AUTOMATIC DISCOVERY OF LOGICAL DOCUMENT STRUCTURE

A Dissertation

Presented to the Faculty of the Graduate School

of Cornell University

in Partial Fulfillment of the Requirements for the Degree of

Doctor of Philosophy

by

Kristen Maria Summers

August 1998

AUTOMATIC DISCOVERY OF LOGICAL DOCUMENT STRUCTURE

Kristen Maria Summers, Ph.D.
Cornell University 1998

The availability of large, heterogeneous repositories of electronic documents is increasing rapidly, and the need for flexible, sophisticated document manipulation tools is growing correspondingly. These tools can benefit greatly by exploiting *logical structure*, a hierarchy of visually observable organizational components of a document, such as paragraphs, lists, sections, *etc.* Knowledge of this structure can enable a multiplicity of applications, including hierarchical browsing, structural hyperlinking, logical component-based retrieval, and style translation.

Most work on the problem of deriving logical structure from document layout either relies on knowledge of the particular document style or finds a single flat set of text blocks. This thesis describes an implemented approach to discovering a full logical hierarchy in generic text documents, based primarily on layout information. Since the styles of the documents are not known *a priori*, the precise layout effects of the logical structure are unknown. Nonetheless, typographical capabilities and conventions provide cues that can be used to deduce a logical structure for a generic document. In particular, the key idea is that analyses of the text contours at appropriate levels of granularity offer a rich source of information about document structure.

The problem of logical structure discovery is divided into problems of *segmentation*, which separates the text into logical pieces, and *classification*, which labels the pieces with structure types. The segmentation algorithm relies entirely on layout-based cues, and the classification algorithm uses word-based information only when this is demonstrably unavoidable. Thus, this approach is particularly appropriate for scanned-in documents, since it is more robust with respect to OCR errors than a content-oriented approach would be. It is applicable, however, to the problem of analyzing any electronic document whose original formatting style rules remain unknown; thus, it can provide the basis for flexible document manipulation tools in heterogeneous collections.

BIOGRAPHICAL SKETCH

Kristen Summers attended Amherst College, where she majored in computer science and English. She recieved a B.A., magna cum laude in computer science and magna cum laude in English, in May of 1991. She received an M.S. in computer science from Cornell University in May of 1994.

For Mark

ACKNOWLEDGMENTS

I am very grateful to John Hopcroft for his invaluable guidance and support over the years during which this thesis took shape and also to Daniela Rus and Jim Davis for theirs. Many thanks are due Dan Huttenlocher, Lillian Lee, and Tim Murray for careful readings of early drafts and greatly beneficial insights and advice. This research was facilitated by a National Physical Science Consortium Fellowship, with stipend support from the National Security Agency.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

Chapter 1

Introduction

The availability of large, heterogeneous repositories of electronic documents is increasing rapidly. As good internet navigation and resource discovery tools are developed, the text files accessible to a user through the internet will come to form an implicit distributed collection of documents in much the same way as other files form an implicit database collection [90]. Documents designed specifically for such use are likely to be provided in a structured form; older documents and those written *primarily* for paper are not.

As the quantity of available information grows, however, so too does the probability of overwhelming a user; in order to use the internet document collection effectively, users will require sophisticated tools for manipulating its elements. Users will require search tools for filtering information and navigation tools for exploring heterogeneous data, both within and among documents.

Many such useful document manipulation tools can be enabled by a knowledge of the logical structure of a document (defined in Section 1.1); the popularity of markup systems, such as SGML [37] and its application HTML, and structured documents reflects a recognition of this power. (The use of HTML is a bit problematic; although its official *intent* is to describe structure [93], many document providers misuse its structures in order to achieve a layout formatting on the browsers they expect their readers to use [31, 72]. This creates a special interpretation problem.) Unlike layout structure, which is intrinsically present in that it can be meaningfully described automatically, logical structure must either be *imposed* by markup or similar means or *discovered* by document analysis.

This thesis explores the problem of discovering logical structure in text documents in the presence of little or no style information, based primarily on the shapes of blocks of text. It describes and evaluates an approach that proceeds from a few key observations about the significance of shape and spacing, which are discussed in Section 1.3. This approach is implemented in the LABLER (LAyout-Based Logical Entity Recognizer) system. It is differs from previous approaches to logical structure discovery (as discussed in Chapter 2) in that its goal is to find detailed logical structure hierarchies for documents of unknown and possibly varying styles. This goal leads to several differences in approach from previous systems, including the reliance on the aforementioned observations, feedback between separate processes for segmentation and classification, and the incorporation of available style knowledge without requiring it.

Section 1.1 defines logical structure as the term is used in this thesis and the related literature. Section 1.2 details some of the applications enabled by the knowledge of logical document structure. Section 1.3 presents the observations about general structure representation on which LABLER is based. Section 1.4 summarizes the approach of LABLER.

1.1 Logical Structure

Electronic document manipulation tools can benefit greatly from exploiting an understanding of multiple views of document structure, especially the relationship between layout and logical structures. This section discusses some general uses of such information; Section 2.5 discusses several examples of specific research in this area. For example, consider a scanned-in document stored as a set of files, each representing a page, on each of which optical character recognition (OCR) has been performed.[1] Here, physical layout structure is represented directly, at the character and page level. To represent logical structure, the document might be divided into sections, rather than pages; such an indexing scheme allows, for example, retrieval of sections relevant to a query, with relevance determined by traditional techniques of Information Retrieval.

The full logical structure forms a hierarchy of visually observably separate semantic components of the document. An example of the logical structure for a technical paper is given in Figure 1.1. This structure lies at the intersection of content and layout. More precisely, it can be defined as follows.

Definition 1.1 *The* logical structure *of a document consists of a hierarchy of segments of the document, each of which corresponds to a visually distinguished semantic component of the document. Ancestry in the hierarchy corresponds to containment among document components.*

The layout requirement that a segment must be *visually distinguished* requires some non-content-based indication that this piece of text belongs together, separate from its surroundings;[2] thus, it should be possible to identify the segment as a potential component without an understanding of the words of the document. The content requirement that a segment be a *semantic component* requires that it have meaning as a separate piece within the content of the document; thus if the document were translated into an utterly different style, the piece of text would still be identifiable as a segment separable from its surroundings. This structure is identified as the *functional level* of a document in [24].

[1] This is a current form of representation of Cornell's computer science technical reports, using output from Xerox's ScanWorX optical character recognition software.

[2] This is a requirement of the *definition* used throughout this thesis, and it functions as an implicit requirement of the logical structure discovery systems discussed in Chapter 2, as well. It does *not* mean that all items that might correspond to an intuitive notion of a logical component will necessarily have this attribute.

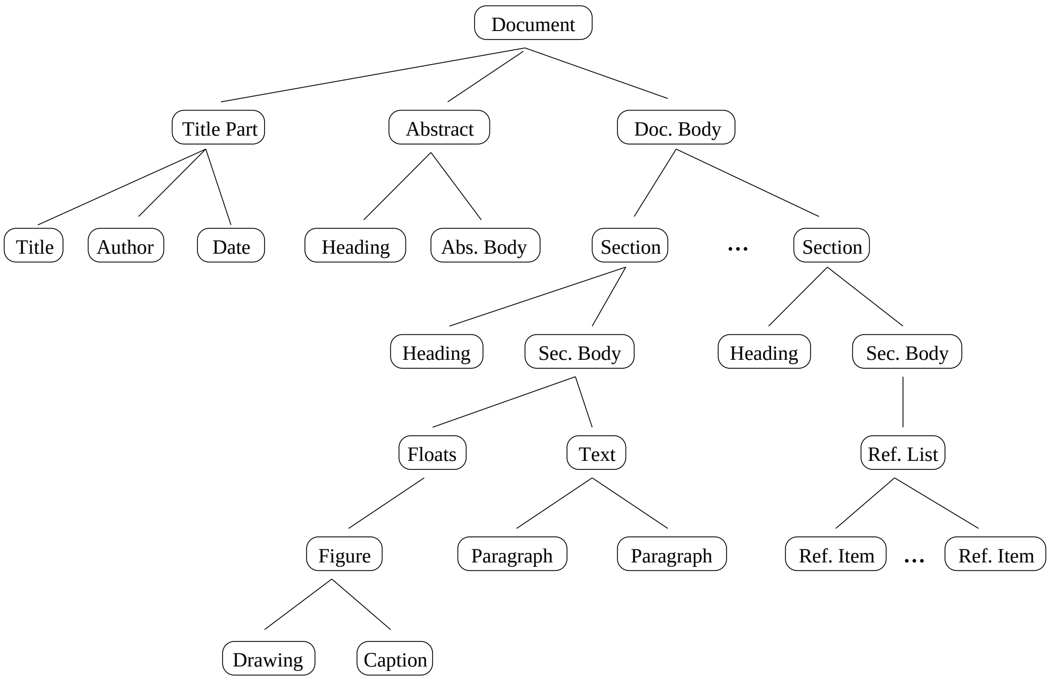


Figure 1.1: A logical structure tree

Logical components generally correspond to organizational pieces of a document, as organization will typically be driven by content considerations and also reflected in layout design. For instance, a section should be both topically cohesive and visually identifiable by its heading. LABLER finds logical hierarchies that are also *complete* in the sense that, with the exception of floats, the concatenation of the document components corresponding to the children of a node forms the document component corresponding to that node.

In an instance of term overloading, the document components that correspond to the nodes of the hierarchy are also called logical structures. For example, any particular paragraph of the document whose full logical structure is given in Figure 1.1 is itself a logical structure of the document.[3]

Other interesting text segments can be found, such as topically cohesive passages or italicized portions, but these do not form logical components; the former is content only, and the latter is layout only. In most cases, the layout of a document is interesting primarily because it represents organizational structure. (An exception arises if the style of the document is under consideration, rather than its message.) Disregarding pure layout thus restricts our attention to structures whose significance is fairly clear. Purely content-based components

[3]These meanings of logical structure are distinguished by the article; "*the* logical structure" of a document is its hierarchy, and "*a* logical structure" of the same document refers to some particular component. The term "logical structure" remains overloaded.

are often interesting in their own right. Identifying them requires significant linguistic analysis and subjective judgement about subject matter,[4] and there is no reason to believe they will consistently form a hierarchy in any given document. Finding and managing such components is an important and interesting task, elements of which are addressed, for example, in [40], [54], and [82]; it is also a very different kind of task from finding and managing logical components, and this thesis addresses the latter.

Non-hierarchical relationships do exist between logical components of a document, such as the relationship formed by references within the text, but these are not part of the logical structure in the sense used here.

1.2 Logical Structure Uses

The popularity of markup systems like SGML reflects the usefulness of this kind of information [94]. In addition to the direct use of SGML (advocated for many purposes in [28], [62], and [94], for instance) and structured documents in other forms (as discussed in, for example, [41], [63], and [80]), this popularity is exemplified by some of the suggestions for the direction of HTML and the World Wide Web. HTML 4.0 emphasizes logical structure; visual formatting is meant to be handled by style sheets and browsers, and the visual commands of earlier versions are deprecated [73]. There is also an interest in subsuming HTML by XML [32], which makes far more use of the power of SGML, as discussed in [31]. Taking this line of thought to its logical conclusion, van Ossenbruggen *et al.* argue for using SGML itself, with stylesheets, on the Web [84].

If an instantiation of a document is available with markup in a known language, then the logical hierarchy is directly available, and it can be used to manipulate the document as described below and in Section 2.5. (The logical structure may not be distinguished from other structures by the markup, but this should not interfere with the usability of the included logical structure information.) Alternatively, if the document is only available in a layout-based format, such as the results of scanning and Optical Character Recognition (OCR) or a Page Definition Language (PDL) such as PostScript, the problem of discovering the logical structure arises.

The automatic discovery of logical document structure can enable a multiplicity of electronic document tools, including (1) automated markup, (2) structural hyperlinking as discussed in [2], (3) hierarchical browsing as discussed in [2] and [14], and (4) logical component-based retrieval as discussed in [77]. The browsing and retrieval thus enabled are relevant to two of Croft's top ten research issues for Information Retrieval: interfaces and browsing, and efficient, flexible indexing and retrieval [20]. (In the latter issue, logical structure knowledge increases flexibility, not efficiency.)

Logical markup, in SGML or another format, can proceed based on logical structure directly. For example, to generate a document instantiation with SGML tags, perform a

[4] An interesting exception to the rule that identifying content-based elements requires content-based information is found in [15], which applies Information Retrieval techniques to word *shapes*.

depth-first traversal of the tree. On entering a node, generate a start tag for its structure; on exiting, generate an end tag. At a leaf, generate the appropriate text between these tags. Of course, if some logical structures are not currently of interest, their nodes need not generate tags.[5]

Hyperlinking can be automated by defining links to exist between certain kinds of structures under given sets of conditions. For instance, within a set of documents, links can be provided between bibliography entries and the documents to which they refer in the following manner. If a list item occurs within a section whose heading matches "References" or "Bibliography" or "Works Cited," then consider this item to be a bibliography entry. If this entry has a substring that matches the title of another document, create a link between the list item and that document.

Browsing of a document can proceed by exploring the logical structure tree; this can be enabled by representing a document as appropriate hypertext nodes and links for currently available browsers, or a browser could provide this approach directly. As this kind of browsing is really only appropriate for a fairly large document, the former approach seems preferable; the responsibility of deciding how much structural navigation is reasonable lies with the document provider. If a summarized form is defined for each type of logical structure, a user may browse the tree by starting at the root and repeatedly deciding which, if any, nodes to expand. The initial view would be of a summary of the root document node, perhaps its title. This would expand to show summaries of its children, which might be a title part, an abstract, and a body. The user might then read the abstract or perhaps expand the body to reveal summaries of its sections. Each step reveals a bit more about the document and would help both a user who must decide whether to read the document and one who is seeking a particuar part of it. An example of this kind of browsing is given in Chapter 9. Note also that the common HTML device of providing a table of contents, each entry of which links to the corresponding document part, is a version of this approach, based on a tree of two levels.

Alternatively, browsing could be enabled by defining a series of document views of increasingly fine granularity, rather than allowing the user to explore the hierarchy at will. One such series might consist of the title only, then the information in the title part (including author, *etc.*), then this information and the abstract, then the section headings, then the entire document. Navigating a document via structural views is discussed in [5].

Retrieval can be based in whole or in part on the existence of and relationships between logical structures. For instance, a user might wish to retrieve all the theorems from a paper or all the sections containing theorems. This can, of course, be combined with classical Information Retrieval techniques such as those in [33, 40, 78] to yield results that are based on both content and logical structure, such as all the theorems in sections relevant to a given topic.

As an example of logical structure use, LABLER generates a document instantiation in HTML. This result can be browsed by any World Wide Web navigator, such as Mosaic or

[5] Also, any needed non-logical structures will, of course, have to be identified and tagged in some other way.

Netscape.[6] Each logical structure has its own page, together with a summary of its position in the hierarchy and descriptive links to its parent, children, and siblings. In this way, the document can be explored via its logical structure tree; at any point it may simply be read, as well.

1.3 Generic Structure Representation

The particular mapping of logical structure to layout presentation varies with document style. If the precise style of a document is known, the logical structure may be inferred from the layout representation of the expected forms. Without such information, more general observations are required. The approach to generic structure discovery explored in this thesis relies on a few observations about the usual manners of representing logical structure.

Since the vast majority of documents must be represented comprehensibly in markings of a single color laid out on a two-dimensional space (*i.e.*, pages), vertical and horizontal spacing play a great role in the representation of logical structure. Moreover, most documents are designed to make the structure easily apparent to human readers, and they use visual cues in fairly consistent ways. This leads to the following observations, on which LABLER is based. Other document structure discovery systems, as discussed in Chapter 2, make use of the general observation 1 but not the particulars of 1a and 1c; some of them implicitly rely on 2, but they do not state it explicitly.

1. The shape and spacing of regions of text provide significant structure cues.
   (a) Structures typically appear multiple times, yielding repeated shapes.
   (b) Structures are typically separated by white space, vertical (blank lines) and/or horizontal (indentation).
   (c) A structure that splits another into unusual parts typically occurs horizontally within the larger structure.
2. Common structures typically have similar representations across document styles. The different representations can be seen as variations on a theme.

The list above forms an example of the phenomenon described in 1c. It separates this paragraph into parts, and its left and right margins fall within those of the surrounding paragraph.[7]

[6] The output uses the `frames` structure of HTML but it also provides a `<noframes>` alternative.

[7] For these purposes, block of text x with the same right margin as another block y, but with a left margin that falls within that of y, falls horizontally within y.

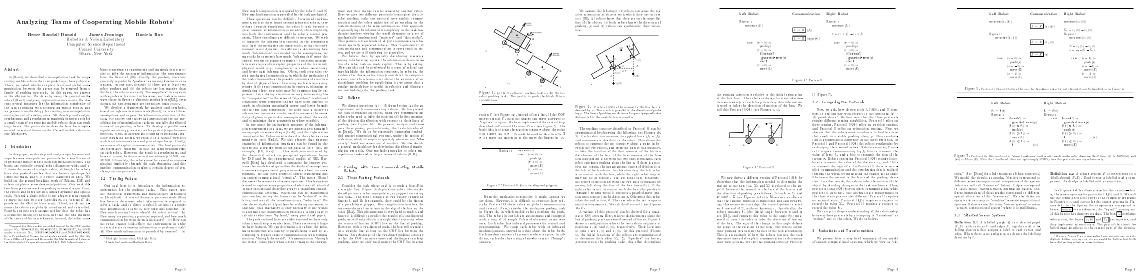


Figure 1.2: A zoomed-out view of a document. The paragraphs, lists, title area, and figures are all identifiable, although the text is not legible.

1.4 LABLER's Approach

In its search for logical structure, LABLER privileges geometry, based on the above observation that shape is an important indicator of logical relationships, through indentation, proximity, and repetition. As a result, the logical structure of a document can be determined, to a large extent, without reference to the text itself. That is, consider a *zoomed-out* view of a document, in which the layout shapes are still visible but the text is not, as shown in Figure 1.2. The portions of the document that belong together can be determined, and in many cases the type of structure can also be identified.

The input to the system is a document view with lines of text and their positions and heights identified. If font information is available, the system uses that as well. No description of the document style is required, but any such available knowledge is incorporated into the process of finding logical structure. For a scanned-in document, the processing has the form shown in Figure 1.3; the heavy boxes are steps taken by this system.

The problem of deriving the logical structure of a document from its inherent layout structure can be divided into two stages: (1) *segmentation* of the document into a hierarchy of logical elements and (2) *classification* of the nodes of the hierarchy according to the type of logical structures they represent. These processes affect each other, but they occur separately. LABLER is unique in that it both keeps the processes separate and allows feedback between them.

The approach described here uses contour- and font-based information only for segmentation, based on the expectation that structures will be repeated and/or will be indicated by vertical proximity or horizontal containment. LABLER alternates searches for repetitions of shape with searches for relevant horizontal containment; vertical proximity is considered when these do not produce readily classifiable segments. Other cues are introduced for classifying the segments as needed, and the classifier results provide feedback for the segmenter. This distinguishes LABLER both from systems that consider only layout, in that it can identify structures that they cannot, and from systems that rely on domain-specific interpretations of text words or characters (such as that in [53]), in that its approach is

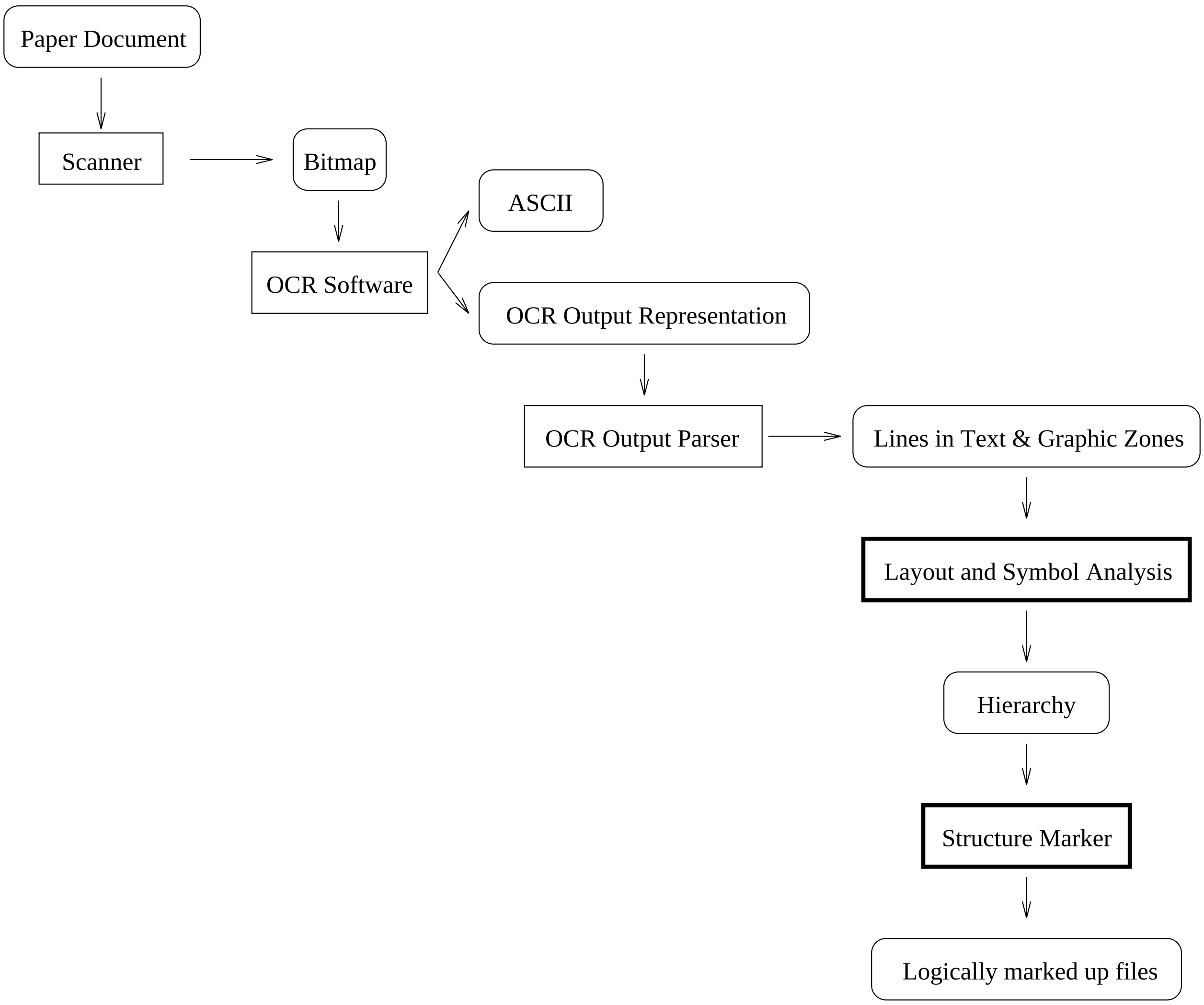


Figure 1.3: Document analysis steps, with those of the logical structure discovery system in heavy boxes.

more generally applicable. If some constraints on the layout representation of the logical structure are known, these are incorporated as well. This distinguishes LABLER both from systems that consider no style knowledge and those that are based entirely or primarily on such information.

1.5 Outline

This thesis is organized as follows. Chapter 2 discusses previous work on related problems. Chapter 3 describes a taxonomy of logical document structures and the implications of the characteristics of different types for their discovery and use. Chapter 4 provides an overview of the logical structure discovery system. Chapter 5 describes the process of segmentation in detail, Chapter 6 describes classification, and Chapter 7 describes the incorporation of specific knowledge. Chapter 8 discusses the evaluation and performance of this approach. Chapter 9 presents an application of the results for browsing (mentioned in Section 1.2), and Chapter 10 offers conclusions and discussion.

Chapter 2

Background and Related Work

The need for logical structure information is implicit in the history of hypertext. Many standard types of hyperlinks require this information, such as links from a table of contents to sections of text, or links from references to the texts to which they refer. The specific problem of automatically discovering this structure from documents designed only for layout-based presentation, however, began to acquire attention relatively recently, as the quantity of available online documents became vast enough to require sophisticated tools for searching, browsing, and filtering.

2.1 Flat Segmentation

Many researchers have concerned themselves with the problem of determining a flat set of document divisions, rather than a hierarchy. This decomposes the document into text blocks, which may then be assembled into a reading order or analyzed to determine their content. For instance, this is the role of logical structure in [65], an outline of a text reading system. Logical structure also plays this role in [66], a system that derives reading orders for newspaper pages.

One form of this task is to separate the text from the graphics in a document. Other forms of analysis can then be applied to the resulting blocks, as appropriate. Wong, *et al.* present a method of making this separation in [92]; Fletcher and Kasturi present another in [30]. Both of these approaches are based on analysis of pixel density. Antonacopoulos and Ritchings present a method based on an analysis of connected white rectangles in [3]. Jain and Yu describe a method in [49] based on grouping aligned runs of black or white pixels and then performing an analysis of height, density, *etc.* Witten *et al.* provide a general overview of this particular problem in [91].

Within the text domain, a document can be divided into blocks at the paragraph level and/or blocks that are specific to a particular document type. Several approaches exist to finding such particular block types. For instance, Jain and Bhattacharjee find address blocks on envelopes [48]; Rahgozar *et al.* find the grid structure in tables [74]; Belaïd *et al.* decompose card catalog entries [10]; Watanabe and Huang decompose business cards [88];

Hao *et al.* find standard parts of office memos [39], and Bayer and Walischewski find similar elements of business letters [9]. Watanabe *et al.* describe several often useful techniques for this kind of task in [89], and Frankhauser and Xu present a general methodology for finding many such specific blocks in [27].

Walischewski presents an approach that learns a document block model, represented as an attributed DAG consisting of a tree with the addition of sibling edges, by including in the model anything seen in the sample set; conflicting possibilities are given probabilities based on their observed frequencies [86]. The node attributes are the vertical and horizontal projections of bounding boxes, and the edge attributes are the spatial relationships between the connected nodes, selected from a complete set of qualitative descriptions. A new image is analyzed by matching it to the tree so as to minimize its overall deviation from the model. This approach can be extended and applied to full document hierarchies, but the implementation described only divides business letters and addressed envelopes into flat sets of blocks.

Baird and Ittner have developed a general approach to segmenting documents into blocks of text that can be read together and are delimited by white space (*i.e.*, each block has the same orientation and has appropriate spacing for reading as a sequence of lines, and blocks are separated by extra white space) [7, 47]. This system uses an analysis of the shape and density of page markings to generate a set of candidate segmentations, with components of varying sizes. White rectangles that fall between text blocks of mid-range sizes are enumerated, sorted, and unified, generating a partial order of sets of *white covers*. In [7], these are compared, and the chosen segmentation separates components at approximately the level of paragraphs or paragraph groups (although a paragraph interrupter, such as a list or equation, is likely to form its own block). In [47], the first result found that satisfies a given stopping rule is chosen, and the results are similar. Lovegrove and Brailsford address this problem for PDF files in [61]; blocks are identified based on matching font size and margins and considering vertical white space.

Ha *et al.* propose a method for finding text lines and the blocks they form in [38], by identifying bounding boxes, then considering the vertical and horizontal projections of pixel density. These identify lines. Differences in within-line projections identify block breakpoints. Déforges and Dominique address the same problem differently in [21]. They identify lines by connection and blocks by uniformity of line height and slope. Ishitani presents an approach based on *emergent* computation; each current piece follows rules for merging with its neighbors based on proximity and similarity and rules for splitting itself based on perceived errors [46]. Poirier and Dagenais address the special case of finding lines and blocks for converting PostScript files to HTML in [70]. They separate the text into lines, then group these into blocks with similar font and spacing characteristics; these are assigned to HTML structures according to a set of predefined rules.[1]

[1]Note that this could involve building a full hierarchy, but the approach the authors chose was simply to identify block types; presumably, they include the higher-level structures, such as lists, that are entailed by their block classifications.

2.2 Document Layout Parsing

Several researchers have proposed approaches to document structure recognition based on a form of parsing. This method applies if the *style* of a document is known, *i.e.*, the logical structure follows a known format, and it has been mapped to the layout structure according to a known set of rules. Typically, the rules to generate such a document can be represented as a Context-Free Grammar (CFG), and the document layout can then be parsed accordingly.

Fujisawa *et al.* describe precisely the above approach in [34, 35, 36, 95]; the logical structure is defined at the section and subsection levels and at the paragraph level, and the relationships among these structures and their layout representations are described by a CFG. The rules used are common to a large group of technical journals, so the results are applicable to articles from any of them. Porter and Rainero advocate a similar approach that uses tree manipulation rather than explicit parsing [71]. In this case, a more complete hierarchy is built for a narrower set of documents; the recovered structure includes such elements as *theorem* and *list*, and it is based on a knowledge of the precise style rules, such as those available if the document is known to have been formatted with the LaTeX `article` style.

Viswanathan *et al.* propose a top-down approach in [85], in which a particular set of attribute grammars is defined for each structure; applying such a set in sequence starts with input of the pixel projections (horizontal or vertical) for this structure and produces output of a representation of the substructures. Repeated application parses the document; the depth of the hierarchy depends on the choices in defining the *block grammars* for the identified structures, and the style must be known precisely enough to unambiguously identify all desired structures. Liu-Gong *et al.* use a similar approach, without explicit grammars and parsing. They model documents of a given style as a physical hierarchy, in which each node contains instructions for separating its contents into its children; the behavior is equivalent to parsing [60]. Finally, Azokly and Ingold in [6] define a language for describing how a document is separated at various levels; this provides appropriate information for deriving a hierarchy by repeated divisions.

The above type of approach is ideal if a precise, unambiguous style is known for the logical structures of interest. A few researchers modify this by using error-tolerant parsing; they are thus able to handle documents that do not conform precisely to the given specifications. This yields an approach appropriate if a style is known but some documents may not conform precisely, whether due to idiosyncratic authoring or to OCR errors.

Hu and Ingold use fuzzy parsing [43]; when a precise parse is impossible, the system chooses one or more near-matches, so as to minimize the total deviation from a correct parse tree, based on a predefined cost function. Thus a document that deviates slightly from the given style can be adequately analyzed. Klein and Fankhauser present a system of character-based parsing in which the user can specify fallback rules, geared to handling the types of errors expected for a given document type [53]. This approach is very sensible for contexts in which deviations from the usual style can be expected to take a small number of known forms. Chou and Kopec use an information-theoretic approach to find the logical parse tree most likely to be associated with the observed layout, given known probabilities of flipping black pixels to white and of flipping white pixels to black [16]. This approach rigorously handles the type of deviations expected from the OCR process; the effects of non-conformity at a higher level of the hierarchy are unclear.

Other researchers take the approach of building a system that will learn the necessary grammar from sample documents, thus allowing a precise parse while relieving the user from the necessity of forseeing all allowable layout forms. This is appropriate for analyzing a set of documents of a single style, which is either incompletely specified or simply unknown.

Akindele and Belaïd present a system that builds a tree grammar, based on two sets of samples [1]. It uses the first set to validate an initial grammar provided by the user and those documents of the second set that conform to the initial grammar are used to extend it to the full grammar used for parsing. Takasu, *et al.* infer a *matrix grammar*, in which concatenation may be either vertical or horizontal, by generalizing from a set of examples; any observed form is acceptable, and any adjacent structure repetition is generalized to accept the Kleene closure [42] of its form of concatenation [83].

Dengel and Dubiel describe a system that learns decision-trees for identifying the style of a business letter, which enables a parse [22]; this approach is based on considering the areas of the page covered with text, at three levels of granularity. Brugger *et al.* take the concept of n-grams from natural language processing and apply them to tree structures, based on 5 kinds of sequences: left sibling, right sibling, ancestor, first child, and last child [12]; their system learns from a training set the probability of encountering a structure based on its preceders in these sequences. (They use trigrams and thus sequences of length 3.) It starts from tree leaves that are based on the physical structure; in the given example, these are text lines.

Derrien and Habib and Esposito *et al.* provide exceptions to this general approach of parsing and its variations. Derrien and Habib build blocks from lines and classify them [23]. They use the interline distance as a guide for segmentation (*i.e.*, lines with less distance between them should be grouped together at a lower level of the hierarchy than lines with greater distance) and then apply known rules for classification. Esposito *et al.* segment documents into lines and two levels of text blocks [26]. Lines are formed from basic blocks based on proximity, overlapping, and similarity of type and height. Then, the first set of blocks is formed by first grouping together lines with sufficiently similar horizontal pixel projections, then grouping together line sets that are sufficiently nearly aligned and sufficiently vertically close, based on preset thresholds. The second group of blocks is formed by combining first-level blocks with overlapping horizontal pixel projections, left or right alignment, and a vertical distance less than the mean plus standard deviation for the first-level blocks. The levels appear to correspond to paragraph groups for level 1 and running text that belongs to the same article for level 2, but the authors do not specify the intended correct answer.[2]

[2] In particular, their blocks at level 1 group together multiple paragraphs but keep these separate from any centered lines surrounded by space, as such a line *may* be a new section heading; it may also be an equation that belong to the paragraph around it, however, in which case this result does not correspond to the usual understanding of logical structure.

Table 2.1: Summary of approaches to logical hierarchy derivation

Approach	Segment & Classify	Observables
Fujisawa *et al.*	Together	Many
Porter & Rainero	Together	Many
Viswanathan *et al.*	Together	Pixel Projections, Run Lengths
Liu-Gong *et al.*	Together	Variable
Hu & Ingold	Together	Many
Chou & Kopec	Together	Pixels
Klein & Fankhauser	Together	Characters
Dengel & Dubiel	Together	Bounding Boxes (Three Levels)
Brugger *et al.*	Together	Surrounding Structures
Derrien & Habib	Separate	Segment: Vertical White Space Classify: Many
Esposito *et al.*	Segment only	Pixel Projections, Line Edges, Vertical White Space
LABLER	Separate	Segment: Line Contours Classify: Many

2.3 Comparisons

This thesis addresses the problem of generic logical structure discovery; the goal is to build a full hierarchy for a document whose style remains unknown. The correctness of the result cannot be guaranteed, since indistinguishable layout formats may in some cases represent different logical structures. Some conventions of spacing and repetition are consistent enough to enable a reasonable approach, but this should only be used when necessary. That is, the extant algorithms for block segmentation and layout parsing described above clearly should be chosen if only a single set of divisions is required or if the document style is known *a priori* or can be reasonably expected to match precisely the styles in an available sample set. (The differences among the various approaches that build hierarchies are summarized in Table 2.1.[3]) To build a full hierarchy without knowledge of a fixed style, however, requires a more general and flexible approach, such as that described in this thesis. Derrien & Habib take an approach that can also be so used, but it seems unlikely to fare well on complex documents; straightforward use of vertical white space provides a baseline for comparison to LABLER's results in Chapter 8.

[3]An entry of "variable" indicates that the approach does not specify this, and it appears to depend entirely on the desired application.

Table 2.1 (Continued)

Approach	Matching	Levels
Fujisawa *et al.*	Precise	Sections & Paragraphs
Porter & Rainero	Precise	Many
Viswanathan *et al.*	Precise	Variable
Liu-Gong *et al.*	Precise	Variable
Hu & Ingold	Approximate	Many
Chou & Kopec	Approximate	Variable
Klein & Fankhauser	Precise (With Alternatives)	Variable
Dengel & Dubiel	Precise	Few
Brugger *et al.*	Probablistic	Variable
Derrien & Habib	Precise	Many
Esposito *et al.*	Within Thresholds	Two Levels
LABLER	Cues (Very Approximate)	Many

The research discussed below on structured document representation and manipulation provides a context for this logical discovery work; although these issues are not the current focus, the representation of results does maintain both logical structure and some layout structure, and a browsable document instantiation is created.

