

Chapter 7

LAPIS User Interface

The ideas embodied in this thesis are implemented in a user interface called LAPIS. LAPIS combines a web browser and a plain-text editor into a single user interface, so that all three kinds of text relevant to this thesis can be edited and manipulated: web pages, plain text, and source code.

LAPIS has a number of interesting and novel features, the discussion of which will occupy the remainder of this dissertation. The current chapter describes multiple selections — specifically, how the user can make multiple selections with pattern matching or the mouse, how multiple selections are rendered on the screen, and how multiple selections are used for editing. The chapter concludes by describing a user study of multiple selections in LAPIS, focusing in particular on making selections with TC patterns.

Later chapters discuss the other important features of LAPIS: Unix-style tools and scripting (Chapter 8), inferring multiple selections from examples (Chapter 9), and finding outliers in multiple selections (Chapter 10).

7.1 Overview of LAPIS

A screenshot of LAPIS is shown in Figure 7.1. The LAPIS window has several main components:

- the *browser pane*, which displays a rendered HTML page or a text file.
- the *command bar*, where the user can enter the URL of a web page or file to load. As its name suggests, the command bar can also interpret typed commands in a scripting language (Tcl), which is described in more depth in Chapter 8. At the end of the command bar is the *View As* control, which allows the user to toggle the view back and forth between rendered HTML and plain text.
- the *toolbar*, which has not only the typical browser controls (Back, Forward, Reload, Home, and Stop) but also file-editing controls (New, Open, Save; Cut, Copy, Paste). The toolbar also has three buttons that control the selection inference mode (Manual, Guessing, and Simultaneous Editing), which are described in more depth in Chapter 9.
- the *pattern pane*, in which the user can type and execute TC patterns. The pattern pane is also used to display patterns automatically generated by other components of LAPIS, such as selection inference.

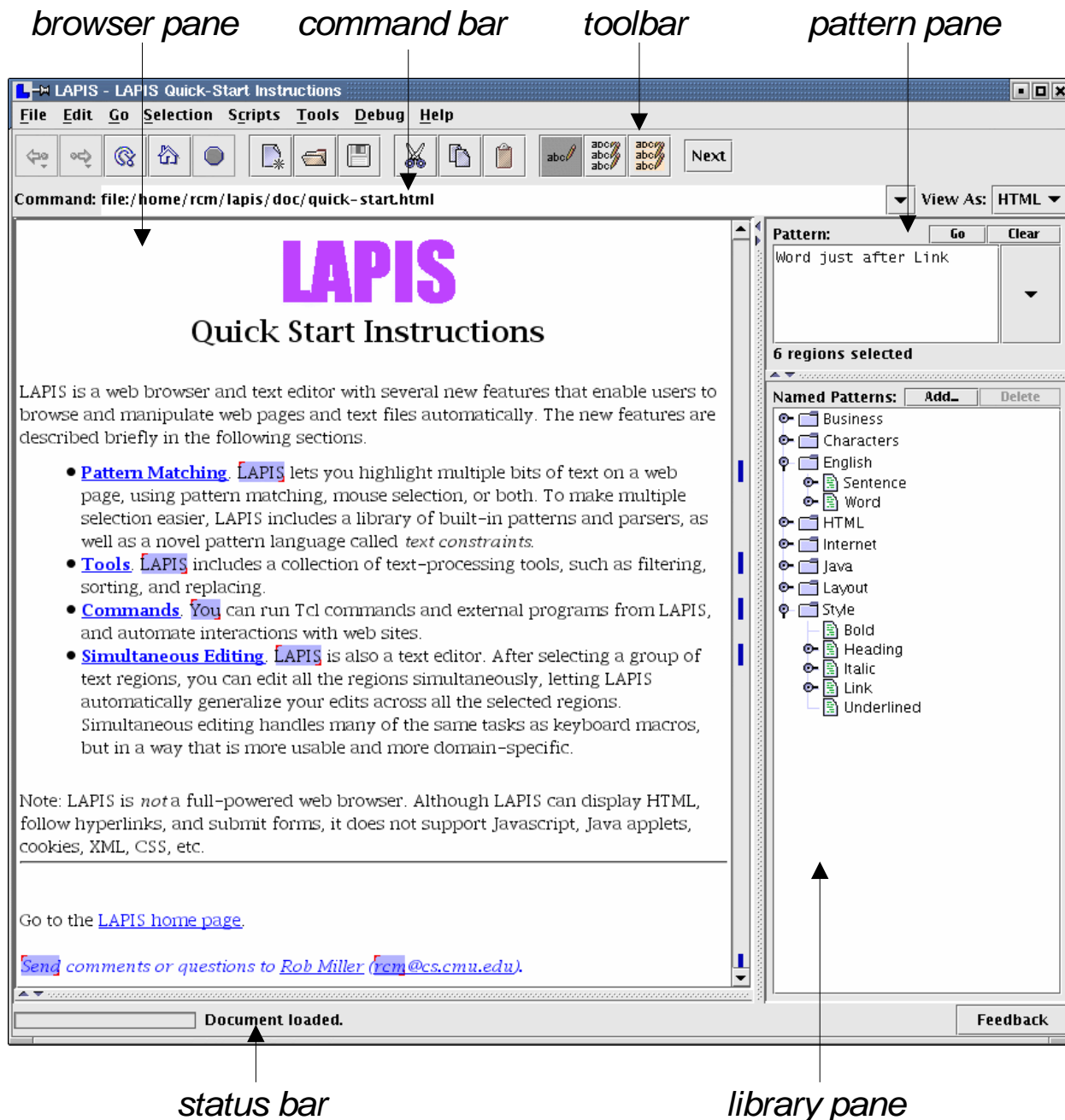


Figure 7.1: Screenshot of a LAPIS window showing its major components.

- the *library pane*, which displays the pattern identifiers that are defined in the global pattern library.
- the *status bar*, which displays progress messages and a percent-done indicator while LAPIS is downloading URLs, searching for patterns, or performing inference.

Although LAPIS can display both rendered HTML and plain text, its behavior is somewhat different in each case. When LAPIS is showing rendered HTML, it behaves like a web browser. The user can click on hyperlinks and fill out HTML forms, but cannot directly edit the content of the page. When LAPIS is showing plain text, on the other hand, it behaves like a text editor, allowing the user to insert and delete text. The user can toggle a page back and forth between the rendered, read-only view and the plain-text, editable view using the *View As* drop-down menu. A future version of LAPIS may lift this restriction, so that editing can be done in the rendered HTML view as well.

7.2 Multiple Selections

Unlike many web browsers and text editors, LAPIS permits arbitrary *multiple selections* to be made in the browser pane. Multiple selections are used for two purposes in LAPIS: to display the result of a pattern match, and to specify the arguments of editing commands (like Copy and Delete) and text-processing tools (like Extract and Sort). The screenshot in Figure 7.1 shows multiple selections: the word just after every hyperlink is selected.

Most browsers and editors support only one contiguous selection at a time. Some editors, notably Emacs and Microsoft Word, also allow *rectangular* selections, consisting of all the characters in a rectangle of row and column positions. The latest version of Microsoft Word, Word XP, supports multiple selections, but not multiple insertion points (zero-length selections).

Internally, the current selection is represented as an arbitrary region set, which may include nested or overlapping regions. This raises a number of questions:

- *Highlighting*: How should LAPIS display which regions are selected, if regions may overlap or nest?
- *Manual mouse selection*: How can the user add and remove selections from an arbitrary region set using the mouse?
- *Commands*: How should editing commands like Copy or Delete behave on an arbitrary region set?

The following sections give some partial answers to these questions. Generally, however, most of the design effort in LAPIS has been focused on the common case, in which the selection is a flat, non-overlapping region set. The user study described at the end of this chapter used only flat selections.

7.3 Highlighting Multiple Selections

This section describes how LAPIS highlights the current selection in the browser pane. For simplicity of presentation, the discussion is broken down by type of region set. First, techniques for highlighting flat region sets are presented. These techniques have been implemented in LAPIS and tested on users. Next is presented a technique for visually depicting the result of a unary relational operator, such as *containing* or *starting* or *just after*. This technique has been implemented in LAPIS but not tested on users. Then, some proposals for depicting nested and overlapping regions are presented, although these proposals have neither been implemented nor tested on users. The section ends by showing how the LAPIS scrollbar is augmented to indicate where selections are found in a long document.

7.3.1 Flat Region Sets

Even when the selection is a flat region set, LAPIS selection highlighting is subtly different from conventional text highlighting. Conventional text editors use a solid colored background that completely fills the selected text's bounds. Using this technique to highlight a flat region set has two problems.

First, two selected regions that are adjacent would be indistinguishable from a single selection that spans both regions. This problem is solved by shrinking the colored background by one pixel on all sides, leaving a two-pixel white gap between adjacent selections.

