ペンインタフェースを用いた視覚的な Lisp 教育環境

原 謙治[*]　五十嵐 健夫[**]

[*]東京大学理学部情報科学科　[**]東京大学大学院情報理工学系研究科

[*] [**]{hara2001,takeo}@ui.is.s.u-tokyo.ac.jp

概要

従来のテキストエディタにおける Lisp プログラムは多くの括弧を用いて書かれており、プログラムの木構造が見えにくいものとなっている．また、テキストエディタで編集している様子を見ても何をしているのかがわかりにくいという問題も存在する．我々はそうした問題を解決するためにペンインタフェースを用いた視覚的な Lisp プログラミング環境を開発した．このシステムにおいては、ユーザはジェスチャを用いて、可視化された木構造のノードに対して追加、削除、開閉操作を直感的に行うことができ、編集している人の意図の推測しやすさも期待される．

1．はじめに

現在一般的な Lisp プログラミング教育はテキストエディタを用いて行われている．しかし、テキストエディタにおいて多重にネストした Lisp の式を表示した場合には括弧が多くなり木構造が見えにくく、キーボードタイピングに慣れていない人にとっては文字入力もしにくい．

本論文では Lisp プログラムの木構造を直感的に表示し、初心者でもペンを用いて容易に木構造を作成できるツリーエディタを提案する(図 1)．このツリーエディタにおいてユーザは木構造を直接編集してプログラムを作成できる．木構造の編集はドラッグ＆ドロップに基づいたジェスチャによって行うことができ、直感的なノード追加、ノード削除、ノードの開閉、ノードのコピー＆ペースト、ノードのプログラム実行が可能となっている．また、文字入力方法には Graffiti[5]の使用が可能となっており、初心者に対してキーボードタイピングの習得は要求しない．

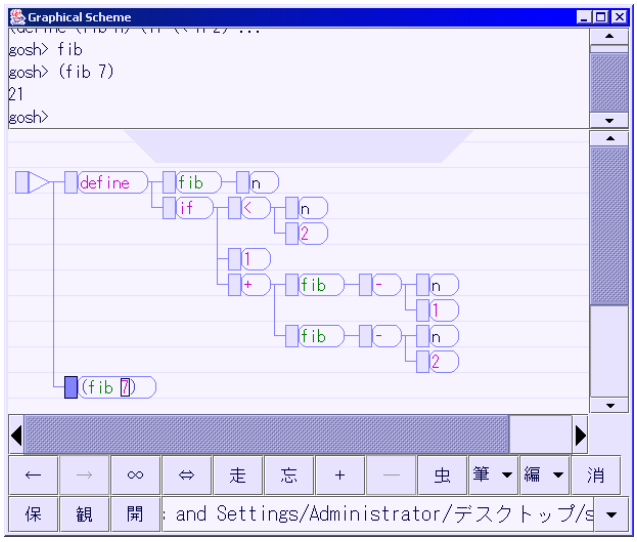

図 1. 可視化された木構造とプログラム実行の様子

2．関連研究

ペンを用いたプログラミング環境の研究としては Hyperflow[1]が挙げられる．Hyperflow はペンインタフェース、オブジェクト指向、データフロー型、といった特徴を持った教育用ビジュアルプログラミング言語である．また、ダイアグラム編集におけるペンとマウスの操作性の違いについての実験では、ペンの方がマウスより 2 倍早い操作性を得られるという結果が出ている[3]．ビジュア

Pen-based Visual Programming Environment for Teaching Lisp
Kenji Hara[*],Takeo Igarashi[**]
[*]Department of Information Science,Tokyo Univ.
[**]Graduate School of Information Science and Technology, Tokyo Univ.

