Technische Universität München
Institut für Informatik

Diplomarbeit

Eine TEI/SGML-Edition der textkritischen Ausgabe von James Joyces *Ulysses*

Tobias Rischer
Dachauer Str. 147
80335 München

rischer@informatik.tu-muenchen.de

Technische Universität München
Institut für Informatik

Diplomarbeit

Eine TEI/SGML-Edition der textkritischen Ausgabe von James Joyces *Ulysses*

Tobias Rischer
Dachauer Str. 147
80335 München

rischer@informatik.tu-muenchen.de

Aufgabenstellerin:	*Prof. Anne Brüggemann-Klein* (TUM)
Betreuer:	*Prof. Hans-Walter Gabler* (LMU)
Abgabedatum:	*15. Februar 1997*

Ich versichere hiermit, daß ich diese Arbeit selbständig verfaßt und nur die angegebenen Quellen und Hilfsmittel benutzt habe.

München,

Abstract

Diese Arbeit beschreibt ein umfangreiches Konvertierungsprojekt. Eine komplexe textkritische Edition von James Joyces *Ulysses* stand außer als Buch auch in Form der Dateien zur Verfügung, nach denen die gedruckte Ausgabe mit dem TUSTEP-System hergestellt worden ist. Mein Ziel war die Erzeugung eines SGML-Dokuments aus diesen Dateien.
Im ersten Teil der Arbeit leiste ich eine gründliche Analyse der dafür relevanten Informationsstrukturen sowohl im eigentlichen Text wie im editorischen Metatext der Buchausgabe.
Auf dieser Grundlage und auf der Basis der *Document Type Definition*, die von der *Text Encoding Initiative* für die Erfassung literarischer Texte entworfen wurde, beschreibe ich im zweiten Teil das von mir erstellte SGML-Modell, in dem die gefundenen Strukturen sinnvoll ausgedrückt werden können.
Die Durchführung der automatisierten Konvertierung aus den TUSTEP-Dateien in das erstellte SGML-Format und die dafür geschriebenen Programme stelle ich im dritten Teil der Arbeit dar.

Nach einem halben Jahr intensiver Arbeit an einem Thema, das bis zum Schluß seinen Reiz nicht verloren hat, und nach fünf Jahren Studium in zwei wunderschönen Fächern möchte ich mich bedanken:

– bei meinen 'Diplomeltern', Frau Brüggemann-Klein und Herrn Gabler, für ihre Freundlichkeit, Offenheit und die großzügige Hilfsbreitschaft, mit der sie diese Arbeit ermöglicht und unterstützt haben;

– bei Klaus Bartenschlager und Klaus Prahl, die meine Lehrlingsjahre in Anglistik und Informatik als Lehrer und Menschen geprägt und bereichert haben;

– bei der Universität von Tromsø, an der ich als Gaststudent das schönste und intensivste Jahr meines Studiums verbracht habe.

Inhaltsverzeichnis

Anhang

1. Die Aufgabenstellung der Arbeit

James Joyces 1922 erschienener Roman *Ulysses* ist ein nicht nur inhaltlich sondern auch formal sehr komplexer Text: neben in gewöhnliche Absätze gegliederter Erzählung enthält er unter anderem eingeschobene Gedichte und Briefe, eine bibliographische Liste und eine tabellarische Übersicht über Ausgaben und Einnahmen der Hauptfigur; das siebte Kapitel ist durch Zeitungsüberschriften untergliedert, das siebzehnte ist als Folge von Fragen und Antworten gestaltet, und das fünfzehnte ist durchgängig als Drama geschrieben.

Daneben hat der Text auch eine komplexe Entstehungs- und Druckgeschichte, und die Stufen, in denen Joyce seinen Roman schrieb, erweiterte und redigierte, während sich gleichzeitig bei Abschriften und im Satz Fehler einschlichen, sind zwar interessant zu verfolgen, aber nicht einfach zu überblicken.

Die 1984 erschienene Ausgabe *Ulysses. A Critical and Synoptic Edition* von Hans Walter Gabler ist ein gelungener Versuch, die stufenweise Entstehung des Textes optisch zu veranschaulichen, einen möglichst fehlerfreien Text des Romans zu präsentieren und die Abweichungen in erhaltenen Vorstufen und wichtigen gedruckten Ausgaben zu dokumentieren und zu erklären. Diese Ausgabe kann ohne weiteres als ein gedruckter Hypertext bezeichnet werden: mit Hilfe von Symbolen im Text, Fußnoten, längeren Anmerkungen im Anhang, Tabellen und Variantenlisten, sowie zwei parallelen Textdarstellungen mit jeweils eigenem Referenzsystem gehen die drei Bände an die Grenze dessen, was im gedruckten Buch zu erreichen und für Leser nachzuvollziehen ist. Dabei demonstrieren sie auf eindrucksvolle Weise, wie klar und kompakt auch eine hochkomplizierte Dokumentarchitektur im Druck darstellbar ist.

Die Herstellung solcher, sogenannter *historisch-kritischer* Ausgaben in elektronischer Form bekommt, eben wegen ihrer komplexen Struktur und der Schwierigkeit ihrer typographischen Wiedergabe, immer größere Bedeutung. Auch für andere philologische Zwecke kann es interessant sein, einen literarischen Text in elektronischer Form zur Verfügung zu haben (z.B. für automatisierte linguistische Untersuchungen, zur Konkordanzerstellung oder als leichteren Zugang zu komplex strukturierten Texten). Die *Text Encoding Initiative* hat deswegen eine umfangreiche SGML *Document Type Definition* erstellt, mit der Texte aus dem Bereich der Geisteswissenschaften, insbesondere auch Transkriptionen von Manuskripten und literarische Texte mit textkritischem Apparat, adäquat repräsentiert werden können.

Das Ziel meiner Diplomarbeit war es, den gedruckten Text von Hans Walter Gablers *Ulysses*-Ausgabe in einem TEI/SGML-Dokument darzustellen, die in der Buchform vorhandenen Strukturen möglichst sinnvoll abzubilden und dabei gleichzeitig die im TEI/SGML-Modell vorgesehenen Strukturen dem Standard entsprechend zu benutzen. Das entstandene SGML-Dokument könnte nun zum Beispiel als Datenbasis in einer interaktiven *Ulysses*-Edition verwendet werden, aber auch andere Anwendungen sind denkbar.

Die gedruckte Ausgabe wurde vom ersten Textvergleich bis zur Satzvorbereitung mit dem Softwarepaket TUSTEP der Universität Tübingen hergestellt. Als Ausgangsmaterial standen mir neben der Buchausgabe Dateien zur Verfügung, die aus der Druckvorbereitung des *Ulysses* 'übriggeblieben' waren. Die Symbole, Absatzformate, Fußnoten, Schriftarten, Seiten- und Zeilennummern *et cetera* der gedruckten Ausgabe sind in diesen TUSTEP-Dateien durch eine Vielzahl unterschiedlicher Kontrollsequenzen codiert.

Um aus diesen Dateien eine SGML-Edition herzustellen, war eine aufwendige Konvertierung nötig; die dafür erforderlichen Verfahren und Programme waren auszuwählen beziehungsweise zu schreiben.

Die Bearbeitung der Diplomarbeit bestand daher in etwa aus folgenden Schritten:

- Einarbeitung in TEI/SGML und Grundlagen der Editionsphilologie.
- Dokumentanalyse der *Ulysses*-Ausgabe: welche Strukturen liegen vor, auf welche Besonderheiten ist zu achten?
- Identifizieren der TUSTEP-Codes in den Quelldateien (es lag praktisch keine Dokumentation vor).

- Auswahl bzw. Entwurf geeigneter SGML-Modelle für die vorgefundenen Strukturen auf der Basis der TEI DTD.
- Erstellen von Programmen, um die automatisch konvertierbaren Elemente zu verarbeiten.
- Kontrolle und Handbearbeitung der generierten SGML-Dateien.

Das Ergebnis der Dokumentanalyse macht den ersten Teil meiner Arbeit aus; im zweiten Teil folgt die Beschreibung der korrespondierenden SGML-Strukturen; im dritten Teil beschreibe ich das Konvertierungsverfahren und die dafür verwendeten Programme.

Im Verlauf des Konvertierungsprojekts bin ich immer wieder auf Probleme gestoßen, die prinzipielle Fragen der Editionswissenschaft, der SGML-Codierung oder der Programmierung berühren. Ich bin diesen Fragen nicht ausgewichen, aber was ich vorlege, ist in erster Linie eine praktische Arbeit. Darum werden prinzipielle Fragen in ihrem pragmatischen Kontext aufgegriffen und nur soweit verfolgt, wie sie für die Umsetzung meiner praktischen Ziele wichtig waren.

⋆

Noch eine technische Anmerkung: Alle in der Arbeit zitierten oder angesprochenen Bücher und Aufsätze sind im Literaturverzeichnis am Ende der Arbeit enthalten; ich zitiere deswegen im Text ohne große Formalität, aber immer so, daß der betreffende Text eindeutig identifiziert ist.

Um Ballast zu vermeiden, habe ich die sehr häufigen Zitate aus dem *Ulysses* durch Kurzangaben nach den beiden Referenzsystem der Gabler-Edition belegt. Es steht '*c15 ℓ285*' für die Zeile 285 im fünfzehnten Kapitel des Lesetextes und '*p1476 ℓ12–13*' für die gleiche Textstelle in den Zeilen 12–13 auf der Seite 1476 des synoptischen Textes.

Dokumentanalyse

2. Zur systematischen Dokumentanalyse

Die folgenden Abschnitte führen eine Dokumentanalyse des gedruckten *Ulysses* durch und bilden die Grundlage des Entwurfs der elektronischen Edition; daran erst kann sich die Durchführung der automatischen und manuellen Konvertierung anschließen – in der Theorie.

In der Praxis fanden diese drei Schritte von Anfang bis Ende gleichzeitig statt, und die vorliegende elektronische Edition ist das Ergebnis eines Wachstumsprozesses aller drei Teile. Viele Details der Dokumentanalyse kamen erst während der Konvertierung zum Vorschein, viele Konvertierungsansätze erwiesen erst bei der Durchführung ihre Schwächen, die auch auf die Struktur der elektronischen Edition zurückschlugen, und im Verlauf einer intensiven Beschäftigung mit den Strukturen des Textes und den Möglichkeiten und Zielen einer SGML-Ausgabe haben sich auch meine Zielvorstellungen für eine elektronische Edition mehrfach geändert.

Auch wenn ich in einzelnen Fällen durch größere Erfahrung und sorgfältigeren Entwurf sicher Sackgassen vermieden hätte und das Ergebnis wertvoller geworden wäre – im Wesentlichen halte ich diesen Prozeß für natürlich und unvermeidbar. Es wäre für den SGML-*Ulysses* und wohl auch für andere derartige Unternehmungen wünschenswert, wenn man an dieser Stelle den langen Atem hätte, noch einmal mit der gewonnenen Erfahrung von vorne anzufangen, um die erkannten Fehler von vorneherein zu umgehen und ein konzeptuell geschlossenes und überzeugendes Gesamtprodukt herzustellen. Diese Erkenntnis ist weder neu noch auf elektronische Editionen beschränkt. Frederic Brooks schreibt in seinem 1975 erschienenen Klassiker *The mythical man-month*:

> In most projects, the first system built is barely usable. It may be too slow, too big, awkward to use, or all three. There is no alternative but to start again, smarting but smarter, and build a redesigned version in which these problems are solved ... Where a new system concept or new technology is used, one has to build a system to throw away, for even the best planning is not so omniscient as to get it right the first time.
>
> (Brooks, S. 116)

Aus zeitlichen Gründen habe ich mich nicht an den Ratschlag halten können, den Brooks noch auf der gleichen Seite gibt, den ich aus meiner Erfahrung aber durchaus empfehlen kann: "plan to throw one away; you will, anyhow."

⋆

Die systematische Dokumentanalyse der folgenden Abschnitte ist also, was meinen tatsächlichen Arbeitsablauf betrifft, eine Fiktion; sie ist aber gleichzeitig eines der Ergebnisse meiner Arbeit, und möglicherweise eine Grundlage für weitere Arbeit an der elektronischen *Ulysses*-Ausgabe.

Ich gehe so vor, daß ich nach einem Überblick über prinzipielle Grundlagen und den Aufbau des Buches zunächst die Elemente des Lesetextes, und damit des eigentlichen, von Joyce verfaßten Textes des *Ulysses* beschreibe, von der Buchstabenebene zu den größeren Strukturen hin. Daran schließt sich die Beschreibung des Metatextes an, mit dem der Herausgeber die Ausgabe angereichert hat; das sind im Wesentlichen die synoptische Darstellung der Textentstehung, die Fußnoten mit dem textkritischen Apparat (Emendationen) sowie die Liste der Textabweichungen von früheren Druckausgaben (Historische Kollation). Die

Behandlung der elektronischen Edition im zweiten Teil der Diplomarbeit habe ich ähnlich strukturiert und gebe immer wieder Querverweise, damit die Umsetzung von der einen in die andere Form besser zu verfolgen ist. Die zahlreichen Beispiele machen den Text hoffentlich auch für nicht eingearbeitete Leser nachvollziehbar.

3. Editorische Grundkonzepte der 'Gabler-Edition'

Die Arbeit des Herausgebers eines wissenschaftlich editierten Textes ist durch zwei Pole bestimmt: zum einen sind die ihm überlieferten Textträger (die sogenannten Zeugen) durch materielle Beschädigung und fehlerhafte Überlieferung korrumpiert. In dieser Hinsicht ist es seine Aufgabe, durch Textvergleich, Kenntnis der Überlieferungsgeschichte und gegenenfalls auch durch einen *educated guess* eine möglichst bereinigte, ursprüngliche Form des Textes wiederherzustellen (unter der Hypothese, daß es eine solche ideale Textgestalt einmal gab, zumindest im Kopf des Autors). Der andere Aspekt der editorischen Tätigkeit ist es, den Entstehungs- und Bearbeitungsprozeß des Textes durch den Autor, oder auch in der weiteren Überlieferung, zu dokumentieren. Dem interessierten Leser gibt dies Einblick in die Arbeitsweise des Autors, zusätzliche Perspektiven auf die Strukturen und Bedeutungen des Textes, und nicht zuletzt Anschauungsmaterial dafür, daß ein literarischer Text nicht nur ein fixierter, zeitloser Monolith ist, sondern auch ein verzweigter Prozeß, und der Erstdruck (oder jede andere Fassung) nur eine bestimmte Momentaufnahme davon. Das Ergebnis dieser Arbeit ist die 'historisch-kritische Ausgabe'.[1]

Unterschiedliche Schulen der Editionswissenschaft haben unterschiedliche Positionen in diesem Spannungsfeld als Zielsetzung bestimmt, meist auch aus der Prägung durch unterschiedliche editorische Aufgaben heraus. Die Edition von griechischen oder lateinischen Klassikern, wo normalerweise keinerlei Originalquellen sondern stattdessen ganze Stammbäume fehlerhafter und umgestalteter Abschriften vorliegen, stellt andere Aufgaben als die Edition Hölderlins, wo es eher um die Auswahl aus unterschiedlichen Fassungen autorisierter Texte oder die sinnvolle Darstellung der im Manuskript erkennbaren Entstehungsgeschichte geht. Zudem gehen die Meinungen darüber, welche Informationen und damit welche Varianten in einem Text als wesentlich gelten, weit auseinander.

Es läßt sich als editorische Aufgabe aber auf jeden Fall summarisch vereinfachend festhalten, daß zum einen schwer zugängliches Material für ein breiteres Publikum dokumentiert wird, und zum anderen ein geschulter Spezialist dieses Material bearbeitet, um die Informationsfülle zu ordnen und zu bewerten, und schließlich den Lesern in Form eines edierten Textes oder/und einer übersichtlichen Darstellung der Textvarianten zur Verfügung zu stellen. Diese Abstraktion ist einerseits eine große Dienstleistung am Leser, der sich bei seiner Lektüre auf die jahrelange Vorarbeit des Herausgebers stützen kann; andererseits ist es aber auch eine Erfordernis des Mediums Buch, das auf eine kompakte Darstellung angewiesen ist.

Die technischen Möglichkeiten und auch die Preisentwicklung der Computertechnik haben diese Situation radikal verändert. Es ist heute einfacher und billiger, Originaldokumente als hochauflösende Grafiken zu digitalisieren, sie eventuell noch durch Subunternehmer in Billiglohnländern abschreiben zu lassen und das Ergebnis auf CD oder im Netzwerk zur Verfügung zu stellen, als ein Team von Philologen jahrelang mit der Herstellung einer konventionellen historisch-kritischen Ausgabe zu beschäftigen. Zu diesem ökonomischen

[1] Die Begriffe 'historisch-kritisch', 'textkritisch' oder nur 'kritisch' werden oft unscharf gebraucht. Ich bin hier der natürlichen Herleitung von 'historisch-kritisch' gefolgt, wie sie z.B. bei Herbert Kraft, *Editionsphilologie*, steht.

Aspekt treten auch theoretische Bewertungen: Während viele Philologen eine solche 'Archiv-Edition' nur als Vorarbeit zur eigentlichen Editionsarbeit sehen würden,[2] wird es von anderen Wissenschaftlern auch theoretisch abgelehnt, die lebendige textuelle Vielfalt und die historische Realität für die Leser schon im Voraus zu bereinigen und dabei zu verflachen und zu simplifizieren.[3] Sie begrüßen die Offenheit des elektronischen Mediums, durch die es den Lesern überlassen werden kann, ob sie die (gescannten) Manuskripte oder verschiedene mögliche daraus konstituierte Fassungen sehen wollen. Eine aus solchen Grundsätzen entstandene elektronische Edition ist beispielsweise das *Rosetti Archive*, ein Editionsprojekt, in dem digitalisierte Originaldokumente mit Transkriptionen und kritischer Information eng verknüpft werden sollen.[4]

⋆

Die *Ulysses*-Ausgabe von Hans-Walter Gabler und seinen Mitarbeitern befindet sich mitten im oben dargestellten Spannungsfeld.[5] Das Ziel der Ausgabe war es, den Entstehungsprozeß des Textes noch einmal von der ersten zusammenhängenden Manuskriptfassung bis zur ersten gedruckten Ausgabe nachzuvollziehen, ohne die gleichzeitig beim Abschreiben, Korrigieren und Setzen aufgetretenen Fehler mitaufzunehmen (und auch Joyce selbst hat beim Abschreiben und Korrigieren Fehler gemacht). Das Ergebnis dieser Arbeit ist ein durch die Herausgeber idealisierter Text der Erstausgabe, 'wie er eigentlich hätte sein sollen', aber so vor 1984 nie existiert hat.

Zusätzlich hielten die Herausgeber den stufenweisen Entstehungsprozeß des Textes unter Joyces Bearbeitungen, wie sie ihn anhand der erhaltenen Quellen erschließen konnten, für so interessant, daß sie sich entschlossen, die Entstehungsstufen durch typographische Kennzeichnung im Text zu dokumentieren. Das editorische Ziel war eine vollständige Dokumentation jedes von Joyce stammenden Textelements, einschließlich der nur indirekt zu erschließenden und der später verworfenen oder verlorengegangenen Elemente. Die Darstellung von diesem auf zahlreiche Dokumente verteilten 'Text als Entstehungsprozeß', von Gabler im Nachwort der Ausgabe als *continuous manuscript text* bezeichnet, ist das eigentliche Ergebnis der editorischen Textarbeit.

Das Ziel der Edition ist hingegen nicht eine vollständige Dokumentation aller erhaltenen Textzeugen. Auch wenn die Schichten der synoptischen Darstellung mit den entsprechenden Dokumenten eng zusammenhängen, dürfen sie nicht als Transkription dieser Dokumente mißverstanden werden. Auch die Emendationen dienen nicht einer vollständigen Rekonstruktion der zitierten Zeugen, sondern liefern lediglich eine Beschreibung der relevanten Zeugenlage an den Stellen, wo der Editionstext in seiner Konstruktion des *continuous manuscript text* aus der Dokumentsituation heraus begründet werden muß.

---

[2] Klaus Kanzog hat diesen Begriff für eine rein dokumentierende, eben nicht 'kritische' Edition in den *Prolegomena zu einer historisch-kritischen Ausgabe der Werke Heinrich von Kleists* eingeführt. Der Unterschied wird, ohne einen Buchstaben zu lesen, augenfällig, wenn man die fünfzehn schweren Bände der reproduzierten Textzeugen zum *Ulysses* neben die drei vergleichsweise schlanken Bändchen der Gabler-Edition stellt.

[3] John Lavagnino gibt diesen Standpunkt, den er 'localism' im Gegensatz zu 'idealism' nennt, in seinem Aufsatz *Completeness and Adequacy in Text Encoding* wieder.

[4] Siehe dazu Jerome McGanns Aufsatz *The Rosetti Archive and Image-Based Electronic Editing*.

[5] Die folgenden Erklärungen basieren vor allem auf dem Nachwort zum *Ulysses*, auf S. 1859–1907.

4. Gesamtaufbau der gedruckten Ausgabe

Der Vorspann der *Ulysses*-Ausgabe[6] besteht aus dem Inhaltsverzeichnis (S. v) einem Vorwort (S. vi–vii), einem Vorwort zur zweiten Auflage mit einer Liste von Korrekturen gegenüber der Ausgabe von 1984 (S. vii–ix) und einer Aufschlüsselung der verwendeten Symbole (S. x–xiii).

Den Hauptteil bildet der Text des Romans (S. 1–1727), der sich in drei *parts* und insgesamt achtzehn *episodes* gliedert (vergleiche die Übersicht auf S. 21); "Part I" besteht aus den ersten drei Episoden, "Part II" aus den mittleren zwölf, und "Part III" aus den letzten drei. Die Episoden (ich spreche im Folgenden meist von 'Kapiteln') werden traditionell mit Titeln bezeichnet, die der homerischen Odyssee entnommen sind. Diese Arbeitstitel gehen zwar auf Joyce zurück, sind von ihm aber für das fertige Buch nicht vorgesehen gewesen.

Der Romantext ist zweimal parallel abgedruckt. Auf den rechten (ungeraden) Seiten ist der vom Herausgeber erarbeitete Lesetext als normaler Buchtext abgedruckt; am Rand befinden sich Zeilennummern, die pro Kapitel durchgezählt sind. Dieses Referenzsystem wird auch in reinen Leseausgaben die von dieser Ausgabe abgeleitet sind, eingehalten und eignet sich daher sehr gut für das Zitieren von Textstellen.[7]

In einem Anhang befindet sich die sogenannte historische Kollation, in der alle Textunterschiede zwischen dem hier etablierten Lesetext und einigen anderen Ausgaben des *Ulysses* erfaßt sind (nämlich den eigenständigen zu Joyces Lebzeiten erschienen Ausgaben, auf die er zumindest theoretisch noch Einfluß nehmen konnte, und der langjährigen Standardausgabe von *Random House*, 1961).

Auf den linken (geraden) Seiten befindet sich der gleiche Text noch einmal, als synoptische Darstellung der Textgeschichte: hier ist der editorisch rekonstruierte Entstehungsprozeß von der ersten Manuskriptfassung bis zum Erstdruck graphisch so eingearbeitet, daß man Joyces Arbeitsprozeß und das Wachsen des Textes mitverfolgen kann. Die einzelnen Phasen der Überarbeitung wurden dazu in Stufen (*levels*) gegliedert, die mit Buchstaben- und Zahlenkürzeln bezeichnet werden. Zusätzliche Symbole geben weitere Informationen über Einzelheiten der Textentstehung (Streichungen, Auslassungen und ähnliches).

In Fußnoten zu dieser Textform befinden sich die Emendationen; hier werden die für die Konstruktion des *continuous manuscript text* relevanten Details der Textüberlieferung in den erhaltenen Dokumenten bis zum Erstdruck aufgeschlüsselt. Dazu werden etablierte und neu geschaffene typographische Konventionen verwendet, um möglichst präzise Information auf engem Raum zu vermitteln. Dort wo Sachverhalte ausführlicher erläutert werden müssen, steht in der Fußnote ein Verweissymbol auf die *textual notes*, die ausführlichen Diskussionen einzelner Probleme im Anhang. Das Referenzsystem für die linken Seiten ist eine gewöhnliche Seiten-Zeilen-Zählung; der Zeilenfall der linken Seiten unterscheidet sich wegen des zusätzlichen Materials von dem der rechten Seiten.

Neben den *textual notes* (S. 1729–1753) und der Historischen Kollation (S. 1755–1854) enthält der Anhang noch eine Bibliographie der bisher erschienenen *Ulysses*-Editionen und ausgewählter wissenschaftlicher Literatur zum Text und zur Textentstehung (S. 1855–1858), ein Nachwort, in dem die Entstehungsgeschichte des Textes ausführlich beschrieben wird und die sich daraus ergebenden Editionsprinzipien der Ausgabe erläutert werden (S. 1859–1907), den Dank des Herausgebers an Helfer und Mitarbeiter (S. 1908–1911), eine Tabelle zur Datierung der Fahnenkorrekturen von Joyce (S. 1914–1915) und eine Tabelle zur (sehr komplizierten) Reihenfolge der Maschinenabschrift der siebzehnten Episode (S. 1916–1919).

---

[6] Ich habe für meine Arbeit den korrigierten Nachdruck von 1986 verwendet; die Unterschiede zur ersten Ausgabe von 1984 sind aber geringfügig.

[7] Zum Beispiel die *Student's Edition* der *Penguin Books* von 1986. Merkwürdigerweise gibt es auch Ausgaben des Gablerschen Lesetextes, die den Zeilenumbruch zwar sorgfältig einhalten, aber keine Zeilennummern dazu drucken, was den Ansatz natürlich entwertet.

5. Elemente des Lesetextes

In diesem Abschnitt stelle ich die Strukturen des *Ulysses*-Textes selbst dar, wie man sie auf den rechten Seiten der Buchausgabe (oder in einer reinen Leseausgabe) vorfindet. Dabei beginne ich auf der Ebene der einzelnen Zeichen, beschreibe dann wiederkehrende Strukturen, wie Prosaabsätze und Lyrik, sowie die selteneren Spezialfälle, wie Briefe; schließlich werden die besonderen kapitelspezifischen Prosaformen dargestellt, wie die dramatische Form der Circe-Episode. Ich möchte betonen, daß die typographischen Konventionen der Gabler-Edition, die hier ziemlich detailliert analysiert werden, nicht das Ergebnis willkürlicher Entscheidungen des Herausgebers oder seines Verlegers sind, sondern sehr sorgfältig aus dem Schriftbild des Manuskripts und der von Joyce gebilligten Form des Erstdrucks hergeleitet wurden.

Die folgende Tabelle gibt zunächst einen Überblick über die achtzehn Kapitel des Buchs, ihren Umfang und die darin verwendeten Textformen.

1: *Telemach*	744 Zeilen	Prosa, Lyrik
2: *Nestor*	449 Zeilen	Prosa, Lyrik
3: *Proteus*	505 Zeilen	Prosa, Lyrik
4: *Calypso*	551 Zeilen	Prosa, Lyrik, Brief
5: *Lotus Eaters*	572 Zeilen	Prosa, Postadresse, Prosadisplay, Brief
6: *Hades*	1033 Zeilen	Prosa, Lyrik
7: *Aeolus*	1075 Zeilen	Durch Schlagzeilen gegliedert: Prosa, Lyrik
8: *Lestrygonians*	1193 Zeilen	Prosa, Lyrik, Prosadisplay
9: *Scylla and Charybdis*	1225 Zeilen	Prosa, Lyrik, Prosadisplay, Gregorianischer Cantus, Blankvers, Drama, Theaterankündigung, Besetzungsliste
10: *Wandering Rocks*	1282 Zeilen	Durch Sternchen in Abschnitte unterteilte Prosa, Lyrik
11: *Sirens*	1294 Zeilen	Liste von Kurzabsätzen als 'Präludium', Prosa, Lyrik, Postadresse, Prosazeilen mit synchronisierten Wortabständen
12: *Cyclops*	1918 Zeilen	Prosa von zwei inhaltlich deutlich unterscheidbaren Erzählern, mit Unterbrechungen vorgelesener Brief, Lyrik
13: *Nausikaa*	1306 Zeilen	Prosa, Lyrik
14: *Oxen of the Sun*	1591 Zeilen	Prosa, Lyrik
15: *Circe*	4967 Zeilen	Drama, Lyrik und lyrikähnliche Formen
16: *Eumaeus*	1894 Zeilen	Prosa, Lyrik
17: *Ithaca*	2332 Zeilen	Prosa als Abfolge von Frage und Antwort, Lyrik, Lied (mit Noten), Bücherverzeichnis, Bilanz (Tabelle), Akrostichon
18: *Penelope*	1609 Zeilen	Prosa (extrem lange Absätze, keine Interpunktion)

5.1. *Der Zeichenvorrat*

Der Text wurde für den konventionellen Buchsatz geschrieben und enthält auf der Zeichenebene nur wenige ungewöhnlichen graphischen Phänomene. Da aus verschiedenen Sprachen zitiert (und parodiert) wird, werden eine Reihe von diakritischen Zeichen verwendet, auch andere typographische Alltäglichkeiten jenseits des englischen Alphabets treten hier und da auf. Auffällige Sonderzeichen sind der große Punkt, der die letzte Frage im siebzehnten Kapitel beantwortet,[8] das altertümliche lange 's' (wie in "aſtray"), das in der Beschreibung eines Buches aus dem 18. Jahrhundert verwendet wird (*c17 ℓ1398–1407*) und die durchgestrichenen Buchstaben in 'symp~~h~~athy' und 'ne~~y~~phew' (*c18 ℓ730*), mit denen Molly Blooms Schreibfehler optisch markiert werden.

Die folgende Aufstellung ist eine Gesamtübersicht der über das englische Alphabet und normale Interpunktion hinausgehenden Zeichen im Lesetext.

Á à á ä ā Æ æ Ç ç É É é ê ~~h~~ î ǩ ñ ó ò ô

ö œ ř š ṡ ſ ú û ~~w~~ £ & % + / ∗ † • $^{c}/_{o}$ $^{a}/_{c}$

$^{1}/_{2}$ $^{1}/_{3}$ $^{1}/_{4}$ $^{1}/_{6}$ $^{2}/_{7}$ $^{5}/_{11}$ 50° 1^{st} 2^{nd} 3^{rd} 4^{th} $8^{s}/6^{d}$ $M^{c}Coy$

M'Coy H_2O SO_4 rec^{d} - – — [Malteserkreuz][9]

Es ist hervorzuheben, daß der Text zwar sowohl das linke wie das rechte einfache Anführungszeichen enthält, aber nicht in ihrer Funktion als Anführungszeichen. Das rechte Anführungszeichen taucht natürlich als Apostroph auf, das linke in irischen Namen wie "M'Coy".

Neben der Schreibweise "$^{c}/_{o}$" wird gelegentlich auch die Form "c/o" verwendet. Es ist nicht klar, ob diese Schreibweisen wirklich durchgängig semantisch unterschieden werden, allerdings wird ein solcher Unterschied in der Historischen Kollation zu *c5 ℓ63* als Variante verzeichnet. Allein aus Joyces Manuskript dürfte eine solche Differenzierung aber wohl kaum zu begründen sein.

Eine Problemstellung in allen Textausgaben ist die Behandlung von Bindestrichen am Zeilenende. Sie dienen einerseits zur Trennung von normalerweise zusammengeschriebenen Wörtern, wie "goodness", das dann aus typographischen Gründen zu "good-ness" wird; andererseits kann es auch passieren, daß Wörter grundsätzlich mit Bindestrich zusammengesetzt werden, wie "good-natured", und nun zufällig an dieser Stelle ein Zeilenwechsel erfolgt. Da der Gebrauch solcher Schreibweisen individuell verschieden sein und auch Nuancen der Bedeutung ausdrücken kann, unterscheiden viele Herausgeber zwischen Bindestrichen, die beim Satz ihrer Ausgabe eingefügt wurden und solchen, die im Quelltext standen. Oft dokumentieren sie auch, wie sie mit solchen Zweifelsfällen, die bereits im Quelltext auftreten, umgegangen sind.

In dieser Hinsicht ist die *Ulysses*-Ausgabe etwas ungenau; sie vermeidet die Einführung neuer Bindestriche und die Positionierung von festen Bindestrichen am Zeilenende nach Möglichkeit, und wo Wortrennungen auftreten, sind sie mit normalem Menschenverstand als solche zu erkennen, zumal der Text ja immer zweimal und meist mit unterschiedlichem Zeilenfall abgedruckt ist; eine explizite typographische Unterscheidung oder eine Auflistung dieser Fälle gibt es aber nicht. (Die in Frage kommenden Stellen sind, soweit mir bekannt,

---

[8] Dieser Punkt belegt, daß Joyce seinen Text auch auf typographischer Ebene mitgestaltet hat. "Ce point doit être plus visible" wies er den Setzer schriftlich an (zitiert nach der Emendation zu *p1632 ℓ22*)

[9] Das Malteserkreuz ist ein aufrecht stehendes Kreuz aus zwei gleichlangen, am Ende breiteren Balken, ähnlich einem Pluszeichen mit Serifen. Es ist in meinen TEX-Fonts leider nicht enthalten.

c3 ℓ52 (li. Seite), *c7 ℓ874*, *c12 ℓ567*, *c15 ℓ643*, *c15 ℓ3809*, *c15 ℓ3812*, *c15 ℓ4353* und *c15 ℓ4355* (li. Seite)).

5.2. *Prosa*

Der Text ist in durch Einzug deutlich markierte Prosaabsätze gegliedert. Direkte Figurenrede ist, wie überall im Buch, nicht durch Anführungszeichen gekennzeichnet, sondern in eigenen Absätzen wiedergegeben, die ohne Einrückung und mit einem Geviertstrich (*dialogue dash*) beginnen. Außer der wörtlichen Rede selbst enthalten diese Absätze auch *inquit*-Formeln oder nähere Beschreibungen des Sprechers. Zum Beispiel:

> Buck Mulligan peeped an instant under the mirror and then covered the bowl smartly.
> —Back to barracks! he said sternly.
>
> (*c1 ℓ17–19*)

Der Absatz ist im englischen Sprachraum noch mehr als im Deutschen nicht nur die kleinste typographische Einheit unformatierten Prosatextes, sondern vor allem auch ein inhaltliches Strukturelement. Infolgedessen könnten solche Figurenreden, ebenso wie eingeschobene Lyrik und andere Displays zumindest an einigen Stellen als Teil des vorangehenden Absatzes aufgefaßt werden; das Kriterium dafür ist aber nicht typographisch zu definieren.

Nach rein formalen Kriterien bieten sich im Wesentlichen zwei Möglichkeiten: man kann Absätze als die kleinste Einheit frei fließend formatierten Blocksatzes definieren, dann bildet jede wörtliche Rede einen neuen, eigenen Absatz; oder man definiert einen Absatz als den Textabschnitt bis zum nächsten Absatzeinzug, dann gehören wörtliche Reden zum vorhergehenden Absatz. Die praktische Relevanz dieser Unterscheidung ist nicht allzu groß.

Fremdsprachliche Wendungen wie "*Introibo ad altare Dei*" (*c1 ℓ5*) sind im Druck durch Kursivschrift gekennzeichnet, ebenso literarische Zitate (*c1 ℓ567*), Wiedergabe der Worte Dritter (*c1 ℓ198f*), zitierte Titel von Büchern, Musik- oder Theaterstücken (*c1 ℓ258*) und ähnliches.

5.3. *Lyrik*

Wenn Lyrik im Text vorkommt, ist sie kursiv und (natürlicherweise) in einzelnen Zeilen gesetzt. Lyrik die Bestandteil des Erzählertextes ist (zum Beispiel *c1 ℓ260–262*), wird als in sich linksbündiger Block auf die längste Zeile zentriert. Lyrik die Bestandteil einer Figurenrede ist (zum Beispiel Buck Mulligans Lied ab *c1 ℓ584*), wird wie Prosarede ohne Einrückung links gesetzt und mit einem Geviertstrich eingeleitet.

Lyrik tritt in unterschiedlichen Formen auf; oft werden nur einzelne Zeilen anzitiert, aber gelegentlich folgen auch mehrere Strophen hintereinander. Als spezielle Form tritt im siebzehnten Kapitel ein Akrostichon auf, ein Gedicht, bei dem die Anfangsbuchstaben jeder Zeile hintereinander gelesen ein sinnvolles Wort ergeben; um das hervorzuheben, sind die Anfangsbuchstaben fett gesetzt (*c17 ℓ412–416*).

An zwei Stellen werden den Texten Noten beigegeben. Im neunten Kapitel ist die Melodie zu einem "Gloria in excelsis Deo" in spätmittelalterlicher Notation angegeben (*c9 ℓ500*); im siebzehnten Kapitel singt Stephen Dedalus ein Lied, dessen Melodie in modernen Noten handschriftlich wiedergegeben ist (*c17 ℓ808* und *c17 ℓ828*).

Eine weitere Sonderform tritt im siebten Kapitel auf, wo ein Vers nur bruchstückhaft zitiert wird, und die ausgelassenen Zeilenanfänge mit weit gesperrten Auslassungspunkten aufgefüllt sind:

> . . . . *la tua pace*
> . . . . *che parlar ti piace*
> *Mentre che il vento, come fa, si tace.*
>
> (*c7 ℓ717–719*)

Auch kurze Zitate aus Dramen in Blankvers (vor allem Shakespeare), die also ebenfalls eine Zeilenstruktur haben, und andere lyrikähnliche Formen kommen mehrfach vor.

Im neunten Kapitel wird ein Gespräch über das Verhältnis zwischen Shakespeare und seiner Frau von der Hauptfigur Stephen Dedalus auf die folgende Weise wahrgenommen und wiedergegeben:

> He faced their silence.
>
> To whom thus Eglington: You mean the will.
> But that has been explained, I believe, by jurists.
> She was entitled to her widow's dower
> At common law. His legal knowledge was great
> Our judges tell us.
> Him Satan fleers,
> Mocker:
> And therefore he left out her name
> From the first draft but he did not leave out
> The presents for his granddaughter, for his daughters,
> For his sister, for his old cronies in Stratford
> And in London. And therefore when he was urged,
> As I believe, to name her
> He left her his
> Secondbest
> Bed.
>
> *Punkt*.
>
> Leftherhis
> Secondbest
> Leftherhis
> Bestabed
> Secabest
> Leftabed.
>
> Woa!
>
> (*c9 ℓ684–700*)

Die literarische Form, auf die im ersten Teil angespielt wird, ist auch durch die Einrückungen, die die vorherige Zeile fortsetzen, charakterisiert. Diese Satzweise ist typisch für Versdramen, wenn innerhalb einer Zeile die Sprecher wechseln; ihre Funktion hier ist weniger klar, weil keine wechselnden Sprecher markiert sind. In den *Textual Notes* zu dieser Stelle wird die Heraushebung der Zeile "Him Satan fleers, | Mocker:" als eine Reflektion Stephens über die sie umgebende Rede Eglingtons gedeutet.

Während im gleichen Kapitel Zitate aus *Hamlet* kursiv gesetzt sind, ist dieser Abschnitt recte, was stärkere Nähe zur Prosa andeutet. Die beiden ausgerückt erscheinenden Zeilen sind, der Typographie nach, eigene Absätze ("*Punkt*" ist kursiv, weil es fremdsprachig ist).

Der zweite Teil ist aber nicht als unabhängiges Gedicht eingerückt, sondern folgt immer noch der Einrückung des ersten Teils (der, als Lyrik, auf seine längste Zeile zentriert ist). Dieser zweite Teil erscheint als in lyrische Zeilen gegliederter Bewußtseinsstrom und bewegt sich damit nahe der Grenze zur Prosa; an dieser Stelle scheint der Zeilenumbruch, der Prosa im typographischen Sinne zu Lyrik macht, weniger ein von der Umgebung deutlich getrenntes Gedicht zu markieren, als eine Methode zur Rhythmisierung der Sprache zu sein.

⋆

So interessant solche Textstellen, gerade auch durch ihr Zusammenspiel von graphischer Gestalt und Inhalt, für den Leser sind – sie lassen sich nicht ohne weiteres in einfache

Kategorien von Textfunktionen einordnen, und das Layout ist nicht mehr nur ein Mittel, eine klar definierbare Textfunktionen zu markieren.

5.4. *Prosadisplays*

Neben der Lyrik gibt es auch andere Texte, die zeilenorientiert dargestellt werden. Eine öfter wiederkehrende Textform ist der in der Außenwelt wahrgenommene oder erinnerte Reklametext:

What is home without
Plumtree's Potted Meat?
Incomplete.
With it an abode of bliss.

(c5 ℓ144–147)

Diese Texte unterscheiden sich typographisch von Lyrik dadurch, daß sie zeilenweise zentriert sind. Die Schrift ist normalerweise kursiv, wird aber für besondere Effekte variiert, zum Beispiel im folgenden Display aus dem siebzehnten, in Frage-Antwort-Sequenzen gegliederten Kapitel:

In what order of precedence, with what attendant ceremony was the exodus from the house of bondage to the wilderness of inhabitation effected?

Lighted Candle in Stick
borne by
BLOOM
Diaconal Hat on Ashplant
borne by
STEPHEN

(c17 ℓ1023–1028)

Auch hier ist es, wie in Grenzfällen der Lyrik, schwer zu sagen, welche Textfunktion durch dieses spezielle Layout markiert werden soll, so daß die Dokumentanalyse an solchen Stellen darauf beschränkt bleibt, das besondere Layout als solches festzustellen.

Die Hervorhebung von Personennamen durch Kapitälchen wird auch in den *dramatis personae* (siehe unten) und den Dramenteilen des *Ulysses* (siehe S. 30) verwendet; Joyce hat wohl beide Formen nur durch Unterstreichung gekennzeichnet und die Entscheidung der Setzer, die etablierten Konventionen folgt,[9] implizit gebilligt.

5.5. *Dramatis Personae*

Diese spezielle Form des Prosadisplays wird hier extra behandelt, weil sie, zur erfreulichen Abwechslung, eine klare Zuordnung von Layout zu Funktion bietet. Am Ende des neunten Kapitels entwirft Buck Mulligan im Scherz ein Theaterstück, für das er folgende Personen mitteilt:

—Characters:
TOBY TOSTOFF (a ruined Pole)
CRAB (a bushranger)
MEDICAL DICK
and
MEDICAL DAVY
$\left.\begin{array}{l}\\ \\ \\ \end{array}\right\}$ (two birds with one stone)

[9] Was den Satz von Eigennamen in Kapitälchen betrifft, vergleiche Tschicholds 'Ausgewählte Aufsätze', S. 130. Zum Satz speziell von Dramen kenne ich leider keine Literatur.

MOTHER GROGAN (a watercarrier)
FRESH NELLY
and
ROSALIE (the coalquay whore).

(*c9 ℓ1181–1189*)

Das komplexe Layout kann hier eindeutig funktionalisiert werden: Figurennamen stehen in Kapitälchen, Rollenbeschreibungen anschließend in Klammern; zwei Figuren können unter einer gemeinsamen Rollenbeschreibung zusammengefaßt werden (DICK und DAVY), aber es können auch zwei Figuren zu einer Gruppe zusammengefaßt werden, von denen nur einer eine Rollenbeschreibung zugeordnet ist (FRESH NELLY und ROSALIE).

5.6. *Briefe und Postadressen*

Im vierten Kapitel bekommt Leopold Bloom einen Brief von seiner Tochter Milly; der Brief wird zusammenhängend in den Erzähltext eingeschoben und graphisch als Brief wiedergegeben (*c4 ℓ397–414*), er besteht aus den typischen Briefelementen Anredeformel, Brieftext, abschließender Gruß, Unterschrift, Postscriptum, nochmal Unterschrift (abgekürzt). Diese Elemente sind in gängiger Weise durch Einrückung und Zeilenabstand brieftypisch formatiert, ohne aus dem Prosalayout des umgebenden Textes mehr als nötig herauszufallen; die Anrede ist nur halb so weit eingerückt wie normale Absätze und vom Text durch vergrößerten Zeilenabstand abgesetzt, der Text in normalen Absätzen formatiert, die Abschiedsformel $\frac{2}{5}$ der vollen Breite eingezogen, die Unterschrift $\frac{3}{5}$, das PS ist nicht eingerückt; diese Schlußelemente sind ebenfalls durch größere Zeilenabstände voneinander getrennt.

Dearest Papli

Thanks ever so much for the lovely birthday present. It suits me splendid. Everyone says I'm quite the belle in my new tam.
...
Tell him silly Milly sends my best respects. I must now close with fondest love

Your fond daughter
Milly

P.S. Excuse bad writing am in hurry. Byby.

M.

(*c4 ℓ397–414*)

Im fünften Kapitel liest Leopold Bloom den Brief, den er von seiner heimlichen Korrespondentin Martha Clifford bekommen hat (*c5 ℓ241–259*); er besteht aus ähnlichen Elementen wie Millys Brief, die auch entsprechend formatiert sind. Im elften Kapitel schreibt er einen Antwortbrief; dessen Text erscheint aber nur stückweise als Teil eines Bewußtseinsstroms und ist typographisch in keiner Weise hervorgehoben oder strukturiert.

Sowohl im fünften wie im elften Kapitel werden die Postadressen auf den jeweiligen Briefen zeilenweise formatiert als Prosadisplay wiedergegeben; die Formatierung dieser beiden Adressen ist aber unterschiedlich, sie werden also nicht nur streng nach ihrer Funktion formatiert, um dem Leser als Postadressen kenntlich gemacht zu werden, sondern geben auch die individuellen Schreibweisen der Absender wieder.[10]

[10] Es mag etwas verkrampft wirken, wie ich immer wieder auf diese Unterscheidung hinweise; ich will auch nicht sagen, daß Joyce sich allzutiefe Gedanken über die individuellen Adressierstile seiner Figuren gemacht hat. Diese Beobachtungen sind aber für die SGML-Codierung wichtig.

Henry Flower Esq,
$^c/_o$ P.O. Westland Row,
City.

(c5 ℓ62–65)

Miss Martha Clifford
c/o P. O.
Dolphin's Barn Lane
Dublin

(c11 ℓ897–900)

Im zwölften Kapitel wird ebenfalls ein Brief wiedergegeben; dieser Brief beginnt mit der Absenderadresse oben rechts, die Anredeformel ist komplizierter gestaltet und der Absender gibt nach der Unterschrift noch seinen Beruf an, noch etwas weiter ausgerückt. Vor allem aber wird dieser Brief vorgelesen, ist also Bestandteil einer wörtlichen Rede, und wird mehrfach durch Einwürfe der Zuhörer und Erzähltext unterbrochen. Der eigentliche Brieftext ist, wie andere wörtlichen Reden innerhalb wörtlicher Reden (wenn also A erzählt, was B gesagt hat), in Kursivschrift gesetzt; dadurch hebt er sich auch besser vom Kontext ab, in den er eingeflochten ist.

