Kapitel 7

Formelerkennung

Neben der reinen Texterkennung eröffnet die Erkennung handgeschriebener Formeln eine Reihe weiterer interessanter Anwendungen und Fragestellungen. Führt man sich vor Augen, wie unzureichend bis heute die Eingabemöglichkeiten an einem Computer für mathematische Formeln realisiert sind, wird der Bedarf nach dieser Form von Mensch-Maschine-Schnittstellen schnell deutlich.
Für einen handschriftlichen Formeleditor sind im Desktop-Bereich Textverarbeitungssysteme das wichtigste Anwendungsszenario. Hier sind derzeit im wesentlichen zwei konkurrierende Ansätze gebräuchlich. Der erste ist die klassische Menü-gesteuerte Variante, wie sie bei WYSIWYG- (What You See Is What You Get) Textverarbeitungssystemen eingesetzt wird. Hierbei muß der Benutzer die einzelnen Symbole mittels Tastatur eingeben oder (z.B. bei Sonderzeichen oder Operatoren) über Auswahlmenüs selektieren. Der zweite Ansatz geht über die Verwendung von Satzsystemen wie z. B. LaTeX, bei dem die verschiedenen Symbole durch Befehle in einer Quelltextdatei gesetzt werden. LaTeX bietet somit die Möglichkeit auch ausgefallene Ausdrücke darzustellen, erinnert dafür aber stark an eine höhere Programmiersprache und zwingt den Benutzer sich eine Vielzahl verschiedener Befehle zu merken.
Ein weiteres Anwendungsszenario ergibt sich im Desktop-Bereich bei Mathematik-Tools und Gleichungslösern. Die Problematik der Mensch-Maschine-Schnittstelle ist hier ähnlich. In beiden Fällen ließe sich eine deutliche Verbesserung des Bedienkomforts durch eine handschriftliche Formeleingabe erzielen.
Ebenso ließen sich bei Mobilcomputern der nächsten Generation (Palm-Tops, PDAs und PIMs) in eleganter Weise über die HSE-Schnittstelle komplexe wissenschaftliche Taschenrechnerfunktionen steuern. Der zu berechnende Ausdruck könnte so direkt auf das Display geschrieben werden. Nach der Erkennung der handschriftlichen Eingabe und der Ergebnisberechnung kann das Ergebnis dann in gewohnter Form ausgegeben werden.
Um die spezielle Darstellung mathematischer Ausdrücke und die zweidimensionale Anordnung zu erkennen und zu interpretieren, wird das Gesamtproblem in zwei Teilprobleme zer-

legt. Diese Teilprobleme lassen sich wiederum in insgesamt sechs Prozesse aufteilen, die in dieser oder ähnlicher Form in den meisten Ansätzen wiedergefunden werden [Blo97, Blo95]:

1. Erkennung einzelner Symbole
   (a) Vorverarbeitung, Rauschfilter, Skew-Reduktion
   (b) Segmentierung
   (c) Symbolerkennung
2. Analyse und Interpretation der Symbolanordnung
   (a) Identifikation räumlicher Symbolbeziehungen
   (b) Identifikation logischer Symbolbeziehungen
   (c) Konstruktion der Bedeutung (z. B. in Syntaxform oder Baumstruktur)

Ein wesentliches Problem ist hier die Aufteilung der Segmentierung und der Symbolerkennung in zwei getrennte Prozesse. Speziell für die Handschrifterkennung gilt, dass eine korrekte Segmentierung häufig nur mit Kenntnis der Klassenzugehörigkeit möglich ist. Ebenso wie eine unvollständige oder falsche Segmentierung zwangsläufig zu einer falschen Symbolerkennung führt. Dieses Henne-Ei-Problem läßt sich effizient durch die Verwendung von HMMs umgehen, da mit der Viterbi-Dekodierung neben dem eigentlichen Erkennungsergebnis auch immer die Segmentierung bekannt ist.
Die folgenden Abschnitte stellen einen Ansatz dar, mit dem die Erkennung handschriftlicher Formeln realisiert werden kann. Nach einer generellen Übersicht im folgenden Abschnitt, werden in den weiteren Abschnitten die einzelnen Verarbeitungsschritte, insbesondere die Strukturanalyse und das Parsing beschrieben. Im Abschnitt 7.6 werden über die eigentliche Erkennung hinaus einige integrierte handschriftliche Korrekturfunktionen beschrieben, bevor in den letzten beiden Abschnitten die Ergebnisse dargestellt werden und die Zusammenfassung dieses Kapitel abschließt.

7.1 Grundlegende Funktionsweise

Das hier beschriebene System ist in einem schreiberabhängigen Modus in der Lage, zusammenhängende mathematische Ausdrücke bestehend aus einem Zeichenvorrat von ca. 100 verschiedenen Zeichen zu erkennen. Neben Ziffern, Klein- und Großbuchstaben können die wichtigsten mathematischen Symbole und Operatoren erkannt werden ($+ - \cdot : / \text{——} \hat{} \sqrt{} \sum \prod \int , ' = < > \leq \geq \rightarrow \leftrightarrow \approx ! \infty$), wie auch eine Reihe weiterer griechischer Buchstaben ($\alpha, \beta, \gamma, \lambda, \mu, \Delta, \pi, \omega, \epsilon, \tau, \phi$). Zur Modellierung der Symbol-Zwischenräume

wird - ähnlich wie bei einer Satzerkennung - ein *Space*-Modell eingesetzt. Auch bei der Formelerkennung findet die Initialisierung anhand von vorsegmentierten Trainingsbeispielen statt, wie sie beispielhaft in Abb. B.1 dargestellt sind. Das anschließende Training der Modelle, sowie die Evaluierung wird dann anhand ganzer zusammenhängender Ausdrücke, wie sie in Abschnitt B.3 beispielhaft dargestellt sind, durchgeführt.
Um zum einen eine möglichst konsistente Parameterschätzung des Space-Modells zu erzielen, zum anderen aber auch um den Aufwand für das zweidimensionale Parsing des Erkennungsergebnisses zu begrenzen, werden insgesamt vier Benutzungsregeln definiert, die bei der handschriftlichen Eingabe der Formeln beachtet werden sollen. Diese Benutzungsregeln - dargestellt in einer graphischen Übersicht in Abb. 7.1 - sind jedoch so formuliert, dass sie einer möglichst natürlichen Schreibweise entgegenkommen [Bec97, Kos98b] und die Benutzbarkeit des Systems damit nicht nennenswert einschränken:

1. *left-right, top-down*

   Ganz allgemein formuliert müssen die handschriftlichen Eingaben von links nach rechts und von oben nach unten erfolgen. Am Beispiel eines Bruchs bedeutet diese Regel, dass zunächst der Zähler zu schreiben ist, anschließend der Bruchstrich gezogen werden muß, bevor schließlich der Nenner geschrieben werden kann. Klammerausdrücke sind entsprechend dieser Regel in folgender Form einzugeben: *linke Klammer, Ausdruck, rechte Klammer*.

2. *Integrale, Summen, Produkte*

   Aus Beobachtungen ergab sich, dass es in einigen Fällen günstig ist, eine Ausnahme von Regel 1 zu erlauben. Dies ist beispielsweise der Fall bei Integralen, Summen oder Produkten. Tritt im Zusammenhang mit diesen Symbolen eine Untergrenze und/oder Obergrenze auf, so neigt der Benutzer dazu, nach dem Schreiben des Hauptsymbols zuerst die Untergrenze und erst dann die Obergrenze zu schreiben. Ganz ähnlich verhält es sich bei tiefgestellten Zeichen (Subskripte) in Kombination mit hochgestellten Zeichen (Superskripte), einfachen Anführungsstrichen oder Vektor- bzw. Matrixpfeilen. Auch hier wird es scheinbar als effizienter empfunden nach der Eingabe

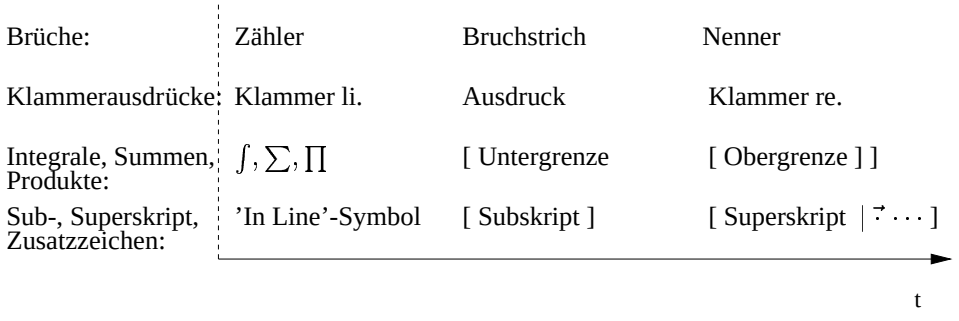


Abbildung 7.1: Benutzungsregeln zur zeitlichen Abfolge handschriftlicher Eingaben

des 'in-line'-Symbols das Subskript vor dem Superskript zu schreiben. Konkret bedeutet diese Ausnahmeregelung, dass z. B. ein Integral mit entsprechenden Grenzen in der Reihenfolge $< Integral,\ untere\,Grenze,\ obere\,Grenze >$ zu schreiben ist.

3. *Verlängerung von Brüchen und Wurzelzeichen*
   In einigen Fällen kann es sinnvoll sein, Zeichen wie Bruchstriche oder Wurzelzeichen zu verlängern, nachdem der Nenner bzw. der Radikand geschrieben wurde und dieser länger ausgefallen ist als ursprünglich vermutet. Aufgrund der Verarbeitung dynamischer Eingangsdaten ist eine Modifikation in der Form nicht möglich, da auf den unteren Systemebenen, d. h. während der Symbolerkennung, die räumlichen Beziehungen noch nicht berücksichtigt werden. Zeitliche Beziehungen sind jedoch bei dieser Form von Korrekturen im Allgemeinen nicht gegeben, sodass Symbol und Modifikation unter Umständen mit großem zeitlichen Abstand auftreten.