Second, two selections separated by a line break would be indistinguishable from a single selection that spans the line boundary. LAPIS solves this problem by adding small handles to each selection, one in the upper left corner and the other in the lower right corner, to indicate the start and end of the selection.

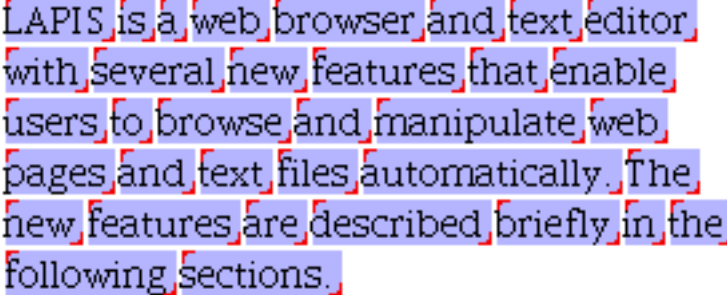
These small changes to conventional text highlighting allow any flat region set to be clearly and unambiguously rendered. Figure 7.2 shows how LAPIS highlights three different region sets in the same paragraph. Each of the region sets completely spans the given paragraph, in the sense that every character is included in some highlighted region. Thus conventional highlighting would highlight the entire paragraph in all three cases. In LAPIS, however, the white gaps and handles clearly mark the region boundaries, making it easy to distinguish the region set broken on word boundaries and the set broken on line boundaries from the set consisting of the entire paragraph.

7.3.2 Region Sets Created by Unary Relational Operators

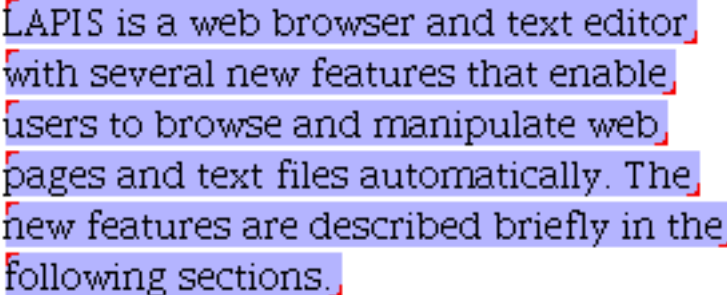
A unique feature of the TC pattern language, relative to other structured text pattern languages, is that its relational operators are *unary*. The user can write a pattern like

```
contains "web browser"
```

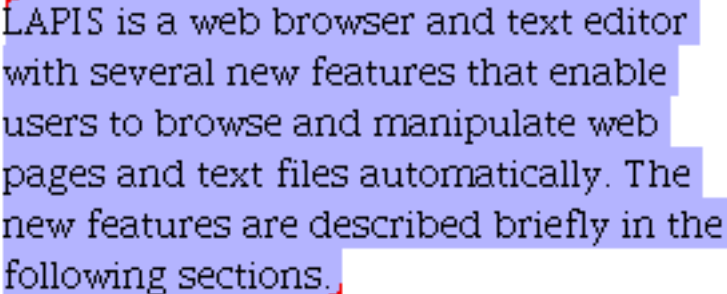
to match the set of all regions that contain an occurrence of the string “web browser”. A pattern like this would probably not be useful for editing, since it is underconstrained. But the ability to visualize the region set matched by this pattern may still be useful, for two reasons. First, `contains "web browser"` may be a subexpression in a larger pattern that the user is trying to understand or debug. Second, unary relational operators are used as features for machine learning

A screenshot of a text editor showing a paragraph of text. Each individual word, including the space after it, is highlighted with a blue background. The text is: "LAPIS is a web browser and text editor with several new features that enable users to browse and manipulate web pages and text files automatically. The new features are described briefly in the following sections."

(a) words (including the space after each word)

A screenshot of a text editor showing the same paragraph of text. Each entire line of text is highlighted with a blue background. The text is: "LAPIS is a web browser and text editor with several new features that enable users to browse and manipulate web pages and text files automatically. The new features are described briefly in the following sections."

(b) lines

A screenshot of a text editor showing the same paragraph of text. The entire paragraph, from the first line to the last, is highlighted with a blue background. The text is: "LAPIS is a web browser and text editor with several new features that enable users to browse and manipulate web pages and text files automatically. The new features are described briefly in the following sections."

(c) entire paragraph

Figure 7.2: Different region sets in the same paragraph, highlighted in LAPIS.

(Chapters 9 and 10), and the ability to display those features in the context of the document is another way to give the user feedback about how the system is learning.

With these goals in mind, a technique was developed that can visualize any region rectangle as a highlight in the browser pane. The highlight for flat regions described in the previous section is a special case of this technique. The technique is implemented in LAPIS, and it can display the results of unary relational operator patterns like the example shown above.

Recall from Section 4.1 that applying a unary relational operator like `contains` to a region produces a rectangle in region space. Each rectangle resulting from the relational operator is rendered by LAPIS using the following rules:

- characters included in every region in the rectangle are highlighted in dark blue;
- characters included in some but not all regions in the rectangle are highlighted in light blue;
- the boundary between dark blue and light blue highlighting is indicated by a smooth gradient;
- if all regions in the rectangle share the same start point (or end point), then that point is marked by a start handle (or end handle).

Figure 7.3 shows the effect of these rules on some common unary operators. For example, consider Figure 7.3(a). The characters in "web browser" are highlighted in dark blue, because every region that satisfies the condition `contains "web browser"` must include these characters. The characters outside "web browser" are also highlighted, but in light blue, because *some* region matching the condition includes them.

Figure 7.3(b), which shows the result of the `in` operator, is treated as a special case. Following the rules above, the highlight for `in` would simply be light blue, with no gradients. Since this would be indistinguishable from conventional text highlighting, however, LAPIS adds a gradient to each end of the highlight.

7.3.3 Nested and Overlapping Region Sets

The highlighting technique described previously is used to highlight all region sets in LAPIS. Nested or overlapping regions are simply drawn on top of each other. The handles are always drawn last, so at least the endpoints of regions can be distinguished. Still, the results are not ideal. For example, Figure 7.4 shows that LAPIS highlights two different region sets, one nested and the other overlapping, in exactly the same way.

Solving this problem requires augmenting the highlight to distinguish nested regions from overlapping regions. Several alternative solutions are presented below and illustrated in Figure 7.5.

- **Underlining.** The region notation shown on the left side of Figure 7.4, which underlines each region separately, is clearly unambiguous. Inspired by that notation, highlights for nested and overlapping regions could be augmented with multiple underlines, as shown in Figure 7.5(a). Although this technique is unambiguous, it has two disadvantages. First, deeply nested regions may require more space between each line than is normally available, forcing the lines of text to move apart to accommodate a stack of underlines. Second, if the regions span multiple text lines, it would be hard for the user to follow the underlines across linebreaks.

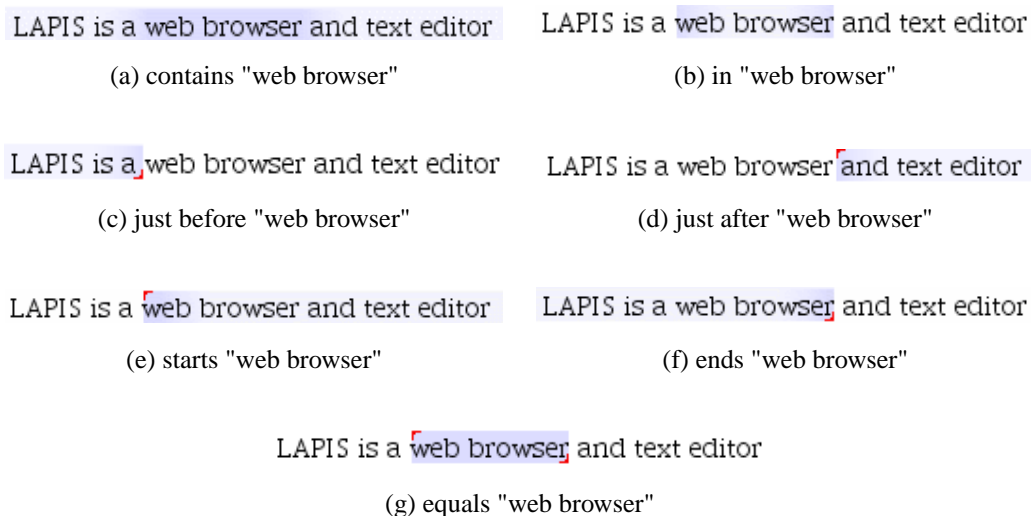


Figure 7.3: Gradient highlights displayed by LAPIS for common unary relational operators. (a) *contains* fades out in both directions away from the center; (b) *in* fades out inward to the center; (c) *just before* fades out to the left; (d) *just after* fades out to the right; (e) *starts* fades out to the right; (e) *ends* fades out to the left; and (f) *equals* has no gradient at all.

