ヒューマンインタフェース 64－7
（1996. 1. 11）

テキストにおける過去の変更の位置に基づく参照の実現

上田 康裕　　八木 啓介　　美濃 導彦　　池田 克夫

京都大学工学部

〒606-01 京都大学工学部情報工学科
e-mail: ueda@kuis.kyoto-u.ac.jp

本報告では，ユーザが過去に行った変更について，その変更前の状態の参照や利用が容易に行えるようなテキストエディタの実現について述べる．本システムでは，局所的かつ連続的に行われた操作系列をひとまとまりとして扱うことにより，コマンドヒストリから変更の系列をリアルタイムで生成する．また，ユーザが過去に変更を行った領域は色づけされており，ユーザはそこをクリックするだけで変更前の状態を簡単に参照できるようになっている．このようなインタフェースを実現するにあたり，単に変更の履歴を保持しておくだけでは効率が悪いので，テキストをラティス構造とし変更を管理する方法を用いた．

Position Specified Retrieval of Text Change

Yasuhiro UEDA　　Keisuke YAGI　　Michihiko MINOH　　Katsuo IKEDA

Kyoto University

Department of Information Science, Faculty of Engineering,
Kyoto University, 606-01, Japan

We realized a text editor that gives color to region where the user made change in the past, and he can easily retrieve the previous version before the change by clicking there. We propose a method to retrieve changes by grouping edit-commands that are executed locally and sequentially, and a method to manage the changes by a lattice structured text. This managing method is suitable for the coloring of changes and retrieval of the previous version.

1 はじめに

本報告では，文書等の作成においてユーザが過去に行った変更の前の状態を容易に参照や利用できるようなテキストエディタの実現について述べる．

ユーザがテキストエディタで，文章やプログラムなどのテキストを書いているときに，ユーザは何の迷いや入力ミスもなく，いきなり求めるものを得られるわけではない．入力ミスや既に行った変更を取り消すための手段の一つとして，多くのテキストエディタやグラフィックエディタは，UNDO機能を持っている．

GNU Emacs[1] はテキストエディタの中では比較的強力なUNDO機能を持つ．このUNDO機能では直前に実行されたコマンド列を無効化することができる．つまり，それらのコマンド列が実行される前の状態に戻ることができる．しかし，Emacsの提供するUNDO機能が他に比べ強力であるとはいえ，ユーザはそれで満足しているわけではない．Yangは“EmacsのUNDO機能はどのようなものが理想的か”というアンケートをネットニュース上のグループgnu.emacsで行った[2]．それに対する解答を要約すると以下のようになる．

1. UNDOの粒度はユーザの実行するコマンドと同じであるべき．

2. タスク粒度でのUNDOも実現して欲しい．

3. 領域限定UNDOなどもう少し自由なUNDOを実現して欲しい．

4. ユーザがコマンドヒストリを見ることができるようにして欲しい．

1.については，アンケート当時のEmacsは一括置換をUNDOすると個々の置換を単位としたUNDOになるなど，ユーザの実行するコマンドより小さな粒度でのUNDOとなることがあったが，現在のバージョンでは改善されている．

2.については，現バージョンでは，限定的なものではあるが，連続する挿入コマンドや連続する削除コマンドをひとまとめにして扱うようになっている．一方で，連続した同一操作にとどまらず，タスク粒度のUNDOについて，Yangもその必要性を認めている．タスクとは，ユーザのその時点その時点での編集の小目的であり，複数のコマンドによって構成される．しかしながら，既存の技術では自動的にタスクを完全に認識することが困難であるため，実現には至っていない．

ユーザが文章を作成する場合には，ある段落を書く，特定の文を修正するなど，位置について局所的に作業していることが多い．このような連続して行われた局所的編集作業をひとまとまりのタスクとして捉えることができる．そこで，本研究では，完全にタスクを認識することは目指さず，この局所的編集作業に基づくタスクだけを取り扱うことにする．以降，局所的編集作業に基づくタスクを‘変更’とよぶことにする．

3.については，GNU Emacsで実装されているUNDOでは，最近実行したコマンドラインから順番にUNDOすることしかできず，ヒストリ上の各コマンドはそれ以降に実行されたコマンドをUNDOすることなく，UNDOすることはできない．しかし，ヒストリ途中のコマンドだけをUNDOしたい状況もあり，とくに望まれているのが，テキストの一部分だけを過去の状態に戻す領域限定UNDOである．ヒストリ上の任意のコマンドのUNDOを実現するアルゴリズムが，Prakashら[3]などにより提案されている．

