xindy

Ein Flexibles Indexierungssystem

Studienarbeit

Roger Kehr

November 1995

Institut für Theoretische Informatik
FG Systemprogrammierung
Technische Hochschule Darmstadt

Zusammenfassung

Dieser Bericht beschreibt die Ergebnisse einer Studienarbeit, die 1995 am Institut für Theoretische Informatik an der Technischen Hochschule Darmstadt erstellt wurde. Ziel des Projektes war eine theoretische Analyse der Anforderungen an Indexierungssysteme und eine prototypische Implementierung.

Indexierungssysteme verarbeiten die von einem Textsatzsystem erzeugten Indexierungsinformationen, um daraus einen sortierten und mit Ausgabeinformationen versehenen Index zu generieren. Dieser wird in der Regel dann wieder dem Textsatzsystem zugeführt. Gegenstand der Studienarbeit war eine Analyse der Leistungsfähigkeit bestehender Systeme und eine darauf aufbauende Entwicklung eines theoretischen Gesamtmodells der Indexverarbeitung. Besonderen Wert wurde dabei auf die Verarbeitung von Lokationsreferenzen und die Ausgabeformatierung gelegt. Hohe Benutzerkonfigurierbarkeit und Flexibilität waren weitere Ziele bei der Entwicklung. Eine Teilimplementierung der Kernaspekte des Modells wurde vorgenommen, um die wesentlichen Aspekte zu überprüfen.

Danksagung

Ich möchte mich zuerst bei meinen Betreuern Joachim Schrod und Klaus Guntermann für ihre hervorragende Unterstützung bei der Realisierung dieser Studienarbeit bedanken. Des weiteren bedanke ich mich bei Gabor Herr und Christine Detig für Korrekturen, Tips und Hilfestellungen, die sie mir bei vielerlei Problemen gegeben haben. Außerdem bedanke ich mich bei Karin Wolf für ihre Geduld und Hilfe während dieser Arbeit und für ihre Anmerkungen zum vorliegenden Text.

Erwähnt werden sollen auch alle die Mitglieder der Internetgemeinde, die mir durch ihre Arbeiten von Emacs und clisp bis zu noweb und TEX eine optimale Entwicklungsumgebung zur Verfügung gestellt haben.

Inhaltsverzeichnis

1 Vorwort **1**

2 Einführung **3**

3 Grundanalyse und Begriffsdefinitionen **5**
3.1 Stichwort 5
3.2 Lokationsreferenz 6
3.3 Indexklassen 7
3.4 Stichwortverarbeitung 7
3.4.1 Hierarchiestruktur 7
3.4.2 Sortieren und Mischen von Stichwörtern 8

4 Analyse **10**
4.1 Strukturelle Analyse 10
4.1.1 Klassen von Lokationsreferenzen 10
4.1.2 Alphabete 12
4.1.3 Komponententypen 13
4.1.4 Komposition 14
4.1.5 Verträglichkeit 15
4.1.6 Zuordnungsstrategien 16
4.1.7 Kategorien von Lokationsklassen 17
4.2 Sortieren und Mischen von Lokationsreferenzen 18
4.2.1 Totale Ordnung 19
4.2.2 Direkter Nachfolger 19
4.2.3 Bereichsbildung 20
4.2.4 Sortieren und Mischen 21
4.2.4.1 Sortieren und Mischen mit Attributen 21
4.2.4.2 Sortieren und Mischen in einer Lokationsklasse . 24

5 Modellentwurf **26**
5.1 Definitionen 26
5.2 Daten- und Strukturmodell des Index 26
5.2.1 Indexeintrag 26
5.2.2 Lokationsreferenz 27
5.3 Regeln bei der Lokationsverarbeitung 28
5.3.1 Mischregeln 29
5.3.2 Ausgaberegeln 29

5.4 Ausgabeformatierung 30
5.4.1 Ausgabebaum 31
5.4.2 Traversierung des Ausgabebaums 32
5.4.3 Stichwortgruppen 34
5.5 Indexstyle 34

6 Indexverarbeitung **37**
6.1 Indexstyle einlesen 37
6.2 Indexeinträge einlesen und normalisieren 37
6.3 Stichwortmischung und -sortierung 38
6.4 Lokationsreferenzmischung und -sortierung 38
6.5 Ausgabeformatierung 39

7 Implementierung **40**
7.1 Entwicklungsumgebung 40
7.2 Struktur 41
7.3 Portabilität 41
7.4 Aktueller Implementierungsstand 41

8 Zusammenfassung **49**

9 Glossar **51**

Abbildungsverzeichnis

2.1 Datenfluß in einem Indexierungssystem 4

3.1 Index mit Seitennummern aus einem Sachbuch über Algorithmen 5

4.1 Index mit Abschnittsnummern aus einem Sachbuch über Algorithmen 10

5.1 Schematischer Ausgabebaum für die Ausgabeformatierung 31

7.1 Schematisches Datenmodell der wichtigsten Klassen der Implementierung mit *enthält*-Beziehungen 42
7.2 Vererbungsgraphen der wichtigsten Klassen der Implementierung 42
7.3 Modulgraph mit *benutzt*-Relationen 43
7.4 Basisalgorithmus der `mark-object`-Methoden 48

9.1 Index mit Bezeichnung der Komponenten eines Indexeintrags . . 52

Tabellenverzeichnis

4.1 Tabelle mit Komponententypen von Strukturkomponenten 14
4.2 Lokationsreferenzmatrix 17
4.3 Kategorien der Darstellung von Lokationsreferenzen 22
4.4 Beispiele für die Sortierung/Mischung innerhalb einer Lokationsklasse 25

5.1 Tabelle mit Ausgabematrizen für unterschiedliches Markup eines Ausgabebaums 33

Kapitel 1

Vorwort

Sometimes it is desirable to index words that don't actually appear on the page. . . . For example, Appendix I lists page 1 under 'beauty', even though page 1 only contains the word 'beautiful'. (The author felt that it was important to index 'beauty' because he had already indexed 'truth'.)

Donald E. Knuth, *The TEXbook* (1984)

The compiling of an index is interesting work, though some authors are apt to find it tedious and delegate the work to others. The proofreader who undertakes it will find that it is splendid mental exercise and brings out his latent editorial capability.

Albert H. Highton, *Practical Proofreading* (1926)

Glücklicherweise ist die Erstellung eines Stichwortverzeichnisses heutzutage weniger eine manuelle Tätigkeit als noch im Jahre 1926, als Albert H. Highton wohl noch mit Indexkärtchen Stichwörter und Seitennummern aus einem Buch herausschreiben mußte, um sie anschließend zu sortieren und für den Textsatz zusammenzustellen. Heutzutage ist der Computer ein adäquates Werkzeug, um einen Index zu generieren, aber die Tätigkeit des Proofreaders und die Auswahl der Stichwörter durch den Autor sind auch heute noch genauso wichtig wie seinerzeit.

Index als Bestandteil von Literatur

Die Qualität von guten Lehr- und Sachbüchern ist in erster Linie eine Frage der methodischen und didaktischen Vermittlung von Wissen. Weiterhin kann man auf die Funktion eines Buches als Nachschlagewerk besonderen Wert legen. Es sollte möglich sein, durch Nachschlagen nach Stichwörtern und Begriffen schnell an gewünschte Informationen zu gelangen. Lehrbücher sind in der Regel ohne einen gut stukturierten Index nicht als Nachschlagewerk verwendbar. Viele Lehrbücher weisen leider einen sehr unvollständigen Index auf, in welchem wichtige Begriffe fehlen oder die Verweise auf die entsprechenden Seiten im Buch fehlerhaft oder ungenügend sind. Sie enthalten dadurch ein schlecht strukturiertes und unübersichtliches Stichwortverzeichnis und verlieren dadurch entscheidend an Qualität.

Mängel bestehender Systeme

Grund für diese Mängel sind oft nicht ausreichende Systeme zur Erstellung solcher Indexe. Ist es bei maschinell lesbaren Texten meistens möglich, durch entsprechende Suche nach bestimmten Begriffen ansatzweise trotzdem an die gewünschten Textstellen zu gelangen, ist dies prinzipiell bei gedruckter Literatur nicht möglich, und der Leser ist grundsätzlich immer auf das Stichwortverzeichnis angewiesen.

Besondere Bedürfnisse

Weiterhin trifft man insbesondere in den Naturwissenschaften immer häufiger auf spezielle Wünsche der Autoren. Zu diesen Wünschen zählen Indexe mit mehrsprachigen Einträgen, die Verwendung verschiedenartiger Indexverweise auf Seitenummern, Gliederungsebenen, Anhänge, Symbole etc., welche mit herkömmlichen Indexierungssystemen nicht zufriedenstellend erfüllt werden können. Prinzipiell sollte ein Indexierungsprogramm auch in der Lage sein, spezielle Indexinformationen wie HyperText-Links o.ä. zu verwalten. Viele der existierenden Systeme sind in einzelnen Details sehr leistungsfähig, aber ein übergreifendes Konzept fehlt meist und sie sind deshalb nur begrenzt einsetzbar. Beispiele für solche Systeme sind das makeindex-System [CH88] in Verbindung mit dem TEX- [Knu84] bzw. LATEX-System [Lam86] oder moderne interaktive Textverarbeitungssysteme wie *StarWriter* [Sta94] oder *Word für Windows* [Mic94].

Die grundsätzliche Problematik, welche Begriffe und Stichwörter in einen Index aufzunehmen sind, auf welche Stellen im Text verwiesen werden soll, wie Referenzen optisch unterschiedlich hervorzuheben sind usw., ist nicht Gegenstand dieser Arbeit. Diese Aufgabe wird üblicherweise vom Autor eines Buches übernommen, denn sie erfordert Kenntnis von den Inhalten und der Struktur des Textes und soll auch auf Schwerpunkte innerhalb des Buches hinweisen. Weitere Betrachtungen zu diesem Themenaspekt finden sich in [LL93] und [Chi82].

Gegenstand der Studienarbeit

Gegenstand der Arbeit ist eine grundsätzliche Analyse der Erfordernisse an ein Indexierungssystem und die Entwicklung eines möglichst universellen Modells der Indexverarbeitung. Dieses Modell soll den o.a. Ansprüchen in möglichst vielen Bereichen gerecht werden. Neue erweiterte Mechanismen zur Generierung eines Indexes werden vorgestellt und entsprechende Algorithmen eingeführt.

Basierend auf diesem Modell wird eine Implementierung der grundlegenden Verfahren des Modells vorgestellt, die zur Verifizierung der grundsätzlichen Mechanismen in der Praxis dienen sollte.

Kapitel 2

Einführung

Wir werden in dem nun folgenden Abschnitt die grundsätzliche Funktion eines Indexierungssystems und seine Einbettung in den Textsatzprozeß darlegen. In Abschnitt 3 werden wir eine vertiefte Analyse bestehender Systeme durchführen. Eigene Erkenntnisse werden dargelegt und insbesondere die Lokationsreferenzen werden als Schwerpunkt dieser Arbeit in Abschnitt 4 vertieft untersucht. In Abschnitt 5 werden wir die aus der Analyse gewonnenen Informationen in ein Modell der Indexverarbeitung abbilden. Es folgt ein Überblick über die Verarbeitungsvorgänge innerhalb des Modells in Abschnitt 6. In Abschnitt 7 finden sich Informationen über die erfolgte Implementierung des Systems. Den Abschluß bilden die Zusammenfassung mit einem Ausblick auf weiterführende Arbeiten in Abschnitt 8 und ein Glossar in Abschnitt 9.

Datenfluß in einem Indexierungssystem

In Abbildung 2.1 ist der Datenfluß dargestellt, an welchem ein Indexierungssystem beteiligt ist. Das Textsatzsystem liefert die Rohdaten der Indexeinträge in einem festgelegten Format. Das Indexsystem liest zunächst sämtliche für den Verarbeitungsprozeß benötigten Informationen aus der Indexstyle-Datei und anschließend die Rohdaten des Indexes ein. Die Indexeinträge werden daraufhin gemischt und sortiert. Dieser Prozeß besteht aus der Verarbeitung der Stichwörter und den dazugehörigen Verweisen auf das Dokument. Diese Verweise werden mit *Lokationsreferenz* bezeichnet. Die entsprechenden Misch- und Sortierregeln sind im Indexstyle angegeben. Zuletzt wird der komplette Index mit einem Markup versehen und dem Textsatzsystem zur Weiterverarbeitung zugeführt.

Indexstyle

Besondere Wichtigkeit liegt dabei auf der benutzerdefinierbaren Konfiguration des Indexstyles. Möglichst viele Parameter des Systems sollten von Benutzer einstellbar sein, um eine möglichst hohe Anpassungsfähigkeit zu gewährleisten.

Benutzerdefinierbare Konfiguration

Wie man aus diesem Konzept erkennen kann, ist unser System eine Art *Indexprozessor*. Ein Index wird in einem kompletten Durchlauf erzeugt. Eine inkrementelle Einbindung in ein System, welches durch Einfüge- oder Löschoperationen eine dynamische Veränderung des Indexes erzeugt, ist nicht vorgesehen.

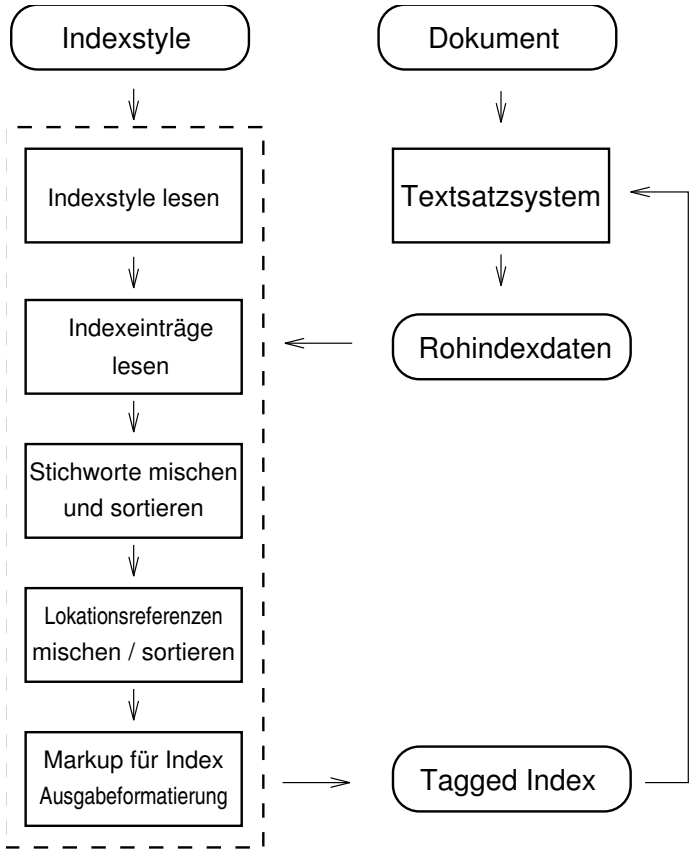


Abbildung 2.1: Datenfluß in einem Indexierungssystem

Dies ist normalerweise auch nicht erforderlich, da ein Index nie direkt bearbeitet werden sollte, sondern immer aus einem Generierungsprozeß mit vohergehendem generischen Markup der Indexbegriffe hervorgehen sollte.

Auch typische interaktive Textverarbeitungssysteme wie *Word für Windows* [Mic94] oder *StarWriter* [Sta94] verwenden diese Methode, indem zuerst Begriffe im Dokument für den Index speziell markiert werden und das System später auf expliziten Befehl hin Indexe erstellt.

Kapitel 3

Grundanalyse und Begriffsdefinitionen

In diesem Abschnitt werden die grundsätzlichen Aspekte und Verfahren der Indexverarbeitung zusammengefaßt und die Funktionsweisen verschiedener Systeme untersucht. Um einen leichteren Einstieg in die generelle Problematik der Indexverarbeitung zu erreichen, nehmen wir als Beispiel einen Index, anhand dessen wir die prinzipiellen Grundgedanken vermitteln wollen.

3.1 Stichwort

Hierarchische Struktur

Stichwort

Vertikale Anordnung

Wir untersuchen nun anhand eines Beispielindexes die Grundstruktur von Indexen. Schauen wir uns Abbildung 3.1 an, so können wir allgemein sagen, daß ein Index eine hierarchisch gegliederte, vertikal angeordnete und sortierte Struktur aus Stichwörtern und Unterstichwörtern mit jeweils dazugehörigen Referenzen wie Seiten- und Abschnittsnummern ist. Unter einem *Stichwort* verstehen wir einen hierarchisch untergliederten Begriff wie z.B. 'Bäume, natürliche'. Die Struktur von Stichwörtern ergibt sich aus der *vertikalen* Anordnung, in welche

Bäume
    AVL, **23**, 22–25, 38
    natürliche, **20**, 18–21
Fibonacci Queues, *siehe* Priority Queues
Suche
    binäre, **5**, 4–7, 11
    sequentielle, **7**, 6–8, 10
    geordnete, 6, *siehe auch* ungeordnete Suche
    ungeordnete, 7
Priority Queues
    Fibonacci, **35–36**, 41*f.*

Abbildung 3.1: Index mit Seitennummern aus einem Sachbuch über Algorithmen

die Unterstichpunkte gemäß vorgegebener Sortierreihenfolge eingeordnet werden. Im folgenden verwenden wir die Notation

$$(\ \langle Ebene_1\rangle\ :\ \langle Ebene_2\rangle\ :\ \ldots\ :\ \langle Ebene_n\rangle\ )\ ,$$

um die hierarchische Untergliederung von Stichwörtern zu beschreiben. In dieser Notation werden Ebenen durch Doppelpunkte getrennt, und ihre oberste Gliederungsstufe wird zuerst angeführt. Die einzelnen Hierarchieebenen eines Stichworts bestehen aus Zeichenketten. Wir werden in Abschnitt 5.1 noch auf die genaue Definition einer Zeichenkette eingehen.

