Entwicklung eines Kodierungssystems für Sonderzeichen und spezielle Notationen im CSLBIB2000-Bibliothekssystem

Holger Westphal

30. Oktober 1995

Zusammenfassung

Im Zusammenhang mit der Konzeption des CSLIB2000 Bibliothekssystems ergibt sich die Frage nach der Repräsentation von Umlauten und anderen nationalen Sonderzeichen bei Beschränkung auf 7- oder 8-Bit Codes in der Datenbank. Der hier gemachte Vorschlag berücksichtigt sowohl den ISO 8859-Latin1 Zeichensatz als auch die Darstellungsmöglichkeiten des Textformatiersystems LaTeX. Beim Entwurf der Kodierung findet die sich aus der Kodierung ergebene Sortierreihenfolge und die Herabsetzung der Kodierungsunsicherheit mittels regulärer Ausdrücke bei der Anfrage besondere Beachtung.

1 Einleitung

Das Problem der Zeichenkodierung ist bei dem Vorhaben der Entwicklung des CSLIB2000 Bibliothekssystems als besonders dringlich einzuschätzen, da es gelöst sein sollte, bevor größere Menge an Daten in das System wandern und da es die Konzeption jeder Benutzeroberfläche beeinflußt. Hierbei sollte die Prämisse gelten, soviel Information wie möglich zu erhalten, ohne die Retrievalleistung zu schmälern.

2 Problembestimmung

Das Problem besteht darin, ein Kodierungsschema zu finden, welches folgende Eigenschaften aufweist:

- Es beschränkt sich auf die Verwendung von 7 bit Zeichen, wobei nur „printable character" verwendet werden, also keine Steuerzeichen. Konkret bedeutet dies eine Beschränkung auf den Codebereich von 32-126, entsprechend ISO-646/ASCII. Hiermit werden keine besonderen Anforderungen an speichernde und verarbeitende Systeme gestellt.
- Unsicherheit bzgl. der Kodierung, welche sich durch unterschiedliche Qualität der ins System eingebrachten Daten ergeben (wurde z.B. „ae" statt „ä" gewählt oder „a" statt „à"), sind relativ schematisch, z.B. durch reguläre Ausdrücke, zu reduzieren.
- Eine Sortierung auf der Ersatzdarstellung liefert, wenn auch keine korrekte, so zumindest eine „vernünftige" Anordnung der Wörter, ohne daß zuerst eine Sortierwert-Repräsentation hergestellt werden muß. Als „vernünftig" soll hierbei bereits gelten, wenn ein „ä" immerhin noch vor einem „b" erscheint, aber nicht erst nach einem „z". Diese Bedingung kann den Implementierungsaufwand für die Aufbereitung von Daten reduzieren.

2.1 Abgrenzung

Prinzipiell macht es für ein im deutschen Sprachraum angesiedeltes System keinen Sinn, Zeichensysteme zuzulassen, welche nicht auf der lateinischen Schrift basieren. Für wichtige, nicht-lateinische Schriften sollten stets Transkriptions- oder Transliterationssysteme zu Anwendung kommen (siehe z.B. [DGKM73, S.787]).

Die hiermit verbundene Transkriptions- bzw. Transliterationsunsicherheit ist natürlich durch ein Kodierungssystem allein nicht behebbar. Hierzu können, insbesondere bei Personennamen, nur entsprechende Verweistechniken zur Anwendung kommen, welche in Standards für die alphabetische Katalogisierung auch vorgesehen sind (z.B. [Mün92, S.75]).

Auch prinzipiell nicht aufhebbar sind Transkriptions- bzw. Transliterationsunterschiede in unterschiedlichen Sprachen; z.B. wenn derselbe (z.B. russische) Name im Amerikanischen anders transkriptiert bzw. transliteralisiert wird als im Deutschen. Hier bleibt zunächst auch nur die Verweistechnik.

