Structured Text Modification Using Guided Inference

Luke Church

Computer Laboratory
Cambridge University
luke@church.name

Alan F. Blackwell

Computer Laboratory
Cambridge University
Alan.Blackwell@cl.cam.ac.uk

Keywords: POP-I.C. end-user applications, POP-II.A. end-users, POP-IV.A. programming by example

Abstract

We describe a technique that allows end-users to specify automated transformations of structured text by inferring an underlying model. Inference is achieved with a novel algorithm, Structured Prediction by Partial Match (SPPM), a generalisation of the well-known PPM approach to predictive text entry and compression. We created two simple applications, as examples of "first steps" end-user programming tasks that can be achieved using SPPM. In empirical evaluations, each of these applications proved to be substantially superior to equivalent facilities in leading commercial products.

1. Introduction

Many digital text documents have regular structure. An unfortunate consequence for a large number of users is that editing highly structured texts can be boring and repetitive. But regular structure also brings the opportunity to automate repetitive operations by exploiting information redundancy. If a computational model can be constructed to describe that regularity, then information-efficient interaction techniques become possible. Such models of redundancy in text are proving valuable in user interfaces for text entry, where redundancy in the English language (which has information content of only around 1.3 bits per character) allows text to be entered efficiently using a predictive model. This then allows the designers of text entry interfaces to reduce the size of the keyboard (Grover et al. 1998), reduce the need for accuracy in striking the keys (Zhai & Kristensson 2003) or even replace the keyboard altogether (Ward et al. 2000).

We build on the predictive model inference technologies that are proving effective in text entry, to create a system for efficient modification of text. We have created a structured text interaction method that uses machine learning techniques to infer a model of the text structure, and as a result, makes repetitive editing far more efficient. The construction of the model, and the specification of the required repeated modifications, can be considered a kind of end-user programming, in which case the application of machine-learning methods offers a “programming by example” interaction paradigm.

We were initially inspired by the approach of Miller (2002), who developed an interaction technique for editing texts with a regular structure by using multiple edit-point cursors. If the same structure appeared at 100 places in a document, Miller’s Lapis system would display 100 cursors, and every user keystroke would simultaneously update the text at all 100 locations. In addition to being powerful, this technique promotes user confidence through directly visible and incremental feedback of every change. In the simplest case, the Lapis interaction style might be regarded as a novel alternative to search-and-replace, incorporating a preview of the results, and with single-click rather than sequential replacement. In more complex cases, Lapis could be used to specify and execute global structure transformations, of the kind supported by sophisticated programmers’ tools.

Structured editors like Lapis, and indeed most programming editors, rely on a grammar that specifies the language to be edited. Many programming editors can be customised to support different language models by changing the grammar. Lapis also supported a selection of different grammars. However most users do not deal with programming languages, but with texts that can only partially be described in terms of a formal grammar. Even in those portions of the text that can be described

formally, the repetitive structure in the text might well be an ad hoc invention of the user rather than a known language standard. If users are to interact efficiently with such texts, it is therefore necessary to help the user specify the grammar model, either with a guided grammar editor such as Lieberman's Grammex (1998), or a machine-learning approach such as Witten and Mo's TELS (1993).

Machine-learning approaches can either be deterministic (Blackwell 2001), in which case the user must accurately specify a training set as a basis for generalisation, or they can be based on statistical inference, in which case the system can anticipate which model structures are most likely, with guidance from the user. Our research focuses on this last method, of statistical rather than deterministic model inference.

In the remainder of this paper, we start by describing two simple demonstrator applications, in order to illustrate the basic interaction principles that we propose, showing how those principles apply in the context of a realistic end-user task. We then present a generalised view of the interaction approach that underlies both demonstrator applications. This approach is built around an interactive guided learning technique that assists end-users in constructing models of local repeated structure within a document. The technique combines a novel statistical inference method with multi-point editing in the style of Miller's Lapis (2002).

The statistical inference approach is also capable of supporting a far larger variety of models, and we therefore discuss the details of the inference technique in order to illustrate the generality of the approach for future applications.

We briefly report evidence from user trials, demonstrating that the resulting editing methods, as implemented in our simple demonstrators, can be substantially more effective than conventional techniques for repetitive editing that are provided by industry-standard applications. Finally, we review the trade-off between power and ease of use that is an inevitable concern in the design of end-user programming tools, and consider how our approach is located within this design space.

2. Sample applications

To illustrate our approach, we built two simple multi-point editing tools; one for search-and-replace in documents, and the other for bulk renaming files.

2.1. SmartRegex

SmartRegex is, like Lapis, an application that facilitates multipoint editing of a document. However, whilst Lapis offers a powerful approach to building programming editors, it offers sophisticated functionality beyond the requirements of many end-users. We therefore applied the multi-point editing approach to the simple case of supporting more effective global find and replace operations. Whilst find and replace might appear to be trivial as an example of end-user programming, it demonstrates the applicability of the underlying principles while also dealing effectively with a tedious and repetitive real task. Find and replace also happens to be the experimental test case that was used to validate the original cognitive model of Attention Investment (Blackwell 2002), and as we discuss later, Attention Investment is a particularly relevant explanatory model for this work.