3.2 Lokationsreferenz

Lokationsreferenz

Zu jedem Stichwort gehört eine Menge von *Lokationsreferenzen*. Diese Referenzen liegen in einer *horizontalen* Anordnung vor. Im Beispiel besitzt das Stichwort (Bäume:natürliche) die Referenzmenge {**20**, 18–21}. Die optische Hervorhebung der Seitennummer '20' wird oft benutzt, um auf besonders wichtige Stellen im Text hinzuweisen. Im folgenden wird der Begriff *Referenz* nahezu synonym zu *Lokationsreferenz* und *Lokation* verwendet, da wir im weiteren nicht zwischen der Lokation selbst (also einer bestimmten Seite oder einem Abschnitt in einem Dokument) und der Referenz auf diese Lokation (z.B. aus dem Stichwortverzeichnis heraus) im einzelnen unterscheiden müssen.[1]

Referenz

Lokation

Lokationsreferenzen wie '*siehe...*' oder '*siehe auch...*' bilden einen speziellen Typ von Referenzen. Diesen Referenztyp bezeichnen wir mit *Verweisreferenz*. Eine Verweisreferenz besteht aus einem Typ (*siehe*, *siehe unter*, *siehe auch* etc.) und dem verwiesenen Stichwort, welches wir mit *Verweisstichwort* bezeichnen. Wir notieren Verweisreferenzen in der Form

Verweisreferenz

Verweisstichwort

$$(\langle Verweisart\rangle \rightarrow \langle Verweisstichwort\rangle).$$

Im Beispiel ist also bei (*siehe* → Priority Queues) '*siehe*' der Verweisreferenztyp und 'Priority Queues' das zugeordnete Verweisstichwort.

Die Referenz '41*f.*' aus unserem Beispiel bildet noch einen weiteren Referenztyp. Hierbei handelt es sich um eine Lokation, der noch ein zusätzliches Attribut zugeordnet ist. Die Schreibweise '⟨*Lokation*⟩*f.*' bezeichnet in einem Index üblicherweise einen Verweis auf die angegebene und die ihr folgenden Seiten bzw. Lokationen. Die beigefügte Markierung '*f.*' stellt damit eine zusätzliche Eigenschaft einer Referenz dar. Da es sich um ein der Lokation zugeordnetes Attribut handelt, bezeichnen wir diesen Lokationstyp im folgenden als *Attributierte Lokationsreferenz*. Wir werden später noch weitere solcher Attribute einführen. Zunächst jedoch definieren wir auch hier eine eigene Notation der Form

Attributierte Lokationsreferenz

$$(\langle Lokationsreferenz\rangle\,;\langle Attribut\rangle)$$

für diese Referenzart.

Üblicherweise faßt man aufeinander folgende Lokationsreferenzen zu einem

Bereiche

[1] Trotzdem benutzen wir den Begriff *Lokation* eher dann, wenn die reale Lokation — also die Seite oder der Abschnitt — gemeint ist.

Bereich zusammen. Solche Bereiche findet man meistens in der Form $\langle Lok_1 \rangle$–$\langle Lok_2 \rangle$. Ein Bereich stellt damit die Abkürzung einer Liste von Lokationsreferenzen dar. In unserem Beispiel hat das Stichwort (Bäume:natürliche) den Bereich 18–21. Wir verwenden im folgenden die Notation

$$[\mathrm{Lok}_1{:}\mathrm{Lok}_2] \ ,$$

um einen Bereich zu notieren.

3.3 Indexklassen

Nach den Betrachtungen zu Stichwörtern und Lokationsreferenzen wenden wir uns dem Thema *Indexklassen* zu. Umfangreichere wissenschaftliche Werke besitzen heute neben einem Stichwortverzeichnis häufig noch Indexe für Autoren, Symbole etc. In [Ker92] werden beispielsweise insgesamt vier verschiedene Indexe unter den Oberbegriffen *Mythologische Namen*, *Geographische Namen*, *Wörter* und *Sachen* geführt. Separate Indexe für Befehle und Kommandos sind in vielen technischen Handbüchern üblich. Diese verschiedenen Indexe sind darüber hinaus noch nach Bedarf unterschiedlich im Layout, können aber prinzipiell aus einem einzigen Ursprungs-Datenstrom von Indexeinträgen stammen. Unser Modell sollte also die Verarbeitung von verschiedenen Indexklassen mit unterschiedlichen Eigenschaften wie Lokationsklassen, Ausgabeformatierung usw. grundsätzlich ermöglichen. Jeder Indexeintrag muß also Informationen über seine Klassenzugehörigkeit enthalten. Eine gleichzeitige Zuordnung zu mehreren Klassen ist möglich. Beispielsweise findet sich zu verschiedenen thematisch gegliederten Indexen noch ein Gesamt- oder Master-Index, in dem nochmals alle Begriffe der Teilindexe zusammengefaßt sind.

Indexklassen

3.4 Stichwortverarbeitung

Wie wir bereits in Abschnitt 2 erwähnt haben, führt jedes Stichwort eine Liste von Lokationsreferenzen mit sich. Wir haben also als Oberordnung eines Indexes die hierarchische Ordnung der Stichwörter, die im folgenden beschrieben wird. Die Ordnung der Lokationsreferenzen wird in Abschnitt 4 behandelt.

Dadurch ist der Indexierungsprozeß grundsätzlich in die beiden Aufgaben *Stichwortverarbeitung* und *Lokationsreferenzverarbeitung* zerlegbar. Beide Arbeitsprozesse sind getrennt voneinander durchführbar. Wir analysieren zunächst die Mischung und Sortierung von Stichworthierarchien.

3.4.1 Hierarchiestruktur

An unserem Beispiel in Abbildung 3.1 ist die hierarchische Struktur der Stichwörter offensichtlich zutage getreten. Zwar gibt es verschiedene Beispiele für Indexe, welche keine Untergliederung aufweisen, aber solche *flachen* Indexe sind eher die Ausnahme als die Regel. Es ist aber nicht immer so, daß die Untergliederung auch in einer entsprechenden optischen Form sichtbar wird, wie in unserem Beispiel. So definiert das *Chicago Manual of Style* [Chi82] u.a. den Stil

flush-and-hang style, welcher unserem Beispiel entspricht, und *run-in indended style*, welcher Substrukturen nicht explizit optisch hervorhebt, um dadurch eine Platzersparnis zu erreichen. Weitere Arten von unterschiedlichen Ausgabestilen wie z.B. in [BGB91] sind darüber hinaus denkbar.

Ausgabestile von Indexeinträgen

Prinzipiell ändert sich aber nur die Art der Ausgabe des fertigen Indexes. Dieser Vorgang wird üblicherweise als *Ausgabeformatierung* bezeichnet und meint das Anheften von Formatierungsinformationen an den generierten Index, aus welchem dann das Textsatz- oder Informationssystem ein formatiertes Dokument erzeugt.

3.4.2 Sortieren und Mischen von Stichwörtern

Als grundsätzliches Problem stellt sich nun die Frage nach der Sortierung der Stichwörter. In [Chi82] und [LL93] wird auf die Alphabete eines Indexes und damit auf die Ordnungsrelationen zwischen Stichwörtern eingegangen. Die Sortierungsfrage wird dort durch das Beispiel von englischen bzw. amerikanischen Sortierregeln aufgeworfen. Als Beispiel wird das Stichwort 'St. Louis' nach amerikanischen Sprachregeln so sortiert, als ob es ausgeschrieben würde. Es taucht also im Index an der Position von 'Saint Louis' auf. Es werden weitere Beispiele für die Einsortierung von Namen in einen Index aufgeführt, wobei man als Beispiel entsprechende Sortierregeln für das Spanische und Vietnamesische analysiert. Grundsätzlich läßt sich also festhalten, daß die Sortierung der Stichwörter von Sprachen und deren orthographischen Regeln abhängt.

nationale Sortierregeln

Im makeindex-System wird das Problem teilweise gelöst, indem zum Sortierschlüssel *sort key* ein weiterer Druckschlüssel *actual key* eingeführt wird, welcher beim der abschließenden Ausgabeformatierung den Sortierschlüssel substituiert. Dieses Verfahren umgeht die 'St. Louis'-Problematik, indem man in einem solchen Fall die Substitution explizit angibt, ist aber prinzipiell sehr unbequem zu handhaben. Man bedenke, daß in der Regel viele spezielle nationale Zeichensätze mit Umlauten nicht innerhalb des lateinischen Alphabets sortierbar sind. Nach deutschen Orthographieregeln wird der Buchstabe 'ä' je nach Fall wie 'ae' oder auch wie 'a' einsortiert. Es gibt also noch nicht einmal einheitliche Standardregeln, nach denen ein Indexierungssystem sortieren könnte. Für alle diese Stichwörter immer separat den Druckschlüssel angeben zu müssen, ist für den Benutzer eine unakzeptable Lösung.

Druckschlüssel

nationale Alphabete

Im *International MakeIndex* [Sch91] wird die Problematik durch ein regelbasiertes Substitutionsschema angegangen. Innerhalb dieses Substitutionsschemas können Sortierregeln für beliebige Alphabete angegeben werden. Dort kann der Benutzer Regeln definieren, um in einem deutschen Text den Buchstaben 'ä' vor der Sortierung z.B. auf die Buchstaben 'ae' abzubilden. Diese Regeln werden verwendet, um aus dem Stichwort einen Sortierschlüssel zu generieren, welcher implizit zur Sortierung verwendet wird. Die Abbildung des Stichworts in den Sortierschüssel wird als *sort mapping* bezeichnet. Darüber hinaus ist dieses Konzept allgemein genug, daß es auch auf die Sortierung von Zahlen, Symbolen, Formeln etc. angewendet werden kann. Sollten diese Regeln in Einzelfällen nicht genügen, so kann immer noch auf die explizite Angabe des Sortierschlüssels ausgewichen werden.

sort mapping

Durch die Einführung der Sortierungsabbildung treten weitere grundsätzliche Probleme auf, die durch die zusätzliche Einführung einer Mischabbildung (*merge mapping*) gelöst werden konnten. Sie generiert auf analoge Weise einen Mischschlüssel, der entscheidet, welche Einträge als „gleich“ angesehen werden. Man erreicht mit dieser Abbildung eine *Normalisierung* des Mischschlüssels, um von einer konkreten Repräsentation eines Schlüssels zu einer abstrakten Repräsentation zu gelangen. Eine ausführliche technische Beschreibung der beiden Abbildungsverfahren findet sich in [SH91].

merge mapping

Normalisierung

Eine Möglichkeit, den Index zu erweitern, besteht in der Hinzunahme von *Permutierten Indexschlüsseln*. Darunter versteht man das automatische Generieren eines Stichwortes (binäre:Suche) aus dem Stichwort (Suche:binäre) durch Wortpermutationen. Diese sind in [BK88] implementiert, gelingen jedoch teilweise lediglich deshalb, weil die englische Sprache eine Permutation z.B. von ‘*sorting a book index; a book index, sorting; book index, sorting a; index, sorting a book*’ erlaubt, dies jedoch nicht ohne weiteres auf die deutsche oder andere Sprachen übertragbar ist.

Permutierter Index

Zusammenfassung

Wir haben also anhand unseres Beispiels festgestellt, daß ein Index eine hierarchische Struktur besitzt. Wir haben eine vertikale Struktur von Stichwörtern und eine horizontale Struktur von Lokationsreferenzen erkannt und grundsätzlich dargelegt. Des weiteren haben wir verschiedene Lokationstypen ausgemacht. Diese Typen werden im späteren Verlauf als Lokationsklassen weiterbehandelt werden. Lokationsreferenzen können bestimmte Attribute zugeordnet werden. Wir unterscheiden attributierte Lokationsreferenzen und Verweisreferenzen.

Kapitel 4

Analyse

Nach der Grundanalyse der Stichwörter werden wir nun eine vertiefte Analyse der Lokationsreferenzen vornehmen. Sie bilden das Kernstück der Studienarbeit und wir werden insbesondere ihre Struktur und mögliche Operationen darauf präsentieren. Wir beginnen mit einer strukturellen Analyse und wenden uns später dem Sortieren und Mischen von Lokationsreferenzen zu.

4.1 Strukturelle Analyse

4.1.1 Klassen von Lokationsreferenzen

Zur struktuellen Analyse von Lokationsreferenzen greifen wir unseren Index aus Abbildung 3.1 auf und erweitern ihn mit neuen Lokationsreferenzen, anhand derer wir den Lokationsbegriff neu klassifizieren. Um den Blick auf das Wesentliche zu lenken, sind die Seitennummern des vorigen Indexes größtenteils entfernt worden. Der so veränderte Index ist in Abbildung 4.1 dargestellt.

Wie man sehen kann, sind in diesen Index Abschnittsnummern neu hinzugekommen. Abschnittsnummern (hier serifenlos gedruckt) bilden eine weitere Möglichkeit auf Lokationen innerhalb eines Dokuments zu verweisen. Die Abschnittsgliederung ‘2.1’ ist eine hierarchisch in Kapitel und Abschnitt zerlegte

Hierarchienstruktur

Bäume
    AVL, 2.3
    natürliche, A-1, 2.1
Suche
    binäre, 11, 11a, 1.3
    sequentielle, A-2, 1.2
    ungeordnete, 1.2.2
Priority Queues
    Fibonacci, 3.3

Abbildung 4.1: Index mit Abschnittsnummern aus einem Sachbuch über Algorithmen

Lokation mit dem Trennzeichen '.'. Die Referenz 'A-1' ist gleichfalls eine mehrstufige Referenz. Sie bildet sich aus dem Buchstaben 'A' (sie soll auf den Anhang des Buches verweisen) und der Nummer des Abschnitts (in diesem Fall '1'). Dieses Beispiel demonstriert die Unterschiedlichkeit von Lokationstypen. Eine Lokation bezeichnet eine bestimmte Dokumentstelle. Ein Dokument kann von verschiedenen Lokationsstrukturen durchzogen sein. Beispielsweise läßt sich ein Dokument nach Seiten einteilen, was eine *externe* — also nicht dem Dokument innewohnende — Strukturierung darstellt. Im Gegensatz dazu bilden Gliederungen eine *interne*, das Dokument betreffende Strukturierung. Auf welche Weise nun auf ein Dokument verwiesen werden soll, ist insbesondere eine Frage, ob auf die interne oder externe Struktur oder beides verwiesen werden soll.

Unterschiedliche Lokationstypen

externe

interne Dokumentstruktur

Die Verschiedenheit von Lokationen wird in den untersuchten Systemen [CH87, Sta94, Mic94, BK88, LL93] nicht beachtet. Diese Systeme sind nur in der Lage, Seitennummern und Verweisreferenzen zu verwalten. Zu diesem Aspekt finden sich weitere untersuchenswerte Lokationstypen.

Beispielsweise werden im *System Management Guide* [IBM93] die Seitennummern kapitelweise gezählt, so daß sich die Seitennummern aus einer zweistufigen Nummer der Form '⟨*Kapitel*⟩–⟨*Kapitelseite*⟩' wie z.B. '7–13' bilden.

Des weiteren ist die Referenz auf Seite '11a' des Stichworts `(Suche:binäre)` auch eine hierarchisch strukturierte Lokationsreferenz. Häufig werden Manuals durch Hinzunahme von Seiten wie '11a' erweitert, um bei nachträglich eingefügten Seiten die ursprüngiche Seitenfolge nicht zu stören. Dabei sollen bei verschiedenen Versionen eines Manuals möglichst alle Numerierungen gleichbleibend sein.

Im juristischen Umfeld gibt es häufig Bücher, die nach Gesetzbüchern und Paragraphen strukturiert sind [BGB91]. Dort wird nach den Stichwörtern die Nummer des Gesetzbuches angegeben und anschließend werden die zugehörigen Paragraphen aufgezählt. Als Beispiel soll der folgende Auszug dienen.

> **Lasten** beim Kauf **1** §436, §446; auf der Mietsache **1** §546; ... [1]

Grundsätzlich können wir also nach diesen Betrachtungen sagen, daß eine Lokationsreferenz eine hierarchische Struktur aufweist. Die Struktur besteht aus einzelnen Hierarchieebenen wie Kapitelnummer, Abschnittsnummer usw. oder zusätzlichen Aufzählungen wie '11', '11a'. Drücken wir diese Hierarchiestruktur auch durch eine spezielle Notation aus, so müssen wir beachten, daß hierbei noch Trennzeichen wie Bindestriche oder Punkte zwischen den Hierarchien auftreten können. Wir definieren also die Notation

Lokationsstruktur

$$(\![Ebene_1]Trenner_1[Ebene_2]Trenner_2[\ldots]Trenner_{n-1}[Ebene_n]\!) \ ,$$

um eine Hierarchie von Lokationen zu beschreiben. In dieser Notation wird die Lokation '1.2.2' als `([1].[2].[2])` geschrieben. Im folgenden bezeichnen wir diesen Typ von Lokationsreferenz mit *Strukturreferenz*. Eine Strukturreferenz besteht aus allen Hierarchieebenen inklusive der Trennzeichen. Des weiteren bezeichnen wir die Ebenen einer Strukturreferenz mit *Strukturebenen*.

Strukturreferenz

Strukturebene

[1] Fette Nummern bezeichnen hier die Nummer des Gesetzbuches, während die normalen Nummern auf die zugehörigen Paragraphen hinweisen.

4.1.2 Alphabete

Wir haben im vorangehenden Abschnitt die Struktur der Lokationsreferenzen dargelegt und wollen uns jetzt mit der Zuordnung von Lokationsreferenzen zu Lokationsklassen befassen. Da unser System — wie man aus dem Datenflußdiagramm in Abbildung 2.1 erkennen kann — nichts über die Interna des Textsatzsystems weiß, muß unser System in der Lage sein, aus Zeichenfolgen mittels der Angaben im Indexstyle die Lokationsreferenzen zu erkennen. Wir bezeichnen das der Indexverarbeitung zugrundeliegende Alphabet von Zeichen als das *Dokumentalphabet*. Dieses Alphabet ist der gemeinsame Zeichenvorrat des Satzsystems und des Indexsystems.[2] Die Rohindexdaten und damit auch die Lokationsreferenzen liegen also im Format dieses Dokumentalphabets vor, und das Indexsystem muß daraus die Lokationsreferenzen in ihre Struktur zerlegen. Wir unterscheiden die einzelnen Strukturkomponenten einer Lokationsreferenz in zwei Kategorien:

Dokumentalphabet

1. *Aufzählbare Typen* wie z.B. Seiten-, Abschnitts- oder Kapitelnummern.
2. *Endliche Typen* wie z.B. alphabetische Aufzählungen von Abschnitten. So enthält die Lokationsreferenz ([Anhang]–[A]) in der zweiten Strukturkomponente das Alphabet der Großbuchstaben.

