ビジュアルシステムにおける
図形文法の定義方法に関する研究

工学研究科

筑波大学

2004年3月

亀山 裕亮

概要

図形を扱うビジュアルシステムでは、入力はある規則のもとに組み合わせられた長方形や円などである。この規則を定義する文法を図形文法と呼び、ビジュアルシステムの図形文法記述からパーサを自動的に作成する生成系を空間解析器生成系と呼ぶ。しかしこれまで図形文法の定義において、構成要素の 2 次元的な関係の記述を、テキストによる 1 次元的な記述を用いて行なっていた。そのため、文法が表わしている意味を直感的に理解しつつ定義をすることが困難であった。本論文では図形文法として拡張 CMG を対象としている。拡張 CMG は構成要素、属性、制約、アクションから構成されているが、これらの 2 次元な規則についてはテキストによる定義ではなく、図式への直接操作を用いて定義することで、視覚的に入力できる。

本論文では、ビジュアルシステムの開発時における図形文法の定義の複雑さと、定義を行っている文法の意味の理解の困難さを解決するために、図式表現を用いてグラフィカルに拡張 CMG を定義する図形文法定義システム GIGA を実現した。GIGA では、生成規則の各構成要素を視覚的に表現し、それらの要素に対する直接操作を行うことで、生成規則の定義を行うことができ、さらにひとまとめに図式表現された図形文法の生成規則からその意味を容易に把握できる。GIGA を用いることで、従来の空間解析器生成系において図形文法を定義する際に問題となっていた、制約の把握・定義、属性の記述の煩雑さ、アクションの結果の把握しづらさ、などを解決でき、ビジュアルシステムを効率よく開発することが可能になった。

次に、図形文法定義システム GIGA に対して、空間解析器生成系と制約解消系を統合し、グラフィカルな定義インタフェースを持つ空間解析器生成系 Viola を実現した。Viola では図形文法の定義とビジュアルシステムの実行の作業を一貫した一つの図形エディタ上で行うため、定義を行っている文法を確認しなが

らビジュアルシステムの実行を行うことができる。さらに、図形文法の定義に用いている図形に対しても文法に基づいた解析を行うことにより、再帰的な生成規則の定義を容易に行うことができる。

本論文で述べたこれらの技法を用いることで、従来の空間解析器生成系において問題となっていた、制約の把握・定義、属性の記述の煩雑さ、アクションの結果の把握しづらさ、などの定義の際の問題を解決できる。さらに、グラフィカルに図形文法を定義し、同時にビジュアルシステムの実行を行うことにより、文法の正しさを検査するという作業が容易になる。これらにより、空間解析器生成系を用いたビジュアルシステムの開発作業を効率よく行うことが可能になった。

目 次

図 目 次

表 目 次

第1章　序論

本章では、まず図形を扱うビジュアルシスエムと空間解析器生成系について述べる。次にこれまでの空間解析器生成系における問題点を明らかにし、本研究の目的を述べる。

1.1　ビジュアルシステムと空間解析器生成系

E–R 図 [Sil86]、OMT のオブジェクト図 [Rum91, Rum92]、回路図、状態遷移図、数式などの図式は長方形や円などの図形を決められた規則のもとに組み合わせることで描かれている。このような図式を我々は図形言語と呼ぶ。図形言語は、ソフトウェアの設計を行う際や、データ構造を表す時などに用いられる。図形言語に対して、図式を自動的にレイアウトする、回路図のシミュレーションを行う、などといった処理を行なうシステムをビジュアルシステムと呼ぶ。

従来、ビジュアルシステムは個々の図形言語に特化した形で実現されてきた。すなわち、ビジュアルシステムを開発する際には、始めにそのシステム固有の図形言語の仕様を決定し、次にその図形言語を解析するためのビジュアルシステムを一から実現する、という手順を踏む。このような実現形態では、図形言語の仕様に変更が生じた場合、ビジュアルシステムをはじめから開発し直す必要がある。

これに対して、図形言語の図形間の規則を定義するための文法を図形文法と呼び、図形文法記述からビジュアルシステムのための空間解析器を自動的に生成する生成系を空間解析器生成系と呼ぶ。空間解析器とはビジュアルシステムのユーザが入力した長方形、円、線分、テキストなどの基本図形に対して、それらの図形間の空間的な構造の解析を行なう解析器のことである。一般のプログラミング言語の処理系では、プログラミング言語の文法記述から自動的に解

析器を生成する生成系として Yacc[Joh79]、Bison[Cor91]、Rie[Sas93] などが存在する。空間解析器はプログラミング言語における構文解析器や意味解析器に相当する。

空間解析器生成系を用いることで、ビジュアルシステムの空間解析器を簡単に作成できる。また、図形言語の仕様に変更が生じた場合にも、図形文法を修正するだけで新しい空間解析器が得られるため、ビジュアルシステムの開発を効率的に行うことができる。

1.2　従来の空間解析器生成系の問題点

これまで、ビジュアルシステムのための空間解析器生成系についてはいくつか提案されているが、これらのシステムには 2 つの問題があった。

1 つは図形文法の定義方法に関する問題である。従来の空間解析器生成系では図形文法の記述はテキスト形式による記述が用いられていた。図形文法の記述は、図形言語を構成する図形 (長方形、線分、画像など) 同士が満たすべき制約 (含む、接する、中央に存在する等の位置関係や色など) などを記述した規則の集合である。仕様の編集や確認作業では、このような図形の位置関係のような 2 次元の構造をテキスト形式の仕様記述から読み下し、その構成要素を把握し、関係を理解する必要がある。しかしながら、テキストによる記述では仕様記述が表している意味を直感的に理解することが困難であった。

もう 1 つはビジュアルシステムの実行時における問題である。従来の空間解析器生成系では、図形文法の定義を行う画面と実行を行う画面が別々であった。ユーザは図形文法を定義しながら図形を描いて文法の正しさを検査することを繰り返すことよって文法を定義していく。しかしながら、定義と実行の画面が分かれていると、どのような文法が定義されているのかを実行時に確認することができないため、その作業が妨げられてしまうという問題があった。

1.3　研究の目的

本研究では従来の空間解析器生成系における問題点を解決し、空間解析器生成系を用いたビジュアルシステムの開発環境を実現する。

まず、図形文法の定義方法の問題に対しては、図形文法をグラフィカルに編集するための手法を提案し，図式表現を用いたグラフィカルな図形文法編集システム GIGA を実現した．GIGA を用いてグラフィカルに文法の編集を行うことにより，図形文法をより直感的かつインタラクティブに編集することができる。

次に、ビジュアルシステムの定義と実行に関する問題に対しては、グラフィカルな図形文法定義インタフェースと空間解析器生成系を統合することで、グラフィカルな図形文法定義インタフェースを持つ空間解析器生成系 Viola を実現した。Viola では図形文法の定義とビジュアルシステムの実行の作業を一貫した一つの図形エディタ上で行うため、定義を行っている文法を確認しながらビジュアルシステムの実行を行うことができる。さらに、図形文法の定義に用いている図形に対しても文法に基づいた解析を行うことにより、再帰的な生成規則の定義を容易に行うことができる。

1.4　本論文の構成

本論文の構成は以下の通りである。2 章では図形文法や空間解析器生成系について用語の解説を行う。3 章では、テキストを用いて図形文法を定義する際の問題点について述べた後、提案手法に基づいて実現したグラフィカルな文法編集システム GIGA について述べる。4 章では、GIGA に空間解析器生成系の機能と制約解消系の機能を追加することで、図形エディタ上でビジュアルシステムの定義と実行を同時に行うことが可能な空間解析器生成系 Viola について述べる。5 章では、本研究の貢献について述べる。

第2章　準備

本章では、準備として図形文法や空間解析器生成系について用語の解説を行う。まず、図形文法である CMG について述べる。次に CMG にアクションの記述を追加した拡張 CMG について述べる。最後に、具体的な図形言語の例を用いて拡張 CMG の記述例を示す。

2.1　図形文法

図形言語における図形間の規則や処理を定義するものが図形文法であり、プログラミング言語における文法に相当する。図形文法とそれを解析するアルゴリズムについては Positional Grammars (PG)[Cos93, Cos98]、Relation Grammars (RG)[Fer94]、Picture Layout Grammars (PLG)[Gol91]、Constraint Multiset Grammars (CMG)[Mar94] などが提案されている。各図形文法は記述方法や記述することのできる図形言語のクラスなどに違いがある。我々は、記述することのできる図形言語のクラスが広い、記述が簡潔であるといった特徴から、対象とする図形文法に CMG を採用した。

2.1.1　CMG

CMG は Marriott らによって提案された図形文法である [Mar94]。Marriot らは文献 [Mar96] において PG、RG および Uniffication Grammar[Wit91] を用いて表した文法はすべて CMG を用いて書き換えることができることを示している。また、他の図形文法では図形言語の規則の構成要素や制約を混在させて記述するが、CMG ではこれらを別々に記述することができるため、記述がわかりやすい。Chok らは文献 [Cho98] において、CMG を用いて記述されたビジュ

アルシステムの例として、状態遷移図、数式、フローチャート、2 分木、n 分木などの図形言語のエディタを報告している。

CMG では図形間の規則を生成規則 (プロダクション・ルール) として定義し、生成規則を複数定義することにより図形言語を定義する。CMG における生成規則の構文を以下に示す。

$$
\begin{array}{rcl}
T & ::= & T_1, \cdots, T_n \ \ \textbf{where} \ (\\
& & \quad Constraints \\
& &) \{ \\
& & \quad AttributeAssignments \\
& & \}
\end{array}
$$

生成規則の左辺 T は図形単語と呼ばれるトークンである。 生成規則が適用されると新規に図形単語が生成される。右辺の $T_1, \cdots, T_n$ は生成規則の構成要素と呼ばれるトークンである。 構成要素となることができるのはエディタが標準で備えている長方形や円などの基本図形、あるいは図形単語である。$Constraints$ は制約条件であり、各構成要素が満たさなければならない制約が、構成要素の持つ属性間の等式関係や、不等式関係を組み合わせた条件式として示されている。$AttributeAssignments$ は属性定義であり、図形単語 T の属性の定義を行なう。つまり、制約条件をすべて満たす図形や図形単語が入力された場合、それらの図形を構成要素として生成規則が適用され、図形単語 T へと還元される。

上記の生成規則で用いられる構成要素は normal な構成要素と呼ばれる。CMG ではこの他に exist、not exist、all という構成要素を用いることであいまいな文法に決定性を持たせることができる。これらを含めた場合の生成規則の構文を以下に示す。

$$
\begin{array}{rcl}
T & ::= & T_1, \cdots, T_n \ \ \textbf{all} \ \ T_1''', \cdots, T_n''' \ \ \textbf{where} \ (\\
& & \quad \textbf{exist} \ \ T_1', \cdots, T_n' \ \ \textbf{where} \ (\\
& & \quad\quad ExistConstraints \\
& & \quad)
\end{array}
$$

$$
\begin{array}{l}
\quad \textbf{notexist}\ \ T''_1, \cdots, T''_n\ \ \textbf{where}\ (\\
\qquad NegativeConstraints \\
\quad) \\
\quad Constraints \\
)\{ \\
\quad AttributeAssignments \\
\}
\end{array}
$$

$T_1, \cdots, T_n$ の構成要素が normal 構成要素である。normal 構成要素であるトークンは生成規則が適用された時に図形単語へと還元される。$T'_1, \cdots, T'_n$ の構成要素は exist 構成要素である。生成規則が適用されるために図全体の中のどこかに存在する必要がある構成要素を exist として定義する。exist 構成要素のトークンは図形単語の部品としては使用されない。$T''_1, \cdots, T''_n$ の構成要素は not exist 構成要素である。not exist で指定された構成要素の属性を用いた制約条件はネガティブ制約条件と呼ばれる。通常の制約条件と違い、ネガティブ制約条件を満たしている場合には他の制約条件を満たしていたとしても、生成規則は適用されない。$T'''_1, \cdots, T'''_n$ の構成要素は all 構成要素である。all 構成要素とは制約条件を満たす全てのトークンのことである。not exist 構成要素、exist 構成要素、all 構成要素にはすでに図形単語に還元されたトークンも指定することができる。

2.1.2　拡張 CMG

図形言語の規則を CMG の文法に従って定義することにより、図形の解析を行なう空間解析器を生成することが可能となる。しかし、実際のビジュアルシステムにおいては図形の解析だけではなく、図の書き換えのような解析結果に応じた動作を行う必要がある。このような動作は CMG では記述することができない。そこで馬場らは生成規則が適用される時に、図形の追加や削除、図形の属性の変更などを行えるように、CMG にアクションの概念を導入した拡張 CMG を提案した [Bab98b]。

拡張 CMG における生成規則の構文を以下に示す。

$$
\begin{array}{lll}
T & ::= & T_1, \cdots, T_n \ \ \textbf{all} \ \ T_1''', \cdots, T_n''' \ \ \textbf{where} \ (\\
& & \quad \textbf{exist} \ \ T_1', \cdots, T_n' \ \ \textbf{where} \ (\\
& & \quad\quad ExistConstraints \\
& & \quad) \\
& & \quad \textbf{notexist} \ \ T_1'', \cdots, T_n'' \ \ \textbf{where} \ (\\
& & \quad\quad NegativeConstraints \\
& & \quad) \\
& & \quad Constraints \\
& &){\{} \\
& & \quad AttributeAssignments \\
& &){\{} \\
& & \quad Actions \\
& & {\}}
\end{array}
$$

Actions は「生成規則が適用されたときにスクリプト言語のプログラムとして実行される文字列」として定義する。拡張 CMG で追加されたアクションとは図形の生成 (create)、図形の削除 (delete)、図形の属性値の書き換え (alter) である。

2.2　図形言語の例

ここで拡張 CMG による図形言語の定義の例として、2 つの図形言語を取りあげる。まず、図形の解析だけを行う例として、文献 [Cho03] に文法記述が示されている状態遷移図を取りあげる。次に、図形の解析だけではなくアクションによる図形の書き換えを行う例として、文献 [Bab98a, Bab98b, Bab99] に文法記述が示されている計算の木を取りあげる。

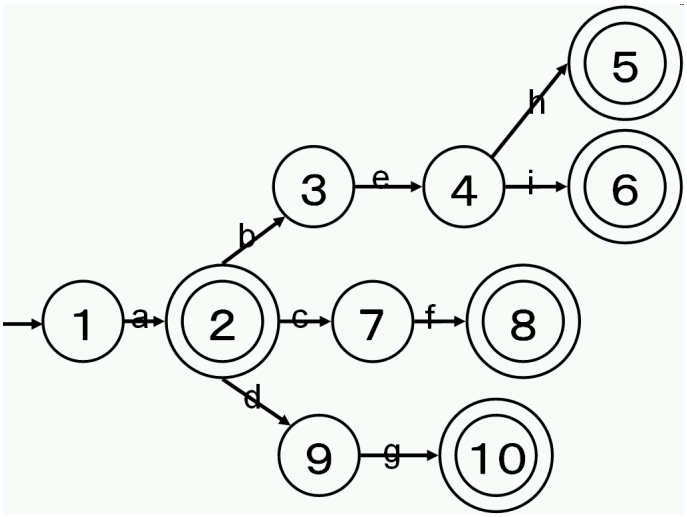

図 2.1: 状態遷移図

2.2.1　状態遷移図

まず、状態遷移図の図式を図 2.1 に示す。文献 [Cho03] では状態遷移図を 10 個の生成規則により定義している。

- 1 つ目の生成規則は図 2.2 に示す状態から状態への遷移を表わす線 (arc) の定義である。1 行目では図形単語 arc は矢印 (arrow) である A とテキスト (text) である T により構成されていることを定義している。構成要素である図形は、2 行目の C の中心座標 (mid) と T の中心座標が同じであるという制約条件 (”C.mid == T.mid”) を満たさなければならない。また、4 行目から 7 行目において、図形単語 arc は 4 つの属性 start、end、mid、label を持っていて、属性の値はそれぞれ矢印の始点座標 (start)、矢印の終点座標 (end)、矢印の中心座標 (mid)、テキストの文字列 (label) であることを定義している。