2.4 Logical Structure Representation

Some researchers in this area have turned their attention to the question of appropriate representation formats for the structure of a document.

Murata proposes a document model designed to provide easy access by both logical and layout structure [64]. The representation is based primarily on intermediate nodes that represent logical (layout) structures that have precisely corresponding layout (logical) structures. The necessary basic logical and layout nodes are included, and containment relationships among all the nodes are specified; together, this information provides an easily traversable description of a document in both logical and layout terms.

Clarke *et al.* discuss a representation that makes no distinction between various types of structure [17, 18]. Logical, layout, and content structures are all represented equivalently. With this model, a hierarchy is an inadequate representation, since two structures (*e.g.*, a page and a paragraph) may overlap without sharing a containment relationship. They recommend instead a marked-up version of the document, in which tags mark the beginning and end of each structure. The relationships the structures may have with each other are unconstrained. This freedom somewhat complicates document manipulation tasks, and the authors discuss in detail a model for retrieving specified structures.

Davis proposes an approach that circumvents some of the difficulty introduced by Clarke's approach; he recommends maintaining multiple, separate structure hierarchies.[4] Each leaf node in each hierarchy contains pointers to its beginning and ending positions in a single instantiation of the document itself. This model allows for well-defined hierarchies of an arbitrary number of kinds of structure, each of which may be manipulated independently without risking any unexpected side effects on other structural hierarchies. The trade-off is that it is not optimized for the combined use of multiple structure types, although such combinations can certainly be used.

2.5 Structured Document Manipulation

A good deal has been written about the kinds of document manipulation tools that are enabled by the knowledge of logical document structure. These include tools for browsing, hyperlinking, style translation, and retrieval.

Burrill's VORTEXT system provides a framework for document creation and browsing based on logical structure [14]. The user must specify logical structure upon creation of the text, so the problem of structure discovery is obviated. In browsing, a user can follow links to sections or subsections (like turning directly to a page from the table of contents in a book) and to references. Wang describes a system with a similar capacity, based on an object-oriented database [87]; Ayres and Wesley [5] and Bauwens *et al.* [8] describe similar approaches, particularly designed to increase document accessibility for the visually disabled.[5] This kind of browsing is based on a form of hypertext; logical structure information can provide a basis for creating multiple kinds of hyperlinks within and between documents. Allan *et al.* discuss in [2] some of these link types, based both on structure alone and on a combination of structure with Information Retrieval definitions of relevance.

Translation of document style from one layout format to another can be performed in a general way, based on knowledge of logical structure. An example of such style translation is to convert a document formatted for a given technical journal into a format appropriate for a Ph.D. thesis while maintaining an accurate depiction of the logical structure. This problem forms part of the motivation for Arnon's Scrimshaw language for document description [4], and Feng and Wakayama discuss the task in detail in [29], as do Kuikka and Penttonen in [56]. Raman's AsTeR system performs a particularly interesting form of document translation: it renders LaTeX documents in an aural style [75].

Logical structure can also provide a basis for retrieval, either of specified logical document portions or of entire documents, based on the structures they contain. Kay's Textmaster system, for example, uses this advantage to base retrieval on a combination of logical structure and the presence of keywords [52]. Scrimshaw, in addition to style translation, provides a

[4] James R. Davis, personal communication, 1994.

[5] In fact, in [97], a conference on computers for the disabled, all papers on documents focused either on the process of reading text aloud or on document structure.

framework for document queries based on logical structure. Pfeifer *et al.* have extended WAIS to index structures separately and allow structure specification in queries [69]. Kuikka and Salminen present a system that bases queries on specifications of structures and keywords they contain, presuming prior knowledge of a logical structure grammar, much like an SGML Document Type Definition (DTD) [57]. Kaneko and Makinouchi discuss the processing of structure-based document queries in an object-oriented database [51].

Some researchers have presented query mechanisms tailored specifically to structured documents. Lalmas, for instance, proposes a fetch-and-browse strategy for identifying the best document structure with which to satisfy a conventional IR query in [58]; if a document is deemed relevant, its structure is browsed, top-down, to find the most specific element of high relevance. Yeh *et al.* present a query language called SSQL for structured documents and an approach to use with it in [96]. The language looks like SQL but also allows specifications of document structure elements. The approach iterates over three steps. It starts with an SSQL query; the user may optionally peruse the results and prune them by hand to identify the elements of interest; then, the user may follow hypertext links, either following the hierarchy or using cross-links. Niyogi and Srihari describe a three-part query for a collection of documents of different (parseable) types in [67]. The first part specifies the document type, narrowing the range for the rest to documents of that type; the second specifies the structure type to consider, and the third specifies the characteristics it must have and which part of it to return. Finally, Buford discusses the usefulness of a particular query mechanism based in part on logical structure in [13].

Chapter 3

Logical Structure Types

The problem of discovering logical document structures raises questions about the nature of the structures themselves. The structures of interest can be categorized according to several criteria, including: *fundamental* distinctions, based on properties intrinsic to structure definitions; *discovery* distinctions, based on the observables involved in finding the structures, both those which are demonstrably necessary and those which are simply useful; and *usage* distinctions, based on expectations about the effects and use of the structures once they are found.

The categories are formed in a general way, but the position of a particular structure in the taxonomy is task-specific. For instance, different cues may be used to indicate a *theorem* in one document style than in another, leading to different discovery characteristics for the same structure. Similarly, usage characteristics may vary from one application to another. All specific examples in this chapter assume documents of the approximate style of this thesis; most are generally applicable to technical documents and many standard English documents.

3.1 Fundamental Distinctions

The most basic divisions of logical structures rest on the definitions of the structures themselves. These distinctions have obvious, direct implications for the structure discovery; what is included in the definition of a structure affects the preferred method of identifying this structure.[1] They also affect the other categorizations. These divisions include distinctions between primary and secondary structures and between content-oriented and layout-oriented structures.

[1]The structure definition does not completely determine how to discover it, however; extra-definitional cues may be quite useful, and at times it may be appropriate to categorize a document piece as a structure whose definition it does not match precisely.

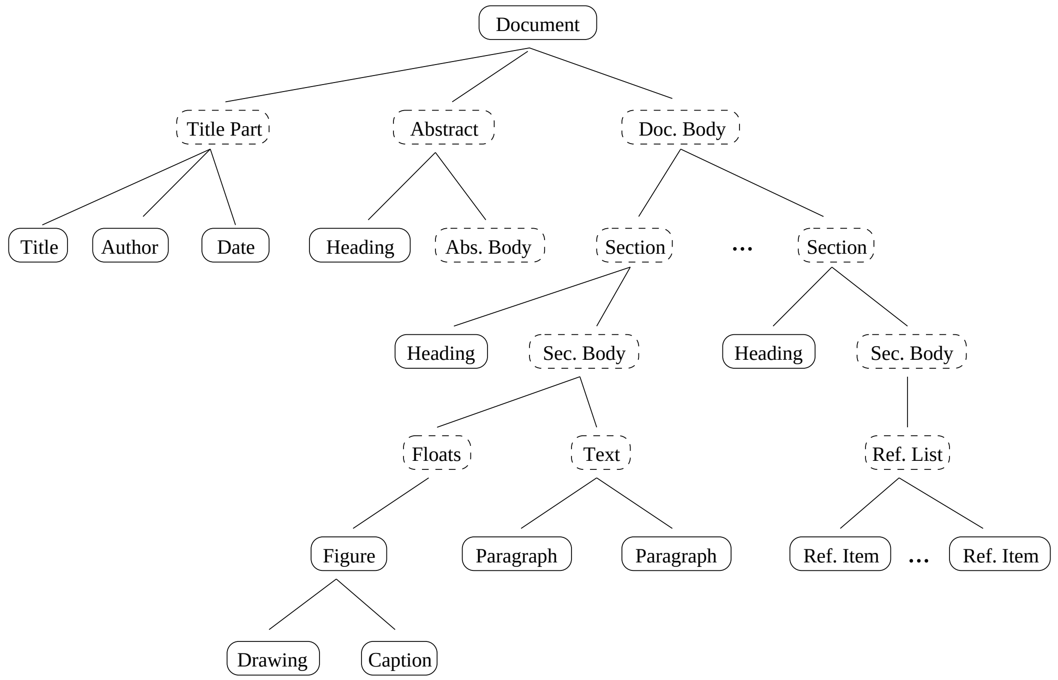


Figure 3.1: The tree in Figure 1.1 (page 3), with primary and secondary structures distinguished. Primary structures are in solid boxes; secondary structures are in dashed boxes.

3.1.1 Primary vs. Secondary

Primary structures are defined, at least in part, by their own attributes; *secondary* structures can be completely defined by their positions in the hierarchy and hierarchy relationships to other structures. For example, a *section heading* is a primary structure; it is identifiable by its appearance and separation from the surrounding text. This primary structure provides the basis for finding the secondary structures *section body* and *section*. A *section body* is a right sibling of a *section heading* with, in turn, no right sibling of its own;[2] a *section* is a node whose children are exactly a *section heading* and a *section body*. Figure 3.1 shows Figure 1.1, with primary structures in solid boxes and secondary structures in dashed boxes.

3.1.2 Content- vs. Layout-Orientation

Another fundamental distinction can be made based on the roles of content and layout in the definition of a logical structure. Although both must be included, some logical structures can

[2]This definition refers to an ideal tree, in which the *sections* have been correctly identified. In the process of forming a tree with an imperfect method, a more useful definition might be: a *section body* is a right sibling of a *section heading* whose own right sibling, if it exists, is also a *section heading*.

be considered *content-oriented*, and some can be considered *layout-oriented*. For example, a *definition* is a logical structure when it is distinguished by its presentation, as in Chapter 1 of this thesis; it remains, however, a content-oriented structure. On the other hand, a *special paragraph* (a paragraph presented in other than the usual format for a given document[3]) is a layout-oriented structure. These descriptions are relative; a logical structure is more content-oriented than another if its definition relies more heavily on internal meaning; similarly, a more layout-oriented structure has a definition that relies more heavily on visual presentation.

To make this precise, consider the hierarchy that can be formed among logical structures themselves, in which the children of a node are subtypes of that node's structure.[4] A portion of this hierarchy is given in Figure 3.2. If a structure is distinguished from its siblings entirely by content, it is content-oriented; if it is distinguished from its siblings entirely by layout, it is layout-oriented; otherwise, it is neither.

Degrees of this kind of orientation are distinguished by the degree to which this definition can be extended. That is, a structure that is distinguished from its siblings and its parent's siblings by content alone is more content-oriented than one that is distinguished from its own siblings by content alone but from one or more of its parent's siblings in part by layout. (Note that if a node is distinguished from its parent's siblings by content alone, it is therefore also distinguished from its first cousins, *i.e.*, its parent's siblings' children, by content alone.) A structure that is also distinguished from its grandparent's siblings by content alone is more content-oriented still, *etc.* This is equivalent to the idea that the degree of content-orientation corresponds to the number of immediate ancestors of a content-oriented structure that are also content-oriented. This definition of degrees of orientation also applies analogously to layout-orientation.

3.2 Discovery Distinctions

Primary logical structures can be characterized by the cues that are necessary and/or useful in their discovery. Ideally, the combination of necessary and useful characteristics would provide a sufficient criterion for structure identification. Unfortunately, we have no guarantee of this; some structures may not be reliably distinguishable without full natural language understanding, and some may not be reliably distinguishable at all. Consider, for example, a paper in which all paragraphs are left-justified and the font is never changed; to distinguish between a theorem in such a paper and a paragraph about theorems may require in-depth understanding of content. Similarly, consider a page break, preceded by a complete sentence that reaches to the right edge of the page. Whether the next block of text is a continuation of the same paragraph or the start of a new one may be ambiguous even to a human reader [68]. Hence, we cannot make general claims about sufficiency, but we can discuss the cues that are typically necessary and those which are typically useful additions in recognizing primary

[3]In this thesis, *definition* is a subtype of *special paragraph*.

[4]Typically, the logical structure of a document will be given in terms of the structures at the leaves of this hierarchy.

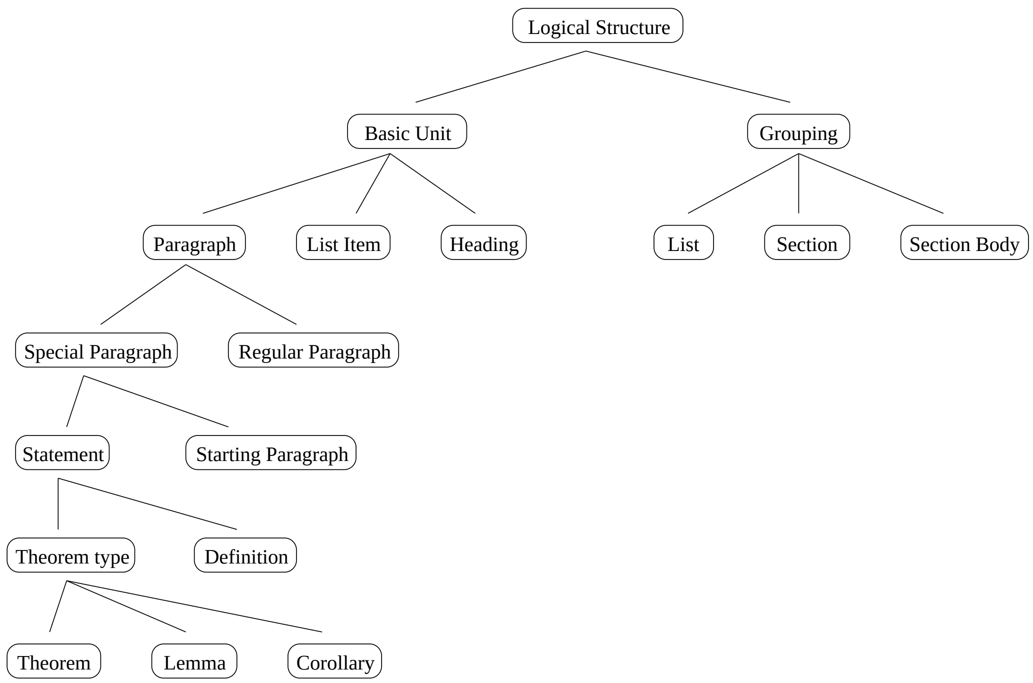


Figure 3.2: A partial hierarchy of logical structures. Each node represents a structure type that is a subtype of its parent node's structure.

structures. (Secondary structures can, of course, be discovered by applying their definitions after primary structures have been found; thus, no additional cues are needed.)

The types of cues can be derived by considering the observables of a set of marks on a background. The most basic facts are where the marks are located and what the marks are; the latter can be divided into purely visual descriptions and meanings. Additionally, any of these descriptions may be considered with respect to the surroundings. This leads to four basic categories of cues: *geometric*, *marking*, *linguistic*, and *contextual*.

Table 3.1 at the end of this section provides several examples of necessary and useful discovery cues for primary logical structures, in terms of these categories. Necessary cues are marked with an "N", and useful cues are marked with a "U." Note that a structure must require at least the observables required by its ancestors in the structure hierarchy of Figure 3.2.

3.2.1 Geometric Observables

Geometric observables include the (external) *contours* and the *internal shape* of a piece of text. (*Height* is a special case of *contours*.) Both of these kinds of cues may be necessary; for instance, the *contours* of an indented list provide its identifying shape of a hanging indent, but a table is characterized by the *internal shape* of its columnization [59].[5] Since geometry involves the shapes formed by the marks on the paper or screen, its contribution can (inversely) be found by an analysis of the white space in a document.

3.2.2 Marking Observables

Marking observables consist of non-linguistic marks on the paper or screen; this includes attributes like font type and weight, as well as non-alphanumeric *symbols*, such as bullet points and rule lines. Bullet points and dashes, for instance, can aid in the identification of *indented list items*; symbols can be necessary to find left-justified *list items*.

For example, consider Figure 3.3, in which a portion of an actual e-mail message is represented with different sets of observables. In all cases, alphabetic characters have been replaced with the letter "x." In the upper left version, all symbols and characters have been so replaced; in this representation, no difference in the format of the text blocks is visible. In the upper right version, the observables include *symbols*; the lower two text blocks can be observed to begin with a parenthesized character, suggesting that they are items in a left-justified list. In fact, this is so, as is quite clear in the lower right version, in which both *symbol*s and numbers (which belong to the *linguistic* category) are included. In this case, either *symbol*s or numbers are sufficient to suggest the presence of a list without the other, but either may be required, depending on the form of marking the list items.

[5]Identifying tables is a substantial problem in its own right, and it does not form a part of this thesis. Some of the literature on this issue is [68], [25], and [74].

Figure 3.3: Paragraphs and a justified list, with and without observable symbols and numbers

3.2.3 Linguistic Observables

Linguistic observables include combinations of *numeric* and *alphabetic* symbols. (These cues enter a gray area between *symbolic* and *linguistic* when they are character-based rather than word-based.) The observation of words is necessary for structures such as *theorem* to be recognized and distinguished from similar structures (*e.g.*, *definition*s, in many cases). The identification of *indented list item*s is aided by *numeric* cues just as by *symbolic* ones; again, these cues can be necessary to identify justified list items. For example, if the item numbers in Figure 3.3 were not enclosed in parentheses, *symbol*s would not identify the list, and *numeric* cues would be necessary to find it. In its current form, *numeric* cues are enough to suggest that it is a list, as can be observed from the lower left representation.

Typically, content-oriented structures will require linguistic cues, since content is usually contained in the language of a document. This linguistic analysis can remain quite shallow or become very complex; naturally, the subtlety of the content aspects of discoverable structures depends in part on the depth of the analysis. For example, consider an attempt to distinguish an author's institutional *affiliation* (one structure) from *address* (another structure), without making use of further analysis than checking for the presence of keywords. In order to find most institutions, the relevant keywords would probably include "University, College, Corporation, Company," to name but a few; the effect would be that streets like "University Avenue" and towns like "College Park" would be incorrectly identified as *affiliation*s.[6] The goal is probably not reasonable for the amount of included analysis.

3.2.4 Contextual Observables

Contextual observables can be divided into *local* and *global* context-based cues. Local contexts use information about some limited number of surrounding nodes: siblings, parents, children, or neighbors within a level (which may or may not be siblings). For instance, consider a typical business letter; the *return address* and the *closing* (including the signature, *etc.*) are both internally left-justified blocks indented approximately halfway across the page; in this setting, they can be distinguished easily by local context, since the *return address* is not preceded by any text, but the *closing* is.

Global contexts use information about the document as a whole. For example, a *special paragraph* is a paragraph that differs in its presentation from the typical paragraphs within the document.[7]

Contextual information may, of course, include information of any of the preceding varieties; moreover, it may make use of available structure type information.

[6]Some cases could be filtered out by requiring that *affiliations* not contain numeric values, but this would not cover every case.

[7]Identifying the typical is a significant problem, as the standard may not always occur more frequently than the non-standard.

Table 3.1: Some primary structures and their discovery cues. Necessary cues are marked "N," and useful cues are marked "U."

Cue	*Paragraph*	*Special Paragraph*	*Theorem*	*Indented List Item*
Geometry				
Contours	N	N	N	N
Internal	U	U	U	-
Marking				
Font	-	U	U	U
Symbols	U	U	U	U
Linguistic				
Words	-	-	N	-
Numbers	-	-	-	U
Context				
Global	-	U	U	-
Local	-	U	U	U

3.3 Usage Distinctions

The logical structures of a document may also be characterized according to their use. This kind of categorization attempts to capture information about the relative significance of different logical structures; it has implications for performance evaluation of logical structure discovery.

The relative importance of logical structures is, of course, application-specific. As an extreme example, consider the application of a theorem extractor. For this tool, *theorem* is the only structure of direct significance. Its ancestors in the structure hierarchy are also important, to the extent that errors in their identification may lead to errors in identifying theorems; no other structures matter, so errors in their identification are insignificant. Finding the full logical structure would be unnecessary for this application, but the point stands that if a structure discovery mechanism is designed for a particular application, its output should be evaluated with respect to structure importance within that application.

In the more general case, however, the logical structure is derived for possible use with many applications, and the kind of information described above is therefore unavailable. A more general (and necessarily less precise) concept of structure significance is required. This raises a variety of different issues, including: classifier implications, expected user references, hierarchy role, and generality. These are described below.

Table 3.2 at the end of this section (on page 27) provides several examples of the usage characteristics of logical structures, in terms of these categories.

3.3.1 Classifier Implications

This attribute refers to the relevance of a structure to the identification of other structures. To some extent, this is dependent upon the classifier itself, but it also depends on the intrinsic definitions of the structures; the definitions of secondary structures highlight the importance of certain other structures, as do the definitions of primary structures that depend in part on their contexts (*e.g.*, *special paragraph* relies on other *paragraph*s). For example, section headings are quite significant in this respect, as two secondary structures (*section bodies* and *section*s) rely on them for correct identification.[8]

3.3.2 Expected User References

Structures that users refer to more frequently than others are, in an important sense, particularly significant. For example, if users write queries that ask for full sections more often than groups of paragraphs, then sections are more significant for retrieval than are paragraphs (unless, of course, these sections are frequently identified by particular paragraphs they contain).

This attribute is task-dependent; different structures may be commonly used in retrieval from those commonly used in browsing, for example. Furthermore, it depends on the user population, because of differences in both cognitive models and underlying goals. That is, users who think about information in different ways may tend to access document structures differently; also, users who want information in order to complete one goal might be likely to access this information differently from users attempting to complete a different goal with it.

Determining precise expectations for this would require tracking the behavior of differing groups of actual users with a fully general system, including different kinds of documents. Such a study would provide a solid basis for according relative weights to different logical structures, with respect to this attribute.

In the absence of strong empirical evidence, however, certain general observations can be made, based on the natures of commonly suggested applications.

- In hierarchical browsing as described in [14, 77], structures at higher levels of the tree are more significant than those at lower levels. Since this browsing is based on tree navigation, starting from the root, higher-level structures will be used for earlier decisions, on which later decisions will in part rely. Furthermore, for any node that is accessed, all of its ancestors must have been accessed as well, but its descendents need not be. So for this application significance corresponds (to a large degree) to height.

- For hyperlinking, bibliographic structures have a special significance. Links will often be desirable based on bibliographic matches (such as articles that share authors, or a match between a reference in one article and the title of another), so the structures

[8]Here, as elsewhere in this chapter, the word "section" may be replaced by "subsection" or "sub^{n}section" where $n \geq 0$.

Table 3.2: Some Structures and Usage Characteristics

Attribute	*Paragraph*	*Heading*	*Section*
Implications	Paragraph Group	Section Body, Section	
Task Importance		Browsing	Browsing
Hierarchy Role	Useful	Useful	Useful
Generality	High	High	High

Attribute	*Section Body*	*Theorem*	*Definition*
Implications		Proof	
Task Importance	Browsing	Retrieval	Retrieval
Hierarchy Role	Filler	Useful	Useful
Generality	High	Low	Low

that provide this information are particularly important. Floats (figures, tables, *etc.*) have a similar importance for linking, as they are typically referenced in the text.

Other kinds of hyperlinks are often desirable as well, of course, including links that reflect relationships based more on content than structure and links that reflect indirect relationships. It is not obvious how these will relate to logical structure, however, as the criteria for their inclusion are still emerging.

- In the retrieval of previously-seen documents or document portions, highly significant structures are likely to be those which differ greatly from their surroundings. Since, according to Cowan, this kind of difference demands attention and people are likely to remember what has attracted their attention [19], such salient structures are likely to be of particular use in characterizing a remembered document. (Determining which structures differ greatly from their surroundings in this sense, however, is far from trivial.)

The above is not meant to be exhaustive; it simply provides an example of the kinds of issues that can provide insight into the significance of different logical structures from the direct perspective of a user.

3.3.3 Hierarchy Role

A significant distinction can be drawn between those structures that exist in order to express a useful piece of the document and those that exist in order to complete the hierarchy. For example, the structure *paragraph part* is not, in itself, useful; it exists in order to complete the children of a *paragraph* that contains an *equation* or an *indented list* or some other interruption. *Filler* structures that exist only to complete the hierarchy are a proper subset of secondary structures. Useful structures are, of course, more significant than are fillers (although distinguishing the two may be extremely significant!)

3.3.4 Generality

Consider the structure hierarchy, partially shown in Figure 3.2. Distinguishing among structures that appear at lower levels of the tree changes the meaning of the result less than distinguishing between structures at higher levels; thus those at higher levels are more significant, in that their correct identification provides more new content. If a system attempts to distinguish structures at a low level of the tree, it is unlikely to return an answer of a structure at a high level, but classifying a block as a structure implicitly places it in the category of each ancestor of the structure, as well. So in this sense, an error that misidentifies a block as a close relative of its structure is less significant than an error that identifies the same block as a more distant relative of the structure.

3.4 Effects of the Types

The distinctions among logical structure types provided in this chapter have implications for the work presented in the rest of this thesis, especially the process of classifying text segments.

The distinction between primary and secondary types has an obvious implication for structure classification; primary types are found first, and secondary types are found later, based on their definitions and the primary structures that have been discovered. The effects of content- vs. layout-orientation are less direct; these influence the discovery distinctions, which do have direct implications.

The discovery distinctions influence the cues used for the classification of text segments. Necessary observables must be included, and useful kinds of observables are usually included as well. The particular cues that are considered are discussed in Chapter 6. The distinctions based on usage are more difficult to apply to the problem; they can influence performance evaluation, which is discussed in Chapter 8, but this would require a quantification of their effects. Since the distinctions made here are qualitative and relatively rough, for lack of concrete, user-based evidence, they do not currently form a part of the evaluation mechanism.

Chapter 4

System Overview

The problem of document structure discovery is divided into two main subproblems: segmentation and classification. The hierarchy is formed from the bottom up, with both segmentation and classification applied as each level is formed. Segmentation is performed on the basis of shape only, including white space and font information; classification considers the non-geometric information discussed in Chapter 3 as well, when this is appropriate. (Table 3.1, in Section 3.2, provides some examples of the cue types on which certain structures rely.)

LABLER builds a level of the hierarchy in the following manner. First, the segmenter identifies the text blocks to be grouped together, based on contour relationships for lower levels and proximity for higher ones. The details of this procedure are discussed in Chapter 5. Next, the classifier assigns a structure to each text block, based on a set of generic, idealized prototypes. A block is assigned the structure to whose prototype it is most similar; this similarity measure is preserved as a degree of certainty. The classifier then provides feedback to the segmenter in the following manner. A text block assigned a structure with a low degree of certainty causes an attempt at resegmentation of a small surrounding area; if a segmentation is found that improves the classification results for that area, this is used instead of the original result. After each ordinary level is formed and its nodes are classified, it is searched for components of secondary structures (*i.e.*, those structures defined entirely by their relationships to other structures), and these are added.

LABLER incorporates available knowledge of the style of a document into the above approach. If a CFG is provided for some identifiable document portion, this part of the document is parsed first, providing a subhierarchy that is later linked to the hierarchy found by the above method. Partial parses are also performed, after the initial parsing attempt, in case the document contains components of the specified structure but not all of it. (For example, in a collection for which a *title part* is specified that includes a *title page* and a subordinate *title part* on a subsequent page that also contains text, a document may have lost its title page; a partial parse will find the remaining subordinate *title part*.)

If the available knowledge for all or a part of the document does not form a grammar, less detailed knowledge can also be used. If the form of certain structures that may appear is known, the classification procedure is modified accordingly; a new description in terms of the available observables may be identified as necessary, sufficient, or typical. If it is necessary, a mismatch leads to a similarity measure of 0; if sufficient, a match leads to a similarity measure of 1; if typical, the generic prototype is replaced by the new description.

Once the entire hierarchy is built, a final postprocessing stage adjusts the hierarchy. It applies a few linguistic heuristics to identify, *e.g.*, various elements of the title part, checks for nodes near each other with ascending numeric values (usually *list item*s) and ensures that they are grouped appropriately, checks that other groupings of nodes of the same general classification are separated only when there are appropriate reasons,[1] and manipulates the tree to eliminate useless structures (such as a division of a *paragraph* into *paragraph part*s without any non-filler structure).

The system is sketched in Figure 4.1. The details are given in Chapters 5 (segmentation), 6 (classification), and 7 (knowledge incorporation).

[1]Groups of nodes of the same general classification may be separated from each other based on the existence of greater vertical white space between the groups than within them or in order to form groups of the same specific classification.

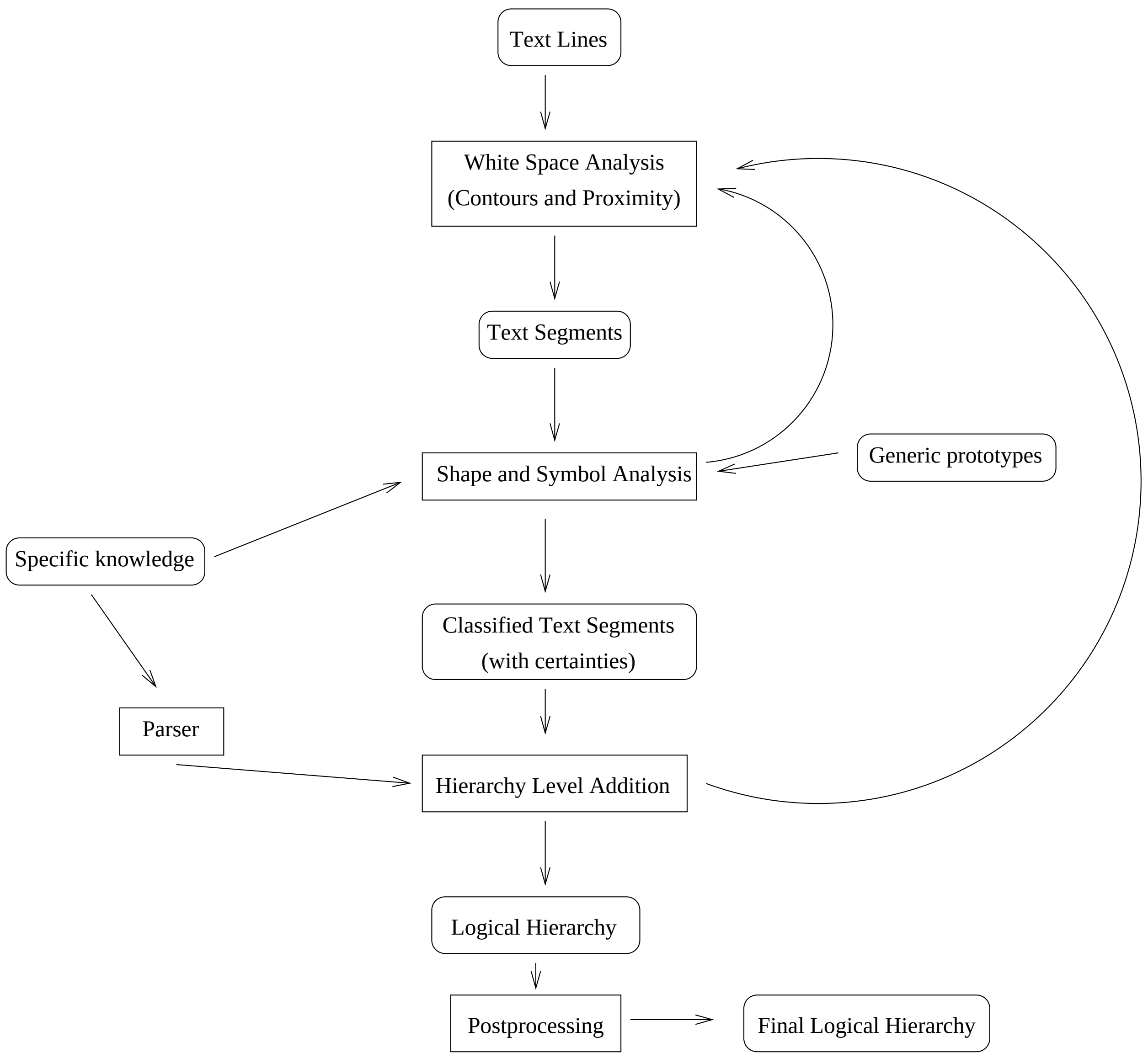


Figure 4.1: The logical structure discovery system overview

Chapter 5

Segmentation

This chapter presents an algorithm that takes an electronic document and generates a hierarchy of logical divisions. The observation of geometric relationships, including contour similarity, vertical distance, and horizontal containment, is used to partition the document into a hierarchy of divisions, including arbitrary degrees of nesting. The resulting tree is suitable for indexing, browsing, and searching: its divisions may be indexed according to information retrieval techniques, a document may be browsed by navigating through the tree, and a set of trees may be searched for documents that include components with required properties.

```
1.  Assumptions and Definitions                                            1
                                                                           2

     In all characterizations of WDGs, we make the following               3
assumptions.  They enable us to characterize the WDGs of                   4
any polygon precisely.                                                     5
                                                                           6
        • The block of text for which the WDG is given                     7
           contains exactly one figure.                                    8
                                                                           9
        • The block and the figure are of the same height                 10
           n + 1, with rows numbered from 0 to n.                         11
                                                                          12
The above assumptions refer to the choice of text block.                  13
     There are also assumptions about the granularity of the view of the  14
figure to be considered.                                                  15
                                                                          16
        • There is at least one column of white space to the              17
           left and one column of white space to the right of             18
           the figure.                                                    19
                                                                          20
        • All lines have width 1.  Thus, points have height               21
           and length 1.  This affects the precise value of               22
           the WDG.                                                       23
                                                                          24
This concludes our listing of assumptions.  We now                        25
move on to definitions.                                                   26
     Portions of WDGs are characterized by their heighs and slopes.  A    27
special case occurs when the height remains constant.  A plateau          28
in a WDG is a segment of length at least 1 with slope 0.                  29
A transition in a WDG is a segment of length at most 2 that               30
lies between two plateaus.                                                31
     This concludes our section of definitions.  We now move on to        32
use them.                                                                 33
```

Figure 5.1: A miniature document with its lines numbered

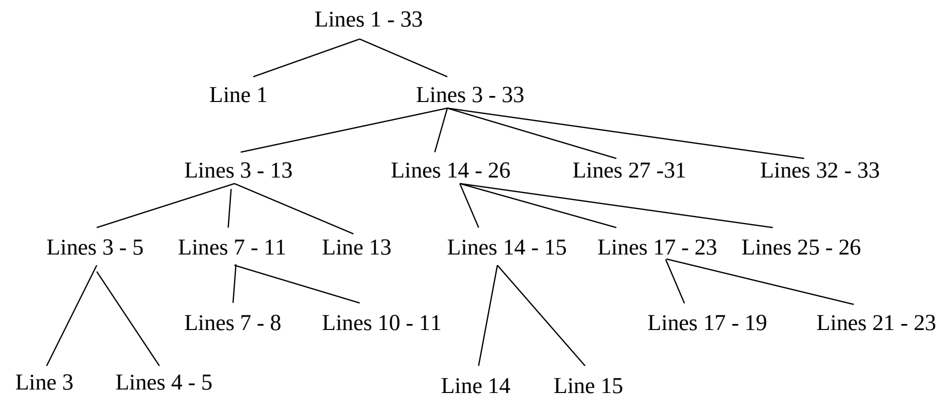


Figure 5.2: Indentation tree for the miniature document in Figure 5.1

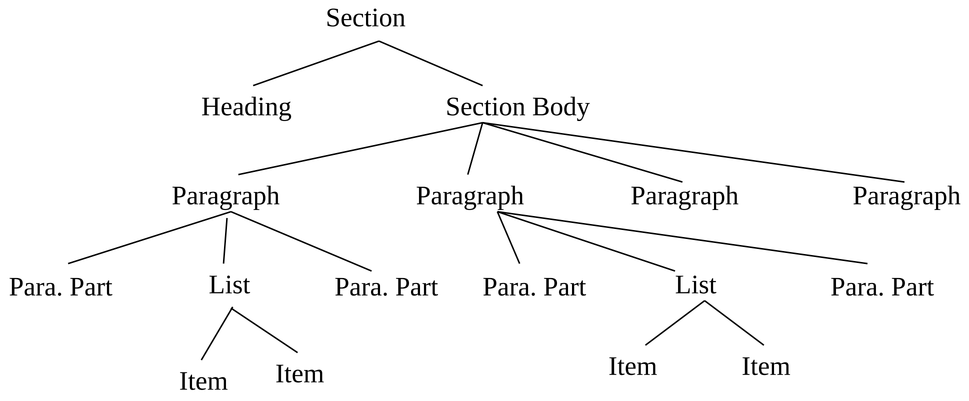


Figure 5.3: Goal tree for the miniature document in Figure 5.1

This layout segmentation captures the divisions of the document's logical structure. In order to generate a tree that describes the logical structure the remaining task is to classify its nodes. For example, Figure 5.1 contains a miniature document of one section. At the coarsest level, it consists of the heading (the first line) and the body (the rest). The section body contains four paragraphs, two of which contain lists, each of which in turn contains two list items. The segmentation algorithm generates the tree in Figure 5.2. This tree constitutes the topology of the tree in Figure 5.3, with a few superfluous leaves added. The node classification task needed for the construction of Figure 5.3, including both the labelling of the nodes and the identification of those to delete, is addressed in Chapter 6. In fact, as discussed in Chapter 4, classification is performed on each level of nodes immediately after its creation.