ところで，従来提案されている領域限定UNDOのインタフェースは，限定した領域で実行されたコマンドだけについて，通常のUNDOと同様のインタフェースを提供するもので，以下の2種類が存在する．

a. GNU Emacs同様，領域内で実行されたコマンドを最近実行されたものから順にUNDOしていき，それに伴い現テキストが変化していくもの．

b. 領域内で実行されたコマンドの一覧を示しておき，ユーザがそこで選んだコマンドだけを

UNDOするもの．これは，アンケートでの意見4.を反映したものと言える．

a.の方は，かなり前の状態に戻すのに手間がかかるとともに，UNDOした結果どうなるのかをユーザが前もって予測することが，ほぼ不可能であるという欠点がある．b.の方では，非常に自由にコマンドをUNDOでき，かつコマンドが見えているために結果がある程度予測できる．一方で，コマンドを直接ユーザに見せた場合，とりわけ初心者には分かりにくいという問題がある．また，一つのコマンドを一行以内で表示しなければならないなど，表示サイズが制約されるという問題がある．このため，コマンドの引き数についての情報を省略して出力せざるを得ず，ユーザがUNDOのもたらす結果を必ず完全に予測できるわけではない．

本システムでは，これらの問題を解決するために，変更されたことのある領域を色づけしたテキストをユーザに提示することにした(図2)．これにより，ユーザは変更に関して位置と変更後の状態を完全に知ることができる．一方で変更された時間と変更前の状態といった情報が欠落している．変更前の状態については，色づけされた領域をクリックすれば変更前のテキストを簡単に参照できるようにすることで解決できる．また，ユーザが領域限定UNDOを望む場合には，位置情報こそがユーザにとって重要な情報であり，時間情報はたいして重要ではないと思われるので，時間情報を見せるために特別なことはしない．

このようなインタフェースを提供するためには，コマンドヒストリのようにコマンドが時系列で並んでいることに意義は少ない．むしろTEXTNET[4]などのように，テキストをいくつかに分割して，各部分テキストをノードするようなグラフとしてテキストを管理し，グラフ上で変更前を表す経路と変更後を表す経路を残すことにより，変更を管理するほうが適していると考えられる．このグラフはラティス構造となるため，グラフ構造のテキストをラティス構造テキストとよぶことにする．

以下，2節ではコマンド系列から局所的な編集操作をまとめ，変更コマンドを生成する方法について，3節では変更コマンドの列をラティス構造テキストとして管理する方法について，4節では，過去に行った変更を参照するためのインタフェースおよび，それをラティス構造テキストを用いて実現する方法について述べる．

2 変更コマンドの生成

ユーザが文章を作成する場合には，ある段落を書く，特定の文を修正するなど，位置について局所的に作業していることが多い．そこで，このような連続して行われた局所的編集作業をひとまとまりのタスクとして捉えることにした．本節では，コマンドヒストリを適当に区切り，局所的編集作業のまとまりに分割する方法を述べる．

コマンドヒストリに含まれているコマンドは，

- 文字列挿入 (挿入先頭位置，挿入文字列)
- 文字列削除 (削除先頭位置，削除文字列)

の2種類だけであるとする．ユーザの入力コマンドレベルでは，より複雑な編集を行うコマンドも存在するが，そのようなものは挿入と削除に分解されているとする．GNU Emacsでもコマンドヒストリは，挿入と削除が最小単位となっており，この前提は実現が困難なものではない．また，カーソル移動などテキスト以外のシステム状態を変更するコマンドについては取り扱わない．

局所的編集作業を構成するコマンド系列を定めるにあたって，直前のヒストリの区切れからその時点までに編集されてきた領域であるカレントリージョンというものを考える．カレントリージョンは直接に編集された領域であり，その近傍の領域をいっさい含まない．コマンドが実行されたときに，もし，その実行位置がカレントリージョン外であれば，このコマンドからは別の領域での編集が始まったことになるので，その直前のコマンドまでをひとまとまりとし，ヒストリをそこで区切る．コマンドがカレントリージョン内で実行されていれば，直前のコマンド系列で扱っていたの

と同一の領域で実行されているので，同じまとまりに属していることになる．

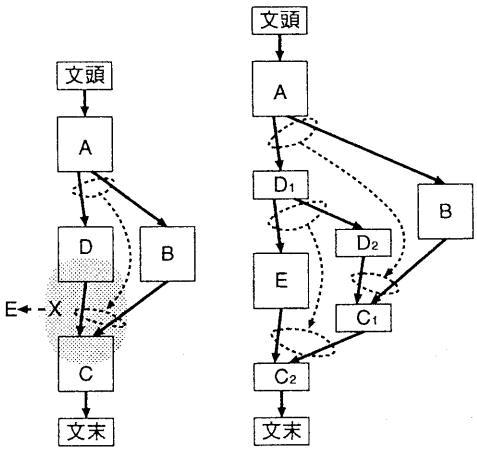
具体的には以下のようになる．

1. 初期状態ではカレントリージョンは存在しないとする．
2. 文字列挿入が行われた場合．
   - 挿入位置がカレントリージョン内(境界上も含む)であるならば，カレントリージョンに挿入文字列を挿入した領域を新しいカレントリージョンとする．
   - それ以外の場合は，カレントリージョン外への挿入であるので，この挿入の直前までをひとまとまりとし，新しいカレントリージョンは，この挿入文字列となる．
3. 文字列削除が行われた場合．
   - 削除文字列の一部がカレントリージョン内に存在していれば，カレントリージョンから削除文字列を削除した領域を．新しいカレントリージョンとする．
   - それ以外の場合は，カレントリージョン外での削除であるので，この削除の直前までをひとまとまりとし，新しいカレントリージョンは，この削除が行われた位置でのヌル文字列となる．