- 2 つ目の生成規則は図 2.3 に示す開始状態への遷移を表わす開始線 (startArc) の定義である。1 行目で定義されているように sratArc は矢印 1 つにより構成されている。しかし、1 つ目の生成規則にあった arc と startArc の定義は類似しており、その違いは矢印の中心テキストがあるかどうかだけで

```
1: R:arc ::= A:arrow,T:text where (
2:     A.mid == T.mid
3:  ){
4:     R.start = A.start;
5:     R.end = A.end;
6:     R.mid = A.mid;
7:     R.label = T.label;
8:  }{}
```

図 2.2: 状態遷移図の定義 (遷移線)

```
1: S:startArc() ::= A:arrow where (
2:     not exist R:text where (R.mid == A.mid)
3:  ){
4:     S.start = A.start;
5:     S.end = A.end;
6:     S.mid = A.mid;
7:  }{}
```

図 2.3: 状態遷移図の定義 (開始線)

ある。そのため、arc として還元されるべき矢印が startArc に還元されてしまうというあいまいさが存在する。そこで、startArc の定義では矢印の中心にテキストがない場合にのみ生成規則が適用されるように、not exist 構成要素を用いている。2 行目が not exist 構成要素とネガティブ制約条件の定義であり、矢印 A のと中心座標が一致するテキスト R が存在する場合には、この生成規則は適用されないことを定義している。4 行目から 6 行目では 1 つ目の生成規則と同じく、図形単語 startArc の属性の定義を定義している。

- 図 2.4 に示す次の 3 つの生成規則は全て状態 (state) の定義である。このように同じ図形単語として定義することで、終了状態、開始状態、通常の状態の 3 つの状態を、状態という同じ図形単語として扱うことが可能になる。1 行目から 10 行目が終了状態の定義である。終了状態は 2 つの円 (circle) と 1 つのテキストで構成される。構成要素は 2 行目から 4 行目で定義されている制約条件を満たす必要がある。つまり、2 重の円の中心に

テキストがあるものが終了状態である。12 行目から 21 行目が開始状態の定義である。開始状態は 1 つの円 (circle) と 1 つのテキスト、1 つの開始線で構成される。14 行目の関数 OnCircle は開始線の終点の座標が円の周上に存在する場合に成立する関数である。開始状態と終了状態を区別するために、15 行目において not exist 構成要素として円を指定し、円が 2 重に存在する場合には、開始状態の生成規則が適用されないことを定義している。23 行目から 32 行目が通常の状態の定義である。通常の状態は 1 つの円 (circle) と 1 つのテキストで構成される。開始状態や終了状態と区別するために、24 行目と 25 行目において not exist 構成要素として円と開始線を指定し、円が 2 重に存在する場合や、開始線が円周上に存在する場合には通常の状態の生成規則が適用されないことを定義している。

- 図 2.5 に示す 2 つの生成規則は状態の遷移 (transition) の定義である。状態の遷移の生成規則は 2 種類あり、1 つ目はある状態 (S1) から別の状態 (S2) への遷移、2 つ目はある状態 (S) から同じ状態への遷移を表す生成規則である。遷移は構成要素として矢印が 1 つ、状態が 2 つで構成されている。ここで、状態を normal 構成要素として定義してしまうと、生成規則が適用された時に状態が遷移を表す図形単語へ還元されてしまうため、他の遷移の生成規則を適用する際の構成要素として使用することができない。そこで生成規則の 2 行目のように、遷移における 2 つの状態は exist 構成要素として定義している。これにより、3 行目、4 行目で定義されている制約条件を満たす矢印と状態が存在した場合に、この生成規則が適用され新たに状態を表す図形単語のトークンが生成された後でも、exist 構成要素として使用されたトークンは他の図形単語の構成要素になることができる。

- 図 2.6 に示す 2 つの生成規則は状態と遷移に還元されたトークンを収集するための生成規則である。all の構成要素を用いることで、制約条件を満たす全てのトークンを指定し、1 つのトークンとして扱うことができる。1 行目では制約条件は常に真であるので、state に還元されたすべてのトークンが収集される。収集されたトークンは 2 行目の属性定義で属性 set に保持される。

```
 1: S:state ::= C1:circle,C2:circle,T:text where (
 2:    C1.mid == C2.mid &&
 3:    C1.mid == T.mid &&
 4:    C1.radius <= C2.radius
 5: ){
 6:    S.mid = C1.mid;
 7:    S.radius = C2.radius;
 8:    S.label = T.label;
 9:    S.kind = "final";
10: }{}
11:
12: S:state ::= C:circle,T:text,A:startArc where (
13:    T.mid == C.mid &&
14:    OnCircle(A.end,C.mid,C.radius) &&
15:    not exist M:circle where (M.mid == C.mid)
16: ){
17:    S.mid = C.mid;
18:    S.radius = C.radius;
19:    S.label = T.label;
20:    S.kind = "start";
21: }{}
21:
23: S:state ::= C:circle,T:text where (
24:    not exist M:circle where (M.mid == C.mid) &&
25:    not exist A:startArc where (OnCircle(A.end,C.mid,C.radius))&&
26:    T.mid == C.mid
27: ){
28:    S.mid = C.mid;
29:   S.radius = C.radius;
30:   S.label = T.label;
31:   S.kind = "normal";
32: }{}
```

図 2.4: 状態遷移図の定義 (状態)

```
 1: T:transition ::= A:arc where (
 2:    exist S1:state,S2:state where (
 3:      OnCircle(A.start,S1.mid,S1.radius) &&
 4:      OnCircle(A.end,S2.mid,S2.radius))
 5: ){
 6:    T.start = S1.label;
 7:    T.tran = A.label;
 8:    T.end = S2.label;
 9: }{}
10:
11: T:transition ::= A:arc where (
12:    exist S:state where (
13:      OnCircle(A.start,S.mid,S.radius) &&
14:      OnCircle(A.end,S.mid,S.radius))
15: ){
16:    T.start = S.label;
17:    T.tran = A.label;
18:    T.end = S.label;
19: }{}
```

図 2.5: 状態遷移図の定義 (状態遷移)

```
1: SS:states ::= all S:state where (true) {
2:    SS.set.Add(S);
3: }{}
4:
5: TS:transitions ::= all T:transition where (true) {
6:    TS.set.Add(T);
7: }{}
```

図 2.6: 状態遷移図の定義 (トークンの収集)

```
1: F:std ::= SS:states,TS:transitions where (
2:    exist S:state where (S.kind = "start")
3: ){
4:    F.ss = SS;
5:    F.ts = TS;
6: }{}
```

図 2.7: 状態遷移図の定義 (状態遷移図全体)

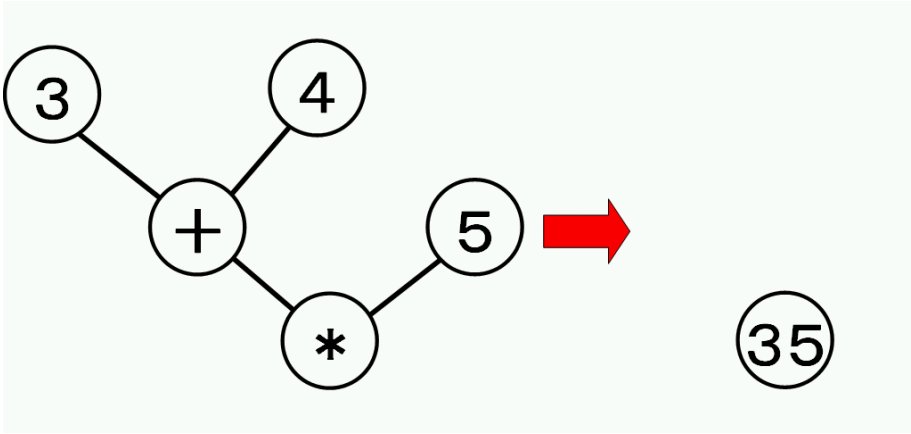

図 2.8: 計算の木

- 最後に図 2.7 に示す生成規則で状態遷移図全体を定義している。 状態遷移図は state と transitions により構成されている。つまり、すべての状態のトークンとすべての遷移のトークンが状態遷移図の構成要素となる。2 行目において、exist の構成要素として state を用いている。これは、状態遷移図は開始状態を必ず含むことを定義している。

2.2.2　計算の木

計算の木の図式を図 2.8 に示す。計算の木では、木の葉にあたる部分の値を木の節にあたる部分の演算子を用いて計算を行う。図 2.8 の場合、$(3+4)*7$ という計算が行われ、結果の 35 という葉だけが残る。

計算の木の拡張 CMG 記述を図 2.9 に示す。計算の木は 2 つの生成規則により定義されている。node は再帰的に定義されているため、計算の木では木の葉が

```
 1: N:node ::= C:circle, T:text where(
 2:    not exist L:Line where ( C.mid == L.mid)&&
 3:    C.mid == T.mid
 4: ){
 5:    N.mid := C.mid
 6:    N.left := C.lu.x
 7:    N.right := C.rl.x
 8:    N.value := T.label
 9: }{}
10:
11: N:node ::= C:circle,T:text where(
12:    exist L1:line,L2:line,N1:node,N2:node where(
13:      L1.start == C.mid &&
14:      L1.end == N1.mid &&
15:      L2.start == C.mid &&
16:      L2.end == N2.mid &&
17:      N1.right < N2.left ) &&
18:    C.mid == T.mid
19: ){
20:    N.left := C.left
21:    N.right := C.right
22:    N.mid := C.mid
23:    N.value := eval(N1.val,t.label,N2.val)
24: }{
25:    delete {N1 N2 L1 L2}
26:    alter T.label N.value
27: }
```

図 2.9: 計算の木の拡張 CMG 記述

node として認識され、node と節の演算子が再帰的に node として認識される。

図 2.9 の 1 行目から 9 行目では計算の木の葉の部分を図形単語 node として定義している。node は円 (circle) とテキスト (text) により構成されている。2 行目でネガティブ制約条件が定義されているので、条件を満たす直線 (line) が存在する場合にはこの生成規則は適用されず、node への還元は行われない。

図 2.9 の 11 行目から 27 行目では計算の木の節の部分と node の組みを図形単語 node として定義している。node は円 (circle) とテキスト (text) により構成されている。1 つ目の生成規則と区別するために、12 行目で exist の構成要素として node が 2 つと直線 2 つが定義されている。つまり、13 行目から 17 行目で定義されている制約条件を満たす node と直線が存在する場合にのみ、この生成規則が適用され円とテキストが node へと還元される。23 行目にある eval() 関数は引数に指定された 2 つの葉の属性値 (value) を節が持つつ演算子を用いて計算を行い計算の結果を返す関数である。25 行目と 26 行目が拡張 CMG の特徴であるアクションを定義している部分である。この生成規則では 2 つのアクションを定義している。まず、25 行目では delete アクションを用いて、この生成規則が適用された時にノードを 2 つ、直線を 2 つ削除することを定義している。26 行目では alter アクションを用いて、テキストの文字列 (label) を node の属性値 (value) に変更することを定義している。

2.3　空間解析器生成系

図形言語の図形間の規則を図形文法を用いて記述することで、図形文法記述からビジュアルシステムのための空間解析器を自動的に生成するシステムが空間解析器生成系である。空間解析器としては SPARGEN[Gol93]、VLCC[Cos95, Cos99]、Penguins[Cho95a, Cho98], 恵比寿 [Bab98c, Bab98a, Bab98b] などが提案されている。

SPARGEN は Picture Layout Grammars を用いて記述された図形言語の定義から図形言語のための空間解析器を自動的に生成するシステムである。図形エディタなどのビジュアルシステムのフロントエンドに生成された空間解析器を組み込むことで、図形言語の解析を行うビジュアルシステムを作成することが

できる。

VLCC はビジュアルプログラミングシステムを自動的に生成するシステムである。 図形言語における図形間の空間的な関係を Positional Grammar で記述し VLCC に与えることで、VLCC は図形言語の空間解析器を持ったビジュアルプログラミングシステムのソースコードを自動的に生成する。ユーザは生成されたソースコードをコンパイルすることで、ビジュアルプログラミングシステムを利用することが可能になる。

Penguins は CMG で記述された図形言語の文法を与えると、自動的にインクリメンタルな解析器を生成するシステムである。生成された解析器は図形エディタを持っている。Penguins では QOCA[Bor97] という制約解消系を利用し、解析の際に図形間の位置関係を保存し、図式のレイアウトを行っている。

恵比寿は空間解析器生成系と図形エディタを備えたシステムであり、拡張 CMG で記述された図形言語の文法を与えることで、与えられた図形言語に対応することができるシステムである。恵比寿では図形文法として拡張 CMG を用いている。また、恵比寿では Penguins と同様に、制約解消系 SkyBlue[San94a, San94b] を利用して図式のレイアウトを行っている。

恵比寿にレイアウト制約を導入し、レイアウトの機能を拡張したシステム Rainbow[Jou00b, Jou00a, jou01] や、ジェスチャ認識システム SATIN[Hon00] を用いて、手書き入力のストローク情報を解析することで、空間解析器で手書き入力を扱えるようにした Handragen[Yam03, Shi03]、入力された図形から図形文法の定義を推測を行う VIC[Fuj99, Fuj01] などが提案された。

第3章　直接操作を用いた文法編集システム:GIGA

本章では、提案手法である図形を用いた図形文法の定義手法について述べる。まず、従来の定義方法における問題点についてあげる。次に、問題を解決するための提案する手法について述べ、作成したシステムである GIGA について述べる。

3.1　従来の図形文法の定義における問題点

拡張 CMG を用いて様々なビジュアルシステムを定義することが可能だが、文献 [Fuj01]、文献 [Bab99]、文献 [Cho03] では例として、下記のようなビジュアルシステムを記述している。

(1) 2.2.1 節で取りあげた状態遷移図を解析するビジュアルシステム

(2) 2.2.2 節で取りあげた計算の木を編集し、実際に計算を行うビジュアルシステム

(3) スタック構造を図として編集し、そのスタックの図に対して操作することができるビジュアルシステム (図 3.1)

(4) 長方形と直線からなるリスト構造を解析するビジュアルシステム (図 3.2)

