テクニカルノート

単一ビュー / モードに基づくビジュアルプログラミング環境の構築

遠藤浩通[†] 田中二郎[††]

現在，発表されているビジュアルプログラミングシステムの多くが，プログラムの編集と実行をそれぞれ別環境で行なわせるような実装となっている．そのようなシステム構成におけるプログラミングの形態は，従来のテキスト言語環境で行なわれてきたものであり，必ずしもビジュアルプログラミングシステムの特徴を行かすものではない．

我々は，プログラムの編集・実行が単一の環境・単一の操作モードによって行なえることを重視した．そのために，プログラムの論理的構造と視覚表現を同時に操作することのできる”プリミティブ”によるプログラムの表現を提案する．それを用いて，我々はプロトタイプシステム LivePP-proto を実装した．LivePP-proto では，ビューと呼ばれる単一の領域上でプログラムの操作を行なうことができる．

プリミティブを用いたビジュアルプログラミングシステムでは，プログラムの構造および実行過程を柔軟に操作することが可能である．その特徴を生かし，プログラムの自由な加工の例として，プログラムの部分的実行と，自動ルール生成について考察する．

Visual Programming System based on Single View/Manipulation Mode

Hiromichi ENDOH [†] and Jiro TANAKA [††]

Most of existing Visual Programming Systems are based on the traditioanl “Edit-and-Run” programming paradigm inherited from the text-based programming environments. Are such systems based on “separated architecture” really intuitive?

We had designed and implemented a new program visualization system in which the logical structure of the program and its visual expression are built in a object called “primitive.” In our system, the static and dynamic aspects of the program can be treated in the same framework, so program editing and execution can be performed on single view. We can apply this framework to provide more intuitive program manipulation interface such as partial execution and auto-rule-generation from evaluation results assumed by user.

1. はじめに

従来のテキストベースのプログラミング言語環境では，プログラムの構造という静的な側面と，実行という動的な側面を同一の環境で扱うことが困難であったため，エディタによる編集 → (コンパイル →) 実行という形態が主流であった．

既存の多くのビジュアルプログラミングシステムも，その多くは専用のエディタと実行過程の視覚化を行なうためのトレーサから構成されており，それらの間では，プログラムは単なる内部表現として扱われている．このように，編集・実行に対してシステムのモードをはっきり区別しているという点で，やはり従来のプログラミングスタイルを受け継いだものであると言える．

しかし，このようにプログラミングにおいて異なる環境でプログラムを扱わなければならないような構成はユーザにとって煩雑なものとなる可能性があり，充分に直感的な操作環境を提供しているとはいい難い．

我々の研究しているビジュアルプログラミングシステム PP[1)2)] では，直接操作[3)4)]・直感的操作の提供に重点を置いているが，現在は特に，単一の環境上におけるプログラムの操作の枠組みに注目している．[5)6)]

† 筑波大学大学院修士課程理工学研究科
Master's Program in Science and Engineering, University of Tsukuba
†† 筑波大学電子・情報工学系
Institute of Information Sciences and Electronics, University of Tsukuba

2. データ構造と視覚表現の統合

2.1 単一モード環境

1節で述べた通り，これまでに提案されてきたビジュアルプログラミングシステムは，前述したエディタとトレーサに相当するサブシステムがシステム中ではっきり区別されていることが多い．これらはちょうどemacsとccのように個別のプログラムとして実装されているか，あるいは単一のビューを提供するようなシステムであっても，編集・実行でそれぞれモードを切り替える必要がある．

我々は，このように個別の環境上でプログラムの編集・実行を行なうのではなく，単一のビューの上でこれらすべての操作を行なえるようなビジュアルプログラミング環境が重要であると考えている．ただし，「いわゆる統合環境」のようなものではなく，編集・実行のための各操作がすべてモードの切り替えなしで行なわれ，プログラムを実行させた結果についても同じビューに提示されるような環境である．イメージとしては，ビュー上のプログラムを直接操作でき，実行させるとそれがそのまま変形していって結果になるというものである．

2.2 プリミティブの導入

既存のシステムでは，それぞれ編集・実行のためのサブシステムが分離して存在していたこともあり，プログラムの論理的構造を内部表現として共有し，それを各サブシステムが視覚表現に変換して表示するという**図1**のような構成が一般的である．

