizierung ungefilterter Muster erfordert.

Zur Erstellung einer Netzstruktur lernt das Rumelhartnetz zunächst die *gefilterten* Prototypen. Das berechnete Netz wird dann modifiziert, indem die Teilmatrix der Synapsenmatrix, die die Struktur der untersten Synapsenschicht festlegt, mit der Filtermatrix multipliziert wird.

1.5 Erkennung von gebundener Schrift durch elastischen Mustervergleich

Elastischer Mustervergleich ist ein Verfahren, das ursprünglich zur Spracherkennung entwickelt wurde. Mit Hilfe dynamischer Programmierung werden zwei diskrete Kurvenzüge

$$K_i : [1 \ldots l_{K_i}] \to A$$

möglichst gut zur Deckung gebracht. Dazu wird eine Kostenfunktion

$$d : A^2 \to \mathbf{N}$$

erklärt, die sich auf Worte über A^2 ausdehnen läßt:

$$d(a_1 \ldots a_n, b_1 \ldots b_n) = \sum_{i=1}^{n} \frac{d(a_i, b_i)}{n}$$

Drückt man mit Hilfe einer Relation Π aus, wie K_1 in K_2 deformiert wird, so läßt sich die Güte dieser Deformation bestimmen, indem die Kosten von Π berechnet werden.

Zu K_1, K_2 wird nun eine totale, surjektive und monotone Relation[7]

$$\hat{\Pi}(K_1, K_2) \subset [1 \ldots l_{K_1}] \times [1 \ldots l_{K_2}]$$

gesucht, die die Kosten

$$d\big(\Pi(K_1, K_2)\big) = \sum_{(i,j) \in \Pi(K_1, K_2)} \frac{d\big(K_1(i), K_2(j)\big)}{\#\Pi(K_1, K_2)}$$

[7] Π heißt monoton, wenn $(i,j), (i+1, l), (k, j+1) \in \Pi \Rightarrow l \geq j, k \geq i$ gilt.

über allen Relationen mit den oben genannten Eigenschaften minimiert.

$\hat{\Pi}(K_1, K_2)$ beschreibt, wie man K_1 durch Dehnen und Stauchen am besten auf K_2 abbildet und läßt sich iterativ berechnen durch die Vorschrift

$$\hat{\Pi}(K_1[1\dots i], K_2[1\dots j] = \begin{cases} \hat{\Pi}(K_1[1\dots i-1], K_2[1\dots j-1] \cup (i,j) & \text{oder} \\ \hat{\Pi}(K_1[1\dots i], K_2[1\dots j-1] \cup (i,j) & \text{oder} \\ \hat{\Pi}(K_1[1\dots i-1], K_2[1\dots j] \cup (i,j), & \end{cases} \tag{1.2}$$

wobei die Relation der rechten Seite auszuwählen ist, deren Kosten am geringsten sind. $K_i[1\dots j]$ bezeichnet die Einschränkung von K_i auf den Bereich $[1\dots j]$, falls $j \leq l_{K_i}$ ist.

Abbildung 1.12: Berechnung der optimalen Relation innerhalb der Matchingmatrizen

Die iterative Rechnung läßt sich durch spaltenweises Ausfüllen einer $l_{K_1} \times l_{K_2}$ *Matchingmatrix* durchführen, indem an der Position (i,j) der Matrix die Relation $\hat{\Pi}(K_1[1\dots i], K_2[1\dots j])$ und deren Kosten $d\left(\hat{\Pi}(K_1[1\dots i], K_2[1\dots j])\right)$ eingetragen wird. Man benötigt dazu $l_{K_1} \cdot l_{K_2}$ Rechenschritte.

Die Relation $\hat{\Pi}(K_1[1\dots i], K_2[1\dots j])$ wird in dem Berechnungsschema natürlich nicht explizit, sondern durch einen Verweis auf den in 1.2 ausgewählten Vorgänger und das Paar (i,j) beschrieben. $\hat{\Pi}(K_1[1\dots i], K_2[1\dots j])$ entspricht also einem Weg durch die Matchingmatrix (siehe Abb. 1.13).

Elastischer Mustervergleich läßt sich als Buchstabenerkennungsverfahren verwenden, wenn auf den (geschwärzten) Buchstabenpixeln eine Ordnung erklärt werden kann, die den pixelweisen Vergleich zweier Buchstaben erlaubt **[Tap]**. Das Verfahren wird deshalb vor allem in Versuchsaufbauten eingesetzt, in denen die zeitliche Entstehung der Buchstaben bekannt ist und die Zeit die benötigte Ordnung definiert[8].

Ein zu klassifizierendes Muster B wird elastisch an alle Prototypen P angepaßt. Danach steht der Prototyp $\hat{P}$ mit minimalem Abstand $d\left(\hat{\Pi}(B,P)\right)$ zu B fest. Die Ausgabe der Klassifikation ist also $\hat{P}$.

Als Distanzfunktion d zur Berechnung der Unterschiedlichkeit zweier Kurvenpunkte hat sich die Wahl einer gewichteten Kombination von Ordinatendifferenz, Abszissendifferenz und Differenz der Tangentenwinkel an den betrachteten Punkten eingebürgert.

[8] Ist die zeitliche Entwicklung nicht bekannt, läßt sich eine künstliche Ordnung z.B. durch einen Linienverfolgungsalgorithmus einführen. Lexikographisches Sortieren ist wegen seiner Unstetigkeit bei Variationen der Schreibweise nicht zu empfehlen.

Abbildung 1.13: Die optimale Relation, dargestellt als Weg durch die Matchingmatrix. Es sind nur die Prototypen abgebildet, die in dem erkannten Wort "weg" vorkommen

1.5.1 Segmentierung mit elastischem Mustervergleich

Der wesentliche Vorteil des elastischen Mustervergleichs ist, daß die Segmentierung von Schreibschriftworten in einzelne Buchstaben als Abfallprodukt anfällt, wenn das eben geschilderte Verfahren geringfügig erweitert wird.

Der Algorithmus erhält als Eingabe ein zu klassifizierendes Wort, das aus mehreren zusammenhängenden Buchstaben besteht und eine Menge von Prototyp- Buchstaben. Er berechnet eine optimale Relation

$$\hat{\Pi}\big(\{p_1, \ldots, p_k\}, m\big) = \min_i \hat{\Pi}(p_i, m)$$

zwischen den Punkten des Musters m und den Punkten von Kombinationen der Prototypen $p_1, \ldots, p_k$. Für jeden Prototyp p_i wird eine eigene Relation $\hat{\Pi}(p_i, m)$ bestimmt, die nun eine Teilmenge von

$$\bigcup_i \{p_i\} \times [1 \ldots l_{p_i}] \times [1 \ldots l_m]$$

ist. Der Algorithmus berechnet deshalb für jeden Prototyp eine eigene Matching-Matrix. Die Matching-Matrizen werden für alle Prototypen parallel spaltenweise ausgefüllt. Im Inneren einer Spalte wird dabei die Berechnung in Analogie zu Gleichung 1.2 durchgeführt. Am Beginn einer Spalte berücksichtigt der Algorithmus, daß die kostenminimale Relation das letzte Teilstück des Musters einem anderen Prototypen als dem gerade betrachteten zugeordnet haben kann, indem bei der Suche nach der günstigsten Relation alle Prototypen berücksichtigt werden (siehe Abb. 1.14).

$$\hat{\Pi}_i\big(p_i(1), m(j)\big) = \hat{\Pi}_k(l_{p_k}, j-1) \cup \big((p_i, 1), j\big) \tag{1.3}$$

Der Prototyp p_k ist dabei so gewählt, daß die Kosten von $\hat{\Pi}_k(l_{p_k}, j-1)$ geringer sind als die entsprechenden Kosten der anderen Prototypen am Ende der $j-1$-ten Spalte.

Abbildung 1.14: Berechnung der optimalen Relation am oberen Rand der Matchingmatrizen

Die optimale Relation für das zu erkennende Wort erhält man, indem man am Ende der l_m-ten Spalte der Matching-Matrizen aller Prototypen die optimale Relation auswählt. Diese Relation gibt an, in welcher Reihenfolge welche Prototypen am besten auf das zu erkennende Wort abgebildet werden. Dies bestimmt natürlich auch die Segmentierung des Wortes in Buchstaben.

1.5.2 Beurteilung des Verfahrens

Mustererkennung durch elastischen Mustervergleich ist nur möglich, wenn die Prototypen in dem zu klassifizierenden Wort klar erkennbar sind. Gerade bei der Verwendung von Schreibschrift wird die Schreibweise einzelner Buchstaben in Abhängigkeit des Kontextes, in dem sie auftreten, stark variieren.

Dies drückt sich in gravierenden, ad hoc schwer nachvollziehbaren Erkennungsfehlern aus. Der Einsatz von elastischem Mustervergleich verlangt also nach einer sehr sorgfältigen Schreibweise des Benutzers.

Um Erkennungsfehler soweit wie möglich zu reduzieren, wird elastischer Mustervergleich häufig in Verbindung mit einem Nachbesserungsverfahren angewandt. Dieses Postprocessing versucht unter Zuhilfenahme eines Wörterbuches oder durch Erkennen besonders unwahrscheinlicher Buchstabenkombinationen typische Erkennungsfehler herauszufinden und zu korrigieren.

Da elastischer Mustervergleich an sich schon sehr rechenaufwendig ist[9], fällt der zusätzliche Aufwand für ein Postprocessing aber sehr unangenehm ins Gewicht.

Abbildung 1.15: Beispiel eines zu erkennenden Wortes mit einem Teil der verwendeten Prototypmenge

Abbildung 1.16: Ergebnis des Algorithmus: Segmentierung und Klassifizierung

1.5.3 Beschleunigung des Verfahrens

- **Parallelisierung:** Elastischer Mustervergleich ist sehr gut parallelisierbar. Diese Eigenschaft wurde ausgenutzt, indem das Verfahren im Rahmen des Projektes auf einem Transputer-Netzwerk unter Occam implementiert wurde.

  Jedem Prozessor wurde eine Teilmenge der Prototypmenge zugewiesen. Die Matchingmatrizen der Prototypen werden parallel spaltenweise berechnet. Nachdem jeder Prozessor für alle seine Prototypen eine neue Spalte der Matchingmatrix berechnet hat, synchronisiert er sich mit den restlichen Prozessoren zur gemeinsamen Berechnung des minimalen Eintrags am unteren Ende der Spalte (Siehe Gleichung 1.3). Dazu sind die Prozessoren ringförmig vernetzt. Kompliziertere Netztopologien ließ die verwendete Hardware nicht zu.

  Wegen der etwas enttäuschenden Integer-Performance der Transputer (ein T800 braucht 32 Taktzyklen für eine Integer-Multiplikation, das entspricht bei einem Takt von 20MHz 1,6 ms) waren erst beim Einsatz von sechs Transputern deutliche Geschwindigkeitsgewinne meßbar. Wegen der guten Parallelisierbarkeit des Verfahrens ist beim Einsatz weiterer Transputer ein linearer Speedup zu erwarten.

  Durch die Parallelisierung des Vefahrens wurde das Erkennungsergebnis nicht verändert, da die Berechnung nicht verändert wurde.

- Die **Einschränkung der Gestalt der Matchingmatrix** ist ein weiterer Weg, elastischen Mustervergleich zu beschleunigen. Bereits in der oben zitierten Arbeit zu diesem Thema werden Matching-Beschränkungen formuliert, die aber den Zweck haben, entartete Relationen von der Betrachtung auszuschließen und keine wesentliche Ersparnis an Rechenaufwand mit sich bringen.

[9] Bei 3×26 Prototypen durchschnittlich ca. 2 sec. auf einer 4 Mips-Maschine

In Bild 1.13 sieht man, daß der kostenoptimale Weg nie sehr weit entfernt von der 'Hauptdiagonalen' der Matching-Matrizen verläuft.

Dies legt die Idee nahe, nur ein relativ schmales Band um die Hauptdiagonale der Matchingmatrix zu berechnen.

Abbildung 1.17: Elastischer Mustervergleich bei Einschränkung der Matchingmatrix auf Bandform

Da bei der Berechnung der Matchingmatrix eines Prototypen nicht erkennbar ist, an welcher Position in dem zu erkennenden Muster der Prototyp vorkommt, reicht es nicht, tatsächlich nur ein schmales Band zu berechnen. Statt dessen läuft das Band mehrfach durch die Matchingmatrix, so daß der in Abb. 1.17 erkennbare Bereich der Matrix berechnet werden muß.

Die Ersparnis an Rechenoperationen ist also unabhängig von der Länge des zu erkennenden Musters. Sie fällt aber – natürlich besonders bei kurzen Mustern, die in unserer Anwendung die Regel sein werden – spürbar ins Gewicht.

