筑 波 大 学 大 学 院 修 士 課 程

理 工 学 研 究 科 修 士 論 文

ビジュアルシステムにおける Action の視覚化

電子･情報工学専攻

西名　毅

平成１２年３月

筑 波 大 学 大 学 院 修 士 課 程

理 工 学 研 究 科 修 士 論 文

ビジュアルシステムにおける Action の視覚化

電子･情報工学専攻

西名　毅
主任指導教官　電子情報学系　　田中二郎

平成１２年３月

目次

要旨

ビジュアルシステムを扱う際に図形の削除や生成、属性の値の書き換えなど動的なものを扱うことが多い。ビジュアルシステム生成系 Eviss は、CMG に Action の概念を加え CMG を拡張したものを使用している。CMG で図形言語を解析し、Action を用いることによってその解析結果を使用して動的なことができる。Eviss は、拡張 CMG を用いて図形文法を自分で定義に基づいて図形言語を解析するための Spatial Parser を生成することができる。Eviss は、その Spatial Parser を用いて図形言語を解析することによってさまざまなビジュアル言語に対応できるシステムである。現在、Eviss では、図形言語を定義する際に拡張 CMG への入力をテキストを用いて行なっている。しかし、図形言語をテキストを用いて表現しているために理解をするのに困難な時があり入力にも時間がかかった。

本研究では、拡張 CMG を使用した図形言語の定義の入力をユーザにとって直感的かつ一目で理解しやすいようにするために、テキストで入力するのではなく定義に使用した図形言語をそのまま用いて Action を視覚化することを試みた。また、1 つの CMG で連続した Action を扱う場合の表示手法を工夫した。その結果、テキストに比べて入力を簡略化することができ、テキスト入力時には難しかった Action の流れを容易に把握することが可能になった。

1. はじめに（研究背景）

最近、テキストを用いたプログラミングによる表現の限界が指摘されている。そこで新たなプログラミングの手法として文字より直感的な図形を用いて視覚的な表現を使ったビジュアルプログラミングシステム[8][9][10][11][12][13]が研究されてきている。

これらビジュアルプログラミングシステムの研究の対象となるものは図形(図形言語)を用いた言語である。そこで、実装をする際には図形言語の解析を行なう部分（Spatial Parser）が必要になってくる。しかし、既存のビジュアルプログラミングシステムは特定のビジュアルプログラミング言語の仕様に固定されていた。１つ１つのビジュアルプログラミングシステムに Spatial Parser を実装するのは困難である。そこで、Eviss[1][5][6][7]が開発された。Eviss は、CMG[3][4]を拡張した拡張 CMG[1][5][6][7]を使用して図形文法をユーザが定義でき、その定義に基づいて Spatial Parser を生成できる。そして、その Spatial Parser を用いて図形を解析することができる。また、拡張された Action は、図形の生成、削除、図形の属性の値の書き換えなどができ、解析結果を利用することができる。

Eviss では、拡張 CMG を用いて図形文法を自分で定義して記述する。この拡張 CMG では図形の属性や制約を定義できる。属性が図形の座標や線の長さなどを定義でき、制約はある属性と属性もしくは定数の間に条件を課すものである。この時、図形間の関係は 2 次元的な情報が扱われている。しかし、Eivss ではこの関係を 1 次元的なテキストで入力するために理解しづらい。そこで、視覚化して図形を用いて入力することによってより直感的で、理解しやすくする。

本論文では、Eviss の特徴の１つである Action に注目し、この拡張 CMG の入力、編集、入力したものの表示方法についてより理解しやすいインタフェースを提案する。

本論文の構成は次の通りである。まず２章でビジュアル言語の定義をおこない、Eviss について説明をする。３章では Eviss の特徴である Action について述べる。４章で Action の視覚化について述べ、次の５章でその視覚化した Action の表示方法について述べる。最後の６章では Eviss とその他のシステムとの比較をする。

2. 準備

2-1 ビジュアル言語

Eviss[1][5][6][7]はビジュアル言語を扱うものである。ここで、ビジュアル言語に関する用語の定義をする。

単語を構成するために通常のテキスト言語では文字を 1 次元に配置する。これに対して円や直線などの図形を 2 次元、もしくはそれ以上に配置するものを「**ビジュアル言語**」と呼ぶ。そして、ビジュアル言語におけるこれらの基本的な図形を「**図形文字**」と呼ぶ。各図形文字は、種類、色、大きさ、位置などの属性を持つ。

テキスト言語で単語に相当するものがビジュアル言語にもあると考えられる。テキスト言語では、文字が集まって単語ができている。ビジュアル言語でも図形文字がいくつか組み合わさったものを「**図形単語**」と定義する。図形単語も属性を持つ。１つの図形単語を作るためのいくつかの図形を、構成要素と呼ぶ。そして、図形単語を組み合わせて構成されるものを「**図形文**」として定義する。ここで、図形文に現れるのは正確には図形単語のインスタンスであることに注意する必要がある。本論文では、図形文に現れる図形単語のインスタンスをトークンと呼ぶことにする。

Eviss はビジュアルシステム生成系だがビジュアルシステムとはビジュアル言語を処理するシステムとして定義する。

2-2 CMG と拡張 CMG

CMG（Constraints Multiset Grammars）[3][4]は図形文法を記述するためのものである。

トークンには、終端トークン、非終端トークン、開始トークンとがある。図形文はトークンのマルチセットである。終端トークンのみを含む図形文を終端文と呼ぶ。CMG の図形文の例を以下に示す。

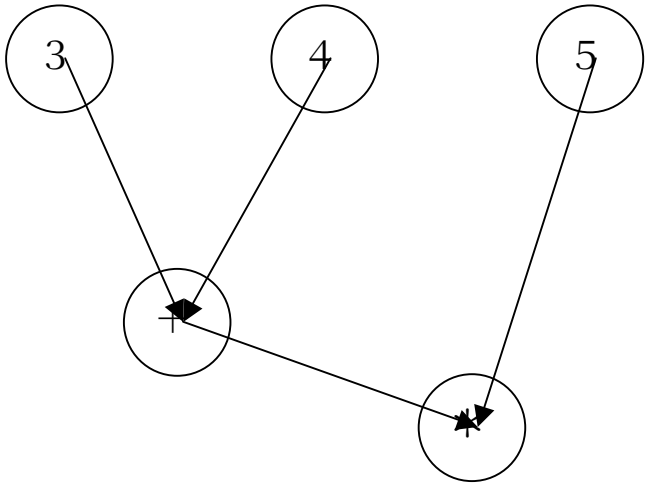

図 2.1　計算の木（（３＋４）＊５）

$$\{circle(mid = (20,20),\ radius = 10),\ circle(mid = (60,20),\ radius = 10),$$
$$circle(mid = (120,20),\ radius = 10),\ circle(mid = (40,60),\ radius = 10),$$
$$circle(mid = (100,20),\ radius = 10),\ text(mid = (20,20),\ text = '3'),$$
$$text(mid = (60,20),\ text = '4'),\ text(mid = (120,20),\ text = '5'),$$
$$text(mid = (40,60),\ text = '+'),\ text(mid = (100,80),\ text = '*'),$$
$$line(start = (20,20),\ end = (40,60)),\ line(start = (60,20),\ end = (40,60)),$$
$$line(start = (40,60),\ end = (100,80)),\ line(start = (120,20),\ end = (100,80))\}$$