4. *Subskript, Superskript*
   Die Erkennung von geschachtelten Subskripten oder Exponenten wie z. B. x^{y^z} sind in diesem System derzeit nicht berücksichtigt.

Unter Beachtung dieser vier Benutzungsregeln ist es nun möglich, den zu erkennenden Ausdruck in kontinuierlicher Form einzugeben. Kontinuierlich bedeutet in diesem Zusammenhang, dass die einzelnen Symbole weder spatial noch temporal separiert geschrieben werden müssen. Vielmehr läßt sich der hier verfolgte Ansatz wie eine Satzerkennung ohne vorhergehende Wortsegmentierung betrachten, wobei die Identifikation von Wort- bzw. Symbolgrenzen einen integralen Bestandteil des HMM-Ansatzes unter Verwendung des 'space'-Modells darstellt. Ein weiterer Vorteil der Verwendung von HMMs ist die Möglichkeit, bei Bedarf gewisse syntaktische Randbedingungen in den Dekodierprozeß direkt zu integrieren. Abb. 7.2 gibt eine Übersicht über das grundlegende System zur Formelerkennung, wie es in [Kos98c, Kos99b] vorgestellt wurde. Die Funktionsweise der einzelnen Verarbeitungsstufen ist in den nachfolgenden Abschnitten dargestellt.

7.2 Vorverarbeitung, Merkmalsextraktion und Dekodierung

Wie bereits in Abschnitt 2.1 beschrieben wurde, erfolgt auch für die Formelerkennung eine räumliche Neuabtastung der Stifttrajektorie. Auch hier gilt es durch die räumlich äquidistante Abtastung die Geschwindigkeitsinformation zu eliminieren und die virtuellen Sequenzen zwischen den Strokes linear zu interpolieren. Dabei wird die Sequenz der *zeitlich äquidistant* abgetasteten Kartesischen Koordinaten $(x, y)(n \cdot \Delta T)$ in die *räumlich äquidistante* Abtast-

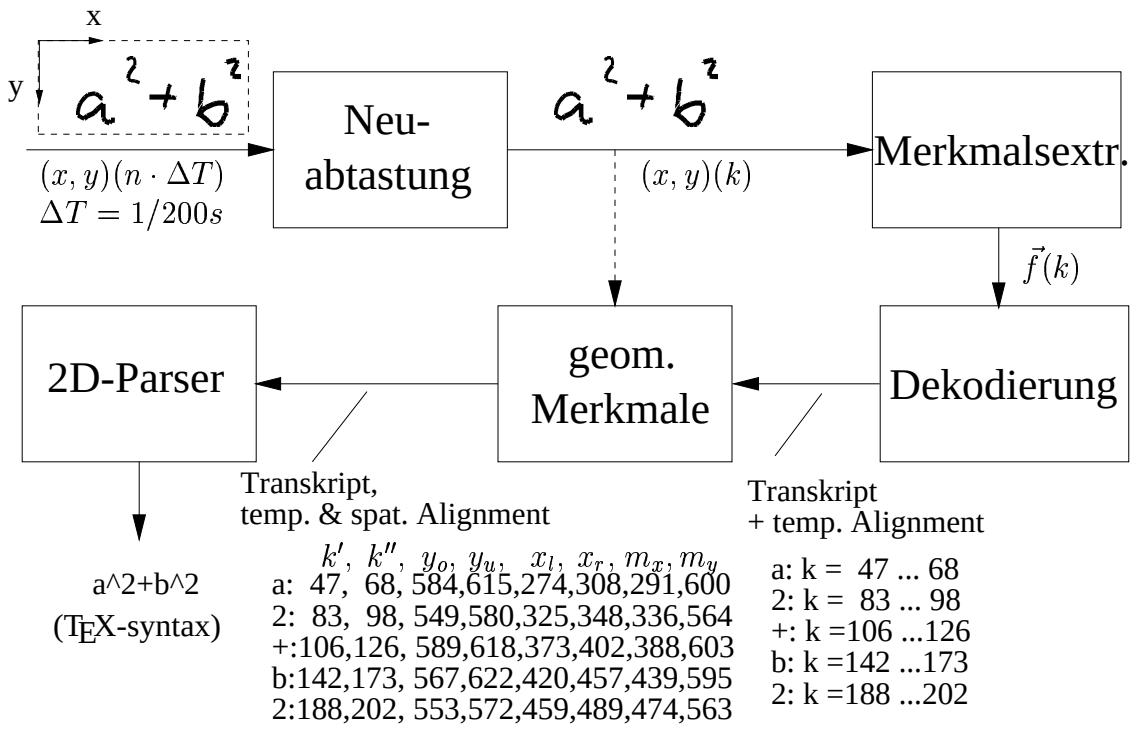


Abbildung 7.2: Systemübersicht des Formeleditors

squenz $(x, y)(k)$ überführt.

Wie auch bei dem Standardsystem zur Wort- bzw. Satzerkennung, werden bei der Formelerkennung Trajektorien- und Bitmap-Merkmale verarbeitet (vergl. Abschnitte 3.1.1 und 3.2.1), sowie zusätzlich die Information über den binären Stiftdruck.

Nach einer Vektorquantisierung, die i. d. R. mittels eines MMI-Netzes - ausgeführt in Multi-Codebuch-Technik - realisiert wird, werden die diskreten, multiplen Merkmalsströme mit Hidden Markov Modellen modelliert bzw. zu Erkennungszwecken dekodiert. Da die Länge der Beobachtungssequenz aufgrund der verschieden großen Symbole bei der Formelerkennung sehr stark schwankt, werden HMMs mit variierender Zustandsanzahl eingesetzt. Zur Modellierung von Großbuchstaben und größeren mathematischen Operatoren und Symbolen (Summe, Integral, Produkt) wurden HMMs mit 12 Zuständen eingesetzt. Kleinbuchstaben und Symbole mittlerer Größe (z. B. $+,\ (,\ ),\ \rightarrow$) wurden mit HMMs bestehend aus acht Zuständen modelliert. Sehr kleine Symbole hingegen, werden schließlich durch HMMs mit drei Zuständen repräsentiert. Für alle genannten Gruppen wurde eine lineare HMM-Topologie verwendet, d. h. es existieren von jedem Zustand nur Selbsttransitionen und Übergänge zum Nachfolgezustand.

Die initialisierten und anschließend trainierten Modelle λ werden schließlich bei der Erkennung verwendet, um eine Viterbi-Approximation in Kombination mit einer Strahlsuche, auf eine unbekannte Beobachtungssequenz Y anzuwenden. Diese Viterbi-Dekodierung liefert nun - im Sinne der Viterbi-Approximation - die wahrscheinlichste Zustandssequenz S^* einer

Menge von HMMs.

$$P(Y, S^*|\lambda) = \max_S P(Y, S|\lambda) \tag{7.1}$$

Die Aussage über die wahrscheinlichste Zustandssequenz S^*, welche die Dekodierung liefert, ist von entscheidender Bedeutung für die Realisierung des Gesamtansatzes. Zum einen ist durch diese Zustandssequenz festzulegen, um welche Symbole und deren Reihenfolge es sich im einzelnen bei der Beobachtungssequenz handelt. Zum anderen erfolgt für die Berechnung der Wahrscheinlichkeiten auch eine Zuordnung der einzelnen 'Frames' zu den jeweiligen Zuständen. Mit anderen Worten heißt das, dass zu *jedem* Frame der Beobachtungssequenz Y bekannt ist, zu welchem Abschnitt dieser Frame der Symbolfolge zuzuordnen ist. Im Rückschluß läßt sich somit die Folge der erkannten Symbole angeben mit jeweils einer exakten Indizierung k von Start- und Endframe.

7.3 Strukturanalyse

Wie bereits angemerkt wurde, liefert die Viterbi-Dekodierung neben dem eigentlichen Erkennungsergebnis auch die Information zu den einzelnen Segmentgrenzen innerhalb der Gesamtsequenz. Die Information zu den Segmentgrenzen (gegeben in zeitdiskreten Abtastschritten k) läßt sich zusammen mit den neuabgetasteten Vektordaten auswerten, um die räumliche Position der einzelnen Segmente zu bestimmen. Die Kenntnis über das aktuelle Symbol, sowie dessen räumliche Zuordnung wird anschließend benötigt um die erkannte Symbolsequenz zu parsen. Anhand des Beispiels in Abb. 7.3 soll das Vorgehen zur Bestimmung der geometrischen Merkmale, d. h., der räumlichen Position erklärt werden.
Dazu wird für das erkannte Symbol 'a' aus der gesamten Sequenz neuabgetasteter Vektoren zunächst der Bereich betrachtet, der ausgehend von den Segmentgrenzen das entsprechende Symbol enthält. In dem Beispiel lieferte die Viterbi-Dekodierung die Segmentgrenzen $k' = 35$ und $k'' = 59$. Beginnend bei $k = 35$, wird das Segment für jeden diskreten Zeitschritt k auf Minima und Maxima in x- und y- Richtung geprüft. Indem diese Extremwerte für jedes Segment gespeichert werden, erhält man am Segmentende eine exakte Information über das umschreibende Rechteck zu dem betrachteten Segment. Die Koordinaten und Abmessungen dieses umschreibenden Rechtecks werden für das anschließende Parsing als geometrische Merkmale weiterverarbeitet. Zuzüglich zu den Kartesischen Koordinaten der Kanten des umschreibenden Rechtecks wird der Mittelpunkt des Segments bestimmt, der als geometrischer Mittelpunkt des umschreibenden Rechtecks approximiert wird ($m_x = (x_l - x_r)/2$ und $m_y = (y_u - y_o)/2$). Mit dieser räumlichen Zuordnung der zeitdiskreten Abtastfolge sind nun Aussagen über Position und Größe der erkannten Symbole verfügbar, die während des Parsings ausgewertet werden.
Die Sequenz der erkannten Zeichen wird zusammen mit den einzelnen zeitlichen Segment-