Sicherlich wird in einzelnen Fällen eine Verschlechterung des Erkennungsergebnisses zu beobachten sein. Die Erkennungsgüte läßt sich jedoch mit Hilfe des Banddurchmessers einstellen und erreicht im Extremfall wieder den Wert des herkömmlichen Verfahrens. Die Einschränkung der Matching-Matrix läßt sich auch 'verfahrenstechnisch' begründen. Man nimmt an, daß die Gangdifferenzen zwischen Muster und Prototypen nur lokaler Natur sind, d.h. Musterpunkte nicht weit verschoben werden brauchen, um auf einen Prototyppunkt zu passen.

1.5.4 Vorverarbeitung für elastischen Mustervergleich

Versuche mit dem Verfahren zeigen, daß eine Normalisierung der Schriftgröße und das Feststellen der (gedachten) Schreiblinie empfehlenswert ist, obwohl aus der Beschreibung des Verfahrens die Notwendigkeit einer Vorverarbeitung bisher nicht zu ersehen ist.

Das Verfahren ist nicht robust gegen Veränderungen der Schriftgröße. Man steht also vor der Wahl, in der Größe abgestufte Prototypen zu verwenden oder die Eingabe durch eine Skalierung auf die Größe der Prototypen zu normieren.

Da der erste Weg zu einer Vergrößerung der Prototypmenge und so zu einer nicht akzeptablen Erhöhung des Rechenaufwandes führt, wird man in einer Preprocessingstufe die Schriftgröße des eingegebenen Musters feststellen und die entsprechende Skalierung vornehmen.

In einem zweiten Schritt wird die Schreiblinie des eingegebenen Musters festgestellt. Dies ist nötig, um die relative Größe der Schwankung der Abszissendifferenz in etwa konstant zu halten und so das Erkennungsergebnis unabhängig von der verwendeten Schreiblinie werden zu lassen.

Die Berechnung der Schrifthöhe und der Schreiblinie kann durch eine Projektion aller Musterpunkte und der Punkte der Verbindungslinien auf die Ordinate des Koordinatensystems durchgeführt werden. Wir setzen die Schreiblinie dabei direkt unterhalb der Stelle an, an der die Ableitung der Projektion ihr Maximum hat, d.h. auf der Unterkante von Buchstaben ohne Unterlänge.

Probeweise wurden in einem weiteren Preprocessingschritt Segmentierungsverbote an Stellen des Musters erteilt, an denen sich mindestens zwei Linien des Musters übereinander befinden. Musterpunkte, die mit einem Segmentierungsverbot belegt wurden, durften bei der Berechnung von Gleichung 1.2 nicht berücksichtigt werden. Es zeigte sich jedoch, daß die Segmentierungsfehler des elastischen Mustervergleichs in großer Mehrheit an Stellen begangen werden, die keinem Segmentierungsverbot unterliegen.

Die erzielte Verbesserung der Erkennungsgüte war also vergleichsweise klein.

1.6 Die Mustererkennungsstufe des Prototypen

Wir haben uns in diesem Abschnitt im wesentlichen zwischen dem konnektionistischen Verfahren auf der Grundlage des Rumelhart-Perzeptrons und dem nicht-konnektionistischem Verfahren mit Hilfe von elastischem Mustervergleich zu entscheiden. Mustererkennung durch Hopfieldnetze scheidet wegen der Ergebnisse in Abschnitt 1.2.3 als Wahlmöglichkeit aus.

Beide Verfahren weisen Vor- und Nachteile auf. Rumelhartnetze arbeiten im Erkennungsmodus sehr schnell (ca. 5 Zeichen pro Sekunde) und bieten eine ausreichende Erkennungsgüte im Bereich zwischen 90% und 97%. Das Lernverfahren ist sehr rechenaufwendig und nimmt bei 300 Prototypen Rechenzeiten von 1-2 Tagen in Anspruch (allerdings auf einer Maschine mit nur mäßiger Floating Point Leistung von ca. 0.4 MFLOP/s).

Elastischer Mustervergleich kommt quasi ohne Lernverfahren aus, d.h. das Training des Verfahrens beschränkt sich auf das Aufschreiben der Prototypen in Form von einigen Kunstwörtern und deren manueller Segmentierung. Die Erkennungsgüte ist bei Einzelbuchstaben mit ca. 90% akzeptabel. Das herausragende Merkmal des elastischen Mustervergleichs ist seine Fähigkeit, zusammenhängende Worte zu segmentieren. Der gravierende Nachteil des Verfahrens ist die langsame Erkennung, die aus dem hohen Rechenaufwand resultiert.

Elastischer Mustervergleich wird deshalb in dem Prototypen noch nicht verwendet.

Zunächst scheint eine gemischte Mustererkennung denkbar, die Einzelbuchstaben mit Hilfe eines Rumelhart-Perzeptrons schnell erkennt und zusammenhängende Worte mit elastischem Mustervergleich klassifiziert.

Dazu müßte die Vorverarbeitungsstufe so erweitert werden, daß sie Einzelzeichen von zusammenhängenden Worten unterscheiden kann und beide Arten von Mustern an den jeweils passenden Algorithmus weiterleitet. Diese Unterscheidung müßte heuristisch auf der Ebene der Koordinatenpunkte getroffen werden und wäre damit sicher fehlerbehaftet (wie unterscheidet man z.B. ohne Kenntnis der Bedeutung zwischen dem Schreibschriftwort ‘el’ und dem Einzelbuchstaben ‘d’?). Man sieht, daß die Unterscheidung zwischen Worten und Einzelbuchstaben genauso schwer ist, wie die eigentliche Mustererkennung.

Deshalb scheint eine Mischform der beiden Erkennungsalgorithmen nicht ratsam zu sein.

Kapitel 2

Syntaktische Analyse zweidimensionaler Strukturen

Die Bedeutung einer mathematischen Formel läßt sich aus der Bedeutung der einzelnen Symbole und der Konstellation der Zeichen, d.h. der Struktur der Formel ablesen.

Der im vorigen Kapitel beschriebenen Stufe zur Mustererkennung — bestehend aus Segmentierung, Normalisierung und Klassifizierung — muß sich also eine syntaxgesteuerte Übersetzung der Formel anschließen, die die Bedeutung der Formel aus ihrer Struktur erschließt.

Eine bei der Syntaxanalyse eindimensionaler Strukturen übliche Vorgehensweise ist die Vorgabe eines Konstruktionsmechanismus — z.B. in Form einer kontextfreien Grammatik [1] — mit deren Hilfe alle Elemente der betrachteten *Welt* — in unserem Fall also die Menge aller zulässigen mathematischen Formeln — erzeugt werden können. Der Konstruktionsmechanismus muß so beschaffen sein, daß auch die Analyse von Elementen mit vertretbarem Aufwand möglich ist.

Oft benutzt man eine nützliche Erweiterung des Formalismus, die *Attributierung.* Attributierte Grammatiken verknüpfen mit jeder Produktion der Grammatik eine Anzahl von Berechnungsregeln, mit deren Hilfe sich *ererbte Attribute* der rechten Seite der Produktion aus ebensolchen der linken Seite und *abgeleitete Attribute* der linken Seite der Produktion sich aus ebensolchen der rechten Seite berechnen lassen.

Die Übersetzung einer Struktur erfolgt durch ihre Analyse und schrittweise Synthese der übersetzten Version in Abhängigkeit von den verwendeten Konstruktionsregeln. Die Übersetzung einer Struktur stellt damit ein spezielles Attribut dar.

Bei der Analyse eindimensionaler Strukturen hat sich die Verwendung eingeschränkter kontextfreier Sprachen eingebürgert, die eine deterministische Analyse durch LL(k), LR(k) oder LALR(k) Parser erlauben. Ein wichtiger Aspekt ist hier die systematische und damit automatisierbare Konstruktion des Parsers aus der zugrundeliegenden kontextfreien Grammatik.

[1] Das Konzept kontextfreier Grammatiken im eindimensionalen Fall und die weiter unten genannten Parsingstrategien setzen wir als bekannt voraus [**AU**].

Der Analyse zweidimensionaler Strukturen stehen solche systematischen Ansätze bislang nicht zur Verfügung. Die zur *Konstruktion* eindimensionaler Objekte verwendeten Kontrollmechanismen lassen sich zwar relativ leicht auf den zweidimensionalen Fall verallgemeinern (Netzgrammatiken, PLEX-Grammatiken [**GT**] und Array-Grammatiken [**Ros3**], es stehen aber keine systematischen Methoden zur Syntax*analyse* zur Verfügung.

2.1 Ein Konzept zweidimensionaler kontextfreier Grammatiken

Wir geben zunächst ein Grammatikkonzept an, mit dessen Hilfe zweidimensionale Strukturen, die aus Rechtecken aufgebaut sind, generiert werden können. Dabei beschränken wir uns auf Strukturen, die aus isoorientierten Rechtecken bestehen und verlangen zusätzlich, daß sich jeweils zwei Rechtecke entweder überhaupt nicht schneiden oder eines der beiden echt in dem anderen enthalten ist.

Diese zusätzliche Einschränkung orientiert sich zunächst an unserer Problemstellung mathematische Formeln zu analysieren, in denen sich die (umfassenden Rechtecke der) Zeichen nicht überlappen sollten[2]. Sie entspricht aber auch der Klammerstruktur eindimensionaler kontextfreier Sprachen.

Der Kernpunkt unseres Konzeptes für zweidimensionale Grammatiken besteht darin, daß die relative Lage der Symbole, die auf der rechten Seite einer Produktion auftreten, nicht nur (wie im eindimensionalen Fall) durch *eine* Ordnungsrelation ("links von"), sondern zusätzlich durch eine zweite Ordnungsrelation ("oberhalb von") charakterisiert wird.

Im eindimensionalen Fall ist die relative Lage von je zwei Symbolen bekannt, wenn die relative Lage aller direkt benachbarten Symbole bekannt ist. Bei der Verallgemeinerung zur zweidimensionalen Grammatik stehen wir vor der Wahl, nur die relative Lage der direkt benachbarten Symbole festzulegen oder die relative Lage von je zwei Symbolen zu spezifizieren.

Wir entscheiden uns hier für die erste Variante, da sie zur Beschreibung von Formeln natürlicher erscheint (beim Schreiben werden Zeichen an ihren Nachbarn ausgerichtet) und weil die Gültigkeit der lokalen Lagerelationen bei der Syntaxanalyse einfacher zu überprüfen ist.

In Analogie zur Syntaxanalyse eindimensionaler Strukturen versuchen wir, den Begriff der zweidimensionalen Struktur zu fassen, ohne eine Vektorraumstruktur auf der Menge der Punkte vorauszusetzen.

Wir beschränken uns bereits in der folgenden Definition auf isoorientierte Rechtecke. Diese Einschränkung ist nicht zwingend, entspricht aber unserer Problemstellung und vereinfacht die Syntaxanalyse.

Definition 2.1 (n-dimensionale Strukturen) *Eine n-dimensionale Struktur ist ein Paar $(A, \mathcal{L})$, wobei A eine Menge von "rechteckförmigen" Zeichen $(A_1, 1), \ldots, (A_n, n)$ ist*[3].

[2] Schwaches Unterschneiden von Zeichen ist akzeptabel und läßt sich durch geringfügiges Verkleinern der umfassenden Rechtecke kaschieren.

[3] Die zusätzlichen Indizes haben nur den Zweck, den Zeichen eindeutige Namen zu verleihen.

$\mathcal{L}$ ist die disjunkte Summe von n Quasiordnungen *(die den Lagerelationen entsprechen) auf der Menge der Rechtecke. Unter Quasiordnung verstehen wir dabei eine Relation, die reflexiv und transitiv, aber nur in einem schwachen Sinne antisymmetrisch ist. Wir fordern lediglich*

$$(b_1 \leq \cdots \leq b_i \leq b_{i+1} \leq \cdots \leq b_n \text{ und } b_{i+1} \not\leq b_i) \Longrightarrow b_n \not\leq b_1.$$

Zwei n- dimensionale Strukturen, A_1 und A_2, sind gleich, wenn durch eine (bijektive) Umbenennung der Symbole erreicht werden kann, daß $A_1 = A_2$ und $\mathcal{L}_1 = \mathcal{L}_2$ gilt.

