筑波大学大学院修士課程

理工学研究科修士論文

単一モードに基づくビジュアルプログラミングシステムの構築

遠 藤 浩 通

平 成 11 年 2 月

筑波大学大学院修士課程

理工学研究科修士論文

単一モードに基づくビジュアルプログラミングシステムの構築

著者氏名　　遠藤 浩通

主任指導教官　電子・情報工学系 田中 二郎

目次

要旨

本研究では、視覚的にプログラミングを行なうシステムの構成法として、従来のシステムでは別々に扱われていたプログラムの論理的構造の表現とその視覚的表現の両方を統合して扱う方法について提案する。また、その枠組みを用いて実装されたプロトタイプシステム LivePP-proto について述べ、プログラムの編集操作と実行をモードの切り替えなしに並行して行なえることを示す。

また、 LivePP-proto で実装された単一モード統合環境を強化して、プログラムを自由かつ直感的に加工できるプログラミングシステムを構築する手法として、プログラムの実行過程そのものも含めたプログラムの加工を行なう方法の考察と行ない、その有効性について述べる。

第 1 章

はじめに

近年の計算機の急激な能力向上を背景として、ビジュアルプログラミングシステムと呼ばれるプログラミング環境の研究が多く行なわれている。これは、「図形・アイコンなどの視覚的表現を用いてプログラムを表し、またそれらを操作することによってプログラムの編集・実行を行なうための環境」と定義される [9][13]。ビジュアルプログラミングシステムは、テキストのみによる表現と比較して、

- 構成要素が図形などで表現されており、プログラムの構造を直接視覚によって認識し、直感的に理解することができる。
- 対象プログラムは 2 次元、あるいはそれ以上の空間内に展開されるため、プログラムの全体構造を把握するのが容易である。

という利点がある。

しかし、従来のビジュアルプログラミングシステムではプログラムの視覚表現がより重視され、プログラムの操作に関して従来のテキスト環境よりも改善されているとは言えないものが多い。もともと、テキスト環境、特に主流であった手続き型言語によるプログラミング環境では、期待する処理結果と、それを得るためのプログラムのそれぞれの表現形式が異なるため、プログラムの編集をエディタで、実行をシェルやインタプリタというように異なる環境で行なっていた。

既存の多くのビジュアルプログラミングシステムでも、専用のエディタ上で視覚化されたプログラムを編集し、トレーサなどによってその実行過程をユーザに提示するという形態をとっているが、このような構成では、ユーザは 1 つのプログラムを扱っているにもかかわらず、エディタとトレーサを切り替えつつ操作を行なわなければならない。

このような構成は充分に直感的であるとは言えず、ビジュアルプログラミングシステムにおけるプログラムの操作環境として適当ではない。そこで、本研究では、従来のような「編集 $\rightarrow$ 実行」というように、複数のモードにわたって操作を行なうのではなく、単一のモードのみでプログラムの操作を行なうことのできるビジュアルプログラミングシステムの構築を目的とする。

本論文では、第 2章で背景を述べたのち、第 3章でプログラムの論理的構造の表現と視覚表現の統合について述べる。次に、第 4章において、この統合されたプログラム表現を用いて実装されたシステム LivePP-proto の詳細について説明する。また、LivePP-proto を発展させ、さらに直感的なプログラミングを可能とするための操作環境について第 5章で考察を行なう。

第 2 章

ビジュアルプログラミングシステム

この章では、本研究の背景となるビジュアルプログラミングシステムについて述べる。

2.1 ビジュアルプログラミングシステムの言語パラダイム

ビジュアルプログラミングシステムにおいてプログラムを記述するための枠組みは、しばしば「ビジュアル言語」と呼ばれている。

テキストベースのプログラミング言語と同様に、ビジュアル言語も言語パラダイムによっていくつかに分類することができる。代表的なものとして、 Burnett らによる分類 [6] を挙げることができる。

ビジュアルプログラミングシステムを分類する主要な項目の 1 つに言語パラダイムがある。プログラムの形態や表現法などによる分類であるが、我々は、プログラムを視覚的に記述する言語パラダイムとして、宣言型言語、その中でも特に並列論理型言語と呼ばれるものに注目している。これらのパラダイムでは、論理型言語と同様に、意味を持った基本要素によってデータを構成し、それらの間に成り立つべき関係規則をプログラムとして定義する。

並列論理型言語は、次のように、ビジュアル言語に適用する場合に有利な特徴を持っている。

- 少数の基本要素のみによって言語が構成されており、非常にシンプルである。
- プログラムとデータが同一の表現法で記述できる。

  これらの特徴は、視覚化した際に必要以上に表示が複雑化するのを防ぎ、ユーザの認知的負荷を軽減するのに有利である。手続き型言語では一般に、プログラムの記述に必要な基本要素の種類が多かったり、データとプログラムの表現が異なったりするために、プログラムの規模の増加に伴なって表示が急速に複雑化しやすい。

- プログラムの実行過程が記述順序に依存しない。

- 単一代入という概念によって、変数の値の共有関係がプログラム中で明示的に示されている。

  手続き型言語におけるプログラムの状態は、それまでに呼ばれた手続きや分岐の履歴などの時間的要素を含んでおり、それらを明示的に視覚化すると、結果として表示の自由度を 1 つ消費してしまう。宣言型の言語では、プログラムの状態は静的 (時間的要素に依存しない) であり、柔軟な視覚化が可能である。

これらの特徴から、ビジュアルプログラミングシステムの中には並列論理型言語をベースとしているものが多く存在する。

2.1.1 GHC

GHC[12] は、これらの並列論理型言語の 1 つである。論理型言語である Prolog と類似した言語構造を持っているが、ガードによる定義節の選択実行と、変数ごとの同期機構により、プログラムの並列実行を可能にしている。

GHC の定義節の構文は以下のようになっている。定義節とは、データ中に現われるゴールをサブゴールへと置き換える (リダクション) 規則を記述したものである。

```
述語名(<引数>, <引数>...) :- ガード節 | ボディ節.
```

GHC のプログラムは、このような定義節が複数集まって構成されている。プログラムの実行は、データ中のゴールを、それと同じ述語名を持つ定義節を用いてリダクションすることによって進む。最終的にすべてのゴールがリダクションに成功するか、あるいは途中で失敗するとそこで実行は終了する。

":-" から "|" の間に記述される節は「ガード節」と呼ばれ、ゴールのリダクションの際、この部分の評価に成功した定義節が選択される。また、ガードで参照している変数の値が未定義であった場合、ここで実行が一時中断されるため、自動的に同期がとられるようになっている。

2.2 関連研究

2.2.1 ビジュアルプログラミングシステム PP

我々は、ビジュアルプログラミングシステムを構築する試みとして、並列論理型言語 GHC の構造、および実行過程を視覚化するシステム PP の研究を行なっている [4][1]。これまでに、プログラムの編集や実行過程の表示などにおける表示方式や操作環境、といった要素技術について研究を行なっている。

2.2.2 その他の研究

Pictorial Janus

Pictorial Janus[10] は、並列論理型言語 Janus を視覚化し、実行させるシステムである。汎用のグラフィックエディタで描画した図形をプログラムの視覚表現として用いることが可能である。実行過程は、 Janus の処理系によって得られた実行結果を視覚化して表示される。

Visulan

Visulan[3] は、ビットマップの置換を用いたビジュアルプログラミングシステムである。画面上のビットマップ領域に、置換規則 (プログラム) を表すビットマップと、書き換えの対象となる (データ) ビットマップをそれぞれ定義する。プログラムの実行は、データビットマップを、定義された置換規則を適用して書き換えることによって行なわれる。

Forms/3

Forms/3[7] は、スプレッドシートの概念を取り入れたビジュアルプログラミングシステムである。 Form と呼ばれるプログラム領域にセルと呼ばれるオブジェクトを貼り付け、各セルに、セル間で満たされるべき制約を書く。セルには、図や絵などを使うことができるので、セルの記述によってこれらのオブジェクトを制御することができる。