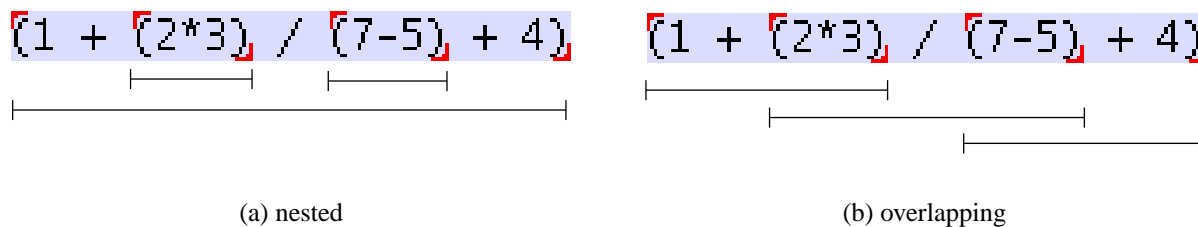


Figure 7.4: LAPIS highlighting cannot distinguish between nested and overlapping region sets.

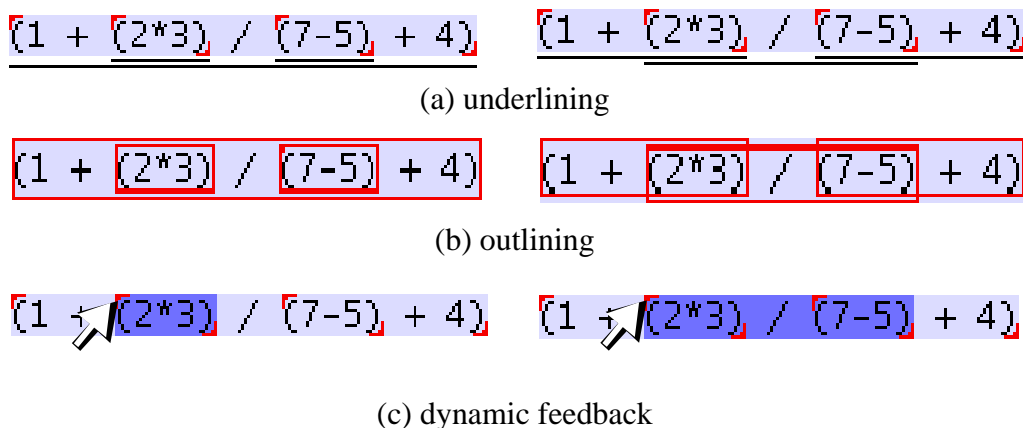


Figure 7.5: Techniques for distinguishing between nested and overlapping highlights.

- **Outlining.** Drawing an outline around each region is another way to distinguish between overlapping and nested regions. Figure 7.5(b) shows one way this outline might be rendered. Like underlines, however, outlines require extra space between lines of text to indicate deep nesting, and outlines may be hard to follow across multiple lines.
- **Dynamic feedback.** Unlike the previous techniques, this solution requires interaction from the user. When the user moves the mouse over a handle, the corresponding region receives some extra highlighting — perhaps deepening in saturation, acquiring a drop shadow or underline, or blinking briefly. Figure 7.5(c) shows what this might look like. This technique is probably far easier to implement than the previous ones, but it requires the user to actively explore the highlight in order to understand it. This technique also requires rendering multiple handles when several regions start or end at the same place.

Other plausible ideas include color-coding or numbering of highlights and handles to distinguish them. Implementation and evaluation of these techniques are left as future work.

7.3.4 Scrollbar Augmentation

To help the user keep track of multiple selections in a long document, the browser pane’s scrollbar is augmented with marks indicating where selections are located (Figure 7.6). Other systems have used the scrollbar for displaying secondary information about a document, such as bookmarks [Dan92] reading or editing frequency [HH92], and the current selection [MSC⁺86].

Each mark drawn in the scrollbar corresponds to the vertical extent of a selected region. As in flat region highlighting, 1-pixel vertical gaps are left between marks from adjacent lines if sufficient resolution is available. This allows, for example, selections of individual lines (Figure 7.6(a)) to be distinguished from a selection that spans all the lines (Figure 7.6(b)). If the document is too long or the scrollbar too short, however, the vertical gaps are sacrificed, and regions from different lines may map to the same pixel in the scrollbar. Marks are drawn on top of the scrollbar’s existing components, so that the scrollbar thumb does not obscure them, but as narrow rectangles, so that the thumb is not obscured either.

Scrollbar marks are also used in LAPIS to indicate the outliers in a selection (Chapter 10).

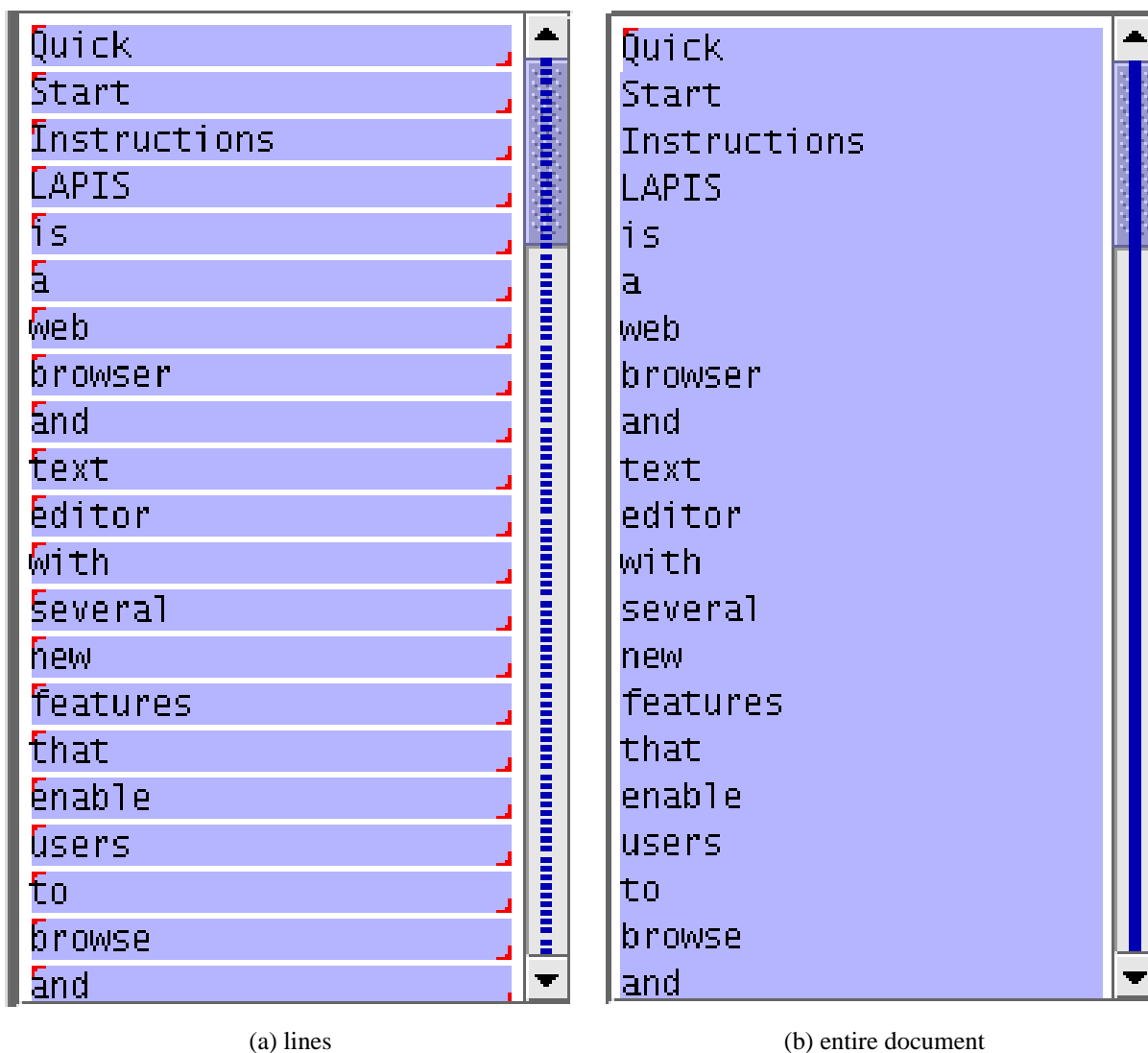


Figure 7.6: The scrollbar is augmented with marks to help the user find selections in a long document. The marks can distinguish between different region sets, such as (a) and (b).

7.4 Making Multiple Selections

LAPIS provides four basic ways to set or modify the current selection: using the mouse; choosing a pattern identifier from the library pane; writing a pattern in the pattern pane, and inferring selections from examples. The first three techniques are described in the following sections. Discussion of the fourth technique, inference from examples, is deferred to Chapter 9.

