Appendix A

Pattern Library

This appendix lists the pattern identifiers built into the LAPIS library, describing what each identifier is designed to match. When an identifier is defined by TC patterns, the TC patterns are specified afterward.

A.1 Business

Identifiers in the `Business` namespace are defined by TC patterns in the file `USEnglish.tc`.

Address

`State` matches U.S. state names and their two-letter abbreviations.

`ZipCode` matches 5-digit and 9-digit U.S. zip codes appearing just after a `State` in an address.

```
State is either "Alabama" or "Alaska"
                ...
             or "Wisconsin" or "Wyoming"
             or word = case-sensitive
                          either "AL" or "AK"
                                 ...
                              or "WI" or "WY"
                    ignoring nothing

@ZipCode is Number equal to /\d\d\d\d\d/
                  ignoring nothing
               just after State
ZipCode is flatten either @ZipCode
                   or @ZipCode
                       then "-"
                       then Digits equal to /\d\d\d\d/
                       ignoring nothing
```

Date

`DayOfMonth` matches a numeric day of the month, 1-31, appearing in a date, with optional “th”, “nd”, or “st” suffix.

`DayOfWeek` matches the English name of a weekday or its three-letter abbreviation.

`LongMonth` matches the English name of a month or its three-letter abbreviation.

`ShortMonth` matches a numeric month, 1-12, appearing in a date.

`Month` matches either `LongMonth` or `ShortMonth`

`LongYear` matches a four-digit year from either the 20th or 21st centuries

`ShortYear` matches a two-digit year appearing in a date.

`Year` matches either `LongYear` or `ShortYear`

`Date` matches a date, which contains at least a month and a year, and optionally a day of the month, day of the week, and a time.

```
    @DayOfMonth is Number equal to /[12][0-9]|3[01]|0?[1-9]/
                           ignoring nothing
    @DayOfMonth is either @DayOfMonth
                        or @DayOfMonth
                             then "th"
                             ignoring nothing
                        or @DayOfMonth
                             then "nd"
                             ignoring nothing
                        or @DayOfMonth
                             then "st"
                             ignoring nothing

    @DayOfWeek is Word equal to either "Sun"
                                   or "Sunday"
                                   or "Mon"
                                   or "Monday"
                                   or "Tue"
                                   or "Tues"
                                   or "Tuesday"
                                   or "Wed"
                                   or "Wednesday"
                                   or "Thu"
                                   or "Thurs"
                                   or "Thursday"
                                   or "Fri"
```

```
                                    or "Friday"
                                    or "Sat"
                                    or "Saturday"
                          ignoring nothing

    @LongMonth is Word equal to either "Jan"
                                    or "January"
                                    or "Feb"
                                    or "February"
                                    ...
                                    or "Dec"
                                    or "December"
                               ignoring nothing

    @ShortMonth is Number equal to /1[012]|0?[1-9]/
                               ignoring nothing

    @Month is either LongMonth or @ShortMonth

    @LongYear is Number equal to /(19|20)\d\d/
                               ignoring nothing

    @ShortYear is Number equal to /\d\d/
                                ignoring nothing

    @Year is either LongYear or @ShortYear

    Date is flatten either
                            either @LongMonth
                                    then @DayOfMonth
                                    then @LongYear
                            or @LongYear
                                then @LongMonth
                                then @DayOfMonth
                            or @DayOfMonth
                                then @LongMonth
                                then @LongYear
                            ignoring either Spaces
                                         or Punctuation

                       or either @LongMonth then @DayOfMonth
                            or @LongMonth then @LongYear
                            ignoring either Spaces
                                         or Punctuation
```

```
                    or either @Month
                              then @DayOfMonth
                              then @Year
                           or @Year
                              then @Month
                              then @DayOfMonth
                           or @DayOfMonth
                              then @Month
                              then @Year
                          ignoring /[-\/]/

                   or @DayOfWeek
                         then @LongMonth
                         then @DayOfMonth
                         then Time
                         then Word
                         then @LongYear
                         ignoring Spaces

    DayOfMonth is @DayOfMonth in Date

    DayOfWeek is @DayOfWeek

    LongMonth is @LongMonth

    ShortMonth is @ShortMonth in Date
                               not in Time
    Month is either LongMonth or ShortMonth

    LongYear is @LongYear

    ShortYear is ShortYear in Date
                           not in Time

    Year is either LongYear or ShortYear
```

Money

`Money` matches a number preceded by a dollar sign

```
    Money is "$" then Number
                 ignoring nothing
```

Number

`Number` matches a number with optional comma separators and decimal point. Comma separators are recognized only up to 999,999,999. A better definition of `Number` would use a regular expression, as `ScientificNotation` does.

`ScientificNotation` matches a number with optional exponent (E+/-).

```
@Number is either Digits
               or Digits
                    then ","
                    then Digits
                    ignoring nothing
               or Digits
                    then ","
                    then Digits
                    then ","
                    then Digits
                    ignoring nothing
@Number is either @Number
                or @Number
                     then "." then Digits
                     ignoring nothing
Number is either @Number
               or "-" not just after Word
                   then @Number
                   ignoring nothing
ScientificNotation is /-?\d+(\.\d+)?(\s*[Ee][-+]?\d+)?/
```

PhoneNumber

`PhoneNumber` matches a 7-digit or 10-digit US phone number separated by dashes.

`AreaCode` matches the 3-digit area code that starts a 10-digit phone number.

`FaxNumber` matches a phone number labeled by the word “fax” before or after it.

```
@PhoneNumber is Number equal to /\d\d\d/
                then "-"
                then Number equal to /\d\d\d\d/
                ignoring nothing
AreaCode is Number equal to /\d\d\d/
                   ignoring nothing
                   just before @PhoneNumber
```

```
PhoneNumber is either @PhoneNumber
                    or AreaCode then "-"
                                 then @PhoneNumber
                    or AreaCode then @PhoneNumber
                    or "(" then AreaCode
                           then ")" then @PhoneNumber
FaxNumber is PhoneNumber
                    either just after "fax"
                        or just before "fax"
                    ignoring either Punctuation
                                 or Spaces
```

Time

`Time` matches a 12-hour or 24-hour time specification, with optional seconds and optional AM or PM.

```
Time is Number equal to /[012]?\d/
         then ":"
         then Number equal to /\d\d/
         ignoring nothing
Time is either Time
            or Time
                  then ":"
                  then Number equal to /\d\d/
                  ignoring nothing
Time is either Time
            or Time then Word equal to either "am"
                                          or "pm"
                                ignoring nothing
Time is either Time
            or "midnight"
            or "noon"
```

A.2 Characters

Identifiers in the `Characters` namespace are defined by the Java class `lapis.parsers.CharacterParser`.