5.7. *Bücherverzeichnis*

Im siebzehnten Kapitel wird der Inhalt von Blooms Bücherregal in einer Form aufgelistet, wie man sie wohl in einem Antiquariatskatalog finden würde (*c17 ℓ1362–1407*); von jedem Buch werden der Autor und der kursivgesetzte Titel angegeben, gefolgt von Angaben zu Ausstattung und Zustand in Klammern. Bei einem Buch werden nach der Titelangabe noch Informationen zu Ausleihe- und Rückgabedatum in der städtischen Leihbücherei gegeben, und dem letzten Buchtitel der Liste folgen ironisierende, umständlich-altertümliche Angaben zum Buch, die auch durch den Gebrauch des langen 's' im Wortinnern eine zusätzliche charakteristische Note erhalten. Auch einige der vorhergehenden Angaben zu Ausstattung und Zustand gehen in unterschiedlicher Weise über ihre vordergründige Funktion hinaus, was sehr unterhaltsam zu lesen, aber schwer zu kategorisieren ist.

Jedes Buch wird in einem eigenen Absatz beschrieben; die erste Zeile beginnt ohne Einzug links, die folgenden Zeilen sind ein Geviert eingerückt, dadurch wird der Listencharakter der Aufstellung unterstrichen.

> Lockhart's *Life of Napoleon* (cover wanting, marginal annotations, minimising victories, aggrandising defeats of the protagonist).
>
> *Soll und Haben* by Gustav Freytag (black boards, Gothic characters, cigarette coupon bookmark at p. 24).

(c17 ℓ1381–1384)

5.8. *Prosazeilen mit synchronisierten Wortabständen*

Im elften Kapitel, das auf verschiedene Weise nach musikalischen Strukturen gestaltet ist, stehen folgende zwei Zeilen als normale Prosaabsätze untereinander im Text:

> Far. Far. Far. Far.
>
> Tap. Tap. Tap. Tap.

(c11 ℓ1185–1186)

Man würde das regelmäßige Druckbild vielleicht übersehen oder für Zufall halten, wenn in der Fußnote nicht Joyces Anweisung an den Setzer wiedergegeben wäre, "Imprimez directement une parole sous l'autre".

So aber wird die unauffällige optische Synchronisation bewußt, die die Verbindung zwischen dem Geräusch eines Blindenstocks (das ist der Kontext für das “Tap.”) und einem Gedanken Blooms unterstützt.

5.9. *Bilanz*

Im siebzehnten Kapitel, das in Fragen und Antworten gegliedert ist, lautet eine der ‘Fragen’ “Compile the budget for 16 June 1904” (den Tag an dem die Handlung spielt). Die Antwort ist eine tabellarisch angeordnete Bilanz.[11]

Debit		Credit	
	£ - s - d		£ - s - d
1 Pork kidney	0 - 0 - 3	Cash in Hand	0 - 4 - 9
1 Copy *Freeman's Journal*	0 - 0 - 1	Commission rec^{d} *Freeman's Journal*	1 - 7 - 6
1 Bath and Gratification	0 - 1 - 6	Loan (Stephen Dedalus)	1 - 7 - 0
Tramfare	0 - 0 - 1		
...			
Balance	0 - 16 - 6		
	£ 2 - 19 - 3		£ 2 - 19 - 3

(*c17 ℓ1456–1475*)

In dieser Bilanz ist die Typographie wieder rein funktional; für jede der beiden Seiten besteht ein Eintrag aus dem Ausgabe- beziehungsweise Einnahmeanlaß und dem Geldbetrag; dazu kommen die Spaltentitel, der Saldo und die Spaltensummen.

5.10. *Aeolus: Schlagzeilen*

Das siebte Kapitel spielt zum größten Teil in einer Zeitungsredaktion; dazu passend ist das Kapitel in relativ kurze Abschnitte unterteilt, denen jeweils eine schlagzeilenartige Überschrift vorangestellt ist. Diese Überschriften sind in Versalien gesetzt und zentriert, der restliche Text besteht aus gewöhnlicher Prosa und Lyrik.

> EXIT BLOOM
>
> —I'm just running round to Bachelor's walk, Mr Bloom said, about this ad of Keyes's. Want to fix it up. They tell me he's round there in Dillons's.
>
> He looked indecisively for a moment at their faces. The editor who, leaning against the mantleshelf, had propped his head on his hand, suddenly stretched forth an arm amply.
>
> —Begone! he said. The world is before you.
>
> (*c7 ℓ429–435*)

5.11. *Wandering Rocks: Unterabschnitte*

Im zehnten Kapitel werden einzelne Personen und Ereignisse in neunzehn relativ kurzen Episoden verfolgt und beschrieben; der Übergang von einer Episode zur nächsten ist ein übergangsloser ‘harter Schnitt’, der im Text durch drei im Dreieck angeordnete Sternchen bezeichnet ist.

[11] Wer sich mit Buchhaltung besser auskennt als ich, möge für sich entscheiden, ob es sich tatsächlich um eine Bilanz im strengen Wortsinn handelt. Ich bleibe bei der handlichen Bezeichnung.

5.12. *Sirens: Ein Präludium aus Kurzabsätzen*

Am Beginn des von Musik geprägten elften Kapitels steht eine Reihe von kurzen Absätzen, die hintereinander gelesen wenig Sinn ergeben (*c11 ℓ1–63*). Beim Lesen des restlichen Kapitels stellt man aber fest, daß jeder dieser Kurzabsätze in chronologischer Reihenfolge ein Motiv des weiteren Textes anzitiert, bevor mit dem Ein-Wort-Absatz "Begin!" und zwei Zeilenhöhen Abstand die Kurzabsätze in den Kapiteltext übergehen.

Optisch unterscheiden sich diese Kurzabsätze dadurch vom Text, daß ihre erste Zeile, meist auch die einzige, auf volle Breite gesetzt ist, während die Folgezeilen mit einem Geviert eingezogen sind, wobei allerdings nur drei von sechzig Absätzen überhaupt mehr als eine Zeile haben. Die gleiche Absatzgestalt wurde bei der Bücherliste im siebzehnten Kapitel verwendet (siehe S. 27); sie ist ein Standard der typographischen Behandlung von Listen,[12] das 'Präludium' des elften Kapitels kann also auch als Liste aufgefaßt werden.

> O rose! Castile. The morn is breaking.
> Jingle jingle jaunted jingling.
> Coin rang. Clock clacked.
> Avowal. *Sonnez.* I could. Rebound of garter. Not leave thee. Smack. *La cloche!*
> Thigh smack. Avowal. Warm. Sweetheart, goodbye!
> Jingle. Bloo.
>
> (*c11 ℓ15–19*)

5.13. *Cyclops: zwei deutlich unterscheidbare Prosastile*

Im zwölften Kapitel sprechen abwechselnd zwei Erzähler. Der eine erzählt in einfacher, saftiger Umgangssprache, was er in der geschilderten Nachmittagsstunde im Pub erlebt, dicht am unmittelbaren Erleben und mit vielen wörtlichen Zitaten. Der andere Erzähler entwirft in manieriertem Stil phantastische Szenerien, die auf unterschiedliche Weise die Wirklichkeit des ersten Erzählers spiegeln oder fortspinnen.

Obwohl diese zwei Erzählstränge typographisch nicht unterschieden werden, ist der Unterschied so markant, daß ich ihn bei meiner Dokumentation der äußeren Strukturen nicht übergehen möchte. (Zum Verständnis des folgenden Beispiels: der "citizen" ist ein wüster irischer Nationalist, das Spiegelbild des Zyklopen der Odyssee; sein Hund heißt Garryowen.)

> So howandever, as I was saying, the old dog seeing the tin was empty starts mousing around by Joe and me. I'd train him by kindness, so I would, if he was my dog. Give him a rousing fine kick now and again where it wouldn't blind him.
> —Afraid he'll bite you? says the citizen, jeering.
> —No, says I. But he might take my leg for a lamppost.
> So he calls the old dog over.
> —What's on you, Garry? says he.
> Then he starts hauling and mauling and talking to him in Irish and the old towser growling, letting on to answer, like a duet in the opera. Such growling you never heard as they let off between them.
> ...
> All those who are interested in the spread of human culture among the lower animals (and their name is legion) should make a point of not missing the really marvelous exhibition of cynanthropy given by the famous old Irish red setter wolfdog formerly known by the *sobriquet* of Garryowen and recently rechristened by his large circle of friends and acquaintances Owen Garry. The exhibition,

[12] Nach Tschichold, *Erfreuliche Drucksachen*, S. 93.

> which is the result of years of training by kindness and a carefully thoughtout dietary system, comprises, among other achievements, the recitation of verse.
>
> (*c12 ℓ697–719*)

5.14. *Circe: Drama*

Das fünfzehnte Kapitel des *Ulysses*, das mit fast fünftausend Zeilen merh als ein Achtel des gesamten Textvolumens ausmacht, ist vollständig als Drama geschrieben, ebenso ein kleiner Teil des neunten Kapitels. Die Dramenform besteht im *Ulysses* aus einer Folge von Sprecherreden und szenischen Regieanweisungen (*narrative directions*) die gleichberechtigt zwischen den Sprecherreden stehen. Sprecherreden bestehen aus der Angabe des Sprechers, auf einer eigenen Zeile zentriert und in Kapitälchen gesetzt, gefolgt von dem gesprochenen Text. Der gesprochene Text kann Prosa sein (ohne Einzug am Absatzanfang), aber auch Lyrik, oder beides nacheinander. In die Prosa der Sprecherrede eingestreut sind kurze Regieanweisungen (*speech directions*). Beide Formen von Regieanweisungen sind in Klammern eingeschlossen; im fünfzehnten Kapitel sind sie in schräggestellter Schrift gesetzt, im neunten Kapitel kursiv. Die Begründung für die schräggestellte Schrift im fünfzehnten Kapitel gibt Gabler im Anhang, S. 1747. Die dort getroffene Entscheidung scheint aber nicht auch auf die Dramenpassage im neunten Kapitel angewandt worden zu sein. Im Manuskript des fünfzehnten Kapitels schreibt Joyce die Sprecher zentriert und unterstrichen, die Absätze ohne Einzug, und kennzeichnet Bühnenanweisungen nur durch Klammerung, nicht durch Unterstreichung.

Die *narrative directions* bilden eigene Absätze aus kompletten Sätzen, sie beginnen mit Großbuchstaben und enden mit einem Punkt. Außer durch die Schrift sind sie durch den Einzug jeder Zeile hervorgehoben. Die *speech directions* hingegen sind nur kurze Anweisungen innerhalb des gesprochenen Textes, die meist nur aus einem einfachen Satz oder einem Satzbruchstück bestehen; sie beginnnen nicht mit Großbuchstaben und enden ohne Satzzeichen. Fremdsprachige Wendungen und ähnliches Material im gesprochenen Text sind natürlich nach wie vor kursiv, Lyrik hingegen ist im fünfzehnten Kapitel *nicht* kursiv gesetzt. Der folgende Abschnitt enthält fast alle typischen Merkmale:

CISSY CAFFREY

> More luck to me. Cavan, Cootehill and Belturbet. (*she sings*)
>
> I gave it to Nelly
> To stick in her belly,
> The leg of the duck,
> The leg of the duck.
>
> (*Private Carr and Private Compton turn and counterretort, their tunics bloodbright in a lampglow, black sockets of caps on their blond cropped polls. Stephen Dedalus and Lynch pass through the crowd close to the redcoats.*)
>
> (*c15 ℓ54–63*)

5.15. *Ithaca: Frage und Antwort*

Das siebzehnte Kapitel besteht aus einer langen Folge von Fragen und Antworten, durch die die Handlung wiedergeben und ein weiterer Hintergrund zu den Hauptfiguren und dem bisher berichteten Geschehen geliefert wird.

Die Fragen bilden jeweils einen nicht eingezogenen eigenen Absatz und sind durch vergrößerten Zeilenabstand von der nachfolgenden (und auch der vorausgehenden) Antwort getrennt. Die Antworten sind selten länger als ein Absatz, die Absatzanfänge sind ebenfalls nicht eingezogen.

> Having set the halffilled kettle on the now burning coals, why did he return to the stillflowing tap?
>
> To wash his soiled hands with a partially consumed tablet of Barrington's lemon-flavoured soap to which paper still adhered, (bought thirteen hours previously for fourpence and still unpaid for), in fresh and cold neverchanging everchanging water and dry them, face and hands, in a long redbordered holland cloth passed over a wooden revolving roller.
>
> (*c17 ℓ229–235*)

5.16. *Penelope: freier Prosafluß*

Die Struktur des achtzehnten Kapitels ist wohl die bekannteste im Ulysses: in acht extrem langen Absätzen folgt die Prosa ohne Punkt und Komma dem Gedankenfluß von Molly Bloom (ein einziger Punkt beschließt das Kapitel und das Buch). Die Anmerkung zu Entstehungsort und -zeit am Schluß sollte man wohl nicht als Teil des eigentlichen Romantextes betrachten.

> ...and then I asked him with my eyes to ask again yes and then he asked me would I yes to say yes my mountain flower and first I put my arms around him yes and drew him down to me so he could feel my breasts all perfume yes and his heart was going like mad and yes I said yes I will Yes.
>
> Trieste–Zurich–Paris
> 1914–1921
>
> (*c18 ℓ1605–1611*)

6. Elemente des Metatextes

In diesem Abschnitt behandle ich die Strukturen mit denen auf den linken Seiten der *Ulysses*-Ausgabe, in den Fußnoten und im Anhang die Textgeschichte dokumentiert ist; ich bezeichne diese Elemente zusammenfassend als 'Metatext', weil sie Aussagen über den eigentlichen Text des *Ulysses*, den ich im folgenden auch einfach 'Lesetext' nenne, machen.

Im Gegensatz zum Lesetext, der eine ästhetisch-literarische Funktion hat, dient der Metatext der kompakten Vermittlung klar strukturierter Information. Das wird erreicht durch ein System von Siglen und diakritischen Zeichen, deren Beschreibung in der Buchausgabe mehrere Seiten umfaßt (vergleiche S. 104 im Anhang der Arbeit).

Bei gründlicher Betrachtung, und vor allem bei dem Versuch, diese Strukturen von einem Programm automatisch erkennen zu lassen, stellt man aber fest, daß sie trotz aller Formalisierung von Menschen für Menschen geschrieben sind und sich deswegen formale Unschärfen leisten können. Genau das unterscheidet eine Edition als wissenschaftliche Kommunikationshandlung von einem stur ausgefüllten Fragebogen – es setzt aber Grenzen, sobald man versucht, die Strukturen der elektronischen Edition möglichst klar und eindeutig zu definieren und mechanisch zu erzeugen.

6.1. *Überblick*

Die synoptische Darstellung der Textentstehung auf den linken Seiten der Buchausgabe strukturiert die Textenstehung konzeptuell in Stufen, die jeweils einem neuen Quelldokument oder einem neuen Bearbeitungsdurchgang entsprechen (*inter-document changes*); zusätzlich werden Bearbeitungsvorgänge und andere wesentliche Textbefunde innerhalb einer Stufe dokumentiert (*intra-document changes*), dies betrifft vor allem die Handschriften, die die Grundschicht des Textmodells bilden. In der Sicht der Herausgeber wird die erste zusammenhängende Manuskriptfassung durch die späteren, umfangreichen Revisionen fortgeschrieben; aus vielen übereinanderliegenden Textschichten entsteht so ein die Quelldokumente überspannender Gesamttext. Zur wissenschaftlichen Begründung für den Wortlaut dieses *continuous manuscript text* dienen die Fußnoten, die die Abweichung von vorhandenen Dokumenten an kritisch relevanten Stellen dokumentieren oder andere Belege zur Textsituation bringen. Wo eine ausführlichere Diskussion notwendig ist, verweisen sie auf die *textual notes* im Anhang.

Die *historische Kollation* im Anhang dokumentiert stark formalisiert die Abweichungen des neu etablierten Lesetextes vom Text der veröffentlichten Ausgaben von 1922 (Erstdruck), 1926, 1932, 1936 und 1961.

⋆

Für die vollständige Auflistung und Beschreibung der im gedruckten Metatext verwendeten Symbole verweise ich auf die Seiten x–xiii der Buchausgabe des *Ulysses*, wie ich sie im Anhang dieser Arbeit abgedruckt habe. Die sorgfältige Beschreibung der editorischen Konzeption im Nachwort der Buchausgabe (S. 1891–1905) gebe ich hier nur ansatzweise wieder, soweit sie zum praktischen Verständnis meines Vorgehens notwendig ist. Obwohl ich mich bemüht habe, für die Wiedergabe im Anhang und die Beispiele im Text die Typographie der Buchausgabe zu imitieren, habe ich die Qualität des Originals nicht erreicht.

6.2. *Die Synopsis: Textschichten*

Ein Teil der in den Text der linken Seiten eingearbeiteten Textgeschichte ist die Wiedergabe von Überarbeitungen innerhalb einer Textschicht, den *intra-document changes*. Dazu gehören Streichungen, Einfügungen und Ersetzungen sowie nachträgliche Trennung von Absätzen. Generell stehen spitze Klammern für Streichungen und Einschaltungszeichen für Hinzufügungen, wie in “the genuine ^⟨▨⟩ christine^:” (*c1 ℓ21*), wo das ursprüngliche Wort so gründlich gestrichen wurde, daß es unlesbar ist (“▨”), und stattdessen “christine” eingefügt wurde. Wird ein eingefügter Text wiederum überarbeitet, werden die Einschaltzeichen um 90° gedreht:

> A shock of red hair he has sticking out of him behind like a furzebush! ^Wait for nine months, my lad! ˃⟨It's⟩ Holy ginger, it's˂ kicking and coughing up and down in her guts already!^
>
> (*p1176 ℓ10–12*)

Die dadurch etablierten Level der Einfügung haben nur lokale Bedeutung sie sind normalerweise nicht wirklich getrennten Durchgängen durch das Manuskript zuzuordnen. Dennoch wird an einigen Stellen, offenbar der Textlogik wegen, eine Überarbeitung des Grundtextes als eine Überarbeitung ‘zweiten Grades’ verzeichnet, wenn sie mit einer danebenstehenden Überarbeitung, die tatsächlich ‘zweiten Grades’ ist, zusammenhängt:

> ˃Hair on end.˂ As he and others see me. ^⟨˃ Hair on end.⟩^ Who chose this face for me?
>
> (*p8 ℓ37–p9 ℓ1*)

Der Grundtext lautete hier “As he and others see me. Who chose this face for me?”. In einem ersten Überarbeitungsschritt wurde “Hair on end.” zwischen den beiden Sätzen eingefügt, in einem zweiten Überarbeitungsschritt dann vor den ersten Satz gestellt. Diese Verwendung der Zeichen für Überarbeitungsstufen ist anschaulich sinnvoll, aber so nicht explizit dokumentiert.[13] Nebenbei zeigt das Beispiel, wie reine Streichungen bei einem späteren Überarbeitungsdurchgang durch ein entsprechendes Einschaltungszeichen innerhalb der spitzen Klammer wiedergegeben werden.

Eine Spezialform dieser intratextuellen Bearbeitungen ist der sogenannte *pseudo overlay*, wenn in ein Manuskript erneut Text eingefügt wird, der in einem erhaltenen früheren Entwurf bereits enthalten war. Solche Pseudo-Einfügungen werden mit “‵...′”, wenn sie verschachtelt sind, mit “‶...″” codiert, wie zum Beispiel in:

> And the dark one with the mop head ‵and the ‶nigger″ mouth′. I knew she could whistle.
>
> (*p798 ℓ11–12*)

Wo der *continuous manuscript text* in einer Fußnote den Quelldokumenten gegenübergestellt wird oder anderer textueller Erläuterung bedarf, wird der Verweis auf die Fußnote durch einen kleinen Kreis, bzw. für eine Phrase aus mehreren Wörtern durch zwei Halbkreise, angezeigt. Im folgenden Beispiel wird “Lusk. Putting” und “Palace” im Apparat behandelt. “⟨righ⟩” ist ein Beispiel für eine Streichung *currente calamo*, also eine Streichung, die nicht bei einer späteren Überarbeitung, sondern direkt während der Niederschrift erfolgt ist.

> ^⟨Turnkey⟩ Turnkey's daughter^ got him out of Richmond, off from Lusk.◜ Putting◝ up in the Buckingham Palace° hotel ⟨righ⟩ under their very noses.
>
> (*c8 ℓ9–11*)

⋆

Textänderungen die in einem neuen Quelldokument oder einem unabhängigen Bearbeitungsdurchgang eingeführt worden sind, werden jeweils kapitelweise zu einer Stufe der Textentstehung, einer Textschicht, zusammengefaßt. Diese Stufen werden durch Indizes bezeichnet, die in den *general notes* zu jedem Kapitel definiert werden. Die Grundschicht ist jeweils die erste vollständige Manuskriptfassung des Textes; die in den späteren Stufen erfolgten Änderungen werden dann in diesen Text eingetragen. Da die Bearbeitungen einer Stufe normalerweise die Ausgangsbasis für die darauffolgenden Bearbeitungen waren, bilden die Stufen fast immer eine korrekt verschachtelte Klammerhierarchie.

Der hinzugefügte Text wird durch kleine rechte Winkel markiert, die mit der Sigle der Textschicht bezeichnet sind; der weggefallene Text durch eckige Klammern. Zur Verwendung der Stufenindizes vergleiche auch den Anhang, S. 104.

Ein kurzes Beispiel:

A ⟨DARKVISAGED⟩ DARKVISAGED MAN

> (*in disguised accent*) ⌜D[Hundred] ⌜5[Hunderd] ⌜9[Hundert] Hoondert9⌝5⌝D⌝ ⌜B[pounds] ⌜5[pound] ⌜8[pount] punt8⌝5⌝ ⌜D[sterling] sterlinkD⌝.B⌝
>
> (*p1174 ℓ1–3*)

Stellt man die einzelnen Textschichten über- statt ineinander, ergibt sich als Zuruf des *Darkvisaged Man* jeweils:

Grundschicht:	Hundred pounds [14]
B:	Hundred pound sterling.
D:	Hunderd pound sterlink.
5:	Hundert pount sterlink.
8:	Hundert punt sterlink.
9:	Hoondert punt sterlink.

[13] Um ehrlich zu sein: sie ist mir erst so spät aufgefallen, daß sie bei der Konvertierung der *intra-document changes* nicht mehr berücksichtigt wurde.

[14] Daß der Punkt am Ende völlig fehlt, ist möglicherweise ein Versehen der Herausgeber.

Die letzte Form steht dann auch, von ihrer Vorgeschichte bereinigt, im Lesetext auf der rechten Seite.

⋆

Die so codierten Revisionen erstrecken sich nicht nur auf die Einfügung oder Streichung von Worten innerhalb fester Strukturen. Es werden oft ganze Absätze, Gedichte, Sprecherreden *et cetera* eingefügt; eine Revision kann große Textmengen und mehrere Strukturgrenzen überspannen. Wo sich eine Revision über mehrere Absätze oder mehrere Liedstrophen erstreckt, wird sie allerdings graphisch in mehrere Einzelrevisionen aufgeteilt.

Eine Revision kann auch den strukturellen Charakter eines Textstücks ändern, wenn z.B. im fünfzehnten Kapitel eine in den gesprochenen Text integrierte Bühnenanweisung zu einer eigenständigen szenischen Anweisung erweitert wird (*p1046 ℓ1–7*), oder wenn ein ursprünglich wie Prosa in den Text eingefügtes Zitat durch Einfügung von Zeilenumbrüchen in Gedichtzeilen zerlegt wird:

He walked along the upwardcurving path.

|⁴|*Liliata rutilantium.*
|⁴|*Turma circumdet.*
|⁴|⌜⁴*Iubilantium*° *te virginum.*⁴⌝°

(*p42 ℓ31–34*)

⋆

Gelegentlich werden mit Hilfe der Textstufencodes auch Phänomene beschrieben, die sich nicht in das Modell der linear aufeinander aufbauenden Textschichten einfügen lassen. Eines dieser Phänomene sind die abgelehnten kollateralen Varianten in den Reinschriften einiger Kapitel:

Das achte Kapitel, beispielsweise, wurde von Joyce zunächst in einem letzten Entwurf (*final working draft*) niedergeschrieben; diesen Entwurf schrieb er noch einmal ins Reine (*fair copy*) – es war aber der Entwurf, nach dem das Typoskript getippt wurde. Der Entwurf selbst ist zwar nicht mehr enthalten, sein Text läßt sich aber weitgehend editorisch ermitteln durch den Vergleich der Typoskriptfassung mit der Reinschrift, unter Ausschluß der Übertragungsfehler durch den Typisten bzw. durch Joyce selbst bei Erstellung der Reinschrift. An Stellen, an denen die textkritisch so ermittelte Grundschicht des *continuous manuscript text* von der Lesart der Reinschrift abweicht, wird letztere im Text mit angeführt, obwohl sie außerhalb der linearen Stufenfolge der dargestellten Textentwicklung steht:

⌜V{Stopping} StoppedV⌝ in Citron's saint Kevin's parade.

(*p326 ℓ27–28*)

"Stopping" ist die vom Herausgeber verworfene Lesart der Reinschrift; die geschweiften Klammern sollen daran erinnern, daß es sich hier um ein spezielle Anwendung der Textstufen-Codierung handelt. An anderen Stellen folgt der Herausgeber aber durchaus der Reinschrift, dann nämlich, wenn sie offenbar eine Textentwicklung dokumentiert, die in den Entwurf nicht übernommen wurde, und deswegen auch nicht im Typoskript und später im Satz berücksichtigt wurde:

Provost's house. ... ⌜RLike a mortuary chapel.R⌝ Wouldn't live in it if they paid me.

(*p346 ℓ16–18*)

Die Indizes 'V' und 'R' bezeichnen keine eigenständige Stufe der linearen Textentwicklung im *continuopus manuscript text*, sie sind aus praktischen Gründen in das Schichtenmodell aufgenommen worden.

⋆

Eine weitere Dehnung des Textstufenprinzips aus praktischen Gründen tritt im zehnten Kapitel auf; wegen seiner Augenkrankheit mußte Joyce Teile dieses Kapitels seinem Freund Frank Budgen diktieren; die Indices “⌜A...A⌝” (*author*) und “⌜S...S⌝” (*scribe*) werden in diesem Kapitel verwendet, um die beiden Schreiber zu unterscheiden, dabei werden auch Budgens Korrekturen in Joyces Text und umgekehrt mit diesen Indizes ausgezeichnet. Diese beiden Indizes bezeichnen also ebenfalls keine streng aufeinanderfolgenden Stufen der Textgeschichte.

Die Sonderform “⌜()...()⌝” bezeichnet einen Abschnitt des fünfzehnten Kapitels, wo das Manuskript, die Quelle für die Grundschicht des *continuous manuscript text*, nicht überliefert ist und die Typoskriptfassung die Grundschicht liefern muß.

Die Sonderform “⌜+...+⌝” im siebzehnten Kapitel markiert die Einfügung ganzer Frage-Antwort-Sequenzen im Manuskript (ist also eigentlich ein Spezialfall von “^...^”).

Mit einem Stern markierte Revisionen bezeichnen im fünfzehnten Kapitel die Normalisierung von Groß- bzw. Kleinschreibung und Zeichensetzung im Zusammenhang mit Bühnenanweisungen; im achtzehnten Kapitel bezeichnen sie die stufenweise Elimination von Satzzeichen, die auf diese Weise pauschal, ohne Zuordnung zu Textschichten, wiedergegeben ist.

⋆

Die Symbole sowohl für *intra-* wie *inter-document changes* können zusätzliche Information tragen: ein Paar beigestellte leere Klammern weisen darauf hin, daß die genaue Position einer Einfügung nicht belegt ist und vom Herausgeber bestimmt wurde. Ein beigestelltes Sternchen zeigt an, daß eine Revision nicht belegt ist, sondern infolge anderer Änderungen vom Herausgeber konstruiert wurde. Wird der Index einer Textstufe in Klammern gesetzt, handelt es sich um eine Textstufe, deren zugehöriger Textzeuge nicht erhalten ist.

6.3. *Die Synopsis: sonstige Symbole*

Neben der Dokumentation der Entwicklungsstufen des Textes unter der Hand des Autors werden noch einige weitere für die Textgeschichte relevante Phänomene des Manuskriptbefundes durch graphische Symbole beschrieben:

- Für eine Einfügung freigelassener Platz wird durch “∅” bezeichnet;
- Unleserliche Worte werden mit “□”, unleserliche Wortteile oder Zeichen mit “▯” bezeichnet (kommt vor allem in gestrichenem bzw. ersetztem Text vor);
- Ein gelöschtes Wort wird durch “⧄” bezeichnet;
- Eine eingezeichnete Einfügung, für die kein Text angegeben wurde, wird mit “△” bezeichnet;
- Ein textlich relevanter Seitenwechsel im Quelldokument wird durch “||” bezeichnet, ein Zeilenumbruch durch “|”, bzw. “-|” oder “-|-”, wenn am Ende der Zeile oder zusätzlich auch am Anfang der neuen Zeile ein Bindestrich stand;
- Auf Emendationen zu einer Textstelle wird durch das Symbol “°” bzw. “⸢...⸣” hingewiesen, eine Fußnote zu einer Änderung der generellen Dokumentsituation wird durch “⌑” angezeigt (siehe unten).

6.4. *Die Fußnoten*

Die Symbole “°” und “⸢...⸣” verweisen auf eine Anmerkung zu den so markierten Textstellen am Fuß der Seite. Mit “°” wird das vorausgehende Wort markiert, mit “⸢...⸣” eine ganze Passage, die sich vom Wort vor dem “⸢”-Symbol bis (einschließlich) zu dem Wort vor dem entsprechenden "⌉"-Symbol erstreckt. Diese Markierungen können ineinander geschachtelt werden, es kann sich also eine Fußnote auf einen ganzen Absatz erstrecken, eine weitere auf einen Satz innerhalb dieses Absatzes und eine dritte auf ein Wort in diesem Satz.

Die Fußnoten enthalten, mit ganz wenigen Ausnahmen, 'Emendationen', also Belege, wo und evt. warum die überlieferten Textzeugen und der erstellte Text nicht übereinstimmen. Dafür wird eine in der Tradition historisch-kritischer Textausgaben bereits eingeführte, aber mit neuen Elementen bereicherte Symbolik verwendet. Der Anfang des ersten Kapitels liefert einige typische Beispiele:

> Stately,° plump Buck Mulligan came from the stairhead, bearing a bowl of lather on which a mirror and a razor lay crossed. A yellow dressinggown, ungirdled, was sustained gently behind him ⌈⁽ᴮ⁾[by] on°⁽ᴮ⁾⌉ the mild morning air. ...
>
> ———
>
> *PRECEDING PAGE* I] *CF* Faire préceder par une page vide marquée au milieu ainsi I a1   *GENERAL NOTE* → TN   1 Stately,] (tB); *NOT INDENTED* aR   3 on] LR,Eg; by (tB),(aC) *PCU*; *CF 892.23*
>
> (*p2 ℓ3*)

Die erste Fußnote bezieht sich auf die römische Eins, die auf der vorangehenden Seite den ersten Teil des Romans einleitet. Die zweite Fußnote findet sich so am Anfang jedes Kapitels und weist auf die allgemeinen Hinweise zu Dokumentenlage, verwendeten Textschichten-Indizes und sonstigen Besonderheiten des Kapitels in den *Textual Notes* hin. Die dritte Fußnote bezieht sich auf das erste Wort des Textes, das, dem Typoskript folgend, eingerückt wurde, in der Reinschrift des Kapitels aber ohne Einrückung steht. Die vierte Fußnote bezieht sich auf das Wort "on" im Text; zusammen mit der Textschichtencodierung '⌈⁽ᴮ⁾[by] on°⁽ᴮ⁾⌉' ergibt sich, daß im (in drei Exemplaren hergestellten) Typoskript 'by' stand, Joyce dies auf den beiden Exemplaren, nach denen die Fortsetzungsabdrucke in der *Literary Review* und *The Egoist* gesetzt wurden, zu 'on' geändert hat, nicht aber im dritten Exemplar, nach dem das Buch gesetzt wurde (*PCU* steht für *PRINTER'S COPY UNCHANGED*). Geht man dem Hinweis auf *p892 ℓ23* nach, findet man dort eine stark verfremdete Wiedergabe derselben Szene, in der ebenfalls 'on' verwendet wird.

Die Fußnoten erweisen sich also, wenn man sich in ihre Symbolik eingelesen hat, als sehr kompakte und präzise Informationsquelle. Eine normale emendatorische Fußnote enthält folgende Elemente:

- Die Nummer(n) der Zeile(n) der linken Textseite, auf die sie sich bezieht, ausnahmsweise auch eine andere Ortsangabe.
- Die Wiederholung des betroffenen Textes, das sogenannte *Lemma*. Falls das Lemma aus mehr als zwei Worten besteht, wird nur das erste und das letzte Wort angegeben, durch einen doppelten Bindestrich getrennt.
- Eine geschlossene eckige Klammer, die das Ende des Lemmas bezeichnet.
- Die Siglen für die Textzeugen, die die Lesart des Lemmas stützen. Großbuchstaben und Zahlen geben die Dokumente an, vorangestellte Kleinbuchstaben 'a', 's', 't' oder 'e' stehen für die Quelle der Lesart: Handschrift des Autors, anderer Schreiber, Typoskript, oder eine Ergänzung des Herausgebers. Wenn die Quellenangabe in Klammern steht, ist das Dokument selbst nicht erhalten, sondern kann nur indirekt erschlossen werden.
- Ein Semikolon, als Trennzeichen zwischen Quellenbelegen und nachfolgender Lesart.
- Abweichende Lesarten, jeweils wieder mit Quellenbeleg und, falls eine weitere Lesart folgt, abschließendem Semikolon.
- Dazwischengestreut erscheinen Kommentare zu Lesarten oder Quellenbelegen, in schräggestellten Kapitälchen; manchmal treten die Kommentare auch als Beschreibungen ganz an die Stelle des jeweiligen Elements, im obigen Beispiel

> *NOT INDENTED* anstelle einer Lesart, oder *PRECEDING PAGE* anstelle der Ortsangabe durch Zeilennummer. Einige wiederkehrende Formeln werden durch Abkürzungen wiedergegeben, zum Beispiel *PCU* für *PRINTER'S COPY UNCHANGED*.

Die Emendationen haben also größtenteils eine feste formale Struktur. Die Verwendung von Kommentaren macht diese Struktur aber flexibel handhabbar, bis hin zu Extremfällen, in denen die Emendationen zu gewöhnlichen kurzen Fußnoten werden. Diese Flexibilität ist wertvoll für die Kommunikation zwischen Herausgeber und Leser, sie begrenzt aber die Möglichkeiten der automatischen Verarbeitung der emendatorischen Fußnoten.

Ein Beispiel für eine reine Fußnote, die sich nicht einmal auf eine Ortsangabe im Quelltext bezieht, ist der Hinweis '*GENERAL NOTE* → TN' (siehe oben). Ein anderes Beispiel sind Anmerkungen zu wichtigen Änderungen in der Dokumentsituation, die mit Hilfe des '⊞'-Symbols und einer zweckentfremdeten Emendation in den Text eingefügt werden:

> … On Newcomen bridge the very reverend John Conmee ⊞ S.J. of …
>
> ———
>
> 8 ⊞] *TS LOST*
>
> *(p476 ℓ8)*

Zuletzt muß noch festgehalten werden, daß nicht jeder Text in den Fußnoten, selbst wenn er von Joyce stammt und wie eine Textvariante notiert ist, wirklich als Text des *Ulysses* gedacht ist; unter den Beispielen am Anfang des Abschnitts findet sich die Wiedergabe einer Notiz von Joyce an den französischen Setzer des Buchs.

6.5. *Die historische Kollation*

Formal ähnlich aufgebaut wie die Emendationen sind die Einträge der historischen Kollation im Anhang. Sie dokumentieren sämtliche Stellen, an denen fünf ausgewählte frühere Textausgaben vom Lesetext der Gabler-Ausgabe abweichen. Zusätzlich wird jeweils die Seiten-Zeilen-Referenz der weitverbreiteten Ausgabe von 1961 angegeben, und das Dokument benannt, auf dem die Abweichung beruht. Die historische Kollation ist strenger formalisiert als die Emendationen, aber arbeitet ebenfalls mit Kommentaren, die die formalen Elemente ergänzen oder ersetzen.

> 3.8 on] by :: 22–61 ⟨3.3⟩ \(aC) *PCU*
> 176.5 *omphalos.*] *NO ITALICS* :: 22–61 ⟨7.33⟩ \aR
> 194.7 and] *ADDED* I :: 22–61 ⟨8.14⟩ \(tB)+a1
> 239.1–241.6 —And - - *cars.*] *PRINTED AS INSET, NO DIALOGUE DASH* :: 22–61 ⟨9.22–24⟩ \s1
> 314.4 briskly] *MISINSERTED* about the hearth :: 22-61 ⟨11.25⟩ \(tB)
> 629.5 Steeeeeeeeeeeephen!] Steeeeeeeeeeeephen. :: 22–26 \(tB):1; Steeeeeeeeeephen. :: 32–61 ⟨20.17⟩
>
> (Auswahl aus S. 1755–1757)

Die formale Struktur wird fast schon aus den Beispielen allein deutlich; im einzelnen ist ein Eintrag der historischen Kollation folgendermaßen aufgebaut:

- Eine Referenz auf die Zeile(n) und die Nummer des Wortes (der Worte) innerhalb der Zeile nach der kapitelweisen Zeilenzählung des Lesetextes auf den rechten Seiten. Die historische Kollation ist kapitelweise in Abschnitte unterteilt, so daß die Kapitelnummer, die die Referenz vollständig macht, implizit ist. Was genau für die Edition ein Wort darstellt, ist nicht explizit definiert.[14]

- Die Wiederholung des betroffenen Textstücks im Lesetext, bei mehr als zwei Worten werden nur das erste und das letzte Wort gegeben, durch zwei Bindestriche getrennt (wie in den Emendationen); darauf folgt eine geschlossene eckige Klammer.
- Den Rest des Eintrags bilden eine oder mehrere Lesarten (oben das Beispiel mit "Steeeeeeeeeeeephen"), die alle wie folgt aufgebaut sind:

- Die abweichende Lesart wird wiedergegeben, evt. ergänzt oder beschrieben durch Kommentar; das Verständnis des Kommentars ist wesentlich, um die Art der Abweichung zu bestimmen. Das Ende der Lesart wird durch zwei Doppelpunkte "::" markiert.
- Wenn die abweichende Lesart in der *Random House Edition* von 1961 enthalten ist, wird in spitzen Klammern eine Seiten-Zeilen-Referenz für diese Ausgabe gegeben.
- Wo es möglich ist, wird hinter einem *backslash* das Dokument benannt, das den Ursprung der Abweichung darstellt; handelt es sich um das Zusammenwirken mehrerer Dokumente, werden sie durch Pluszeichen verbunden. Zur Bezeichnung der Dokumente werden die gleichen Siglen verwendet wie in den Emendationen (siehe S. 36).

Bei der Verzeichnung der Lesarten wird gelegentlich das Symbol für Zeilenwechsel "|" verwendet, um Zeilenumbrüche in Gedichtzitaten und ähnlichem platzsparend zu repräsentieren. In der Historischen Kollation zum fünfzehnten Kapitel wird ein Stern verwendet, um anzuzeigen, daß das nachfolgende Zeichen nicht der Normalisierung dieses Kapitels entspricht; wenn eine Lesart ausschließlich in den normalisierten Zeichen abweicht, wurde sie aber nicht in die historische Kollation aufgenommen.

6.6. *Die 'Textual Notes'*

Das letzte Element des Metatextes, das zu behandeln ist, sind die ausführlichen Anmerkungen zu einzelnen Textstellen, die *Textual Notes* im Anhang, S. 1729–1753.

Sie sind kapitelweise in Abschnitte untergliedert; zu jedem Kapitel ist der Entstehungszeitraum mit Jahreszahlen in der Überschrift angegeben. Anschließend werden zu jedem Kapitel drei *general notes* unter den Rubriken *Documents*, *Levels* und *Comments* gegeben. Sie fassen Hinweise zu überlieferten Zeugen, zu den verwendeten Siglen für die Textstufen und sonstige Anmerkungen zur Entstehung zusammen. Auf die *general notes* wird im Text am Anfang jedes Kapitels in den Fußnoten verwiesen.

Auf diese allgemeinen Anmerkungen folgen die Anmerkungen zu einzelnen Textstellen. Jede von ihnen beginnt mit einer Seite-Zeile-Referenz auf die linken Textseiten und einer Wiederholung des betroffenen Textstücks, dem Lemma (wie immer werden längere Textstücke auf Anfangs- und Endwort verkürzt). Manche Anmerkungen beziehen sich auf mehrere Textstellen an unterschiedlichen Orten; es werden dann zunächst die Textstellen und dann die Lemmata aufgelistet, durch Schrägstriche getrennt. Das Lemma wird durch eine geschlossene eckige Klammer beendet, dann folgt der Text der Anmerkung.

In diesen Anmerkungen werden naturgemäß Textstücke des *Ulysses* zitiert (in einfachen Anführungszeichen), die zum Teil auch die Codes der synoptischen Darstellung mitenthalten. Diese Verwendung der aus dem Kontext gerissenen Codes stellt ein Zitieren des synoptischen Textes dar, und nicht eine Textschichtung innerhalb der *Textual Notes*. Auch die Siglen für Textschichten bzw. Textzeugen, die für den synoptischen Text bzw. die Emendationen und

[14] Unklarheit könnte bei Sonderfällen auftreten, wenn es um Abkürzungen, Zahlen und ähnliche Phänomene geht.

historischer Kollation eingeführt sind, werden in den *Textual Notes* innerhalb der freien Prosa verwendet.

Die *Textual Notes* sind als Liste mit nicht eingezogenen ersten Zeilen und mit einem Geviert eingezogenen Folgezeilen gesetzt, um das Auffinden einer bestimmten Anmerkung anhand der Seiten-Zeilen-Referenz zu erleichtern.

Die erste der folgenden *Textual Notes* ist eine der *general notes* zum ersten Kapitel, die zweite eine gewöhnliche Anmerkung zum neunten Kapitel:

> DOCUMENTS: Fair copy (Rosenbach Manuscript); Letters and Postcards to Claud Sykes; Lost typescript; *Little Review*; 1st *placards*; 1st – 2nd page proofs; 2nd *placards*; 3rd page proofs; First edition; Autograph errata; Printed errata.
>
> (*Ulysses*, S. 1729)

> 410.34 Synge] The dialogue dash requested in the printed errata list to the first edition (p1E) and introduced in 1926 is of doubtful authority. It turns into direct speech what is clearly designed as speech overheard and conveyed through the filter of Stephen's consciousness.
>
> (*Ulysses*, S. 1738)

Die elektronische Edition

7. Zur Konzeption der SGML-Edition

Das Ziel des elektronischen *Ulysses*, wie er jetzt vorliegt, ist es zum einen, die Strukturen der gedruckten Ausgabe, die die Textentwicklung dokumentieren, möglichst adäquat und wirksam nutzbar in eine SGML-Struktur abzubilden; zum anderen sollte der Text des *Ulysses* für eine möglichst breiten Gruppe von Anwendern und Interessenten erschließbar und nützlich sein. Die Zielvorstellungen waren im Einzelnen das Ergebnis eines längeren Entscheidungs- und Diskussionsprozesses, den ich hier noch einmal nachvollziehen möchte.

Die Konzeption meiner elektronischen Fassung des *Ulysses* bewegte sich von Anfang an zwischen zwei Polen. Zum einen war da der Standard für die Codierung von literarischen Texten als SGML-Dokument, den die *Text Encoding Initiative* als TEI P3 mit einer DTD und einer mehrere hundert Seiten starken Dokumentation, dem '*Green Book*', vorgelegt hat.[14] Es erschien mir natürlich und sinnvoll, einer elektronischen Edition durch die Verwendung solcher Standards eine gewisse Lebensdauer und vielseitige Verwendbarkeit zu ermöglichen. Es gab und gibt leider mit großem Arbeitsaufwand betriebene Editionsprojekte, die sich auf eine bestimmte Software als Medium festgelegt haben, und deswegen bereits kurz nach ihrem Erscheinen, wenn nicht schon vorher, veraltet wirken.[15] Auch heute noch scheinen einige Philologen, von den technischen Möglichkeiten die man ihnen bietet begeistert, deren Kurzlebigkeit nicht sehen zu wollen; die in der Welt des gedruckten Buches nicht vorhandene Trennung zwischen der möglichst langfristig gültigen internen Darstellung, in der die Daten aufbewahrt werden, und ihrer Verarbeitung und Präsentation durch unterschiedliche, aktuelle Software halte ich für die erste Grundbedingung einer sinnvollen elektronischen Edition.

Hinzu kommt, daß ich, von wenigen Ausnahmen wie großen Nachschlagewerken vielleicht abgesehen, nicht an das elektronische Buch als Ersatz für das gebundene Buch glaube – wenn ich den *Ulysses* einfach lesen will, werde ich weiterhin an das Bücherregal und nicht an den Bildschirm gehen. Die elektronische Edition ist in meinen Augen ein zusätzliches Werkzeug, eventuell auch die Datengrundlage für die Herstellung einer Druckausgabe; die wichtigste Eigenschaft dieses Werkzeugs ist die Flexibilität, mit der es für unterschiedliche Fragestellungen gebraucht werden kann. Auch von daher empfiehlt sich die Verwendung eines offenen Standards zur Erstellung eines Dokuments, das dadurch von unterschiedlicher existierender oder ad hoc anzupassender Software unterschiedlich ausgewertet werden kann. SGML im allgemeinen, und die TEI P3 DTD im besonderen, bieten einen solchen Standard an; sich an ihn zu halten bietet einem Kompatibilität mit vielen existierenden Projekten,

[14] Der offizielle Name des von Michael Sperberg Mc-Queen und Lou Burnard herausgegebenen Dokuments ist *Guidelines for Electronic Text Encoding and Interchange*. Die WWW-Adresse der TEI ist '`http://www.uic.edu/orgs/tei`', dort ist auch eine Online-Version des *Green Book* zugänglich. Eine *mailing-list* zur TEI-Codierung wird ebenfalls bei der *University of Chicago* betrieben.

[15] Dazu gehört zum Beispiel die `WordCruncher`-Edition des literarischen Nachlasses von Robert Musil, die 1992 auf CD-Rom erschienen ist, und einem für DM 1400,– die unter Verwendung von MSDOS-Sonderzeichen transkribierten Manuskripte unter einem MSDOS-Retrieval-Programm zur Verfügung stellt. Ansonsten ist die nur in elektronischer Form vorliegende Edition aber sorgfältig und interessant gestaltet, und sie wäre auch relativ leicht in ein SGML-Modell zu transformieren, wenn sich jemand die Mühe machte.

Archiven und Programmen. Außerdem zwingen einen die von außen herangetragenen Kategorien zum präzisen Durchdenken der eigenen Konzeption und machen auf übersehene Details aufmerksam.

Der andere Pol meiner Arbeit war die Datenbasis, die mir zur Verfügung stand: der gedruckte *Ulysses* und die Dateien aus der Druckvorbereitung, die ich von Herr Gabler erhalten hatte. Ich hatte also nicht die Freiheit, ein Editionsprojekt von Grund auf zu entwerfen und durchzuführen (wofür der Bearbeitungszeitraum einer Diplomarbeit auch ein wenig kurz gewesen wäre), sondern hatte eine fertige Edition vorliegen, die in eine elektronische Form zu transformieren war.