2.2. FileRenamer

The second application we developed employs a similar technical approach, but for an application that would be perceived by end-users as very different from structured word processing. FileRenamer uses pattern matching to facilitate renaming a large number of files. Such bulk editing and renaming tasks would be simple for many skilled programmers, who would typically use a regular expression based tool (the second author, for example, habitually carries out complex file renaming by constructing regular expression-based macros within the EMACS directory edit mode). However, as noted by Blackwell (2001) regular expressions are particularly hard for non-programmers to learn; Perl texts such as (Christiansen 1998 and Herrman 1997) warn their readers before the chapters on regular expressions that difficulties are in store. It is for this reason that our own applications have applied the

interactive guided learning approach, in which a suitable regular expression is inferred from examples that the user provides.

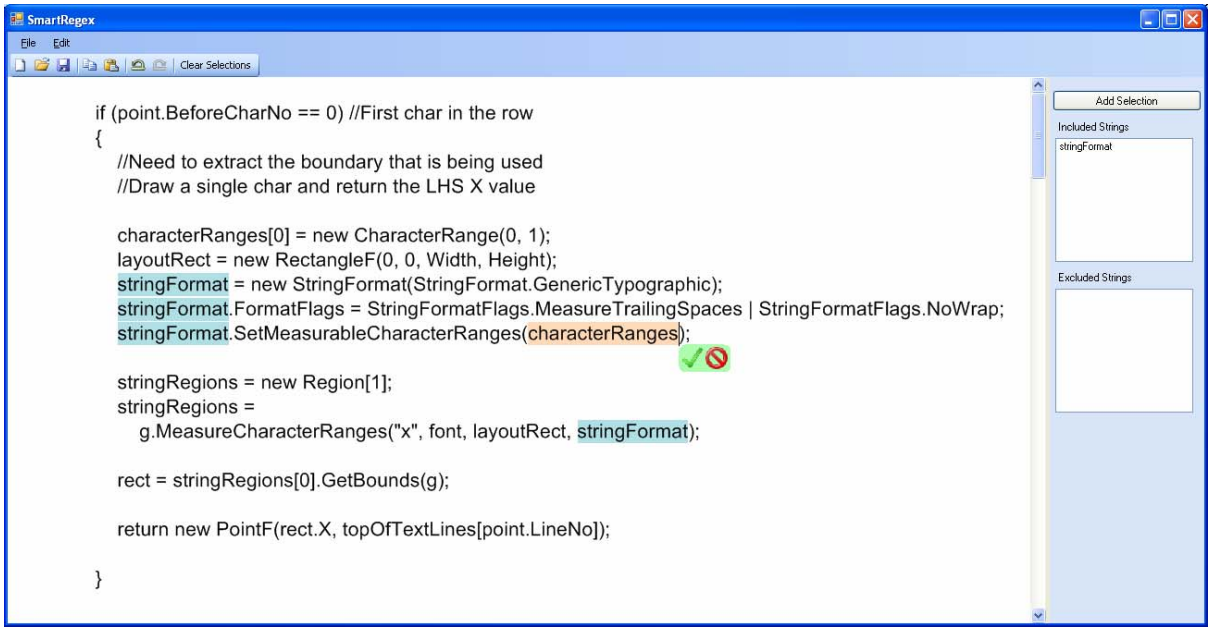

Figure 1 – Screenshot of SmartRegex in use. All strings that match the currently inferred regular expression are highlighted. Placing the cursor within any of these strings allows the user to type modifications that simultaneously change all of them. The screenshot also shows how user can extend or refine the regular expression by selecting a new string, and clicking one of two icons to either accept this string or exclude it from the regular expression