`Token` matches runs of non-whitespace characters.

`Alphanumeric` matches runs of alphanumeric characters (letters or digits).

`Digits` matches runs of digits.

`Letters` matches runs of letters.

`LowerCaseLetters` matches runs of lowercase letters.

`UpperCaseLetters` matches runs of uppercase letters.

`Punctuation` matches runs of punctuation.

`Whitespace` matches runs of whitespace characters.

`Linebreak` matches individual linebreak characters.

`Spaces` matches runs of space characters.

`Tab` matches individual tab characters.

A.3 English

Identifiers in the `English` namespace are defined by TC patterns in the file `USEnglish.tc`.

Sentence

`Sentence` matches English "sentences" that start with a capitalized word and end with a period, exclamation point, or question mark.

```
@EndingPunctuation
   is either "." not just after case-sensitive
                                    either "Dr"
                                        or "Mr"
                                        or "Mrs"
           or "?"
           or "!"
       either just before Whitespace
                           then UppercaseLetters
            or ending Paragraph
@StartingWord
   is CapitalizedWord
         either just after @EndingPunctuation
             or starting Paragraph

Sentence is from @StartingWord
            to @EndingPunctuation
              in Paragraph
              contains Spaces
```

Word

`Word` matches runs of alphanumeric characters.

`AllCapsWord` matches a word which consists entirely of uppercase letters.

`CapitalizedWord` matches a word starting with uppercase letters.

`LowerCaseWord` matches a word consisting entirely of lowercase letters.

`MixedCaseWord` matches a word with both uppercase and lowercase letters, where at least one uppercase letter occurs strictly inside the word.

```
Word is Alphanumeric
CapitalizedWord is Word starting UppercaseLetters
                           ignoring nothing
AllCapsWord is Word equal to UppercaseLetters
                      ignoring nothing
LowerCaseWord is Word equal to LowercaseLetters
                        ignoring nothing
MixedCaseWord is Word
                    contains LowercaseLetters
                              then UppercaseLetters
                              ignoring nothing
```

A.4 HTML

Identifiers in the `HTML` namespace are defined by the Java class `lapis.parser.HTMLParser`.

`Attribute` matches a name-value attribute in an HTML tag.

`name``-attr` matches an attribute named *name*. For example, `href-attr` matches the `href` attribute in an `<a>` tag. An identifier of this form is defined for every attribute named in the HTML 4.0 specification.

`AttributeName` matches the name part of an attribute (the part before the equals sign, or the entire attribute if no value is given).

`AttributeValue` matches the value part of an attribute (the part after the equals sign).

`Element` matches a complete element, running from its start tag to its matching end tag (if any).

`[`*`tagname`*`]` matches a complete element named *tagname*. For example, `[body]` matches the part of the document running from `<body>` to `</body>`. An identifier of this form is defined for every tag in HTML 4.0.

`Tag` matches any tag, which may be either a start tag or an end tag.

`StartTag` matches any start tag.

`<tagname>` matches a start tag named *tagname*. An identifier of this form is defined for every tag in HTML 4.0.

`EndTag` matches any end tag.

`</tagname>` matches an end tag named *tagname*. An identifier of this form is defined for every tag in HTML 4.0.

`Text` matches a run of text between tags.

A.5 Internet

Identifiers in the `Internet` namespace are defined by TC patterns found in the file `Internet.tc`.

`EmailAddress` matches an email address in conventional `user@hostname` form.

`Hostname` matches either an IP address or a domain name.

`IPAddress` matches an IP address in n.n.n.n form.

`RootDomain` matches the most common root domain names, such as `edu`, `com`, and `org`.

`URL` matches a Uniform Resource Locator.

```
EmailAddress is Token containing "@"
                    trim off Punctuation
IPAddress is Digits then "."
                     then Digits
                     then "."
                     then Digits
                     then "."
                     then Digits
                      ignoring nothing

RootDomain is Word equal to either "com"
                                  or "edu"
                                  or "gov"
                                  or "mil"
                                  or "org"
                                  ...
                                  or "it"
                                  or "fi"
                       just after "."
                             ignoring nothing
```

```
Hostname is either IPAddress
                or /[\w\-\.]+/
                     ending "." then RootDomain
 URL is from case-sensitive either "http:"
                                   or "ftp:"
                                   or "mailto:"
                                   or "file:"
                                   or "gopher:"
                                   or "news:"
                                   or "nntp:"
                                   or "https:"
                                   or "telnet:"
                                   or "wais:"
                                   or "prospero:"
                                   or "javascript:"
        to point just before either Whitespace
                                    or '"'
                                    or "'"
                                    or ">"
```

A.6 Java

Some of the identifiers in the `Java` namespace are defined by the class `lapis.parsers.JavaParser`. Others are defined by TC patterns in the file `Java.tc`.

`ActualParameter` matches an expression passed as a parameter to a method call.

`ActualParameterList` matches the parenthesized list of parameters to a method call.

`Block` matches a block of statements surrounded by curly braces.

`Class` matches a Java class declaration, including its body.

`Comment` matches a Java comment, both `//` style and `/*..*/` style.

`Constant` matches a constant expression, such as a number, character or string literal, or the identifier `null`.

`Expression` matches an expression.

`Field` matches a variable declaration in a class.

`FormalParameter` matches the declaration of a method parameter.

`FormalParameterList` matches the parenthesized list of parameter declarations in a method.

`Identifier` matches a user-defined identifier, such as a variable name, method name, or class name.

`Import` matches an `import` statement.

`Interface` matches a Java interface declaration, including its body.

`LocalVariable` matches a local variable declaration inside a method.

`Method` matches a method declaration, including its body.

`MethodBody` matches the body of a method, surrounded by curly braces.

`MethodCall` matches an expression that calls a method.

```
MethodName is Identifier just before FormalParameterList
MethodBody is Block ending Method
                      ignore nothing
MethodCall is Identifier then ActualParameterList
```

`Statement` matches a statement.

`Type` matches a type, such as a class name or primitive type.

`VariableName` matches the name of the variable in a variable declaration.

```
VariableName is Identifier in either field
                                or localvariable
                                or formalparameter
                            not in type
                            not in expression
```

A.7 Layout

Identifiers in the `Layout` namespace are defined by TC patterns, some in the file `Layout.tc`, and others in the file `HTML.tc`.