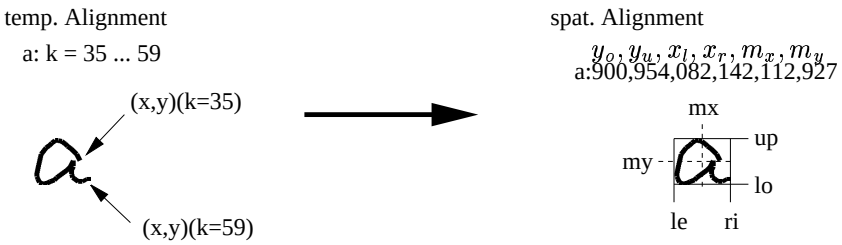


Abbildung 7.3: Bestimmung der räumlichen Segmentierung aus zeitlicher Segmentierung

grenzen und den nun vorhandenen geometrischen Merkmalen zur weiteren Verarbeitung in Form einer doppelt verketteten Liste zusammengefasst. In dem Beispiel in Abb. 7.2 entspricht ein Listenelement demnach einer Zeile der Ausgabe der geometrischen Merkmalsextraktion. Die verkettete Listenstruktur bietet sich deshalb an, da während des nachfolgenden Parsings oder auch um manuelle Korrekturbefehle umzusetzen Elemente an entsprechender Stelle aus der Liste entfernt bzw. hinzugefügt werden müssen. Des weiteren ist so ein einfacher Zugriff auf Vorgänger- oder Nachfolgezeichen möglich.

7.4 Parsing – LRTD-Ansatz

Basierend auf einer erkannten Symbolsequenz und deren geometrischen Merkmalen erfolgt die schrittweise Umsetzung des Erkennungsergebnisses in eine syntaxbeschreibende Form. Der in diesem Abschnitt beschriebene LRTD- (Left-Right-Top-Down-) Ansatz basiert auf den im Abschnitt 7.1 beschriebenen grundlegenden Benutzungsregeln. Als Ausgabeformat wurde hierbei LaTeXgewählt, da das LaTeX-Format zum einen äußerst vielfältige Möglichkeiten der Syntaxbeschreibung bietet und zum anderen die Ergebnisvisualisierung mittels DVI- oder Postscript-Konvertierung in einfacher und übersichtlicher Form ermöglicht. Darüber hinaus ließen sich die Erkennungsergebnisse in dieser Form leicht als Dokumentbausteine im Hinblick auf ein zu realisierendes Gesamtsystem weiterverwenden.
Im wesentlichen setzt sich der LRTD-Parser aus den folgenden sequentiell abzuarbeitenden Modulen zusammen:

- Fehlerkorrektur,
- Parsing spezieller mathematischer Operatoren und
- Detektion und Umsetzung von Sub- und Superskript.

Die Funktionsweise dieser Module wird in den jeweils folgenden Abschnitten beschrieben.

7.4.1 Fehlerkorrektur

Bei komplexen Mustererkennungsaufgaben, wie der Handschrifterkennung, ist eine perfekte, d. h. stets fehlerfreie Erkennung mit heutigen bekannten Ansätzen nicht realisierbar. Dies gilt insbesondere für solche Fälle, bei denen Ambiguitäten auf den unteren Systemebenen nicht aufgelöst werden können. Typisch für diese Fälle ist das in Abb. 7.4 gezeigte Beispiel. Die einzelnen Symbole aus dem oberen und unteren Beispiel ähneln sich dabei sehr stark. Zudem ist die Reihenfolge, in der die jeweilige handschriftliche Eingabe erfolgte, identisch. Dennoch tragen die gezeigten Ausdrücke eine völlig unterschiedliche Bedeutung. Verdeutlicht man sich an dieser Stelle einmal den Ablauf der gesamten Erkennung, wie er in Abb. 7.2 dargestellt ist, wird klar, dass die Dekodierung aufgrund fehlender Möglichkeiten zur Auswertung von Kontextinformation solche Grenzbereiche nicht unterscheiden kann. Mit Kontextinformation sind an dieser Stelle die geometrischen Merkmale der einzelnen Symbole gemeint, die erst nach der Dekodierung bestimmt werden.
Die Fehlerkorrektur des Parsers bezieht sich somit auch nur auf offensichtliche Fehler des Dekoders, wie z. B. die gezeigte Vertauschung eines kurzen Bruchstrichs mit einem Minuszeichen im unteren Beispiel von Abb. 7.4.
Eine weitere korrigierbare Fehlerklasse ist die Vertauschung von einfachen Anführungszeichen bei 'gestrichenen' Variablen mit Kommata ($x' \leftrightarrow x,$) oder auch die fälschliche Einfügung eines Punktes nach den Worten sin oder lim. Diese können durch sog. 'delayed strokes' entstehen, indem zunächst die Buchstabenfolge in verbundener Form geschrieben wird, und anschließend der i-Punkt über den Wortkörper gesetzt wird.
Die Korrektur falsch eingefügter i-Punkte läßt sich einfach realisieren, indem im Erkennungsergebnis auftretende Punkte nach bestimmten Schlüsselwörtern, wie den Worten $\sin$ oder $\lim$ pauschal gelöscht werden. Die Löschung kann vorgenommen werden, da eine solche Zeichenfolge keinen regulären Ausdruck formen kann.
Um die Vertauschung von Minuszeichen und kurzem Bruchstrich zu korrigieren (Bsp. in Abb. 7.4), wird jede Transkribierung zunächst auf vorhandene Brüche oder Subtraktionen überprüft. Tritt ein solches kritisches Zeichen auf, so werden die geometrischen Merkmale

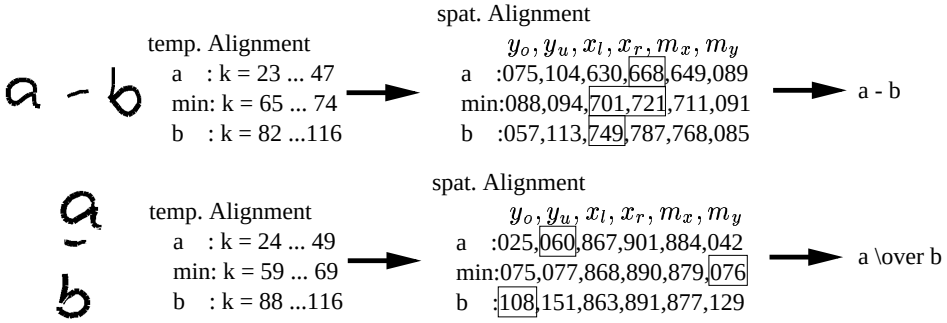
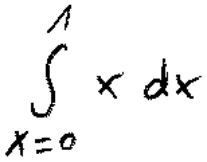


Abbildung 7.4: Beispiel zur Fehlerkorrektur

des Bruchstrichs bzw. des Minuszeichens mit den geometrischen Merkmalen des (zeitlichen) Vorgängersymbols und des (zeitlichen) Nachfolgesymbols verglichen. Der Vergleich bezieht sich dabei auf die in Abb. 7.4 markierten geometrischen Merkmale. Ein Minuszeichen liegt genau dann vor, wenn das Merkmal x_r des Vorgängersymbols (im Bsp. der Buchstabe 'a') kleiner ist als das Merkmal x_l des als 'min' erkannten Symbols. Zusätzlich muß überprüft werden, ob das Merkmal x_r des 'min'-Symbols kleiner als das Merkmal x_l des zeitlichen Nachfolgers ist. Da im oberen Beispiel beide Bedingungen erfüllt sind ($x_r('a') = 668 < x_l('min') = 701$ und $x_r('min') = 721 < x_l('b') = 749$), wird die Erkennung bestätigt. Es muß sich folglich um ein Minuszeichen handeln.
Im zweiten Beispiel in Abb. 7.4 zeigt dieser Vergleich, dass die Bedingungen für ein Minuszeichen nicht erfüllt sind. Vielmehr wird durch den Vergleich der geometrischen Merkmale von (zeitlichem) Vorgänger und Nachfolger mit den geometrischen Merkmalen des 'min'-Symbols klar, dass es sich um einen Bruchstrich handeln muß. Speziell sind bei diesem Vergleich die Merkmale $y_l('a')$, der Mittelpunkt in y-Richtung des 'min'-Symbols $m_y('min')$ und $y_l('b')$ relevant. Ein Bruchstrich liegt dann vor, wenn $y_u('a') < m_y('min)$ und $m_y('min') < y_o('b')$. Die markierten geometrischen Merkmale des unteren Beispiels in Abb. 7.4 bestätigen also, dass es sich um einen Bruchstrich handeln muß, was eine anschließende automatische Korrektur zur Folge hat.
Ein ganz ähnliches Prüfverfahren wird durchgeführt, wenn entweder Kommata oder Hochkommata in der Erkennung auftreten, um Vertauschungen eben dieser Symbole zu korrigieren. Auch hier sind beide Symbole ohne Kontextinformation kaum zu unterscheiden. Der Vergleich bezieht sich in diesem Fall jeweils auf die Merkmale m_y von Komma bzw. Hochkomma und dem Merkmal m_y des Vorgängersymbols. Bei einem Komma muß $m_y(',\ ') < m_y(Vorgänger)$ gelten, bei einem Hochkomma wären die Relationen genau umgekehrt.