このようにしてコマンドヒストリをいくつかのまとまりに分けると，各まとまりについて，編集の行われた位置，編集後得られた部分テキスト，編集前の部分テキストを求めることができる．この結果，挿入と削除から構成されていたヒストリから，

- 変更 (変更開始位置，変更前文字列，変更後文字列)

というコマンドからなる，より粒度の大きなヒストリを得ることができる．このヒストリを変更履歴とよぶことにする．



(a) 変更前　　(b) 変更後

図 1　変更によるラティス構造テキストの変化

3　ラティス構造テキスト

本節では，変更履歴が与えられたときにラティス構造テキストを構築する方法について述べる．

新しくテキストの作成が開始される初期状態では，ラティス構造テキストは，文頭ノード，文末ノードおよびそれらを結ぶリンクからなる．

変更が行われると，変更前の部分を切りだし，変更後の部分テキストに対応した新たなノードを用意し，両者から変更領域の前後の領域に対してリンクを張る．したがって，このとき生じた分岐から合流に至る部分がこの変更を表している．

変更が行われるたびに，分岐，合流が生じるので，一つの変更により生み出された分岐と合流の対応を取るのが困難になる．さらに，複数の変更の開始位置や終了位置がたまたま同じになった場合には，どちらの経路がそれぞれの変更に関係しているのかが分からなくなる．これらを回避するために，各変更に関して，分岐したリンクの対と合流したリンクの対の間に特別なリンクを張る．

例えば，図1-(a)のようなラティス構造テキストが存在していたとする．実線のリンクが通常の

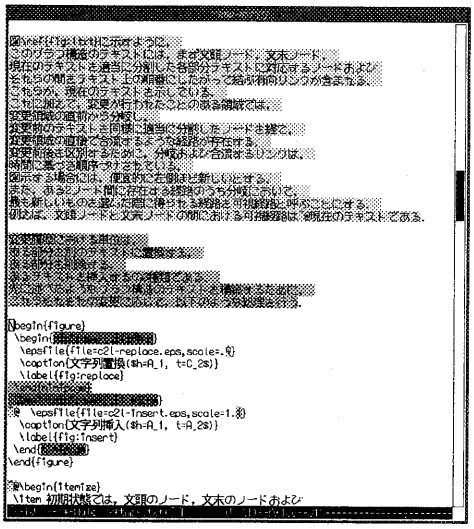


図 2　実行画面 (その 1)

リンクであり，波線のリンクは対応する分岐リンクと合流リンクを結ぶリンクである．便宜的に変更後の部分テキストの方を左に描くことにすると，現テキストを表す経路は ADC である．このような現テキストを表す経路を可視経路とよぶことにする．また，図 1-(a)は過去に $B \rightarrow D$ なる変更が行われたことも示している．この状況から次の変更により，D から C にまたがる領域の一部 X が E に置き換えられたとする．この場合には，X に相当する部分 D_2C_1 が切り出され，その左に E が用意され，双方から，変更領域の直前の領域 D_1 および変更領域の直後の領域 C_2 との間にリンクが張られる．さらに，$(D_1 \rightarrow E, D_1 \rightarrow D_2)$ と $(E \rightarrow C_2, C_1 \rightarrow C_2)$ の間には対応した分岐リンクと合流リンクを示すためのリンクが張られる．この結果，ラティス構造テキストは図 1-(b)のようになる．

4　位置に基づく過去の変更の参照

まず，本システムで提供するインタフェースについて述べる．

本システムでは，位置に基づく過去の変更の

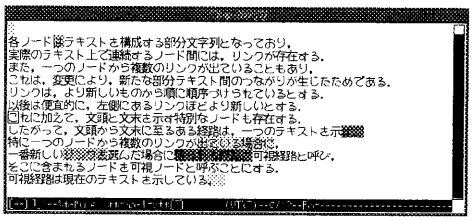


図 3　実行画面 (その 2)

参照を行うために，変更されたことのある領域を色づけして出力するようなインタフェースを提供する．色づけすることにより，変更の存在する位置がはっきりと分るようになり，求める変更へとたどり着きやすくなる．ユーザが変更前の状態を参照したいと思えば，色づけされた変更後の領域にマウスカーソルを合わせ，そこでマウスをクリックするという直接的な操作をすればよい．この結果変更前の部分テキストが別ウィンドウで提示される．