Abbildung 2.1: Direkt benachbarte Rechtecke (Der schraffierte Bereich darf von keinem weiteren Rechteck geschnitten werden)

Definition 2.2 (Direkt benachbarte Rechtecke) *Gegeben sei eine zweidimensionale Struktur. Zwei Rechtecke der Struktur heißen* direkt benachbart, *wenn ihre Lage durch eine der in Bild 2.1 skizzierten Figuren oder deren Spiegelungen und rechtwinklige Drehungen beschrieben wird.*

Definition 2.3 (Zweidimensionale kontextfreie Grammatiken) *Eine zweidimensionale kontextfreie Grammatik wird durch ein Tupel (N, T, P, S) beschrieben. Sie besteht aus zwei Mengen von ('rechteckförmigen') Zeichen — der Menge N der Nichtterminalzeichen und der Menge T der Terminalzeichen —, dem Startsymbol $S \in N$ und einer Menge P von Produktionen*

$$A \longrightarrow \Big(\big\{(B_1, 1), (B_2, 2), \ldots, (B_n, n)\big\}, \mathcal{L}\Big).$$

Wir verlangen, daß $A \in N$ und $B_i \in N \cup T$ ist.

Zur Beschreibung der relativen Lage der Rechtecke, die zur rechten Seite einer Produktion gehören, reicht es, die relative Lage der linken oberen Eckpunkte, $\lceil(B_i, i)$, und der rechten unteren Eckpunkte, $(B_i, i)\rfloor$, aller Rechtecke der Produktion anzugeben. Dies geschieht mit der Lagerelation

$$\begin{aligned} \mathcal{L} \quad \subset \quad & \{\lceil(B_1, 1), (B_1, 1)\rfloor, \ldots, \lceil(B_n, n), (B_n, n)\rfloor\}^2 \times \{\mathit{oberhalb}\} \qquad \cup \\ & \{\lceil(B_1, 1), (B_1, 1)\rfloor, \ldots, \lceil(B_n, n), (B_n, n)\rfloor\}^2 \times \{\mathit{linkerhand}\}. \end{aligned}$$

Die rechten Seiten der Produktionen sind also zweidimensionale Strukturen. An die Lagerelation $\mathcal{L}$ jeder Produktion stellen wir folgende Bedingungen:

- *Die relative Lage aller direkt benachbarten Rechtecke wird vollständig bis auf die unten genannten Ausnahmen angegeben.*
- *Die trivialen Tripel $((\lceil A, A\rfloor), \mathit{oberhalb})$ und $((\lceil A, A\rfloor), \mathit{linkerhand})$ werden nicht aufgeführt, sondern gelten implizit. Außerdem werden Tripel, die sich aus der Transitivität der Lagerelationen schließen lassen, nicht aufgeführt.*

Die in Abbildung 2.2 gezeigte Produktion würde in der Schreibweise der letzten Definition etwa folgendermaßen aussehen:

Abbildung 2.2: Beispielproduktion

$$
\begin{aligned}
A \longrightarrow \Big(\; & \{(A,1),(A,2),(B,3)\}, \\
& \{\, ((A,1)\rfloor, \lceil(A,2), \mathrm{linkerhand}), (\lceil(A,1), \lceil(A,2), \mathrm{oberhalb}), \\
& \quad (\lceil(A,2),(A,1)\rfloor, \mathrm{oberhalb}), ((A,1)\rfloor, (A,2)\rfloor, \mathrm{oberhalb}), \\
& \quad ((A,1)\rfloor, \lceil(B,3), \mathrm{oberhalb}), (\lceil(A,1), \lceil(B,3), \mathrm{linkerhand}), \\
& \qquad\qquad\qquad \ldots \\
& \}\Big)
\end{aligned}
$$

Das Anwenden einer Produktion auf eine Menge von Zeichen, deren relative Lage durch eine Lagerelation beschrieben wird, geschieht durch Ersetzen der linken Seite der Produktion durch die rechte Seite der Produktion. Dabei ändert sich natürlich auch die Lagerelation.

Ersetzt man in der zweidimensionalen Struktur

$$(\{(A_1,1),\ldots,(A_i,i),\ldots,(A_n,n)\}, \mathcal{L})$$

das Symbol (A_i,i) durch $(\{(B_1,1),\ldots,(B_m,m)\}, \mathcal{L}')$, so erhält man

$$(\{(A_1,1),\ldots,(A_n,n),(B_1,n+1),\ldots,(B_m,m+n)\}, \mathcal{L}'')\,.$$

Wir ersparen uns eine größere Formel zur Definition von $\mathcal{L}''$ und geben kurz das Rezept an, nach dem die Relation modifiziert wird:

> Aus $\mathcal{L}$ werden zunächst alle Tripel entfernt, in denen ein Punkt von (A_i,i) vorkommt. Anschließend fügen wir zu der Relationenmenge Tripel hinzu, die sicherstellen, daß keins der Elemente, die für (A_i,i) eingesetzt werden, den Bereich verläßt, den (A_i,i) einnahm:
>
> Für jedes Element am linken Rand der rechten Seite der Produktion[4] und jedes Tripel $((A_l,l), \lceil(A_i,i), linkerhand)$, das ursprünglich in $\mathcal{L}$ enthalten war, fügen wir das Tripel $((A_l,l), \lceil(B_j,j), linkerhand)$ zu $\mathcal{L}''$ hinzu. Verfährt man analog mit den übrigen drei Rändern der rechten Seite der Produktion, so erhält man $\mathcal{L}''$.

Die Relation $\mathcal{L}''$ ist im allgemeinen keine vollständige Beschreibung der zweidimensionalen Struktur $(\{A_1,1),\ldots,(B_m,n+m)\}, \mathcal{L}'')$. Außerdem enthält sie unter Umständen überflüssige Tripel.

Die Unvollständigkeit der Relation spiegelt die Tatsache wider, daß die rechte Seite einer Produktion lediglich unter Berücksichtigung der *inneren* Struktur der rechten Seite der Produktion in die von dem ersetzten Nichtterminal belegte Fläche eingebettet wird, ohne daß zusätzlich Bedingungen an die Plazierung der Symbole relativ zu den Symbolen gestellt werden, die außerhalb der von

[4]Das sind alle Elemente (B_j,j), für die es in $\mathcal{L}$ kein Tripel $((B_k,k), \lceil(B_j,j), \mathrm{linkerhand})$ gibt

dem Nichtterminal belegten Fläche liegen. Solche Bedingungen würden das Prinzip der Kontextfreiheit durchbrechen und sollten bei Bedarf in Form von Kontextbedingungen formuliert werden.

Eine zweidimensionale Struktur gehört zu der Sprache, die von einer zweidimensionalen Grammatik produziert wird, wenn sie durch Wegstreichen der überflüssigen Tripel und Hinzufügen der fehlenden notwendigen Tripel aus einer zweidimensionalen Struktur entsteht, die durch Anwenden von Produktionen der Grammatik aus dem Startsymbol erzeugt werden kann.

2.2 Klassifizierung der Parser nach der Eingabestrategie

Der Rechenaufwand, den LL(k), LR(k) und LALR(k)-Parser zur Analyse von Worten benötigen, die zu einer deterministischen, eindimensionalen Sprache gehören, hängt nur linear von der Länge des zu analysierenden Wortes ab. Dies wird dadurch erreicht, daß jedes Zeichen des Wortes nur ein einziges Mal betrachtet wird und zur Verarbeitung jedes Zeichens nur eine feste Anzahl von Schritten durchzuführen ist.

Die Reihenfolge, in der die Zeichen betrachtet werden, legt auf den Zeichen eine lineare Ordnung fest. Es ist nicht möglich, zweidimensionale Objekte stetig linear anzuordnen[5]. Im allgemeinen werden bei der Errichtung einer linearen Ordnung benachbarte Zeichen auseinandergerissen. In unserem Fall werden räumlich benachbarte Zeichen zeitlich (evtl. weit) auseinanderliegend eingelesen.

Wenn die rechten Seiten der Produktionen während des Analysevorganges durch einen endlichen Automaten lokal erkannt werden sollen, ist es nötig, die evtl. zeitlich auseinanderliegend eingelesenen Bestandteile der rechten Seite einer Produktion auf dem Stack wieder zusammenzufügen.

Der Stack eines Parsers für zweidimensionale Grammatiken muß deshalb auch zweidimensional sein.

Diese Erweiterung des Parsingkonzeptes gegenüber dem eindimensionalen Fall beseitigt jedoch nicht alle Schwierigkeiten, die bei der Analyse zweidimensionaler Strukturen aufgeworfen werden. Um die grundsätzliche Natur der Probleme anzudeuten, diskutieren wir im folgenden Abschnitt informal alle möglichen Eingabestrategien, die ein Parser bei der Verarbeitung einer zweidimensionalen Struktur befolgen kann.

Abbildung 2.3: Klassifizierung der Parsingstrategien nach der Eingabereihenfolge

Jede der Möglichkeiten führt zu einer eigenen Parsingstrategie. Zwischen diesen unterschiedlichen Strategien muß in Abhängigkeit von der zu analysierenden Sprache abgewogen werden. Es geht in der folgenden Diskussion also

[5] Übersetzt in unser Konzept von n-dimensionalen Strukturen bedeutet Unstetigkeit, daß es bei eindimensionaler Anordnung eines zweidimensionalen Objektes i.a. Zeichen geben wird, deren Reihenfolge einer der beiden Ordnungsrelationen zuwiderläuft. Diese Begriffsbildung ergibt sich auch, wenn aus den beiden Quasiordnungen auf die naheliegede Weise eine Metrik auf der Menge der Rechteckpunkte konstruiert wird.

nicht so sehr um die Entscheidung für den optimalen zweidimensionalen Parser, sondern um das Abstecken des Entscheidungsspielraumes.

In der folgenden Systematik werden wir nur *deterministische* Parser betrachten. Parser dieser Kategorie wählen die Eingabereihenfolge nur in Abhängigkeit von der Eingabeinstanz aus. Die Eingabereihenfolge, mit der die Zeichen *einer* Eingabeinstanz eingelesen werden, ist fest, d.h. ändert sich in unterschiedlichen Analysedurchläufen nicht.

Wir verschärfen diese (übliche) Charakterisierung noch und verlangen, daß die Eingabereihenfolge nur von der Art der Zeichen der Eingabeinstanz und der *relativen* Lage der Zeichen (links von, oberhalb von) abhängen darf. Die Eingabereihenfolge darf sich insbesondere nicht erst unter Kenntnis der *exakten* Lage der Zeichen in der Ebene bestimmen lassen. Natürlich ist jegliche Art von Glücksspiel verboten, d.h. die Eingabereihenfolge hängt auch nicht von externen Parametern ab.

Auf den ersten Blick erstaunt die Vernachlässigung der exakten Lage der Zeichen vielleicht etwas. Diese Einschränkung läßt sich aber mit der gewünschten Analogie zu den Parsing-Strategien im eindimensionalen Fall rechtfertigen. In Fällen, in denen die Kenntnis der exakten Lage der Zeichen zur Strukturanalyse notwendig ist, läßt sich das Konzept leicht erweitern, indem eine Attributierung der Grammatik vorgenommen wird und die Aktionen des Parsers in Abhängigkeit von Prädikaten über den Attributwerten durchgeführt werden.

2.2.1 Feste Eingabestrategien

Die Parser dieser Kategorie wählen das nächste einzulesende Zeichen unabhängig von den vorher eingelesenen Zeichen aus. Das nächste einzulesende Zeichen hängt also weder von der Anzahl noch von der Art der vorher eingelesenen Zeichen ab.

Da in diesem Fall nur noch die relative Lage als Ordnungskriterium zur Verfügung steht und aus den beiden Relationen "links von" und "oberhalb von" eine totale Ordnung konstruiert werden muß, können deterministische Parser mit fester Eingabestrategie nur eine der acht Ordnungen verwenden, die sich durch lexikographisches Sortieren gemäß den beiden Lagerelationen ergeben.

Die feste Eingabestrategie führt dazu, daß der Parser eine a priori nicht beschränkte Anzahl von Zeichen lesen muß, die er zur Erkennung der rechten Seite einer Produktion nicht benötigt (siehe Abbildung 2.4).

Abbildung 2.4: Bei fester Eingabestrategie (hier: von links-oben nach rechts-unten) werden evtl. Teile der Struktur gelesen, die zur Reduktion einer Produktion nicht nötig wären.

Bei der Konstruktion des Parsers steht man vor der Möglichkeit, die überflüssigerweise gelesenen Zeichen wieder in die Eingabe zurückzulegen (d.h. sie zu vergessen und später erneut zu lesen, oder sie im Stack zu speichern und später erneut zu lesen), sobald die rechte Seite einer Produktion reduziert werden konnte, oder auch die überflüssigen Zeichen in den Zuständen des Automaten, der die Aktionen des Parsers steuert, zu speichern.