Section 5.1 presents the algorithm that generates the segmentation. This algorithm requires an *indentation alphabet*; this construct is defined and discussed in Section 5.2. Section 5.3 provides an example of the behavior of the algorithm. Section 5.4 discusses the incorporation of feedback from the classification algorithm into the tree creation described in Chapter 4.

5.1 The Segmentation Algorithm

The segmentation algorithm is derived from a more specific version of the elements of Observation 1 in Chapter 1. The key idea is that structures at a basic level are typically repeated, usually adjacent to each other, using the same shapes. Some of these structures will be split by others, such as a paragraph that is split into two parts by a list or block quote inside it. Such splits form lower levels. Groups of identically shaped structures generally form relevant structures at a higher level, and at a higher level still, divisions of a document are indicated by vertical distance.

The algorithm works in a bottom up manner, building a tree of indentation structures from a given set of leaves. At the lowest level, the nodes are characterized according to an *indentation alphabet*.[1] The level is then searched for repeated indentation patterns. The elements of these patterns are taken to be structures at the basic level described above, or parts of them. Sets of leaves that form pattern elements are combined into new nodes at the tree's next level; also, runs of *isolated* nodes that do not participate in patterns are searched for sequences that conform to shapes of previously established patterns, and these too are combined into new nodes. This reflects Observation 1a. The next step uses Observation 1c. It repeatedly looks for structures that fall horizontally within those structures that surround them vertically, such that the second surrounding structure is a continuation of the contour of the first. These are deemed to be structures that split a surrounding structure, and the whole group is merged together. When no more such patterns can be found, this step stops. The resulting top level is represented in the indentation language, and this is again searched for patterns. This time, each pattern is combined into a new node at the next level up. Finally, the top of the hierarchy is formed based on vertical distance; repeatedly, the current closest structures are combined into new nodes, until no further combination is possible.

When a new node is created from a combination of old nodes, each of the old nodes becomes a sub-block of the block described by the new node. (A node that does not participate in a repeated pattern is considered to form a pattern of length one. Thus, at the next two levels it remains a node, and each block it describes has a single sub-block: itself.) The algorithm is given in outline form below.

Algorithm 5.1

1. Divide the document into blocks at the lowest level by an external algorithm.

2. Represent each block by an appropriate string of indentation alphabet characters.

3. (a) Find sets of blocks that form repeating patterns.

[1] Indentation alphabets provide symbols to express the indentation of blocks of text and rules for matching sequences for these symbols. They are defined formally in Subsection 5.2.1.

   (b) Find runs of isolated blocks that conform to patterns found elsewhere.
   (c) Group together the blocks in each element of each pattern, and group the isolated block runs, forming the next level of the tree.

   surrounding blocks. If this generates changes, it creates a level.

4. Repeat until no changes are generated
   (a) Search for adjacent strings of blocks that either:
      - Fall horizontally within their surrounders, such that either:
        - The following surrounder has all its sub-blocks' left edges at the same place as the final left edge of the preceding surrounder

        or

        - The preceding surrounder has a height of 1 line, and the following surrounder has all its sub-blocks' left edges at the left margin

      or

      - Have all their sub-blocks' left edges at the same place as the final left edge of the preceding block, and this position is not the left margin
   (b) Merge the preceding blocks with their continuations, to form blocks at the next level, and represent these new blocks in the indentation alphabet.
5. (a) Search the current top nodes for repeating patterns in the indentation alphabet.
   (b) Group together the elements of each pattern, forming another tree level.
6. Repeat until no new changes are generated:
   Combine the currently closest nodes, in terms of vertical white space between them, to form a new node.

Step 1 currently divides the document into text blocks all of whose edge shifts are in the same direction, inward or outward. (Since each text block's first shift is with respect to the margins, a block with three or more shifts, of which all but the first are outward, is considered to be a block of outward shifts.) Each blank vertical space is its own block. Each document line in such a block constitutes a sub-block. Other initial segmentations are possible, such as the one proposed in [76]. Step 4 reflects both the observation that basic structures may be interrupted by structures that fall horizontally within them and the fact that a basic structure may contain blank vertical space if the margin of the later part is readily identifiable as a continuation of the earlier part.

The hierarchical document divisions allow an arbitrary degree of nesting in the structure of the document. For example, in Algorithm 5.1, the enumerated list in step 4 forms an identifiable indentation structure, which is a part of the observable (higher-level) indentation structure consisting of the whole algorithm. With a carefully defined indentation alphabet (such as that described in Subsection 5.2.3), the structure formed by the subsections and sections is also identifiable.[2] The degree of such nesting in a given paper is arbitrary. Correspondingly, this bottom-up algorithm builds trees of arbitrary depth.

5.2 Indentation Alphabets

Subsection 5.2.1 provides a formal specification of indentation alphabets; Subsection 5.2.3 presents the particular alphabet that LABLER uses. Finally, Subsection 5.2.4 offers a formal specification and analysis of the language defined by patterns in any indentation alphabet that shares certain basic properties.

5.2.1 Formal Definition of an Indentation Alphabet

An indentation alphabet, formally defined below, consists of a set of symbols, their definitions, and a lattice over symbol sequences. A string of the symbols is used to describe the contours of a text block in terms of its component sub-blocks. Each symbol represents a non-empty sequence of sub-blocks. The definition associated with a symbol specifies the sub-blocks that are representable by the symbol. For example, LABLER's indentation alphabet includes the symbol Left-Justified and the definition "one or more consecutive sub-blocks, not all of which are blank, in which the text sub-blocks are all left-justified with respect to the preceding text sub-block." Consider a block formed by the paragraphs of a section, with each paragraph forming a sub-block; in many layout formats, a string of this symbol alone will suffice to express the contours of the given block.

Definition 5.1 *An* indentation alphabet *is a triple $\langle \Sigma, D, R \rangle$, where Σ is a set of symbols; D is a set of definitions for the symbols in Σ, such that, given a block of text made up of one or more sub-blocks, exactly one sequence of definitions describes the sub-blocks that make up the block; and R is a subsumption relation among the symbols in Σ, in the sense of Definition 5.2.*

Definition 5.2 *A* subsumption relation *R in an indentation alphabet $\langle \Sigma, D, R \rangle$ is a lattice over $\Sigma^* \cup \{\top, \bot\}$.*

The subsumption lattice defines a relationship of "more general than" among symbol sequences. That is, the definitions of the symbols of an indentation alphabet may overlap in the following sense: more than one symbol sequence may accurately describe the contours of a given sub-block sequence. In this case, one symbol sequence will be a more general representation of the other; the lattice specifies this relationship. For instance, suppose an indentation alphabet also includes the symbol Both-Justified, defined like Left-Justified (above) with the added constraint that the sub-blocks must be *both* left- and right- justified

[2] This requires distinguishing vertical spaces of different heights from each other. An alphabet that considered all vertical blank space to be equivalent might, for instance, match a heading and the text that follows it to a theorem and the text that follows it.

with respect to their common predecessor. A set of sub-blocks that satisfies this definition also satisfies the definition of Left-Justified. The reverse is not necessarily true; thus, Left-Justified is more general than Both-Justified. This information is captured in the subsumption relation, with Left-Justified $\succeq_R$Both-Justified.

The lattice has two uses. The first use is in the selection of a representation for a sequence of sub-blocks; if two symbol sequences α and β may describe the sub-blocks and $\alpha \succ_R \beta$, then the sub-blocks are described with the sequence β. (To form a useful indentation alphabet, in such cases the lattice must specify either $\alpha \succeq_R \beta$ or $\beta \succeq_R \alpha$.)

The other use of the lattice is to enable matching. If $\alpha \succeq_R \beta$, this is interpreted to mean that α represents an indentation form that subsumes the form represented by β. Thus, in the process of comparing sequences of indentation alphabet symbols, an instance of β may be matched against an instance of α, yielding a pattern of α. Moreover, if $\alpha \succeq_R \gamma$ as well, then both β and γ may be matched as instances of the pattern α (without an explicit appearance of α). More precisely, the interpretation of the subsumption relation is: if the least upper bound of two sequences α and β is not $\top$, then the two sequences match, yielding a pattern of their least upper bound.

For example, consider again the symbols Left-Justified and Both-Justified. Left-Justified $\succeq_R$Both-Justified indicates that the form Left-Justified subsumes the form Both-Justified. Hence, in comparing two sequences α and β, if an instance of Left-Justified is encountered in position i of α and an instance of Both-Justified is encountered in position i of β, with all previous symbols in the sequences matching, then these symbols are considered to match; the examined prefixes of α and β are instances of a pattern γ with Left-Justified in position i.

The effects of the least upper bound can be seen by considering an indentation alphabet with the symbols already described and additional symbols Centered and $\langle$Centered-In, $n\rangle$. Let Centered describe a sequence of sub-blocks, all of which are centered with respect to their predecessor; let $\langle$Centered-In, $n\rangle$ represent a sequence of n sub-blocks, all of which are centered with respect to their common predecessor and each of which has a left edge that lies farther in than (*i.e.*, to the right of) the left edge of its particular predecessor. (Note that the individual predecessor of the first block is the same as the predecessor of the whole group; the individual predecessors of the other blocks lie within the group.) These definitions imply that Centered $\succeq_R\langle$Centered-In, $n\rangle$, and Centered $\succ_R$ Both-Justified. Consider comparing Both-Justified to $\langle$Centered-In, $2\rangle$; neither subsumes the other, but both are subsumed by Centered. That is, lub(Both-Justified, $\langle$Centered-In, $2\rangle$) $=$ Centered. Consequently, the symbols match, yielding a pattern of Centered.

Indentation alphabets can also include the special symbols *start-block* and *end-block*. These symbols begin and end each block description, and they simplify the characterization of the type of patterns appropriate for this task (*i.e.*, those which include only complete blocks). Different choices of indentation alphabets yield different results; the alphabet may be adjusted for different goals without affecting the algorithm. Similarly, the method of deriving blocks and sub-blocks may be altered without affecting the indentation alphabet. This modularity increases the power and ease of use of both elements of the document segmenter.

5.2.2 Useful Subsumption Relation Types

Some subsumption relations have special properties that aid their computational use. In particular, this subsection describes a set of such properties that ensures the possibility of finding the least upper bound of two strings by comparing one symbol at a time, without backtracking.

First, the least upper bound of two strings must be derivable from the least upper bounds of their substrings.

Definition 5.3 *The subsumption relation of an indentation alphabet* $\langle\Sigma,\ D,\ R\rangle$*, is* normal *if*

$$(\alpha \succeq_R \beta \wedge \gamma \succeq_R \delta) \Longrightarrow (\alpha\gamma \succeq_R \beta\delta).$$

That is, in a normal subsumption relation, concatenation does not invalidate subsumption. In order for substring subsumption to fully control string subsumption, the following stronger condition is required.

Definition 5.4 *A normal subsumption relation* R *of an indentation alphabet* $\langle\Sigma,\ D,\ R\rangle$ *is* divisible *if*

$$(\mathrm{lub}(\alpha,\beta) = \gamma) \Longrightarrow (\forall\delta,\eta \in \Sigma^* : \mathrm{lub}(\delta,\eta) \neq \top \Rightarrow \mathrm{lub}(\alpha\delta,\beta\eta) = \gamma\mathrm{lub}(\delta,\eta)).$$

This indicates that a pattern found by looking at the beginning of a sequence will not be invalidated by symbols later in the sequence. In pattern matching based on a divisible subsumption relation, substrings may be considered from left to right, without requiring backtracking. A new lattice can now be defined, made up of the substrings that will need to be considered.

Definition 5.5 *The* minimal subsumption relation $R_{\min}$ *of an indentation alphabet* $\langle\Sigma,\ D,\ R\rangle$ *with a divisible* R *is the lattice over* $\Sigma'^* \cup \{\top,\ \bot\}$*, where* $\Sigma' \subseteq \Sigma$ *such that* $(\alpha \succeq_{R_{\min}} \beta) \Longleftrightarrow$

$$\begin{aligned}&((\alpha \succeq_R \beta) \wedge \\ &\quad(\forall\alpha',\beta' \in \Sigma^* : \\ &\qquad(\exists\gamma,\eta \in \Sigma^* : (\gamma \neq \varepsilon \vee \eta \neq \varepsilon) \wedge (\alpha = \alpha'\gamma \wedge \beta = \beta'\eta)) \Rightarrow \neg(\alpha' \succeq_R \beta'))).\end{aligned}$$

Thus, α subsumes β in $R_{\min}$ if in R, α subsumes β, but no proper prefixes of α and β bear this relationship. The rules of $R_{\min}$ may be used for pattern matching according to a divisible R. For matching to proceed based on individual symbols, the form of $R_{\min}$ must be constrained.

Definition 5.6 *A divisible subsumption relation* R *of an indentation alphabet* $\langle\Sigma,\ D,\ R\rangle$ *is* symbol-oriented *if*

$$\forall\alpha,\beta \in \Sigma^* : ((\alpha \succeq_{R_{\min}} \beta) \Longrightarrow |\beta| = 1).$$

With a symbol-oriented subsumption relation, matches of indentation alphabet strings can be performed on one character at a time, in a manner based on the following definitions.

Definition 5.7 *The* symbol lattice*, R', of an indentation alphabet $\langle\Sigma,\ D,\ R\rangle$with a symbol-oriented R is given by:*
$\forall x, y \in \Sigma : ((x \succeq_{R'} y) \Longleftrightarrow$
$(\exists x_1, \ldots x_n \in \Sigma, \alpha_0, \ldots, \alpha_n \in \Sigma^* :$
$(x\alpha_0 \succeq_{R_{\min}} x_1) \wedge (x_1\alpha_1 \succeq_{R_{\min}} x_2) \wedge (x_2\alpha_2 \succeq_{R_{\min}} x_3) \wedge \ldots \wedge (x_n\alpha_n \succeq_{R\min} y))$.
The sequence $\alpha_0\alpha_1 \ldots \alpha_n$ is called the y-x addition.

Note that, if $\alpha_0 \ldots \alpha_n$ is the *y-x* addition, then $x\alpha_0 \ldots \alpha_n \succeq_R y$. So, matching x to y requires accounting for $\alpha_0 \ldots \alpha_n$ in y's sequence; it is added to y's sequence if y is matched with x and is hence the *y-x* addition.

Then, a pair of sequences to be matched according to a symbol-oriented subsumption relation can be thought of as a pair of stacks of symbols. At each step in the match, a symbol is popped off of each stack. Suppose these symbols are x and y. If the least upper bound z of x and y in the symbol lattice is not $\top$, then the match (so far) succeeds; z is appended to the subsuming pattern formed so far, the *x-z* addition is pushed onto x's stack, and the *y-z* addition is pushed onto y's stack.

The least upper bounds and additions can be calculated in advance, leading to an alternative representation of these subsumption relations as pairs of the form $\langle(x, y, z), (\alpha, \beta)\rangle$, where α is the *x-z* addition, and β is the *y-z* addition. This representation provides matching rules directly.

For instance, consider the sequences $\alpha = \langle$Centered-In, $2\rangle$ and $\beta =$ Left-In, Centered. Let $\langle$Centered-In, $n\rangle$ and Centered be defined as above; let Left-In represent a single sub-block whose left edge is inward (*i.e.*, to the right) of its predecessor. We have Left-In $\succeq_{R_{\min}}\langle$Centered-In, $n > 1\rangle$ with an addition of $\langle$Centered-In, $n-1\rangle$,[3] as well as Centered $\succeq_{R_{\min}}\langle$Centered-In, $n\rangle$, with no addition. That is, we have the matching rules $\langle$(Left-In, $\langle$Centered-In, $n\rangle$, Left-In), (ε, $\langle$Centered-In, $n-1\rangle$)$\rangle$ and $\langle$(Centered, $\langle$Centered-In, $n\rangle$, Centered), (ε, ε)$\rangle$.

Thus, the comparison of the above α and β proceeds as follows. Pop $\langle$Centered-In, $2\rangle$ off α and Left-In of β and compare. These may match; the pattern so far is Left-In, and we push $\langle$Centered-In, $1\rangle$ onto α. Now, we have $\alpha = \langle$Centered-In, $1\rangle$ and $\beta =$ Centered. Pop the first (and only) symbol off of each. These match, according to the second rule above, with nothing to push. The achieved pattern so far is Left-In, Centered; both stacks have been emptied, so the match is successful. In this case, the general pattern, of which both sequences are instantiations, is the same as the second sequence; in some cases it will not be precisely the same as either sequence.

[3]In the case that $n = 1$, we have Left-In $\succeq_{R_{\min}}\langle$Centered-In, $1\rangle$, with no addition.

5.2.3 LABLER's Indentation Alphabet

LABLER uses the following indentation alphabet. When characterizing the first sub-block in a block, the current left and right margins serve as the left and right edges of the "preceding sub-block."[4]

- $\Sigma = \{$ Start-Block, End-Block, Both-Justified, Left-Justified, $\langle$Centered-In, $m\rangle$, $\langle$Centered-Out, $n\rangle$, Centered, $\langle$Left-In, $p\rangle$, $\langle$Left-Out, $q\rangle$, $\langle$Blank, $h\rangle\}$, where m, n and h range over the positive integers.

- $D =$

  Start-Block Represents the beginning of a new block; does not correspond to a sub-block.

  End-Block Represents the end of the current block; does not correspond to a sub-block.

  Both-Justified A set of one or more consecutive sub-blocks, not all of which are blank, in which the text sub-blocks are both left-justified and right-justified with respect to the preceding sub-block.

  Left-Justified A set of one or more consecutive sub-blocks, not all of which are blank, in which the text sub-blocks are left-justified with respect to the preceding sub-block.

  $\langle$**Centered-In,** $m\rangle$ A set of sub-blocks containing m text sub-blocks, all of which are centered with respect to the preceding sub-block. The left edge of each of the m sub-blocks lies to the right of the left edge of its predecessor.

  $\langle$**Centered-Out,** $n\rangle$ A set of sub-blocks containing n text sub-blocks, all of which are centered with respect to the preceding text sub-block. The left edge of each of the n sub-blocks lies to the left of the left edge of its predecessor.

  Centered A set of one or more consecutive sub-blocks, not all of which are blank, in which the text sub-blocks are centered with respect to the preceding sub-block.

  $\langle$**Left-In,** $p\rangle$ A text sub-block whose left edge lies to the right of the left edge of the preceding sub-block, by a difference of p.

  $\langle$**Left-Out,** $q\rangle$ A text sub-block whose left edge lies to the left of the left edge of the preceding sub-block, by a difference of q.

  $\langle$**Blank,** $h\rangle$ A set of one or more completely blank sub-blocks, of total height h.

- R_{min} is given by:
  Left-Justified $\succeq_{R_{\text{min}}}$ Both-Justified,

[4] Usually the current margins are the left and right margin of the entire document, but there may be local deviations, introduced by unusual pages, skew, or figures around which the text flows.

Centered $\succeq_{R_{\min}}$ Both-Justified,
Centered $\succeq_{R_{\min}}$ ⟨Centered-In, m⟩,
Centered $\succeq_{R_{\min}}$ ⟨Centered-Out, n⟩,
indcharLeft-In $\succeq_{R_{\min}}$ ⟨Centered-In, 1⟩,
Left-Out $\succeq_{R_{\min}}$ ⟨Centered-Out, 1⟩,
⟨Centered-In, 1⟩⟨Centered-In, m⟩ $\succeq_{R_{\min}}$ ⟨Centered-In, $m+1$⟩
⟨Centered-Out, 1⟩⟨Centered-Out, n⟩ $\succeq_{R_{\min}}$ ⟨Centered-Out, $n+1$⟩

All matches with Left-In and Left-Out must also match the left edge distance p or q, so this information is maintained for Centered-In and Centered-Out blocks.

R may also be expressed in the alternative form for symbol-oriented divisible subsumption relations as:
{⟨(Left-Justified, Both-Justified, Left-Justified), $(\varepsilon, \varepsilon)$⟩,
⟨(Centered, Both-Justified, Centered), $(\varepsilon, \varepsilon)$⟩,
⟨(⟨Centered-In, m⟩, ⟨Centered-Out, n⟩, Centered), $(\varepsilon, \varepsilon)$⟩,
⟨(Centered, ⟨Centered-In, m⟩, Centered), $(\varepsilon, \varepsilon)$⟩,
⟨(Centered, ⟨Centered-Out, n⟩, Centered) $(\varepsilon, \varepsilon)$⟩,
⟨(⟨Centered-In, 1⟩,Left-In, Left-In), $(\varepsilon, \varepsilon)$⟩,
⟨(⟨Centered-Out, 1⟩,Left-Out, Left-Out), $(\varepsilon, \varepsilon)$⟩,
⟨(⟨Centered-In, $m > 1$⟩, Left-In, Left-In), (⟨Centered-In, $m-1$⟩, ε)⟩,
⟨(⟨Centered-Out, $n > 1$⟩, Left-Out, Left-Out) (⟨Centered-Out, $n-1$⟩, ε)⟩}

In this alphabet, no sub-block is ever initially identified as Centered; this symbol exists for its role in the subsumption relation. Sub-blocks may be determined to follow a pattern that includes the symbol Centered, at which point, an instance of ⟨Centered-In, m⟩ or ⟨Centered-Out, m⟩, or Both-Justified becomes subsumed by the symbol Centered. Note also that there is no Right-Justified symbol; right-justification happens often as a side effect of the combination of indentation and both-justification, but it rarely has significance of its own.

Additionally, the distances of the moves in and out at the left edge are maintained. Two Left-Ins or Left-Outs only match if their distances are considered equal. (Since OCR does not reliably yield precisely the same distance for all equal indentations, two distances are considered equal if their values are very close.) Similarly, Center-Ins can only be subsumed by Left-Ins and Center-Outs by Left-Outs if their distances match.

The height of blank sub-blocks provides a distinction between sections and subsections, as well as other nested structures in which the divisions are indicated in part by vertical white space. Since section headings and subsection headings are typically represented in different font sizes, the space surrounding these types of headings differs. Thus, a section heading and its surrounding blank lines will not match the pattern formed by a set of subsections; the subsections will be recognized as a structure at one level, and the section containing them will form part of a section structure at a higher level. Without considering font information directly, this approach takes advantage of its effects to enable the building of complete logical structure trees.

5.2.4 The Indentation Pattern Language

This subsection provides a formal specification for the language of the indentation patterns found by the algorithm, called the *block indentation patterns*, using a symbol-oriented indentation alphabet that includes the special symbols Start-Block and End-Block. The preparation for this definition and its analysis includes a notion of *fully distinguishable* symbols in an indentation alphabet and a useful relation E_R over the symbols of an indentation alphabet.

Definition 5.8 *Given an indentation alphabet* $\langle \Sigma,\ D,\ R \rangle$*, with* R *symbol oriented, and* $x, y \in \Sigma$*,* x *and* y *are said to be* fully distinguishable *if, in the symbol lattice* $\text{lub}(x, y) = \top$*.*

Intuitively, x and y are fully distinguishable if they can never be considered a match, regardless of what follows them.

Definition 5.9 *Given descriptions of text at a sufficiently fine-grained level that each has an indentation alphabet description of only one symbol in addition to an optional Start-Block and/or End-Block, two text descriptions* t_i *and* t_j *are* fully distinct *if the symbols other than Start-Block and End-Block in their indentation alphabet representations are fully distinguishable and all their other observables have different values.*

Definition 5.10 *Given an indentation alphabet* $\langle \Sigma,\ D,\ R \rangle$*, with* R *symbol-oriented, and* $\alpha \in \Sigma^*$*, the* R-Extension of α*, written* $E_R(\alpha)$*, is given by* $\{\beta | \alpha \succeq_R \beta\}$*.*

The R-Extension of α thus consists of everything α subsumes, *i.e.*, the set of sequences that may match a pattern of α.

Definition 5.11 *An indentation alphabet* $\langle \Sigma,\ D,\ R \rangle$*is said to be* block-oriented *if the following are true:*

- *R is symbol-oriented.*
- *Σ contains the symbols Start-Block and End-Block, which are fully distinguishable from all other symbols in Σ.*
- *$\Sigma - \{$Start-Block, End-Block$\}$ contains at least one pair of fully distinguishable symbols.*

This describes the kind of indentation alphabets appropriate for this kind of segmentation. To seek patterns that include only whole blocks, each pattern is required to begin with Start-Block and end with End-Block. This leads to the following definition of the patterns to be found.

Definition 5.12 *Given a block-oriented indentation alphabet* $I = \langle \Sigma,\ D,\ R \rangle$*, the language of its* block indentation patterns P_I *is given by:*

$$P_I = \{\alpha_1 \alpha_2 \ldots \alpha_n \mid (n > 1) \wedge$$
$$(\exists \beta \in \textit{Start-Block}\ \ \Sigma^*\ \textit{End-Block}:$$
$$(\forall i, 1 \leq i \leq n:\ \ (\alpha_i \in E_R(\beta)))\}$$

Definition 5.13 *For any block-oriented indentation alphabet $I = \langle \Sigma,\ D,\ R \rangle$with block indentation pattern language P_I and sequence of text blocks $B = \{b_1, \ldots b_n\}$, where the representation in I of the contours of a text block b_i is given by $r_I(b_i)$, the language of* text indentation patterns P_{I_t} *is given by*

$$P_{I_t} = \{b_j\ b_{j+1}\ \ldots b_{k>j} | r_I(b_j)\ r_I(b_j + 1)\ r_I(b_k) \in P_I\}$$

At first glance, parsing might seem like a natural method of finding patterns in this language. Unfortunately, the following result leads us to reject this approach.

Theorem 5.1 *For any block-oriented indentation alphabet $I = \langle \Sigma,\ D,\ R \rangle$, P_I is not a context-free language.*

Proof: Application of the pumping lemma for context-free languages to the string Start-Block $(xy)^n$ End-Block Start-Block $(xy)^n$ End-Block Start-Block $(xy)^n$ End-Block in which x, y are fully distinguishable elements of $\Sigma - \{$Start-Block, End-Block$\}$. Consider pumping down once. $\Box$

This language is, however, context-sensitive. This result has computational implications for the recognition of the indentation structures described here. Since context-sensitive parsing is not sufficiently efficient for use in the domain of large documents, the documents are not parsed. Instead, an obvious quadratic algorithm is used that is guaranteed to find some repeated pattern if any exist. Specifically, the algorithm moves through the list of symbols in the following manner. The current first symbol is identified; the list is scanned for the next object that may match it. (Since all matches must use whole blocks, the first current first symbol is considered to be the first symbol of the current block, and the next object that may match it must be the first symbol of its own block, as well as belonging to a type that may match.) The sequence of objects up to the match is compared to the objects following the match; if these lists match successfully, a pattern has been found. In this case, the following part of the list is checked for any repetitions of this pattern. Otherwise, this pattern is rejected and the list is scanned for the next possible match of the first symbol, and the process is repeated until a match succeeds or the list is exhausted. After a successful match, all identified patterns are dropped from the current list, and the process repeats. If no pattern is found, the first symbol is popped from the list (considered a pattern of length one), and the process repeats with the new first symbol. This algorithm is described below, in terms of a sequence s with symbols $s[1]$ through $s[n]$, placing information about the patterns in a structure called *results*.

Algorithm 5.2

```
initialize i = 1, results is empty
WHILE (i < n) {
  find first j > i such that s[i], s[j] may match
  WHILE j exists {
```

```
    compare s[i...j-1] with s[j...]
    IF s[i...j-1] matches s[j...k] with least upper bound α {
      add s[i...j-1], s[j...k] to results as instantiations of α
      DO {                                      /* Find any more matches to this */
        find instantiation of α starting at s[k+1]
        generalize α if necessary for the match
        add instantiation to results
        k = final position of instantiation
      } UNTIL no more instantiations can be found
      i = k+1                                   /* Start over looking for matches */
      j = first j > i such that s[i], s[j] may match
    } ELSE                                      /* This match didn't work. */
      j = next j such that s[i], s[j] may match
  }
  i = i+1
}
```

Lemma 5.2 *The above algorithm is $O(n^2)$, when applied with an indentation alphabet in which whether a pair of symbols matches may be decided immediately on comparing them.*

Proof: In the worst case, the outer `while` loop will occur $n-1$ times, once for each possible value of i. Now, consider the worst case at each iteration. If the first symbol is x, represent all symbols that match it as x. The second symbol must not match x in the worst case, so call it y. The worst case is a level of x, y, followed by $n-2$ symbols x. In this case, every symbol after the second will match, causing an attempted sequence match, each of which will fail. Each sequence match will fail after a single comparison, so the total number of comparisons is $n-1$ initial comparisons $+\ n-2$ next comparisons in attempted sequence matches, which yields $2n-3$ comparisons. This is the worst case, because any change that adds a symbol comparison to a sequence comparison also removes a sequence comparison, thus leaving the total number of comparisons equal. So the worst case for an iteration is $O(n)$, which means that the worst case for the whole algorithm is $O(n^2)$. □

Theorem 5.3 *With* LABLER*'s indentation alphabet, the above algorithm is $O(n^2)$.*

Proof: The only matches that cannot be decided immediately in LABLER's indentation alphabet are of symbols of the form $\langle \mathsf{c},\ m\rangle$ with symbols of the form x, where the former may be matched by a sequence of m instances of x or $k < m$ instances of x together with an instance of $\langle \mathsf{c},\ m'\rangle$. This will add comparisons to an iteration in the case that the first symbol is of the form $\langle \mathsf{c},\ m\rangle$ and it is followed by $k < m$ symbols x, not followed by $\langle \mathsf{c},\ m'\rangle$. In this case, $O(k^2)$ comparisons will be added, since k times, a sequence match will be attempted and not abandoned until all x's following the current one have been used. However, at the next iteration, a match of the k x's will be found, removing k iterations, each with a

Start-Block Left-Justified End-Block
Start-Block $\langle$Blank, 2$\rangle$ End-Block
Start-Block $\langle$Left-In, $p_1\rangle$ End-Block
Start-Block $\langle$Left-Out, $p_1\rangle$ End-Block
Start-Block $\langle$Blank, 1$\rangle$ End-Block
Start-Block $\langle$Left-In, $p_2\rangle$ $\langle$ Left-In, $p_3\rangle$ End-Block
Start-Block $\langle$Blank, 1$\rangle$ End-Block
Start-Block $\langle$Left-In, $p_2\rangle$ $\langle$Left-In, $p_3\rangle$ End-Block
Start-Block $\langle$Blank, 1$\rangle$ End-Block
Start-Block Left-Justified End-Block
Start-Block $\langle$Left-In, $p_1\rangle$ End-Block
Start-Block $\langle$Left-Out, $p_1\rangle$ End-Block
Start-Block $\langle$Blank, 1$\rangle$ End-Block
Start-Block $\langle$Left-In, $p_2\rangle$ $\langle$Left-In, $p_3\rangle$ End-Block
Start-Block $\langle$Blank, 1$\rangle$ End-Block
Start-Block $\langle$Left-In, $p_2\rangle$ $\langle$Left-In, $p_3\rangle$ End-Block
Start-Block $\langle$Blank, 1$\rangle$ End-Block
Start-Block Left-Justified End-Block
Start-Block $\langle$Left-In, $p_1\rangle$ End-Block
Start-Block $\langle$Left-Out, $p_1\rangle$ End-Block
Start-Block $\langle$Left-In, $p_1\rangle$ End-Block
Start-Block $\langle$Left-Out, $p_1\rangle$ End-Block

Figure 5.4: Representation of the initial blocks of Figure 5.1

worst case of $>= k$ comparisons. Thus, $O(k^2)$ comparisons are removed from the worst case possibility by this sequence, so the worst case remains the same as if all matches can be decided immediately. By Lemma 5.2, this is $O(n^2)$. □

5.3 A Segmentation Example

This section presents an example application of the logical hierarchy generation algorithm. The algorithm is applied to the first section of the miniature document in Figure 5.1.

The initial blocks of the document correspond to the leaves of the tree in Figure 5.2; their indentation alphabet representation is given in Figure 5.4. This representation contains two repeated patterns: a block of $\langle$Left-In, $p_1\rangle$ followed by a block of $\langle$Left-Out, $p_1\rangle$ and a single block of $\langle$Left-In, $p_2\rangle$, $\langle$Left-In, $p_2\rangle$. These patterns correspond to the shapes of the paragraphs and list items, respectively. The third and fourth paragraphs offer the repetition of their pattern; the initial parts of the first and second paragraphs also match it, including the distances of the Left-In and Left-Out, so they are isolated elements of the pattern. Both

Start-Block	Left-Justified	End-Block		
Start-Block	⟨Blank, 2⟩	End-Block		
Start-Block	⟨Left-In, p_1⟩	⟨Left-Out, p_1⟩	End-Block	
Start-Block	⟨Left-In, p_2⟩	⟨Blank, 1⟩	⟨Left-In, p_2⟩	End-Block
Start-Block	Left-Justified	End-Block		
Start-Block	⟨Left-In, p_1⟩	⟨Left-Out, p_1⟩	End-Block	
Start-Block	⟨Left-In, p_2⟩	⟨Blank, 1⟩	⟨Left-In, p_2⟩	End-Block
Start-Block	Left-Justified	End-Block		
Start-Block	⟨Left-In, p_1⟩	⟨Left-Out, p_1⟩	End-Block	
Start-Block	⟨Left-In, p_1⟩	⟨Left-Out, p_1⟩	End-Block	

Figure 5.5: Representation of the blocks of Figure 5.1 after combining pattern elements

Start-Block	Left-Justified	End-Block	
Start-Block	⟨Blank, 2⟩	End-Block	
Start-Block	Left-In	Left-Out	End-Block
Start-Block	Left-In	Left-Out	End-Block
Start-Block	Left-In	Left-Out	End-Block
Start-Block	Left-In	Left-Out	End-Block

Figure 5.6: Representation of the blocks of Figure 5.1 after merging

lists form repetitions of the list item pattern. Grouping according to Step 3 yields a level made up of the following: the heading, the start of the first paragraph, the first list, the end of the first paragraph, the start of the second paragraph, the second list, the end of the second paragraph, the third paragraph, and the fourth paragraph. The blocks of this level are represented in the indentation alphabet as shown in Figure 5.5.