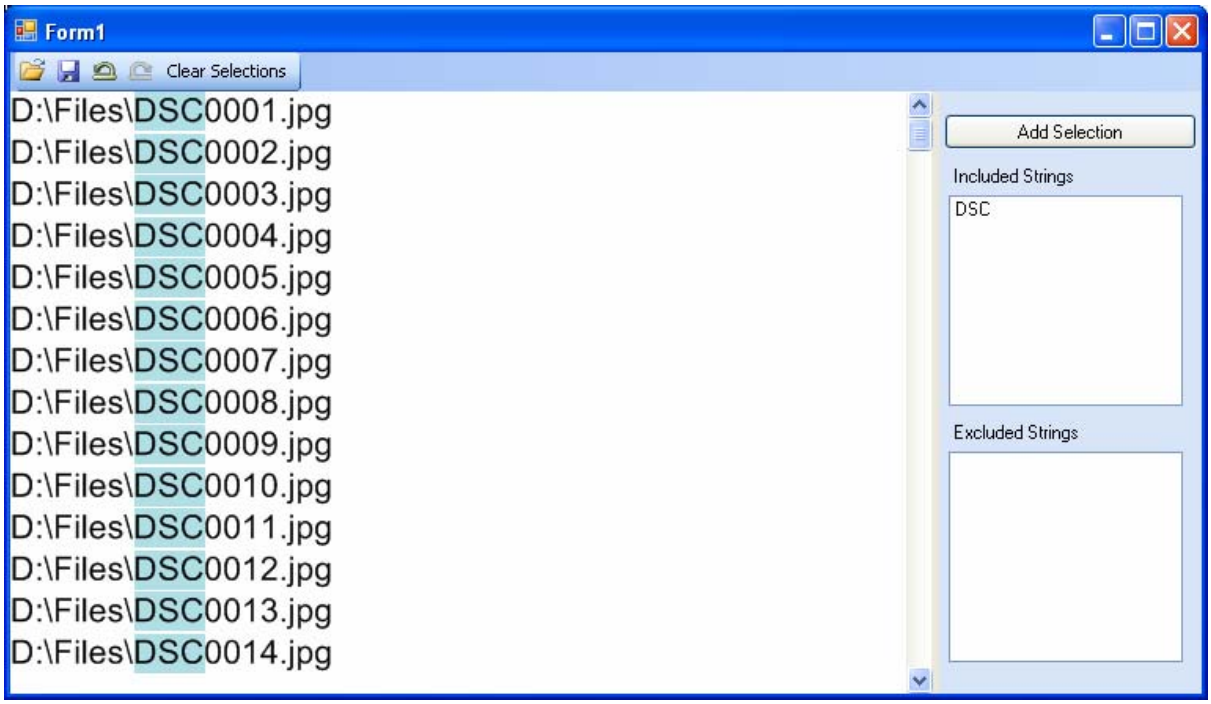

Figure 2 - Screenshot of FileRenamer in use. Using the same interaction paradigm as SmartRegex, users generate a match pattern by selecting those parts of the file names that should be changed (in this case, removing unwanted parts of an automatic filename generated by a digital camera). All candidates for renaming are highlighted, and the user can change all highlighted instances together by typing within any of them.

3. Interactive guided learning

The aim of guided learning is that the user 'trains' the computer to perform some task, in our case matching strings. Our computational perspective on the training process is a Bayesian one. The computer initially has some model (a Bayesian "prior") of the text structure that the user wishes to match. The user then interacts with the computer to modify this model by providing new Bayesian evidence – for example a new string that the user would like the model to match in future, or a string already being matched that the user would prefer not to be matched in future. This negotiation continues until the user is satisfied with the model. In order for the user to provide appropriate evidence, and indeed for the user to evaluate when the computer's model is adequately complete, it is necessary that the computer's internal model be communicated to the user, and also that the user is able to manipulate the model.

This communication and manipulation could be performed by direct manipulation of a notation representing the model. The SWYN system (Blackwell 2002) took this approach, displaying the inferred regular expression grammar to the user in the form of a visual language specifically designed to improve the understanding of regular expressions by end-users. However, this approach retains a number of problems: Firstly, whilst it makes the learning slope of the language gentler, it doesn't fully remove the need for the user to understand potentially complex syntax. Secondly, the requirement to visualise the model constrains the choice of modelling language. This would make the use of more expressive languages, such as context free grammars, difficult.

Rather than displaying and manipulating the model directly, an alternative approach is to show the effect of the model on the user's own data. The model can then be manipulated by changing the training set, the data from which the computer inferred the model, rather than the model itself. This is the approach we chose.

Various design possibilities were sketched. A selection of the sketches that led to the simple direct technique for refining the selection model are shown in figures 3-5. The final technique chosen, as shown earlier in the two demonstrator applications, is that text strings in the document are selected and the user instructs the computer whether a selected string should or should not be included in the model. A Lapis-like display of the current multi-point selections is provided to the user, so that they can determine if the model is complete (because the current set of selections corresponds to their intention) or what further refinements are needed.

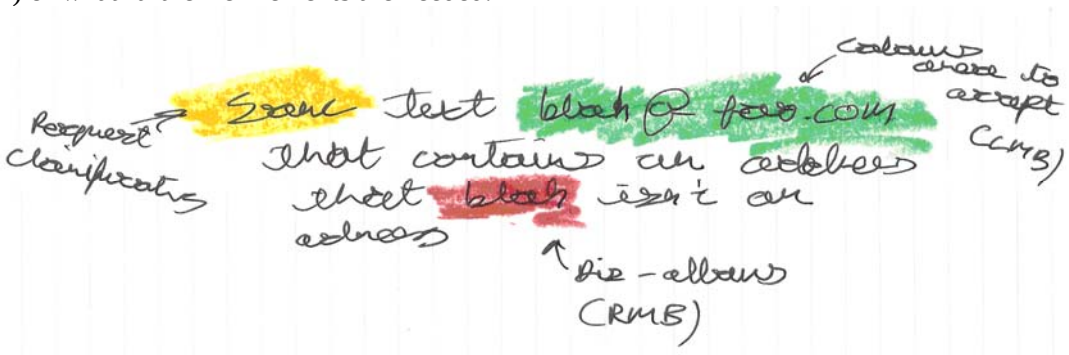

Figure 3 – An early design sketch showing text being 'painted' to indicate inclusion or exclusion from the model