ToonTalk

ToonTalk[11] は、子供でも容易にプログラミングを行なえることを目的としているビジュアルプログラミングシステムである。演算や操作といった要素を漫画のキャラクタに見立て、それらが動く過程がプログラムとなっている。プログラミングは、すべてその漫画の世界で行なわれるようになっている。

第 3 章

プログラム構造とその視覚表現の統合

プログラムを統合環境上で扱うためには、システム内部でのプログラムの表現そのものについても検討する必要がある。本章では、一般的なシステムにおけるプログラムの表現についての問題点を挙げ、それに対する解決として、論理的構造とそれに対する視覚表現を統合した表現を提案する。

3.1 従来のプログラム表現

既存のビジュアルプログラミングシステムでは、プログラムの編集をエディタで行ない、それをトレーサ上で実行させて過程を得る構成になっているものが多い。

このようなシステムの場合、ユーザのプログラムは内部表現の形で各サブシステムに共有され、図 3.1のように、サブシステムが内部表現から視覚表現を生成してプログラムの編集・実行に用いる。

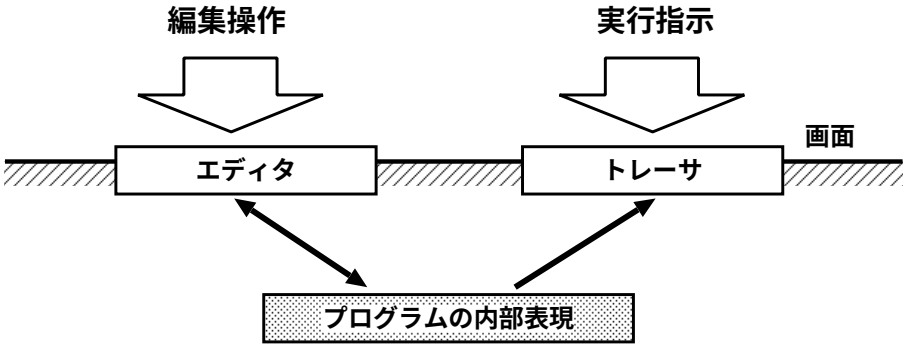


図 3.1: 一般的な構造

これらのシステムでは、サブシステム間の結合が弱いため、１つのサブシステムにおけるプログラムの構造の変化が残りのサブシステムへ反映されにくく、システム全体としてその変化に直ちに追従することができない。ビジュアルプログラミングシス

テムでは、ユーザの操作に対し、システムがそれに対して適切な反応を直ちに返すというインタラクティブ性が重要であるが、このようなプログラム表現を用いるシステムでは、高いインタラクティブ性の確保は困難であると考える。

3.2 プログラム構造と視覚表現の統合

本研究では、従来のようにプログラム全体の内部表現から視覚表現を生成するという縦割りの構造ではなく、プログラムの各構成要素と 1 対 1 に対応するオブジェクトを用い、それら自身が視覚表現の生成を行なうようなアプローチをとる。

構成要素に対応するオブジェクトを「プリミティブ」と呼ぶことにする。各プリミティブは図 3.2に示したように、プログラムの論理的な構造を表す部分と、それに対応する視覚表現を画面上に描画したり、ユーザの操作を扱う、いわゆるユーザインタフェース (UI) 部からなっている。プログラムはプリミティブの集合、およびそれらの間の関係として表すことができる。

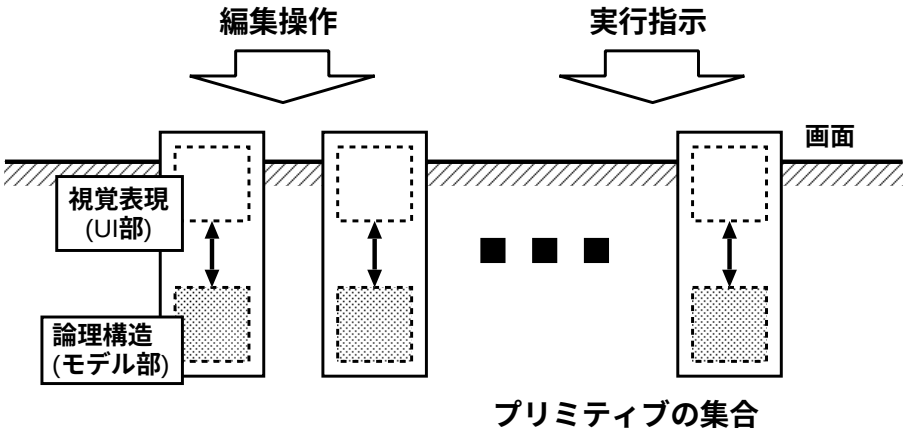


図 3.2: モデル部と視覚的表現を結合した構造

各構成要素の視覚化は、それぞれに対応するプリミティブの UI 部が、そのプリミティブ自身の状態を視覚表現として画面上に描画することによってなされる。全体としてみると、プログラムの構造が視覚表現の形で画面上に投影されているように見える。プログラムの構造に変化が生じると、それはただちに視覚表現の変化として画面上に現われ、逆に、視覚化されたプログラムに対してユーザが操作を行なうと、それはプリミティブの UI 部を通じてプリミティブの状態を変化させることになり、結果としてプログラムの構造を直接操作することになる。

第 4 章

LivePP-proto

本章では、単一のモードでプログラムを扱うビジュアルプログラミングシステムのプロトタイプとして実装したシステム “LivePP-proto” について述べる。

4.1 概要

LivePP-proto は、本研究において目標としている、単一のビュー上においてプログラムの編集操作および実行などの必要な操作を行なうことのできるビジュアルプログラミングシステムのプロトタイプとして実装したものである [2][8]。

図 4.1に、 LivePP-proto の概観を示す。 LivePP-proto には、ビューと呼ばれる領域が存在し、そこにプログラムが表示されている。プログラミングに関するほとんどの操作はこの上だけで行なわれる。また、プログラムを 3章で述べたようにプリミティブの集合として扱っているため、プログラムの視覚表現への直接操作によって、プログラムの構造自体を操作することができる。

4.2 設計

4.2.1 IntelligentPad

LivePP-proto において、 3章で述べたプログラムと視覚表現の統合を実装する手段として、北海道大学で開発された IntelligentPad[14] を用いた。

IntelligentPad は、パッドと呼ばれる基本要素を組み合わせ、アプリケーションを構成していくシステムである。作ったアプリケーションは、動作させたまま新たにパッドを追加したりしてその構造を変えることができる。

それぞれのパッドは、図 4.2に示すような 2 層からなる構造になっている。モデル部は、パッド自身や、それを利用するアプリケーションの内部状態を格納する部分であり、また、他のパッドとデータをやりとりするための窓口という役割を持っている。