上の図形文はサークルが５つ、そしてそのサークルの中心にテキストが書かれていて４つのラインで結ばれていることを表している。図形文の形式は、最初に構成要素の種類が書かれていて、カッコの中にその構成要素の属性が書かれている。サークルの属性として中心の座標と半径が、テキストの属性として中心の座標とその値、ラインの属性としてラインの始点の座標と終点の座標が書かれている。radius は円の半径を表している。mid はその type の中心座標を表している。start と end は直線や矢印の始点と終点の座標を表している。text はどのような文字が書かれているかを表している。

$circle(mid = (20,20),\ radius = 10)$）の場合、円の中心が（20、20）で半径が 10 の円を表している。

また、上記の図形文には出てこないが、このままでは図 2.1 のような計算の木の意味を成さない。実は制約というものが存在していて、定義する際には「円の中心とテキストの中心が一致している」や「矢印の終点と円の中心が一致している」などの制約が必

要となってくる。

CMG は図形文字の集合T_T、図形単語の集合T_{NT}、生成規則の集合Pから構成される。例として次のような計算の木＝（T_T, T_{NT}, P）を定義する。
$T_T = \{circle, text, line\}$、$T_{NT} = \{Node\}$であり、$P$は次のような生成規則から構成されている。

$$
\begin{aligned}
&Node\ (mid, value) ::= \\
&\qquad circle : C\ (mid, radius),\ \ text : T(mid, Ttext)\ \ where \\
&\qquad C.mid = T.mid\ \ and \\
&\qquad mid = C.mid,\ \ valuea = T.text.
\end{aligned}
$$

生成規則は次の形をしている。

$$
\begin{aligned}
&T(\vec{x})\ ::=\ T_1(\vec{x}_1), \cdots, T_n(\vec{x}_n) where \\
&\qquad exists T_1'(\vec{x}_1'), \cdots, T_m'(\vec{x}_m') \\
&\qquad where\ C\ and\ \vec{x}\ =\ F.
\end{aligned}
$$

ここで、Tは生成される図形単語、$T_1, \cdots, T_n$は$n \geq 1$であるような図形単語もしくは図形文字であり、normal の構成要素である。$T_1', \cdots, T_m'$は、$m \geq 0$であるような図形単語もしくは図形文字であり、exist の構成要素である。この図形言語が存在しなければ生成規則は適用されない。$\vec{x}$, $\vec{x}_i$, $\vec{x}_i'$は異なる変数、Cは$\vec{x}_1, \cdots, \vec{x}_n, \vec{x}_1', \cdots, \vec{x}_m'$の間の制約の連接、Fは$\vec{x}_1, \cdots, \vec{x}_n, \vec{x}_1', \cdots, \vec{x}_m'$を引数とする関数である。$T(\vec{x})$は$T_1(\vec{x}_1), \cdots, T_n(\vec{x}_n)$、そして$T_1'(\vec{x}_1'), \cdots, T_m'(\vec{x}_m')$が存在し、その要素の属性が制約Cを満たしているときに、$T_1(\vec{x}_1), \cdots, T_n(\vec{x}_n)$から$T(\vec{x})$が生成される。また、その生成規則が適用されたときに$T(\vec{x})$の属性値が $\vec{x}\ =\ F$ によって定義される。

しかしながら、normal、exist、のみの CMG では計算の木を決定性のある文法として書くことはできない。CMG にはネガティブ制約な制約として not exisit というものがあり、存在しない構成要素を記述することによって決定性を持たせることができる。

$$
\begin{aligned}
&T(\vec{x})\ ::=\ T_1(\vec{x}_1), \cdots, T_n(\vec{x}_n) where \\
&\qquad exists T_1'(\vec{x}_1'), \cdots, T_m'(\vec{x}_m') \\
&\qquad not\ exists T_1''(\vec{x}_1''), \cdots, T_l''(\vec{x}_l'') \\
&\qquad where\ C\ and\ \vec{x}\ =\ F.
\end{aligned}
$$

以前 CMG を使ったシステムはあったがパースをしたらそのままで、その結果を利用することはできなかった。そこで、Eviss ではより記述表現を広げるために Action の概念を導入した。CMG に Action を導入することによってパースされた図形の削除や生成などができるようになった。

Action をもった CMG すなわち拡張 CMG の生成規則は下の式のようになる。

$$
\begin{aligned}
T(\vec{x}) \ ::= \ & T_1(\vec{x}_1),\cdots,T_n(\vec{x}_n) \mathit{where} \\
& \mathit{exists} T_1'(\vec{x}_1'),\cdots,T_m'(\vec{x}_m') \\
& \mathit{not\ exists} T_1''(\vec{x}_1''),\cdots,T_l''(\vec{x}_l'') \\
& \mathit{where\ C\ and\ } \vec{x} \ = \ F \ \mathit{and\ Action.}
\end{aligned}
$$

その他に本論文ではふれられていないが、all という構成要素も用意されている。これは、ある図形文の中で同じ種類の図形単語や図形文字を指定するときに使われる。

2-3 Eviss とは

Eviss はビジュアルシステム生成系である。図形言語を解析するための Spatial Parser を CMG によって定義された文法に基づいて自動的に生成してくれる。また、図形文法に Action を導入することによってさまざまなビジュアルプログラミング言語に対応できる。

Eviss は Spatial Parser Generator と制約解消系を持つ。Spatial Parser Generator は図形文法にしたがって図形言語を解析する部分（Spatial Parser）を生成し、制約解消系は図形間に制約を課す。

図形言語がどういうものかということを定義するときには CMG Input Window を使う。これは、定義 Window にある rule 欄の make new production rule を選ぶと表示される。この CMG Input Window で拡張 CMG を使ってユーザが図形言語を定義すると、その定義にしたがって Eviss はその図形言語を解析するためのパーサ(Spatial Parser)を生成する。そして、1 度パースが行われるとその制約などを使って定義されたものが常にみたされる。このシステムは Tcl/Tk[14]で実装されている。

2-3-1 Eviss の使用方法

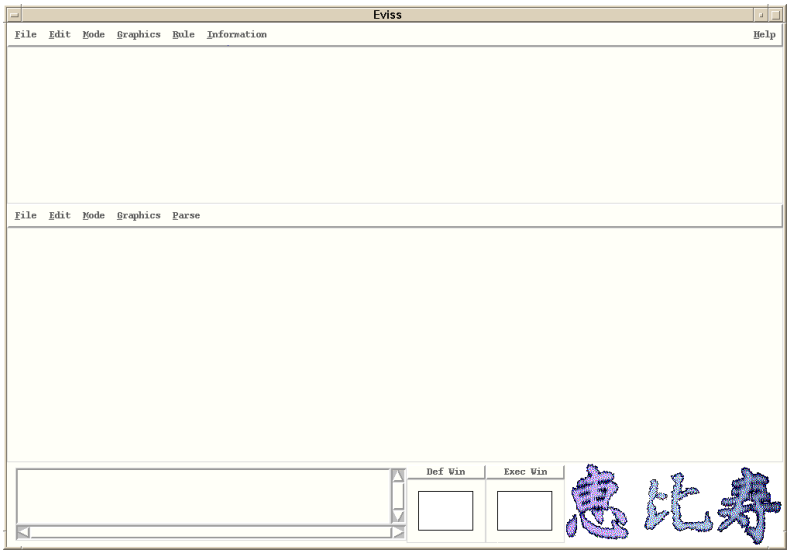

図 2.2　Eviss のシステム概観

Eviss のメイン画面を図 2.2 に示す。上側の Window は定義 Window で、下が実行 Window である。Eviss の使用法は、まず、上側の定義 Window で、図と対応させて 1 つの図形言語につき 1 つのルール（文法）を、CMG Window の形式で作成する。そして、下の実行 Window で解析したい図を描くと、文法に従って解析結果が実行されるようになっている。