Die Komplexität des Materials, das Hans Walter Gabler zur Verfügung stand, wurde für die Buchform in einer dieser Form angemessenen Weise aufbereitet; für die elektronische Form wäre möglicherweise eine andere Art der Aufbereitung optimal. Hätte ich eine elektronische Edition von Grund auf geplant, wären die Basis sicher Abbildungen der erhaltenen Dokumente mit exakten Transkriptionen gewesen; mit diesen verknüpft wäre dann als Zugabe ein kritisch editierter Text zugänglich. So ähnlich werden auch tatsächlich viele elektronische Editionen heute durchgeführt bzw. projektiert, wobei der kritisch editierte Text nicht immer dazugegeben wird – teils aus prinzipiellen Gründen, teils weil dieser scheinbar unspektakuläre Schritt der personal- und kostenaufwendigste Teil des Editionsprojekts wäre. Außerdem existieren von den meisten der bearbeiteten Werke bereits edierte Texte, und eine Hauptaufgabe der elektronischen Editionen ist die Bereitstellung der Quellen für ein breiteres Publikum.[16]

7.1. *Die Codierung des Metatextes*

Sobald mir in vollem Umfang klar war, daß die Schichten der synoptischen Darstellung des *continuous manuscript text* nicht einfach eine Transkription der zugrundeliegenden Textzeugen darstellen, habe ich mich von der Vorstellung verabschiedet, das Material der Gabler-Edition als Steinbruch zu benutzen, um daraus eine elektronische Dokumentation der Textzeugen zu generieren. Ich habe mich stattdessen entschlossen, das Editionskonzept der Gabler-Edition in vollem Umfang beizubehalten und es durch die elektronische Form effektiver und vielseitiger nutzbar zu machen. Dazu waren aber Kompromisse mit dem TEI Standard nötig, denn die Informationsfülle von textgeschichtlicher Synopsis, Emendationen, *Textual Notes* und historischer Kollation wären nur unter Informations- und Strukturverlust in die von der TEI vorgesehenen Strukturen zu transportieren gewesen.

Also habe ich mich bei der Gestaltung des elektronischen Metatextes, wo immer es nötig schien, rigoros von den Modellen der TEI getrennt. Dadurch erst wurde es andererseits auch möglich, die einzelnen Stufen des *continuous manuscript text* auf eine Weise vollautomatisch extrahierbar zu machen, daß die einzelnen Textstufen auch in Textstrukturen wie Überschriften, Absätzen oder Lyrikblöcken ein konsistentes SGML-Dokument bilden.

Die SGML-Form des Metatextes sollte, zusammengefaßt, folgende Zugangsweisen ermöglichen:

1. Den Zugriff auf den Lesetext mit Referenzsystem; dies ist für Anwendungen, die an der Textgeschichte nicht interessiert sind, der einzige Sinn der elektronischen Ausgabe.
2. Den Zugriff auf beliebige andere Textschichten (im Sinne der Gabler-Edition) in einer dem Lesetext entsprechenden Form. Es sollte also möglich sein, ein korrektes, TEI-konformes SGML-Dokument jeder Entwicklungsstufe des *Ulysses* zu erzeugen.
3. Eine Darstellung des synoptischen Textes, in etwa den linken Seiten der gedruckten Ausgabe entsprechend; dabei sollte jede zu einem bestimmten Punkt

[16] Das "Early American Fiction" Projekt der *University of Virginia* ist ein umfangreiches Unternehmen dieser Art, siehe '`http://etext.lib.virginia.edu/eaf/`'.

des Textes gehörende Information auch von diesem Punkt aus zugänglich sein (die graphisch bzw. über Referenzangaben gegebene Verbindung vom Text über Fußnoten zu den *Textual Notes*, wo wiederum Details zu den Siglen der Textschichten stehen, aber auch die historische Kollation soll also als automatisch nutzbare Informationsstruktur bereitgestellt werden).
4. Die inneren Strukturen vor allem der Fußnoten und der historischen Kollation sollen soweit wie möglich in explizite Informationsstrukturen umgewandelt werden, ohne die informellen Elemente zu beseitigen oder zu formalisieren (letzteres ist automatisiert nicht durchführbar).

Vor allem die zweite Anforderung geht an die Grenzen dessen, was in SGML sinnvoll zu realisieren ist, denn es erfordert eine Art von Meta-SGML, um auch strukturelle Änderungen des Textes (Zusammenziehen von Absätzen, Umwidmung von Prosa in Lyrik und dergleichen) so festzuhalten, daß jede Textschicht in sich ein strukturell sinnvolles SGML-Dokument ist. Gleichzeitig soll aber auch das Gesamtdokument ein korrektes und verwendbares SGML-Dokument sein. Die gewählten Lösungen, die ab S. 63 beschrieben werden, sind sicher kritisierbar, scheinen aber das Geforderte zu leisten. Die Problematik wird dort ausführlich diskutiert.

Die Abwendung vom TEI-Standard erscheint mir insofern gerechtfertigt, als das vollständige SGML-Dokument in jedem Fall zu komplex geworden wäre, um beispielsweise in einem *browser* mit einem TEI-angepaßten *style sheet* aus dem Stand darstellbar zu sein; eine gewisse Einarbeitung und Anpassung wäre so oder so nötig gewesen. Die gewählte Struktur macht dies hoffentlich leichter, da sie die Funktion der einzelnen Elemente des elektronischen Metatextes besser deutlich und unterscheidbar macht. Sollte sich aber, durch künftige Erweiterung des Standards oder durch bessere Ideen, eine Möglichkeit ergeben, den *Ulysses* ohne solche Erweiterungen zu codieren, wäre ich der erste, der das begrüßen würde.

Der TEI-Standard sieht einen Mechanismus der geordneten Modifikation und Erweiterung der DTD vor (*Green Book* Kap. 29, S. 737–744), mit dem sich bestehende *content models* modifizieren und neue in die DTD einhängen lassen. Die für den Metatext vorgenommenen Modifikationen verwenden diesen Mechanismus, und sie bilden eine saubere Erweiterung der TEI DTD insofern, als die *Ulysses*-DTD jedes Dokument akzeptieren würde, das von der unmodifizierten TEI Standard-DTD akzeptiert wird. Von daher sind meine Modifikationen, dem Buchstaben nach, wohl auch TEI *conformant* im Sinne des *Green Book* (siehe S. 738), dem Geiste nach allerdings kann die *Ulysses*-Ausgabe nur noch bedingt TEI *conformance* für sich beanspruchen.

⋆

Die beiden Textformen der gedruckten Edition, Lesetext und synoptische Darstellung des *continuous manuscript text*, brauchen in der elektronischen Fassung nicht getrennt zu werden; der Lesetext kann ohne weiteres aus der komplexeren Textform erzeugt werden, indem alle Einfügungen vorgenommen und alle Streichungen übergangen werden. Da für viele Anwendungen der Lesetext von vorrangigem Interesse sein wird, habe ich mich bemüht, ihn durch die Form der SGML-Codierung leicht zugänglich zu machen. Sowohl historische Kollation als auch emendatorische Fußnoten sind mit in den Textstrom integriert, was zwar die Lesbarkeit der SGML-Dateien erschwert, aber die automatische Verarbeitung erleichtern dürfte. Die Referenzsysteme für beide Textformen des gedruckten Buchs sind erhalten, so daß sowohl der Lesetext wie die synoptische Darstellung vollständig aus dem SGML-Text generiert werden können.

Die *textual notes* sind in der elektronischen Ausgabe im Anhang enthalten. Vom übrigen Material im Anhang der Buchausgabe wurden nur die bibliographischen Angaben zu denjenigen *Ulysses*-Ausgaben verwendet, die in die historische Kollation einbezogen sind, außerdem wurden die Datierungsinformationen aus der Tabelle S. 1914 und den *textual notes* implizit verwertet.

Während der editorische Metatext also in erster Linie den Strukturen der Buchausgabe verpflichet ist, sind der Lesetext und alle anderen Textformen des eigentlichen *Ulysses* so streng wie möglich TEI-konform. Wo nötig, habe ich die DTD allerdings den Richtlinien von Kapitel 29 des *Green Book* folgend erweitert, um den Text adäquat zu repräsentieren. Beispielsweise wäre in Zeilen angeordnete Prosa im reinen Standard nur als Lyrik oder als Folge von Absätzen auszuzeichnen gewesen; analog zu den vorhandenen Tags `<lg>` (*line group*, also Strophe o.ä.) und `<l>` (*line*) habe ich deswegen die Tags `<display>` und `<dline>` hinzugefügt. Bei Bedarf ließen sich alle solche Erweiterungen auf strukturell gleiche Tags des reinen TEI Standards abbilden, zu jedem neuen Element gibt es ein grammatisch äquivalentes Standardelement.

Die TEI sieht sehr unterschiedliche Tiefen von Auszeichnung vor, bis hin zur linguistischen Analyse jedes einzelnen Wortes. Ich habe mich mit wenigen Ausnahmen auf Auszeichnungen beschränkt, die automatisch durchzuführen waren und die zu einem sinnvollen und konsistenten Codesystem wenigstens innerhalb eines Kapitels führen. Die Beschränkung auf automatische Konvertierung war aus Zeitgründen nötig, und sichert auch gegen unsystematische Fehler bei der Bearbeitung ab; es wäre mit mehr Zeit aber sicher noch einiges Sinnvolle zu tun. Andererseits würde ich, bevor ich mit Eifer und großem Zeitaufwand eine detaillierte logische Codierung in Angriff nähme, ein paar grundsätzliche Fragen über SGML-Codierung literarischer Texte zu klären oder zumindest aufzuwerfen versuchen. Darauf werde ich im Folgenden etwas näher eingehen.

⋆

In einem literarischen Werk wie dem *Ulysses* sind die äußere Form und der Inhalt nicht ohne weiteres trennbar, und das ist von grundsätzlicher Bedeutung für den Entwurf einer elektronischen Ausgabe. Es ist die mehr oder weniger ausgesprochene Grundlage der Arbeit mit SGML und anderen Formen von 'abstrakten' Dokumenten, daß es eine klare Grenzlinie gibt zwischen dem Inhalt und seiner logischen Struktur einerseits und der konkreten typographischen Wiedergabe andererseits. Wer heute beispielsweise eine technische Dokumentation schreibt, wird deswegen auf SGML zurückgreifen, weil dadurch die Information auf verschiedenste Weise dargestellt, gefiltert, umgeformt werden kann. Dieser Ansatz ist nicht auf SGML beschränkt. So schreibt der Autor des Satzsystems `latex`: "As you are writing your document, you should be concerned with its logical structure, not its visual appearance" (Lamport, S. 7). Während logische Auszeichnung des Textes in `latex` (das zwar logische Formatierung durch Makros ermöglicht, aber immer eine Programmiersprache zur visuellen Dokumentformatierung bleibt) ein Ideal ist, das in der Praxis beliebig dehnbar gehandhabt werden kann, ist "logische Formatierung" in SGML wesensmäßig eingebaut. Nur durch die als Notbehelf vorgesehenen *Processing Instructions* kann die SGML-Auszeichnung die Formatierung unmittelbar beeinflussen, ansonsten wird selbst ein Element wie `<hi rend=italics>` nur dann tatsächlich zu kursivem Text, wenn die nachgeschaltete Konvertierung das Element entsprechend umsetzt.

Wie kaum anders zu erwarten, bekennt sich Charles Goldfarb, der maßgebliche Erfinder von SGML, klar zu dieser Eigenschaft als Stärke des Konzepts:

> Markup should describe a document's structure and other attributes rather than specifying processing to be performed on it...
>
> (Goldfarb, S. 9–10)

> In generalized markup, the term "document" does not refer to a physical construct such as a file or a set of printed pages. Instead a document is a logical construct that contains a document element, the top node of a tree of elements that make up the documents content.
>
> (Goldfarb, S. 18)

Nun gibt es auch in modernen Vielzweckdokumenten Elemente, die in einer bestimmten Anordnung oder Formatierung dargestellt werden müssen und selten rein logisch codiert werden, wie Tabellen, mathematische Formeln oder Grafiken. Der Autor solcher Dokumente weiß aber, daß diese Elemente besonders gekennzeichnet werden müssen, weil sein Quelldokument auf verschiedenste Weise aufbereitet werden wird. Und wenn Autoren solcher Vielzweckdokumente explizit über die typographische Gestaltung ihres Textes schreiben müssen, beispielsweise um im Handbuch eines Programms zu erklären, wie Tastatureingaben dargestellt sind, greifen sie oft auf Tricks zurück und schreiben: "Tastatureingaben werden `so wie dieses Beispiel` dargestellt."

James Joyce hingegen wußte nichts von meiner SGML-Edition, er schrieb konkret für eine bestimmte Form von gedrucktem Buch, dessen Entstehung er anhand der Druckfahnen kritisch begleiten konnte. Er kannte die Konventionen des Buchdrucks, akzeptierte sie oder widersetzte sich ihnen, wo sie ihm mißfielen, und nutzte die visuelle Form gezielt, teilweise spielerisch, als Komponente des fertigen Textes, wie ihn die Leser zu sehen bekommen würden. Deswegen muß die Anwendbarkeit von Charles Goldfarbs eben zitierten Prämissen auf Bücher wie den *Ulysses* kritisch gesehen werden.

Ein kleines Detail als Beispiel: Wenn aus dem offiziellen Aufkleber "POST NO BILLS" durch Wegkratzen die Aufforderung "POST IIO PILLS" wird (*c8 ℓ101*), funktioniert dieser Scherz nur dann überzeugend, wenn die Typographie darauf sorgfältig abgestimmt ist; zumindest muß der Text des Aufklebers in gleichhohen Großbuchstaben und die Zahl "IIO" nicht in gewöhnlichen Ziffern gesetzt werden, damit der Sinn dieser Stelle nicht völlig verlorengeht. In der eben wiedergegebenen Form wirken die unterschiedlichen Serifen, vor allem am zweiten 'I', und die unterschiedliche Dicke der senkrechten Linien in 'N' und 'I' bereits störend, aber man versteht den gemeinten Effekt. Treffender wäre es eventuell, den Aufkleber in einer endstrichlosen Schrift mit gleichmäßiger Strichstärke wiederzugeben (was auch eine typische Schriftgestalt für solche Aufkleber sein könnte), oder die erforderlichen Buchstaben 'N' und 'B' mit entfernten Linien extra zu erzeugen; aber solche Extravaganzen können sich nicht mehr auf Joyce berufen, der das keineswegs bessere Druckbild der Erstausgabe offenbar gelten ließ und also bereit war, dem Leser ein wenig Abstraktion abzufordern. Es ist jedenfalls die graphische Form, von der logische Textauszeichnung ja abstrahieren möchte, die hier ein wesentlicher Teil des Inhalts ist.

Der Herausgeber einer SGML-Edition ist noch stärker als seine konventionell arbeitenden Kollegen zur Bildung klarer Kategorien und zu interpretierenden Maßnahmen gezwungen. Die TEI erkennt dieses Problem an, indem sie sich ihm (berechtigterweise) entzieht:

> In these Guidelines, no hard and fast distinction is drawn between "objective" and "subjective" information or between "representation" and "interpretation". These distinctions, though widely made and often useful in narrow well defined contexts, are perhaps best interpreted as distinctions between issues on which there is a scholarly consensus and issues where no such consensus exists. Such consensus has been, and no doubt will be, subject to change.
>
> (*Green Book*, S. 6)

Aus solchen Beobachtungen muß man natürlich nicht folgern, daß die logische Textauszeichnung von Literatur überhaupt sinnlos und falsch ist. Aber Vorsicht und Behutsamkeit bei der Repräsentation eines konkreten Textes in abstrakten Kategorien und ein möglichst klares Bewußtsein für die interpretierenden Eingriffe, die man dabei vornimmt, halte ich für wichtig. Sie bestimmen in großem Maß den Wert und die Nutzbarkeit der geleisteten Arbeit.

⋆

Um zusammenzufassen und ins praktische Leben zurückzukehren: Während SGML auf eine abstrakte Dokumentformatierung ausgerichtet ist, ist in der zu konvertierenden Quelle die Grenzlinie zwischen hinweg zu abstrahierender Form und zu erhaltendem Inhalt keineswegs klar. Da es sich bei meiner Arbeit nicht um die unabhängige Neuerstellung einer Edition aus den Originaldokumenten handelt, sondern um die Umsetzung einer bestehenden Edition ins elektronische Medium, war mir schon durch meine Vorlage eine gewisse Abstraktionsstufe vorgegeben.

Von daher war mein Leitziel, die strukturellen Möglichkeiten der SGML-Edition zu nutzen um die in der Gabler-Ausgabe geleisteten Abstraktionen zu erschließen, dabei aber möglichst wenig von der implizit im Layout enthaltenen Information verloren gehen zu lassen.

Die TEI-Elemente haben alle ein `rend`-Attribut zur Angabe der graphischen Gestaltung; darauf habe ich zurückgegriffen, um Schrift und Absatzformen anzugeben. Die Schlüsselwörter für das `rend`-Attribut definieren gewissermaßen den typographischen Code, mit dem sich der Text auf der Ebene des Layouts mitteilt.[17] Komplizierte Sonderfälle habe ich in einem Anhang verbal beschrieben und diese Beschreibungen mit `IDREF`s an die entsprechenden Elemente im Text geknüpft.

Andererseits habe ich durch Typographie gestaltete Strukturelemente wo immer möglich als Strukturen codiert – das fünfzehnte Kapitel, das in Dramenform geschrieben ist, ist zum Beispiel mit den TEI-Tags für Drama ausgezeichnet, und nicht nur als eine Folge von unterschiedlich formatierten Prosazeilen. Diese logische Auszeichnung erschien mir aber nur insoweit sinnvoll, als sich wirklich klare funktionale Einheiten abgrenzen ließen. So würde ich die Auszeichnung von fremdsprachigen Elementen im Text, etwa noch mit Angabe der Sprache in einem Attribut, für zwar zu 95 Prozent möglich, aber doch problematisch halten, da die Grenze zwischen fremdsprachigen, englischen und erfundenen Wörtern nicht überall einwandfrei zu ziehen ist. In der parodistischen Schilderung einer spiritistischen Sitzung (*c12 ℓ338–373*) fallen beispielsweise neben den Begriffen "Māyā" und "prālāyā" auch solche wie "tālāfānā" (telephone) oder "wātāklāsāt" (water closet). Sollte man die letzgenannten Formen als verballhorntes Englisch auffassen, oder als verballhorntes Sanskrit? In jedem Fall könnte ein Benutzer der Edition, der sich auf die Auszeichnung der Sprachen verläßt (zum Beispiel um eine Liste der im *Ulysses* verwendeten fremdsprachigen Ausdrücke zu erstellen) in die Irre geführt werden.

An anderer Stelle erscheint in einer langen Liste von sprechenden Eigennamen als Vertreter der Deutschen ein "Nationalgymnasiummuseumsanatoriumandsuspensoriumsordinaryprivatdocentgeneralhistoryspecialprofessordoctor Kriegfried Ueberallgemein" (*c12 ℓ566–569*). Hier sind zum einen zwei Sprachen endgültig unentwirrbar verflochten, zum anderen wirkt es unfreiwillig komisch, die von der TEI bereitgestellten Tags für Namen und Anrede dieses Herrn anzuwenden, der in keiner Weise eine Figur der Handlung ist.[18]

Während man die genannten Beispiele als unbedeutende Extremfälle abtun kann, sind andere Anforderungen an eine SGML-Edition des *Ulysses* von zentraler Wichtigkeit für die philologische Erschließung: Die unterschiedlichen Erzähler in den einzelnen Kapiteln und die Sprecher der direkten Reden unterscheiden sich oft charakteristisch in ihrem Stil. Die Auszeichnung von wiedergegebener direkter Rede und Erzählertext und möglicherweise die Zuordnung der direkten Reden und inneren Monologe zu Personen der Handlung erscheint deswegen sicherlich für philologische Arbeit wünschenswert.

Die saubere Abgrenzung direkter Rede ist in einigen Kapiteln gut möglich, in anderen nahezu überhaupt nicht. Die Identifikation von Personen der Handlung und die Markierung ihres Auftretens und ihrer Figurenreden wäre weitgehend möglich und sicher oft hilfreich. Im fünfzehnten Kapitel allerdings, in dem auch Figuren wie "THE SOAP" oder "THE END OF THE WORLD" auftreten (*c15 ℓ337* bzw. *c15 ℓ2180*) und die Hauptfigur Leopold Bloom zwischenzeitlich ihr Geschlecht wechselt, erscheint auch die Festlegung auf abgegrenzte Personen der Handlung problematisch.

[17] Sie sind auf S. 51 zusammengestellt.

[18] Als beständige Mahnung habe ich folgende TEI-Stilblüte im Hinterkopf, die Peter Flynn am 10. Novemeber 1995 der TEI *mailing list* mitgeteilt hat: …`hanged, drawn and <num val=0.25>quartered</num>`…

Generell erscheint es mir ehrlicher, eine unscharfe Kategorie überhaupt nicht anzuwenden, als durch ihre Anwendung eine scharfe Trennbarkeit zu suggerieren. Aber natürlich stand neben allen prinzipiellen Überlegungen das banale aber entscheidende Kriterium *sine qua non* für die abstrahierende Auszeichnung regelmäßiger Phänomene im Lesetext: die Frage, ob eine automatisierte Codierung möglich war, da die Bearbeitungszeit einer Diplomarbeit eine gründliche und kritische Handauszeichnung des gesamten *Ulysses* von vorneherein ausschließt.

7.3. *Die Struktur der elektronischen Ausgabe*

Die elektronische Ausgabe des *Ulysses* ist ein auf 26 Dateien verteiltes SGML-Dokument von knapp acht Megabyte Umfang; dazu kommen noch drei gescannte Grafiken mit den im Text vorkommenden Liednoten, `gloria.tif`, `harry1.tif` und `harry2.tif`, eine Datei `README.uly` und die TEI-konforme Dokumentation der *Ulysses*-spezifischen Erweiterungen der DTD in `uly_tag.sgml`.

Die 'Masterdatei' des SGML-Dokuments ist die Datei `ulysses.sgml`; sie enthält den TEI-Header, der das elektronische Dokument und seine Quelle beschreibt, und von dieser Datei aus werden alle anderen Bestandteile über *entity references* aufgerufen. Diese Datei ist im Anhang vollständig abgedruckt.

Jedes der achtzehn Kapitel des *Ulysses* ist in einer eigenen Datei enthalten, `uly01.sgml` bis `uly18.sgml`. Der Lesetext ist nicht gesondert abgelegt, sondern ist in den Textstrom integriert, ebenso die emendatorischen Fußnoten und die Einträge der historischen Kollation. Die *Textual Notes* sind in der Datei `uly_tn.sgml` gesammelt abgelegt, aber vom Text aus über `IDREF`s zu identifizieren.

Die vorgenommenen Modifikationen und Erweiterungen der TEI DTD sind in den Dateien `uly_ext.dtd` und `uly_ext.ent` enthalten; ich habe bei diesen Modifikationen die Richtlinien des *Green Book* befolgt.

Diejenigen Sonderzeichen, die nicht in den *character sets* in ISO 8879 enthalten sind (vergleiche Goldfarb, S. 505–529), sind als `SDATA` *enitities* in der Datei `uly_new.ent` definiert.

Eine formalisierte Kurzbeschreibung der synoptischen Textschichten jedes Kapitels wurde aus im Buch vorhandenen Informationen neu zusammengestellt, sie befindet sich in der Datei `uly_syn.sgml`. Alle Textschichten im synoptischen Text werden durch `IDREF`s bezeichnet, die sich auf die `ID`s dieser Kurzdefinitionen beziehen.

Für die Elemente des Textes, die vermutlich eine Nachbearbeitung oder Sonderbehandlung des Layouts benötigen, befinden sich informelle Beschreibungen des Layouts im Anhang; diese Beschreibungen sind über `IDREF`s mit den betreffenden Textstellen verknüpft. Sie stehen in einer `<list>` zusammengefaßt in der Datei `uly_spec.sgml`.

Ebenfalls im Anhang steht ein Verzeichnis der gedruckten *Ulysses*-Ausgaben, auf die sich die historische Kollation mit `IDREF`s bezieht. Dieses Verzeichnis befindet sich physisch in der Datei `uly_bib.sgml`.

⋆

Ein TEI-Dokument besteht normalerweise aus einem `<TEIheader>`, in dem das elektronische Dokument und seine gedruckte Quelle (falls vorhanden) mehr oder weniger detailliert beschrieben werden, und dem eigentlichen `<text>`, der das elektronische Dokument enthält.

Der TEI Header des *Ulysses* ist im Anhang, als Teil von `ulysses.sgml`, vollständig wiedergegeben, die Struktur des TEI Headers generell ist im *Green Book*, Kapitel 5 (S. 89–137) ausführlich beschrieben, ich verzichte daher hier auf eine nähere Beschreibung.

Ein TEI `<text>` besteht aus einem optionalen `<front>`-Element (*front matter*, also Vorworte und dergleichen), dem `<body>`-Element, das den Haupttext enthält, und dem optionalen `<back>`-Element (*back matter*). Ich habe von `<front>` keinen Gebrauch gemacht; `<back>` enthält drei nicht zum eigentlichen Text gehörende Anhänge, nämlich die *Textual*

Notes, die bibliographischen Angaben zu den in die historische Kollation eingegangenen Ausgaben und eine formalisierte Beschreibung aller synoptischen Level. Der `<body>` enthält den Text des *Ulysses*.

8. Der elektronische Lesetext

In diesem Abschnitt behandle ich die SGML-Strukturen, die der Repräsentation des Lesetextes dienen, die Gliederung und die Überschriften sollten die Bezüge zur Beschreibung des gedruckten Lesetextes deutlich werden lassen. Der Lesetext ist in der elektronischen Edition mit dem Metatext verflochten, und die resultierenden Dateien sind an vielen Stellen unübersichtlich. Ich habe die SGML-Beispiele deswegen durch Weglassen von Elementen so vereinfacht, daß die jeweils besprochenen Codes in einem übersichtlichen Kontext vorgestellt werden können.

Das `<body>`-Element enthält also den Text des Romans; er besteht aus achtzehn Episoden, die wiederum zu drei Teilen gruppiert sind; die TEI stellt dafür die schachtelbaren `<div>`-Tags zur Verfügung (alternativ dazu auch die Tags `<div1>`, `<div2>` und so weiter, mit expliziter Bennenung der Schachtelungstiefe, das erschien mir aber unnötig). Die drei Teile haben ihre römischen Ziffern, die Episoden ihre homerischen Überschriften als `<head>` vorangestellt. Zusätzlich sind die Episoden mit dem `n`-Attribut von 1 bis 18 durchnumeriert; diese Zahlen sind immerhin Bestandteil des Referenzsystems für den Lesetext. Der elektronische Text des Ulysses beginnt also:

```
<text lang=eng>
<body>
<div type=part>
  <head><num value=1>I</num></head>
  <div type=episode>
    <head>Telemachus</head>
      &uly01;
  </div>
```

Die *entity reference* `uly01` steht dabei für den Text der ersten Episode; die weiteren Episoden und Teile schließen sich entsprechend an.

Die Zeilenreferenz des Lesetextes wird, ebenso wie die Seiten- und Zeilenreferenz der synoptischen Darstellung, durch sog. *milestone tags* im elektronischen Text mitgeführt, also durch Elemente ohne Inhalt, die durch ihre bloße Anwesenheit und zusätzliche Attributwerte Strukturinformation repräsentieren. Die TEI stellt für diesen Zweck die Tags `<lb ed=.. n=..>` für Zeilenbeginn und `<pb ed=.. n=..>` für Seitenbeginn zur Verfügung; der Lesetext hat als Code für das Attribut `ed` (Edition) die ID `gab`, der synoptische Text die ID `syn`. Für diese Elemente des Referenzsystems kann nur garantiert werden, daß sie irgendwo zwischen dem letzten Stück Text der vorherigen und dem ersten Stück Text der folgenden Zeile erscheinen; wo genau (zum Beispiel innerhalb oder außerhalb eines `<p>`-Elements) ist aus technischen Gründen nicht festzulegen gewesen.

8.1. *Der Zeichensatz*

Der *document character set* der *Ulysses*-Ausgabe ist der 7-Bit ASCII-Zeichensatz. Gelegentlich wird empfohlen, nur eine Teilmenge dieses Zeichensatzes tatsächlich zu verwenden, um größtmögliche Austauschbarkeit zu gewährleisten, ich bin generell aber von der Verfügbarkeit des gesamten ASCII-Zeichensatzes ausgegangen. Einige der ASCII-Zeichen haben syntaktische Funktionen in SGML und konnten natürlich nicht als Textzeichen verwendet werden.

Nicht unmittelbar im 7-Bit-ASCII-Code verfügbare Zeichen wurden zunächst in den *character entity sets* des ISO-Standards gesucht; die in diesen Zeichensätzen nicht definierten Sonderzeichen werden in der Datei **uly_new.ent** neu definiert. Einige der definierten Zeichen erscheinen vielleicht redundant oder überhaupt durch *entity references* fehlcodiert, zum Beispiel die Brüche, die Ellipsen mit unterschiedlich vielen Punkten oder der **dotfill**, mit dem die Zeilenanfänge des auf S. 23 zitierten Gedichts aufgefüllt sind. Ich habe mich für diese Form der Codierung entschieden, weil ich einen abgeschlossenen Text vor mir hatte, der mich beispielsweise nicht mit immer neuen Bruchzahlen überraschen würde, weil ich keine unnötigen Erweiterungen des TEI-Standards einführen wollte (beispielsweise **<fraction num=.. den=..>**), und weil es leichter ist, einen relativ reichen, aber wohldefinierten Satz von *entities* in eine andere, vielleicht strengere Codierung umzusetzen, als in einer minimalistischen Codierung bestimmte Muster zu finden.[19] Ich hätte hier vielleicht sogar noch weiter gehen, und beispielsweise die Kombination aus großem M und linkem Anführungszeichen in irischen Namen wie "M'Coy" als eigene *entity* definieren sollen; bei einem eventuellen Satz von den SGML-Dateien wären so spezielle Druckanordnungen erleichtert, und eine Verwechslungsgefahr mit anderen möglichen Verwendungen des linken Anführungszeichens wäre vermieden.

Daß ich die Codierung nicht in diesem Sinne weiter verfeinert habe, lag daran, daß ich den Text noch einigermaßen verläßlich handcodierbar und lesbar halten wollte, um Korrekturen vornehmen zu können. Auch hätte jede durch detailliertere Codierung gegebene Zusatzinformation, die nicht schon in den Quelldateien zur Verfügung stand, bevor sie automatisch erzeugt worden wäre, erst manuell auf ihre Korrektheit überprüft werden müssen, wie ich es im Falle der Anführungszeichen getan habe: es kommen im Lesetext linke Anführungszeichen nur in der angegebenen Funktion in irischen Eigennamen vor. Dieser Aufwand erschien aber generell nicht gerechtfertigt.

Folgende *entities* wurden also neu definiert (vergleiche auch die Sonderzeichenliste auf S. 22) – der Einfachheit halber gebe ich auch solche Zeichen, die nur für die Codierung des Metatextes nötig waren, hier mit an:

frac27	$^2/_7$	**frac511**	$^5/_{11}$
aoverc	$^a/_c$	**longs**	ſ
highc	^{c}	**highr**	^{r}
highd	^{d}	**ddot**	ḍ
highnd	nd	**highrd**	rd
highs	^{s}	**highst**	st
highth	th	**highv**	^{v}
low2	$_2$	**low4**	$_4$
oelig	œ	**dotteds**	ṡ
kcaron	ǩ	**hslash**	ħ
wslash	w̸		
ell2	..	**ell5**	.....
ell3	...	**ell6**	......
ell4	....	**ell7**	....
ell14	....	**tristar**	* * *
negkern	– kleine Unterschneidung (*negative kerning*) –		
dotfill	– variable Füllung mit Punkten, vgl. S. 23 –		

Aus den ISO-Zeichenlisten wurden verwendet:

[19] Bei der Entscheidungsfindung in dieser und einigen anderen Fragen haben mir die Teilnehmer der **TEI-L** *mailing-list* durch ihre Beiträge geholfen. Ich möchte mich dafür herzlich bedanken.

Latin 1:	`aacute Aacute agrave auml aelig AElig ccedil Ccedil eacute Eacute egrave Egrave ecirc icirc ntilde oacute ocirc ograve ouml uacute ucirc ugrave uuml Uuml`
Latin 2:	`amacr rcaron scaron`
Numeric:	`frac12 frac14 sup2 plus pound percnt amp ast rsqb lsqb verbar deg apos hyphen sol lsquo nbsp shy`
Publishing:	`thinsp hairsp mdash ndash frac13 frac16 incare bull malt dagger caret`
Technical:	`lang rang sim`

Trennstriche, die in der Druckausgabe aus satztechnischen Gründen eingeführt wurden, werden als `shy` (*soft hyphen*) codiert, ebenso die in den Quelldateien bereits markierten möglichen Trennstellen in extrem langen Wörtern.

8.2. *Leerraum*

Bei der Behandlung von Leerraum war ich bis zum Schluß unsicher, ob ich rein logisch codieren, also einfach den spezifischen Leerraum zur Worttrennung codieren sollte, oder ob ich die unterschiedlichen Kontrollsequenzen der Quelldateien für alle verschiedenen Formen von Leerraum in entsprechende SGML-Tags umsetzen sollte, um keine Information der Druckausgabe zu verlieren. Obwohl allzugroße Detailliertheit die anderweitige Benutzung und Nachbearbeitung der Dateien erschwert hätte, ein Druck nach den SGML-Dateien nicht vorgesehen ist und schließlich die Details der typographischen Leerraumbehandlung auch von der gewählten Schriftart und anderen Details des konkreten Kontexts abhängen, habe ich mich zu einer Codierung durch mehrere Entity References entschlossen, die aber eindeutig zum logischen oder zum rein typographischen Leerraum gerechnet werden können.

Logischer Leerraum wird im Allgemeinen durch SGML *space tokens* codiert, also durch ASCII Leerzeichen oder Zeilenwechsel. Die Frage, wann ein Leerraum, insbesondere ein Zeilenwechsel, in einem SGML-Quelldokument auch einem Leerraum in den codierten Textdaten entspricht, ist nicht unkompliziert, ich habe die S. 321–324 bei Goldfarb gründlich durchgearbeitet und bei der Formatierung so korrekt und umfassend wie möglich berücksichtigt.

Weitere Formen von Leerraum sind `nbsp`, ein 'logischer' Leerraum, der aber nicht in einen Zeilenumbruch verwandelt werden sollte, sowie die nicht-logischen `thinsp` und `hairsp`, die aus typographischen Gründen in den Quelldateien erhalten sind und übernommen wurden. Ebenso wie `negkern`, eine leichte Unterschneidung, die nur im Song über *Harry Hughes* in *c17 ℓ802–828* auftritt, können diese nicht funktionalen Leerräume für logisch-funktional orientierte Anwendungen und auch für einfache Bildschirmdarstellung ignoriert werden, ich garantiere auch keine systematische Anwendung in mit der Hand nachbearbeiteten Teilen (für die automatische Konvertierung kann ich ohnehin nur die Übernahme einer eventuellen Systematik aus den Quelldateien versichern). Sie verhelfen aber bei Ausdrucken möglicherweise zu höherer Qualität und wurden deshalb beibehalten. Tatsächliche Maße für `thinsp` und `hairsp` kann ich nicht angeben, ich habe die Codezuordnung nur aus dem Vergleich von Quelldatei und Druckbild erraten.

8.3. *Schriftarten und Absatzformen*

Schriftarten und Absatzformen haben zwar oft eine logische Funktion im Dokument, und die SGML-Codierung zielt in erster Linie darauf ab, diese logische Funktion wiederzugeben. Da dies aus technischen und prinzipiellen Gründen in dieser *Ulysses*-Ausgabe aber nur bedingt möglich ist, und da es in jedem Fall sinnvoll ist, die 'typographischen Codes' des Textes in ihrer Gestalt zu dokumentieren, habe ich für dieses SGML-Dokument relativ großen Wert auf die eindeutige Benennung des Layouts gelegt. Die folgende Übersicht gibt die

Schlüsselwörter an, die in `rend`-Attributen verwendet werden, um die äußere Textgestalt auf der Zeichenebene und der Ebene des Absatzformats zu beschreiben.

`rm`	Normale lateinische Schrift, recte (wie dieses Beispiel);
`it`	Kursive Schrift (*wie dieses Beispiel*);
`sl`	Schräggestellte Schrift (*wie dieses Beispiel*);
`caps`	Versalien (WIE DIESES BEISPIEL);
`csc`	Kapitälchen mit Versalien (WIE DIESES BEISPIEL);
`slcsc`	Schräggestellte Kapitälchen mit Versalien (*WIE DIESES BEISPIEL*);
`bfit`	Fette Kursiv (für die Buchstaben ***POLDY***);
`mdash`	Repräsentiert den Geviertstrich (—), mit dem wörtliche Rede eingeleitet wird;
`indent`	Normaler, deutlicher Einzug der ersten Zeile (zwei Geviert in der Druckausgabe), der Rest auf volle Breite gesetzt;
`halfindent`	Einzug der ersten Zeile des Absatzes mit nur einem Geviert.
`noindent`	Kein Einzug der ersten Zeile, der gesamte Absatz ist auf volle Breite gesetzt;
`revindent`	Die erste Zeile ist auf volle Breite gesetzt, die Folgezeilen mit einem Geviert eingezogen;
`fullindent`	Der gesamte Absatz ist mit dem Absatzeinzug (zwei Geviert) eingezogen;
`indent25`	Mit zwei Fünfteln der vollen Breite eingezogen (einzelne Zeile);
`indent35`	Mit drei Fünfteln der vollen Breite eingezogen (einzelne Zeile);
`centered`	Eine Zeilengruppe ist zeilenweise zentriert;
`centerblock`	Eine Zeilengruppe ist als in sich linksbündiger Block auf die längste Zeile zentriert;
`inline`	Ein logischer Absatz (verwendet für Überschrift in den *Textual Notes*) endet nicht mit einem Zeilenwechsel, der folgende Text schließt sich direkt an;
`join`	Ein logischer Absatz beginnt nicht auf einer neuen Zeile, sondern schließt sich direkt an das Ende des vorhergehenden Textes an (für den auf `inline` folgenden Absatz in den *Textual Notes* verwendet);
`flushright`	Eine Zeilengruppe ist in sich rechtsbündig anzuordnen – wird nur für zwei Spezialfälle verwendet, deren Layout ohnehin Nachbearbeitung erfordert;

Da jedes TEI-Element ein `rend`-Attribut hat, kann die Textgestaltung für jedes Element exakt angegeben werden; wo immer ein Zweifelsfall auftreten könnte, wird das auch getan. Um die SGML-Tags nicht unnötig zu überladen, wird aber davon ausgegangen, daß ein Element, für das keine Spezifizierung des Layouts angegeben ist, die des Vorgängers im Baum (also des umgebenden Elements) übernimmt. Wird also in einem kursiven Kontext mit `<add>` Text eingefügt, dann ist dieser auch kursiv anzunehmen, wenn `<add>` kein eigenes `rend`-Attribut setzt. Damit folge ich also der Semantik des TEI-spezifischen *Default Value* `INHERITED`, nicht etwa dem Standard-SGML *Default Value* `CURRENT`. Der tatsächliche *Default Value* für `rend` ist aber `IMPLIED`.

Die etwas merkwürdige `inline-join`-Konstruktion war für die *Textual Notes* nötig, weil dort die Überschriften zu den Unterabschnitten der allgemeinen Bemerkungen zum Kapitel nur durch Kapitälchen ausgezeichnet den Anfang des entsprechenden Absatzes bilden. Da es sich nicht um den eigentlichen Text des *Ulysses* handelt, sind aber keine allzugroßen Skrupel bei der Umsetzung dieses Layouts erforderlich.

8.4. *Prosa*

Absätze sind im elektronischen Dokument die kleinsten Einheiten frei umbrochener Prosa; ein zentriertes Gedicht oder ähnliches unterbricht also einen Absatz nicht, sondern beendet den vorhergehenden Absatz; ebenso das Einsetzen einer wörtlichen Rede mit Geviertstrich auf einer neuen Zeile. Diese Auffassung von Absatz ist nicht die einzig mögliche, aber sie scheint der TEI DTD weitgehend zugrunde zu liegen, und sie ist klar zu definieren.

Die Absätze der Prosa sind durch das Element `<p>` wiedergegeben; Details der Absatzform werden durch das Attribut `rend` codiert (siehe oben). Gewöhnliche Prosaabsätze haben `rend=indent`, Absätze mit wörtlicher Rede haben `rend='noindent mdash'`. Der Geviertstrich wurde nicht im Text, sondern im Attribut codiert, weil er eher funktionale Markierung als echter Textbestandteil ist; ausserdem sollte der Attributwert `mdash` das Auffinden der Absätze mit direkter Rede für spätere, bessere Auszeichnung durch das TEI-Element `<q>` erleichtern.

Hervorhebungen durch andere Schriftarten sind mit dem Element `<hi>` codiert; das Attribut `rend` gibt die Schriftform an; im Lesetext mögliche Werte sind `it` für Kursiv, `rm` für Recte innerhalb von kursivem Kontext, `caps` für Versalien, `csc` (*caps and small caps*) für Kapitälchen mit Versalien, `sl` für schräggestellte lateinische Buchstaben (in den Bühnenanweisungen im fünfzehnten Kapitel), `bfit` für fette Kursiv (nur im Akrostichon *c17 ℓ412–416*).

Der Anfang des Buches lautet in der SGML-Form des Lesetextes:

```
<p rend=indent><lb ed=gab n=1>Stately,
plump Buck Mulligan came from the stairhead, bearing a
<lb ed=gab n=2>bowl of lather on which a mirror and a
razor lay crossed. A yellow
<lb ed=gab n=3>dressinggown, ungirdled, was sustained
gently behind him on the mild
<lb ed=gab n=4>morning air. He held the bowl aloft
and intoned:
</p>
<lb ed=gab n=5><p rend='noindent mdash'><hi rend=it>Introibo
ad altare Dei.
</hi>
</p>
```

8.5. *Lyrik*

Für Lyrik stellt die TEI die Elemente `lg` (*line group*) und `l` (*line*) zur Verfügung, zur `lg` ist noch die Art von Zeilengruppe (z.B. Strophe / Refrain) anzugeben. Auf die längste Zeile zentrierte Lyrik wird durch `<lg type=stanza rend=centerblock>` codiert, Lyrik als wörtliche Rede erscheint als `<lg type=stanza rend='noindent mdash'>` (Das Attribut `type` wurde nicht zu genauerer Differenzierung genutzt). Jede einzelne Zeile ist innerhalb dieser *line groups* als `<l rend=it>` codiert.

Die von Buck Mulligan in *c1 ℓ239–241* zitierten Zeilen sind im SGML-Text so codiert:

```
<lg type=stanza rend='noindent mdash'>
<l rend=it>And no more turn aside and brood</l>
```

```
<l rend=it>Upon love&apos;s bitter mystery</l>
<l rend=it>For Fergus rules the brazen cars.</l>
</lg>
```

Die durch Fettdruck hervorgehobenen Anfangsbuchstaben jeder Zeile des Akrostichons *c17 ℓ412–416* sind durch `<hi rend=bfit>` ausgezeichnet; die durch Punkte aufgefüllten Gedichtzeilen *c7 ℓ717–719* sind, als Behelfslösung, durch die *entity* `dotfill` wiedergegeben. Die Liednoten habe ich aus der Buchausgabe eingescannt und als Graphiken mit dem TEI-Element `<figure>` eingebunden; gemäß der TEI DTD enthält dazu das Attribut `entity` einen Verweis auf die Graphik als eine externe `NDATA` Entity; zusätzlich kann in einer `<figDesc>` eine Beschreibung der Graphik gegeben werden, die ausgegeben wird, wo das System eine graphische Wiedergabe nicht zuläßt. Das “Gloria” im neunten Kapitel erscheint so im Dokument:

```
<figure entity=graphic.gloria rend=centered>
<figDesc>The melody for the <q>Gloria in excelsis Deo</q>
in Gregorian notation.
</figDesc>
</figure>
```

Die Entity `graphic.gloria` und das Graphikformat sind in `ulysses.sgml` wie folgt definiert:

```
<!NOTATION tiff PUBLIC
  '-//Aldus Corp.//NOTATION Tagged Image File Format//EN'>
<!ENTITY graphic.gloria SYSTEM "gloria.tiff" NDATA tiff>
```

⋆

Die Passage *c9 ℓ684–700* (siehe S. 24) gehört zu den Fällen, bei denen ich die Waffen gestreckt habe. Solche Elemente im Buch haben eine `ID` erhalten, auf die im Anhang verwiesen wird, dort steht jeweils eine kurze Beschreibung des Druckbilds; dies soll helfen, solche Stellen zumindest identifizieren und bei der Ausgabe oder sonstigen Verarbeitung als Spezialfall behandeln zu können.

8.6. *Prosadisplays*

Für Prosadisplays habe ich keine adäquate Darstellungsform in der TEI DTD gefunden; ich habe deswegen die Elemente `<display>` und `<dline>` analog zu den Lyrikelementen `<lg>` und `<l>` definiert. Prosadisplays sind im Allgemeinen zeilenweise zentriert, das wird durch den Attributwert `rend=centered` codiert, wie im folgenden Beispiel (*c17 ℓ1023–1028*):

```
<display rend=centered><dline>Lighted Candle in Stick
</dline>
<dline>borne by
</dline>
<dline><hi rend=csc>Bloom
</hi>
</dline>
<dline>Diaconal Hat on Ashplant
</dline>
<dline>borne by
</dline>
<dline><hi rend=csc>Stephen
</hi>
</dline>
```

```
</display>
```

8.7. *Dramatis Personae*

Die auf S. 25 wiedergegebene Liste von Personen eines fiktiven Theaterstücks eignet sich gut für funktionale Codierung, auch wenn sie hier weniger Sinn als normalerweise macht, da es ja hier kein Drama gibt, auf daß sich die Personenliste bezieht und mit dem sie verknüpft werden könnte.

Die Codevorrat für dramatische Texte (*Green Book*, Abschnitt 10.1.4, S. 276–279) gestattet folgende Wiedergabe einer Rollenbeschreibung:

```
<castlist>
<castitem><role rend=csc>Toby Tostoff
</role>
<roleDesc>(a ruined Pole)
</roleDesc>
</castitem>
 ...
</castlist>
```

Es ließe sich diskutieren, ob die Klammern um die Rollenbeschreibung nicht aus dem Text in ein `rend`-Attribut wandern sollten, da sie ja zur funktionellen Charakterisierung dienen; für die Klammern in den Regieanweisungen des fünfzehnten Kapitels habe ich eine Speziallösung durch Entity References geschaffen, für den hier vorliegenden Fall schien mir das, mangels Masse, unnötig.

8.8. *Postadressen und Briefe*

An zwei Stellen im Buch werden Postadressen zeilenweise formatiert dargestellt, so wie die sie gelesen bzw. geschrieben werden. Die TEI sieht für Adressen ein einfaches Element `<address>` vor, das aus einer Folge von als `<addrLine>` codierten Zeilen besteht. Diese Struktur wurde zur Codierung der Adressen verwendet; da die Zeilen der Adressen aber sehr unterschiedlich eingerückt sind, wurden die beiden Adressen ebenfalls als Spezialfälle des Layouts im Anhang verzeichnet.