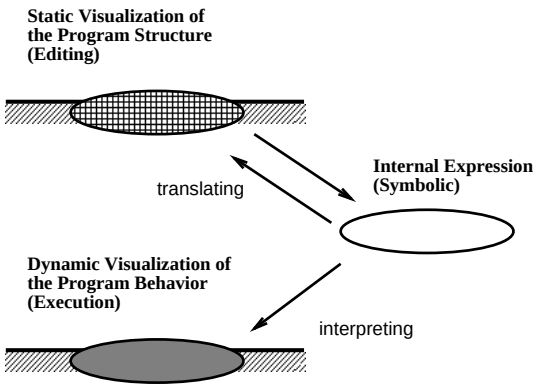

図1 既存のシステム構成

このような構成の場合，サブシステム間でプログラムの状態を直接共有することが難しく，例えばエディタでプログラムを変更したが，それがトレーサ上でのプログラムの実行状態に即座に反映されないといった問題が生じる．

我々は，プログラムの論理構造と視覚表現を同時に扱うためには，プログラムの内部表現があり，それを視覚表現に変換して表示するという層割りの構造ではなく，**図2**のように，プログラムの構成要素を単位として，その1つ1つが自らの視覚表現を生成するようにすればよいと考えた．この構成要素をプリミティブと呼ぶ．プリミティブは，自分の名前や引数の数のようなシンボリックな情報を持つ部分と，それをもとにビュー上に自分を描画する部分からなっており，これらの集合としてプログラムが表現される(**図3**)．プリミティブの1つ1つが，自分自身をビュー上に描画することにより，全体としてそのときのプログラムの状態がビュー上に投影されているように見える．もし，プログラムの構造が変化すればビューの状態も変化し，ユーザがビュー上でプリミティブを操作すれば，同時にプログラムの構造が変更する．

プリミティブの集合はビューと1対1に対応しており，編集操作と実行操作はどちらもそのプリミティブ集合を対象として行なわれる．いずれも，結局はプリミティブ集合の書き換えであり，プログラムの編集はユーザが，実行はシステムが行なうという，書き換えを行なう主体の違いだけとなる，このため，編集操作・実行のそれぞれが異なる操作モードを持つ必要がなくなる．

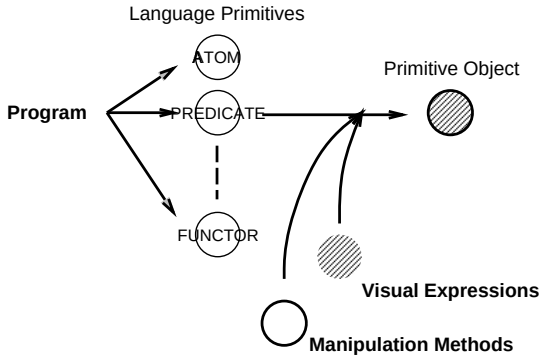

図2 言語要素と視覚表現の統合

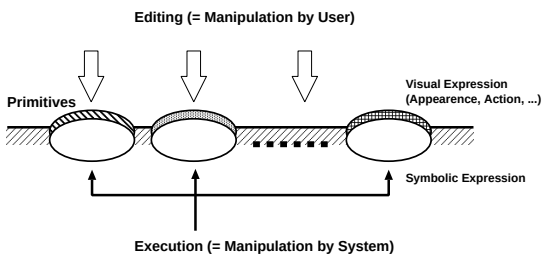

図3 プリミティブを用いたプログラムの表現

3. LivePP-proto

我々は，プリミティブの概念を用いて，プログラムの論理的構造と視覚表現を統合して扱うことにより，単一ビュー上でプログラムの操作を行なうシステム LivePP-proto を設計，実装した．本節では主にその設計とプログラミング例について述べる．**図 4** に LivePP-proto の概観を示す．

LivePP-proto では，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として，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