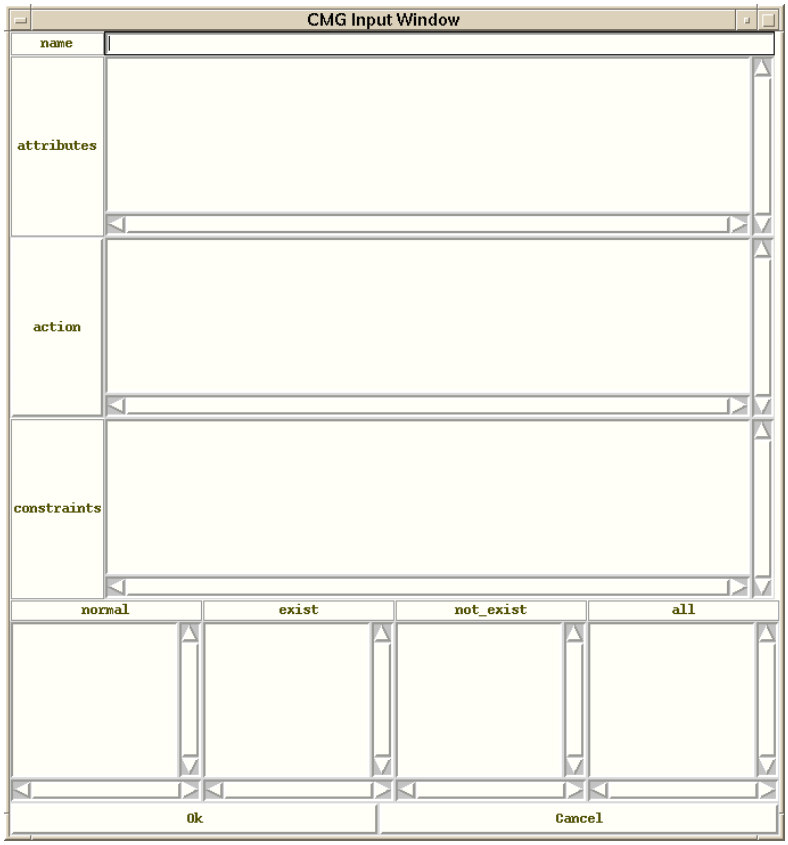

図 2.3 CMG Input Window

CMG Input Window は、図形の属性、アクション、制約を記述する Attributes、Action、Constraints 部分と、図形言語の構成要素の種類を表す normal、 exist、not_exist、all の欄から成る。

2-3-2 従来の入力方法

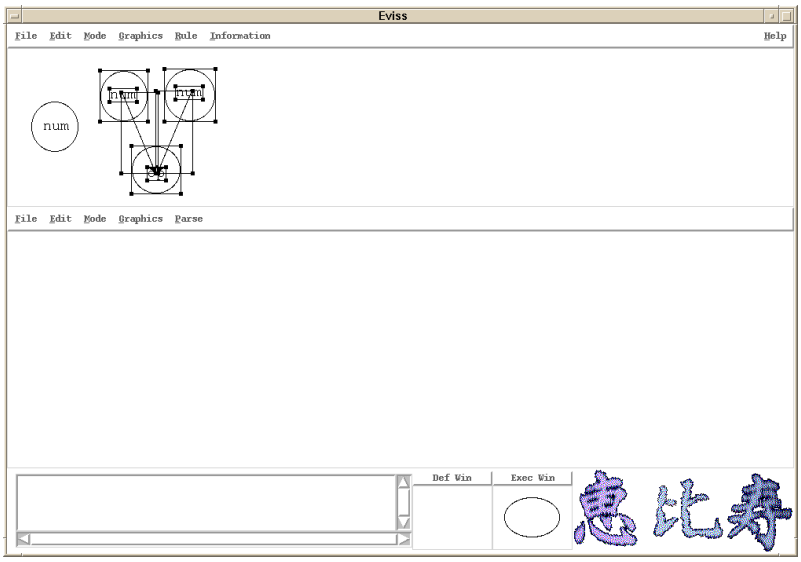

図 2.4 定義 Window に定義したいものを描いて選択した状態

従来の入力方法を計算の木を例にとって説明する。計算の木は 2 つの文法から定義される。

① ノードは、円の中に数字が描いてあり、その円の中心が一致している。

② ノードは、2 つのノードが矢印によって円につながれ、その円の中には演算子が描かれていて、矢印の先と終点は、円とノードの中心と一致する。そして、計算を実行する。

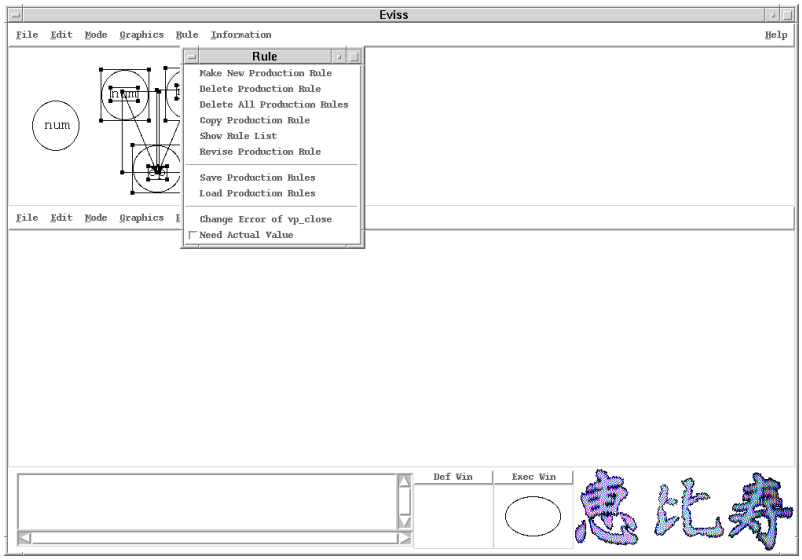

図 2.5　定義したもの選択して rule 欄の make new production rule を選択

図 2.4 の定義 Window には 2 つの図形言語の例が示されている。図形言語の左側が文法の①、右側が②と対応している。1 つの図形言語をセレクトし、メニューの rule 欄から make new production rule を選択すると、CMG Input Window が表示される。この時、CMG Input Window には、自動認識された制約と構成要素が記入されている。