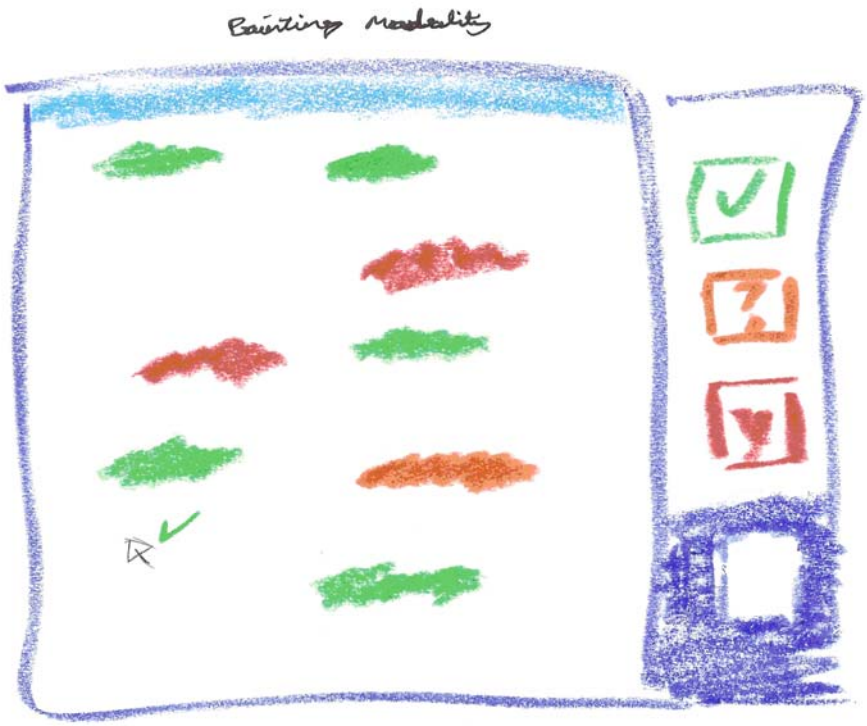

Figure 4 – Concerns with decreasing the modal behaviour of the interface led to a refinement of 'select and click' interaction rather than painting

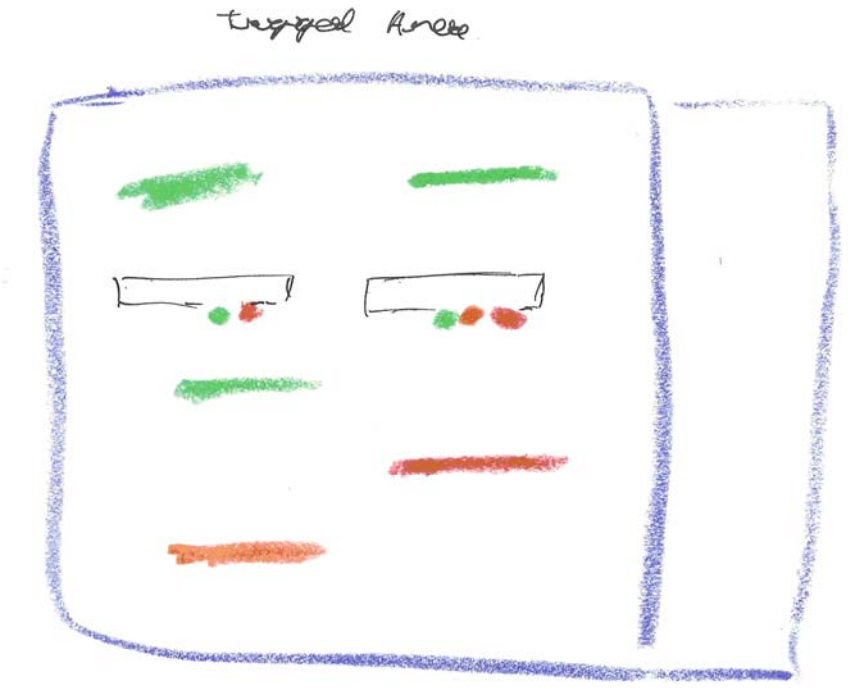

Figure 5 – Final design sketch, inspired by the 'SmartTag' paradigm of MS Office: when some text has been selected, floating buttons appear below it that can be used to mark the text as either included or excluded in the model

This section has addressed the question of how an interactive guided learning approach can be presented to end-users. However, we also need to consider the question of how the computer can work from the evidence (a series of positive and negative text examples provided by the user) to a hypothetical model of the general class of text structures that the user wishes to refer to. This task involves inferring a model for structured text, which we shall discuss next.

4. Inferring models of structured text

4.1. Models for natural language

Natural language has been modelled explicitly by many different disciplines over the centuries, from cryptanalysis to linguistics. Our view of text modelling was motivated by a passage in Shannon's famous description of communications theory (Shannon 1948). Shannon asks how many guesses it takes, on average, to determine the next character in an English sentence. The number of guesses needed is considered to be an estimator for the information gained by learning that character.

This view of characters as information has inspired a number of user interfaces, notably Dasher (Ward et al. 2000), in which the user zooms through a field of characters with probable characters being allocated a larger area, making them easier to select. A statistical model of English (PPM – Prediction by Partial Match) is used in Dasher to estimate the probability of the next character, given the previous characters.

Generally, PPM predicts the next symbol in a stream of characters by considering the relative frequencies that have been seen for the next symbol given the context (i.e. the previous few characters in the stream) as shown in figure 6. The 'order' of the PPM model (denoted by a numeral after PPM) is the number of previous characters that make up the context.

