

Lightweight Structure in Text

Robert C. Miller

May 2002

CMU-CS-02-134

CMU-HCII-02-103

School of Computer Science
Computer Science Department
Carnegie Mellon University
Pittsburgh, PA

Thesis Committee:

Brad A. Myers, Co-chair

David Garlan, Co-chair

James H. Morris

Brian Kernighan, Princeton University

*Submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy*

Copyright © 2002 Robert C. Miller

This research was sponsored by the National Science Foundation under grant no. IRI-9319969 and grant no. IIS-0117658, the Army Research Office (ARO) under National Defense Science and Engineering Grant no. DAAH04-95-1-0552, the Defense Advanced Research Project Agency (DARPA) - US Army under contract no. DAAD1799C0061, DARPA - Navy under contract no. N66001-94-C-6037, N66001-96-C-8506, and the USENIX Association.

The views and conclusions contained herein are those of the author and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of any sponsoring party or the U.S. Government.

Keywords: text processing, structured text, pattern matching, regular expressions, grammars, web automation, repetitive text editing, machine learning, programming-by-demonstration, simultaneous editing, outlier finding, LAPIS

Abstract

Pattern matching is heavily used for searching, filtering, and transforming text, but existing pattern languages offer few opportunities for reuse. *Lightweight structure* is a new approach that solves the reuse problem. Lightweight structure has three parts: a model of text structure as contiguous segments of text, or *regions*; an extensible library of structure abstractions (e.g., HTML elements, Java expressions, or English sentences) that can be implemented by any kind of pattern or parser; and a *region algebra* for composing and reusing structure abstractions. Lightweight structure does for text pattern matching what procedure abstraction does for programming, enabling construction of a reusable library.

Lightweight structure has been implemented in LAPIS, a web browser/text editor that demonstrates several novel techniques:

- *Text constraints* is a new pattern language for composing structure abstractions, based on the region algebra. Text constraint patterns are simple and high-level, and user studies have shown that users can generate and comprehend them.
- *Simultaneous editing* uses multiple selections for repetitive text editing. Multiple selections are inferred from examples given by the user, drawing on the lightweight structure library to make fast, accurate, domain-specific inferences from very few examples. In user studies, simultaneous editing required only 1.26 examples per selection, approaching the 1-example ideal.
- *Outlier finding* draws the user's attention to inconsistent selections or pattern matches — both possible false positives and possible false negatives. When integrated into simultaneous editing and tested in a user study, outlier finding reduced user errors.
- *Unix tools for structured text* extend tools like `grep` and `sort` with lightweight structure, and the *browser shell* integrates a Unix command prompt into a web browser, offering new ways to build pipelines and automate web browsing.

Theoretical contributions include a formal definition of the region algebra, data structures and algorithms for efficient implementation, and a characterization of the classes of languages recognized by algebra expressions.

Lightweight structure enables efficient composition and reuse of structure abstractions defined by various kinds of patterns and parsers, bringing improvements to pattern matching, text processing, web automation, repetitive text editing, inference of patterns from examples, and error detection.

Contents

1	Introduction	1
1.1	Applications	4
1.2	LAPIS	5
1.3	Limitations	8
1.4	Contributions	9
1.5	Thesis Overview	10
2	Related Work	13
2.1	Parser Generators	13
2.2	Markup Languages	14
2.3	Structured Text Editors	15
2.4	Structure-Aware User Interfaces	17
2.5	Pattern Matching	17
2.6	Web Automation	19
2.7	Repetitive Text Editing	19
2.8	Inferring Text Patterns From Examples	20
2.9	Error Detection	21
3	Region Algebra	23
3.1	Regions	25
3.2	Region Relations	25
3.3	Region Space	26
3.4	Region Set Types	31
3.5	Region Algebra	33
3.5.1	Set Operators	33
3.5.2	Relational Operators	34
3.5.3	Iteration Operator	35
3.6	Derived Operators	35
3.6.1	Adjacency	35
3.6.2	Strictness	37
3.6.3	Overlap	37
3.6.4	Min and Max	38
3.6.5	Counting	39
3.6.6	Span	40
3.6.7	Concatenation	40