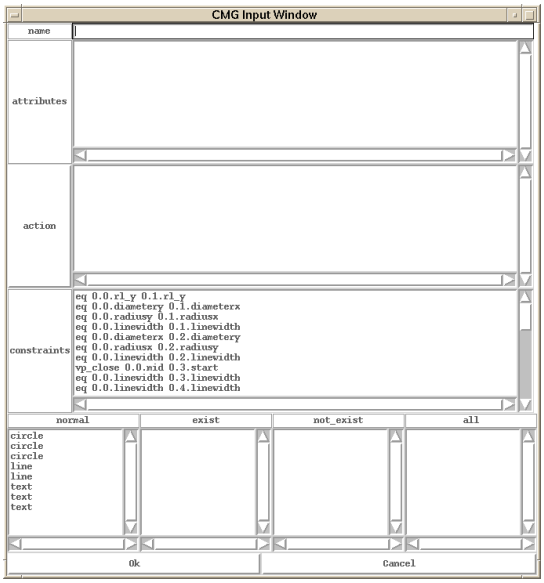

図 2.6　make new production rule を選択し自動認識された構成要素と制約が記入されている状態

以下に②に対応する図形を描き、自動認識させた制約と構成要素を示す。

自動認識させた構成要素

normal ----- text、text、text、circle、circle、circle、line、line

自動認識された制約

eq 0.0.diameterx 0.1.diameterx
eq 0.0.diametery 0.1.diametery
eq 0.0.radiusx 0.1.radiusx
eq 0.0.radiusy 0.1.radiusy
eq 0.0.linewidth 0.1.linewidth
eq 0.0.diameterx 0.2.diameterx
eq 0.0.diametery 0.2.diametery
eq 0.0.radiusx 0.2.radiusx
eq 0.0.radiusy 0.2.radiusy
eq 0.0.linewidth 0.2.linewidth
vp_close 0.0.mid 0.3.start
eq 0.0.linewidth 0.3.linewidth
eq 0.0.linewidth 0.4.linewidth
vp_close 0.0.mid 0.5.mid
eq 0.1.diameterx 0.2.diameterx
eq 0.1.diametery 0.2.diametery
eq 0.1.radiusx 0.2.radiusx
eq 0.1.radiusy 0.2.radiusy
eq 0.1.linewidth 0.2.linewidth
eq 0.1.linewidth 0.3.linewidth
vp_close 0.1.mid 0.4.start
eq 0.1.linewidth 0.4.linewidth
vp_close 0.1.mid 0.6.mid
vp_close 0.2.mid 0.3.end
eq 0.2.linewidth 0.3.linewidth
eq 0.2.mid_y 0.4.end_y

vp_close 0.2.mid 0.4.end
eq 0.2.linewidth 0.4.linewidth
vp_close 0.2.mid 0.7.mid
vp_close 0.3.end 0.4.end
eq 0.3.linewidth 0.4.linewidth
eq 0.3.start_y 0.5.mid_y
vp_close 0.3.start 0.5.mid
eq 0.3.end_x 0.7.mid_x
vp_close 0.3.end 0.7.mid
eq 0.4.start_x 0.6.mid_x
vp_close 0.4.start 0.6.mid
vp_close 0.4.end 0.7.mid

これには、最初に制約の種類が書かれている。eq は、等しい、vp_close は、ほぼ等しい、といった意味である。その他には、lt という大小関係を記述できる制約もある。制約はある属性と属性もしくは定数の間に条件を課すものであるので、制約の種類の後には、どのような属性と属性との間かがかかれている。前の 2 つの数字は CMG Input Window の構成要素の欄の位置を表していて、そこには図形言語の種類が書かれている。そして、３つめで図形言語の属性の種類を表している。

この中から円や直線の線の太さ（linewidth）、円の半径（radiusx, radiusy）などに関する必要ない制約を削除し、足りないものを追加して書く。文法①と②を CMG Input Window に書いて定義した後に計算の木を下の実行 Window で描いてパースすると計算が実行され答えを得ることができる。図 2.7 に②に対応する CMG Input Window を示す。

図 2.7　自分で CMG Input Window を完成させた状態

　完成させた時の制約と構成要素を以下に示す。

構成要素

normal ----- circle、text

exist ----- line、line、node、node

制約

vp_close 1.0.start 1.3.mid

vp_close 1.0.end 0.0.mid

vp_close 1.1.start 1.2.mid

vp_close 1.1.end 0.0.mid

vp_close 0.0.mid 0.1.mid

lt 1.2.right 1.3.left

これらの制約は上から

ノードの中心と矢印の始点がほぼ一致している
矢印の終点と円の中心がほぼ一致している
ノードの中心と矢印の始点がほぼ一致している
矢印の終点と円の中心がほぼ一致している
円の中心とテキストの中心がほぼ一致している
左のノードの右の座標 ＜ 右のノードの左の座標

という意味の制約である。

３. Eviss における Action

Action は、パースした結果を利用して図形の描き換えなどを行うような機構を CMG に導入したものである。

Eviss における Action の実行は、図形言語を認識するときに CMG Window の Action 欄に書いてある Action を実行することによって行われる。また、Action は「生成規則が適用された時に実行されるプログラム」として定義されていて、任意の Tcl[14]のスクリプトを扱うことができる。

Eviss では Action の中で Tcl のスクリプトを書くのに便利な手続きとして、delete（図形の削除）、alter（図形の属性値の書き換え）、create（図形の生成）などを供給している。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化表現を書き換えることによって実行の視覚化を行なうことが多い。図形に関しての描き換えや生成の際には、この 3 つの Action でほとんど対応することができる。そのために Action のなかでも delete（図形の削除）、alter（図形の属性値の書き換え）、create（図形の生成）は重要であると考える。

これらは Action 中で次のように記述する。

生成　　*create* 生成するものの type とその座標
　　line ならば始まりの座標と終わりの座標を記述
　　四角形ならば対角線の左上と右上の座標を記述
　　円ならば中心座標と半径を記述する

削除　　*delete* {@id@ @id@ … @id@}
　　id は削除する図形文字、図形単語、図形文の id である

属性値の書き換え　　*alter* @書き換えられる構成要素の id@ 書き換えられるものの属性 @書き換えるもの@

ここでは@を前後につけることによって構成要素や属性値を参照することが可能になっている。

以下に、2-3-2 章で述べた文法②で使われている Action のスクリプトを示す。上式

（delete）は、図形の削除でノード２つとライン２本を削除するのを表している。@で囲まれた数字の１つめが構成要素の種類を表している。２つめはその構成要素の何番目かを表している。下式（alter）は、図形の書き換えで、演算子を計算結果に書き換えることを表している。左の@で囲まれた数字が書き換えられるもので、右の@で囲まれたものが書き換えるものを表している。

```
delete {@1.0@ @1.1@ @1.2@ @1.3@}
alter @0.1@ text @value@
```

実際に実行すると図のようになる。図 3.1 はパースする前の図である。図 3.1 の状態のもの（３＋４の計算の木）を書いてパースすると、図 3.2 のように Action が行われノードと矢印が削除されて演算子が計算の結果に書き換わる。