The merging step finds that the lists interrupt their surrounding paragraphs, yielding a level consisting of the heading and the four paragraphs. Since merging combinations of nodes retains the current level of granularity and does not introduce the effects of the interrupters, this yields the representation in Figure 5.6.

The next step finds a pattern of four repeated elements, each consisting of a single block of the form Left-In, Left-Out. These are combined, yielding a level that consists of the heading and the section body. Finally, these two blocks are the vertically closest to each other, since they are the only blocks, so they are combined to form the root level of the whole section.

5.4 Classification Effects

The preceding sections describe the segmentation algorithm as it is used to build a hierarchy without intervening classification steps. In the actual system, not only are the nodes of each level classified immediately upon formation of that level, but the results of this classification also provide feedback for the segmenter; some areas of the level may be resegmented as a result. Moreover, the algorithm itself is adjusted slightly to make use of the classification results.

Step 5b is adjusted to exploit classification information. When sequences of nodes that form patterns are grouped together to form new nodes, so are sequences of isolated nodes that share a classification. Thus, for example, several *theorems* in a row will be grouped together into their own structure. Additionally, classifications may be specified as ungroupable; blocks with these classifications will not be combined with any other blocks until a specified step has been reached. LABLER saves blocks of the title part until the vertical spacing step, and it saves headings for special processing into sections at the end, in order to form the sections according to the numbering scheme of the headings.

Finally, the classifier provides feedback, once the segmentation algorithm has been performed. If one or more adjacent segments cannot be convincingly classified, an attempt is made at resegmentation. When the classifier has been trained, as will be described in Section 6.3, convincing classification means that the text block has a similarity to the chosen structure of at least .5.[5] Resegmentation may alter up to one structure at each end of the run of badly classified segments. The first attempt is to group sub-blocks by vertical proximity, rather than repeated shapes. That is, all text sub-blocks in the considered area that are separated by blank space of the minimum height found within the area are joined to form new blocks. If this does not improve the classification overall, then resegmentation is attempted first by extending the original blocks, and then by regrouping the sub-blocks from scratch. If these desperate measures are called for, then all possible groupings are considered; this is theoretically a costly process (exponential time), but in practice the sequences of bad classifications are quite short, so this does not pose a large problem.[6] Also, hierarchy leaves (blocks formed to start with by the external algorithm) that yield no useful classification are removed from the final tree; these blocks are presumed to be parts of basic structures, and they may be needed to find the patterns of which the basic structures form elements, but once this purpose has been served, they are superfluous.

[5]Without such a basis, a cutoff similarity must be chosen based on some kind of heuristic.

[6]Also, in practice, this kind of wholesale resegmentation is rarely necessary.

Chapter 6

Structure Classification

Classification of primary logical structures proceeds based on comparison of the text blocks to predefined structure prototypes. The prototype of a structure describes the cues that would typically identify this structure. This does not imply that *all* indicated cues would typically be present, but rather that *any* of them might commonly contribute to recognition of the structure.

The prototypes rely heavily on geometric cues. The use of geometry in the segmenter makes this emphasis natural; heavier use of shape information than characters and words also suits a system designed to use the results of OCR, since OCR cannot yet provide assurance of every character. All the cue types (except *internal shape*) discussed in Chapter 3 are included, however, since they can be necessary in some cases and highly useful in others.

A prototype consists of a set of attribute/value pairs. Each attribute describes some facet of a text block that falls into the domain of the observables discussed in Chapter 3. (These do not, however, have a one-to-one relationship; their correspondences are illustrated in Figure 6.1.) Some attributes have sub-attributes, and some values may remain unspecified. Section 6.1 describes the attributes and their possible values.

A text block is compared to a prototype by finding the distance between the two with respect to each attribute, according to the measurement discussed in Subsection 6.1.2. These distances are combined to yield the total distance. Each segment of text is classified by assigning it the structure of the prototype to which it is nearest (*i.e.*, most similar). Thus, primary classification can be described as follows. Given a set of prototypes $P = \{p_1, \ldots, p_n\}$, corresponding to classes $C = \{c_1, \ldots, c_n\}$ respectively, and given a distance function D, the classification function Cl (applied to a text block b) can be defined:

$$Cl(b) = c_i \text{ s.t. } (D(b, p_i) = \min_{p_j \in P} D(b, p_j))$$

This process is discussed in detail in Section 6.2, along with the process of classifying secondary structures based on these primary results. Section 6.3 presents a method of reflecting the various degrees of importance of the attributes in a prototype.

6.1 Logical Structure Prototypes

The prototypes proposed here for logical structures comprise the following attributes: `shape`, `context`, `height`, `symbols`, `fonts`, and `children`. Any attribute may remain unspecified in a prototype; all segments are considered to match an unspecified attribute. Each attribute is described in detail, together with any subattributes, in Subsection 6.1.1. Figure 6.1 summarizes the attributes and subattributes, together with their relationships to the observable categories defined in Chapter 3. The complete lists of resulting observables and prototypes for primary structures are given in Appendix A.

The distance measurement is described in Subsection 6.1.2. The details of the resulting algorithm are supplied in Section 6.2. Sample existing prototypes are shown in Figure 6.2

6.1.1 Prototype Attributes

The Shape Attribute

The `shape` of a document segment is a representation of its text contours in the indentation alphabet described in Chapter 5. This belongs to the *contours* subcategory of *geometric* observables in Chapter 3.

The Context Attribute

The `context` attribute enables a small degree of context-dependence in the classification algorithm; it provides a description of a prototypical predecessor and successor,[1] as well as a special `typicality` attribute if appropriate. The `predecessor` and `successor` subattributes belong to the *local context* category of observables; `typicality` belongs to the *global context* category.

The `predecessor` and `successor` subattributes consist of the following descriptive subattributes.

- `Shape` Again, this is the representation of a text block in the indentation alphabet described in Chapter 5.
- `Sameness` This indicates whether the `geometry` of the predecessor/successor is prototypically the same as that of the segment in question. For example, a *list item* is typically geometrically similar to both its predecessor and successor; a *special paragraph* is not. (These expectations are violated in the case of the first and last items in a *list*, as well as by a run of adjacent *special paragraphs*; this fact highlights the differences in meaning and basis for inclusion between the *cues* indicated by prototypes and the *requirements* imposed by parsing-based systems.) Font is also considered in this comparison, when the information is available.

[1]This refers to the actual predecessor and successor of a node within its level; if the node has a left (right) sibling, it will be the predecessor (successor), but the sibling relationship or its lack does not affect the analysis described here.

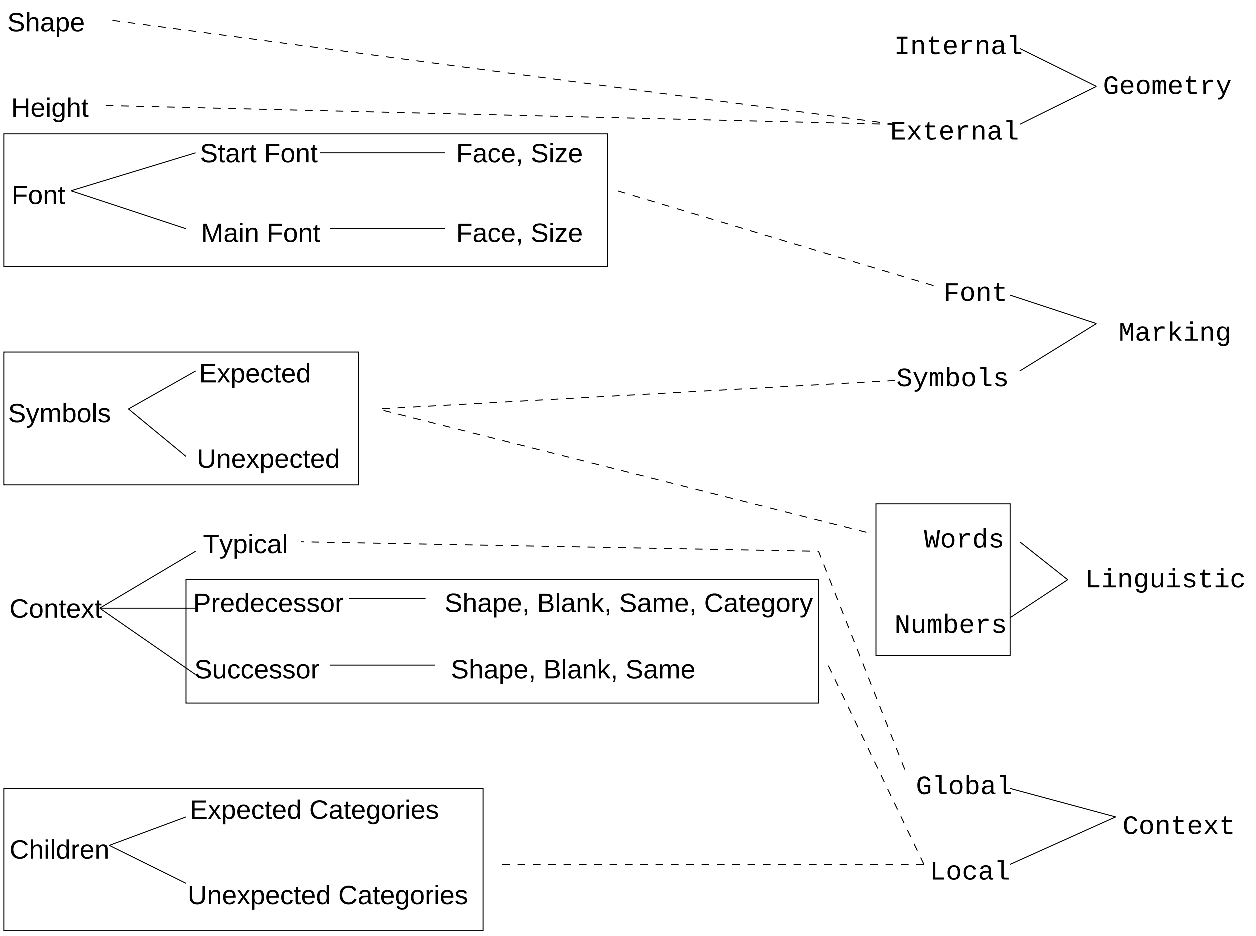


Figure 6.1: Attributes and their observable types. Attributes that belong to the same observable have been grouped together in boxes. Multiple observables required to cover the same attribute have also been grouped in boxes. Attributes are connected to their observables by dashed lines.

- `Blank` This indicates whether a blank space (greater than the interline spacing of the page) should exist between the text under consideration and its predecessor/successor.

- `Category` This applies only to the predecessor of a segment, and it describes the classification that would prototypically be assigned to it. For example, the `category` of the predecessor of a prototype *proof* is *theorem*, where *theorem* includes theorems and lemmas.

The typicality attribute specifies whether the structure is expected to have the shape typical of a relevant (specified) ancestor in the structure tree (Figure 3.2). For instance, a *theorem* is a *special paragraph* and is therefore expected to differ in shape from the typical *paragraph* within the document. The method of determining typicality is described in Subsection 6.1.2.

The Height Attribute

The `height` specifies the typical length of the structure, measured in lines. For instance, a *heading* has a typical height of 1. (Any value *may* be specified for the `height`, but in practice this is only used for structures expected to consist of a single line.) As mentioned in Chapter 3, this attribute is a special case of the *contours* subcategory of the *geometry* category of observables.

The Symbols Attribute

The `symbols` attribute consists of a description of typically appearing or non-appearing symbols. It includes both non-linguistic symbols and the small amount of word-based processing included in the classification step. It thus combines the *symbols* subcategory of the *marking* category and the *linguistic* category of observables of Chapter 3. This attribute consists of a list of positive and/or negative expectations. (Positive expectations typically appear; negative expectations typically do not.) Each of these specifications includes a list of symbols (or words) and a position indicator, which may be one of: `start`, `end`, `any`, `all`. A positive (negative) expectation with a position of `start` indicates that one of its symbols should (should not) start the structure; `end` refers to ending the structure; `any` indicates an expectation about whether the symbol(s) will appear anywhere in the text segment.

`All` functions a bit differently; it indicates an expectation that each *word*, defined as a space-delimited string of characters, in the text segment will (or will not) contain one of the specified symbol strings. Naturally, the symbol strings specified for `all` may not contain the space character, ␣. (In all discussions below, a space-delimited word must never contain ␣; a space-delimited word within a given string is a substring that falls between two ␣ characters or is bounded at one end by an end of the string and at the other by ␣).

For example, headings typically do not end with a punctuation mark that may end a sentence;[2] hence, the *heading* prototype includes a negative expectation of ⟨ {., !, ?}, `end` ⟩.

[2]Sometimes they do; this indicates the value of using prototypes, rather than strict definitions.

The string list in an expectation must not be $\emptyset$ or a superset of Σ, as these choices always lead either to an expectation that is impossible to meet or to an expectation that is always met. Moreover, none of the strings may consist only of $\sqcup$.

For `start` and `end`, including positive and negative expectations is a notational convenience; given a finite alphabet of recognizable symbols, the negative expectations can be expressed in terms of positive expectations. Let expectations be expressed as above, with an additional tuple element of `P` or `N`, to indicate positive or negative expectations, respectively. Let s be a string of length k, $s = s_1, ..., s_k$, and let the alphabet of symbols be Σ. Using the notation $a\Sigma$ as a shorthand for the set of strings formed by concatenating the symbol a with each individual symbol in Σ, we have the following.

$$\begin{aligned}\langle\ \mathtt{N}, \{s\}, \mathtt{start}\ \rangle &= \langle\ \mathtt{P}, (\Sigma - \{s_1\}) \cup\ (s_1(\Sigma - \{s_2\}))\ \cup\ (s_1 s_2(\Sigma - \{s_3\})) \\ &\quad \cup\ \ldots\ \cup\ (s_1, \ldots s_{k-1}(\Sigma - \{s_k\})),\ \mathtt{start}\ \rangle\end{aligned}$$

$$\begin{aligned}\langle\ \mathtt{N}, \{s\}, \mathtt{end}\ \rangle &= \langle\ \mathtt{P}, (\Sigma - \{s_k\}) \cup\ ((\Sigma - \{s_{k-1}\})s_k) \\ &\quad \cup ((\Sigma - \{s_{k-2}\})s_{k-1}s_k)\ \cup\ \ldots \\ &\quad \cup ((\Sigma - \{s_1\})s_2 \ldots s_k),\ \mathtt{end}\ \rangle\end{aligned}$$

Now, let S represent a group of strings, in which the maximum length is m, and let $P_i(s)$ represent the symbol in position i of string s. Let $P_i(S) = \cup_{s \in S} P_i(s)$. Let $\Sigma_1\Sigma_2$ represent all strings formed by selecting one symbol from Σ_1 and following it with a symbol from Σ_2.

$$\begin{aligned}\langle\ \mathtt{N}, S, \mathtt{start}\ \rangle &= \langle\ \mathtt{P},\ (\Sigma - P_1(S))\ \cup\ (P_1(S)\ (\Sigma - P_2(S)))\ \cup\ \ldots \\ &\quad \cup (P_1(S)\ P_2(S)\ \ldots\ P_{m-1}S\ (\Sigma - P_m(S))),\ \mathtt{start}\ \rangle \\ \langle\ \mathtt{N}, S, \mathtt{end}\ \rangle &= \langle\ \mathtt{P},\ (\Sigma - P_m(S))\ \cup\ ((\Sigma - P_{m-1}(S))\ P_m(S))\ \cup\ \ldots \\ &\quad \cup ((\Sigma - P_1(S))\ P_2(S)\ \ldots\ P_{m-1}(S)\ P_m(S)),\ \mathtt{end}\ \rangle\end{aligned}$$

That is, negative expectations may be expressed by enumerating in positive expectations the choices they do not exclude.

For `any` and `all`, however, the negative expectations are more than a notational convenience; they add value.

Proposition 6.1 $\forall S \neq \emptyset\ (\not\exists S'$, `position` $\in \{$ `start`, `end`, `any`, `all`$\}$ $(\langle\ \mathtt{N},\ S,\ \mathtt{any}\ \rangle = \langle\ \mathtt{P},\ S', \mathtt{position}\ \rangle))$.

Proof: Suppose such an S' and `position` exist. Consider each possible value of `position`.

Case 1 `Position is start.`

Consider $s \in S, s' \in S'$. Consider the string $s's$. This meets the expectation $\langle\ \mathtt{P},\ S', \mathtt{start}\ \rangle$, but it violates the expectation $\langle\ \mathtt{N}, S, \mathtt{any}\ \rangle$.

Case 2 `Position` is `end`.
Consider $s \in S, s' \in S'$. Consider the string ss'. This meets the expectation $\langle$ `P`, S', `end` $\rangle$, but it violates the expectation $\langle$ `N`, S, `any` $\rangle$.

Case 3 `Position` is `any`.
Consider $s \in S, s' \in S'$. Consider the string ss'. This meets the expectation $\langle$ `P`, S', `any` $\rangle$, but it violates the expectation $\langle$ `N`, S, `any` $\rangle$.

Case 4 `Position` is `all`.
Consider cases of the contents of S' and S.

Case 4a $\exists s \in S \cap (\Sigma - \sqcup)^*$
Consider s above and $s' \in S'$. Consider the string ss'. This meets the expectation $\langle$ `P`, S', `all` $\rangle$, but it violates the expectation $\langle$ `N`, S, `any` $\rangle$.

Case 4b $\exists s \in S$ s.t.
$\forall s_w$ (s_w a space-delimited word within $s \Rightarrow s_w \in S'$)
Consider the string s from above. This meets the expectation $\langle$ `P`, S', `all` $\rangle$, but it violates the expectation $\langle$ `N`, S, `any` $\rangle$.

Case 4c $\forall s \in S$
(s contains $\sqcup \;\wedge\; \exists s_w$ a space-delimited word within s s.t. $s_w \notin S'$)
Consider some string $s \in S$, and consider the string consisting only of its space-delimited substring $s_w \notin S'$. This violates the expectation $\langle$ `P`, S', `all` $\rangle$, but it meets the expectation $\langle$ `N`, S, `any` $\rangle$.

Each subcase leads to a contradiction, so this case leads to a contradiction.

Each case leads to a contradiction. Thus, the supposition leads to a contradiction, so the S' and `position` do not exist. $\Box$

Proposition 6.2 $\forall S \neq \emptyset$
$$(\nexists S', \texttt{position} \in \{\texttt{start}, \texttt{end}, \texttt{any}, \texttt{all}\}$$
$$((\langle\ \texttt{N},\ S,\ \texttt{all}\ \rangle = \langle\ \texttt{P},\ S',\ \texttt{position}\ \rangle))).$$

Proof: Suppose such an S' and `position` exist. Consider each possible value of `position`.

Case 1 `Position` is `start`.
Consider $s \in S, s' \in S'$. Consider the string $s's$. This meets the expectation $\langle$ `P`, S', `start` $\rangle$, but it violates the expectation $\langle$ `N`, S, `all` $\rangle$. Thus, the assumption that the two expectations are equal leads to a contradiction.

Case 2 `Position` is `end`.
Consider $s \in S, s' \in S'$. Consider the string ss'. This meets the expectation $\langle$ `P`, S', `end` $\rangle$, but it violates the expectation $\langle$ `N`, S, `all` $\rangle$. Thus, the assumption that the two expectations are equal leads to a contradiction.

Case 3 `Position` is `any`.
Consider $s \in S, s' \in S'$. Consider the string ss'. This meets the expectation $\langle$ `P`, S', `any` $\rangle$, but it violates the expectation $\langle$ `N`, S, `all` $\rangle$. Thus, the assumption that the two expectations are equal leads to a contradiction.

Case 4 `Position` is `all`.
Consider $s \in S, s' \in S'$. Consider the string ss'. This meets the expectation $\langle$ `P`, S', `all` $\rangle$, but it violates the expectation $\langle$ `N`, S, `all` $\rangle$. Thus, the assumption that the two expectations are equal leads to a contradiction.

Each case leads to a contradiction. Thus, the supposition leads to a contradiction, so the S' and `position` do not exist. $\square$

The Fonts Attribute

The `fonts` attribute consists of expectations about the fonts in a particular structure, described in terms of the predominant font in the document. This belongs to the *font* subcategory of the *marking* category of observables in Chapter 3.

This attribute has two subattributes: `start` and `main`. The `start font` attribute describes the expected font to begin the structure; the `main font` describes the font expected to predominate within the structure. The descriptions are not absolute, but rather given by comparison to the main font of the document, and they specify characteristics of the font `face` and `size`. The `face` of a font may be expected to be heavier than the typical or not, and the `size` of a font may be expected to be the same as typical (*i.e.*, normal), larger, or smaller. For example, the *theorem* prototype includes the specification of `start font` as ⟨heavy, normal⟩.

The Children Attribute

The `children` attribute describes the typical or atypical categories of the hierarchical children of a given structure. It thus constitutes a special case of the *local context* category of observables of Chapter 3. It consists of a list of positive and negative expectations, each of which is a list of children. A positive expectation indicates that the structure should have a child that is a member of its category list; a negative expectation indicates that the structure should not. (Observe that an empty category list makes a positive expectation impossible to meet and makes a negative expectation always hold vacuously.)

For instance, a *list*'s children are typically list items or sub-lists; moreover, any other categories of children are atypical and contribute to a structure's distance from the *list* prototype.[3] Hence, the `children` attribute of the *list* prototype includes the positive expectation {*list item*, *list*} and the negative expectation Σ − {*list item*, *list*}, where Σ is the set of all possible structure categories.

[3] A *list* is not, however, considered a secondary structure; a structure with many children, of which one is a *paragraph* and the rest *list items* is probably a list that contains an interruption.

Paragraph

`Geometry`: Left In Left Out

`Context`: **Pred.:** `Blank`: yes
Succ.: `Blank`: yes

First Paragraph

`Geometry`: Left In Left Out

`Context`: **Pred.:** `Blank`: yes, `Same`: no, *`Category`: *heading*
Succ.: `Blank`: yes, `Same`: no

Heading

`Context`: **Pred.:** `Same`: no, `Blank`: yes
Succ.: `Same`: no, `Blank`: yes*

`Height`: 1

`Symbols`: **Positive:** $\langle$ {1, ... 9}, `start` $\rangle$
Negative: $\langle$ {., !, ?}, `end` $\rangle$

`Font`: **Main:** $\langle$heavy, large$\rangle$

Figure 6.2: Sample prototypes. Attributes with a * immediately preceding their name and specification are marked as necessary. Note that the *paragraph* prototype includes both the shape cue for indented paragraphs and the cue of surrounding white space, often used without indenting. *First paragraph* is a type of *special paragraph*, a paragraph that differs from its surrounding context.

The inclusion of negative as well as positive expectations adds expressiveness to the children attribute. Expressing children expectations as pairs of `P` or `N` (corresponding to positive or negative expectations) and a list of structure categories, we have the following result.

Proposition 6.3 $\nexists C_1, C_2(\langle P, C_1\rangle = \langle N, C_2\rangle)$.

Proof: Suppose such a C_1 and C_2 exist. Consider a structure with no children. It meets the expectation $\langle N, C_2\rangle$, but it does not meet the expectation $\langle P, C_1\rangle$. This contradicts the assumption that the expectations are equal. Thus, such a C_1 and C_2 cannot exist. $\square$

6.1.2 Distance Measures

The measurement of the distance from a segment to a prototype is given on a scale from 0 to 1 by a combination of the following attribute distances (each of which is normalized to fall within the interval $[0,1]$.) An unspecified attribute always yields distance 0. Attributes may also occasionally be marked as necessary; a non-zero distance from a necessary attribute automatically generates a distance of 1 from the prototype.

The distances with respect to the attributes are combined to form the total distance. Thus, given an attribute set $A = \{a_1, \ldots, a_m\}$ with corresponding distance functions $\{d_1, \ldots d_m\}$ and a set of weights $\{w_1 \ldots, w_m\}$, the distance between a prototype p and a text block b is given by:

$$D(p,b) = \sum_{i=1}^{m} (w_i)(d_i(a_i, b)) \ .$$

The immediately following description discusses attributes and subattributes, with the distances of subattributes contributing to the distance of their containing attribute. This is equivalent to combining all lowest-level subattributes directly, with the appropriate set of weights. That is, if each attribute a_i has s_i subattributes $sa_{i,1} \ldots sa_{i,s_i}$, each of which contributes to d_i with a distance function $d_{i,j}$ and weight $w_{i,j}$, $1 \leq j \leq s_i$, then we have the following.

$$\begin{aligned} D(p,b) &= \sum_{i=1}^{m} (w_i)(d_i(a_i, b)) \\ &= \sum_{i=1}^{m} ((w_i) \sum_{j=1}^{s_i} (w_{i,j})(d_{i,j}(sa_{i,j}, b))) \\ &= \sum_{i=1}^{m} \sum_{j=1}^{s_i} ((w_i)(w_{i,j})(d_{i,j}(sa_{i,j}, b))) \end{aligned}$$

We can reparameterize as follows. Let $n = \sum_{i=1}^{m} s_i$, and let $f(k) = (i,j)$ such that $\sum_{q=1}^{i-1} s_i + j = k$. Then, given $k, f(k) = (i,j)$, let $w'_k = (w_i)(w_{i,j})$, let $d'_k = d_{i,j}$, and let $sa'_k = sa_{i,j}$. This yields:

$$D(p,b) = \sum_{k=1}^{n} (w'_k)(d'_k(sa'_k, b))$$

The attribute distances are defined as follows.

`Shape` The gross geometric distance between a text block and a structure prototype is given by the Levenshtein distance [79] between the indentation alphabet representations. The Levenshtein distance is given by a weighted sum of the minimum inserts, deletes, and (optionally) changes required to convert one string to another (the minimum here is with respect to the total weighted sum); in this case, inserts and deletes are weighted equally, and changes are disallowed. This is normalized by dividing by the sum of the lengths of the two representations.

`Context` The context distance is a combination of its predecessor and successor distances and its typicality distance. The predecessor and successor distances combine their sub-attribute distances, which are described below.

`Shape` This distance is precisely as described above.

`Sameness` This distance is 0 if the sameness expectation is satisfied; otherwise it is 1.

`Blank` This distance is 0 if the expectation is satisfied; otherwise it is 1.

`Category` This distance is 0 if the expectation is satisfied; otherwise it is 1.

The `typicality` distance is 0 if the typicality is as expected and 1 otherwise. It is determined in the following manner. First, classification is performed without considering this sub-attribute. Then, for each structure type whose subtypes may consider typicality (*e.g.*, *paragraph*, since typicality is relevant to *special paragraphs*), the typical shape at this level is found. The most common shape among segments classified with this structure or its descendents whose identification as typical will allow some structures to match prototypes expected to be non-typical is chosen as the typical shape. This allows the distance to be found, and classification is performed again for all segments that may be affected.

`Height` The gross height distance is given by the difference in height between the prototype and the text segment; this is normalized by dividing by the maximum of the two heights. Thus, a difference of one or two lines, while quite significant in a structure expected to be short, will be relatively insignificant in a structure expected to be large. (As mentioned in Subsection 6.1.1, however, the only prototypes that use `height` expect a value of 1.)

`Symbols` The gross symbol distance is given by the number of unmet symbol expectations, *i.e.*, positive expectations for which none of the symbols is found in the text block at an appropriate position and negative expectations for which one of the symbols is found at the specified position. This distance is normalized by dividing by the total number of expectations.

`Fonts` The font distance is given by a combination of the `start font` distance and the `main font` distance. Each of these is defined as the number of font characteristics that violate expectations, divided by the number of characteristics considered. Since the prototypes only consider 2 font characteristics, this means that the font distance is 0 if both face and size are as expected, .5 if face or size (but not both) meets the expectation, and 1 if neither does.

`Children` The children distance is a combination of the distance due to positive expectations and that due to negative expectations, which are determined slightly differently. The gross positive expectation distance is given by the number of *unmet* positive expectations, *i.e.*, the number of positive expectations for which none of the children match any of the specified categories. This distance is normalized by dividing by the number of positive expectations. The gross negative expectation distance is given by the number of *violations* of the negative expectations, *i.e.*, the number of children that belong to a category given in a negative expectation. This distance is normalized by dividing by the total number of children; hence, a single bad child forms a small deviation in a large group but a large deviation in a small group.[4]

6.2 The Classification Algorithm

The prototypes are designed with easy application in mind; since the only surrounding categories that are considered are of the children of a node and of its predecessor on the same level, classification proceeds as the tree is built from the bottom up, moving through each tree level sequentially immediately after it is formed. Each node is examined and assigned to the category identified with the prototype to which it is nearest, *i.e.*, that from which its distance is minimal. This requires a second pass over structures initially identified to have types for which the `typicality` subattribute of `context` may be relevant.

In the absence of further information about how to combine distance measures, the distance with respect to each attribute is given by the average of its sub-attribute distances; the total distance is the average of main attribute distances. Section 6.3 discusses a weighted approach to distance combination, based on machine learning.

In the case of a tie, the more specific category is chosen. (The more specific category resides at a deeper level in the tree of Figure 3.2.)

Lemma 6.4 *Two identically formatted text blocks that appear in sufficiently similar contexts, without distinguishing linguistic markers, will receive the same primary classification.*

Proof: The classifications are determined by the distances from the prototypes. For each prototype p, a text block b's distance is given by

$$D(p,b) = \sum_{k=1}^{n}(w'_k)(d'_k(sa'_k, b))$$

where each sa'_k is a prototype attribute. For identically formatted text blocks, the contributions of all layout attributes, including geometry and marking, will be the same. Thus, the only attributes that may contribute to a different total distance are contextual and linguistic attributes; if these are sufficiently similar, the results will be the same. $\square$

The above process assigns primary categories only. When this is complete, a second phase begins; this phase assigns secondary categories. First, node repetitions are coordinated, in the following sense. If a sequence of nodes $n_1, n_2, ..., n_k$ can be found such that $k \geq 2$ and

---

[4]For instance, a *paragraph* child in a group with 10 *list item* children will not by itself prevent the parent from matching the *list* prototype, but a *paragraph* child with a single *list item* sibling probably will.

n_{i+1} is the sole descendent of n_i for all $1 \leq i < k$, this sequence forms a repetition that may need to be coordinated. If there is no pair n_i, n_{i+1} such that the assigned classifications of these nodes c_i, c_{i+1} bear a special relationship allowing the latter to be a sole descendent of the former, then the sequence does need to be coordinated. Coordination collapses the nodes into one (with parent the parent of n_1 and children the children of n_k); if the nodes in the repetition belonged to different categories, the replacement node is assigned the category from which the distance was least.

For example, if primary classification generates a hierarchy partially shown in Figure 6.3, then nodes 2, 3, and 4 form a sequence to be coordinated. Node 5 is part of the chain of single descendents, but *items* are allowed to be sole descendents of *lists*, so it does not need to be coordinated. Node 4 has the minimal distance of the group, so it replaces the entire chain, with the parent of node 2 as its parent and node 5 as its child. (Note that if node 2 or 3 had replaced the chain, the process would have to repeat, since node 5 would no longer be exempt from coordination.)

The newly resulting tree is searched for nodes that meet the descriptions of the defined secondary categories, and these nodes are assigned the corresponding categories.

This process will treat identically formatted text blocks identically, unless their contexts or relevant linguistic markers are substantially different. The linguistic caveat is desireable, since the same formatting rule may occasionally be used for different (content-oriented) structures, *e.g.*, *theorem* and *definition*. That treatment may vary with context is unavoidable in a system that seeks to make use of the cues context provides, to help distinguish truly similarly formatted structures from structures that have coincidentally similar appearances.

Theorem 6.5 *Two identically formatted text blocks that appear in sufficiently similar contexts, without distinguishing linguistic markers, will receive the same classification.*

Proof: The original primary classifications will be the same, by Lemma 6.4. Coordination and secondary classification are controlled by context, so if the contexts are sufficiently similar, these will affect the final result the same way. Thus, under these conditions, the classifications will be the same. □

6.3 Machine Learning for Attribute Weights

Some attributes contribute more strongly to the intuitive recognition of a particular structure than others; correspondingly, some attributes should be considered more important components of a particular prototype than others. Similarly, strong indications that a text block belongs to a given structure should sometimes inhibit the match between the block and the prototype of another structure. That is, the partition of n-space formed by classification should consist of regions of differing shapes.

In general, we have no *a priori* reason to assign any particular attribute weights; thus, without additional information, the distances are simply combined in averages reflecting

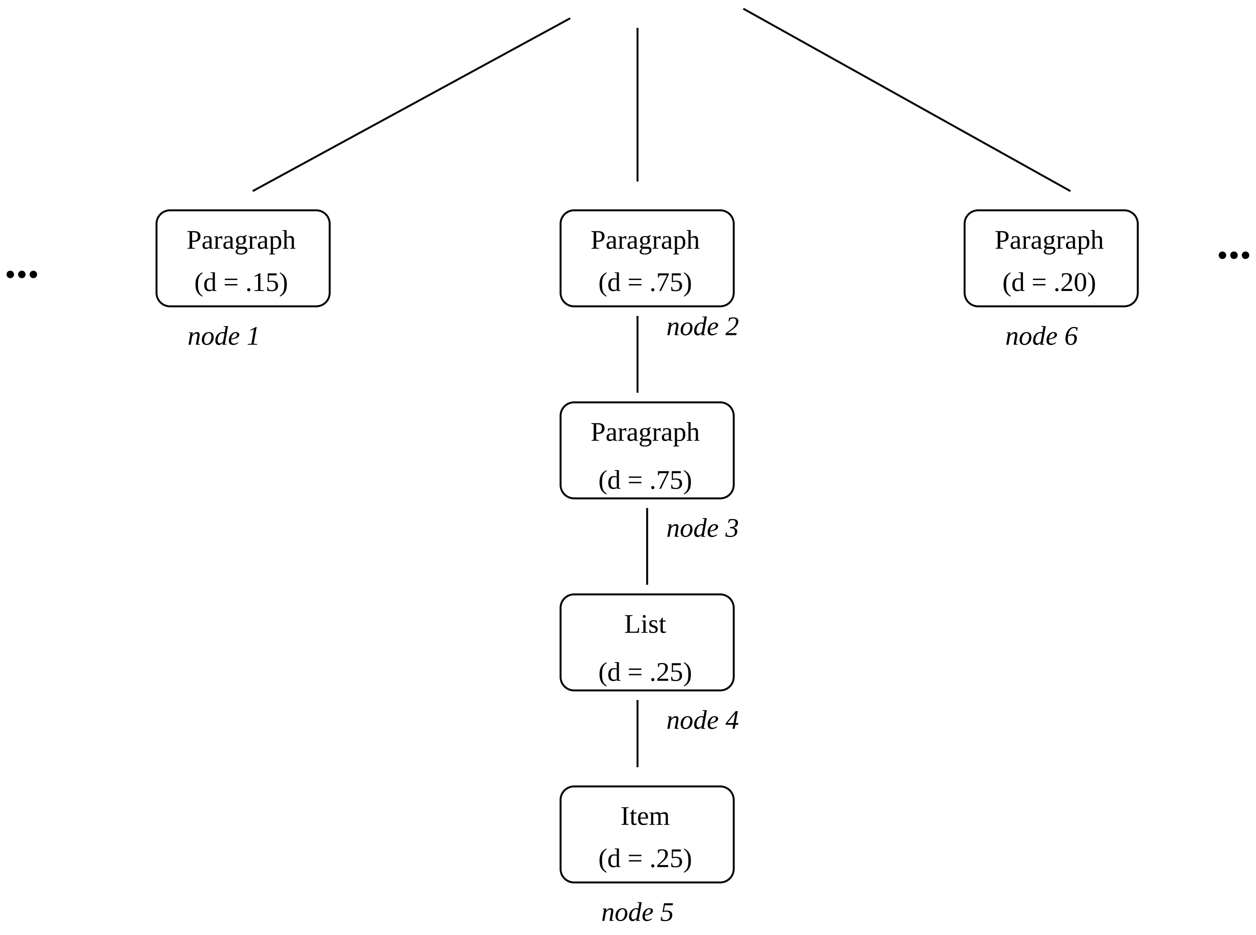


Figure 6.3: A coordination example. Nodes 2, 3, 4, and 5 form a chain of single descendents, the top 3 of which need to be coordinated by collapsing them into node 4.

their hierarchy, as described in Section 6.2. Given access to a set of documents that provide an adequate sampling of the types and styles likely to be processed, however, we can use these as a training set for a machine learning approach to this problem.