(5) スクロールバーやボタンなどの GUI 部品を組み合わせたインタフェースを編集できるインタフェースビルダ (図 3.3)

(6) パイプの機能をもったシェルを視覚化したビジュアルシェル (図 3.4)

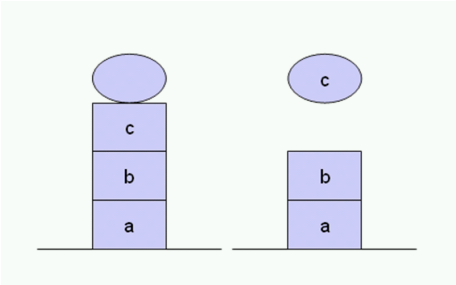

図 3.1: スタック

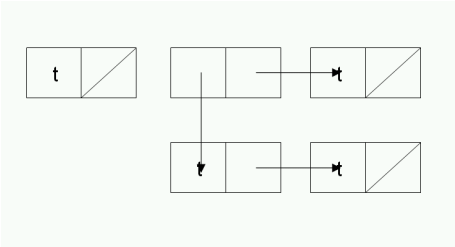

図 3.2: リスト

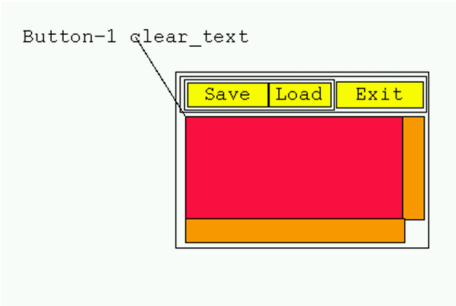

図 3.3: GUI

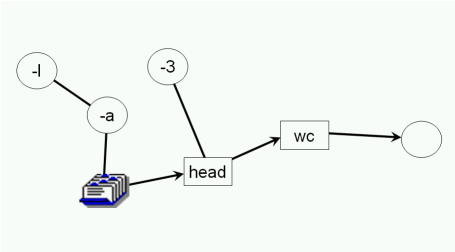

図 3.4: VSH

(7) ファイルやコマンドを図的に表現したアイコンを二つ以上重ねあわせることによって動作を指定することができるビジュアルシステム HI-VISUAL [Hir91] のサブセット (図 3.5)

(8) 書き換えのルールを図形を用いて定義し、マウスのクリックなどのユーザの操作により図形の書き換えを行うビジュアルシステム VISPATCH[Har97] のサブセット (図 3.6)

これらの拡張 CMG 記述について、含まれる生成規則の数や制約の数などを調べた結果を表 3.1 に示す。

空間解析器生成系に与える図形文法の記述は、図形言語を構成する図形 (長方形、線分、画像など) 同士が満たすべき制約 (含む、接する、中央に存在する等の位置関係や色など) などの 2 次元の構造を記述した規則の集合である。

そのような 2 次元の構造をテキストのみを用いて記述する場合、以下に挙げる問題が生じる。

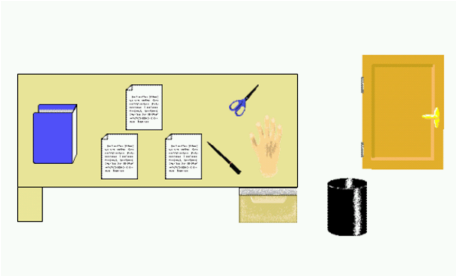

図 3.5: HI-VISUAL

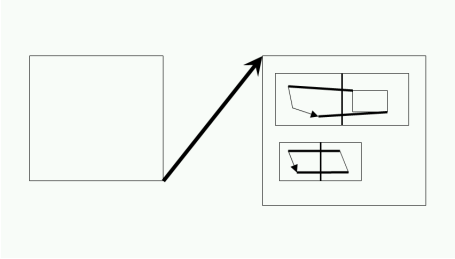

図 3.6: VISPATCH

表 3.1: 拡張 CMG で記述されたビジュアルシステムの例

	(1)	(2)	(3)	(4)	(5)	(6)	(7)	(8)	合計
総生成規則数	10	2	4	4	14	11	15	24	84
総制約数	16	8	5	16	79	53	43	129	349
座標の一致	8	7	1	16	2	35		6	75
x 座標 (or y 座標) の一致			4		22			7	33
x 座標 (or y 座標) の比較		1			40		28	94	163
属性値同士の一致									0
属性値同士の比較	1				5		7		13
属性値と定数値の一致	1				10	18	8	12	49
属性値と定数値の比較								10	10
onCircle	6								6
総属性数	29	8	18	11	101	29	50	100	349
1 つの属性値を継承	24	7	14	9	83	17	48	84	286
その他	5		4	2	18	12	2	16	63
総アクション数	0	7	6	3	1	1	13	5	36
create			1					1	2
alter		1					1		2
delete		4	1				3	2	10
その他		2	4	3	1	1	9	2	22

1. 制約の把握・定義が困難である
   生成規則が適用される条件のひとつである制約は、座標間の関係を書き下すための複数の式から成ることが多く、これらを読み、理解するには時間がかかる。2 章で用いた図形言語の例である計算の木でも、以下に示す文法記述から制約条件の把握するためには、6 つの式を読み、各構成要素間の関係を理解しなければならない。

```
N:node ::= C:circle,T:text where(
   exist L1:line,L2:line,N1:node,N2:node where(
     L1.start == C.mid &&
     L1.end == N1.mid &&
     L2.start == C.mid &&
     L2.end == N2.mid &&
     N1.right < N2.left ) &&
   C.mid == T.mid{
){
   ...
}{
   ...
}
```

   さらに、文法を定義するためには、制約で使用される属性を把握し、適用条件を正確に書き下す必要がある。

2. 属性の記述が煩雑である
   属性の値には構成要素の図形の座標や大きさがそのまま使われることが多い。例えば計算の木の 1 つ目の生成規則では right、left、mid という属性が、構成要素の属性をそのまま代入することによって定義されている。このように、そのまま使われる属性について、そのひとつひとつを記述し定義を行うことは煩雑な作業となっている。

3. アクションの結果が把握しにくい

アクションが実行された結果、図全体がどのような形状に変化するのかは、テキストを用いて記述されたアクションでは把握しにくい。

本研究では、図形文法の定義の困難さを解決するため、以下のアプローチを採用した。

- 図形言語の図式表現を編集するという例示操作により図形文法の定義を行い、成立している制約についてはシステムにより画面上に制約をインタラクティブに図示することにより、現在定義を行っている制約を直感的に理解できるようにする。
- 関係する図形を明示的にユーザが関連付けることによって、不必要な制約の推論を抑えるようにする。
- 入力した図式表現を隣接領域にコピーし、そのコピーに対して行った編集操作をアクションとして定義する。

3.2　GIGA システム

前節で述べた提案手法を用いることにより、より直感的に図形文法を定義するシステムとして GIGA を設計し実装を行った [Kam02, Kam03]。GIGA の定義インタフェースを図 3.7 に示す。

マルチプラットフォームで実行が行えるよう、実装には Java 言語（J2SE v1.4.1）を用いた。システムのコードの量はおよそ 5000 行である。システムは図形の編集を行う部分、入力された図形から生成規則の推論を行う部分、推論した生成規則をテキスト形式でファイルへ出力する部分に分かれている。

以降では、拡張 CMG の各要素を GIGA を用いてどのように定義するかについて述べる。

3.2.1　構成要素の定義

構成要素となる基本図形もしくは図形単語を定義するために、GIGA の下部にあるボタンから目的の図形を選択し、図形を画面上に描画する。基本図形と

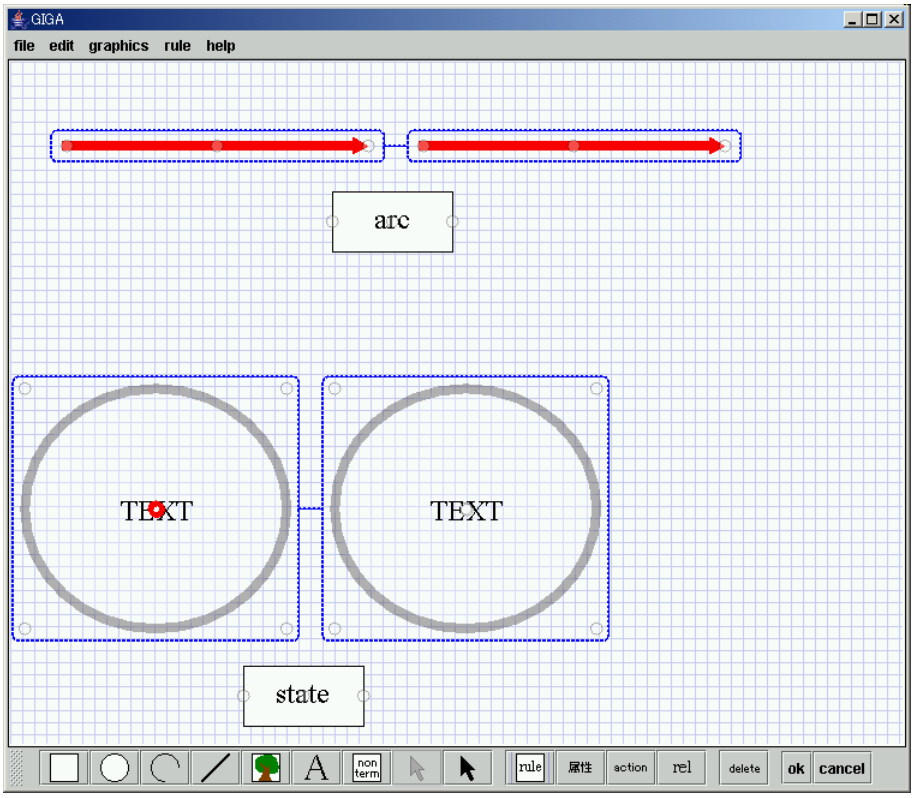

図 3.7: GIGA の図形文法定義インタフェース

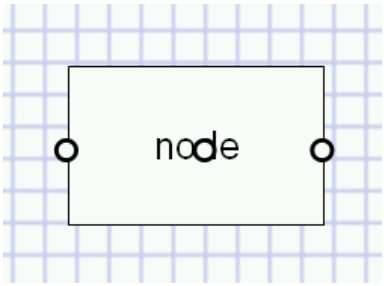

図 3.8: 図形単語の属性表示

して使用できる図形には、円 (circle) や長方形 (rectangle)、テキスト (text)、直線 (line)、画像 (image) がある。あらかじめ定義した図形単語を入力することもできる。なお、図形単語は図 3.8 に示すように中心に図形単語の名前の付いたボックスとして表示され、座標値を表す属性もあわせて表示される。例えば node という図形単語が 3 つの属性 (中心の座標値、左端の座標値、右端の座標値) を持つ場合には、図 3.8 に示すように、各座標に円が表示される。

3.2.2　制約の定義

制約の定義を行うには、構成要素の図形を制約を満たすように配置する。GIGA は配置された構成要素間の属性を比較し制約を自動生成する。GIGA では、自動生成された制約は図 3.9 に示すように制約を画面上で強調して表示することで、定義した制約が直感的に理解できるように手助けを行っている。

- 図形の座標が他の図形の座標と完全一致している場合には図 3.9 –(a) のように、一致している座標を表す属性を強調表示する。
- 図形の座標が他の図形の座標とと完全に一致してはいないが、X 座標 (もしくは Y 座標) が一致している場合には図 3.9–(b) のように、座標を表す属性同士を通る太い点線でガイドラインを表示する。
- 図形の大きさを表す属性が、他の図形の大きさの属性と完全に一致している場合には、図 3.9–(c) のように大きさの一致している部分を表す属性を強調表示する。

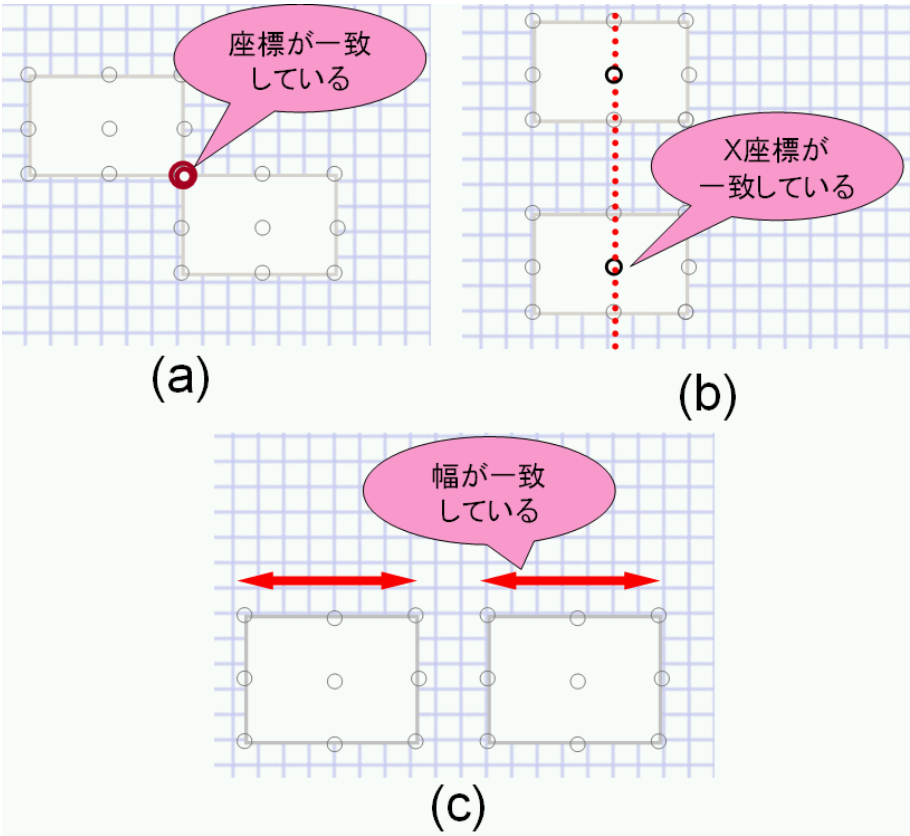

図 3.9: 制約の強調表示

また、入力されたすべての構成要素間で属性の比較を行うと、不必要な制約が多数生成されるという問題がある。そこで、GIGA では重ね合せによる図形の属性の関連付けと基本属性の属性指定により不必要な制約の推論を抑えている。

重ね合わせによる属性の関連付け

GIGA では制約を定義したい構成要素の属性があった場合に、その属性同士を重ね合わせることでユーザにより明示的に図形の属性の関連付けを行なわせ、関連付けされた属性間に成立する制約だけを推論することによって、不必要な制約の出力を抑えている。関連付けは、座標値を持つ属性であれば、図形を移動し別の図形と属性の座標を表わす円同士を重ね合わせることにより行うことができる。座標値を持つ属性以外 (図形の幅や色など) の属性については、属性を重ねることができないので、属性を持つ図形同士を重ねることで属性の関連付けが行う。一度関連付けを行うと、図形を移動し属性同士が離れても関連付けは保持される。