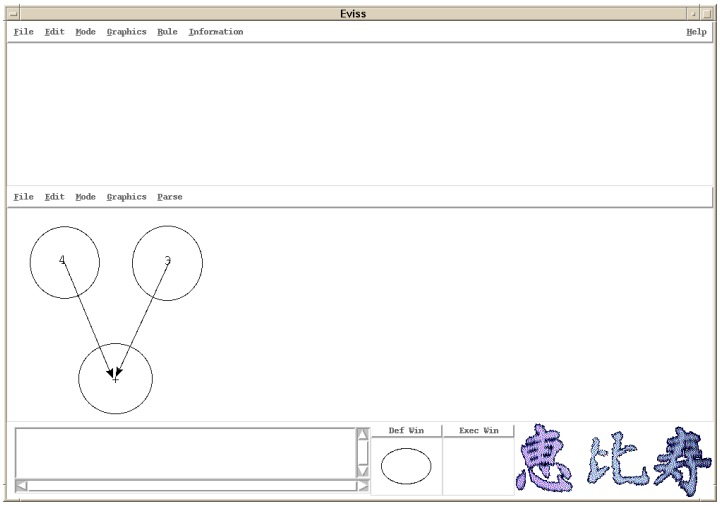

図 3.1　Action が実行される前（パースされる前）

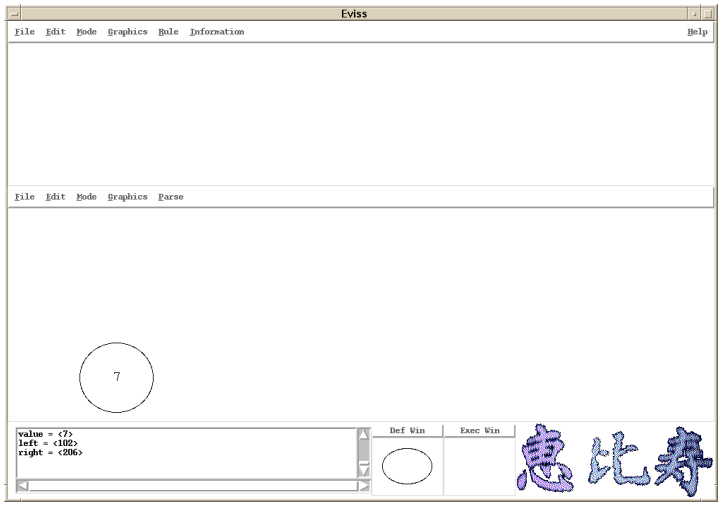

図 3.2　パースして Action が実行された後

4. Action の視覚化

4-1 視覚化手法

Eviss では CMG をテキストで入力するが、これは、ユーザにとって直感的でない。これは、図形文法を自分で定義して記述する際には図形間の関係は 2 次元的な情報が扱われていが、Eivss ではこの関係を 1 次元的なテキストで入力するからであると思われる。そこで、ユーザにとって理解しやすいように入力を図形を用いて視覚化することが重要である。また、現在の入力手法では、テキスト入力時の@で囲まれた部分をわざわざ下の構成要素の欄から参照したり、定義したい図と比較しながら入力しなければならない。よって一個所で定義と参照できることをめざす。また、Action の中でも図形を扱う際には delete（図形の削除）、alter（図形の属性値の書き換え）、create（図形の生成）でほとんど対応できるためこの 3 つの Action について視覚化することは重要である。

delete（図形の削除）、alter（図形の属性値の書き換え）の視覚化手法を計算の木を使い、create（図形の生成）はリストを使って以下に示す。まず最初に現在の使い方と同様に定義画面に定義したい図を描く。

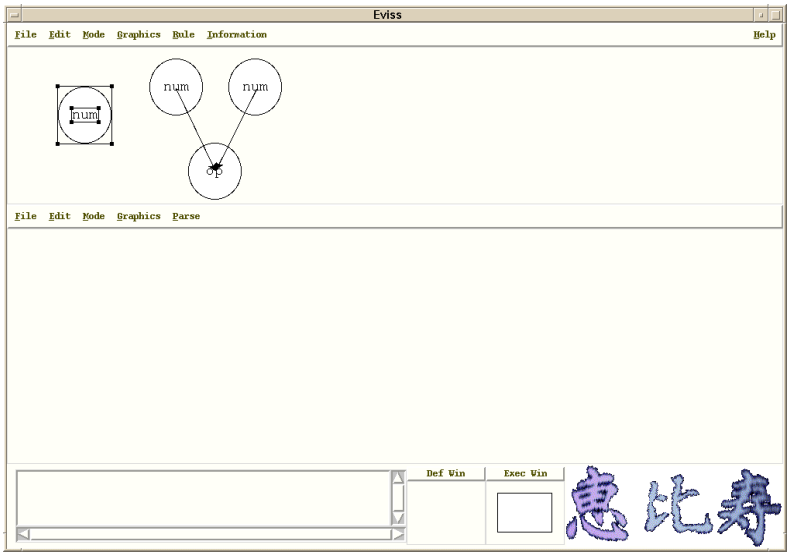

図 4.1　定義したい図形言語を選択した状態

そして、図形言語をセレクトし、メニューの rule 欄から make new production rule

を選択すると、CMG Input Window が表示する。そうしたら Action 以外の図形言語の名称、図形言語の属性、図形言語の制約、図形言語の構成要素の欄をうめる。最後に、Action 欄のボタンを押す。そうすると Action 定義 Window が表示される。