3.1 言語のモデル化

我々の提案する，単一のビューによるプログラミング環境では，次のような理由から，モデルとする言語に並列論理型言語の 1 つである GHC[7] を採用した．

- 言語を構成する基本要素の数が少なく，規則がシンプルである
  GHC を含む宣言型の言語は，データ構造の要素となる意味単位 (アトム) や演算子，そしてそれらのデータ構造間の関係規則によって構成される．また，実行規則も，ゴールをサブゴールに置換するというもののみで，非常に簡潔なものである．このため，手続き呼び出し・変数・制御構造など多くの要素・規則を含む手続き型言語などに比べて視覚化が容易である．
- データ及びプログラムが単一の表現によっている
  GHC のプログラムは，データ構造から新しいデータ構造への変換規則を定義するものであり，データ構造と同じ構成要素を用いて記述される．従って，単一ビュー上でプログラムを操作する場合，操作対象による表現の違いを意識する必要がなく，ユーザの負担も軽くて済む．

3.2 プログラミング例

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．

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 5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，プリミティブのシンボル (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．

プリミティブや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 6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．

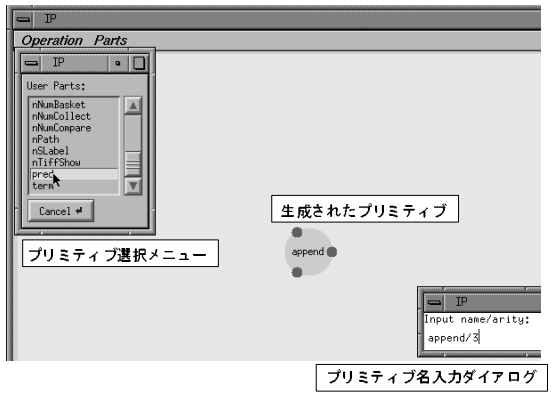

図 5 プリミティブの生成

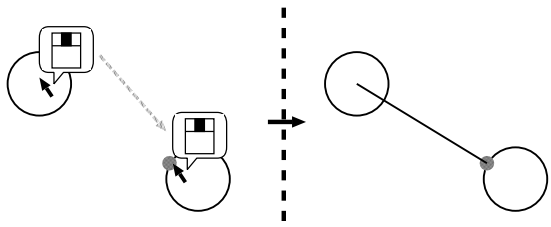

図 6 論理変数の接続

プログラムの実行

LivePP-proto は，プログラム中のゴール状態を監視しており，リダクションが可能であると判断した場合はそのゴールの評価を自動的に開始させる．従って，プログラムの主要部分を 3.2節のようにして作成しておき，そこに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．2 つのリストを連結する述語 append を用いて実行例を示す．

図 7 に，完成前のプログラムを示す．中央にあるのがゴール append/3，左は連結されるべき 2 つのリスト，そして右は結果を保持するダミーゴールである．これらを論理変数で接続すると，すべてが接続された時点で append/3 がリダクション可能と判断され，プログラム実行が開始される．まず，図 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

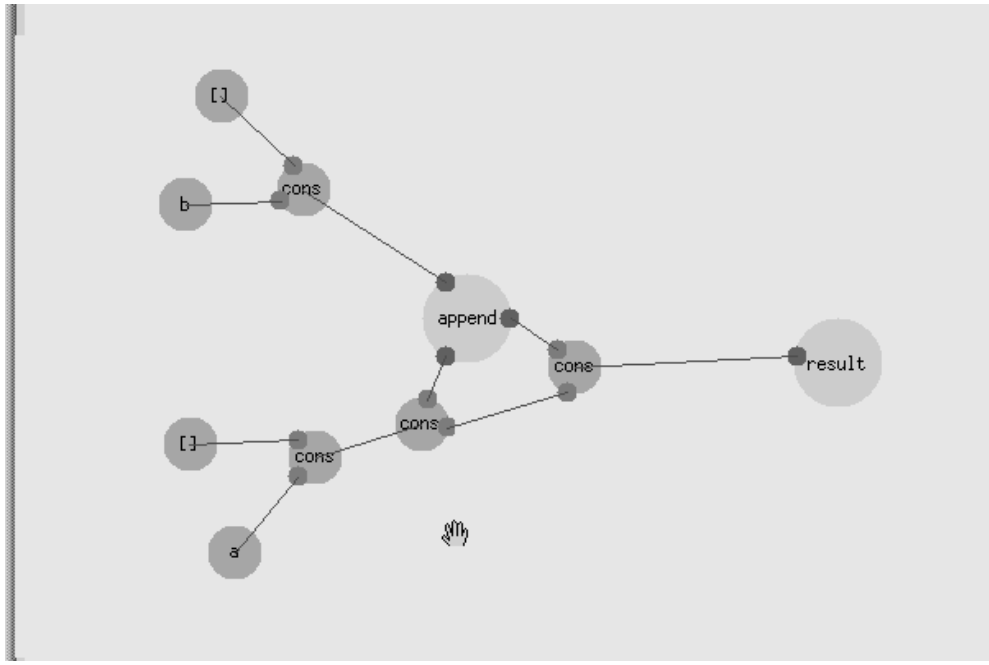

図 4 LivePP-proto

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9では，図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．