Wir definieren zusammenfassend:

Definition 4.1.1 *Ein* Dokumentalphabet *ist eine endliche Menge von Zeichen.*

Definition 4.1.2 *Ein* Symbol *ist ein Wort über einem Dokumentalphabet.*

Definition 4.1.3 *Eine* Aufzählung $\mathcal{I}$ *ist eine unendliche linear geordnete Menge von Symbolen. Insbesondere existiert eine bijektive Abbildung* $\psi : \mathcal{I} \rightarrow \mathbb{Z}$ *des Alphabets auf die Menge* $\mathbb{Z}$ *der Ganzen Zahlen oder die Menge* $\mathbb{N}$ *der Natürlichen Zahlen.*

Definition 4.1.4 *Ein* Alphabet $\mathcal{A}$ *ist eine endliche Aufzählung von Symbolen. Insbesondere existiert eine bijektive Abbildung* $\phi : \mathcal{A} \rightarrow \mathbb{N}$ *des Alphabets auf ein endliches Intervall der Natürlichen Zahlen.*

Wir folgern daraus:

Lemma 4.1.1 *Die Strukturkomponenten einer Lokationsreferenz bestehen aus Symbolen.*

Betrachten wir als Beispiel Seitennummern im Dokumentalphabet $\mathcal{D}$ vom Typ ISO-Latin, so ist die Seitennummer '13' ein Wort des Dokumentalphabets und damit nach Definition 4.1.2 ein Symbol. Die Seitennummern bilden sich nun aus einer Untermenge $\mathcal{D}_{sn} = \{\text{'0'}, \ldots, \text{'9'}\}$[3] des Dokumentalphabets $\mathcal{D}$. Die Menge aller Worte über $\mathcal{D}_{sn}$ bilden nach Definition 4.1.3 eine Aufzählung. Wir definieren:

[2] Üblicherweise wird dort das ASCII-Alphabet, die ISO-Latin–Familie oder Unicode [Uni91] Verwendung finden.

[3] also die der Ziffern des arabischen Zahlensystems

DEFINITION 4.1.5 *Ein* Sortierungsalphabet $\mathcal{S}$ *ist eine Aufzählung oder ein Alphabet. Die Symbole des Sortierungsalphabets sind Wörter über einer Untermenge* $\mathcal{D}_\mathcal{S} \subseteq \mathcal{D}$ *des Dokumentalphabets.*

Wir bringen mit dieser Definition zum Ausdruck, daß ein Sortierungsalphabet sowohl ein endliches Alphabet (wie die Menge der Großbuchstaben) als auch eine unendliche Aufzählung (wie im Falle der Seitennummern) sein kann.

DEFINITION 4.1.6 *Auf jedem Sortierungsalphabet* $\mathcal{S}$ *ist gemäß der Definitionen 4.1.3 und 4.1.4 eine bijektive Abbildung* $\sigma_\mathcal{S} : \mathcal{S} \rightarrow \mathbb{Z}$ *definiert. Diese Abbildung stellt eine* linere Ordnung $<_{alph}$ *bezüglich paarweiser Elemente* $s_1, s_2 \in \mathcal{S}$ *auf dem Sortierungsalphabet dar. Die Ordnungsrelation* $s_1 <_{alph} s_2$ *gilt genau dann, wenn* $\sigma_\mathcal{S}(s_1) < \sigma_\mathcal{S}(s_2)$.

DEFINITION 4.1.7 *Ein* Basisalphabet $\mathcal{D}_\mathcal{S}$ *eines Sortierungsalphabets* $\mathcal{S}$ *ist die kleinste Untermenge des Dokumentalphabets* $\mathcal{D}$*, aus der die Worte des Sortierungsalphabets* $\mathcal{S}$ *vollständig gebildet werden können.*

Betrachten wir nun folgendes Beispiel: Die Strukturreferenz ([*Großbuchstabe*]–[*Abschnittsnummer*]) mit dem Dokumentalphabet $\mathcal{D}$ hat auf der ersten Strukturkomponente das Sortierungsalphabet $\mathcal{S}^1$ mit den Symbolen $\mathcal{S}^1_1$ = 'A', $\mathcal{S}^1_2$ = 'B', ..., $\mathcal{S}^1_{26}$ = 'Z'. Daraus ergibt sich das Basisalphabet $\mathcal{D}_{\mathcal{S}^1} = \{\text{'A'}, \ldots, \text{'Z'}\}$. Auf der zweiten Ebene haben wir das Sortierungsalphabet $\mathcal{S}^2 = \{\text{'1'}, \text{'2'}, \ldots, \text{'9'}, \text{'10'}, \ldots\}$ mit dem Basisalphabet $\mathcal{D}_{\mathcal{S}^2} = \{\text{'0'}, \ldots, \text{'9'}\}$.
Mit diesen Definitionen sind wir nun für weitere Untersuchungen gerüstet.

4.1.3 Komponententypen

Komponententypen

Wir betrachten nun diverse *Komponententypen* von Strukturkomponenten. Ein Komponententyp ist ein Tupel $B = (\mathcal{S}, \mathcal{B})$ mit $\mathcal{S}$ Symbolalphabet und $\mathcal{B}$ Basisalphabet. Tabelle 4.1 zeigt uns einen Teil der üblicherweise verwendeten Komponententypen. Sie sind nach Aufzählungen und Alphabeten geordnet. Um die Komponententypen mit einfachen Bezeichnungen zu identifizieren, benennen wir sie auf geeignete Weise.

Grundsätzlich muß für jeden Basishierarchietyp eine Ordnung definiert sein, damit man die Elemente des Typs in einer numerischen Form sortieren kann. Aus den Definitionen 4.1.3 und 4.1.4 geht bereits hervor, daß entsprechende Abbildungen von der Symbolmenge auf eine Zahlenmenge definiert sein müssen. Für arabische und römische Ziffernfolgen ist diese Abbildung bereits durch das üblicherweise benutzte Zahlensystem gegeben.[4] Die hier aufgeführten Komponententypen decken bereits einen Großteil aller Erfordernisse ab. Trotzdem müssen wir zulassen, daß besondere Alphabete, wie das griechische Alphabet oder die numerische Indexierung, prinzipiell benutzerdefinierbar und frei verwendbar sind, auch wenn deren Verwendung nur in Spezialfällen nötig sein wird.

[4] Man beachte, daß die o.a. „Zahlen" zunächt einmal nur Zeichenfolgen (also Worte über dem Dokumentalphabet) darstellen, deren Abbildung in das Zahlensystem zuerst definiert werden muß.

Aufzählungen			
Komponententyp	*Symbolalphabet*	*Basisalphabet*	*Name*
numerisch	0, 1,. . . , 10,. . .	{0 . . . 9}	[num]
römisch (klein)	i, ii,. . . ,x, . . .	{i,v,x,l,c,d,m}	[roman]
römisch (groß)	I, II,. . . ,X,. . .	{I,V,X,L,C,D,M}	[ROMAN]
Indexaufzählung	$x_1, x_2, \ldots, x_9, \ldots$	$\{x, 0, \ldots, 9\}$	

Alphabete			
Komponententyp	*Symbolalphabet*	*Basisalphabet*	*Name*
alphabetisch (klein)	a, b, c,. . . , z	{a,. . . ,z}	[alpha]
alphabetisch (groß)	A, B, C,. . . , Z	{A,. . . ,Z}	[ALPHA]
griechisch (klein)	$\alpha, \beta, \gamma, \delta, \ldots, \omega$	$\{\alpha, \ldots, \omega\}$	
—	Kapitel, Anhang	{a,A,e,g,h,i,K,l,n,p,t}	

Tabelle 4.1: Tabelle mit Komponententypen von Strukturkomponenten

Wesentlich häufiger dagegen ist die Verwendung von festgelegten Worten wie 'Kapitel', 'Anhang', 'Glossar' usw. als Symbolalphabet, welche in unserer Hierarchieebene ebenfalls eine Ebene belegen können.[5] Hierbei muß dann ebenfalls eine geeignete Ordnungsrelation definiert werden. Dies könnte durch eine einfache Aufzählung der Symbole geschehen wie in der letzten Zeile der Tabelle dargestellt. Dort sind die Worte 'Kapitel' und 'Anhang' Symbole des Symbolalphabets, denen aufgrund der Aufzählungsreihenfolge eine Ordnungsrelation zugewiesen wurde.

4.1.4 Komposition

Aus den Komponententypen bilden wir nun durch *Komposition* die Lokationsklassen. Mit ihnen beschreiben wir dem System die Struktur der vorkommenden Lokationsreferenzen und können die Erkennung und Zuordnung von Zeichenströmen zu Lokationsreferenzen formal beschreiben. *Lokationsklassen*

Betrachten wir erneut die in der Praxis verwendeten Lokationstypen, so können wir grundsätzlich die folgenden beiden Klassen unterscheiden:

1. *Standardklassen*, deren Struktur immer gleich ist, wie z.B. Seitennummern der Form ([num]). *Standardklassen*

2. *Varklassen*, deren Struktur sich variabel aus mehreren Strukturkomponenten zusammensetzt. Dazu gehören Gliederungsnummern der Form ([num].[num].[num]. . .). Beispielsweise gehören die Lokationsreferenzen ([2]), ([2].[1]) und ([2].[1].[4]) der gleichen logischen Gliederungsstruktur an. Man kann hier auch argumentieren, daß jede der vorkommenden *Varklassen*

[5] Insbesondere kann man auch Trennzeichen als Alphabete ansehen, die nur aus einem einzigen Symbol bestehen.

Strukturen eine eigene Lokationsklasse bildet. Prinzipiell bildet eine Varklasse jedoch eine Menge von Lokationsklassen, denen wir in den folgenden Abschnitten noch weitere gemeinsame Eigenschaften zuordnen können.

Um dem System adäquate Beschreibungen zu liefern, müssen wir ihm für jede Klassendefinition deren Typ — Standard- oder Varklasse — mitteilen, da wir für beide Klassen unterschiedliche Zuordnungsstrategien anwenden müssen.

Wir werden im folgenden die Standard-Lokationsklassen analog zur Schreibweise von Lokationsreferenzen in der Form `([num].[num])` notieren, und die Varklassen durch das Anfügen eines '`*`' wie in `([num].[num])*`. Bei Varklassen wird durch die Strukturdefinition auch gleichzeitig die maximale Tiefe festgelegt.

4.1.5 Verträglichkeit

Um Strukturkomponenten zu Hierarchien zu verknüpfen, muß die Komposition von Komponententypen zu einer kompletten Hierarchie inklusive aller Trennzeichen speziell untersucht werden. Insbesondere muß ein System in der Lage sein, einem Zeichenstrom die richtige Lokationsklasse zuzuordnen. Die folgenden Beispiele sollen in die Problemstellung einführen:

Zuordnungsproblem

1. `([num].[num])`

   definiert eine Lokationsklasse, deren erste und zweite Hierarchiestufe durch arabische Zahlen repräsentiert werden. Gültige Repräsentanten dieser Klasse sind beispielsweise die folgenden, aufsteigend sortierten Referenzen 1.1, 1.2, 1.3, ..., 2.0, 2.1, ..., 2.20, ...

   Die Sortierung erfolgt demnach streng von der obersten Hierarchiestufe bis zur untersten. Insbesondere ist das Trennzeichen '**.**' unbedingt erforderlich, da sonst nicht zwischen den Zählebenen unterschieden werden kann (12 kann sein „eins zwei“ oder „zwölf“).

2. `([num][roman])`

   definiert eine Aufzählungsform, die nicht unbedingt ein Trennzeichen verlangt, da das Basisalphabet von `[num]` und das der römischen Ziffern `[roman]` disjunkt und damit eindeutig unterscheidbar sind. Beispiele für solche Klassenvertreter sind 1i, 1ii, 1iii, 1iv, ..., 2i, ...

3. `([roman][alpha])`

   In diesem Fall ist das Basisalphabet des Komponententyps `[roman]` echte Untermenge des Komponententyps `[alpha]`, da alle römischen Ziffern ebenfalls Elemente des lateinischen Alphabets sind. Im Fall der Zeichenkette 'ivi' kann man sowohl auf die Lokationsreferenz `([iv][i])` als auch `([i][vi])` schließen, wobei letztere genau genommen einen Fehler darstellen würde. Man kann demnach keine exakte Hierarchieebene ermitteln, wenn keine eindeutigen Trennzeichen verwendet werden und die Zeichenmengen nicht disjunkt sind.

Aus diesen Überlegungen können wir also folgern:

LEMMA 4.1.2 *Zwei Sortierungsalphabete $\mathcal{A}$ und $\mathcal{B}$ sind grundsätzlich miteinander* verträglich, *wenn für die zugehörigen Basisalphabete $\mathcal{D}_A$ und $\mathcal{D}_B$ gilt: $\mathcal{D}_A \cap \mathcal{D}_B = \emptyset$.*

Verwenden wir geeignete Trennzeichen, die mit den Basisalphabeten der Komponententypen disjunkt sind, so können wir das Verträglichkeitsproblem explizit umgehen. Alle Kompositionen von Komponententypen, welche nicht verträglich sind, müssen durch implizite Konventionen oder spezielle eindeutige Festlegungen spezifiziert werden. *Verträglichkeit*

4.1.6 Zuordnungsstrategien

Um einen Zeichenstrom anhand der Beschreibungen der Komponententypen und Lokationsklassen genau einer Lokationsklasse zuzuordnen, müssen wir geeignete Zuordnungsstrategien entwerfen.

Standard-Lokationsklassen sind duch ihre gleichbleibende Struktur definiert. Wir verlangen daher, daß ein Zeichenstrom auf alle Komponententypen und Trennzeichen einer Standardklasse passen muß, um akzeptiert zu werden. Wir nennen diese Erkennungsstrategie *Exaktes Zuordnen*. *Exaktes Zuordnen*

Varklassen definieren eine Menge von Lokationsklassen und die Zuordnungsstrategie muß in der Lage sein, alle möglichen Teilklassen für die Zuordnung heranzuziehen. Wir müssen daher auch eine Teilüberdeckung zulassen. Da eine Überdeckung nur streng von links nach rechts erfolgen kann, sprechen wir im folgenden von einer *Präfixüberdeckung*. Demnach lassen wir hier eine Zuordnung auch dann zu, wenn ein Zeichenstrom nur auf den Präfix einer Varklasse paßt. Wir nennen diese Zuordnungsstrategie *Präfixzuordnung*. *Präfixzuordnung*

Kombinieren wir beide Lokationsklassentypen derart, daß wir Lokationsklassen in der allgemeinen Gestalt

$$\langle \textit{Standard-Hierarchien} \rangle \langle \textit{Var-Hierarchien} \rangle$$

ausdrücken, so können wir noch präzisere Erkennungsstrategien für bestimmte Lokationsklassen gewinnen. Jeweils einer der beiden Hierarchieanteile ist dabei optional. Verwenden wir die Zuordnungsstrategie Exaktes Zuordnen für den Standard-Anteil und die Präfixzuordnung für den Var-Anteil, so erhalten wir noch einen dritten, kombinierten Lokationsklassentyp, der aus einer Mischung von Standard- und Varklassen besteht.

Für die in der Praxis vorkommenden Lokationsreferenzen reicht die einfache Beschreibungsform der separaten Standard- und Varklassen jedoch grundsätzlich aus und wir werden sie nicht weiter in die Untersuchungen mit einbeziehen.

Zuordnung zu Lokationsklassen

Verwenden wir Lokationsklassen, die nicht miteinander verträglich sind, so müssen wir heuristische Verfahren untersuchen, um trotzdem eine Zuordnung nach bestimmten Kriterien zuzulassen. Wir betrachten nun Tabelle 4.2, die Lokationsklassen zu der folgenden Referenzmenge enthält

1, 2, 1a, 1b, 1c, 1i, 1ii, 1iii, 1iv.

Beispiel 1		
Lokationsklasse	*eindeutig*	*mehrdeutig*
`([num][alpha])*`	1a, 1b, 2a	1, 2, 1c, 1i
`([num][roman])*`	1ii, 1iii, 1iv	1, 2, 1c, 1i

Beispiel 2		
Lokationsklasse	*eindeutig*	*mehrdeutig*
`([num])`	1, 2, 3	
`([num][alpha])*`	1a, 1b, 2a	1c, 1i
`([num][roman])*`	1ii, 1iii, 1iv	1c, 1i

Tabelle 4.2: Lokationsreferenzmatrix

Die Spalte *eindeutig* bezeichnet Referenzen, welche aufgrund ihrer Struktur als Präfix einer einzigen Lokationsklasse zugeordnet werden können oder sie auf eine einzige Varklasse passen. In Beispiel 2 führt dies zu der Zuordnung der Seitenummern 1, 2 und 3 zur Lokationsklasse `([num])`. Die Referenzen der Spalte *mehrdeutig* können nicht eindeutig einer einzigen Klasse zugeordnet werden.

Da der Leser die korrekte Zuordnung ebenso verstehen muß, treten solche Fälle in der Praxis normalerweise nicht auf. Sie sollten deshalb von Autoren auch nicht zur Dokumentstrukturierung verwendet werden. Man muß in diesem Fall davon ausgehen, daß die Angabe der Lokationsklassen unvollständig ist und der Benutzer korrigierend eingreifen sollte. Eine Korrektur kann darin bestehen, bestimmte Klassen noch mit geeigneten Hilfs-Präfixen zu versehen. Für die Lokationsklassen `([num][num])` und `([num][alpha])` könnte man letztere z.B. in `([Seite]:[num][alpha])` abwandeln und bei der Ausgabeformatierung mit geeigneten Markups versehen, die den Hilfspräfix unterdrücken.

4.1.7 Kategorien von Lokationsklassen

Es ist üblich, Lokationsreferenzen, auf die besonders hingewiesen werden soll, durch eine optische Darstellung hervorzuheben, welche die Bedeutung ihres Inhaltswerts unterstreicht. Man möchte Lokationen z.B. mit den Attributen *Definition*, *Verwendung* oder *wichtige Verwendung* versehen, um deren Bedeutung zu charakterisieren. Diese Attribute können dann zum Satzzeitpunkt vom Textsatzsystem mit beliebigen Darstellungsformen wie z.B. *normal*, *fett* und *kursiv* verknüpft werden. Weiterhin sollen solche unterschiedlich attributierten Lokationsreferenzen unterschiedlich in die Lokationsliste einsortiert werden. Wir ordnen daher einer Lokationsreferenz ein Attribut zu, welches für verschiedenartige Kategorisierungszwecke verwendet werden kann. Wir verwenden hierfür den Begriff *Kategorieattribut*.