7.4.2 Spezielle mathematische Operatoren

In einem weiteren Parsingschritt wird nach einigen speziellen Symbolen gesucht, deren zeitlicher Kontext eine besondere Bedeutung haben könnte. Dies sind insbesondere die Symbole $\int$, $\sum$, $\prod$ und $\lim$. Wird in der Transkribierung ein Summen-, Integral- oder Produktsymbol gefunden, so muß das Ergebnis auf mögliche Unter- oder Obergrenzen hin untersucht werden. Dabei können nun wieder die in Abschnitt 7.1 definierten Benutzungsregeln zur Hilfe genommen werden. Dort wurde mit Bedingung (2) festgelegt, dass eine Untergrenze - beispielsweise eines Integrals - zeitlich direkt nach dem Integral selbst geschrieben wird. Eine mögliche Obergrenze wird zeitlich nach der Untergrenze geschrieben.
Anhand des Beispiels in Abb. 7.5 läßt sich nun das Parsing dieser speziellen Operatoren erklären. Wie auch bei der oben beschriebenen Unterscheidung zwischen kurzen Brüchen



temp. Alignment

int	: k = 21 ... 56
x	: k = 108 ... 132
=	: k = 144 ... 264
0	: k = 172 ... 189
1	: k = 241 ... 260
x	: k = 330 ... 356
d	: k = 373 ... 409
x	: k = 417 ... 444

spat. Alignment

	$y_o, y_u, x_l, x_r, m_x, m_y$
int	:455,559,080,131,105,507
x	:583,618,041,068,054,600
=	:591,613,084,109,096,602
0	:587,609,121,142,131,598
1	:395,435,101,136,118,415
x	:498,532,178,209,193,515
d	:475,532,249,280,264,503
x	:495,534,295,334,315,514

Abbildung 7.5: Detektion von Unter- und Obergrenze am Beispiel eines Integrals

und Minuszeichen wird zunächst jeder Ausdruck auf das Vorhandensein dieser speziellen Operatoren (Integral, Summe etc.) hin untersucht. Tritt - wie im Beispiel gezeigt - ein solcher Operator auf, so wird zunächst die Position der Unterkante des umschreibenden Rechtecks $y_u('int')$ mit dem geometrischen Merkmal y_o des Nachfolgers verglichen (hier: $y_o('x')$). Aus dem spatialen Alignment wird sofort ersichtlich, dass sich das Nachfolgesymbol 'x' unterhalb des Integrals befindet ($y_u('int') = 559 < y_o('x') = 583$). Da eine Untergrenze nicht zwangsläufig aus einem einzigen Symbol bestehen muß, sind zusätzlich noch die weiteren Nachfolgezeichen zu untersuchen. Diese Nachfolgezeichen werden nacheinander solange zu einer Untergrenze zusammengefasst, wie deren Oberkante des umschreibenden Rechtecks unterhalb des Integrals liegt. In dem gezeigten Beispiel sind dies neben dem Zeichen 'x' noch das Gleichheitszeichen und die Null ($y_u('int') = 559 < y_o('=') = 591$ sowie $y_u('int') = 559 < y_o('0') = 587$). Bei dem fünften Zeichen der Transkribierung (hier: die '1') ist diese Bedingung jedoch nicht mehr erfüllt, sodass die Gruppierung zur Untergrenze abgebrochen werden muß. Ungeklärt ist an dieser Stelle allerdings noch, ob für das Integral eine Obergrenze angegeben wurde. Dazu muß zunächst für das nächste Symbol nach der Untergrenze die relative Position zum Integral betrachtet werden. Insbesondere werden dazu die Merkmale $y_o('int')$ und $y_u('1')$ verglichen, wobei sich zeigt, dass $y_u('1') = 435 < y_o('int') = 455$ und damit eine Obergrenze vorliegt. Bei einem positiven Test für eine Obergrenze wird zwecks Gruppierung nachfolgender Symbole in analoger Weise zur Gruppierung der Untergrenze verfahren. Nach Abbruch der Gruppierung zur Obergrenze werden die nachfolgenden Symbole zum Integranden zusammengefasst.

Wird das Wort 'lim' erkannt, so ist zu erwarten, dass nachfolgend die Variable genannt wird, auf die sich die Grenzwertbildung bezieht, gefolgt von einem Pfeil und dem Grenzwert selbst. Die Gruppierung erfolgt auch hier durch Vergleich der Unterkante des Wortes 'lim' mit den Oberkanten der Nachfolgesymbole.

Nach diesem Prinzip erfolgt auch die Gruppierung zu Teilausdrücken bei auftretenden Wurzeln oder Brüchen (Abb. 7.6). Bei diesen Strukturen ist es allerdings durchaus wahrscheinlich, dass ein Teilausdruck, d. h. der Radikand, der Zähler oder der Nenner eines Bruches über das Wurzelzeichen bzw. den Bruchstrich hinausragt. Dies kann insbesondere dann be-

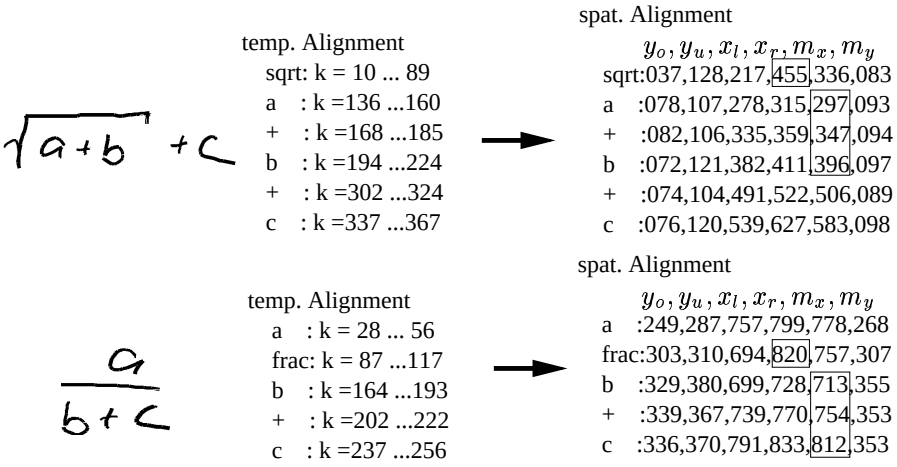


Abbildung 7.6: Gruppierung von Teilausdrücken

obachtet werden, wenn das letzte Zeichen eines Teilausdruckes noch mit Zusatzzeichen oder Indizes versehen wird. Zur Gruppierung von Zeichen zu Teilasudrücken wird daher nicht das umschreibende Rechteck als Entscheidungskriterium herangezogen, sondern der geometrische Mittelpunkt der Zeichen. Wie in Abb. 7.6 am Beispiel einer Wurzel gezeigt wird, werden die nachfolgenden Symbole zu einem Radikanden zusammengefasst, deren Mittelpunkt m_x unterhalb des Wurzelzeichens liegt ($m_x(Nachfolger) < x_r('sqrt')$). Im oberen Beispiel in Abb. 7.6 sind dies die Zeichen 'a', '+' und 'b'. Die Suchrichtung für den Radikanden ist hier wiederum aus der Bedingung (1) in Abschnitt 7.1 bekannt.
Das zweite Beispiel in Abb. 7.6 zeigt eine Situation, in der die Gruppierung, basierend auf den Merkmalen des umschreibenden Rechtecks, fehlschlagen würde, da das 'c' im Nenner etwas über den Bruchstrich hinausragt ($x_r('c') > x_r('-')$. Da jedoch auch bei der Gruppierung von Brüchen von den Mittelpunkten der Zeichen ausgegangen wird, kann der Nenner korrekt zusammengefasst werden ($m_x('c') < x_r('-')$). Die Suchrichtungen für Zähler und Nenner sind ebenfalls aus Bedingung (1) in Abschnitt 7.1 bekannt: einem Bruchstrich muß der Zähler vorausgehen und der Nenner folgen.

7.4.3 Sub- und Superskript

Die zuvor beschriebenen Methoden zur Strukturierung des Erkennungsergebnisses basierten stets auf der Existenz gewisser Schlüsselwörter, wie z. B. 'int' bei Integralen oder 'sqrt' bei Wurzeln. Auf solche Hinweise kann bei der Suche nach Sub- oder Superskripten nicht zurückgegriffen werden. Die Situation stellt sich hier vielmehr so dar, dass bis auf Ausnahmen jedes erkannte Zeichen zu einem Index oder Exponenten zählen könnte. Als ein wesentlicher Anhaltspunkt kann hier lediglich die in Abschnitt 7.1 definierte Bedingung (1) verwendet werden, wonach ein Index oder Exponent in zeitlicher Reihenfolge unmittelbar nach dem 'in-line' Zeichen einzugeben ist.

Um den Suchaufwand nach Indizes oder Exponenten einzuschränken und auch das Potential für mögliche Fehlinterpretationen zu reduzieren, ist es zunächst sinnvoll, mögliche Zeichentypen für Indizes und Exponenten zu spezifizieren. Für Indizes lassen sich die gültigen Ausdrücke so z. B. auf Zahlen, Buchstaben und deren Kombination beschränken. Für das Parsing kann weiterhin ausgenutzt werden, dass die Basis zu einem Exponenten oder das 'in-line' Zeichen eines Index - bis auf Ausnahmen - ebenfalls Variablen oder Zahlenwerte sind. Als Ausnahmen sind hierbei noch die Symbole $), \ ], \ \}, \ \sqrt{}, \ !, \ \infty$ zu berücksichtigen, welche darüber hinaus als 'in-line' Symbol auftreten können. Für den Exponenten gelten prinzipiell jedoch keine Restriktionen, sodass auch komplexe Strukturen, wie z. B. Brüche als Exponenten möglich sind. Die einzige Einschränkung hier ist, dass keine Exponenten höherer Ordnung (d. h. Exponent vom Exponent) zugelassen sind. Diese Einschränkung ist notwendig, da die verschiedenen Ebenen bei handschriftlicher Eingabe kaum mehr zu unterscheiden sind. Ein durchschnittlicher Exponent ist zudem nur zwei bis drei Millimeter groß. Bei einer Auflösung von 300 dpi bedeutet das schließlich, dass solche kleinen Zeichen nach der Neuabtastung lediglich durch 10-20 Abtastpunkte repräsentiert werden. Dies erschwert nicht nur die eigentliche Erkennung der Zeichen und die Unterscheidung zwischen Groß- und Kleinschreibung bei Buchstaben, sondern auch die zuverlässige relative Positionsbestimmung der einzelnen Zeichen und damit die eindeutige Zuordnung zu den verschiedenen Ebenen des Exponenten.