Beim Aufbau von Thesauri oder Volltextindexen könnte derartigen Mehrdeutigkeiten prinzipiell durch Synonymeinträge begegnet werden. Allerdings wären hier für alle denkbaren Varianten Einträge vorzusehen. Außerdem sollte eine der eingespeicherten Formen eine Form sein, welche den geringsten Informationsverlust aufweist, was einen nicht von einer Darstellung dieser Formen entbindet (z.B. sollten aus der Datenbank erzeugte Literaturverzeichnisse die korrekten Schreibweisen enthalten: Umlaute, Akzente, usw.). Eine geeignete Kodierung könnte, in Verbindung mit regulären Ausdrücken bei der Anfrage, einen Teil dieser Einträge überflüssig machen und damit bereits auf lexikalischer Ebene für Voraussehbarkeit sorgen (z.B.: ein „ä" ist entweder als „a" oder als „ae" oder als „ä" kodiert).

Es wurden bereits Methoden entwickelt, welche sich mit der Ähnlichkeit von Zeichenketten befassen (siehe [Fal85] für einen Überblick und [PPF95] für einen Vergleich der Retrieval-Qualität). Diese Verfahren sollen vorwiegend der Behandlung von Problemen im Zusammenhang von Schreibfehlern und der Suche nach phonetisch verwandten Wörtern dienen. Eine explizite Anwendung dieser Methoden im Zusammenhang mit internationalen Kodierungen und Mehrsprachigkeit ist mir nicht bekannt. Das Problem der „Identifikation von Namen" wird weiterhin als offen angesehen [PPF95].

Die erwähnten Methoden erfordern den Aufbau spezieller Indexe, welche dann jeweils Identität oder Ähnlichkeit nach einem festen Schema realisieren. Neben der damit verbundenen Unflexibilität weisen sie also auch Redundanz auf. Die hier beschriebene Methodik der Verwendung regulärer Ausdrücke bei der Anfrage ist hingegen prinzipiell auf den unveränderten Originaldaten (in der korrekten oder genauesten Schreibweise) möglich. Die hier behandelte Methodik bezieht sich auf Probleme, welche im Zusammenhang mit der Kodierung und Sortierung dieser Daten stehen und weniger im Zusammenhang mit einer umfassenderen Steigerung des Recall auf lexikalischer Ebene. Sie kann deshalb als Ergänzung der erwähnten Methoden verstanden werden, wobei allerdings nicht untersucht wird, ob die Ansätze sinnvoll integrierbar sind.

Darstellungsmittel zur Hervorhebung oder speziellen Formatierung sollen keine Berücksichtigung finden, da diese Informationen aus zum Retrieval bestimmten Daten entfernt sein sollten. Gliederungsinformationen, wie sie z.B. für die Volltextrecherche sinnvoll sein können [Rei94], werden hier ebenfalls nicht betrachtet, obwohl eine Integration nicht ausgeschlossen werden sollte.

2.2 Bestehende Kodierungsysteme

Eine Variante, welche versuchen würde, immer die (landesübliche) Originalschreibweise zu speichern (was ja gerade im Regelfall nicht möglich sein wird), würde es erforderlich machen, potentiell jede Schrift kodieren zu können. Ansätze, welche sich mit dem letzteren Problem befassen, sind z.B. Unicode bzw. ISO-10646 (Überblick in [Beh92]). Davon abgesehen, daß dies tatsächlich die aufwendigste Lösung wäre, würde sie erstaunlicherweise keine der eingangs gemachten Bedingungen erfüllen.

Eine weitere Alternative bestünde in der Beschränkung auf einen Zeichensatz, welcher auch tatsächlich von einem bestehenden System unterstützt werden kann (d.h. bei Anzeige und Eingabe). Für Unix bzw. das darauf aufsetzende X-Windowssystem wäre das z.B. der ISO 8859-Latin1 Zeichensatz. Diese Beschränkung kann jedoch als etwas willkürlich erscheinen, wenn man sie mit den Standards anderer Systeme vergleicht. Wieder werden jedoch keine der eingangs gemachten Bedingungen erfüllt.