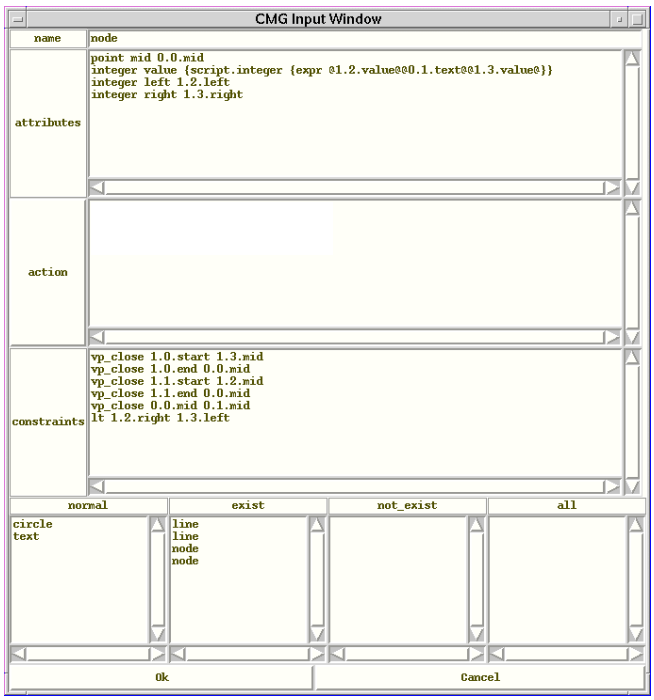

図 4.2　Action 以外の欄をうめた状態の CMG Input Window

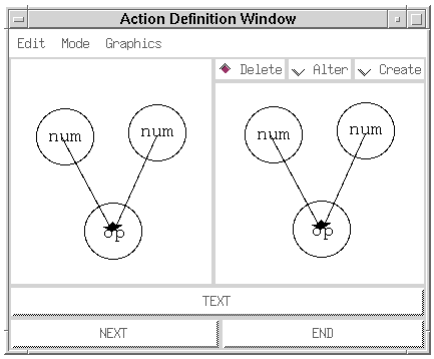

図 4.3　Action 定義 Window

4-2 Delete の視覚化

直感的にするために図形言語を消す作業をそのまま図形言語の削除として定義できる

ようにした。まず、最初に Action の種類かの delete を選択する。そして、削除したい図形を選択し実際に消す。

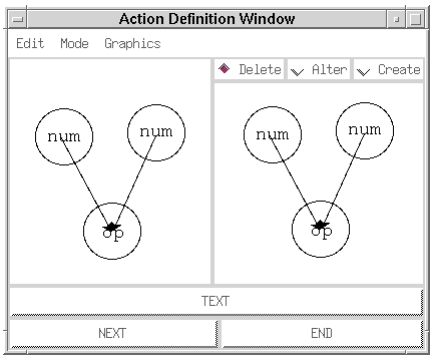

図 4.4　delete を定義する前の状態

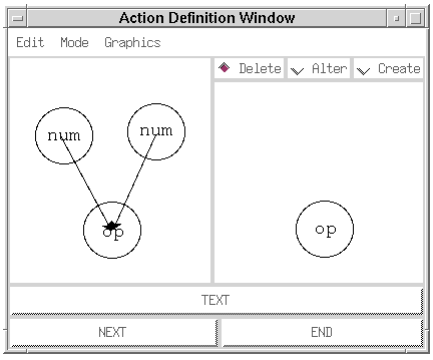

図 4.5　delte を定義したあと

定義した後に NEXT のボタンを押すと CMG にその内容が反映され、次の Action 定義 Window が出てくる。

図 4.6　Action 定義 Window の内容（delete 図形の削除）が反映された CMG Input Window

4-3 Alter の視覚化

delete 入力終了後に右下の NEXT を押すと、前の Action の結果が左側に描かれている画面が現れる。alter は、◇Alter を押し右にコピーされたものを書き換えることによってそれを表現する。描き換える際に消したものを“描き換えられるもの”、描き換えたものを“描き換えるもの”という具合に認識させる。ここで、@value@というのは計算結果の値の事で、前後の@は CMG で構成要素の属性の値を参照するために用いられる記号である。図に、書き換えた後の Window を示す。

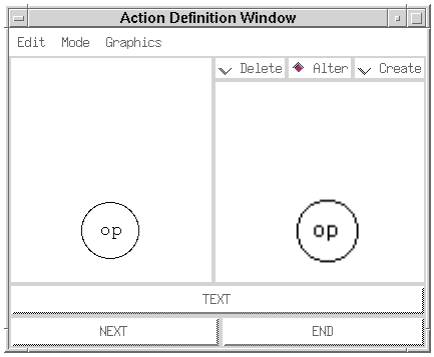

図 4.7　Alter を定義する前の Action 定義 Window

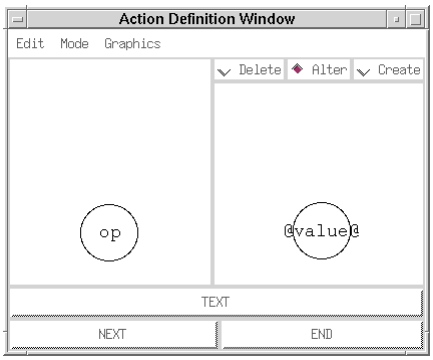

図 4.8　Alter を定義した後の Action 定義 Window

4-4 Create の視覚化

Create は計算の木ではなくリストをつかって説明する。◇create を押して実行後の Window の図形言語に、実際に生成したい図形言語を書き加えることによって create を表現する。このとき、書き加えられた図形言語の座標は、既にある図形言語の座標との相対的な位置から計算される。図 4.9 に図形言語を生成した後の図を示す。

このように定義し四角を１つ描けばリストがどんどん生成される。制約の欄に描かれる四角の数の上限を指定すると指定した数のリストを作ることができる。

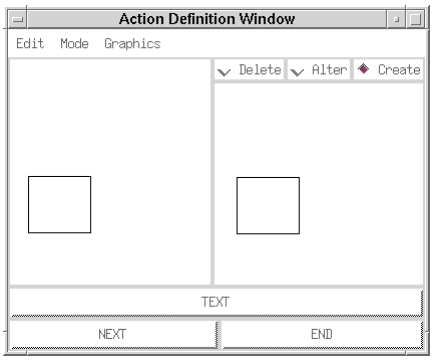

図 4.8　Create を定義する前の Action 定義 Window

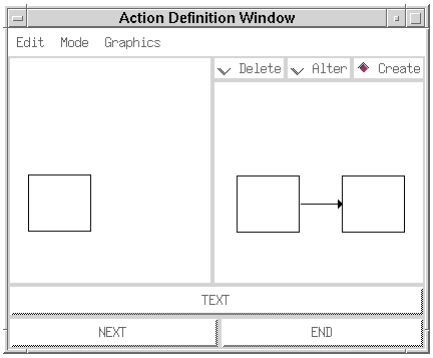

図 4.9　Create を定義した後の Action 定義 Window

4-5 応用例

視覚化の応用例として、「１本の線を引いたら直角にそれより短い直線が引かれていく」という例をあげる。この例は渦巻き状に次々と直線が引かれていく。図 4.10 は、定義する前の画面である。図 4.11 で直線太さを１にしている。図 4.12 は create の定義前の図である。図 4.13 で同じ長さの直線を前の直線の last の座標とこれから引く直線の start の座標を一致させて直角に引く。この時に前の直線より短い直線になるように属性で直線の終点の座標を計算している。図 4.14 に実行結果の例をあげる。