Delimiters

`CurlyBraces` matches a balanced set of curly braces, {...}.

`Parentheses` matches a balanced set of parentheses, (...).

`SquareBrackets` matches a balanced set of square brackets, [...].

```
Parentheses is balances "(" with ")"
SquareBrackets is balances "[" with "]"
CurlyBraces is balances "{" with "}"
```

Form

`Form` matches an HTML form.

`Control` matches an HTML form control, such as a button or text field.

`Button` matches an HTML form button.

`Checkbox` matches an HTML checkbox.

`Menu` matches an HTML drop-down menu.

`RadioButton` matches an HTML radio button.

`Textbox` matches an HTML text field.

```
Form is [form]
Control is either [input] or [textarea] or [select]
view source
  Textbox
   is either [input]
               contains type-attr
                          contains either "text"
                                       or "password"
          or [input] not containing type-attr
          or [textarea]
  Button
    is [input]
         contains type-attr
                    contains either "button"
                                 or "submit"
                                 or "reset"
                                 or "image"
  Menu is [select]
  Checkbox
      is [input]
            contains type-attr contains "checkbox"
  RadioButton
      is [input] contains type-attr contains "radio"
```

Image

`Image` matches an image in a web page

```
Image is [img]
```

Line

`Line` matches a line in a plain text file.

`BlankLine` matches a line with nothing but whitespace characters in it.

`Break` is a synonym for `Linebreak`.

```
Line is nonempty from either start of page
                           or end of linebreak
                    to either linebreak
                           or end of page
Break is Linebreak
BlankLine is Line not containing Token
```

List

`List` matches an HTML list.

`Item` matches a single item in a list.

`BulletList` matches a bulleted list.

`Bullet` matches a single item in a bulleted list.

`NumberedList` matches a numbered list.

`NumberedItem` matches a single item in a numbered list.

`DefinitionList` matches an HTML definition list.

`Term` matches a term in a definition list.

`Definition` matches a definition in a definition list.

```
List is either [ul]
            or [ol]
            or [dl]
Item is either [li]
            or [dt] then [dd]
BulletList is [ul]
Bullet is [li] in BulletList
NumberedList is [ol]
NumberedItem is [li] in NumberedList
DefinitionList is [dl]
Term is [dt]
Definition is [dd]
```

Page

`Page` matches the entire page or file.

```
Page is all
```

Paragraph

`Paragraph` matches (in plain text) a group of lines separated by blank lines, or (in HTML) a block-level element, such as `[p]` or `[li]`. Paragraph is defined by the Java class `lapis.parsers.SystemParser`.

Rule

`Rule` matches a horizontal rule (`<hr>` element in HTML).

```
Rule is [hr]
```

Table

`Table` matches an HTML table.

`Row` matches a row in a table.

`Cell` matches a cell in a row.

```
Table is [table]
Row is [tr]
Cell is either [td] or [th]
```

A.8 Style

Identifiers in the `Style` namespace are defined by TC patterns found in the file `HTML.tc`.

`Bold` matches text in boldface.

`Heading` matches a heading.

`Heading`*`N`* matches headings of size *`N`*, where *`N`* can range from 1-7.

`Italic` matches text in italics.

`Italics` is a synonym for `Italic`.

`Link` matches a hyperlink.

`Target` matches a hyperlink target (an `<a name=>` element in HTML).

`Underlined` matches underlined text.

```
Bold is either [b]
            or [strong]
Italic is either [i]
              or [em]
Italics is Italic
Underlined is [u]

Link is [a] starts <a> contains href-attr
Target is [a] starts <a> contains name-attr

Heading is either [h1]
               or [h2]
               or [h3]
               or [h4]
               or [h5]
               or [h6]
               or [h7]

Heading1 is [h1]
Heading2 is [h2]
Heading3 is [h3]
Heading4 is [h4]
Heading5 is [h5]
Heading6 is [h6]
Heading7 is [h7]
```

Appendix B

TC Pattern Operators

This appendix presents an alphabetical list of operators in the TC pattern language. For more details about the syntax and use of the pattern language, see Chapter 6.

B.1 And

```
expr1 and expr2
```

Matches regions that match both expressions. Equivalent to algebra operator ∩ (Section 3.5.1). See also Section 6.2.15.

B.2 Anywhere After

```
anywhere after expr
```

Matches regions anywhere after some match to *expr*. Equivalent to algebra operator *after* (Section 3.5.2).

B.3 Anywhere Before

```
anywhere before expr
```

Matches regions anywhere before some match to *expr*. Equivalent to algebra operator *before* (Section 3.5.2).

B.4 Balanced from-to

```
balanced from expr1 to expr2
```

Matches a nested set of regions whose start delimiters match *expr1* and end delimiters match *expr2*. Equivalent to algebra operator *balances* (Section 3.6.8).

B.5 Case Sensitive

`case sensitive` *expr*

Forces literal and regular expression matches inside *expr* to be case-sensitive.

`not case sensitive` *expr*

Forces literal and regular expression matches inside *expr* to be case-insensitive (the default).

See also Section 6.2.8.

B.6 Contains

`contains` *expr*

Matches regions that contain at least one match to *expr*. Equivalent to algebra operator *contains* (Section 3.5.2).

B.7 Either-Or

`either` *expr1* `or` *expr2*

Matches regions that match either *expr1* or *expr2*. The `either` is optional. Equivalent to algebra operator $\cup$ (Section 6.2.15).

See also Section 6.2.15.

B.8 End of

`end of` *expr*

Matches the end points of regions matching *expr*. Always returns a set of zero-length regions. Equivalent to algebra operator *end-of* (Section 3.6.1).

B.9 Ends

`ends` *expr*

Matches regions that end at the same point as *expr*. Ignores background regions around the end point. Equivalent to algebra operator $ends_W$ (Section 3.6.13).

B.10 Equals

`equals` *expr*

Matches regions that match *expr*. Ignores background regions around both start point and end point. Equivalent to algebra operator $equals_W$ (Section 3.6.13).

B.11 Flatten

`flatten` *`expr`*

Flattens the regions that match *`expr`*, by combining nested and overlapping regions into a single region. Equivalent to algebra operator *flatten* (Section 3.6.12).

B.12 From-To

`from` *`expr1`* `to` *`expr2`*

Matches a flat region set consisting of regions that start with a match to *`expr1`* and end with the next match to *`expr2`*. Equivalent to algebra operator *fromto* (Section 3.6.8).

B.13 Identifier

`identifier`

Matches the regions matched by the pattern bound to *`identifier`* in the pattern library.