基本図形の属性指定

基本図形にはシステムにより様々な属性が用意されている。例えば長方形には座標値や幅、高さ、色が属性として用意されている。これらの属性は生成規則を定義するために用いられるが、基本図形が持つすべての属性が定義に使用されることは少ない。「長方形であればどのような色や幅を持っていても解析を行う」という規則のように、生成規則の定義に必要な属性だけが使用される。GIGA では制約を推論する際に、入力された図形の属性をすべて使用せずにあらかじめユーザに指定された属性のみを用いて制約を出力する。例えば何も属性が選択されていない状態では、長方形は図 3.10–a に示すように半透明で表示されている。

長方形の上に表示されている 9 つの円は長方形の 4 つの頂点と各辺の中心、対角線の交点の座標を表している。ユーザはこの中から生成規則を定義するために必要な属性のみを選択する。例えば長方形の中心の座標を生成規則で使用したい場合には、中心にある円をマウスでクリックする。これにより図 3.10–b

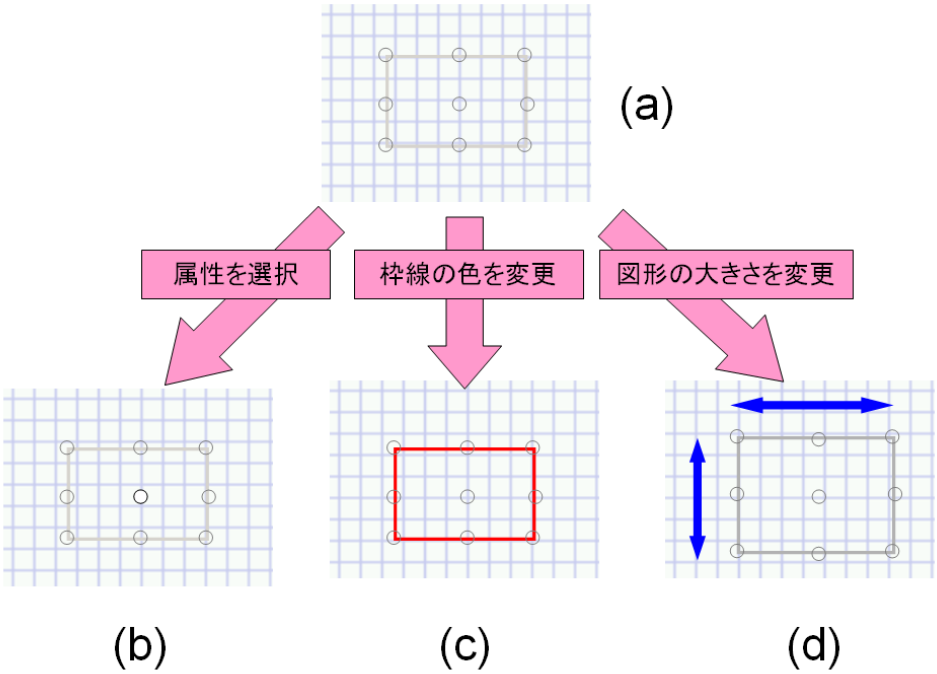

図 3.10: 基本図形の属性表示

に示すように、選択された円 (属性) が半透明ではなくなり、この属性を制約の定義を行う際に使用することができるようになる。長方形の枠線の色を生成規則で使用したい場合には、長方形の枠線の色を変更する。枠線の色を変更すると図 3.10–c に示すように、枠線の色が実際の色で表示されるため、枠線の色を属性として使用することを確認することができる。長方形の幅や高さを生成規則で使用したい場合には、長方形の大きさを変更する。大きさを変更すると図 3.10–d に示すように、変更された部分が属性として使用されることを矢印を用いて表示される。

3.2.3　生成規則の定義

構成要素となる図形を入力し、構成要素間の制約まで定義が終わった時点で、ツールバーの Rule ボタンを押し、入力した図形を全て選択する (図 3.11–(b))。これにより、選択した図形が生成規則の構成要素であることが定義される。同時に、定義画面上に図形単語を表すボックスが生成される (図 3.11–(c))。ボックスの中心に書かれた文字列が定義した生成規則から還元される図形単語名であり、GIGA により任意の名前が自動的に付けられる。この名前は変更することが可能であり、複数の生成規則に同じ名前を付けることで、それらの生成規則から還元される図形単語を同じものとして扱うことが可能になる。

また、構成要素に図形単語を含む生成規則を定義する際にはこのボックスを使用することで、図形単語を構成要素として定義することができる。

3.2.4　アクションの定義

拡張 CMG のアクションとは、解析が行われた時に実行される動作の集合である。アクションの中で図形の書き換えに関する規則については、入力した図式表現の他にアクションが実行された後の図式表現を併せて編集することにより、アクションの定義を行う。GIGA では構成要素、制約についての定義を行った後、画面下にあるアクションと書かれたボタンを押すことで、作成した図形がそのまま複製される。この図を編集し、アクションが実行された後の図を作

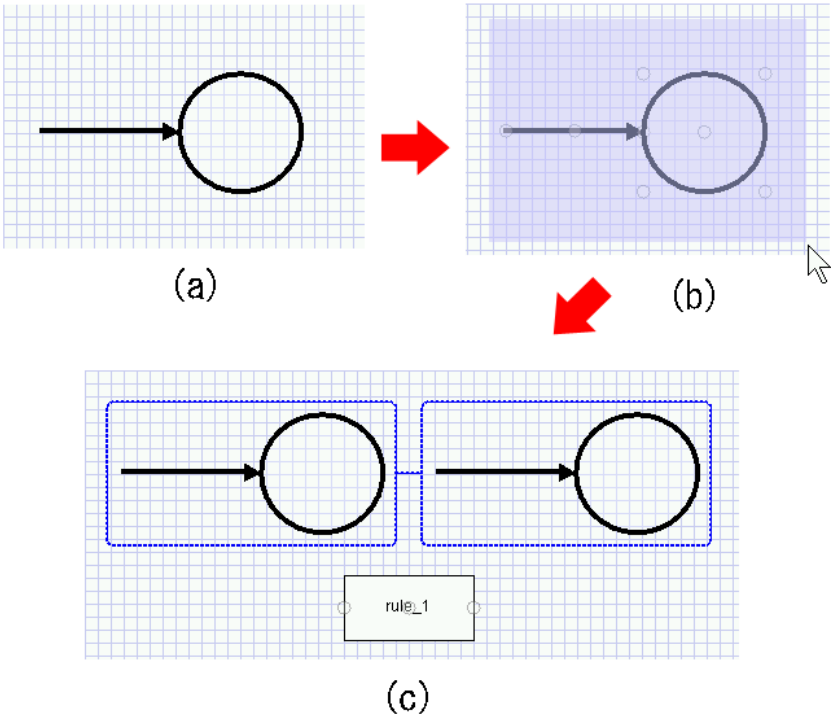

図 3.11: 生成規則の定義

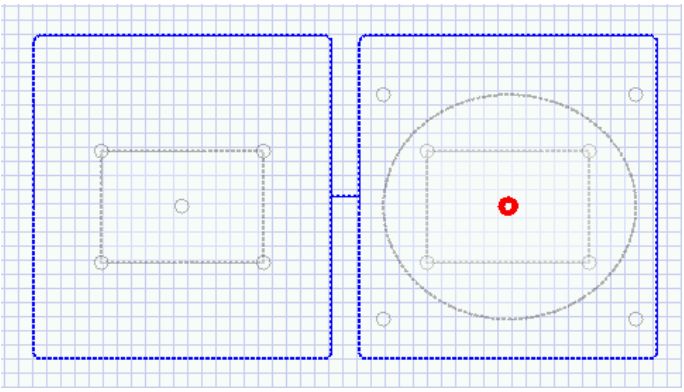

図 3.12: create アクションの定義

成することで、アクションの定義を行う。GIGA ではこれらの 2 つの図の違いと図形に対する操作履歴から拡張 CMG のアクションを推論し生成する。GIGA を用いて拡張 CMG のアクションを定義するためには、以下の操作を行う。

- create アクションを定義するには、定義インタフェースの右側に追加したい図形を描く。例えば、長方形が 1 つ解析された後に中心が同じ円を作成したい場合には、定義インタフェースの右側にある長方形と中心が同じ円を描くことにより、円を作成する create アクションを定義できる（図 3.12）。create アクションにより生成される図形の位置座標の属性値については、構成要素となる図形との相対的な座標を用いて決定する。座標の指定方法は制約の定義と同様に、図形の座標の属性を表す円同士を重ねることで指定する。

- delete アクションを定義するには、定義インタフェースの右側で対象の図形を削除する。例えば、2 つの長方形が重なった時に内側の長方形を削除したい場合には、定義インタフェースの右側で内側にある長方形を削除することによって、長方形を削除する delete アクションが定義できる（図 3.13）。

- alter アクションを定義するには、定義インタフェースの右側で属性値を変更したい図形を修正する。例えば、解析された長方形の色を黒くしたい場合には、定義インタフェースの右側でその長方形の色を実際に変更す

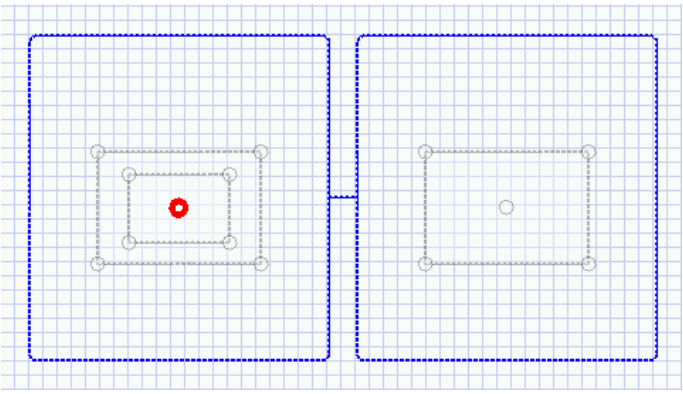

図 3.13: delete アクションの定義

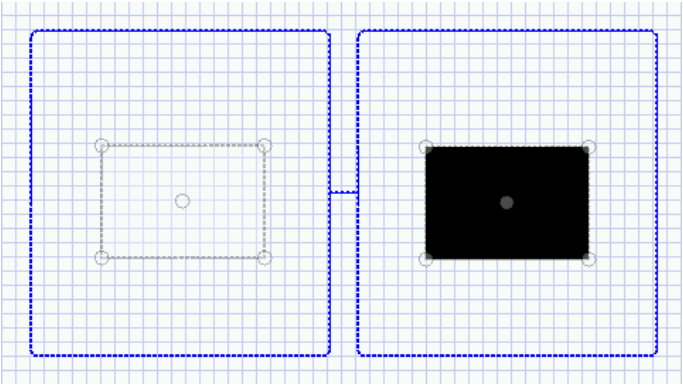

図 3.14: alter アクションの定義

ることにより、長方形の色を変更する alter アクションが定義できる（図 3.14）。

3.2.5　属性の定義

図形単語の属性として定義される値は構成要素となる図形の属性の値をそのまま用いて定義されることが多い。そこで、構成要素の属性値を図形単語の属性値として定義する場合は、構成要素の属性を直接指定することで定義を行う。具体的には、図 3.15 に示すように、構成要素となる図形の属性を表示している円をクリックすることでその座標が図形単語の属性として定義し、その円の色を黄色に変更することで図形単語の属性であることの表示を行う。属性名は用

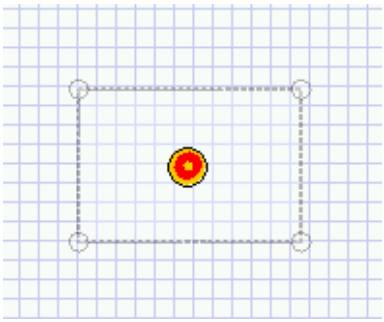

図 3.15: 属性の定義

いられた構成要素の属性名をそのまま使用する。

座標値以外の値を持つ属性を定義する場合には、まず属性を入力するためのボックスを入力する。次に、ボックスの中に属性の計算式を入力することで座標値以外の値を持つ属性を定義することができる。

3.2.6　ビジュアルシステムの定義例

GIGA を用いてビジュアルシステムを定義する手順を解説するために、状態遷移図の編集を行うビジュアルシステムを用いる。状態遷移図の図式の一例を図 3.16 に示す。また、状態遷移図の規則は文献 [Cho03] に示されている規則に変更を加えた。拡張 CMG 記述を図 3.17 に示す。

以降では GIGA を用いて各生成規則を定義する手順を解説する。

状態遷移図の生成規則 1 の定義　生成規則 1 は、状態の遷移を表わす線 (arc) を定義している。遷移を表わす線 (arc) を定義する手順を図 3.18 に示す。遷移を表わす線は直線とテキストで構成されていて、それぞれの中心が一致しているものとする。これを定義するために構成要素として直線とテキストを 1 つ画面上に描画する (図 3.18–(a))。描画した各図形の中心にある円をクリックし、中心の座標値を属性として使用することを指定する (図 3.18–(b))。次に 2 つの図形の中心を重ねる。先ほど指定した属性同士を重ねることで、GIGA が制約を推論し、座標が一致していることを図 3.18–(c) のように強調表示してくれる。制約を定義した構成要素をまとめ

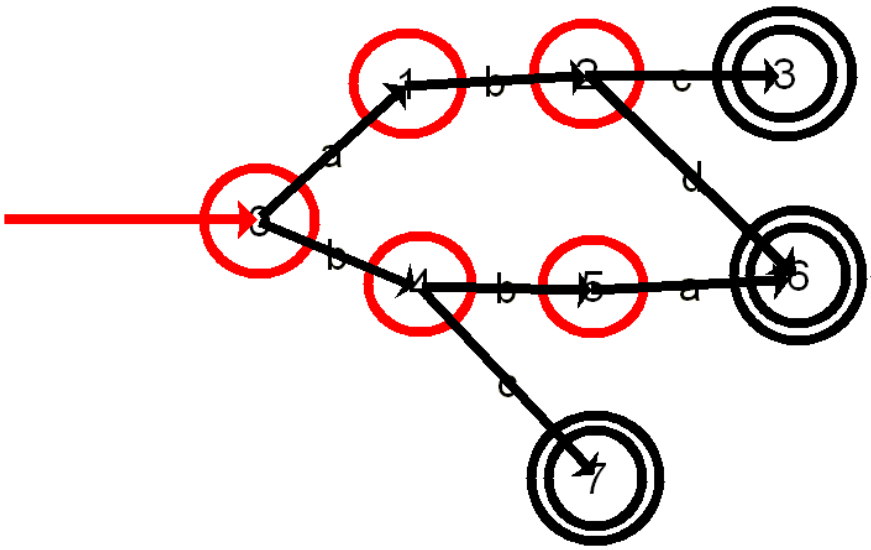

図 3.16: 状態遷移図

て選択し、生成規則を定義すると、図形単語を表す図形が生成される (図 3.18–(d))。生成された図形のテキストを変更することで図形単語の名前を定義する (図 3.18–(e))。最後に直線の始点、終点、中心点を属性として定義するために右側の画面で直線の各属性を指定し、図形単語の属性として定義する (図 3.18–(f))。