Kategorieattribut

In den folgenden Beispielen werden wir diesem Attribut Werte der Form `bold` oder `italic` zuordnen. Es soll helfen, die Lokationsreferenzen gleichen Kategorieattributs sofort erkennen zu können. Trotzdem müssen wir beachten, daß

die semantische Zuordnung eines Attributswertes nicht vom Indexierungssystem geleistet wird, sondern dies erst zur Markup-Zeit vom Textsatzsystem vorgenommen wird und dort ein Attribut `bold` mit anderen Markup-Informationen versehen werden kann, die z.B. eine kursive Darstellung bewirken könnten.

Ergebnisse der strukturellen Analyse

Die strukturelle Analyse von Lokationsreferenzen hat uns zu folgenden Ergebnissen geführt:

- Es gibt verschiedene Arten von Lokationsreferenzen, die wir als Lokationsklassen bezeichnen. Eine Lokationsklasse enspricht dabei einem bestimmten Lokationstyp des Dokuments wie z.B. Seite, Kapitel, Abschnitt, Unterabschnitt.
- Strukturreferenzen werden durch Kompositionen von Komponententypen gebildet. Die verwendeten Komponententypen sind u.U. nicht miteinander verträglich, was besondere Zuordnungsstrategien erforderlich macht. Wir müssen einen Zeichenstrom einer Lokationsklasse zuordnen können und daraus eine konkrete Lokationsreferenz bilden.
- Wir haben Kategorieattribute charakterisiert, so daß wir allgemein die Klasse der *attributierten Strukturreferenzen* definieren können.

Nach der Analyse der grundlegenden Strukturen kommen wir nun zu den Prozessen der Indexverarbeitung.

4.2 Sortieren und Mischen von Lokationsreferenzen

Der nun folgende Abschnitt stellt eines der Kernthemen der Arbeit dar. Aufgrund der Komplexität der Betrachtungen ist es sinnvoll, schrittweise in das Thema einzuführen und die einzelnen Verfahren und Überlegungen zunächst getrennt zu erklären, um sie dann später im Modellentwurf zusammenzuführen.

Das Problem des Sortierens und Mischens von Lokationsreferenzen beruht auf den folgenden Anforderungen an ein Indexierungssystem:

1. Wir möchten die zu einem Stichwort gehörenden Lokationsreferenzen in einer — für den Leser — geordneten und sortierten Darstellung ausgeben.
2. Das System sollte in der Lage sein, aufeinanderfolgende Lokationsreferenzen auf Wunsch zu einem Bereich zusammenzufassen.

Diese hier nur unpräzise formulierten Ziele führen uns zu den zugrunde liegenden Problemen dieses Verarbeitungsprozesses. Sie bestehen aus den Begriffen der *Totalen Ordung* und des *Nachfolgers*.

4.2.1 Totale Ordnung

Betrachten wir die folgenden unsortierten Lokationsreferenzen

1.1, 1.2, Anhang–B, 1.3, Anhang–A, 1e, 1c,

so müssen wir in der Lage sein, ihnen eine Sortierreihenfolge zuzuordnen. Intuitiv ordnen wir die Referenzen gruppenweise an, wobei sich eine Gruppe direkt aus einer Lokationsklasse bildet. In unserem Beispiel sind die Lokationsklassen (`[num].[num]`), (`[Anhang]–[Alpha]`) und (`[num][alpha]`) gegeben. Mit dieser Reihenfolge gelangen wir beispielsweise zur sortierten Ausgabefolge

1.1, 1.2, 1.3, Anhang–A, Anhang–B, 1c, 1e.

Zuerst bestimmen wir also die Reihenfolge der Lokationsklassen. Die *Totale Ordnung* innerhalb einer Klasse ist bereits implizit durch die Klassenstruktur definiert. Wir benutzen deshalb die Definition der linearen Ordnung auf einem Symbolalphabet in Definition 4.1.6 für die Definition der Totalen Ordnung auf einer Lokationsklasse und formulieren analog:

Implizite Totale Ordnung

Definition 4.2.1 *Eine* Totale Ordnung *auf einer Lokationsklasse L mit der Struktur $S = ([s_1] \dots [s_n])$ mit s_k Sortieralphabet ist für paarweise Lokationsreferenzen $l^1, l^2 \in L$ mit der Struktur $([l_1] \dots [l_n])$ definiert als eine Relation $<_{loc}$ für die gilt:*

$$l^1 <_{loc} l^2 \;\equiv\; \exists i : \; (l^1_j = l^2_j \;\; \forall j : 1, \dots, i-1) \quad \wedge \quad (l^1_i <_{alph} l^2_i).$$

Diese Definition besagt, daß die Strukturen zweier Lokationsreferenzen bis zu einer bestimmten Strukturkomponente $i-1$ gleich sind und die Strukturkomponente i beide Referenzen unterscheidet. Die Totale Ordnung der Strukturkomponente i bestimmt dann die Totale Ordnung der Lokationsreferenzen. Wir verwenden das Relationssymbol $=_{loc}$, um Gleichheit zweier Lokationsreferenzen zu notieren.

Wir definieren also alle Elemente der Unterebenen einer Strukturkomponente als zwischen die einzelnen Symbole des Sortieralphabetes *eingereiht*. Dieses ist die implizit festgelegte Ordnung auf Lokationsklassen.

Implizite Einreihung

4.2.2 Direkter Nachfolger

Bei der Bereichsbildung von Lokationsreferenzen genügt nicht bloßes Wissen über die Totale Ordnung, sondern wir müssen genaue Kenntnis davon haben, ob zwei Lokationsreferenzen direkt aufeinander folgen.

Bei der Einführung von Alphabeten haben wir bijektive Abbildungen verwendet, aus denen wir nun auch in bestimmten Fällen den Nachfolger einer Lokationsreferenz bestimmen können. Ein Alphabet der Länge n kann in jedem Fall mit einer Abbildung ϕ bijektiv auf ein Intervall $[1 \dots n] \in \mathbb{N}$ abgebildet werden, wobei der Nachfolger eines Symbols s durch $\phi^{-1}(\phi(s)+1)$ ermittelt werden kann. Gleiches gilt für ein Sortierungsalphabet in Form einer Aufzählung.

Allgemein stellen wir fest, daß wir den Nachfolger anhand geeigneter bijektiver Abbildungen direkt aus der Totalen Ordnung des Sortierungsalphabets ermitteln können, wenn wir Lokationsreferenzen gleicher Klasse betrachten. Wir

definieren dann als den Nachfolger einer Lokationsreferenz l^1 mit der letzten Strukturkomponente l^1_k genau die Referenz l^2, die sich nur in der letzten Strukturkomponente von l^1 unterscheidet und dort genau den Nachfolger l^1_{k+1} im Symbolalphabet von l^1_k enthält. Wir werden darauf in Abschnitt 4.2.4.2 noch zurückkommen.

Das Wissen über Nachfolger ist zum Teil direkt im Dokument selbst verankert. Nehmen wir als Beispiel den Fall von eingefügten Seiten der Varklasse `([num][alpha])*`, wie sie oft in Handbüchern nach Änderungen vorkommen, so wissen wir nicht, ob auf die Seite 11 nun die Seite 12 oder die Seite 11a folgt. Dieses ist *Dokumentwissen*, welches wir unserem System explizit mitteilen müssen. Andernfalls könnte es zu der Bereichsbildung 11–12 kommen, obwohl Seite 11a existiert und die entsprechende Lokation dort nicht markiert worden ist. Die Bereichsbildung wäre demnach ungültig. *Dokumentwissen*

Wir definieren zusätzlich zum impliziten Wissen über die Nachfolgerfunktion die Operation $succ(l_1, l_2)$, welche explizit festlegt, daß l_2 direkter Nachfolger von l_1 ist, wenn gilt: $l_1 <_{loc} l_2$. Analog definieren wir $not\text{-}succ(l_1, l_2)$, um explizit die Nachfolgerfunktion außer Kraft zu setzten. Sie hat Vorrang vor allen impliziten und expliziten Nachfolgerfestlegungen. Des weiteren definieren wir das Prädikat $succ\text{-}p(l_1, l_2)$, welches wahr ist, wenn auf implizite oder explizite Weise $succ(l_1, l_2)$ und nicht $not\text{-}succ(l_1, l_2)$ definiert wurde. *succ()* *succ-p()*

Wir bezeichnen im folgenden für eine Lokation l das zugehörige Kategorieattribut mit $catattr(l)$ und die zugehörige Strukturreferenz mit $strucref(l)$.

4.2.3 Bereichsbildung

Nach der Definition des direkten Nachfolgers wollen wir noch kurz eine formale Beschreibung der *Bereichsbildung* hinzufügen.

Definition 4.2.2 *Seien $l_1, \ldots, l_n$ Lokationsreferenzen gleicher Lokationsklasse mit $succ\text{-}p(l_i) = l_{i+1}$ für $i = 0, \ldots, n-1$; $catattr(l_i)$=$catattr(l_j)$, $\forall i, j = 0, \ldots, n$; so ist $[l_1{:}l_n]$ ein* Bereich.

Wir sagen, daß l_1 die *untere* oder *linke* Bereichsgrenze ist, und l_n die *obere* oder *rechte* Bereichsgrenze ist.

Definition 4.2.3 *Ein Prädikat joinable-p(), ob zwei Bereiche zu einem Bereich zusammengefaßt werden können, ist folgendermaßen definiert: Seien $L = [l_l{:}l_r]$, $K = [k_l{:}k_r]$ Bereiche mit gleichem Kategorieattribut, so gilt:*

$$joinable\text{-}p(L,K) = \begin{cases} wahr & falls \begin{cases} l_l \leq_{loc} k_l \leq_{loc} l_r \ \vee \\ l_l \leq_{loc} k_r \leq_{loc} l_r \ \vee \\ k_l \leq_{loc} l_l \leq_{loc} k_r \ \vee \\ k_l \leq_{loc} l_r \leq_{loc} k_r \ \vee \\ succ\text{-}p(l_r) = k_l \ \vee \\ succ\text{-}p(k_r) = l_l \end{cases} \\ falsch & \text{sonst.} \end{cases}$$

Eine Funktion join-range() für die Bereichsbildung ist folgendermaßen definiert:

$$\textit{join-range}(L,K) = \begin{cases} [min(l_l,k_l) : max(l_r,k_r)] & \textit{falls joinable-p}(L,K) = \textit{wahr} \\ \bot & \text{sonst.} \end{cases}$$

Wir lassen bei der Bereichsbildung nur Lokationsreferenzen mit gleichem Kategorieattribut zu. Würden wir die Bereichsgrenzen mit unterschiedlichen Attributen wie z.B. '**14**–17' kennzeichnen, so ist nicht einsichtig klar, welche semantische Bedeutung daraus folgt. Gleiches gilt für die Beschränkung auf Lokationsreferenzen der gleichen Lokationsklasse. Ein Bereich der Gestalt 'Kapitel-2 – 1.5.3' würde den Leser eher verwirren als eine Suchhilfe darstellen.[6]

4.2.4 Sortieren und Mischen

Nach den theoretischen Grundlagen müssen wir uns nun den konkreten Problemen der Sortierung und Mischung von Lokationsreferenzen zuwenden. Folgende Probleme sind zu lösen:

1. Wie wirken sich Kategorieattribute auf die Totale Ordnung und die Definition des Nachfolgers aus?

2. Wie erweitern wir das Modell der Totalen Ordnung über die Grenzen von Klassenstrukturen hinweg und lassen benutzerdefinierte Änderungen zu?

Wir beginnen zuerst mit der Untersuchung der Sortier- und Mischvorgänge innerhalb von Standardklassen. Anschließend erläutern wir die Modifikationsmöglichkeiten bezüglich der Nachfolgerbehandlung von Varklassen.

4.2.4.1 Sortieren und Mischen mit Attributen

Kategorieattribute sollen eine verschiedenartige Darstellung von Lokationsreferenzen bewirken. Wir müssen also Regeln definieren, die das Sortieren und Mischen von Lokationsreferenzen gleichen Lokationswertes aber mit unterschiedlichem Kategorieattribut steuern, da bei einer Ausgabe eine Entscheidung für einen der Kandidaten erfolgen muß.

In Tabelle 4.3 sind verschiedene Darstellungsformen einer Lokationsreferenzliste kategorisiert worden. Wir unterscheiden prinzipiell 8 verschiedene Kategorien, die durch die Markierungen in den jeweiligen Spalten unterschieden werden.

Folgende Darstellungskategorien sind unterscheidbar:

separate-sorting
: Wir separieren in der Ausgabe alle Kategorieattribute. Es werden nach einer fest definierten Ausgabereihenfolge bzgl. der Kategorieattribute die jeweiligen Lokationsreferenzen ausgegeben.

  Wir notieren eine solche Reihenfolge durch eine Aufzählung der Kategorieattibute in Listenform wie z.B.: `(default bold italic...)`

[6] Dies hat sich bereits bei den Diskussionen um dieses Thema gezeigt.

Nr.	*Typ*	*Bereiche*				*Lokationsreferenzen*
	Sep oder Mix	ohne	einfach	zugeordnet	unterdrückt	11 13 14 15 17 25 **12 15 25**
1	Sep	○				11 13 14 (15) 17 (25) **12 15 25**
2	Sep		○			11 13–15 17 (25) **12 15 25**
3	Sep			○		11–15 17 (25) **12 15 25**
4	Sep				○	11–15 17 (25) **25**
5	Mix	○				11 **12** 13 14 **15** 17 **25**
6	Mix		○			11 **12** 13–15 **15** 17 **25**
7	Mix			○		11–15 **12 15** 17 **25**
8	Mix				○	11–15 17 **25**

Tabelle 4.3: Kategorien der Darstellung von Lokationsreferenzen

Die Kategorien 1–4 der Tabelle entsprechen der Sortierform `(default bold)`.

Für die Totale Ordnung $<$ zweier Lokationsreferenzen x, y und der Menge der Kategorieattribute $C = \{c_1, \ldots, c_n\}$ soll dann gelten:

$$\begin{array}{r} \forall x, y : catattr(x) = c_i \\ \wedge\ catattr(y) = c_j \end{array} \quad \Rightarrow \quad \begin{cases} x < y & \text{falls } i < j \\ y < x & \text{falls } j < i \\ x < y & \text{falls } i = j \ \wedge\ x <_{loc} y \\ y < x & \text{falls } i = j \ \wedge\ y <_{loc} x \\ x = y & \text{falls } i = j \ \wedge\ y =_{loc} x. \end{cases}$$

mixed-sorting

Wir fassen alle Vertreter bestimmter Kategorieattribute zu einem *virtuellen Kategorieattribut* zusammen und definieren innerhalb dieses virtuellen Attributs eine gemischte Anordnung der unterschiedlichen Attribute. Ein solches virtuelles Attribut kann selbst wieder in einer separate-sorting-Aufzählung enthalten sein.

virtuelle Kategorieattribute

Für die Totale Ordnung der Lokationsreferenzen eines virtuellen Attributes bedeutet dies, daß

$$\begin{array}{r} \forall x, y : catattr(x) \neq catattr(y) \\ \wedge\ strucref(x) = strucref(y) \end{array} \quad \Rightarrow \quad x =_{loc} y.$$

Des weiteren definieren wir innerhalb einer solchen Aufzählung eine Rangordnung, die die Wichtigkeit der Attribute festlegt. Wir notieren solche virtuellen Attribute, indem wir in der Aufzählung der Ausgabereihenfolge die entsprechenden Attribute einklammern.

Beispiel: `( (bold default) italic )`

> Innerhalb des virtuellen Attributs `(bold default)` (siehe Kategorien 5–8) definieren wir eine Abnahme des Vorrangs, die sich in der semantischen Bedeutung des *Verdrängens* widerspiegelt. In unserem Beispiel wird in Kategorie 5 die Lokationsreferenz 15 durch die Referenz **15** ersetzt, wenn die Alternative eine Ausgabe beider Lokationsreferenzen ist.

Wir fordern zur Einschränkung, daß gemischte Attribute nicht weiter mit anderen Attributen zusammengefaßt werden dürfen. Des weiteren darf ein Kategorieattribut nicht mehrfach in einer solchen Reihenfolge auftreten.

Ohne Bereichsbildung (Kategorien 1 und 5) erhalten wir eine listenförmige Ausgabe. In allen Fällen können die in Klammern dargestellten Lokationsreferenzen auf Wunsch auch unterdrückt werden, da sie doppelt in der Referenzliste auftreten. Dazu besteht eine weitere Wahlmöglichkeit, die wir mit substitute-if-double bezeichnen wollen. In Kategorie 5 ersetzt '**15**' die Referenz '15' durch die implizite Vorrangregelung des mixed-sorting.

Mit einer *einfachen* Bereichsbildung werden aufeinanderfolgende Lokationsreferenzen eines Kategorieattributs zu einem Bereich zusammengefaßt (Kategorien 2 und 6).

Mit der *zugeordneten* Bereichsbildung (Kategorien 3 und 7) werden die Lokationsreferenzen mit Kategorieattribut `bold` bei der Bereichsbildung von `default`-Lokationen mitverwendet. Solches Verhalten wird erreicht, indem die Definition des Nachfolgers dahingehend manipuliert wird, so daß gilt:

$$\begin{array}{rcl} \forall\, l_1, l_2 \text{ Lokationsreferenzen mit} \\ \textit{succ-p}((l_1, \texttt{default}), (l_2, \texttt{default})) \end{array} \quad \Rightarrow \quad \begin{array}{l} 1.\ \textit{succ}((l_1, \texttt{default}), (l_2, \texttt{bold})) \\ 2.\ \textit{succ}((l_1, \texttt{bold}), (l_2, \texttt{default})). \end{array}$$

Lokationsreferenzen gleichen Inhalts aber mit unterschiedlichen Kategorieattributen werden also in der Nachfolgerbehandlung als gleich angesehen. Für die Verarbeitungstechnik lassen wir zu, daß Attribute neben dem *Primärattribut*, welches der Lokationsreferenz initial zugeordnet wurde, noch weitere *sekundäre Attribute* zugeordnet werden können. Wir erzeugen dann z.B. aus der Lokationsreferenz `(`**`15`**`;bold)` eine Referenz `(15;default,{bold})` mit einem Sekundärattribut `bold`. Die Totale Ordnung bleibt unverändert. Die Nachfolgerbestimmung wird dann auf die Menge der Sekundärattribute erweitert. Bei der Ausgabe müssen diese Lokationsreferenzen einfach ignoriert werden, falls sie nicht in einen Bereich aufgenommen werden konnten,

Primärattribut

Sekundärattribute

In den Kategorien 4 und 8, die durch die Spalte *unterdrückt* gekennzeichnet sind, werden alle Lokationsreferenzen, von denen Kopien — auf die eben beschreibene Weise — mit Sekundärattributen erfolgreich gemischt werden konnten, bei der Ausgabe in der Kategorie des Primärattributs unterdrückt. Man erkennt dies am Vergleich der Kategorien 7 und 8. In Kategorie 8 sind die Lokationsreferenzen **12** und **15** weggefallen, weil die virtuellen Referenzen `(12;default,{bold})` und `(15;default,{bold})` in den Bereich 11–15 eingemischt werden konnten.