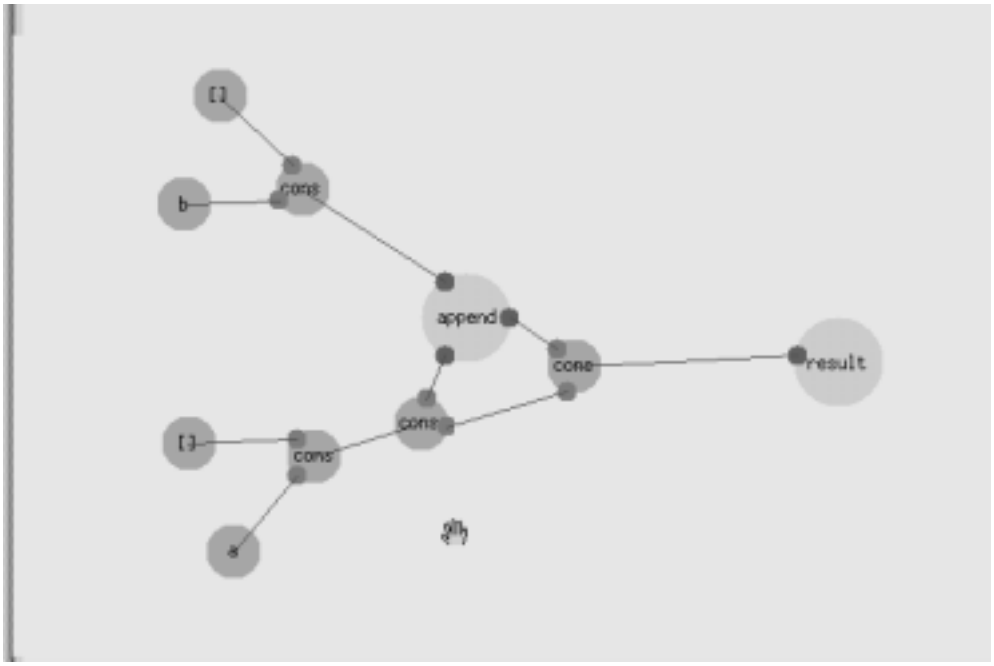


図 4.1: LivePP-proto

UI 部[†1]は、パッドの外観を画面に描画したり、ユーザの操作によるパッドの状態の変化をモデル部に伝える働きをしている。

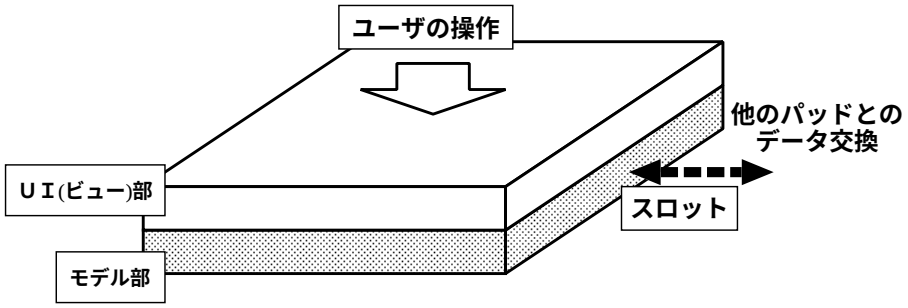


図 4.2: パッドの構造

パッドに対してユーザの操作が加わると、 UI 部によって操作イベントの情報がモデル部に伝えられ、それに従ってモデル部の状態が変わる。その結果を他のパッドに伝達したい場合は、スロットと呼ばれる機構を通じ、相手のパッドのモデル部にデータを伝えることができる。また逆に、他のパッドから受け取ったデータなどによりモデル部の状態が変化した場合はモデル部から UI 部に更新メッセージが送られる。 UI

[†1]IntelligentPad システムでは、この部分はビューと呼ばれることが多いが、ここでは LivePP-proto のビューとの混同を避けるために「UI 部」として説明する。

部は、モデル部の内部状態を用いて自分自身の描画を行なう。

このようなパッドの構造は、 3章で述べた、プログラムの論理的構造とその視覚的表現を 1 つにまとめたプリミティブの概念を実装するのに適している。そこで、LivePP-proto では、プリミティブの機能を実装したパッドを新たに定義して、それらの組み合わせでプログラムを表現することとする。

4.2.2 視覚表現の設計

LivePP-proto では、視覚化の対象とする言語として、並列論理型言語の 1 つである GHC を採用した。 GHC のプログラムとデータ構造は

- アトム
- ファンクタ
- 述語

という基本要素から構成されている。これらについて、それぞれ図 4.3のような視覚表現を定義する。

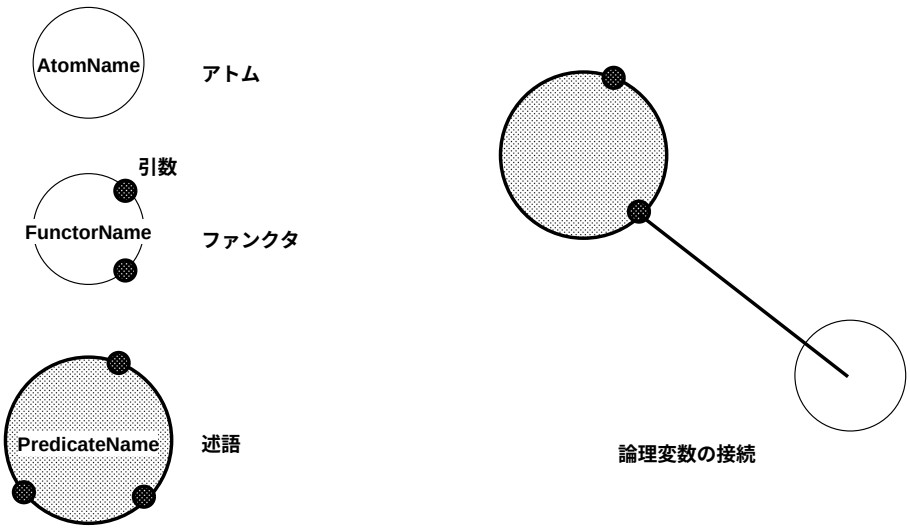


図 4.3: プリミティブの視覚表現

4.2.3 LivePP-proto によるプログラミング

実装したプロトタイプシステム LivePP-proto では、一般的なビジュアルプログラミングシステムにおける操作環境のサブセットに当たる、以下のような操作をサポートしている。

- プログラムの作成・編集

- プログラムの実行
- プログラムのファイルへの保存・ファイルからの呼び出し

本節では、 LivePP-proto におけるプログラミングの手順を、実際の操作例を用いて説明する。なお、詳細な操作方法は付録 Aに収録した。

プログラムの作成

LivePP-proto でプリミティブを生成するには、図 4.4で示したように、メニューからプリミティブの種類を選択することで行なう。このとき、プリミティブにつけるシンボル (名前) の入力を求められる。生成されたプリミティブはビュー上に配置され、マウスの左ボタンでドラッグして自由に動かすことができる。削除・コピーなどの操作はマウスの右ボタンを押すと出るメニューから行なう。

あるプリミティブまたはその引数どうしを論理変数で接続するためには、マウスの中央ボタンを使用する。図 4.5のように、始点、終点となるプリミティブまたはその引数の上でそれぞれクリックすると、それらの間に論理変数を表す線分が生成される。

図 4.4: プリミティブの生成

プログラムの実行

4.2.3で述べた操作を用いてプログラムを作成した後、プログラムに対して必要なデータを与えることにより、自動的にプログラムの実行を開始させることができる。

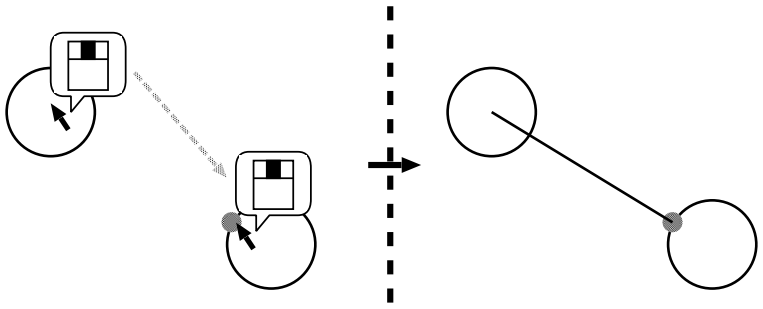


図 4.5: 論理変数の接続

例として、次に示す GHC のプログラムを LivePP-proto 上で作ることを考える。

```
:- module main.

main :- X = [a], Y = [b], append(X, Y, Z), result(Z).
```

サンプルプログラム

append/3[†2]は 2 つのリストを連結して出力する述語であり、次のように定義されている。

```
append(IN1, IN2, OUT) :- IN1 = [A | B] |
                         append(B, IN2, C), OUT = [A | C].

append(IN1, IN2, OUT) :- IN1 = [] |
                         OUT = IN2.
```

まず、ビュー上にプリミティブを配置し、いくつかを論理変数で結んだ状態を図 4.6に示す。ビューの中央の円形が append/3 を表すゴールプリミティブである。また、ビューの左側に 2 つある、 3 つのプリミティブが接続ざれたものは入力となるリストである。そして、ビューの右側のプリミティブは、 append/3 の出力を保持するためのダミーゴール result/1 である。