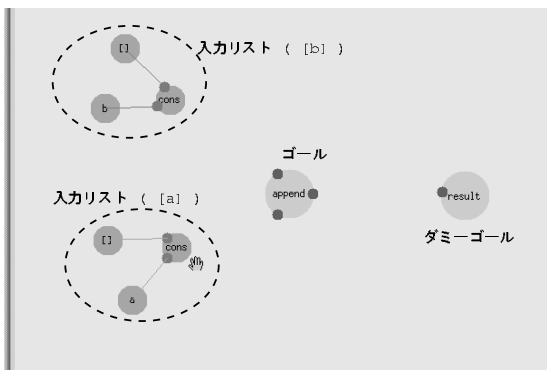

図 7 プログラム実行例: 初期状態

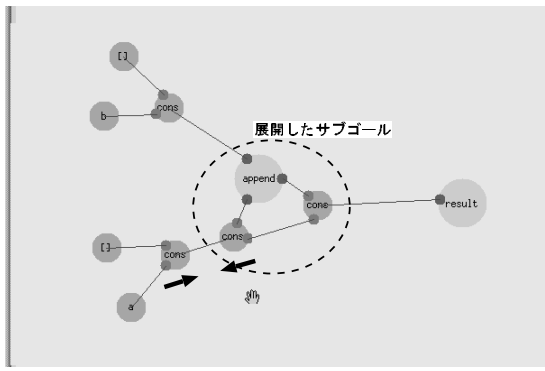

図 8 プログラム実行例: 実行開始

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．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．

4. 単一環境を応用したプログラミング

LivePP-proto で実装された，プログラムの論理構造と視覚表現を統合したプログラム表現は，直感的かつ自由にプログラムを加工できるような操作環境を構築するための基礎として有用であると考えている．本節では，

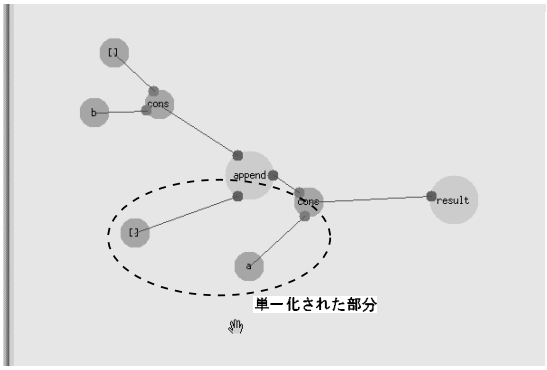

図 9 プログラム実行例: 単一化直後

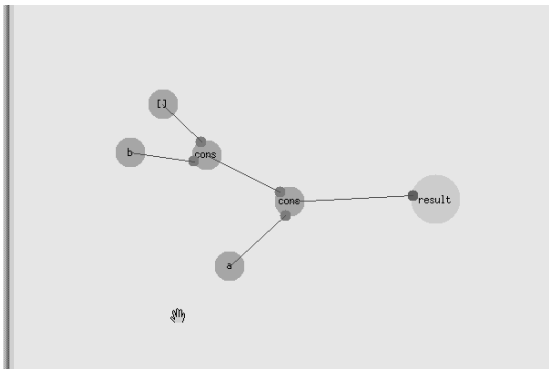

図 10 プログラム実行例: 最終状態

LivePP-proto の環境の発展として，次に挙げるような環境の有用性について考察する．

- プログラムの自動評価
- ルール定義・追加の支援

2.2節で挙げたような，プログラムの編集 / 実行を明確に分離している構成のビジュアルプログラミングシステムでは，これらの環境の実現は困難と思われるが，前節で示した枠組を用いることでこれらを自然に達成できると思われる．

4.1 プログラムの部分的評価

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 2.2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，そこだけの評価を行なわせることができる．