Aufgrund der großen Anzahl potentieller Kandidaten für Indizes oder Exponenten muß die Suche nach Sub- bzw. Superskript demnach für eine relativ große Symbolmenge durchgeführt werden. Die Strukturierung stützt sich dabei lediglich auf die relativen Positionen der einzelnen Symbole. Zu diesem Zweck wird jedes Zeichen, beginnend mit dem ersten Zeichen in der Liste erkannter Symbole, untersucht. Erfüllt ein erkanntes Zeichen in der Liste das logische Kriterium, d. h. es ist vom zulässigen Typ, so werden durch Positionsvergleiche von Symbolmittelpunkt und umschreibendem Rechteck zwischen dem aktuell untersuchten Zeichen und deren Nachfolger entsprechende Gruppierungen zu Indizes oder Exponenten vorgenommen, wenn zudem die geometrischen Kriterien erfüllt werden. Für Indizes (Exponenten) gelten $m_y(Basis) < y_o(Nachfolger)$ ($m_y(Basis) > y_u(Nachfolger)$). Zudem muß der Nachfolger in beiden Fällen rechts von der Basis liegen ($m_x(Basis) < x_l(Nachfolger)$).

Da der geometrische Mittelpunkt eines Zeichens mit dem Mittelpunkt des umschreibenden Rechtecks approximiert wird, liegen die Mittelpunkte von Buchstaben mit Unterlängen (f, g, j, p, q, y) tendenziell etwas tiefer als die übrigen Buchstaben oder Zahlen. Um zu vermeiden, dass nun nachfolgende in-line Zeichen fälschlicherweise als Exponent erkannt werden, wird zu Beginn des Parsings die Position m_y eines jeden Zeichens mit Unterlänge um einen gewissen Bias in y-Richtung nach oben korrigiert.

Matrix- bzw. Vektorpfeile werden von dem Parser als eine spezielle Art von Exponenten

behandelt. Die Strukturierung des Erkennungsergebnisses kann bei dieser Form von Symbolen unter Berücksichtigung geometrischer Merkmale allerdings wieder auf die Existenz von Schlüsselworten zurückgreifen.
Nachdem schließlich alle Teilausdrücke gruppiert wurden, erfolgt die abschließende Umsetzung in das LaTeXFormat [Kop94]. Dazu ist es lediglich notwendig die gefundenen Teilausdrücke jeweils mit logischen Klammern zu versehen und diese zusammen mit den übrigen erkannten Zeichen und Operatoren entsprechend der LaTeX-Syntax in eine Ausgabedatei zu schreiben. Funktionen werden im Gegensatz zu Variablen mit einem '\' gekennzeichnet. Die in Abb. 7.6 gezeigten Beispiele resultieren somit nach abgeschlossener Erkennung in zwei reinen (ACSII-) Textdateien mit einer sehr kompakten Beschreibung der Syntax: `\sqrt{a+b}+c` bzw. `a \over{b+c}`.

7.5 2D-Parsing mit kontextuellen Graph-Grammatiken

Neben dem in den vorangegangenen Abschnitten beschriebenen LRTD-Ansatz, wurde in Zusammenarbeit mit dem *Insitut National de Recherche en Informatique et en Automatique* (INRIA) ein weiterer Ansatz realisiert [Kos99a], der auf Benutzungsregeln, wie sie zunächst für den LRTD-Ansatz formuliert wurden weitgehend verzichtet. Dieser Ansatz basiert im wesentlichen auf der Kombination zweier existierender Systeme für die Formelerkennung, bei deren unabhängiger Entwicklung jedoch zunächst unterschiedliche Ziele verfolgt wurden.
Bei dem einen System handelt es sich um den bereits beschriebenen handschriftlichen Formeleditor. Das zweite System ist das am INRIA entwickelte *OFR* (Optical Formula Recognition), welches für die Offline-Erkennung von Formeln in gedruckten Dokumenten entwickelt wurde. Dieses System besteht aus den drei weitgehend unabhängigen Hauptkomponenten OCR, Graph-Builder und Graph-Parser [Lav97, Lav98].
Das OCR-Modul liefert dabei die Eingabe für den Graph-Builder in Form von erkannten Einzelzeichen, und deren Position und Größe. Da es sich um eine Offline-Erkennung handelt, existiert eine *zeitliche* Zeichenfolge im Sinne der erfolgten Zeicheneingabe nicht. Der auf der OCR-Ausgabe aufsetzende Graph-Builder und der nachfolgende Graph-Parser werten lediglich Position und Größe der erkannten Zeichen für die Strukturierung aus. An dieser Stelle wird bereits klar, dass sich die Vorteile beider Systeme elegant miteinander verknüpfen lassen, wodurch sich eine handschriftliche Formelerkennung ohne vorgegebene Schreibrichtung realisieren läßt. Als Schnittstelle zwischen beiden Systemen dient hier lediglich die Transkribierung der handschriftlichen Eingabe und deren geometrische Merkmale.
Aus der Transkribierung wird zunächst zusammen mit den geometrischen Merkmalen

ein Graph konstruiert, der die geometrischen Zusammenhänge zwischen den einzelnen initialen lexikalischen Einheiten repräsentiert. Die gerichteten Kanten, welche die Knoten miteinander verbinden, tragen somit die Information über die relativen Positionen der Knoten. Der Graph wird in der Form konstruiert, dass zunächst versucht wird, jedes Symbol der Formel in acht Richtungen (rechts, links, oben, unten, oben rechts, oben links, unten links, unten rechts) mit seinen nächsten Nachbarn zu verbinden, wobei - in Abhängigkeit des untersuchten Zeichens - bestimmte Gültigkeitstests bezüglich der Richtung potentieller ein- und ausgehender Kanten durchgeführt werden. Darüber hinaus wird bei diesem Test ein elliptisches Potenzialmodell verwendet, welches der bevorzugten Schreibrichtung (von links nach rechts) entspricht. Damit läßt sich modellieren, dass in horizontaler Richtung ein rechts liegendes Zeichen beispielsweise über weitere Abstände mit einer ausgehenden Kante verbunden werden kann, als ein links liegendes Zeichen (vergl. [Lav98]).
Nachdem auf diese Weise ein initialer Graph der Formel konstruiert wurde, kann damit begonnen werden diese zweidimensionale Beschreibung der Formel zu parsen. Dabei wird eine kontextabhängige Graph-Grammatik [Pfa69] verwendet, die bestimmte Syntaxregeln in Abhängigkeit der verwendeten Symbole auswertet. In iterativer Weise werden aus dem eingangs generierten Graphen Teilgraphen zu einem Knoten zusammengefasst. Der resultierende Knoten enthält eine abstrahierte Prefix-Syntaxbeschreibung des Teilausdrucks. Die geometrischen Merkmale dieses Knotens werden nach der Zusammenfassung auf die Parameter des umschreibenden Rechtecks des gesamten Teilausdrucks gesetzt. Bei diesem Ansatz, der in der Literatur auch als *Graph-rewriting* [Blo96] bekannt ist, wird diese Zusammenfassung von Teilausdrücken iterativ wiederholt, bis schließlich ein verbleibender Knoten die Syntaxbeschreibung der gesamten Formel enthält.

7.6 Manuelle Korrekturfunktionen

Die bisher vorgestellten Ansätze zur Realisierung eines handschriftlichen Formeleditors basierten stets auf dem Prinzip eine ganze Formel vollständig einzugeben und anschließend die Erkennung zu starten. Nachträgliche Modifikationen oder auch partielle Korrekturen von möglichen Erkennungsfehlern waren dabei ausgeschlossen. Das bedeutet, dass ein Benutzer ohne die Unterstützung manueller Korrekturfunktionen gezwungen ist, selbst bei kleinsten Erkennungsfehlern eine u. U. sehr komplexe Formel komplett neu einzugeben. Der erhöhte Gebrauchswert bei Einführung manueller Korrekturfunktionen zeigt sich jedoch nicht nur bei auftretenden Erkennungsfehlern. Gerade bei einem handschriftlichen Formeleditor kann es zudem ausgesprochen nützlich sein, nach der Erkennung der Eingabe noch Erweiterungen oder Veränderungen vorzunehmen, um die gewünschte Formel auf dem ’elektronischen Pa-

pier' schrittweise zu entwickeln. Zu diesem Zweck wurden die Editierfunktionen 'Löschen', 'Ersetzen', 'Einfügen', 'Rückgängig und Wiederholen' und 'Neuzeichnen' in den Demonstrator integriert.
Im Prinzip ist die Umsetzung von Editierfunktionen bei einem handschriftlichen Formeleditor auf zwei Arten denkbar:

- Zum einen könnten Modifikationen auf dem Erkennungsergebnis selbst erfolgen. Bei der hier gewählten Ausgabeform wäre das sehr leicht umzusetzen, da es sich bei der Ausgabe um reine Textdateien handelt.
- Zum anderen ist es aber auch denkbar die Änderungen auf dem temporär gespeicherten Schriftbild vorzunehmen, anschließend die handschriftlichen Editierbefehle und Korrekturzeichen erkennen zu lassen, die Befehle auszuführen und die Änderungen dann schließlich mit der zuvor erzeugten Ausgabe zusammenzuführen.