次に、ゴールプリミティブの 3 つの引数と、 2 つの入力リスト、そしてダミーゴールを、図 4.7のようにそれぞれ論理変数で接続する。これらがすべて接続された時点で初めてさきほどのサンプルプログラムが完成して、直ちにその実行が開始される。まず、図 4.8のように、 append/3 のリダクションが発生し、サブゴールがビュー上に展開する。

[†2]名前の後の “/ 数字” は引数の個数を表している。 GHC では、名前が同じでも引数の個数が違うと別のものとして扱われる。

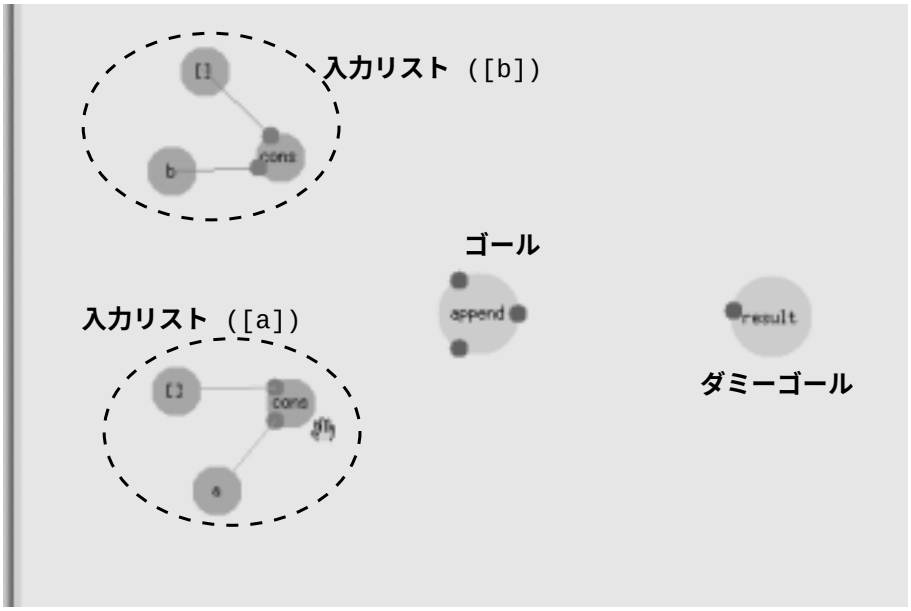


図 4.6: プログラム実行例: 初期状態

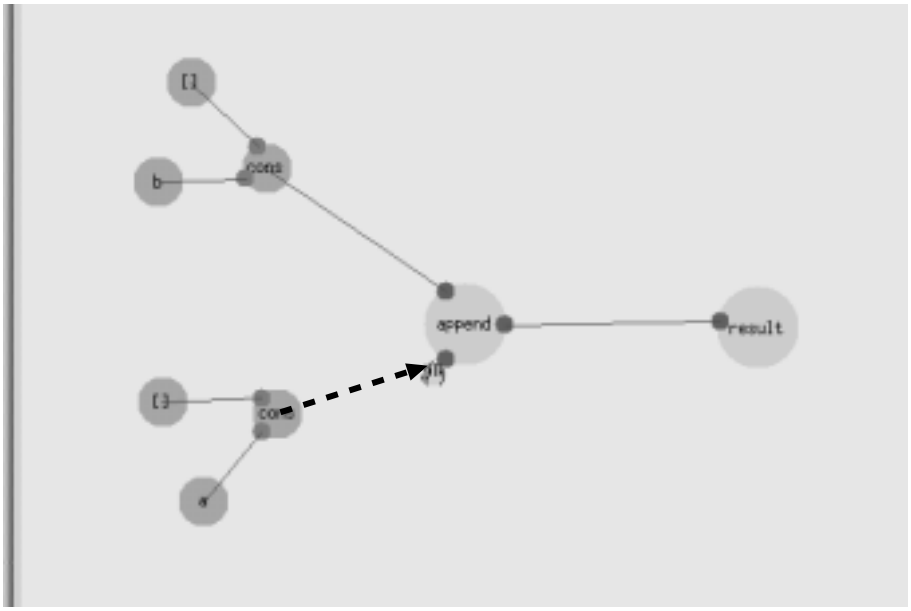


図 4.7: プログラム実行例: 論理変数を接続する

ゴールの展開が終了すると、次に、論理変数で接続されているプリミティブどうしで単一化が起こる。図 4.9では、図 4.8上に矢印で示された部分が単一化されている。その結果リダクションが可能になったゴールがあれば、さらにサブゴールの展開が行われる。