(1) プログラムから評価したい部分を切り出す (**図 11** 上).
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ことができる．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ら枝として管理される．

4.2 ルール定義・追加の支援

ユーザは，ビュー上にアトムやゴールなどの部品を配置してプログラムを編集する．ゴールの引数に対してデータが与えられると，LivePP-proto と同様に，それがゴールの定義節におけるリダクション条件を満たすかどうかがチェックされる．

もしもそのゴールに定義節が存在しないか，または与えられたデータがどの定義節のリダクション条件にも合致しない場合，システムは**図 12** のようにそのゴールを待機状態にして，ユーザにルールの定義を促す．ユーザは，新たに追加するルールの適用後の状態を仮定としてビュー上に作成する (**図 13**)．システムは，そこからユーザが仮定したルールを自動的に作成する．

プログラムの実行中に，それまでに定義されているルールの適用に失敗した場合にも，そこでプログラム全

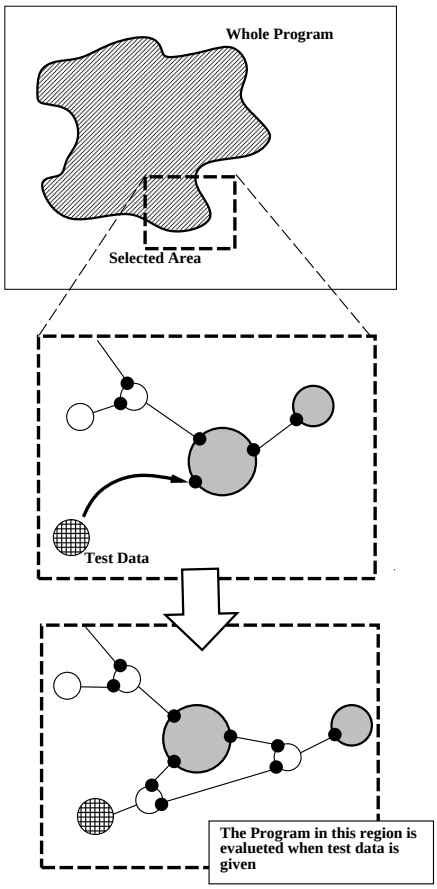

図 11 領域の切り出しとテストデータの適用

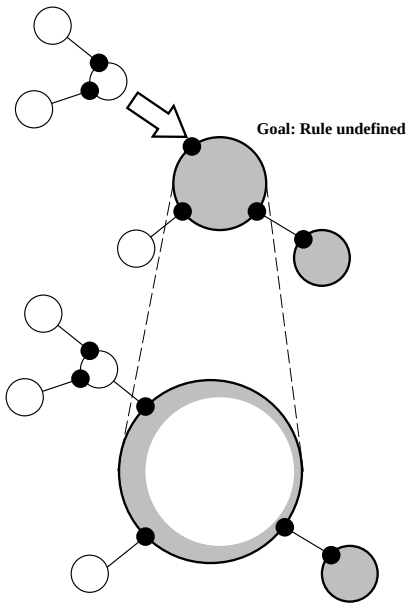

図 12 ルール定義の開始 (1)

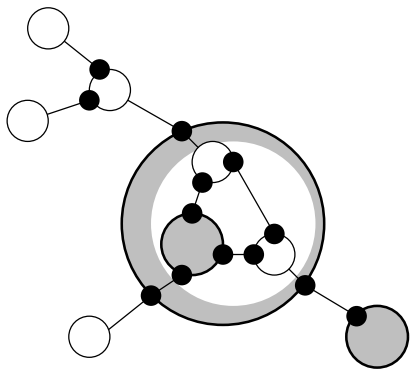

図 13 ルール定義の開始 (2)

体の実行を放棄するのではなく，ルール適用に失敗した部分で，ユーザに新たなルールの定義を求めるようにする．

例えば，プログラムを実行中，**図 14** のように，ゴールに対してどの定義節のリダクション条件も満たさないようなデータが引数として与えられると，このゴールに対するリダクションが失敗する．通常，GHC ではこのような場合にはプログラム全体の実行が放棄されるが，ここですぐに実行を放棄せず，リダクションに失敗したゴールを待機状態にして，ユーザに新しいルールの追加を促す．もし，新しいルールの追加によってその部分の評価が可能になった場合，追加されたルール状態でプログラムの実行が続行される．