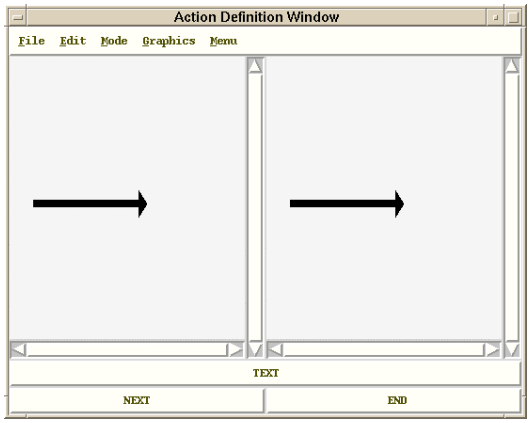

図 4.10　alter の定義前

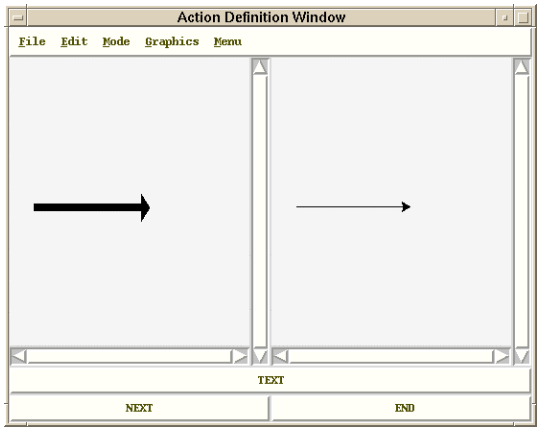

図 4.11　alter 定義後（直線の属性の書き換え）

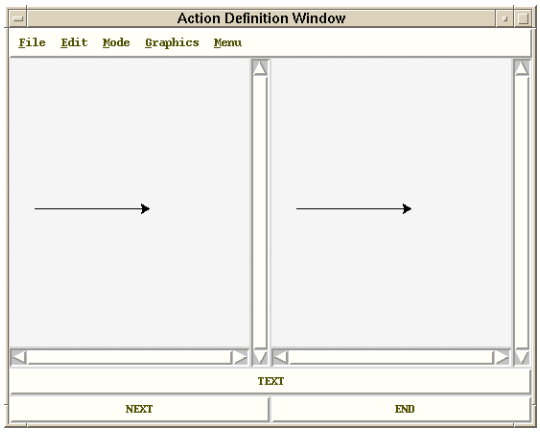

図 4.12　create の定義前

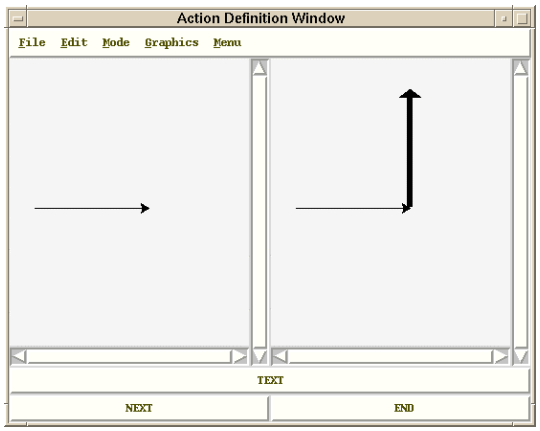

図 4.13　create の定義後（直線の生成）

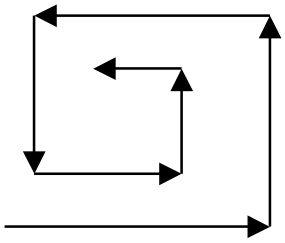

図 4.14　実行結果

テキストの入力は以下のようになる。

　属性では前に引いた直線の太さから１ひく計算と次に引く直線の終わりの座標を計算している。アクションでは引いた直線の太さを１に書き換え、次の直線を生成している。制約では線の太さを２以上としている。構成要素は lline だけである。

属性（attribute）

```
integer lw {script.integer {expr @0.0.linewidth@ - 1}}
integer ex {script.integer {expr @0.0.end_x@-((@0.0.start_y@-@0.0.end_y@)*7)/8}}
integer ey {script.integer {expr @0.0.end_y@-((@0.0.end_x@-@0.0.start_x@)*7)/8}}
```

アクション（action）

alter @0.0@ linewidth 1

create line @0.0.end_x@ @0.0.end_y@ @ex@ @ey@ -width @lw@

制約（constraints）

ge 0.0.linewidth {integer 2}

構成要素

normal

line

５．連続した Action の表示手法

４章で delete（図形の削除）や alter（図形の属性の変更)、create（図形の生成）の視覚化をすることによって直感的になり理解しやすくなった。

しかし、このように delete（図形の削除）や alter（図形の属性値の変更）などの Action が複数個あった時に、複数の Window を表示したり、いくつも横に並べたりすると見づらくなる。そこで、複数の Action 定義 Window を縦につないでミニチュアで表示する方法を提案する。

ここでは、縦に隣り合った 2 枚で 1 つの Action を表している。ミニチュアにして全体の Action を表示する事によってすべての Action を見ることができる。マウスのカーソルをミニチュアの Window にあわすとその部分が拡大表示されて見える。図にその表示図を示す。

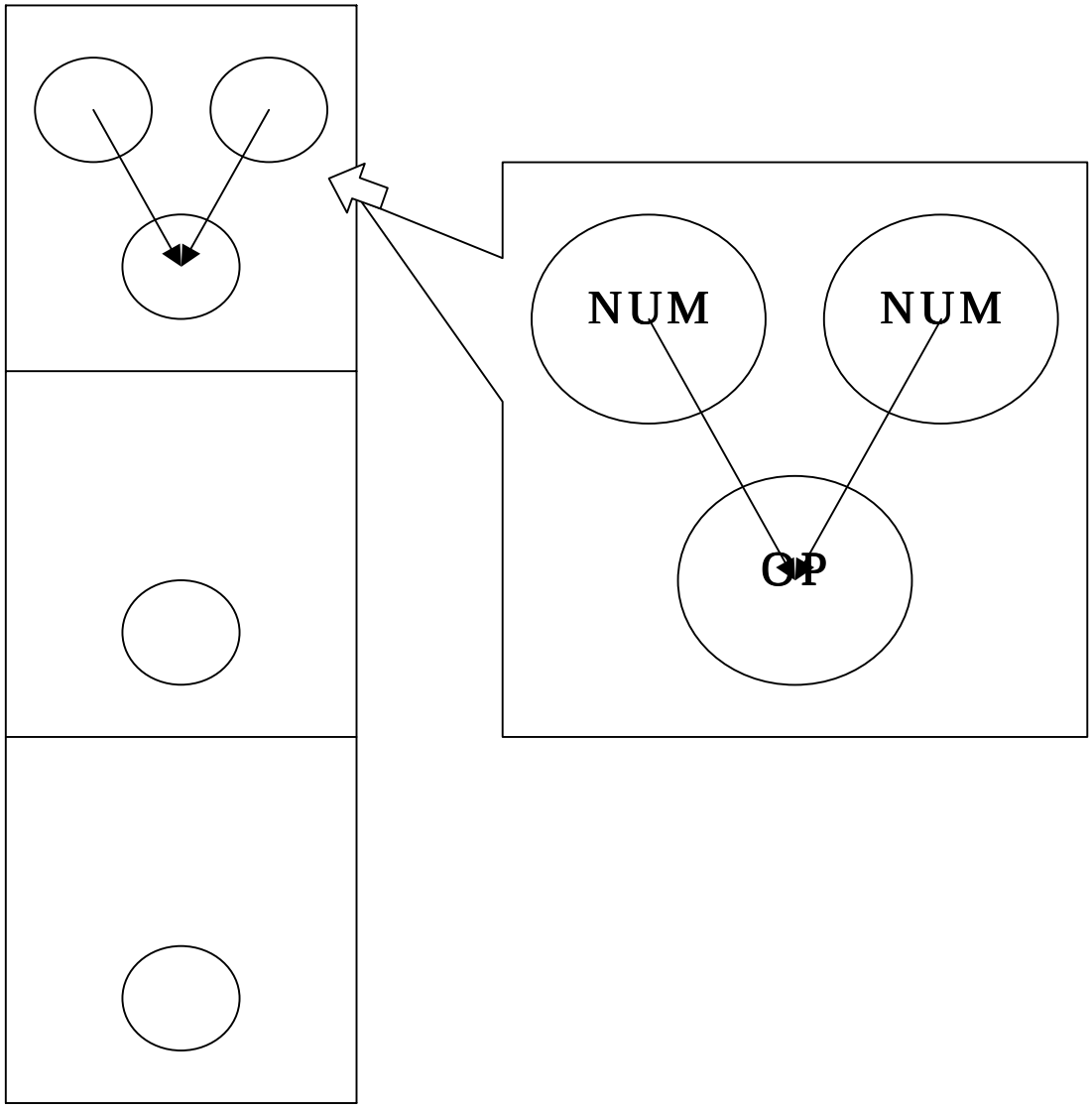

図 5.1　Action 複数時の表示手法（マウスカーソルがミニチュアの Window にあわされて拡大表示されている状態）

これは、４章の計算の木を使用し、Action が複数ある場合である。delete（図形の削除）の定義に使用された Window を縦につないで１番目と２番目のミニチュア Window によって表現している。そして、Alter（図形の属性の書き換え）の定義に使用された Window を縦につないで２番目と３番目の Window を表現している。この時、delete の結果と alter の実行前の Window が同じなので省略している。また、ミニチュアにする際にある程度構成要素を省略している。左のミニチュアの Window の１番上には計算の木の最初の状態が描かれている。ノードの削除が行われていない。２番目は、ノード２つとライン２本が削除されて演算子と円が残った状態、３番目にはその円の中の演算子

