Repeat and Predict – Two Keys to Efficient Text Editing

Toshiyuki MASUI
Software Laboratories
SHARP Corporation
2613-1 Ichinomoto-cho
Tenri, Nara 632, Japan
Tel: +81-7436-5-2468
E-mail: masui@shpcsl.sharp.co.jp

Ken NAKAYAMA
Department of Information Science
Faculty of Science
The University of Tokyo
3-8-1 Komaba, Meguro, Tokyo 153, Japan
Tel: +81-3-5478-0520
E-mail: ken@is.s.u-tokyo.ac.jp

ABSTRACT

We demonstrate a simple and powerful predictive interface technique for text editing tasks. With our technique called the *dynamic macro* creation, when a user types a special "repeat" key after doing repetitive operations in a text editor, an editing sequence corresponding to one iteration is detected, defined as a macro, and executed at the same time. When we use another special "predict" key in addition to the repeat key, wider range of prediction schemes can be performed depending on the order of using these two keys.

KEYWORDS: Text Editing, Predictive Interface, Programming By Example, PBE, Programming by Demonstration, PBD, Keyboard Macro, Dynamic Macro Creation

DYNAMIC MACRO

Various techniques for programming by demonstration (PBD) and predictive user interface have been proposed to support easy programming or to reduce the burden of doing similar operations repeatedly[1][2]. We propose a new simple and powerful method of creating a keyboard macro from repetitive user operations, which we call the *dynamic macro* creation method.

Dynamic macro works as follows: All the recent user operations in a text editor is logged as a string, and when a special "repeat" command is issued by typing a special key denoted as REPEAT, the system looks for repetitive operations from the end of the string. If such operations are found, they are defined as a macro and then executed at the same time. If REPEAT is typed again, the macro is executed again. For example, when a user enters a string "**abcabc**" and types REPEAT after that, the system detects the repetition of "**abc**," defines it as a macro, and executes the macro, resulting in another "**abc**." When the user types REPEAT again, one more "**abc**" is inserted.

The process of detecting repetitive operations consists of the following two strategies.

Rule1: If there exist two same consecutive sequences of operations just before typing REPEAT, define the sequence as a macro. If there exist more than one such sequences, take the longest one. For example, if the user types REPEAT after "**abccabcc**," define "**abcc**" as the macro, not "**c**."

Rule2: If there exists no such sequence, look for a pattern XYX just before REPEAT, where X and Y denote nonempty sequences of operations. If there exist such sequences, define XY as a macro, executing only Y for the first REPEAT. If there exist more than one such sequences, take the longest X and take the shortest Y with that X. For example, if the user types REPEAT after "**abracadabra**," take "**abra**" as X and "**cad**" as Y, not "**a**" as X and "**br**" as Y.

Example

Figure 1 shows the case when a user types REPEAT after typing % ␣ ^N ^A % on GNU Emacs. In this case, Rule2 applies and % ␣ ^N ^A is executed repeatedly.

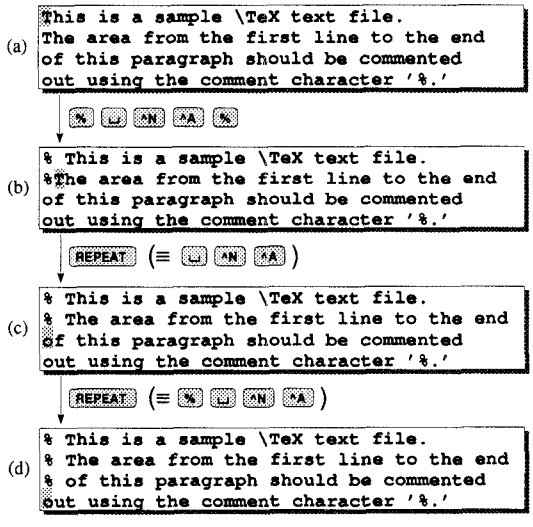


Figure 1: Adding comment characters to each line.

USING DYNAMIC MACRO WITH OTHER PREDICTION TECHNIQUES

Dynamic macro predicts right in most cases, but it sometimes guesses differently from the user's expectation. For example, if a user types REPEAT after TAB TAB a b c RET TAB TAB, Rule1 applies and another TAB is executed, which may be different from the user's expectation of a b c RET. This problem can be solved by using a "predict" key, or PREDICT, with which users can change the prediction scheme after the system made a wrong guess. PREDICT normally acts like conventional predictive keys such as the "file name completion" key or "dynamic abbreviation" key of GNU Emacs. Using PREDICT after REPEAT, users can try different candidates whenever the prediction was an unexpected one. In Figure 2, changing the prediction scheme by PREDICT makes Rule2 active, and next candidate, a b c RET, is predicted instead. If this prediction is the one in the user's mind, the user can then type more REPEAT to go on the prediction.

