ビジュアルプログラミングシステムにおける入力法の効率化

An Efficient Input Method for Visual Programming Systems

田中 二郎 Jiro TANAKA
後藤和貴[†1] Kazutaka GOTO
馬場 昭宏 Akihiro BABA

筑波大学
University of Tsukuba

概 要

ビジュアルプログラミングシステムの構築に当たっては、如何にして図形入力を効率的に行なえるかが実用化の鍵となる。我々は並列論理型言語 GHC を基に、ビジュアルプログラミング言語の視覚化モデルを設計し、図形入力の定義節エディタを Tcl/Tk を用いて実装した。

本論文では、視覚化モデルの図的表現、新しい定義節エディタにおける図形入力、実装と内部コード、新しい図形入力システムの機能、自動レイアウト機能などについて述べる。

1 はじめに

視覚的言語 (Visual Language) とは、図的情報を操作し、視覚的インタラクションをサポートするプログラミング言語を指す。また、ビジュアルプログラミングシステムとは、ユーザがプログラムの構造に沿った空間的な制約に基づく文法に従い、視覚的言語をインタラクティブに操作できる環境を指す。我々は、より使いやすいビジュアルプログラミングシステムを目指すために、まずその図的入力法について考察し、ビジュアルプログラミングシステムのプロトタイプを試作した。本論文ではそれらについて報告する。

2 視覚化モデルの図的表現

ターゲットとなるビジュアルプログラミング言語の仕様を、宣言型言語である並列論理型言語 GHC [Ueda85]に基づくものとし、ビジュアルプログラミングシステムの視覚化モデルの図的表現を以下のように定めた。

引数 定義節の引数は引数の領域の円周上に置き、入力引数は○、出力引数は●で表す。また、ゴールや項の入力引数や出力引数はゴールや項から出入りする矢印として表現され、陽には表現されない。

ゴール、項 ゴールは円形で表す。項はゴールより小さな円形で表す。

論理変数 共有する引数、ゴール、項の間を矢印で結ぶことにより表現する。

ガード及びボディ 引数ガード及びボディは円形にし、ガードはボディの外側となるように配置する。

なお、以後、引数、ゴール、項を総称して**アイコン**と呼ぶ。

3 新しい定義節エディタにおける図形入力

図的表現に基づき、ビジュアルプログラミングにおけるグラフィック・ビューの見直しを計り、図形入力のためのシステムの新しい定義節エディタを以下のようにデザインした (図 1)。これは、従来か

†1 現在 日本オラクル株式会社

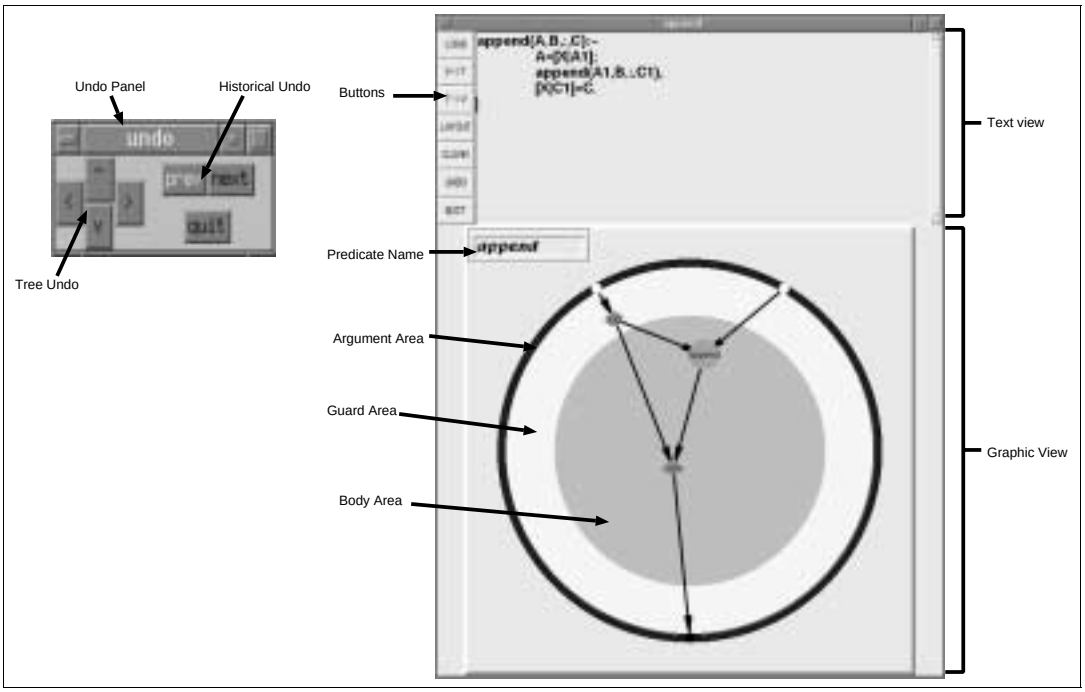


図 1　図形入力のための新しい定義節エディタ

ら研究を進めていた PP(Pictorial Programming) [Tanaka94a] [Tanaka94b]をベースに再設計を行なったものである。

新しい定義節エディタでの入力方法は以下のようにまとめることができる。

select 状態　各アイコンは select された状態と select されていない状態の二つの状態を持ち、select されているいくつかのゴールに対して操作を行うことができる。