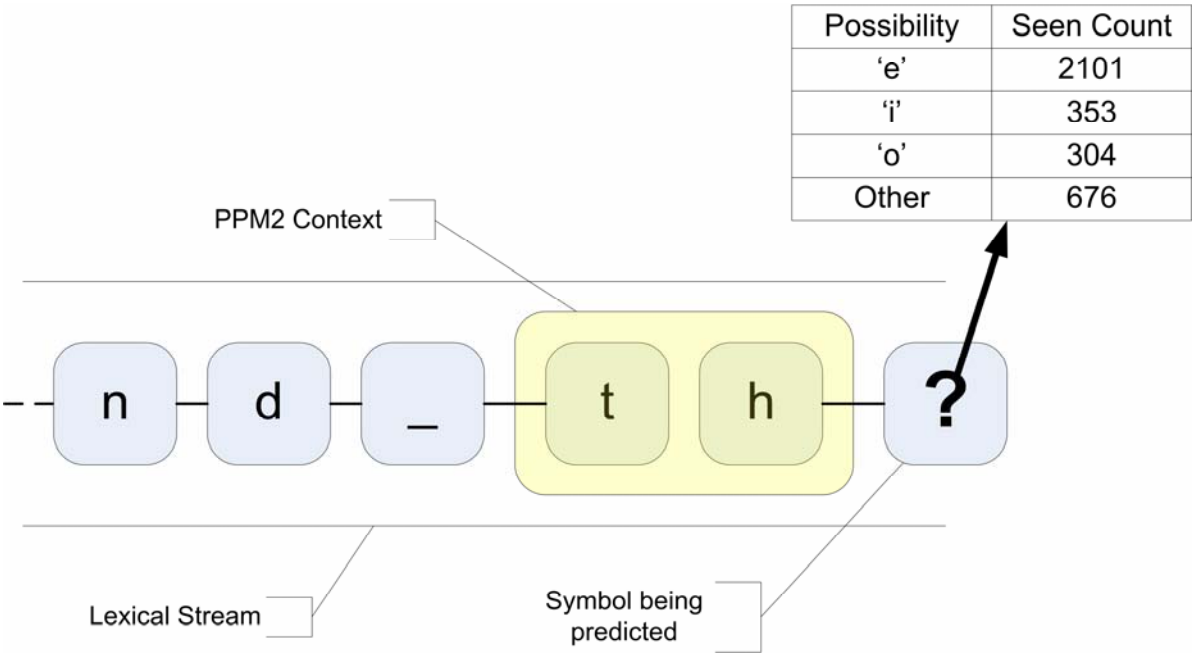

Figure 6 – PPM2 operating on an English text stream. The yellow highlighted area indicates the context being used to predict the position labelled '?'. The table indicates the relative frequencies of the next symbol given the context 'th' in the Canterbury Text Corpus

.

Our algorithm, SPPM (Structured Prediction by Partial Match), generalises PPM by modelling arbitrary contexts, instead of just the previous symbols in the character stream. For example, when

modelling natural language it can use a combination of previous characters (like PPM) and also Part Of Speech (POS) tags for previous words, see figure 7. We found that the use of POS tags improves the efficiency of prediction of the next character for the same memory usage for models over order four.

POS-tagging is a useful predictor as it starts to include information about the structure of the text in the model. Due to the nature of human languages, there are limits to what can be achieved in this manner for natural language applications. However, for applications to programming languages and other formal systems SPPM can do much better, because these languages typically have much more explicit structure.

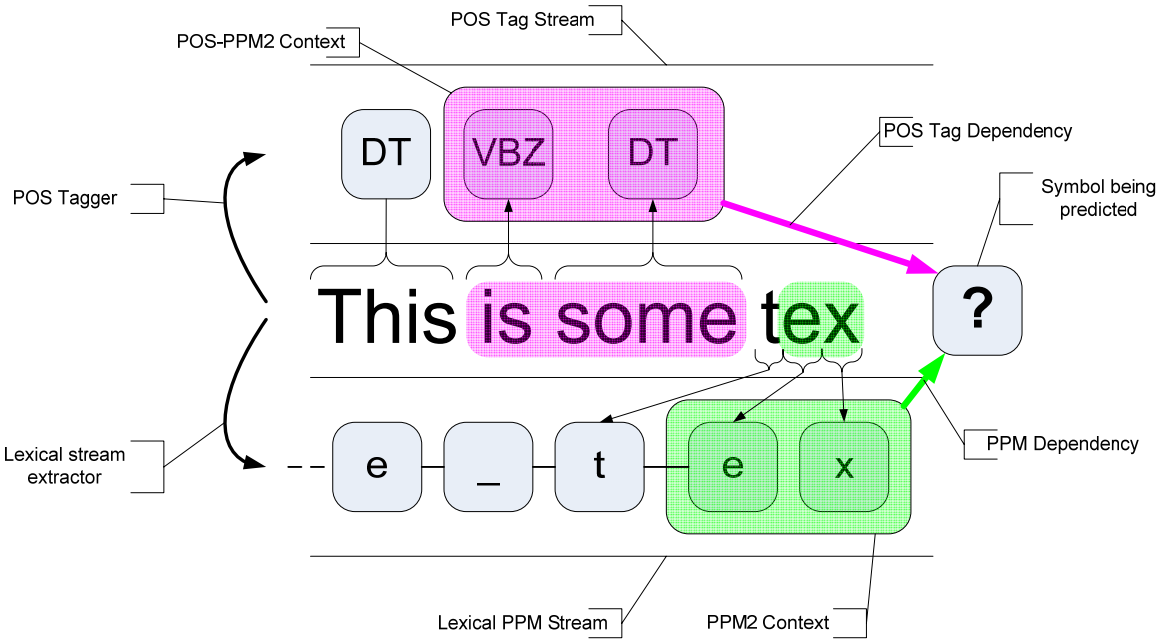

Figure 7 – Shows a single text stream (centre) being converted into a Part Of Speech (POS)-Tag stream (top) and a PPM stream (bottom). Using only POS Tags, the prediction would depend on the items highlighted above in pink; using only PPM the prediction would only depend on the items highlighted below in green. SPPM combines both of these streams, with a model that depends on both the green items (below) and the pink items (above). DT is the POS tag for 'determiner' and VBZ is the tag for 'verb, present tense 3rd person single'

4.2. Models for formal languages

As an example of a highly structured formal language, figure 8 shows the parse tree of the partial regular expression '[A-Za-z0-9'. Whilst PPM can only model the lexical stream, SPPM can also directly take into account structure of the tree, allowing far more context to be incorporated into the model and used for prediction.