7.4.1 Selecting Regions with the Mouse

Mouse selection in LAPIS generalizes the single-region selection techniques used in other text editors. When the left mouse button is pressed over the browser pane, the selection is first cleared. If the user drags the mouse with the left button held down, a single region is selected. An insertion point (zero-length region) is selected by clicking and releasing the left button. Keyboard keys can also be used to move the insertion point or make a single-region selection, following the usual conventions. For example, a single region can be selected by holding down Shift while moving the cursor with the keyboard arrows.

To add more regions to the selection, the user holds down the Control key while clicking and dragging. This use of Control as a modifier for discontinuous selection is also conventional, mimicking its use in file managers and drawing editors. Microsoft Word XP, which was released after LAPIS, also uses Control for making discontinuous selections. At one point in the development of LAPIS, the Shift key could also be used for the same effect, but several users objected because Shift is conventionally used to *extend* a single-region selection. As a result, the conventional behavior of Shift was restored, and only Control can be used to add more regions to a selection.

At present, mouse selection can only be used to add *flat* regions to the selection. If the new region overlaps an already-selected region, it is automatically merged with the existing selected region, rather than added as a new region. This decision was made for several reasons. First, flat selections are far more common in practice, particularly when the user is editing. Second, the current LAPIS highlighting is poor at displaying nested and overlapping region sets (Section 7.3.3), so it isn't clear that users would notice selection mistakes that accidentally introduced nesting or overlapping in a flat region set. Indeed, in the user study described below (Section 7.6), which used an early version of LAPIS in which mouse selections could add overlapping and nested regions, one user accidentally created overlapping regions and never noticed the mistake. LAPIS was subsequently redesigned to make the common case (flat region sets) the default. In the future, it might be useful to support an additional modifier combination, such as Control-Alt, which would allow users to make nested or overlapping selections.

Originally, the handles at the start and end of each selected region (e.g., in Figure 7.2) were interactive, so that the user could click and drag a handle with the mouse to resize the region. However, when this behavior was tested in the user study, it was found that users tended to click the handles by accident when trying to click next to the selected region. Since there are other ways to extend a selected region — for example, using Shift-click, or making an overlapping selection that is automatically merged with it — the handles were subsequently made noninteractive.

To remove a region from the selection, the user holds down Control and clicks on the selected region, essentially toggling it off. The user can also right-click on a region to bring up the context menu, which (in addition to other commands) includes an Unselect command that unselects the

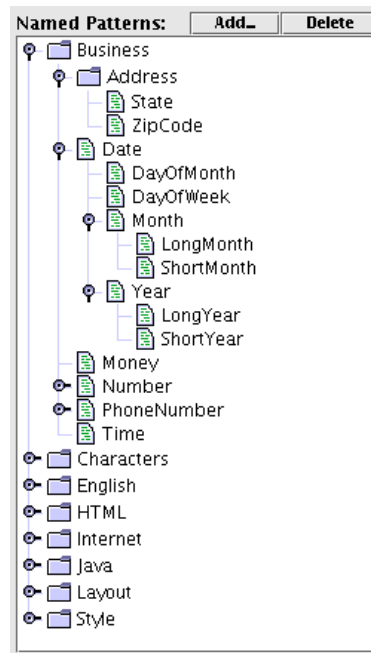


Figure 7.7: The library pane shows the patterns in the pattern library.

region that was clicked. The context menu also has an Unselect All command that clears all selections.

7.4.2 Selecting from the Library

The library pane provides another way to make a selection in LAPIS. The library pane displays the hierarchy of visible pattern identifiers in the pattern library using a tree widget. There are three kinds of items in the library pane:

- A *pure namespace* is a component of a namespace path. A pure namespace is denoted by a folder icon and a toggle switch that opens and closes its children, which are all the names in the namespace. Clicking on the name or icon also opens the namespace's children. All the top-level names, such as `Business`, `Characters`, and `English`, are pure namespaces.
- A *leaf* is a name bound to a pattern. It is denoted by a document icon filled with little marks that suggest highlighted pattern matches. Clicking on a leaf runs the associated pattern and selects the matching region set in the browser pane. In Figure 7.7, for example, `State` is a leaf name, which corresponds to the absolute pattern identifier `Business.Address.State`.
- A *bound namespace* is a namespace that is also bound to a pattern. A bound namespace is denoted by a document icon, like a leaf, but it also has children, like a namespace. In Figure 7.7, for example, `Date` is a bound namespace. Its absolute identifier, `Business.Date`, is bound to a pattern that matches entire dates, but it also acts as a namespace for other identifiers, such as `Business.Date.DayOfMonth`, which match parts of dates. Clicking

on a bound namespace not only runs its associated pattern and selects the matches in the browser pane, but also opens its children.

Named patterns can be added and removed using the Add and Delete buttons. The Add button allows the user to name the current selection. Clicking Add brings up a dialog box in which the user can enter a name. If the current selection in the browser pane was produced by a pattern, then that pattern is assigned to the entered name, with the same effect as evaluating the TC expression *name is pattern* in the root namespace.

If the current selection was made by mouse selection, however, then LAPIS creates a *constant pattern* representing the selected region set. This constant pattern is not a TC pattern, but rather an internal Java object implementing the `Pattern` interface (Section 6.2.3) that encapsulates a region set and a pointer to the document. When the constant pattern is later applied to the same document, it returns the same region set. If the document has been edited since the constant pattern was created, the offsets specified in the region set are adjusted to account for the document changes using a coordinate map. (More about coordinate maps can be found in Section 6.2.12.) Since a constant pattern refers to specific character offsets, it is only useful on the document it was created for, so applying a constant pattern to a different document always returns the empty region set. To generalize mouse selections into a pattern that can apply to other documents, the user can use selection guessing (Chapter 9).

Selecting a name and clicking the Delete button removes the selected named pattern from the library. If the deleted name was a leaf, then it disappears from the library pane. If the deleted name was a bound namespace, then it becomes a pure namespace, having lost its pattern association. A pure namespace cannot be deleted in this way. A pure namespace does not disappear from the library pane until all of its children have been deleted.

In the current version, library additions and deletions do not persist across different runs of LAPIS. Making library changes persistent currently requires changing the LAPIS configuration files by hand.

7.4.3 Selecting by Pattern Matching

The third way to make a selection in LAPIS is by writing a TC pattern in the pattern pane (Figure 7.8). Typing a pattern into this pane and pressing the Enter key or the Go button sets the current selection to all regions that match the pattern.

In order to accommodate multi-line TC patterns, the pattern pane is a multi-line text editor. Since Enter is used to run the pattern, linebreaks are inserted by pressing Control-Enter or Shift-Enter. Originally, these shortcuts were reversed: pressing Enter inserted a linebreak, and pressing Control-Enter (or the Go button) ran the pattern. In the user study, however, several users commented that using Control-Enter to run the pattern was counterintuitive — they expected to just push Enter to trigger the search, as in a Find dialog or a web search engine query form. Patterns entered interactively tend to be short anyway, so it makes sense to use the Enter key for the common case (evaluating a pattern) rather than the less common one (creating a multi-line pattern).

As shown in Figure 7.8, the pattern pane automatically displays *tie lines* to give feedback on how the system will parse the pattern. Tie lines are displayed by running the tokenization and structuring phases of TC expression parsing (Section 6.2.17) in the background as the user is editing the pattern. After each edit, the current expression is parsed into a token tree. The token

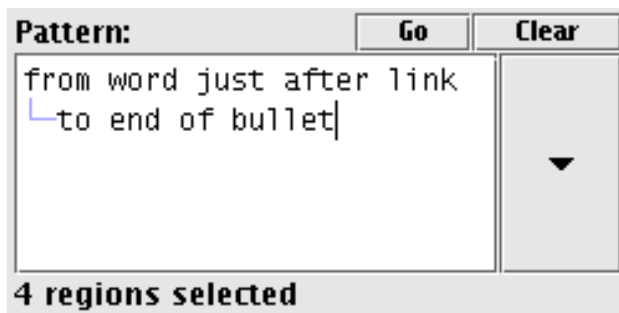


Figure 7.8: The pattern pane allows the user to enter and run TC patterns.

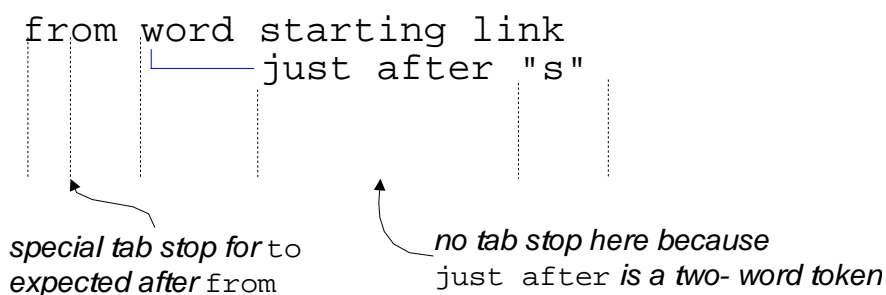


Figure 7.9: Adaptive tab stops.

tree is then used to draw a light blue line between the first token of each indented line (e.g., `to` in Figure 7.8) and its parent token in an earlier line (e.g., `from`).