図 2は本システムの実行画面である．この図の場合には，変更が行われたことがある区間は灰色で示されている．灰色の濃さが領域によって異なるのは，二つの変更領域が連続した場合に両者を区別するためであり，それ以外には意味はない．ユーザが上半分の位置にある灰色領域の変更前の部分テキストを参照したいと思い，その領域にマウスカーソルを合わせクリックすれば，別ウィンドウが開き，図 3のような変更前のテキストがそこに提示される．そして，このウィンドウ内でも，更に細かい変更について色づけが為される．参照した結果，この変更を取り消したいと思った場合には，この領域だけを変更前の状態に戻すことができる．逆に，求める変更でなかったり，現状のほうが良いと思った場合には，取り消さずに参照を終えることもできる．

次に，色づけすべき変更を判定する方法について述べる．全ての変更を色づけして見せることは不可能である．なぜならば，現在のテキスト状態において，他の変更がまず取り消されなければ，取り消すことが不可能な変更が存在するからであ

る．例えば，前節の例では，$B \rightarrow D$を取消し．Bを含む状態に戻したければ，まず，$X \rightarrow E$を実行する前の状態に戻っていなければならない．

現在のテキスト状態で取り消すことが可能な変更というのは，変更後の状態に相当する経路に含まれているノードが全て見えている変更であり，つまり，その変更に関する分岐ノードも合流ノードも可視経路上に存在する．したがって，いずれか一方でも可視経路上にない変更については，現在のテキスト状態から取り消すことは不可能であり，これにより，現状では取消し不可能な変更を判別することが容易にできる．

次に，色づけされた領域における変更に関して，変更前の状態を得るための処理について述べる．変更前の状態を得たい変更に対して，対応する分岐リンクと合流リンクを一意に定めることができる．そして，それらの分岐リンクおよび合流リンクのうちで，左側のリンクに基づく経路は，変更後の状態に対応する経路で可視経路に含まれている経路である．そして，右側のリンクに基づく経路が変更前の状態を表している．分岐リンクと合流リンクを結ぶような経路を見つけ出すには，この分岐リンクの次のノードを始点として，最も左側の経路を取るように文末の方へたどっていけばよい．このように文末へと進むうちに，必ず合流リンクに至るので[1]，そこまでに得られた経路が変更前の状態を表すものとなっている．

ある変更について，変更前状態へと戻す操作は，その変更に関する分岐リンクおよび合流リンクについて，変更前に対応するものと変更後に対応するものを入れ替える，つまり右側にあったものを左側にするだけでよい．

本システムのようなインタフェースを実現しようとした際に，従来のように，変更をヒストリとして管理していたのでは，変更を取り消す際には変更の依存関係を調べ，変更を取り消した場合には全コマンドに対して，コマンド実行位置の補整を行わなければならないなど，手間のかかる処理が必要となる．これに対して，本手法のように変更履歴をラティス構造テキストで管理すれば，ラティス構造テキストを構築した段階でこれらの処理がほぼ済んでおり，比較的簡単な処理で変更の参照や取消を実現できる．

[1]証明についてはここでは省略する．ある時点のラティス構造テキストにおいてこのことが成立すると仮定して，変更や変更前に戻す操作が加えられて得られる新しいラティス構造テキストでも，このことが成立することが証明できる．

5 おわりに

本報告では，局所的かつ連続的に行われた操作系列をひとまとまりとして扱うことにより，ある程度意味的にまとまりを持つ変更という，より粒度の大きな操作にまとめる方法を示した．さらに，得られた各変更について，変更前の状態を参照したり回復したりすることを容易に行えるようなインタフェースを示し，これを実現するのに適した変更管理法として，ラティス構造で管理する方法を示した．

今後は，得られた各変更が実際に意味的なまとまりとなっているのかということや，システムの提供するインタフェースが実際に使い易いものなのかという点について，実際にユーザに利用してもらうなどして，評価実験を行う必要がある．

参考文献

[1] R. M. Stallman: GNU Emacs Manual (Free Software Foundation, Inc, 1987).

[2] Y. Yang: Anatomy of the Design of an Undo Support Facility, *International Journal of Man-Machine Studies*, **36-1**, pp.81–95, 1992.

[3] A. Prakash and M. J. Knister: Undoing Actions in Collaborative Work, *Proceedings of ACM CSCW'92 Conference on Computer-Supported Cooperative Work*, pp.273–280, 1992.

[4] R. H. Trigg and M. Weiser: TEXTNET: A Network-Based Approach to Text Handling, *ACM Transactions on Office Information Systems*, **4-1**, pp.1–23, 1986.