3.6.8	Delimiters	41
3.6.9	Hierarchies	42
3.6.10	Before/After with Restricted Range	44
3.6.11	Split	44
3.6.12	Flatten	45
3.6.13	Ignoring Background	46
3.6.14	Trim	48
4	Region Algebra Implementation	51
4.1	Region Rectangles	51
4.2	Rectangle Collection	54
4.3	Implementing the Region Algebra	59
4.4	Data Structures	61
4.4.1	R-Trees	61
4.4.2	Quadtrees	66
4.4.3	Rectangles as Points	68
4.5	Specialized Data Structures	71
4.5.1	Region Arrays	71
4.5.2	Syntax Trees	73
4.6	Optimizations	76
4.6.1	Trimming Overlaps	76
4.6.2	Preserving Lexicographic Order	79
4.6.3	Point Collections	79
4.6.4	Reordering Operands	81
4.6.5	Tandem Traversal	81
4.6.6	Quadtree Traversal	84
4.6.7	Plane-Sweep Intersection	84
4.6.8	Counting, Min, and Max	86
4.7	Performance	87
4.7.1	Random Microbenchmarks	89
4.7.2	Realistic Benchmarks	92
5	Language Theory	97
5.1	Preliminary Definitions	97
5.2	Finite State Transducers	98
5.3	Languages Recognized by a Region Expression	100
5.4	Restricted Algebras	102
5.5	Algebra Semantics	103
5.6	Trivial Ground Terms (RST_{\emptyset})	104
5.7	Finite Ground Terms ($RST_{\mathcal{F}}$)	105
5.7.1	Noncounting Languages (NC)	106
5.7.2	$RST_{\mathcal{F}} = NC$	106
5.8	Regular Ground Terms ($RST_{\mathcal{R}}$)	108
5.9	Context-Free Ground Terms ($RST_{\mathcal{CFL}}$)	109
5.10	The <i>forall</i> Operator ($RSTA_{\mathcal{R}}$ and friends)	113

5.10.1	N-way Regular Relations	113
5.10.2	Algebra Semantics With <i>forall</i>	115
5.10.3	$RSTA_{\mathcal{R}} = R$	117
5.10.4	Open Questions: $RSTA_{\emptyset}$, $RSTA_{\mathcal{F}}$, $RSTA_{\mathcal{C}\mathcal{F}\mathcal{L}}$	118
5.11	Summary	119
6	Text Constraints	121
6.1	Goals	121
6.2	Language Description	123
6.2.1	Basic Lexical Rules	123
6.2.2	Basic Syntactic Rules	124
6.2.3	Pattern Identifiers	124
6.2.4	Parsers	125
6.2.5	Pattern Definitions	126
6.2.6	Visible and Hidden Identifiers	128
6.2.7	Namespaces	129
6.2.8	Literals	133
6.2.9	Inferring Quotes	134
6.2.10	Regular Expressions	136
6.2.11	Matching Against HTML	138
6.2.12	Coordinate Maps	139
6.2.13	Pattern Matching Operators	140
6.2.14	Ignoring Background	142
6.2.15	Boolean Operators	143
6.2.16	Operator Precedence and Indentation	145
6.2.17	Parsing Indentation Structure	148
6.3	Examples	154
6.3.1	Plain Text	154
6.3.2	Web Pages	155
6.3.3	Source Code	156
7	LAPIS User Interface	159
7.1	Overview of LAPIS	159
7.2	Multiple Selections	161
7.3	Highlighting Multiple Selections	162
7.3.1	Flat Region Sets	162
7.3.2	Region Sets Created by Unary Relational Operators	162
7.3.3	Nested and Overlapping Region Sets	164
7.3.4	Scrollbar Augmentation	166
7.4	Making Multiple Selections	168
7.4.1	Selecting Regions with the Mouse	168
7.4.2	Selecting from the Library	169
7.4.3	Selecting by Pattern Matching	170
7.5	Editing with Multiple Selections	172
7.6	User Study	173