Im ersten Fall müßten Teile der Eingabe wiederholt gelesen werden. Es lassen sich Grammatiken konstruieren, deren Satzformen nur mit (in der Anzahl der Zeichen) quadratischem Aufwand analysiert werden könnten.

Im zweiten Fall kann die Anzahl der aktiven Items [**AU**] beliebig groß werden, so daß der Steuerautomat i.a. nicht mit einer endlichen Anzahl von Zuständen konstruiert werden kann.

2.2.2 Flexible Eingabestrategien

Wir gestatten dem Parser nun, die Entscheidung für das nächste einzulesende Zeichen in Abhängigkeit von den bisher eingelesenen Zeichen zu treffen.

Der Parser hat die Möglichkeit, die rechte Seite einer bestimmten Produktion bevorzugt abzufertigen, bevor er den Rest der Eingabe betrachtet. Die neue Freiheit ist aber gleichzeitig mit einem neuen Entscheidungsdruck verbunden. Wenn der Parser effizienter arbeiten soll als die Parser, die eine feste Eingabestrategie verfolgen, muß er sich in Extremfällen anhand eines Zeichens für die anzuwendende Produktion entscheiden.

Dies beschränkt die Mächtigkeit der Parsingstrategie von vornherein auf die Mächtigkeit des LL(k)-Parsings. Bei dieser top-down-Strategie muß die anzuwendende Produktion ebensfalls anhand der ersten k Zeichen, die reduziert werden sollen, getroffen werden.

2.3 Ein Graphreduktionsalgorithmus zur Analyse zweidimensionaler Strukturen

Die Syntaxanalysestufe des Prototypen wurde zweistufig ausgelegt.

Die erste Stufe legt die Bestandteile der eingegebenen Formel in einem zusammenhängenden Graphen ab, der die räumliche relative Lage der Bestandteile widerspiegelt. Eine zweite Stufe — die eigentliche Graphreduktion — reduziert den von der ersten Stufe produzierten Graphen schrittweise in Abhängigkeit der zugrundeliegenden zweidimensionalen Grammatik.

Der Algorithmus, die Implementierung und die Grammatik, die die Formeln beschreibt, die der Prototyp verarbeiten kann, werden ausführlich in [**Kap**] beschrieben. Hier werden nur die wesentlichen Charakteristika skizziert.

2.3.1 Aufbau des Grundgraphen

Die erste Stufe des Syntaxanalysemoduls liest die Zeichen in der Reihenfolge ihres zeitlichen Auftretens ein, d.h. in der Reihenfolge, in der sie der Schreiber auf der Eingabeeinheit schreibt.

Folgen wir unserer Systematik der Eingabestrategien, so handelt es sich bei unserem Graphreduktionsalgorithmus also um einen nichtdeterministischen Parser. Wie wir später sehen werden, gilt diese Einstufung auch, wenn man die Graphaufbaustufe als eine Art Preprocessing ansieht und nur die Graphreduktionsstufe als Parser im engeren Sinne gelten läßt.

Die Berücksichtigung der Reihenfolge, in der die Zeichen geschrieben wurden, geht von dem Prinzip der Stetigkeit aus: In der Regel sind zeitlich aufeinanderfolgend geschriebene Zeichen räumlich ebenfalls benachbart. In diesem Fall läßt sich ein neues Element besonders leicht in den Grundgraphen einbauen, da es nur mit seinem zeitlichen Vorgänger verknüpft wird.

Jeder Knoten des Graphen trägt die wesentliche Informationen über ein Symbol der Formel: Art des Symbols, Abmessungen und (absolute) Lage durch Spezifizierung zweier Eckpunkte, zusätzlich einige Attributfelder.

Darüberhinaus verfügt jeder Knoten über acht Verweise, die auf Nachbarknoten gerichtet sein können oder als unbenutzt gekennzeichnet sind. Die acht Verweise entsprechen den acht Nachbarfeldern eines Quadrates bei Einteilung der Ebene in ein Gitter, das aus Quadraten besteht. Die Verweise zweier benachbarter Elemente sind umkehrbar. Zeigt Verweis i des ersten Elementes auf das zweite Element, so zeigt Verweis $(i+4) \bmod 8$ des zweiten Elementes auf das erste Element.

Der Grundgraph ist also aus symmetrischen Kanten aufgebaut, d.h. ungerichtet. Wir sorgen beim Aufbau des Graphen in der Regel dafür, daß keine Kante ein Element überspringt. Jeder von einem Knoten ausgehende Verweis ist auf das nächste in der entsprechenden Richtung liegende Element gerichtet oder als unbenutzt markiert.

Obwohl wir die *stetige Fortsetzung*[6] des Grundgraphen als den Regelfall beim Niederschreiben mathematischer Formeln ansehen, müssen wir Sonderregelungen für Unstetigkeitsstellen treffen. Diese treten in ihrer harmlosesten Form als *natürliche Unstetigkeiten beim Schreiben* auf, z.B. wenn beim Schreiben eines Bruches zunächst der Zähler geschrieben wird und dann der Nenner begonnen wird. In diesem Fall wird durch globale Suche der nächste Nachbar des neuen Zeichens festgestellt und das neue Zeichen mit Hilfe der passenden Verweise in den Grundgraphen eingebaut.

Der Unterschied zur Normalfallbehandlung (stetiges Weiterschreiben) besteht darin, daß die Normalfallbehandlung das neu eingelesene Zeichen nicht notwendigerweise mit dem nächsten Nachbarn verknüpft. Statt dessen wird das neue Zeichen mit seinem zeitlichen Vorgänger verknüpft, solange dies vertretbar ist, insbesondere solange die — relativ zur Größe der Zeichen gemessene — Entfernung beider Zeichen nicht zu groß wird. Die Normalfallbehandlung kommt mit lokalen Untersuchungen des Grundgraphen aus.

Überschreibt der Benutzer ein Zeichen durch ein anderes, so stellt der Algorithmus zum Graphaufbau zunächst eine Unstetigkeit beim Schreiben fest. Das überschriebene Zeichen ergibt sich bei der Suche nach dem nächsten Nachbarn und wird durch das neue Zeichen ersetzt. Alle Verweise des überschriebenen Zeichens werden an das neue Zeichen vererbt. Gleichzeitig wird überprüft, ob das neue Zeichen nicht noch weitere Zeichen berührt und somit eine unzulässige Konfiguration vorliegt.

Sollen mehrere Zeichen überschrieben oder gelöscht werden, so muß der Benutzer eine *Radieroperation* durchführen. Die ausradierten Symbole werden

[6] Die räumliche Nachbarschaft der Zeichen entspricht der zeitlichen Nachbarschaft.

dann aus dem Grundgraphen entfernt. Weil die lokale Reparatur der Graphstruktur unter Umständen sehr kompliziert sein kann[7], wird der Graph mit den verbleibenden Symbolen vollständig neu aufgebaut. Als Eingabereihenfolge wird weiterhin die Reihenfolge, in der der Benutzer die Zeichen eingab, verwendet.

Nach dem Wiederaufbau steht der Grundgraph für Operationen in einem der vier geschilderten Modi zur Verfügung.

2.3.2 Aufgabenteilung zwischen Graphaufbau und Graphreduktion

Die Aufgabenverteilung zwischen den beiden Stufen der Syntaxanalyse wurde von der Philosophie bestimmt, möglichst wenige Entscheidungen bereits während des Graphaufbaus zu treffen und möglichst alle (unsicheren) Entscheidungen bis zur Graphreduktion aufzuschieben.

Diese Philosophie verdient diese Bezeichnung, da während des Graphaufbaus noch Änderungen des Graphen durch Korrekturen zu erwarten sind und die bereits getroffenen Entscheidungen evtl. mit großem Aufwand revidiert werden müssen.

Entscheidungen, die erst während der Reduktionsphase gefällt werden, können zumindest schon auf einen Teil der (i.a. nicht mehr lokalen) Strukturinformationen zurückgreifen und sind deshalb mit geringerer Wahrscheinlichkeit fehlerbehaftet.

Versuche zeigten, daß es keinen Sinn hat, während des Graphaufbaus mehr Information über die Struktur der eingegebenen Formel zu berechnen, als die relative Lage von jeweils zwei benachbarten Symbolen und die aus der lokalen Perspektive am günstigsten erscheinenden Verweise. Versuchsweise zusätzlich berechnete Informationen waren unter Umständen sehr fehlerhaft, so daß in der Reduktionsphase eine aufwendige Fehlerbehandlung nachgeschaltet werden mußte, statt eine Ersparnis an Aufwand zu erzielen.

Die augenblickliche Version des Syntaxanalysealgorithmus stellt an die Graphaufbauphase nur geringe Anforderungen. Dies geht so weit, daß im Extremfall leichte Fehler toleriert werden, die in der Reduktionsphase wieder behoben werden können.

Während des Graphaufbaus werden — außer in den oben genannten Ausnahmefällen — nur lokale Betrachtungen angestellt. Die nicht-lokalen Operationen beschränken sich auf das Auffinden des nächsten Nachbarn mittels Durchmustern des kompletten Graphen bei Unstetigkeiten und den kompletten Neuaufbau des Graphen.

2.3.3 Reduktion des Grundgraphen

Wir haben weiter oben schon bemerkt, daß der Aufbau des Grundgraphen eine Art Aufbereitung der Eingabe darstellt. Die eigentliche Syntaxanalyse der Formel findet in der zweiten Stufe des Analysealgorithmus, der Graphreduktion, statt, nachdem der Benutzer die Eingabe der Formel abgeschlossen hat.

[7]Wenn der Grundgraph zusammenhängend sein soll, ist die lokale Reparatur sogar unmöglich, da Zusammenhang nicht lokal überprüft werden kann.

Die Graphreduktion gehorcht einem festen Rhythmus: Feststellen einer Stelle des Graphen, an der die rechte Seite einer Produktion der zugrundeliegenden Grammatik vorliegt und darauffolgend die Reduktion der rechten Seite der Produktion auf die linke Seite der Produktion. Der Graph wird schrittweise verkleinert, bis nur noch ein Nichtterminal, das Startsymbol der Grammatik, vorhanden ist, oder die Reduktion als fehlerhaft abgebrochen werden muß.

Der eigentlich schwere Teil eines Reduktionsschrittes besteht im Feststellen der Reduktionsstelle und — in engem Zusammenhang damit — auch in der Bestimmung der anwendbaren Produktion. Die reine Reduktion auf das Nichtterminal ist vergleichsweise naheliegend, erfordert aber einen geschickten Entwurf der Grammatik.

Potentielle Reduktionsstellen werden durch eine Heuristik aufgefunden: Kandidaten für Stellen im Graph, an denen eine Produktion reduziert werden kann, wird man dort vermuten, wo Symbole der eingebenen Formel (oder Nichtterminale, die bereits reduzierte Teilformeln repräsentieren) eng benachbart sind.

Für die Graphreduktion werden die Kanten des Graphen deshalb mit der Entfernung (gemessen von Eckpunkt zu Eckpunkt der beteiligten Symbole) markiert und entsprechend der Markierung sortiert. Beginnend bei den "kurzen" Kanten wird die Menge der Kanten nun dieser Ordnung gemäß durchmustert, bis eine Reduktionsstelle gefunden wurde.

Die Produktion der kontextfreien Grammatik, die an einer Stelle des Graphen evtl. vorliegt, kann durch Betrachtung der Knoten, die die gerade aktuelle Kante verbindet, und — falls die rechte Seite der Produktion aus mehr als einem Nichtteminal besteht — deren Nachbarknoten im Grundgraph festgestellt werden.

Die beteiligten Knoten dürfen nun aber nicht ohne weiteres zur linken Seite der Produktion reduziert werden, da zunächst sichergestellt werden muß, daß die Reduktion zulässig ist. Eine Reduktion gilt als zulässig, wenn die zu reduzierenden Knoten nicht zu anderen Reduktionsstellen gehören und Konflikte bzgl. der anwendbaren Produktion bei Mehrdeutigkeiten entschieden werden können. Als Entscheidungskriterien in Konfliktfällen werden Bindungsprioritäten[8], Abstandsregeln und die Größe der beteiligten Zeichen verwendet.

Erst wenn alle Konflikte ausgeräumt werden konnten, wird die Reduktion tatsächlich durchgeführt. Dazu werden die reduzierten Zeichen aus dem Grundgraph entfernt und durch ein einziges Nichtterminalzeichen ersetzt. Dieses hat die Größe und Lage des umfassenden Rechteckes der entfernten Zeichen. Die Nachbarschaftsverweise des neuen Zeichens ergeben sich aus den Verweisen der ersetzten Zeichen.