The ability of SPPM to take account of the formal structure of a language means that SPPM implementations need to make several decisions that were not needed for PPM. Principally, it is necessary to implement a scheme to decide what elements of the structure to use as context. In PPM the decision as to what to use for the context was simple – it is always the previous characters. However in SPPM the data could come from, for example, a higher level in a parse tree or a neighbour in a parse tree, or the previous characters. Whilst balancing the use of those additional contexts involve extra optimisation work, the flexibility allows for modelling of data more efficiently

than in PPM, and also for modelling languages in which simple character matching is of limited use, such as source code.

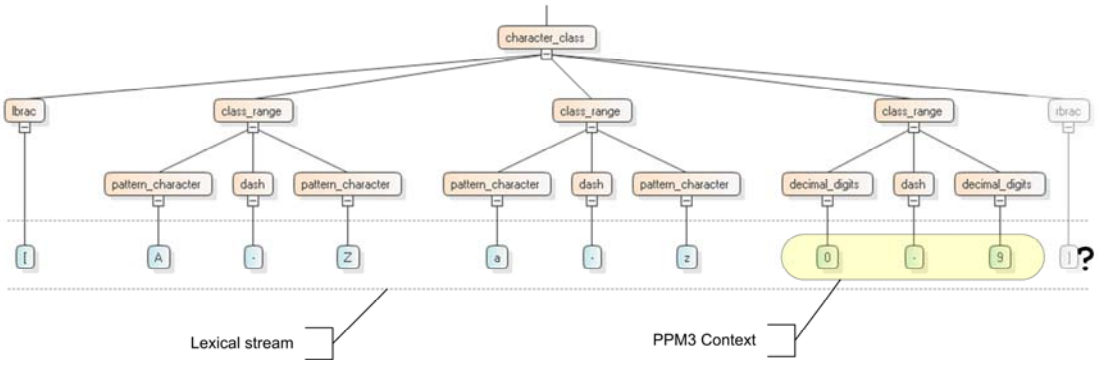

Figure 8 – A PPM3 view on predicting characters in a regular expression. The '?' denotes the symbol being predicted, and the highlighted context is the only posterior information that is considered by PPM, whereas SPPM considers both the highlighted context and the implicit repeated structure

To demonstrate its generality and understand challenges in features such as error support, we implemented SPPM models for a number of different languages, at various levels of support. These are summarised in the Table 1.

Language	Reason for support
Regular Expressions	Support for commonly used subset of Perl style regular expressions
English and POS tagging	Demonstrates multiple stream support
XHTML	Demonstrates techniques for error recovery
C#	Demonstrates complex grammar support
Parsed English	Demonstrates integration with a NLP library

Table 1 – SPPM support for various languages

This generality provides interesting research possibilities for the future, including using SPPM as a predictive model for Integrated Development Environments, either as an enhancement to code completion systems or to support a novel programming interface like #Dasher (Church 2005).

5. Empirical comparison

As observed earlier, there are many ways in which a predictive language model could be used. To demonstrate the practical value of the approach, we empirically compared the two demonstrator applications discussed in section two to equivalent features in leading commercial products. The basis for comparison was the length of time taken by users to perform representative tasks. These included two conventional find and replace tasks for SmartRegex, and pattern-match bulk renaming (e.g. "DSC0035.jpg" had to become "Holiday Photos 0035.JPEG") for FileRenamer. Our experiment also included two search tasks, not related to our principal concern with text transformation, which suffered from an experimental design flaw. Those tasks did not show significant differences, and are not reported further.

The group of six test subjects was gender-balanced and chosen from a population intended to be representative of end-users who might work with semi-structured text, from students to office professionals. The subjects were presented with four editing tasks, of which they performed two in

SmartRegex and two in MS Word. They were also given four file renaming tasks, of which they performed two in FileRenamer and two in Adobe Bridge. The orders in which they performed the tasks were counterbalanced.

As shown in figure 9 there was a statistically significant difference between time taken to complete the editing tasks using MS Word and SmartRegex. The average task completion time, across all subjects, was 81% faster when using SmartRegex (Kruskal-Wallis test shows significance at df=1, $p < 0.01$).

As shown in figure 10, there was also a statistically significant difference between time taken to complete the renaming tasks in FileRenamer and Adobe Bridge. The average task completion time, across all subjects, was 82% faster when using FileRenamer (Kruskal-Wallis test shows significance at df=1, $p < 0.01$).