このようにして、ゴールの評価に成功している間はサブゴールの展開と単一化が反復され、評価に失敗するとそこでプログラムの実行は停止する。サンプルプログラム

の最終状態を図 4.10に示す。 [a] と [b] という 2 つのリストが連結され、 [a,b] というリストになっていることがわかる。

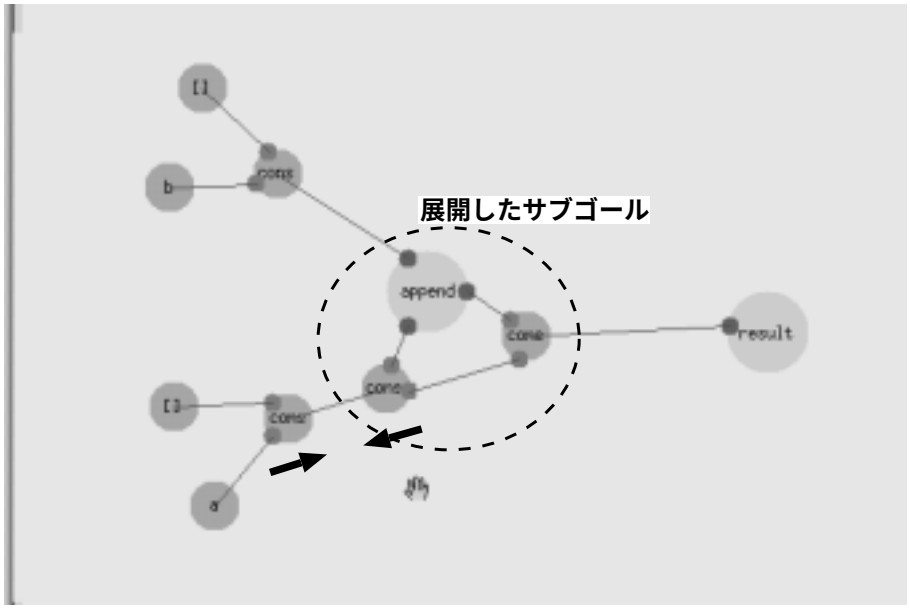


図 4.8: プログラム実行例: 実行開始

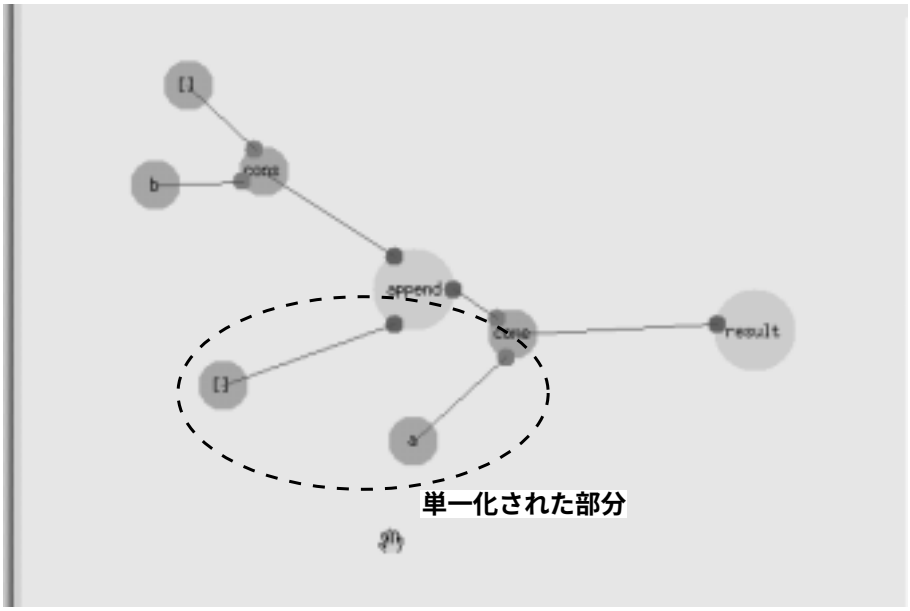


図 4.9: プログラム実行例: 単一化直後

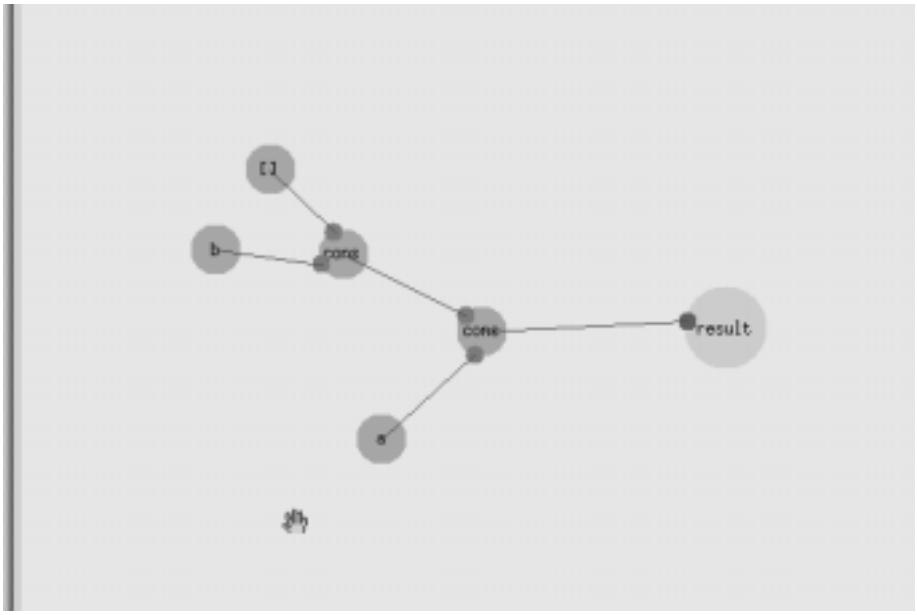


図 4.10: プログラム実行例: 最終状態

実行中の編集

プログラムが実行されている間も、ユーザは自由にプリミティブの生成などを行なうことができる。また、実行の一時停止を指定しておいて、実行中のプログラム自体を再編集することも可能である[†3]。一時停止を解除すると、再編集を行なった状態からプログラムの実行が再開される。

プログラムの保存・呼び出し

ビュー上で編集したプログラムは、ビューを右クリックすると出るメニューから、“Save” を選択することによってファイルに保存することができる。

ファイルに保存したプログラムを呼び出すには、ビュー上部のメインメニューから “Operation” → “Load” と選択し、メニューから目的ファイルを選択すると、プログラムがビュー上に呼び出される。

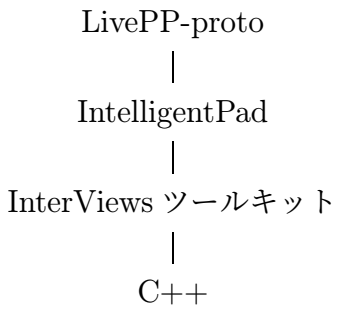
[†3]再編集操作自体はプログラムの実行を停止させることなく行なうことができ、プログラムの挙動を動的に変更することができる。

4.3 実装

本節では、 LivePP-proto におけるプログラムの実装の詳細と実行の機構について述べる。なお、下に引用した LivePP-proto のソースコードは、ベースである IntelligentPad 自身が C++ で書かれているのに合わせ、 C++ によって記述されている。

4.3.1 実装プラットフォーム

LivePP-proto の実装プラットフォームの階層構造は次のようになっている。



従って、以下に示す LivePP-proto のソースコードも C++ によるものであることを断わっておく。

4.3.2 プログラムの論理的構造

LivePP-proto におけるプログラムの論理的構造は、クラス `Node` によって表現されている。クラス `Node` の定義を以下に示す。

```
class Node {
  public:
    /* Node type */
    enum NodeType {Pred, Term, Base};

  public:
    Capsule* ID;
    NodeType type;
    char name[64];
    int arity;
    Coord size;
    IpXY* argPos;
    Edge** conns;
};
```

`type` はプリミティブの種類、 `name`、 `arity` はそれぞれ名前と引数の数である。また、 `conns` は、各引数から接続している論理変数へのポインタを保持するテーブルである。

2つのプリミティブを接続する論理変数は、クラス Edge において次のように定義されている。

```
class Edge {
  public:
    Node* node[2];
    int arg[2];
    Coord freeLength;
};
```

node[] と arg[] で、それぞれ両端に接続されるプリミティブと引数の番号を指定している。

各プリミティブは、このクラス Node のオブジェクトをメンバオブジェクトの形で保持しており、それらのオブジェクトが論理変数を介して互いに接続することで、プリミティブ全体がプログラムとしての構造をなしている。 (図 4.11)

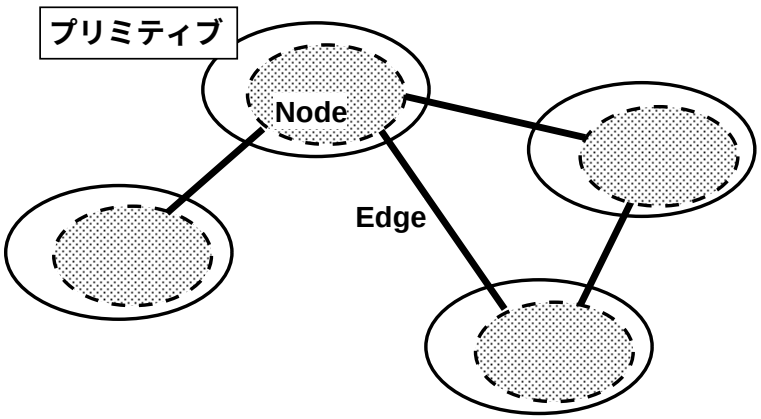


図 4.11: プログラムの論理的構造

4.3.3 実行の機構

LivePP-proto において、プログラムはプリミティブの集合という形をとっているが、プログラムの実行の主体もプリミティブ自身にある。プリミティブには、種類ごとに表 4.1のような実行時のふるまいのルールが定義されており、プリミティブどうしがメッセージのやりとりをしながらそれぞれのルールに従って動作することで、全体としてプログラムの状態が変化していく。

4.2.3で述べたように、 LivePP-proto では、プログラム中のゴールに対して引数として適切なデータを与えてやることで、自動的にプログラムの評価を開始させることができる。以下にその機構について述べる。

表 4.1: 各プリミティブの実行ルール

アトム ファンクタ	自分と接続している論理変数の先に自分と同じ種類のプリミティブがあった場合、自分とそのプリミティブとで単一化を試みる
ゴール	自分のすべての引数が具体化されたら (アトムまたは項が接続されたら) ガードのチェックを開始

1. ユーザがマウスの中ボタンを使い、プリミティブどうしを論理変数で接続すると、論理変数オブジェクトが生成され、システムに登録される。

2. 新たに生成された論理変数にゴールプリミティブが接続されていた場合、そのゴールは評価が行なわれる可能性があるとみなされ、リダクション判定候補のキューにも入れられる。

3. キューから出されたゴールに対し、リダクションが可能であるかを問い合わせるメッセージが発行される。メッセージを受け取ったゴールは、まずすべての引数がほかのプリミティブに接続されているかどうかチェックされる。接続されていない引数があった場合、そのゴールの評価は失敗する。