Als weiterer Ansatzpunkt könnten die Möglichkeiten eines Textverarbeitungssystems dienen. Im Bereich der Informatik und Mathematik ist hier, aufgrund der großen Anzahl von Publikationen, die mit ihm erstellt werden, insbesondere das Textformatiersystem LaTeX zu nennen [Lam86]. Die Bedeutung dieses Systems für das CSLIB2000 Bibliothekssystem wird noch dadurch erhöht, daß es ein Textformat für die Datensätze einer Literaturdatenbank definiert, welche durch ein spezielles Programm (BibTeX, [Lam86, Pat88]) ausgelesen und zur Zitierung in einem Dokument formatiert werden können. Inzwischen gibt es bereits Literaturserver, welche Anfrageergebnisse in diesem Textformat liefern [Ach95]. Die Kodierung von LaTeX erfüllt zumindest bereits die Bedingung, daß sie mit 7 bit auskommt.

Im Rest des Dokuments wird nun ein Kodierungsschema vorgeschlagen und diskutiert, welches alle drei eingangs gemachten Bedingungen erfüllt.

3 Kodierungssystem

Es wird das prinzipielle Schema der Kodierung vorgestellt und danach der Umfang der zu kodierenden Zeichen festgelegt.

3.1 Kodierungsschema

Neben den bereits erwähnten Kodierungssystemen gibt es solche, welche auf sog. diakritischen Zeichen aufbauen. Diakritische Zeichen sind dadurch gekennzeichnet, daß ihr Ausdruck keine automatische Weiterbewegung der Schreibposition bewirkt, so daß dieses Zeichen mit dem folgenden Zeichen übereinander gedruckt wird (Non-Spacing Characters, [Bor91, S. 5.11]). ISO 6937 sieht hierzu (mindestens) 15 Zeichen vor, welche sich am Beispiel mit dem Folgezeichen „a“ folgendermaßen darstellen:

à á â ã ā ă ȧ ä a̋ å a̧ a̱ ? ạ ǎ

ISO 10646 definiert über 60 solche Zeichen.

Es ist klar, daß ein System, welches diakritische Zeichen voranstellt, das Sortierkriterium nicht erfüllen kann.

Dies gilt auch für die entsprechenden LaTeX -Darstellungen:

```
\`{a} \'{a} \^{a} \~{a} \={a} \u{a} \.{a} \"{a} \H{a} \aa\ \c{a} \b{a} ? \d{a} \v{a}
```

Der hier gemachte Vorschlag geht deshalb einen grundsätzlich anderen Weg. Er geht davon aus, daß sich zu jedem darzustellenden Zeichen ein lateinisches Grundzeichen aus dem 7 bit ASCII-Zeichensatz finden läßt, von dem es abgeleitet werden kann (oder wo diese Ableitung definiert wird). Jedes Zeichen wird dargestellt als Grundzeichen *gefolgt* von einem Erweiterungs- bzw. Ableitungszeichen[1].

Die folgenden Kodierungen mögen dies verdeutlichen:

ä ← `a"`, à ← `a‘`, ...

Damit nun einerseits aber die Kodierung eindeutig ist, andererseits die Sortierbedinung eingehalten wird, müssen auch die Grundzeichen selbst durch zwei Zeichen dargestellt werden:

a ← `a!`, c ← `c!`, ...

Das Erweiterungszeichen „!“ wurde dabei so gewählt, daß das Grundzeichen, selbst bei einer Sortierung nach den ASCII-Codes, stets vor allen abgeleiteten Zeichen kommt. Als erstes folgen die deutschen Umlaute mit dem Erweiterungszeichen „`"`“.