Bei der Konstruktion der zweidimensionalen Grammatik ist darauf zu achten, daß dieses Vererben der Verweise konfliktfrei ausgeführt werden kann. Es dürfen nicht mehrere Verweise für eine Richtung bei den reduzierten Zeichen

[8] D.h. wir ordnen die Prioritäten nicht Operatoren, sondern den Bindungen zwischen Zeichen zu. Es gilt beispielsweise Priorität(Zeichen, *Zeichen*) $>$ Priorität(Zeichen, $\cdot$) $>$ Priorität(Zeichen, $+$)

vorliegen, da dann ein Verweis ausgewählt und die restlichen Verweise vernachlässigt werden müßten. In diesem Fall wäre der Grundgraph nach einer Reduktion unter Umständen nicht mehr zusammenhängend, die Graphreduktion würde schließlich mit einer Fehlermeldung beendet oder falsch durchgeführt. Will man diese Einschränkung nicht tolerieren, so muß das Konzept geringfügig dahingehend erweitert werden, daß jedes Element beliebig viele Verweise für eine Richtung besitzen darf.

Der heuristische Ansatz[9] führt zu akzeptablen Analysezeiten. Die bisherigen Erfahrungen mit der Reduktionsstrategie haben gezeigt, daß die Reduktionsstelle schnell gefunden wird.

2.3.4 Erzeugung der Ausgabe

Während der Graphreduktion erzeugt die Syntaxanalysestufe eine interne Darstellung der eingegebenen Formel, die die syntaktische Struktur der Formel wiedergibt. Dieser *Syntaxgraph* wird in einem letzten Schritt durch einen Depth-First-Search-Algorithmus in ein eindimensionales Format übersetzt.

2.3.5 Scanner

Der Syntaxanalysestufe ist ein als endlicher Automat realisierter Scanner vorgeschaltet, der komplexe Zeichen aus ihren Bestandteilen zusammensetzt (:=, =, i, j). Diese komplexen Zeichen gelten als atomar. Die Bestandteile müssen zeitlich aufeinanderfolgend eingegeben werden. Es ist nicht möglich, einzelne Bestandteile komplexer Zeichen durch Radieren oder Überschreiben zu entfernen.

Schwierigkeiten entstanden beim Entwurf des Scanners durch die Aufgabe, Inkonsistenzen des Systemzustandes zu vermeiden, die durch die Notwendigkeit, Zeichen im Scanner zu puffern, einerseits und die Philosophie, den Benutzer jederzeit jede Operation ausführen zu lassen, andererseits entstehen können. Der Scanner wird deshalb von dem im folgenden Kapitel beschriebenen Modul zur Befehlsbehandlung besonders kontrolliert.

2.3.6 Erweiterbarkeit der Syntaxanalysestufe

Die Entwicklung der Syntaxanalysestufe wurde von Anfang an auf die Analyse mathematischer Formeln zugeschnitten. Zunächst wurde eine zweidimensionale Grammatik entworfen, mit deren Hilfe sich alle Formeln generieren lassen, die als mögliche Instanzen der Eingabe vorgesehen waren.

Jede Produktion der Grammatik wurde um Kontextbedingungen erweitert, in denen die Abstandsregeln und die Regeln über die Größenverhältnisse formuliert wurden. Zusätzlich wurden die Bindungsprioritäten festgelegt. Die Produktionen der Grammatik wurden schließlich von Hand in die entsprechenden Softwareroutinen des Graphreduzierers umgesetzt.

Die geschilderte Situation legt die Frage nach der Erweiterbarkeit der Grammatik (und damit des Funktionsumfanges der Syntaxanalyse) nahe.

[9] Gemäß unseres Klassifikationsschemas für "zweidimensionale Parser" handelt es sich auch bei der zweiten Stufe um einen nichtdeterministischen Parser, da die absolute Lage der Zeichen verwendet wird.

Wir haben es — ähnlich wie bei einem Prolog-Programm — mit einem Satz von Regeln und einem ziemlich universellen Auswahlverfahren zu tun, das in Abhängigkeit von der vorliegenden Eingabeinstanz feststellt, welche Regeln der Regelmenge anwendbar sind.

Die Erweiterbarkeit der Grammatik oder die Änderung der Kontextbedingungen ist also zunächst grundsätzlich durch Erweiterung des Satzes von Regeln möglich. In unserer Anwendung stellt sich die Frage der Erweiterbarkeit aber eher im Sinne der Automatisierbarkeit.

Der Benutzer sollte die Grammatik möglichst einfach auf einer für ihn überschaubaren Abstraktionsebene vornehmen können. Dazu ist der im Augenblick gewählte Ansatz nicht geeignet, da der Benutzer Programmierarbeit leisten müßte und diese Einblick in die Funktionsweise des Reduktionsalgorithmus voraussetzt.

Wir konstatieren also Bedarf an der Erweiterung des Konzeptes durch ein Generatorkonzept, mit dessen Hilfe Erweiterungen der zweidimensionalen Grammatik (oder auch Neuentwürfe) durchgeführt werden können. Zusätzlich könnte der Benutzer durch eine Graphikoberfläche zur Definition von Grammatikregeln und Kontextbedingungen unterstützt werden. [**Gab**] schlägt eine Umgebung zur graphischen Definition von syntaxgesteuerten Formeleditoren vor[10], die sich so erweitern ließe, daß unsere zweidimensionalen Grammatiken damit definiert werden könnten.

Mit Hilfe eines solchen Werkzeuges sollten auch 'uneingeweihte' Benutzer rasch in der Lage sein, das Konzept attributierter, kontextfreier Grammatiken soweit zu beherrschen, daß sie einfachere Spracherweiterungen vornehmen können. Problematisch bleibt sicher die Berücksichtigung von Prioritäten und die Vermeidung von Mehrdeutigkeiten der Grammatik bei weiterreichenden Erweiterungen der Sprache.

2.3.7 Augenblicklicher Leistungsumfang der Syntaxanalysestufe

Der überwiegende Teil gängiger mathematischer Formeln kann mit dem implementierten Prototypen bereits übersetzt werden: Integrale, Summen, Produkte, Indizes, Exponenten und Brüche dürfen in beliebig komplizierten Kombinationen verwendet werden.

Die augenscheinlichste Lücke im Leistungsumfang bilden Matrizen, die zur Zeit noch nicht verarbeitet werden können. Die Implementierung von Operationen, mit deren Hilfe Matrizen bearbeitet werden können, wurde aus mehreren Gründen nicht mehr in den Entwicklungsumfang des Prototypen aufgenommen:

- Das Algebrasystem (Reduce) unterstützt das Arbeiten mit Matrizen nicht weitreichend genug. Das Rechnen mit Matrizen ist zwar möglich, doch läßt das Algebrasystem keinen Zugriff auf Zeilen, Spalten oder Blöcke einer Matrix zu, was gerade dem handschriftlichen Arbeiten entgegenkäme. Die bloße Erweiterung der Arbeitsumgebung um die Möglichkeit, Matrizen einzugeben, erscheint deshalb unbefriedigend. Statt dessen sollte die

[10] Der von [**Gab**] vorgeschlagene Formalismus beinhaltet kein Parsingkonzept. Der syntaxgesteuerte Editor fragt die Bestandteile einer Formel der Reihe nach ab. Dieser Ansatz scheint zum handschriftlichen Arbeiten völlig ungeeignet.

Erweiterung der Syntaxanalysestufe Hand in Hand mit der Erweiterung des Algebrasystemes erfolgen, um eine homogene Unterstützung aller Arbeitstechniken zu erreichen, die das Arbeiten mit Matrizen elegant und effizient werden lassen: Direkte Produkte, flexible Indizierung, Operationen auf Blöcken, effiziente Eingabe von Matrizen spezieller Gestalt und rekursiv definierter Matrizen.

- Handgeschriebene Matrizen lassen sich unter Umständen nur dann eindeutig interpretieren, wenn der Kontext, in dem sie geschrieben wurden, bekannt ist. Nicht immer ist die Zeilen- und Spaltenstruktur eindeutig erkennbar.

  Die Implementierung der Analyse von Matrizen, die auf einem vernünftigen Kompromiß zwischen Benutzerkomfort und technischer Realisierbarkeit beruhen sollte, stellt also auch aus dieser Perspektive einen größeren, eigenständigen Entwicklungsschritt dar.

Bei der Entwicklung von Formeln werden Ähnlichkeiten zwischen Variablen häufig durch Verwendung des gleichen Symbols in Verbindung mit einem Akzent zum Ausdruck gebracht.

Das Arbeiten mit Akzenten wird bislang ebenfalls nicht unterstützt. Die notwendigen Erweiterungen von Mustererkennung und Syntaxanalyse werfen aber keine wesentlichen Schwierigkeiten auf.

2.4 Andere Parsingverfahren

In der Literatur wird im Zusammenhang mit der Analyse mathematischer Formeln auch heute noch das Verfahren von Anderson [**And**] zitiert, das wohl als das einschlägige Verfahren zur Analyse mathematischer Formeln angesehen werden kann.

Andersons Algorithmus strukturiert die Formel top-down mit Hilfe der in der Formel enthaltenen Sonderzeichen (Integralzeichen, Produktzeichen, Summenzeichen, Bruchstrich und Klammern).

Die anwendbare Produktion wird anhand nur eines Zeichens erkannt. Deshalb muß das signifikante Zeichen an einer ausgezeichneteten Position stehen. Andersons Algorithmus erfordert, daß das Sonderzeichen jeweils das am weitesten links stehende Symbol ist. Jedes Sonderzeichen teilt die Ebene in mehrere Streifen ein, die der Algorithmus rekursiv (ebenfalls von links nach rechts) nach Teilformeln untersucht. Andersons Algorithmus stellt also im wesentlichen einen *Recursive-Descent-Parser* [**AU**] dar, der die Formeln mit der wenig mächtigen LL(k)-Strategie strukturiert.

Dieser Ansatz setzt sich der Kritik aus, da die Einschränkung bzgl. der Position des maßgeblichen Zeichens beim Niederschreiben von Formeln häufig nicht erfüllt ist (siehe Bild 2.5).

Da Formeln nach anderen Kriterien niedergeschrieben werden, wird der Benutzer in der Regel von den Analysefehlern völlig überrascht. Die Ungenauigkeit des Eingabegerätes erschwert es zusätzlich, Andersons Schreibkonventionen einzuhalten.

Abbildung 2.5: Diese Formel könnte mit Andersons ll(k)-Verfahren nicht korrekt analysiert werden, weil das Integralzeichen nicht das am weitesten links stehende Zeichen ist.

Der Nachteil des Verfahrens ist bereits im Konzept begründet und läßt sich deshalb nicht reparieren. Als Fazit kann man also festhalten, daß Andersons Verfahren zwar einfach zu implementieren und damit auch zu erweitern ist, aber im Benutzerbetrieb unzulänglich ist.

Die Schwächen von Andersons Ansatz und die in Abschnitt 2.3.6 aufgezeigten Schwierigkeiten bei der automatischen Generierung von Parsern nach dem hier vorgestellten Graphreduktionskonzept legen die Idee nahe, das Parsing nach dem LR(k)-Schema auf zweidimensionale Strukturen zu übertragen.

Tatsächlich ist die Definition für zweidimensionale Grammatiken im Hinblick auf dieses Entwicklungsziel komplizierter angelegt worden, als es für den Graphreduzierer notwendig gewesen wäre.

Geplant ist zunächst die Definition eines zweidimensionalen Stacks mit den beiden Operationen *Push* und *Pop*. Anschließend soll das LR(k)-Schema Schritt für Schritt auf die neuen Gegebenheiten übertragen werden: Definition von Items, Definition des Abschlusses und Definition der Übergangsrelation.

Untersuchungen mit einer sehr eingeschränkten Version von zweidimensionalen Grammatiken, bei der die rechten Seiten der Produktionen nur eindimensionalen Charakter haben, aber in horizontaler oder vertikaler Richtung verlaufen können, führten zu ermutigenden Ergebnissen.

Diese eingeschränkten Grammatiken können zweidimensionale Strukturen erzeugen, die nur baumartig verzweigt sind. Weil die Produktionen der Grammatik eindimensional sind, reicht zur Syntaxanalyse ein eindimensionaler Stack.

Die Zustände eines Parsers für diese Form zweidimensionaler Grammatiken müssen aber wesentlich mehr Information speichern, als die Zustände eines Parsers für eindimensionale Grammatiken: Der Parser wählt eine von vier möglichen Leserichtungen aus und verwendet für jede Leserichtung einen eigenen Lookahead.

Die Versuche beschränkten sich deshalb bisher auf die Implementierung der Abschlußbildung und die Berechnung der Übergangsrelation, um an sehr einfachen Beispielen die Größe des erzeugten Automaten zu messen.