See also Section 6.2.3.

B.14 Ignoring

`expr1` `ignoring` *`expr2`*

Sets the background set to the regions matching *`expr2`*, then evaluates *`expr1`* and returns its matches.

See also Section 6.2.14.

B.15 In

`in` *`expr`*

Matches regions lie in some match to *`expr`*. Equivalent to algebra operator *in* (Section 3.5.2).

B.16 Is

`identifier` `is` *`expr`*

Assigns the pattern *`expr`* to *`identifier`* in the pattern library, and returns the matches to *`expr`*.

See also Section 6.2.5.

B.17 Just After

```
just after expr
```

Matches regions that lie after and adjacent to some match to *expr*. Ignores background regions when testing for adjacency. Equivalent to algebra operator *just-after*$_W$ (Section 3.6.13).

B.18 Just Before

```
just before expr
```

Matches regions that lie before and adjacent to some match to *expr*. Ignores background regions when testing for adjacency. Equivalent to algebra operator *just-before*$_W$ (Section 3.6.13).

B.19 Literal

```
"string"
'string'
```

Matches regions consisting of the literal characters *string*. Either single or double quotes may be used to delimit the string.

See also Section 6.2.8.

B.20 Melt

```
melt expr
```

Melts the regions that match *expr* by combining nested, overlapping, or adjacent regions into a single region. Equivalent to algebra operator *melt* (Section 3.6.12).

B.21 Nonzero

```
nonzero expr
```

Matches regions that match *expr* and contain at least one character. Equivalent to algebra operator *nonzero* (Section 3.6.4).

B.22 Not

```
expr1 not expr2
```

Matches regions matching *expr1* that do not match *expr2*. Equivalent to algebra operator $-$ (Section 6.2.15).

See also Section 6.2.15.

B.23 Nth

```
nth expr
nth expr1 in expr2
nth expr1 before expr2
nth expr1 after expr2
```

The first form matches the nth region in the document that matches *expr*. The other forms match the nth match to *expr1* that lies in, before, or after each match to *expr2*.

The "nth" can be written in a variety of ways:

- 1st, 2nd, 3rd, 4th, ...
- first, second, third, ..., tenth
- last, 2nd from last, 3rd from last, ...
- second from last, third from last, ...

Equivalent to algebra operator nth_n (Section 3.6.5).

B.24 Or

```
expr1 or expr2
```

Matches regions that match either *expr1* or *expr2*. Equivalent to algebra operator $\cup$ (Section 6.2.15).

See also Section 6.2.15.

B.25 Overlaps

```
overlaps expr
```

Matches regions that overlap some region matching *expr*. Equivalent to algebra operator *overlaps* (Section 3.6.3).

B.26 Overlaps End Of

```
overlaps end of expr
```

Matches regions that overlap the end point of some region matching *expr*. Equivalent to algebra operator *overlaps-end* (Section 3.5.2).

B.27 Overlaps Start Of

```
overlaps start of expr
```

Matches regions that overlap the start point of some region matching *expr*. Equivalent to algebra operator *overlaps-start* (Section 3.5.2).

B.28 Prefix

```
prefix identifier expr
```

Changes the current namespace to *identifier* for the scope of *expr*.

See also 6.2.7.

B.29 Regular Expression

```
/regexp/
```

Matches regions that match the regular expression *regexp*.

See Section 6.2.10 for the regular expression operators supported by LAPIS.

B.30 Start of

```
start of expr
```

Matches the start points of regions matching *expr*. Equivalent to algebra operator *start-of* (Section 3.6.1).

B.31 Starts

```
starts expr
```

Matches regions that start at the same point as *expr*. Ignores background regions around the start point. Equivalent to algebra operator $starts_W$ (Section 3.6.13).

B.32 Then

```
expr1 then expr2
```

Matches regions that are the concatenation of a region matching *expr1* with a region matching *expr2* that lies after and adjacent to it. Ignores background regions when determining whether *expr1* and *expr2* are adjacent. Equivalent to algebra operator $then_W$ (Section 3.6.13).

B.33 Trim

```
expr1 trim expr2
```

Matches regions that match *expr1* with an overlapping match to *expr2* removed from the start or end point. Equivalent to algebra operator *trim* (Section 3.6.14).

B.34 View

```
view source expr
view rendered expr
```

Forces literals and regular expressions inside *expr* to be matched against the HTML source or the rendered view of a web page. Has no effect on a plain text document.

See also Section 6.2.11.

Appendix C

LAPIS Commands

This appendix lists the script commands recognized by LAPIS, in alphabetical order. Standard Tcl commands are not included in this appendix; only new commands defined by LAPIS. For more information about LAPIS scripting, see Chapter 8.

C.1 Back

```
back [n]
```

Backs up to the previous document in the page history. The optional argument `n` is the number of pages to back up. This command has the same effect as the Back toolbar button.

See also: Section 8.8.

C.2 Calc

```
calc pattern
     [-count]
     [-sum]
     [-average|-mean|-avg]
     [-min]
     [-max]
     [-stddev]
```

Calculates statistics on the regions matching *`pattern`*. Only numeric regions are included in the statistics; nonnumeric regions are ignored. The statistics returned depend on which options are given:

- `-count` returns the number of numeric regions matching the pattern
- `-sum` returns the sum
- `-average` returns the mean of the regions. `-mean` and `-avg` are synonyms.

- `-min` returns the minimum of the matching regions
- `-max` returns the maximum
- `-stddev` returns the standard deviation

If only one option is given, then `calc` returns only the computed value. If multiple options are given, then `calc` returns a Tcl list of values in the order the options were given. If no options are given, `calc` computes all the statistics and returns a formatted display.

See also: Section 8.1.5.

C.3 Click

`click` *`pattern`*

Clicks on the hyperlink or form control described by *`pattern`*. Throws a Tcl exception if *`pattern`* does not match exactly one hyperlink or form control.

See also: Section 8.10.

C.4 Count

`count` *`pattern`*

Returns the number of matches to *`pattern`*.

C.5 Delete

`delete` *`pattern`*

Deletes all regions matching *`pattern`*. Synonym for `omit`.

See also: Section 8.1.3.

C.6 Doc

`doc [`*`string`*`]`

Returns the current document

`doc` *`string`* `[-type` *`type`*`]`

Sets the current document to *`string`*. With `-type`, the document is created with content type *`type`*. Possible content types are `text` and `html`. Without this argument, the content type is guessed from the content of *`string`*, defaulting to `text` if no valid HTML tags are found.

See also: Section 8.12.