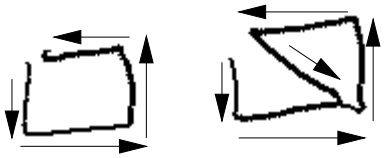
Während erste Arbeiten zur Integration von Erkennung und automatischer Ausführung handschriftlicher Korrekturen in OCR-Systemen bekannt sind [Mor97] blieb der zweite o. g. Ansatz bis auf die an der Gerhard-Mercator-Universität - Duisburg durchgeführten Untersuchungen [Kos00, Mes99] völlig unberücksichtigt. Doch gerade dieser Ansatz bietet eine Reihe von interessanten Fragestellungen und Vorteilen bezüglich der Benutzbarkeit. Änderungen bzw. Korrekturen können so vom Benutzer durchgeführt werden ohne einen lästigen Wechsel der Eingabemodalitäten vorzunehmen (vom Stift zur Tastatur oder Maus). Bei Endgeräten, die konsequent auf die Stifteingabe setzen, ließen sich mit einem solchen Verfahren in konsistenter Form z. B. Taschenrechnerfunktionen integrieren.

7.6.1 Funktionsweise manueller Korrekturfunktionen

Für die Erkennung und Ausführung manueller Korrekturfunktionen ist es zunächst notwendig Steuersymbole zu definieren. Diese Symbole werden wie die übrigen Zeichen, Buchstaben und Ziffern anhand von Trainingsbeispielen gelernt und durch ein oder mehrere HMMs repräsentiert. Wird anschließend ein solches Steuersymbol erkannt, können daraufhin entsprechende Editieroperationen auf dem Original vorgenommen werden.
Ein solches Steuersymbol sollte daher gewisse Eigenschaften aufweisen. Zunächst ist es wichtig, mit einem Steuersymbol die relevanten Bereiche, auf die sich die Änderungen beziehen, exakt markieren zu können. Eine weitere wichtige Eigenschaft des Steuersymbols sollte die gute Unterscheidbarkeit zu sämtlichen übrigen Zeichen sein, da Vertauschungen zwischen regulären Zeichen und Steuersymbolen zwangsweise einen starken unerwünschten Einfluß auf das Gesamtergebnis zur Folge haben. Erste Tests mit unterschiedlichen Editiersymbolen [Kos00] zeigten, dass die in Abb. 7.7 dargestellten Zeichen die erforderlichen

Eigenschaften aufweisen. Da bei der Online-Handschrifterkennung auch die zeitliche Abfolge der ’Strokes’ relevant ist, kann dies als zusätzliches Unterscheidungsmerkmal einbezogen werden, indem die in Abb. 7.7 mit den Pfeilen angedeutete Schreibrichtung eingehalten wird. Mit entsprechenden Variationen dieser Symbole hinsichtlich Ausdehnung und Kantenverhältnis in der Trainingsdatenbasis sind natürlich auch verschiedene Realisierungsformen bezüglich der Form und Größe in der Erkennungsphase möglich.

Wenngleich beide Varianten in Abb. 7.7 von dem Demonstrator unterstützt werden, ist den-



(1) (2)

Abbildung 7.7: Implementierte Editierbefehle

noch Variante (2) zu bevorzugen, da dieses Symbol sehr sicher erkannt wird. Wie in den nachfolgenden Abschnitten noch im Detail gezeigt wird, können diese Editiersymbole universell für die Funktionen ’Löschen’, ’Ersetzen’ und ’Einfügen’ verwendet werden.

Für die Funktion ’Löschen’ sowie für die übrigen Editierfunktionen gilt, dass diese sich ohne Änderungen an der gewählten Gesamtarchitektur des Demonstrators integrieren lassen. Die Änderungen beschränken sich lediglich auf die Einführung neuer Modelle für die Steuersymbole in den Dekoder und einen erweiterten Parser, der bei erkannten Steuersymbolen entsprechende Änderungen auf vorherigen Versionen des Dokumentes vornimmt. Dazu ist es natürlich erforderlich, dass der Parser über eine entsprechende Verwaltung der Historie verfügt, sowie über eine Synchronisation der verschiedenen Versionen von handschriftlichen Eingabedokumenten, Transkribierungen mit geometrischen Merkmalen und Ausgabedokumenten. Die Historie ist als Stapel mit vorzugebender Maximalgröße ausgeführt. Ein Stapelelement wiederum entspricht jeweils einer Eingabe mit Erkennungsergebnis und enthält demnach einen Verweis auf die handschriftlichen Eingabedaten, sowie die verkettete Liste mit den Transkribierungen, Segmentgrenzen und geometrischen Merkmalen.

Der Vorteil der Listenstruktur kommt auch bei der Ausführung manueller Korrekturbefehle zum tragen, da hier teilweise ganze Zeichensequenzen zu entfernen oder durch neue Sequenzen zu ersetzen sind.

7.6.2 Löschen

Um bestimmte Teile einer Formel zu löschen, ist der betreffende Bereich mit dem Steuersymbol zu markieren. Nach erneuter Erkennung wird der entsprechende Bereich aus dem Ausgabedokument entfernt. Abb. 7.8 verdeutlicht dies an einem Beispiel. Der erste Schritt

zeigt das Original der Eingabe, wie es auf dem *Canvas* des Systems (s. auch Abb. 8.1) angezeigt wird. Eine erstmalige Erkennung generiert ein entsprechendes Ausgabedokument (`mx+b=y`). Im zweiten Schritt wird dann der zu löschende Bereich markiert und die Erkennung erneut gestartet. Das Ergebnis ist eine Ausgabedatei mit dem Inhalt `mx=y`.

Während im Canvas sämtliche Stiftaktionen visualisiert werden, schickt die Benutzerober-

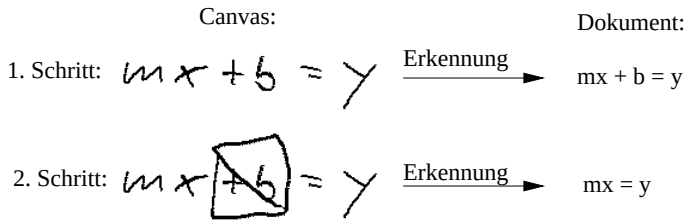


Abbildung 7.8: Löschen von Teilausdrücken

fläche immer nur die Daten in den Erkennungsprozeß, die seit dem *letzten* Erkenneraufruf hinzugekommen sind. In dem gezeigten Beispiel bedeutet dies also, dass der Dekoder lediglich das (hinzugefügte) Steuersymbol zu sehen bekommt. Die einzige Ausgabe des Dekoders ist somit die Transkribierung des Steuersymbols 'ctrl' und das temporale Alignment der Abtastvektoren. Daraufhin werden wiederum geometrische Merkmale (umschreibendes Rechteck und Mittelpunkt) des Steuersymbols bestimmt und dies zusammen mit der Transkribierung 'ctrl' an den Parser weitergeleitet.

Bei dem nun aufgetretenen Steuerbefehl wird von dem Parser zunächst das letzte, d. h. oberste Stapelelement (Verweis auf handschriftliche Eingabe und verkettete Liste) vom Stapel geholt, auf welches die Operation anzuwenden ist. Zusammen mit den Parametern des umschreibenden Rechtecks des Steuersymbols können die zu löschenden Symbole nun bestimmt werden, indem die Liste erkannter Zeichen nach solchen Einträgen durchsucht wird, deren Mittelpunkt (m_x, m_y) innerhalb des umschreibenden Rechtecks des Steuersymbols liegt. Die auf diese Weise bestimmten Zeichen werden anschließend aus der Liste gelöscht.

Um größere Lücken zwischen den Formelelementen zu vermeiden, werden die geometrischen Merkmale der nachfolgenden verbleibenden Zeichen um einen gewissen Offset verschoben. Dieser Offset wird bestimmt, indem ein umschreibendes Rechteck für die gesamte Sequenz zu löschender Zeichen berechnet wird. Das umschreibende Rechteck muß dabei jedoch nicht mehr aus den Vektordaten ermittelt werden, sondern kann durch Maximum- bzw. Minimumbildung über alle enthaltenen umschreibenden Rechtecke bestimmt werden. Die Verschiebung der Zeichen betrifft an dieser Stelle zunächst die Korrektur der geometrischen Merkmale um den ermittelten Offset. Für eine spätere Angleichung des Schriftbildes (Funktion 'Neuzeichnen') wird der Offset für das jeweilige verschobene Zeichen in einer zusätzlichen Variablen innerhalb des Listenelementes gespeichert.

Abschließend erfolgt die Ausgabe des aktualisierten Dokumentes und die Ablage der aktua-

lisierten Liste als neues Stapelelement.
Auf diese Weise ist es natürlich auch möglich mehrere Korrekturzeichen in einem Arbeitsschritt zu verarbeiten (Abb. 7.9), indem die Suche nach von Steuersymbolen überdeckten Zeichen für jedes auftretende Steuersymbol durchgeführt wird.

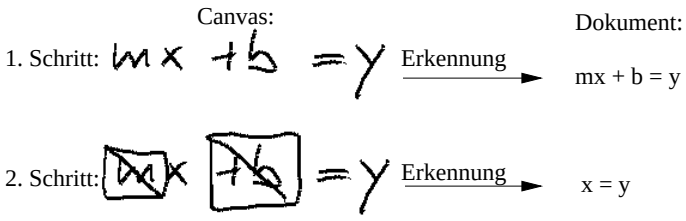


Abbildung 7.9: Löschen mehrerer Teilausdrücke

7.6.3 Ersetzen

Die Funktion ’Ersetzen’ basiert wie die Funktion ’Löschen’ ebenfalls auf dem Prinzip, die von der Modifikation betroffenen Zeichen mittels eines handgezeichneten Steuersymbols zu selektieren (Abb. 7.10). Beachtlich ist, dass trotz der deutlich unterschiedlichen Größen der in Abb. 7.10 und Abb. 7.8 gezeigten Steuersymbole, beide Realisierungen durch nur ein einziges HMM repräsentiert werden.
Die Handhabung der Funktion ’Ersetzen’ soll an dem in Abb. 7.10 gezeigten Beispiel verdeutlicht werden. Der erste Schritt zeigt das eingegebene Original (Canvas) mit der entsprechenden Ausgabe nach erfolgter Erkennung (Dokument). Im zweiten Schritt wurde dann der zu ersetzende Bereich mittels des Steuersymbols selektiert. Schritt drei zeigt, wie nach dem Einfügen des Steuersymbols handschriftlich der ersetzende Ausdruck hinzugefügt wurde.