状態遷移図の生成規則 2 の定義 生成規則 2 では、状態遷移の始めに開始状態へと遷移する線 (start_arc) を、通常の遷移に用いられる直線と区別するために赤色の直線として定義を行っている。開始線 (start_arc) を定義する手順を図 3.19 に示す。開始線は直線のみで構成されているものとする。これを定義するために構成要素として直線を 1 つ画面上に描画する (図 3.19–(a))。描画した直線の色を赤色に変更することで、直線の色が赤色であるという制約を定義する (図 3.19–(b))。構成要素をまとめて選択し、生成規則を定義する (図 3.19–(c))。図形単語の名前を定義する (図 3.19–(d))。最後に直線の終点を属性として定義するために、右側の画面で直線の終点を指定し属性として定義する (図 3.19–(f))。

```
// 生成規則 1
arc(point start, point end,
point mid) ::=
   L:line, T:text where(
   L.mid == T.mid
){
   start = L.start;
   end = L.end;
   mid = L.mid;
}

// 生成規則 2
start_arc(point end) ::=
   L:line where(
   L.linecolor == red
){
   end = L.end;
}

// 生成規則 3
state(pont mid) ::=
  C:circle,T:text where (
   C.linecolor == red &&
   T.mid == C.mid
){
   mid = C.mid;
}
```

```
// 生成規則 4
state(point mid)::=
  A:start_arc, C:circle,
  T:text where(
   C.linecolor == red &&
   A.end == C.mid &&
   T.mid == C.mid
){
   mid = C.mid;
}

// 生成規則 5
state(pont mid, string kind)
   ::= C1:circle, C2:circle,
       T:text where(
   C1.mid == C2.mid &&
   T.mid == C1.mid
){
   mid = C1.mid;
}
// 生成規則 6
transition ::=
  A:arc, exist S1:state,
  S2:state where(
   A.start == S1.mid &&
   A.end == S2.mid
){
}
```

図 3.17: 状態遷移図の拡張 CMG 記述

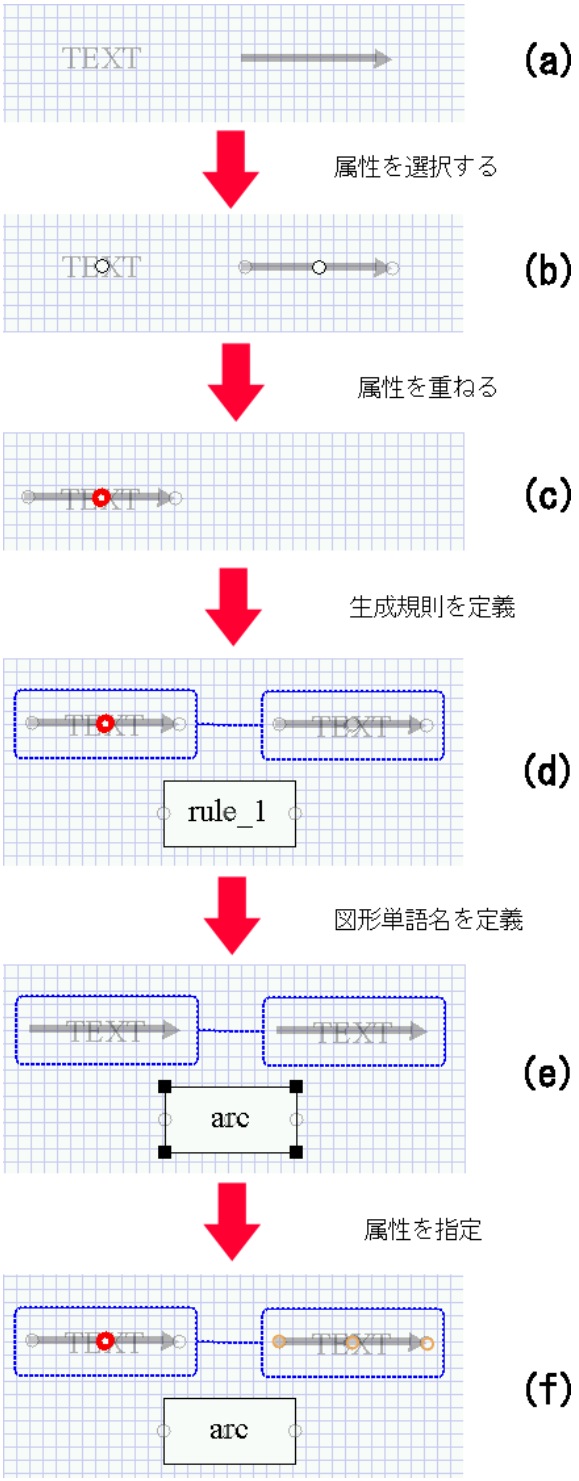

図 3.18: 遷移を表わす線 (arc) の定義

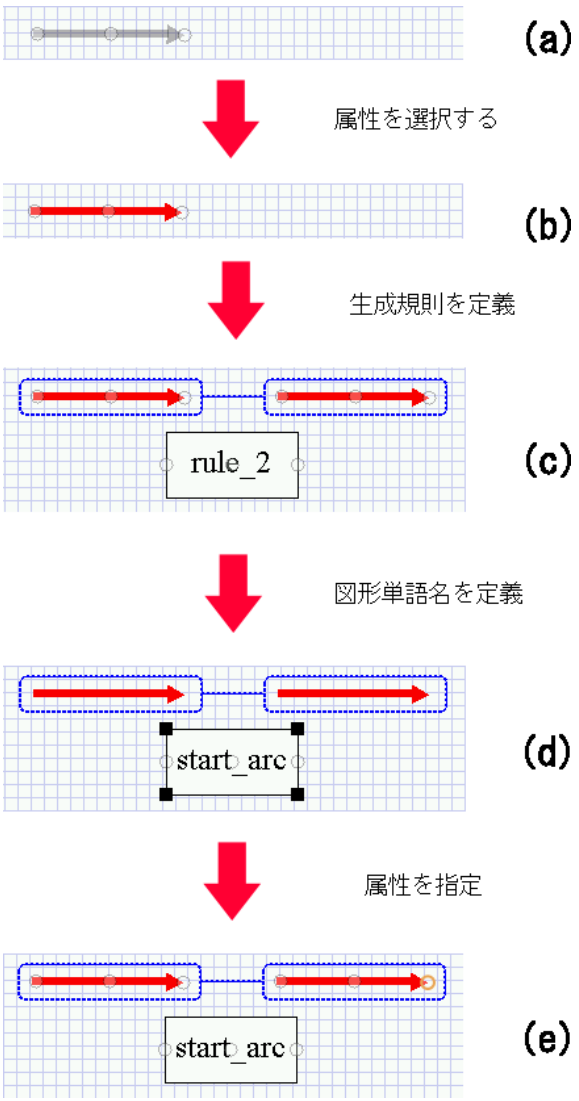

図 3.19: 開始線 (start_arc) の定義

状態遷移図の生成規則 3 の定義 生成規則 3 では、状態遷移図の各状態 (state) は、テキストと赤い円の組み合わせとして定義を行っている。状態遷移図の状態 (state) を定義する手順を図 3.20 に示す。構成要素としてテキストを 1 つと楕円を 1 つ画面上に描画する (図 3.20–(a))。状態における円の色は赤色であるという制約を定義するために、円の色を赤色に変更する。(図 3.20–(b))。描画した各図形の中心にある円をクリックし、中心の座標値を属性として使用することを指定する (図 3.20–(c))。次に円とテキストの中心を重ねる。先ほど指定した属性同士を重ねることで、GIGA が制約を推論し、座標が一致していることを図 3.20–(d) のように強調表示してくれる。次に構成要素をまとめて選択し、生成規則を定義する (図 3.20–(e))。図形単語の名前を定義する (図 3.20–(f))。最後に円の中心点を状態の属性として定義するために、右側の画面で円の中心点を指定し属性として定義する (図 3.20–(g))。

状態遷移図の生成規則 4 の定義 生成規則 4 では、状態遷移の開始を表わす開始状態 (state) は、開始線と状態の組み合わせとして定義を行っている。開始状態 (state) を定義する手順を図 3.21 に示す。まず、構成要素として状態と開始線を 1 つずつ画面上に描画する (図 3.21–(a))。状態と開始線の中心にある円をクリックし、中心点の座標値を属性として使用することを指定する (図 3.21–(b))。次に状態の中心点、開始線の終点を重ね、先ほど指定した属性同士を重ねることで、GIGA が制約を推論し、座標が一致していることを図 3.21–(c) のように強調表示してくれる。さらに構成要素をまとめて選択し、生成規則を定義する (図 3.21–(d))。図形単語の名前を定義する (図 3.21–(e))。最後に状態の中心点を状態の属性として定義するために、右側の画面で状態の中心点を指定し属性として定義する (図 3.21–(f))。

状態遷移図の生成規則 5 の定義 生成規則 5 では、状態遷移の終了を表わす終了状態 (state) は他の状態と区別するために、テキストと二重の円の組み合わせとして定義を行っている。状態遷移図の終了状態 (state) を定義する手順を図 3.22 に示す。構成要素として楕円を 2 つとテキストを 1 つ画面上に描画する (図 3.22–(a))。描画した各図形の中心にある円をクリックし、

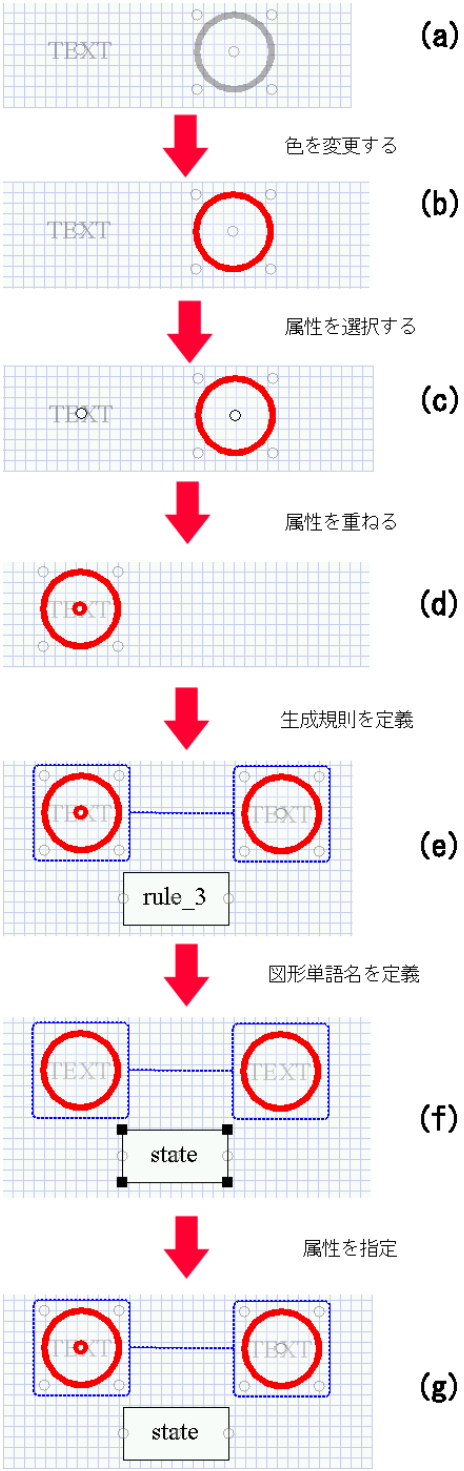

図 3.20: 状態 (state) の定義

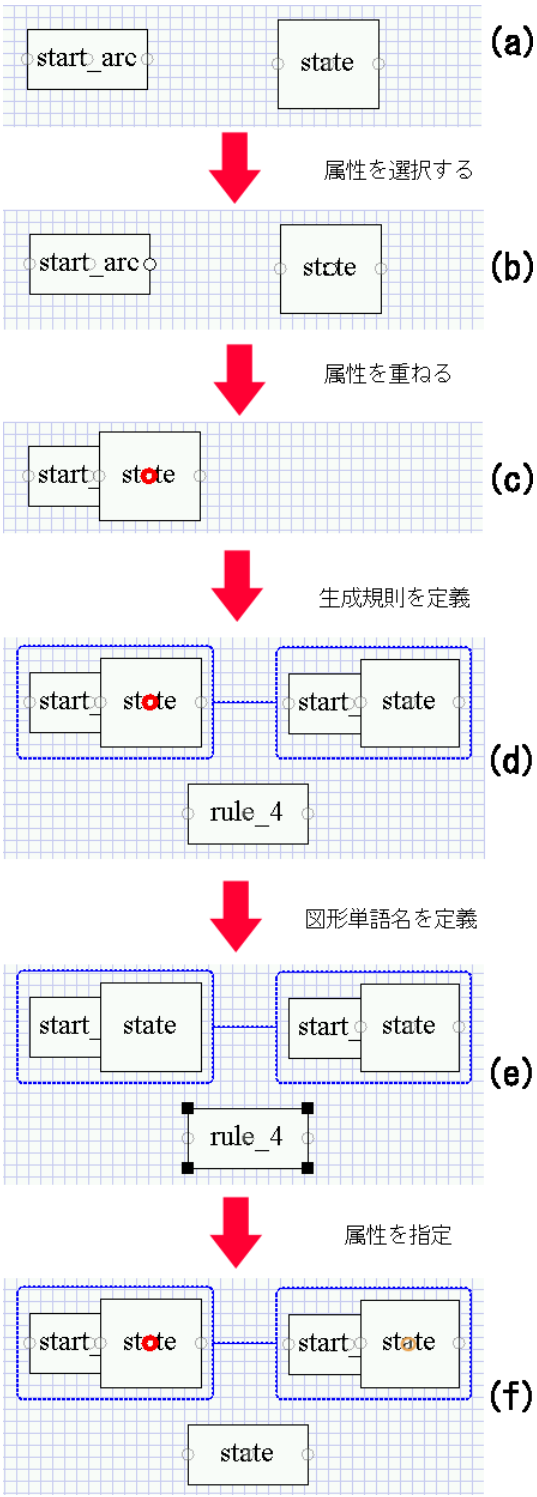

図 3.21: 開始状態 (state) の定義

中心点の座標値を属性として使用することを指定する (図 3.22–(b))。先ほど指定した 3 つの図形の属性同士を重ねることで、GIGA が制約を推論し、座標が一致していることを図 3.22–(c) のように強調表示してくれる。次に、構成要素をまとめて選択し、生成規則を定義する (図 3.22–(d))。図形単語の名前を定義する (図 3.22–(e))。最後に円の中心点を状態の属性として定義するために、右側の画面で円の中心点を指定し属性として定義する (図 3.22–(f))。

状態遷移図の生成規則 6 の定義 生成規則 6 では、状態の遷移 (transition) は 2 つの状態と遷移線の組み合わせとして定義を行っている。遷移 (transition) を定義する手順を図 3.23 に示す。構成要素として状態 (state) を 2 つと遷移線 (arc) を 1 つ画面上に描画する (図 3.23–(a))。状態は exist な構成要素として定義するために、メニューから 2 つの状態 (state) を exist に変更する。遷移線 (arc) の始点と終点、状態 (state) の中心点の座標を座標値を属性として使用することを指定する (図 3.23–(b))。遷移線 (arc) の始点と終点をそれぞれの状態 (state) の中心と一致させることで、座標が一致していることを図 3.23–(c) のように定義する。次に、構成要素をまとめて選択し、生成規則を定義する (図 3.23–(d))。図形単語の名前を定義する (図 3.23–(e))。