Als nächstes ist die kodierungstolerante Selektion mittels regulärer Ausdrücke zu untersuchen. Wir nehmen zunächt den häufig anzutreffenden Fall, daß Akzente einfach weggelassen werden. Selektiere „?“ genau ein beliebiges Zeichen und durch [] eingeschlossene Zeichen genau eines dieser Zeichen, dann selektiert die Anfrage

„`A!m!p!e?r!e!`“ Ampere, Ampère, Ampére, Ampëre, ...
„`A!m!p!e[!‘]r!e!`“ Ampere, Ampère
„`A!m!p!e‘r!e!`“ Ampère

Speziell im Deutschen treten häufig die Ersatzdarstellungen für die Umlaute und ß auf. Selektiere „`*`“ null oder beliebig viele Zeichen, dann selektiert die Anfrage

„`G!r!u"s"e!`“ Grüße
„`G!r!u!e!s!s!e!`“ Gruesse
„`G!r!u[!"][es]["!]*e!`“ Grüße, Gruesse, Gruße, Gruene Nuesse, ...
„`G!r!u[!"][es]["!][es]!*`“ Grüße, Gruesse, Gruße, ...

Keine Anfrage liefert ohne Erweiterung der regulären Ausdrücke (um den Oder-Operator „`|`“, z.B.: „`G!r!u("s"|!e!s!s!)e!`“) sicher nur Grüße und Gruesse. Diese Erweiterungen unterstützen aber viele Systeme nicht. Allerdings besteht die Möglichkeit der Realisierung als vollständiges Oder, was einer Anfrage „`(G!r!u"s"e!|G!r!u!e!s!s!e!)`“ entspräche.

Die Probleme mit den deutschen Umlauten bei der Anfrage sind allerdings unabhängig von dem hier behandelten Kodierungsverfahren zu sehen, da es keinen Unterschied macht, ob ein Zeichen durch einen Code oder durch zwei Codes repräsentiert ist.

3.2 Zeichensatz

Nachdem das Kodierungsschema festgelegt wurde, ist als nächstes zu klären, welche Zeichen kodiert werden sollen. Damit verbunden ist die Frage, welche Zeichen als Grundzeichen auftreten sollen und welche als Erweiterungszeichen.

Die Zeichenauswahl könnte nach einem der folgenden Kriterien erfolgen:

- Es werden nur deutsche Sonderzeichen berücksichtigt.

[1] Dies entspricht nach [Beh92] auch dem Vorgehen in ISO 10646, soweit es die Zeichen betrifft, welche durch Zeichenkombination aus einem Grundzeichen und einem diakritischen Zeichen gewinnbar sind.

- Es werden nur Zeichen berücksichtigt, welche in der geplanten Anwendungsumgebung unterstützt werden. In unserem Falle hieße dies ISO 8859-Latin1 bzw. ISO 646/Latin1 bzw. ISO 10646 Group 0, Plane 0, Row 0.
- Es werden nur Zeichen berücksichtet, die LaTeX unterstützt.
- Alle lateinischen Zeichen nach ISO 10646 bzw. Unicode werden berücksichtigt. D.h. mindestens Group 0, Plane 0, Row 0, 1, 2, 14 (was allerdings noch anhand der vollständigen Dokumentation geklärt werden müßte).

Es wird hier der Dringlichkeit wegen eine Verbindung aus LaTeX und ISO 8859-Latin1 vorgeschlagen, wobei eine Erweiterung in Richtung Unicode (lateinischer Teil) offen stehen sollte. Spezielle Berücksichtigung sollten die griechischen Buchstaben finden, da sie häufig an der Bildung feststehender Begriffe beteiligt sind (z.B. λ-Kalkül).

Zunächst sind die Grundzeichen zu definieren. Ein Blick in ISO 10646 Group 0, Plane 1, Row 1 ergibt, daß fast alle Buchstaben außer b, f, m, p, q, v und x als Grundzeichen denkbar sind. Es werden also alle Buchstaben von a bis z und von A bis Z als Grundzeichen definiert.