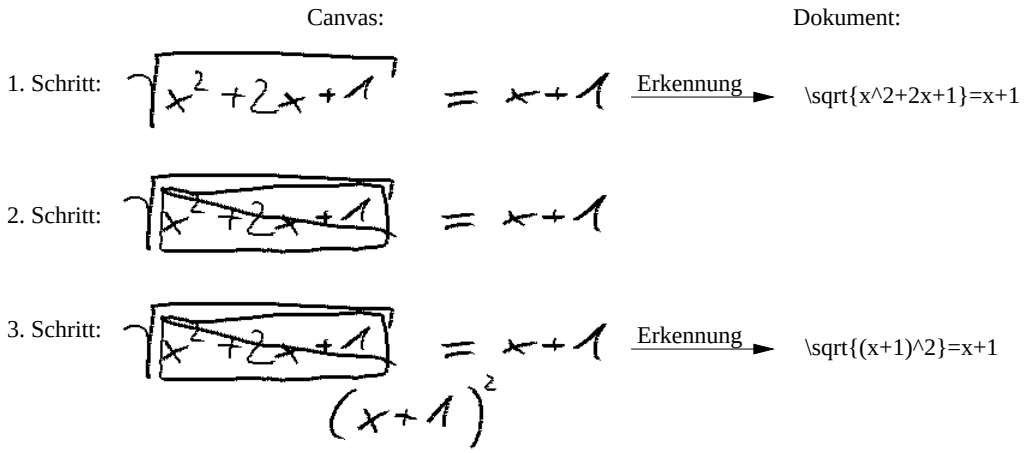


Abbildung 7.10: Ersetzen von Teilausdrücken

Nach einer abschließenden Erkennung in Schritt drei werden die Änderungen in das Ausgabedokument übertragen.
Die im Beispiel gezeigte Reihenfolge der Eingabe von

1. Steuersymbol und

2. ersetzendem Ausdruck bestehend aus Nicht-Steuersymbolen

ist ein wesentliches Merkmal für die Umsetzung der Funktion ’Ersetzen’. Wie bereits erwähnt, werden im zweiten Erkenneraufruf lediglich die hinzugekommenen Daten von der Oberfläche an den Erkenner übergeben. In dem gezeigten Beispiel heißt dies, dass der Parser nach der Ermittlung der geometrischen Merkmale die folgende Zeichensequenz mit entsprechenden geometrischen Merkmalen empfängt: `’ctrl ( x + 1 ) 2’`. Tritt - wie in dem vorliegenden Fall - ein Steuersymbol auf, ist dies grundsätzlich ein Signal für den Parser Änderungen an einem vorherigen Dokument vorzunehmen. Im Unterschied zu dem Beispiel in Abb. 7.8, wo auf ein Steuersymbol kein Nicht-Steuersymbol folgte, erkennt der Parser im Beispiel in Abb. 7.10, dass eine Substitution vorzunehmen ist daran, dass dem Steuersymbol eine Zeichenkette aus Nicht-Steuerzeichen folgt. Die Bestimmung der zu ersetzenden Zeichen verläuft zunächst analog zu der Vorgehensweise bei der Funktion ’Löschen’. Im einem weiteren Schritt wird dann das Erkennungsergebnis strukturiert, welches dem Steuersymbol folgt. Dies erfolgt in der beschriebenen Weise, wie bei einer Erkennung ohne manuelle Korrekturen. Um mehrere Substitutionen in einem Arbeitsschritt vornehmen zu können, ist es auch hier möglich wiederholt die Folge ’Steuersymbol, ersetzender Ausdruck’ einzugeben.
Die eigentliche Ersetzung kann nun erfolgen, indem der durch das führende Steuerzeichen selektierte zu ersetzende Ausdruck aus der verketteten Liste entfernt wird und an dieser Stelle der ersetzende Ausdruck eingefügt wird. Für multiple Substitutionen wird dieser Vorgang analog für jedes auftretende Steuersymbol wiederholt. Um auch hier Überlappungen oder Lücken zu vermeiden, werden die nachfolgenden verbleibenden Zeichen um einen Offset verschoben. Dieser Offset wird aus den Differenzen der umschreibenden Rechtecke von ersetzendem und zu ersetzendem Ausdruck ermittelt. Das Ergebnis wird wieder ausgegeben und aktualisiert auf dem Stapel abgelegt.

7.6.4 Einfügen

Bei der Funktion ’Einfügen’ handelt es sich im Grunde im eine spezielle Variante einer Substitution. Wie das Beispiel in Abb. 7.11 zeigt, wird mittels eines sehr schmalen Steuersymbols die Stelle markiert, an die der ersetzende Ausdruck eingefügt werden soll. Der Parser erkennt den Korrekturbefehl ’Einfügen’ daran, dass wie bei einer Substitution ein Steuersymbol von Nicht-Steuersymbolen gefolgt wird, wobei das Steuersybmol keinen Mit-

telpunkt irgend eines anderen Zeichens des vorausgegangenen Erkennungsergebnisses überdecken darf. Nach erfolgter Strukturierung der nachfolgenden Nicht-Steuerbefehle werden diese wiederum an entsprechender Stelle in die Liste eingefügt. Die ursprünglich vorhandenen Zeichen werden wiederum um einen Offset verschoben, um Überlappungen zu vermeiden. Auch hier gilt, dass mehrere Korrekturanweisungen in einem Erkenneraufruf abgearbeitet werden können.

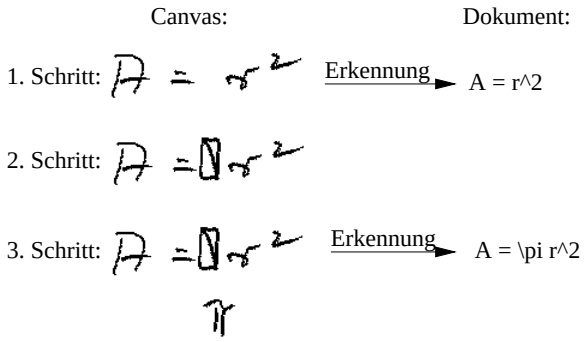


Abbildung 7.11: Einfügen von Teilausdrücken

7.6.5 Rückgängig und Wiederholen

Um während der Entwicklung einer Formel den Zugriff auf verschiedene Versionen zu ermöglichen, wurden die Funktionen ’Rückgängig’ und ’Wiederholen’ realisiert. Mit Hilfe dieser Funktionen können vom Benutzer eingeführte Änderungen zurückgenommen bzw. wieder hinzugefügt werden.
Die Umsetzung dieser Funktionen ist aufgrund der gewählten Struktur für die Historie relativ effizient umzusetzen. Ausgehend von der aktuellen Version eines Dokumentes entspricht der Aufruf von ’Rückgängig’ (’Wiederholen’) dem absteigen (aufsteigen) innerhalb des Stapels. Bei wiederholtem Aufruf der Funktion ’Rückgängig’ liegt das aktuelle Stapelelement natürlich unterhalb des obersten Stapeleintrags. Modifikationen der jetzt aktuellen Version lassen alle späteren Einträge auf dem Stapel ungültig werden. Diese Einträge werden somit aus dem Stapel gelöscht.

7.6.6 Neuzeichnen

Wie die vorangegangenen Beispiele zeigen, kann bereits nach einer Korrekturmaßnahme die handschriftliche Eingabe recht unleserlich werden. Ein weiterer Effekt durchgeführter Modifikationen ist, dass die Divergenz zwischen Eingabe- und Ausgabedokument mit jeder weiteren Änderung zunimmt. Sollen nun zusätzliche Änderungen auf einem bereits modifizierten Dokument vorgenommen werden, könnte diese Divergenz zu Problemen führen, da

vom Benutzer gesetzte Steuersymbole - bedingt durch Verschiebungen aus früheren Ersetzungen oder Einfügungen - nicht mehr exakt die nun zu ersetzenden Zeichen treffen. Die visuelle Rückkopplung zur Stiftführung an den Benutzer ist nicht mehr gegeben. Zu diesem Zweck wurde die Funktion 'Neuzeichnen' eingeführt, welche die Handschrifteingabe an das Ausgabedokument anpasst und automatisch nach jeder manuellen Korrektur ausgeführt wird. Optional kann anschließend das automatisch angeglichene handschriftliche Eingabedokument geladen und im Canvas visualisiert werden.
Die Funktion 'Neuzeichnen' erzeugt aus den aktuellen Änderungen und dem vorangegangenen Erkennungsergebnis mit entsprechender Segmentierungsinformation und geometrischen Merkmalen der handschriftlichen Eingaben eine aktualisierte Synthese der handgeschriebenen Formel.
Die einzelnen Segmente einer Formel werden dazu zunächst anhand der zeitlichen Segmentgrenzen aus dem Eingabedatenstrom ausgeschnitten. Wenn nun Verschiebungen aufgrund von Löschungen oder Einfügungen vorgenommen wurden, existiert für das betroffene Listenelement ein von Null verschiedener Offset, der aus jeder Verschiebung ggf. individuell mitgeführt wurde. Dieser Offset kann nun auf die Kartesischen Koordinaten der Abtastvektoren eines jeden Segmentes aufaddiert werden.
Diese Vorgehensweise soll anhand des Beispiels in Abb. 7.12 verdeutlicht werden. Schritt eins zeigt bereits die im Ausgabedokument durchgeführte Änderung und die handschriftliche Eingabe mit hinzugefügtem Steuersymbol und einzufügendem π. In dem gezeigten Beispiel ist das 'r' sowie die hochgestellte '2' von der Verschiebung betroffen, da diese sich rechts neben dem Steuersymbol befinden (vergl. LRTD-Bedingungen). Die das 'r' und die '2' betreffenden Listenelemente enthalten also einen von Null verschiedenen Offset, der anzeigt um wieviel und in welche Richtung die geoemtrischen Merkmale korrigiert wurden. Um