Nach dieser grundsätzlichen Klassifizierung werden wir im Modellentwurf in Abschnitt 5.3 Regeln angeben, mit denen man die hier vorgestellten Ausgabeformen steuern kann.

4.2.4.2 Sortieren und Mischen in einer Lokationsklasse

Nach der Darstellung von attributierten Lokationsreferenzen wollen wir nun untersuchen, wie das Sortieren und Mischen über mehrere Strukturkomponenten innerhalb einer Lokationsklasse hinweg funktioniert. Als Beispiel betrachten wir die Lokationen

1, 2, 3, 2.1, 2.2, 2.3, 2.2.1 der Varklasse `([num].[num].[num])*` .

Wir haben nun folgende Möglichkeiten zur Auswahl:

1. Wir fassen auf bestimmten Hierarchiebenen Bereiche zusammen:
   1–3, 2.1–2.3, 2.2.1 oder 1–3, 2.1, 2.2, 2.2.1, 2.3 .
2. Wir lassen bestimmte Unterabschnitte zugunsten Zusammenfassungen auf höherer Ebene wegfallen:
   1–3, 2.1–2.3 oder nur 1–3 .
3. Wir fassen nicht alles zu Bereichen zusammen und variieren die expliziten Aufzählungen:
   1, 2, 2.1–2.3, 3
   1, 2, 2.1–2.3, 2.2.1, 3
   1, 2, 2.1, 2.2, 2.2.1, 2.3, 3 .

Da u.U. alle der aufgeführten Beispiele durchaus sinnvoll sein können, müssen wir Regeln und Verfahren angeben können, wie die entsprechenden Sortierungen und Zusammenfassungen durchgeführt werden sollen. Wir haben also eine weitere Komplexitätsstufe erreicht, da wir prinzipiell beachten müssen, daß wir es insgesamt sowohl mit verschiedenen Hierarchieebenen als auch verschiedenen Attributen zu tun haben.

Im theoretischen Modell verändern wir hierbei die implizite Definition der Totalen Ordnung auf einer Lokationsklasse und die Nachfolger-Vereinbarungen. Um die gewünschten Ausgabeverfahren zu erhalten, müssen wir den in Abschnitt 4.2.2 eingeführten Begriff des Nachfolgers jetzt genauer definieren. Allgemein läßt sich sagen:

LEMMA 4.2.1 *Für eine Lokationsklasse C mit den Lokationsreferenzen $L = (l_1 \dots l_n)$, $K = (k_1 \dots k_n) \in C$ der Strukturtiefe n; l_i, k_i Strukturkomponenten gilt: succ-p(L, K) falls succ-p(l_n, k_n) und $\forall i = 1 \dots n-1 : l_i = k_i$.*

Wir definieren also für Lokationsklassen nur eine implizite Nachfolgerfunktion für Elemente der untersten Ebene.[7] Dies entspricht der intuitiven Vorstellung eines Nachfolgers, denn wir können ohne Dokumentwissen nichts über Nachfolger auf den oberen Ebenen aussagen. Wir können nun bestimmte Ebenen einer Lokationsklasse zusätzlich explizit mit der Nachfolgerfunktion belegen (Fall 1). Des weiteren benötigen wir eine Möglichkeit, bei einer erfolgreichen Zusammenfassung einer höheren Ebene niedrigere Ebenen zu überdecken (Fälle 2 und 3).

In Tabelle 4.4 haben wir die obigen Beispiele in ein Schema eingeordnet, wobei in jeder Ebene die Eigenschaft der Nachfolgerbildung (NF) und die Überdeckung von niedrigeren Ebenen (2 bzw. 3) eingetragen wurde. Ein Eintrag der

[7] Also $succ(3.1.2, 3.1.3)$; aber nicht $succ(3.1, 3.2)$.

Spezifikationen				*Beispiele*
Ebene 1		Ebene 2		
NF				1–3, 2.1, 2.2, 2.2.1, 2.3
NF		NF		1–3, 2.1–2.3, 2.2.1
NF	3	NF	3	1–3, 2.1–2.3
NF	2			1–3
		NF		1, 2, 2.1–2.3, 2.2.1, 3
		NF	3	1, 2, 2.1–2.3, 3

Tabelle 4.4: Beispiele für die Sortierung/Mischung innerhalb einer Lokationsklasse

Form (NF 3) bedeutet, daß die Ebene 3 bei einer Zusammenfassung auf der entsprechenden Ebene unterdrückt werden soll. Mit dieser Matrix läßt sich das gewünschte Verhalten einfach spezifizieren.

Zusammenfassung

Die geführten Überlegungen haben die grundsätzliche Komplexität der Lokationsverarbeitung aufgezeigt. Sinnvolle Konventionen mußten festgelegt werden. Die Festlegungen, wie im Falle von unverträglichen Lokationsklassen verfahren werden soll, stellen normalerweise eine Ausnahmesituation dar. Die Ambiguität der Zuordnung von Lokationsreferenzen zu Lokationsklassen bringt auch den Leser eines Indexes in Verständnisprobleme, die bei einer gut strukturierten Dokumentgliederung nicht auftreten sollten. Es ging hier um die Definition eines sinnvollen Standardverhaltens in diesem Ausnahmefall. Die hier eingeführte Lokationsmatrix bietet eine einfachere Sichtweise des Zuordnungsproblems.

Die Klassifizierung von Kategorieattributen mit Hilfe von virtuellen Attributen liefert uns ein einfaches und sinnvolles Modell für die (Ein-)Sortierung dieser Attribute.

Die Darstellungsformen der Lokationsliste wurden zunächst aus der Frage nach möglichen Wünschen entwickelt. Nach der Erkennung der tieferliegenden Problematik der Totalen Ordung und des Nachfolgerbegriffs konnte so eine umfassende Analyse gewonnen werden, die ein einfaches und überschaubares Modell der Lokationsverarbeitung liefert.

Die hier aufgezeigten Analysen werden in die Modellbildung münden, die in Abschnitt 5.3 durchgeführt wird.

Kapitel 5

Modellentwurf

Nach der Analyse bestehender Systeme, ihrer Vor- und Nachteile und eigenen Überlegungen kommen wir nun zur Beschreibung des Indexmodells. Nach der Definition der grundlegenden Begriffe, auf denen das Modell aufbauen soll, wird das eigentliche Datenmodell definiert, welches aus den o.a. Überlegungen herausgearbeitet wurde. Im Anschluß an das Datenmodell folgen die Mechanismen der Sortier- und Mischvorgänge eines Indexes. Hierbei wird im wesentlichen nur auf die neuen Details eingegangen. Auf die Vorarbeiten anderer Autoren wird zu gegebener Zeit noch speziell verwiesen.

5.1 Definitionen

Ein **Zeichen** oder **Character** ist die kleinste vom Indexsystem verarbeitbare Einheit. Zeichen werden aus Dateien oder Datenströmen gelesen und vom System weiterverarbeitet. Die Anzahl der zur Verfügung stehenden Zeichen ist endlich und fest. Auf der Menge der zur Verfügung stehenden Zeichen ist eine mathematische Relation $ord()$ definiert für die gilt, daß zwei verschiedene Zeichen nie die gleiche Ordnung besitzen dürfen, also gilt: $\forall x, y \,.\, x \neq y \rightarrow ord(x) \neq ord(y)$

Zeichenketten oder **Strings** sind Listen von Zeichen. Jeder String hat eine definierte Länge, welche sich aus der Anzahl der Zeichen ergibt, aus denen er besteht. Ein Zeichen und ein String der Länge 1 sind unterschiedlich.

5.2 Daten- und Strukturmodell des Index

Definition 5.2.1 *Ein* ***Index*** *i ist eine Liste von Indexeinträgen.*

5.2.1 Indexeintrag

Definition 5.2.2 *Ein* Indexeintrag *ist ein Tupel*

$$idxent = (\ k\_key,\ p\_key,\ m\_key,\ s\_key,\ locref_{lst},\ idxcls_{set}\ )$$

und besteht aus den Komponenten

k_key : **Indexschlüssel**, *Liste von Strings*

p_key : **Druckschlüssel**, *Liste von Strings*
m_key : **Mischschlüssel**, *Liste von Strings*
s_key : **Sortierungsschlüssel**, *Liste von Strings*
$locref_{lst}$: *Menge der* ***Lokationsreferenzen***
$idxcls_{set}$: *Menge der* ***Indexklassen***

Weiterhin definieren wir:

Sei e ein Indexeintrag, so soll gelten: e.p_key $\equiv$ *p_key,*
e.p_key $\equiv$ *p_key, e.m_key* $\equiv$ *m_key, e.s_key* $\equiv$ *s_key sowie*
$e.locref_{lst} \equiv locref_{lst}$ *und* $e.idxcls_{set} \equiv idxcls_{set}$.

Diese Abkürzungsform soll im weiteren auch für alle anderen Referenzierungen auf Komponenten von Tupeln verwendet werden.

Definition 5.2.3 *Eine* ***Lokationsmenge*** $locref_{lst}$ *ist eine Menge bzw. Liste von Lokationen.*

Definition 5.2.4 *Eine* ***Indexklasse*** *idxcls ist ein String.*

5.2.2 Lokationsreferenz

Definition 5.2.5 *Eine* ***Lokationsreferenz*** *ist ein Tupel*

$$locref = (\ strucref,\ catattr,\ loccls,\ refattr\ ) .$$

Definition 5.2.6 *Eine* ***Strukturreferenz*** *strucref ist eine Liste* $layer_{set}$ *von Hierarchieebenen.*

Definition 5.2.7 *Eine* ***Hierarchieebene*** *layer ist ein Tripel*

$$layer = (\ laystr,\ sepstr,\ ordnum\ )$$

mit den Komponenten

laystr : **Ebene**, *String*
sepstr : **Trennzeichen**, *String*
ordnum : **Ordnungszahl**, *ganze Zahl*

Definition 5.2.8 *Ein* ***Referenzattribut*** *refattr ist ein Paar*

$$refattr = (\ ratype\ ,\ refarg_{lst}\ )$$

mit den Komponenten

ratype : *Ein* ***Referenzattribut-Typ*** *ist ein String.*
$refarg_{lst}$: *Die* ***Referenzattribut-Argumente*** *sind eine Liste von Strings.*

Beispiel: *refattr* = ('*Crossreference*' , '(Suche:binäre)')

Definition 5.2.9 *Ein* ***Kategorieattribut*** *catattr ist ein String.*

Definition 5.2.10 *Eine* ***Lokationsklasse*** *loccls ist ein String, mit dem auf eine Lokationsklasse verwiesen wird. Für genauere Details sei hier auf die Implementierung verwiesen.*

5.3 Regeln bei der Lokationsverarbeitung

In Abschnitt 4.2 haben wir uns mit dem Mischen und Sortieren von Lokationsreferenzen beschäftigt. Diese Überlegungen müssen nun in ein geeignetes Modell für die Lokationsverarbeitung einfließen. Mit Hilfe der Lokationsreferenzmatrix in Tabelle 4.2 haben wir bereits ein Unterscheidungsverfahren für die verschiedenen Lokationsreferenzen eingeführt, auf welchem wir im folgenden aufbauen werden.

Sortier- und Mischregeln

Wir definieren nun einen Satz von Regeln, die dem Benutzer des Systems eine möglichst vielfältige Gestaltung des Sortier- und Mischvorgangs gestatten soll.

DEFINITION 5.3.1 *Eine* ***Lokationsverarbeitungsregel*** *ist ein Tupel*

$$locrefrule^{is} = (\ rulename,\ catattr_{set},\ loccls,\ hier_{set},\ tgtattr,\ tgthier_{set},\ rulearg\ )$$

mit den Komponenten

$rulename$	:	***Regelname***.
$catattr_{set}$	:	***Kategorieattribute***.
$loccls$	:	***Lokationsklasse***.
$hier_{set}$	:	*Menge der* ***Hierarchiebereiche***.
$tgtattr$	:	***Zielattribut***.
$tgthier_{set}$	:	*Menge der* ***Zielhierarchiebereiche***.
$rulearg_{set}$	:	*Menge der* ***Zusatzargumente***.

Je nach Regeltyp kann es sein, daß einige der Tupelelemente undefiniert bleiben. Für die zu definierenden Regeln definieren wir folgende Schreibweisen:

1. Kategorieattribute werden durch ihre Bezeichner wie `default` oder `bold` angegeben.
2. Lokationsklassen werden durch ihren Klassennamen angegeben.
3. Hierarchiemengen werden durch die Auflistung von Hierarchienummern oder Hierarchiebereichen wie z.B. `(1-2 4)`, `(3-)` oder `(-4)` definiert. Bereiche dürfen dabei nach beiden Richtungen geöffnet sein.

Wir sprechen im Zusammenhang von Hierarchieebenen auch von *höherer* bzw. *tieferer* Ebene und meinen damit Ebenen mit kleinerer bzw. größerer Ebenennummer, wobei Ebene 1 die höchste Ebene ist.

> *Für alle in den folgenden Abschnitten aufgeführten Beispiele gehen wir von der Sortierreihenfolge*
>
> `( (bold default) italic )`
>
> *aus, die die Kategorieattribute* `bold` *und* `default` *zu einem virtuellen Attribut zusammenfaßt. Man beachte auch den höheren Vorrang des Attributs* `bold` *gegenüber* `default`*. Als Lokationsklassen betrachten wir die Klasse* `section` *der Struktur* `([num].[num])` *und* `manpage` *der Struktur* `([num].[alpha])*`.

Die in Klammern angeführten Lokationsreferenzen klassifizieren die einzelnen Attribute welche zunächst nur interne Ergebnisse des Mischprozesses sind. Die optische Ausgabe wird dann u.U. noch von anderen Optionen beeinflußt.

5.3.1 Mischregeln

join

Für die angegebenen Lokationsklassen wird ein Zusammenfassen von Lokationsreferenzen innerhalb der angegebenen Hierarchiestufen zu Bereichen zugelassen.

Der optionale Parameter *number* gibt die untere Grenze an, ab wievielen Zähleinheiten aufeinanderfolgende Lokationen zu einem Bereich zusammengefaßt werden können. Bei Fehlen dieses Parameters wird ein Defaultwert angenommen.

Syntax: join *loccls hierarchien* [*number*]

join `section (1-2)` 3 : 1.1 1.2 1.3 → 1.1–1.3
join `section (1)` 3 : 1.1 1.2 1.3 → 1.1 1.2 1.3
join `section (1)` 3 : 1 2 3 1.1 1.2 1.3 → 1–3 1.1 1.2 1.3

ignore-for-join

Die angegebenen Hierarchiestufen dürfen aus der Referenzliste entfernt werden, wenn ein Bereich von Lokationen auf höherer Ebene diese Lokationen mit einschließt.

Syntax: ignore-for-join *loccls hierarchien zielhier*

ignore-for-join `section (-) (2-)` : 5 6 6.1 6.2 7 → 5–7
ignore-for-join `manpage (-) (2-)` : 11 11a 12 13 13a 13b → 11–13

merge-to

Die Referenzen der Attributklasse $optattr_{from}$ dürfen auch von der Attributklasse $optattr_{to}$ als Mitglieder angesehen werden und bei der Bereichsbildung mitverwendet werden.

Syntax: merge-to $optattr_{from}$ $optattr_{to}$

merge-to `italic default` : (1.1 1.3) (*1.2*) → (1.1–1.3) (*1.2*)

5.3.2 Ausgaberegeln

drop-if-merged

Als Erweiterung der merge-to-Regel bedeutet diese Regel, daß eine Lokation beim Mischen mit einem Bereich einer anderen Attributklasse für die Ausgabe in der eigenen Attributklasse unterdrückt wird.

Syntax: drop-if-merged $optattr_{from}$ $optattr_{to}$

merge-to `italic default` : (1.1 1.3) (*1.2*) → (1.1–1.3) (*1.2*)
drop-if-merged `italic default` : (1.1 1.3) (*1.2*) → (1.1–1.3)

substitute-if-double

Diese Regel erzwingt das Verdrängen von Lokationsreferenzen eines Kategorieattributs bei Separate-Sorting durch ein anderes.

Syntax: substitute-if-double *optattr optattr*
substitute-if-double `bold default` : 3 4 5 **4** → 3 5 **4**
normalerweise : 3 4 5 **4** → 3 4 5 **4**

Das Standardverhalten von virtuellen Attributen wollen wir noch an einem Beispiel mit den Attributen `default` und `bold` zeigen.

1. Grundmenge: 11 13 14 15 18 **12 13 14 18**
2. Verdrängung: 11 **12 13 14** 15 **18**
3. Ausgabe: 11, **12**–**14**, 15, **18**

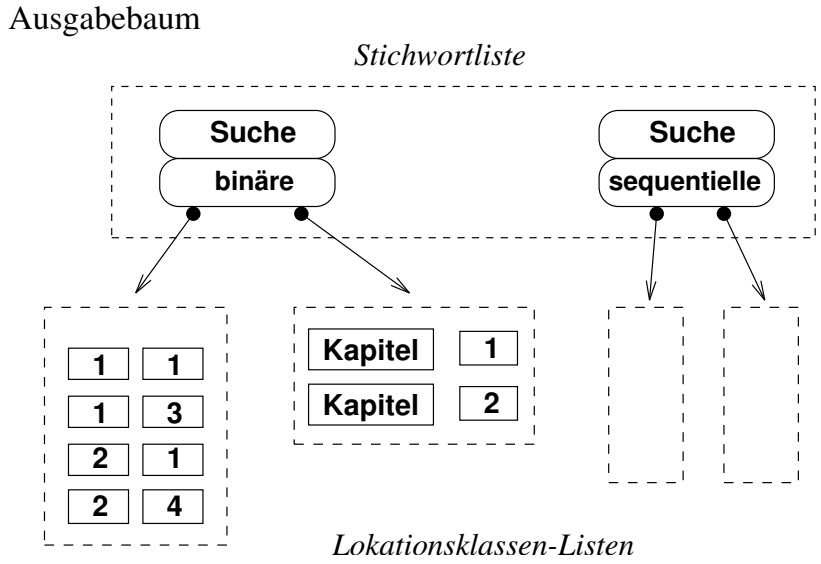
Zusammenfassung

Wir haben in diesem Abschnitt die Spezifikation von Regeln geleistet, die das Mischen und Sortieren von Lokationsreferenzen steuern sollen. Bei eingehender Untersuchung sind mit Sicherheit noch weitere solcher Regeln auffindbar. Wir haben uns hier bewußt auf die potentiellen Bedürfnisse der Anwender konzentriert und versucht eine zu komplexe Beschreibungsform zu vermeiden.