4. ゴールは次に、引数に接続されているデータを用い、自分自身のガードを評価する。これに成功した場合、システムに対してリダクション判定に成功したことを伝え、続いてサブゴールをビュー上に展開する。

5. 各論理変数に接続するプリミティブに対して単一化が行なわれる。ここまでのリダクションや単一化の結果生成されたプリミティブや論理変数があれば、これらもシステムに登録される。

6. リダクション候補のゴールがなくなるか、ゴールの評価に失敗するまで、 2 からの過程が繰り返される。

4.3.4 メッセージ交換の機構

4.3.3で述べた実行の制御など、 LivePP-proto ではプリミティブどうしのメッセージ交換によって動作の制御を行なう。メッセージは、 IntelligentPad に用意されているスロットというデータ交換のための機構を用いて交換されるが、各プリミティブがこれらのメッセージを仲介するために、ベースと呼ばれるオブジェクトを用いている。

図 4.1などの画面写真などで、一見背景に見えるものがベースである。ベースはプリミティブには属さないが、プリミティブと同様に、 IntelligentPad のユーザパッドとして実装されている。

図 4.12にベースの動作を表す。 LivePP-proto のすべてのプリミティブはこのベースにスロット結合しており、各プリミティブからのメッセージはいったんベースが受け取った後、メッセージの宛先となるプリミティブへ転送される。

ベースは、ビュー上に存在するプリミティブに関する情報の管理も行なっている。ビュー上にプリミティブが生成され、ベースと新規にスロット結合されるときに、プリミティブから ID などの情報がベースに送られ、リストに加えられる。

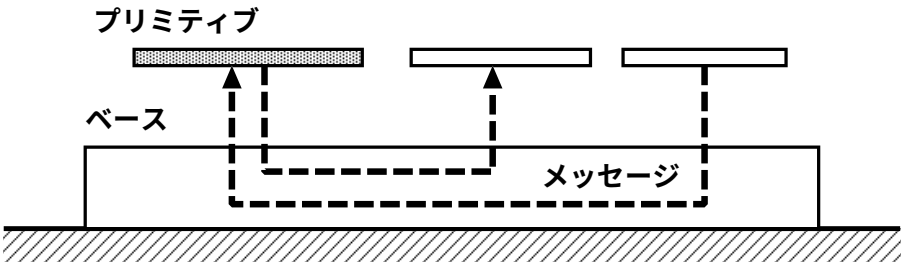


図 4.12: ベースの動作

4.3.5 ユーザインタフェース

LivePP-proto には、プリミティブ以外にも InteligentPad を用いて実装されている部分が存在する。ユーザの操作を処理する部分は主に、 IntelligentPad で提供されているマウスクリック、ドラッグなどのイベント処理メソッドの一部を利用している。また、ダイアログやメニューについては、 IntelligentPad の実装プラットフォームである InterViews ツールキットで提供されているものを再定義して使用している。

第 5 章

単一モード環境の発展

本章では、 LivePP-proto で実装されたプログラムの表現や実行環境に基づいたビジュアルプログラミングシステムを構成するに当たり、プログラムを自由かつ直感的に加工するための操作環境について考察する。

5.1 時間軸操作の自由化

時間軸操作とは、一般のシステムにおける undo や redo、ヒストリなどに相当するものであり、ここではプログラムの実行過程を操作することを指す。本節では、それらもプログラムの操作に含めて扱うことについて述べる。

5.1.1 時間軸操作の必要性

既存のビジュアルプログラミングシステムでは、与えられたプログラムを解釈・実行し、トレーサがその過程を視覚化してユーザに提示する。実行過程の中の任意の時点の状態を見るために、実行開始からの時間を表すスライドバーなどを用意して、それを動かすことで任意の時点の実行状態を見ることのできるシステムもある。

しかし、これらはいわば予め録画されたビデオを再生しているようなもので、早送りや巻戻しはできても録画されたものに手を加えることはできないのと同様、ユーザはトレーサが表示するプログラムを操作することはできない。

例えば、プログラムの実行中に期待と違う挙動が見られた場合を考える。当然そのような場合にはプログラムを修正する必要があるが、 3章で述べたような、トレーサによってプログラムの実行過程を視覚化して見せるようなシステムでは、ほとんどの場合、図 5.1のように、初期状態まで戻って実行をやり直さなければならない。そのようなシステムであっても、トレーサにヒストリ機構などを組み込むことで、いったんプログラムを実行して得た実行過程をもとにして任意の時点におけるプログラムの状態を得ることは可能である。しかし、そのようにしても、プログラムの編集後の実行は避けることができない。

プリミティブによって表現されたプログラムでは、ビュー上に存在するプリミティブの状態がそのままプログラムの状態を表し、そのときのビューの状態によって 1 ス

テップ実行後のビューの状態が決定する。また、ユーザはシステムとは独立に、任意の時点でビュー上のプログラムを操作することができる。従って、図 5.2のように、ユーザがプログラムの実行過程に介入してビューを操作することにより、プログラムの挙動を動的に変更することができる。

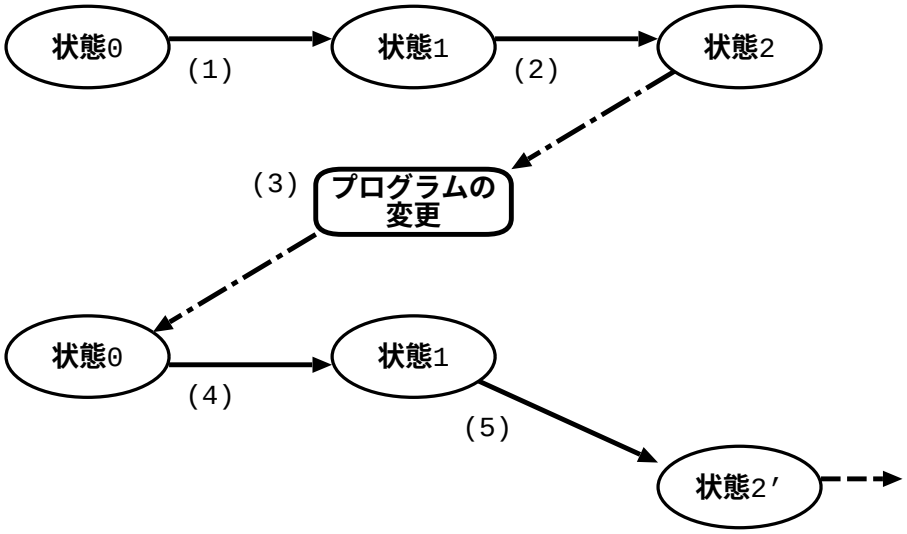


図 5.1: 既存のシステムにおけるプログラムの変更

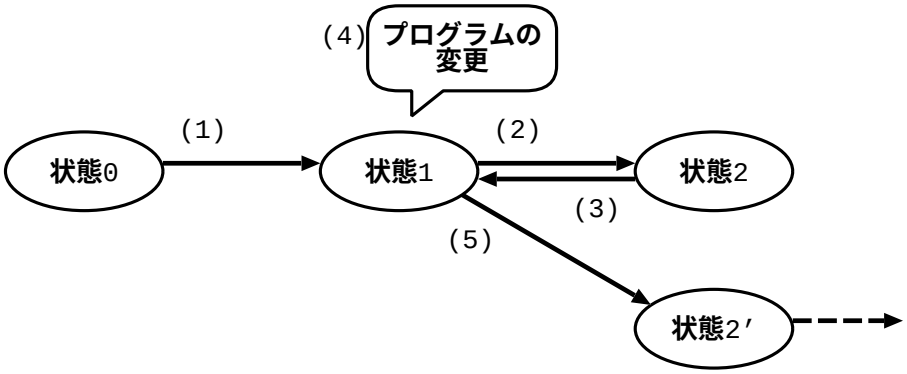


図 5.2: プリミティブからなるプログラムの変更

ただし、この場合に 1 度限りの変更しか許されないのであれば、自由にプログラムに介入できても意味はない。従って、この機構を有効に利用するために、プリミティブベースのシステムにも何らかの undo 機構を組み込んでおき、ユーザが実行過程の任意の時点にアクセスできるようにすることが重要である。また、消極的な理由であるが、実行中のプログラムに対しても介入が可能であるというのは危険なことであり、ユーザが不用意にプログラムを改変したために正常な動作ができなくなった場合に、それを修正できる機構が必要であるというものもある。