7.6.1	Goals	173
7.6.2	Procedure	175
7.6.3	Results	180
7.6.4	Summary	186
8	Commands and Scripting	187
8.1	Text-Processing Commands	188
8.1.1	Extract	188
8.1.2	Sort	193
8.1.3	Keep and Omit	196
8.1.4	Replace	198
8.1.5	Calc	200
8.1.6	Count	201
8.2	The Browser Shell	201
8.3	URLs as Commands	201
8.4	Self-Disclosure in the Command Bar	203
8.5	Command Input and Output	203
8.6	External Programs	205
8.7	The Browser Shell as a System Command Prompt	208
8.8	Page History vs. Command History	210
8.9	Generating Scripts from the Page History	211
8.10	Web Automation	211
8.11	Web Automation by Demonstration	213
8.12	Other Tcl Commands	214
8.13	Document Metadata	215
8.14	Command Line Invocation	217
9	Selection Inference	221
9.1	User Interface	221
9.1.1	Selection Guessing Mode	223
9.1.2	Simultaneous Editing Mode	226
9.2	Implementation	232
9.2.1	Selection Guessing Algorithm	233
9.2.2	Simultaneous Editing Algorithm	238
9.3	User Studies	243
9.3.1	Simultaneous Editing Study	243
9.3.2	Selection Guessing Study	249
10	Outlier Finding	253
10.1	Motivation	254
10.2	Outlier Highlighting	255
10.3	Unusual Matches Display	256
10.4	Implementation	260
10.4.1	Feature Generation	260
10.4.2	Feature Weighting	261

10.4.3	Feature Pruning	262
10.4.4	Ranking	262
10.5	User Study	263
10.5.1	Results	263
11	Conclusion	267
11.1	Discussion	267
11.2	Open Questions	269
11.3	Summary of Contributions	275
11.4	Looking Ahead	276
A	Pattern Library	279
A.1	Business	279
A.2	Characters	284
A.3	English	285
A.4	HTML	286
A.5	Internet	287
A.6	Java	288
A.7	Layout	289
A.8	Style	292
B	TC Pattern Operators	295
B.1	And	295
B.2	Anywhere After	295
B.3	Anywhere Before	295
B.4	Balanced from-to	295
B.5	Case Sensitive	296
B.6	Contains	296
B.7	Either-Or	296
B.8	End of	296
B.9	Ends	296
B.10	Equals	296
B.11	Flatten	297
B.12	From-To	297
B.13	Identifier	297
B.14	Ignoring	297
B.15	In	297
B.16	Is	297
B.17	Just After	298
B.18	Just Before	298
B.19	Literal	298
B.20	Melt	298
B.21	Nonzero	298
B.22	Not	298
B.23	Nth	299

B.24 Or	299
B.25 Overlaps	299
B.26 Overlaps End Of	299
B.27 Overlaps Start Of	300
B.28 Prefix	300
B.29 Regular Expression	300
B.30 Start of	300
B.31 Starts	300
B.32 Then	300
B.33 Trim	301
B.34 View	301

C LAPIS Commands 303

C.1 Back	303
C.2 Calc	303
C.3 Click	304
C.4 Count	304
C.5 Delete	304
C.6 Doc	304
C.7 Enter	305
C.8 Exec:	305
C.9 Extract	305
C.10 File:	305
C.11 Forward	306
C.12 Ftp:	306
C.13 History	306
C.14 Http:	306
C.15 Insert	306
C.16 Keep	306
C.17 Omit	307
C.18 Parse	307
C.19 Property	307
C.20 Relocate	308
C.21 Replace	308
C.22 Save	308
C.23 Show	308
C.24 Sort	309
C.25 Submit	309