The distance function $D(p_i, b)$ applied to a particular prototype and a text block then becomes:

$$D(p_i, b) = \sum_{j=1}^{m}(w_{i,j})(d_j(a_j, b))$$

with a set of weights $\{w_{1,1}, \ldots w_{1,m}, \ldots, w_{n,1}, \ldots, w_{n,n}\}$ corresponding to pairs of prototypes and attributes.

We have a set of values, the distances with respect to each most basic attribute, and we wish to find weights to assign to them, in order to achieve the correct answer with a weighted average. This is precisely the problem that perceptrons (*i.e.*, connectionist networks without hidden nodes) are designed to solve. Perceptrons function by performing a gradient descent on the error space defined by the activations of the output nodes, over each *epoch* of training, in which all training inputs are activated and their results compared to the desired outputs.

Thus, we can assign weights to each attribute/value pair (including even those created for other prototypes) for each prototype by training a single network with an input node for each expected attribute/value pair and an output node for each structure classification. For each training instance, the inputs are the distances with respect to the attribute values, and the correct outputs are 0 for each inappropriate classification and 1 for each appropriate classification. (Usually, exactly one classification will be appropriate, but not always; in the case of *special paragraph*s, both the classification *paragraph* and that of the particular type, *e.g.*, *theorem* are appropriate.)

Since some attributes involve the results of earlier classification, however, the learning task acquires a bit of subtlety. Simply using the correct values for these attributes can lead to networks that rely heavily on this correctness; the result is that when the network is used for classification, any errors are propagated wildly. Furthermore, secondary structure classifications are unavailable at the time primary structure classifications are made, so using their values is clearly inappropriate. Instead, the network is trained in two stages, one of which provides part of the input for the other. The first stage excludes the attributes that require classification information (*children*, *predecessor category*, and *typicality*) and trains the rest of the network in the normal fashion. The next stage uses the results of this partially trained network in order to provide classification values; it changes only the weights from these attributes.

When we have trained the network to provide weights, we then replace the above classification for primary structures with the process of activating the network with the distances of the text block with respect to the attribute values of the input nodes. The output node with the highest activation indicates the choice of structure classification, with one exception: if this choice yields a structure that is a generalization of other structure classifications and one or more of the more specific structures has an activation greater than .5, then the specific structure (*i.e.*, descendent of the original choice in the Figure 3.2 tree) of the highest activation is chosen. (Prototypes with unmet required attributes are eliminated, as before.) This procedure is used in the example of the technical report collection discussed in Chapter 8.

Chapter 7

Specific Knowledge

The preceding chapters discussed the general form of the problem addressed in this thesis, *i.e.*, discovering logical structure in the absence of style knowledge. If, however, some information about the style of a document is available, this knowledge is utilized as described in this chapter.

7.1 Types of Knowledge

The available knowledge may take any of several forms; it also may include a combination of them. If a complete style specification convertible to a CFG with fully describable nonterminals is available, however, the document should by analyzed by one of the parsing approaches discussed in Chapter 2, not the approach in this thesis.

Information about the style of a document falls into three major categories: coordination, structure relationships, and location. The first category refers to the coordination of logical structure with its layout effects. For example, a common specification is that a *paragraph* consists of an indented line followed by a sequence of zero or more lines flush to the left margin; another is that a *heading* consists of one or two lines of heavy, large type. The second refers to the expected relationships among logical structures. For instance, in some settings (but not this thesis), a *section* consists entirely of *subsection*s. Finally, location information specifies the position in the hierarchy expected of a structure or a set of structures. This location may be specified horizontally in terms of layout or vertically in terms of logic. For instance, a title page appears at the far left of the hierarchy (corresponding to the first page of the document, a layout specification), and it appears at the top (corresponding to the most general logical divisions).

Any of these types of information may specify conditions that are necessary, sufficient, or typical for a given structure. This specification and the knowledge type together determine how the information is used. Observe that a knowledge set including either a necessary and sufficient coordination specification or a necessary and sufficient structure relationship specification for each structure, together with necessary location information for leaf nodes, provides the kind of complete information used by the parsing approaches discussed in Chapter 2.

7.2 Parsing

If the partial style knowledge provided takes a particular form, parsing can be included and combined with more general document analysis. In particular, a set of symbols must be described in terms of the types of conditions described below; they either describe coordination or provide a description of the children of a given structure, as a regular expression (a relationship condition). In both cases, the conditions are sufficient to make a decision.

Given a relationship condition, consider two structures x and y to have the relationship x *description-child* y if x forms a part of the regular expression describing the children of y and $x \neq y$. Sufficient coordination conditions must be provided for some nonempty subset; the rest of the set is formed by taking the transitive closure with respect to the *description-child* relationship. These specifications and the resulting grammar may introduce new structures.

A grammar or set of grammars is formed, in which terminals correspond to structures for which sufficient coordination conditions are provided; text matches such a terminal if it meets the conditions. Nonterminals correspond to structures included through the *description-child* relationship. The rules of the grammars are the obvious; the rule $A \rightarrow \alpha$ is included if A is a nonterminal whose sufficient children condition is described by the regular expression α. (The coordination conditions used for matching in the grammar must be based on the same essential observables used by the rest of the system, but they may involve more processing; for example, a greater degree of linguistic detail may be appropriate.) Each topmost nonterminal (*i.e.*, a nonterminal corresponding to a structure that does not have the relationship *description-child* to any included structure) is the start symbol of a grammar. An example of a grammar for the title page of a CS technical report at Cornell University is given in Figure 7.1.[1] Unless necessary horizontal location information is also included, each grammar is automatically augmented to include repetitions of this symbol.

At appropriate locations (as specified by any necessary location conditions), parsing is attempted based on each grammar in turn. As soon as a successful parse is found, the attempts stop. Partial parses are also attempted, *i.e.*, parses that begin with a nonterminal other than the start symbol. Each grammar generates these in a partial order. Let $str(X)$ indicate the structure corresponding to the symbol X, and let S be the original start symbol (before augmentation). Then, the symbols in the set $\Sigma_1 = \{X \mid str(X)\ \textit{description-child}\ str(S)\}$ are used as possible starts after S; next the symbols in $\Sigma_2 = \{Y \mid str(Y)\ \textit{description-child}\ str(X \in \Sigma_1)\}$ are used, and so on.

This procedure *can* be applied to any level formed in the hierarchy in the usual way, so no location information is strictly necessary. Without any indication of the correct location at which to perform this, however, all possible parses must be attempted at each level; this can

[1] The grammar is shown with regular expression right-hand-sides; these rules are easier to see than the form without regular expressions, and they are equivalent.

Title Page	→	*Title Author number date address*
Title	→	*title_start title_continue**
Author	→	*author_line*$^+$

Terminal Descriptions:

title_start	Centered
title_continue	Centered, does not follow extra blank space
author_line	Centered, meets linguistic criteria for a name or list of names
number	Centered, does not meet linguistic criteria for a date expression
date	Centered, meets linguistic criteria for a date expression
address	Left-In, follows extra blank space, immediately precedes a blank page

Figure 7.1: A CFG for the title page of a computer science technical report at Cornell. This is constrained to fall at the start of the document, and its start forms a top-level division.

become very costly. LABLER in its current form considers only cases in which a particular form of location information is provided: the terminals of the grammar must correspond to leaves in the full structure tree. (It is easy to see how to extend this to more general vertical location specifications, however; parsing could be applied at any prespecified level.)

This procedure finds logical subtrees of the full hierarchy. Other subtrees are formed by applying the more general method to the sequential runs of text blocks not successfully parsed. These trees are combined with their neighboring parse trees when they reach the same height, and the ordinary hierarchy building process continues; an exception is formed if the topmost structure of a parse tree has vertical location information specifying that it must be a top-level division. In that case, its parse tree is combined with any neighboring trees as a final step.

7.3 Classification Procedure Adjustment

Many cases of less detailed partial information call for slight changes in the classification procedure. In this case, the conditions should be limited to precisely the observables that the prototypes use; this allows for simple and direct alterations to utilize this knowledge. Both primary and secondary structure classification may be affected.

7.3.1 Coordination Conditions

Necessary coordination conditions essentially create new required attributes for the corresponding prototypes. Thus, to incorporate such a condition, the appropriate attribute is altered in the prototype and marked as necessary. A text block that does not match this attribute will automatically generate a distance of 1 from the prototype (*i.e.*, similarity of 0) and will therefore be excluded from matching.

Sufficient coordination conditions create a new, complementary situation. The corresponding attributes of the prototype are again altered if necessary, in order to reflect the specifications. The sets in which they are given are stored, and any text block that matches an entire set automatically generates a distance of 0 from the corresponding prototype (*i.e.*, similarity of 1) and will therefore match this structure. It may still match a more specific structure as well, in which case that structure will be its classification.

Typical coordination conditions describe precisely what the prototypes are meant to describe in a general way. Thus, their effect is to alter the prototype attribute values to correspond to the style-specific knowledge they provide.

7.3.2 Relationship Conditions

Necessary relationship conditions, in terms of the kinds of structure relationships the system recognizes, provide secondary-style structure specifications. If a necessary relationship is also sufficient, a structure is redefined as a secondary structure, based on the specified relationships. Otherwise, a primary structure retains its prototype, with the attributes corresponding to the relationships in question adjusted and marked as required.

Sufficient relationship conditions, when not also necessary, add a secondary form to an originally primary structure. (If the structure was originally secondary, they simply change its definition.) The structure is found as usual during primary classification; it is also found by matching the sufficient conditions during secondary classification.

Typical relationship conditions are incorporated into the corresponding prototypes, as described above for typical coordination conditions.

7.3.3 Location Conditions

Most location conditions are not captured by the attributes included in LABLER prototypes. Thus, they usually contribute to useful knowledge only when combined with the conditions that can form a grammar. The starts and ends of document portions separated by blank pages, however, are indicated in the `context` attributes of the corresponding first and last structures. (In many cases, these will simply be the start and end of the document.) Thus, conditions based on whether a structure starts or ends such a document portion, such as a requirement that a *title part* begin one, are incorporated identically to coordination conditions.

7.4 LABLER's Knowledge

LABLER uses style knowledge to parse the title parts of technical reports. This is the only parsing it performs; the hierarchy corresponding to the body of a document is built up according to the algorithm described in Chapters 4 through 6. (It does also use two sufficient coordination conditions; *paragraph*s and *list item*s are sufficiently, but not necessarily, identified by their shape.) The grammar LABLER uses for the title parts includes the title page description given in Figure 7.1. It is described in detail in Appendix B.

Chapter 8

Evaluation

8.1 Evaluation Methodology

Although segmentation and classification interact, they are essentially separate processes, calling for separate evaluations. Since segmentation consists of identifying certain pieces of text as units, as opposed to all other possible groupings, a natural analogy is formed with information retrieval. For classification, a pair of measures are described to capture both the rate of all errors and the rate of errors of relative significance.

LABLER's performance on these measures is given, together with some baseline results, in Section 8.2.

8.1.1 Segmentation Evaluation

The evaluation of the segmentation performance relies on the ideas of precision and recall from information retrieval. In the realm of IR, these are defined as follows [78]:

$$\begin{aligned} Precision &= \frac{Relevant\ retrieved}{Total\ retrieved} \\ Recall &= \frac{Relevant\ retrieved}{Total\ relevant} \end{aligned}$$

Logical groupings are analogous to relevant documents, and the groups formed are analogous to retrieved documents. This yields:

$$\begin{aligned} Precision_{segment} &= \frac{Logical\ groups\ formed}{Groups\ formed} \\ &= \frac{Good\ groups}{Good\ groups + Bad\ groups} \\ Recall_{segment} &= \frac{Logical\ groups\ formed}{Logical\ groups} \end{aligned}$$

$$= \frac{Good\ groups}{Good\ groups + Missing\ groups}$$

It is also interesting to consider segments that are almost right, which can be defined as follows: if a proposed segment that does not precisely match any good segment does share an edge with a good segment, and they have the same classification, the proposed segment can be considered to have been found *instead* of the good segment. For instance, if the last line of a paragraph has been incorrectly separated from the rest of it, the proposed segment that includes the first part of the paragraph may have been found instead of the paragraph itself. (Only one structure is counted as found instead of any good structure that is not found, to keep the accounting reasonable.) In many imaginable applications, such as browsing and retrieval, the proposed structure would be likely to occur where the correct structure should, yielding a result that, from a human perspective, is *almost* right and thus is of interest but includes too little or too much.

We can then consider this more forgiving measure:

$$Precision_{segment_{withalmosts}} = \frac{Logical\ (or\ almost)\ groups\ formed}{Groups\ formed}$$

$$= \frac{Good\ groups + Instead\ groups}{Good\ groups + Bad\ groups}$$

$$Recall_{segment_{withalmosts}} = \frac{Logical\ (or\ almost)\ groups\ formed}{Logical\ groups}$$

$$= \frac{Good\ groups + Instead\ groups}{Good\ groups + Missing\ groups}$$

Note that if group x is found instead of group y, then x is still counted among the bad groups, and y is still counted among the missing groups.

Another measure of the performance of the segmenter is the percentage of correct decisions it makes. Each decision either combines a pair of adjacent text blocks or leaves them separate. Thus, as each level is formed, if there are n text blocks in the preceding level, the segmenter must make $n-1$ decisions. The segmenter stops creating levels when it no longer chooses to combine any text blocks; thus, considering one additional level identical to the topmost allows us to evaluate the decisions that led the algorithm to stop generating levels.

This measure can be described as follows, for a performance that generates levels 1 through k, starting with an initial set of text blocks at level 0.

$$Decision_{Segment} = \sum_{i=1}^{k+1} Adjacent\ pairs\ at\ level\ (i-1)\ correctly\ grouped\ at\ level\ i \div \sum_{i=0}^{k} (Text\ blocks\ at\ level\ i) - 1$$

This measure can reasonably take into account decisions that include previously formed incorrect segments, if we define correct decisions in one of the following manners. Let s_i and s_{i+1} be two segments. For a strict measure, if the next correct structure that includes all of s_i is also the next correct structure that includes all of s_{i+1}, then the correct decision is to combine the two; otherwise, the correct decision is to leave them separate. For a more generous measure, also include as acceptable the choice to combine the structures, if the next structure that contains one of them also contains any part of the other one.

Finally, a more results-oriented measure can be found by *aligning* the generated structure hierarchy with the correct one. Alignment of trees is defined in [50], in the following manner. Consider all information about a node other than its position in the surrounding tree to be contained in a label for that node. To form an alignment, first add nodes with blank labels to each tree in such a manner that the resulting trees have identical shape. Then form a tree by overlaying these on each other, *i.e.*, form a tree with the same shape as the two generated trees, in which the label of each node is an ordered pair of the label from the corresponding position in the first tree and the label from the corresponding position in the second tree. This hierarchy is an alignment. Its *value* is found by defining a distance measurement on labels and summing the distances between the elements of the label pairs over the entire tree. The *optimal alignment* of two trees is the alignment with minimal value; its value is the *alignment distance* between the original two trees. Figure 8.1 shows two trees and their optimal alignment.

Alignment applies more naturally to the evaluation problem than does the more traditional tree edit. In the latter, all nodes are presumed to come out of the same pool of possibilities; either two nodes are really the same node, or they are not. This means that a single insert or delete adds or removes a node and all its descendents (the quantity of which may affect the cost of the operation) and that no accommodation is made for degrees of similarity among nodes. Alignment treats each node separately and compares the contents of nodes; thus, nodes are compared separately from their descendents, and the results may reflect finer-grained judgements about the degree of similarity among nodes. The alignment distance between ordered trees T_1 and T_2 can be found in time $\mathcal{O}(|T_1| \cdot |T_2| \cdot (\deg(T_1) + \deg(T_2)))$, using the algorithm described in [50].

Complete document structure hierarchies form a special case of ordered trees. In comparing segmentations, the relevant attributes of a node are the position at which it begins and the position at which it ends. Represent these by line numbers, and call them `start`(n) and `stop`(n), where n is a node. The children of a node represent a way of dividing the lines covered by the node. Thus, `start` of the leftmost child of n is `start` of n, and `stop` of the rightmost child of n is `stop` of n. Moreover, `start` of any n not a root or leftmost child is the (text) line following `stop` of its left sibling. Since the root node must begin at line 1 and end at the end of the document, the `start` of every node is predetermined (by this fact, its parent, or its left sibling), and the `stop` of any node with no right sibling is predetermined.

To reflect the above facts, each node is labeled by its `stop` value and a tag that indicates whether it is a rightmost child. Rightmost children are always considered to match each other. This corresponds to allowing changes to a parent to automatically change the

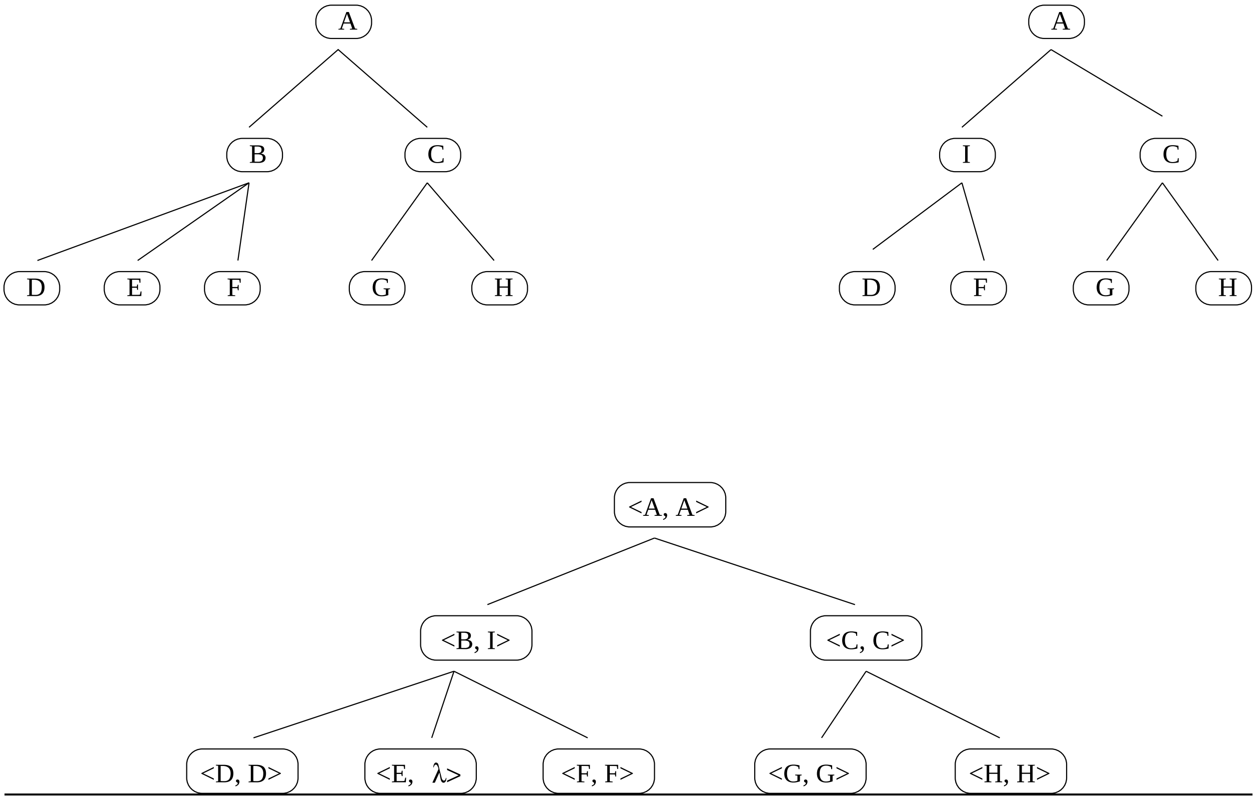


Figure 8.1: Two trees and their optimal alignment. If each pair of non-equal labels has a distance value of 1, the alignment distance is 2.

`start` and `stop` of its leftmost and rightmost child, respectively, and allowing a change to the `stop` of a node to automatically adjust the `start` of its right sibling. (A node whose range automatically shrinks to 0 does not disappear, however; it acquires an empty label.) Represent a label l_i of a node n_i as a pair $\langle s_i, r_i \rangle$, such that s_i is $\texttt{stop}(n_i)$ and r_i is `true` if n_i is a rightmost child and `false` otherwise. Represent a blank label as λ. Then, interpreting an alignment distance between tree T_i and T_j as a distance from T_i to T_j, the distance of a label pair (l_i, l_j) is defined to be:

$$d(l_i, l_j) = \begin{cases} 0 & \text{if} \quad (s_i = s_j) \vee ((r_i = \texttt{true}) \wedge (r_j = \texttt{true})) \\ \textit{insertion cost} & \text{if} \quad l_i = \lambda \\ \textit{deletion cost} & \text{if} \quad l_j = \lambda \\ \textit{shift cost} & \text{otherwise} \end{cases}$$

For the purposes of comparison in this thesis, all costs are 1 to maintain simplicity, but they could be adjusted to weight different kinds of changes differently, just as in string comparison or traditional tree edit. In particular, the shift cost could depend on the difference between the `stop` values.

For example, figure 8.2 shows two hierarchies of the form of full document structure trees, their representations for alignment as performed in this thesis, and their optimal alignment. With all costs 1, the alignment distance between the trees is 2.

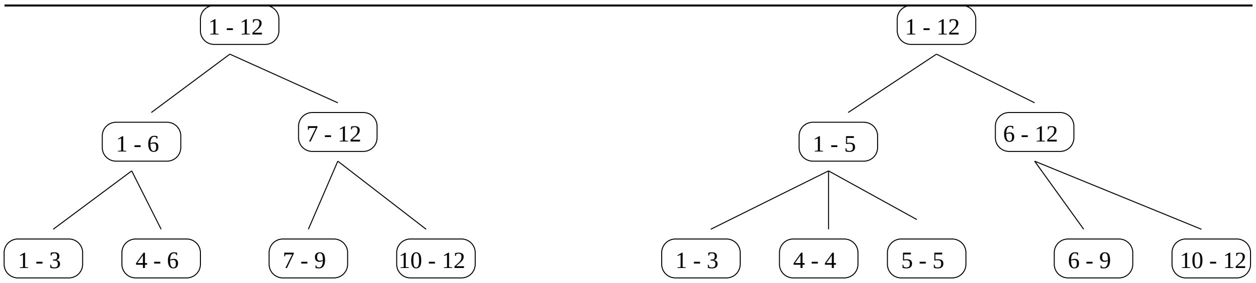
Alignment distances as raw integer figures provide very little information; an alignment distance is meaningful relative to the size of a hierarchy. Hence the alignment distances given in this thesis are normalized by the sum of the sizes of the correct hierarchy and the derived hierarchy.

Note that alignment essentially combines precision and recall in the following way. If a precision error and a recall error are found involving overlapping sets of lines at the same level of the tree, these are summarized together as a *shift*; otherwise, a precision error requires a *delete* and a recall error requires an *insert*.

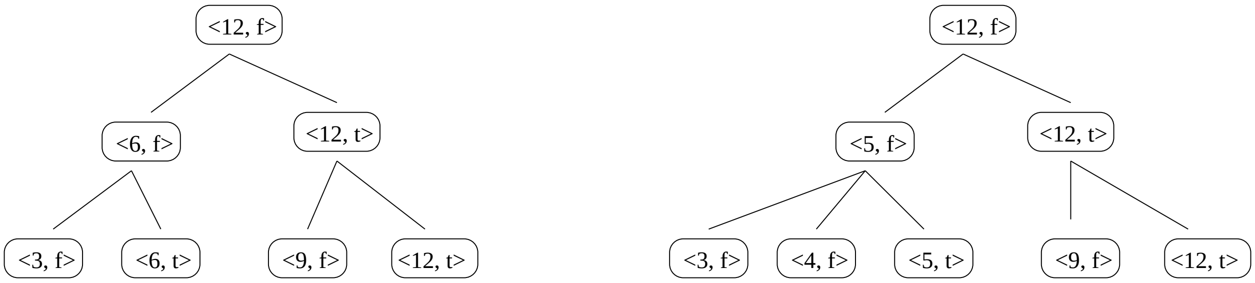
8.1.2 Classification Evaluation

Each classified node in a document hierarchy is characterized as *correct*, *overgeneralized*, *overspecialized*, or *incorrect*. A correct node is one that has been classified as the most specific logical structure in the system that appropriately describes it. An overgeneralized node has been classified as an appropriate logical structure, but there exists in the system a more precise logical structure that is also appropriate. (For example, a theorem may be classified as a *paragraph*.) An overspecialized node has been assigned a structure that is a subtype of its appropriate structure. Finally, an incorrect node has been assigned an inappropriate structure type.

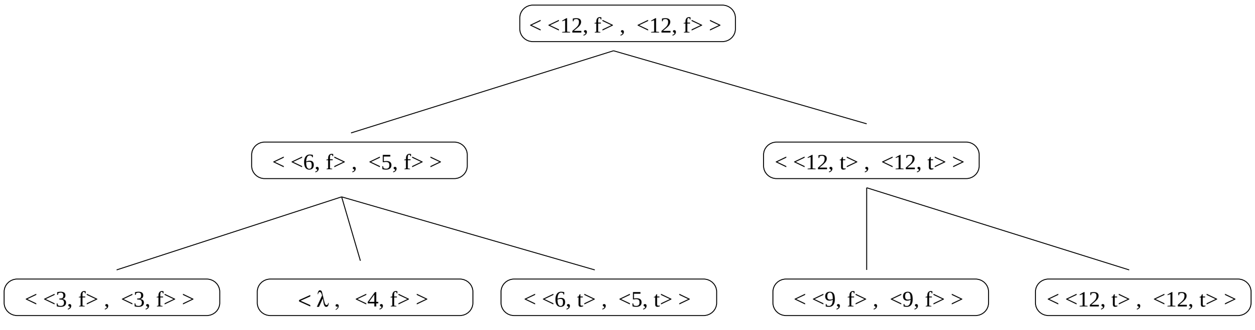
These characterizations yield two slightly different measures of accuracy. *Precise accuracy* is the percentage of classifications that are characterized as correct; *generalized accuracy* is the percentage of classifications that are characterized as correct, overgeneralized, or overspecialized, combining the overgeneralized and overspecialized into an "almost."



(a) Two trees of the form of full document structure hierarchies



(b) Representations of the above trees for alignment



(c) The optimal alignment of the above trees

Figure 8.2: Alignment of hierarchies of the form generated by document structure

$$\text{Precise accuracy} = \frac{Correct}{Correct + Almost + Incorrect}$$

$$\text{Generalized accuracy} = \frac{Correct + Almost}{Correct + Almost + Incorrect}$$

8.2 Performance

This section describes LABLER's performance on a test set of 13 computer science technical reports, comprising about 360 pages and over 5000 structures. LABLER's results were compared to correct hierarchies defined by versions of the documents that were marked up by hand. (This markup was performed after an initial version of the algorithm was developed but before the algorithm reached its current form.) Floats are handled by a preprocessor and not counted in the performance, as they would artificially boost the results; this reduces the number of structures under consideration to 4780.

The title part is found by parsing according to the style knowledge in Appendix B, and results are given both with and without considering the title part; without title parts, there are 4474 structures. The results with the title part demonstrate the practical effects of using LABLER, as a whole. Since the heart of the approach lies in the non-parsing algorithm, the results without the title part indicate the performance of LABLER's real contribution.[1]

The indentation values generated from OCR were corrected by hand when they were far from their correct values. (This typically happens when an unrecognized character is interpreted by the OCR as a figure and is thus skipped in forming the line of text.) LABLER relies heavily on reasonably accurate indentation values, which interferes with its direct usefulness on some OCR output; a fruitful area for further research would be the automatic handling of likely indentation errors, perhaps by identifying which values are likely to be errors (and then finding better values or presenting them interactively to a user for correction) or by incorporating an approach such as that in [16] that integrates into the analysis a consideration of the probability of flipped pixels.

8.2.1 Segmentation Performance

LABLER achieves an overall precision of 0.823, with $Precision_{segment_{withalmosts}}$ of 0.912, and an overall recall of 0.797, with $Recall_{segment_{withalmosts}}$ of 0.899. Figure 8.3 shows the results for the reports, together with baseline results of forming a hierarchy based entirely on vertical distance; this baseline yields an overall precision of 0.358 and recall of 0.582. This baseline corresponds to the segmentation method in [23], as described in Sections 2.2 and 2.3. The

[1] Since title parts follow somewhat different rules from running text, LABLER's main algorithm is of limited application to them; direct use of vertical distance tends to work well for title part segmentation, but the identification of the separation point at which to stop using this and start using LABLER's algorithm is non-trivial.

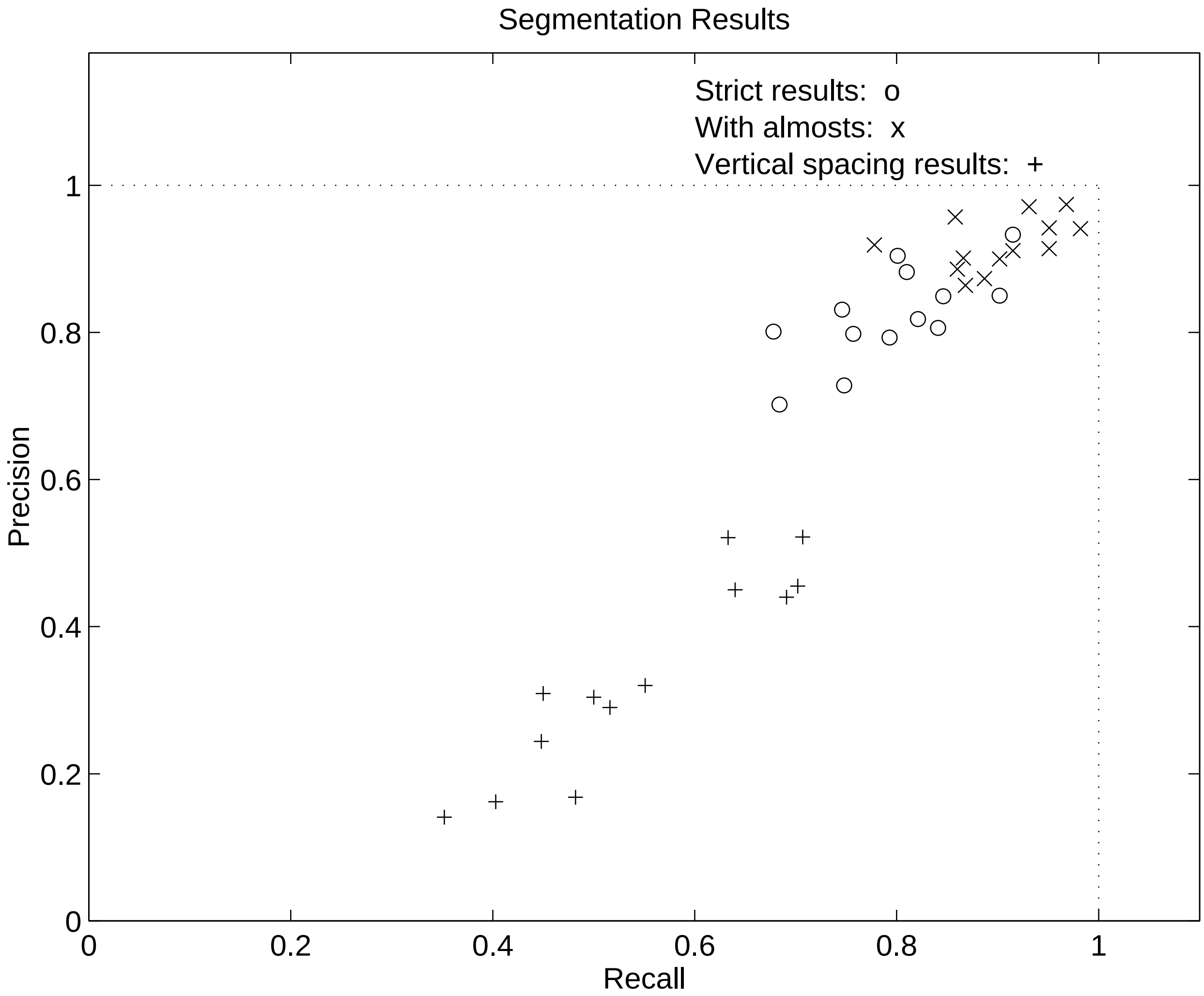


Figure 8.3: Precision and recall achieved on various technical reports

idea is that in general coarser-grained document divisions will be separated by larger quantities of vertical spacing, as, for instance, sections are separated by larger spaces than are subsections, which in turn are separated by larger spaces than are paragraphs. This correspondence does not hold over enough of the kinds of structures found in complex documents, such as those considered here, for its effects to dominate the layout of the documents.

Figure 8.4 shows the results and baseline without consideration of the title part. In this case, LABLER achieves an overall precision of 0.822 and recall of 0.803 (with $Precision_{segment_{withalmosts}}$ of 0.909 and $Recall_{segment_{withalmosts}}$ of 0.901), and the baseline yields precision 0.356 and recall 0.586.