ルプログラミング言語におけるダイアグラムの入力方法については、ペンのみとマウス＆キーボードで実験している[2]．この実験においてはモード切替はジェスチャで行うか、入力デバイス選択で行うべきであると結論付けている．また、コピー＆ペースト操作の重要性とジェスチャかポップアップメニューを用いることの有効性を主張している．Graffiti については MacKenzie らがその覚えやすさについて実験を行っており、5 分の練習の後に 95%以上の精度での入力が可能となっている[4]．また、Byrne らはアルゴリズム・アニメーションの効果について実験を行い、ただアニメーションを用いるだけではあまり効果は得られないと結論付けている[6]．本論文では Lisp 系言語 Scheme のビジュアルプログラミング環境における直感的なインタフェースを提案する．

3．インタフェース

本ツリーエディタの操作インターフェースには「ペン、マウス＆キーボード、キーボード」のうちから１つをユーザが選択することができる．ここではペンかマウスを用いたジェスチャによる木構造の編集操作、キーボードショートカットによる木構造の編集操作の特徴について説明する．

3.1．ジェスチャ操作

以下では木構造の編集におけるカーソル移動、ノード追加、ノード削除、ノードの開閉、文字入力、コピー＆ペースト、プログラム実行、スケッチ描画のジェスチャ操作を具体的に説明する．

(1)カーソル移動

クリックした位置にカーソルが移動可能であればその位置へのカーソル移動が行われる．

(2)ノード追加

ノードの追加はノードの外から中へのドラッグ操作で行う．ノードの外から中へドラッグすると、追加されるノードが青色で表示される．この時ペンを離すと赤色で表示されていたノードが追加される．(図 2-a,2-b,2-c)

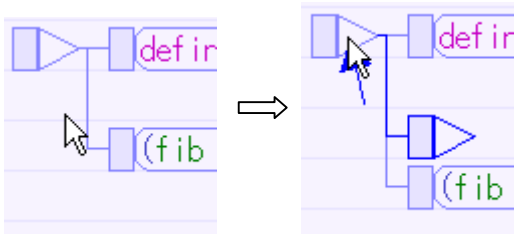

図 2-a. **ドラッグ開始**　　**図** 2-b. **ドラッグ中**

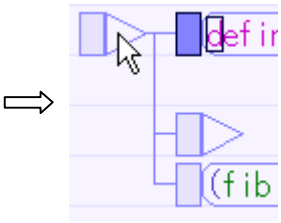

図 2-c. **ドラッグ完了**

(3)ノード削除

ノードの削除はノードを繋ぐ枝を横切るようなドラッグ操作で行う．枝を横切るようにドラッグを行うと削除されるノードが赤色で表示される．そこでペンを離すと赤色で表示されていたノードが削除される．(図 3-a,3-b,3-c)

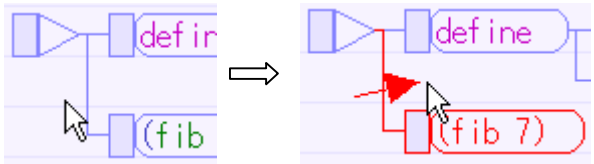

図 3-a. **ドラッグ開始**　**図** 3-b. **ドラッグ中**

図 3-c. **ドラッグ完了**

(4)ノードの開閉

ノードの開閉はノードの先端にあるヘッダ（ノードのすぐ左の部分）から左右にドラッグを開始することによって行う．ドラッグを開始すると縦に紺色の線が表示される．選択されているノード以下にあるノードのうち、その線より左側にあるノードは開かれ、右側にあるノードは閉じられる．(図 4-a,4-b,4-c.4-d)