List of Figures

1.1	An email message with some of its structure labeled.	1
1.2	A web page that lists electronic books available for downloading.	2
1.3	A web page from Yahoo.	2
1.4	Part of a Java program.	2
1.5	The LAPIS web browser/text editor, which demonstrates how lightweight structure can be used in a text-processing system.	6
1.6	Not all useful abstractions can be represented by a contiguous region. In this plain text file laid out in two columns, each <i>address</i> actually consists of several non-contiguous regions. The closest we can get in the lightweight structure model is <i>address-line</i>	8
3.1	Example region sets. (a) words; (b) styles; (c) expressions; (d) a selection made by a user.	24
3.2	Equivalent definitions of a region $[s, e]$: (a) s and e are offsets from the start of the string; (b) s and e denote positions between characters, and $[s, e]$ is a closed interval; (c) s and e denote characters, and $[s, e]$ is a half-open interval.	25
3.3	Fundamental region relations.	26
3.4	Region space. A region $[s, e]$ corresponds to the point (s, e) in region space. Only integer-valued points on or above the 45° line are valid regions.	27
3.5	Areas of region space representing region relations relative to a fixed region b . The shaded rectangles represent all regions a satisfying a before b , a overlaps-start b , and a contains b , respectively.	28
3.6	Region space map. “ $op\ b$ ” labels the set of regions a such that $a\ op\ b$	29
3.7	Degenerate region space maps, for regions b that lie on boundaries of region space.	30
3.8	When a -45° line is swept across an overlapping region set, it can intersect at most one region at every sweep position, because the rest of the line lies in the off-limits <i>in</i> and <i>contains</i> areas.	32
3.9	The result of <i>in Word</i> depicted in region space. The isolated points make up the <i>Word</i> region set. The shaded areas are <i>in Word</i>	34
3.10	Adjacency operators.	37
3.11	<i>touches</i> and <i>overlaps</i>	38
3.12	<i>min</i> and <i>max</i>	38
3.13	Region space areas for <i>less-than</i> and <i>greater-than</i>	39
3.14	<i>max-span</i> operator in region space.	40
3.15	<i>descendant-of</i> and <i>ancestor-of</i>	43
3.16	<i>separated-by D</i>	45