To test the significance of the precision improvement of LABLER over the baseline, let $p =$ the baseline precision, let $good =$ the number of good structures retrieved by LABLER, and let $n =$ the number of good structures retrieved by the baseline. (Obviously, we must

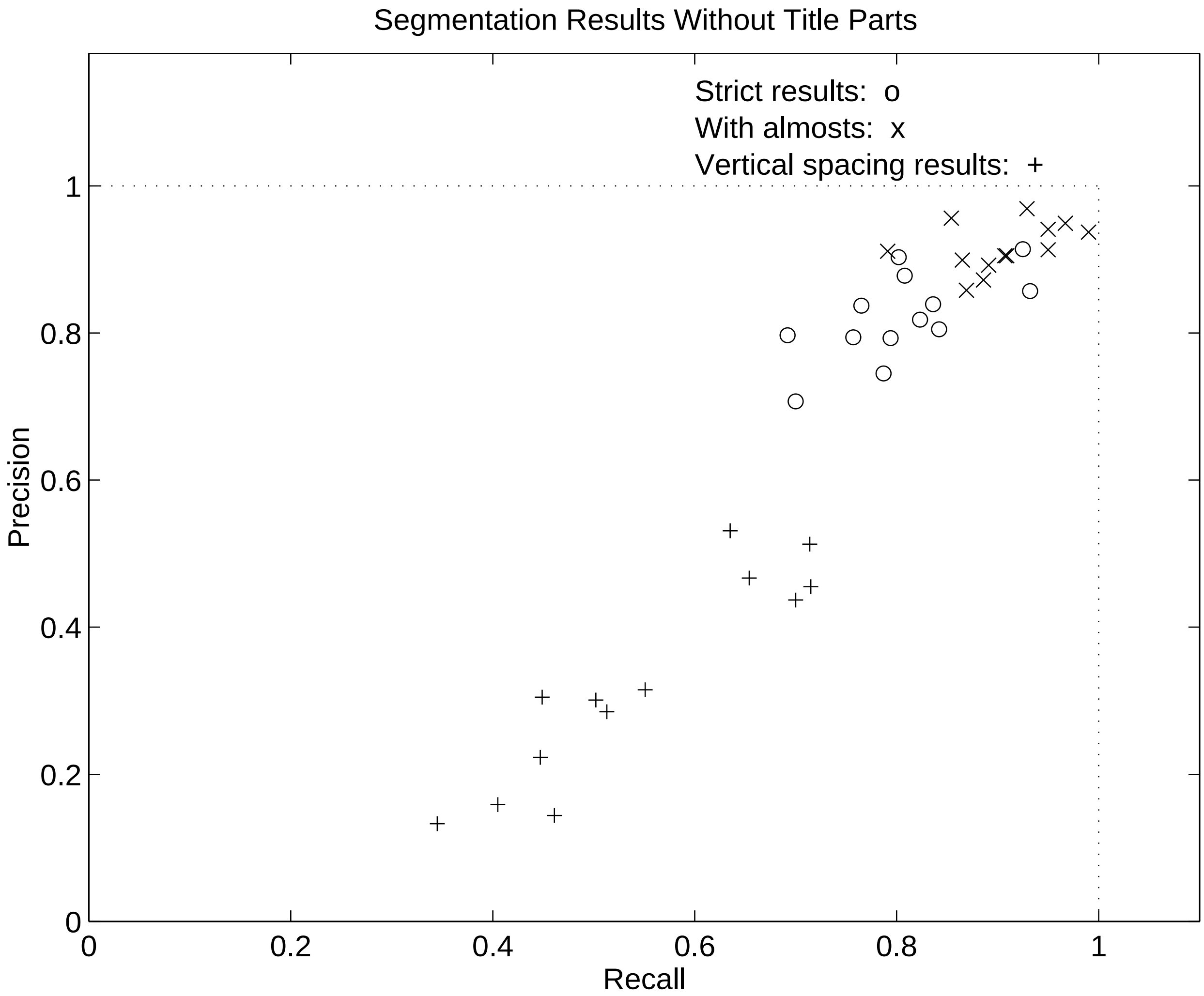


Figure 8.4: Precision and recall on various technical reports, without title parts

use the stricter measure of LABLER's performance here, to compare it fairly to the baseline.) Then we have, with one degree of freedom:

$$\chi^2 = \frac{(good - n\ p)^2}{n\ p} + \frac{((n - good) - (n\ (1 - p)))^2}{n\ (1 - p)} = \frac{(good - n\ p)^2}{n\ p\ (1 - p)}$$

Including the title parts, this yields $\chi^2 = 4242$; without the title parts, it yields $\chi^2 = 4167$. Both results are well beyond the value of 10.87 needed for significance at the 0.1% level.

To test the significance of the recall improvement of LABLER over the baseline, replace the precision errors above with the corresponding recall errors. Including the title parts, this yields $\chi^2 = 886$; without the title parts, it yields $\chi^2 = 874$. Both results are well beyond the value of 10.87 needed for significance at the 0.1% level.

The overall measures of correct decisions achieved are 0.872 and 0.899. The results are shown in Figure 8.5, with the strict measure indicated by `o`'s and the generous measure indicated by `x`'s. Without considering the title parts, the measures of correct decisions become 0.872 and 0.895; these results are shown in Figure 8.6, according to the same representation scheme. Note that these results are approximately the same; LABLER's performance is not artificially boosted by the title part grammar. In fact, the results suggest that the observed deviations from the expected title part format (due sometimes to actual deviations and sometimes to problems such as the creation of original blocks that contain groups of lines that should belong to different title part blocks) are almost as common as deviations from conformance to LABLER's expectations about running text.

Figure 8.7 presents the alignment results; as noted in Subsection 8.1.1, these are given as alignment distances normalized by the sum of the sizes of the correct hierarchy and the derived hierarchy. To combine these into a single figure, we divide the sum of the distances by the sum of the sizes of all the correct hierarchies and derived hierarchies; this can be thought of as an overall alignment result, in the sense that it corresponds to performing alignment on a pair of trees with dummy roots whose children correspond to the correct trees' roots and the derived trees' roots, respectively. This result is 0.208. The baseline is again the result of forming a hierarchy based on vertical distance only; this yields an overall alignment result of 0.600. Figure 8.8 provides the results without considering the title parts; in this case, LABLER achieves 0.211, and the baseline yields 0.618.

To test the significance of the alignment improvement of LABLER over the baseline, let $p =$ the overall normalized baseline alignment result, let $d =$ the total of LABLER's alignment distances, and let $n =$ the total by which LABLER's alignment is normalized, and Then we have, with one degree of freedom:

$$\chi^2 = \frac{(d - n\ p)^2}{n\ p} + \frac{((n - d) - (n\ (1 - p)))^2}{n\ (1 - p)} = \frac{(d - n\ p)^2}{n\ p\ (1 - p)}$$

Including the title parts, this yields $\chi^2 = 6224$; without the title parts, it yields $\chi^2 = 6427$. Both results are well beyond the value of 10.87 needed for significance at the 0.1% level.

In general, LABLER performs very well on documents made up primarily of running text, with few interruptions. When the interruptions start to dominate the shape of the text,

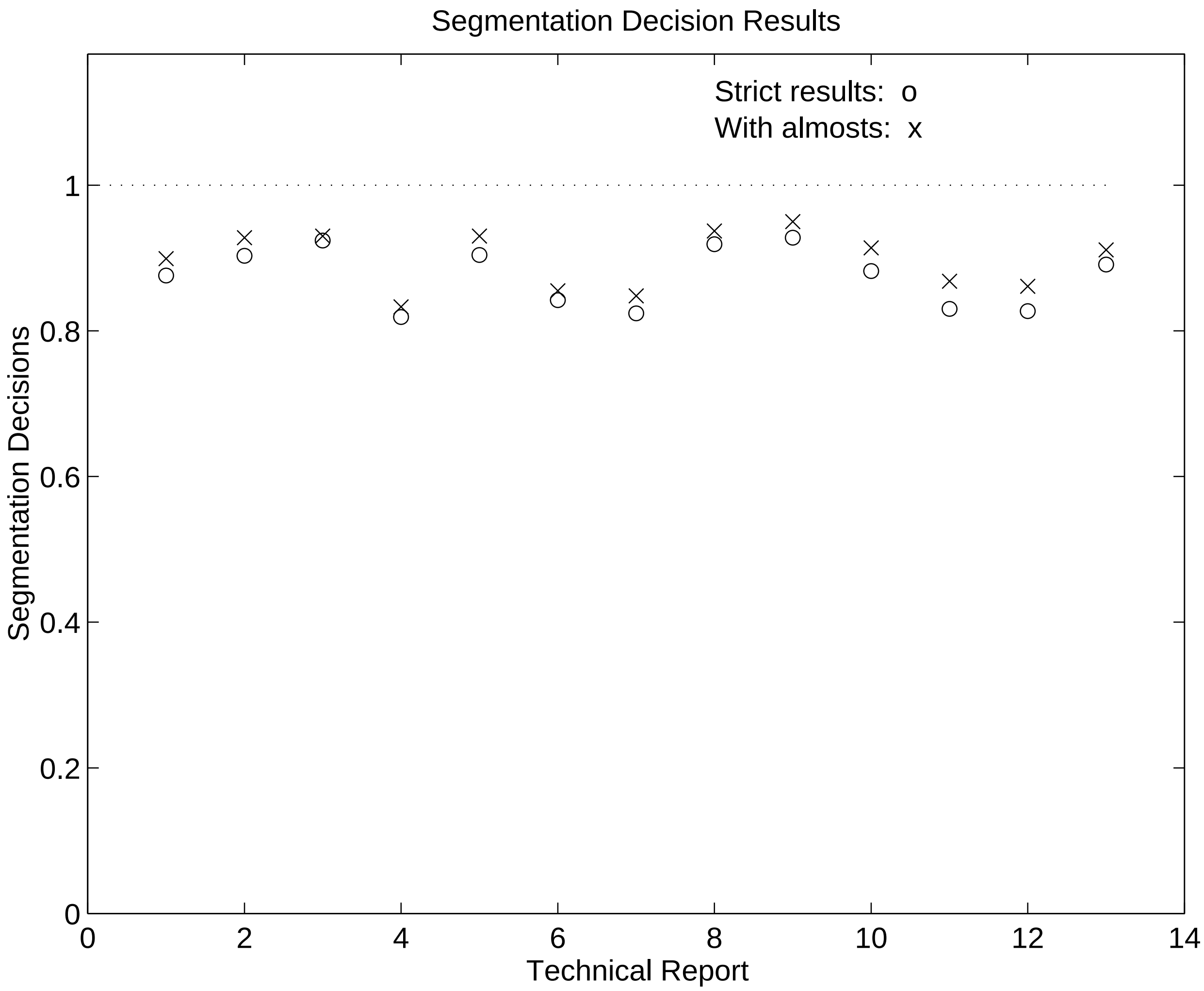


Figure 8.5: Correct decisions on various technical reports

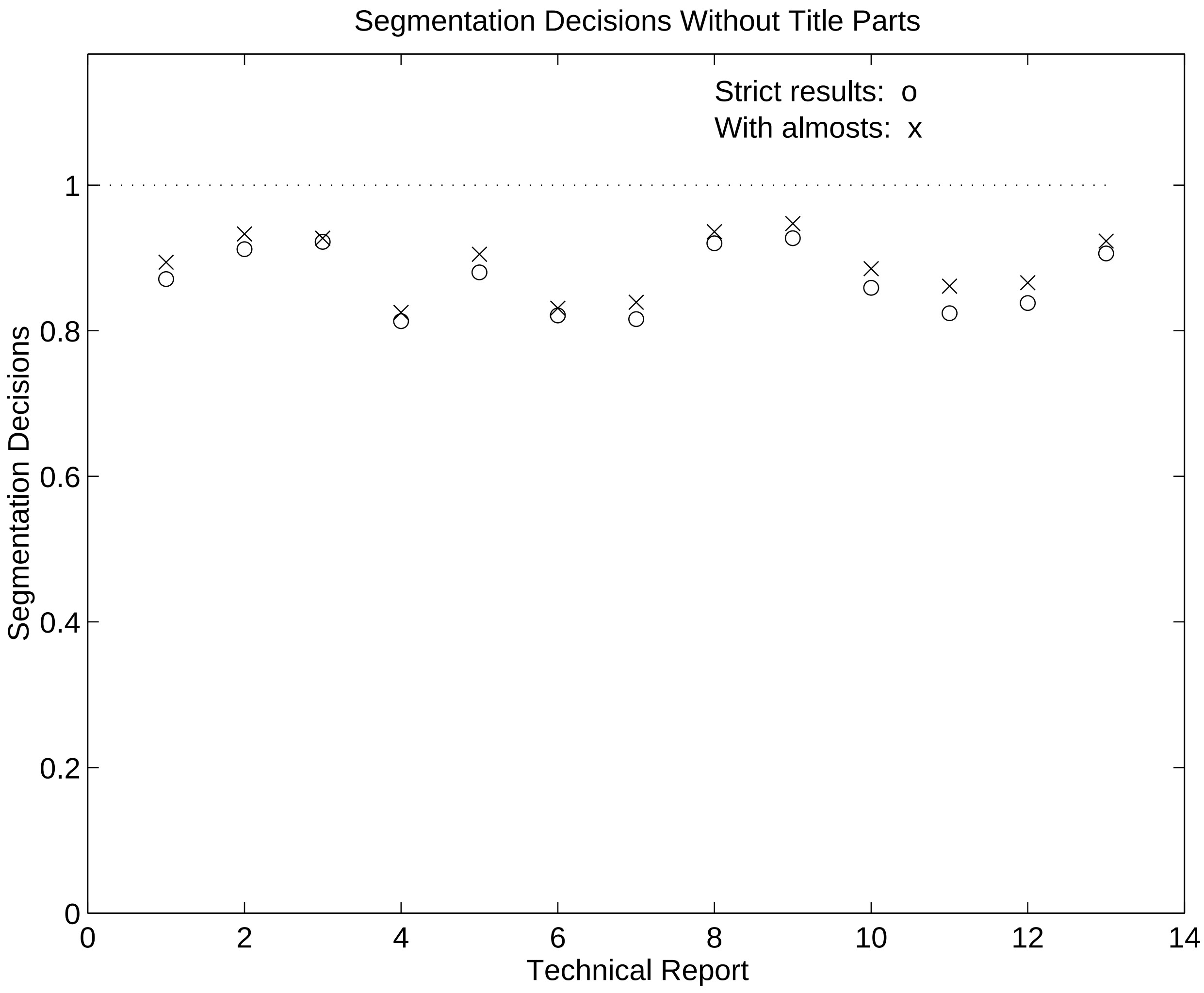


Figure 8.6: Correct decisions on various technical reports without title parts

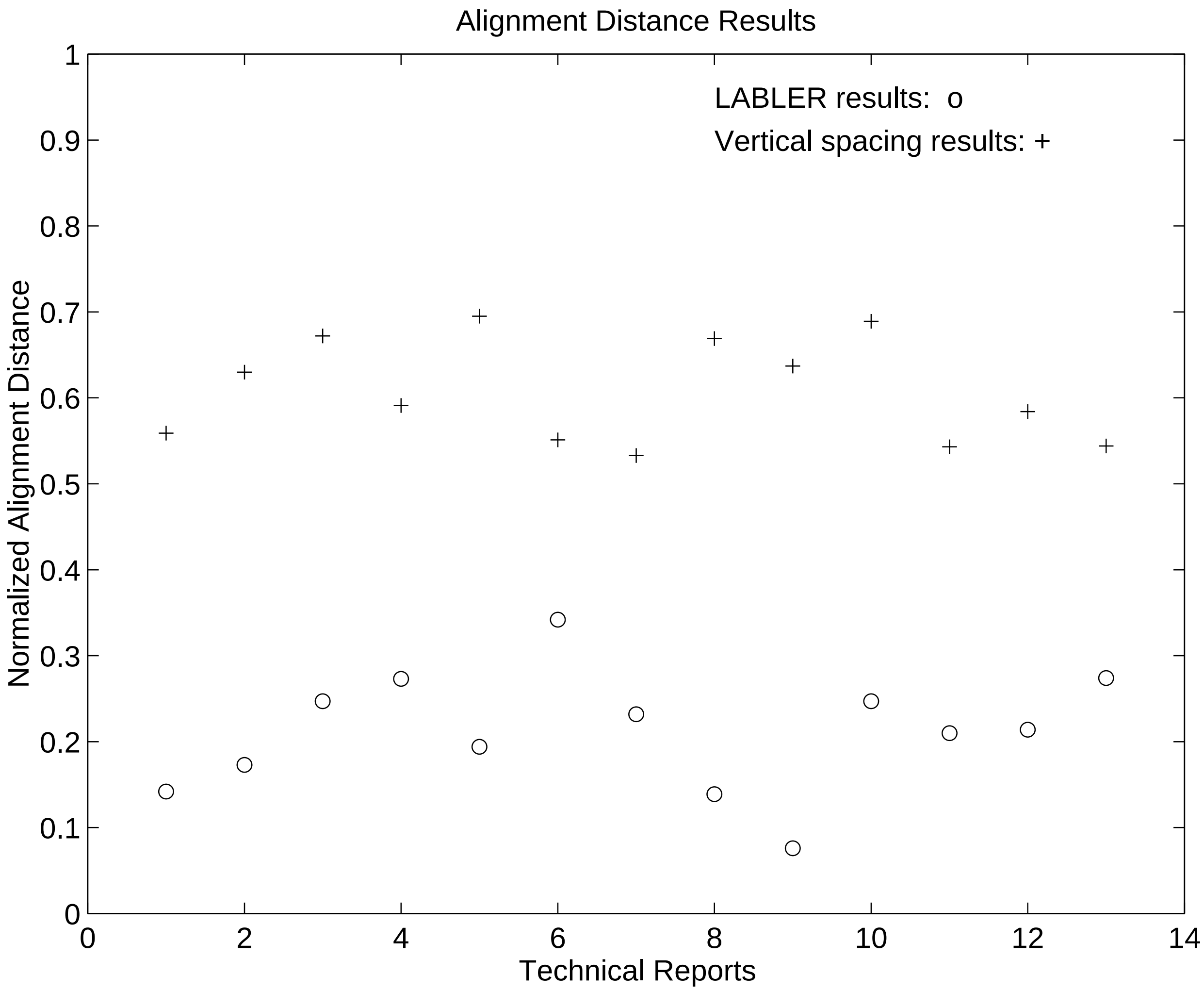


Figure 8.7: Alignment results on various technical reports

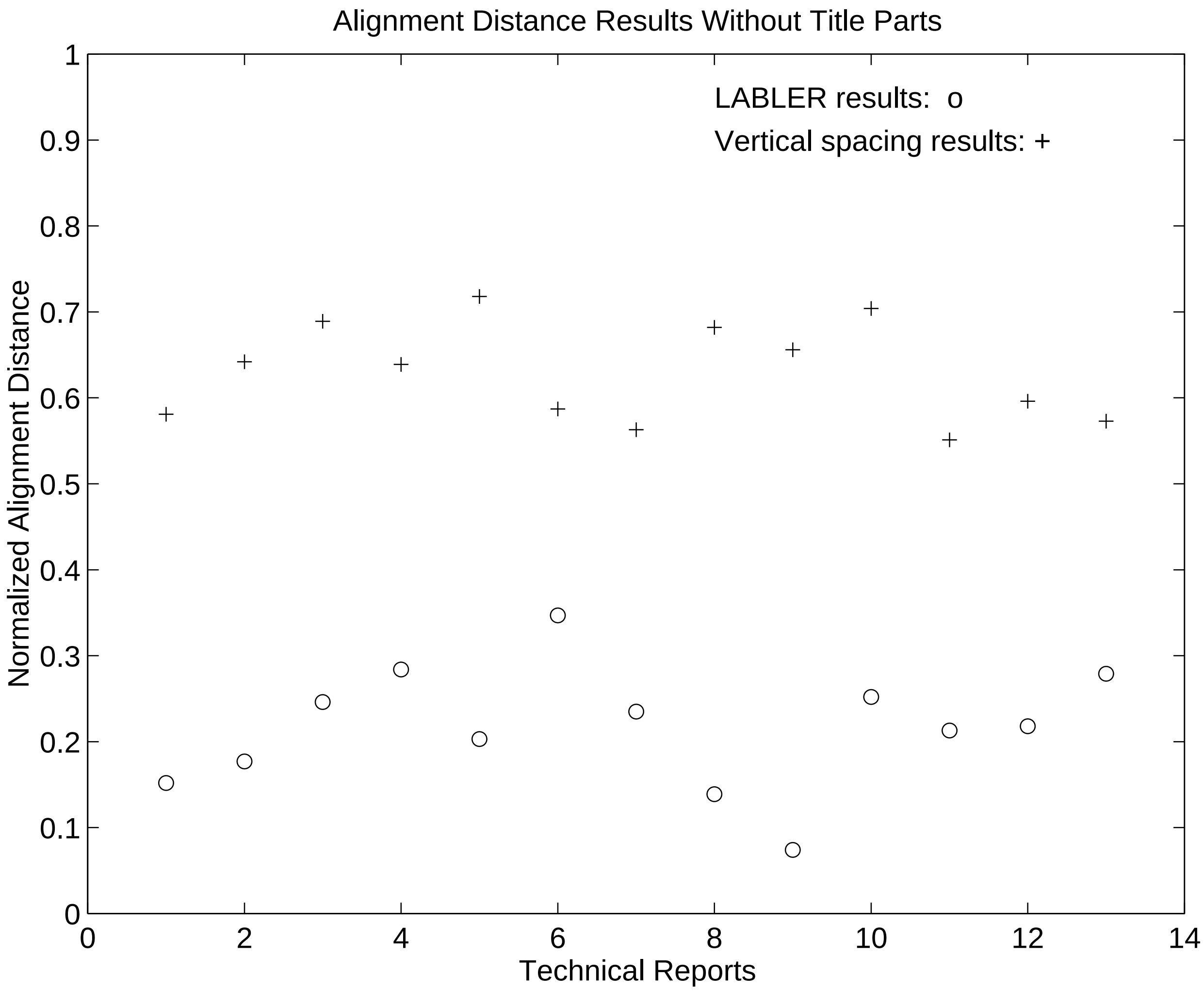


Figure 8.8: Alignment results on various technical reports without title parts

this can create spurious patterns and interfere with the matching of genuine patterns, thus decreasing LABLER's accuracy.

8.2.2 Classification Performance

Considering only the classifications of those structures that were correctly identified segments, the overall precise accuracy of classification is 0.809, and the generalized accuracy is 0.837. The results are shown in Figure 8.9, with precise accuracy indicated by `o`'s and generalized accuracy indicated by `x`'s. The results are compared to a baseline achieved by classifying all nodes as *paragraphs*, the most common structure; this baseline yields a precise accuracy of 0.182 and a generalized accuracy of 0.248. Figure 8.10 presents the results and baseline without considering the title parts. In this case, the results are 0.804 and 0.832, and the baseline values are 0.189 and 0.260.

To test the significance of the classification improvement of LABLER over the baseline, let p_p = the baseline precise accuracy, let p_g = the baseline percentage of overgeneralized or overspecialized results (*i.e.*, the baseline generalized accuracy minus p_p) let *good* = the total number of correctly classified structures, let *close* = the number of overgeneralized or overspecialized structures, and let n = the total number of classified structures. Then we have, with two degrees of freedom:

$$\chi^2 = \frac{(good - n\ p_p)^2}{n\ p_p} + \frac{(close - n\ p_g)}{n\ p_g} + \frac{((n - good - close) - (n - n\ p_p - n\ p_g))^2}{n - n\ p_p - n\ p_g}$$

Including the title parts, this yields $\chi^2 = 9747$; without the title parts, it yields $\chi^2 = 8873$. Both results are well beyond the value of 13.82 needed for significance at the 0.1% level.

Finally, to evaluate the classification process with as little effect as possible from segmentation, Figure 8.12 presents the results of classifying nodes when all their surroundings are correct. To get this result, a correctly segmented tree is generated, in which nodes without preceding and/or following siblings receive context from "cousins" at the nearest level to their own in the tree. Title parts are omitted from consideration here, and all segmenting information supplied for each node is accurate. Using the results of this process to provide the contextual classification information for each node, it yields an overall precise accuracy of 0.764 and a generalized accuracy of 0.799. Using entirely correct contextual information in each case, the classification process yields a precise accuracy of 0.828 and generalized accuracy of 0.861. Figures 8.11 and 8.12 present these results, against the same baseline as above. Note that the former values are lower than those achieved on LABLER's correctly segmented structures, and the latter are higher; this reflects the significance of contextual information and the fact that classification errors can propagate through contexts. (When such propagation occurs near a segmentation error, the erroneous segment is not counted among the classification results, so the effect on the numbers is lessened.)

Applying the above significance test to these results yields $\chi^2 = 9527$ and $\chi^2 = 11778$, both well beyond the value of 13.82 needed for significance at the 0.1% level.

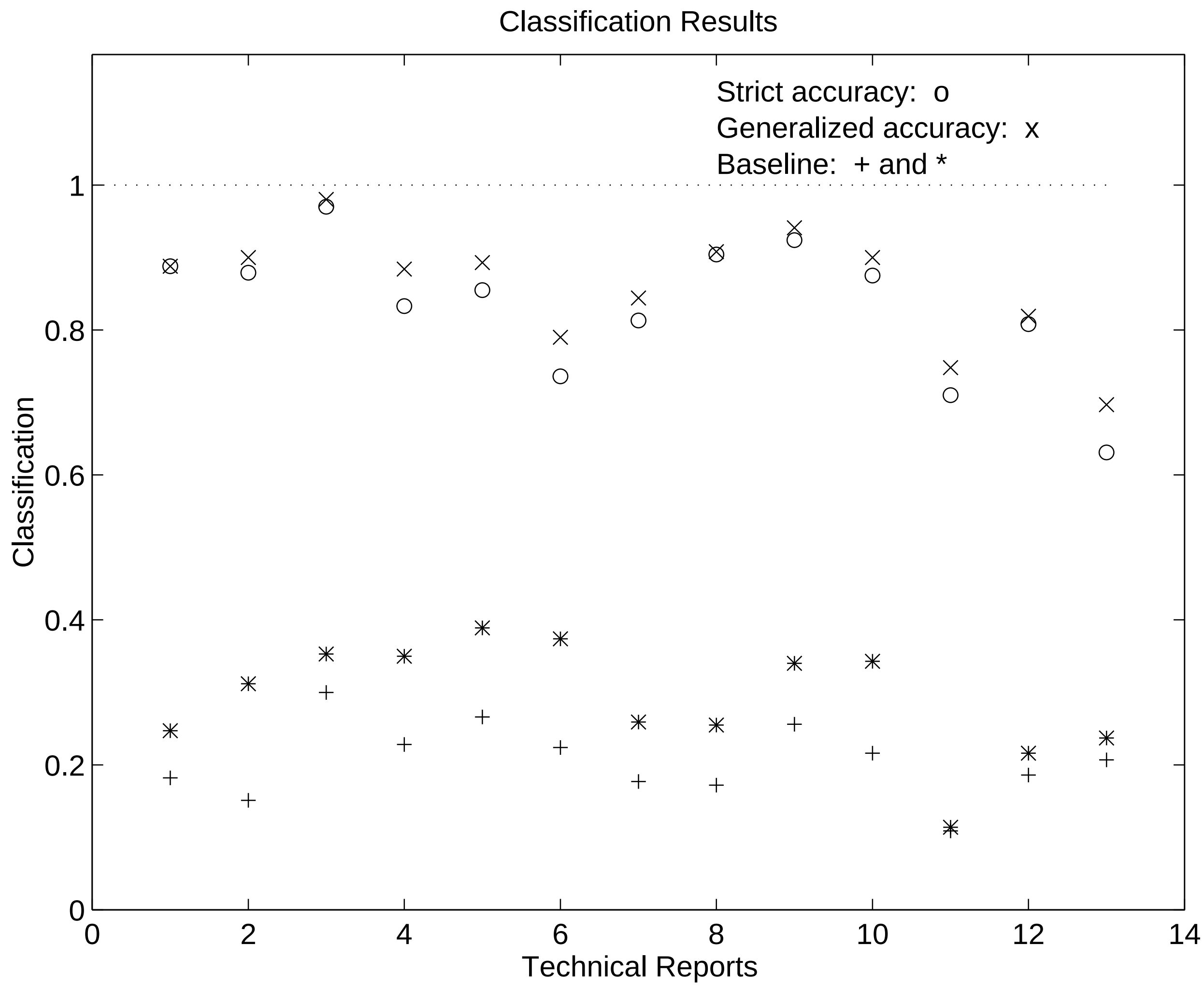


Figure 8.9: Classification results on various technical reports. The baseline has strict accuracy represented by + and generalized accuracy represented by *.

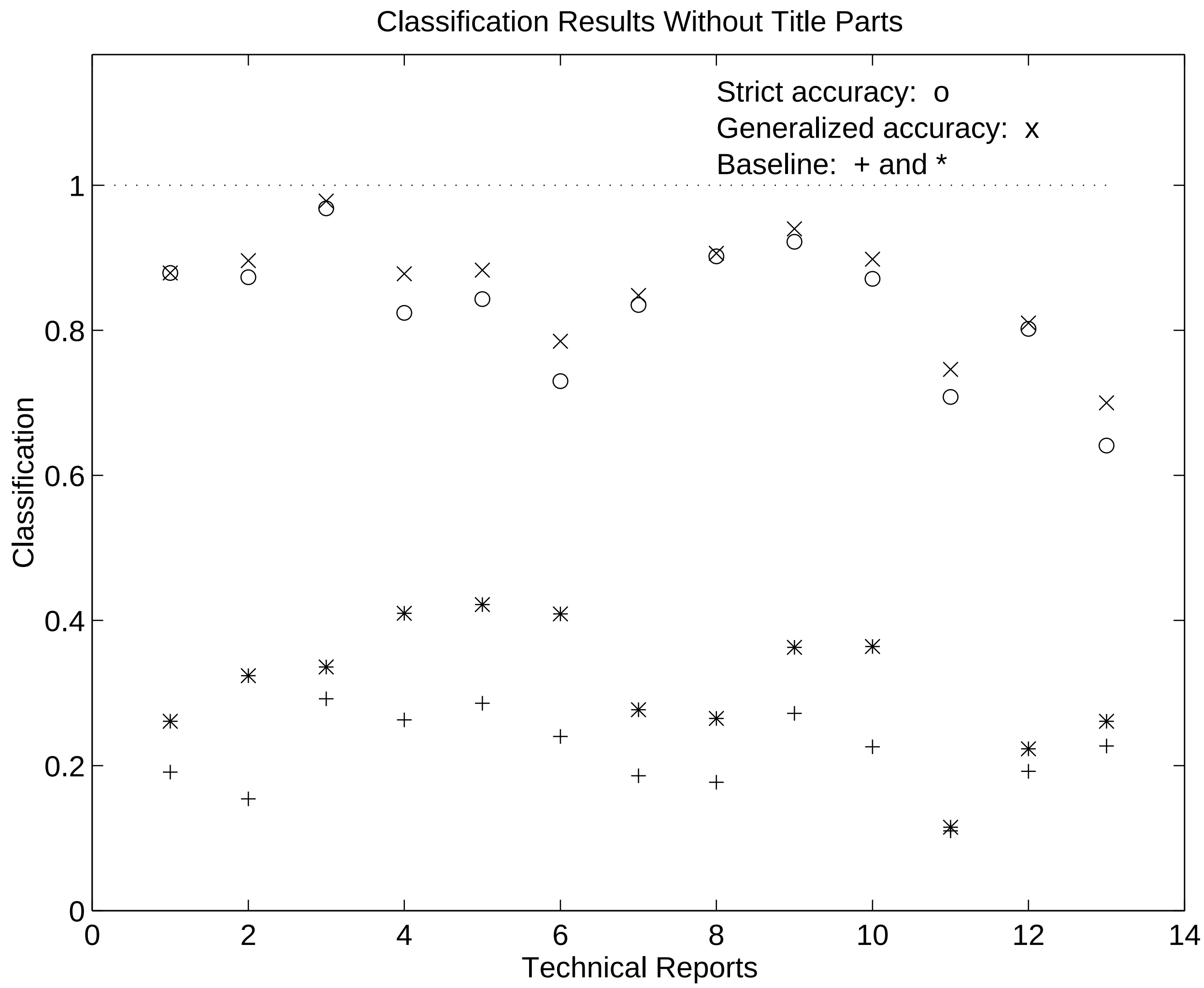


Figure 8.10: Classification results on various technical reports without title parts. The baseline has strict accuracy represented by + and generalized accuracy represented by *.

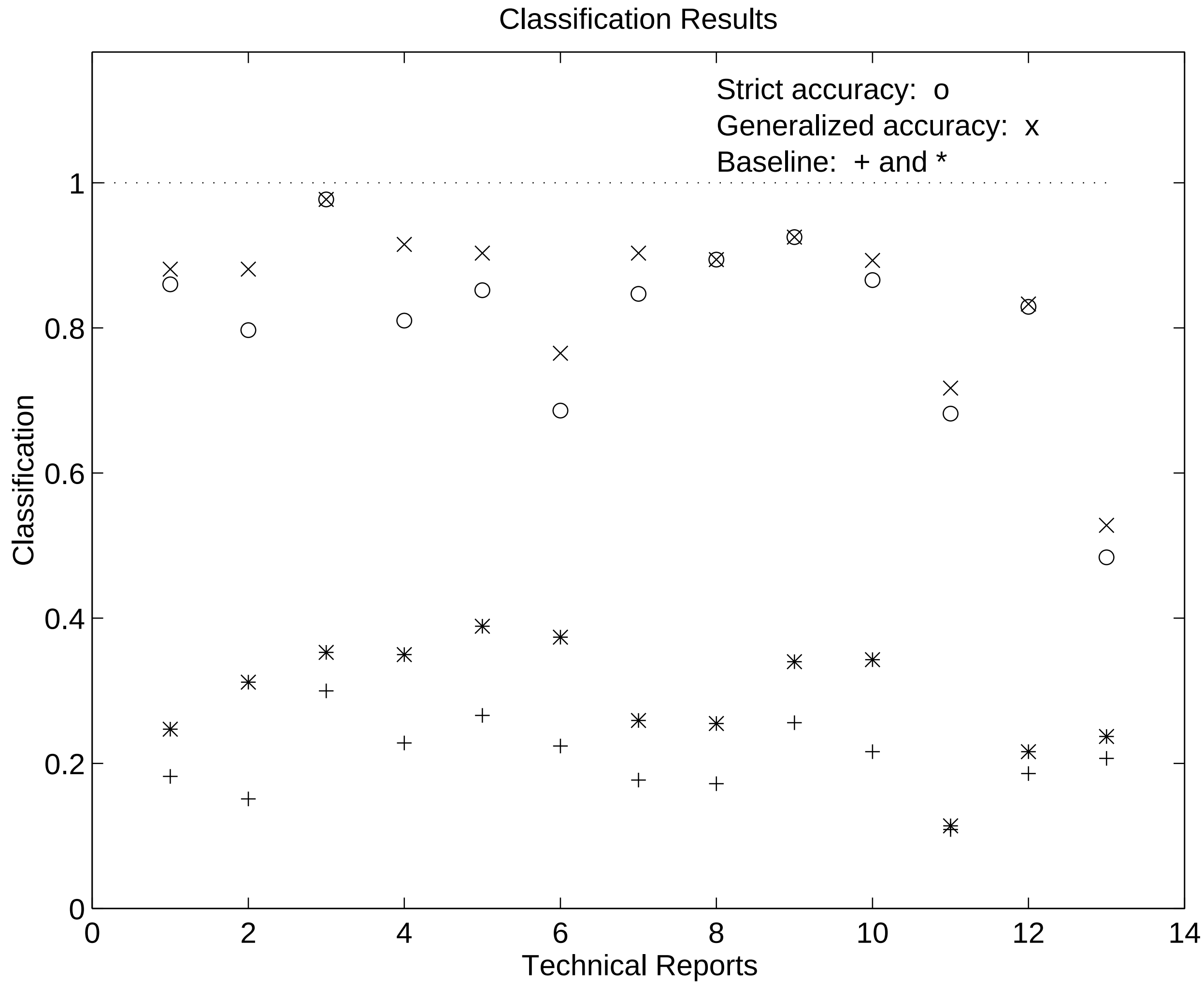


Figure 8.11: Classification results on correct hierarchies for various technical reports. The baseline has strict accuracy represented by + and generalized accuracy represented by *.