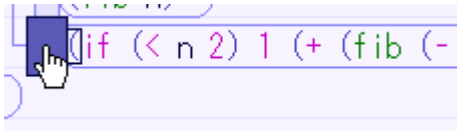

図 4-a. **ドラッグ開始**

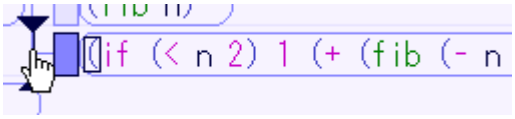

図 4-b. **ドラッグ中** 1

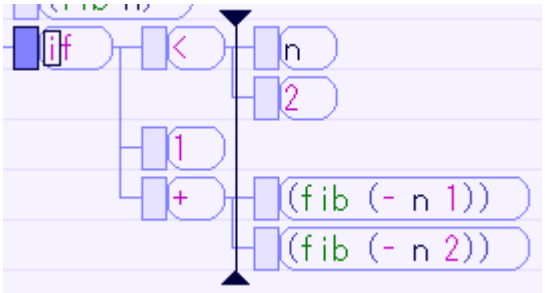

図 4-c. **ドラッグ中** 2

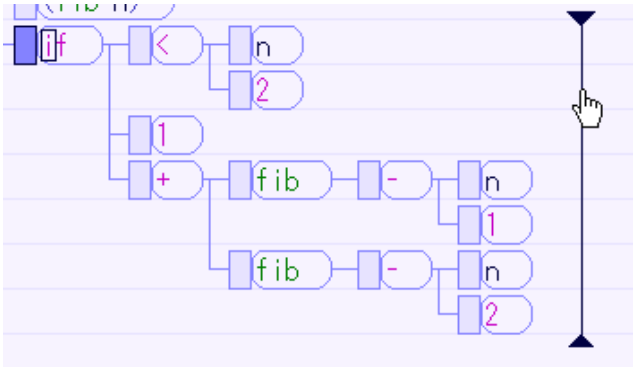

図 4-d. **ドラッグ中** 3

(5)文字入力

ペンを用いた文字入力はノードをタップすることで現れる半透明の入力スペースに Graffiti を用いて入力を行う．(図 5-a,5-b)

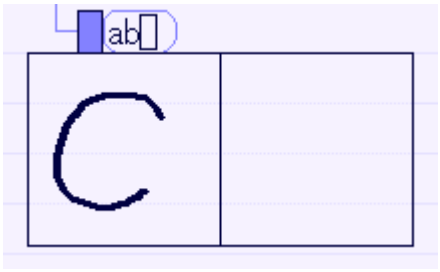

図 5-a. **文字入力中(‘ c ’を入力中)**

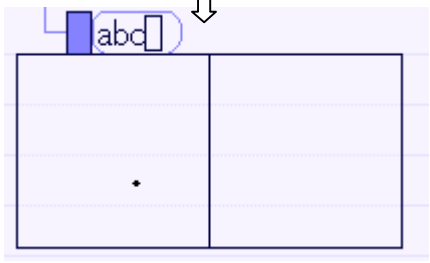

図 5-b. **文字入力完了**

また、左下から右上へ線を引く補完ジェスチャを用いて文字入力数を減らすことも可能である(図 5-c,5-d)

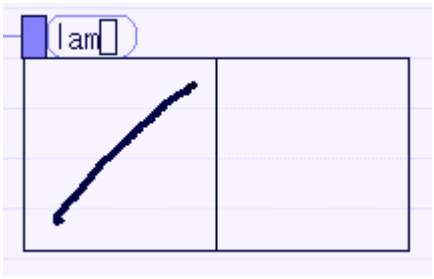

図 5-c. **補完ジェスチャ**

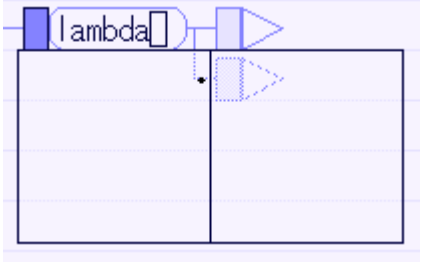

図 5-d. **補完完了**

(6)コピー&ペースト

コピー&ペーストはノードからノードへのドラッグ&ドロップで行う．ノードの中からドラッグを開始するとそのノードの背景が暗く表示される．この状態で別のノードの上へドラッグすると、ポインタがドラッグ&ドロップ可能を表すアイコンで表示される．この時ペンを離すとドラッグを終了したノードの名前がドラッグを開始したノードの名前になる（コピーされる）．（図6-a,6-b,6-c,6-d)