以上の手順により定義を行った 6 つの生成規則を図 3.24 に示す。生成規則の定義を終了した後、GIGA は定義した生成規則をテキストとしてファイルに出力する。このファイルを恵比寿などの空間解析器生成系に与えることで状態遷移図を解析するビジュアルシステムを実行することができる。

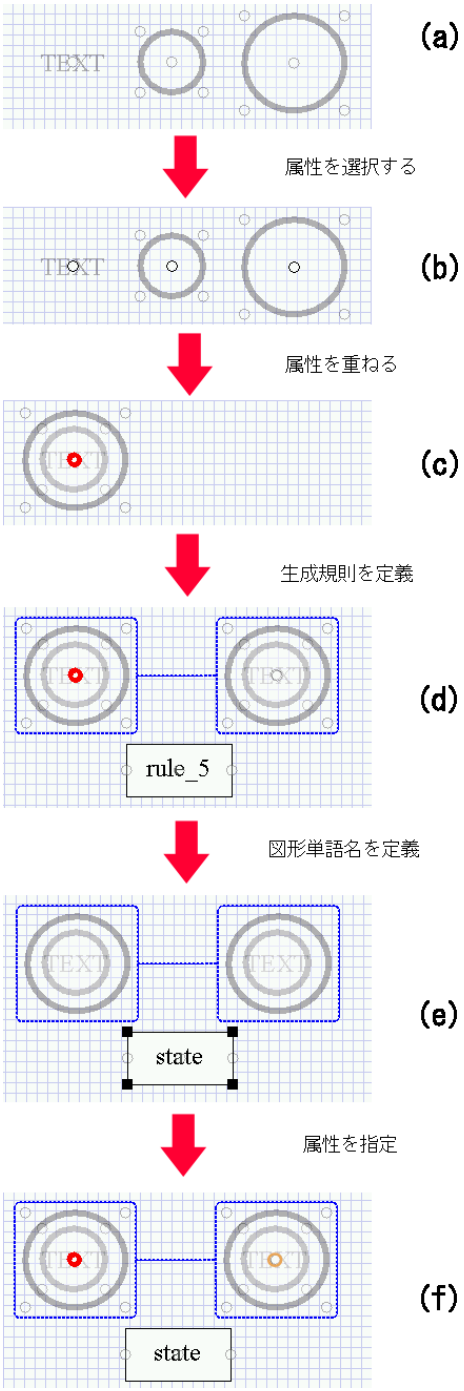

図 3.22: 終了状態 (state) の定義

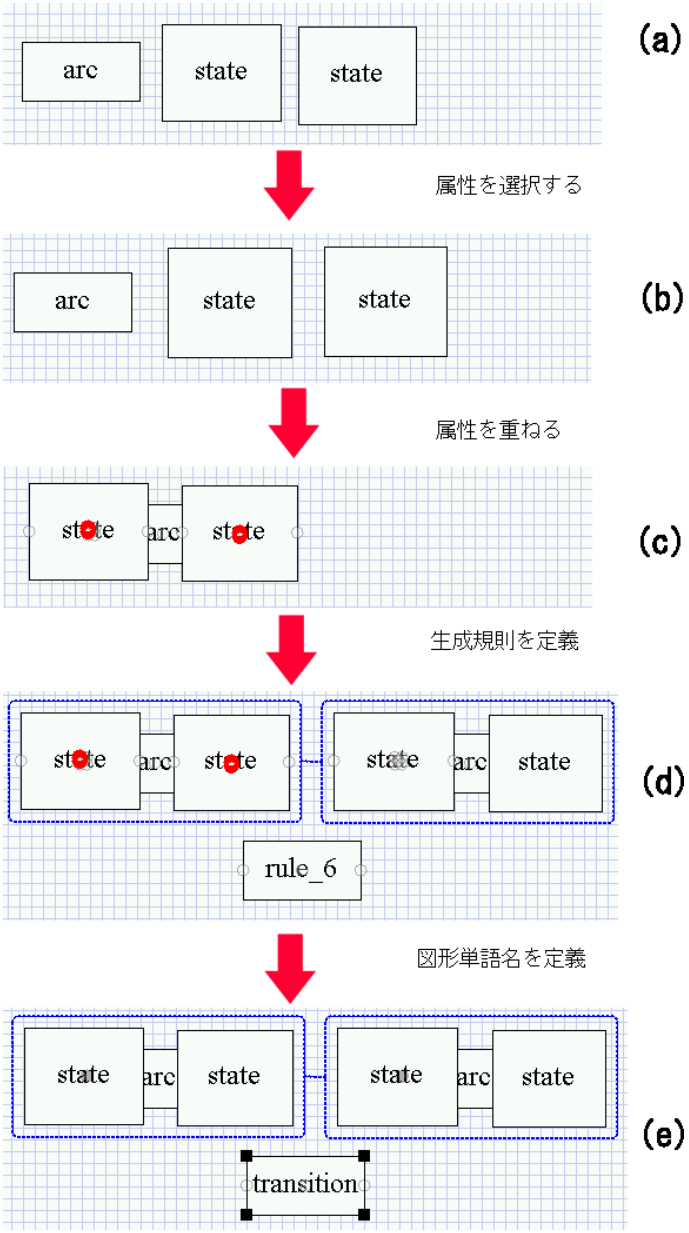

図 3.23: 遷移 (transition) の定義

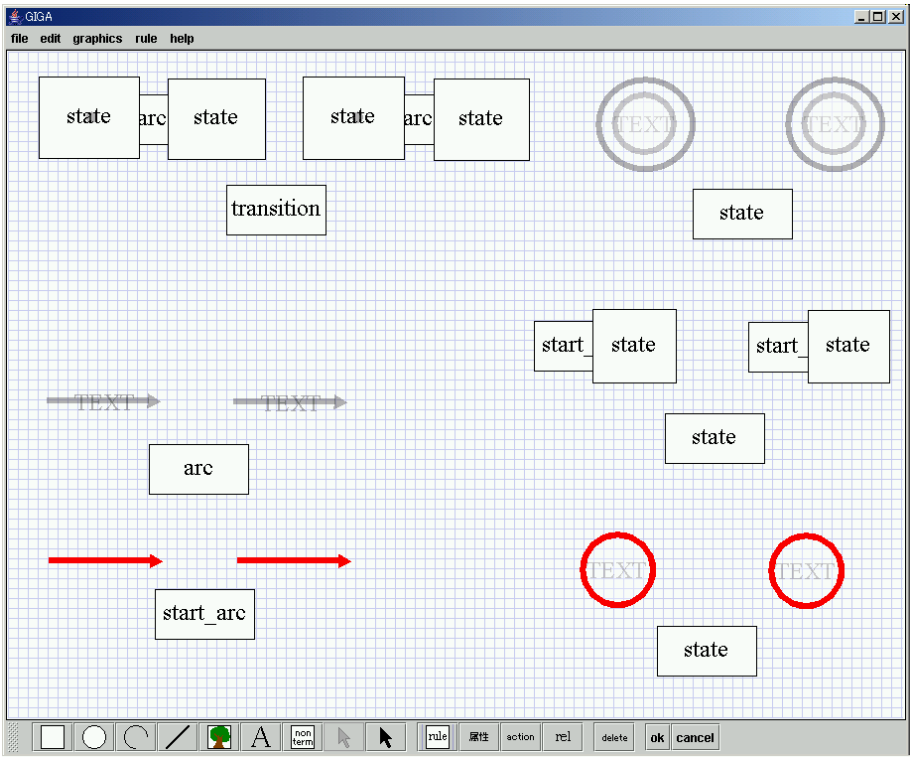

図 3.24: GIGA による状態遷移図の定義

3.3　関連研究

制約の定義と表示方法に関する関連研究として Briar[Gle94] と Pegasus[Iga98] がある。Briar は、ユーザの操作から図形間の制約を推論し、画面上に補助線を表示したり補助線上にマウスカーソルをスナッピングすることにより、図形描画の補助を行うシステムである。Pegasus は、ユーザの手書き入力から制約を自動的に抽出して整形を行ったり、複数の候補を同時に生成することによりユーザに希望するものを選択させる図形描画システムである。

これらの研究が図形描画のためのシステムであるのに対して、本システムは、図形文法の生成規則を入力・編集するためのシステムであるという違いがある。そのため、本システムでは制約を定義するだけではなく、文法の要素である構成要素、属性、アクションについても定義を行うことができる。また、Pegasus では制約の予測の種類を増やしていくと、予測結果として生成される候補の数が増え、画面上に図示される候補の数が増え過ぎてしまうという問題がある。これに対し、本システムでは図形の属性同士に関連付けを行い、候補の数が増え過ぎてしまうことを抑えている。

なお、GIGA が提供するインタフェースのうち、アクションの定義には、図形の書き換え前と書き換え後を表現する 2 つの定義インタフェースを提供している。このような図式表現を用いて書き換え規則を定義する研究として Visulan [Yam96] や VISPATCH[Har97]、KIDSIM[Cyp95] などがある。KIDSIM では物体を動かすことにより物体の移動規則を例示で定義することができる。移動前と移動後の状態の絵の組が変換規則であり、パターンにマッチする変換規則があればそれに従って画面が書き換えられる。Visulan では変化前と変化後の組により絵の変化を表し、その組を 1 つのルールとみなしてプログラムを構築することができる。画面の一部が変化前の絵にマッチしたらその部分を変化後の絵に書き換えを行うことでプログラムが実行される。VISPATCH はルールをもとに図形の書き換えを行うビジュアル言語である。マウスによるイベントが発生すると、入力された図形によって表される条件が成立しているルールを探す。もし条件が成立しているルールがあれば、そのルールで定義されている図形へと書き換えを行う。

書き換えが行われる前後の画面を使って文法を定義するという点ではこれらの研究と GIGA は同じである。しかし、構成要素と図形の書き換え規則 (アクション) 以外にも、GIGA では生成規則の適用条件である制約や、属性についても図式表現を例示することにより、グラフィカルに文法を定義し、その文法に基づいてビジュアルシステムを生成することができる点で違いがある。また、実際のビジュアルシステムを拡張 CMG で記述した場合、ビジュアルシステムは多くの生成規則から構成される。この編集作業において、編集対象となる生成規則を探し出す作業にも GIGA のグラフィカルな文法編集インタフェースは効果を発揮する。GIGA が提供する図式表現を眺めることによって生成規則の意味を大つかみに把握し、探し出す生成規則に見えるものが見つかった時に初めて細かな部分を読めば良いからである。

3.4　まとめ

本章では、空間解析器生成系に与える図形文法をグラフィカルに編集するための手法について述べた。その手法に基づき図式表現を用いてグラフィカルに拡張 CMG を編集するインタフェースを持つシステム GIGA を Java 言語を用いて実現した。GIGA では、生成規則の各構成要素を視覚的に表現し、各種の手法を用いてそれらの要素に対する直接操作を行うことにより、生成規則の定義を行うことができ、さらに図式表現された図形文法の生成規則からその意味を容易に把握することができる。具体的には、生成規則を構成する構成要素、制約、属性、およびアクションのそれぞれについて、以下のように定義・把握することができる。構成要素については、構成要素に対応する図形を定義インタフェースに描画することにより定義を行うため、構成要素を定義インタフェース上で識別することができる。制約については、その図形を直接操作してシステムに制約の推論をさせる。推論の結果が即座に定義インタフェースに提示されるため、制約が意図した通りかどうかをインタラクティブに確認することができる。この際、図形の属性を重ね合わせるという操作によって図形の関連付けを行うことにより、推論対象の図形を絞り込み、不要な推論がシステムによって行われないようにする指定を行なっている。属性については、座標値の場合

には構成要素の対応部分を定義インタフェースで指定することにより定義することができる。定義した属性は定義インタフェースを見るだけで把握することができる。アクションについては、アクションが実行される前の図と実行された後の図を作成することによって定義することができる。定義を行なったアクションは定義インタフェースの左側と右側を見比べることによって一目で把握することが可能である。

第4章　文法編集システムと空間解析器生成系の統合

本章では、まず従来の空間解析器生成系において定義と実行の画面が分かれているために起こる問題点について述べる。次に、グラフィカルな図形文法定義インタフェースと空間解析器生成系を統合することを提案し、提案手法に基づいて実現したグラフィカルな図形文法定義インタフェースを持つ空間解析器生成系 Viola について述べる。

4.1　恵比寿

空間解析器生成系を用いることで、図形言語を解析する空間解析器を図形文法の記述から生成することができる。しかし実際の図形言語の処理系では図形言語の解析だけではなく、解析結果に応じた動作を行い、図形間に制約を課すことによって意味的関係を保存し、さらには描いた図形へのフィードバックを行うことが必要とされる。馬場らは空間解析器生成系と制約解消系を持つことでこれらの要求を満たすようなシステム恵比寿を作成した。恵比寿の実行画面を図 4.1 に示す。

図の画面の上半分を定義ウィンドウと呼び、図形言語の作製者はここで文法を定義する。下半分を実行ウィンドウと呼び、ユーザはここで実際に解析、実行したい図形を入力する。これは通常のテキスト言語ではエディタを用いてプログラミングすることに相当する。定義ウィンドウ、実行ウィンドウにおける図形の入力では、通常のドローツールにあるような、コピー、削除、整列といった操作をおこなうことができる。扱える図形の種類は楕円、長方形、直線、テキスト文字列、円弧、GIF イメージがあり、これらは線の太さ、色、フォント

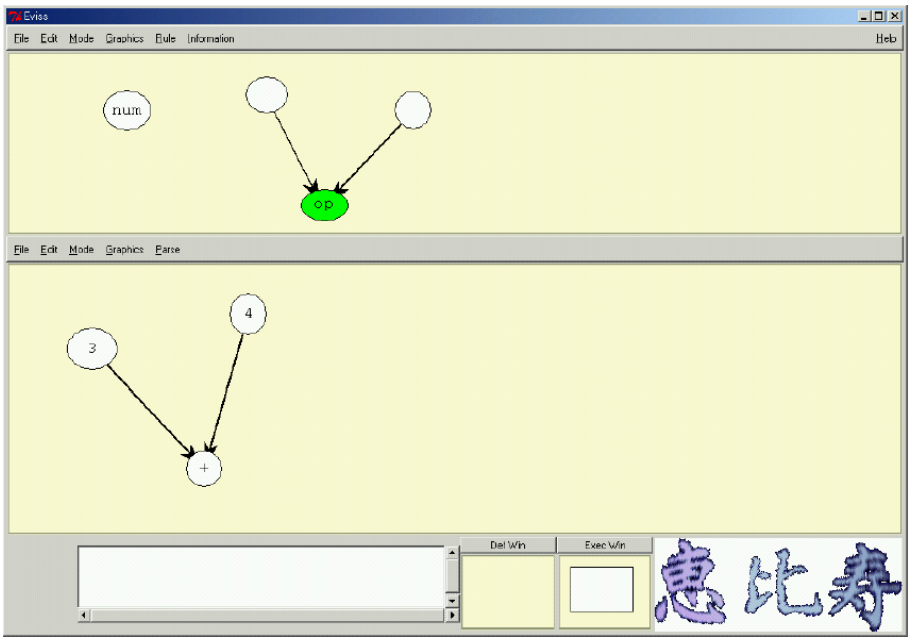

図 4.1: 空間解析器生成系 恵比寿

などの属性を指定できる。

恵比寿における文法の定義

恵比寿では、ユーザはまず定義ウィンドウに大まかな文法と構成要素を描く。次に定義ウィンドウに描いた図形の中から、一つの図形単語としたい図形を選択すると図 4.2 に示す CMG 入力ウィンドウが表示される。同時に恵比寿は選択された図形から、構成要素とそれらの属性間に成り立っている簡単な制約を自動的に生成し、それらを CMG 入力ウィンドウにテキストで書き出す。ユーザは CMG 入力ウィンドウに新たに名前、属性、アクションを追加し、システムによって生成された制約のうち必要のないものを削除していくことでビジュアル言語の文法を定義する。CMG 入力ウィンドウは上から順に名前 (name)、属性 (attributes)、アクション (action)、構成要素間の制約 (constraints)、構成要素を書く欄にわかれている。構成要素を書く欄についてはさらに左から順に normal、exist、not exist、all にわかれている。