5.1.2 実行過程の管理

プログラムの実行過程を操作の対象とするためには、その過程中の各時点におけるプログラムの状態を保存しておく必要がある。プログラムの状態を保存する方法として、大まかに次の 2 通りが考えられる:

1. その時点でのプログラム全体の状態を保存する (スナップショット)

2. 変化のあった要素についてのみ、その情報を記録する

1. のスナップショットによる手法は実装が容易であるが、スナップショットの 1 つ 1 つは独立しており、「ステップ間でどこが変化したのか」という情報を持っていないため、 linear undo などの単純な undo 機構以外には適用しにくい。また、ステップ間で変化のなかった部分も保存するために、無駄が多いという問題点もある。

2. は、「どこが」「どう変わった」という、状態の差分を記録しておく方法である。 1 と違い、記録された情報から直接プログラムの状態を復元できるわけではないが、それらを用いることにより、部分的な undo などの高度な undo モデルを利用することができる。

そのような undo モデルのなかで、 LivePP-proto に適していると思われるのが、[5] で提案されている selective undo である。この手法は、複数のオブジェクトを操作するシステムにおいて、オブジェクトごと、あるいは操作の種類ごとに履歴を設定することができるというものである。通常の linear undo の場合、ある操作を取り消そうとする場合、そこから現在までにに行なわれた無関係な操作も一緒に取り消されてしまう可能性があるが、 selective undo では、直接目的とする操作を取り消すことができる。

LivePP-proto を構成するプリミティブごとに実行過程の各ステップで起こった変化を記録しておき、時間軸に対する操作を行なう際にそれらの情報を用いることによって、プログラムの実行過程に対する操作を行なうことができる。

5.2 プログラムの部分実行

プログラミングの過程では、ある程度の試行錯誤が生じることが予想される。ビジュアルプログラミングシステムでは、 2 次元 (あるいはそれ以上) 空間内にプログラムの構造を表現しているため、ユーザはテキストで書かれたプログラムよりも比較的容易にプログラムの構造を把握することができ、ソース上での変更は自由に行なうことができる。

実際には、プログラムのある部分を実行させて、その挙動を見ながら他の部分を作成していきたいという場合がある。特に、プログラミングの経験の少ないユーザは、プログラムが視覚化されていても、その挙動までを予測することは困難であるから、実際にプログラムがどのような挙動を示すのかを見せる必要がある。このような場合に、既存のシステムでは 3.1節で述べたような問題があり、ただ一部分の変更のために最初からプログラムを実行し直すなどの必要がある。

もし、編集中のプログラムのある部分だけを実行させてみたり、それをまたもとの状態に戻してみたりという、時間軸の操作のようなことができれば、部分的な試行錯誤がしやすくなり、その部分の挙動の把握や修正をしたいといった場合に有効であると考えられる。

そこで、 LivePP-proto で実装されたプログラムの自動実行機構を利用して、次のように、プログラムの指定した一部分に全体とは独立した時間軸を与え、そこだけの評価ができるようにする。

1. プログラムから評価したい部分を切り出す (図 5.3)。

   切り出しを指定した部分は独立した時間軸を持っており、巻き戻しなどによって実行過程の任意の時点の状態を呼び出せる。選択部分の評価によって生じた変化は本体のプログラムには干渉しない。

2. 切り出し部分の時間軸を操作して、その部分が正しく評価されているかどうかを観察する。意図した動作をしない場合は、その部分のルールを修正することができる。

3. 必要であれば、切り出した部分に対して新しい入力データを与える。ゴールに適切な引数が与えられていれば、選択された範囲だけが、通常の実行と同様に自動的に評価される。修正後の実行過程は、それまでの実行過程から分岐した新たな枝として管理される。

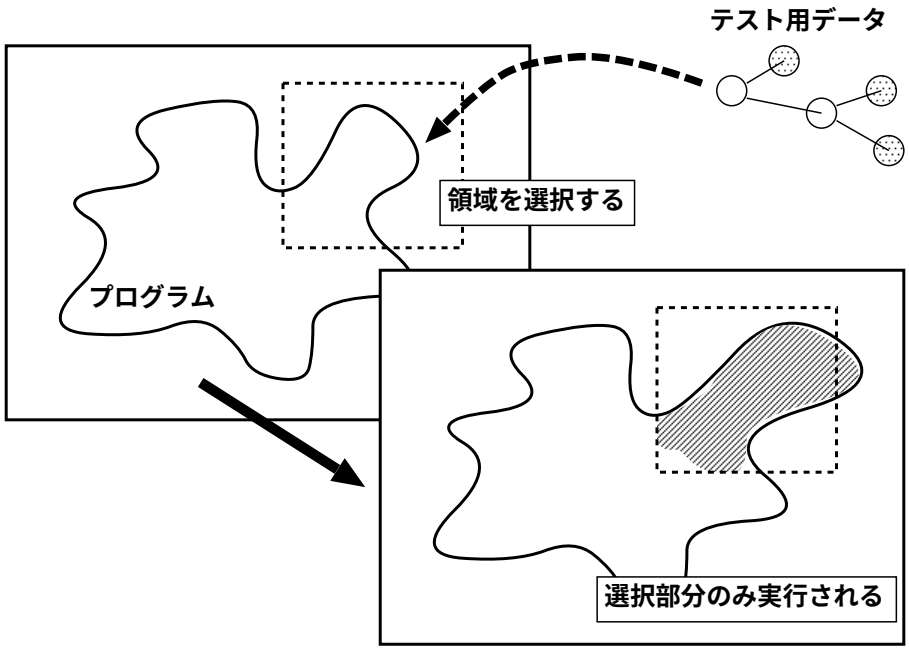


図 5.3: 領域の切り出しと部分実行

5.3　プログラムの自由変形

5.1、5.2で述べた環境は、それぞれ単体で用いてもユーザにとっては意味が薄い。これらを併用することで、プログラムの空間的・時間的側面に対して同時にアクセスすることができ、トレーサ上の実行過程を見比べながらエディタで修正を行なうといった煩雑さを避けることができる。

例として、編集したプログラムを実行させてみたところ、途中で意図しない挙動を示す部分が見つかったとする。その場合、以下のような手順でプログラムの修正を行なう。

1. 時間軸操作を行なって、異常のあったステップまで戻る。
2. 問題の挙動を示した部分に領域選択を行ない、その部分のプログラムを修正。
3. 選択領域を再実行して挙動を確認。うまくいかなければ 2. に戻って再び修正を行なう。
4. 修正が完了したら、領域選択を解除して全体の実行を再開させる。

このように、プログラムの構造と実行過程に関する操作の自由度を上げることで、ユーザがプログラムに対し直感的な操作を行なうことができる。また、これらの操作を同一のビューで行なわせることにより、視点を変更する煩雑さを軽減し、操作の有効性を上げることができる。

第 6 章

まとめ

ビジュアルプログラミングシステムにおける単一モードの操作環境の重要性、望まれるシステムにおけるプログラムの表現について述べた。

プログラムの構造を、抽象化された単なる内部データではなく、自身の視覚表現を生成機能まで含めたオブジェクトの集合として扱うことを提案し、これを利用して、ビジュアルプログラミングシステム LivePP-proto を実装した。 LivePP-proto では、ユーザの操作が直接プログラムの構造に反映され、また、ユーザはモードを意識せずに、単一のビュー上で、プログラムの編集操作と実行を並行して行なうことができる。

LivePP-proto をベースに、プログラムを自由かつ直感的に加工するための操作環境を提案し、それらについて考察を加えた。

謝辞

本研究を進めるにあたり、ご指導、ご助言を頂いた指導教官の田中二郎教授、研究についてのヒントをくださった北海道大学田中譲研究室の岡田義広助手ならびに同研究室の皆様に対し、深く感謝の意を表す。また、支えていただいた田中研究室のメンバー、友人諸氏、そして両親にも深く感謝する。

参考文献

[1] 中野勝次郎 and 田中二郎. ビジュアルプログラミングシステムにおけるモデルの視覚化 アルゴリズム. In インタラクティブプログラミングとソフトウェア *II*, pages pp. 205–214. 近代科学社, 1994.