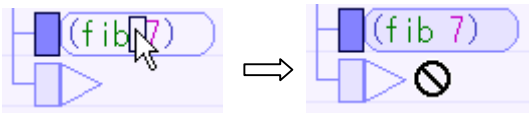

図 6-a. **ドラッグ開始**　**図** 6-b. **ドラッグ中** 1

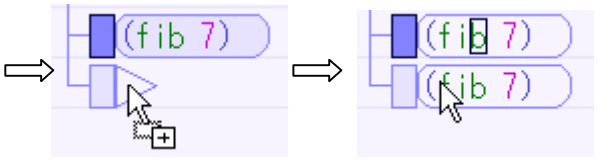

図 6-c. **ドラッグ中** 2　**図** 6-d. **ドラッグ完了**

(6)プログラム実行

プログラムの実行はノードからインタプリタ出力領域へのドラッグ&ドロップで行う．コピー&ペーストと同様にノードから上部のインタプリタ出力領域の上へドロップすると、ドラッグを開始したノードがインタプリタに渡されて実行される．(図7-a,7-b,7-c)

(7)フリーハンド・コメント

スケッチモードに移行することで、木構造の背景にフリーハンド・コメントを描画することができる(図8)．テキストのコメントよりも自由度の高いコメント記述が可能なことから、生徒の理解に対してより効果的なコメント作成が可能になると期待される．

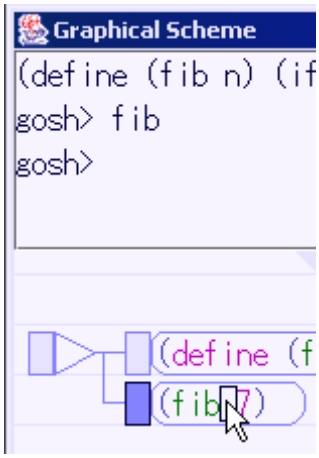

図 7-a. **ドラッグ開始(実行プログラムは“**(fib 7)**”)**

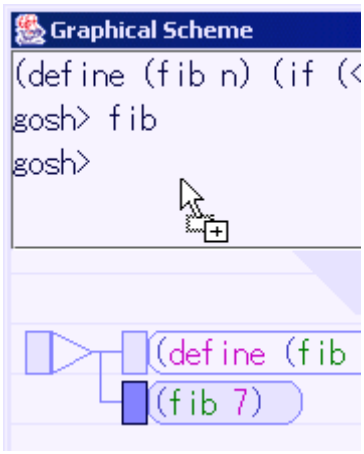

図 7-b. **ドラッグ中**

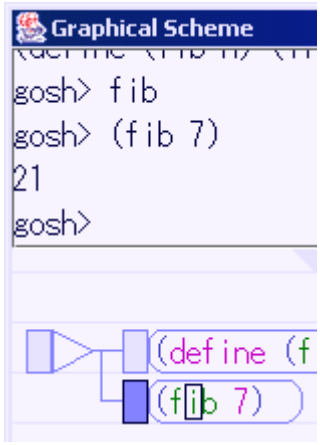

図 7-c. **ドラッグ完了(結果は“**21**”)**

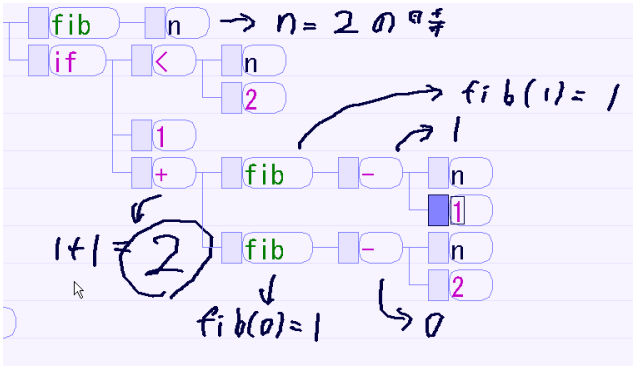

図 8. **コメント例**

3.2. キーボード・ショートカット

本ツリーエディタにおける木構造の編集はキーボード・ショートカットを用いることでキーボードから手を離さずに行うことも可能である．この場合、プログラム作成に要するキータイプ数はテキストエディタの場合とあまり変わらないものとなっており、高速な編集速度が期待できる．