Darüberhinaus ist es sinnvoll, ein Zeichen

- vor den Ziffern „`/`“,
- nach den Ziffern „`:`“,
- vor den Großbuchstaben „`@`“,
- nach den Großbuchstaben „`[`“,
- vor den Kleinbuchstaben „`‘`“,
- nach den Kleinbuchstaben „`{`“

als Grundzeichen zu definieren, damit später Zeichen, die in der Sortierreihenfolge an den genannten Stellen erscheinen sollen, mit diesen Grundzeichen definiert werden können.

Die Menge der Ergänzungszeichen soll zunächst nicht beschränkt werden, obwohl sie die Größe der von Implementierungen genutzten Umkodierungstabellen bestimmt. Es sind alle Zeichen mit den Codes 33 („`!`“) bis 126 („`~`“) zugelassen.

Im Zusammenhang mit den griechischen Buchstaben tritt die Frage auf, ob diese als Literal oder in der Verbalform verwendet werden sollen: Also „der λ-Kalkül“ oder „der Lambda-Kalkül“. Es scheint so, als würde die Literalform eine Anfrage nur unnötig erschweren. Katalogisierungsregeln sehen in der Regel bestmögliche Verbalisierung vor. Ein Kompromiß könnte die Zulassung von LaTeX-Syntax sein: `\lambda-Kalk"ul`. Auf diese Weise könnten alle LaTeX-Zeichen in ihrer Verbalform in den Speicher wandern, ohne daß die Information darüber verloren geht, daß ein einzelnes Zeichen verwendet wurde. Auf diese Weise könnte auch in Formeln gesucht werden. Es wird also zunächst auf eine Literalform für griechische Buchstaben verzichtet.

Damit sich die Aufnahme von LaTeX-Syntax in das bisher beschriebene Kodierungssystem einpaßt, ist eine Menge von Startzeichen für LaTeX-Steuersequenzen zu definieren. Die Menge dieser Zeichen hat als Grundzeichen einen beliebigen Buchstaben, der zugleich der erste Buchstabe der Steuersequenz ist und als Erweiterungszeichen „\“. Die folgenden Zeichen der Steuersequenz werden als normale Buchstaben kodiert, wie bereits beschrieben. Auf diese Weise sind alle LaTeX-Steuersequenzen kodierbar, die mit einem Buchstaben beginnen, nach ihrer Verbalform sortierbar und als Steuersequenz oder verbal anfragbar. Das Beispiel „der λ-Kalkül“ würde also folgendermaßen kodiert werden:

```
d!e!r l\a!m!b!d!a!-K!a!l!k!u"l!.
```

Entsprechend selektiert die Anfrage

`l\a!m!b!d!a!-K!a!l!k!u"l!` λ-Kalkül
`[lL]!a!m!b!d!a!-K!a!l!k!u"l!` Lambda-Kalkül, lambda-Kalkül
`[lL][\!]a!m!b!d!a!-K!a!l!k!u"l!` Lambda-Kalkül, lambda-Kalkül, λ-Kalkül, Λ-Kalkül.

Bei der Übernahme von LaTeX-Texten ist zu beachten, daß viele als Steuersequenz kodierte Zeichen nur im Mathematik-Modus definiert sind.

Die vorläufige Zuordnung von Codes zu Zeichen ist in Tabelle 1 dargestellt. Die Tabelle wird vervollständigt, falls die Kodierung zum Einsatz kommen sollte.