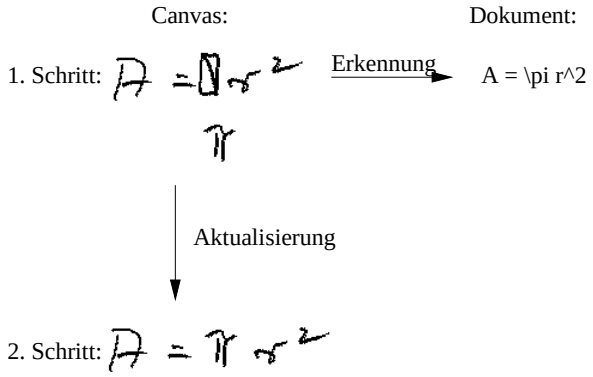


Abbildung 7.12: Aktualisierung des Schriftbildes

das an das Ausgabedokument angeglichene Schriftbild zu generieren, werden anhand der zeitlichen Segmentierung die einzelnen Symbole aus der Abtastsequenz ausgeschnitten. Die Kartesischen Koordinaten dieser Abtastvektoren enthalten zunächst noch die ursprünglichen

Werte. Da für die Zeichen 'A' und '=' kein Offset existiert, wird die originale Abtastfolge übernommen. Die Positionen des nach dem Steuersymbol auftretenden Zeichens π sowie das 'r' und die '2' wurden jedoch in der Funktion 'Einfügen' korrigiert und mit einem entsprechenden Offset versehen, welcher auf die Abtastfolge $(x, y)(k)$ des jeweiligen Segmentes aufaddiert wird.
Für die weitere Verarbeitung insbesondere für eine wiederholte Erkennung ist es notwendig die Abtastfolge der Zwischenräume zu rekonstruieren, da diese ja bei der Erkennung einem speziellen 'space'-Modell zugeordnet werden. Das Fehlen der Zwischenräume in der zeitlichen Abfolge würde zwangsweise zu Erkennungsfehlern führen. Diese Zwischenräume können durch lineare Interpolation zwischen Endpunkt eines Zeichens und Anfangspunkt des Nachfolgezeichens rekonstruiert werden. Die 'Spaces' am Anfang vor dem ersten Zeichen und am Ende nach dem letzten Zeichen können aus dem jeweiligen Original kopiert werden. Die Länge der Abtastvektoren dieser Zwischenräume bzw. deren Anzahl kann willkürlich gewählt werden, da bei jeder Erkennung zunächst eine Neuabtastung vorgenommen wird, die für äquidistante Abtastpunkte sorgt.
Das so generierte Schriftbild, repräsentiert durch die rekonstruierte Abtastfolge, kann schließlich visualisiert werden und für eine Weiterverarbeitung verwendet werden.

7.7 Ergebnisse

Bei der Bewertung des Formeleditors ist es zweckmäßig die Bereiche Ersteingabe und Modifikation zu trennen. Die Erkennungsgenauigkeit bei der Ersteingabe kann, analog zu der Worterkennung, wieder anhand eines Test-Sets geschätzt werden. Die entsprechenden Werte für die drei Schreiber 'ank', 'bec' und 'sla' sowie die Duchschnittswerte sind in Tab. 7.1 zusammengestellt.
Zu unterstreichen ist hierbei jedoch, dass die Erkennungsraten auf der Ausgabe des Dekoders berechnet wurden. Die Bewertung der Formelerkennung auf Basis der Erkennungsraten kann daher nur eine ungefähre Idee von der eigentlichen Benutzbarkeit des Systems geben, da in einfacher Form nur die Fehler der Dekoderausgabe meßbar sind. Um Fehler des Parsers zu messen, müßte ein umfangreiches mathematisches Regelwerk für die Bewertung genutzt werden, da hierbei wiederum verstärkt die kontextbezogene Bedeutung der einzelnen Symbole einzubeziehen wäre.

	ank	bec	sla	ø
Korrektheit	97.4 %	97.5 %	89.5 %	94.8 %
Akkuratheit	96.6 %	97.1 %	89.4 %	94.4 %

Tabelle 7.1: Erkennungsraten des schreiberabhängigen Formeleditors

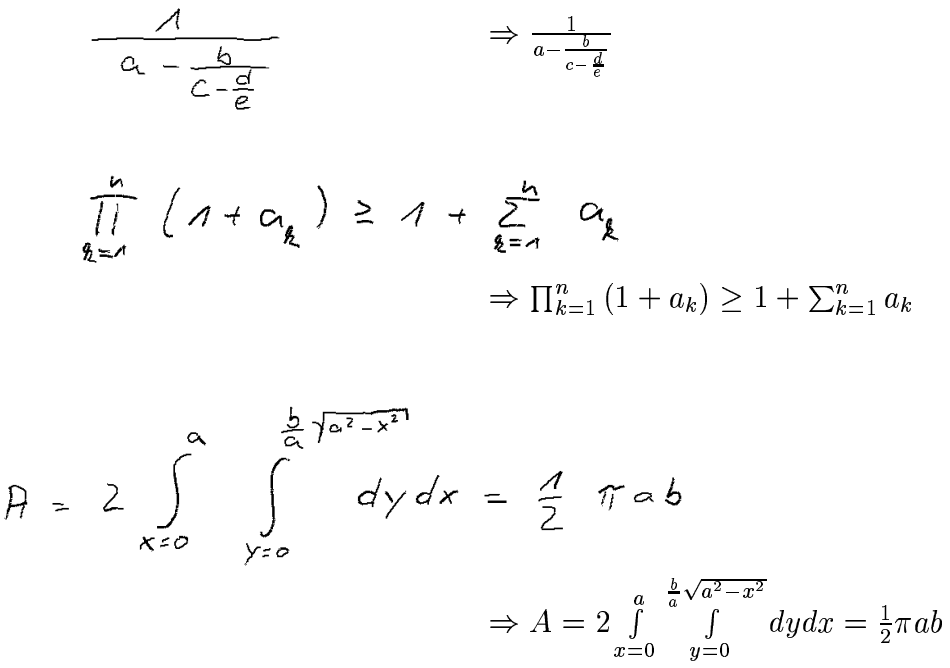


Abbildung 7.13: Korrekt erkannte Beispiele

Dazu betrachte man die folgenden Beispiele. Wird eine Zahl, die als Index durch den Schreiber tiefgestellt wurde, durch den Dekoder korrekt erkannt, durch den nachfolgenden Parser aber fehlerhaft als *in-line* gesetzt, statt als Index, so wird dieser Fehler in der Berechnung der Erkennugsrate außer Betracht bleiben. Ein Durch den Dekoder gelöschter Multiplikationspunkt zwischen einem Koeffizienten und einer Funktion würde hingegen als Fehler gezählt werden, obwohl beide Ausdrücke gültig und darüber hinaus auch gleichwertig sind.
Dennoch ist festzuhalten, dass der Parser um so genauere Ergebnisse liefert je genauer das Alignment von dem Dekoder geliefert wird. Das Alignment wiederum ist dann exakt, wenn die Modelle genau sind. Die Genauigkeit der Modelle spiegelt sich wiederum in der Erkennungsrate - meßbar nach dem Dekoder - wieder, was die Betrachtung der Erkennungsraten in gewisser Weise auch bei dem Formeleditor rechtfertigt. Ergänzend zu den Ergebnissen in Tab. 7.1 sind in Abb. 7.13 einige korrekt erkannte Beispiele des Schreibers 'ank' wiedergegeben, die durchaus komplexe Strukturen aufweisen und von dem System korrekt umgesetzt werden.
Die Ergebnisdiskussion hat gezeigt, dass sich die Erfassung meßbarer Ergebnisse bei der Erkennung von Ersteingaben schwierig gestaltet. Dies gilt um so mehr bei der Bewertung der Editierfunktionen. Die Interaktionen sind hier mit Dialogen vergleichbar, bei denen sich Aktionen und Reaktionen von Mensch und Maschine gegenseitig beeinflussen. Derartige Systeme sind lediglich durch sehr aufwendige Benutzbarkeitsstudien an einer größeren Anzahl Benutzer zu evaluieren und im Rahmen dieser Arbeit nicht realisierbar.

7.8 Kapitelzusammenfassung

Ein Schwerpunkt aktueller Forschungsarbeiten in dem Bereich der Formelerkennung bildet nach wie vor die Untersuchung der unteren Systemebenen, wie Segmentierung und anschließende Erkennung der Einzelzeichen. In dieser Arbeit konnte hingegen gezeigt werden, dass die integrierte Erkennung und Segmentierung durch einen HMM-basierten Ansatz wesentliche Vorteile gegenüber klassischen Verfahren bietet. Das Segmentierungsproblem als solches tritt explizit nicht mehr auf. Des weiteren konnte die Segmentierungsinformation des Dekoders für weitere Verarbeitungsschritte genutzt werden. Insbesondere wurde die Segmentierungsinformation für das Parsing der Dekoder-Ausgabe verwendet.

Im Bereich der handschriftlichen Formelerkennung war bisher die Erkennung und Ausführung manueller Korrekturbefehle unberücksichtigt. Mit der Einführung universell verwendbarer Editiersymbole konnten die Befehle ’Löschen’, Einfügen’ und ’Ersetzen’ realisiert werden. Die anschließende Synthese des Schriftbildes aus dem aktualisierten Erkennungsergebnis sorgt für die nötige Synchronisation zwischen Eingabe und Ausgabedokument. Schließlich wurde die gesamte Funktionalität in einen nahezu echtzeitfähigen Demonstrator integriert, dessen Oberfläche in Abb. 8.1 gezeigt ist.