4. 実装

本ツリーエディタは特定の Scheme 処理系には依存していない．この非依存性により、ユーザは必要に応じた処理系を選択してより自由度の高いプログラミングができる．同様に、Common Lisp のシンタックスでプログラムを書き、Common Lisp 処理系を用いることで Common Lisp プログラミングをすることも可能である．

5. 議論

従来のテキストエディタを用いたプログラミングと比較した時の本手法のメリットとしては 1)キーボードを用いないペンのみでのプログラミングが可能であること、2)プログラム構造が可視化された木構造で表示されているので直感的に理解しやすいこと、3)ペンで操作している様子を見てソフトウェアの使い方が覚えやすいこと、4)プログラムを編集している様子を見て何をしようとしているかが分かりやすいこと、が挙げられる．

本論文で提案した手法は、プログラミング教育用のインタフェースとしての使用を想定している．従来のテキストエディタを用いたプログラミング教育においては、特にキーボードタイピングに慣れていないユーザは授業以外にキーボード入力も気にしなければならず、また、教師の書いたプログラムを写すだけでは深い理解が得にくいという問題もある．しかし、我々のシステムにおいてはキーボードタイピングに気を使う必要はなく、教師のプログラム作成過程の分かりやすさ、作成されたプログラムの分かりやすさ、覚えやすさも期待される．

また、ペンのみで操作できることから巨大なタッチパネルディスプレイを用いて教師が説明し、生徒がタブレット PC で板書(プログラム)を写すというスタイルでの授業も可能となる．我々のシステムにおいてはプログラムの木構造が可視化されているので、遠くからプログラムを見た時も括弧の対応で悩むことはなく、プログラムも写しやすい．このスタイルは現在一般的な授業が行われているスタイルとの差異が小さいので自然な授業を行えることが期待される．

本手法の問題点としては、編集速度の遅さが挙げられる．キーボード操作の場合、習熟度が増せば非常に高速なプログラム編集を行うことができる．しかし、ペン操作の場合熟練してもキーボードほどの高速な編集は行えない．授業中はこの問題はさして大きくないが、独習中は高速な編集速度が要求されると考えられる．我々はテキストエディタを用いた場合とあまり変わらないキータイプ数で扱えるようにシステムを設計したが、その操作方法をいつ覚えるかという問題も存在する．
今後はシステムの使いやすさ、プログラミングの分かりやすさについて評価実験を行い、それに基づいたさらなる改良を行っていきたいと考えている．

参考文献

[1] Kimura, T.D. “Hyperflow: A visual Programming Language for Pen Computers,” Proceedings of 1992 IEEE Work shop on Visual Lunguages, Seattle, Washington, September 1992,pp 12.5132.

[2] Citrin, W., "Requirements for Graphical Front Ends for Visual Languages", IEEESymposiumon Visual Languages, pp. 142-150, Bergen, Norway,August 1993.

[3] Apte, Ajay and Takayuki Dan Kimura. A Comparison Study

of the Pen and the Mouse in Editing Graphic Diagrams. *Proceedings of Visual Languages 1993 Conference*, Bergen, Norway, 1993.

[4] I. Scott MacKenzie and S. Zhang. “The Immediate Usability of Graffiti”. Proceedings of Graphis Interface '97, Canadian Information Processing Society. 1997. pp. 129-137.

[5] Palm Products - How to Enter Data into a Palm Handheld. 2003. Available from http://www.palm.com/products/input/.

[6] Catrambone, R., Stasko, J. T., & Byrne, M. (1996). Do algorithm animations aid learning? TR GIT-GVU-96-18. GVU Center, Georgia Tech, Atlanta, GA.