3.17	<i>split</i> and <i>split-nonzero</i>	46
3.18	<i>flatten</i> creates a flat set by combining overlapping regions into a single region. <i>melt</i> combines not only overlapping regions but also adjacent regions.	47
3.19	<i>just-before_W</i> and <i>just-after_W</i>	48
3.20	<i>equals_Wa</i> matches any region that can be obtained from <i>a</i> by adding or removing characters in <i>W</i>	49
4.1	Region rectangles depicted in a text string and in region space. Region rectangle <i>A</i> is the set of all regions that <i>contains</i> the word “score”, <i>B</i> is the set of regions that are <i>in</i> the word “seven”, and <i>C</i> consists of the single region “years”.	52
4.2	Region relations can be represented by rectangles.	53
4.3	Applying a relational operator to a region rectangle always produces another rectangle.	55
4.4	Rectangle collections for flat, overlapping, and nested region sets. Each rectangle covers only a single point. Dashed lines show that each region set meets the desired definition. Regions in <i>F</i> must be <i>before</i> or <i>after</i> each other. Regions in <i>O</i> may also <i>overlap-start</i> or <i>overlap-end</i> . Regions in <i>N</i> may be related by <i>before</i> , <i>after</i> , <i>in</i> , or <i>contains</i>	56
4.5	Rectangle collections for relational operators applied to the flat region set <i>F</i> from Figure 4.4. Each collection contains four rectangles, which may intersect. Dashed lines show the location of the original region set <i>F</i>	57
4.6	More relational operators applied to the flat region set <i>F</i> from Figure 4.4. Dashed lines show the location of the original region set <i>F</i>	58
4.7	An R-tree containing 5 rectangles, <i>A</i> – <i>E</i>	62
4.8	Different R-trees for the same set of rectangles can have different querying performance. A query for rectangle <i>Q</i> can avoid visiting the C-D subtree in the R-tree on the left, but it must visit all nodes in the R-tree on the right.	63
4.9	Deleting a rectangle from an RB-tree.	65
4.10	Example of a region quadtree (after Figure 1.1 from Samet [Sam90]).	66
4.11	A quadtree used as a rectangle collection.	66
4.12	Bad case for a quadtree: two long but closely-spaced rectangles force the quadtree to drill down to depth $O(\log n)$, requiring $O(n)$ quadtree nodes.	69
4.13	Syntax tree and RB-tree representations of a nested region set.	74
4.14	Heuristics for reducing overlap in a rectangle collection.	77
4.15	Rectangle collections after reducing overlap.	78
4.16	The plane-sweep intersection algorithm (after Preparata & Shamos [PS85], Figure 8.29).	85
4.17	The lexicographic ordering of regions has no simple relationship with lexicographic ordering of rectangles. Finding the <i>n</i> th region in a rectangle collection may require jumping back and forth between rectangles.	87
4.18	Hierarchy of region set types.	88
4.19	Relational operators applied to random region sets of size <i>N</i>	90
4.20	Intersection of random region sets of size <i>N</i>	90
4.21	Union of random region sets of size <i>N</i>	91
4.22	Difference of random region sets of size <i>N</i>	91

4.23	Performance of LAPIS library patterns on Java source files.	93
4.24	Performance of LAPIS library patterns on HTML web pages.	93
4.25	Performance of LAPIS library on <code>ls -l</code> outputs.	94
5.1	A finite state transducer that tests for the letter <i>a</i> and deletes the letter <i>b</i>	99
5.2	Finite state transducer M_0 maps a region language $L(E)$ to the corresponding inner language $L_0(E)$ by erasing symbols outside the square brackets and the square brackets themselves.	101
5.3	Finite state transducer M_1 maps a region language $L(E)$ to the corresponding outer language $L_1(E)$ by erasing just the square brackets.	101
5.4	Finite state transducers for the relational operators.	110
5.5	An ϵ -free FST for <i>in</i>	112
5.6	Summary of relationships among language classes.	119
6.1	The LAPIS pattern library shows the namespace hierarchy.	130
6.2	Regular expression operators supported by LAPIS.	137
6.3	Different views of a simple HTML page: (a) rendered; (b) source.	138
6.4	Coordinate map mapping from source view to rendered view for part of an HTML document.	139
6.5	TC pattern-matching operators with their equivalent region algebra operators from Chapter 3. Several of the relational operators can be either unary or binary, as indicated by the optional first operand <i>expr?</i>	140
6.6	Synonyms for TC operators.	141
6.7	Default background for different file types.	143
6.8	The LAPIS pattern editor automatically displays light blue <i>tie lines</i> to indicate which token is modified by an indented expression.	147
6.9	Comparison of token trees with syntax trees for several arithmetic expressions (parsed with conventional precedence, which gives multiplication higher precedence than addition).	150
6.10	Token trees for several TC expressions.	151
6.11	Rules for translating a token tree into a syntax tree. Light lines are token tree edges, and dark lines are syntax tree edges. Rules are applied to nodes in preorder, left-to-right, until all token tree edges have been transformed into syntax tree edges.	153
6.12	Excerpt from an email message announcing cheap airfares.	154
6.13	Excerpt from a web page describing user interface toolkits.	155
6.14	A Java method with a documentation comment.	157
7.1	Screenshot of a LAPIS window showing its major components.	160
7.2	Different region sets in the same paragraph, highlighted in LAPIS.	163
7.3	Gradient highlights displayed by LAPIS for common unary relational operators. (a) <i>contains</i> fades out in both directions away from the center; (b) <i>in</i> fades out inward to the center; (c) <i>just before</i> fades out to the left; (d) <i>just after</i> fades out to the right; (e) <i>starts</i> fades out to the right; (e) <i>ends</i> fades out to the left; and (f) <i>equals</i> has no gradient at all.	165
7.4	LAPIS highlighting cannot distinguish between nested and overlapping region sets.	165