Since indentation is essential to the interpretation of multi-line patterns, the pattern pane includes several features that make it easier to manage indentation. First, the Tab key does smart indentation, stepping through a set of adaptive tab stops that are determined by the possible parent tokens on earlier lines. Figure 7.9 shows an example. Most tokens produce one tab stop aligned with the start of the token, but tokens that are part of a multi-token expression, such as `from-to`, also produce a tab stop that would align the expected matching token, in this case `to`, with the end of the `from` token. To the right of the adaptive tab stops, the Tab key reverts to a fixed tab stop every 4 spaces. Shift-Tab can be used to move backwards through tab stops. If a group of lines are selected, then Tab and Shift-Tab adjust the indentation of the entire group, following the adaptive tab stops defined for the first line in the group.

In order to preserve expression structure when the user inserts or deletes pattern elements, the pattern pane automatically adjusts the indentation of lines below and to the right of the cursor. In particular, when an insertion or deletion changes the column position of a token, the pattern editor automatically reindents the token's children in the token tree in order to preserve their relative column positions. Figure 7.10 shows an example of automatic reindentation. Automatic reindentation makes TC indentation notation less *viscous* (Section 6.1) by allowing the user to make small changes to a pattern without having to fix the indentation manually.

The pattern pane provides a few other controls in addition to the pattern editor (Figure 7.8). The Go button runs the pattern, the same as pressing Enter. The Clear button clears the pattern editor. Finally, the large arrow on the right side of the pattern editor displays a drop-down history

<pre> from word starting "s" just after link to end of bullet just after "." </pre>	<pre> from allcaps word starting "s" just after link just after "." </pre> <p style="text-align: center; margin: 0;"><i>pushed over automatically</i> →</p>
--	---

Figure 7.10: Automatic subexpression reindentation preserves expression structure during pattern editing.

menu showing recently-executed patterns, most recent first. Choosing a pattern from the history menu loads it into the pattern editor, where it can be edited if desired and then executed again.

The pattern pane is also synchronized with the library pane, in the following sense. When the user types a (valid) pattern identifier into the pattern pane and runs it, the identifier becomes selected in the library pane, to show the user where the identifier is located in the library. Name-space categories are opened and the library pane is scrolled as necessary to make the selected identifier visible. Conversely, when the user clicks on an identifier in the library pane, the identifier is loaded into the pattern pane, to show the user how to obtain the same selection using a pattern. If the identifier alone would be ambiguous, then the pattern pane displays the shortest unambiguous suffix of the identifier's canonical name. For example, if the library contains both `Programming.Verilog.State` and `Business.Address.State`, and the user clicks on the former identifier in the library pane, then the pattern pane would display `Verilog.State`, which is sufficiently long to disambiguate it from the other pattern named `State`.

7.5 Editing with Multiple Selections

Once multiple selections have been made, editing with them is a straightforward extension of single-selection editing. Typing a sequence of characters replaces every selection with the typed sequence. Pressing Backspace or Delete deletes all the selections if the multiple selection includes at least one nonzero-length region. If the selection is all insertion points, then Backspace or Delete deletes the character before or after each insertion point. Other editing commands, such as capitalization changes, are applied to each selection separately. If LAPIS were capable of editing rendered HTML as well as plain text, it would also include editing commands for changing paragraph styles and character styles that would be extended to multiple selections in the same way.

Clipboard operations are slightly more complicated. Cutting or copying a multiple selection puts a list of strings on the clipboard, one for each selection, in document order. If the clipboard is subsequently pasted back to a multiple selection with the same number of regions, then each string in the clipboard list replaces the corresponding target selection. If the target selection has more or fewer regions than the copied selection, then the paste operation is generally prevented, and a dialog box pops up to explain why. Exceptions to this rule occur when the source or the target is a single selection. When the source is a single selection, it can be pasted to any number of targets by replication. When the target is a single selection, the strings on the clipboard are pasted one after another, each terminated by a line break, and an insertion point is placed after each pasted string. Line breaks were chosen as a reasonable default delimiter. The user can delete the line breaks and replace them with a different delimiter using the new multiple selection.

For comparison, Microsoft Word XP supports a limited form of multiple-selection editing. Multiple selections can be created either with the mouse, by holding down Control while clicking and dragging, or by searching for a pattern, which can be either a literal string or a regular expression. Once multiple selections are created, the user can delete or apply a style command to all the selections. Multiple selections in Word XP do not act as insertion points for typing, however, nor can they be copied and pasted with the clipboard.

In LAPIS, if the selection contains nested or overlapping regions, those regions are flattened together before an editing command is applied to them. This design decision was taken for the same reason that mouse selection can only produce flat region sets — to avoid presenting a confusing editing model to the user. Editing with multiple nested selections may be useful for some tasks, particularly transformations on source code or XML, but the details and evaluation are left for future work.

Editing is not the only use that can be made of the selection in LAPIS. Chapter 8 describes some text-processing tools, such as Keep and Sort, that act on the current selection. Unlike editing commands, these tools are not limited to plain-text mode, but can also be used in the rendered HTML view.

7.6 User Study

After the initial design and implementation of the TC pattern language described in Chapter 6 and the LAPIS user interface described in this chapter, a small user study was conducted to evaluate their usability. The user study was designed as a formative evaluation, with the primary goal of discovering usability issues and resolving them to improve the language and user interface. Formative evaluations tend to answer qualitative questions like “Can a user do this kind of task using this feature?” and “What kinds of errors are commonly made?” Formative evaluations generally do not answer questions like “Is technique X more usable than technique Y?” More quantitative evaluations have been performed on some aspects of LAPIS, particularly selection inference (Chapter 9) and outlier finding (Chapter 10). Discussion of these other evaluations is postponed until those chapters.

7.6.1 Goals

The user study was primarily a test of the TC pattern language, since TC is the most complex piece of the user interface. Two questions are of chief interest:

- **Generation.** Given a region set in a document, can users write a TC pattern that matches it?
- **Comprehension.** Given a TC pattern, can users predict which region set it would match?

These questions guided the overall design of the experiment, which was split into two parts: generation and comprehension. In the generation part, the user wrote TC patterns to create certain selections in a document displayed by LAPIS. In the comprehension part, the user used a highlighting marker to make selections on paper. The tasks in each part were carefully chosen to test various features of the TC pattern language, and in particular to expose several hypothesized usability problems:

- **Literals.** Do users remember to quote literal strings?
- **Selecting attributes vs. whole objects.** One difference between TC queries and other kinds of queries is that a TC pattern specifies not only search constraints (e.g. `contains "Reynolds", or not in Bold`) but also the space of text objects that should be searched (e.g. `Word`, or `Line`, or maybe `CourseTitle`). In other query systems, the query only specifies search constraints, and the object space is either fixed or specified some other way (e.g., by choosing a collection or database). For example, a web search engine searches for web pages, and a library catalog searches for books. Queries to these systems need not specify *what* the user wants to retrieve. TC patterns may return only certain attributes of the objects being searched — e.g., the titles of books matching the search constraints. Does this distinction cause confusion and erroneous patterns?
- **Operator names.** Do users use the relational operators correctly? Are any relational operators confused with each other? What other operator synonyms are suggested by common user mistakes?
- **Indentation.** Can users use indentation to structure complex expressions correctly?
- **Booleans.** Does the TC approach to Boolean operators (omitting `and` and encouraging an explicit `either` with each `or`) help users avoid common mistakes when formulating Boolean queries?
- **Abstractions.** Can users name patterns and use the named abstractions later? Do users name abstractions spontaneously, without being instructed to do so?

For reasons of time, the study did not comprehensively test *all* features of the TC language — only features that typical users would be likely to need during interactive use of LAPIS. In particular, advanced features like namespaces (Section 6.2.7), visible and hidden identifiers (Section 6.2.6), regular expressions (Section 6.2.10), and changing background (Section 6.2.14) were not tested by the study.

The study also did not ask users to generate or comprehend nested or overlapping selections, because those aspects of the LAPIS user interface are somewhat raw. Nevertheless, the study did attempt to test users' understanding of the underlying region set *model* that permits combinations of overlapping and nested regions:

- **Arbitrary region set model.** Can users comprehend patterns that combine two region sets that are not hierarchically compatible, e.g., lines and sentences?

In addition to addressing these questions about TC, the study also exercised other components of the LAPIS interface:

- **Highlighting.** Do users understand what region set is highlighted (particularly when some highlights are adjacent, as in Figure 7.2)?
- **Mouse selection.** Can users make multiple selections using the mouse? What typical errors are made? Do users combine mouse selection with pattern matching to make a difficult selection?