が計算の結果に書き換わっている。マウスをミニチュアにした Window の上から下へなぞっていくと拡大された Window が流れていき Action の流れが把握しやすくなっている。

６. Eviss と他のシステムとの比較

6-1 Action を導入して拡張された CMG を持った Eviss と属性文法

属性文法[15]とは文脈自由文法を拡張したものである。静的意味を定式化する方法の代表的なものでプログラムの意味を形式的に記述することが可能である。属性文法は、属性、意味規則、属性評価、属性評価器から成る。属性は、文脈自由文法の各文法記号に意味の情報を表すものとして付加するものである。意味規則は属性に対する値を決めるもので生成規則に付随している。属性評価は属性の値を計算することである。属性評価器は属性評価を行うプログラムのことである。
属性文法では属性評価にあたる部分は Eviss の Action にあたる。

6-2 ビジュアルシステム生成系 Eviss と YACC

YACC（Yet Another Compiler Compiler）[15]は Lex で字句解析してものを使い構文解析するものである。構文解析を行うＣのプログラムを自動生成するツールである。規則は BNF によく似た記述になっていて構文解析のアルゴリズムには LALAR(1)が使用されている。動作（Action）には任意の C プログラムを書くことができる。
Eviss と違う点は、まず、Eviss はビジュアルなものも扱えるが YACC はテキストしか扱えない所である。そして YACC は C プログラム（コンパイラ）を生成するが Eviss では解析するためのプログラムを生成しない。

6-3 Eviss と Penguins

Penguins[18]は、図形言語の定義をテキストだけを使用して行なっている。定義には Eviss と同様に CMG が用いられている。しかしながら Eviss では定義の段階から図形を使用していてよりビジュアルでインタラクティブなシステムとなっている。

6-4 その他のシステム

Mondorian (Henry Lieberman 1993)

基本的な図形をいくつか組み合わせたり引いたりして新しい図形を定義する過程を視覚化している。そして、その定義する過程の1番最初と最後の１枚づつをつかってコマン

ドとして表示している。

Chimera(Divid Kurlander, Steven Feiner 1993)

図形エディタでの作業編集過程を視覚化している。フォントの変更や図形のコピーなどの作業の１つ１つをパネルにして左から右にならべている。

7. 結論

我々は、図形言語を描き換えたりする際によく用いられる Action の delete（図形の削除)、alter（図形の属性の変更)、create（図形の生成）を視覚化し、編集できるようにした。視覚化することによってより直感的になり、Action の実行結果を見ることができるようになった。そして、テキスト入力ではわかりにくかった構成要素の欄から参照して対応を調べることや、定義したい図と比較しながらの入力が、1 個所で理解できるようになった。また、視覚化するだけでなく、Action 複数時の表示を工夫することによって Action の流れを把握しやすくなった。

謝辞

本研究を進めるにあたりご指導くださった主査の田中二郎教授および副査の福井幸男教授、細野千春助教授に心から感謝いたします。また、研究全般においてなにかと助けていただいた飯塚和久さん、丁錫泰さん、藤山健一郎さん、奥村穂高さんに感謝します。そして、田中研究室の皆さんには研究の方針などゼミの際に貴重な意見を頂きました。ここに感謝の意を表します。

参考文献

[1] 馬場昭宏, 田中二郎: Spatial Parser Generator を持ったビジュアルシステム, 情報処理学会論文誌, Vol.39, No5, pp1385-1394, 1998

[2] E. J. Golin and T. Magliery: A Compiler Generator for Visual Languages. *Proceedings of the 1993 IEEE Symposium on Visual Languages*, pp.314-321, 1993

[3] S. S. Chok and K. Marriott: Automatic Construction of User Interfaces from Constraint Multiset Grammars. *Proceedings of the 1995 IEEE Workshop on Visual Languages*, pp.242-249, 1995

[4] K. Marriott: Constraint Multiset Grammars, *Proceedings of the 1994 IEEE Symposium on Visusal Languages*, pp245-252, 1994

[5] 馬場昭宏, 田中二郎: 「恵比寿」を用いたビジュアルシステムの作成, 情報処理学会論文誌, Vol.40, No2, pp497-506, 1999

[6] A. Baba and J. Tanaka: Eviss: A Visual System Having a Spatial Parser Generator, *Proceedings of Asia Pacific Computer Human Interaction* 1998 (APCHI'98), July, pp.158-164, 1998

[7] 馬場昭宏, 田中二郎: GUI を記述するためのビジュアル言語. インタラクティブシステムとソフトウェアⅤ, pp.135-140. 近代科学社, 1997

[8] H. Lieberman: Mondrian: A Teachable Graphical Editor, *Watch What I do*, pp340-338, 1993

[9] D. Kurlander and S. Feiner: A History-Based Macro by Example System, *Watch What I do*, pp322-338, 1993

[10] P. P. Piernot and M. P. Yvon: The AIDE Project: An Application-lndependent Demonstrational Enviroment, *Watch What I do*, pp382-401, 1993

[11] D. Kurlander: Chimera: Example-Based Graphical Editing, *Watch What I do*, pp270-290, 1993

[12] Y. Harada, K. Miyamoto, and Y. Inagaki: VISPATCH: Graphical rulebased language controlled by user event. *Proceedings of the 1997 IEEE Symposium on Visual Language*, 1997

[13] I. Yoshimoto, N. Monden, M. Hirakawa, M. Tanaka, and T. Ichikawa: Interactive Iconic Programming Facility in HI-VISUAL. *Proceedings of the 1986 Workshop on Visual Languages*, pp23-41, 1986

[14] 宮田重明, 芳賀敏彦共著: tcl/tk プログラミング入門, オーム社, 1995

[15] 佐々政孝　プログラミング言語処理系　岩波書店, 1989

[16] 西名　毅, 田中二郎 ビジュアルシステム生成系 Eviss における Action の視覚化, 日本ソフトウェア科学会 第１６回, pp9-12, 1999

[17] E. J. Golin. :Parsing Visual Language with Picture Layout Grammars. *Journal of Visuzl Languages and Computing*, No2, pp.371-393, 1991

[18] S. S. Chok and K. Marriott : Automatic Construction of Intelligent Diagram Editors, *Proceedings of the ACM Symposium on User Interface Software and Techolgy*, pp185-194, 1998