C.7 Enter

```
enter pattern value
```

Sets all form fields matching *pattern* to the value *value*.

- For text fields, *value* is a string which is entered in the field.
- For menus and lists, *value* is the name of the selected value.
- For radio buttons and checkboxes, *value* should be one of the following: `on`, `off`, `yes`, `no`, `true`, `false`, `0`, `1`.

See also: Section 8.10.

C.8 Exec:

```
exec:command
```

Runs *command* as an external program.

See also: Section 8.6.

C.9 Extract

```
extract pattern
        [-startswith start]
        [-endswith end]
        [-separatedby sep]
        [-as type]
```

Extracts all regions matching *pattern*.

- `-startswith` prints *start* before each extracted region.
- `-endswith` prints *end* after each extraction region.
- `-separatedby` prints *sep* between each pair of regions (after the previous region's *end* and before the next region's *start*).
- `-as` converts the extracted regions to *type*, which may be either `text` or `html`.

See also: Section 8.1.1.

C.10 File:

```
file:filename
```

Loads the file named *filename* and returns it as the current document.

See also: Section 8.3.

C.11 Forward

```
forward [n]
```

Goes forward to the next document in the page history. The optional argument `n` is the number of pages to go forward. This command has the same effect as the Forward toolbar button.

See also: Section 8.8.

C.12 Ftp:

`ftp://`*`hostname/pathname`*

Retrieves a file by FTP and returns it as the current document.

See also: Section 8.3.

C.13 History

```
history
```

Prints the page history to standard output. This command is designed for the LAPIS typescript shell (`lapis -tty`), which would otherwise have no other way to show the history.

See also: Section 8.8.

C.14 Http:

`http://`*`hostname/pathname`*

Retrieves a web page by HTTP and returns it as the current document.

See also: Section 8.3.

C.15 Insert

`insert` *`pattern string`*

Inserts *`string`* at all points matched by *`pattern`*. Synonym for `replace`.

See also: Section 8.1.4.

C.16 Keep

`keep` *`pattern`* `[-outof` *`recordpattern`*`]`

Keeps only records matching *`pattern`* and deletes the rest.

- `-outof` specifies a record set rather than inferring it from *`pattern`*.

See also: Section 8.1.3.

C.17 Omit

```
omit pattern
```

Deletes all regions matching *pattern*.
See also: Section 8.1.3.

C.18 Parse

```
parse parser
```

Binds the patterns described by *parser* into the pattern library. The *parser* can be one of three possibilities:

- a filename ending in `.tcl`, which is interpreted as a script of Tcl commands;
- a filename ending in `.tc`, which is interpreted as a file of TC patterns;
- the name of a Java class implementing `lapis.Parser`. The class is loaded, an instance is created, and its `bind` method is called.

See also: Section 8.6.

C.19 Property

```
property [-get] name
```

Returns the value of the property named *name* on the current document.

```
property -set name value
```

Sets the *name* property to *value*.

```
property -list
```

Returns a Tcl list of the property names defined on the current document.
See also: Section 8.13.

C.20 Relocate

```
relocate
   [-base url]
   [-override]
```

Adds the HTML element <base href=*url*> to the current document. The *url* is obtained as follows:

1. From the -base argument, if specified.
2. From the base property of the current document, if any.
3. From the url property of the current document, if any.

If none of these can be found, relocate makes no change to the current document. If the current document already has a <base> element, relocate does nothing unless the -override option forces it to replace the existing <base>.

See also: Section 8.13.

C.21 Replace

```
replace pattern template
```

Replaces all regions matching *pattern* with *template*. The template may include embedded pattern substitutions surrounded by curly braces.

See also: Section 8.1.4.

C.22 Save

```
save [filename]
     [-backup extension]
```

Saves the current document to *filename*, or to the file the current document was loaded from if no *filename* is specified.

If the file already exists, the old contents are backed up to *filename*~ by default. The -backup option changes the backup extension from ~ to *extension*. If *extension* is the empty string, backup is disabled.

C.23 Show

```
show [-brief] [-all]
```

Prints the content of the current document to standard output. This command is designed for use in the LAPIS typescript shell (lapis -tty) and for scripts run from the command line.

- `-brief` displays at most a fixed number of lines, half from the start of the document and half from the end. The number of lines displayed is controlled by the Tcl variable `lapis::displayLimit`, which defaults to 25. This is the default when `show` is called interactively.
- `-all` displays the entire document. This is the default when `show` is used in a script.

C.24 Sort

```
sort pattern
     [-by keypattern]
     [-order [reverse] dictionary|numeric|unicode|random]
```

Sorts the regions matching *`pattern`*.

- `-by` specifies a sort key in each record. If no `-by` option is specified, the entire record is used as a sort key.
- `-order` specifies the sort order. Default is `dictionary`.

Multiple sort keys may be specified with multiple `-by` and `-order` arguments.

See also: Section 8.1.2.

C.25 Submit

```
submit [-form formpattern]
       [-button buttonpattern]
```

Submits a web form in the current page.

- `-form` specifies the form to submit. If `-form` is omitted, the first form in the page is used.
- `-button` specifies the button that should be pressed to submit the form. If no `-button` argument is given, the first button of type `submit` is pressed.

See also: Section 8.10.

Bibliography

[AKW88] Alfred V. Aho, Brian W. Kernighan, and Peter J. Weinberger. *The AWK Programming Language*. Addison-Wesley, 1988.

[All83] James F. Allen. Maintaining knowledge about temporal intervals. *Communications of the ACM*, 26(11):832–843, November 1983.

[And73] Michael R. Anderberg. *Cluster Analysis for Applications*. Academic Press, 1973.

[App02] Hypercard 2.4.1. http://www.apple.com/hypercard/, 2002.

[AS95] Eric Z. Ayers and John T. Stasko. In *Proceedings of the 4th International World Wide Web Conference (WWW4)*, pages 259–270, 1995.

[Aut02] Autocad. http://www.autodesk.com/, 2002.

[Bad99] Greg J. Badros. JavaML: A markup language for java source code. In *Ninth International World Wide Web Conference*, 1999.

[Ben75] Jon L. Bentley. Multidimensional binary search trees used for associative searching. *Communications of the ACM*, 18(9):509–517, September 1975.

[BKSS90] Norbert Beckmann, Hans-Peter Kriegel, Ralf Schneider, and Bernhard Seeger. The R*-tree: an efficient and robust access method for points and rectangles. In *ACM SIGMOD International Conference on Management of Data*, pages 322–331, 1990.