T1.	Highlighting
T2.	Patterns
T3.	Mouse Selection
T4.	Named Patterns
T5.	Naming a Pattern
T6.	Constraints
T7.	More Constraints
T8.	Either/Or
T9.	Not
T10.	Then
T11.	Counting
T12.	Summary

Figure 7.11: Tutorial sections.

- **Library pane.** Can users use the library pane to explore the pattern library and discover and use built-in patterns?
- **Pattern pane.** Can users use the pattern pane to enter, edit, and run TC patterns? What kinds of errors are made?

Multiple-selection editing was not tested in this user study, because it had not yet been implemented. Editing was tested in the later studies described in Chapters 9 and 10.

7.6.2 Procedure

The experiment consisted of four parts: pretest questionnaire, tutorial, generation, and comprehension.

The pretest questionnaire asked the user to evaluate his or her own computer experience in a number of areas: web browsing, word processing, search-and-replace, regular expressions, web search engines, and general computer programming. Users were also asked whether English was their native language, since TC uses English keywords and an English-like syntax.

The tutorial was a sequence of 12 web pages displayed in LAPIS, each of which described one feature of LAPIS or the TC pattern language. An outline of the tutorial is shown in Figure 7.11; the tutorial itself is included with the LAPIS distribution. Each page of the tutorial included one or more examples demonstrating the feature. Since the user viewed the tutorial inside LAPIS, the user was asked to try all such examples using the LAPIS user interface — for instance, typing example patterns into the pattern editor and running them to see what selection was made. The tutorial was designed to take about 30 minutes to complete. Afterwards, the user was provided with a printed copy of the tutorial, including a two-page summary of TC operators, which the user could consult as necessary during the remainder of the experiment.

The generation section consisted of 15 tasks. For each task, the user was asked to make a certain selection in a web page displayed in LAPIS. All tasks used the same web page, a short list of CMU computer science courses. Each task showed the desired selection marked in highlight marker on a printed copy of the web page, along with a brief English description of the selection.

An example of a generation task is shown in Figure 7.12. The English descriptions of all 15 tasks are shown in Figure 7.13.

As observed by Pane and Myers [PRM01], there is some danger that describing tasks in English will bias the test by inadvertently suggesting the correct answer. Nevertheless, some kind of intentional description of the desired selection seems unavoidable, particularly for complex Boolean queries. The English descriptions were carefully worded to avoid using TC operator names and library pattern names, instead using nouns and verbs appropriate to the semantic domain. For example, the description of task G5 uses the phrase “taught by Moore” instead of “*containing* Moore”, which would be too suggestive of the correct TC operator. Similarly, G13 avoids use of the word “not”. Some task descriptions must still use words that correspond to TC operators, but only when the meaning of the word in the description differs from its meaning in TC, so that a naive translation from English to TC would actually lead *away* from the correct pattern. For example, task G8 uses the phrase “in Wean Hall” to describe the location of the class, not of the desired selection. Naively translating this English phrase to the TC expression `in "WeH"` would not give the correct result; instead, the user must say *contains* "WeH". Similarly, G12 uses “start at 10:30” in a sense different from the TC operator *starting*.

LAPIS provides several ways to make selections, and it was desirable to have users try all the methods in order to discover their usability problems. As a result, some of the tasks specify that the user should use a particular method to make a selection. For example, G1 must be done by writing a pattern, G2 using the mouse, and G3 by picking a pattern from the library. Some of the more challenging tasks (G10-G13) must be done by writing patterns, in order to force the user to try to write complex patterns. For some tasks, however, users were allowed to make the selection however they liked (G6, G7, G9, G14), in order to see which techniques users would prefer, and in particular whether users would combine pattern matching with mouse selection on the same task.

While the user was interacting with LAPIS, both in the tutorial section and in the generation section, the system recorded a transcript of selection-related user interface actions: TC patterns that were run, mouse selections that were made, and library names that were clicked on. Users were urged to “think aloud” as much as possible. The experimenter took notes about the user’s verbal comments and the context in which they were uttered, but no audio or video recordings were made.

After the generation section came the comprehension section, which consisted of 10 tasks presented and performed entirely on paper. Each task showed a TC pattern, indented and augmented with tie lines as it would be displayed in the LAPIS pattern editor, and a printed copy of a web page. The user was asked to use a green highlighting marker to indicate what selection would be made in the web page if the given pattern were entered into LAPIS.

The 10 patterns used in the comprehension section are shown in Figure 7.14. Most of the tasks used the same web page as in the generation section, the listing of CMU computer science courses (Figure 7.15). Tasks C7 and C8 used a different web page, consisting of the first 8 sentences of the Gettysburg Address ((Figure 7.16). A different page was used for these two tasks so that the patterns could refer to two region sets which are not hierarchically compatible, specifically lines and sentences. Users were given a key that defined the named patterns used in the tasks (`Units`, `Row`, `StartTime`, `Line`, and `Sentence`) by showing the region set selected by each pattern.

The comprehension tasks were designed so that different ways of interpreting the pattern would result in different answers. For example, there are several plausible interpretations of C4, depending on how the constraint `starting "1"` is applied. The correct interpretation applies this

Use the **mouse** to make this highlight

15-712	Advanced Operating Systems	Gibson	MW	10:00-11:20	WeH 5409A&B	12
15-750	Algorithms	Blum/Sleator	TR	1:30-2:50	WeH 5409A&B	12
15-780/16-731	Advanced AI Concepts	Moore	TR	10:30-11:50	WeH 5409A&B	12
20-602/15-802	Statistical Approaches to Learning	Fienberg/Lafferty	TR	12:00-1:20	PH A22	12
15-810	Verification of Real-Time Programs	Clarke	M	12:30-1:50	WeH 4601	6
15-819	Typed Compilation	Crary	MW	1:30-2:50	WeH 5409B	12
15-819	Denotational Semantics of Types	Reynolds	TR	10:30-11:50	WeH 4601	12
15-812	Semantics of Programming Languages	Brookes	MWF	10:00-10:50	WeH 4615A	12
80-713	Category Theory	Awodey	TR	1:30-2:50		12
15-822	System Design and Implementation	Satya	T	3:00-5:50	WeH 8220	12
15-824	Mobile and Wireless Networking	Johnson	TR	10:30-11:50	WeH 4615A	12
15-839	What Makes Good Research?	Shaw	MW	1:30-2:50	WeH 5409A	12
15-840	Research Issues in Computer Systems	Steenkiste	M	12:00-1:20	WeH 7220	2
15-854	Approximation and On-Line Algorithms	Blum	MW	1:30-2:50	WeH 4615A	12
11-741	Information Retrieval	Yang/Callan	TR	1:30-2:50		12
15-882	Introduction to Artificial Neural Networks	Touretzky	MW	3:00-4:20	WeH 4615A	12
15-887	Planning, Execution, and Learning	Simmons	TR	1:00-2:20	WeH 4601	12

Figure 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.

- G1. Please use a **pattern** to highlight all the occurrences of **WeH**
- G2. Use the **mouse** to make this highlight
- G3. Use a **built-in name** to make this highlight
- G4. (1) Use a **built-in name** to make this highlight; then,
(2) Name the highlight **Course**.
- G5. Use only **patterns** to highlight **all the courses taught by Moore**.
- G6. (1) Make this highlight however you like; then,
(2) Name the highlight **StartTime**
- G7. (1) Highlight **the course titles** however you like; then,
(2) Name the highlight **Title**
- G8. Use only **patterns** to highlight **all the titles of courses taught in Wean Hall (WeH)**
- G9. Highlight **all the 12-unit courses** however you like
- G10. Use only **patterns** to highlight **all the course numbers in Department 15**
- G11. Use only **patterns** to highlight **all the 12-unit courses that meet in Wean Hall (WeH)**
- G12. Use only **patterns** to highlight **all the courses that start at 10:30 or else meet in WeH 4601**
- G13. Use a **pattern** to highlight **all the courses taught by anybody other than Clarke**
- G14. Highlight **all the course numbers** however you like
- G15. Use only **patterns** to highlight **all the courses meeting in WeH 5409 starting at 10:30 or 1:30**

Figure 7.13: Generation tasks.

- C1. "WeH" just before "4601"
- C2. "WeH" anywhere before "8220"
- C3. "4601" anywhere before last "4615"
- C4. Units in Row contains StartTime
 └─ starting "1"
- C5. Row either contains "10:30"
 └─ or contains "Semantics"
 └─ contains "4615"
- C6. Row either contains "10:30"
 └─ contains "4615"
 └─ or contains "Semantics"
- C7. Line in Sentence
- C8. Sentence starting Line
- C9. Row not containing "10:"
 └─ containing ":50"
 └─ containing "WeH"
- C10. Row not containing "WeH"
 └─ not containing "MW"

Figure 7.14: Comprehension tasks.