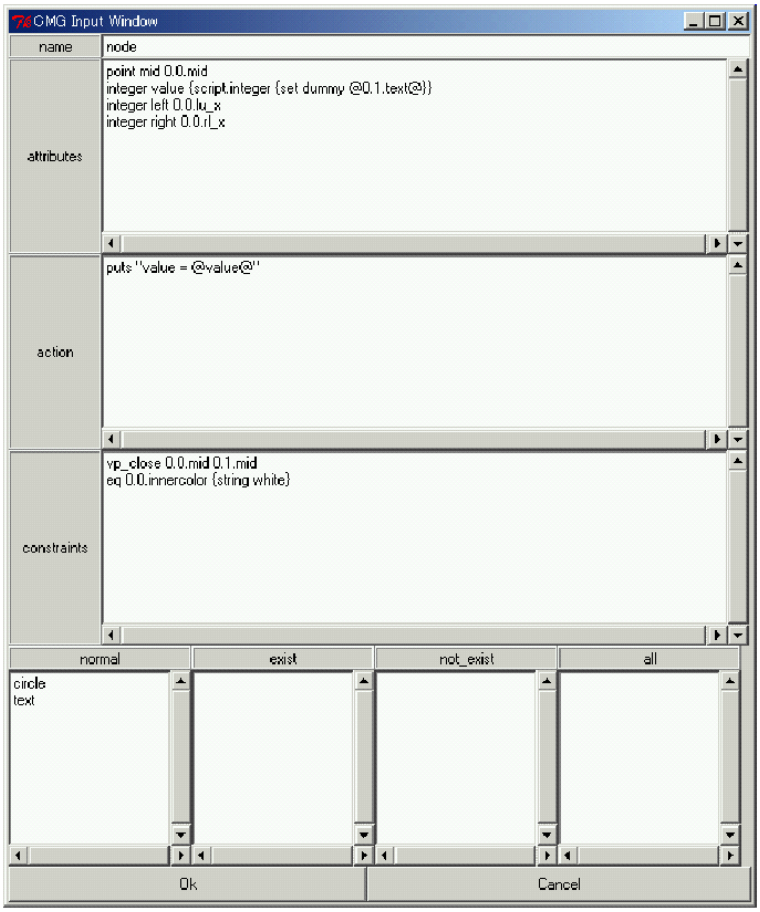

図 4.2: CMG 入力ウィンドウ

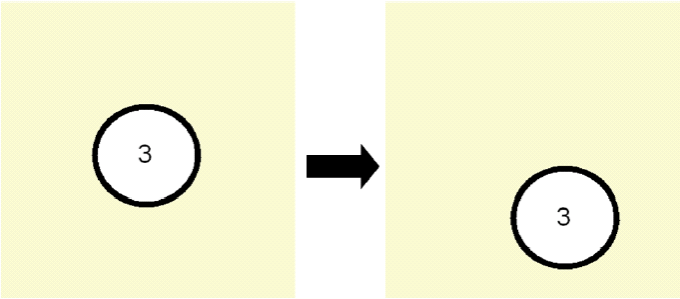

図 4.3: 図形単語の移動

4.1.1　恵比寿におけるビジュアルシステムの実行

恵比寿では、図形言語の文法の定義が終わったら、実行ウィンドウに実際に解析・実行をしたい図形を入力する。入力された図形が空間解析器によってトークンとして認識されると、そのトークンを認識した生成規則に書かれた制約がトークンの属性間に課せられる。たとえばある図形単語は円とテキストを構成要素としていて、円の中心とテキストの中心が一致しているという制約条件であった場合。円とテキストが認識されると、その制約が円とテキストの属性間に課せられ、以降の編集においては円を移動させるとテキストもその中心が円の中心と一致するように移動し、結果として円とテキストは一つの図形単語として移動を行うことができる (図 4.3)。

このように恵比寿では、トークンの属性を変数とし、それらの属性の間に制約を課す。各トークンの間に成り立っている制約が常に成り立つように維持するシステムを制約解消系と呼び、恵比寿では制約解消系として SkyBlue の C 言語による実装に手を加え、これを Tcl から呼び出すインタフェースを作成し、使用している。

ここで、ビジュアルシステムの実行例として、実際に計算を行うビジュアルシステムである計算の木を恵比寿上で実行した様子を図 4.4 に示す。入力された計算の木 $(3+4)*5$ に対し、まず始めに $(3+4)$ の部分にあたる計算の木が解析器により 7 というノードに書き換えられる。さらに、書き換えられたノードと残りの部分が計算の木を構成しているため、引き続き解析が行われ $7*5$ を表す計算の木が 35 というノードに書き換えられ、実行が終了する。

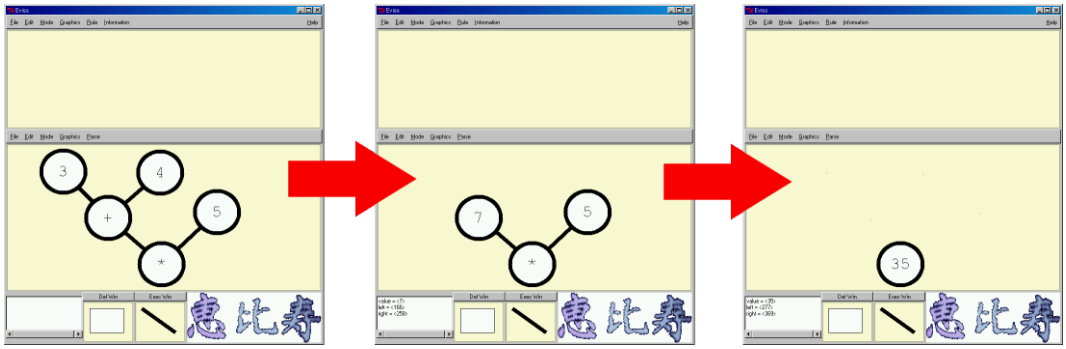

図 4.4: 計算の木の実行

4.2　文法定義画面と実行画面の統合

恵比寿の図形エディタでは、図形言語の文法を定義する定義ウィンドウと実際に図形の解釈を行う実行ウィンドウに分れていた。そのため、恵比寿では構成要素と簡単な制約については定義ウィンドウに描画された図から、恵比寿が自動的に生成するのであるが、他の文法要素を入力するためには CMG 入力ウィンドウを開き、そこで定義を行う必要があり、どのような文法を定義したのかを実行時に確認することができない。

この問題を解決するために恵比寿を拡張し、定義部分と実行部分を統一した空間解析器生成系 VIC[Fuj99, Fuj01] と Rainbow[Jou00b, Jou00a, jou01] が開発された。

4.2.1　Rainbow

Rainbow は、拡張 CMG の制約にレイイアウト制約を導入した空間解析器生成系である。レイアウト制約とは図形が解析された後、図形要素間の複雑な構造や関係を分かりやすく、かつバランス良く表す位置関係を作り出す制約である。

Rainbow では恵比寿における定義ウィンドウと実行ウィンドウを統合し、定義と実行を同じ画面で行うことが可能となっている。図 4.5 に Rainbow の図形エディタを示す。

Rainbow では、生成規則を定義することにより、図形エディタ上に描いた図

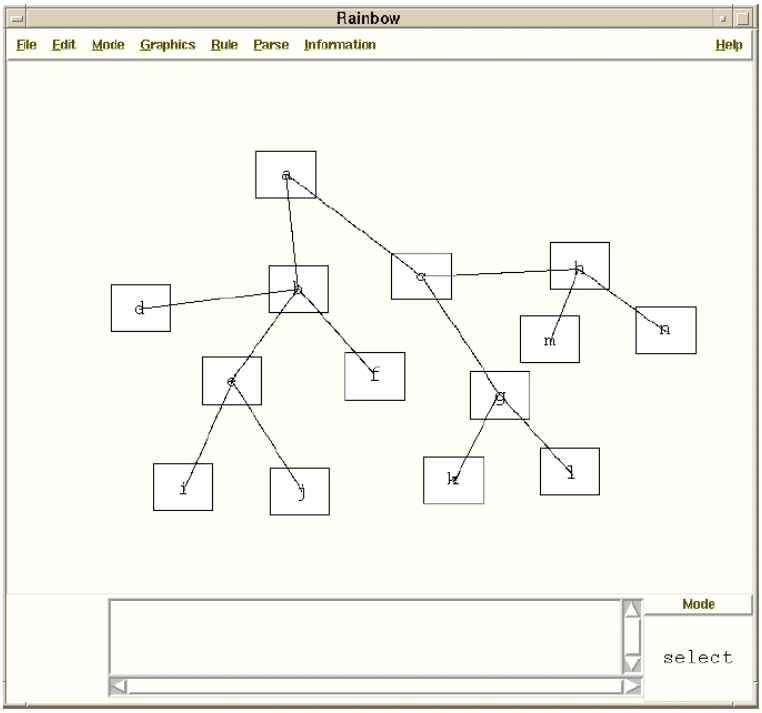

図 4.5: Rainbow

形が定義した規則に従って解析されるが、図形文法を定義するには以下の手順により定義を行う必要がある。まず、ユーザは構成要素として定義したい図形を図形エディタに描く。次に、その図形を選んで図 4.6 に示す CMG 入力ウィンドウを開く。Rainbow は選択された図形から構成要素とそれらの属性間に成り立っている制約を推論し CMG 入力ウィンドウに書き出す。CMG 入力ウィンドウは恵比寿のそれと同様に、上から順番に図形単語名、属性、制約、構成要素を書く欄になっている。ユーザはこのウィンドウ上で制約を修正し、また図形単語名、属性、アクションを編集することにより、生成規則を定義する。

生成規則の定義が終わったら実際に図形エディタに図形を入力することで、文法に従った図形の解析が行われる。

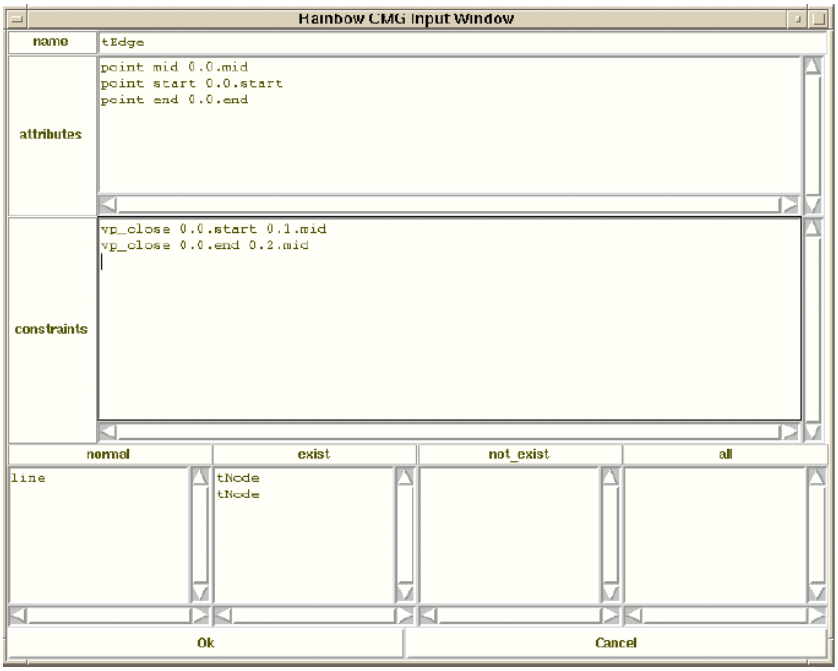

図 4.6: Rainbow の文法入力ウィンドウ

4.2.2　VIC

VIC は図形文法の編集に例示入力図という視覚的な表現を利用した空間解析器生成系であり、テキスト編集を行わず、ほぼ例示入力図に対するマウス操作のみで生成規則を定義できる。定義と実行の境界をなくすことにより、ユーザは文法の定義と実行の作業を一貫して一つのウィンドウで行うことができる。VIC の画面を図 4.7 に示す。

VIC で文法を定義するためには、まず生成規則の構成要素となる図形を画面上に描画する。次に、描画した図形を選択し、文法を入力するための文法定義ウィンドウを表示する。VIC の文法入力ウィンドウを図 4.8 に示す。このウィンドウへのマウス操作により制約や属性などの詳細な文法を記述することができる。