5.4 Ausgabeformatierung

Als Gegenstück zum Einlesen der Rohindexdaten muß nun die Ausgabe des fertig generierten Indexes in den Ausgabestrom modelliert werden. Hier ist eine hohe Konfigurierbarkeit wünschenswert, da sich die in der Literatur gefundenen Ausgabeformen verschiedener Indextypen erheblich voneinander unterscheiden. Wir können folgende Punkte als grundlegende Voraussetzungen an die Indexausgabe festhalten, um deren Grenzen zu umreißen:

1. Die Stichwörter werden in der Reihenfolge ihrer Sortierung ausgegeben. Dabei werden alle Unterstichpunkte ebenfalls in ihrer Sortierreihenfolge ausgegeben.
2. Die Lokationsreferenzen werden klassenweise ausgegeben. Eine Mischung zwischen verschiedenen Klassen ist nicht wünschenswert. Die Reihenfolge der Klassen wird im Indexstyle vorgegeben. Innerhalb einer Lokationsklasse haben wir nun eine sortierte Menge von Referenzen, die bezüglich ihrer Kategorieattribute vermischt sein können. Die Ausgabe der Lokationsreferenzen kann allerdings nicht eine Lokationsreferenz als kleinste Einheit auffassen, wie das Beispiel in Abschnitt 4.1.1 zeigt. Greifen wir dieses Beispiel auf, so erkennen wir die Klassenstruktur `([num]⊔[num])`. Allerdings wurden bei der Ausgabe die Lokationsreferenzen zerlegt aufgelistet, so daß wir als kleinste Darstellungseinheit die Strukturebenen betrachten müssen.
3. In Textsystemen werden Formatierungsanweisungen üblicherweise durch *Umgebungen* ausgedrückt. Ein Textfragment wird mit geeigneten Zeichenfolgen umklammert, um damit besondere Attribute wie Zeichensatz oder Schriftgröße zu verändern. Wir sollten also in der Lage sein, Ausgabeeinheiten mit Umgebungen zu versehen.



(Suche:binäre): 1.1, 1.3, 2.1, 2.4; Kapitel 1, Kapitel 2, ...

Abbildung 5.1: Schematischer Ausgabebaum für die Ausgabeformatierung

5.4.1 Ausgabebaum

Aufgrund dieser Überlegungen müssen wir ein Verfahren entwickeln, das es uns ermöglicht, eine Ausgabeanordnung vorzusehen, deren kleinste Ausgabeeinheit die Strukturkomponenten der Lokationsreferenzen bzw. Stichwörter sind. Die Datenstruktur des Indexes beruht auf einer Baumstruktur, die aus mehrstufigen Listen aufgebaut ist. Diesen Baum bezeichnen wir mit *Ausgabebaum*. Ein Beispielbaum ist in Abbildung 5.1 dargestellt. Die erste Ebene bildet die Liste der Stichwörter. Jedes Stichwort besteht dabei aus einer Liste aller Teilstichwörter. Die nächste Ebene besteht aus einer Liste von Lokationsklassen (durch dicke Punkte dargestellt). Eine Lokationsklasse besteht aus einer Liste von Lokationsreferenzen (vertikal angeordnet), die ihrerseits aus einer Liste von Strukturkomponenten (horizontal angeordnet) bestehen. Diese Strukturkomponenten bilden zusammen mit den Teilstichwörtern die kleinsten Einheiten des Ausgabebaums.

Da sowohl Stichwörter als auch Lokationsreferenzen aus Listen von Teilkomponenten bestehen, die in bestimmter Form ausgegeben werden müssen, werden wir die Ausgabestrategie allgemein genug formulieren, so daß sie grundsätzlich für beide gleichermaßen verwendbar ist.

Bei der Ausgabeformatierung müssen wir nun den kompletten Baum geeignet traversieren und entlang der Knoten und Verbindungen entsprechende Zeichenketten in den Ausgabestrom schreiben. Die Traversierung muß auf die gewünschte Ausgabeform anpaßbar sein. Betrachten wir die Lokationen in der

Abbildung, so sind z.B. die folgenden Ausgabeformen denkbar:

1. 1.1, 1.3, 2.1, 2.4; Kapitel 1, Kapitel 2
2. . . . ; Kapitel 1, 2
3. 1 1,3; 2 1,4; . . .

Wir müssen demnach in der Lage sein, Strukturkomponenten einzeln ausgeben zu können und zusätzlich die Ausgabe von bestimmten Elementen des Ausgabebaums gezielt zu unterdrücken.

5.4.2 Traversierung des Ausgabebaums

Nun stellt sich die Frage nach den notwendigen Ausgabekommandos und dem Traversierungsalgorithmus. Folgende Ausgabekommandos sind anhand der Traversierung und der Durchsicht verschiedener Indexe entworfen worden:

- Beim erstmaligen Eintritt in einen Knoten und beim endgültigen Verlassen sollten Ausgabekommandos definierbar sein: pre-print-node und post-print-node.
- Vor und nach einer Strukturkomponente: pre-print-layer und post-print-layer.
- Die Strukturkomponente, also das Element selbst: print-element.
- Bei listenförmigen Aufzählungen mit Trennzeichen ist es wünschenswert, Trennzeichen *nach* dem letzten Element unterdrücken zu können. Wir bezeichnen das Ausgabekommando, welches immer nach einer Strukturkomponente, aber nicht nach der letzten Strukturkomponente ausgegeben wird, als optional-post-print-layer.

  Beispiel: 1.1, 1.3, 1.5[,]
- Bei der Aufzählung von Bereichen (z.B. 1.2–1.5) muß das Bereichstrennzeichen (in diesem Fall '–') definierbar sein. Es tritt beim Ausdruck von Bereichen an die Stelle von optional-post-print-layer. Wir bezeichnen es mit optional-range.
- Es ist vor allem bei Stichwörtern notwendig, ein Wiederholungssymbol definieren zu können. Oft findet man Indexe in der Form

  Suche, binäre . . .
  ~ , sequentielle . . .

  Hier wird das Tilde-Zeichen als Wiederholungssymbol für das Stichwort der ersten Ebene verwendet. Üblicherweise wird das Wiederholungszeichen an jedem Seiten- oder Spaltenanfang nicht verwendet, um dem Leser eine leichtere Orientierung zu ermöglichen. Das Erkennen des Seitenanfangs oder anderer Positionen im Dokument kann jedoch nur durch das Textsatzsystem geleistet werden. Dieses Wiederholungssymbol bezeichnen wir mit repetition-symbol.
- Wir ordnen jedem Knoten die Eigenschaft *aktiv* oder *passiv* zu. Ein aktiver Knoten erzwingt bei der Ausgabe seine Einreihung in den momentanen

Ausgabematrix

Ausgabeform: 1.1.1, 1.1.2, 1.2.1–1.2.3, ... 2.1.1 ... $x.y.z$

Typ	pre-node	pre-layer	layer	rep.-symbol	post-layer	optional-post-layer	optional-range	post-node
A			*num*			‘**,**␣’		
A		‘**.**’	*num*					
A		‘**.**’	*num*				‘_’	

Ausgabeform: **1** 1.1, 1.2, 2.1–2.3, ...; **2** $x.y$, ...

Typ	pre-node	pre-layer	layer	rep.-symbol	post-layer	optional-post-layer	optional-range	post-node
P		*bold-on*	*num*		*bold-off*	‘**;**␣’		
A	‘␣’		*num*			‘**,**␣’		
A		‘**.**’	*num*				‘_’	

Tabelle 5.1: Tabelle mit Ausgabematrizen für unterschiedliches Markup eines Ausgabebaums

Ausgabestrom. Ein passiver Knoten überläßt die Entscheidung über seine Einreihung der Formatierungsroutine, die mittels Kontextvergleich die passende Einordnungsstrategie ermittelt.

Daraus ergibt sich nun eine Definitionsmatrix, in der für jede Strukturkomponente einer Lokationsklasse die Ausgabekommandos definiert werden können und die Komponenten als aktive (A) oder passive Einheiten (P) festgelegt werden müssen. Eine entsprechende Beispielmatrix ist in Tabelle 5.1 dargestellt.

Als natürliches Durchlaufverfahren des Ausgabebaums wählen wir eine modifizierte Version der Traversierung *Hauptreihenfolge*.

Jeder Ebene eines Stichworts und jeder Strukturkomponente einer Lokationsklasse wird ein entsprechendes Markup zugewiesen. Bei der Ausgabe eines neuen Stichworts wird als *Kontext* der Druckschlüssel des vorherigen Stichworts mitgegeben. Die Formatierungsroutine kann durch einen Vergleich dieses Kontexts mit dem aktuellen Stichwort entscheiden, ab welcher Ebene sich beide unterscheiden. Passive Knoten werden bei Gleichheit mit dem Kontext unterdrückt. Unterscheiden sich Kontext und Schlüssel, oder ist das Markup einer Ebene aktiv oder mit einem Wiederholungssymbol versehen, wird die Ausgabe des Knotens gestartet.

Kontext

Entscheidet sich die Formatierungsroutine für eine Ausgabe einer Strukturkomponente, so werden die Formatierungsanweisungen für das Öffnen von Umgebungen in den Ausgabestrom hineingeschrieben.

Die Formatierungsanweisungen zum Schließen der jeweiligen Umgebungen werden auf einem Stapel abgelegt, da ihre Ausgabe verzögert werden muß, bis die Unterstrukturen abgearbeitet sind. Da sich die Stichwörter und Lokationsreferenzen an bestimmten Stellen in den Ausgabeprozeß einreihen, muß bei der Einreihung dafür gesorgt werden, daß der Stapel mit den Umgebungen immer geeignet geleert wird und die Anweisungen in den Ausgabestrom eingefügt wer-

Umgebungsstapel

den. Dies beschreibt ein einfaches aber mächtiges Verfahren, die Ausgabe von Lokationsreferenzen zu steuern.

5.4.3 Stichwortgruppen

Stichwortgruppen

Bei der Darstellung von Indexen ist es üblich, Stichwortgruppen durch spezielle Informationen visuell darzustellen. Oft bildet man Stichwortgruppen nach dem ersten Buchstaben eines Stichworts und trennt beispielsweise in einem Index jede Buchstabengruppe optisch voneinander ab. Da eine solche optische Abtrennung Teil der Ausgabeformatierung ist, müssen wir auf geeignete Weise die Definition von Stichwortgruppen und deren Markup vorsehen.

Da die Sortierung der Stichwörter bereits durch das sort-mapping festgelegt wurde, müssen wir uns einfach nur eine Liste von Stichwortgruppen durch Tupel der Form

$$keywordgrp = (keyword, markup) \text{ mit } keyword,\ markup : String$$

definieren und können beim Traversieren des Stichwortbaums bei Überschreiten der Gruppengrenzen die entsprechenden Markups in den Ausgabestrom einfügen. Im makeindex-System von [CH88] werden Stichwortgruppen grundsätzlich nur aus den Anfangsbuchstaben der Stichwörter gebildet und optische Trennungen in den Ausgabestrom eingefügt. Im *International MakeIndex* [Sch91] werden Anweisungen eingeführt, welche die Bildung von zusätzlichen Buchstabengruppen und deren Markup erlaubt.

Die in diesem Verfahren angewandte Grundidee besteht aus einem zusätzlichen Mapping des Sortierungsschlüssels eines Indexeintrags auf einen *Stichwortgruppenschlüssel*. Dieses Mapping wird in den Systemen implizit als die Abbildung auf den ersten Buchstaben angenommen.

Wir erweitern die Ausgabe zu einem zweischichtigen Modell durch die Bildung von Stichwortgruppen, denen jeweils ein zusätzliches Markup zugeordnet wird. Wir beschreiben die Ausgabeinformationen in einer Liste G von Stichwortgruppen mit $G = \{g_1, \ldots, g_n\}$ mit $g_i = (\{l_1, \ldots, l_m\}, markup)$. Die Elemente g_i sind also Obergruppen, denen als Gesamtheit ein Markup zugeordnet ist, während die l_i Stichwortgruppen der üblichen Form sind. Ein Markup besteht in der einfachsten Form aus einem Paar von Strings, welche die entsprechenden Formatierungs-Umgebungen einleiten und abschließen.

Diese Definition erscheint uns im Moment ausreichend genug, um auch komplexere Indexe geeignet auszugeben.

5.5 Indexstyle

Indexstyle

Mit dem Abschluß des vorigen Abschnitts haben wir nun die grundlegenden Parameter des Indexsystems umrissen. Dies gibt uns nun die Möglichkeit, das Datenmodell des *Indexstyles* zu entwerfen. Der Indexstyle ist die Zusammenfassung aller Parameter und Spezifikationen, mit denen unser System in der Lage sein soll, einen Indexierungsprozeß durchzuführen. Zu diesen Parametern zählen u.a. die folgenden Daten:

- Beschreibung der Lokationsklassen mit Struktur und Regelwerk.
- Beschreibung der Indexklassen, insbesondere deren Ausgabeformatierung.
- Regelwerk zum Mischen und Sortieren der Stichwörter. Insbesondere das regelbasierte Substitutionsschema analog zum *International MakeIndex*.
- Informationen zur Ausgabeformatierung des kompletten Indexes mit der Definition von Stichwortgruppen.

Aus der Menge der Parameter wird auch das Ziel der Arbeit deutlich, ein hoch parametrisierbares und individuell vom Benutzer definierbares Indexsystem zu entwerfen. Möglichst viele Parameter sollen in die externe Spezifikation ausgelagert werden, um höchste Konfigurierbarkeit zu erlangen.

Bei den folgenden Definitionen verwenden wir bei Bezeichnern den Exponenten ^{is}, um kenntlich zu machen, daß es sich um Datenelemente des Indexstyles handelt. Weil wir hier die Parameter von realen Objekten des Indexeintrags beschreiben, bietet es sich an, ähnliche Bezeichner dafür zu verwenden, um die Analogie kenntlich zu machen. Gleichzeitig schließen wir eine Verwechslung mit den Elementen des Indexeintrags aus, die ohne Exponenten geschrieben werden.

DEFINITION 5.5.1 *Ein* ***Indexstyle*** *idxsty ist eine Liste von Indexklassen*

$$idxsty = (\ idxcls^{is}_{set}\ )\ .$$

DEFINITION 5.5.2 *Eine* ***Indexklasse*** $idxcls^{is}$ *ist ein Tupel*

$$idxcls^{is} = (\ idxclsname^{is},\ basetype^{is}_{set},\ maprule^{is}_{lst},\ idxtag^{is}_{lst}\ )$$

mit den Komponenten

$idxclsname^{is}$	:	***Klassenname****, String*
$basetype^{is}_{set}$	:	***Komponententypen****, Alphabete und Aufzählungen*
$loccls^{is}_{set}$	:	***Lokationsklassen***
$idxtag^{is}_{lst}$	:	***Markup-Informationen****.*

DEFINITION 5.5.3 *Eine* ***Lokationsklasse*** $loccls^{is}$ *ist ein Tupel*

$$loccls^{is} = (\ locclsname^{is},\ laystruc^{is},\ locrefrule^{is}_{set}\ )$$

mit den Komponenten

$locclsname^{is}$	:	***Lokationsklassenname****, String*
$laystruc^{is}_{set}$	:	*Menge der* ***Lokationsstrukturkomponenten***
$locrefrule^{is}_{set}$	:	*Menge der* ***Lokationsverarbeitungsregeln****, siehe Definition 5.3.1 auf Seite 28.*

DEFINITION 5.5.4 *Eine* ***Lokationsstrukturkomponente*** $laystruc^{is}$ *ist ein Tupel*

$$laystruc^{is} = (\ basetype^{is},\ separator^{is}\ )$$

mit den Komponenten

basetype^{is} : ***Komponententyp***, *Element aus Basetypes*^{is}
separator^{is} : ***Trennzeichen***, *String.*

Mit diesen Definitionen haben wir das schematische Datenmodell des Indexstyles festgelegt. Wir verzichten auf eine detailliertere Beschreibung aller Komponenten, da sie für das Verständnis des Gesamtsystems nicht relevant sind.

Um die Benutzbarkeit des Systems zu erhöhen, definieren wir eine `default`-Indexklasse, deren Definitionen in allen anderen Indexklassen ebenfalls gelten sollen. Die Festlegungen dieser Indexklasse werden jedoch von Definitionen in den jeweiligen Klassen überschrieben.

Kapitel 6

Indexverarbeitung

Wir kommen nach der Daten- und Strukturmodellierung in Abschnitt 5 zu den einzelnen Verfahren der Indexverarbeitung. Vergegenwärtigen wir uns dazu noch die Einbettung des Indexierungssystems in den Prozeß der Indexerstellung, wie er in Abbildung 2.1 auf Seite 4 dargestellt ist.

Wir betrachten nun die einzelnen Abschnitte der Indexverarbeitung und behandeln insbesondere die Verarbeitung der Lokationsreferenzen vertieft, weil sie im Hinblick auf die bestehenden Systeme neu entwickelt wurde.

6.1 Indexstyle einlesen

Das Einlesen des Indexstyles dient dazu, dem System alle Parameter bekannt zu machen, die von Benutzerseite spezifiziert wurden. Der Indexstyle ist eine Menge von Dateien, in denen in deklarativer Form die Angaben über die Parameter des Indexsystems spezifiziert sind. Diese Spezifikationen werden eingelesen und geeignet abgelegt.

6.2 Indexeinträge einlesen und normalisieren

Die Indexeinträge werden eingelesen und jeweils ein internes Objekt *idxent* erzeugt. Dabei fallen noch die folgenden Aufgaben an:

- Der Indexeintrag muß auf Gültigkeit überprüft werden.
- Die Indexschlüssel müssen durch die Sortier- und Mischregeln gegebenenfalls normalisiert werden.
- Dem Indexeintrag muß die passende Lokationsklasse zugeordnet werden.
- Die Ordnungsnummern der einzelnen Strukturkomponenten müssen anhand der Lokationsklassendefinition errechnet werden.