5. 関 連 研 究

山本の visulan[8] は，1 枚のビットマップをプログラミング空間とみなし，その中でパターンマッチと置換を繰り返すことによってプログラムを実行するシステムである．しかし，ビットマップの各ドットは意味を持たないために完全にフラットなプログラムしか記述できない，実行規則の適用が単純なパターンマッチのみによるなどの相違点がある．

Burnett らによる Forms/3[9)10)] は，スプレッドシートの動作をもとにしたプログラミングシステムである．オブジェクトの動作は，

セル内の記述によって制御され，セル内容が更新されると，自動的に再評価が行なわれる．制約の向きはセル → オブジェクトの一方通行であって，オブジェクトに対する操作をセルに反映させることはできない．

ToonTalk[11] は，データや演算などを漫画のキャラクタに見立て，それらの動きによってプログラムを表現する．プログラミングは単一のビュー上で行なわれる．プログラムの編集は例示的であり，ユーザの操作をロボット (プログラム) に覚えさせることで行なわれる．

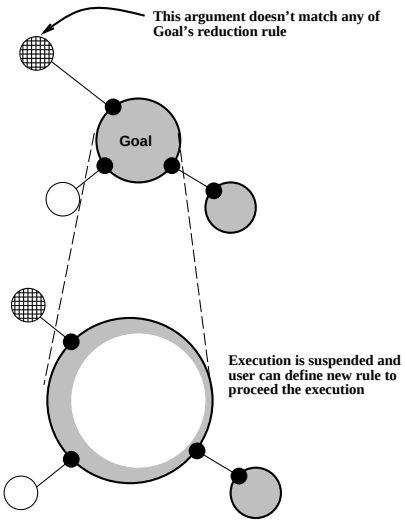

図 14 実行失敗時のルール追加定義

6. ま と め

我々は，プログラムの論理的な構造と，その視覚的表現を同一のデータ構造内に組み込むことにより，プログラムの静的な面と動的な面を同等に扱うことができるようになった．これにより，単一のビュー上においてプログラムの編集及び実行という性質の異なる操作を並行して自由に行なえるという，従来のビジュアルプログラミングシステムにみられなかった環境を実現できるようになった．この枠組を利用するとにより，4.1，4.2節で示したように，プログラムの編集と同時にその部分の評価結果を得てそれを編集操作にフィードバックさせることができたり，プログラムの実行中に発見された誤りをその場で訂正し，そのまま評価を続行させたりすることが可能になる．

参 考 文 献

1) 田中二郎: 並列論理型言語 GHC のビジュアル化の試み, インタラクティブプログラミングとソフトウェア I, 近代科学社, pp. 205–214 (1994).
2) 中野勝次郎, 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, インタラクティブプログラミングとソフトウェア II, 近代科学社, pp. 205–214 (1994).
3) B.Schneiderman: Direct Manipulation: A Step beyond Programming Languages, *IEEE Computer*, Vol. 19, No. 8, pp. 57–69 (1983).
4) Schneiderman, B.: *Designing the User Interface*, Addison-Wesley, 2nd edition, chapter 5 (1992).
5) 遠藤浩通, 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, Vol. 15, No. 1, pp. 50–53 (1998).
6) Endoh, H. and Tanaka, J.: Integrating Data/Pro-gram Structure and their Visual Expressions in the Visual Programming System, *Asia Pacific Computer Human Interaction 1998*, pp. 453–458 (1998).
7) K.Ueda: Guarded Horn Clauses, ICOT tech. report TR-103, Institute for New Generation Computer Technology (1985).
8) 山本格也: ビットマップに基づくプログラミング言語 Visulan, インタラクティブプログラミングとソフトウェア III, 近代科学社, pp. 151–160 (1995).
9) John Atwood, M. B. et al.: Steering Programs via Time Travel, *1996 IEEE Symposium on Visual Languages* (1996).
10) Carlson, P., Burnett, M. and Cadiz, J.: A Seamless Integration of Algorithm Animation into a Declarative Visual Programming Language, *Proceedings Advanced Visual Interfaces (AVI'96)* (1996).
11) Kahn, K.: ToonTalk(TM) – An Animated Programming Environment for Children, *Journal of Visual Languages and Computing*, Academic Press (1996).