Ein Nachteil besteht darin, daß der Parsingvorgang in der Startproduktion beginnen muß. Diese muß also leicht auffindbar sein, d.h. an ausgezeichneter Position stehen (was aber auch schon eine erheblich geringfügigere Einschränkung darstellt als die von Anderson geforderte).

2.4.1 Zusammenspiel zwischen Syntaxanalyse und Mustererkennung

Bisher arbeiten Syntaxanalysestufe und Mustererkennung unabhängig voneinander. In dieser Konfiguration unterlaufen der Mustererkennungsstufe Fehler, die ein Menschen durch Berücksichtigung des Kontextes vermeiden kann.

Da die 'Welt' der mathematischen Formeln einfach strukturiert ist, sollte es möglich sein, den Kontext, in dem ein Zeichen geschrieben wurde, in die Mustererkennung einfließen zu lassen. Die Mustererkennungsstufe müßte dazu von der Syntaxanalysestufe über den aktuellen Kontext unterrichtet werden.

In der gegenwärtigen Form des Prototypen ist dieses Zusammenspiel nicht möglich, weil die Mustererkennung und die eigentliche Syntaxanalyse (Graph*reduktion*) zeitlich aufeinanderfolgen. Jedoch nimmt die Syntaxanalysestufe selbst Mustererkennungsaufgaben — wenn auch nur in bescheidenem Ausmaß — wahr: Unterscheidung von Π und Produktzeichen, Σ und Summenzeichen, Minus und Bruchstrich, Komma und Schrägstrich, i-Punkt und Operatorzeichen 'Punkt'.

Kapitel 3

Zum Arbeiten benötigte Infrastruktur

3.1 Anforderungen

Wie bereits in der Einleitung dargestellt wurde, soll das entwickelte System den Benutzer beim Rechnen mit mathematischen Formeln unterstützen. Um ihm die ganze Bandbreite der Rechnerunterstützung zu bieten, reicht es sicher nicht, nur einen einfachen Frage-Antwort-Rhythmus zuzulassen.

Unser Ziel ist, den Benutzer soweit wie möglich von mechanischer Arbeit zu entlasten, damit seine volle Aufmerksamkeit dem Entwickeln einer Strategie zugute kommen kann. Unter Strategie verstehen wir dabei, daß die Entwicklung einer Formel nicht nur durch Anwendung kontextfreier Regeln erfolgt, sondern auch durch übergeordnete Informationen wie die besondere Form der Formel oder den Kontext, in dem die Formel entstand, beeinflußt wird. Algebrasysteme arbeiten nicht in diesem Sinne strategisch[1].

Andererseits können Rechner gerade bei Arbeitsschritten, die bei Ausführung durch den Benutzer wahrscheinliche Fehlerquellen darstellen, ausgezeichnete Hilfe leisten. Die Tugend der Sorgfalt ist Rechnern angeboren, so daß der Benutzer eine Menge Zeit, Konzentration und Energie, die er auf Plausibilitätstests anwenden müßte, einsparen kann.

Soll der Benutzer in seinem gewohnten Arbeitsablauf sowenig wie möglich gestört werden, muß die Bedienungsoberfläche diesem gewohnten Arbeitsablauf angepaßt werden, d.h. die übliche Arbeitsumgebung möglichst naturgetreu imitieren. Diese Imitation schützt vor Fehlern beim Entwurf des Systemverhaltens. Laut [**NV**] gilt es, drei unterschiedliche Aspekte beim Entwurf interaktiver Programme zu berücksichtigen:

- Der *Physiologische Aspekt* zielt auf die Bewertung des Systems aus ergonomischer Sicht. Er ist besonders beim Entwurf des Eingabemediums und der Gestaltung der Ausgabe zu berücksichtigen.

[1] [**Dre**] argumentiert, daß regelbasierte, symbolisch arbeitende Systeme - wie Algebrasysteme sie darstellen - nie das auf Erfahrung beruhende, gewandte Vorgehen menschlicher Spezialisten nachahmen können

- In *psychologischer Hinsicht* gilt es, Bevormundung des Benutzers durch das System zu vermeiden. Die Reihenfolge von Verarbeitungsschritten wird — wo immer möglich — durch den Benutzer bestimmt. Er darf Programmteile mehrfach ausführen oder auf ihre Ausführung verzichten. Das System verschafft dem Benutzer eine übergeordnete Sicht über die Bearbeitung des Problemes, statt ihn von Eingabe zu Eingabe zu führen.

- Das *Systemverhalten* ist durch einen hohen Grad an Transparenz dem Benutzer gegenüber gekennzeichnet. Die Aufeinanderfolge und die Ergebnisse der einzelnen Verarbeitungsschritte sind für ihn nachvollziehbar.

Schon unsere Zielsetzung legt es nahe, die Benutzeroberfläche der Realität nachzuempfinden. Als *konzeptionelles Modell* [**FZ**] wählen wir die Arbeitsumgebung, die ein Benutzer von seiner Schreibtischoberfläche her kennt. In dieser Modellwelt findet er die *Objekte* (Formeln, Papier) wieder, die er bearbeiten möchte.

Die Objekte können durch *generische Funktionen* (Löschen, Verschieben) und *objektspezifische Funktionen* (Auswerten einer Formel, Lösen einer Gleichung) manipuliert werden. Die *Syntax der Bedienung* – wie werden welche Funktionen auf welche Objeke angewendet? – ist dem Benutzer aus seiner herkömmlichen Arbeitsumgebung bekannt. Finden sich die Objekte des konzeptionellen Modells auf der *physikalischen Ebene* der Realisierung im Programm wieder, so bezeichnet man das eben beschriebene Szenario als *direkte Manipulation* [**FZ**].

Systeme zur direkten Manipulation sollten den von [**NV**] an interaktive Programme gestellten Anforderungen auf natürliche Weise genügen. Besondere Aufmerksamkeit erfordert jedoch die Zusammenstellung des konzeptionellen Modells, der wir uns im folgenden zuwenden.

Neben der Verwendung eines möglichst 'natürlichen' Eingabegerätes, wie es in Abschnitt 0.4 beschrieben wurde, ist die Verwendung aller Arbeitsweisen, die der Benutzer auf einem üblichen Blatt Papier anwenden kann, wesentlich:

- *Handschriftliches Schreiben* in beliebiger Reihenfolge,

- *Korrektur* durch Überschreiben und nachträgliches Einfügen

- sowie *sukzessives Erweitern* von Formeln zum inkrementellen Arbeiten.

Diese Arbeiten sollten auf die naheliegende, dem Benutzer vertraute Weise – ohne Umwege wie die Bedienung von Menüs oder Tasten – erfolgen können.

Auf jedem Schreibtisch finden sich nun noch zusätzliche Werkzeuge, die die Arbeit weiter unterstützen. Auch diese Hilfsmittel sollte man auf der Bedienoberfläche des Systems wiederfinden: Einen Radiergummi zum Löschen von Zeichen, eine Schere zum Ausschneiden von Teilformeln, einen Stapel Papier zum Notieren von Zwischenergebnissen. Allerdings werden diese Werkzeuge nicht andauernd benötigt, so daß hier die Bedienung von Menüs o.ä. akzeptabel ist. Dies gilt auch für die letzte Gruppe von Hilfsmitteln.

Eine häufige Fehlerquelle beim Ausführen von Rechnungen liegt in Flüchtigkeitsfehlern beim Übertragen von Formeln. Auf den ersten Blick scheint diese

Fehlerquelle mit dem Einsatz von Formelmanipulationssystemen nicht mehr zu bestehen, da das Formelmanipulationssystem die Auswertung der Formel ja übernehmen soll.

Dies stellt sich aber als eine etwas verklärte Sicht der Realität heraus. In der Praxis zeigt es sich, daß Algebrasysteme dazu neigen, im Verlauf einer längeren Rechnung immer größere, unübersichtlichere Ausdrücke zu produzieren, so daß der Benutzer, der eine Vorstellung von dem gerade bearbeiteten Problem hat, notgedrungen von Zeit zu Zeit vereinfachend eingreifen muß. Als Beispiel für diesen Sachverhalt mag das Arbeiten mit Summenformeln dienen, die von Algebrasystemen nur expandiert werden können. Eingriffe geschehen durch geschicktes Umgruppieren von Formeln oder durch Umbenennen oder Ersetzen von Teilausdrücken. Werden diese Arbeitstechniken vom System zugelassen, ist der Benutzer ausreichend gerüstet, den Verlauf einer Berechnung zu steuern.

Es verbleiben noch die Hilfsmittel, die erst bei Benutzung eines Rechners nötig werden: Der Benutzer muß durch Fehlermeldungen auf Schwierigkeiten bei der Interpretation eingegebener Formeln aufmerksam gemacht werden. Diese Fehlermeldungen bieten ihm die Möglichkeit, durch leichte Korrekturen der Formel das System auf die richtige Spur zu bringen, indem z.B. unsauber gesetzte Symbole der Formel neu eingegeben werden. Die Arbeitsergebnisse einer Sitzung lassen sich auf Dateien sichern. Selbstverständlich ist dann auch das Einlesen bereits erstellter Dateien möglich.

Bei der Implementierung aller Dienste ist die Einfachheit der Bedienung vorrangig. Dem Benutzer vertraute Arbeitsvorgänge werden naheliegend auf die Systemoberfläche übertragen. Die Bedienung von Menüs darf nur bei Arbeitsschritten nötig werden, die selten vorkommen oder bei denen der übliche Arbeitsfluß sowieso unterbrochen werden muß. Systemmeldungen an den Benutzer erfolgen auf der Ebene der Eingabesprache.

3.2 Zusätzlich benötigte Komponenten

Die im letzten Abschnitt beschriebenen Merkmale der Benutzeroberfläche erfordern einige Komponenten, die sich um die in den vorangegangenen Kapiteln beschriebene zentrale Pipeline zum Vorverarbeiten, Erkennen, Strukturieren und Verarbeiten der Zeichen bzw. Formeln gruppieren. Obwohl jede dieser Komponenten nicht so kompliziert und damit reizvoll ist wie die Pipelinestufen, ist die sinnvolle Integration der Einzelkomponenten eine anspruchsvolle und sehr zeitaufwendige Aufgabe. Schließlich verleihen erst die zur Benutzeroberfläche gehörenden Komponenten dem aus der Sicht des Benutzers starren Gerippe der Pipeline Beweglichkeit und Attraktivität.

Die Pipeline selbst implementiert Schreiben, Überschreiben und Löschen von Zeichen. Wir brauchen hier also nur noch auf die Bearbeitung bereits fertiggestellter Formeln einzugehen. Diese werden zunächst in der Reihenfolge ihres Entstehens in einem Journal gespeichert. Mit Hilfe der Operationen "Blättern" und "Verschieben" lassen sich alle Formeln, die im Laufe einer Sitzung entstanden sind, auf dem Display einblenden. Die Operation "Verschieben" dient dabei dem schrittweisen Einblenden längerer Formeln, die nicht komplett auf den Bildschirm passen. Die Abmessungen der Formeln lassen sich mit einer Zoom-Funktion verändern.

Realisiert wurde das Journal als lineare Liste ohne weitere (Seiten-) Struktur. Jeder Eintrag der Liste enthält eine Formel und ihr umfassendes Rechteck. Die Formel selbst ist auf zwei verschiedene Arten abgelegt.

- Zunächst als Folge von Paaren, bestehend aus je einem Symbol und dessen umfassenden Rechteck. Diese Darstellung spiegelt die zweidimensionale Anordnung der Formel wieder und ist adäquat zur schnellen Ausgabe der Formel auf dem Display und zur erneuten Clusterung nach eventueller Modifikation.

- Zusätzlich werden die Formeln als vollständig geklammerte Ausdrücke abgelegt, da diese Darstellung notwendig ist zum Ersetzen von Teilausdrücken in bereits bestehenden Formeln.

Das Journal entspricht dem gewohnten Stapel Papier beim herkömmlichen Arbeiten. Auf eine Seitenstruktur wurde verzichtet, da der Umbruch der Formeln auf Seiten manchmal größere Lücken entstehen läßt und die Seitenstruktur keine besonderen Vorteile bietet.

Alle Zugriffe auf das Journal und die mit ihm in Zusammenhang stehenden Systemvariablen (Anzahl der Formeln, gerade bearbeitete Formel, durch Blättern oder Verschieben entstandene Offsets) werden von der *Befehlsverwaltung* abgewickelt. Die Einführung der Befehlsverwaltung bewirkt eine Abschirmung des Journals und des in den Zustandsvariablen codierten Systemzustandes gegenüber Zugriffen anderer Komponenten, was die konsistente Programmierung aller Kommandos sehr erleichtert. Modifikationen des Systemzustandes oder Auskünfte über ihn sind für andere Komponenten nur durch die Erteilung wohldefinierter Kommandos an die Befehlsverwaltung möglich.