The above results suggest that the classification routine has the potential to perform better than it currently does, if given better input, but improved segmentation may also bring to light propagated classification errors.

Note that in all the versions of classification, the relative results on various documents are similar. That is, some documents appear simply to be harder than others. This consistency reflects a more general consistency in the types of classification errors LABLER makes. These errors fall into two main categories: part/whole distinctions and primarily content-based structure identification. LABLER's errors often confuse parts with wholes, by identifying, for instance, a *paragraph* as a *paragraph group* or *paragraph part*. Another set of errors occurs in structures for which content is an especially significant cue, such as *equations*; simply identifying the presence and frequency of a few unusual characters or strings often does

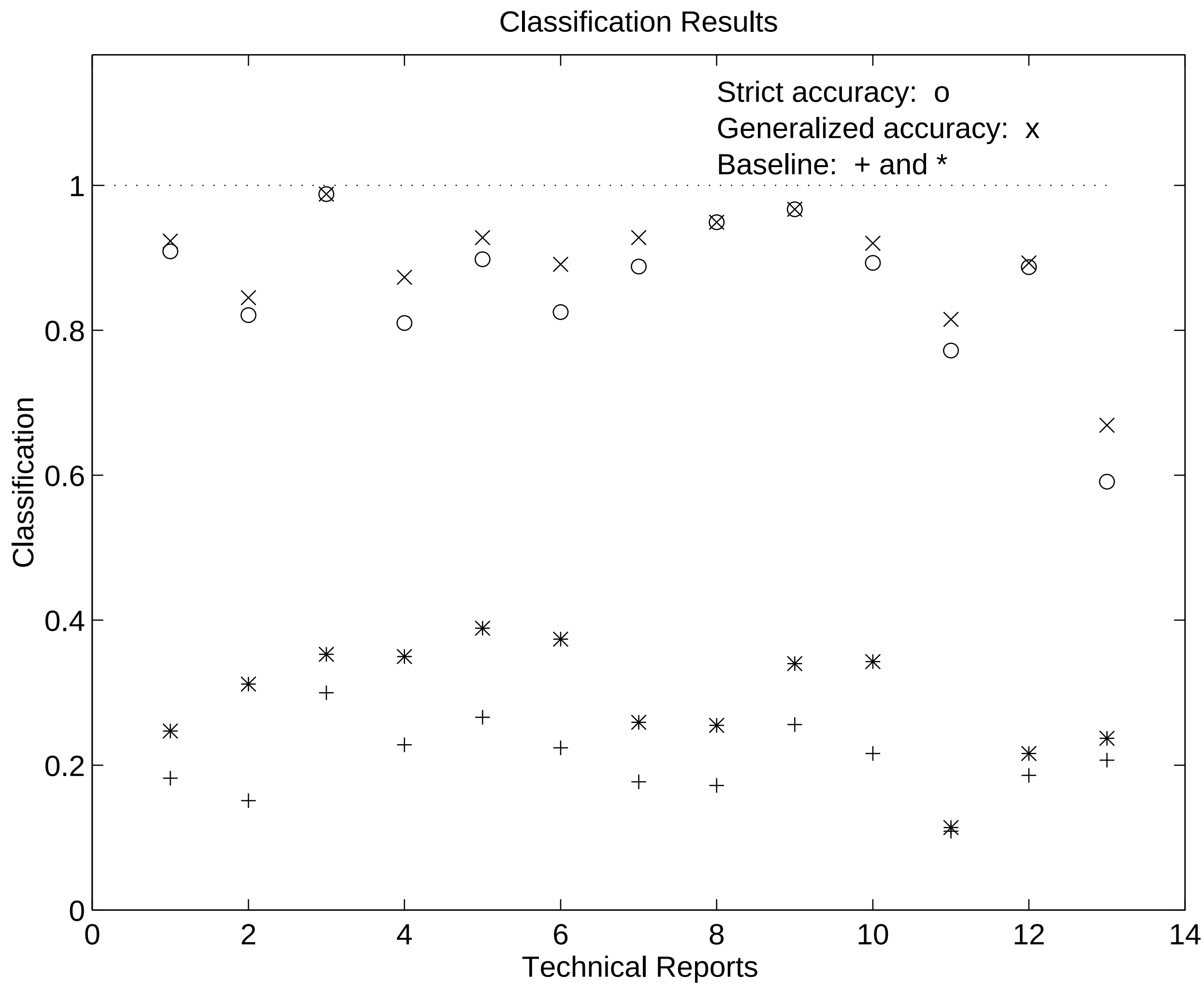


Figure 8.12: Classification results on correct hierarchies for various technical reports, with correct contextual classification. The baseline has strict accuracy represented by + and generalized accuracy represented by *.

not suffice to identify these structures. (The fact that OCR errors may cause an inaccurate depiction of these characters adds to the problem but is not its primary cause.)

Tables 8.1 and 8.2 show error matrices for LABLER's classification. Each cell shows the percentage of misclassifications of a given type, for any type that accounts for at least 1%. Specializations of a single general type have been combined, and in some cases individual structures and groups of the same have been combined; an error entry with the same row and column heading indicates errors in choosing between the classifications covered by the heading.

Table 8.1 shows these results on the correctly segmented structures LABLER finds, and table 8.2 shows these results starting from full, correct hierarchies. Part/whole distinctions account for 51.0% of the errors in the first case and 29.7% of the errors in the second. Note that in the second case, the correct contextual information may include an earlier, accurate part/whole distinction, which can aid in making the current decision; these errors tend to occur in clustered groups, and correcting the earlier decisions can propagate the right information through the group of related structures. The difference in part/whole errors between the cases reflects this phenomenon, which suggests that even a small change to improve the part/whole performance could have a large effect on these results.

Content distinctions, including identifying a paragraph or paragraph group as the wrong *kind* of paragraph or group and errors in finding equations, constitute 35.1% of the errors in the first case and 44.5% in the second. Clearly, these two error types dominate the results; the more difficult documents demanded more of these distinctions relative to their size than the others did.[2]

8.2.3 Performance Conclusions

On a variety of measures, LABLER performs far better than the baselines of simple vertical distance segmentation and classification according to the most common category. These baselines use simple, available, and generic information in the most straightforward way; that LABLER far outperforms them indicates that its additional observables and processing are in fact productive.[3]

In particular, the intuition that more vertical space corresponds to a more significant separation (and thus separation at a higher hierarchy level) only holds very roughly. In building a detailed hierarchy, many exceptions arise, such as equal vertical space of different importance and greater vertical space that may actually indicate a less significant separation. For instance, the vertical space surrounding an equation or list within a paragraph may be greater than that surrounding a subheading. Consideration of the horizontal relationship

[2]Note that the presence of equations generally contributes to both types of problems, since an equation itself must be identified, and it usually splits a paragraph into pieces, generating the need for a part/whole distinction.

[3]It is difficult to compare a more generic method to one based on parsing, as the results of a parse depend greatly on the grammar chosen. Moreover, a set of observables and grammar can be constructed for any particular fixed set of documents; the advantage to a generic approach lies in its greater applicability, not a higher success rate on the documents to which it is applicable.

Table 8.1: Classification error matrix, on correct segments from full results

	Correct Classificiation		
Found	Paragraph Type	Para. Group Type	Para. Part
Paragraph Type	13.2	1.2	29.2
Para. Group Type	3.5		
Paragraph Part	4.3		

	Correct Classificiation			
Found	List Item	List	Equation(s)	Section(s)
Paragraph Type	2.9		10.3	
Para. Group Type	1.2			
Paragraph Part	1.0		1.2	
List Item		1.7	5.5	
List	4.5		1.9	
Equation(s)	2.2		1.3	
Heading	1.2			
Section(s)				5.3

Table 8.2: Classification error matrix, on correct hierarchies

	Correct Classificiation		
Found	Paragraph Type	Para. Group Type	Para. Part
Paragraph Type	10.0	1.5	13.6
Para. Group Type	7.1	3.8	
Paragraph Part	6.8		

	Correct Classificiation			
Found	List Item	List	Equation(s)	Title Type
Paragraph Type	3.5	1.2	10.5	2.0
Paragraph Part	1.1		11.9	
Heading				2.3
Section Body				1.0
List Item			6.6	
Equation(s)	1.4			

between vertically separated components enables the necessary distinctions in many cases; feedback from the classifier helps substantially, as well.

Similarly, the classifier itself makes use of its observables; many standard structures are highly identifiable by shape and context, and the consideration of simple content and font characteristics makes a good deal of headway at identifying the others.

There is clearly room for improvement, especially in the areas of reducing the reliance on OCR indentation accuracy, discriminating between genuine separations and spurious appearances of separation in the segmentation phase, and addressing part/whole distinctions and content-based structures in classification. The directions these issues suggest for future work are discussed in Chapter 10.

Chapter 9

Logical Browsing

Once LABLER has analyzed a document and derived its logical hierarchy, it generates output in HTML, suitable for browsing with a World Wide Web navigator. This enables a user to traverse the logical tree, beginning with the root node containing the entire document.

Each node is represented by a separate page, and each page contains three types of information about its node: general representational information about the node itself, structural relationships to other nodes, and the text (or other contents) of the node. The user may thus choose among the options of reading the current node or moving to another related node, based on a description of the current node and briefer descriptions of available related nodes. The accessible related nodes always include the root node, so the option always remains of simply reading the entire document by jumping to the root node and reading its content. The three types of information are contained in different sections; they are specified as separate `frame` subwindows for browsers that support this.

9.1 Node Representation

The usefulness of this kind of browsing depends in large part on the node descriptions available. They need to provide adequate information with which to decide how to continue browsing. They must *summarize* the information in a node, rather than *present* it all, however; otherwise, the advantage of browsing over reading is lost.

To this end, LABLER includes a concept of *representative components*. One of the descendents of each interior logical structure node is deemed to contain information of particular use in describing that node; this descendent is the representative component of the original node. (Leaf nodes have no representative components.) The relationship can be defined by structure classification or by hierarchical relationship. An obvious example is that the representative component of a *section* is its *heading*, *i.e.*, the representative component of a node with classification *section* is its child with classification *heading*; in this case, a large part of the purpose of the representative component structure is to convey information about the content of the main structure.

In LABLER, several such relationships are predefined based on structure type. In the remaining cases, the leftmost child of a node is considered to be its representative component; this heuristic reflects the intuition that the content the writer expects to be seen first usually provides more cues about what follows than vice versa.

The content of a node is described by a presentation of the leaf node that ends a chain of representational components. That is, consider a sequence of nodes $n_0 \ldots n_k$ such that $\forall 0 \leq i < k$, n_{i+1} is the representative component of n_i, and n_k is a leaf node in the structural hierarchy. Then, the content of n_0 is described by a presentation of the content of n_k, called the *representative leaf* of n_0. In fact, the content of each n_i such that $0 \leq i < k$ is so described. The presentation may vary, from a display of all the content to the first few words, depending on the degree of detail required.

9.2 HTML Page Sections

In the main informational section of a node's HTML page, the node's representative component is given in its entirety, after headings that specify both the structure class of the node and the structure class of its representative component. Direct links are provided to the parent, left sibling, right sibling, and leftmost child of the node. (This makes a breadth-first traversal particularly easy and direct.) In the case that `frames` are in use, this section also contains links that control the form of presentation of the information in the others.

The structural relationships are presented both purely locally and in a greater context. Local structural information consists of the parent, siblings, and children of a node; the greater context includes the path from the root to the current node, as well as the children of each node on this path. (The latter is, of course, a superset of the former.) These structures are presented in an outline format; each entry includes the name of the structure classification of the node, which also serves as a link to the node itself. The local representation is designed to be compact, so it includes only this information. The contextual structure representation also includes a presentation of the representative leaf of each node. If this leaf is reached from the node entirely by means of predefined structure type relationships, the entire contents of the leaf are shown; if it is reached in part by using the "first child" heuristic, then only the first text line of the leaf is shown instead. This precludes the inclusion of very long content descriptions within the outline. In the contextual outline, fonts distinguish nodes on the direct path from the root to the current node from the other children of such nodes and also distinguish the current node from the rest.

The content of a node is presented in two alternative manners, as text or as a set of GIF images. For the text option, the results of OCR and rebuilding of text lines are presented, providing a view of precisely the input corresponding to the current node that LABLER analyzed to find the presented logical structure. The GIF image option presents the scanned images of the pages that cover the current node. The latter offers a more complete and accurate view of the relevant portion of the original document than does the former, although it also includes irrelevant parts of the document.

Table 9.1: Predefined representative components of structures in LABLER

Structure Type	Representative Component Type
Section	*Heading*
Document	*Title Part*
Title Part	*Title*
Abstract	*Abstract Body*
Author Information	*Name* or *Names*

Table 9.2: Some effects of the "first child" representative component rule

Structure Type	Typical Representative Component
List	First *Item*
Paragraph Group	First *Paragraph*
Section Body	*First Paragraph*
Sections	First *Section*

9.3 Representative Components

In the cases in which one kind of descendent of a given type of structure is clearly likely to be more informative about the structure as a whole than the others, this descendent type is predefined as a representative component of the main type. The first descendent of this type, based on a breadth-first ordering, is the representative component of the main node. (Typically, this node is a child of the original node.) These cases are summarized in Table 9.1.

Otherwise, as discussed above, the leftmost child is used. Some typical examples of the effects of this heuristic are summarized in Table 9.2.

9.4 A Browsing Example

A typical step in browsing a technical report might lead to the representation shown in Figure 9.1. This is a Netscape rendering of the HTML page for the first section of a particular technical report. The top frame indicates that the current node in the hierarchy is a section, provides its heading, and offers direct links up to the set of sections, right to the next section, and down to the heading. The "left" link leads nowhere, as the first section has no left sibling. It also contains links to control the form of presentation in the other frames.

The middle frame shows the relevant structure. In Figure 9.1, it shows the local structure only, of the parent, siblings, and children of the section, together with links to them. The parent, of course, is the set of sections; the siblings are the other sections; the children are the heading and section body. Figure 9.2 shows the structure in context, in its own window.

Here, the entire path from root to the current node is shown, together with the children of each node on this path and at least the first line of the representative leaf of each structure. (The path from the root is: Document → Body → Sections → Section_1.)

The bottom frame shows the contents of the node. Its default value is the results of OCR for precisely the node itself, as shown in Figure 9.1. The GIF images of all pages containing any of the node can also be displayed, by choosing the corresponding link. Figure 9.3 shows part of this result in its own window.

Portions of the Mosaic rendering of the non-frames version of the page are given in Figures 9.4 through 9.5.

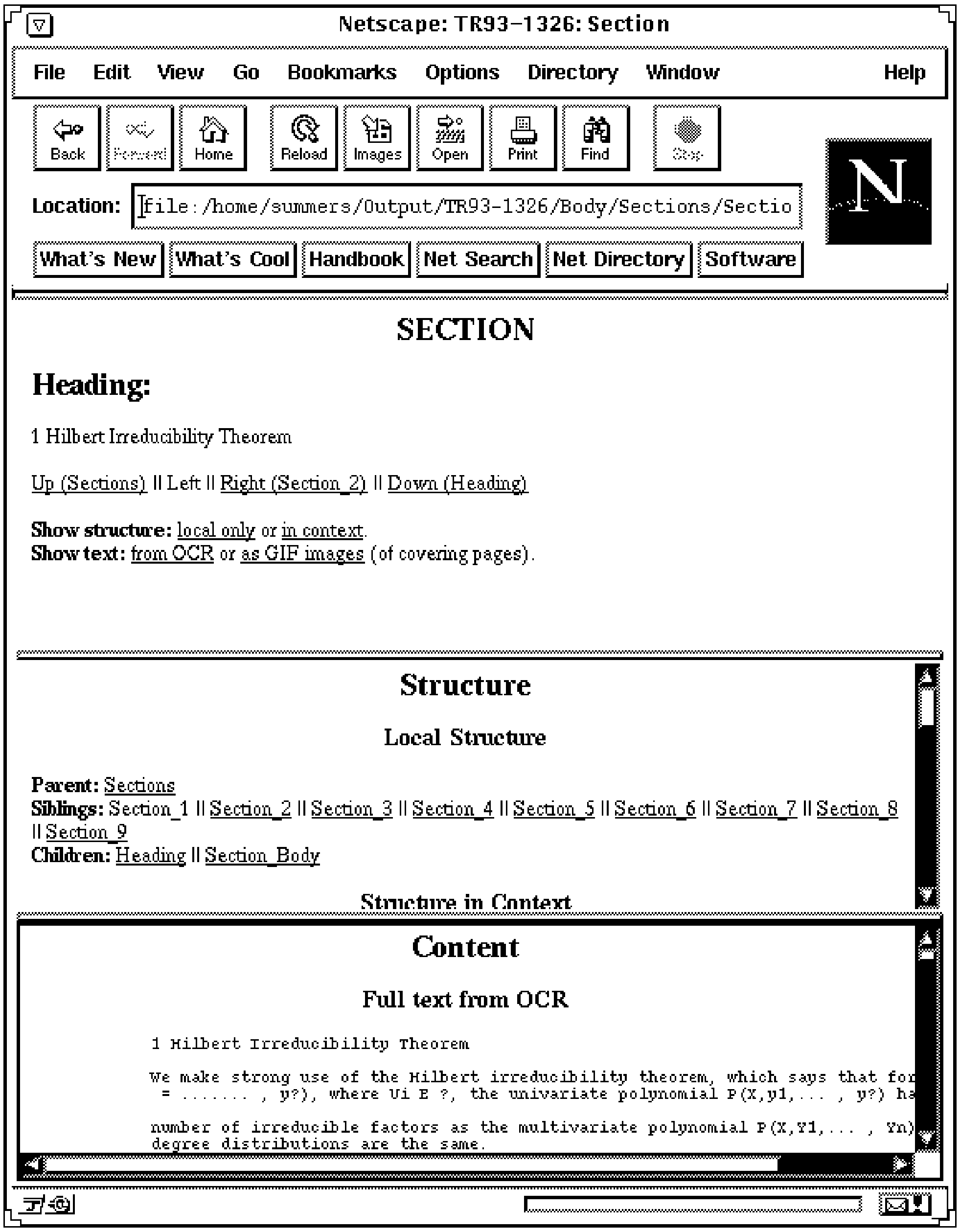


Figure 9.1: A representation of the first section of a technical report

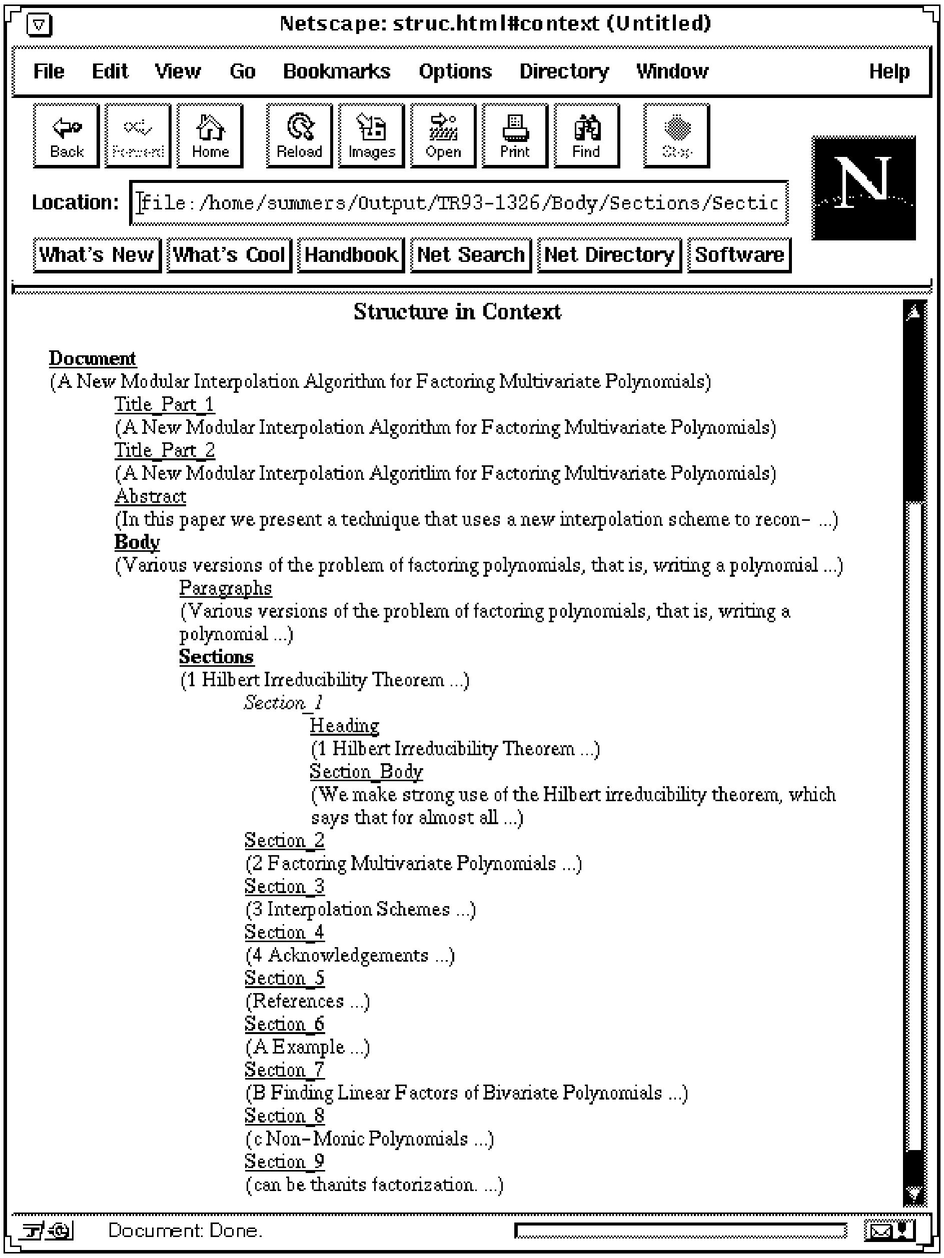


Figure 9.2: The structure in context from Figure 9.1, in its own window

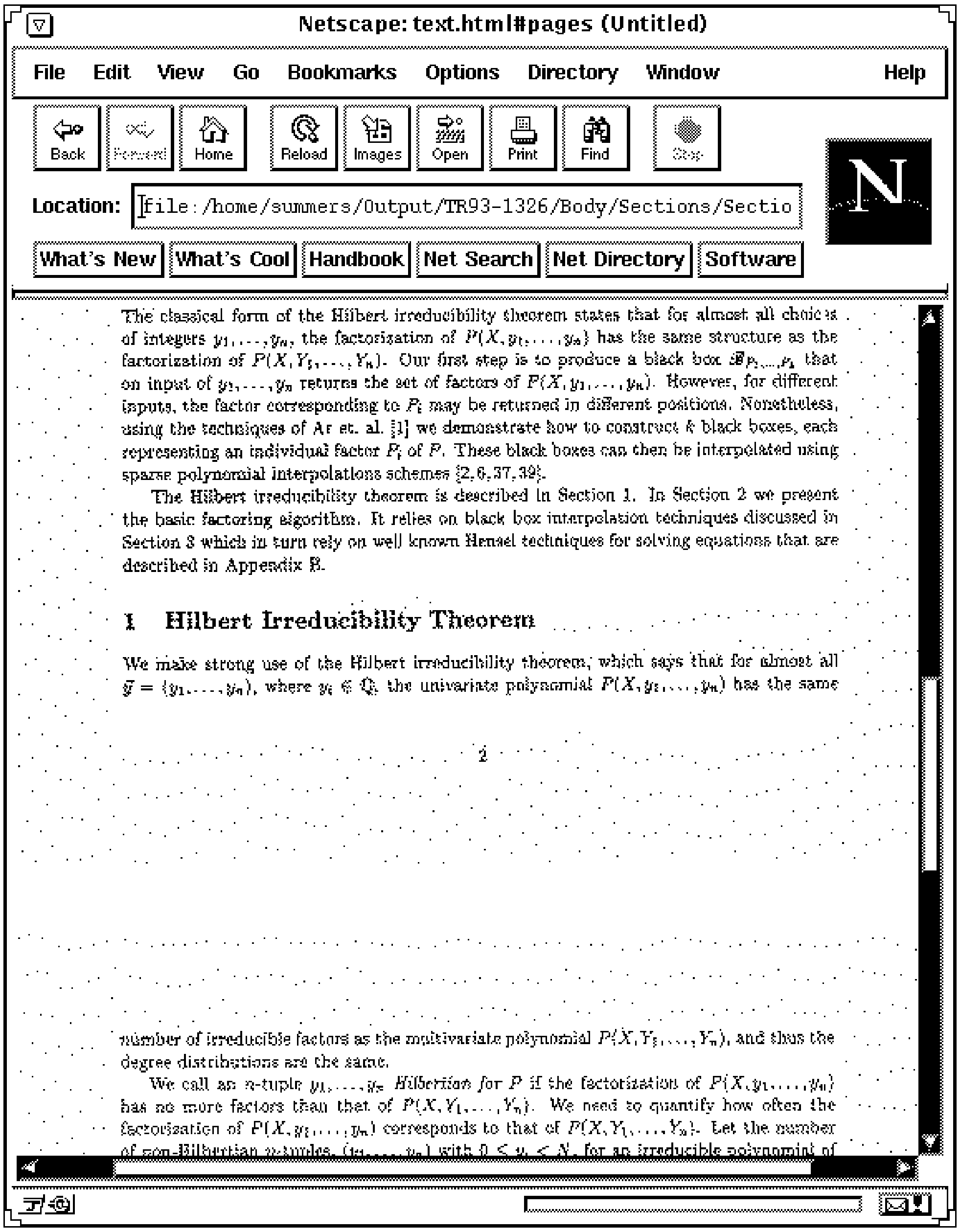


Figure 9.3: Part of the relevant gif images for Figure 9.1

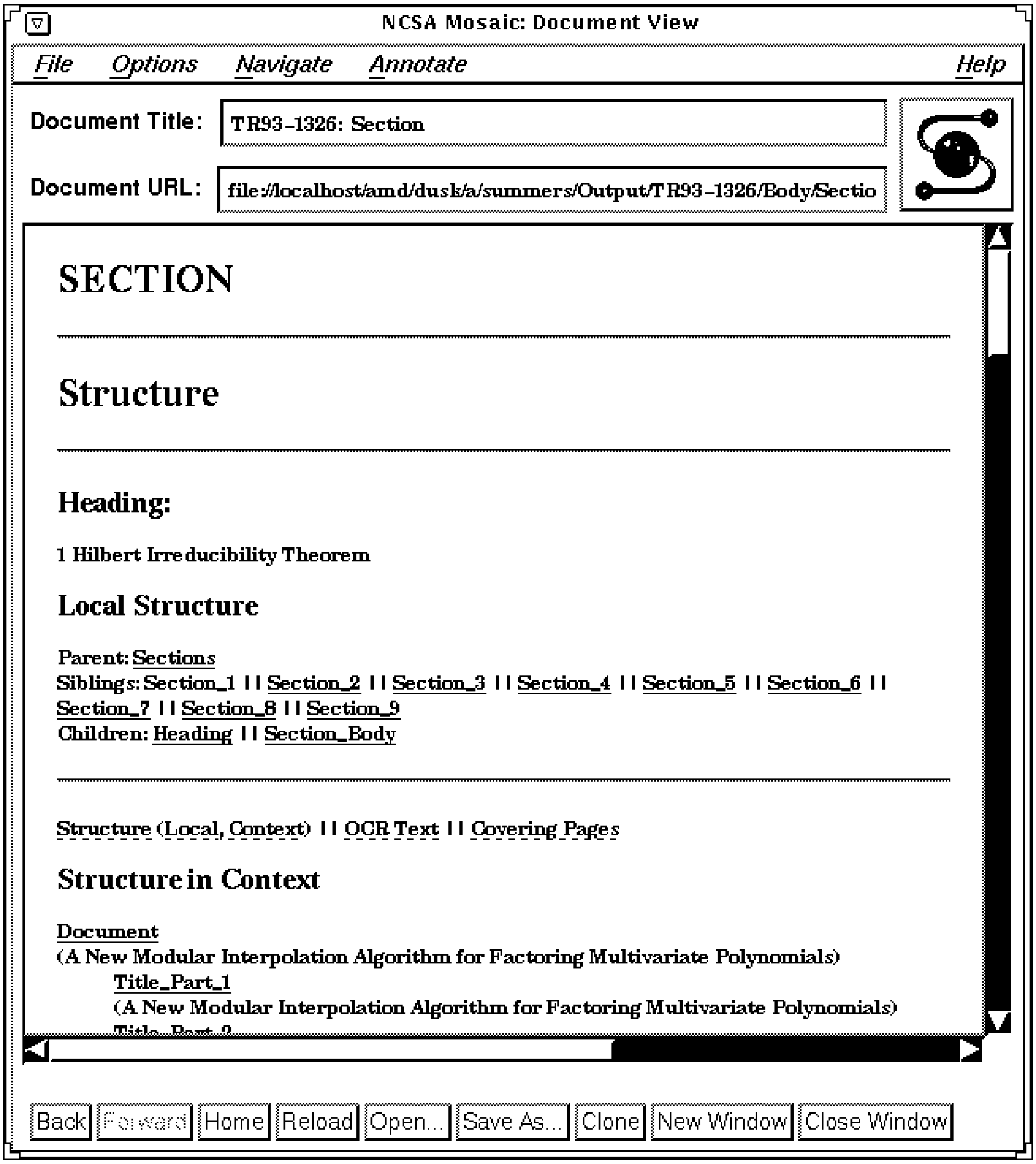


Figure 9.4: Beginning of the Mosaic non-frames rendering of the page in Figure 9.1

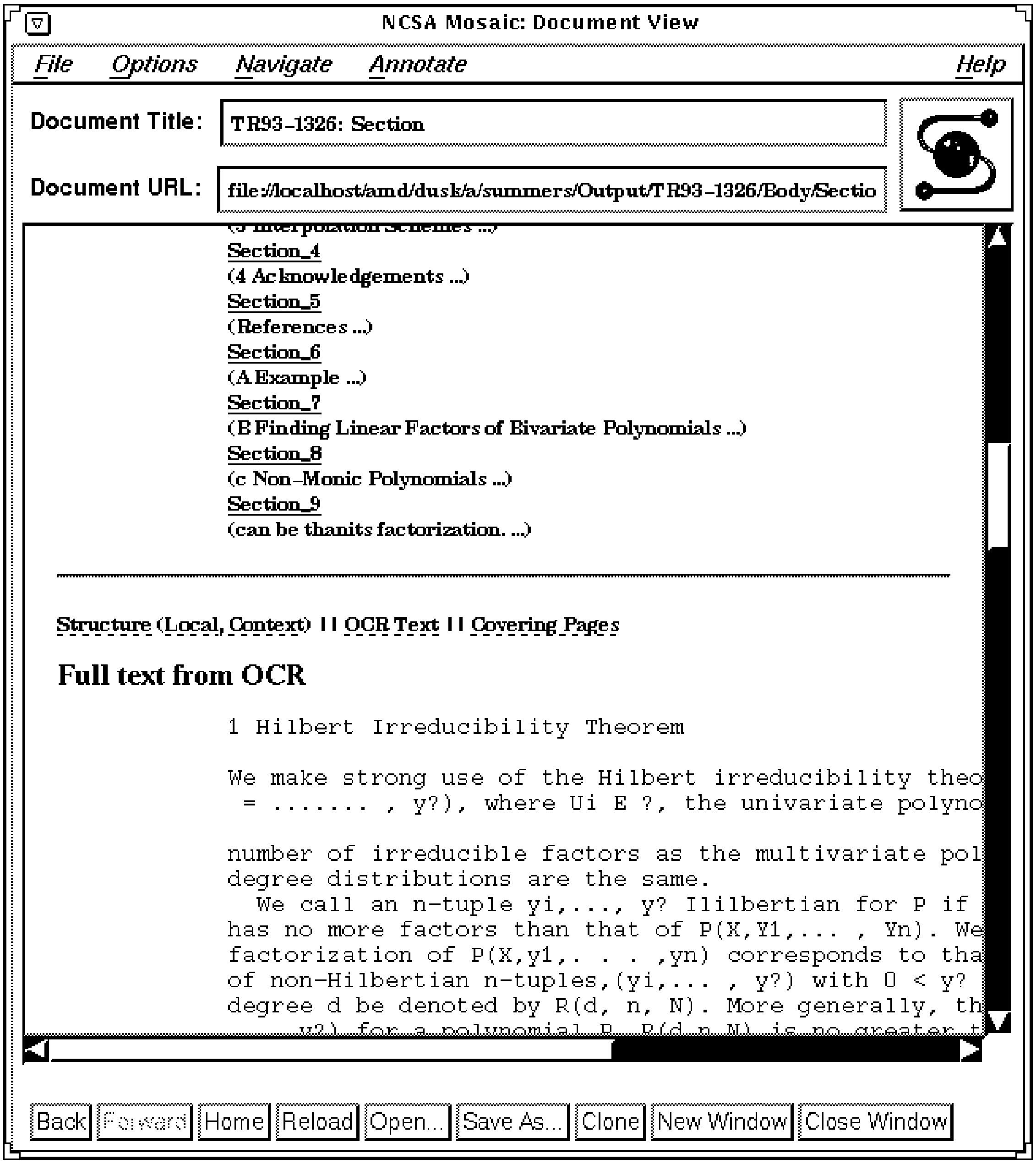


Figure 9.5: More of the Mosaic non-frames rendering of the page in Figure 9.1

Chapter 10

Conclusions

10.1 Contributions

This thesis addresses the problem of deriving a detailed logical hierarchy from the layout of a document of unknown style. Other work has presented appropriate methods for discovering logical structure in documents of known style, possibly including small deviations, and for discovering a single, flat set of blocks in a generic document. This thesis extends such work by presenting an approach that is appropriate for the version of the problem described above.

This thesis contributes an algorithm that differs from previous work in several ways, pursuant to the above goal.

- It incorporates the generic observations that structural forms tend to repeat within a document and that a structure that interrupts another is generally located horizontally within the latter.
- Classification is based on *cues*, which amount to a highly approximate matching system, rather than on precise matches or small deviations from a model.
- It separates the segmentation and classification processes while allowing feedback between them.
- It incorporates style knowledge (of particular types) when available, but it does not require this information to proceed.

LABLER implements the described algorithm for the discovery of a hierarchy of logical structure in a generic document, based on its layout. Its performance is a highly significant improvement over the simple baseline of forming a hierarchy based on vertical distance between lines and assigning all nodes to the most common category.

LABLER also generates output that represents the structure of a given document in HTML, with links to the document itself; this makes the logical hierarchy suitable for browsing.

10.2 Future Work

The performance of LABLER suggests several areas for future work.