Eine echte Prüfung für funktionale, logische Codierung stellen die zitierten Briefe dar; eine aufwendige, aber saubere Lösung wäre es, sie als in den Erzähltext eingebettete Subtexte darzustellen. Das andere Extrem ist, ihre Rolle als Text-im-Text zu ignorieren, und sie als Zitat oder nur als Layoutphänomen zu erfassen.

Für zwei der drei Briefe habe ich mich für die erste Variante entschieden – sie sind jeweils als eigener `<text>` codiert. Die TEI-Elemente `<salute>` und `<signed>`, die für Grußformel und Unterschrift vorgesehen sind, habe ich durch ein `<ps>` für die in beiden Briefen vorkommenden Postscripta ergänzt. Damit ist die Struktur der Elemente in der Lage, das Layout befriedigend zu beschreiben, wie in folgender Kurzfassung des Briefs von Milly Bloom an ihren Vater (*c4 ℓ397–414*, vergleiche das Druckbild auf S. 26).

```
<text><body><salute rend=halfindent>Dearest Papli</salute>
<p rend=indent>Thanks ever so much for the lovely
birthday present.
 ...
I must now close with fondest love</p>
<salute rend=indent25>Your fond daughter</salute>
<signed rend=indent35><name>Milly</name></signed>
<ps rend=noindent>P. S. Excuse bad writing am in hurry.
    Byby.</ps>
```

```
<signed rend=indent35>M.</signed>
</body>
</text>
```

Der dritte Brief, der im zwölften Kapitel (*c12 ℓ415–431*) stückweise von einer Figur vorgelesen wird, entzieht sich dieser Codierungsweise. So sehr es dem Sinn der SGML-Auszeichnung zuwiderlaufen mag, ich habe ihn einfach, so gut es ging, als Prosadisplay bzw. als kursiven Text ausgezeichnet und die idiosynkratischen Zeilenanordnungen in Adresse, Anrede und Schluß als Sonderfälle markiert.

8.9. *Bücherverzeichnis*

Das Bücherverzeichnis im siebzehnten Kapitel des *Ulysses* ist, auch durch seine Absatzform, strukturiert genug um eine explizite Codierung dieser Struktur zu rechtfertigen, zumal die TEI verschiedene Elemente für die Codierung bibliographischer Verweise oder ausführlicher Bibliographien anbietet (siehe *Green Book*, Abschnitt 6.10). Allerdings lassen sich diese Elemente nur in einer eher informellen Weise auf die Beschreibung von Blooms Büchersammlung anwenden.

Das Element `<bibl>` kann für solche frei formatierten bibliographischen Angaben verwendet werden – im Gegensatz zu `<biblStruct>`, das eine formal strukturierte Angabe von Autor, Titel, Verlag *et cetera* verlangt und das ich für die bibliographischen Nachweise im Anhang verwendet habe.

Innerhalb der Buchbeschreibungen läßt sich noch der Titel problemlos auszeichnen, er ist durch Kursivschrift gekennzeichnet und wurde als `<title rend=it>` codiert. Die Liste wurde nur als Folge von Absätzen codiert; gegen die im *Green Book* auf S. 195 vorgeschlagene Möglichkeit, eine Folge von `<bibl>`-Elementen in einer `<listBibl>` zu einer Liste zusammenzufassen, würde auch nichts sprechen, es schien mir aber nicht notwendig, da die Liste doch sehr in die umgebende Prosa integriert ist, und die verwendete Absatzform (erste Zeile auf volle Breite gesetzt, Folgezeilen ein Geviert eingezogen) am Anfang des elften Kapitels auch als Prosaform vorkommt. Allerdings ließe sich auch diese Stelle als Liste auffassen, siehe dazu den Abschnitt 8.13 unten.

In der momentanen Form besteht die Bücherliste also aus einer Folge von Absätzen mit `rend=revindent`, die jeweils ein `<bibl>`-element enthalten; darin wiederum steht die freie Prosa des Absatzes, in der nur der Titel jeweils durch `<titel rend=it>` ausgezeichnet ist:

```
<p rend=revindent><bibl>Lockhart&apos;s
<title rend=it>Life of Napoleon</title>
 (cover wanting, marginal annotations, minimising victories,
aggrandising defeats of the protagonist).
</bibl>
</p>
```

8.10. *Prosazeilen mit synchronisierten Wortabständen*

Für dieses Einzelphänomen in *c11 ℓ1185–1186* habe ich keine adäquate und kompakte Codierung gefunden. Die beiden Zeilen als Tabelle zu codieren erscheint mir falsch. Die TEI Elemente der Gruppe *Linking, Segmentation and Alignment* (Kapitel 14 des *Green Book*) sind zwar prinzipiell dazu geeignet, Verbindungen zwischen einander auf irgendeine Weise entsprechenden Elementen zu dokumentieren (zum Beispiel um einen Text mit seiner Übersetzung zu synchronisieren), aber die singuläre Anwendung auf einen solchen Spezialfall erscheint mir merkwürdig. Ich habe mich darauf beschränkt, die Absätze als Sonderfälle zu kennzeichnen und im Anhang zu dokumentieren.

8.11. *Bilanz*

Die Abrechnung der finanziellen Transaktionen Leopold Blooms am 16. Juni 1904 läßt sich in der TEI DTD am besten als vierspaltige Tabelle repräsentieren. Das wird der Semantik natürlich nicht voll gerecht, aber doch hinreichend für die mir vorstellbaren Anwendungen.

Die Tabelle hat folgende Struktur:

```
<table id=spec06 cols=4>
<row role=label>
<cell role=label cols=2 rend=centered>Debit</cell>
<cell role=label cols=2 rend=centered>Credit</cell>
</row>
...
<row role=data>
<cell role=item>1 Pork kidney</cell>
<cell role=amount>0 - 0 - 3</cell>
<cell role=item>Cash in Hand</cell>
<cell role=amount>0 - 4 - 9</cell>
</row>
...
```

Die Tabelle ist also in durchgehende Zeilen gegliedert, die jeweils aus Zellen bestehen. Sowohl Zellen wie ganze Zeilen können eine rein beschreibende Rolle haben oder Datenwerte enthalten. Einzelne Zellen können mehrere Spalten abdecken, in unserem Fall die Kolumnentitel "*Debit*" und "*Credit*".

Um die Semantik der Tabelle etwas besser zu repräsentieren, habe ich die Schlüsselwörter für das `role`-Attribut etwas über die TEI-Vorschläge `label` und `data` hinaus erweitert. Ich verwende `label` für die Bezeichnungen der Tabellenspalten, `item` für die Textfelder, `amount` für die Geldbeträge, leere Zellen haben keine `role` gesetzt. Der Summenstrich ist ebenfalls durch eine Zeile repräsentiert, mit `<cell role=rule>` an den entsprechenden Positionen. Aus dem Blickwinkel der logischen Auszeichnung ist das natürlich angreifbar, aber die Linien werden immerhin auch im Referenzsystem der Druckausgabe als eigene Zeile gezählt. Der Saldo (*Balance*) ist durch `role=balanceitem` bzw. `role=balanceamount` gekennzeichnet. Die Summen schließlich haben `role=sum` gesetzt.

Alle Details der Semantik werden durch diese einfache Struktur natürlich nicht befriedigend erfaßt; ebensowenig werden wohl die letzten Feinheiten des Layouts ohne zusätzliche Nachbesserung aus den SGML-Quellen zu generieren sein. Deshalb ist auch diese Tabelle in der Liste der Sonderfälle enthalten, die `id=spec06` ist die Verbindung dorthin.

8.12. *Untergliederung in Aeolus, Wandering Rocks, Ithaca*

Im siebten Kapitel (*Aeolus*) ist der Text in kurze Abschnitte gegliedert, vor die jeweils eine schlagzeilenartige Überschrift gesetzt ist; das zehnte Kapitel (*Wandering Rocks*) besteht aus unbetitelten Teilabschnitten und das siebzehnte Kapitel (*Ithaca*) besteht aus einer Folge von Frage-Antwort-Einheiten. Diese Untergliederungen wurden alle durch `<div>`-Elemente realisiert, wobei das `type`-Attribut jeweils ein charakteristisches Schlüsselwort enthält (`aeolus`, `wanderingrocks`, `ithaca`).

Die Schlagzeilen im siebten Kapitel sind als `<head type=aeolus rend='centered caps'>` codiert. Die drei Sternchen, die die Grenze zwischen zwei Abschnitten im zehnten Kapiteln markieren, sind am Ende des jeweiligen Abschnitts als Entity `tristar` in einem `<trailer rend=centered>`-Element codiert.[20] Die Fragen im siebzehnten Kapitel schließlich werden im SGML-Dokument jeweils ein `<head type=ithaca rend=noindent>`.

8.13. *Sirens: Ausgerückte Absätze als Präludium*

Dieser Abschnitt ist einer der Fälle, wo sich die elektronische Ausgabe des *Ulysses* mit nicht allzuviel Aufwand interessanter gestalten ließe. Die Kurzabschnitte des Sirenenpräludiums sind im Moment nur als `<p rend=revindent>` gekennzeichnet, ohne das deutlich wäre, warum das so ist. Sowohl die Typographie wie auch die Lektüre des Kapitels macht aber deutlich, daß es sich im Grunde um eine Liste handelt (Tschichold schlägt in den *Erfreulichen Drucksachen* genau dieses Absatzformat für Auflistungen vor, und in der Bücherliste im siebzehnten Kapitel wird es ebenfalls verwendet). Jeder Kurzabsatz ist ein Verweis auf eine Textstelle im anschließenden Kapiteltext, in der Form eines zusammengezogenen, leicht umgeformten Zitats (darum nenne ich diesen Teil auch 'Präludium').

Wenn man versucht, ausgehend von der bereits geleisteten Layout-orientierten Darstellung des *Ulysses* in SGML eine inhaltlich orientierte Tiefenstruktur zu codieren, wäre es sicher eine interessante und verhältnismäßig kleine Aufgabe, das Präludium als eine Liste von Verweisen auf die entsprechenden Textabschnitte explizit zu codieren. Ich habe mir bei ohnehin schon knapper Zeit nicht die Mühe gemacht, dieses eine Puzzleteil zu legen.

8.14. *Cyclops: Untergliederung durch Prosastil*

Genau dasselbe Vermächtnis wie im vorhergehenden Abschnitt hinterlasse ich auch in diesem: es war nicht die Aufgabe meiner Konvertierungsarbeit, Sinnstrukturen zu codieren, die sich nicht in typographischer Form manifestieren; und nur wenige solcher Sinnstrukturen sind so eindeutig zu definieren, daß sie sich leichten Herzens in SGML formulieren lassen. Eine solche Sinnstruktur wäre die Zuordnung der Textteile im zwölften Kapitel zu ihren deutlich unterscheidbaren Erzählern gewesen. Amerikanische Bücher sagen in solchen Fällen gern mit einem Augenzwinkern, *this is left as an exercise for the reader*.

8.15. *Circe: Drama*

Die Elemente des dramatischen Textes lassen sich mit TEI Tags gut darstellen: Figurenrede wird in `<sp>`-Elementen angeführt, die mit einem `<speaker>` beginnen und anschliessend in beliebiger Folge `<p>`s und `<lg>`s (also Prosa und Lyrik) enthalten können (mit den vorgenommenen Erweiterungen auch `<display>` für Prosadisplays); `<stage>`-Elemente für Bühnenanweisungen können sowohl innerhalb von Figurenreden erscheinen als auch zwischen Figurenreden. Damit kann das fünfzehnte Kapitel ohne Probleme dargestellt werden.

Die Sprecher werden als `<speaker rend='centered csc'>` codiert, Prosarede als `<p rend=noindent>`, Lyrik ist im fünfzehnten Kapitel nicht kursiv und besteht daher aus zu `<lg>`s zusammengefaßten `<l rend=rm>` (die Verwendung von recte wurde explizit gemacht, um die Abweichung von der Norm des restlichen Buches zu verdeutlichen). Die von der TEI vorgeschlagene Möglichkeit, Sprecher durch IDs eindeutig zu identifizieren, macht für dieses 'Drama' generell wenig Sinn. Für die identifizierbaren und wiederkehrenden Figuren wie Bloom, Stephen Dedalus und die Mitarbeiterinnen des Bordells wäre eine solche Kennzeichnung für manche Zwecke vielleicht interessant, aber die (nicht vollständig automatisierbare) Arbeit ist nicht unerheblich.

---

[20] Das Element `<trailer>` der TEI DTD dient dazu, den 'Abspann' eines Abschnitts, der oft nicht zum eigentlichen Text gehört, wie z.B. "End of First Part of 'Ulysses'", zu kennzeichnen.

Ein Problem stellen die Klammern dar, in die die Bühnenanweisungen eingeschlossen sind. Sie sind lediglich typographische Signale für die Funktion des eingeschlossenen Textes (daß es sich nämlich um Bühnenanweisungen handelt, nicht um gesprochenen Text), sie gehören nicht zum Text der Bühnenanweisungen selbst. Falls jemand zum Beispiel die Syntax der Bühnenanweisungen statistisch untersuchen wollte, könnte er sie ohne weiteres automatisch aus dem SGML-Dokument extrahieren, aber die Klammern wären ein störender Ballast. Dies spräche also dafür, sie wie die *dialogue dashes* aus dem eigentlichen Text zu entfernen, da sie durch die SGML-Markierung impliziert werden. Andererseits sind sie als Zeichen des geschriebenen Textes so oft Objekt des Metatextes, daß ihre Abwesenheit zu vielen Schwierigkeiten geführt hätte. Darum habe ich den salomonischen Ausweg gewählt, sie durch die Entities `stagepo` und `stagepc` (*stage direction parenthesis open* bzw. *closed*) darzustellen, die am Anfang von `ulysses.sgml` als gewöhnliche Klammern definiert werden. Wenn die Klammern wirklich einmal stören, können diese Entities problemlos anders definiert werden.

Bühnenanweisungen werden durch ihr `type`-Attribut in `narrative` (szenische Anweisungen in einem selbständigen Absatz) und `directive` (eingebettet in Figurenrede) unterschieden. Szenische Bühnenanweisungen bestehen aus Absätzen `<p rend=fullindent>`.

Insgesamt wird damit die auf S. 30 zitierte Passage zu folgendem SGML-Text:

```
<sp><speaker rend='centered csc'>Cissy Caffrey</speaker>
<p rend=noindent>More luck to me.  Cavan, Cootehill and
Belturbet.
<stage type=directive rend=sl>&stagepo;she sings&stagepc;
</stage>
</p>
<lg type=stanza rend='centerblock'>
<l rend=rm>I gave it to Nelly</l>
<l rend=rm>To stick in her belly,</l>
<l rend=rm>The leg of the duck,</l>
<l rend=rm>The leg of the duck.</l>
</lg>
</sp>
<stage type=narrative rend=sl>
<p rend=fullindent>&stagepo;Private Carr and Private
Compton turn and counterretort, their tunics bloodbright
in a lampglow, black sockets of caps on their blond cropped
polls.  Stephen Dedalus and Lynch pass through the crowd
close to the redcoats.&stagepc;
</p>
</stage>
```

8.16. *Penelope: freier Prosafluß*

Das letzte Kapitel ist, was die Codierung des Lesetextes betrifft, ein leichter Fall. Die Gliederung in acht Unterabschnitte erfordert nicht unbedingt eine Aufteilung in `<div>`s, da die Absätze `<p rend=indent>` die Unterteilung bereits eindeutig markieren; ich habe deswegen keine `<div>`s eingeführt.

Viele Verarbeitungsprogramme werden mit den extrem langen Absätzen Schwierigkeiten haben (TEX zum Beispiel); das ist aber nicht unbedingt die Sorge des Codierers;

außerdem bieten vorhandene Codes, vor allem die *milestones* der Referenzsysteme, die Möglichkeit, geeignete 'Sollbruchstellen' zu finden.[21]

Ich schließe diesen Teil der Arbeit mit den letzten Worten des Lesetextes von `uly18.sgml`:

```
<p rend=indent>no thats no way for him has he no manners
nor no refinement nor no nothing in his nature slapping us
behind like that on my bottom
 ...
yes and his heart was going like mad and yes I said yes
I will Yes.
</p>
<trailer rend=noindent>Trieste&hyphen;Zurich&hyphen;Paris
</trailer>
<trailer rend=noindent>1914&hyphen;1921
</trailer>
```

[21] Auch schon in früheren Kapiteln sind einige Absätze ungewöhnlich lang; außerdem lauern einige überlange Wörter als Zeitbomben für den Textexport, zum Beispiel die Anrede des potentiellen TEXbrechers Kriegfried Ueberallgemein auf S. 46.

9. Der elektronische Metatext

Ich gebe zunächst einmal eine unkommentierte Aufstellung der verwendeten SGML-Elemente, um für die folgenden Ausführungen festen Boden unter den Füßen zu haben. Elemente aus der TEI DTD sind mit [TEI] gekennzeichnet.

Streichung *currente calamo*	<del type=cc>[TEI]...</del>
Einfügung im Manuskript	<overlay><add>[TEI]...</add></overlay>
Streichung im Manuskript	<overlay>~~[TEI]...~~</overlay>
Ersetzung im Manuskript	<overlay>~~[TEI]...~~ <add>[TEI]...</add></overlay>
Ersetzung in anderer Textstufe	<syn><old>...</old> <new>...</new></syn>
Strukturen in Revisionen	<boundary> (als Stellvertreter für Tags, wo eine andere Darstellung technisch nicht möglich war)
Fußnote (Emendation)	<emen>...</emen> (Enthält die Elemente <lem>[TEI], <wit>[TEI], <rdg>[TEI], <comm>)
Eintrag für Historische Kollation	<hico>...</hico> (Enthält die Elemente <lem>[TEI], <wit>[TEI], <rdg>[TEI], <comm>, <plref>, <depart>)
Textual Notes	<tn>...</tn> (Enthält entweder <head>[TEI], <div>[TEI]oder <plref>, <lem>[TEI], <p>[TEI])
Leerraum im MS	<space>[TEI]
Leere Einfügung im MS	<emptyadd>[TEI]
Unlesbarer und gestrichener Text	<gap>[TEI]
Änderung der Quellenlage	<docChange>
Informationen zur Worttrennung	<wbreak>
Normalisierung	<reg>[TEI]...</reg>
Nicht normalisierte Form	<orig>[TEI]...</orig>
Ergänzung der Grundschicht	<supplied>[TEI]...</supplied>

Die Elemente <lem>, <rdg> und <wit> wurden gegenüber den TEI-Modellen so erweitert, daß sie auch das Element <comm> enthalten können, auch die *content models* anderer TEI-Elemente mußten naturgemäß erweitert werden um die neu definierten Elemente in der Syntax verfügbar zu machen; die Einführung neuer Elemente ermöglicht außerdem die Verwendung von TEI-Tags in neuen Kontexten.

Wo in den folgenden Abschnitten ein neudefiniertes Element eingeführt wird, gebe ich ihre Definitionen aus der Datei `uly_ext.dtd` wieder. Die TEI DTD ist mit Hilfe von *parameter entities* stark modularisert aufgebaut, was sich auch auf den Mechanismus zur Erweiterung der DTD auswirkt. Zum Verständnis der in den Dateien `uly_ext.dtd` und `uly_ext.ent` verwendeten Technik und der dort getroffenen Entscheidungen ist die Kenntnis von Kapitel 3 "Structure of the TEI Document Type Definition" und Kapitel 29 "Modifying the TEI DTD" des *Green Book* sicher hilfreich.

9.1. *Die Codierung der nicht-textuellen Befunde*

Die Gabler-Edition bietet zwar an sich nur eine inhaltliche Transkription aller vom Autor stammenden Textteile. Es werden jedoch auch einige nicht-textuelle Befunde mitgeteilt, die für die Beurteilung der Textentwicklung eine Rolle spielen könnten. Die von der Gabler-Edition durch graphische Symbole eindeutig vorgegebenen Kategorien für diese Befunde sollten durch die Codierung nicht verwässert werden, deswegen wurde hier eine eindeutige, umkehrbare Zuordnung vorgenommen.

Das Drucksymbol "∅" für im Manuskript freigelassenen Leerraum wird durch das TEI-Element `<space>` codiert; die von der TEI vorgesehenen Attribute `dim` für die Richtung des Leerraums (horizontal oder vertikal) und `extent` für seine Größe konnte ich nicht verwenden, da die Gabler-Edition diese Informationen nicht bereitstellt.

Nichtlesbarer beziehungsweise gelöschter Text wird durch das TEI-Element `<gap>` codiert; dem Drucksymbol "▯" entspricht `<gap reason=illegible extent='letter(s)'>`, für "□" steht `<gap reason=illegible extent='word(s)'>`,[22] und für "⧄" steht `<gap reason=erasure extent='word(s)'>`.

Dem Symbol "△" der Druckausgabe, das als "insert indicated but missing" definiert wird, entspricht kein spezifischer TEI-Code. Die Codierung durch `<add><space></add>` wäre natürlich denkbar; der Eindeutigkeit zuliebe habe ich für diese besondere Form von Leerraum das Element `<emptyadd>` eingeführt, mit der Definition:

```
<!ELEMENT emptyadd      - O     EMPTY >
<!ATTLIST emptyadd      %a.global; >
```

Die Angabe von Zeilen- und Seitenumbrüchen, die wegen Unklarheiten bei der Worttrennung oder bei weggelassenen oder verdoppelten Wörtern wichtig sein könnten, erfolgt in der Druckausgabe durch die Symbole "| -| -|- ||". Im synoptischen Text habe ich zu ihrer Codierung das neudefinierte Element `<wbreak>` verwendet; das Attribut `where` kann die Werte `lb` oder `pb` (für *line break* und *page break*) annehmen. Mit dem Attribut `hyphen` kann angegeben werden, ob eine Worttrennung mit keinem, einem oder zwei Bindestrichen markiert ist. Das `level`-Attribut, mit dem die beschriebene Worttrennung einer bestimmten synoptischen Textschicht zugeordnet werden kann, sollte das Element flexibler machen. Diese Information ist aber im Grunde redundant und ist bei der Konvertierung auch nicht zuverlässig erzeugt worden; momentan wird das `level`-Attribut am besten ignoriert.

```
<!ELEMENT wbreak        - O     EMPTY >
<!ATTLIST wbreak        %a.global
          level         IDREF           #REQUIRED
          where         (lb | pb)       #REQUIRED
          hyphen        (0 | 1 | 2)     #REQUIRED >
```

Der senkrechte Strich wird außerhalb des synoptischen Textes, zum Beispiel in den *Textual Notes* und der Historischen Kollation als Symbol für den Zeilenumbruch des Textes verwendet, wenn beispielsweise mehrere Zeilen Lyrik kompakt zitiert werden sollen. Dort ist er behelfsmäßig als `<lb>` ohne weitere Attribute codiert.

Wenn sich innerhalb eines Kapitels die Quellenlage relevant ändert, wird das in der Druckausgabe durch das Symbol "⊓" und eine Fußnote verzeichnet. Die SGML-Ausgabe verwendet dafür das neudefinierte Element `<docChange>`.

```
<!ELEMENT docChange     - O     EMPTY >
<!ATTLIST docChange     %a.global; >
```

9.2. *Die Codierung der intratextuellen Bearbeitung*

Obwohl es sich um strukturell ähnliche Phänomene handelt, unterscheidet die Gabler-Edition zwischen den großen Entwicklungsstufen des Textes, die sich eigenen Dokumenten oder Bearbeitungsdurchgängen zuordnen lassen, und den Textrevisionen innerhalb einer Entwicklungsstufe. Die letzteren werden in diesem Abschnitt behandelt.

---

[22] In der Druckausgabe wird "□" definiert als "word or words illegible", in der Praxis scheint aber ein Symbol pro Wort gesetzt zu werden.

Keine echte Überarbeitung, sondern eine Korrektur unmittelbar während der Niederschrift stellt die Streichung *currente calamo* dar; sie ist durch das Element `<del type=cc>` codiert:

```
Pineapple rock, lemon platt, butter scotch. A
<del type=cc>lem</del>
sugarsticky girl shovelling scoopfuls of creams for a
christian brother.
```

(*p316 ℓ1–2*)

Der eigentliche *overlay* im Manuskript stellt nachträgliche Überarbeitungen dar und wird durch die TEI-Elemente `<add>` und `<del>` innerhalb des neudefinierten Elements `<overlay>` codiert. Die zusätzlichen Informationen zu einer Revision, die im Druck durch Spezialsymbole mitgegeben werden, werden im `<overlay>`-Element durch Attributwerte festgehalten.

```
<!ELEMENT overlay  - - (%n.add | (%n.del , (%n.add)?))>
<!ATTLIST overlay  %a.global;
          type     (normal | pseudo | ch10)  'normal'
          entrydoc (extant | lost)           'extant'
          status   (certain
                   | position-inferred
                   | content-supplied)       'certain'
          hand     IDREF                     'author' >
```

Das `status`-Attribut gibt an, ob die Einfügung mit Einfügeort und Inhalt gesichert ist oder ob der Ort oder sogar der Inhalt vom Herausgeber erschlossen wurde.

Das `type`-Attribut unterscheidet zwischen tatsächlichen Einfügungen (`normal`), Pseudoeinfügungen von Material, das früher schon einmal existierte (`pseudo`), und den besonderen Einfügungen im zehnten Kapitel, die im Druck mit den Level-Codes “⌜A. . .A⌝” beziehungsweise “⌜S. . .S⌝” bezeichnet wurden, in meinem SGML-Modell aber als mit `ch10` bezeichnete Sonderform des `<overlay>` geführt werden (vergleiche die Diskussion auf S. 35).

Nur für den Sonderfall im zehnten Kapitel sind auch die beiden folgenden Attribute vorgesehen: Das `entrydoc`-Attribut ermöglicht die Angabe, ob eine Einfügung tatsächlich im erhaltenen Dokument zu sehen ist (`extant`) oder in einem verlorenen Dokument stattgefunden haben muß (`lost`).

Das `hand`-Attribut schließlich gibt an, ob die Einfügung vom Autor selbst oder einem anderen Schreiber (`scribe`) stammt. Außer in den `<overlay ch10>` gilt immer `entrydoc=extant` und `hand=author`.

Es folgt ein Beispiel für verschachtelte Überarbeitung (das Druckbild dazu ist auf S. 32 wiedergegeben).

```
A shock of red hair he has sticking out of him behind
like a furzebush!
<overlay><add>Wait for nine months, my lad!
<overlay><del>It&apos;s
</del><add>
 Holy ginger, it&apos;s
</add>
</overlay>
kicking and coughing up and down in her guts already!
</add>
</overlay>
```

(*p1176 ℓ10-12*)

Während Entwurf und Durchführung der Konvertierung hielt ich den Level der Überarbeitung, der durch die Rotation der Einfügesymbole angezeigt wird, für redundante Information, die durch die Verschachtelung implizit gegeben ist und habe ihn infolgedessen nicht explizit durch ein Attribut festgehalten. Erst vor kurzem habe ich zufällig das erste Beispiel gefunden, in dem der Level nicht durch die lokale Schachtelung bestimmt ist (vergleiche S. 32), womit die vorliegende SGML-Edition leider revisionsbedürftig geworden ist.

Im Allgemeinen betreffen intratextuelle Revisionen, so wie sie in der Gabler-Ausgabe wiedergegeben werden, keine Strukturen sondern nur *phrase level* Text, das gilt aber nicht durchwegs. Einfügungen erstrecken sich in manchen Fällen über Absatzgrenzen hinweg, in anderen Fällen ist die Einfügung eines Strukturelements (zum Beispiel von ganzen Figurenreden im fünfzehnten Kapitel), offensichtlich aus technischen Gründen, künstlich in ihre *phrase level* Bestandteile zerlegt.

Eine sinnvolle Codierung muß also auch das Einfügen oder Streichen von Strukturelementen erlauben; für den `<overlay>` habe ich das Problem etwas anders gelöst als für das im nächsten Abschnitt beschriebene `<syn>`-Element. Anstatt Strukturelemente als Inhalt von `<add>` und `<del>` zuzulassen, habe ich mit Hilfe des ebenfalls unten beschriebenen `<boundary>`-Elements die betroffenen Strukturen verflacht. Diese Entscheidung ist natürlich anfechtbar. Ich habe sie getroffen, weil ich davon ausging, daß Strukturen im `<overlay>` eher eine seltene und lokal begrenzte Ausnahme darstellen, was sich aber nur bedingt als richtig herausgestellt hat. Zu der Problematik von strukturellen Revisionen wird im nächsten Abschnitt mehr gesagt, durch die unterschiedlichen Techniken für `<syn>` und `<overlay>` können in praktischen Experimenten jedenfalls zwei Lösungsansätze verglichen werden.

9.3. *Die Codierung der intertextuellen Bearbeitung*

Strukturell besteht kein großer Unterschied zwischen den Revisionen innerhalb einer Textschicht und den Änderungen, die das Schichtenmodell des synoptischen Textes ausmachen. Der eine wesentliche Unterschied ist aber, daß diese Schichten mit Siglen benannt sind; jeder (kapitelweise vergebenen) Sigle kann eine bestimmte Textfassung zugeordnet werden.

Mehr als für den *overlay* gilt, daß nicht nur nicht nur Textstücke hinzugefügt und gestrichen werden, sondern auch Strukturelemente: Absatztrennungen werden eingefügt oder aufgehoben, Prosa wird zu Lyrik umgewidmet, eine in eine Figurenrede eingefügte Bühnenanweisung wird solange erweitert, bis sie schließlich den Status einer selbständigen szenischen Anweisung erhält, und so weiter.

Die Codierung solcher struktureller Revisionen ist ein grundsätzliches Problem für SGML, da aus dieser Sichtweise die SGML-Tags des Lesetextes zusammen mit den Textdaten zu Objekten der textgeschichtlichen Struktur werden; was man braucht, ist eine Art Meta-SGML. Die zweite Forderung auf S. 42, daß jede Textschicht als korrektes SGML-Dokument extrahierbar sein solle, wäre am leichtesten umzusetzen, wenn die Textgeschichte überhaupt nicht mit SGML codiert wäre, sondern zum Beipiel als `RCS`-Dokument.

Natürlich ist aber eine reine SGML-Lösung notwendig, denn die dritte Forderung, graphische Darstellbarkeit der Synopsis, kehrt das Verhältnis zwischen umgebender Struktur und eingebetteter Information völlig um: unter dieser Voraussetzung sind die Lesetext-Elemente strukturbildend und die Tags des Metatexts könnten genausogut einfache *Milestones* oder *Character Entities* sein.

Generell kann man sagen, daß die `<overlay>`-Implementation eher der dritten Forderung, die `<syn>`-Implementation eher der zweiten Forderung zuarbeitet. Eine wirklich gute Lösung für das Dilemma scheint mir noch auszustehen.

⋆

Revisionen im Textschichtenmodell habe ich mit den neudefinierten Elementen `<syn>`, `<old>` und `<new>` codiert; `<syn>` erhält die Informationen zur varianten Stelle in Attributen und klammert die zusammengehörigen `<old>`- und `<new>`-Elemente zusammen.

```
<!ELEMENT syn           - - ( old , new )>
<!ATTLIST syn           %a.global;
          level         IDREF           #REQUIRED
          entrydoc      (extant | lost)      'extant'
          status        (certain
                        | position-inferred
                        | content-supplied)  'certain'
          hand          IDREF                'author'
          key           CDATA           #IMPLIED>
<!ELEMENT old           - O  ANY >
<!ATTLIST old           %a.global;
          levels        IDREFS          #REQUIRED>
<!ELEMENT new           - O  ANY >
<!ATTLIST new           %a.global;
          levels        IDREFS          #REQUIRED>
```

Die Attribute **entrydoc**, **status** und **hand** haben dieselbe Bedeutung wie für `<overlay>` (siehe oben); **hand** nimmt außer **author** und **scribe** noch den Wert **editor** an, wenn **status=content-supplied** gesetzt ist.

Jeder Level des Schichtenmodells hat im SGML-Text eine kapitelspezifische ID; der Level, auf dem die betreffende Änderung stattfindet, wird im Attribut `level` festgehalten; das Attribut **key** enthält die Bezeichnung des Levels in der Druckausgabe, um eine äquivalente Wiedergabe zu erleichtern. Da die Symbole der Druckausgabe am Anfang und Ende einer Markierung nicht identisch, sondern spiegelsymmetrisch angeordnet sind, werden Anfangs- und Endform in das `key`-Attribut aufgenommen.

Die Elemente `<old>` und `<new>` geben in ihrem `levels`-Attribut an, für welche Textstufe der in ihnen enthaltene Text gilt; dadurch kann jede der Textstufen einfach rekonstruiert werden, indem man die zugehörige ID in den `levels` sucht und das gesamte `<syn>`-Element durch den Inhalt des entsprechenden `<new>` oder `<old>` ersetzt. Der Lesetext der Edition hat eine kapitelübergreifende ID `lvl.gab` und kann ebenfalls auf diese Weise aus dem Gesamtdokument extrahiert werden.

Eine knappe, formalisierte Beschreibung zu jedem Level ist in einem `<synlevel>`-Element unter dieser ID im Anhang des SGML-Dokuments abgelegt; sie soll eine aufgeschlüsselte, weniger kompakte Darstellung des Textschichtenmodells ermöglichen und außerdem die Beantwortung von Fragen wie 'welche Änderungen im fünfzehnten Kapitel wurden nach der ersten Niederschrift des siebzehnten Kapitels vorgenommen?', also Verknüpfungen zwischen den Textschichten über Kapitelgrenzen hinweg, erlauben.

Es folgen die Definition und zwei Beispiele; etwas problematisch ist die Verwendung des `<date>`- beziehungsweise `<daterange>`-Elements ohne Inhalt – die TEI stellt diese Elemente eigentlich bereit, um vorhandene Zeitangaben zu codieren und in den Attributen zusätzlich zu erfassen. Vielleicht sollte man die Datumsinformation als Inhalt der Elemente wiederholen. Da ich ohnehin über zweihundert `<synlevel>`-Einträge mit der Hand erstellt habe, habe ich erstmal darauf verzichtet.

```
<!ELEMENT synLevel      - O ((%n.date; | %n.dateRange;),
                                synLevelDesc ) >
<!ATTLIST synLevel      %a.analysis; -- exploded a.global --
                        %a.linking;
                        %a.terminology;
           -- changed from a.global: id --
          id            ID              #REQUIRED
          n             CDATA           #IMPLIED
          lang          IDREF           %INHERITED
          rend          CDATA           #IMPLIED
```

```
            -- additional attributes: --
           key          CDATA          #REQUIRED
           entrydoc     (extant | lost | partial) #REQUIRED>
<!ELEMENT synLevelDesc  - O (%paraContent;) >
<!ATTLIST synLevelDesc  %a.global;>

<synlevel id=lvl1.c key="C" entrydoc=lost>
  <date value=1921></date>
  <synleveldesc>typescript overlay 1921</synleveldesc>
</synlevel>
<synlevel id=lvl1.1 key="1" entrydoc=extant>
  <daterange from=1921-06-11 to=1921-06-17></daterange>
  <synleveldesc>proof</synleveldesc>
</synlevel>
```

⋆

Es wird Zeit für ein Beispiel: Aus dem synoptischen Text “all about ⌜*B[that°] the B*⌝ old Barbary ⌜B[ape] apes ⟨△⟩B⌝” (*p1680 ℓ5*) wird, von der emendatorischen Fußnote abgesehen, folgender SGML-Text:

```
all about
<syn extant content-supplied level=lvl18.b hand='editor'
        key='*B B*'>
<old levels='lvl18.ros lvl18.a'>that</old>
<new levels='lvl18.b lvl18.1 lvl18.2 lvl18.3 lvl18.4 lvl18.5
        lvl18.6 lvl.gab'>
the
</new></syn>
old Barbary
<syn extant certain level=lvl18.b hand='author' key='B B'>
<old levels='lvl18.ros lvl18.a'>ape</old>
<new levels='lvl18.b lvl18.1 lvl18.2 lvl18.3 lvl18.4 lvl18.5
        lvl18.6 lvl.gab'>
 apes
<del type=cc><emptyadd></del>
</new></syn>
```

⋆

Um die Möglichkeit zu haben, neben reinem Text auch die Einfügung ganzer Absätze, Unterabschnitte, Gedichte und ähnlicher Strukturen mit dem `<syn>`-Element zu erfassen, war ein sehr offenes *content model* nötig; ich habe mich in einem radikalen Schritt für das *content model* `ANY` entschieden. Die Zusatzbedingung war, daß der Inhalt sowohl des `<old>`- wie des `<new>`-Elements an die Stelle des gesamten `<syn>`-Elements treten könnte ohne die syntaktische Korrektheit des SGML-Dokuments zu verändern. Ebenso sollte es möglich sein, die Inhalte von `<old>` und `<new>` direkt hintereinander zu hängen, ohne die syntaktische Korrektheit zu gefährden. Solche Anforderungen bleiben aber natürlich informelle Wünsche, sie können in der Syntax von SGML nicht erzwungen werden.

Experimentieren mit den Möglichkeiten und Einschränkungen dieser Darstellungsform wird vielleicht zu anderen Lösungen führen; ich fürchte aber, daß SGML immer Schwierigkeiten haben wird, strukturelle Varianten wirklich flexibel zu repräsentieren.

⋆

Eine weitere Schwierigkeit tritt dann auf, wenn die Varianz nicht ganze, abgeschlossene Elemente umfaßt, sondern nur die Grenzen oder die Art von Element. Es ist beispielsweise nicht möglich, die Einfügung einer neuen Absatztrennung so zu beschreiben:

```
<p rend=indent>
...like a stallfed heifer.
<syn><old></old>
<new></p><p rend=indent></new> <!-- Illegal!! -->
</syn>
He took a page up from the pile...
</p>
```

Für solche Fälle habe ich ein leeres Element `<boundary>` eingeführt, das als Stellvertreter der nicht zulässigen Tags eingesetzt werden kann. Soll die Revision als solche dargestellt werden, kann es in eine graphische Darstellung, zum Beispiel die Symbole "|B|" beziehungsweise "←B→" der Gabler-Edition, verwandelt werden. Soll die Revision hingegen *durchgeführt* werden, enthält das `<boundary>`-Element alle Informationen, um die eigentlich gemeinten Tags wieder herstellen zu können. Das obige Beispiel sähe folgendermaßen aus (von den Attributen für `<syn>`, `<old>` und `<new>` abgesehen):

```
<p rend=indent>
...like a stallfed heifer
<syn><old></old>
<new><boundary gi=p btype=split rend=indent></new>
</syn>
He took a page up from the pile...
</p>
```

Das Attribut `gi` enthält den *generic identifier* des ersetzten Tag, das Attribut `btype`, das die Werte `start`, `end` und `split` annehmen kann, definiert, ob ein *start tag*, ein *end tag*, oder, wie im Beispiel, ein *end* gefolgt von einem neuen *start tag* ersetzt wird. Weiterhin sind für `<boundary>` die typischen TEI-Attribute vorgesehen, die an den wiederhergestellten Tag weitergegeben werden können, so wie `rend=indent` im Beispiel.

```
<!ELEMENT boundary - O  EMPTY>
<!ATTLIST boundary      %a.global;
          gi            NAME       #REQUIRED
          btype         (start | end | split)  'split'
          type          CDATA      #IMPLIED>
```

Leider ist die praktische Umsetzung dieser Codierungstechnik noch nicht gut gelungen, so daß die `<boundary>`-Codierung, wo sie nicht mit der Hand verbessert wurde, oft mangelhaft geblieben ist.

Ein in diesem Zusammenhang noch ungelöstes Problem sind beispielsweise die 'Umwidmungen' von Text, wenn zum Beispiel Prosa zu Lyrik wird. Ich zitiere noch einmal die auf S. 34 angeführte Passage aus dem ersten Kapitel:

He walked along the upwardcurving path.

|4| *Liliata rutilantium.*
|4| *Turma circumdet.*
|4| ⌜4*Iubilantium*° *te virginum.*°4⌝

|4| The priest's grey nimbus in a ^⟨☒⟩niche^ where he dressed discreetly. [4*Jubilantium te virginum.*]

(*p42 ℓ31–p43 ℓ2*)

Vor der Stufe Revision auf Stufe vier sah diese Stelle also so aus:

> He walked along the upwardcurving path. *Liliata rutilantium. Turma circumdet.* The priest's grey nimbus in a niche where he dressed discreetly. *Jubilantium te virginum.*

In dieser Form wäre der Text als ein Absatz <p rend=indent> zu codieren, der zwei Phrasen <hi rend=it> enthält. In der revidierten Form endet der Absatz <p rend=indent> nach “path”, daran schließt sich eine Strophe Lyrik <lg type=stanza rend=centerblock> aus drei Zeilen <l rend=it> an, die am Ende natürlich auch mit den End-Tags </l> und </lg> abgeschlossen werden muß, dann beginnt ein neuer <p rend=indent> mit “The priest's”.

Diese relativ komplexe Umstrukturierung (und es gibt noch kompliziertere Fälle, zum Beispiel *p1046 ℓ1–7*) ist durch automatische Konvertierung nur mühsam zu bewerkstelligen; und auch wenn man bereit ist, mit der Hand zu codieren, muß man eventuell zusätzliche <syn>-Elemente einfügen, um die Transformation korrekt codieren zu können (im vorliegenden Fall ist das nicht nötig).

Dazu kommt, daß die gesamte in der vierten Stufe hinzugekommene Struktur der <lg> und die dadurch implizierten Absatztrennungen davor und danach nur durch <boundary>-Elemente codiert werden können; die Verwendbarkeit des Dokuments ist also stark davon abhängig, daß diese Elemente sinnvoll behandelt werden, was innerhalb von SGML nicht erzwungen werden kann.

Durch die strukturellen Revisionen gerät man alles in allem in ein Netz von Problemen, aus dem ich keinen einfachen Ausweg weiß. Ich habe an einigen solchen Stellen so sinnvoll wie möglich mit der Hand codiert, der Rest steht so dürftig da, wie er von der automatischen Konvertierung erzeugt wurde.

⋆

Schließlich will ich noch kurz die Behandlung der Grenzfälle des Schichtenmodells darstellen:

Das SGML-Modell des synoptischen Textes fordert eine konsequente Durchführung des Gablerschen Schichtenmodells – jedes Stück Text muß einer klar definierten Menge von Textschichten zuzuordnen sein, jede Textschicht wiederum muß einen durchgängigen Text ergeben und für jede Textschicht muß klar definiert sein, wer ihre Vorgänger und Nachfolger sind.

Die gewisse Flexibilität, mit der das Schichtenmodell der Druckausgabe in einzelnen Fällen gehandhabt wird, mußte also aufgestöbert und in das kompromißlosere SGML-Modell integriert werden. Ich habe bereits erklärt, wie und warum die ‘Textstufen’ “⌜A…A⌝” und “⌜S…S⌝” des zehnten Kapitels als <overlay> dargestellt werden. Ebenfalls aus der Reihe tanzen die Auszeichnungen “⌜R…R⌝” und “⌜V…V⌝” (vergleiche S. 34). Sie lassen sich aber in das Stufenmodell einbauen: Die mit ‘R’ bezeichneten Lesarten werden ausschließlich den Stufen lvl.ros und lvl.gab zugeordnet, bilden also als eine Textform, die nur in der Grundschicht und dem edierten Text der Gabler-Edition enthalten ist. Die mit ‘V’ bezeichneten Lesarten werden ausschließlich dem lvl.ros zugeordnet, da der edierte Text diesen Lesarten nicht folgt. Die Auszeichnung “⌜+…+⌝” im siebzehnten Kapitel macht hingegen keine Schwierigkeiten; ich habe ihr einen eigenen lvl.plus zugeordnet und das Einfügen kompletter <div>-Elemente ist dank des liberalen *content model* von <new> kein Problem. Die mit “⌜○…○⌝” bezeichneten Passagen werden ohne jeden Rückgriff auf das Schichtenmodell mit dem TEI-Tag <supplied reason='base text lost' source='typescript'> codiert. Die Alternative, diese Passagen mit <syn>-Elementen als nicht zur Grundschicht gehörend auszuzeichnen, also mit Hinzufügungen im Typoskriptlevel gleichzustellen, wäre falsch gewesen, da sie ja nicht wirklich später hinzugefügt worden sind.

9.4. *Die Codierung der Fußnoten*

Ich habe schon in der Dokumentanalyse darauf hingewiesen, daß sich die Fußnoten in einem Übergangsfeld zwischen streng formalisierter und frei formulierter Information bewegen. Eine Codierung, die die Fußnoten inhaltlich erhalten sollte und obendrein automatisch durchgeführt werden mußte, kann darum ihren Informationsgehalt nicht völlig formalisieren und dadurch explizit machen. Andererseits sollten vorhandene Strukturen durchaus sinnvoll codiert werden, was einmal mehr eine Neudefinition von Elementen notwendig gemacht hat, die ich zunächst angebe:

```
<!ELEMENT emen          - - ( (%n.lem, (%n.wit;)?,
                              (%n.rdg;, (%n.wit;)?)*)?)
                                                  +(comm) >
<!ATTLIST emen          %a.global
          type          (note | normal | fseop) #REQUIRED
          from          IDREF                   #IMPLIED
          to            IDREF                   #IMPLIED >
<!ELEMENT comm          - - (%paraContent;) >
<!ATTLIST comm          %a.global; >
```

Sobald man die Klammerebenen richtig zusammengebracht hat, erkennt man, daß eine Emendation im Extremfall leer sein, beziehungsweise nur aus Kommentarelementen `<comm>` bestehen kann. Ansonsten ist sie aber als geordnete Abfolge von einem Lemma `<lem>` und einigen Lesarten `<rdg>` angelegt, die jeweils optional von einer Angabe `<wit>` des zugehörigen Zeugen gefolgt werden. Die Unterelemente `<lem>`, `<rdg>` und `<wit>` sind TEI-Elemente; sie werden in der TEI DTD in `<app>`-Elementen verwendet, um textkritischen Apparat zu codieren. Die TEI sieht für `<app>` allerdings die Kommentare `<comm>` nicht vor, die überall in `<emen>` vorkommen können.

Ich habe mich lange mit der Frage beschäftigt, ob es nicht möglich und sinnvoll gewesen wäre, einige oder alle Formen von Textvarianten im *Ulysses*, also das Textschichtenmodell, die Emendationen und/oder die historische Kollation als TEI-`<app>`s zu codieren. Ich habe mich letztlich dagegen entschieden, weil es in jedem Einzelfall Abstriche bei der Genauigkeit der Übertragung ins elektronische Modell bedeutet hätte, und weil das gleichzeitige Vorhandensein von so vielen unterschiedlichen Typen von `<app>` die Verhältnisse eher undurchsichtiger gemacht hätte. Wenn man bereit ist, die TEI-Konformität mit teilweise erheblichem Informationsverlust zu bezahlen, wird man zumindest Emendationen und historische Kollation vollautomatisch in `<app>`s umwandeln können; das mächtige *content model* der `<syn>`-Elemente ist dazu wohl nur bedingt geeignet.