Normalisieren

Das *Normalisieren* der Indexschlüssel wird analog zum Verfahren des *International Makeindex* durchgeführt. Aus dem Schlüssel *k_key* wird mit den Mischregeln der Mischschlüssel gebildet, aus welchem mit Hilfe der Sortierregeln der Sortierschlüssel generiert wird.

6.3 Stichwortmischung und -sortierung

Stichwortmischung dient dazu, alle Indexeinträge zusammenzufassen, deren Mischschlüssel *m_key* identisch ist. Wir definieren dazu eine Funktion *join*, die zwei Indexeinträge miteinander vereinigt.

Diese Funktion kann folgendermaßen beschrieben werden. Seien $\mathcal{I}_1$ und $\mathcal{I}_2$ Indexeinträge, so ist

$$\begin{aligned} join(\mathcal{I}_1, \mathcal{I}_2) \equiv (\quad & \mathcal{I}_1.k\_key \triangleleft \mathcal{I}_2.k\_key,\ \mathcal{I}_1.m\_key, \\ & \mathcal{I}_1.s\_key \triangleleft \mathcal{I}_2.s\_key,\ \mathcal{I}_1.p\_key \triangleleft \mathcal{I}_2.p\_key, \\ & \mathcal{I}_1.locref_{\,lst} \ \cup\ \mathcal{I}_2.locref_{\,lst}). \end{aligned}$$

Die Funktion $\triangleleft$ ist hier wie folgt definiert:

$$x \triangleleft y = \begin{cases} y & \text{falls } x \text{ leer} \\ x & \text{sonst.} \end{cases}$$

Nach dem Mischen der zusammengehörenden Indexeinträge wird der Index lexikographisch anhand der Sortierschlüssel sortiert.

6.4 Lokationsreferenzmischung und -sortierung

Durch die Hinzunahme von verschiedenen Lokationsklassen muß die Mischung und Sortierung von Lokationsreferenzen anhand der in Abschnitt 5.3 definierten Misch- und Sortierregeln erfolgen. Der Verarbeitungsprozeß der Lokationsreferenzen eines Indexeintrags gliedert sich in die folgenden Phasen:

1. Lokationsklassenmatrix (siehe Abbildung 4.2) aufstellen. Den Lokationsreferenzen wird ihre Lokationsklasse zugeordnet. Die Kategorieattribute werden bei diesem Klassifizierungsprozeß noch nicht beachtet.
2. Auflösen der Matrix und Mischen und Sortieren jeder einzelnen Lokationsklasse gemäß ihrer Kategorieattribute. Dieser Prozeß läuft in folgender Weise ab:
   i. Unterteilung der Lokationsreferenzen in Mengen, deren Elemente zu einer Lokationsklasse gehören.
   ii. Die gebildeten Lokationsreferenzmengen werden gemäß der Unterscheidung in Separate- und Mixedsorting bezüglich ihrer Kategorieattribute unterteilt.
   iii. Auflösen der merge-to-Regeln, indem die entsprechenden Referenzen zusätzlich in die Referenzmengen der Kategorie-Nachbarn eingefügt werden (Primärattribut), wobei ein Hinweis auf ihr originales Kategorieattribut beibehalten wird (Sekundärattribut).
   iv. Sortieren dieser Mengen gemäß der definierten Totalen Ordnung.
   v. Erzeugung von Bereichen durch das Zusammenfassen von aufeinanderfolgenden Lokationsreferenzen. Dabei werden sowohl das Primärattribut als auch die Sekundärattribute verwendet.

vi. Verdrängung von Lokationsreferenzen innerhalb von virtuellen Attributen durch Referenzen mit höherer Vorrangstufe.

vii. Eliminierung von Lokationsreferenzen, welche durch eine merge-to-Regel nicht in einem Bereich aufgenommen werden konnten oder die aufgrund einer erfolgreichen drop-if-merged-Regel wegfallen. Des weiteren Eliminierung von Lokationsreferenzen aufgrund der substitite-Regel.

Nach diesem Arbeitsabschnitt ist die grundsätzliche Arbeit am Index vollendet. Wir müssen uns zum Abschluß mit der Ausgabe des kompletten Indexes in den Ausgabestrom befassen.

6.5 Ausgabeformatierung

Die Ausgabe des Indexes ist im wesentlichen die in Abschnitt 5.4 beschriebene Traversierung des Ausgabebaums. Wir müssen dem Ausgabestrom noch zusätzliche Informationen beifügen, die im Indexstyle definiert sind. Des weiteren müssen die Stichwortgruppen während der Traversierung korrekt erkannt und die entsprechenden Ausgabekommandos aufgerufen werden.

Kapitel 7

Implementierung

In diesem Abschnitt soll auf die konkrete Implementierung des **xindy**-Systems eingegangen werden. Dabei wird kurz auf die verwendete Programmiersprache und auf die Entwicklungsumgebung eingegangen. Allgemeine Anforderungen an das System waren:

1. Das System sollte möglichst portabel sein. Die bestehenden Indexsysteme werden besonders intensiv im Zusammenhang mit dem TEX- bzw. LATEX-System genutzt. Es sollte möglich sein, das zu entwickelnde System weiterhin in diesem Bereich einzusetzen, wenngleich das System auf andere Anwendungssysteme hin konfigurierbar ist.
2. Das System sollte eine einfache und schnelle Umsetzung des entworfenen Datenmodells in eine Programmiersprache ermöglichen und leicht erweiterbar sein.
3. Es sollte eine möglichst schnelle Implementierung gestatten und die wesentlichen Punkte des Modells umfassen. Zur Realisierung sollten auch fertige Bibliotheken verwendet werden.

7.1 Entwicklungsumgebung

Aus den aufgeführten Forderungen haben wir uns zunächst für die Programmiersprache C++ entschieden, da eine einfache Abbildung in entsprechende Klassen möglich ist. Diese Sprache ist außerdem auf den wichtigsten Betriebssystemen verfügbar.

Nach weiteren Überlegungen wurde eine Implementierung in Common Lisp [Ste84, Ste90] vorgenommen, um die Entwicklungszeit zu verkürzen und im Rahmen dieser Arbeit möglichst viele neue Punkte des Modells zu verifizieren. Dabei wurde umfangreichen Gebrauch von den objektorientierten Möglichkeiten des *Common Lisp Object System* CLOS [Kee88, BDG+88] gemacht.

Um außerdem Erfahrung mit *Literate Programming* zu sammeln, wurde das System mit Hilfe des LP-Systems[1] `noweb` [Ram94] erstellt und mit Hilfe von LATEX 2_ε gesetzt.

[1] LPS bedeutet Literate Programming System. Man versteht darunter die parallele Beschreibung von Programm und Dokumentation in gemischter Form innerhalb eines Dokumentes.

Wertvolle Hilfe für textuelle Erstellung der Studienarbeit in LaTeX 2_ε und Nachformatierung der `noweb`-Programme waren [GMS94, Lam86, Kop94]. Als mathematisches Nachschlagewerk wurde im wesentlichen [Ihr93] benutzt.

7.2 Struktur

Da das System in Common Lisp implementiert wurde, bot sich die Möglichkeit, den Indexstyle mit einer deklarativen Beschreibung in Form von Lisp-Ausdrücken anzugeben, die der Interpreter direkt lesen kann. Dadurch ist die ursprünglich angestrebte Erstellung eines Parsers für den Indexstyle entfallen und bietet letztendlich einen höheren Konfigurierungsgrad als ursprünglich angestrebt.

Für die Rohindexerkennung war ursprünglich ein eigener Parser in Form eines `perl`-Skripts [WS92, Sch93] vorgesehen. Diese skriptorientierte Programmiersprache bietet vielfältige Möglichkeiten zur Erkennung von Mustern und verwendet Reguläre Ausdrücke, um die Indexeinträge zu filtern und in eine Folge von Lisp-Ausdrücken umzuwandeln. Diese werden dann von **xindy**-System gelesen und ausgewertet.

Das entsprechende Skript ist leicht auf andere Eingabeformate und Textsysteme modifizierbar. Das hier vorgestellte Konzept nutzt die Stärken vorhandener Systeme und bildet mit der Symbiose von `perl` und Common Lisp eine schnelle und leicht wart- und adaptierbare Gesamtlösung.

7.3 Portabilität

Das **xindy**-System wurde auf den folgenden Rechnerarchitekturen und Betriebssystemen getestet:

- IBM RS/6000, AIX 3.2 mit `perl-4.0.36` und `clisp-94-10-26`.
- IBM-AT-kompatibler Rechner auf Intel-i486-Basis, Linux 1.2, `perl-4.0.36` und `clisp-94-10-26`.

Das `clisp`-System ist eine Public-Domain-Implementierung von Common Lisp und auf diversen ftp-Servern verfügbar.

Das **xindy**-System sollte grundsätzlich auf allen Plattformen lauffähig sein, auf denen Common Lisp und `perl` verfügbar sind. Sollte letzteres nicht verfügbar sein, so muß evtl. eine Lösung mit einer anderen Sprache verwendet werden. Die Rohindexerkennung und Generierung der Lisp-Ausdrücke kann auch auf beliebige andere Weise erfolgen, solange das Lisp-Kernmodul entsprechend mit Informationen versorgt wird.

7.4 Aktueller Implementierungsstand

Im folgenden Abschnitt soll die bisherige Implementierung skizziert werden und insbesondere die Modellierung der Klassen und Module vorgestellt werden. Abbildung 7.1 zeigt einen Überblick über die Beziehungen zwischen den zentralen

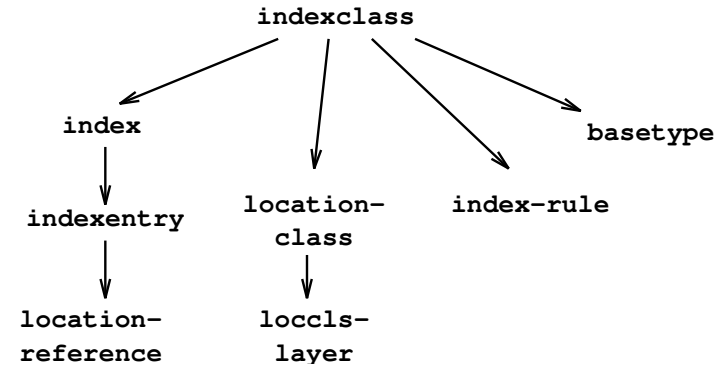


Abbildung 7.1: Schematisches Datenmodell der wichtigsten Klassen der Implementierung mit *enthält*-Beziehungen

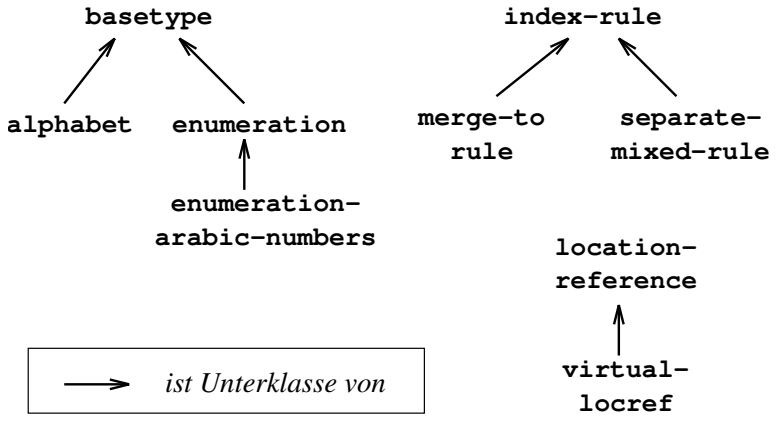


Abbildung 7.2: Vererbungsgraphen der wichtigsten Klassen der Implementierung

Klassen des Systems. Sie stellt die *enthält*-Relationen der wichtigsten Klassen dar und beschreibt welche Klasse *Container* für eine andere Klasse sind.

In Abbildung 7.2 sind die wichtigsten Vererbungsbeziehungen zwischen Klassen dargestellt. Sie sollen einen Überblick über die dem System innewohnenden Beziehungen geben. Weitere implementationsspezifische Vererbungsbeziehungen sind nicht dargestellt. Der Modulgraph in Abbildung 7.3 zeigt die *benutzt*-Relationen der Module.

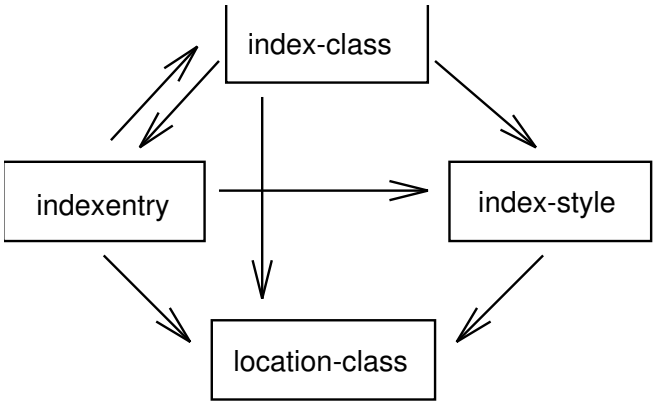


Abbildung 7.3: Modulgraph mit *benutzt*-Relationen

Klassenbeschreibungen

Wir beschreiben nun für die angeführten Klassen ihre jeweilige Funktion und die wichtigsten Komponenten und Methoden bzw. Schnittstellen, um einen Überblick über das Gesamtsystem zu erhalten.

`indexclass`

Diese Klasse ist die oberste Datenstruktur im System. Sie enthält alle Definitionen, die zu einer Indexklasse gehören. Dazu gehören sowohl die Komponenten, die den Indexstyle definieren, als auch der Index selbst.

Komponenten:

- `name` – Hier ist der Name der Indexklasse abgelegt. Es exisitiert immer eine Indexklasse mit Namen `default`.
- `basetypes` – Alle definierten Komponententypen werden hier abgelegt. Die Komponententypen werden über ihre Namen angesprochen. Die Elemente sind Instanzen der Klasse `basetype`.
- `locclasses` – Alle vom Benutzer definierten Lokationsklassen werden hier abgelegt. Die Elemente sind Instanzen der Klasse `location-class`.
- `index` – Hier ist der Index abgelegt. Es handelt sich um eine Instanz der `index`-Klasse.
- `keyword-markup` – Die Ausgabeformatierung für die Stichwörter werden hier abgelegt.
- `succ-table` – ist die Tabelle mit den Nachfolgerdefinitionen. Hierin ist das *Dokumentwissen* enthalten.
- `merge-to-rules` – sind die merge-to-Regeln. Sie sind Instanzen der Klasse `index-rule`.
- `sep-mix-rule` – enthalten die separated-mixed-Regeln.

Methoden:

Da die Klasse oberste Verwaltungseinheit ist, stehen im wesenlichen nur Operationen zum Füllen und Zugreifen der Komponenten zur Verfügung.

`basetype`

Diese Klasse ist die abstrakte Basisklasse für Alphabete und Aufzählungen.

Komponenten:

`name` – enthält den Namen des Komponententyps.

`base-alphabet` – das Basisalphabet des Komponententyps.

Methoden:

`prefix-match` *string basetype* [*Generische Funktion*]

Das Argument *string* ist eine Zeichenkette, *basetype* ist eine Instanz eines Komponententyps. Die Funktion versucht, einen Präfix der Zeichenkette dem angegebenen Komponententyp zuzuordnen. Die Methoden müssen für alle abgeleiteten Klassen von `basetype` definiert sein und einen multiple-value der Form (*matched-str rest-str ordnum*) zurückliefern. Dabei bezeichnet *matched-str* eine Zeichenkette, die dem Komponententyp zugeordnet werden konnte, *rest-str* der Rest der Zeichenkette und *ordnum* die Ordnungszahl, die der Zeichenkette zugeordnet wurde.

`alphabet`

Diese Klasse ist Basisklasse aller Alphabete und von `basetype` abgeleitet.

Komponenten:

`symbols` – enthält eine Liste aller Symbole des Alphabets. Ein Symbol ist dabei ein Wort über dem Basisalphabet. Wir ermitteln bei Alphabeten das Basisalphabet durch Analyse der Symbole des Alphabets.

Methoden:

Es ist eine allgemeine Methode `prefix-match` definiert, die das oben beschriebene Verhalten für alle Alphabete nachbildet.

`enumeration`

Diese Klasse ist Basisklasse aller Aufzählungen und von `basetype` abgeleitet.

Komponenten:

Konstruktor – Der Konstruktor bekommt das Basisalphabet übergeben, da keine weiteren Informationen über die Aufzähung bekannt sind.

Methoden:

Da Aufzählungen wie die arabischen oder die römischen Zahlen eine grundsätzlich andere Struktur aufweisen als Alphabete, muß für jede konkrete Aufzählung eine entsprechende Methode `prefix-match` implementiert werden.

`location-class`

Diese Klasse ist Container für alle zu einer Lokationsklasse gehörenden Informationen.

Komponenten:

`name` – enthält den Namen der Lokationsklasse.

`layers` – enthält die Liste der Ebenen der Lokationsstruktur. Die Ebenen sind Instanzen der Klasse `loccls-layer`.

`join-layers` – enthält eine Liste der Nummern aller Strukturkomponenten, die prinzipiell zur Bereichsbildung herangezogen werden dürfen.

`ordnum` – die einer Lokationsklasse zugeordnete Ordnungsnummer. Dies wird zur Definition eines Ordnungskriteriums auf Lokationsklassen für die Sortierung von Lokationsreferenzen benötigt. Das momentan implementierte Ordnungskriterium bezieht sich auf die Reihenfolge der Definitionen der Lokationsklassen. Lokationsreferenzen von früher definierten Lokationsklassen erscheinen bei der Ausgabe zuerst.

Methoden:

Die zugeordneten Methoden dienen dem Zugriff auf die Komponenten der Instanzen.

`location-reference`

Diese Klasse verwaltet eine Lokationsreferenz. Sie ist Container für die Strukturkomponenten und Attribute der Lokationsreferenz.

Komponenten:

`layers` – hierin finden sich die einzelnen Ebenen der Lokationsreferenz. Eine Ebene ist ein Objekt der Klasse `locref-layer`.