[2] 遠藤浩通 and 田中二郎. パッドベースシステムによるビジュアルプログラミングシス テムの構成. コンピュータソフトウェア, 15(1):pp.50–53, 1998.

[3] 山本 格也. ビットマップに基づくプログラミング言語 visulan. In インタラクティブプログラミングとソフトウェア *III*, pages 151–160. 近代科学社, 1995.

[4] 田中二郎. 並列論理型言語 GHC のビジュアル化の試み. In インタラクティブプログラミングとソフトウェア *I*, pages pp. 205–214. 近代科学社, 1994.

[5] T. Berlage. A selective undo mechanism for graphical user interfaces based on command objects. *ACM Transactions on Computer Human Interaction*, pages 269–294, September 1994.

[6] Margaret Burnett and Maria Baker. A classification system for visual programming languages. *Journal of Visual Languages and Computing*, pages 287–300, September 1994.

[7] P. Carlson, M. Burnett, and J. Cadiz. A Seamless Integration of Algorithm Animation into a Declarative Visual Programming Language. In *Proceedings Advanced Visual Interfaces (AVI'96)*, 5 1996.

[8] H. Endoh and J. Tanaka. Integrating Data/Program Structure and their Visual Expressions in the Visual Programming System. In *Asia Pacific Computer Human Interaction 1998*, pages pp.453–458, 1998.

[9] E. J. Golin and S. P. Reiss. The specification of visual language syntax. *J. Visual Languages and Computing*, Vol.1(No.2):pp.141–157, 1990.

[10] K. M. Kahn and V. A. Saraswat. Complete visualizations of concurrent programs and their execution. In *Proc. of 1990 IEEE Workshop on Visual Languages*, pages pp.7–15, 1990.

[11] Ken Kahn. ToonTalk(TM) – An Animated Programming Environment for Children. *Journal of Visual Languages and Computing*, June 1996.

[12] K.Ueda. Guarded Horn Clauses. ICOT tech. report TR-103, Institute for New Generation Computer Technology, 1985.

[13] B. A. Myers. Taxonomies of visual programming and program visualization. *J. Visual Languages and Computing*, Vol.1(No.1):97–123, 1990.

[14] Y. Tanaka and T. Imaktaki. Intelligentpad: A hypermedia system allowing functional composition of active media objects through direct manipulatios. In *Proc. of IFIP 11th World Computer Congress*, pages pp.541–546, 1989.

付録 A

LivePP-proto の操作説明

本章では、 LivePP-proto の操作の詳細を説明する。

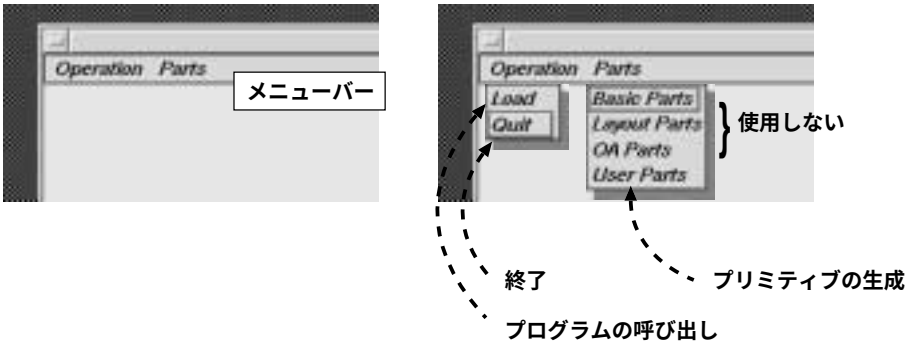
A.1 実行環境

LivePP-proto は、以下の環境において動作する。

OS	Solaris2.4, 2.5 + X11R6
言語	C++
ツールキット等	InterViews3.1 + 日本語化パッチ
	IntelligentPad リファレンスシステム

A.2 LivePP-proto の操作手順

A.2.1 操作部の概観



A.2.2 LivePP-proto の起動

1. IntelligentPad システムを起動する。

```
aria% cd ~/IP/is/here
aria% ./IP
```

2. LivePP-proto のベースを生成する。

   メニューバーの “Parts” から “User Parts” を選択するとパッドのメニューが現われるので、その中の”base” を選択する。

A.2.3 プログラムの編集

プリミティブの生成

1. IntelligentPad のメニューバーから “Parts”→“User Parts” と選択し、現われたメニューから “term”(アトム・ファンクタ) または “pred”(述語) を選択する。

2. プリミティブのシンボル名とアリティ (引数の個数) を尋ねるダイアログが現われるので、“シンボル名 / アリティ” のように入力する。 1. で term プリミティブを選択していた場合は、アリティとして 0 を指定するか、“ / アリティ” を省略するとアトム、 1 以上であればファンクタと認識される。

プリミティブの操作

プリミティブは、マウスで自由にドラッグして移動させることができる。

プリミティブ上でマウスの右ボタンをクリックすると図 A.1のようなメニューが現われ、各種操作をすることができる。

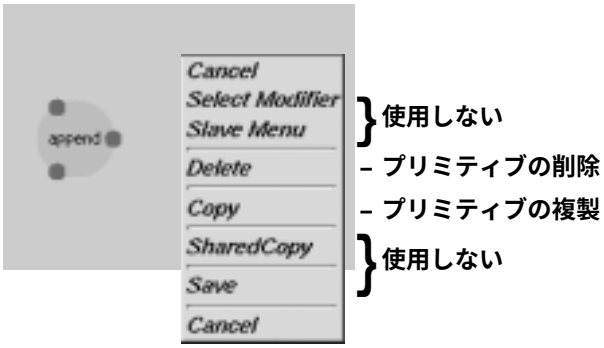


図 A.1: プリミティブ操作メニュー

論理変数の接続

1. 始点となるプリミティブまたはその引数の上でマウスの中ボタンをクリックする。

2. 終点となるプリミティブまたはその引数の上でマウスの中ボタンをクリックする。

3. 論理変数が生成され、プリミティブに接続される。どちらかのプリミティブにほかの論理変数が接続された場合、もとの論理変数は削除される。

A.2.4 プログラムの実行

プログラムの実行は、ビュー上の述語のいずれかにおいて、その述語のガードが満たされるようなデータが結合されたときに自動的に開始される。実行は、ゴールがすべて正常にリダクションしてしまい、ビュー上に存在しなくなるか、あるいはリダクションに失敗するまで続けられる。

A.2.5 プログラムの保存・呼び出し

ベース上でマウスの右ボタンをクリックする。プリミティブの操作メニューと同様のメニューがポップアップするので、 “save” を選択して、識別名をつけることにより、プログラムの状態が保存される。プログラムを呼び出すときは、メニューバーから “Operation”、続けて “Load” を選び、保存のときにつけた識別名を入力することで、ビュー上にプログラムが呼び出される。

A.2.6 LivePP-proto の終了

IntelligentPad のメインメニューから “Operation”→“Quit” と選択し、それに続けて “Confirm” を選択する。

付録 B

LivePP-proto ソースプログラム

以下に、 LivePP-proto のすべてのソースプログラムリスト、ヘッダファイルを添付する。内容は以下の通りである。

- プリミティブ共通部分 (`elem.c`, `elem.h`, `data.c`, `data.h`)
- アトム・ファンクタプリミティブ (`term.c`, `term.h`)
- 述語プリミティブ (`pred.c`, `pred.h`)
- ベース (`base.c`, `base.h`)