[BL84] Vic Barnett and Toby Lewis. *Outliers in Statistical Data*. Wiley, 2nd edition, 1984.

[Bla01] Alan F. Blackwell. Swyn: A visual representation for regular expressions. In *Your Wish is My Command: Giving Users the Power to Instruct Their Software*. Morgan Kauffman, 2001.

[BLLJ98] Bert Bos, Hakon Lie, Chris Lilley, and Ian Jacobs. Cascading style sheets, level 2 (CSS2) specification. Technical Report http://www.w3.org/TR/REC-CSS2/, W3C, 1998.

[BM80] Jon L. Bentley and Hermann A. Maurer. Efficient worst-case data structures for range searching. *Acta Informatica*, 13(2):155–168, 1980.

[Bru97] Amy Bruckman. *MOOSE Crossing: Construction, Community, and Learning in a Networked Virtual World for Kids*. PhD thesis, Masschusetts Institute of Technology Media Lab, May 1997.

[BYN96] Ricardo A. Baeza-Yates and Gonzalo Navarro. Integrating contents and structure in text retrieval. *ACM SIGMOD Record*, 25(1):67–79, 1996.

[Car72] Lewis Carroll. The Jabberwocky. *Through the Looking-Glass and What Alice Found There*, 1872.

[CC97] Charles L. A. Clarke and Gordon V. Cormack. On the use of regular expressions for searching text. *ACM Transactions on Programming Languages and Systems*, 19(3):413–426, May 1997.

[CCB95] Charles L. A. Clarke, Gordon V. Cormack, and F. J. Burkowski. An algebra for structured text search and a framework for its implementation. *The Computer Journal*, 38(1):43–56, May 1995.

[CLR92] Thomas H. Cormen, Charles E. Leiserson, and Ronald L. Rivest. *Introduction to Algorithms*. MIT Press, 1992.

[Cov96] Robin Cover. Dsssl – document style semantics and specification language. Technical Report 10179:1996, ISO/IEC, 1996.

[Cre97] Roger F. Crew. ASTLOG: a language for examing abstract syntax trees. In *Proceedings of the USENIX Conference on Domain-Specific Languages*, 1997.

[Cyp93] Allen Cypher. Eager: Programming repetitive tasks by demonstration. In Allen Cypher, editor, *Watch What I Do: Programming by Demonstration*, pages 205–218. MIT Press, 1993.

[Dan92] Dan R. Olsen, Jr. Bookmarks: An enhanced scroll bar. *ACM Transactions on Graphics*, 11(3):291–295, July 1992.

[DAW98] Anind K. Dey, Gregory A. Abowd, and Andrew Wood. CyberDesk: a framework for providing self-integrating ubiquitous software services. In *Proceedings of the International Conference on Intelligent User Interfaces (IUI '98)*, pages 47–54, 1998.

[DE95] Chris DiGiano and Mike Eisenberg. In *Symposium on Designing Interactive Systems (DIS '95)*, pages 189–197, 1995.

[DeJ98] Jacl and Tcl Blend. http://www.scriptics.com/software/java, 1998.

[DMO01] Steve DeRose, Eve Maler, and David Orchard. Xml linking language (xlink) version 1.0. Technical Report http://www.w3.org/TR/xlink/, W3C, 2001.

[Ede80] Herbert Edelsbrunner. Dynamic data structures for orthogonal intersection queries. Rep. F59, Univ. Graz, Institute fur Informationsverarbeitung, 1980.

[FB74] Raphael A. Finkel and Jon L. Bentley. Quad trees: a data structure for retrieval on composite keys. *Acta Informatica*, 4(1):1–9, 1974.

[FGK93] Robert Fourer, David M. Gay, and Brian W. Kernighan. *AMPL: a modeling language for mathematical programming*. Duxbury Press, 1993.

[Fin80] Craig A. Finseth. Theory and practice of text editors, or, a cookbook for an EMACS. Technical Memo 165, MIT Lab for Computer Science, May 1980.

[Fre98] Dayne Freitag. *Machine Learning for Information Extraction in Informal Domains*. PhD thesis, Computer Science Department, Carnegie Mellon University, November 1998.

[Fuj98] Yuzo Fujishima. Demonstrational automation of text editing tasks involving multiple focus points and conversions. In *Proceedings of the International Conference on Intelligent User Interfaces (IUI '98)*, pages 101–108, 1998.

[GAM96] William G. Griswold, Darren C. Atkinson, and Collin McCurdy. Fast, flexible syntactic pattern matching and processing. In *Proceedings 4th Workshop on Program Comprehension*, pages 144–153, 1996.

[GDCG90] Sharon L. Greene, Susan J. Devlin, Philip Cannata, and Louis M. Gomez. No IFs, ANDs, or ORs: a study of database querying. *International Journal of Man-Machine Studies*, 32(3):303–326, 1990.

[GG83] Ralph E. Griswold and Madge T. Griswold. *The Icon Programming Language*. Prentice-Hall, 1983.

[Gol90] Charles F. Goldfarb. *The SGML Handbook*. Oxford University Press, 1990.

[GPP71] Ralph E. Griswold, James F. Poage, and Ivan P. Polonsky. *The SNOBOL4 Programming Language*. Prentice-Hall, 1971.

[GT87] Gaston H. Gonnet and Frank W. Tompa. Mind your grammar: a new approach to modelling text. In *Proceedings of the ACM Conference on Very Large Databases (VLDB '87)*, pages 339–345, 1987.

[Gus97] Dan Gusfield. *Algorithms on Strings, Trees, and Sequences: Computer Science and Computational Biology*. Cambridge University Press, 1997.

[Gut84] Antonin Guttman. R-tree: a dynamic index structure for spatial searching. In *ACM SIGMOD International Conference on Management of Data*, pages 47–57, 1984.

[Han71] Wilfred J. Hansen. User engineering principles for interactive systems. In *AFIP Conference proceedings, Fall joint computer conference*, pages 523–532, 1971.

[HH92] William C. Hill and James D. Hollan. Edit wear and read wear. In *Proceedings of ACM Conference on Human Factors in Computing Systems (CHI '92)*, pages 3–9, 1992.

[HN83] Klaus Hinrichs and Jürg Nievergelt. In *Proceedings of the WG'83 International Workshop on Graph-theoretic Concepts in Computer Science*, pages 100–113, 1983.

[HN86] Nico Habermann and David Notkin. Gandalf: Software development environments. *IEEE Transactions on Software Engineering*, 12(12):1117–1127, December 1986.