Highlight **all matches** to this pattern:

"WeH" anywhere before "8220"

15-712	Advanced Operating Systems	Gibson	MW	10:00-11:20	WeH 5409A&B	12
15-750	Algorithms	Blum/Sleator	TR	1:30-2:50	WeH 5409A&B	12
15-780/16-731	Advanced AI Concepts	Moore	TR	10:30-11:50	WeH 5409A&B	12
20-602/15-802	Statistical Approaches to Learning	Fienberg/Lafferty	TR	12:00-1:20	PH A22	12
15-810	Verification of Real-Time Programs	Clarke	M	12:30-1:50	WeH 4601	6
15-819	Typed Compilation	Crary	MW	1:30-2:50	WeH 5409B	12
15-819	Denotational Semantics of Types	Reynolds	TR	10:30-11:50	WeH 4601	12
15-812	Semantics of Programming Languages	Brookes	MWF	10:00-10:50	WeH 4615A	12
80-713	Category Theory	Awodey	TR	1:30-2:50		12
15-822	System Design and Implementation	Satya	T	3:00-5:50	WeH 8220	12
15-824	Mobile and Wireless Networking	Johnson	TR	10:30-11:50	WeH 4615A	12
15-839	What Makes Good Research?	Shaw	MW	1:30-2:50	WeH 5409A	12
15-840	Research Issues in Computer Systems	Steenkiste	M	12:00-1:20	WeH 7220	2
15-854	Approximation and On-Line Algorithms	Blum	MW	1:30-2:50	WeH 4615A	12
11-741	Information Retrieval	Yang/Callan	TR	1:30-2:50		12
15-882	Introduction to Artificial Neural Networks	Touretzky	MW	3:00-4:20	WeH 4615A	12
15-887	Planning, Execution, and Learning	Simmons	TR	1:00-2:20	WeH 4601	12

Figure 7.15: Example of the first kind of comprehension task. All but tasks C7 and C8 used this web page.

Highlight **all matches** to this pattern:

Line in Sentence

Four score and seven years ago our fathers brought forth on this continent a new nation, conceived in liberty and dedicated to the proposition that all men are created equal. Now we are engaged in a great civil war, testing whether that nation or any nation so conceived and so dedicated can long endure.

We are met on a great battlefield of that war. We have come to dedicate a portion of that field as a final resting-place for those who here gave their lives that that nation might live. It is altogether fitting and proper that we should do this.

But in a larger sense, we cannot dedicate, we cannot consecrate, we cannot hallow this ground. The brave men, living and dead who struggled here have consecrated it far above our poor power to add or detract. The world will little note nor long remember what we say here, but it can never forget what they did here.

Figure 7.16: Example of the second kind of comprehension task. Tasks C7 and C8 used this web page.

constraint to Row, in which case the user would highlight the units column in all rows that start with "1". There are two other plausible interpretations of this pattern, however: that the Units should start with "1", or that the StartTime should start with "1". Each of these interpretations produces a different highlight, so that particular misinterpretations can be identified by looking at the user's answer.

For complex patterns involving literal strings (e.g., C5, C6, C9, C10), there is some danger that the user will make errors not because of pattern misinterpretation, but merely because the user has overlooked some occurrence of a literal string. To distinguish between these kinds of errors, and to encourage the user to be methodical, the user was asked to circle all occurrences of literal strings in the document (using a pencil or ballpoint pen instead of the highlight marker). In retrospect, it would have been better if the printed document for each task simply made all occurrences of the relevant literal strings obvious, perhaps by printing them in contrasting colors. Then the test would focus more on pattern comprehension than on the user's ability to scan text for string matches. Fortunately, no users' answers were uninterpretable due to overlooked occurrences of a literal string.

The study was run with 3 unpaid pilot users followed by 6 paid users. The paid users were found by advertising on campus newsgroups (`cmu.misc.jobs` and `cmu.misc.market`) for subjects with at least some word-processing and web-browsing experience. Paid users received \$12 for about 90 minutes of work. Observations of both pilot users and paid users are combined in the discussion in the next section. From their answers to the pretest questionnaire, all users had substantial experience in web browsing and word processing, and a range of experience in programming: 6 users with "lots" of programming experience, 2 users with "some" experience, and 1 user with "little" experience. Only 2 of the 9 users were non-native speakers of English.

Since the user study was intended as a *formative* evaluation, to guide further development of LAPIS, some usability problems were addressed by making changes to LAPIS during the study, in hopes that observation of the remaining users would show whether the changes had the desired effect. More changes to LAPIS were made after the conclusion of the study. Both kinds of changes are discussed in the next section.

7.6.3 Results

The discussion of the study results is organized around the goals identified earlier (Section 7.6.1).

Literals

Users neglect to quote literal strings in TC patterns. Here are some examples of patterns generated by users that omitted quotes from a literal string (the code in parentheses is the generation task the user was trying to solve):

```
WeH (G1)
Row starting 15-780 (G5)
Title in Row containing WeH (G8)
Title just before Gibson (G8)
7th Cell in Course not containing 12 (G9)
```

This problem has been noted by authors of other programming languages [Bru97]. The solution used for TC is automatic single-word quoting, which was described in detail in Section 6.2.9. Automatic quoting was one of the changes introduced after the first 5 users. The tutorial was not changed, however, and subsequent users were not told that automatic quoting would be available. Nevertheless, automatic quoting successfully prevented errors for the remaining users, two of whom solved task G1 using the quoteless pattern WeH.

Object Space

Two tasks were designed to test whether users could generate and comprehend patterns in which the search constraints (predicates that control whether a region was selected) were distinct from the object space (the regions actually selected).

Generation task G8 asked the user to write a pattern selecting the titles (the object space) of all courses taught in Wean Hall (the search constraint). One correct pattern is

```
Title in Course containing "WeH"
```

where `Title` and `Course` were named by the user in earlier generation tasks. One might expect users who misunderstand this aspect of TC to write a pattern that puts `Course` or `"WeH"` as the first component. Seven out of nine users successfully generated a correct pattern on the first try. One of the remaining users started with

```
title just before "WeH"
```

which is incorrect because the course title is not adjacent to the building in each row, but nevertheless identifies the correct object space.

The last user started with

```
WeH in title
```

which seems to indicate an object space problem. This user continued experimenting with other patterns, including `Title before "WeH"`, `"WeH" in Course`, and `Title containing "WeH"`. The user's think-aloud comments suggested that he/she was looking for a way to "correlate fields within a course" and "look for WeH outside of this set [the set of titles]". The user finally achieved the desired selection by first naming the set of courses that meet in Wean Hall:

```
pippo is Course containing "WeH"
```

and then using this named pattern to select the titles of the set:

```
2nd Cell in pippo
```

The other task that tested the object space concept was comprehension task C4, which asked the user to highlight the units column (the object space) in rows that start with the digit "1" (the search constraint). Although not all users highlighted the *correct* set of units, all nine users highlighted only units — not rows or start times or the digit "1". This suggests that they correctly identified the object space for this pattern.

Operator	Suggested synonyms	
	Typed	Aloud
in	of, in each	belonging to, within
contains	containg	whose, including
equals	is	
just before		right before
starts	begin	starts with
then	and	extending to more stuff

Figure 7.17: Synonyms for TC operators suggested by users.

Operator Names

The most interesting discovery about operator names was users' tendency to misspell containing. Three of the first 7 users misspelled the keyword at some point, two of them writing `containg` and the third `containging`. After these errors, LAPIS was changed so that the official keyword described in the tutorial was `contains`, with `containing` accepted as a synonym, without any mention of this fact in the tutorial. Subsequent users did use the official keyword `contains` (27 out of 47 occurrences), but also `containing` (19/47) and `containg` again (1/47). After the study, LAPIS was further changed to permit `containg` as yet another synonym.

The generation tasks offered an opportunity to discover other synonyms that users might find natural, shown in Figure 7.17. Some of these synonyms were typed in as part of attempted patterns; others were uttered verbally in the user's think-aloud protocol. Except for `containg`, each of these synonyms was suggested by only one user. Several of these synonyms were already available in TC, such as `begin` for `starts`, and `right before` for `just before`. Others were added to TC as a result of the study, particularly `containg`. (The current list of synonyms can be found in Figure 6.6.) Other suggested synonyms were deemed too vague or ambiguous, particularly `of` and `whose`. It is interesting to note that `and` was proposed as a synonym for `then`, confirming previous studies that `and` is commonly used to describe sequencing in natural language descriptions of programming tasks [PRM01].

Several comprehension tasks were designed to test potential confusions between operators. In particular, tasks C1 and C2 test whether the user can distinguish between `just before` and `anywhere before`. All users correctly answered both tasks.