Die unterschiedlichen Verwendungsformen der emendatorischen Fußnoten sind nicht scharf voneinander abzugrenzen; einen Versuch in dieser Richtung macht das `type`-Attribut. Die klare Kategorie `fseop` (*full stop, end of paragraph*) bezeichnet Emendationen, die im Druck gar nicht erscheinen, in den Quelldateien aber vorhanden sind. Sie geben an, wo ein Punkt am Absatzende eingefügt wurde. Die Kategorie `type=note` wurde eingefügt, um die zur reinen Fußnote 'degenerierten' Emendationen auszeichnen zu können; die Forderung nach automatischer Erkennbarkeit und die fließenden Übergänge zwischen Emendation und kommentierender Fußnote haben die Anwendbarkeit dieser Kategorie aber sehr begrenzt. Praktisch wird sie nur für die Hinweise "*GENERAL NOTE* → TN" am Anfang jedes Kapitels verwendet.

Die Textpassage, auf die sich eine Emendation bezieht, ist durch `<anchor>`-Elemente abgegrenzt, auf deren IDs die Attribute `from` und `to` von `<emen>` verweisen. Diese Passage wird, eventuell verkürzt, im `<lem>`-Element wiederholt, so wie sie im Druck vor der eckigen Klammer erscheint. Das ist natürlich redundante Information, aber es erleichtert die Verarbeitung, und nebenbei auch die Kontrolle, ob der ID-Verweis korrekt erzeugt wurde. Zudem gibt es emendatorische Anmerkungen, die aus technischen Gründen ohne den ID-Verweis auskommen müssen; sie beziehen sich auf das letzte vorhergehende Wort, und die Wiederholung des Lemmas ist hier eventuell nützlich.

Die in den Kommentaren immer wieder vorkommenden Abkürzungen wurden mit dem TEI-Element `<abbr>` aufgelöst, so daß für eine weniger komprimierte Darstellung auch die ausgeschriebene Form des Kommentars ausgegeben werden kann. Die in Kommentaren vorkommenden Verweise auf andere Textstellen wurden zwar als `<ref>` markiert, aber nicht in automatisch verfolgbare Links aufgelöst. Verweise auf die *Textual Notes* durch das Symbol "→ TN" hingegen werden mit dem TEI-Element `<ref>` als explizites Link codiert.

Wenn in `<lem>` oder `<rdg>` der Emendationen, aber auch in der historischen Kollation und den *Textual Notes* Textpassagen wiedergegeben werden, in denen ein 'harter' Zeilenwechsel vorkommt, also beispielsweise mehrere Zeilen eines Gedichts, steht statt dem "|" der Druckausgabe der Code `<lb>`. Das ist keine wirklich befriedigende Lösung, da dieses Element eigentlich zur Codierung von Zeilenumbrüchen für Referenzsysteme gedacht ist (und so im Text auch verwendet wird). Die Codierung durch `<l>` beziehungsweise `<dline>` für Lyrik- oder Prosazeilen würde aber bedeuten, daß die gesamte Lesart als Lyrik oder Prosadisplay zu codieren ist, auch wenn sie nicht so dargestellt ist und eventuell nur zwei aus dem Zusammenhang gerissene Worte enthält. Die Codierung durch ein entsprechendes graphisches Sonderzeichen (ISO `verbar`) schließt wiederum die Möglichkeit aus, den Zeilenumbruch tatsächlich als solchen zu erkennnen und darzustellen.

Dies führt zu der generellen Problematik, wie innerhalb von `<lem>` und `<rdg>` aus Überschriften, Bühnenanweisungen, Gedichten und dergleichen zitiert werden kann; für die *Textual Notes* erweitert sich das Problem noch auf das Zitieren von Strukturen des synoptischen Stufentextes. Würde man die Strukturelemente wie `<head>`, `<stage>` und so weiter in diese Apparate übernehmen, wären sie außerhalb ihres Kontextes gestellt. Sie machen dort als Strukturelemente wenig Sinn und sind ohne eine beträchtliche Erweiterung des *content models* von `<lem>` und `<rdg>` auch nicht syntaktisch zulässig. Aus diesen Gründen erfolgt in den Apparaten typographische Auszeichnung nur mit dem `<hi>`-Element; die Symbole der Manuskripttranskription in den *Textual Notes* werden aus demselben Grund als Sonderzeichen codiert, nicht als `<overlay>`.

⋆

Das folgende, etwas längere Beispiel gibt den auf S. 36 zitierten Anfang des Buches wieder und enthält zwei der dort zitierten emendatorischen Fußnoten.

```
<p rend=indent><anchor id=id01c1n1>Stately,<anchor id=id01c1n2>
<emen type=normal from=id01c1n1 to=id01c1n2>
<lem>Stately,</lem>
<wit>(tB)</wit>
<rdg><comm rend=slcsc>not indented</comm></rdg>
<wit>aR</wit>
</emen>
plump Buck Mulligan came from the stairhead, bearing a bowl
of lather on which a mirror and a razor lay crossed. A yellow
dressinggown, ungirdled, was sustained gently behind him
<syn lost certain level=lvl1.b hand='author' key='(B) (B)'>
<old levels='lvl1.ros lvl1.a'>by</old>
<new levels='lvl1.b lvl1.c lvl1.1 lvl1.2 lvl1.3 lvl1.4 lvl1.5
        lvl1.6 lvl.gab'>
<anchor id=id01c1n3>on<anchor id=id01c1n4></new>
</syn>
<emen type=normal from=id01c1n3 to=id01c1n4>
<lem>on</lem>
<wit>LR,Eg</wit>
<rdg>by</rdg>
<wit>(aC) <comm rend=slcsc>
<abbr expan="printer's copy unchanged">pcu</abbr>
</comm>
</wit>
<rdg><comm rend=slcsc>cf</comm>
```

```
<comm rend=slcsc><ref>892.15-17</ref></comm>
</rdg>
</emen>
the mild morning air.
```

9.5. *Die Codierung der historischen Kollation*

Die Einträge der historischen Kollation sind den Emendationen ähnlich und werden auch ähnlich codiert; der Unterschied besteht in der größeren formalen Strenge der Einträge und in der Anwesenheit der zusätzlichen Elemente <plref> und <depart> (der Aufbau eines Eintrags ist auf S. 37 beschrieben).

```
<!ELEMENT hico          - - (%n.lem, (%n.rdg, %n.wit,
                                      plref?, depart?)+)
                                                +(comm) >
<!ATTLIST hico          %a.global;
          from          IDREF           #REQUIRED
          to            IDREF           #REQUIRED >
<!ELEMENT plref         - - (%paraContent;) >
<!ATTLIST plref         %a.global;
          ed            IDREF   #REQUIRED
          fromp         NUMBER  #REQUIRED
          froml         NUMBER  #REQUIRED
          top           NUMBER  #REQUIRED
          tol           NUMBER  #REQUIRED >
<!ELEMENT depart        - - (%paraContent;) >
<!ATTLIST depart        %a.global; >
```

Die Zeilen- und Wortreferenz, die am Anfang eines gedruckten Eintrags steht, wird ebenso wie die Zeilenreferenz einer Emendation als implizit gegeben gesehen und nicht codiert; stattdessen verweisen die Attribute `from` und `to` auf die IDs der <anchor>-Elemente, die das Lemma im Text umgeben. Für das <lem>-Element ist hier keine Angabe von Zeugen nötig, es ist immer die Lesart des Gablerschen Lesetextes. Die <rdg>-Elemente werden von einem <wit>-Element gefolgt, in dem die zugehörigen Quellen in der Schreibweise der Druckausgabe verzeichnet sind; zusätzlich wurde diese Information auch noch automatisch abgreifbar codiert in Form von IDs, die im `wit`-Attribut der <rdg>s gegeben werden. Die Schlüsselwörter `e22`, `e26`, `e32`, `e36`, `e37` und `e61` bezeichnen die Ausgaben mit dem jeweiligen Erscheinungsjahr, sie sind gleichzeitig die IDs der bibliographischen Einträge zu diesen Ausgaben im Anhang. Diese explizite Auszeichnung macht es möglich, beispielsweise eine Liste der Abweichungen nur zwischen der 1961er Ausgabe von *Random House* und dem Gablerschen Lesetext zu erzeugen.

Das Element <plref> habe ich als Spezialisierung des TEI-Elements <ref> neu definiert, um Verweise auf einen Text mittels Seiten- und Zeilenangabe gut codieren zu können. In der historischen Kollation dient es zur Codierung der Referenz der Ausgabe von *Random House*. Die Attribute `fromp` und `froml` enthalten Seiten- und Zeilennummer des Beginns einer Textstelle, `top` und `tol` Seiten- und Zeilennummer des Endes. Der Inhalt von <plref> ist, in Analogie zu <ref>, die Klartextangabe der betreffenden Referenz.

Das Element <depart>, das die Angaben zum 'Ort' der Abweichung enthält, besteht ähnlich wie ein <wit>-Element aus einer Zeugenangabe als unstrukturierter Zeichenfolge. Auch wenn im Druck mehrere Angaben durch ein Pluszeichen verbunden werden, wird die gesamte Zeichenfolge als ein einziges <depart>-Element dargestellt.

Ich gebe die SGML-Form des ersten Eintrags der historischen Kollation (zum zweiten Satz des Textes) als Beispiel:

```
A yellow dressinggown, ungirdled, was sustained
gently behind him <anchor id=id01c1n3>on<anchor id=id01c1n5>
<hico from=id01c1n3 to=id01c1n5>
<lem>on</lem>
<rdg wit='e22 e26 e32 e36 e37 e61'>by</rdg>
<wit>22-61</wit>
<PLref ed=e61 fromp=3 froml=3 top=3 tol=3>&lt;3.3&gt;</PLref>
<depart>(aC) <comm rend=slcsc>
<abbr expan="printer's copy unchanged">pcu</abbr>
</comm>
</depart>
</hico>
the mild morning air.
```

9.6. *Die Codierung der 'Textual Notes'*

Die *Textual Notes* sind ausführliche Kommentierungen einzelner Textstellen; zusätzlich gibt es allgemeine Hinweise zu jedem Kapitel. Das *content model* für `<tn>` ist deswegen auch effektiv zweigeteilt:

```
<!ELEMENT tn    - - ( ((%n.head;)+, (%n.div;)+)
                    | (plref+, (%n.lem)+, (%n.p)+) ) >
<!ATTLIST tn    %a.global
          type  (general | specific) #REQUIRED >
```

Wenn `type=general` gesetzt ist, handelt es sich um eine allgemeine Anmerkung, die aus einer oder mehreren Überschriften `<head>` und einem oder mehreren Unterabschnitten `<div>` besteht. Wenn `type=specific` gesetzt ist, handelt es sich um eine Anmerkung zu einer oder mehreren Textstellen: für jede dieser Stellen gibt es eine `<plref>` mit Seiten- und Zeilenverweis auf den synoptischen Text (`<plref>` wurde im vorigen Abschnitt vorgestellt) sowie ein Lemma `<lem>`. Entsprechend dem Druckbild erscheinen erst alle `<plref>`s, dann alle `<lem>`s, wobei auch mehrere `<plref>`s für ein `<lem>` erscheinen können, wenn die Lemmata gleich sind. Anschließend folgen ein oder mehrere Absätze Text zu diesen Stellen. Das Problem von in diesem Text vorkommenden Zitaten aus dem synoptischen Text wurde bereits im Abschnitt über die Codierung der Emendationen auf S. 69 besprochen.

Vom Text aus zugänglich sind die *Textual Notes* über die Fußnoten. Der Hinweis auf den Anhang ist dort als `<ref type=tn>` mit der ID der entsprechenden Anmerkung codiert. Zum Beispiel steht im Text des ersten Kapitels auf Seite 52 folgende Passage:

```
Isn&apos;t the sea what Algy calls it: a
<anchor id=id01c1n52>great<anchor id=id01c1n53>
<emen type=normal from=id01c1n52 to=id01c1n53>
<lem>great</lem>
<comm rend=rm><ref type=tn target=id03tn1>
<abbr expan='see Textual Notes'>&rarr;TN</abbr>
</ref>
</comm>
</emen>
sweet mother?
```

Die `<ref type=tn target=id03tn1>` ist einer der beiden Verweise auf folgende `<tn>` im Anhang des SGML-Dokuments:

```
<tn type=specific id=id03tn1>
<plref ed=gab fromp=6 froml=14 top=6 tol=14>6.14</plref>
```

```
<plref ed=gab fromp=6 froml=23 top=6 tol=23>6.23</plref>
<lem>great</lem>
<lem>grey</lem>
<p rend=join>The reading at <plref ed=gab fromp=6 froml=23
top=6 tol=23>6.23</plref> ‘great searching eyes’ (for R, LR
and Eg: ‘quick searching eyes’) which first appears in the
<hi rend=it>placards</hi> (Pl) seems to be an error in the
transcription of a revision. The post&hyphen;1922 autograph
corrections specify ‘grey searching eyes’.  The revision
introduces a traditional epic colour epithet. The mistyping
of Joyce’s corrections provided a faulty line reference. The
correction was applied to the nearest ‘great’, resulting in
an unwarranted change of the quotation from Swinburne.</p>
</tn>
```

Zu erkennen ist an dem Beispiel, daß auch innerhalb des Textes der Anmerkungen Textverweise mit Seiten- und Zeilenangabe als `<plref>` codiert werden. Dadurch sollen Browser die Möglichkeit haben, diese Hinweise als Hyperlinks zu nutzen. Textzitate hingegen sind nicht besonders gekennzeichnet, da der Gebrauch von Anführungszeichen keine eindeutige automatische Identifikation zuläßt.

9.7. *Ein unbeschönigtes Beispiel*

Zum Abschluß der Behandlung des elektronischen Textes möchte ich, nachdem alle bisher angeführten Beispiele aus didaktischen Gründen vereinfacht wurden, noch ein Rohbeispiel geben, so wie es tatsächlich in den Quelldateien der SGML-Edition steht. Es handelt sich um den ersten Absatz des achten Kapitels, den ich zur Orientierung zunächst dem synoptischen Text der Druckausgabe entsprechend wiedergebe (wie immer mit anderem Zeilenfall):

> Pineapple rock, lemon platt, butter scotch. A ⟨lem⟩ sugarsticky girl shovelling scoopfuls of creams for a christian brother. Some school treat. ⌜(B)Bad for their tummies.(B)⌝ Lozenge and comfit manufacturer to His Majesty the King. God. Save. Our. Sitting on his throne° sucking ⌜D[jujubes.] red jujubes white.D⌝
>
> ————
>
> GENERAL NOTE → TN   4 throne] aR; throne, tC
>
> (*p316 ℓ1–5*)

Die Datei `uly08.sgml` beginnt mit einer von der automatischen Konvertierung erzeugten Kopfzeile mit Datum und Uhrzeit der Erstellung, gefolgt von der *revision history* der anschließenden Bearbeitung, die vom *Revision Control System* `RCS` dort fortgeschrieben wird. Dieser Vorspann ist in SGML-Kommentare eingebettet und hat also keinerlei Einfluß auf den Parser.[23] Daran schließt sich der SGML-Text der momentanen Version an (überlange Zeilen werden im Ausdruck umbrochen und die Fortsetzung in der Folgezeile etwas eingezogen). Die Ausgaberoutine der automatischen Konvertierung versucht, sich die Regeln für Leerraum in SGML-Dateien zunutze zu machen um ein Mindestmaß an Übersichtlichkeit zu erzeugen (verrgleiche S. 100). Allerdings werden Tags nie auf mehrere Zeilen verteilt, um Suchen und Ersetzen in Texteditoren zu erleichtern, was sich bei der manuellen Bearbeitung auch sehr bewährt hat. Die nachträglich generierte Version 0.1.2.1, auf die sich der Logeintrag zu Version 1.2 bezieht, taucht in der *revision history* nicht explizit auf.

```
<!- Automatic Conversion Output -*-fundamental-*- ->
<!- Generated on Nov 22, 1996 at 13:35h ->
```

[23] Es sei denn, man verwendet – wie gelegentlich geschehen – aus Versehen die Zeichenfolge ‘`--`’ in einem Logeintrag, die dann den Kommentar vorzeitig beendet.

```
<!-

$Log: uly08.sgml,v $
# Revision 1.2  1997/01/22  09:47:27  tobsi
# Merged in v0.1.2; corrected errors with <boundary> and <ref>.
#
# Revision 1.1  1997/01/09  20:03:38  tobsi
# Added cut-line for demo version. Set to Emacs fundamental mode.
#
# Revision 1.0  1996/11/28  11:58:20  tobsi
# Ulysses makes it through the parser
#
# Revision 0.3  1996/11/22  12:41:46  tobsi
# Versionen 0.2 und 0.1.1.1 zusammengebracht.
#
# Revision 0.1.1.1  1996/11/22  12:40:19  tobsi
# Neu generiert nach Konvertierungsfehlern.
#
# Revision 0.2  1996/11/20  12:34:34  tobsi
# Der TEI Header und Episode 1-9 parsen (im ersten pass) fehlerfrei;
# einzelne Layoutkorrekturen in 1-9.
#
# Revision 0.1  1996/11/14  15:23:00  tobsi
# Initial Revision

->
<pb ed=syn n=316><lb ed=syn n=1><p rend=indent><lb ed=gab n=1>Pineapple
<emen type=note><comm rend=slcsc>general note
</comm>
<comm rend=rm><ref type=tn target=id03gen8><abbr expan='see Textual Notes'>&rarr;TN
</abbr>
</ref>
</comm>
</emen>
rock, lemon platt, butter scotch. A
<del type=cc>lem
</del>
sugarsticky girl
<lb ed=gab n=2><lb ed=syn n=2>shovelling scoopfuls of creams for a
christian brother. Some school treat.
<lb ed=gab n=3><lb ed=syn n=3><syn lost certain level=lvl8.b hand='author' key='(B) (B)'><old
      levels='lvl8.ros'>
</old><new levels='lvl8.b lvl8.c lvl8.d lvl8.1 lvl8.2 lvl8.3 lvl8.4 lvl8.5 lvl8.6 lvl8.7 lvl8.8
      lvl.gab'>Bad
for their tummies.
</new>
</syn>
Lozenge and comfit manufacturer to His
<lb ed=syn n=4>Majesty
<lb ed=gab n=4>the King. God. Save. Our. Sitting on his
<anchor id=id01c8n1>throne<anchor id=id01c8n2><hico from=id01c8n1 to=id01c8n2>
 <lem>throne
</lem>
<rdg wit='e22 e26 e32 e36 e37 e61'>throne,
</rdg>
<wit>22-61
</wit>
<PLref ed=e61 fromp=151 froml=5 top=151 tol=5>&lt;151.5&gt;
</PLref>
<depart>tC
</depart>
</hico>
<emen type=normal from=id01c8n1 to=id01c8n2><lem>throne
</lem>
```

<wit>aR
</wit>
<rdg>throne,
</rdg>
<wit>tC
</wit>
</emen>
sucking
<lb ed=syn n=5><syn extant certain level=lvl8.d hand='author' key='D D'><old levels='lvl8.ros lvl8.b
lvl8.c'>jujubes.
</old><new levels='lvl8.d lvl8.1 lvl8.2 lvl8.3 lvl8.4 lvl8.5 lvl8.6 lvl8.7 lvl8.8 lvl.gab'>
red jujubes white.
</new>
</syn>
</p>

Das Konvertierungsverfahren

10. Die Quelldateien

Das Ausgangsmaterial für die Herstellung meiner elektronischen *Ulysses*-Ausgabe waren Dateien aus der Herstellung der Druckausgabe, die mir von Herrn Gabler zur Verfügung gestellt wurden. Die Druckausgabe ist von der Erfassung der einzelnen Textzeugen und ihrem Vergleich bis hin zur Satzvorbereitung mit dem Tübinger TUSTEP-System hergestellt worden;[24] auch für andere Zwecke wurden die Basisdateien in der Vergangenheit immer wieder transformiert, außerdem wurden für die zweite Auflage von 1986 noch einige Änderungen vorgenommen. Die Dateien, die mir zur Verfügung standen, sind offenbar nicht unmittelbar die, nach denen gedruckt wurde, sondern von einer logischen Vorstufe abgeleitet. An einzelnen Stellen gab es Inkohärenzen zwischen Lesetext und synoptischem Text, weil die mir vorliegende Fassung des Lesetextes anscheinend bereits einige der Änderungen zur zweiten Auflage hin enthielt, während der synoptische Text der ersten Auflage entsprach. Von diesen Einzelheiten abgesehen entsprechen die Dateien aber zeichengenau der Buchausgabe des *Ulysses*.

Es liegen drei Gruppen von Quelldateien vor: die Dateien `i01.ris` bis `i18.ris` enthalten den synoptischen Text, also die linken Seiten der Buchausgabe, für die achtzehn Kapitel. Fußnoten stehen nach ihrem Lemma im Text; die Zeilenreferenz wird von TUSTEP intern verwaltet und wurde beim Exportieren für meine Zwecke am Ende jeder Zeile explizit angehängt. Die Dateien `c01.ris` bis `c18.ris` enthalten den Lesetext, also die rechten Seiten der Buchausgabe, für die achtzehn Kapitel. Die Einträge für die historische Kollation stehen ebenfalls unmittelbar nach ihrem Lemma im Text. Die Zeilenreferenz des Lesetextes ist wiederum explizit in jeder Zeile codiert, hier allerdings am Anfang.[25] Die Datei `tn.ris` schließlich enthält den Text der *Textual Notes*.

Ich möchte die in diesen Quelldateien verwendeten Codes nicht erschöpfend behandeln, sondern nur auf einzelne, für die Verarbeitung wichtige Punkte hinweisen. Um überhaupt eine Vorstellung vom Aussehen dieser Dateien zu vermitteln, gebe ich den ersten Absatz des achten Kapitels (der auf S. 72 als synoptischer Text und daran anschließend als SGML-Text abgedruckt ist), zunächst aus dem synoptischen Text und dann aus dem Lesetext wieder (überlange Zeilen werden umbrochen und die Folgezeile etwas eingezogen):

```
$Pineapple%;316.1/0|><|
&~++}general_note{_&.TN~&%;316.1/01|><|
rock, lemon platt, butter scotch. A <lem> sugarsticky
     girl%;316.1/02|><|
shovelling scoopfuls of creams for a christian brother. Some school
     treat.%;316.2/0|><|
|B()Bad for their tummies.()|B{ Lozenge and comfit manufacturer to
     His%;316.3/0|><|
Majesty the King. God. Save. Our. Sitting on his
     throne##%;316.4/0|><|
&~##throne%]_aR;_throne,_tC~&%;316.4/01|><|
```

[24] Dieses System wird in angenehm lesbarer Form im *Lernbuch TUSTEP* vorgestellt.

[25] Es liegt keinerlei Notwendigkeit in Ort und Form dieser Codierungen, es hatte sich einfach so ergeben.

```
sucking%;316.4/02|><|
|D[jujubes.] red jujubes white.|D{%;316.5/0|><|

|>80001.1<|$Pineapple rock, lemon platt, butter scotch. A
     sugarsticky^M
|>80001.1/01<|girl^M
|>80002.1<|shovelling scoopfuls of creams for a christian brother.
     Some school treat.^M
|>80003.1<|Bad for their tummies. Lozenge and comfit manufacturer
     to His Majesty^M
|>80004.1<|the King. God. Save. Our. Sitting on his throne &x[]&x{^M
|>80004.1/01<|&xx1\xa3\xa3_throne]_throne,_::=22&.hy61<151.5>
     _&.sltC&xx{ sucking red jujubes white.^M
```

Am Ende der Zeilen in der Lesetextfassung befinden sich noch die *Carriage Return*-Codes '`^M`', weil der Text als MS-DOS-Datei importiert wurde. Da ich einige Zeit lang mit immer wieder neuen Versionen dieser Dateien experimentiert habe, habe ich bei der Dateiübertragung nicht immer darauf geachtet, die Zeilenenden der UNIX-Konvention entsprechend zu konvertieren, sondern stattdessen eventuelle *Carriage Returns* erst während meiner SGML-Konvertierung abgeschnitten.

Die Codes '`\xa3\xa3`' in der letzten Zeile stehen für die entsprechenden hexadezimalen Bytewerte; es kommen einige 8-Bit-Zeichen in den Dateien vor, die vermutlich in irgendeiner MS-DOS-*Codepage* Sinn ergeben. Im vorliegenden Fall sind sie Bestandteil einer Kontrollsequenz und das intendierte Zeichen spielt keine Rolle; wo solche Nicht-ASCII-Zeichen für druckbare Zeichen stehen, ließ sich ihre Bedeutung durch Vergleich mit dem gedruckten Buch ermitteln. Andere Zeichen werden durch Kontrollsequenzen codiert, zum Beispiel der *backslash* durch das '`&.sl`' in der letzten Zeile.

Die Gruppierung aufeinanderfolgender Kontrollseqeuenzen zu für die SGML-Konvertierung sinnvollen Token ist nicht immer trivial; auf jeden Fall braucht man einen gewissen *lookahead*, um beispielsweise '`|B`' (für "⌜B") von '`|B{`' (für "B⌝") unterscheiden zu können oder zu erkennen, daß die Klammern am Anfang von '`()|B{`' (für "(B⌝") nicht im normalen Text zu setzen sind, sondern zur Kontrollsequenz gehören.

Ein drastischer Fall von Kontextabhängigkeit ist die Zeichenfolge '`|1<|2`' (für "$^{\wedge}\langle^{>}$", zum Beispiel im auf S. 32 zitierten Textstück "$^{\wedge}\langle^{>}$ Hair on end.$\rangle^{\wedge}$"), die aus logischen Gründen als '`|1`' (Einfügung im Overlay erster Stufe) gefolgt von '`<|2`' (Streichung im Overlay zweiter Stufe) gelesen werden muß. Die Zeichenfolge '`|2<|1`' (für "$^{>}\langle^{\wedge}$" hingegen, in der nur die Stufenparameter für den Overlay vertauscht sind, muß aus logischen Gründen als '`|2<`' (Ersetzung im Overlay zweiter Stufe) gefolgt von '`|1`' (Einfügung im Overlay erster Stufe) gelesen werden. Das '`<`'-Zeichen wird also einmal der ersten Kontrollsequenz zugeordnet, einmal der zweiten. Für das Druckbild ist diese Differenzierung bedeutungslos, für die SGML-Darstellung aber ist sie wesentlich.[26]

Durch die Parametrisierung mit Siglen, zusätzlichen Symbolen des Textschichtenmodells und Indizes für Overlaystufen und durch die spezifischen Codierungen in Fußnoten und historischer Kollation ergibt sich eine erhebliche Anzahl zu unterscheidender Kontrollsequenzen; man kann natürlich unterschiedlich zählen, das Ergebnis liegt in etwa zwischen 400 und 700.

[26] Besonders fatal an diesem Beispiel ist, daß der lauernde Fehler erst sehr spät in Erscheinung tritt – eventuell erst beim Versuch, das konvertierte SGML-Dokument zu parsen, und zu diesem Zeitpunkt meint man natürlich, man hätte die Phase der Tokenerkennung längst hinter sich gelassen. Diese Form von weitreichender Rückkoppelung war für die Entwicklung der Konvertierungsverfahren charakteristisch; sie ist zunächst frustrierend, wenn man genug Zeit und Nerven hat, kann sie aber auch sehr lehrreich sein.

Die Kontrollsequenzen in den Quelldateien sind zum großen Teil frei gewählt und müssen keiner bestimmten Syntax gehorchen, da sie in TUSTEP durch Textersetzung mit regulären Ausdrücken zu einer satzfähigen Datei weiterverarbeitet werden. Beispielsweise sind einzelne Ziffern in den Dateien des synoptischen Textes Kontrollsequenzen für den Beginn einer Textschicht mit dieser Ziffernsigle; sollen Ziffern im Text gesetzt werden, wird ihnen ein Prozentzeichen vorangestellt. In den Fußnoten und in den Dateien des Lesetextes hingegen gilt das nicht, die dort im Text stehenden Ziffern werden in jedem Fall gesetzt.

Textdateien werden in TUSTEP durch ein explizites Referenzsystem optional in Seiten, auf jeden Fall aber in Datensätze und Wörter strukturiert, wobei ein Datensatz normalerweise einer Zeile entspricht, aber durch sogenannte 'Unterscheidungsnummern' in mehrere physische Zeilen unterteilt werden kann (vergleiche das TUSTEP-Lernbuch, S. 26–29). Diese interne Referenz wurde für das Referenzsystem der Buchausgabe genutzt und deswegen beim Dateiexport explizit in den Text eingefügt. Im ersten Beispiel oben steht der Zeilencode '`%;316.4/02|><|`' beispielsweise für die zweite Zusatzzeile zum Datensatz (logische Zeile) Nummer 4 auf Seite 316. Wörter sind die durch echte Leerzeichen getrennten Texteinheiten. Wenn beispielsweise "Y. M. C. A." als '`Y.=M.=C.=A.`' geschrieben wird, gilt die Abkürzung dadurch trotz des Ausschlusses zwischen den Buchstaben als ein einziges Wort; diese Definition von Wort ist wichtig, wenn der Geltungbereich des Fußnotensymbols ermittelt werden muß. Stünde im Text zum Beispiel "Y. M. C. A.°", so wäre das Lemma, auf das sich die Emendation bezieht, nicht nur "A.", sondern die gesamte Abkürzung.

Einige unerwartete technische Schwierigkeiten und Konvertierungsfehler, die sich am Ende der Arbeit bemerkbar machten, haben mir nachträglich gezeigt, wie wichtig es gewesen wäre, diese Grundgegebenheiten des TUSTEP-Systems stärker zu berücksichtigen. Die elektronische Edition die ich vorlege ist in mancher Hinsicht noch mangelhaft, und das liegt zum großen Teil daran, daß ich in meinen Konvertierungsverfahren das TUSTEP-Modell zu wenig berücksichtigt und dadurch wertvolle Strukturinformation verwischt habe. Solche Versäumnisse waren allerdings, angesichts der vielen Enden, von denen aus ich die Konvertierungsaufgabe gleichzeitig aufzudröseln hatte, wahrscheinlich unvermeidlich.

11. Der Entwurf des Konvertierungsverfahrens

Als ich mit nichts als fünfeinhalb Megabyte Rohdaten und zwei Bänden TEI Spezifikation in der Hand das Konvertierungsprojekt plante, ergaben sich nach und nach folgende Probleme und Lösungsansätze:

Die Konvertierung würde kaum von einem einzigen monolithischen Filterprogramm erledigt werden, das die TUSTEP-Dateien einlesen und fertiges SGML ausgeben würde. Die praktische Alternative war es, einen Satz von Modulen zu programmieren, die in einer Pipeline einzelne Konvertierungsaufgaben nacheinander erledigen konnten. Ich hatte eine Zeit lang daran gedacht, mit `lex` und `yacc` einen monolithischen *Ulysses*-Compiler zu schreiben, habe davon aber Abstand genommen. Entwurf und Test eines solchen Monolithen hätte wohl noch mehr Probleme bereitet, als es die `perl`-Module ohnehin taten, zumal die Codierung des Quelltextes ja keineswegs in einer geschlossenen Dokumentation beschrieben war, sondern im Verlauf der Konvertierung heuristisch ermittelt werden mußte. Außerdem ermöglicht der etwas anarchische Ansatz der in `perl` programmierten Konvertierungspipeline einige pragmatische Lösungen, die in der reinen Luft einer `yacc`-Grammatik vielleicht schwierig zu implementieren gewesen wären. Ob der gesamte *Ulysses* sich überhaupt in eine einigermaßen handhabbare Grammatik hätte bringen lassen, ist ohnehin nicht sicher.

Sobald ich mich aber von der Idee des monolithischen Konverters verabschiedet hatte, stellte sich die Frage eines Formats für die Zwischenergebnisse. Es mußte eine Möglichkeit gefunden werden, in einer gemischten Datei bereits erzeugte SGML-Daten von den noch vorhandenen TUSTEP-Daten zu unterscheiden. Ich dachte zunächst an besondere Zeichen im Text, die den Wechsel zwischen TUSTEP und SGML markieren sollten, oder daran, die

SGML-spezifischen Sonderzeichen in den TUSTEP-Quellen durch andere Kobinationen zu ersetzen, um so den SGML-Text unterscheiden zu können.

Die entscheidende Anregung zur Lösung dieses Problems verdanke ich aber Stefan Hermann, einem Mitarbeiter von Frau Brüggemann-Klein. Das Ergebnis ist ein Dateiformat, in dem der Textstrom in sinnvolle Token zerlegt wird, die zeilenweise abgespeichert werden. Jedem Token ist am Zeilenanfang ein Buchstabencode vorangestellt, der es in Typen und Untertypen klassifiziert. Durch diese Typologie kann '*out of band*' zwischen SGML-Daten und TUSTEP-Daten unterschieden werden; außerdem können beliebige weitere Differenzierungen durchgeführt werden, dazu später mehr. Daß auf diese Weise der Text in sinnvollen, kleinen Einheiten vorportioniert wird, ist für die Verarbeitung mit `perl` ebenfalls hilfreich. Dieser Ansatz hat allerdings auch dazu beigetragen, die impliziten Strukturen des TUSTEP-Formats zu verschleiern und so einige Ungenauigkeiten in das Ergebnis gebracht. Wenn ich von vorneherein stärker auf dieses Problem geachtet hätte, wäre das aber zu vermeiden gewesen.

Die für dieses Format nötige Zerlegung in Token erfolgt durch einen `lex`-Scanner. Während `perl` nur durch Tricks dazu zu bringen ist, den Textstrom sequentiell Zeichen für Zeichen durchzugehen, ist das für `lex` die natürliche Arbeitsweise. Außerdem erfolgt die Unterscheidung auch komplexerer Token schnell und zuverlässig. Nur für sehr wenige Sonderfälle mußte ich kleine Tricks anwenden, zum Beispiel für das oben geschilderte '`|2<|1`'-Phänomen.

Ein anderes Problem war, daß der Lesetext mit seiner Zeilenreferenz und der historischen Kollation einerseits und der synoptische Text mit den Emendationen andererseits in getrennten Dateien vorlagen; für die SGML-Edition mußten die Informationen aus beiden Formen aber in einem Textstrom zusammengefaßt werden. Der Schlüssel zu diesem Problem war, daß der synoptische Text, wenn man die eingearbeiteten Revisionen als Korrekturanweisungen auffaßt und durchführt, den Wortlaut des Lesetextes ergibt. Das machte es möglich, ein Programm zu schreiben, das beide Texte synchron durchgeht, dabei für den Vergleich nur den Lesetextgehalt des synoptischen Textes berücksichtigt, aber den vollständigen synoptischen Text wieder ausgibt. Sobald es dabei auf Daten aus dem Lesetext stößt, die in den synoptischen Text aufgenommen werden sollen, werden diese in den synoptischen Ausgabetextstrom eingefügt. Dieses Programm offenbarte bei seinem buchstabengenauen Vergleich auch viele der kleinen Fehler und Inkonsistenzen in den Quelldateien, von denen ich im vorigen Abschnitt gesprochen habe.

Es war abzusehen, daß die automatische Konvertierung nicht beim ersten Versuch perfekte Resultate liefern würde, also zur schrittweisen Verbesserung mehrmals durchlaufen werden müßte. Andererseits sollte es ab einem gewissen Punkt möglich sein, den SGML-Text an einzelnen Stellen manuell zu verändern. Das Dilemma dabei ist, daß nachträgliche Änderungen am Konvertierungsergebnis von erneuten automatischen Konvertierungsläufen überschrieben werden; eine manuelle Änderung erhebt die konvertierten Dateien in den Status von Quelldateien und verbietet eigentlich spätere Konvertierungsläufe.

Ich sah dafür zunächst nur die Lösung, die Fehlerfreiheit des Konvertierungsergebnisses möglichst zweifelsfrei sicherzustellen, bevor manuelle Änderungen beginnen würden. Auch eine automatisierte Form von manuellen Änderungen habe ich vorgesehen, in der durch ein `perl`-Modul bestimmte Stellen im Text durch die Referenzsysteme und Muster im Text identifiziert und dann auf vorgegebene Weise modifiziert werden. Diese Technik wird auch für einzelne Korrekturen innerhalb der Pipeline verwandt, die notwendig sind, um korrekt fortfahren zu können. Für die manuelle Bearbeitung des SGML-Textes in größerem Umfang ist diese Technik aber zu mühselig und auch zu fehleranfällig.

Die optimale Lösung fiel mir mehr oder weniger in den Schoß: ich hatte nicht sehr viel Erfahrung mit dem *Revision Control System* `RCS` (oder irgendeinem anderen dieser Systeme), entschloß mich aber, es zur weiteren Verwaltung der SGML-Dateien nach Abschluß der automatischen Konvertierung zu benutzen, um unbedachte Änderungen leicht zurücknehmen zu können. Als dann, nach den ersten manuellen Eingriffen, ein systematischer Konvertierungsfehler offenbar wurde, habe ich mich notgedrungen mit den Möglichkeiten von `RCS` beschäftigt. Es stellte sich heraus, daß die Funktion `rcsmerge` (beziehungsweise `co -j`) einen eleganten Ausweg aus dem dargestellten Dilemma bietet, indem sie es erlaubt, die Unterschiede zwischen zwei Ergebnissen des automatischen Konvertierungslaufs in die handbearbeiteten SGML-Dokumente hineinzumischen. Nur wo so ein Unterschied auf eine manuell geänderte Stelle berührt, ist eine Entscheidung des Benutzers nötig. Da die manuellen Änderungen immer nur verhältnismäßig kleine Teile des Textes betreffen, sind solche Kollisionen sehr selten.

⋆

Insgesamt ergibt sich so ein System, daß sehr modular aufgebaut ist und nachträgliche Änderungen gut toleriert. Diese Eigenschaften waren für das Projekt auch notwendig, da die Spezifikation der Konvertierung, die in einer idealen Welt am Anfang der gesamten Arbeit steht, hier ein heuristischer Prozeß war, der die praktische Konvertierungsarbeit bis zum Schluß begleitet hat. An einigen Stellen der vorangegangenen Abschnitte ist sicher deutlich geworden, daß ich diesen Prozeß nach wie vor nicht für abgeschlossen halte.

Diese allmähliche Herauskristallisierung von Strukturen und Lösungstechniken hat natürlich auch ihre Spuren in den Konvertierungstechniken hinterlassen. Einige elegante Techniken, die ich in den späteren Modulen entwickelt und verwendet habe, wären auch schon in den früheren Modulen nützlich gewesen. Einige der Erkenntnisse, die sich in der Zusammenschau (nicht zuletzt beim Niederschreiben dieser Arbeit) ergeben haben, könnten sinnvoll auf die Konzeption des Konvertierungsverfahrens angewandt werden. Die natürliche Trägheit, ein immerhin laufendes System noch einmal grundlegend zu verändern, hat dazu geführt, daß solche späten Erkenntnisse sich eher in kleinen Verbesserungen als in großen Neuentwürfen geäußert haben.

Ich möchte noch einmal betonen, daß es für eine qualitativ wirklich hieb- und stichfeste elektronische Edition des *Ulysses* günstig wäre, mit den gewonnenen Erkenntnissen neu anzusetzen und den Weg, den ich mit der vorgelegten Arbeit gebahnt habe, noch einmal und besser nachzugehen. Aber auch die vorgelegte Edition kann, mit einigen weiteren Verbesserungen, als solide Basis für einen elektronischen *Ulysses* verwendet werden.

Wenn die konkreten Programme auch zu spezifisch sind, um für andere Konvertierungsprojekte unmittelbar nützlich zu sein, so sind die erarbeiteten Verfahren und die dabei gewonnenen Erfahrungen doch sehr gut übertragbar.

12. Das Token-Format

Sämtliche Konvertierungsroutinen arbeiten auf einem Token-Format, in dem der Textstrom in kleine Einheiten zerlegt ist, die durch einen voranstehenden Buchstaben einen Typ zugewiesen bekommen; zusätzlich gehört jedes Token zu einer bestimmten Textsorte: dem normalen Text, den Fußnoten, der historischen Kollation oder den *Textual Notes*. Alle Textsorten außer dem normalen Text werden durch einen dem Typ vorangestellten Buchstaben bezeichnet. Die Datei `uly08.toks` beginnt zum Beispiel so (die entsprechenden TUSTEP-Originaldateien sind auf S. 75 abgedruckt, der gedruckte Kapitelanfang auf S. 72):

```
l [r 8 1]
l [c 316 1]
c $
w Pineapple
b 0a
n c &~
n c ++}
n w general
```

```
n b 5f
n w note
n c {
n b 5f
n c &.TN
n c ~&
b 0a
w rock
c ,
b 20
```

Eine Zeile in einer Tokendatei ist also folgendermaßen aufgebaut: Das erste Zeichen jeder Zeile ist ein Kleinbuchstabe, gefolgt von einem Leerzeichen. Falls der Kleinbuchstabe '`h`', '`n`' oder '`t`' ist, handelt es sich um eine Textsortenangabe, ansonsten um eine Typangabe. Wenn das erste Zeichen eine Textsortenangabe ist, ist das dritte Zeichen die Typangabe, wiederum gefolgt von einem Leerzeichen. Alle folgenden Zeichen bis zur Zeilenendemarkierung (in UNIX ein *Line Feed*) sind die Daten.

Das Verarbeiten der Tokendateien würde natürlich leichter fallen, wenn alle Textsorten einen Kennbuchstaben hätten und die Fallunterscheidung wegfiele. Am Anfang der Arbeit war mir die Zweckmäßigkeit der Textsortenbildung aber noch nicht klar gewesen, und einige Routinen waren bereits geschrieben. Zu dem Zeitpunkt, als die Fallunterscheidung lästiger wurde und auch die Rechenzeit der Konvertierung ins Gewicht zu fallen begann, erschien mir die Umstellung aller bisherigen Routinen als zu aufwendig – vielleicht wäre es aber auch dann noch günstig gewesen.

Ein noch schwererer Fehler war es, nicht sorgfältig zwischen Kontrollsequenzen des eigentlichen Textes und denen des Metatextes zu unterscheiden. Wenn beispielsweise ein `<anchor>` für ein Lemma plaziert wird, sollte er innerhalb von Metatext-Kontrollsequenzen zu liegen kommen, ohne dabei aber Satzzeichen oder Zahlen zu übergehen. Diese Unterscheidung, die bei der Tokenerkennung relativ leicht vorzunehmen gewesen wäre, erschien mir am Anfang noch nicht notwendig. Später mußte ich sie mit zusätzlichem Aufwand als Mustererkennung in einzelne Routinen einbauen, weil ich dann wiederum schon die Nebenwirkungen einer grundlegenden Umstellung des Tokensystems fürchtete.

Die folgende Übersicht beschreibt die verschiedenen Kennbuchstaben und ihre Bedeutung.

Gruppe 1:

- `n` Textsorte für emendatorische Fußnoten.
- `h` Textsorte für Einträge der historischen Kollation.
- `t` Textsorte für die *Textual Notes*.

Gruppe 2:

- `w` 'word', Daten können unverändert in die SGML-Datei übernommen werden.
- `c` 'code', normalerweise eine TUSTEP-Kontrollsequenz. Jedenfalls müssen die Daten weiterverarbeitet werden.
- `l` 'line', ein Referenzpunkt; die Daten haben entweder das Format '`[r` *Kapitel Zeile*`]`' oder '`[c` *Seite Zeile*`]`', je nachdem ob sie sich auf den Lesetext oder den synoptischen Text beziehen.
- `b` 'blank', ein Leerraum; die Daten sind der Hexadezimalcode des Leerzeichens, so daß zwischen gewöhnlichem Leerzeichen und Zeilenwechsel unterschieden werden kann.
- `d` Ein 8-Bit Sonderzeichen, das ebenfalls durch seinen Hexadezimalcode repräsentiert wird, damit es leichter zu lesen und zu verarbeiten ist.
- `x` Eine unbekannte Kontrollsequenz – dieser Typ war während der Entwicklung der Konvertierung sehr wichtig, kommt jetzt aber nicht mehr vor.

Gruppe 3:

- `s` Ein SGML-Tag.
- `e` Eine SGML-Entity Reference.

Gruppe 4:

- `a` Eine der Standardabkürzungen im editorischen Kommentar, wie '*PCU*' für '*PRINTER'S COPY UNCHANGED*.'
- `r` Ein Querverweis auf eine andere Textstelle, wie '316.4'.
- `z` Eine Zeugensigle wie 'tC' oder 'aR'.
- `f` Eine Angabe zu den gedruckten Editionen, die eine Lesart der historischen Kollation enthalten, wie '26–61'.

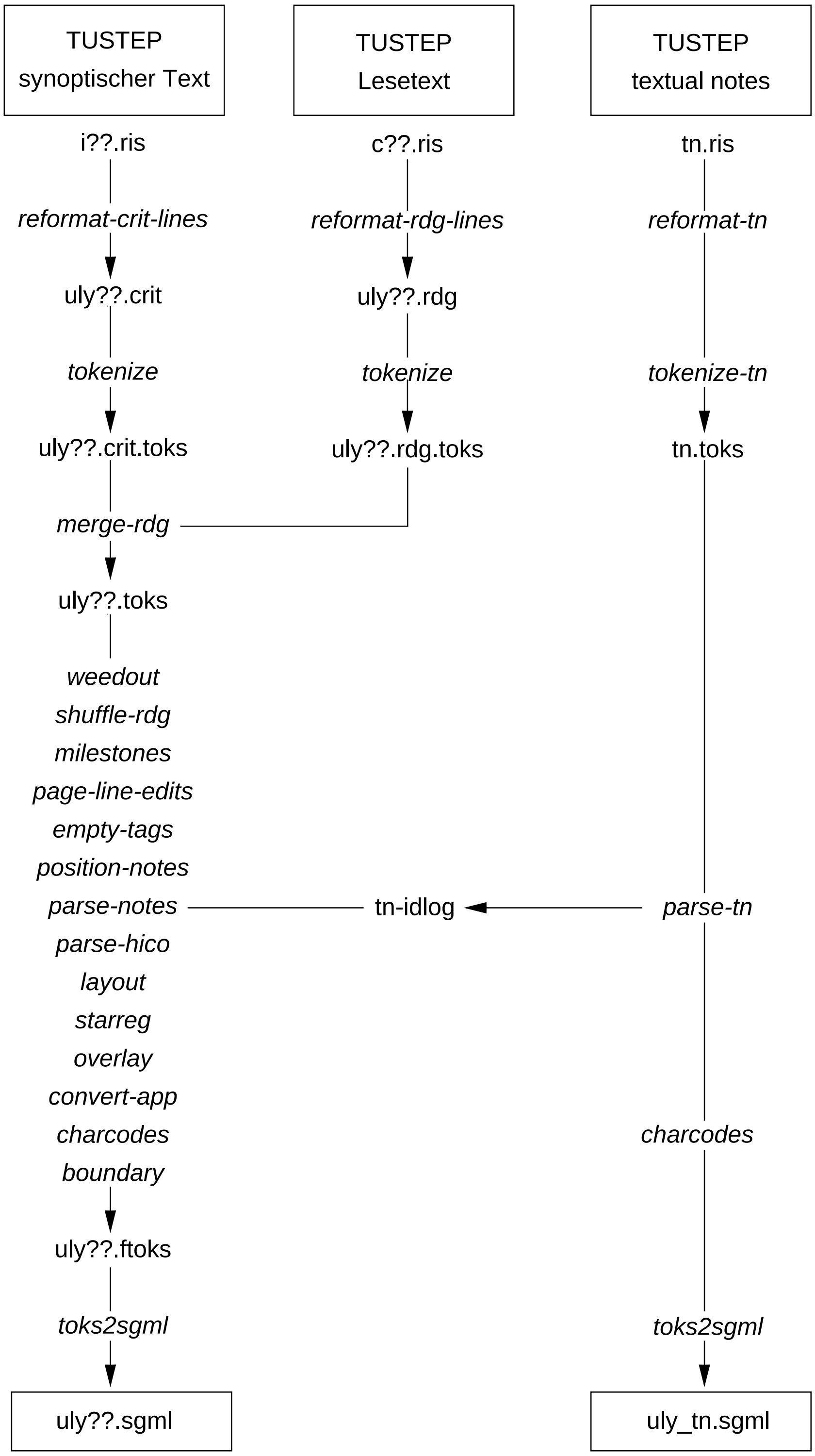
Die erste oben aufgeführte Gruppe sind die Textsortenzeichen. Die zweite Gruppe stellen eine minimale Gruppe von Typen für die Codierung der TUSTEP-Dateien dar. Die dritte Gruppe sind die beiden Typen, mit denen SGML-Daten in die Tokendatei eingefügt werden; die Unterscheidung zwischen Tags und Entity References ist für die Formatierung der resultierenden SGML-Datei nützlich. Die letzte Gruppe erleichtert die Verarbeitung der Fußnoten, der *Textual Notes* und der historischen Kollation, indem sie bestimmte Datenmuster bereits vorerkennt.