[HU79] John E. Hopcroft and Jeffrey D. Ullman. *Introduction to Automata Theory, Languages, and Computation*. Addison-Wesley, 1979.

[Imm88] Neil Immerman. Nondeterministic space is closed under complementation. *SIAM Journal of Computing*, 17(5):935–938, 1988.

[Jak99] Jakarta Project. http://jakarta.apache.org/regexp/, 1999.

[Jav00] JavaCC. http://www.webgain.com/products/java_cc/, 2000.

[JB97] David Jackson and Michael A. Bell. String-pattern matching in a visual programming language. *Journal of Visual Languages and Computing*, 8(5-6):545–561, 1997.

[JK96] Jani Jaakkola and Pekka Kilpelainen. Using sgrep for querying structured text files. Report C-1996-83, University of Helsinki, Department of Computer Science, 1996.

[Joh75] Stephen C. Johnson. Computing Science Tech. Rep. 32, AT&T Bell Labs, Murray Hill, NJ, 1975.

[Kay01] Michael Kay. Xsl transformations (xslt) version 2.0. Technical Report http://www.w3.org/TR/xslt20/, W3C, 2001.

[KDE02] KDE desktop environment. http://www.kde.org/, 2002.

[KK94] Ronald Kaplan and Martin Kay. Regular models of phonological rule systems. *Computational Linguistics*, 20(3):331–378, 1994.

[KLM+97] Gregor Kiczales, John Lamping, Anurag Mendhekar, Chris Maeda, Cristina Videira Lopes, Jean-Marc Loingtier, and John Irwin. Aspect-oriented programming. In *European Conference on Object-Oriented Programming (ECOOP)*, 1997.

[KM93] Pekka Kilpelainen and Heikki Mannila. Retrieval from hierarchical texts by partial patterns. In *Proceedings of the 16th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 214–222, 1993.

[KM98] Thomas Kistler and Hannes Marais. WebL – a programming language for the web. In *Proceedings of the 7th International World Wide Web Conference (WWW7)*, 1998.

[KN98] Edwin M. Knorr and Raymond T. Ng. Algorithms for mining distance-based outliers in large datasets. In *Proceedings of the 24th International Conference on Very Large Databases (VLDB)*, pages 392–403, 1998.

[KP84] Brian W. Kernighan and Rob Pike. *The UNIX Programming Environment*. Prentice-Hall, 1984.

[KP99] Brian W. Kernighan and Rob Pike. *The Practice of Programming*. Addison-Wesley, 1999.

[KR88] Brian W. Kernighan and Dennis M. Ritchie. *The C Programming Language*. Prentice-Hall, 2nd edition, 1988.

[Kru97] Bruce Krulwich. Automating the internet: Agents as user surrogates. *IEEE Internet Computing*, 1(4):34–38, 1997.

[Kul97] Zenon Kulpa. Diagrammatic representation for a space of intervals. *Machine Graphics and Vision*, 6(1):5–24, 1997.

[KWD97] Nicholas Kushmerick, Daniel S. Weld, and Robert Doorenbos. In *Proceedings of International Joint Conference on Artificial Intelligence (IJCAI)*, pages 729–737, 1997.

[Les75] Michael E. Lesk. Lex – a lexical analyzer generator. Computing Science Tech. Rep. 39, AT&T Bell Labs, Murray Hill, NJ, 1975.

[LH95] Jurgen Landauer and Masahito Hirakawa. Visual AWK: a model for text processing by demonstration. In *Proceedings of the 11th International IEEE Symposium on Visual Languages '95*, pages 267–274, 1995.

[Lis81] Barbara Liskov. *CLU Reference Manual*. Springer-Verlag, 1981.

[LNW98] Henry Lieberman, Bonnie A. Nardi, and David Wright. Grammex: Defining grammars by example. In *Proceedings of Human Factors in Computing Systems (CHI 98)*, pages 11–12, 1998.

[Lut00] Mark Lutton. Report on hacker altering mit grades: Not! *comp.risks*, 20(84), March 2000.

[LW77] Der-Tsai Lee and C.K. Wong. Worst-case analysis for region and partial region searches in multidimensional binary search trees and quad trees. *Acta Informatica*, 9(1):23–29, 1977.

[LWDW01] Tessa Lau, Steven Wolfman, Pedro Domingos, and Daniel S. Weld. Learning repetitive text-editing procedures with SMARTedit. In Henry Lieberman, editor, *Your Wish Is My Command: Giving Users the Power to Instruct Their Software*, pages 209–226. Morgan Kaufmann, 2001.

[Lyx02] Lyx – the document processor. http://www.lyx.org/, 2002.

[Mac91] Ian MacLeod. A query language for retrieving information from hierarchic text structures. *The Computer Journal*, 34(3):254–264, 1991.

[Mau94] David Maulsby. *Instructible Agents*. PhD thesis, Department of Computer Science, University of Calgary, 1994.

[MB98a] James R. Miller and Thomas Bonura. From documents to objects: An overview of LiveDoc. *SIGCHI Bulletin*, 30(2):53–58, 1998.

[MB98b] Robert C. Miller and Krishna Bharat. SPHINX: a framework for creating personal, site-specific web crawlers. *Computer Networks and ISDN Systems*, 30(1–7):119–130, 1998.

[MC74] Robert Morris and Lorinda L. Cherry. Computer detection of typographical errors. Technical Report 18, Bell Laboratories, July 1974.

[McC81] Edward M. McCreight. Priority search trees. Tech Report CSL-81-5, Xerox PARC, 1981.

[Mic02] Microsoft. Complete tasks quickly with Smart Tags in Office XP. http://office.microsoft.com/assistance/2002/articles/oQuickSmartTags.aspx, 2002.

[Min92] Sten Minor. Interacting with structure-oriented editors. *International Journal of Man-Machine Studies*, 37(4):399–418, 1992.

[MM97] Robert C. Miller and Brad A. Myers. Creating dynamic world wide web pages by demonstration. Technical Report CMU-CS-97-131 (and CMU-HCII-97-101), CMU School of Computer Science, May 1997.

[MM99] Robert C. Miller and Brad A. Myers. Lightweight structured text processing. In *Proceedings of the 1999 USENIX Annual Technical Conference*, pages 131–144, June 1999.

[MM00] Robert C. Miller and Brad A. Myers. Integrating a command shell into a web browser. In *USENIX 2000 Annual Technical Conference*, pages 171–182, June 2000.

[MM01a] Robert C. Miller and Brad A. Myers. Interactive simultaneous editing of multiple text regions. In *Proceedings of the 2001 USENIX Annual Technical Conference*, pages 161–174, June 2001.