7.5	Techniques for distinguishing between nested and overlapping highlights.	166
7.6	The scrollbar is augmented with marks to help the user find selections in a long document. The marks can distinguish between different region sets, such as (a) and (b).	167
7.7	The library pane shows the patterns in the pattern library.	169
7.8	The pattern pane allows the user to enter and run TC patterns.	171
7.9	Adaptive tab stops.	171
7.10	Automatic subexpression reindentation preserves expression structure during pattern editing.	172
7.11	Tutorial sections.	175
7.12	Example of a generation task. All generation tasks used the same data (a table of CMU computer science courses), but the instructions at the top and the selected region set varied.	177
7.13	Generation tasks.	178
7.14	Comprehension tasks.	178
7.15	Example of the first kind of comprehension task. All but tasks C7 and C8 used this web page.	179
7.16	Example of the second kind of comprehension task. Tasks C7 and C8 used this web page.	179
7.17	Synonyms for TC operators suggested by users.	182
8.1	The Tools menu lists the text-processing tools built into LAPIS.	188
8.2	Examples of the output produced by Extract for various selections.	189
8.3	Using copy and paste to extract a selection. (a) The selection (word) is copied to the clipboard. (b) After making a new document with File/New, the clipboard is pasted, leaving an insertion point after each pasted region. (c) The user deletes the linebreaks that were inserted by default, and types) , (instead. (d) The user fixes up the boundary cases, before the first and after the last extracted region.	192
8.4	The Sort dialog.	193
8.5	Examples of using Sort on various tasks.	194
8.6	Sorting the words in a document produces gibberish. (Note that the big purple “LAPIS” was not sorted with the other words because it is an image, not text.) . . .	195
8.7	Images can be deleted from a web page by selecting them (with the Image pattern) and applying the Omit command.	196
8.8	Keep and Omit are complementary. Starting with the selection shown in (a), the Keep command keeps the selected books and deletes the rest (b), while the Omit command deletes the selected books and keeps the rest (c). Book is defined as Row anywhere after "Description".	197
8.9	The Replace dialog.	198
8.10	Replacing selections with a template. Every region in the selection on the left (made by the pattern Paragraph) is replaced by a template with two slots — one for the person’s name (Name, defined as first Line in Paragraph) and the other for the person’s phone number (Phone, defined as from end of "Phone:" to start of Linebreak). When a template pattern fails to match, e.g., when there isn’t a phone number, that slot in the template is left blank.	199