Die Token-Dateien werden durch die Programme `tokenize` und `tokenize-tn` erzeugt. Dank der Typenunterscheidung im Textstrom können anschließend die einzelnen Konvertierungsmodule den Text nach und nach mit SGML-Material anreichern ohne sich gegenseitig zu beeinflussen, bis das Exportprogramm `toks2sgml` schließlich nur die Datentypen `w`, `b`, `s` und `e` berücksichtigt und zu einer SGML-Datei formatiert.

13. Der Ablauf der automatischen Konvertierung

Die Graphik auf der folgenden Seite dient dazu, einen Überblick über den Ablauf zu gewinnen. Eine weniger anschauliche Darstellung derselben Struktur ist das `Makefile`, das diesen Ablauf steuert und im Anhang abgedruckt ist. Eine weitere Handreichung ist die folgende Tabelle, die zu jeder Routine kurz ihre Aufgabe skizziert.

`reformat-crit-lines` `reformat-rdg-lines` `reformat-tn`	Normalisierung von Zeilennummern und Zeilenenden sowie Ausbessern von Fehlern, die `tokenize` bzw. `tokenize-tn` behindern.
`tokenize` `tokenize-tn`	Mit `lex` erzeugte Programme, die TUSTEP-Kontrollsequenzen erkennen und das Tokenformat erzeugen.
`merge-rdg`	Liest synchron Lesetext und synoptischen Text und gibt den mit Zeilenreferenz und historischer Kollation angereicherten synoptischen Text aus.
`weedout`	Entfernt nicht verwendete Kontrollsequenzen.
`shuffle-rdg`	Versucht, die von `merge-rdg` eingebrachten Kontrollsequenzen etwas sinnvoller zu positionieren.
`milestones`	Erzeugt `<pb>`s und `<lb>`s für die Referenzsysteme.
`page-line-edits`	Ein Versuch, die notwendigen lokalen Eingriffe in einer Routine zusammenzufassen, dank `RCS` nahezu überflüssig.
`empty-tags`	Fügt die leeren SGML-Elemente `<space>`, `<gap>`, `<emptyadd>` einfach durch lokales Ersetzen ein.
`position-notes`	Fügt die `<anchor>`s um die Lemmata von Emendationen und historischer Kollation ein, positioniert den Text dieser Anmerkungen dort, wo er hin soll.
`parse-tn`	Erzeugt alle Strukturen für die *Textual Notes* und legt die Datei `tn-idlog` mit der Zuordnung der IDs an.
`parse-notes`	Erzeugt die Strukturen für die Emendationen, für Verweise auf die *Textual Notes* wird die Datei `tn-idlog` benötigt.
`parse-hico`	Erzeugt die Strukturen für die historische Kollation.
`layout`	Erzeugt Strukturen des Lesetextes wie `<hi>`, `<p>`, `<div>`, `<sp>`, `<lg>` usw.
`starreg`	Nur für Kapitel 15 und 18, wo ein Sternchen im Text eine Normalisierung bzw. eine Satzzeichenstreichung ohne Level angibt.
`overlay`	Erzeugt `<del type=cc>` und die `<overlay>`-Strukturen.
`convert-app`	Erzeugt die `<syn>`-Strukturen.
`charcodes`	Erzeugt Entity References anstelle diverser Kontrollsequenzen und Sonderzeichen.
`boundary`	Ersetzt in bestimmten Kontexten Struktur-Tags durch `<boundary>`.
`toks2sgml`	Liest den für die SGML-Erzeugung relevanten Teil aus Tokendateien und erzeugt eine formatierte SGML-Datei.



Eine graphische Veranschaulichung der automatischen Konvertierung; Dateinamen sind in Helvetica recte gesetzt, Programmnamen in schräggestellter Helvetica. Die ?? in den Dateinamen stehen für die Zahlen 01 bis 18.

Nicht alle diese Konvertierungsroutinen sind ein Lehrstück in sauber strukturierter und dokumentierter Programmierung, viele sind typische Einwegprodukte – was durch die Programmiersprache Perl auch eher unterstützt als vermieden wird. Das Werkzeug `perl` hat sich in mehrerer Hinsicht als zweischneidiges Schwert erwiesen. Innerhalb des Arbeitsbereichs, für den es geschaffen wurde, erlaubt es schnelles und flexibles Programmieren; verläßt man aber diesen Bereich, machen sich vor allem die Beschränkungen der Datenstrukturen und ihrer Verarbeitung unangenehm bemerkbar. Außerdem verführt die Sprache zu einem unstrukturierten BASIC-Stil, der sich schnell gegen den Programmierer kehrt, wenn er später obskure Konvertierungsfehler lokalisieren will.

Ein weiteres Problem, das ich anfangs völlig unterschätzt habe, war die Rechenzeit, die eine Pipeline aus bis zu zwanzig `perl`-Programmen braucht. Die Laufzeit von ein bis zwei Stunden für einen vollen Konvertierungsdurchgang spielt für die einmalige Produktion der elektronischen Fassung natürlich keine Rolle, behindert das Testen aber erheblich.

Aus diesen Gründen würde ich heute in jedem Einzelfall überlegen, ob eine bestimmte Routine nicht besser durch ein C-Programm erledigt werden kann. Natürlich sind die sehr flexiblen regulären Ausdrücke, die problemlose Speicherverwaltung und die assoziativen Stringzuordnungen als eingebaute Hilfsmittel zunächst ein Pluspunkt für `perl`; der einmalige Zeitaufwand für das Einarbeiten in eine C-Funktionsbibliothek könnte sich aber für einige komplexere Aufgaben trotzdem amortisieren.

Da laut Frederic Brooks aber die Arbeitszeit und Fehlerhäufigkeit in etwa proportional zur Länge des Programmtextes ist, egal in welcher Sprache, wird man die aufwendigere C-Programmierung wohl in erster Linie nur für fest definierte, häufig gebrauchte und möglichst wiederverwendbare Lösungen vorsehen.

⋆

Den Einsatz von spezifischen Programmgeneratoren, insbesondere `lex` und `yacc`, kann ich für geeignete Aufgaben nur sehr empfehlen. Die Routinen `tokenize` und `tokenize-tn` liefen schnell und fehlerfrei, waren leicht zu schreiben und zu erweitern, und die Syntax und Arbeitsweise von `lex` waren mit ganz wenigen Ausnahmen wie maßgeschneidert für die gestellte Aufgabe. Hier hätte der Einsatz eines `perl`-Programms ungleich mehr Arbeitsaufwand und wohl einen spürbaren Effizienzverlust bedeutet.

Eine andere mögliche Alternative zu handgestricktem `perl` wären kommerzielle Konvertierungstools gewesen, und wo sie fertig installiert zur Verfügung stehen, sollte man diese Alternative sicher gründlich prüfen. Für mich kamen sie aus praktischen Gründen kaum in Frage, da die zu verwendenden Werkzeuge auf meinem häuslichen Linux-PC installierbar sein mußten. Für meine Zwecke unbrauchbar wären Programme, die effektiv nur eine DTD in eine andere transformieren, also die Formulierung einer DTD für das Ausgangsformat voraussetzen. Ebenfalls große Bedenken hätte ich, jedenfalls für Projekte von der Komplexität der *Ulysses*-Konvertierung, gegen monolithische Konverter, die keine wirklich allgemeine Programmierschnittstelle zur Verfügung stellen. Die Gefahr, irgendein Teilproblem mit den vorhandenen Hilfsmitteln nicht lösen zu können und keine neuen herstellen zu dürfen, scheint mir relativ groß.

Die letzte zu diksutierende Möglichkeit wäre die Konvertierung innerhalb des TUSTEP-Systems gewesen, das für akademische Zwecke kostenlos zur Verfügung steht. Die Arbeit mit TUSTEP hätte den Vorteil gehabt, daß man in der 'natürlichen Umgebung' der Quelldateien arbeitet, was manche Schwierigkeiten vermeiden hilft. TUSTEP ist aber ein ziemlich hermetisches System, vergleichbar einer UNIX-Umgebung, in der hauptsächlich `ed`, `sed` und `troff/nroff` zur Verfügung stehen. Trotz zusätzlicher Spezialroutinen und der Möglichkeit, Kommandosequenzen genauso wie Textdaten zu verarbeiten, verfügt es nicht über eine allgemeine Programmierschnittstelle zur Bearbeitung der Textdaten, womit wieder die eben angesprochene Gefahr der Sackgasse droht.

Zudem hatte ich, offen gestanden, wenig Lust auf die Abwanderung von den mir vertrauten Werkzeugen einer offenen UNIX-Plattform in die sehr eigene Welt dieses Programms. So blieb es letztlich bei `perl`, das die meisten allgemeinen Funktionen von TUSTEP äquivalent anbietet.[27] Das sollte aber nicht als eine Herabwürdigung eines nicht nur wegen seines Alters sondern auch wegen vieler besonderer Leistungsmerkmale beachtlichen Programms mißverstanden werden. Das auf S. 334-344 im *Lernbuch TUSTEP* beschriebene Kommando '`vergleiche`' beispielsweise erledigt den Kern der Textvorbereitung für eine kritische Edition und hat bei der Ermittlung der Textschichten im Gabler-Modell die Hauptarbeit erledigt. Ebenfalls interessant finde ich, daß das TUSTEP-Satzprogramm den Zeilenfall eines umbrochenen Textes als Referenzsystem in eine Kopie der Quelldateien zurückschreiben kann; so wurde auch die Zeilenreferenz für den *Ulysses* erzeugt.

⋆

In den folgenden Abschnitten möchte ich die Verfahren der automatischen Konvertierung im Einzelnen darstellen. Zu den interessanteren Routinen werden auch technische Details angegeben. Alle diese Routinen sind Filter, die den Eingabedatenstrom bearbeiten und das Ergebnis als Ausgabedatenstrom liefern. Bis auf den Anfang der Pipeline arbeiten alle auf dem oben beschriebenen Tokenformat, das sich für die Verarbeitung mit `perl` gut eignet.

Die Möglichkeit der Anreicherung der Tokendateien mit SGML-Daten macht die Routinen einigermaßen unabhängig voneinander, dennoch ist die Reihenfolge nicht völlig beliebig; zum Beispiel werden Kontrollsequenzen zur Zeichensatzwahl von einigen Routinen spezifisch ausgewertet, bevor die übrigen derartigen Sequenzen von `layout` durch `<hi>`-Elemente ersetzt werden. Andererseits ist beispielsweise `parse-notes` auf die Vorarbeit von `position-notes` angewiesen, und `boundary` macht nur nach `overlay` Sinn.

13.1. *reformat-crit-lines, reformat-rdg-lines, reformat-tn*

Diese Routinen lesen die rohen aus TUSTEP exportierten Dateien, wie ich sie von Herrn Gabler erhalten habe und bereiten sie für die Zerlegung in Token vor. Dazu gehört erstens die Beseitigung von MS-DOS-Zeilenenden und eine vereinheitlichte Darstellung der Kapitel / Zeilen- beziehungsweise Seiten / Zeilen-Referenz am Anfang jeder Zeile. Zweitens wurden einige Zeichenumstellungen und Ersetzungen systematisch durchgeführt, um die Tokenbildung zu erleichtern oder weil sie im noch zusammenhängenden Text am leichtesten vorzunehmen sind. Beispielsweise werden Bindestriche am Zeilenende an dieser Stelle durch die Kontrollsequenz für *soft hyphens* ersetzt, und eine Kontrollsequenz, die vermutlich überlange Absätze in technisch verdauliche Portionen eingeteilt hat, wurde, um die spätere Behandlung echter Absätze zu erleichtern, durch ein Leerzeichen ersetzt.

Drittens ist dieses Programm die erste Stelle, an der die Referenzsysteme des Textes ausgenützt werden, um bestimmte Stellen zu korrigieren; die hier vorgenommenen Korrekturen sind teilweise Korrekturen für fehlende oder defekte Kontrollsequenzen im Text, teils sind sie nötig gewesen, um die erwähnten Inkonsistenzen zwischen Lesetext und synoptischem Text zu reparieren. Von diesen Korrekturen konnten die rein technisch bedingten, die vor allem mit Worttrennung zu tun hatten, nach einer Verbesserung von `merge-rdg` wieder wegfallen. Bei den *Textual Notes* waren einzelne Referenzangaben falsch und mußten korrigiert werden.

Die in anderen Routinen wiederkehrenden Programmzeilen für solche lokalen Eingriffe sehen zum Beispiel so aus:

```
if (($page == 30) && ($line == 37)) {
    s/(sorrow\:)(_a1)/$1\%\]$2/;
```

[27] Ich habe mit der Version 4 gearbeitet. Version 5 führt unter anderem objektorientierte Programmiertechniken ein, die möglicherweise das strukturierte Programmieren erleichtern. Möglicherweise verstärken sie aber auch nur die Unübersichtlichkeit.

```
} elsif (($page == 36) && ($line == 33)) {
    s/ferule/ferrule/;
```

Die Reihenfolge der Abfrage von `page` beziehungsweise `chapter` und `line` verdient, der Effizienz wegen, ein kurzes Nachdenken, da immerhin allein in `reformat-crit-lines` 36 Korrekturkandidaten auf 'ihre' Textzeile warten: Für die synoptische Referenz nach Seiten und Zeilen sollte zuerst die Seite überprüft werden, da dann in nur rund 35 Fällen die zweite Bedingung überprüft werden muß, im Gegensatz zu rund 850 Fällen sonst. Für die Lesetext-Referenz hingegen sollte die Zeilenabfrage zuerst erfolgen, da dann die zweite Bedingung höchstens achtzehn Mal überprüft wird, anstelle von circa 1000-5000 Fällen bei umgekehrter Reihenfolge. Für längere Korrekturlisten kann es noch sinnvoller sein, die Zeilenabfragen kapitelweise zu gruppieren und nur durchzuführen, wenn wir uns im betreffenden Kapitel befinden.

Durch geschickte Bereichsabfragen könnte man die Zahl der nötigen Vergleich natürlich noch deutlich senken; aber da die Liste dieser Eingriffe ständig ergänzt werden muß und die benötigte Rechenzeit trotz allem nicht übermäßig ins Gewicht fallen dürfte, habe ich keinen weiteren Aufwand getrieben. Wenn ich allerdings noch einmal von vorne anfangen würde, würde ich schon als Arbeitserleichterung für diese Korrekturen eine effiziente Funktion mit einer klaren Schnittstelle programmieren.

13.2. *tokenize, tokenize-tn*

Diese Programme fallen insofern aus dem Rahmen, als sie mit `lex` erstellt worden sind. Ihre Aufgabe war es, den Zeichenstrom der TUSTEP-Dateien in die auf S. 80 definierten Token zu zerlegen, insbesondere also die Kontrollsequenzen richtig zu erkennen. Mit 371 Regeln ist `tokenize.l` ein ziemlich umfangreicher Scanner; allerdings ließen sich diese Regeln drastisch vereinfachen und zusammenfassen. Der Scanner war zu Beginn der Arbeit ein Werkzeug, um noch unbekannte Codes im Text zu finden, um so ihre Funktion zu bestimmen; deshalb wurden die bereits erkannten Codes so genau wie möglich spezifiziert, damit keine noch nicht identifizierten Codes als Trittbrettfahrer durch die Maschen der regulären Ausdrücke schlüpfen konnten.

Für in den Regeln immer wiederkehrende Zeichengruppen, vor allem die Siglen im Textschichtenmodell, in den Fußnoten und der historischen Kollation, wurden Ersetzungsvariablen (*Substitutions*, siehe S. 153 in 'lex & yacc') definiert, um die Regeln übersichtlicher zu machen und unsystematische Fehler zu vermeiden.

Der Zuordnung der Token zu Textgruppen entspricht in der Scanner-Spezifikation die Einführung von Startzuständen; der Grundzustand entspricht dem synoptischen Text, daneben gibt es einen nicht-exklusiven Zustand `RDG` für den Lesetext und exklusive Zustände `FOOT` und `HIST` für die Emendationen und die historische Kollation.[28]

Daß die Zustände `FOOT` und `HIST` trotz einiger für alle Zustände gemeinsamer Kontrollsequenzen exklusiv sind, liegt daran, daß viele Kontrollsequenzen in diesen Anmerkungen anders interpretiert werden müssen; deswegen lag mir an einer separaten Identifikation der vorkommenden Kontrollsequenzen. Ein Beispiel ist die bereits an anderer Stelle besprochene Zeichenfolge '`%1`', die im Haupttext eine Kontrollsequenz für die Ziffer eins ist, während es diese Kontrollsequenz in den Anmerkungen nicht gibt und die Ziffer eins durch eine einfache '`1`' dargestellt wird.

An manchen Stellen waren Tricks nötig, um die richtigen Token zu erzeugen; das Problem der '`|2<1`'-Zeichenfolge habe ich bereits auf S. 76 geschildert. Um es zu lösen, erkennt der Scanner die gesamte Zeichenfolge, vergleicht dann die Parameter und gibt die jeweils richtigen zwei Token nacheinander aus:

[28] Nicht-exklusive Startzustände können, im Gegensatz zu den exklusiven, die allgemeinen Regeln mitbenutzen; vergleiche S. 171–173 im Handbuch. Nicht alle Varianten von `lex` implementieren exklusive Startzustände.

```
"|"{Level}"<|"{Level}   { if (yytext[1] > yytext[4]) {
                                printf ("c %c%c%c\n",
                                yytext[0], yytext[1],
                                yytext[2]);
                                printf ("c %c%c\n",
                                yytext[3], yytext[4]);
                          } else {
                                printf ("c %c%c\n",
                                yytext[0], yytext[1]);
                                printf ("c %c%c%c\n",
                                yytext[2], yytext[3],
                                yytext[4]);
                        }}
```

Ein anderes Problem war, daß ein bestimmter Typ von Emendation die Buchstabenfoge '`FSEOP`' unmittelbar vor dem ersten und nach dem letzten Wort der Fußnote stehen hat; da `lex` stets die längstmögliche Zeichenfolge erkennt, funktioniert eine Regel, die nur die Buchstabenfolge '`FSEOP`' erkennen soll, nicht. Hier war es am einfachsten, in der vorgeschalteten Routine `reformat-crit-lines` die '`FSEOP`'s mit je einem '`@`' zu umrahmen, so daß sie mit den benachbarten Wörtern nicht zusammenlaufen.

Der Scanner für die *Textual Notes* ist eher simpel, kommt mit 36 Regeln aus und hat wenig bemerkenswerte Eigenschaften.

13.3. *merge-rdg*

Dieses Programm ist der erste große Zeitfresser der Pipeline – sowohl im Hinblick auf die Rechenzeit angeht, als auch im Hinblick auf die Arbeitszeit bei Entwurf, Programmierung und Nachbearbeitung. Es dient dazu, in den Tokenstrom des synoptischen Textes die Token für historische Kollation und Lesetextreferenz einzufügen.

Dazu werden die `w`-Token von synoptischem Text und Lesetext eines nach dem anderen verglichen, wobei für den synoptischen Text nur diejenigen Token berücksichtigt werden dürfen, die nicht innerhalb von Streichungen oder Fußnoten liegen. Da Streichungen auch ineinandergeschachtelt vorkommen, werden die jeweils erwarteten Endezeichen in einem Stapel verwaltet; erst, wenn der Stapel völlig abgebaut ist, gehören die `w`-Token wieder zum Lesetext.

Sollten die so gefundenen Lesetexte sich unterscheiden, liegt entweder ein Programmfehler oder ein Fehler in einer der Quelldateien vor. In diesem Fall versucht das Programm nicht, neu aufzusetzen, sondern beendet die Arbeit mit einer Fehlermeldung.

Während der Lesetext von einem `w`-Token zum nächsten weitergelesen wird, werden die unterwegs gefundenen `l`- und `h`-Token zwischengespeichert. Sie werden in den Datenstrom des synoptischen Textes eingefügt, der anschließend seinerseits bis zum nächsten `w`-Token weitergelesen und dabei ausgegeben wird. Da zwischen zwei zum Lesetext gehörenden `w`-Token im synoptischen Text sehr viel Material stehen kann, ist das Finden einer sinnvollen Position für die eingefügten Token keine einfache Aufgabe; das wird von `merge-rdg` aber ignoriert – die eingefügten Token stehen einfach an der frühest möglichen Stelle. Die nachgeschaltete Routine `shuffle-rdg` versucht eine bessere Plazierung, aber das Ergebnis ist trotzdem oft dürftig.

Eine zusätzliche Erschwernis war, daß der Text in der Lesefassung manchmal anders als in der synoptischen Fassung auf die `w`-Token verteilt ist. Das liegt an unterschiedlichen Silbentrennungen, an eingefügten *soft hyphens*, manchmal auch an Metatext-Material. Die ersten derartigen Probleme habe ich durch das künstliche Aufspalten der schuldigen Wörter in `reformat-crit-lines` beziehungsweise `reformat-rdg-lines` zu lösen versucht.

Als die Zahl dieser Fälle jedoch immer weiter stieg, habe ich das oben skizzierte Verfahren so abgeändert, daß der eingelesene Text in beiden Textströmen in Strings als *queue* gepuffert wird. Beim Vergleich wird der kürzere String mit dem entsprechenden Teilstück des jeweils anderen verglichen, die verglichenen Stringanfänge können dann gelöscht werden. Im Eingabestrom wird erst weitergelesen, wenn die entsprechende *queue* leer ist, mit dem gelesenen Token wird der Stringpuffer aufgefüllt. Die unten wiedergegebene Hauptschleife von `merge-rdg` veranschaulicht das Verfahren vielleicht; da das Programm meiner Ansicht nach eines der interessantesten der ganzen Pipeline ist, habe ich es im Anhang vollständig abgedruckt. (Zum Verständnis: die Variablen `hicostring` und `linestring` dienen als Zwischenspeicher für die Token, die aus dem Lesetext in den synoptischen Text übertragen werden. `rdgstream` und `critstream` sind die Queuepuffer.)

```
mainloop: for (;;) {
    $rdgstream || &next_rdgword;
    print $hicostring;
    print $linestring;
    $critstream || &next_critword;
    $hicostring = $linestring = '';
    &match;
```

Ein kleine Schwäche dieses Programms liegt in der Fehlerkontrolle: es überprüft zwar am Schluß, daß die Zahl der gefundenen Lesetextzeilen mit der zuletzt ausgegebenen Zeilennummer übereinstimmt, dadurch wird aber noch nicht entdeckt, wenn der synoptische Text aus irgendeinem Grund irgendwo aufhört, Lesetext-Token zu finden – typischerweise, weil ein Streichungsende nicht gefunden wurde. In so einem Fall bleibt ein großer Teil des Lesetextes eines Kapitels unausgewertet, ohne daß eine Fehlermeldung erfolgt. Als ich dieses Phänomen endlich bemerkt hatte, habe ich es während der Verbesserungen durch Vergleich der vom Programm ausgegebenen letzten Zeilennummer mit der tatsächlichen Zeilenzahl des Kapitels kontrolliert. Es wäre klüger gewesen, von vorneherein einen solchen Vergleich ins Programm einzubauen.

13.4. *weedout*

Es gibt Momente beim Schreiben einer Diplomarbeit, während man die Arbeit der vergangenen Monate noch einmal mit Abstand und im Überblick sieht, wo man stolz ist, ja fast staunt über die Entwürfe und ihre Realisierungen, die man zustande gebracht hat. In anderen Momenten allerdings fragt man sich, ob, und wenn ja, was man überhaupt gedacht hat – und ein solcher Fall ist `weedout`. Es dient dazu, die für die Konvertierung nicht benötigten und eventuell störenden Codes '`[[]]`' und '`%\\`' in allen Textgruppen zu entfernen. Warum ich das aber nicht, wie für einige andere derartige Codes, in `reformat-crit-lines` erledigt habe, weiß ich beim besten Willen nicht.

13.5. *shuffle-rdg*

Die Aufgabe, die aus dem Lesetext in den synoptischen Text transportierten Token an eine sinnvolle Stelle zu bringen, wurde nicht in die Routine `merge-rdg` aufgenommen, um das Jonglieren mit Texten dort durchschaubar zu halten. Die Routine `shuffle-rdg` erledigt diesen Teil des Mischens, indem sie die entsprechenden Token aus dem Textstrom entfernt und zwischenspeichert, bis sie das nächste `b`-Token oder einen Absatzanfang, also Leerraum, findet. Da an diesem Leerraum das vorhergehende Wort zu Ende sein muß, werden vor dem Leerraum die gefundenen Token einer historischen Kollation eingefügt. Weil andererseits ein Zeilenwechsel fast (s.u.) immer mit einem Leerraum einhergeht, wird der Zeilenreferenzcode nach diesem Leerraum abgelegt.

Die Position der Einträge für die historische Kollation scheint auf diese Weise sinnvoll getroffen zu werden – da das Lemma, auf das sie sich beziehen, durch ihre Position bestimmt wird, ist es wichtig, sie nicht von 'ihrem' Wort zu trennen. Die Referenz-Elemente für den

Zeilenwechsel des Lesetextes allerdings liegen im Endergebnis manchmal an merkwürdigen Stellen; was am Zusammenspiel mit anderen Plazierungsverfahren, zum Beispiel für Absatzendecodes, liegen dürfte. Sie erfüllen aber auf jeden Fall die Anforderung, *nach* dem letzten Wort der vorhergehenden und *vor* dem ersten Wort der nachfolgenden Zeile zu stehen.

Das 'fast' im vorvorigen Absatz weist auf eine weitere Anomalie hin, die ich sehr lange übersehen hatte: im seltenen Fall einer Worttrennung am Zeilenende sollte der Referenzcode überhaupt nicht verschoben werden, falls man auf korrekte Zeilenreferenz wert legt. Falls man (eigentlich unberechtigterweise) fürchtet, dadurch das betroffene Wort zu zerteilen, ist es hingegen tatsächlich sinnvoll, die Referenz nach das getrennte Wort zu stellen.

13.6. *milestones*

Das Referenz von Lesetext und synoptischem Text ist im Text durch `l`-Token eingetragen. Diese `l`-Token bleiben durch die gesamte Pipeline erhalten, da sie verschiedenen Routinen zum Auffinden bestimmter Textpositionen dienen.

Das Programm `milestones` fügt die SGML-Elemente `<lb>` und `<pb>` für beide Referenzsysteme ein. Redundante Tags werden dadurch vermieden, daß die Werte für die 'laufende' Seite, synoptische Zeile und Lesetextzeile mitgeführt werden; nur wenn ein `l`-Token einen dieser Werte ändert, wird ein entsprechendes SGML-Element ausgegeben. Dieser Mechanismus betrifft vor allem die Seitenangaben mit `<pb>`, da mehrfache Zeilen mit der gleichen Nummer, wie sie in den TUSTEP-Dateien noch vorkommen, schon von `reformat-crit-lines` beziehungsweise `reformat-rdg-lines` abgefangen werden.

13.7. *page-line-edits*

Nach den `reformat`-Routinen ist das Programm `page-line-edits` die zweite Stelle, an der mit Hilfe des Referenzsystems und der Suche nach bestimmten Textmustern Modifikationen an einzelnen Textstellen vorgenommen werden. Während die Eingriffe in den `reformat`-Routinen in erster Linie Reparaturen der Quelldateien waren, handelt es sich bei den `page-line-edits` hauptsächlich um das Einfügen von SGML-Elementen, die an ganz bestimmten Stellen für Sonderfälle nötig sind und nicht durch die allgemeinen Verfahren erzeugt werden.

Dieses Programm ist also eine erste Antwort auf das Dilemma zwischen der Wiederholbarkeit des Konvertierungsprozesses und der Notwendigkeit manueller Eingriffe (vergleiche die Diskussion auf S. 78). Die lokal nötigen Eingriffe sollten durch ein Programm vorgenommen werden, so daß sie bei neuen Konvertierungsläufen nicht verlorengehen würden. Es ist selbstverständlich mühsam und langwierig, einen Text auf diese Weise zu ändern; und die Zahl von manuellen Änderungen, die ich später direkt in den SGML-Dateien vorgenommen habe, wäre auf diese Weise sicher nicht möglich gewesen.

Seit ich die Möglichkeiten des *Revision Control System* zu nutzen gelernt habe, ist das Verfahren von `page-line-edit` nur noch von historischem Interesse. Allerdings hat es eine nötige Korrekturfunktion mit übernommen, indem es für das siebte, vierzehnte und achtzehnte Kapitel die erste Lesetextreferenz, die von `merge-rdg` und `shuffle-rdg` zu spät plaziert wird, an den Kapitelanfang holt. Schließlich wird noch ein kleiner Trick ausgeführt, der für das siebte Kapitel nötig ist, um die Überschrift "? ? ?" (*p274 ℓ1*) für die `layout`-Routine als Wort kenntlich zu machen.

13.8. *empty-tags*

Das Programm `empty-tags` ist einfach und kurz; es fügt alle SGML-Elemente mit dem *content model* `EMPTY` ein, für die also kein End-Tag plaziert werden muß. Das funktioniert größtenteils über eine einfache Ersetzungstabelle, aufgrund der bestimmte Kontrollsequenzen durch das entsprechende `s`-Token ersetzt werden.

Eine Sonderbehandlung ist lediglich für das "⌷"-Symbol notwendig, das, wenn es im normalen Text auftritt, als Lemma für eine Fußnote gedacht ist. Im Gegensatz zu anderen Lemmata folgt ihm aber kein "°"-Symbol als Fußnotenverweis. Die Kontrollsequenz dafür wird in **`empty-tags`** eingeführt, damit die Fußnote in **`position-notes`** korrekt verarbeitet werden kann.

13.9. *Die Vergabe der IDs*

Da die folgende Routine **`position-notes`** die erste ist, in der SGML-IDs vergeben werden, möchte ich kurz die Überlegungen zum ID-Namensraum zusammenfassen.

SGML-IDs bestehen nach der TEI *Concrete Syntax* aus den Zeichen '`A`'–'`Z`', '`0`'–'`9`' sowie '.' und '-', wobei das erste Zeichen ein Buchstabe sein muß; zwischen Groß- und Kleinschreibung wird nicht unterschieden. Im Abschnitt 6.9.2 des *Green Book* (S. 186) wird in anderem Zusammenhang auf systematische Methoden zur Vergabe von IDs eingegangen, die sich aber für meinen Fall nicht eignen.

In meinem Verfahren beginnen alle automatisch vergebenen IDs mit den Buchstaben '**`id`**', gefolgt von einer zweistelligen Nummer, die für das erzeugende Programm spezifisch ist; dadurch haben unterschiedliche Programme getrennte Namensräume. Der anschließende Teil kann dann vom jeweiligen Programm frei gestaltet werden. Es hat sich herausgestellt, daß nur zwei Programme überhaupt IDs vergeben, **`position-notes`** im Haupttext und **`parse-tn`** für die *Textual Notes*.

Im Haupttext wird **`position-notes`** für jedes Kapitel einzeln aufgerufen, deswegen muß die Kapitelnummer ebenfalls Bestandteil der ID werden. Das ID-Format für die von **`position-notes`** vergebenen **`<anchor>`**-IDs ist deswegen '`id01c`*kapitelnr*`n`*seriennr*'.

Für die *Textual Notes* habe ich zwischen den *general notes*, von denen es genau eine pro Kapitel gibt, und den spezifischen Anmerkungen unterschieden. Erstere haben das Format '**`id03gen`***kapitelnr*', letztere '**`id03tn`***seriennr*'.

13.10. *position-notes*

Die Verarbeitung der in den Textstrom eingebetteten Anmerkungen (Fußnoten und historische Kollation) geschieht in zwei Stufen. In **`position-notes`** werden die Lemmata mit **`<anchor>`**s markiert und die Anmerkungen an ihre endgültigen Plätze verschoben. In **`parse-notes`** und **`parse-hico`** wird dann der Inhalt der Anmerkungen konvertiert.

In den TUSTEP-Dateien sind die Lemmata durch dem Wort nachgestellte Kontrollsequenzen markiert; bei Anmerkungen zu mehr als einem Wort folgt der Text der Anmerkung dem ersten markierten Wort. Für die SGML-Dateien erschien es mir sinnvoller, den tatsächlichen Anfang des Lemmas durch einen dem ersten Wort vorangestellten **`<anchor>`** zu markieren, anstatt sich auf einen impliziten Wort-Begriff zu verlassen. So wird die genaue Ausdehnung des Lemmas formal erfaßt. Der Text der Anmerkung sollte, der einfacheren Verarbeitung zuliebe, möglichst unmittelbar nach dem Ende des Lemmas stehen; die Anmerkung mitten ins Lemma zu stellen, schien mir unsinnig. Es hätte die Lesbarkeit des unverarbeiteten SGML-Textes sicher verbessert, wenn die Anmerkungen getrennt gesammelt wären. Ich habe mich im Endeffekt aber dagegen entschieden, da es leichter ist, den räumlichen Zusammenhang bei Bedarf zu zerreissen, als ihn nachträglich wieder herzustellen.

Die Aufgabe von **`position-notes`** wird durch die Möglichkeit verschachtelter Anmerkungen erschwert. Es sind bis zu vier Ebenen von ineinanderligenden Fußnoten und bis zu drei Ebenen ineinanderliegender Einträge der historischen Kollation möglich, die getrennt verwaltet werden müssen. Im Gegensatz zu späteren Programmen, die bei Bedarf den Tokenstrom zurückspulen, wird hier auch nur streng vorwärts gelesen, so daß relativ viele Zustandsdaten gehalten werden.

$\star$

Das erste Problem ist die Plazierung der `<anchor>`s. Um den `<anchor>` vor ein markiertes Wort legen zu können, wird der gelesene Tokenstrom zwischen zwei Blanks gepuffert; wird anschließend ein Markierungs-Startsymbol gefunden, kann der `<anchor>` unmittelbar nach dem vorhergehenden Blank ausgegeben werden, gefolgt vom markierten Wort.

Es kommt relativ häufig vor, daß sich historische Kollation und Emendation auf dieselbe Textstelle beziehen. Um den SGML-Text nicht unnötig mit `<anchor>`s zu füllen, können sich darum mehrere Anmerkungen einen geeignet plazierten `<anchor>` teilen. Grundsätzlich würde es auch genügen, anstelle des `<anchor>` irgendein Element, das sich zufällig an der richtigen Stelle befindet, mit einer ID zu versehen. Das scheint mir aber eine unnötige Erschwernis der Konvertierung zu sein. Die Möglichkeit, `<anchor>`s gemeinsam zu benutzen, ist auch der Hauptgrund, warum die ansonsten unabhängig voneinander arbeitenden Routinen zur Positionierung von Emendationen und historischer Kollation in einem `perl`-Programm zusammengefaßt wurden.

Wenn ein Start-`<anchor>`s plaziert wird, wird die vergebene ID zwischengespeichert; für Anmerkungen zu mehr als einem Wort auch der Text der Anmerkung. Sobald der dazugehörige End-`<anchor>` plaziert wird, wird der Fußnotentext eingefügt und mit den SGML-Start- und End-Tags umgeben, damit die IDs der `<anchor>`s eingetragen werden können.

Der Ablauf ist nicht eigentlich kompliziert, wird aber durch die verschiedenen Sonderfälle und die verschachtelbaren Anmerkungen etwas undurchsichtig. Das knapp vierhundert Zeilen lange Programm ist von keinem prinzipiellen Interesse, ich habe es deshalb nicht abgedruckt.

13.11. *Die 'readlib'-Funktionen*

Die Programme der Konvertierungspipeline wurden im Großen und Ganzen in der Reihenfolge geschrieben, in der sie arbeiten und hier besprochen werden; bei `parse-tn` verläuft dabei in gewisser Hinsicht eine Generationengrenze. Bis einschließlich `position-notes` bestand die Arbeitsweise meiner Programme darin, streng sequentiell über die Eingabe zu laufen und dabei wenn nötig durch verschiedene Puffer und Zustandsflags Information zu transportieren. Schon in `position-notes` ist dieses Verfahren mühsam und unanschaulich. Die sich anschließenden Routinen `parse-tn`, `parse-notes`, `parse-hico` und `layout` sollen alle nicht nur vorhandene Kontrollsequenzen in SGML-Tags verwandeln, sondern anhand von eher spärlichen Indizien umfassende Strukturen erzeugen. Dafür war eine andere Sicht auf die Daten nötig.

Diese vier Programme haben zweierlei gemeinsam: sie sind als *recursive descent* Parser geschrieben und sie verwenden für den Zugriff auf den Eingabestrom die in `readlib` zusammengestellten Funktionen, wobei `parse-tn` allerdings von ihren Möglichkeiten praktisch keinen Gebrauch macht.

Die `readlib`-Bibliothek ist nur 50 Zeilen lang und enthält fünf `perl`-Funktionen (sie ist im Anhang abgedruckt). Sie stellt Puffer für Ein- und Ausgabe bereit und ermöglicht es dadurch, die Ausgabe wieder zurückzuholen und in die Eingabe zurückzuschreiben. Dadurch liegt die Eingabe auf einem 'Schneidetisch', auf dem sie während der Bearbeitung hin- und hergefahren werden kann.

Die Funktionen, mit denen das realisiert wird, sind:

- `read`, zum Einlesen des nächsten Tokens;
- `output`, um ein oder mehrere Token auszugeben;
- `readback`, um das letzte ausgegebene Token zurückzuholen;
- `unread`, um ein oder mehrere Token zurück in die Eingabe zu schreiben;
- `flush`, um den Ausgabepuffer endgültig auszugeben; typischerweise am Ende eines Absatzes oder einer Anmerkung.

Wenn mit **read** das Ende der Eingabe erreicht wird, wird der Ausgabepuffer mit **flush** geleert und das Programm beendet. Davor wird noch ein **rl_exitstring** ausgegeben, in dem Programme die End-Tags offener Strukturen ablegen können. Dieser Mechanismus ist mehr eine hinzugefügte Bastelei, ebenso wie die Zeilennumerierung des Ausgabepuffers, die in **outline** mitgeführt wird und das Zurückspulen zu einem bestimmten Punkt erlauben soll. Beides sind nützliche Funktionen, die es verdient hätten, vernünftig implementiert zu werden. Weil sie aber 'nur mal schnell' nötig waren, sind sie improvisiert geblieben.

Der flexible Umgang mit Ein- und Ausgabe war eine wichtige Voraussetzung für die Parser in den genannten vier Routinen, die oft weit in die Eingabe hineinlesen müssen, bevor sie entscheiden können, welches SGML-Element sie erzeugen sollen. Diese Technik der Programmierung, bei der viele Daten hin- und herbewegt werden, ist verführerisch und kostet Rechenzeit, aber sie erleichtert das Programmieren, insbesondere wenn man sich nur langsam und durch Versuch und Irrtum der endgültigen 'Grammatik' für komplexe Strukturen annähert.

Das Stichwort 'Grammatik' legt die Frage nahe, ob für diese Routinen nicht die Verwendung von **yacc** sinnvoll gewesen wäre. Ich habe zwischendurch darüber nachgedacht, bin mir aber nicht sicher – ich hatte mich zu diesem Zeitpunkt in die Arbeit mit **perl** eingelebt, und die C-Programmierung mit regulären Ausdrücken und dynamischer Speicherverwaltung hätte zusätzlichen Aufwand bedeutet, der für diesen begrenzten Zweck wohl übertrieben gewesen wäre.

13.12. *parse-tn*

Parallel zu der langen Pipeline, in der der Haupttext des *Ulysses* von der Tokenisierung an verarbeitet wird, steht die Umwandlung der *Textual Notes* direkt von der ersten Tokenform zum beinahe ausgabefähigen Ergebnis. Lediglich die gemeinsam genutzte Routine **charcodes** und der Export in echtes SGML durch **toks2sgml** schließen sich an **parse-tn** noch an.

Die Routine muß beim Parsen feststellen, ob sie die Überschriften und die drei *general notes* zu Kapitelbeginn oder eine gewöhnliche Anmerkung zu einer Textstelle vor sich hat. Da das bereits am ersten Token unterschieden werden kann, kann der Text ohne Probleme sequentiell durchlaufen werden. Die Methode des *recursive descent* erlaubt es, die Strukturen auf eine sehr anschauliche und übersichtliche Weise zu verarbeiten. Das Hauptprogramm von **parse-tn** lautet zum Beispiel:

```
&nextline;
&do_title;
while (1) { # &read quits on EOF.
    &try_general
    || &try_tn
    || die "parse-tn: General parse error: $word Stopped";
}
```

In **parse-tn** wird für jede Anmerkung eine ID vergeben, die in den Verweisen im Text benötigt wird; dafür muß mit der zuständigen Konvertierungsroutine für den Haupttext, **parse-notes**, koordiniert werden, welche Anmerkung welche ID trägt. Die *general notes* stellen kein Problem dar, weil es von ihnen genau eine pro Kapitel gibt; ihre IDs konnten also so geformt werden, daß sie in **parse-notes** unabhängig reproduziert werden konnten (vergleiche den Abschnitt über ID-Vergabe oben).

Für die übrigen Anmerkungen wird in der Hilfsdatei **tn-idlog** eine Zuordnungstabelle von Seiten-Zeilen-Angaben zu IDs abgelegt, die von **parse-notes** gelesen wird. Es werden Anfangs- und Endzeile der Textstelle protokolliert, selbst wenn das für einzeilige Anmerkungen dieselbe Angabe ist:

...

```
156.1 id03tn15
156.2 id03tn15
164.13 id03tn16
164.13 id03tn16
166.36 id03tn17
166.37 id03tn17
 ...
```

Die Seiten-Zeilen-Referenzen stehen am Anfang jeder Anmerkung nach den zugehörigen Lemmata, während diese Angaben verarbeitet werden, werden die Tabelleneinträge erzeugt. Für eventuelle Seiten-Zeilen-Referenzen innerhalb des Textes, die auch als `<ref>` codiert werden, werden keine Einträge erzeugt. In `parse-notes` wird die 'laufende' Seite und Zeile mitgeführt; stößt `parse-tn` auf den Hinweis " → TN" in einer Anmerkung, kann so die ID der betreffenden *textual note* ermittelt werden.

13.13. *parse-notes*

Nachdem die Fußnoten durch `position-notes` an die endgültigen Stellen plaziert und die `<emen>`-Tags mit den IDs der das Lemma umgebenden `<anchor>`s eingefügt worden sind, und nachdem die von `parse-tn` erstellte ID-Tabelle für die *Textual Notes* bereit steht, kann der Inhalt der Fußnoten verarbeitet werden.

Fußnoten bestehen aus den Einheiten `<lem>`, `<rdg>`, `<witness>` und `<comm>`. Versuche, die durch das `<comm>`-Element mögliche Abstufung von streng formalen bis zu völlig freien Fußnoten zu kategorisieren, hatten keinen Erfolg. Die einzige Unterscheidung ist, daß Fußnoten, die kein `<lem>`-Element aufweisen, mit `type=note` gekennzeichnet werden. Es hat sich herausgestellt, daß das nur die Hinweise auf die *general notes* sind.

Die Unterscheidung, ob ein `<comm>` als Teil eines `<rdg>` oder `<wit>` auftritt oder stattdessen, wird nach einfachen formalen Kriterien durchgeführt; ob sie immer der tatsächlichen Verwendung des Kommentars entspricht, sei dahingestellt. Es erschien mir aber sinnvoll, die Unterscheidung zumindest zu versuchen, da Kommentare wie "*UNDERLINED*" wirklich stellvertretend für die Wiedergabe einer Lesart eingesetzt werden, während solche wie "*CF 1370.11*" nichts mit einer Lesart zu tun haben.

Diese Unterscheidungen verlangen ein gewisses Maß an Versuch und Irrtum; deswegen nutzen sie die Möglichkeiten der `readlib`-Funktionen reichlich. Die unten abgedruckte Funktion `try_lemma` versucht beispielsweise, den Anfang der Fußnote als `<lem>` zu interpretieren. Dazu übergeht sie zunächst eventuelle Leerzeichen, setzt dann den Start-Tag für `<lem>`, geht weiter durch den Text und konvertiert eventuelle Anweisungen für Schriftartwechsel. Findet sie eine geschlossene eckige Klammer, entfernt sie sie aus dem Tokenstrom, fährt über die letzten Leerzeichen zurück, setzt den End-Tag und gibt 1 zurück. Findet sie stattdessen die Kontrollsequenz für das Fußnotenende, dann gab es kein Lemma in dieser Fußnote, und sie kehrt zum Anfang zurück, löscht den `<lem>`-Start-Tag und gibt 0 zurück. Im ersten Fall wird das aufrufende Programm als nächstes versuchen, eine `<witness>`-Angabe ab der momentanen Position zu finden, im zweiten Fall handelt es sich um eine formlose Fußnote, in der Kommentare erkannt und Schriftarten durch `<hi>` codiert werden.

```
sub try_lemma {
    # Copy initial blank(s)
    while ($type eq 'b') {
        &output ($line);
        &nextline;
    }
    # Start tag
    &output ($o_lemma);
    # Copy text until '%]' or end of note.
```

```
    while (($word ne '%]') && ($word ne '~&')) {
        &try_caps
        || &try_italic
        || &try_slant
        || &try_comm
        || do {
            &output ($line);
            &nextline;
        };
    }
    if ($word eq '~&') {
        # Read back the whole lemma content, swallow the start tag.
        while ($line ne $o_lemma) {
            &unread ($line);
            &readback;
        }
        &nextline;              # Swallow '<lem>'.
        0;                      # There was no lemma.
    } else {
        # End Tag
        &output ($c_lemma);
        &nextline;              # Swallow '%]'
        1;                      # There was a lemma.
    }
}
```

13.14. *parse-hico*

Die Einträge der historischen Kollation unterscheiden sich hauptsächlich durch ihre festeren formalen Strukturen von den Emendationen; parallel dazu ist auch das Programm `parse-hico` seiner Schwester `parse-notes` recht ähnlich geraten, muß aber weniger erraten und kann dafür detaillierter parsen. Aus diesem Grund habe ich `parse-hico` als den interessantesten Vertreter dieses Programmtyps im Anhang abgedruckt.

