

LAPIS: Smart Editing with Text Structure

Robert C. Miller and Brad A. Myers

School of Computer Science
Carnegie Mellon University
{rcm,bam}@cs.cmu.edu

Keywords

programming-by-demonstration, PBD, automated text editing, pattern matching, search-and-replace, LAPIS

INTRODUCTION

Text editing is full of small repetitive tasks. Users have a basket of tools for automating such tasks, including search-and-replace, keyboard macros, and text-processing languages like Perl or awk. Among the drawbacks of existing tools are too little power, too little domain specificity, and too-steep learning curves. Automated text editing can also be dangerous; witness the reckless applications of global search-and-replace featured in *comp.risks*, among them “eLabourated” in a news report about the British government and “arjpgicial turf” on a web site that evidently switched from TIFF to JPEG. Existing automation tools do nothing to help the user catch these kinds of errors.

We have developed a system for automated text editing, LAPIS, that is based on the idea of *lightweight structure*, the ability to recognize text structure automatically using an extensible library of patterns and parsers. The current LAPIS library includes parsers for HTML, Java, plain text layout, addresses, phone numbers, etc. Extending the library is as easy as writing a pattern and giving it a name — actually easier, since LAPIS can also infer patterns from examples.

Lightweight structure enables four novel features in LAPIS: (1) *text constraints*, a new text pattern matching language based on lightweight structure; (2) *selection guessing*, where the user gives positive and negative examples and the system infers a pattern describing it; (3) *simultaneous editing*, where the user controls multiple cursors to do a group of repetitive edits simultaneously; and (4) *outlier finding*, where the system draws attention to the most unusual elements in a selection or pattern match, in case some of them are errors. These features combine to create a powerful environment for automated text editing.

TEXT CONSTRAINTS

All text editors have a search-and-replace command that searches for a string pattern and replaces it with another

```
paint (rectangle, 0, 0);
for (int i = 0; i < circle.length; ++i) {
  paint (circle[i], x, y);
  paint (circle[i].center, x + 2, y + 2);
}

rectangle.paint (0, 0);
for (int i = 0; i < circle.length; ++i) {
  circle[i].paint (x, y);
  circle[i].center.paint (x + 2, y + 2);
}
```

Figure 1: Simultaneous editing in action. (a) User selects “rectangle”; system infers the pattern “first ActualParameter” to make other selections. (b) User cuts and pastes all selections simultaneously.

string. Some editors also support regular expressions, but the learning curve can be intimidating, and some patterns are impossible to express with low-level regular expressions. LAPIS has a new pattern language called *text constraints* that addresses these problems [4]. Text constraints combine literal string searches, lightweight structure from the library, and relations like *before*, *after*, *in*, and *contains*, to produce simple but powerful patterns:

- “-” in PhoneNumber
- Link containing “My Yahoo”
- Java.Method containing Java.MethodName= “toString”
- last Word in Sentence

In LAPIS, text constraints are used not only for search-and-replace, but also to define new structure for the structure library, to automate web browsing, and to apply Unix-style tools like *grep* and *sort* to structured text [4].

SELECTION GUESSING

Any language has a learning curve, and text constraints is no exception. LAPIS can smooth out the curve by inferring a pattern from positive and negative examples provided by the user. The user gives examples by entering *selection guessing mode*. Multiple examples are given by making a multiple selection in the text editor, holding down the Control key to add selections (additional positive examples) or remove selections (negative examples). After each selection, the system suggests a pattern that matches the user’s selections and highlights other matches to the inferred pattern.

ational Conference on Information and Knowledge Management, Baltimore, 1995.
 6. Finin, T., Fritzson, R., McKay, D. and McEntire, R. KQML A Language and Protocol for Knowledge and Information Exchange. In Fuchi, K. and Yokoi, T., Eds. Knowledge Building and Knowledge Sharing. Ohmsha and IOS Press, 1994.
 7. Hayes-Roth, B., Pfeleger, K., Morignot, P. and Lalanda, P. Plans and Behavior in Intelligent Agents, Technical Report KSL-95-35, Stanford University, 1995.
 8. Kosbie, D.S. and Myers, B.A. A System-Wide Macro Facility Based on Aggregate Events: A Proposal. In Cypher, A., Ed. Watch What I Do: Programming by Demonstration. The MIT Press, Cambridge, Mass., 1993.

Figure 2: Red outlier highlighting draws attention to a selection error in simultaneous editing: only “Hayes” is selected instead of the full name “Hayes-Roth”.