OCR Errors LABLER relies heavily on accurate OCR input, with respect to line indentation, but such input may not be available. (For instance, several OCR errors were hand-corrected for the experiments in Chapter 8.) It would be a great improvement to the usability of LABLER's approach to have an automatic method of accounting for such errors. The approach in [16], based on probabilities of flipped pixels, is promising; how to combine this with the much looser comparisons used in a generic system like LABLER remains to be explored.

Separation Indentation patterns are sometimes repeated by chance, leading to segmentation errors. An accurate method of determining which layout attributes may serve as separators would diminish this problems; a pattern would only be identified as indicating a structure if it began and ended with indications of structure separation.

Integration with Block Segmentation The segmenter might also be substantially improved by integrating it with a flat block segmenter, such as one of those described in 2.1. Depending on their level of granularity, such blocks could be used as the original blocks for LABLER's algorithm, or they could form a required internal level of the hierarchy. In the latter case, one possibility would be to insert the blocks whenever LABLER's usual next level would have crossed their boundaries.

Parts and Wholes Many of LABLER's classification errors involve confusing two structures with a part/whole relationship, such as a paragraph and a paragraph part. This mistake follows naturally from the use of cues for classification; a part will contain many, if not all, of the cues that a whole does. Special processing may be required to provide reliable part/whole distinctions.

Character-Based Processing The other primary source of classification errors lies in identifying structures that are primarily identified by content, such as equations. This difficulty suggests that a more sophisticated form of character processing than LABLER's prototypes may be required in some cases; an extremely useful area of future work would be to find an appropriate middle ground between LABLER's simple measure of character presence and frequency and a costly analysis of content. Perhaps Information Retrieval techniques may be adapted to apply to the collection of characters found in a structure.

Evaluation The measurements in Chapter 8 offer a reasonable representation of the performance of a logical structure discovery algorithm, but they do not complete the picture. As discussed in Chapter 3, some structures have a greater significance than others, in general or for a particular application. A formal specification of how to reflect this fact in performance measures would be a great asset in presenting and comparing document analysis results.

Another open question in evaluation is how to combine the issue of generality with that of performance on appropriate documents, so that an approach, such as error-tolerant parsing, designed for use on documents of a single type with perhaps small deviations, may be compared to an approach designed for more general applicability. The former should yield better results than the latter on the documents to which it applies, but this should be a smaller set. How these issues are combined should probably be task-specific, but unless the task specification makes the choice of approach obvious, some kind of combination is needed for comparison.

Queries and Retrieval Approaches to querying structured documents and integrating logical structure with Information Retrieval were discussed in Section 2.5; these generally assume knowledge of the document type. Does the preferred approach change for a heterogeneous collection of documents of unknown types?

Semi-Structured Documents LABLER starts from the document layout, assuming no structure is available. In some cases, however, a document may be available in a form that mixes logical and layout specifications. For instance, many LaTeX documents have this property; an author may mix the use of logically-named macros with visual commands. (The ideal LaTeX author would define a macro with a logical name for every command in the document, but many authors are not primarily concerned with specifying logical structure.) The primary observations still apply, but structure discovery should work around the known structures; this situation is similar to the incorporation of partial style knowledge, but the known structures may be intermixed with structures to discover in any manner. HTML documents may also be considered semi-structured, although they incorporate the additional challenge that the provided logical labels are often misused [31, 72].

LABLER provides one step towards the use of layout-based documents in a structure-based setting, such as a digital library. With additional work on topics such as those outlined above, we can hope to advance to a point at which unstructured documents, both physical and electronically page-based, can be seamlessly incorporated into a structured environment.

Appendix A

Prototypes and Attributes

This appendix describes the generic prototypes used to classify primary structures. It also provides a list of all the attributes included in this process.

For the purposes of these specifications, let $\mathcal{C}$ be the set of all possible node classifications, both primary and secondary. Let $\mathcal{T}$ be the set of all node classifications that represent part or all of the title part of a document.

A.1 Prototypes

Necessary attributes are marked with asterisks.

Paragraph

`Geometry:` **General**: Left In, Left Out
Local: Left In, Left Out

`Context:` **Pred.**: `Blank`: yes
Succ.: `Blank`: yes

`Children:` **Negative**: { *List-Item*, *List* }, unmerged only
***Negative**: { *Section*, *Section Body*, *Sections*, *Abstract*, *Body* } $\cup\ \mathcal{T}$

`Fonts:` `Main Font`: normal weight, medium size

First Paragraph

`Geometry:` **General**: Left In, Left Out
Local: Left In, Left Out

`Context:` **Pred.**: `Blank`: yes, *`Category`: *Heading*
Succ.: `Blank`: yes
Typical: no

`Children:` **Negative**: { *List-Item*, *List* }, unmerged only
*__Negative__: { *Section*, *Section Body*, *Sections*, *Abstract*, *Body* } $\cup \mathcal{T}$

`Fonts:` `Main Font`: normal weight, medium size

Theorem

`Geometry:` **General**: Left In, Left Out
Local: Left In, Left Out

`Context:` **Pred.**: `Blank`: yes
Succ.: `Blank`: yes
Typical: no

*`Symbols:` **Positive**: ⟨ { Theorem, Lemma, Corollary, Proposition, Claim }, `start` ⟩

`Children:` **Negative**: { *List-Item*, *List* }, unmerged only
*__Negative__: { *Section*, *Section Body*, *Sections*, *Abstract*, *Body* } $\cup \mathcal{T}$

`Fonts:` `Main Font`: normal weight, medium size

Proof

`Geometry:` **General**: Left In, Left Out
Local: Left In, Left Out

`Context:` **Pred.**: `Blank`: yes, *`Category`: *Theorem*
Succ.: `Blank`: yes
Typical: no

*`Symbols:` **Positive**: ⟨ { Proof }, `start` ⟩

`Children:` **Negative**: { *List-Item*, *List* }, unmerged only
*__Negative__: { *Section*, *Section Body*, *Sections*, *Abstract*, *Body* } $\cup\ \mathcal{T}$

`Fonts: Main Font`: normal weight, medium size

Definition

`Geometry:` **General**: Left In, Left Out
Local: Left In, Left Out

`Context:` **Pred.**: `Blank`: yes
Succ.: `Blank`: yes
Typical: no

*`Symbols:` **Positive**: $\langle$ { Definition }, `start` $\rangle$

`Children:` **Negative**: { *List-Item*, *List* }, unmerged only
*__Negative__: { *Section*, *Section Body*, *Sections*, *Abstract*, *Body* } $\cup\ \mathcal{T}$

`Fonts: Main Font`: normal weight, medium size

Heading

`Context:` **Pred.**: `Blank`: yes, `Same`: no
Succ.: *`Blank`: yes `Same`: no

`Symbols:` **Positive**: $\langle$ { $0, 1, 2, 3, 4, 5, 6, 7, 8, 9$, Chapter, Section, Sub, Part, Book, Volume }, `start` $\rangle$
Negative: $\langle$ { (,), !, ., ? }, `end` $\rangle$

`Height:` 1
`Fonts: Main Font`: heavy weight, large size

Title Start `Geometry:` **General**: Centered

*`Context:` **Pred.**: `Blank`: yes, `Same`: no, `Category`: *Top*, `Geometry`: Top

*Children: **Negative**: $\mathcal{C} - \mathcal{T}$
Fonts: **Main Font**: heavy weight, large size

Title Element Geometry: **General**: Centered
Context: **Pred.**: Blank: yes, Same: yes, *Category: $\mathcal{T}$
Children: **Negative**: $\mathcal{C} - \mathcal{T}$
Fonts: **Main Font**: heavy weight, large size

Equation Geometry: **General**: Left In

Context: **Pred.**: Blank: yes, Same: no,
Category: { *Paragraph, First Paragraph, Theorem, Proof, Definition* }
Symbols: **Positive**: $\langle$ { : }, end $\rangle$
Succ.: Blank: yes, Same: no

Height: 1

Symbols: **Positive**: $\langle$ $\{=, >, <, >=, <=, +, /, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$, all $\rangle$
***Positive:** $\langle$ $\{=, >, <, >=, <=, +, /\}$, any $\rangle$

List Item

Geometry: **General**: Left In, Left In
Local: Left Justified, Left In

Context: **Pred.**: Blank: yes, Same: yes, Category: *List Item*
Succ.: Blank: yes, Same: yes

Symbols: **Positive**: $\langle$ $\{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$, start $\rangle$
Fonts: **Main Font**:: normal weight, medium size

List

Geometry: **General**: Left In
Local: Left Justified

Context: **Pred.**: Blank: yes, Same: no
Succ.: Blank: yes, Same: no

`Children:` ***Positive**: *List Item, List*
Negative: $\mathcal{C}$ − { *List Item*, *List* }

`Fonts:` **Main Font**: normal weight, medium size

Abstract

`Context:` **Pred.**: `Blank`: yes, `Same`: no, *`Category`: $\mathcal{T}$
Succ.: `Blank`: yes, `Same`: no

*`Symbols:` **Positive**: ⟨ { Abstract }, `start` ⟩
`Children:` **Positive**: *Paragraph*

A.2 Attributes

`Geometry`

General

- Left In, Left Out
- Left In, Left In
- Left In
- Centered

Local

- Left In, Left Out
- Left Justified, Left In
- Left Justified

`Context`

Predecessor

`Blank` : yes

`Same` : yes, no

`Geometry` : Top

`Symbols` : ⟨ { : }, `end` ⟩

`Category` :

- *Top*
- { *Title Start*, *Title Element* }
- *Heading*

- *List Item*
- { *Paragraph, First-Paragraph, Theorem, Proof, Definition* }
- *Theorem*

Successor

`Blank` : yes

`Same` : yes, no

Typicality : no

`Height:` 1

`Symbols`

Positive

- ⟨ { Theorem, Lemma, Corollary, Proposition, Claim }, `start` ⟩
- ⟨ { Proof }, `start` ⟩
- ⟨ { Definition }, `start` ⟩
- ⟨ {0, 1, 2, 3, 4, 5, 6, 7, 8, 9, Chapter, Section, Sub, Part, Book, Volume}, `start` ⟩
- ⟨ {0, 1, 2, 3, 4, 5, 6, 7, 8, 9}, `start` ⟩
- ⟨ {=, >, <, >=, <=, +, /, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9}, `all` ⟩
- ⟨ {=, >, <, >=, <=, +, /}, `any` ⟩
- ⟨ { Abstract }, `start` ⟩

Negative

- ⟨ { (,), !, ., ? }, `end` ⟩

`Children`

Positive

- { *List Item, List* }
- { *Paragraph* }

Negative

- { *List-Item, List* }, unmerged only
- { *Section, Section Body, Sections, Abstract, Body* } $\cup \mathcal{T}$
- $\mathcal{C} - \mathcal{T}$
- $\mathcal{C} -$ { *List Item, List* }

`Fonts`

Main Font

- normal weight, medium size
- heavy weight, large size

Appendix B

Style Knowledge for Cornell Computer Science Technical Reports

This appendix describes the specific knowledge LABLER uses in processing technical reports from the Cornell Department of Computer Science. These technical reports follow a strict format for the title page. A context-free grammar describing this format is given in Figure B.1. In this grammar, as well as all others in this thesis, the following conventions are observed: terminal symbols begin with lowercase letters; nonterminal symbols begin with uppercase letters; the left hand side of the first rule is the start symbol; right hand sides are given as regular expressions with concatenation represented by $\cdot$, optionality by ?, inclusion 0 or more times by *, inclusion 1 or more times by +, and grouping by parentheses.

Ideally, the Figure B.1 grammar would suffice, with all its terminals recognizable by the pattern of spacing and certain fixed linguistic patterns, such as dates. Errors in OCR can interfere with this identification, and the initial grouping of adjacent lines with identical left margins can also interfere. LABLER therefore uses a slightly more flexible grammar, in which many of the original terminals become nonterminals, which can be fuzzily matched. When parsing is complete, irrelevant nonterminals nodes in the parse tree are discarded.

$$
\begin{array}{rcl}
\text{TitlePage} & \rightarrow & \text{Title} \cdot \text{Author} \cdot \text{id} \cdot \text{date} \cdot \text{address} \cdot \text{thanks ?} \\
\text{Title} & \rightarrow & \text{title start} \cdot \text{Title}' \text{ ?} \\
\text{Title}' & \rightarrow & \text{title line} \cdot \text{Title}' \text{ ?} \\
\text{Author} & \rightarrow & \text{name} \mid \text{Names} \\
\text{Names} & \rightarrow & \text{name} \cdot \text{Names}' \\
\text{Names}' & \rightarrow & \text{name} \cdot \text{Names}' \text{ ?}
\end{array}
$$

Figure B.1: Grammar for Cornell CS Technical Report Title Pages

$$\begin{aligned}
\text{Document Start} &\rightarrow \text{Title Part} \cdot \text{Abstract} \\
\text{Title Part} &\rightarrow \text{Title} \cdot ((\text{Author} \cdot \text{date ?}) \mid \text{Authors}) \\
\text{Title} &\rightarrow \text{title start} \cdot \text{Title}' \text{ ?} \\
\text{Title}' &\rightarrow \text{title line} \cdot \text{Title}' \text{ ?} \\
\text{Authors} &\rightarrow \text{Author} \cdot \text{Authors}' \\
\text{Authors}' &\rightarrow \text{Author} \cdot \text{Authors}' \text{ ?} \\
\text{Author} &\rightarrow (\text{name} \mid \text{Names}) \cdot \text{Author Information ?} \\
\text{Names} &\rightarrow \text{name} \cdot \text{Names}' \\
\text{Names}' &\rightarrow \text{name} \cdot \text{Names}' \text{ ?} \\
\text{Author Information} &\rightarrow \text{affiliation line} \cdot \text{Author Information ?} \\
\text{Abstract} &\rightarrow \text{abstract heading} \cdot \text{Abstract Body} \\
\text{Abstract Body} &\rightarrow \text{sole paragraph} \mid \\
& (\text{start paragraph ?} \cdot \text{Abstract Paragraphs}) \\
\text{Abstract Paragraphs} &\rightarrow \text{Last Paragraph} \mid \\
& (\text{Middle Paragraph} \cdot \text{Abstract Paragraphs}) \\
\text{Middle Paragraph} &\rightarrow \text{indent} \cdot \text{outdent_followed_by_text} \\
\text{End Paragraph} &\rightarrow \text{indent} \cdot \text{outdent_followed_by_blank}
\end{aligned}$$

Figure B.2: Grammar for Cornell CS Technical Report Text Following Title Page

The observed group of technical reports also displays consistency in the text immediately following the title page. This text includes a reiteration of title and author information followed by an abstract. The format is not strict, but the style can be expressed by the grammar in Figure B.2.

Bibliography

[1] O.T. Akindele and A. Belaïd. Construction of generic models of document structures using inference of tree grammars. In *Proceedings of the Third International Conference on Document Analysis and Recognition* [44], pages 206–209.

[2] James Allan, Jim Davis, Dean Krafft, Daniela Rus, and Devika Subramanian. Information agents for building hyperlinks. In *Proceedings of the ACM Conference on Information and Knowledge Management*, Washington DC, November 1993.

[3] A. Antonacopoulos and R. T. Ritchings. Representation and classification of complex-shaped printed regions using white tiles. In *Proceedings of the Third International Conference on Document Analysis and Recognition* [44], pages 1132–1135.

[4] Dennis S. Arnon. Scrimshaw: A language for document queries and transformations. *Electronic Publishing: Origination, Dissemination and Design*, 6(4):385–396, 1993.

[5] Nick Ayres and Tom Wesley. Using structure within electronic documents to make editors more accessible. In Zagler et al. [97], pages 198–205.

[6] Antoine Azokly and Rolf Ingold. A language for document generic layout description and its use for segmentation into regions. In *Proceedings of the Third International Conference on Document Analysis and Recognition* [44], pages 1123–1126.

[7] Henry S. Baird. Anatomy of a versatile page reader. *Proceedings of the IEEE*, 80(7):1059–1065, 1992.

[8] Bart Bauwens, Jan Engelen, Filip Evenepoel, Chris Tobin, and Tom Wesley. Structuring documents: the key to increasing access to information for the print disabled. In Zagler et al. [97], pages 214–221.

[9] T. A. Bayer and H. Walischewski. Experiments on extracting structural information from paper documents using syntactic pattern analysis. In *Proceedings of the Third International Conference on Document Analysis and Recognition* [44], pages 476–479.

[10] Abdel Belaïd, Julian C. Anigbogu, and Yannick Chenevoy. Qualitative analysis of low-level logical structures. *Electronic Publishing: Origination, Dissemination and Design*, 6(4):435–446, 1993.

[11] Allen Brown, Anne Bruggemann-Klein, and An Feng, editors. *Electronic Publishing: Origination, Dissemination and Design,* 8(2–3). Special Issue: *Proceedings of the Sixth International Conference on Electronic Publishing, Document Manipulation and Typography.* Wiley, 1996.

[12] R. Brugger, A. Zramdini, and R. Ingold. Modeling documents for structure recognition using generalized N-Grams. In *Proceedings of the Fourth International Conference on Document Analysis and Recognition* [45], pages 56–60.

[13] John F. Buford. Evaluation of a query language for structured hypermedia documents. In James Ford, Fillia Makedon, and Samuel A. Rebelsky, editors, *Electronic Publishing and the Information Superhighway: Proceedings of the Dartmouth Institute for Advanced Graduate Studies*, pages 105–116, Boston, May 1995. Birkhäuser.

[14] Victoria A. Burrill. VORTEXT: VictORias TEXT reading and authoring system. In J. C. van Vliet, editor, *Text Processing and Document Manipulation: Proceedings of the International Conference*, British Computer Society Workshop Series, pages 43–57, Nottingham, April 1986. Cambridge University Press.

[15] Francine R. Chen and Dan S. Bloomberg. Extraction of indicative summary sentences from imaged documents. In *Proceedings of the Fourth International Conference on Document Analysis and Recognition* [45], pages 227–232.

[16] Philip A. Chou and Gary E. Kopec. A stochastic attribute grammar model of document production and its use in document image decoding. In Luc M. Vincent and Henry S. Baird, editors, *Proceedings: Document Recognition II*, volume 2422 of *SPIE Proceedings Series*, pages 66–73. SPIE–The International Society for Optical Engineering, 1995.

[17] C. L. A. Clarke, G. V. Cormack, and F. J. Burkowski. An algebra for structured text search and a framework for its implementation. *Computer Journal*, 38(1):43–56, 1995.

[18] Charles L. A. Clarke, G. V. Cormack, and F. J. Burkowski. An algebra for structured text search and a framework for its implementation, August 1994. URL: `ftp://cs-archive.uwaterloo.ca/cs-archive/CS-94-30/structxt.dvi`.

[19] Nelson Cowan. *Attention and Memory: An Integrated Framework.* Oxford Psychology Series. Oxford University Press, Clarendon Press, New York, 1995.

[20] Bruce Croft. What do people want from information retrieval? (The top 10 research issues for companies that use and sell IR systems). *D-Lib Magazine*, November 1995. URL: `http://www.dlib.org/dlib.november95/11croft.html`.

[21] Olivier Déforges and Barba Dominique. Segmentation of complex documents multilevel images: a robust and fast text bodies-headers detection and extraction scheme. In *Proceedings of the Third International Conference on Document Analysis and Recognition* [44], pages 770–773.

[22] Andreas Dengel and Frank Dubiel. Computer understanding of document structure. In *International Journal of Imaging Systems and Technology,* 7(4) [81], pages 271–278.

[23] Denise Derrien and Michel Habib. Approche objet pour l'analyse de la structure logique des documents. In Jacques André and Jean Bézivin, editors, *Woodman '89: Workshop on Object-Oriented Document Manipulation*, pages 226–235, Rennes, May 1989.

[24] D. Doermann, A. Rosenfeld, and E. Rivlin. The function of documents. In *Proceedings of the Fourth International Conference on Document Analysis and Recognition* [45], pages 1077–1081.

[25] Shona Douglas, Matthew Hurst, and David Quinn. Using natural language processing for identifying and interpreting tables in plain text, September 1994. URL: `file://ftp.cogsci.ed.ac.uk/pub/shona/Tables.ps.gz`.

[26] Floriana Esposito, Donato Malerba, and Giovanni Semeraro. A knowledge-based approach to layout analysis. In *Proceedings of the Third International Conference on Document Analysis and Recognition* [44], pages 466–471.

[27] Peter Fankhauser and Yi Xu. *MarkItUp!* An incremental approach to document structure recognition. *Electronic Publishing: Origination, Dissemination and Design*, 6(4):447–456, 1993.

[28] Jon Fausey and Keith Shafer. All my data is in SGML. Now what? *Journal of the American Society for Information Science*, 48(7):638–643, 1997.

[29] An Feng and Toshiro Wakayama. SIMON: A grammar based transformation system for structured documents. *Electronic Publishing: Origination, Dissemination and Design*, 6(4), 1993.

[30] Lloyd Alan Fletcher and Rangachar Kasturi. A robust algorithm for text string separation from mixed text/graphics images. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 10(6):910–918, November 1988.

[31] Peter Flynn. W[h]ither the Web? the extension or replacement of HTML. *Journal of the American Society for Information Science*, 48(7):614–621, 1997.

[32] Peter Flynn, Terry Allen, Tom Borgman, Tim Bray, Robin Cover, Chirstopher Maden, Eve Maler, Peter Murray-Rust, Liam Quin, Michael Sperberg-McQueen, Joel Weber, and Makoto Murata. *Frequently Asked Questions about the Extensible Markup Language: The XML FAQ, Version 1.21*, February 1998. URL: `http://www.ucc.ie./xml/`.

[33] E. Fox, Q. Chen, and R. France. Integrating search and retrieval with hypertext. In E. Berk and J. Devlin, editors, *Hypertext/Hypermedia Handbook*, pages 329–355. McGraw-Hill, New York, 1991.

[34] Hiromichi Fujisawa, Itsuko Kiuchi, Takuo Koguchi, and Hidefumi Kondo. A visual user interface for a personal information base using a concept network. In *Database Systems for Advanced Applications '91: Proceedings of the Second International Symposium*, pages 69–78, Tokyo, April 1991. World Scientific.

[35] Hiromichi Fujisawa, Yoshihiro Shima, Masashi Koga, and Tatsuya Murakami. Automatically organizing document bases using document understanding techniques. In *Future Databases '92: Proceedings of the Second Far-East Workshop on Future Database Systems*, pages 244–253, Kyoto, April 1992. World Scientific.

[36] Hiromichi Fujisawa, Hiroshi Yashiro, Jun'ichi Higashino, Yoshihiro Shima, Yasuaki Nakano, and Tatsuya Murakami. Document analysis and decomposition method for multimedia contents retrieval. In *Proceedings of the Second International Syposium on Interoperable Information Systems*, pages 231–238, Tokyo, November 1988.

[37] Charles F. Goldfarb. *The SGML Handbook*. Clarendon Press, Oxford, 1990.

[38] Jaekya Ha, Robert Haralick, and Ihsin Phillips. Document page decomposition by the bounding-box projection technique. In *Proceedings of the Third International Conference on Document Analysis and Recognition* [44], pages 1119–1122.

[39] Xiaolong Hao, Jason T. L. Wang, and Peter A. Ng. Information extraction from the structured part of office documents. *Information Sciences*, 91(3-4):245–274, 1996.

[40] Marti A. Hearst and Christian Plaunt. Subtopic structuring for full-length document access. In *Proceedings of the Sixteenth Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 59–68, Pittsburgh, 1993.

[41] M. H. Heine. A provisional notation for describing the information structure of documents. *Journal of Documentation*, 51(4):339–359, 1995.

[42] John E. Hopcroft and Jeffrey D. Ullman. *Introduction to Automata Theory, Languages, and Computation*. Addison-Wesley, Reading, 1979.

[43] Tao Hu and Rolf Ingold. A mixed approach toward an efficient logical structure recognition from document images. *Electronic Publishing: Origination, Dissemination and Design*, 6(4):457–468, 1993.

[44] International Assocation for Pattern Recognition TC-11, TC-10, Canadian Image Processing and Pattern Recognition Society, Centre for Pattern Recognition and Machine Intelligence, Institute of Electrical and Electronics Engineers, Section Montréal, Laboratoire Scribens, International Graphonomics Society, Centre de recherche Informatique de Montréal, and Institute for Robotics and Intelligent Systems. *Proceedings of the Third International Conference on Document Analysis and Recognition*. IEEE Computer Society Press, August 1995.

[45] International Association for Pattern Recognition, TC 10 and 11, International Graphonomics Society, German Association for Computer Science, and German Association for Information Technology. *Proceedings of the Fourth International Conference on Document Analysis and Recognition.* IEEE Computer Society Press, August 1997.

[46] Yasuto Ishitani. Document layout analysis based on emergent computation. In *Proceedings of the Fourth International Conference on Document Analysis and Recognition* [45], pages 45–50.

[47] D. J. Ittner and H. S. Baird. Language-free layout analysis. In *Proceedings of the Second Annual International Conference on Document Analysis and Recognition*, pages 336–340. IEEE Computer Society Press, October 1993.

[48] A. Jain and S. Bhattacharjee. Address block location on envelopes using Gabor filters. *Pattern Recognition*, 25(12), 1992.

[49] Anil K. Jain and Bin Yu. Page segmentation using document model. In *Proceedings of the Fourth International Conference on Document Analysis and Recognition* [45], pages 34–37.

[50] Tao Jiang, Lusheng Wang, and Kaizhong Zhang. Alignment of trees – an alternative to tree edit. In *Combinatorial Pattern Matching: 5th Annual Symposium, CPM 94*, Lecture Notes in Computer Science, pages 75–86. Springer-Verlag, Asilomar, California, June 1994.

[51] Kunihiko Kaneko and Akifumi Makinouchi. Data storage and query processing for structured document databases. In Roland R. Wagner, editor, *Proceedings: Eighth International Workshop on Database and Expert Systems Applications*, pages 92–97. IEEE Computer Society, 1997.

[52] Michael H. Kay. Textmaster – document filing and retrieval using ODA. In J. C. van Vliet, editor, *Text Processing and Document Manipulation: Proceedings of the International Conference*, British Computer Society Workshop Series, pages 125–139, Nottingham, April 1986. Cambridge University Press.

[53] Bertin Klein and Peter Fankhauser. Error tolerant document structure analysis. In *IEEE International Forum on Research and Technology on Advances in Digital Libraries: ADL '97*, pages 116–127. IEEE Computer Society Press, 1997.

[54] Gary E. Kopec. Document image decoding in the UC Berkeley digital library. In Luc M. Vincent and Jonathon J. Hull, editors, *Proceedings: Document Recognition III*, volume 2660 of *SPIE Proceedings Series*, pages 2–13. SPIE–The International Society for Optical Engineering, 1996.

[55] D. Kroemker, editor. *Computer Networks and ISDN Systems: The International Journal of Computer and Telecommunications Networking,* 27(6). Special Issue: *Proceedings of the Third International World-Wide Web Conference.*, 1995.

[56] E. Kuikka and M. Penttonen. Transformation of structured documents. *Electronic Publishing: Origination, Dissemination, and Design*, 8(4):319–341, 1995.

[57] E. Kuikka and A. Salminen. Filtering structured documents in the SYNDOC environment. In *Electronic Publishing: Origination, Dissemination and Design,* 8(2–3) [11], pages 181–193.

[58] Mounia Lalmas. Dempster-Shafer's theory of evidence applied to structured documents: modelling uncertainty. In Nicholas J. Belkin, A. Desai Narasimhalu, and Peter Willett, editors, *Proceedings of the 20th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval.* Special Issue of the *SIGIR Forum*, pages 110–118, 1997.

[59] Christopher Lewis, Daniela Rus, and Matthew Scott. A structure detector for tables. Forthcoming Technical Report.

[60] Y. H. Liu-Gong, B. Dubuisson, and H. N. Pham. A general analysis system for document's layout structure recognition. In *Proceedings of the Third International Conference on Document Analysis and Recognition* [44], pages 597–600.

[61] William S. Lovegrove and David F. Brailsford. Document analysis of PDF files: methods, results and implications. In *Electronic Publishing: Origination, Dissemination and Design,* 8(2–3) [11], pages 207–220.

[62] Yves Marcoux and Martin S'evigny. Why SGML? Why now? *Journal of the American Society for Information Science*, 48(7):584–592, 1997.

[63] Ethan V. Munson. A new presentation language for structured documents. In *Electronic Publishing: Origination, Dissemination and Design,* 8(2–3) [11], pages 125–138.

[64] Makoto Murata. File format for documents containing both logical and layout structures. *Electronic Publishing: Origination, Dissemination and Design*, 8(4):295–317, 1995.

[65] G. Nagy, S. Seth, and M. Vishwanathan. A prototype document image analysis system for technical journals. *Computer*, 25(7), 1992.

[66] Debashish Niyogi and Sargur Srihari. An integrated approach to document decomposition and structural analysis. In *International Journal of Imaging Systems and Technology,* 7(4) [81], pages 330–342.

[67] Debashish Niyogi and Sargur Srihari. The use of document structure analysis to retrieve information from documents in digital libraries. In Luc M. Vincent and Jonathon J. Hull, editors, *Proceedings: Document Recognition IV*, volume 3027 of *SPIE Proceedings Series*, pages 207–218. SPIE–The International Society for Optical Engineering, 1997.

[68] Pat Norrish. Semantic structures of text. In Jacques André, Richard Furuta, and Vincent Quint, editors, *Structured Documents*, The Cambridge Series on Electronic Publishing, pages 143–159. Cambridge University Press, Cambridge, 1989.

[69] Ulrich Pfeifer, Norbert Fuhr, and Tung Huynh. Searching structured documents with the enhanced retrieval functionality of freeWAIS-sf and SFgate. In *Computer Networks and ISDN Systems: The International Journal of Computer and Telecommunications Networking,* 27(6) [55], pages 1027–1036.

[70] Benoit Poirier and Michel Dagenais. An interactive system to extract structured text from a geometrical representation. In *Proceedings of the Fourth International Conference on Document Analysis and Recognition* [45], pages 342–346.

[71] Gilbert B. Porter and Emil V. Rainero. Document reconstruction: A system for recovering document structure from layout. In *Proceedings of the Conference on Electronic Publishing*, pages 127–141, 1992.

[72] Vincent Quint, Cécile Roisin, and Irène Vatton. A structured authoring environment for the World-Wide Web. In Kroemker [55], pages 831–840.

[73] Dave Raggett, Arnaud Le Hors, and Ian Jacobs. *HTML 4.0 Specification*, 1997. URL: `http://www.w3.org/TR/REC-html40/`.

[74] M. Armon Rahgozar, Zhigang Fan, and Emil V. Rainero. Tabular document recognition. In *SPIE Proceedings*, San Jose, February 1994.

[75] T. V. Raman. *Audio System for Technical Readings.* PhD thesis, Cornell University, May 1994. URL: `http://www.cs.cornell.edu/Info/People/raman/phd-thesis/`.

[76] Daniela Rus and Devika Subramanian. Multi-media RISSC Informatics: Retrieving Information with Simple Structural Components. In *Proceedings of the ACM Conference on Information and Knowledge Management*, Washington DC, November 1993.

[77] Daniela Rus and Kristen Summers. Using non-textual cues for electronic document browsing. In Nabil R. Adam, Bharat K. Bhargava, and Yelena Yesha, editors, *Digital Libraries: Current Issues*, Lecture Notes in Computer Science, chapter 9, pages 129 – 162. Springer-Verlag, 1995.

[78] Gerard Salton. *Automatic Text Processing: The Transformation, Analysis, and Retrieval of Information by Computer.* Addison-Wesley Publishing Company, Reading, 1989.

[79] D. Sankoff and J. Kruskal. *Time Warps, String Edits, and Macromolecules: The Theory and Practice of Sequence Comparison.* Addison-Wesley, 1983.

[80] Philip N. Smith and David F. Brailsford. Towards structured, block-based PDF. In *Electronic Publishing: Origination, Dissemination and Design,* 8(2–3) [11], pages 153–165.

[81] Sargur N. Srihari and Debashish Niyogi, editors. *International Journal of Imaging Systems and Technology,* 7(4). Special Issue: *Document Analysis and Recognition*, 1996.

[82] Kazuo Sumita, Seiji Miike, and Kenji Ono. Automatic abstract generation based on document structure. *Systems and Computers in Japan*, 26(13):32–42, 1995. Translated from *Denshi Joho Rsushin Gakkai Ronbunshi*, Volume J78-D-II, Number 3, pp. 511–519.

[83] Atsuhiro Takasu, Shin'ichi Satoh, and Eishi Katsura. A rule learning method for academic document image processing. In *Proceedings of the Third International Conference on Document Analysis and Recognition* [44], pages 239–242.

[84] Jacco van Ossenbruggen, Anton Eliëns, and Bastiaan Schönhage. Web applications and SGML. In Brown et al. [11], pages 51–62.

[85] Mahesh Viswanathan, Edward Green, and M. S. Krishnamoorthy. Document recognition: an attribute grammar approach. In Luc M. Vincent and Jonathon J. Hull, editors, *Proceedings: Document Recognition III*, volume 2660 of *SPIE Proceedings Series*, pages 101–111. SPIE–The International Society for Optical Engineering, 1996.

[86] Hanno Walischewski. Automatic knowledge acquisition for spatial document interpretation. In *Proceedings of the Fourth International Conference on Document Analysis and Recognition* [45], pages 243–247.

[87] Bing Wang. The design of an integrated information system. In Roland R. Wagner and Helmut Thoma, editors, *Database and Expert Systems Applications: 7th International Conference, DEXA '96 Proceedings*, pages 479–488. Springer-Verlag, 1996.

[88] Toyohide Watanabe and Xiaou Huang. Automatic acquisition of layout knowledge for understanding business cards. In *Proceedings of the Fourth International Conference on Document Analysis and Recognition* [45], pages 216–220.

[89] Toyohide Watanabe, Qin Luo, and Noboru Sugie. Structure recognition methods for various types of documents. *Machine Vision and Applications*, 6:163–176, 1993.

[90] Gio Wiederhold. Mediators in the architecture of future information systems. *Computer*, pages 38–49, March 1992.

[91] Ian H. Witten, Alistair Moffat, and Timothy C. Bell. *Managing Gigabytes: Compressing and Indexing Documents and Images*, chapter 8 (Mixed Text and Images), pages 295–327. Van Nostrand Reinhold, New York, 1994.

[92] K. Y. Wong, R.G. Casey, and F. M. Wahl. Document analysis system. *IBM Journal of Research and Development*, 26(6):647–656, November 1982.

[93] W3C's HTML home page. URL: `http://www.w3.org/MarkUp/`, March 1998. Maintained by the World Wide Web Consortium.

[94] Haviland Wright. SGML frees information. *Byte*, 17, June 1992.

[95] Hiroshi Yashiro, Tatsuya Murakami, Yoshihiro Shima, Yasuaki Nakano, and Hiromichi Fujisawa. A new method of document structure extraction using generic layout knowledge. In *International Workshop on Industrial Applications of Machine Intelligence and Vision*, Tokyo, April 1989.

[96] Lin-Ju Yeh, Hsiu-Hsen Yao, and Yuan-Kuo Chen. SSQL: a semi-structured query language for SGML document retrievals. In *The 14th Annual International Conference on Computer Documentation: Conference Proceedings*, pages 221–228, 1996.

[97] Wolfgang L. Zagler, Geoffrey Busby, and Roland R. Wagner, editors. *Computers for Handicapped Persons: 4th International Conference, ICCHP '94 Proceedings*. Lecture Notes in Computer Science. Springer-Verlag, 1994.