Die Angaben zu den Editionen, die eine bestimmte Lesart enthalten, werden ebenso wie Abkürzungen und Seiten-Zeilen-Referenzen bereits von `tokenize` erkannt und in Spezialtoken zusammengefaßt, so daß deren Verarbeitung hier leicht fällt.

Die deutlichste Schwäche dieser Routinen ist, daß sie jede für sich die Umwandlung der Schriftartanweisungen vornehmen, obwohl diese, von den schräggestellten Kapitälchen für Kommentare abgesehen, keine strukturelle Bedeutung haben und von einem eigenen Programm pauschal hätten konvertiert werden können. Das hätte die Parser schlanker gemacht und Fehlerquellen vermieden.

13.15. *layout*

Das Programm `layout` verarbeitet Textstrukturen wie Prosaabsätze, Lyrik, Prosadisplays, Dramenform und die übrigen besonderen Kapitelformen. Es ist mit 680 Zeilen das mit Abstand längste Programm der Konvertierungspipeline, weswegen es sich weder zum Abdruck noch zur detaillierten Besprechung sehr gut eignet. Da es aber diejenigen Aspekte der Konvertierung behandelt, die auch für andere Projekte interessant sind, möchte ich doch auf einige der Überlegungen und Verfahren eingehen.

Für die Verarbeitung der Kontrollsequenzen für Schriftartenwechsel habe ich hier (endlich) eine einigermaßen gute und effiziente Lösung gefunden. Die Verarbeitung dieser Kontrollsequenzen ist in die Ausgaberoutine `normalline` eingebunden, so daß sie nicht an allen möglichen Stellen explizit aufgerufen werden muß. Es wird mit Hilfe von Zustandsflags überprüft, ob die Kontrollsequenzen korrekt beendet werden. Welche Schriftartwechsel wie codiert werden ist aber kontextabhängig, deswegen wird diese Routine, zum Teil mit davorgeschalteten Filtern, von den Routinen für die einzelnen Absatzformen aufgerufen.

Beispielsweise wird in den stets schräggestellten Bühnenanweisungen im fünfzehnten Kapitel Hervorhebung durch normale recte ausgedrückt. Die Routinen für die Bühnenanweisungen sorgen also zunächst dafür, daß die Hervorhebung `<hi rend=sl>` unterdrückt wird, da sie bereits im `rend`-Attribut von `<stage>` ausgedrückt ist; wenn sie auf die Kontrollsequenz für Hervorhebungsende stoßen, untersuchen sie, ob dieser Wechsel nur technische Gründe hat und vor dem nächsten Wort wieder aufgehoben wird. Falls nicht, wird die Routine `do_hilite_rm` aufgerufen, die das entsprechende Textstück korrekt als `<hi rend=rm>` auszeichnet. Diese Funktion wiederum ruft für die Ausgabe der normalen Textzeilen `normalline` auf, um für den unwahrscheinlichen Fall einer weiteren Hervorhebung innerhalb der Passage, etwa durch Kapitälchen, vorgesorgt zu haben. Außerdem muß sie darauf achten, die `<hi>`-Tags richtig zu plazieren, damit nicht unsinnigerweise die umgebenden Leerzeichen mit in das `<hi>`-Element aufgenommen werden.

Für die Codierung von Lyrik wurde auf ähnliche Weise die allgegenwärtige Auszeichnung durch Kursivschrift in das `rend`-Attribut der Struktur übernommen und das entsprechende `<hi>`-Element in den Zeilen unterdrückt.

Bei der Erstellung der `layout`-Routine hat sich überraschenderweise herausgestellt, daß gewöhnliche Prosaabsätze ohne Einzug der ersten Zeile in den Quelldateien nicht von solchen mit Einzug unterschieden werden. Ich kann mir das nur dadurch erklären, das die mir zur Verfügung stehenden TUSTEP-Dateien nicht unmittelbar für den Druck weiterverwendet wurden und irgendwann einmal auf diese Weise irreversibel vereinfacht worden sind. Dieser Defekt war leider nicht mehr zu beheben, er wirkt sich aber nicht allzusehr aus, weil solche Absätze fast nur systematisch vorkommen, wo sie durch den Kontext als eingezogen oder nicht eingezogen bestimmt werden können.

Ich hatte mich nach kurzem Zögern entschlossen, die Absatzenden, die in TUSTEP nicht extra markiert sind, explizit durch End-Tags zu codieren; zwar kann der SGML-Parser oder ein Normalisierungsprogramm diese Arbeit auch übernehmen, es ist aber keineswegs garantiert, daß die Endtags dann auch da stehen, wo man sie erwartet hätte. Gerade für die Konvertierung des *Ulysses*, wo sich die genauen Definitionen vieler Strukturen erst langsam durch Versuch und Irrtum herauskristallisiert haben, hat es sich für mich sehr gelohnt, meine Erwartungen und Vorstellungen von diesen Strukturen explizit zu programmieren und so zu überprüfen. Die Markierung erfolgt praktisch so, daß eine Absatzfunktion, die auf den Startcode für die nächste Absatzstruktur stößt, bis zum letzten druckbaren Wort 'zurückspult',[29] anschließend bis zum nächsten Leerzeichen vorspult, und davor `</p>` oder den entsprechenden End-Tag in den Tokenstrom einfügt. Die `readlib`-Funktionen konnten in solchen Fällen ihre Vorteile voll zur Geltung bringen, allerdings auch ihren Nachteil durch die verlängerte Rechenzeit, die zu diesem Zeitpunkt bereits erschreckende Ausmaße annahm.[30]

Die Sonderfälle der Formatierung haben mich hier ebenfalls beschäftigt, da ich das Dilemma zwischen automatischer Konvertierbarkeit und Handarbeit noch ungemildert vor mir sah. Die grobe Vorcodierung der Bilanz und des Buchkatalogs im siebzehnten Kapitel haben mir die spätere manuelle Codierung aber tatsächlich erleichtert.

---

[29] Dies war wieder eine Stelle, bei der sich die eingeschränkte Definition von 'druckbarem Wort', die den Tokentypen zugrunde liegt, gerächt hat; vergleiche S. 80.

[30] Nach den ersten Stufen der Pipeline reichte die Zeit für ein Lied auf der Gitarre; am Ende konnte ich während eines vollständigen Testlaufs schon ein Abendessen inklusive Abwasch erledigen.

13.16. *starreg*

Auf dieser Stufe der Pipeline ist, von der Ersetzung der Sonderzeichen durch Entity References abgesehen, die Codierung des Lesetextes und der Anmerkungen abgeschlossen. Was bleibt ist 'nur' noch die Codierung der synoptischen Darstellung der Textgeschichte, die im Wesentlichen durch die Programme `overlay` und `convert-app` erledigt wird. Davor werden aber noch drei relativ harmlose Sonderfälle bearbeitet, die alle das gleiche Druckbild und ähnliche Struktur haben und deswegen von einer Routine, `starreg`, erledigt werden.

Es geht um die Kontrollsequenzen '`*`*zeichen*' und '`*<`*zeichen*`>`', die im Text von Kapitel 15 und 18 eine Normalisierung des markierten Zeichens anzeigen, in der historischen Kollation dieser Kapitel hingegen die Wiedergabe der nicht normalisierten Schreibweise bezeichnen. Die entsprechenden SGML-Elemente sind `<reg orig=..>` und `<orig reg=..>`.

Im achtzehnten Kapitel betrifft die Normalisierung Satzzeichen, die mit '`*`zeichen' als gestrichen gekennzeichnet werden und direkt in das `orig`-Attribut übernommen werden können. Im fünfzehnten Kapitel werden die Bühnenanweisungen normalisiert und dazu Satzzeichen sowohl gestrichen als auch eingefügt und außerdem von Groß- zu Kleinbuchstaben gewechselt und umgekehrt.

Die kleine Besonderheit dieses Programms ist, daß in den SGML-Elementen `<reg>` und `<orig>` die jeweils andere Schreibweise in einem Attribut angegeben wird; das Programm bestimmt sie in der Funktion `should_have_been` automatisch.

Die Entscheidung, welches Element gemeint ist, also zwischen historischer Kollation und Text, wird dadurch erleichtert, daß der Stern in der historischen Kollation nicht durch '`*`' sondern durch '`&.st`' dargestellt wird; es ist deswegen nicht nötig, die Start- und Endcodes mitzuverfolgen.

13.17. *overlay*

Die nächsten beiden Programme der Pipeline, `overlay` und `convert-app`, bilden die Kontrollsequenzen, mit denen die synoptische Darstellung der Textgeschichte codiert ist, in die SGML-Strukturen `<overlay>` und `<syn>` mit ihren Unterelementen ab.

Durch die unterschiedlichen Level der Bearbeitungen und die zusätzlich assoziierten Symbole ergeben sich eine enorme Zahl von Kontrollsequenzen. Für eine sinnvolle Verarbeitung müssen die Level und die Extrasymbole nach Möglichkeit als Parameter aufgefaßt werden. Das reduziert die Zahl der zu unterscheidenden Kontrollsequenzen, erfordert aber zusätzlichen organisatorischen Aufwand bei der Erzeugung der SGML-Tags.

Charakteristisch ist für beide Programme, daß zu Beginn eines Elements festgelegt wird, wie das Ende und oft auch noch ein Mittelstück aussehen muß, daß dazwischen aber rekursiv geschachtelt weitere solche Elemente liegen können. Insbesondere die zweigeteilten `<syn>`-Strukturen, wo nach nach dem `<old>` ein `<new>` mit zu setzenden Attributen folgt, erfordern einen umfangreichen Transfer von Statusinformation vom Anfangs- zum Mittelteil.

Hierbei muß man mit kleinen Tricks arbeiten, um die nötigen komplexeren Datenstrukturen mit `perl` zu realisieren. In `overlay` genügte es aber schon, ein $6n$ langes Array als $2 \times 3 \times n$-Array zu interpretieren, um die Übersetzungstabelle einrichten zu können. Anstatt Paare von Strings auf einem Stapel abzulegen, wurden außerdem zwei parallele Stapel eingerichtet, die synchron auf- und abgebaut werden.

Für jede (parametrisierte) Kontrollsequenz sind in dieser Tabelle folgende sechs Felder ausgefüllt:

1. Ein regulärer Ausdruck, der die gesuchte Kontrollsequenz beschreibt; der String, der dem ersten geklammerten Teilausdruck entspricht, kann im dritten und fünften Feld verwendet werden (die Wirkungsweise von Klammern in regulären Ausdrücken ist im `perl`-Handbuch auf S. 104 beschrieben).
2. Ein String, der das oder die einzusetzenden SGML-Token enthält. Ein Einsetzen von Parametern ist in `overlay` nicht vorgesehen, weil die Levelinformation

scheinbar implizit durch die Schachtelungstiefe gegeben war und einfache Ersetzung damit ausreichend schien. Das das ein Irrtum war, hat sich zu spät herausgestellt (vergleiche S. 32).

3. Ein regulärer Ausdruck, der den ‘Trigger’ für die zweite Ersetzung beschreibt; bei einfachen Streichungen oder Einfügungen repräsentiert er das Ende der Revision, bei Ersetzungen das Ende des alten und den Anfang des neuen Textes. Nach diesem Ausdruck wird nur gesucht, wenn das dazugehörige Anfangselement offen ist, mehr dazu unten. Im regulären Ausdruck kann ‘`%s`’ erscheinen und wird durch den im ersten Schritt abgespeicherten String ersetzt.
4. Die SGML-Token, die an der Stelle des ‘Triggers’ eingesetzt werden sollen.
5. Ein regulärer Ausdruck für die dritte Ersetzung, falls eine solche nötig ist (nur für zweiteilige Elemente, also echte Ersetzungen).
6. Die dazugehörigen SGML-Token.

Ich gebe zur besseren Anschaulichkeit die sechs Felder der Übersetzungstabelle für eine gewöhnliche Überarbeitung, wie sie beispielsweise verwendet würde für die Übersetzung von

```
Memories |1<crowded> beset|1{ his brooding brain.
```

zu

```
Memories <overlay><del>crowded</del><add> beset</add> his brooding
brain.
```

Der zuständige Eintrag der Übersetzungstabelle, wie gesagt, lautet:[31]

```
"\\|([1-3])\\<",
"s <overlay>\ns <del>",
"\\>",
"s </del>\ns <add>",
"\\|%s\\{",
"s </add>\ns </overlay>",
```

Die Token, die den Beginn einer Revision markieren, kennzeichnen sie eindeutig. Es ist aber wichtig, daß die Fortsetzung dieser Struktur, also die ‘Trigger’, nur im korrekten Kontext gesucht werden, weil sonst manche Verschachtelungen zu Fehlern beim Parsen führen, wie in folgendem konstruierten und albernen Beispiel: “^⟨Sehr ge⟨teerter⟩ehrter⟩ Lieber^ Herr Bundeskanzler!” In diesem Beispiel liegt eine Streichung *currente calamo* mitten in einem später als Ganzes gestrichenen Textstück. Nach Beginn der inneren Streichung schließt das ‘`>`’-Symbol das `<del type=cc>`-Element, anschließend ist ein ‘`>`’-Symbol wieder, wie vor Beginn der inneren Streichung, das Zeichen für das Ende der ersetzten und den Beginn der neuen Lesart.

Diese Rekursion wird im Programm `overlay` durch eine Stapelstruktur mit erwarteten Triggerelementen und einem zweiten Stapel mit den korrespondierenden Ersetzungsstrings realisiert. Beginnt innerhalb einer Revision eine weitere, verdrängen der aktuelle Trigger und Ersetzungsstring die bisher aktuellen auf dem Stapel nach unten; sobald die entsprechende Kontrollsequenz gefunden und ersetzt ist, werden die vorherigen Werte wieder aktuell. Strukturen mit zwei Triggern legen den zweiten Trigger und Ersetzungsstring ebenfalls auf den Stapel, die ersten sind sofort gültig.

Das Programm vergleicht sequentiell jedes Kontrollsequenz-Token zuerst mit dem ersten regulären Ausdruck für jeden Eintrag in der Tabelle, um zu prüfen, ob eine neue Overlaystruktur beginnt. Falls ja, wird die Anfangsersetzung vorgenommen und der Triggerzustand wie eben beschrieben verändert; falls nein, wird mit dem aktuellen Trigger verglichen.

[31] Da die regulären Ausdrücke in Strings abgelegt werden, muß jeder Backslash doppelt geschrieben werden, damit er als einfacher Backslash im regulären Ausdruck ankommt.

Falls der paßt, ist er abgearbeitet, der Ersetzungsstring wird eingefügt und der Stapel- und Triggerzustand aktualisiert.

Das Programm `overlay` ist im Anhang abgedruckt. Das eigentliche Programm ist nur circa 50 Zeilen lang; die restlichen 100 Zeilen verbraucht die Übersetzungstabelle.

13.18. *convert-app*

Das Programm `convert-app` ist das letzte 'große' Konvertierungsprogramm der Pipeline, danach kommen nur noch zwei Kleinigkeiten und die Ausgabeformatierung. Es ist ähnlich aufgebaut wie `overlay`, es muß aber etwas mehr und etwas kompliziertere Probleme lösen. Dem entsprechend ist auch die die Übersetzung tragende Datenstruktur etwas komplizierter. Ich halte ihre Diskussion trotzdem für wesentlich, weil sie zeigt, wie auch mit den Möglichkeiten von `perl` komplexere Strukturen und Abläufe gestaltet werden können.

Die Übersetzungstabelle ist, notgedrungen, ein einfaches assoziatives Array; die Substruktur entsteht durch eine Feldunterteilung mit Doppelpunkten und durch Verwendung von Platzhaltern '`x`' und '`y`'. Um deutlich zu machen, wovon die Rede ist, gebe ich zunächst einige Beispielzeilen aus dieser Tabelle:

```
%general_info = (
'[x',           'delete:$y:$y:]::extant:certain:author',
'[x(**)',       'delete:*$y:$y*:]::extant:content-supplied:editor',
'x()',          'insert:($y):($y):()$x{::lost:certain:author',
'x()(**)[',
     'replace:*($y):($y)*:]:(**)()$x{:lost:content-supplied:editor',
'x()[',         'replace:($y):($y):]:()$x{:lost:certain:author',
'p:jx',         'join:$y:$y:::extant:certain:author',
'p:sx()',       'split:($y):($y):::lost:certain:author',
'x-|S',         'insert:$y-S:S-$y:|S{-$x{::extant:certain:scribe',
'x-|S[',        'replace:$y-S:S-$y:]:|S{-$x{:extant:certain:scribe',
);
```

In dieser Struktur, die auf den ersten Blick etwas erschlagend wirkt, steht der erste String für die TUSTEP-Kontrollsequenz, wobei aber der Code für den Level durch das '`x`' repräsentiert wird. Dieser Code ist entweder ein senkrechter Strich gefolgt von einem Buchstaben, oder eine Zahl (ohne Strich). Der Platzhalter '`y`' entspricht dem Levelcode in '`x`', ist aber in jedem Fall vom Strich befreit.

Der zweite String der Übersetzungstabelle ist durch Doppelpunkte in acht Felder unterteilt. Das erste Feld gibt die durchzuführende Aktion an, die restlichen Felder spezifizieren Attributwerte und Ersetzungstrigger. Die einzusetzenden SGML-Token sind, von den Attributen abgesehen, durch die Aktion bereits eindeutig definiert. Den Variablen '`x`' und '`y`' ist hier ein Dollarzeichen vorangestellt, so daß keine versehentlichen Ersetzungen, beispielsweise im Wort '`extant`', vorkommen.

Das zweite und dritte Feld geben die Klartextschreibweise der Levelbezeichnung an, wie sie im Attribut `key` festgehalten wird (siehe S. 64). Das vierte und fünfte Feld sind, analog zur Konstruktion in `overlay`, die Triggerstrings für die Folgeersetzungen. Die restlichen drei Felder sind die Attributwerte für `entrydoc`, `status` und `hand`.

Diese Ersetzungsparameter werden wieder mit einer Stapelstruktur wie in `overlay` verwaltet. Den Schlüsselwörtern im ersten Feld entsprechen Routinen, die die Stapel bedienen und die entsprechenden SGML-Tags erzeugen.

Hinzu kommt aber noch die wesentlichsten Parameter der `<syn>`-Struktur: die Spezifikation des Levels der Revision in `<syn>` und der jeweils gültigen Level in `<old>` und `<new>`. Diese Aufgabe wird durch einen eigenen Mechanismus bewältigt. Das vordefinierte Array `chaplevels` enthält in achtzehn Einträgen die für jedes Kapitel vorgesehenen Level-IDs, die so durch Indizierung mit der Kapitelnummer erreicht werden können. Der Eintrag für das erste Kapitel zum Beispiel lautet:

```
"lvl1.ros lvl1.a lvl1.b lvl1.c lvl1.1 lvl1.2 lvl1.3 lvl1.4 lvl1.5
    lvl1.6",
```

Einer der in der Stapelstruktur verwalteten Stapel enthält für jede Textposition die momentan gültigen Level-IDs. Wenn eine neue Revision beginnt, werden in der Funktion `find_levels` anhand des Level-Parameters '`y`' und der aktuell gültigen Level die für den ersten und den zweiten Teil einer Revision (also für `<old>` und `<new>`) gültigen IDs bestimmt, indem der String, der sämtliche aktuellen IDs enthält, an der richtigen Stelle geteilt wird.

⋆

Zusätzliche Maßnahmen werden durch die Sonderformen "⌜R...R⌝" und "⌜V...V⌝" nötig. Diese Sonderfälle wurden durch Zugehörigkeit ausschließlich zu `lvl`*kapitel*`.ros` und, je nachdem, `lvl.gab` codiert; in `find_level` werden diese beiden Level darum gesondert nachbehandelt; `lvl.gab` wird ohnehin erst nachträglich für die dem Lesetext entsprechende Verzweigung angehängt.

Die Ausführungsprozeduren zu den Schlüsselwörtern im ersten Feld der Übersetzungstabelle weisen immer noch einige Redundanz auf. Es wäre vielleicht möglich, bessere Verfahren zu finden, um die etwas sperrige Übersetzung des Schichtenmodells in den Griff zu bekommen. Ich halte es auch durchaus für möglich, daß die Mühseligkeiten und Sonderfälle der Übersetzung ein Indiz für eine gewisse Schwammigkeit der SGML-Struktur des Schichtenmodells sind. Sowohl die SGML-Struktur wie die Konvertierungsroutine sind jedenfalls das Beste, was ich unter den gegebenen Umständen zustandegebracht habe.

Als leichtes Anhängsel der schwergängigen Textstufenkonvertierung ist noch die Übersetzung der Symbole für relevante Zeilen- und Seitenwechsel in diese Routine geraten. Es handelt sich um eine einfache wenn-dann-Ersetzung ohne Neuigkeitswert.

13.19. *charcodes*

Im Programm `charcodes` werden die Kontrollsequenzen, die für einzelne Zeichen des Textes stehen und die 8-Bit-Sonderzeichen der TUSTEP-Dateien eins zu eins übersetzt, größtenteils in Entity References. Da in Einzelfällen Unterschiede zwischen den Textgruppen gemacht werden müssen, gibt es neben einer allgemeinen Ersetzungstabelle noch spezifische Tabellen für die Textgruppen '`n`' und '`t`'. Wenn sie einen passenden Eintrag enthalten, haben sie Vorrang vor der allgemeinen Tabelle.

Alle Zeichen die im SGML-Dokument erscheinen sollen, müssen mit dem Tokentyp '`e`' codiert werden, auch die gewöhnlichen Interpunktionszeichen, die ansonsten unverändert übernommen werden.

13.20. *boundary*

Wie auf Seite 63 besprochen, habe ich zwei mögliche Ansätze zur Bewältigung struktureller Revisionen erwogen, und letztlich beide umgesetzt. Die Methode, Strukturelemente innerhalb von Revisionen durch leere `<boundary>`-Elemente zu markieren, war der erste Ansatz; für die `<syn>`-Tags bin ich davon wieder abgekommen, für `<overlay>` habe ich ihn beibehalten.

Das Programm `boundary` führt die dafür erforderliche Verflachung der Struktur, also die Umwandlung von Strukturelementen in `<boundary>`-Tags, durch. In der Liste `context` sind die *generic identifiers* der Elemente aufgeführt, innerhalb von denen verflacht werden muß – inzwischen nur noch das Element `<overlay>`. In der Liste `boundary` sind die *generic identifiers* aufgeführt, die verflacht werden sollen. Da `<overlay>` verschachtelt auftreten kann, wird mit dem bewährten Trigger-Stapel-System Buch geführt. Der Text wird sequentiell durchlaufen, alle 's'-Token untersucht, und solange das Programm sich innerhalb eines `<overlay>` befindet, werden die Start- und End-Tags der Elemente `<p>`, `<lg>`, `<l>`, `<display>` und `<dline>` durch entsprechende `<boundary>`-Tags ersetzt.

13.21. *toks2sgml*

Am Ende der automatischen Konvertierung steht die Umwandlung der erzeugten Tokendaten in eine SGML-Datei. Dazu ist im Prinzip nicht mehr nötig, als alle Token vom Typ `w`, `s`, `e` und `b` herauszufiltern und ihre Daten unmittelbar aneinanderzuhängen, mit Ausnahme der `b`-Token, die natürlich in Leerzeichen verwandelt werden.

Die Ausgabe sollte allerdings formatiert werden, um ein Mindestmaß an Lesbarkeit zu erzeugen. Auch für die Bearbeitung unter `RCS`, das zeilenorientiert arbeitet, ist eine Aufteilung in relativ kurze Textzeilen sinnvoll. Die Formatierung muß aber natürlich so erfolgen, daß der Dateninhalt der SGML-Elemente unverändert bleibt.

Das Programm versucht, SGML-Tags auf eine eigene Zeile zu stellen und keine Zeile länger als 72 Zeichen werden zu lassen.Es besteht immer die Möglichkeit, statt Leerzeichen innerhalb von SGML-Tags einen Zeilenwechsel einzufügen. Davon habe ich aber keinen Gebrauch gemacht, um bei der Bearbeitung im Texteditor besser nach Tags suchen zu können.

Stattdessen wird in `toks2sgml` von den Regeln zur Behandlung von Zeilenwechseln in den Daten (*Record Ends* in den Begriffen von SGML, da *Record Starts* praktisch keine eigenständige Rolle spielen) Gebrauch gemacht, so gut es mit vertretbarem Aufwand möglich war. Dabei bin ich davon ausgegangen, daß es für die Daten unerheblich ist, ob anstelle eines Leerzeichens mehrere Leerzeichen oder Zeilenwechsel auftreten; entscheidend ist, daß Wortzwischenraum nicht völlig verschwindet oder an einer falschen Stelle erzeugt wird.

Die Regeln des ISO-Standards, die Goldfarb auf Seite 321–324 abdruckt und kommentiert, laufen darauf hinaus, das Zeilenwechsel nach einem regulären Start-Tag und vor einem regulären End-Tag ignoriert werden; Zeilenwechsel nach dem End-Tag werden in gewissen Fällen ignoriert, aber nicht in allen. Zeilenwechsel nach einem leeren Element, das ohne Kenntnis der DTD nicht von einem Start-Tag zu unterscheiden ist, werden nicht ignoriert. Für die praktische Vorgehensweise ergeben sich daraus folgende Regeln:

1. Beim Aufbau einer reinen Textzeile wird bei jedem neuen Leerzeichen die erreichte Länge überprüft. Wenn eine Zeile länger als 72 Zeichen geworden ist, wird am zuletzt gesehenen Leerzeichen umbrochen. Kam bisher kein Leerzeichen vor, wird die Zeile vollständig ausgegeben und eine neue begonnen. Leerzeichen können beim Umbruch durch den Zeilenwechsel ersetzt werden.
2. Wird ein Start-Tag in eine Textzeile eingefügt, wird am zuletzt gesehenen Leerzeichen umbrochen. Im Anschluß an den Start-Tag wird an den nächsten Leerzeichen wieder umbrochen, solange bis wieder Textdaten erscheinen. Das Leerzeichen nach einem Start-Tagwird in die nächste Zeile übernommen.
3. Wird ein End-Tag in eine Textzeile eingefügt, wird unmittelbar davor umbrochen. Ein eventuelles Leerzeichen wird in der neuen Zeile vor dem End-Tageingefügt. Nach dem End-Tag wird an den nächstmöglichen Stellen wieder umbrochen, bis wieder Textdaten erscheinen.

Das Programm `toks2sgml` steht im Anhang. Dort werden diese Regeln durch die Statusflags `keepspace`, `blankstate` und `wastag` sowie die Position des letzten Leerzeichens in `lastblank` umgesetzt, während die Hauptschleife den Tokenstrom sequentiell verarbeitet und je nachdem die Routinen `opentag`, `closetag`, `dospace` (für Leerzeichen) oder `append` (für Textdaten) aufruft. Ein Beispiel für die Ausgabe ist der Anfang des achten Kapitels, der ab Seite 72 abgedruckt ist.

14. Die weitere Bearbeitung

Die Arbeit an der elektronischen Fassung war mit der Produktion der automatisch konvertierten Dateien nicht beendet. Bei den manuellen Verbesserungen und der Montage zu einem einheitlichen SGML-Dokument waren folgende Programmpakete besonders wichtig:

Der SGML-Parser `nsgmls` von James Clark diente zur Überprüfung der SGML-Dateien auf syntaktische Korrektheit. Während einige kommerzielle SGML-Produkte mit der modularen Struktur der TEI DTD nicht zurechtkommen, scheint `nsgmls` allen Anforderungen des ISO Standards zu entsprechen und alle *features* außer `DATATAG` und `CONCUR` zu unterstützen.

Das `psgml`-Paket von Lennard Staflin ist ein *major mode* für `emacs`, mit dem `emacs` sich in einen SGML-Editor verwandelt, der sich meiner Ansicht nach für die Eingabe von SGML-Dateien mindestens ebenso gut eignet wie beispielsweise `author/editor`, ein kommerzielles Programm von SoftQuad. Seine 'DTD-*awareness*' war allerdings bei der Bearbeitung der einzelnen Kapiteldateien, die ja keine vollständigen SGML-Dokumente sind, eher hinderlich; dafür habe ich den einfachen *fundamental mode* verwendet.

Das Loblied auf das `RCS`-Paket habe ich bereits an anderer Stelle gesungen (auf Seite 78). Ich kann nur noch einmal betonen, daß allein die Option '`-j`' des Befehls '`co`' mehr zur Qualität des elektronischen Endprodukts beigetragen hat, als ich in wochenlanger Extraarbeit hätte leisten können, da erkannte systematische Fehler der Konvertierung repariert werden konnten, ohne die bereits vorgenommenen manuellen Eingriffe zu verlieren.

Leider kam es aus Zeitgründen nicht mehr zur Anwendung der `sgmls.pl`-Bibliothek von David Megginson, die ein `perl`-Backend zu James Clarks Parser darstellt und ein mögliches Werkzeug zur Weiterverarbeitung der *Ulysses*-Dateien gewesen wäre. Auch mit James Clarks DSSSL-Maschine `jade` hätte ich gerne noch experimentiert.

15. Ausblick

Ich habe mich bei dieser Diplomarbeit relativ blauäugig an ein sehr umfangreiches Konvertierungsprojekt gemacht. Ich habe dafür eine erschöpfende Dokumentanalyse inclusive einer Entschlüsselung der Codierung der Quelldateien geleistet, eine einigermaßen tragfähige SGML-Struktur für das Zieldokument entworfen und mit Hilfe von mehreren tausend Zeilen Software und etwas Handarbeit dieses Zieldokument tatsächlich erzeugt. So weit, so gut.

Ich habe bereits mehrfach darauf hingewiesen, daß die Konvertierung des *Ulysses* kein linearer Denk- und Arbeitsprozeß war, in dem auf die Dokumentanalyse die Definition der SGML-Strukturen und darauf die Programmierung gefolgt ist und jeder Schritt das vorausbestimmte Resultat fehlerfrei abgeliefert hat. Viele der bis jetzt beschriebenen Arbeitsschritte haben korrigierend und gestaltend auf die jeweils vorausgehenden zurückgewirkt.

Ich bin überzeugt, daß bei unterschiedlichen Anwendungen der jetzt vorliegenden SGML-Edition echte Fehler, Verbesserungsmöglichkeiten und Ideen für die Gestaltung und Erweiterung der Edition auftauchen würden, die wiederum neue Anforderungen an die vorhergehenden Stufen der Herstellung stellen. Wenn ich weitere Zeit auf das Projekt verwenden würde, wäre der nächste Schritt sicher eine Phase des Experimentierens mit der Darstellbarkeit in verfügbaren Browsern, der Export des Textes in einer dem gedruckten Buch ähnlichen Form, der Extraktion einzelner Textschichten und andere, komplexere Zugriffe auf die 'eingebaute' Information. Auch ein 'Beta-Test' durch breitere Benutzergruppen wäre sicher hilfreich, dem steht allerdings das Urheberrecht der *Trustees of the Estate of James Joyce* noch für einige Jahrzehnte im Weg.

Ich halte es durchaus für möglich, daß das Ergebnis dieser Experimente der Entschluß ist, auf der Basis der in dieser Arbeit dokumentierten Erfahrungen die Konvertierung noch einmal von Grund auf durchzuführen, um auf direktem Weg zu einer besseren Fassung zu kommen. Für grundlegende Änderungen der SGML-Struktur wäre dies sicher unumgänglich, für bescheidenere Modifikationen kann die vorliegende Form aber durchaus als Basis dienen.

Zwei Erweiterungen würde ich besonders begrüßen: zum einen die Einbindung gescannter Originaldokumente in den Text, zumindest der Joyceschen Manuskripte; es ist allerdings gründlich zu überlegen, wie reibungslos sich das vor allem für Quellen jenseits der Grundschicht mit dem Textbegriff der Gabler-Edition verträgt. Zum anderen würden die meisten Leser (wie auch ich) sicher inhaltliche Anmerkungen begrüßen, wie sie zum Beispiel Giffords und Seidmanns Buch *Ulysses annotated* bietet; solche Anmerkungen ließen sich durch einen Mechanismus, wie ich ihn für die *Textual Notes* verwendet habe, ohne weiteres einbinden.

Solche Pläne sind aber, nach einem arbeitsreichen halben Jahr, zunächst nur die virtuelle Zukunftsmusik des Verfassers.

Anhang

Die Symbole der Buchausgabe

Im folgenden gebe ich die Erklärung der Symbole für synoptischen Text, Emendationen und historische Kollation wieder, so wie sie sich auf den ersten Seiten jedes Bandes der gedruckten *Ulysses*-Ausgabe findet.

1. Synopsis

Revisions within one document (intra-document changes):
The extent of additions and replacements is indicated by pairs of carets. These are rotated for successive levels of revision within one document. *Currente calamo* cancellations and deletions in revision are enclosed in pointed brackets.

<table>
<tr><td>⟨text cancelled⟩</td><td>deletion only within one document (currente calamo cancellation)</td></tr>
<tr><td>^text new^</td><td>revision (addition) within one document at overlay level</td></tr>
<tr><td>^⟨text old⟩ text new^</td><td>revision (deletion and replacement) within one document at overlay level</td></tr>
<tr><td>⟨^text old⟩</td><td>deletion only within one document at overlay level</td></tr>
<tr><td>^() . . . . . ()^</td><td>revision, positioning inferred</td></tr>
<tr><td>*^ . . . . . ^*</td><td>revision implied by other changes and editorially construed</td></tr>
<tr><td>|^|</td><td>paragraph separation at overlay level</td></tr>
<tr><td>←^→</td><td>paragraph cancellation at overlay level</td></tr>
<tr><td>^ . . . . . ^</td><td>first overlay level</td></tr>
<tr><td>> . . . . . <</td><td>second overlay level</td></tr>
<tr><td>v . . . . . v</td><td>third overlay level</td></tr>
<tr><td>< . . . . . ></td><td>fourth overlay level</td></tr>
<tr><td>\ . . . . . / and \\ . . . . . //</td><td>pseudo-overlay at first and second levels, e.g., interlinear addition of text already present in extant early draft</td></tr>
</table>

Revisions in the progression of documents (inter-document changes):
The extent of additions and replacements is indicated by pairs of half-brackets. The levels of revision are indexed by letters of the alphabet before the first typesetting in proof (1st *placards*) and by rising numbers thereafter. Deletions are enclosed in square brackets.

<table>
<tr><td>⌜Btext newB⌝</td><td>revision (addition) at level B</td></tr>
<tr><td>⌜B[text old] text newB⌝</td><td>revision (deletion and replacement at level B</td></tr>
<tr><td>[Btext old]</td><td>deletion only at level B</td></tr>
<tr><td>⌜(B) . . . . . (B)⌝</td><td>revision (addition) at level B, document of entry lost</td></tr>
<tr><td>⌜B() . . . . . ()B⌝</td><td>revision (addition) at level B, positioning inferred</td></tr>
<tr><td>⌜*B . . . . . B*⌝</td><td>revision (addition) at level B, implied by other changes and editorially construed</td></tr>
<tr><td>|B|</td><td>paragraph separation at level B</td></tr>
<tr><td>←B→</td><td>paragraph cancellation at level B</td></tr>
</table>

Index letter A generally indicates changes between a holograph and a typescript typed from it.

Index letters B, C and D indicate the stages of revision belonging to the typescripts extant or lsot (i.e., their reproduction of a lost revised final working draft and/or their revisional autograph overlay).

Index letter R indicates collateral revisions uniquely present in the fair copy (Rosenbach Manuscript) and editorially admitted.

Index letter V indicates collateral revisions uniquely present in the fair copy and editorially not admitted (NB: the unique fair-copy reading is bracketed {.....}).

Index letter W in episode 15 ("Circe") only indicates collateral revisions present in the typescript in a brief mid-chapter section.

Index letter S indicates the entry of a revision in a scribal hand.

Index letters S and A in episode 10 ("Wandering Rocks") only distinguish the strata of scribal inscription and overlay and of authorial overlay in the Rosenbach Manuscript.

Index numbers 1 to 12 indicate the levels of revision at the successive proof stages; an independent series begins for each chapter.

⌜()..... ()⌝ in episode 15 ("Circe") only: missing section in the autograph

⌜+.....+⌝ in episode 17 ("Ithaca") only: addition of complete question-and-answer segments in the final working draft

Further symbols:

∅	space reserved in the autograph
□	word or words illegible
▯	letter, letter sequence or mark of punctuation illegible
⧄	erasure
△	insert indicated but missing
‖	turn of leaf in the autograph
-|	word division at line-break in the autograph, hyphenated
|	word division at line-break in the autograph, not hyphenated
-|-	word division at line-break in the autograph, doubly hyphenated
°	reference to a single-word emendation note or footnote
‹	reference to a phrase-extension emendation note or footnote
›	end of phrase extension
⌧	reference to a notable change in the document situation (footnoted)
~	identical text in roman to text italicized in revision
* in episode 15 ("Circe") only:	emendation without specification of source in direction-text surroundings (upper- or lower case initial letters supplied, full stops and parentheses supplied or deleted)
* in episode 18 ("Penelope") only:	emendation without specification of level (marks of punctuation and apostrophes eliminated)

2. List of Emendations and Footnotes; Historical Collation List

The symbols of the Synopsis recur by analogy in the list of emendations and footnotes and the historical collation list. Modifiers prefixed to the index letters and numbers of the synoptic presentation provide sigla to specify the source, the quality and, directly or by inference, the authority of readings accepted or rejected; thus 'a' stands for authorial inscription, 's' for scribal inscription, 't' for typist's inscription. Additional level or document indicatios not used in the Synopsis are:

P and PP (in the combinations aP and aPP)	extant early drafts preceding lost final working draft or extant fair copy at one or two removes
W (in the combination (aW))	lost final working draft
LR	*Little Review* serialisation

Eg	*Egoist* serialisation

The following are some recurrent sigla and typical combinations:

aR	authorial reading in Rosenbach Manuscript
aR?	authorial reading in Rosenbach Manuscript, doubtful scribal reading in Rosenbach Manuscript
sR	scribal reading in Rosenbach Manuscript
tB	typed reading in a typescript copied from the Rosenbach Manuscript
(tB)	*do.*, document lost
aB, aC	authorial reading (correction or revision) as ‘B’ or ‘C’ typescript overlay
tC	typed reading in a typescript copied from a lost final working draft
(aW):tC	authorial reading conjectured for the lost final working draft by the evidence of the typescript
aC, aD	authorial reading (correction or revision) as ‘C’ or ‘D’ typescript overlay
sC, sD	scribal reading at relevant copy-editing level of typescript
sB in episode 15 (“Circe”) only:	scribal reading in amanuensis copy
1	type-set reading in proofs defined as level 1 substratum
a1	authorial reading (correction or revision) as level 1 overlay
1926, 1932, 1936, 1937	an edition in Joyce’s lifetime as a source of emendation
e	editorial emendation, independent or combining features from several document readings cited
e:tB	editorial emendation in main agreement with typescript tB
e:1960	editorial emendation following edition of 1960
e:JD	editorial emendation in agreement with a suggestion from Jack Dalton
aE	correction in autograph list of errata
tE	correction in typed list of errata
sE	correction in scribal amendment of typed list of errata
p1E	correction in first printed list of errata
p2E	correction in second printed list of errata
JJRec	variant in Joyce’s recording of a section from episode 7 (“Aeolus”)
$\rightarrow$ TN	reference to the list of Textual Notes
TD:	transmissional departure (unauthorized change or error in a scribal copy, typescript or proof)
NU	not underlined
PCU	printer’s copy unchanged
PNR	proofs not returned
ABSENT	reading absent in document cited
DISPLACED	reading displaced in document cited
MISINSERTED	reading misinserted in document cited

Editorial comments are printed in sloped small-caps.

3. Historical Collation List

The symbols and sigla of the list of emendations and footnotes recur in the historical collation list, supplemented by

::	prefix to edition or editions cited
22–61	all collated editions from 1922 to 1961
26–32	editions of 1926 and 1932 only
22.32–61	all collated editions except 1926
61	edition of 1961 only
⟨aE⟩	reference to autograph list of errata
⟨123.45⟩	page.line reference to 1961 Random House edition
\	prefix to the siglum or sigla denoting the source of a first-edition reading rejected
* in episode 15 ("Circe") only:	asterisked element reciprocal to editorial normalisation; collation record included for other variation

NB: The headnotes over the Textual Notes for the episodes should be consulted for the editorial situation and indices relevant to each chapter. A discursive explication of the apparatus will be found in the Afterword.