Selection guessing uses lightweight structure in two ways. First, lightweight structure is used in the inferred pattern, so LAPIS can infer patterns over HTML and Java structure without having to infer an HTML or Java parser. Second, features derived from lightweight structure are used to rank the system’s guesses, so that fewer examples are needed to reach the right inference.

SIMULTANEOUS EDITING

Many repetitive tasks in text editing have a simple form: apply the same edits to a group of similar objects, such as lines, paragraphs, mailing addresses, function calls, or things less easily described. *Simultaneous editing* [5] is a new way to do these kinds of edits all at once. In simultaneous editing, the user first selects a set of regions to edit, called the *records*. This record set can be defined by a pattern, by selection guessing, or by manual multiple selection. After the record set is defined, the system enters a mode in which a selection in one record causes the system to infer a pattern that makes a similar selection in every other record. Subsequent editing operations — such as insert, delete, and cut-and-paste — affect all records simultaneously. Using simultaneous editing feels like controlling multiple text cursors with a single mouse and keyboard (Figure 1).

Like selection guessing, simultaneous editing must infer a pattern for each selection, so it is sometimes wrong. The user can correct an inference by holding down the Control key and making the desired selection. Fortunately, the combination of high-level lightweight structure with the requirement of exactly one selection per record constrains inference so much that most selections are correct after only the first example. In a user study of simultaneous editing, 85% of users’ selections required only one example, and each selection required only 1.25 examples on average [5]. The same study found that simultaneous editing was up to 8 times faster to use on average than another programming-by-demonstration system, DEED [2].

OUTLIER FINDING

The user study of simultaneous editing revealed that users tend to overlook selections that were slightly incorrect. *Outlier finding* [6] is a technique for finding selections which look unusual and highlighting them as possible errors. Outlier finding uses lightweight structure to find features that most selections have in common. Selections which lack those features are deemed outliers and highlighted specially

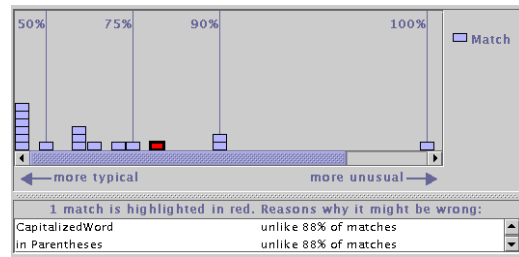


Figure 3: The Unusual Matches window finds possible errors in a pattern match.

to attract the user’s attention (Figure 2). When outliers were highlighted, users were more likely to notice and correct selection errors in simultaneous editing [6].

Outlier finding can also help users fix buggy patterns. The Unusual Matches window (Figure 3) displays a histogram of pattern matches on a linear scale of “weirdness”. Blocks lying alone at the right end of the histogram are outliers that might need the user’s attention. Clicking on a block highlights it in the text editor and explains what features make it unusual. The window can also show pattern *mismatches* on the same scale, so users can check for both false positives and false negatives. The Unusual Matches window can help prevent reckless uses of search-and-replace.

RELATED WORK

LAPIS is similar to other programming-by-demonstration systems for text editing, including TELS, Tourmaline, Cima, DEED, and SMARTedit [1, 2, 3]. LAPIS provides more feedback than other PBD systems, using text constraints, simultaneous editing, and outlier highlighting.

CONCLUSION

LAPIS offers a basket of new tools for automated text editing that demonstrate the usefulness of lightweight structure. LAPIS is freely available, including Java source code, from: <http://www.cs.cmu.edu/~rcm/lapis>

REFERENCES

1. A. Cypher, ed. *Watch What I Do: Programming by Demonstration*. MIT Press, 1993.
2. Y. Fujishima. Demonstrational automation of text editing tasks involving multiple focus points and conversions. In *Proc. IUI*, pp 101–108, Jan 1998.
3. H. Lieberman, ed. *Your Wish Is My Command: Giving Users the Power to Instruct Their Software*. Morgan Kaufmann, 2001.
4. R.C. Miller and B.A. Myers. Lightweight structured text processing. In *Proc. USENIX Tech. Conf.*, pp 131–144, June 1999.
5. R.C. Miller and B.A. Myers. Interactive simultaneous editing of multiple text regions. In *Proc. USENIX Tech. Conf.*, pp 161–174, June 2001.
6. R.C. Miller and B.A. Myers. Outlier finding: Focusing human attention on possible errors. In *Proc. UIST 2001*, pp 81–90, Nov 2001.