Indentation

Indentation is used to structure TC expressions, instead of explicit parentheses. The most telling test of indentation rules was comprehension task C4, which asked users to interpret the pattern:

```
Units in Row contains StartTime
    └─starting "1"
```

The key question in this pattern is where the constraint `starting "1"` should be applied. Three plausible interpretations are possible:

- the Row must start with 1 (the correct interpretation under indentation rules)

- the `StartTime` must start with 1
- the `Units` must start with 1

Each of these interpretations produces a different answer. In the study, 5 out of 9 users chose the correct `Row` interpretation, 4 out of 9 users chose the `StartTime` interpretation, and no users chose the `Units` interpretation. Evidently, indentation is not strong enough to overcome users' tendency to read from left to right.

In generation tasks, two users made errors by expecting TC operators to follow "natural" precedence and associativity rules. Some examples of these incorrect patterns follow:

```
2nd Cell in course not starting "Statistical"
7th Cell in Course not containing "12"
row ending "12" containing "WeH"
```

In all three patterns, the last operator on the line was meant to apply to the first token on the line, which must be done by indentation. The user who wrote the first two patterns never got the hang of using indentation for Boolean `and`, adopting the strategy of naming all subexpressions instead to avoid building complex patterns. The user who wrote the third pattern realized what was wrong in a few seconds and added the indentation needed to correct the pattern.

Because of indentation structuring, newlines are significant in TC patterns. Two users failed to understand this fact and generated erroneous patterns with embedded newlines but no indentation, such as:

```
row either containing "30"
or containing "WeH 4601"

Digits then "-" then Digits in
1st Cell in Row
```

Although indentation structuring has mixed results as far as eliminating precedence errors and comprehension errors are concerned, users were nevertheless largely able to use indentation to structure expressions. For the generation tasks G11, G12, and G15, each of which involved multiple relational operators and at least one Boolean operator (two in the case of G15), a majority of users structured their patterns using indentation: 7 of 9 users for G11, 6 of 9 users for G12, and 6 of 9 users for G15. The remaining users managed to write only single-line patterns for these tasks, which sometimes required naming subexpressions.

Booleans

Several comprehension tasks tested whether users could correctly interpret Boolean combinations of constraints. For tasks C5 and C6, which combined `and` with `or` in the same expression, the results were positive: 8 out of 9 users interpreted both these patterns correctly. The ninth user (the same user in both cases) apparently interchanged the meanings of `and` and `or`. For example, the logical formula represented by task C5 should be

```
(contains 10:30 or contains Semantics) and contains 4615
```

but this user interpreted it as

```
(contains 10:30 and contains Semantics) or contains 4615
```

The user did exactly the same interchange on task C6.

For task C9, which involved a Boolean `and` in the scope of a `not` operator, the results were more mixed. Only 4 out of 9 users interpreted this pattern correctly, as

```
not (contains 10: and contains :50) and contains WeH
```

The remaining 5 users distributed the `not` incorrectly:

```
(not contains 10:) and (not contains :50) and contains WeH
```

In the generation tasks, users were largely able to produce Boolean expressions using indentation, as described in the previous subsection. Several users complained about the absence of `and` from the language, however. As a result of these observations, `and` was added to TC, with a warning about its ambiguity (Section 6.2.15).

Abstractions

Generation tasks G4, G6, and G7 tested whether users could assign names to patterns (`Course`, `Title`, and `StartTime`) and then use the named abstractions in later patterns. All nine users were able to name patterns; 8 of 9 used the `Add` button to name all patterns interactively, while the ninth user used the `is` operator for all naming. Almost all users, 8 of 9, reused `Course` and `Title` in later patterns. The remaining user never referred to any of these named abstractions, instead treating each task as an independent problem to be solved.

Four users assigned names to patterns spontaneously, without being requested to do so by the task. Three of these users used naming to express a complex pattern as a sequence of simpler patterns. For example, one user solved task G11 by first assigning the name `12units` to the 12-unit courses, then writing the pattern `12units containing "WeH"` to constrain it to the ones that meet in Wean Hall.

Two users spontaneously assigned names to natural abstractions in the data, such as `Units`, `Teacher`, and `Coursenumber`, even when these abstractions were not strictly necessary to solve the task. It is interesting to note that only one of these users was a programmer; the other described his or her programming experience as “little”.

Region Set Model

In general, most users were able to conceptualize overlapping sentence and line regions and determine relationships among them. Comprehension tasks C7 and C8 tested this ability with TC patterns involving the nonhierarchically-compatible region sets `Line` and `Sentence`. Most users, 7 out of 9, correctly interpreted task C7, `Line in Sentence`. One user misinterpreted it to mean essentially `Line overlaps Sentence`, i.e., that every line which included some part of a sentence should be highlighted, and proceeded to highlight every line in the document. The remaining user’s answer is simply wrong; no simple hypothesis can explain the user’s mental model.

For task C8, 5 out of 9 users gave the correct interpretation. The other 4 users misinterpreted the pattern in the same way, as if `starting` implied not only coincident start points but also containment — an understandable model, since the only examples of `starting` in the tutorial happened to imply containment.

Highlighting

Generation task G2 (reproduced in Figure 7.12) specifically tested whether the LAPIS highlighting techniques were sufficient to delimit flat region sets. G2 includes examples of both problems mentioned in Section 7.3.1: two adjacent regions that must be separately highlight (M and W), and two lines that form a single region (the two fully-selected rows). All users understood and correctly reproduced the selection in G2, so flat region highlighting can be deemed a success.

On two occasions, however, users inadvertently created overlapping region sets. One user ran the pattern `Word then Word`, which produces an overlapping set, and was confused by the resulting highlight. Another user inadvertently made overlapping regions while selecting with the mouse, but failed to notice the resulting unusual highlight. More design is needed to properly visualize nested and overlapping region sets, as discussed above (Section 7.3.3).

Mouse Selection

Generation task G2 also tested whether users could use the mouse to select multiple selections, and indeed all users could.

In the version of LAPIS used in the study, the handles at each end of a selected region were interactive, so that the user could click and drag them to resize the region. Although one user took advantage of this feature to resize a region, the interactive handles caused errors for other users. Three users accidentally clicked and dragged on handles when they were trying to make a selection adjacent to a selected region. After the study, LAPIS was changed so that the handles are no longer interactive.

Only task G2 explicitly required users to use mouse selection, but users sometimes resorted to mouse selection to accomplish other generation tasks. The most troublesome tasks were G7 and G14. On task G7, three users used mouse selection to make the entire selection; the remaining users were able to come up with a pattern. On task G14, two users made the entire selection manually, three users made the entire selection with a pattern, but four users first wrote an almost-correct pattern, then made a few mouse selections to fix it up. Thus users can and do combine pattern matching with mouse selection to accomplish tasks like these.

Library Pane

Two generation tasks, G3 and G4, requested that the user find a built-in named pattern corresponding to the desired selection. All users solved G3 immediately, probably because the answer (`Business.Time`) was usually still visible in the library pane as a result of a tutorial example that used `Business.Address.State`. Task G4, however, required users to poke around in the library hierarchy, searching for some pattern that selects the rows of the table. Eight of the 9 users successfully found it under `Layout.Table.Row`. The remaining user clicked on `Table` several times, which selected the entire table, but failed to notice that `Table` could be expanded

further and hence never saw the pattern `Row`. This user did not realize that some names in the library can be simultaneously *patterns* that produce a selection and *namespaces* that group together other patterns. In the terminology presented earlier, `Table` is a bound namespace. After the study, to help address this problem, the library pane was changed so that clicking on a bound namespace not only runs its pattern, but also expands its children, so that the user can see that it has both kinds of behavior at the same time.

Interestingly, one user did not bother exploring the library pane to find named patterns. Instead, this user simply typed a name that seemed sensible: `time` for G3, and `row` for G4. This strategy succeeded, and LAPIS not only made the correct selection, but also showed the user where the name was actually defined in the library's hierarchical namespace.

Pattern Editor

Users made no errors with the pattern editor, apart from the indentation and newline problems discussed earlier, which can be attributed more to the underlying TC language than to the editing environment. As described earlier, however, several users objected to the use of Control-Enter to run the pattern, so LAPIS was changed after the study so that pressing Enter runs the pattern instead.

7.6.4 Summary

Overall, the user study identified a number of usability issues in TC and LAPIS, some of which were addressed in subsequent redesign of features. The following changes were motivated by observations in the user study:

- **Forgetting to quote literals.** TC now has automatic single-word quoting.
- **Misspelling “containing”.** The official keyword is now `contains`, and common misspellings are accepted as synonyms.
- **Inadvertently clicking on handles.** The handles of a selection are no longer interactive.
- **Inadvertently selecting overlapping regions.** Mouse selection now selects only flat regions.
- **Expert users want `and`.** TC now supports the `and` keyword, with a warning that can be toggled off.
- **Bound namespaces don't look expandable in the library pane.** Clicking on a bound namespace in the library pane now does two things: runs its pattern, and expands it to show its children.

The user study also confirmed that precedence, expression grouping, and Boolean logic are trouble spots in programming languages, and TC's novel features (indentation structure, eliminating `and`) seem to help, but are not a panacea for these problems.