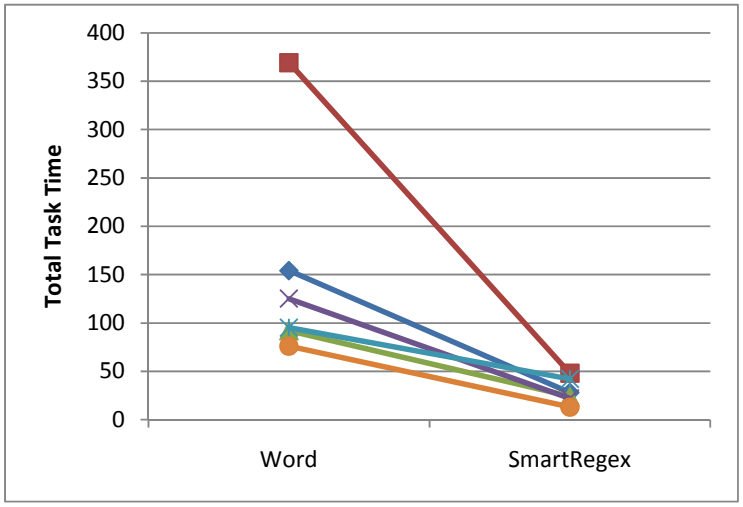

Figure 9 – Reduced task completion time when representative end-users carry out a find-and-replace text transformation task using the SmartRegex demonstrator rather than the (familiar) facilities provided by industry-standard application Microsoft Word. Lines indicate paired comparisons. The observed trend for faster task completion with SmartRegex is statistically significant (p<.01).

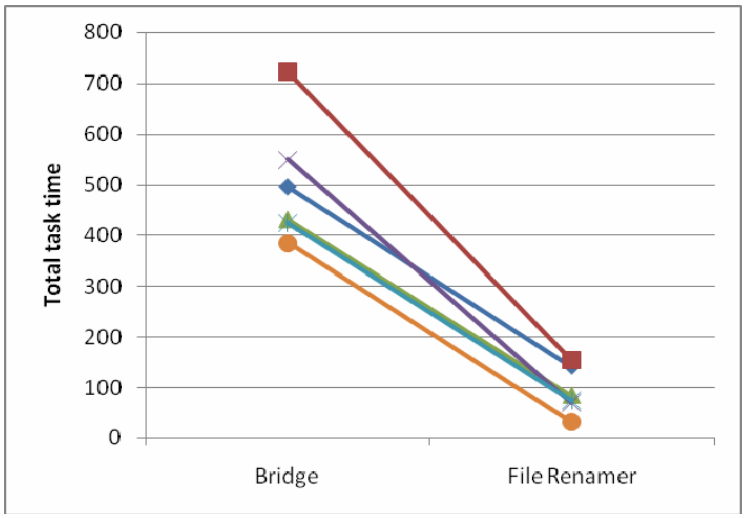

Figure 10 - Reduced task completion time when representative end-users carry out a file renaming task using the FileRenamer demonstrator rather than the facilities provided by industry-standard application Adobe Bridge. The observed trend for faster task completion with FileRenamer is statistically significant (p<.01).

Whilst it must be stressed that these applications are only demonstrators, and that much more sophisticated use could be made of the inference techniques that they demonstrate, even these simple demonstrators show a clear improvement in task performance over the equivalent features offered in industry-standard applications MS Word and Adobe Bridge.

6. Power / learning trade-off

Whenever users are given the opportunity to specify and perform multiple actions at the same time, this changes the intellectual complexity of their task, to a more abstract level of description. Although a shift to more abstract thinking about a problem is completely routine for programmers, non-programmers can find it challenging. Indeed, the "paradox of the active user" observes that many users prefer just to carry on with a repetitive or inefficient task rather than seek out better ways to do it (Carroll & Rosson 1987).

Designers of more efficient UIs can address this by offering "just in time programming" (Potter 1993), where users normally proceed by manual direct manipulation, but are offered a more abstract programming approach only when their task becomes particularly repetitive.

In terms of end-user programming research, this is a vital component of the "gentle slope" requirement that incremental gains in expressive power should be accompanied by no more than incremental increases in tool complexity (Pane & Myers 2006). Blackwell's (2002) Attention Investment model of abstraction use unifies these principles. It suggests that users often face a strategic choice between relatively mundane direct manipulation, or investment of mental effort in more abstract programming-like approaches to a task. This attention investment comes with a cost, but also a risk that the program may not work, or that a programming strategy might even incur further effort to fix bugs or unintended execution results.

Our own interaction design has been directly motivated by these principles and theoretical observations. Inference is started with a routine selection operation, while generalisation takes place in the background, being refined with each further selection. This offers both a "just in time" and a "gentle slope" experience. The perceived attention investment cost of building the training set is both small and incremental, through the use of the accept/reject checkmark approach. The perceived attention investment risk is also minimised, through the fact that all specified execution instances are highlighted in context and can be inspected by the user.

This interaction style has some resemblance to the work of Burnett et. al. on end-user programming of spreadsheets, where benefits of assertions and testing are highlighted in context, and can be modified using simple checkboxes (Rothermel et al. 1998). Their "surprise-explain-reward" design strategy (Wilson et al. 2003) applies the attention investment model to encourage better software engineering discipline from end-users programming spreadsheets, while our inference-based strategy is designed to encourage the use of a programming approach rather than repetitive direct manipulation in the first place.

A number of other inference systems have taken this approach in the past, most notably Cypher's (1991) Eager, an agent which monitored user actions for opportunities to generalise over repeated operations, and "eagerly" popped up to offer automated continuation. An early text interface for generalisation over repeated action was Dynamic Macro (Masui 1994).

These early inference-based interaction prototypes used deterministic generalisation techniques. The power and generality of those techniques quickly becomes limited – even in the simple applications we describe here, inference of regular expressions is a challenging problem in the general case. The more powerful statistical model inference techniques that we have developed promise opportunities for a far wider range of abstract expressive languages. Our simple applications have demonstrated that statistical techniques are a valid alternative to the deterministic techniques used in Eager, and we have empirically verified the usability benefits of such an approach in applications of similar complexity to those early systems. Future opportunities offered by our approach extend to a greater range of structured texts.