アイコンを select する　どのアイコンも select されていない時は、各アイコンをマウスの左ボタンでクリックすることで、そのアイコンを select された状態にする。 いくつかのアイコンが select された状態の時に、ある select されていないアイコンを shift キーを押しながらマウスの 左ボタンでクリックすると、新たにそのアイコンが select された 状態になる。

select をはずす　マウスの右ボタンをクリックすると select された状態のゴールを select されていない状態にする。

アイコンの移動　あるアイコン上でマウスの中ボタンでドラッグすることで、アイコンの移動を行うことができる。移動後にそのアイコンは select された状態になる。

アイコンの入力　アイコンの入力は、マウスの左ボタンをクリックすることで行う。このとき select されているゴールがある場合、それらのゴールからの出力引数を、その新しくできたゴールに対しての入力引数とする。

アイコンの変更、消去　アイコンの変更または消去は、変更または消去したいアイコンの上でマウスの左ボタンをダブルクリックすることによりメニューを呼び出すことで行う。

4　実装と内部コード

以上の考察に基づき、 Tcl/Tk [Ousterhout94]を用いて新しい図形入力システムの実装を行なった。

本システムでは、図形表現からテキスト表現、テキスト表現から図形表現の自動変換を行なう必要がある。この双方の表現の変換を効率的に行うために中間言語として内部コードを用意した。

内部コードは、 Tcl スクリプトでのリスト表現として次のような形をしている。

{ **定義節のゴールのコード** { **ガード部のゴールのコードのリスト** } { **ボディ部のゴールのコードのリスト** }}

また、ゴールのコードは、

{ **ゴールの名前／種類／番号** { **入力引数のコードのリスト** } { **出力引数のコードのリスト** }}

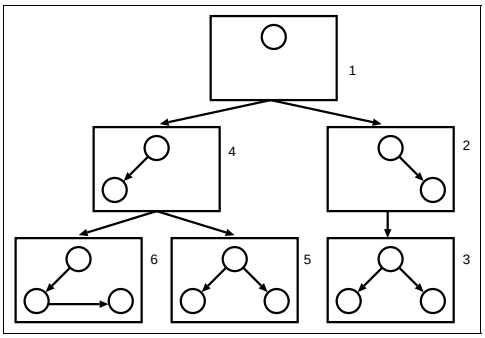


図 2　状態の木

そして引数のコードは、

引数の名前／種類／番号

と表す。

例えば、以下のようなプログラムの場合

```
append(A,B,:,C):-
    A=[X|A1];
    append(A1,B,:,C1),
    [X|C1]=C.
```

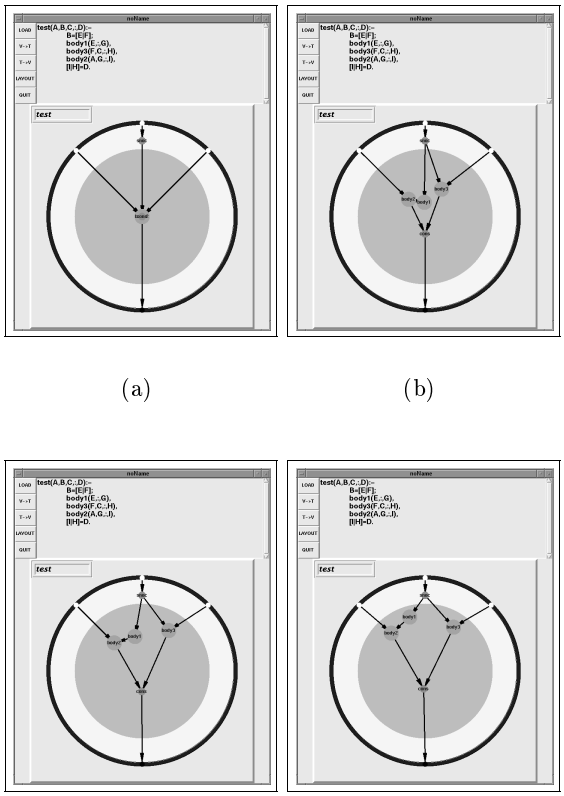
対応する Tcl スクリプトでのリスト表現は以下のようになる。

```
{{append/0/1 {A/0/1 B/0/2} C/0/3}
     {{snoc/2/2 A/0/1 {X/0/4 A1/0/5}}}
     {{append/4/3 {A1/0/5 B/0/2} C1/0/6}
          {cons/1/4 {X/0/4 C1/0/6} C/0/3}}}
```