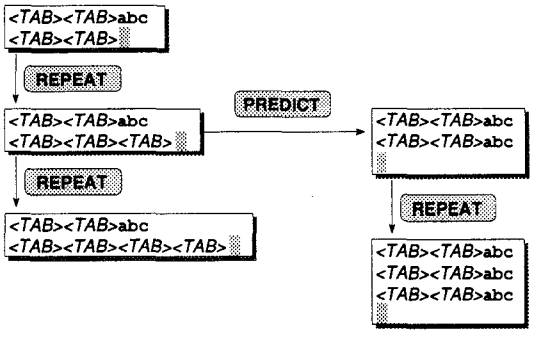


Figure 2: Changing the prediction scheme using PREDICT.

We can go further by extending the semantics of REPEAT as follows: if it is pressed after REPEAT or PREDICT, execute the same prediction scheme again; otherwise, predict the next string using the dynamic macro technique. With this extension, users can first select the prediction strategy using PREDICT, and then apply it repeatedly by REPEAT. Figure 3 shows an example of using this technique.

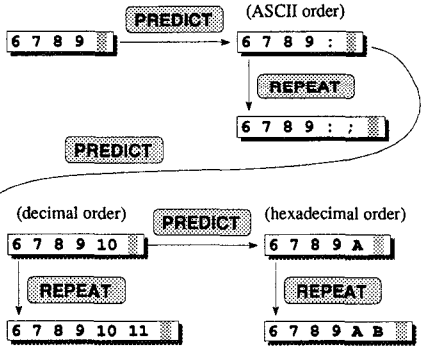


Figure 3: Select and repeat prediction strategies.

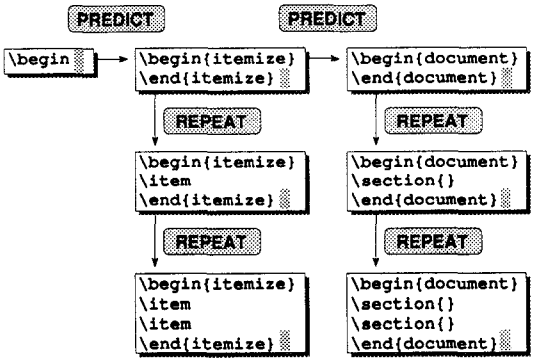
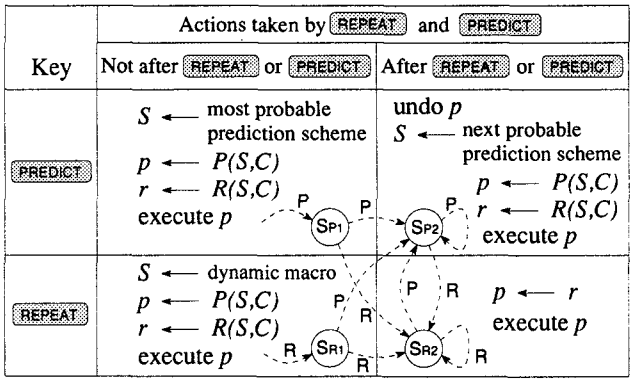


Figure 4: Qualitative and quantitative prediction.

We can go even further to extend the meanings of REPEAT and PREDICT, and use them as mode-specific prediction keys which correspond to quantitative and qualitative prediction, respectively. Figure 4 shows how PREDICT and REPEAT can be used when writing LaTeX documents with this kind of extension.

All together, the functions of REPEAT and PREDICT are shown in Figure 5. When REPEAT is pressed, the system goes to state SR1 and dynamic macro is executed. If PREDICT is pressed there, the system goes to state SP2, undo the last prediction, and performs another new prediction. Various other prediction schemes can be tried based on the state transition shown in Figure 5.



S: prediction scheme *C*: current context *P*,*R*: prediction functions
p,*r*: sequences of operations SP1,SP2,SR1,SR2: prediction states

Figure 5: Actions by REPEAT and PREDICT.

REFERENCES

[1] Cypher, A., Ed. *Watch What I Do – Programming by Demonstration.* The MIT Press, Cambridge, MA 02142, 1993.

[2] Myers, B. A. Demonstrational interfaces: A step beyond direct manipulation. *IEEE Computer 25*, 8 (August 1992), 61–73.