[MM01b] Robert C. Miller and Brad A. Myers. Outlier finding: Focusing human attention on possible errors. In *Proceedings of User Interface Software and Technology (UIST 2001)*, November 2001. To appear.

[MM02] Robert C. Miller and Brad A. Myers. Multiple selections in smart text editing. In *Proceedings of the Sixth International Conference on Intelligent User Interfaces (IUI 2002)*, pages 103–110, 2002.

[MN96] Gail C. Murphy and David Notkin. Lightweight lexical source model extraction. *ACM Transactions on Software Engineering and Methodology*, 5(3):262–292, 1996.

[MP71] Robert McNaughton and Seymour Papert. *Counter-Free Automata*. MIT Press, 1971.

[MSC+86] James H. Morris, Mahadev Satyanarayanan, Michael H. Conner, John H. Howard, David S. H. Rosenthal, and F. Donelson Smith. Andrew: a distributed personal computing environment. *Communications of the ACM*, 29(3):184–201, March 1986.

[Mye93] Brad A. Myers. Tourmaline: Text formatting by demonstration. In Allen Cypher, editor, *Watch What I Do: Programming by Demonstration*, pages 309–322. MIT Press, 1993.

[Mye94] Brad A. Myers. User Interface Software Tools. http://www.cs.cmu.edu/~bam/toolnames.html, 1994.

[Mye98] Brad Myers. Natural programming: Project overview and proposal. Technical Report CMU-CS-98-101, Carnegie Mellon University School of Computer Science, January 1998.

[NBY95] Gonzalo Navarro and Ricardo A. Baeza-Yates. A language for queries on structure and contents of textual databases. In *Proceedings of the ACM Conference on Information Retrieval (SIGIR '95)*, pages 93–101, 1995.

[Nix85] Robert Nix. Editing by example. *ACM Transactions on Programming Languages and Systems*, 7(4):600–621, October 1985.

[NMW98] Bonnie A. Nardi, James R. Miller, and David J. Wright. Collaborative, programmable intelligent agents. *Communications of the ACM*, 41(3):96–104, 1998.

[Omn99] Omnimark. http://www.omnimark.com/, 1999.

[Ous94] John K. Ousterhout. *Tcl and the Tk Toolkit*. Addison-Wesley, 1994.

[Pap91] Athanasios Papoulis. *Probability, Random Variables, and Stochastic Processes*. McGraw-Hill, 3rd edition, 1991.

[Pik87] Rob Pike. The text editor sam. *Software Practice and Experience*, 17(11):813–845, 1987.

[Pik94] Rob Pike. Acme: a user interface for programmers. In *Proceedings of the USENIX 1994 Winter Technical Conference*, pages 223–234, 1994.

[PK97] Milind S. Pandit and Sameer Kalbag. The selection recognition agent: instant access to relevant information and operations. In *Proceedings of the International Conference on Intelligent User Interfaces (IUI '97)*, pages 47–52, 1997.

[PM96] John F. Pane and Brad A. Myers. Usability issues in the design of novice programming systems. Technical Report CMU-CS-96-132, Carnegie Mellon School of Computer Science, August 1996.

[PM00] John F. Pane and Brad A. Myers. Tabular and textual methods for selecting objects from a group. In *Proceedings of VL 2000: IEEE International Symposium on Visual Languages*, pages 157–164, September 2000.

[PRM01] John F. Pane, Chotirat Ratanamahatana, and Brad A. Myers. Studying the language and structure in non-programmers' solutions to programming problems. *International Journal of Human-Computer Studies*, 54(2):237–264, February 2001.

[PS85] Franco P. Preparata and Michael Ian Shamos. *Computational Geometry: an Introduction*. Springer-Verlag, 1985.

[Qui86] J. Ross Quinlan. Induction of decision trees. *Machine Learning*, 1:81–106, 1986.

[Rit86] Jean-Francois Rit. Propagating temporal constraints for scheduling. In *Proceedings of the Fifth National Conference on AI (AAAI-86)*, pages 383–388. Morgan Kaufmann, 1986.

[RT89] Thomas W. Reps and Tim Teitelbaum. *The Synthesizer Generator: A System for Constructing Language-Based Editors*. Springer-Verlag, 1989.

[Sam90] Hanan Samet. *The Design and Analysis of Spatial Data Structures*. Addison-Wesley, 1990.

[Sch99] Gary L. Schaps. Compiler construction with ANTLR and java. *Dr. Dobb's Journal*, March 1999.

[SK98] Atsushi Sugiura and Yoshiyuki Koseki. Internet Scrapbook: Automating web browsing tasks by demonstration. In *Proceedings of User Interface Software and Technology (UIST 98)*, pages 9–18, 1998.

[ST92] Airi Salminen and Frank W. Tompa. PAT expressions: an algebra for text search. Report OED-92-02, University of Waterloo Centre for the New Oxford English Dictionary and Text Research, 1992.

[Sta81] Richard M. Stallman. In *ACM SIGPLAN SIGOA Symposium of Text Manipulation*, pages 147–156, 1981.

[TRH81] Tim Teitelbaum, Thomas Reps, and Susan Horwitz. The why and wherefore of the cornell program synthesizer. In *Proc ACM SIGPLAN SIGOA Symposium on Text Manipulation*, pages 8–16, 1981.

[VGB92] Michael L. Van De Vanter, Susan L. Graham, and Robert A. Ballance. Coherent user interface for language-based editing systems. *International Journal of Man-Machine Studies*, 37(4):431–466, 1992.

[W3C00] W3C. Extensible markup language (XML) 1.0. http://www.w3.org/TR/2000/REC-xml-20001006, October 2000. second edition.

[Wat82] Richard C. Waters. *SIGPLAN Notices*, 17(7):39–46, 1982.

[WCS96] Larry Wall, Tom Christensen, and Randal L. Schwartz. *Programming Perl*. O'Reilly, 2nd edition, 1996.

[WM92] Sun Wu and Udi Manber. Agrep – a fast approximate pattern searching tool. In *Proceedings of the Winter USENIX Technical Conference*, pages 153–162, 1992.

[WM93] Ian H. Witten and Dan Mo. TELS: Learning text editing tasks from examples. In Allen Cypher, editor, *Watch What I Do: Programming by Demonstration*, pages 183–204. MIT Press, 1993.

[Woo81] Steven R. Wood. Z — the 95% program editor. In *Proc ACM SIGPLAN SIGOA Symposium on Text Manipulation*, pages 1–7, 1981.