5　新しい図形入力システムの機能

新しい図形入力システムの機能として以下の機能を用意し、実装した。

- 図形入力
  Tk のキャンバスウィジェットは、その内部に円、直線、多角形、文字などのアイテムと呼ばれる図形群を表示することができる。各視覚化モデルは キャンバスウィジェットのアイテムを使い表現した。
- テキスト入力
  テキストプログラムは、テキスト・ビューにフォーカスをあわせてキーボードから入力するようにした。また、 LOAD ボタンを押してファイルメニューを出してファイルを読み込むことも可能である。
  テキスト・ビューにおいては、 Emacs ライクなキーバインドで入力が可能であるようにした。また、マウスによる他のアプリケーションとの間でカット＆ペーストも可能であるようにした。
- 図形表現から内部コード
  図形表現から内部コードに変換する時は、その空間的な配置を考慮して、ガード部及びボディ部のゴールはそれぞれのノードの上下関係をそのまま内部コードのリストの順番に反映させた。
  また、各ゴールの引数についても、左右の関係を引数の順番とした。
- 内部コードから図形表現
  内部コードから図形的表現に変換する際、定義節の引数は、入力引数の時は引数領域の円周の上半分に、出力引数は引数領域の円周の下半分に、それぞれ引数の数に応じて等分した場所に配置する。この時、変数名は入力引数の左側から順に名前をつける。
  ガード部のゴールは、そのゴールの入力引数となる論理変数とグラフィック・ビューの中心と結んだ線上に配置する。また、ボディ部のゴールは、この時点ではすべてグラフィック・ビューの中心に配置する。
- Undo
  新しい図形入力があるたびに現在の図形表現をを内部コードに変換し、それぞれの状態に番号をつけて保存することで Undo を実現した。
  Undo をするときにはまず UNDO ボタンを押して Undo Panel を出す。
  本システムでは 2 通りの方法の Undo を提供している。
  ひとつ目の方法は Historical Undo である。新しい状態ができたとき、または、 Undo をしたとき、のいずれかの場合に現在の状態につけられた番号は履歴のリストに加えられる。 Historical Undo は、その履歴のなかをたどることで目的の状態まで戻る方法である。
  もうひとつは Tree Undo である。新しい状態を直前の状態の子供だと考えると、全ての状態の集合は一つの木構造を形成する (図 2)。 Tree Undo はこの状態の木の各ノードをたどることにより、目的の状態まで戻る方法である。上向き、下向き、右向き、左向きのボタンを押すと、それぞれ親、もっとも若い子、すぐ隣の



(a) (b)

(c) (d)

図 3　レイアウト過程

兄、すぐ隣の弟の各ノード (状態) に移る。

6　自動レイアウト

本定義節エディタでは、理想的なレイアウトを次のように考えた。

グラフ上の一つのエッジは、そのエッジに繋がる二つのノードの種類により異なる強さを持つバネであるとする。このとき、すべてのバネによる引力がつりあうように各ノードが配置されるのが理想的レイアウトということになる。

アルゴリズムは次のようになる。

1. 自動レイアウトは select されたノードを対象に行う。ただし一つも select ノードがない場合、ボディ領域内の全てのノードを対象に行う。
2. 全てのノードについて、移動の計算をしてから再描画する。
3. 各ノードは、対象ノードとリンクする各ノードについてそのノードへのベクトルを計算し、エッジの種類によって異なる定数をかけたベクトルの合成ベクトルの方向に移動する。
4. レイアウトすべき全てのノードの移動量をその配置のエネルギーとし、エネルギーの変化量がしきい値よりも少なくなるまで続ける。

以上のようなアルゴリズムにより、最適配置までの途中結果を用いることで、アニメーションによるレイアウトを実現した。

実際に、定義節エディタを用いて図形入力をおこなう過程を図 3にしめす。

7　おわりに

本論文では、ビジュアルプログラミングシステムにおける入力法の効率化について述べた。今回は一つの定義節のみの入力であるが、より効率的なプログラム入力が可能になった。

本研究で実装した定義節エディタは、複数個の定義節の編集、三つ以上のゴールの引数として共有される論理変数の表現方法などに関して研究の余地を残している。今後はこれらの課題も含め、より使いやすいビジュアルプログラミングシステムを目指すために、さまざまな手法を採用し研究を深めていく予定である。

参考文献

[Ousterhout94] J.K. Ousterhout: Tcl and the Tk toolkit, Addison-Wesley, 1994.

[Tanaka94a] J. Tanaka: Visual Programming System for Parallel Logic Languages, The NSF/ICOT Workshop on Parallel Logic Programming and its Program Environments, the University of Oregon, pp.175-186, 1994.

[Tanaka94b] 田中二郎: 並列論理型言語 GHC のビジュアル化の試み, インタラクティブシステムとソフトウェア I, 日本ソフトウェア科学会 WISS'93, 竹内彰一 編, 近代科学社, pp265-272, 1994.

[Ueda85] K. Ueda: Guarded Horn Clauses, ICOT Technical Report TR-103, ICOT, 1985.