Wie wir noch sehen werden, ist die Ausführung der Befehlsbehandlung als vollkommen eigenständiger Prozess vorteilhaft. Die Befehlsverwaltung führt Kommandos der folgenden Art aus:

- Löschen der Formeln in einem bestimmten Ordinatenbereich

- Anhängen einer Formel an das Journal

- Vorblättern, Zurückblättern, Verschieben aller Formeln durch Modifikation eines Offsets

- Einlagern oder Auslagern von Formeln auf Dateien

- Feststellen und Bereitstellen der Formel, in deren Bereich ein neu eingegebenes Symbol fällt

- Bereitstellen der in einem bestimmten Rechteck liegenden Symbole

- Bereitstellen der Symbole einer Formel, die außerhalb eines gegebenen Rechteckes liegen

Das vorletzte Kommando ermöglicht die Extraktion von Teilformeln aus bereits fertiggestellten Formeln ('Cut & Paste'). Der Benutzer kann dazu die Funktion *Ausschneiden* anwählen und ein Rechteck markieren, das die zu extrahierende Teilformel enthält. In der Befehlsverwaltung werden mit Hilfe des Journals alle in diesem Rechteck liegenden Symbole festgestellt und an eine vom

Benutzer anzugebende Position verschoben. Hat der Benutzer die Extraktion beendet, stellt die Befehlsverwaltung die Symbole dem Clusterungsmodul zur Verfügung, das sie – evtl. zusammen mit noch weiteren vom Benutzer direkt eingegebenen Symbolen – zu einer Formel strukturiert.

Das Ersetzen von Teilausdrücken wird ähnlich abgewickelt. Hier sind zunächst mehrere Schwierigkeitsstufen in der möglichen Aufgabenstellung zu unterscheiden:

- Die $1{:}n$-Ersetzung erlaubt lediglich das Ersetzen *eines* Zeichens durch mehrere andere. Die Implementation ist besonders einfach, da diese Ersetzung auf der Ebene der linearisierten, vollständig geklammerten Notation durch Stringersetzung durchgeführt werden kann. Hierzu werden zu ersetzender Ausdruck und Substitut zunächst strukturiert und in die linearisierte Form übersetzt. Die Stringersetzung selbst wird in der Regel bereits von dem Algebrasystem zur Verfügung gestellt.

- Es ist vorstellbar, diese Operation ohne großen Aufwand mächtiger zu gestalten, indem eine *eingeschränkte $n : m$ Ersetzung* zugelassen wird. Hierbei kann eine Symbolmenge, deren Übersetzung in linearisierter, vollständig geklammerter Form die Expansion eines Nichtterminals der für die die Eingabesprache des Computeralgebrasystemes zugrundegelegten kontextfreien Grammatik ist, durch eine aus dem gleichen Nichtterminal expandierbare Symbolmenge ersetzt werden. Da diese Erweiterung jedoch die Kenntnisse von Programminterna seitens des Benutzers voraussetzt, widerspricht sie der von uns verfolgten Strategie der direkten Manipulation.

- Beiden bisher angesprochenen Verfahren ist der Nachteil eigen, daß jeweils alle Vorkommen des zu ersetzenden Musters ersetzt werden. Soll die Möglichkeit der *selektiven $n : m$ Substitution* bestehen, muß die Ersetzung auf einer Darstellung erfolgen, die noch reichere Information über die zu behandelnde Formel enthält.

Um dem Benutzer die volle Mächtigkeit der zweidimensionalen Eingabe zukommen zu lassen, wählen wir die Möglichkeit der selektiven, uneingeschränkten $n : m$ Ersetzung innerhalb einer Formel. Der Benutzer wählt innerhalb einer Formel einen rechteckigen Bereich aus, den er ersetzen möchte. Die Befehlsverwaltung stellt die aktivierte Formel fest und extrahiert mit Hilfe des letzten Kommandos der oben angegebenen Liste die Symbole, die außerhalb des markierten Rechteckes liegen. Zu dieser Symbolmenge werden die Symbole des Substituts hinzugefügt und die so erhaltene Symbolmenge vom Clusterungsmodul strukturiert und übersetzt.

Die beiden oben erwähnten unterschiedlichen Darstellungen von Formeln verlangen die Möglichkeit der Konvertierung von jeder der beiden Darstellungen in die jeweils andere.

Die Konvertierung der Darstellung einer Formel als zweidimensional gesetztes Objekt — d.h. als Folge von Symbolen mit umfassenden Rechtecken — in die linearisierte Form ist nötig, um das symbolische Formelmanipulationssystem benutzen zu können. Sie wird durch den Formelparser geleistet. Die andere Richtung wird erstens nötig, um dem Benutzer die Interpretation der eingegebenen Formel durch den Formelparser ohne Rückgriff auf die Ausgabe vollständig geklammerter Terme mitteilen zu können. Zweitens müssen auch die Resultate des Algebrasystemes zur Ausgabe auf dem Display in der zweidimensionalen Form aufbereitet werden.

Das benötigte Modul wurde als Kombination von Scanner und LALR(1)-Parser ausgelegt. Es liest vollständig geklammerte lineare Ausdrücke und erzeugt daraus "schön"- gesetzte zweidimensionale Darstellungen. Aus Geschwindigkeitsgründen arbeitet der Parser in einem einzigen Pass, so daß die Berechnung ererbter Attribute nicht möglich ist.

Scanner und Parser wurden mit den UNIX-Werkzeugen FLEX und YACC generiert, wobei eine selbstkonstruierte attributierte Grammatik zugrundegelegt wurde, die die Eingabesprache des Computeralgebrasystems beschreibt. Zusammen mit zwei Prozeduren zum Entgegennehmen vollständig geklammerter Formeln und zum Verschicken von zweidimensional gesetzten Symbolmengen bilden Scanner und Parser den sogenannen *Pretty-Printer*.

Dieser verfügt aus Einfachheitsgründen noch über eine weitere Eingabeschnittstelle, die das Versenden bereits zweidimensional gesetzter Formeln gestattet. Durch diese Konstruktion gibt es nur eine einzige Schnittstelle zwischen Pipeline und Ein/Ausgabe, über die Formeln zur Ausgabe verschickt werden.

Damit bereits verarbeitete und zur Ausgabe verschickte Formeln nach eventueller Modifikation erneut verarbeitet werden können, muß die Ausgabe des Pretty- Printers zur erneuten Eingabe in den Formelparser geeignet sein. Diese Rückkoppelung ist der erste Schritt zum flüssigen Arbeiten mit Formeln. Sie können nun eingegeben werden, erscheinen als erste Antwort des Systems als unstrukturierte Menge von Klarschriftsymbolen und dann strukturiert als sauber gesetzte Formel auf dem Bildschirm.

Solche vom System erzeugten Formeln können modifiziert wieder als Eingabe in das System dienen. Dazu wird eine bereits verarbeitete Formel lediglich durch Einfügen, Überschreiben oder Löschen von Zeichen wieder "aktiviert". Aktivierte Formeln werden wie vom Benutzer direkt eingegebene Formeln Zeichen für Zeichen vom Formelparser eingelesen und strukturiert.

Der Benutzer erhält so mit wenig Aufwand eine neue Version einer bereits verarbeiteten Formel, die er auswerten lassen kann. Aktivierte Formel, modifizierte Version und Resultat der modifizierten Version stehen dem Benutzer anschließend zur Verfügung.

3.3 Das Gesamtsystem: Struktur, Kommunikation und Hardware

Die in den letzten beiden Abschnitten beschriebenen Module bilden zusammen mit den Modulen der Pipeline und der Ein/Ausgabeansteuerung das Gesamtsystem unseres Arbeitsplatzes zum Rechnen mit handgeschriebenen Formeln. Die Software des Prototypen läuft im wesentlichen auf zwei Rechnern.

Eine unter der Unixversion BSD4.2 arbeitende Workstation trägt die Hauptlast der Berechnung. Auf ihr werden alle Berechnungen außer der Ein/Ausgabeansteuerung erledigt. Die Ein/Ausgabeansteuerung wurde auf einem PC-AT plaziert, der mit der Workstation über eine RS232 Verbindung gekoppelt ist.

Der Grund für den Einsatz des zweiten Rechners liegt dabei in dem Problem, unsere LCD-Digitalisiertablett-Kombination elektronisch ansteuern zu können. Bislang gibt es keine Möglichkeit, das Display direkt an der Workstation zu betreiben. Der PC trägt aber keinen wesentlichen Anteil an der Rechenlast. Vielmehr bremst die langsame serielle Verbindung zwischen Workstation und PC die Verarbeitungsgeschwindigkeit, was beim Blättern und Verschieben durch Nachlaufen störend bemerkbar wird. Da sich zumindest die Integerleistung von Workstations bei gleichbleibenden Preisen während der bisher knapp zweijährigen Projektlaufzeit glatt verfünffacht hat, stehen in jedem Fall genügend Leistungsreserven für die Unterbringung zusätzlicher Lasten auf einer einzigen Workstation zur Verfügung.

Die Struktur der Software erläutern wir mit der Schilderung des Verarbeitungsablaufs einer Formel und dem Blockdiagramm in Abbildung 3.1. Wir riskieren in diesem Abschnitt die Wiederholung einiger bereits in vorigen Abschnitten erwähnter Sachverhalte, doch die fokussierte Darstellung vermittelt wohl am besten das Verständnis für die Funktion des Gesamtsystems.

Der Benutzer schreibt die Symbole der Formel mit Hilfe eines Stiftes auf der LCD-Digitalisiertablett-Kombination. Die vom Digitalisiertablett ertasteten Koordinaten werden ständig von der Ansteuerung erfragt. Ein endlicher Automat, der den wesentlichen Bestandteil der Ansteuerung darstellt, verfolgt die Aktivitäten des Benutzers und unterscheidet zwischen dem Aufruf von Kommandos durch Auswählen von Menüpunkten und dem Niederschreiben von Symbolen.

Gezeichnete Symbole werden als Linienzüge auf dem LCD-Panel ausgegeben, so daß der Benutzer seine Handschrift sofort zu sehen bekommt. Erkannte Befehle und zu Symbolen gehörende Koordinatensätze werden über die serielle Schnittstelle an die auf der Workstation liegenden Programmteile[2] weitergeleitet.

Der Versand verteilt die Befehle und Koordinatensätze an die zusätzlichen Module und nimmt auch deren Rückmeldungen entgegen, die er zur Ausgabe an den PC weiterleitet. Zu geschriebenen Symbolen gehörende Koordinatensätze nimmt die Segmentierungsstufe entgegen. Sie erzeugt kompakt als Liste der geschwärzten Pixel dargestellte Bilder der Symbole, die zusammen mit den Koordinaten des umfassenden Rechtecks des Symbols als Eingabe der Mustererkennung dienen. Nachdem die Symbole dort klassifiziert wurden, faßt der Clusterungsscanner Symbolfragmente eines Symbols, die nicht bereits durch die Segmentierung als zusammengehörend erkannt wurden, zusammen (i-Punkte mit i-Rumpf etc.).

Alle Symbole, die zur selben Formel gehören, sammelt der Clusterungsparser. Er strukturiert die Symbole zu einer Formel, indem er eine Übersetzung in vollständig geklammerter, eindimensionaler Form anfertigt. Der weitere Ablauf der Verarbeitung spaltet sich in zwei Teile.

- Die Befehlsbehandlung läßt im Pretty-Printer eine Rückübersetzung der eindimensionalen Ausgabe der Clusterungsstufe in eine zweidimensionale,

[2] Wir verzichten hier auf eine genaue Schilderung der Funktionsweise der einzelnen Programmteile und begründen auch die gewählten Darstellungsformate nicht, da diese Informationen bereits in den einschlägigen vorangegangenen Kapiteln zu finden sind.

schön gesetzte Darstellung anfertigen. Diese Darstellung der Formel wird im Journal abgelegt und auf dem Weg über den Versand zeichenweise auf der Ein/Ausgabeeinheit ausgegeben.

- Gleichzeitig wertet das Algebrasystem die eindimensionale Darstellung der Formel aus und liefert ein Ergebnis bei der Befehlsverwaltung ab. Auch dieses Ergebnis wandert zunächst in das Journal und gelangt dann auf dem bereits bekannten Weg auf die Anzeige.