Zeichen	Grundzeich.	Erweit.zeich.	ISO 10646	LaTeX
A-Z	A-Z	!	0041-005A	A-Z
a-z	a-z	!	0061-007A	a-z
À Á Â Ã Ä Å Æ	A	‘ ’ \^ ~ "’ A E	00C0-00C6	\‘{A} \’{A} \^{A} \~{A} "A \AA \AE
Ç	C	,	00C7	\c{C}
È É Ê Ë	E	‘ ’ ^ "	00C8-00CB	\‘{E} \’{E} \^{E} \"{E}
Ì Í Î Ï	I	‘ ’ ^ "	00CC-00CF	\‘{I} \’{I} \^{I} \"{I}
D (gestrichen)	D	-	00D0	—
Ñ	N	~	00D1	\~{N}
Ò Ó Ô Õ Ö	O	‘ ’ ^ ~ "	00D2-00D6	\‘{O} \’{O} \^{O} \~{O} "O
Kreuz	X	X	00D7	—
Ø	O	/	00D8	\O
Ù Ú Û Ũ Ü	U	‘ ’ ^ ~ "	00D9-00DC	\‘{U} \’{U} \^{U} \~{U} "U
...	...	...	...	...
ß	s	"	00DF	\"{s} \"s "s
...	...	...	...	...
à á â ã ä å æ	a	‘ ’ ^ ~ " a e	00E0-00E6	\‘{a} \’{a} \^{a} \~{a} "a \aa \ae
ç	c	,	00E7	\c{c}
è é ê ë	e	‘ ’ ^ "	00E8-00EB	\‘{e} \’{e} \^{e} \"{e}
ì í î ï	i	‘ ’ ^ "	00EC-00EF	\‘{i} \’{i} \^{i} \"{i}
delta (gestrichen)	d	-	00F0	—
ñ	n	~	00F1	\~{n}
ò ó ô õ ö	o	‘ ’ ^ ~ "	00F2-00F6	\‘{o} \’{o} \^{o} \~{o} "o
kreuz	x	x	00F7	—
ø	o	/	00F8	\o
ù ú û ũ ü	u	‘ ’ ^ ~ "	00F9-00FC	\‘{u} \’{u} \^{u} \~{u} "u
...	...	...	...	...

Tabelle 1: Kodierung

Latin1-Codes sind an der führenden „00“ des Hexcodes zu erkennen.

4 Sortierverhalten

Die Notwendigkeit, lexikalisch sortierte Listen von Daten zu erzeugen, entsteht an den Stellen, wo einem Menschen die weitere (manuelle) Suche in Datenbeständen nicht erspart werden kann (z.B. umfangreiches Anfrageergebnis, alphabetischer Index als Hypertext).

Zum Abschluß folgt deshalb noch ein Beispiel zum Sortierverhalten der Kodierung anhand von Tabelle 2. Die Reihenfolge ergibt sich aus einer einfachen Sortierung nach den ASCII-Werten der Zeichen der kodierten Zeichenketten, wie sie jedes Datenbanksystem und jede Standard-Sortierroutine durchführen kann. An der Anordnung der Paa-

Kodierung	Text
2-s!t!e!l!l!i!g!	2-stellig
A!m!p!e!r!e!	Ampere
A!m!p!e!r!e!m!e!t!e!r!	Amperemeter
A!m!p!e`r!e!	Ampère
A!p!f!e!l!	Apfel
B!a!n!a!n!e!	Banane
L!a!m!b!d!a!-K!a!l!k!u"l!	Lambda-Kalkül
a!e!p!f!e!l!	aepfel
a!f!f!e!	affe
a!p!f!e!l!	apfel
a!u!s!s!t!a!n!d!	ausstand
a"p!f!e!l!	äpfel
b!a!n!a!n!e!	banane
e!i!n!s!	eins
l!a!m!b!d!a!-K!a!l!k!u"l!	lambda-Kalkül
l!a!u!n!i!s!c!h!	launisch
l\a!m!b!d!a!-K!a!l!k!u"l!	λ-Kalkül
s!c!h!l!u!s!s!	schluss
s!c!h!l!u!s"	schluß
z!w!e!i!	zwei

Tabelle 2: Beispiel zum Sortierverhalten

re Ampere—Ampère, apfel—äpfel und lambda-Kalkül—λ-Kalkül werden die Schwächen dieser Sortierung deutlich.