8.11	Commands disclosed for earlier examples in this chapter.	204
8.12	An external program's standard output and standard error streams are displayed in separate panes. (a) a successful command that prints only to standard output; (b) an unsuccessful command that prints an error message to standard error.	207
8.13	The Konqueror web browser/file manager [KDE02] can include a terminal pane (at the bottom of the window) for executing commands.	209
9.1	The selection mode can be changed with either the Selection menu or toolbar buttons.	222
9.2	Making a selection by example in selection guessing mode.	224
9.3	Selection guessing mode also offers a list of alternative hypotheses.	225
9.4	First step of simultaneous editing: selecting the records using unconstrained inference (selection guessing).	227
9.5	Second step of simultaneous editing: the records turn yellow, and inference is now constrained to make one selection in each record.	227
9.6	Selecting "public" in one record causes the system to infer equivalent selections in the other records.	228
9.7	Typing and deleting characters affects all regions.	228
9.8	The user selects the types and copies them to the clipboard...	229
9.9	... and then pastes the copied selections back.	229
9.10	Changing the capitalization of the method name.	230
9.11	The final outcome of simultaneous editing.	230
9.12	Commands on the Change Case menu change the capitalization of the selection. . .	231
9.13	Block diagram of selection-guessing algorithm.	233
9.14	Finding common substrings using a suffix tree. (a) Suffix tree constructed from first example, <code>rcm@cmu.edu</code> ; \$ represents a unique end-of-string character. (b) Suffix tree after matching against second example, <code>ljc@cmu.edu</code> . Each node is labeled by its visit count. (c) Suffix tree after pruning nodes which are not found in <code>ljc@cmu.edu</code> . The remaining nodes represent the common substrings of the two examples.	235
9.15	Block diagram of simultaneous editing algorithm.	239
9.16	Coordinate map translating offsets between two versions of a document. The old version is the word <code>trample</code> . Two regions are deleted to get the new version, <code>tame</code>	242
9.17	The Simultaneous Editing dialog box used in the user study.	244
9.18	Tutorial task 1: change the numbering of a list.	245
9.19	Tutorial task 2: reformat email message headers into a short list.	246
9.20	Task 1: put author name and year in front of each citation in a bibliography [Fuj98].	247
9.21	Task 2: reformat a list of mail aliases from HTML to plain text [Fuj98].	248
9.22	Task 3: reformat a list of baseball scores into a tagged format [Nix85].	248
9.23	User responses to questions about simultaneous editing.	250
9.24	The Selection Guessing dialog box used in the user study.	250
9.25	User responses to questions about selection guessing.	251

10.1	Incorrect inference of the last two digits of the publication year. The selection is visibly wrong, and all users noticed and corrected it.	254
10.2	Incorrect inference of the author’s name. “Hayes-Roth” is only partially selected, but no users noticed.	254
10.3	Incorrect inference with outlier highlighting drawing attention to the possible error.	256
10.4	The Unusual Matches window showing occurrences of “copy” in a document. The most prominent outlier, which is selected, is found in an italicized word, “rcopy”. .	257
10.5	The Unusual Matches window showing both matches and mismatches to the pattern <code>Line starting "From:"</code> in a collection of email message headers. The most prominent mismatch, which is selected, is a Sender line which appears where the From line would normally appear in the message.	259

List of Tables

4.1	Rectangle-collection data structures. N is the number of rectangles in the collection, F is the number of rectangles found by a query or affected by a deletion, and n is the length of the string. Boldface indicates that the data structure is implemented in LAPIS.	87
4.2	Unary relational operators, such as <i>in</i> , <i>contains</i> , <i>just-before</i> , <i>just-after</i> , etc.	88
4.3	Set intersection.	88
4.4	Set union.	89
4.5	Set difference.	89
4.6	Pattern matching rate for the benchmark documents (mean \pm standard deviation). <i>All patterns in library</i> is the rate at which all TC patterns in the library are matched. <i>Per pattern</i> is the average rate for each of the 82 patterns. <i>Per operator</i> is the average rate for each of the 478 operators in the patterns.	95
9.1	Time taken by users to perform each task (mean [min–max]). <i>Simultaneous editing</i> is the time to do the entire task with simultaneous editing. <i>Manual editing</i> is the time to edit 3 records of the task by hand, divided by 3 to get a per-record estimate.	246
9.2	Equivalent task sizes for each task (mean [min–max]). <i>Novices</i> are users in the user study. <i>Expert</i> is my own performance, provided for comparison. <i>DEED</i> is another PBD system, tested on tasks 1 and 2 by its author [Fuj98].	246
9.3	Time taken by users to perform the task (mean [min–max]). <i>Selection guessing</i> is the time to do the entire task with selection guessing. <i>Manual editing</i> is the time to edit 3 records of the task by hand, divided by 3 to get a per-record estimate.	251
9.4	Equivalent task sizes (mean [min–max]) for task 2, comparing selection guessing, simultaneous editing, and DEED. <i>Novices</i> are users in a user study. <i>Expert</i> is my own performance, provided for comparison.	251
10.1	Fraction of incorrectly-inferred selections that were noticed and corrected by users (number corrected / number total). The denominators vary because some users never made the selection and others made it more than once.	264
10.2	Fraction of tasks completed with errors in final result (number of tasks in error / number total).	264
10.3	Equivalent task sizes (mean [min–max]) for each task in each condition.	265