Die Datei 'ulysses.sgml'

```
<!DOCTYPE tei.2 SYSTEM 'tei2.dtd' [    <!- -*-fundamental-*- ->
<!-

        ulysses.sgml
        ============

        The frame file for the TEI/SGML Ulysses Edition; the text for
        each chapter is included via entity references.

        For the text: (c) 1984, The Trustees of the Estate of James Joyce.

        For genetic synopsis and notes: (c) 1984, Hans Walter Gabler.

        Electronic version created by Tobias Rischer in 1996/7.  There
        are no objections against non-commercial use of this work from
        my side.

->
<!-
        RCS Log.  Revision numbers 0 do not yet parse correctly, numbers
        1 do parse, but are not yet complete (pictures and necessary
        hand tagging missing etc.)  Further revisions can be used when
        extending the edition conceptually.

$Log: ulysses.sgml,v $
# Revision 1.7  1997/01/31  12:49:39  tobsi
# Removed the definition of a coming 'Revision 2' from comment on
# revisions and logging.
#
# Revision 1.6  1997/01/29  19:22:07  tobsi
# Updated the log in the TEI header.
#
# Revision 1.5  1997/01/29  18:24:09  tobsi
# 'n'-Attributes for episodes were not in the last revision, after all.
# Here they come.
#
# Revision 1.4  1997/01/29  18:21:31  tobsi
# Cut out the bibliography list and put that into a new file.
# Introduced file for layout exceptions.
# Added 'n'-Attribute to the episodes.
# Found that the 5th episode never was part of the document and added
# it...
#
# Revision 1.3  1997/01/13  10:09:52  tobsi
# Added code for the three graphic elements needed.
#
# Revision 1.2  1997/01/11  19:32:42  tobsi
# Base tag set changed from Prose to Mixed, because the drama element
# <castlist> is needed in episode 9.
#
# Revision 1.1  1997/01/09  21:36:29  tobsi
# Added some imperial touch to the first lines of this file.
#
# Revision 1.0  1996/11/28  11:58:20  tobsi
# Ulysses makes it through the parser
#
# Revision 0.6  1996/11/28  11:45:35  tobsi
# Fertig gemacht fuer eine brauchbare Version 1.0.
#
# Revision 0.5  1996/11/27  10:28:35  tobsi
# Auskommentieren war Schmarrn.  Jetzt sind die gesamten Parts I und II
```

```
# auskommentiert.
#
# Revision 0.4  1996/11/27  10:26:46  tobsi
# Episoden 1-15 auskommentiert.
#
# Revision 0.3  1996/11/27  10:18:05  tobsi
# Episoden 1-14 auskommentiert, damit das Parsen schneller laeuft
# (und um zu testen, ob in Circe ein Capacity-Problem vorliegt).
#
# Revision 0.2  1996/11/20  12:34:34  tobsi
# Der TEI Header und Episode 1-9 parsen (im ersten pass) fehlerfrei; einzelne Layoutkorrekturen in 1-9.
#
# Revision 0.1  1996/11/14  15:23:00  tobsi
# Initial Revision
#

->

<!- TEI - Modules... ->
<!ENTITY % TEI.mixed    'INCLUDE'>
<!ENTITY % TEI.prose    'INCLUDE'>
<!ENTITY % TEI.drama    'INCLUDE'>

<!ENTITY % TEI.textcrit 'INCLUDE'>
<!ENTITY % TEI.transcr  'INCLUDE'>
<!ENTITY % TEI.linking  'INCLUDE'>
<!ENTITY % TEI.figures  'INCLUDE'>

<!- Writing System Declarations ->
<!ENTITY wsd.english SYSTEM "teien.wsd" SUBDOC>

<!- Character sets ->
<!ENTITY % ISOlat1 SYSTEM "isolat1.ent">
<!ENTITY % ISOlat2 SYSTEM "isolat2.ent">
<!ENTITY % ISOnum  SYSTEM "isonum.ent">
<!ENTITY % ISOpub  SYSTEM "isopub.ent">
<!ENTITY % ISOtech SYSTEM "isotech.ent">
%ISOlat1;
%ISOlat2;
%ISOnum;
%ISOpub;
%ISOtech;

<!- Nonstandard entities for character representation ->
<!ENTITY % uly.new.entities SYSTEM "uly_new.ent">
%uly.new.entities;

<!- TEI extensions ->
<!ENTITY % TEI.extensions.ent SYSTEM "uly_ext.ent">
<!ENTITY % TEI.extensions.dtd SYSTEM "uly_ext.dtd">

<!- The text chapters ->
<!ENTITY uly01 SYSTEM "uly01.sgml">
<!ENTITY uly02 SYSTEM "uly02.sgml">
<!ENTITY uly03 SYSTEM "uly03.sgml">
<!ENTITY uly04 SYSTEM "uly04.sgml">
<!ENTITY uly05 SYSTEM "uly05.sgml">
<!ENTITY uly06 SYSTEM "uly06.sgml">
<!ENTITY uly07 SYSTEM "uly07.sgml">
<!ENTITY uly08 SYSTEM "uly08.sgml">
<!ENTITY uly09 SYSTEM "uly09.sgml">
<!ENTITY uly10 SYSTEM "uly10.sgml">
<!ENTITY uly11 SYSTEM "uly11.sgml">
<!ENTITY uly12 SYSTEM "uly12.sgml">
```

```
<!ENTITY uly13 SYSTEM "uly13.sgml">
<!ENTITY uly14 SYSTEM "uly14.sgml">
<!ENTITY uly15 SYSTEM "uly15.sgml">
<!ENTITY uly16 SYSTEM "uly16.sgml">
<!ENTITY uly17 SYSTEM "uly17.sgml">
<!ENTITY uly18 SYSTEM "uly18.sgml">

<!- Graphics ->
<!NOTATION tiff PUBLIC '-//Aldus Corp.//NOTATION Tagged Image File
 Format//EN'>
<!ENTITY graphic.gloria SYSTEM "gloria.tif" NDATA tiff>
<!ENTITY graphic.harry1 SYSTEM "harry1.tif" NDATA tiff>
<!ENTITY graphic.harry2 SYSTEM "harry2.tif" NDATA tiff>

<!- Additional matter ->

<!ENTITY tn    SYSTEM "uly_tn.sgml"   - the textual notes - >
<!ENTITY bibl  SYSTEM "uly_bib.sgml"  - Bibliography of Ulysses editions ->
<!ENTITY syn   SYSTEM "uly_syn.sgml"  - the synoptic level definitions ->
<!ENTITY specs SYSTEM "uly_spec.sgml" - special layout list ->

<!-
        The following two entities represent the parentheses that
        enclose stage directions in episode 15 (Circe).  It may depend
        on circumstances whether one would want them as data in the text
        or not, therefore they can be redefined here.
->

<!ENTITY stagepo        "("     - stage parenthesis open - >
<!ENTITY stagepc        ")"     - stage parenthesis closed - >

] >
<tei.2>
  <teiheader>
    <filedesc>
      <titlestmt>
        <title>Ulysses. An electronic version of the critical and
synoptic edition.
        </title>
        <respstmt>
          <resp>Electronic version prepared by</resp>
          <name>Tobias Rischer</name>
        </respstmt>
      </titlestmt>
      <extent>Currently ca 8 Megabyte</extent>
      <publicationstmt>
        <p>This electronic edition was created as part of a
<term>Diplomarbeit</term> for the <rs>Technische Universit&auml;t
M&uuml;nchen</rs>.  <emph>Note that the text and the edition are
copyrighted and may not be distributed freely.</emph> The copyright
notices of the printed edition still apply where applicable:</p>
        <p>Draft texts, revisional material, and reading text of
<title>Ulysses</title> &copy; <date>1984</date>, The Trustees of the
Estate of James Joyce</p>
        <p>Presentation of Genetic Synopsis, Notes and Afterword &copy;
<date>1984</date>, Hans Walter Gabler</p>
        <p>The compiler of the electronic version, from his side, agrees
to all non-commercial use of his work.</p>
      </publicationstmt>
      <notesstmt>
        <note>This electronic edition was first automatically converted
from the TUSTEP files used for preparation of the printed edition;
necessary corrections and additional tagging were done by hand
afterwards.</note>
```

```
        <note>The electronic edition comprises the continuous
manuscript text as on the left hand pages of the printed edition (this
also implies the reading text, as on the right hand pages), the
emendatory footnotes, the historical collation and the textual notes in
the Appendix.  The date information from table 1 on page 1913 is also
incorporated, so are some data from the bibliography relevant to the
historical collation.  The preface, afterword, and the additional
information in the bibliography and table 1 and 2 in the appendix are
not included.</note>
      </notesstmt>
      <sourcedesc>
        <biblstruct id=gab>
          <monogr>
            <author>James Joyce</author>
            <editor>Hans Walter Gabler</editor>
            <title type=main>Ulysses</title>
            <title type=subordinate>A Critical and Synoptic Edition</title>
            <edition>Second Impression</edition>
            <imprint>
              <publisher>Garland Publishing Inc.</publisher>
              <pubplace>New York</pubplace>
              <pubplace>London</pubplace>
              <date>1986</date>
            </imprint>
          </monogr>
        </biblstruct>
      </sourcedesc>
    </filedesc>
    <encodingdesc>
      <projectdesc>
        <p>
This electronic edition was prepared as part of a computer science
<term>Diplomarbeit</term> for the <rs>Technische Universit&auml;t
M&uuml;nchen</rs>.  It is an attempt to remodel the information
structure of the printed Gabler edition of <title>Ulysses</title>
explicitely in SGML, as closely as possible to the TEI P3 standard.</p>
        <p>
With the time and resources given, the bulk of the SGML text files had
to be generated automatically from the electronic documents used in the
preparation of the printed edition.  This hopefully gives a very correct
text, but in many places gives crude and/or bad tagging.</p>
        <p>
The tagging applied shall serve two main purposes: To generate a
TEI-compatible version of some particular level of the text, notably the
reading text as established by the Gabler edition, and to create an
online critical edition that allows more intuitive access to the
information structures available in the Gabler edition.</p>
      </projectdesc>
      <editorialdecl>
        <quotation marks=all form=data>
          <p>Quotations are not tagged explicitely.  Direct speech in
the main text is marked exclusively with em dashes at the beginning of a
new paragraph.  What graphically looks like left and right single quotes
in the main text are apostrophes and left single quotes used in
contracted Irish names as <q>M&lsquo;Coy</q> and in similar forms of
transcription (e.g. from Hebrew).  These are coded as entity
references to the standard ISO entities apos and lsquo.  Quotations in
the notes use single or double quotes.</p>
        </quotation>
        <hyphenation eol=none>
          <p>End-of-line hyphenation and soft hyphens introduced for
typographic reasons in the computer files for the printed edition is
discarded or represented by soft hyphen entities (shy).  Explicit
hyphens in the text can be considered as hard hyphens in compound words.
```

```
Hyphenation information related to Joyce's manuscripts that is provided
explicitely in the printed edition is marked with word break tags
(wbreak).</p>
        </hyphenation>
        <segmentation>
          <p>The main text is separated in three divisions, representing
the three parts of the book.  Each part is subdivided into episodes,
eighteen altogether.  The episodes 7 (Aeolus), 10 (Wandering Rocks) and
17 (Ithaca) are again subdivided into their obvious elements.</p>
        </segmentation>
        <stdvals>
          <p>Where dates are given as attribute values, they are either
of the form <q>1904</q> to denote a whole year, or of the form
<q>1904-06-16</q> to denote a particular day (the sixteenth of June 1904
in this case).  There are <q>date</q> and <q>daterange</q> elements
without content where the date ist to be derived from the
attributes.</p>
        </stdvals>
      </editorialdecl>
      <refsdecl>
        <p>Two canonical reference systems are coded into the main text:
There are the page and line-per-page numbers of the left-hand pages (the
synoptic pages) of the printed edition, which are coded with <q>pb</>
and <q>lb</> milestones with the <q>ed</> attribute set to <q>syn</q>
(for <q>synoptic</>).  And there are the lines-per-chapter numbers
serving as a standard reference to the reading text of the Gabler
edition, which are coded with <q>lb</> tags with attribute <q>ed</> set
to <q>gab</q> (for <q>Gabler</q>).  To complete these references, one
needs the chapter number given in the <q>n</> attribute of the episode
level <q>div</> elements.</p>
        <p>It may be unfortunate, but there are also <q>lb</q> tags
in the notes that are not part of a reference system; they indicate a
relevant line break in the source manuscript instead.</p>
      </refsdecl>
    </encodingdesc>
    <profiledesc>
      <langusage>
        <language id=eng wsd=wsd.english>English</language>
        <p>
The main language of the text is English, although a lot of the words
are not Standard English but new coinages that often also mimic other
languages.</p>
        <p>
In the current version of the SGML transcription, no distinction of
languages is made &mdash; all italics are just tagged as
highlighted.</p>
      </langusage>
      <handlist>
<!-
        The <hand> tag is used in a more abstract way than TEI suggests.
        The hand= attribute seems close to a bug to me, at least a
        documentation bug. (this was meanwhile also pointed out by
        someone else on TEI-L)
->
        <hand id=author hand=author scribe='James Joyce'>
        <hand id=scribe hand=scribe scribe='Various typists and scribes'>
        <hand id=editor hand=editor scribe='Hans Walter Gabler'>
      </handlist>
    </profiledesc>
    <revisiondesc>
<!-
      The revision description here refers to the development of the
      Ulysses edition as a whole; the RCS log entries in each file refer
      to the revision stages of each file.  The major steps (first
```

```
      correct parse, first time complete) that are reflected in the
      major revision numbers of the files shall also be identifiable
      here. TR.
->
      <change>
        <date value=1996-11-14>14 Nov, 1996</date>
        <respstmt>
          <resp>First created by</resp>
          <name>Tobias Rischer</name>
        </respstmt>
        <item>SGML version frozen from the last automatic conversion.
Incomplete and faulty in details.</item>
      </change>
      <change>
        <date value=1996-11-28>28 Nov, 1996</date>
        <respstmt>
          <resp>Modified by</resp>
          <name>Tobias Rischer</name>
        </respstmt>
        <item>Pushed and pulled through the needle's eye: it parses.
Corrected some obvious conversion blunders, but not too systematically.</item>
      </change>
      <change>
        <date value=1997-2-1>1 Feb, 1997</date>
        <respstmt>
          <resp>Modified by</resp>
          <name>Tobias Rischer</name>
        </respstmt>
        <item>I have remodelled the file structure for the appendix
slightly and added the layout specs for exceptional cases.  I did a lot
of manual editing at critical spots and have rerun the conversion for
systematic blunders.  The whole thing is a lot better now, but still
needs testing and improvement.  The tag set documentation is not
completed.</item>
      </change>
    </revisiondesc>
  </teiheader>
  <text lang=eng>
    <body>
      <div type=part>
        <head><num value=1>I</num></head>

        <div type=episode n=1>
          <head>Telemachus</head>
          &uly01;
        </div>

        <div type=episode n=2>
          <head>Nestor</head>
          &uly02;
        </div>

        <div type=episode n=3>
          <head>Proteus</head>
          &uly03;
        </div>

      </div>
      <div type=part>
        <head>
          <num value=2>II</num>
        </head>

        <div type=episode n=4>
```

```
    <head>Calypso</head>
    &uly04;
  </div>

  <div type=episode n=5>
    <head>Lotus Eaters</head>
    &uly05;
  </div>

  <div type=episode n=6>
    <head>Hades</head>
    &uly06;
  </div>

  <div type=episode n=7>
    <head>Aeolus</head>
    &uly07;
  </div>

  <div type=episode n=8>
    <head>Lestrygonians</head>
    &uly08;
  </div>

  <div type=episode n=9>
    <head>Scylla and Charybdis</head>
    &uly09;
  </div>

  <div type=episode n=10>
    <head>Wandering Rocks</head>
    &uly10;
  </div>

  <div type=episode n=11>
    <head>Sirens</head>
    &uly11;
  </div>

  <div type=episode n=12>
    <head>Cyclops</head>
    &uly12;
  </div>

  <div type=episode n=13>
    <head>Nausicaa</head>
    &uly13;
  </div>

  <div type=episode n=14>
    <head>Oxen of the Sun</head>
    &uly14;
  </div>

  <div type=episode n=15>
    <head>Circe</head>
    &uly15;
  </div>

</div>
<div type=part>
  <head>
    <num value=3>III</num>
  </head>
```

```
        <div type=episode n=16>
          <head>Eumaeus</head>
          &uly16;
        </div>

        <div type=episode n=17>
          <head>Ithaca</head>
          &uly17;
        </div>

        <div type=episode n=18>
          <head>Penelope</head>
          &uly18;
        </div>

      </div>
    </body>
    <back>
      <div type=appendix>
        &tn;
      </div>
      <div type=appendix>
        <head>Bibliography of cited editions</head>
        &bibl;
      </div>
      <div type=appendix>
        <head>List of all synoptic levels</head>
        &syn;
      </div>
      <div type=appendix>
        <head>Special cases of layout</head>
        &specs;
      </div>
    </back>
  </text>
</tei.2>
```

Die Konvertierungspipeline (Makefile)

```
# Makefile fuer Konvertierung (GNU make)

CONVERTER =\
 tools/weedout \
 tools/shuffle-rdg \
 tools/milestones \
 tools/page-line-edits \
 tools/empty-tags \
 tools/position-notes \
 tools/parse-notes \
 tools/parse-hico \
 tools/layout \
 tools/starreg \
 tools/overlay \
 tools/convert-app \
 tools/charcodes \
 tools/boundary

CONVERSIONPIPE = \
 | weedout \
 | shuffle-rdg \
 | milestones \
 | page-line-edits \
 | empty-tags \
 | position-notes \
 | parse-notes \
 | parse-hico \
 | layout \
 | starreg \
 | overlay \
 | convert-app \
 | charcodes \
 | boundary

NICE = nice -15

FTOKS = uly01.ftoks uly02.ftoks uly03.ftoks uly04.ftoks \
        uly05.ftoks uly06.ftoks uly07.ftoks uly08.ftoks \
        uly09.ftoks uly10.ftoks uly11.ftoks uly12.ftoks \
        uly13.ftoks uly14.ftoks uly15.ftoks uly16.ftoks \
        uly17.ftoks uly18.ftoks

ULYSSES = uly01.sgml uly02.sgml uly03.sgml uly04.sgml \
        uly05.sgml uly06.sgml uly07.sgml uly08.sgml \
        uly09.sgml uly10.sgml uly11.sgml uly12.sgml \
        uly13.sgml uly14.sgml uly15.sgml uly16.sgml \
        uly17.sgml uly18.sgml uly_tn.sgml uly_syn.sgml \
        uly_ext.dtd uly_ext.ent uly_new.ent ulysses.sgml

all:    uly01.sgml uly02.sgml uly03.sgml uly04.sgml \
        uly05.sgml uly06.sgml uly07.sgml uly08.sgml \
        uly09.sgml uly10.sgml uly11.sgml uly12.sgml \
        uly13.sgml uly14.sgml uly15.sgml uly16.sgml \
        uly17.sgml uly18.sgml hico.sgml emen.sgml tn.sgml

parse: $(ULYSSES)
        $(NICE) nsgmls -s ulysses.sgml 2>&1 | tee ulylog

uly%.crit: texts/i%.ris tools/reformat-crit-lines
        reformat-crit-lines $< > $@
```

```
uly%.rdg: texts/c%.ris tools/reformat-rdg-lines
	reformat-rdg-lines $< > $@

uly%.crit.toks: uly%.crit tools/tokenize
	tokenize < $< > $@

uly%.rdg.toks: uly%.rdg tools/tokenize
	tokenize < $< > $@

uly%.toks: uly%.rdg.toks uly%.crit.toks tools/merge-rdg
	$(NICE) merge-rdg < uly$*.crit.toks uly$*.rdg.toks > $@

uly%.ftoks: uly%.toks tn-idlog $(CONVERTER)
	cat $< $(CONVERSIONPIPE) > $@

uly%.sgml: uly%.ftoks tools/toks2sgml tn-idlog
	toks2sgml $< > $@

tn.toks: texts/tn.ris tools/reformat-tn tools/tokenize-tn
	reformat-tn < texts/tn.ris | tokenize-tn > tn.toks

tn-idlog tn.sgml: tn.toks tools/parse-tn tools/charcodes tools/toks2sgml
	parse-tn tn.toks | charcodes | toks2sgml > tn.sgml

##############################

demo:
	makedemo

packagedos: demo
	zip -l -k demo demo/*

clean:
	-rm *~ *.tmp* *.crit.toks *.rdg.toks *.ftoks
```

Das Programm 'merge-rdg'

```
#!/usr/bin/perl

# Perl-script zum Einfuegen der Zeilennummerierung des Lesetextes in das
# toks-file der kritischen Ausgabe.  Die toks-Datei der kritischen
# Ausgabe wird von <stdin> nach <stdout> gefiltert, der Lesetext muss
# als toks-File als command line parameter angegeben werden.

# Verglichen werden die w-Token des Lesetextes mit denjenigen w-Token
# des kritischen Textes, die nicht in einer Fussnote stehen oder zu
# gestrichenen Passagen gehoeren.  Wenn eine Nicht-Uebereinstimmung
# gefunden wird, wird ein Fehler gemeldet.  Ansonsten werden die l-Token
# aus dem Lesetext jeweils vor dem ersten w-Token der Zeile eingefuegt.

%flushpatterns = (
                  "\\<",                         "\\>",
                  "\\<\\|[1-4AS]",               "\\>",
                  "\\&\\~",                      "\\~\\&",
                  "([0-9]+)\\[",                 "\\]",
                  "([0-9]+)\\(\\)\\[",           "\\]",
                  "\\|[A-Z]\\[",                 "\\]",
                  "\\|[1-4AS]\\<",               "\\>",
                  "\\|[A-Z]\\(\\)\\[",           "\\]",
                  "\\[([0-9]+)",                 "\\]",
                  "\\[([0-9]+)\\(\\*\\*\\)",  "\\]",
                  "\\[\\|[A-Z](\\(\\))?",     "\\]",
                  "\\|V\\|\\(",                  "\\|\\)",
                  "\\|\\(\\|V",                  "\\|\\)",
                  "\\&\\&\\&l\\&\\&\\&\\{",   '\\$\\$70\\$\\$\\{',
                  "\\&\\!o",                     "\\\@1",
                  "\\|[A-Z]-\\|S\\[",            "\\]",
                  "5-\\|S\\[",                   "\\]",
                  "\\|[A-Z]\\(\\)\\(\\*\\*\\)\\[", "\\]",
                  "([0-9]+|\\|[A-Z])\\(\\*\\*\\)\\[",     "\\]",
                  "\\[\\|[A-Z]\\(\\*\\*\\)",  "\\]",
                  "\\|[A-Z]\\(\\*\\)\\[",     "\\]",
                  "\\|[1-4]\\(\\*\\*\\)\\<",  "\\>",
                  "\\&\\.26\\[",                 "\\]",
                  "\\*\\<",                      "\\>",
);

sub ende {
    if ($trace) {
        print STDERR "ENDE: $critline";
    }
    while (<STDIN>) { print; }  # restliche crit-Datei ausgeben.
    print STDERR "File(s) ended.\n";
    print STDERR "$linesfound lines found.\n";
    $lastlinestring =~ /([0-9]+)\]/;
    print STDERR "Last line number found was $1\n";
    exit ($linesfound != $1)    # 0, wenn gleich, sonst 1.
}

# next_rdgword sucht das naechste w-Token im Lesetext; falls dabei ein
# l-Token passiert wird, wird es in $linestring abgelegt, ansonsten ist
# $linestring leer.  Das gefundene Wort steht in $rdgword.  Bei EOF wird
# die Subroutine ende aufgerufen.

sub next_rdgword {
    local($type, $done);

    $done = 0;
```

```
    $linestring = "";
    $hicostring = "";
    do {
        $rdgline = <rdgfile>;
        (eof) && &ende;
        ($type, $rdgword) = split (/ /, $rdgline, 2);
        chop $rdgword;

        ($type eq "l") && do {
            $linestring = $rdgline;
            $lastlinestring = $rdgline;
            $linesfound++;
        };
        ($type eq "h") && do {
            $hicostring .= $rdgline;
        };
        ($type eq "w") && do {
            $done = 1;
        };
    } while !($done);
    $rdgstream .= $rdgword;
}

# next_critword sucht das naechste w-Token im kritischen Text, das
# Bestandteil des Lesetextes ist.  Alle Token vor dem gesuchten w-Token
# werden unveraendert ausgegeben.  Das gefundene w-Token steht
# anschliessend in $critline, das Wort in $critword.  Bei EOF wird die
# Subroutine ende aufgerufen.

sub next_critword {
    local ($type, $done, $opat, @cpatstack, $skipping, $done);

    $skipping = 0;
    $done = 0;
    $cpat = "xxx";

    do {
        $critline = <STDIN>;
        print $critline;
        (eof) && &ende;
        ($type, $critword) = split (/ /, $critline, 2);
        chop $critword;

        if ($type eq "c") {
            foreach $opat (keys %flushpatterns) {
                if ($critword =~ /^$opat$/) {
                    push (@cpatstack, $cpat);
                    $cpat =  $flushpatterns{$opat};
                    ++$skipping;
                    last;
                }
            }
            if ($skipping && ($critline =~ /^c $cpat$/)) {
                $cpat = pop @cpatstack;
                -$skipping;
            }
        }
    } until ($type eq "w" && !$skipping);
    $critstream .= $critword;
}

sub fail {
    die ("Word mismatch after $linesfound lines:\n",
         "R: \"$rdgstream\" C: \"$critstream\"\n",
```

```
        "Stopped");
}

sub match {
    local ($lr, $lc) = (length($rdgstream), length($critstream));
    if ($rdgstream eq $critstream) {
        $rdgstream = '';
        $critstream = '';
    } elsif ($lc < $lr) {
        ($critstream ne substr ($rdgstream, 0, $lc)) && &fail;
        $critstream = '';
        substr ($rdgstream, 0, $lc) = '';
    } else {
        ($rdgstream ne substr ($critstream, 0, $lr)) && &fail;
        $rdgstream = '';
        substr ($critstream, 0, $lr) = '';
    }
}

if (@ARGV < 1) {
    die ("usage: $0 <reading-text.toks>\n Stopped");
}

$rdgfile = shift @ARGV;

open (rdgfile) || die ("Cannot open $rdgfile; stopped");
$critline = "";
$rdgstream = '';
$critstream = '';

mainloop: for (;;) {
    $rdgstream || &next_rdgword;
    print $hicostring;
    print $linestring;
    $critstream || &next_critword;
    $hicostring = $linestring = '';
    &match;
}
```

Die 'readlib'-Funktionen

```
#
# Perl library for buffered input.
#

@read_inlines = ();
@outbuf = ();
$outline = -1;
$rl_exitstring = '';

sub read {
    if (@read_inlines) {
        pop (@read_inlines);
    } else {
        <> || do { &output ($rl_exitstring); &flush; exit(0) };
    }
}

sub unread {
    push (@read_inlines, @_);
}

sub output {
    foreach $out (@_) {
        ++$outline;
        push (@outbuf, $out);
        }
}

# The `current token' is created in globals $line, $mode, $type, $word.
sub readback {
    @outbuf || die ("readlib:rewind with empty output. Stopped");
    $line = pop (@outbuf);
    -$outline;
    ($type, $word) = split (/ /, $line, 2);
    if (($type eq 'h') || ($type eq 'n') || ($type eq 't')) {
        $mode = $type;
        ($type, $word) = split (/ /, $word, 2);
    }
    chop $word;
}

sub flush {
    print @outbuf;
    @outbuf = ();
    $outline = -1;
}

1;                              # Always be true...
```

Das Programm 'parse-hico'

```
#!/usr/bin/perl

# Erzeugt SGML fuer Historische Kollation

require ("readlib");

$o_comm = "h s <comm rend=slcsc>\n";
$c_comm = "h s </comm>\n";
$o_abbr = "h s <abbr expan=\"%s\">\n";
$c_abbr = "h s </abbr>\n";
$o_lemma = "h s <lem>\n";
$c_lemma = "h s </lem>\n";
$o_rdg = "h s <rdg wit='%s'>\n";
$c_rdg = "h s </rdg>\n";
$o_wit = "h s <wit>\n";
$c_wit = "h s </wit>\n";
$o_rhref = "h s <PLref ed=e61 fromp=%s froml=%s top=%s tol=%s>\n";
$c_rhref = "h s </PLref>\n";
$o_departure = "h s <depart>\n";
$c_departure = "h s </depart>\n";
$o_italic = "h s <hi rend=it>\n";
$c_italic = "h s </hi>\n";
$o_caps = "<hi rend=caps>\n";
$c_caps = "</hi>\n";

%witnesscodes = (
                '22',                   'e22',
                '22&.hy26',             'e22 e26',
                '22&.hy26.36&.hy61',    'e22 e26 e36 e37 e61',
                '22&.hy26.61',          'e22 e26 e61',
                '22&.hy32',             'e22 e26 e32',
                '22&.hy32.61',          'e22 e26 e32 e61',
                '22&.hy36',             'e22 e26 e32 e36',
                '22&.hy61',             'e22 e26 e32 e36 e37 e61',
                '22.32',                'e22 e26 e32',
                '22.32&.hy61',          'e22 e32 e36 e37 e61',
                '22.36',                'e22 e36',
                '22.61',                'e22 e61',
                '25',                   ' ', # Circe, Z.2400 ??
                '26',                   'e26',
                '26&.hy32',             'e26 e32',
                '26&.hy32.61',          'e26 e32 e37 e61',
                '26&.hy36',             'e26 e32 e36',
                '26&.hy61',             'e26 e32 e36 e37 e61',
                '26.36',                'e26 e36',
                '26.36&.hy61',          'e26 e36 e37 e61',
                '26.61',                'e26 e61',
                '26<aE>',               'e26',
                '26<aE>&.hy61',         'e26 e32 e36 e37 e61',
                '26<p1E>',              'e26',
                '26<p1E>&.hy61',        'e26 e32 e36 e37 e61',
                '26<sE>&.hy61',         'e26 e32 e36 e37 e61',
                '26<tE>&.hy61',         'e26 e32 e36 e37 e61',
                '32',                   'e32',
                '32&.hy36',             'e32 e36',
                '32&.hy61',             'e32 e36 e37 e61',
                '36',                   'e36',
                '36&.hy61',             'e36 e37 e61',
                '37',                   'e37',
                '61',                   'e61',
                );
```

```
%expansion = (
              'pcu', "printer's copy unchanged",
              'pnr', 'proofs not returned',
              'nu', 'not underlined',
              );

sub nextline {
    $line = &read;
    ($mode, $type, $word) = split (/ /, $line, 3);
    chop $word;
    ($mode ne 'h') && do {
        &flush;
        die ("parse-hico: hico overflow. Stopped");
    }
}

sub try_abbr {
    if ($type eq 'a') {
        $word =~ s/(\+\+\})|(\{)//g;
        &output ($o_comm,
                sprintf ($o_abbr, $expansion{$word}),
                "h w $word\n",
                $c_abbr,
                $c_comm);
        &nextline;
        1;
    } else {
        0;
    }
}

sub try_comm {
    if ($word eq '++}') {
        &output ($o_comm);
        &nextline;
        while ($word ne '{') {
            &output ($line);
            &nextline;
        }
        &output ($c_comm);
        &nextline;
        1;
    } else {
        0;
    }
}

sub try_italic {
    if ($word eq '@@}') {
        &output ($o_italic);
        &nextline;
        while ($word ne '{') {
            &output ($line);
            &nextline;
        }
        &output ($c_italic);
        &nextline;
        1;
    } else {
        0;
    }
}
```

```
sub try_caps {
    if ($word eq '++') {
        &output ($o_caps);
        &nextline;
        while ($word ne '{') {
            &output ($line);
            &nextline;
        }
        &output ($c_caps);
        &nextline;
        1;
    } else {
        0;
    }
}

sub do_lemma {
    # Copy initial blank(s)
    while ($type eq 'b') {
        &output ($line);
        &nextline;
    }
    # Start tag
    &output ($o_lemma);
    # Copy text until ']'
    while ($word ne ']') {
        (&try_italic
         || &try_caps
         || do {
             &output ($line);
             &nextline;
         }
         );
    }
    # End Tag
    &output ($c_lemma);
    &nextline;                  # swallow ']'.
}

sub do_rdg {
    # Copy initial blank(s)
    while ($type eq 'b') {
        &output ($line);
        &nextline;
    }
    # Dummy start tag, but remember its line!
    &output ("h s <rdg>\n");
    $rdgtagline = $outline;
    # Copy contents, until '::='
    while ($word ne '::=') {
        (&try_abbr
         || &try_comm
         || &try_italic
         || &try_caps
         || do {
             &output ($line);
             &nextline;
         }
         );
    }
    # Make ::= disappear and back up over last blank(s)
    &readback;                  # $line from output
    while ($type eq 'b') {
        &unread ($line);                # Back to input
```

```
        &readback;              # next $line from output
    }
    &output ($line);            # Last one was not blank.
    # End tag
    &output ($c_rdg);
    &nextline;
}

sub do_witness {
    # Copy blanks
    while ($type eq 'b') {
        &output ($line);
        &nextline;
    }
    ($type ne 'f') && die ("parse-hico: $line No f tag in witness! Stopped");
    # Parse witness data
    $outbuf[$rdgtagline] = sprintf ($o_rdg, $witnesscodes{$word});
    &output ($o_wit);
    $word =~ s/\&\.hy/\-/g;
    $word =~ s/\</\&lt\;/g;
    $word =~ s/\>/\&gt\;/g;
    &output ("h w $word\n", $c_wit);
    &nextline;
}

sub try_rhref {
    # Copy blanks
    while ($type eq 'b') {
        &output ($line);
        &nextline;
    }
    if ($type eq 'r') {
        local ($fpage, $fline, $tpage, $tline);
        if ($word =~ /^\<(\d+)\.(\d+)\>/) {
            $fpage = $tpage = $1;
            $fline = $tline = $2;
        }
        elsif ($word =~ /^\<(\d+)\.(\d+)\&\.hy(\d+)\>/) {
            $fpage = $tpage = $1;
            $fline = $2;
            $tline = $3;
        }
        elsif ($word =~ /^\<(\d+)\.(\d+)\&\.hy(\d+)\.(\d+)\>/) {
            $fpage = $1;
            $fline = $2;
            $tpage = $3;
            $tline = $4;
        }
        $word =~ s/(\<)/\&lt\;/; # Replace angle brackets
        $word =~ s/\>/\&gt\;/;  # with entity reference.
        $word =~ s/\&\.hy/-/;
        &output ("h b 20\n");   # Add blank before RHref.
        &output (sprintf ($o_rhref, $fpage, $fline, $tpage, $tline));
        &output ("h e $word\n");
        &output ($c_rhref);
        &nextline;
        1;
    } else {
        0;
    }
}

sub try_departure {
    # Copy blanks
```

```
    while ($type eq 'b') {
        &output ($line);
        &nextline;
    }
    # Try start tag
    if ($word eq '&.sl') {
        &output ("h b 20\n");   # Add blank before departure
        &output ($o_departure);
        &nextline;
        # Copy SGMLized until end or ';'
         while (($word ne ';') && ($word ne '&xx{')) {
            (&try_abbr
             || &try_comm
             || do {
                 &output ($line);
                 &nextline;
             }
             );
        }
        &output ($c_departure);
        if ($word eq ';') {
            &nextline;
        }
        1;
    } else {
        0;
    }
}

###### Main:

scan:
while ($line = &read) {

    ($line !~ /^h c &xx[123](\[c\])?((\xa3\xa3)|(>+))/) && do {
        print $line;
        next scan;
    };

    print $line;
    &nextline;
    &do_lemma;
    while ($word ne '&xx{') {
        &do_rdg;
        &do_witness;
        &try_rhref;
        &try_departure;
    }
    &flush;
}

&flush;
```

Das Programm ‘overlay’

```
#!/usr/bin/perl

# overlay: Replaces the intra document changes (overlay) with
# corresponding SGML Tags.

@reptable = ("\\<",
            "s <del type=cc>\n",
            "\\>",
            "s </del>\n",
            "", "",

            "\\<\\|([1-4])",
            "s <overlay>\ns <del>\n",
            "\\>",
            "s </del>\ns </overlay>\n",
            "", "",

            "\\|([1-3])",
            "s <overlay>\ns <add>\n",
            "\\|%s\\{",
            "s </add>\ns </overlay>\n",
            "", "",

            "\\|1\\(\\*\\)",
            "s <overlay position-inferred>\ns <add>\n",
            "\\(\\*\\)\\|1\\{",
            "s </add>\ns </overlay>\n",
            "", "",

            "\\|([1-3])\\<",
            "s <overlay>\ns <del>\n",
            "\\>",
            "s </del>\ns <add>\n",
            "\\|%s\\{",
            "s </add>\ns </overlay>\n",

            "\\|1\\(\\*\\*\\)\\<",
            "s <overlay content-supplied>\ns <del>\n",
            "\\>",
            "s </del>\ns <add>\n",
            "\\(\\*\\*\\)\\|1\\{",
            "s </add>\ns </overlay>\n",

            "\\|S\\<",
            "s <overlay ch10 scribe>\ns <del>\n",
            ">",
            "s </del>\ns <add>\n",
            "\\|S\\{",
            "s </add>\ns </overlay>\n",

            "\\|A\\<",
            "s <overlay ch10 author>\ns <del>\n",
            ">",
            "s </del>\ns <add>\n",
            "\\|A\\{",
            "s </add>\ns </overlay>\n",

            "\\|A\\(\\)",
            "s <overlay ch10 lost author>\ns <add>\n",
            "\\(\\)\\|A\\{",
            "s </add>\ns </overlay>\n",
```

```
            "", "",

            "\\|S",
            "s <overlay ch10 scribe>\ns <add>\n",
            "\\|S\\{",
            "s </add>\ns </overlay>\n",
            "", "",

            "\\|A",
            "s <overlay ch10 author>\ns <add>\n",
            "\\|A\\{",
            "s </add>\ns </overlay>\n",
            "", "",

            "\\<\\|S",
            "s <overlay ch10 scribe>\ns <del>\n",
            "\\>",
            "s </del>\ns </overlay>\n",
            "", "",

            "\\<\\|A",
            "s <overlay ch10 author>\ns <del>\n",
            "\\>",
            "s </del>\ns </overlay>\n",
            "", "",

            "p\\:j\\|1",
            "s <overlay>\ns <del>\ns <boundary gi=p>\ns </del>\ns </overlay>\n",
            "", "", "", "",

            "\\|9\\(\\*\\)",
            "s <overlay pseudo position-inferred>\ns <add>\n",
            "\\(\\*\\)\\|9\\{",
            "s </add>\ns </overlay>\n",
            "", "",

            "\\|([89])",
            "s <overlay pseudo>\ns <add>\n",
            "\\|%s\\{",
            "s </add>\ns </overlay>\n",
            "", "",

            );

$curtrigger = "";
$curvalue = "";

sub try_init {
    $entry = 0;
    $found = 0;

    while ($entry < @reptable) {
        if ($word =~ /^$reptable[$entry]$/) {
            $level = $1;
            $_ = $reptable[$entry+1];
            if ($reptable[$entry+2]) {
                push (@trigger, $curtrigger);
                push (@value,   $curvalue);
                $curtrigger = sprintf($reptable[$entry+2], $level);
                $curvalue =   sprintf($reptable[$entry+3], $level);
            }
            if ($reptable[$entry+4]) {
                push (@trigger, sprintf($reptable[$entry+4], $level));
                push (@value,   sprintf($reptable[$entry+5], $level));
```

```
            }
            $found = 1;
            last;
        }
        $entry += 6;
    };
    $found;
}

sub try_trigger {
    if ($word =~ /^$curtrigger$/) {
        $_ = $curvalue;
        $curtrigger = pop (@trigger);
        $curvalue   = pop (@value);
    }
}

 mainloop: while (<>) {

     if (!/^c /) {
         print;
         next mainloop;
     }

     $word = substr ($_, 2);
     chop $word;

     try_init || try_trigger;

     print;
 }
```

Das Programm 'toks2sgml'

```
#!/usr/bin/perl

# Perl-Script zur Ausgabe des Token-Files als SGML-Datei.
# Beruecksichtigt werden die Tags e, s, w und b, wobei Zeilentrennungen
# nur bei b erfolgen sowie nach Moeglichkeit vor und nach jedem SGML-Tag,
# oder wenn eine Zeile ueber 72 Zeichen lang wird.

$linemax=72;
$line = "";
$linelen = 0;
$lastblank = -1;
$wantsbreak = 0;
$word = "";
$wordlen = 0;

sub flush {
    $line =~ s/\s+$// unless $keepspace; # Delete trailing whitespace.
    if ($line) {                # Don't print empty lines.
        print $line, "\n";
    }
    $line = "";                 # Reset line state.
    $linelen = 0;
    $lastblank = -1;
}

sub break {
    if (!$line || ($lastblank == -1)) {
        return;
    }                           # Prevent empty lines.
    print substr ($line, 0, $lastblank), "\n";
    $line = substr ($line, $lastblank + 1);
    $linelen -= $lastblank + 1;
    $lastblank = -1;
}

sub opentag {
    &break;                     # Break before tag, to avoid lengthy lines.
    $line .= $word;
    $linelen += $wordlen;
    $keepspace = 1;             # If there is a space, keep it despite newline.
    $blankstate = 0;            # We can't start with a new line.
    $wastag = 1;                # SGML tag
}

sub closetag {
    &flush;                     # Break before </closetag> always.
    if ($blankstate) {
        $line .= ' ';           # If there was a blank before </closetag>,
        $linelen += 1;          # transfer it to beginning of next line.
    }
    $line .= $word;
    $linelen += $wordlen;
    $keepspace = 0;             # A newline does for a space.
    $blankstate = 0;            # Don't automatically linebreak.
    $wastag = 1;                # SGML tag
}

sub append {
    $line .= $word;
    $linelen += $wordlen;
    $keepspace = 0;             # Linebreak normally.
```

```
    $blankstate = 0;            # We are not in blank-area.
    $wastag = 0;                # Last word was not sgml
}

sub dospace {
    if ($blankstate) {
        return;                 # Mehrere spaces nacheinander.
    }
                                # Zeile reif fuer Umbruch?
    if ($wastag || ($linelen > $linemax)) {
        if ($lastblank == -1) { # Gibt's einen Umbruchpunkt?
            &flush;             # Nein: dann ueberlang raus.
            return unless $keepspace;
        } else {                # Ja: Dann dort brechen
            &break;
        }
    }
    $linelen && ($lastblank = $linelen);
    $line .= " ";
    ++$linelen;
    $keepspace = 0;             # No need to keep more than one.
    $blankstate = 1;            # We are in blank area.
    # wastag is kept over blank area!
}

@monthnames = ('Jan', 'Feb', 'Mar', 'Apr', 'May', 'Jun',
               'Jul', 'Aug', 'Sep', 'Oct', 'Nov', 'Dec');
sub header {
    local (@timeinfo);

    @timeinfo = localtime (time);

    print "<!- Automatic Conversion Output ->\n";
    printf ("<!- Generated on %s %d, %d at %.2d:%.2dh ->\n",
            $monthnames[$timeinfo[4]], $timeinfo[3], 1900 + $timeinfo[5],
            $timeinfo[2], $timeinfo[1]);
}

# MAIN

&header;
 mainloop: while (<>) {
     ($type, $word) = split (/ /, $_, 2);
     if (($type eq 'n') || ($type eq 'h') || ($type eq 't')) {
         $mode = $type;
         ($type, $word) = split (/ /, $word, 2);
     } else {
         $mode = '';
     }

     chop $word;
     $wordlen = length ($word);

     ($type eq "b") && do {
         &dospace;
         next mainloop;
     };
     (-1 != index ("wpez", $type)) && do {
          &append;
          next mainloop;
      };
     ($type eq "s") && do {
         if ($word =~ /\<\// ) {
             &closetag;
```

```
            } else {
                &opentag;
            }
            next mainloop;
        }
    }
   &flush;
```

James Joyce

JAMES JOYCE: *Ulysses. A critical and synoptic edition.* Herausgegeben von Hans Walter Gabler mit Wolfhard Steppe und Claus Melchior. Zweiter, korrigierter Nachdruck. Garland Publishing: New York & London, 1986.
(Der Anlaß des ganzen Unternehmens. Das Nachwort von Hans Walter Gabler enthält Erläuterungen zur Textgeschichte und zum Editionskonzept.)

JAMES JOYCE: *Ulysses. A Facsimile of the Manuscript / The Manuscript and first Printings compared.* 3 Bände. London, 1975.
(Die ersten zwei Bände sind eine photomechanische Wiedergabe des sogenannten 'Rosenbach-Manuskripts' des *Ulysses*, der dritte Band enthält einen verkleinerten Abdruck der Erstausgabe, in den die Abweichungen vom Manuskript handschriftlich eingetragen sind.)

DON GIFFORD WITH ROBERT SEIDMANN (HRSG): *Ulysses annotated.* Zweite, erweiterte Auflage. Berkeley, Los Angeles, London: 1988.
(Eine nützliche Lesehilfe zum Ulysses; Zu jeder Episode wird ein Stadtplanausschnitt und eine generelle Einführung gegeben, daneben gibt es sachliche Erklärungen zu möglicherweise unklaren Stellen, unter Verwendung des Gablerschen Lesetext-Referenzsystems.)

Typographie

ROBERT BRINGHURST: *The Elements of Typographic Style.* Hartley & Marks: Vancouver, 1992.
(Das beste Handbuch zu Schrift und Buchgestaltung, das ich kenne. Es hat meine Wahrnehmung für die typographischen Phänomene im *Ulysses* geschärft.)

JAN TSCHICHOLD: *Ausgewählte Aufsätze über Fragen der Gestalt des Buches und der Typographie.* Zweite Auflage. Birkhäuser: Basel, 1987.
(Dieses wortgewaltige Buch ergänzt durch seine Aufmerksamkeit für Details und seine stärker traditionsorientierte Sichtweise Bringhursts Buch in einigen Punkten; vor allem aber verdanke ich ihm das deutsche typographische Vokabular.)

JAN TSCHICHOLD: *Erfreuliche Drucksachen durch gute Typographie.* Lizenzausgabe des Originals von 1960. Maro-Verlag: Augsburg, 1988.
(Auch in diesem Buch vermittelt Jan Tschichold die klassischen Traditionen der Typographie, auch für Alltagsanwendungen, mit hochgesteckter Meßlatte und ohne faule Kompromisse. Von seiner Praxisnähe und Aufmerksamkeit fürs Detail habe ich besonders profitiert.)

SGML und TEI

CHARLES GOLDFARB: *The SGML Handbook.* Clarendon Press: Oxford, 1990.
(Die definitive SGML-Referenz, die den ISO-Standard in gut strukturierter Form wiedergibt und kommentierend ergänzt. Ausserdem halte ich es – hier gehöre ich wohl einer Minderheit an – für eine sehr gute Einführung in SGML.)

CARL-MICHAEL SPERBERG-MCQUEEN, LOU BURNARD (HRSG): *Guidelines for Electronic Text Encoding and Interchange (TEI P3).* Chicago & Oxford, 1994.
(Das zweibändige *Green Book* der Text Encoding Initiative beschreibt die TEI DTD und gibt die semantische Definition ihrer Elemente. Es ist nicht nur als Referenz unumgänglich gewesen, ich verdanke ihm auch einige Anregungen zu differenzierter semantischer Codierung von elektronischen Texten.)

'`tei-l@uic.edu`' (Verwaltung durch '`listserv@uic.edu`'), die von den Herausgebern der *Guidelines* moderierte *mailing list* zu Fragen der TEI-Codierung. Dieses Gesprächsforum für die auf der Welt verteilten Anwender der TEI Standards war mir eine unschätzbare Hilfe. Besonders Carl Michael Sperberg-McQueen, der mich bei einem Besuch in Tübingen auch persönlich beraten hat, schulde ich großen Dank.

Editionswissenschaft

HERBERT KRAFT U.A.: *Editionsphilologie.* Wissenschaftliche Buchgesellschaft: Darmstadt, 1990.
(Eine gut lesbare und gegliederte Monographie über verschiedene Aspekte der Editionswissenschaft, die sich mit unterschiedlichen Ansätzen kritisch auseinandersetzt. Drei Kapitel stammen von Fremdautoren, darunter auch ein Beitrag 'Edition und Datenverarbeitung' von Wilhelm Ott, der an der elektronischen Betreuung der *Ulysses*-Buchausgabe maßgeblich beteiligt war.)

RICHARD FINNERAN (HRSG): *The Literary Text in the Digital Age.* The University of Michigan Press: Ann Arbor, 1996.
(Eine interessante Aufsatzsammlung, die das momentane Verhältnis der Editionswissenschaft zur Computeranwendung umreißt. Die Palette reicht von konkreten Erfahrungsberichten bis zu allgemeinen Zielentwürfen und der Fortführung grundsätzlicher Debatten der Editorik auf dem Feld der Computeredition.)

JOHN LAVAGNINO: ' Completeness and Adequacy in Text Encoding.' In: *The Literary Text in the Digital Age.* (siehe oben)
(Der Aufsatz betont, daß keine Wiedergabe eines Textes wirklich alle Aspekte eines Originals wiedergibt, weist auf theoretische Strömungen hin, die sich um größere Historizität in der Editorik bemühen und stellt deutlich heraus, daß das *encoding* eines elektronischen Textes immer auch schon ein *editing* ist.)

JEROME MCGANN: ' The Rosetti Archive and Image-Based Electronic Editing.' In: *The Literary Text in the Digital Age.* (siehe oben)
(Der Aufsatz beschreibt einen stark am Textzeugen, und zwar am Faksimile-Textzeugen, orientierten Ansatz für eine elektronische Edition. Er spannt einen interessanten Bogen von der textphilosophischen Diskussion zur konkreten technischen Realisierung.)

KLAUS KANZOG: *Prolegomena zu einer historisch-kritischen Ausgabe der Werke Heinrich von Kleists.* München, 1970.
(Führt den Begriff der bloßen 'Archiv-Edition' gegenüber einer 'historisch-kritischen Edition' ein, eine Unterscheidung, die meiner Ansicht nach in den Zeiten der Computeredition neue Aktualität bekommen hat.)

Software

WINFRIED BAADER (BEARB.): *Lernbuch TUSTEP. Einführung in das Tübinger System von Textverarbeitungsprogrammen.* Eberhard-Karls-Universität: Tübingen, 1995.
(Das TUSTEP-System war das Werkzeug zur Produktion der gedruckten *Ulysses*-Edition. Das Buch bietet einen gut lesbaren Überblick über die Grundfunktionen des Systems.)

DONALD KNUTH: *The TEXbook.* Elfter, revidierter Nachdruck. Addison-Wesley: 1991.
(Dieses Buch hat für den Inhalt der Diplomarbeit keinerlei Rolle gespielt, für ihre Form aber war es unverzichtbar. Außerdem war sein präziser aber informeller Stil und die sehr 'hardwarenahe' Sichtweise der Dokumentverarbeitung eine angenehme Abwechslung zur kühlen Abstraktion von SGML.)

LESLIE LAMPORT: *Latex. A Document Preparation System. User's Guide and Reference Manual.* Zweite Auflage. Addison-Wesley: 1994.
(Leslie Lamport ist der Erfinder von `latex`, einem Makropaket für das Satzsystem TEX. In `latex` wird großer Wert auf eine, SGML nicht unähnliche, logische Auszeichnung gelegt, gewissermaßen durch freiwillige Selbstbeschränkung. Dennoch bleibt es im Wesentlichen ein parametrisierbares Satzsystem.)

JOHN LEVINE, TONY MASON, DOUG BROWN: *lex & yacc.* Zweite Auflage. O'Reilly & Associates: 1992.
(Wie sehr viele Bücher dieses Verlags ein gutes und herstellerunabhängiges Handbuch; die wenig sinnvolle alphabetische Systematik der Beschreibung von `lex` und `yacc` wird aufgewogen durch die vielen Beispiele und den Vergleich verschiedener Implementierungen der Programme.)

LARRY WALL, RANDAL SCHWARTZ: *Programming perl.* O'Reilly & Associates: 1992.
(Larry Wall ist der Erfinder von `perl`, und der Stil des Buchs ähnelt dem der Programmiersprache: "in general they do what you want, unless you want consistency." Das Buch enthält aber nützliche Hinweise und einen ziemlich systematischen Referenzteil. Diese Ausgabe behandelt noch nicht die Version 5 von perl, die objektorientierte Ansätze aufweist und zum Beispiel für das `sgmls.pl`-Tool von David Megginson benötigt wird.)

Sonstiges

ROBERT MUSIL: *Der literarische Nachlaß, hrsg. von Friedbert Aspetsberger, Karl Eibl und Adolf Frisé.* Rowohlt: Reinbek, 1992.
(Ein Beispiel für eine Edition, die nur in elektronischer Form veröffentlicht wurde, mit großer Sorgfalt erstellt ist, und deren entscheidendes Handicap aus heutiger Sicht die Verschwisterung mit einem bestimmten softwaretechnischen Entwicklungsstand ist, der schon zum Zeitpunkt des Erscheinens nicht mehr aktuell war.)

FREDERIC BROOKS: *The Mythical Man-Month. Essays on Software Engineering.* Korrigierter Reprint. Addison-Wesley: 1982.
(Dieses Buch handelt davon, wie man große Softwareprojekte plant und durchführt, und wie nicht. Es hätte sich in einigen Aspekten auf mein Projekt übertragen lassen, wie mir zum Ende hin schmerzhaft klar wurde.)