Genügt die erreichte Sortierqualität nicht, kann in einem Anwendungsprogramm, wieder unter Verwendung von Standard-Sortierroutinen, nachsortiert werden, indem eine Sortierrepräsentation der Zeichenketten durch einfaches Weglassen aller Erweiterungszeichen erzeugt wird. Tabelle 3 zeigt die sich daraus ergebene Reihenfolge, wobei gleichzeitig die häufig unerwünschte Unterscheidung zwischen Groß- und Kleinschreibung aufgehoben wurde. Bei dieser einfachen Variante ist die Reihenfolge für die Zeichenketten, welche gleichen Sortierwert besitzen (Ampere—Ampère, Apfel—apfel—äpfel, Banane—banane, Lambda-Kalkül—lambda-Kalkül—λ-Kalkül) nicht determiniert bzw. vom Sortierverfahren abhängig. Die Behandlung von ß als ss ist auch nicht gewährleistet.

5 Grammatik

Zur Klarheit folgt jetzt zusammenfassend eine Grammatik für eine nach dem in Abschnitt 3 entwickelten Kodierungssystem kodierten Zeichenkette (String).

Kodierung	Sortierwert	Text
`2-s!t!e!l!l!i!g!`	`2-stellig`	2-stellig
`A!m!p!e!r!e!`	`ampere`	Ampere
`A!m!p!e‘r!e!`	`ampere`	Ampère
`A!m!p!e!r!e!m!e!t!e!r!`	`amperemeter`	Amperemeter
`a!e!p!f!e!l!`	`aepfel`	aepfel
`a!f!f!e!`	`affe`	affe
`A!p!f!e!l!`	`apfel`	Apfel
`a!p!f!e!l!`	`apfel`	apfel
`a"p!f!e!l!`	`apfel`	äpfel
`a!u!s!s!t!a!n!d!`	`ausstand`	ausstand
`b!a!n!a!n!e!`	`banane`	banane
`B!a!n!a!n!e!`	`banane`	Banane
`e!i!n!s!`	`eins`	eins
`L!a!m!b!d!a!-K!a!l!k!u"l!`	`lambda-kalkul`	Lambda-Kalkül
`l!a!m!b!d!a!-K!a!l!k!u"l!`	`lambda-kalkul`	lambda-Kalkül
`l\a!m!b!d!a!-K!a!l!k!u"l!`	`lambda-kalkul`	λ-Kalkül
`l!a!u!n!i!s!c!h!`	`launisch`	launisch
`s!c!h!l!u!s"`	`schlus`	schluß
`s!c!h!l!u!s!s!`	`schluss`	schluss
`z!w!e!i!`	`zwei`	zwei

Tabelle 3: Beispiel zur Nachsortierung

String	:=	Token String | empty
Token	:=	ZeichenToken | LatexToken
ZeichenToken	:=	Grundzeichen Erweiterungszeichen | BelassenesZeichen
LatexToken	:=	Buchstabe '\' AlphaString
AlphaString	:=	AlphaToken AlphaString | empty
AlphaToken	:=	Buchstabe '`!`'
Grundzeichen	:=	Buchstabe | '`/`' | '`:`' | '`@`' | '`[`' | '`‘`' | '`{`'
Buchstabe	:=	'`a`' | ... | '`z`' | '`A`' | ... | '`Z`'
Erweiterungszeichen	:=	'`!`' | ... | '`~`' /* Belegung entsprechend Tabelle */
BelassenesZeichen	:=	„Alle 7-bit ASCII-Zeichen, die nicht Grundzeichen sind“

Die Semantik eines Token, also das Zeichen, welches er bedeutet, ergibt sich aus Tabelle 1 bzw. der LaTeX-Dokumentation.

6 Diskussion

Vor einem Einsatz sollte geklärt werden, ob

- das Sortierproblem nicht anders gelöst werden kann
- dann eine sparsamere Kodierung verwendet werden kann
- überhaupt auf die Behandlung von Sonderzeichen im Datenbestand verzichtet werden soll (d.h. eine Umkodierung von Eingabedaten und Anfragen in ASCII).