Vom Benutzer abgesetzte Befehle werden vom Versand an die Befehlsbehandlung adressiert, wo die vom Benutzer gewünschten Maßnahmen getroffen werden. Einige Befehle (Ausschneiden und Ersetzen von Teilformeln, Aktivieren von Formeln) erfordern, daß die Befehlsbehandlung Kontakt mit dem Clusterungsmodul aufnimmt, um ihm die ausgeschnittenen bzw. aktivierten Symbole zur erneuten Clusterung zuzuführen. Umgekehrt erteilt der Clusterungsmodul der Befehlsbehandlung Befehle. Dies geschieht, wenn eine bereits verarbeitete Formel aktiviert werden soll, weil Zeichen einer neu begonnenen Formel in ihren Bereich geschrieben wurden.

Die Gesamtsoftware wurde in mehrere unabhängig voneinander laufende UNIX-Prozesse zerlegt, die in Abb. 3.1 skizziert wurden. Kommunikation zwischen diesen Prozessen erfolgt über Zweipunktverbindungen, die von UNIX mit Hilfe des Socketkonzepts realisiert werden. Sollen zwei Prozesse miteinander kommunizieren, errichten beide eine Kommunikationsendstelle, einen sog. Socket. Mit Hilfe eines festgelegten Protokolls wird dann eine Vollduplexverbindung zur sicheren Übertragung von Byteblöcken hergestellt. 'Sicher' bedeutet hierbei, daß keine Blöcke bei der Übertragung verschwinden oder vervielfältigt werden und daß die Blöcke in der Reihenfolge eintreffen, in der sie abgeschickt wurden. Zumindest kurzfristige Geschwindigkeitsdifferenzen zwischen Empfänger und Sender gleichen die Endstellen durch Puffer aus.

Der Grund für die Zerlegung der Software liegt in der beim Beginn des Projektes gehegten Sorge, die Leistung einer einzigen Maschine würde zur Echtzeitverarbeitung der Eingabe nicht ausreichen. Die über Sockets kommunizierenden Prozesse hätten dann auf mehrere mittels des TCP/IP Protokolls kommunizierende Maschinen plaziert werden können. Diese Option erwies sich zwar als unnötig, da der Prototyp auf einer einzigen Maschine in Echtzeit abläuft, jedoch konnten einige Vorteile aus dem Konzept der kommunizierenden (pseudo-) parallelen Prozesse gezogen werden:

- Die einzelnen Prozesse konnten völlig unabhängig voneinander entwickelt werden. Neuere Versionen oder Varianten eines Prozesses lassen sich ohne Recompilation der Gesamtsoftware problemlos in das System aufnehmen. Bei Implementierung einer geschickten Prozedur zum Andocken zweier Prozesse ist sogar das Austauschen zur Laufzeit möglich. Dies ist auch bei den während der Entwicklung unvermeidlichen 'Abstürzen' einzelner Module hilfreich.

- Die Schnittstellen zwischen den einzelnen Modulen sind völlig überschaubar. Es besteht keine Möglichkeit und somit auch keine Versuchung, nicht dokumentierte Schnittstellen bei der Implementierung auszunutzen.

- Durch Ausnutzen der Pseudoparallelität kann man den Benutzer in die Lage versetzen, mehrere Aktionen gleichzeitig vom Rechner erledigen zu lassen, ohne den Programmcode, der diese Aktionen implementiert, verflechten zu müssen. In unserem Falle kann der Benutzer einen Befehl absetzen, während die Mustererkennung noch eine Folge von sehr schnell geschriebenen Zeichen abarbeitet. Dies kostet den Programmierer aber nicht die Mühe, Mustererkennung und Befehlsbehandlung in einem Modul abwechselnd durchzuführen. Die gemäß ihren unterschiedlichen Aufgaben getrennten Module arbeiten auf der Maschine lediglich pseudoparallel.

Kapitel 4

Beispiele

Die Beispiele dieses Kapitels wurden primär mit dem Ziel ausgesucht, einen Eindruck von dem Systemverhalten des Prototypen zu vermitteln. Es geht nicht so sehr darum, die Leistungsfähigkeit des Algebrasystems bei der Berechnung anspruchsvoller mathematischer Probleme zu erproben. Das erste Beispiel zeigt die Berechnung eines Problems, das sich bei einem Versuch des ersten Kapitels stellte. In Abschnitt 1.3.2 muß die Maximum-Likelihood-Grenze als Menge der Schnittpunkte einer Schar von gedrehten Ellipsen mit einer Schar von Kreisen berechnet werden.

Abbildung 4.1: Die augenblickliche Version der Benutzeroberfläche

Mit dem zweiten Beispiel (ab Abb. 4.18) soll die Fähigkeit des Graphreduzierers demonstriert werden, auch kompliziertere Formeln, die nicht der LL(k)-Einschränkung genügen, zu analysieren.

Abbildung 4.2: Die Formel entsteht: Die Mustererkennung ersetzt die geschriebenen Symbole durch Normsymbole. Die Verzögerung, mit der die Zeichen ersetzt werden, beträgt nur die drei Segmente, die die Segmentierung als Lookahead benötigt.

Abbildung 4.3: Weil die Schreibfläche (aus technischen Gründen) sehr klein ist, können größere Formeln unter Zuhilfenahme des Scrolling eingegeben werden.

Abbildung 4.4: Durch Zoomen können Formeln vergrößert und verkleinert werden, so daß in diesem Fall die komplette Formel auf einen Bildschirm paßt.

Abbildung 4.5: Nach der Auswertung wird die Formel wohlproportioniert gesetzt. Der Benutzer kann das Ergebnis der Auswertung überprüfen.

Abbildung 4.6: Rechteckförmige Teilformeln können ausgeschnitten und dadurch in einen Puffer übernommen werden.

Abbildung 4.7: Ausgeschnittene Teilformeln können an beliebigen Stellen in das Journal eingefügt werden. Dabei können die Proportionen der Teilformel in (weit bemessenen) Grenzen verändert werden.

Abbildung 4.8: Die eingefügte Teilformel wird nach Ablegen des Werkzeugs sichtbar.

Abbildung 4.9: Durch Überschreiben und Einfügen von Symbolen wird die eingefügte Teilformel modifiziert.

Abbildung 4.10: Die Formel wird ausgewertet.

Abbildung 4.11: Große Ausdrücke lassen sich durch Makrodefinitionen übersichtlich strukturieren.

Abbildung 4.12:

Abbildung 4.13:

Abbildung 4.14:

Abbildung 4.15: Wir spezifizieren die Parameter, weil die allgemeine Lösung zu groß wäre.

Abbildung 4.16: Die Lösung der Gleichung soll von dem System berechnet werden.

Abbildung 4.17: Die beiden Lösungen erscheinen auf zwei aufeinanderfolgenden Seiten des Journals.

Abbildung 4.18: Eine Nicht-LL(k)-Formel wird von dem Graphreduzierer analysiert: Das signifikante Sonderzeichen (Bruchstrich, Integralzeichen)ist nicht in jedem Fall das am weitesten links stehende Zeichen.

Literaturverzeichnis

[AU] A. Aho, J. Ullman,
Principles of Compiler Design
Addison Wesley
1977

[And] R. Anderson,
An Online Symbolic Mathematics System Using
Twodimensional Handprinted Notation
Rand Corp., Santa Monica
1970

[Ben] J. Benary,
Attributed Grammars and Two-Dimensional Formal Languages
Technische Universität Dresden
1988

[BS] J.Bruck J. Sanz,
A Study on Neural Networks
IBM RJ 1986

[Bur] H. Burkhardt,
Methoden der digitalen Signalverarbeitung in
der Bildverarbeitung und Mustererkennung
Informatik Fachberichte Bd 125
Springer, 1986

[Dev] P. Devijver, J. Kittler,
Pattern Recognition Theory and Applications
NATO ASI Series F, Vol. 30
Springer, 1987

[Dre] H. Dreyfus, S. Dreyfus,
Künstliche Intelligenz
Rowohlt, 1987

[FZ] K. Fähnrich, J. Ziegler,
Direkte Manipulation als Interaktionsform an
Arbeitsplatzrechnern
in: Software Ergonomie '85
H. Bullinger [Hrsg.]
Teubner, 1985

[Fer] G. Ferraté (Ed.),
Syntactic and Structural Pattern Recognition
NATO ASI Series F, Vol. 45
Springer, 1988

[Fuk] K. Fukushima,
Neocognitron: A Hierarchical Neurol Network
Capable of Visual Pattern Recognition
Neural Networks, Vol. 1,pp 119-130, 1988

[Fun] K. Funahashi,
On the Approximate Realization of Continuous
Mappings by Neural Networks
Neural Networks Vol. 2, pp 183-193, 1989

[GT] R. Gonzalez, M. Thomason,
Syntactic Pattern Recognition
Addison Wesley
1978

[Gab] R. Gabriel,
Ein Formalismus zur Definition graphischer Formeln
Jahresbericht der GMD, 1987

[Gro] G. Groner,
Real-Time Recognition of Handprinted Text
Rand Corp., Santa Monica

[Har] I. Hartmann (Ed.),
Optical Recognition of Chinese Chararcters
Vieweg, 1989

[HN] R. Hecht-Nielson,
Neurocomputing: Picking the Human Brain
IEEE Spectrum, March 1988

[HF] F. Heigl, J. Feuerpfeil,
Stochastik, Leistungskurs
Bayerischer Schulbuchverlag, 1983

[Hop] J. Hopfield,
Neural Networks and Physical Systems with
Emergent Collective Computational Abilities
Proc. Nat. Acad. Sci. USA 79, 1982, 2554-2558

[Hua] T. Huang (Ed.),
Picture Processing and Digital Filtering
Springer, 1979

[HW] D. Hubel, T. Wiesel,
Receptive Fields, Binocular Interaction and
Functional Architecture in the Cats visual Cortex
Journal de Physiologie, 1962, 160(1), 106-154

[IM] B. Irie, T. Miyake,
Capabilities of Three-Layered Perceptrons
IEEE International Conference on Neural Networks
pp 641-648, 1988

[Kap] R. Kappes,
Ein Algorithmus zur syntaktischen
Analyse zweidimensionaler Mathematischer Formeln
Diplomarbeit, Universität des Saarlandes
voraussichtlich 1990

[Kem] C. Kemke,
Der neuere Konnektionismus. Ein Überblick
Informatik-Spektrum, Juni 1988
Springer

[Koh1] T. Kohonen,
An Introduction to Neural Computing
Neural Networks, Vol 1, pp 3-16, 1988

[Koel] J. Kölsch,
Schreibschrifterkennung durch elastischen
Mustervergleich
Diplomarbeit, UniversitÄt des Saarlandes
voraussichtlich 1990

[Koh2] T. Kohonen,
Self-Organization and Associative Memory
Springer, 1989

[Lee] K. Lee,
Neural Network Applications in Handwritten
Symbol Understanding
Siemens 1988

[Lin] Linus Technologies, Inc.
Handwritten Keyboardless-entry
Computer System,
Patentantrag EP 0 254 561
des Europäischen Patentamtes

[Mea] C. Mead,
Analog VLSI and Neural Systems
Addison Wesley, 1989

[NV] J. Nievergelt, A. Ventura,
Die Gestaltung interaktiver Programme
Teubner, 1983

[Pap1] A. Papoulis,
The Fourier Integral and Its Application,
McGraw-Hill, 1962

[Pap2] A. Papoulis,
Probability, Random Variables,
and Stochastic Processes
McGraw-Hill, 1965

[RGD] L. Personnaz, I. Guyon, G. Dreyfus,
Journal de Physique 46, pp 359-365, 1985

[Rit] S. Ritter,
Blockschrifterkennung mit Rumelhartnetzen
Universität des Saarlandes, vorraussichtlich 1990

[RMS] A. Rajavelu, M. Musavi, M. Shirvaikar,
A Neural Network Approach to Character Recognition
Neural Networks, Vol 2, pp 387-393, 1989

[RCW] D. Rumelhart, J. Mc. Clelland,
Parallel Distributed Processing
MIT Press, 1988

[RK] A. Rosenfeld, A. Kak,
Digital Picture Processing
Vol. 1, 2
Academic Press, 1982

[Ros3] A. Rosenfeld,
Picture Languages
Academic Press
1979

[Sch] J. Schürmann,
Polynomklassifikatoren für die
Zeichenerkennung
R. Oldenbourg Verlag, 1977

[Tap] C. Tappert,
Cursive Script Recognition
by Elastic Matching
IEEE Tutorial on Pattern Recognition, 1985

[Wei] F. Weigel,
Automatische Segmentierung
von Hand gezeichneter Diagramme,
Diplomarbeit, Universität des Saarlandes
voraussichtlich 1990