List of Algorithms

3.1	FROMTO returns a flat region set by spanning from each region in L to the closest following region in R	42
3.2	BALANCE returns a nested region set formed by matching opening delimiters from L with closing delimiters from R	42
4.1	RELATION applies a relational operator to a rectangle collection.	59
4.2	UNION finds the union of two rectangle collections.	59
4.3	INTERSECTION finds the intersection of two rectangle collections.	60
4.4	DIFFERENCE finds the difference between two rectangle collections.	60
4.5	FORALL applies a function f to every rectangle in collection C	60
4.6	QUERY (T, r) traverses an R-tree T to find all rectangles that intersect the rectangle r	62
4.7	INSERT (T, r) inserts a rectangle r into a quadtree T	67
4.8	SPLIT (T) converts a quadtree leaf into a node with children.	67
4.9	MERGE (T) converts a quadtree node into a leaf if its children are all leaves and the union of their rectangles is rectangular.	68
4.10	QUERY (A, i, j, r) searches a segment of a region array, $A[i\dots j]$, for all rectangles that intersect the rectangle r	72
4.11	QUERY (A, r) searches a region array for all rectangles that intersect the rectangle r , using a binary search on each coordinate.	73
4.12	QUERY (T, r) searches a syntax tree for all regions that intersect the rectangle r	75
4.13	LEXORDER (T) generates the regions from a syntax tree in lexicographic order. The algorithm is basically a preorder traversal, except that all descendants of T that start at the same point as T must be returned before T itself.	75
4.14	INTERSECTION(C_1, C_2) intersects two point collections by traversing them in lexicographic order.	80
4.15	TANDEMINTERSECTION finds the intersection of two tree-shaped rectangle collections by traversing both trees at the same time.	82
4.16	QUADTREEINTERSECT finds the intersection of two quadtrees by tandem traversal.	84
6.1	PARSEINDENTATION constructs the token tree for a TC expression from its indentation.	152

Acknowledgements

This thesis is dedicated to:

- my dear wife Laura, who puts up with a lot;
- my parents Larry and Marian, and my brothers Craig and Eric, for their love and support, and for only rarely asking when I was going to get a real job;
- Dick Kollin and Dave Fraser, whose shameless exploitation of teenage labor gave me my start in this crazy business;
- Charles Leiserson, who encouraged me to go on to grad school at CMU and predicted I'd come back someday;
- Brad Myers, who pointed me in the right direction, gave me the ball, and let me run with it;
- my other committee members David Garlan, Jim Morris, and Brian Kernighan;
- my fellow grad students James Landay, Rich McDaniel, John Pane, Jeff Nichols, and Jake Wobbrock;
- Sharon Burks and Ava Cruse, who made impossible things seem easy;
- Yuzo Fujishima, who kindly provided materials for my user studies;
- the wonderful people with whom I have shared offices (and laughter) over the years at CMU, including George Nacula, Fay Chang, José Brustoloni, Adam Berger, Chris Colohan, Hakan Younes, Umair Shah, Bartosz Przydatek, and Sanjit Seshia;
- everyone who gave valuable feedback about LAPIS, including Rich Clingman, Monty Zukowski, Julián Jesús Martínez López, John Padula, Ben Bostwick, Franklin Chen, Chris Long, and John Gersh.