Bei einem Einsatz sind folgende Nachteile zu berücksichtigen:

- Die Sortierreihenfolge auf der Kodierung ist nicht korrekt.

- Nachteilig ist der etwa doppelt so hohe Speicheraufwand gegenüber einer 7- oder 8-Bit-Kodierung. Auch bei der Suche müssen etwa doppelt so viele Daten bearbeitet werden (gegenüber einer ASCII-Kodierung).
- Der Tabellenaufwand für die Umkodierung beträgt 60x126 Einträge.
- Ad-Hoc-Anfragen direkt an die Datenbank sind ohne die Unterstützung eines Preprozessors (s.u.) sehr umständlich, Ergebnisse unleserlich.

Der Einsatz erfordert

- eine für den Benutzer transparente Umkodierung, da die Kodierung selbst unhandlich ist
- unterschiedliche Benutzermodi (werden Umlaute verwendet oder nicht, soll tolerant gesucht werden und in welcher Weise, ...).

Die Integration in das System sollte am besten an zentraler Stelle über die Anfragesprache erfolgen. Nur so kann sichergestellt werden, daß alle Textelemente einer Benuzeroberfläche, welche sich auf so kodierte Textfelder im Datenbestand beziehen, von den Anfrage- und Kodierungsmöglichkeiten Gebrauch machen können.

Die unterschiedlichen Modi (Funktionen), nach denen aus einer Benutzereingabe (Zeichen) ein regulärer Ausdruck (Anfrage) generiert wird und aus einer Systemkodierung ein lesbarer String, sind noch zu definieren.

Literatur

[Ach95] Alf Achilles. LitServer: A Mailserver for Bibliographic References in Computer Science, http://liinwww.ira.uka.de/bibliography/litserver/litserver.html, 1995.

[Beh92] Bernd Behr. Welt der Zeichen — Neuer Zeichenstandard der ISO. *c't Magazin für Computertechnik*, 1992(9):241–247, September 1992.

[Bor91] Ute Bormann. ISIS 1: Internationale Standards für Informationstechnik — Systemarchitektur. Skript zur Lehrveranstaltung an der TU-Berlin, 1991.

[DGKM73] Günther Drosdowski, Paul Grebe, Rudolf Köster, and Wolfgang Müller, editors. *DUDEN - Die Rechtschreibung*, volume 1 of *DUDEN — Das Standardwerk zur deutschen Sprache*. Bibliographisches Institut, 1973.

[Fal85] Christos Faloutsos. Access methods for text. *ACM Computing Surveys*, 17(1):49–74, 1985.

[Lam86] Leslie Lamport. *Latex: A Document Preparation System*. Addison-Wesley, 1986.

[Mün92] Monika Münnich. *PC-Katalogisierung mit RAK*. K. G. Sauer Verlag, München, New York, London, Paris, 1992.

[Pat88] Oren Patashnik. Bibtexing. Teil der Latex-Distribution, lokal unter file:/usr/TeX/doc/btxdoc.dvi.gz, February 1988.

[PPF95] Ulrich Pfeiffer, Thomas Poersch, and Norbert Fuhr. Searching proper names in databases. In Rainer Kuhlen and Marc Rittberger, editors, *Hypertext-Information Retrieval-Multimedia— Synergieeffekte elektronischer Informationssysteme, Proceedings HIM'95*, volume 20 of *Schriften zur Informationswissenschaft*, pages 259–275, Konstanz, 1995. Universitätsverlag.

[Rei94] Ulrike Reiner. Anfragesprachen für Textsuchsysteme. In A. Böhm, A. Mengel, and T. Muhr, editors, *Texte verstehen - Konzepte, Werkzeuge, Methoden*, volume 14 of *Schriften zur Informationswissenschaft*, pages 227–256. Universitätsverlag, Konstanz, 1994. Auch erhältlich unter `http://www.cs.tu-berlin.de/~ul/freiall.ps`.