7. Conclusion

We have described the use of a sophisticated statistical inference technique to support programming-like abstract modifications of structured text. Statistical inference, as a basis for "programming by example" systems, is one of the most promising strategies for addressing the end-user programming challenges highlighted by Blackwell's attention investment model.

Firstly, statistical inference, when applied in a guided learning context, can reduce the perceived attention cost of an abstract strategy. Secondly, efficient user interface techniques such as multi-point editing can reduce both the perceived investment risk (because incremental structure changes can be previewed), and also the direct manipulation costs (as shown in our empirical evaluation of two simple demonstrator applications).

Although we have demonstrated the feasibility of this approach using relatively simple demonstrator applications, the SPPM inference method offers significant benefits for a far greater range of end-user programming applications. We believe that a particularly promising opportunity comes from the fact that SPPM can be applied both to natural language models and to more conventional programming language grammars. It is often the case, in end-user programming contexts, that end-users deal with combinations of more and less formal textual data. In these contexts, a technique that offers a common mathematical foundation for representation and inference across different kinds of textual structure provides a powerful tool for future end-user programming innovations.

8. Acknowledgements

Luke's current research is supported by the Eastman Kodak company. The implementations of SPPM, SmartRegex and FileRenamer were completed to fulfil a requirement of the Cambridge Computer Science Tripos.

9. References

Beckwith, L., Kissinger, C., Burnett, B., Wiedenbeck, S., Lawrance, J., Blackwell, A. and Cook, C. 2006. Tinkering and gender in end-user programmers' debugging. Proc. CHI'06, pp. 231-240.

Blackwell, A.F. 2001. SWYN: a visual representation for regular expressions. In *Your Wish Is My Command: Programming By Example* Morgan Kaufmann Publishers, San Francisco, CA, pp. 245-270.

Blackwell, A.F. 2002. First steps in programming: A rationale for Attention Investment models. Proc. IEEE Symposia on Human-Centric Computing Languages and Environments, pp. 2-10.

Carroll, J.M. & Rosson, M.B. 1987. Paradox of the active user. In J. M. Carroll (Ed.) *Interfacing Thought: Cognitive Aspects of Human-Computer Interaction*, Bradford Books, pp. 80-111

Christiansen, T. and Torkington, N. 1998. *Perl cookbook*.: O'Reilly, Sebastopol CA, 1998.

Church, L. 2005. Introducing #Dasher, A continuous gesture IDE, A work in progress paper. Proc. PPIG 17, pp. 227-241

Cypher, A. 1991. EAGER: programming repetitive tasks by example. Proc. CHI'91, pp 33-39

Grover, D.L., King, M T & Kuschler, C A. 1998. Patent No. US5818437, "Reduced keyboard disambiguating computer," Tegic Communications, Inc., Seattle, WA.

Herrman, E. 1997. *Teach yourself CGI programming with Perl 5 in a week.* Sams, Indianapolis, 1997.

Lieberman, H., Nardi, B. A., and Wright, D. 1998. Grammex: defining grammars by example. Proc. CHI '98, Conference Summary on Human Factors in Computing Systems pp. 11-12

Masui, Toshiyuki, and Ken Nakayama. 1994. Repeat and predict—Two keys to efficient text editing, Proc. CHI '94, pp. 118-130

Miller, R C, Myers, B. A., 2002. LAPIS: smart editing with text structure. Proc. CHI'02, pp. 496-497

Pane, J.F. and Myers, B.A. 2006. Natural programming languages and environments. In H. Lieberman, F. Paterno and V. Wulf (Eds.), *End User Development*. Dordrecht: Springer, pp. 31-50.

Potter, R. Just-in-time programming. 1993. In *Watch What I Do: Programming By Demonstration*, A. Cypher, D. C. Halbert, D. Kurlander, H. Lieberman, D. Maulsby, B. A. Myers, and A. Turransky, Eds. MIT Press, Cambridge, MA, 513-526.

Rothermel, G., Li, L., DuPuis, C. and Burnett, M. 1998. What You See Is What You Test: A Methodology for Testing Form-Based Visual Programs. Proc. International Conference on Software Engineering, 1998, pp. 198-207.

Shannon, C. E. "A Mathematical Theory of Communication", *Bell System Technical Journal*, vol. 27, pp. 379-423, 623-656, July, October, 1948

Ward, D. J., Blackwell, A.F., and MacKay, D.J.C. 2000. Dasher - A Data Entry Interface Using Continuous Gestures and Language Models. Proc. UIST 2000, pp. 129-137.

Wilson, A., Burnett, M., Beckwith, L., Granatir, O., Casburn, L., Cook, C., Durham, M., and Rothermel, G. 2003. Harnessing Curiosity to Increase Correctness in End-User Programming, Proc. CHI '03, pp. 305-312.

Witten, I. H. and Mo, D. 1993. TELS: learning text editing tasks from examples. In *Watch What I Do: Programming By Demonstration*, A. Cypher, D. C. Halbert, D. Kurlander, H. Lieberman, D. Maulsby, B. A. Myers, and A. Turransky, Eds. MIT Press, Cambridge, MA, 183-203.

Zhai, S. and Kristensson, P.O. 2003. Shorthand writing on stylus keyboard. Proc. CHI'03, pp. 97-104