`ordnums` – Die Ordnungszahlen der Ebenen werden hier abgelegt. Ursprünglich wurde in jeder Ebene die zugeordnete Ordnungszahl abgelegt. Durch die Zusammenfassung in einer direkt zugreifbaren Liste wurde eine einfache Optimierung vorgenommen.

`optattr` – beinhaltet den Namen des zugeordneten optischen Attributs.

`locclass` – beinhaltet den Namen der zugeordneten Lokationsklasse.

Methoden:

`locref<` *locref-1 locref-2* [*Funktion*]

Diese Funktion vergleicht die Ordnungszahlen zweier Lokationsreferenzen und liefert `t` oder `nil`.

`markup-object` *location-reference markup context env-stack* [*Methode*]

Implementiert die Ausgabeformatierung für eine Lokationsreferenz.

`indexentry`

Diese Klasse enthält alle Informationen über einen Indexeintrag. Sie stellt einen Container dar, der beim Einlesen eines Indexeintrags initialisiert wird.

Komponenten:

`main-key` – enthält den Hauptschlüssel des Indexeintrags.

`merge-key` – ist der Schlüssel zum Mischen zweier Einträge.

`sort-key` – ist der Sortierschlüssel.

`print-key` – ist der Schlüssel, der für die Ausgabe verwendet wird.

`locrefs` – sind die dem Indexeintrag zugeordneten Lokationsreferenzen.

Methoden:

`process-indexentry` *indexentry* [*Funktion*]

Sortiert und mischt alle Lokationsreferenzen dieses Indexeintrags und bildet Bereiche aus aufeinanderfolgenden Referenzen.
Diese Verarbeitung gliedert sich in verschiedene Unterprozesse. Die Funktionsweise folgt der in Abschnitt 6.4 beschriebenen Vorgehensweise. Die Lokationsreferenzen werden in Gruppen unterteilt, die zu verschiedenen Lokationsklassen gehören. Anschließend wird jede Gruppe gemäß der Regeln des separate- bzw. mixed-sorting unterteilt. Es folgt die Anwendung der `merge-to`-Regeln und eine Sortierung gemäß der Totalen Ordnung. Zuletzt werden die Bereiche gebildet.

`markup-object` *indexentry markup-list context-list env-stack* [*Methode*]

Implementiert die Ausgabeformatierung für einen Indexeintrag.

`index`

Diese Klasse ist ein Container für die Liste der Indexeinträge.

Komponenten:

`entries` – enthält die Liste der Indexeinträge.

Methoden:

`indexentry=` *indexentry-1 indexentry-2* [*Funktion*]

Überprüft, ob zwei Indexeinträge bezüglich ihres Mischschlüssels gleich sind.

`merge-indexentry-to-index` *indexentry index* [*Funktion*]

Fügt einen Indexeintrag in den Index ein und vereinigt ihn gegebenenfalls mit einem Indexeintrag gleichen Mischschlüssels.

`merge-indexentries` *indexentry-1 indexentry-2* [*Funktion*]

Vereinigt zwei Indexeinträge miteinander, deren Mischschlüssel gleich sind und liefert einen neuen Indexeintrag (*siehe Abschnitt 6.3*).

`markup-object` *index markup-list context-list env-stack* [*Methode*]

Implementiert die Ausgabeformatierung für einen Index.

`process-index` *indexclass* [*Funktion*]

Verarbeitet den zur Indexklasse *indexclass* gehörenden Index.

`markup`

Diese Klasse definiert die Markup-Struktur für die Ausgabeformatierung. Sie definiert Ausgabeprimitive für die `markup-object`-Methoden und behandelt die Verwaltung der geöffneten Umgebungen mittels eines Stacks.

Komponenten:

– sämtliche Komponenten wie in Abschnitt 5.4.2 beschrieben.

Methoden:

`markup-object` *index markup context env-stack* [*Generische Funktion*]

Diese generische Funktion wird verwendet, um die Komponenten, die bei der Ausgabeformatierung beteiligt sind, mit einer einheitlichen Schnittstelle zu versehen. Jede Komponente benötigt eine an die jeweilige Klasse gebundene Methode, fügt ihre Ausgabekommandos in den Ausgabestrom hinzu und legt die Ausgabestrings für das Schließen von Formatierungsumgebungen auf den Environment-Stack. Dieser Stack wird dann von der Funktion `close-environment` gegebenenfalls entleert.

In Abbildung 7.4 ist der generelle Algorithmus der `markup-object`-Methoden dargestellt. Die Punktnotation *list*.komp liefert die Komponente komp des ersten Elements der Liste *list*.

`close-environments` *env-stack* `&optional` *stack-length* [*Funktion*]

Schließt die auf dem Stack geöffneten Umgebungen der Ausgabeformatierung bis der Stack nur noch die Tiefe *stack-length* besitzt. Fehlt das optionale Argument so werden alle Umgebungen geschlossen. Der Stack wird in den `markup-object`-Methoden aufgebaut.

`mprint` *object* [*Generische Funktion*]

Definiert eine generische Funktion für Ausgabeprimitive wie Strings und Zahlen. Entsprechende Methoden sind dafür implementiert.

```
function markup-object (object markup-list context-list env-stack)
begin
   object-list := ≪hole object-list aus object≫
   while (context-list.value == object-list.value) or
         (markup-list.typ == aktiv)
      counter++
      ≪entferne erste Elemente aus object-list markup-list context-list≫
   endwhile
   env-stack := close-environments(env-stack counter)
   while not empty(object-list)
      mprint(markup-list.pre-node)
      mprint(markup-list.pre-layer)
      ≪drucke nun object-list.value bzw. markup-list.repetition-symbol≫
      mprint(markup-list.post-layer)
      push({markup-list.post-node, ≪evtl. auch optional-post-layer≫ },
           env-stack)
      ≪entnehme erstes Element aus object-list und markup-list≫
   endwhile
   ≪bearbeite Unterelemente von object≫
endfunc.
```

Abbildung 7.4: Basisalgorithmus der `mark-object`-Methoden

Kapitel 8

Zusammenfassung

Das **xindy**-System ist eine Weiterentwicklung der makeindex- und *International MakeIndex*-Systeme.

Durch eine komplette theoretische Neuanalyse von Indexen konnten die Erfordernisse an ein Indexsystem überarbeitet werden. Die intensive Analyse von Lokationsklassen und deren Verarbeitungsprozesse sowie die neuartige universelle Ausgabeformatierung bilden den Hauptteil dieser Studienarbeit.

Die Implementierungszeit konnte durch die vollständige theoretische Erarbeitung des Modells und der Verarbeitungsverfahren drastisch verkürzt werden.

Während der Modellierung führte vor allen Dingen die semantische Bedeutung, die bestimmten Ausgabeformen von Lokationsreferenzen zugeordnet werden mußte, zu Problemen. Besondere Schwierigkeit lag in der Kategorisierung von Ausgabeformen und inwieweit Lokationsreferenzen unterdrückbar sein sollten. Wir haben uns hier zum einen an existierende Indexe gehalten und zum anderen auch eigene Überlegungen angestellt, die in die Modellierung eingeflossen sind. Um die Komplexität niedrig zu halten und auch den Benutzer nicht mit übertriebener Konfigurierbarkeit zu überfordern, haben wir uns letztendlich für pragmatische, aber leistungsfähige Lösungen entschieden.

Die Erstellung eines Indexes ist generell auch eine Geschmacksfrage. An dieser Stelle ist es aufgrund der Vielfalt der potentiellen Wünsche der Autoren schwierig, eine optimale Gesamtlösung auszuarbeiten.

Weiterführende Arbeiten

Im Rahmen der Modellierung eines Indexes und der damit verbundenen Algorithmen sind zwei Probleme grundsätzlicher Natur in den Vordergrund getreten.

Das erste Problem ist die Erkenntnis, daß die Lokationsreferenzen als Informationsträger über das zu indizierende Dokument nicht ausreichend sind, um bestimmte in der Praxis vorkommende Indexe nachzubilden. Das Hauptproblem ist in fehlendem Dokumentwissen begründet. Bereits die einfachste Gliederungsform in Punktnotation (1.2.3...) mit verschiedenen Gliederungstiefen macht die Bestimmung des Nachfolgers einer Lokation unlösbar, sofern dem System die benötigten Informationen nicht zusätzlich zugeführt werden.

Dokumentwissen

Das zweite Problem besteht darin, daß ein Index sich auch typographischen Satzregeln unterwerfen muß, die z.B. verlangen, daß die Ersetzung von Stichwörtern durch Wiederholungssymbole (*siehe Abschnitt 5.4.2*) an Seiten- und Spaltenanfängen unterdrückt werden sollte. Um dieses Problem zu lösen, benötigt das Textsatzsystem allerdings Informationen, die während des Indexierungsvorgangs ermittelt werden. Wie man sieht, sind diese Probleme nur dann lösbar, wenn ein geeigneter Informationsaustausch zwischen beiden Systemen stattfindet. Die untersuchten Textsatzsysteme bieten allerdings keine direkte Unterstützung, um ein Indexierungssystem direkt an den Textsatzprozeß anzukoppeln.

textsatzabhängige Indexausgabe

Interessant ist es die Spezifikation der Ausgabeformatierung und der Regeln zum Sortieren und Mischen der Lokationsreferenzen zu überarbeiten. Im Moment ist Wissen über die zugrundeliegenden Algorithmen nötig, um eine solche Spezifikation zu erstellen. Dies ist jedoch für einen Benutzer eine sehr unbefriedigende Lösung und eine Untersuchung, ob *interaktive* Werkzeuge dieses Problem vereinfachen können, ist mit Sicherheit sinnvoll.

Des weiteren könnte es sinnvoll sein, das Konzept der Indexklassen weiter zu untersuchen. Insbesondere könnte man sich mit der Bildung einer Hierarchie von Klassen beschäftigen, um Klasseneigenschaften durch Vererbung weiterzugeben. In diesem Zusammenhang kann man auch die Gesamt- und Masterindexe und ihre Eigenschaften vertieft untersuchen.

Indexklassenvererbung

Kapitel 9

Glossar

Attributierte Strukturreferenz
: Strukturreferenz, der ein zusätzliches Attribut zugeordnet ist. Die Lokationsreferenz '35*f.*', die auf Seite 35 und die folgenden Seiten verweist, besteht aus der Strukturreferenz ([35]) und dem Attribut '*f.*'.

Basisalphabet
: Teilmenge eines Dokumentalphabets. *Siehe auch Definition 4.1.7, S. 13.*

Dokumentalphabet
: Gemeinsames zugrundeliegendes Alphabet des Textsatzsystems und des Indexsystems. Üblicherweise ASCII, ISO-Latin–Familie oder Unicode. *Siehe auch Definition 4.1.1, S. 12.*

Dokumentwissen
: Bezeichnet Informationen, die in der Struktur des Dokumentes abgelegt sind und dem Indexierungssystem zur korrekten Bildung von Lokationsbereichen mitgeteilt werden müssen. *Siehe Abschnitt 4.2.2.*

Hierarchieebene
: Bezeichnet eine einzelne Ebene eines hierarchisch untergliederten Stichworts. Das Stichwort (Suche:binäre) besteht aus den Hierarchieebenen 'Suche' und 'binäre'.

Index
: Bezeichnet ein Stichwortverzeichnis im üblichen Sinne. Es besteht aus einer Liste von sortierten Indexeinträgen.

Indexeintrag
: Besteht aus einem Stichwort und einer Menge von Lokationsreferenzen. *Siehe auch Abbildung 9.1.*

Kategorieattribut
: Lokationsreferenzattribut, welches zu Kategorisierungszwecken verwendet wird und vom Textsatzsystem zur unterschiedlichen optischen Hervorhebung von Lokationsreferenzen verwendet werden kann. *Siehe auch Abschnitt 4.1.7.*

Komponententyp
: Bezeichnet den Typ einer Strukturkomponente.

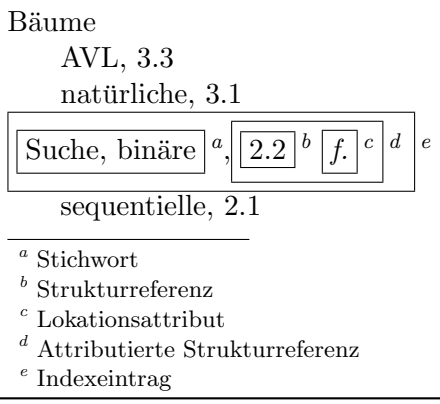


Abbildung 9.1: Index mit Bezeichnung der Komponenten eines Indexeintrags

Lokation
: Strukturelles Objekt bzw. Element eines Textes. Beispiele für solche Lokationen sind *Seiten*, *Kapitel*, *Abschnitte*, *Unterabschnitte*, *Anhänge* etc. Jede Lokation kann in einem Dokument nur ein einziges Mal auftreten und muß eindeutig auffindbar sein.

Lokationsreferenz
: Eindeutiger Bezeichner, der einer Lokation zugeordnet ist. Die Lokationsreferenz 'Seite 51' verweist auf eindeutige Weise auf bestimmtes Objekt des Dokuments, eben die 51. Seite.

Permutierter Index
: Enhält zu jedem hierarchisch gegliederten Stichwort wie `(Suche:binäre)` auch Permutationen der Stichworthierarchien wie z.B. `(binäre:Suche)`. *Siehe dazu auch Abschnitt 3.4.2*.

Referenz ist gleichbedeutend mit *Lokationsreferenz*.

Sortieralphabet
: Definiert ein Alphabet von Worten über einem Basisalphabet. Ein solches Alphabet bestimmt die Sortierordnung innerhalb von Strukturkomponenten einer Lokationsreferenz. *Siehe auch Definition 4.1.5, S. 13.*

Stichwort
: Begriff, auf dessen Vorkommen innerhalb eines Textes verwiesen werden soll. Ein strukturierter Begriff kann sich aus mehreren Ebenen zusammensetzen. Im Beispiel in Abbildung 9.1 haben wir die Stichwörter `(Bäume:AVL)`, `(Bäume:natürliche)` und `(Suche:binäre)`. Wir verstehen hier als Stichwort alle zusammengehörenden Ebenen eines Begriffs.

Strukturkomponente
: Teilkomponente der Struktur einer Lokationsreferenz. Die Lokationsreferenz `([3].[1].[2])` besteht aus den Strukturkomponenten [3], [1] und [2].

Strukturreferenz
: Beschreibt die strukturelle Zusammensetzung einer Lokationsreferenz. Die

Struktur einer Referenz ergibt sich aus der Folge der Strukturkomponenten und der Trennzeichen zwischen den Ebenen. Die Lokation ‘Kapitel–1’ besteht aus der ersten Ebene ‘Kapitel’, dem Trennzeichen ‘–’ und der zweiten Ebene ‘1’. Wir notieren solche Strukturreferenzen als `([Kapitel]–[1])`. Weitere Beispiele sind der Abschnitt `([1].[2].[3])` oder die Seite `([23])`.

Literaturverzeichnis

[BDG+88] Daniel G. Bobrow, Linda G. DeMichiel, Richard P. Gabriel, Sonya E. Keene, Gregor Kiczales, and David A. Moon. *Common Lisp Object System Specification*. ACM Press, New York, NY 10036, USA, 1988.

[BGB91] *Bürgerliches Gesetzbuch*. Beck-Texte im dtv, 1991.

[BK88] J. L. Bentley and B. W. Kernighan. Tools for Printing Indexes. *Electronic Publishing Origination, Dissemination, and Design*, 1(1):3–18, April 1988.

[CH87] Pehong Chen and Michael A. Harrison. Automating index preparation. Technical Report 87/347, Computer Science Division, University of California, Berkeley, CA, USA, March 1987. This is an extended version of [CH88].

[CH88] Pehong Chen and Michael A. Harrison. Index Preparation and Processing. *Software–Practice and Experience*, 19(9):897–915, September 1988. The LaTeX text of this paper is included in the `makeindex` software distribution.

[Chi82] *The Chicago Manual of Style*, chapter 18. University of Chicago Press, thirteenth edition, 1982.

[GMS94] Michel Goossens, Frank Mittelbach, and Alexander Samarin. *The LaTeX Companion*. Addison-Wesley, 1994.

[IBM93] IBM. *AIX System Management Guide*, 1993.

[Ihr93] Thomas Ihringer. *Allgemeine Algebra*. Teubner Studienbücher, 1993.

[Kee88] Sonya E. Keene. *Object–Oriented Programming in Common Lisp–A Programmers Guide to CLOS*. Addison-Wesley, 1988.

[Ker92] Karl Kerényi. *Die Mythologie der Griechen*. dtv Taschenbuch Verlag, 1992.

[Knu84] Donald E. Knuth. *The TeXbook*. Addison-Wesley, 1984.

[Kop94] Helmut Kopka. *LaTeX Band 1 Einführung*. Addison-Wesley, 1994.

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

[LL93] David Alex Lamb and Margaret Anne Lamb. Separation of concerns for indexing. *Electronic Publishing Origination, Dissemination, and Design*, 6(1):23–34, March 1993.

[Mic94] Microsoft. WinWord 6.0 Handbuch, 1994.

[Ram94] Norman Ramsey. Literate-programming can be simple and extensible. Technical report, Department of Computer Science, Princeton University, November 1994.

[Sch91] Joachim Schrod. An International Version of MakeIndex. *Cahiers GUTenberg*, 10(10–11):81–90, September 1991.

[Sch93] Randal L. Schwartz. *Learning Perl*. O'Reilly & Associates, Inc., O'Reilly & Associates, Inc., 1993.

[SH91] Joachim Schrod and Gabor Herr. *MakeIndex Version 3.0*, August 1991.

[Sta94] StarDivision GmbH. StarWriter 2.0 für Windows Handbuch, 1994.

[Ste84] Guy L. Steele Jr. *Common Lisp–The Language*. Digital Press, 12 Crosby Drive, Bedford, MA 01730, USA, 1984.

[Ste90] Guy L. Steele Jr. *Common Lisp–The Language*. Digital Press and Prentice-Hall, 12 Crosby Drive, Bedford, MA 01730, USA and Englewood Cliffs, NJ 07632, USA, second edition, 1990.

[Uni91] The Unicode Consortium. *The Unicode Standard: Worldwide Character Encoding. Version 1.0. Volumes 1 and 2.* Addison-Wesley, 1991.

[WS92] Larry Wall and Randal L. Schwartz. *Programming Perl*. O'Reilly & Associates, Inc., 981 Chestnut Street, Newton, MA 02164, USA, 1992. The authoritative guide to `perl`–the programming language for any serious UNIX users.