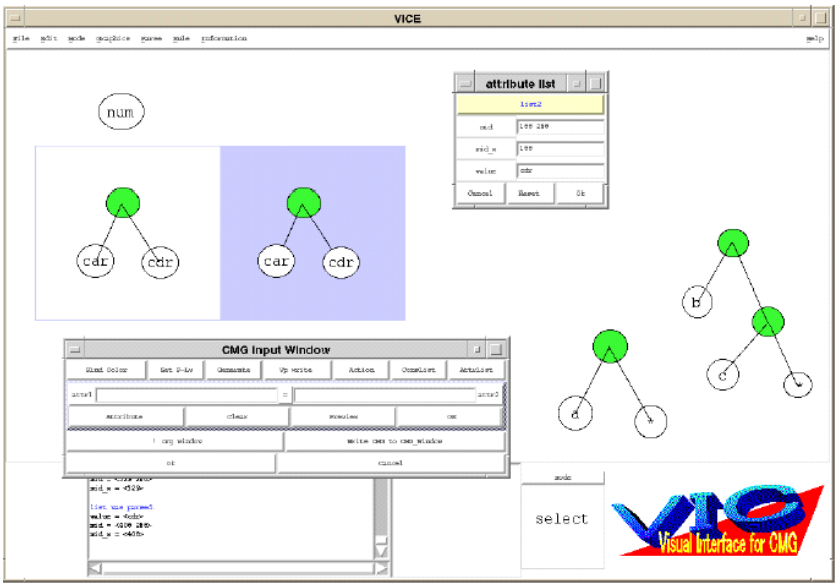

図 4.7: VIC

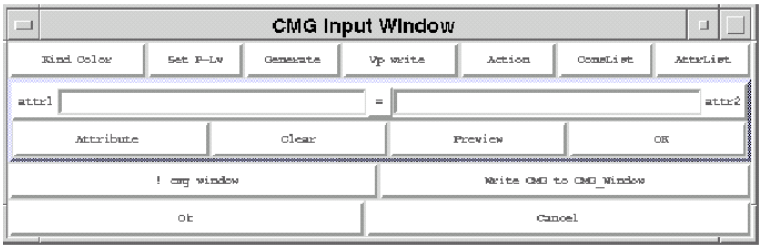

図 4.8: VIC の文法入力ウィンドウ

4.3　空間解析器生成系 Viola

4.3.1　GIGA システムと空間解析器生成系の統合

図形文法編集システム GIGA は、図形文法をグラフィカルに編集するための手法を用いることで、生成規則の各構成要素を視覚的に表現し、構成要素に対する直接操作を行うことにより、生成規則の定義を行うことを可能にした。さらに図式表現された図形文法の生成規則からその意味を容易に把握することが可能である。しかし、GIGA で定義を行ったビジュアルシステムを実行するためには、定義した文法をファイルに出力し、それを恵比寿などの空間解析器生成系に与える必要があった。GIGA の定義画面の上では視覚的に確認をすることができた生成規則も、恵比寿上では CMG 入力ウィンドウを用いてテキストで確認しなければならない。また、複数の生成規則を同時に閲覧することも不可能である。さらに、定義した図形文法に誤りが見つかり生成規則を修正する場合には、GIGA を用いて生成規則を修正した後、それをファイルに出力し、恵比寿で読み込むという手順を繰り返さなければならない。

これらの問題を解決するためには図形文法の定義をグラフィカルに行うだけではなく、定義を行った図形文法から空間解析器を生成し、生成された空間解析器を用いてビジュアルシステムの実行を行うこと、つまり VIC や Rainbow と同様に定義画面と実行画面を統合する必要がある。そこで、我々は GIGA には図式表現を編集するための図形エディタがあることを利用し、GIGA に図形エディタに空間解析器生成系の機能の統合し、図形エディタ上でビジュアルシステムの実行を行えるように改良を加えた。本章では、GIGA に空間解析器生成系の機能と制約解消系の機能を追加し、文法の編集だけではなく、ビジュアルシステムの実行が可能なシステムとして新たに空間解析器生成系 Viola を作成した。Viola では図形文法の定義とビジュアルシステムの実行の作業を一貫した一つの図形エディタ上で行うため、定義を行っている文法を確認しながらビジュアルシステムの実行を行うことができる。

以降では GIGA システムと空間解析器生成系を統合する際に生じた問題点を示しながら、Viola で用いた各要素の定義手法について述べる。

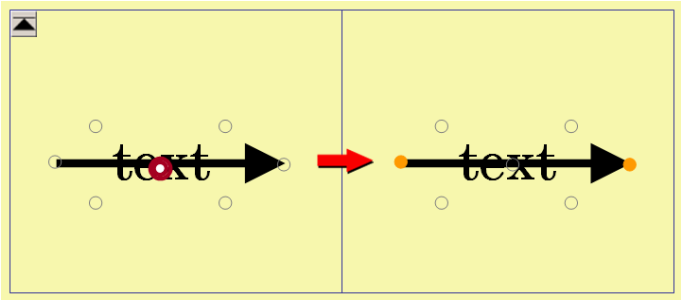

図 4.9: 生成規則の範囲の定義

4.3.2　生成規則の範囲の定義

GIGA では、生成規則の各構成要素を視覚的に表現し、それらの要素に対して直接操作を行うことにより、生成規則の定義を行っていた。例えば、生成規則の構成要素に対応する図形を図形エディタ上に直接描くことにより構成要素の定義を行う。また、構成要素として描いた図形の属性を選択し、それらを直接操作して生成規則の図式表現を例示することで、システムは制約を推論し、図形エディタ上に定義された制約を表示する。この際、図形の属性を重ね合わせるという操作によって図形の関連付けを行うことにより、推論対象の図形を絞り込み、不要な推論がシステムによって行われないようにする指定を行なう。

しかし、構成要素の属性の表示や、成立している制約の表示などは、文法の定義の際には有用であるが、ビジュアルシステムを実行する際には不要である。そこで Viola では生成規則ごとに定義を行なう範囲をあらかじめ指定し、その範囲内にある図形についてのみ、GIGA のように構成要素の属性の表示や、推論した制約の表示などの視覚化を行うことにした。一方、生成規則の定義範囲より外側に描いた図形については、生成規則の定義ではなくビジュアルシステムの実行にのみ使用すると判断し、これらの視覚化を行わない。

生成規則の定義範囲を指定するには、ツールパネルにある rule と書かれたボタンを選択し、図 4.9 のように画面上に範囲を示す枠を描画する。この枠の中で図を編集することで、それらが 1 つの生成規則の定義となる。

構成要素については、左側の枠の中に図形を描くことにより定義を行う。構成要素として描いた図形は、GIGA と同様に図形にあらかじめ定義されている

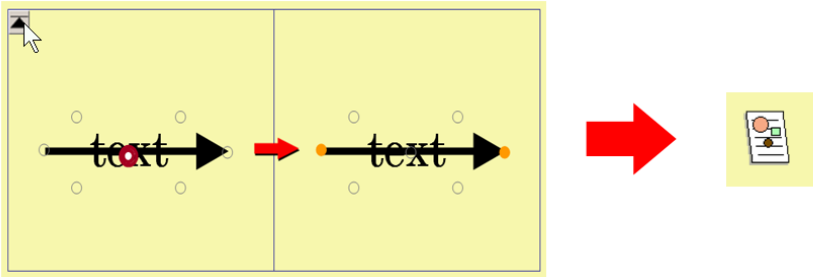

図 4.10: 生成規則の縮小表示

属性が半透明で表示される。その中から生成規則の中で使用したい属性をマウスでクリックすることにより、属性の選択を行うことができる。

制約については、左側の枠の中に構成要素として描いた図形を操作し、定義したい制約を例示することにより定義を行う。この時、定義している制約が画面上に図示される。

構成要素及び制約の定義を終えた後、定義画面の中央にある矢印をクリックすることで、構成要素として描いた図形が右側の画面に複製される。複製された図形に対して操作を行うことで、属性とアクションを定義することができる。

属性については、図形単語の属性として定義したい値を、複製された図形の属性の中から選択し、属性の値を表示している円をクリックすることにより属性の定義を行う。

アクションについては、定義インタフェースの右側の画面で、複製された図形を削除したり、新しい図形を追加するなどの操作を行い、アクションが実行された後の図式を作成することで、定義を行う。

生成規則の数が多い複雑なビジュアルシステムの定義を行う場合、定義画面内ですべての生成規則を定義しきれない場合がある。画面をスクロールすることでこのような状況を回避することができるが、それでは定義した生成規則をすべて同時に閲覧することが不可能になってしまう。そこで図 4.10 に示すように生成規則の定義範囲の左上に縮小ボタンを設け、このボタンを押すことで、生成規則をアイコン表示し表示スペースを節約することができるようにした。

拡張 CMG では生成規則の左辺にある図形単語に同じ名前をつけることで、構成要素や制約が異っていても、同じ図形単語に還元させる生成規則を定義する

ことができる。Viola では図形単語の名前はシステムにより暗黙に決定される。そのため、このような生成規則を定義するためには、同じ図形単語に還元させたい生成規則をアイコン表示し、アイコン同士を重ねあわせることで、明示的に同じ図形単語に還元される生成規則であることを定義する。

4.3.3　定義図形に対する空間解析

GIGA では図形を組み合わせて図形単語となるようにしたものを定義に使用しても、それは認識されず図形単語として構成要素を定義することができない。Viola ではビジュアルシステムの実行時だけではなく、図形文法を定義するために入力した図形に対しても、すでに定義を行った図形文法に基づいた解析を行っている。例えば、テキストと円で構成されている図形単語ノードを構成要素として持つ生成規則の定義を行う際に、まず図形単語ノードの定義を行った後、テキストと円を描くとそれがノードとして認識され、認識されたノードを構成要素として別の生成規則を定義することができる。

4.4　空間解析器生成系 Viola の実装

Viola の実装には Java(j2sdk 1.4.1_01) を用いた。Viola の画面を図 4.11 に示す。Viola は図形言語の文法を定義する場合と実際に図形言語を実行する場合の両方に用いられる図形エディタを備えている。図形エディタで扱うことのできる図形の種類は長方形、楕円、円弧、直線、画像、テキスト文字列があり、画面上部にある各ボタンで描画する図形を選択することができる。線の色や太さ、フォントなどの属性についてはメニューから変更することができる。また一般の図形エディタと同様に、描いた図形に対して移動、コピー、削除といった操作を行うこともできる。

Viola の空間解析器の解析アルゴリズムには、Chok らが提案した CMG の解析アルゴリズム [Cho95a, Cho95b] の一部を変更して使用した。使用した解析アルゴリズムを図 4.12 に示す。

図 4.12 に示したアルゴリズムでは新たな図形が入力されるたびにインクリメン

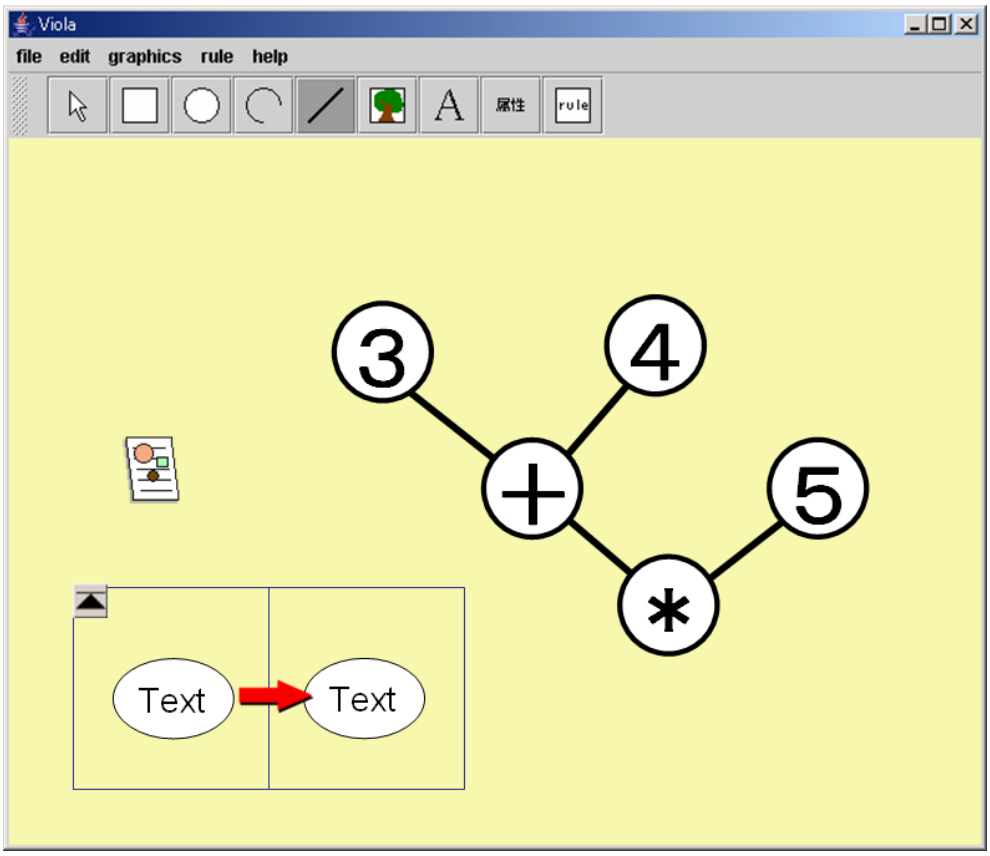

図 4.11: 空間解析器生成系 Viola

```
procedure Parse(t)
 AddToken(t)
  repeat
   for  each rule R in the SCC do
     EvalRule(R)
   end for
 until ParseForest is unchanged
end
```

図 4.12: CMG 解析アルゴリズム

タルに解析が行われる。新しい図形トークンが入力されると、手続き AddToken を用いて、入力された図形トークンを `ParseForest` に挿入する。`ParseForest` は生成規則が適用される対象となるトークンのデータベースである。`SCC` は生成規則のデータベースで、`SCC` 中では低い階層の図形単語を作るための生成規則から順に並べられている。`ParseForest` にある全てのトークンに対して、`SCC` 中の生成規則 R が適用できるか手続き EvalRule を用いて調べ、適用できる生成規則が存在すれば、新たな図形単語を `ParseForest` に挿入し、生成に用いられた図形単語を `ParseForest` から削除する。`ParseForest` にあるトークンが変更されなくなるまで生成規則の適用を続ける。

また、恵比寿で用いていた制約解消系 SkyBlue は非線形な幾何制約を解くことができない。そこで、本システムでは幾何制約を扱うことができる制約解消系である Chorus[Hos01] を用いた。

4.5　Viola を用いた図形言語の文法定義と実行の例

4.5.1　計算の木の定義例

拡張 CMG では再帰的な生成規則を定義することができるが、Viola を用いることで、そのような再帰的な生成規則についても容易に定義することができる。計算の木を実行するビジュアルシステムを例として、Viola を用いた再帰的な生成規則の定義方法を述べる。

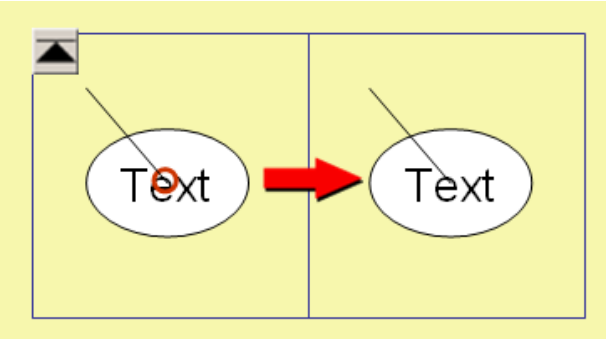

図 4.13: 計算の木のノード–1

計算の木の 1 つ目の生成規則は木の葉にあたる部分の定義であり図形単語 node は円とテキストにより構成されていることを定義する必要がある。まず Viola の図形エディタ上で生成規則の定義範囲を描画する。構成要素を定義するために、定義範囲の左側の枠の中に中心の座標を揃えた円とテキストを描く。 2 つ目のルールと区別をするために not exist 構成要素として直線が 1 つ必要なので、円とテキストと同様に描く (図 4.13)。1 つ目の生成規則にはアクションがないので、以上で node の生成規則を定義できた。

この生成規則を定義したことにより、node の解析を行なう空間解析器が自動的に生成され、これより後で、中心の座標が一致するテキストと円が図形エディタに描画されると、それらは 1 つのノードとして認識され、中心が同じであるという制約が課される。

計算の木の 2 つ目の生成規則は木の節にあたる部分の定義であり図形単語ノード (node) は円とテキスト、2 つのノード、2 つの直線により構成されていることを定義する必要がある。まずは生成規則の定義範囲を描画する。範囲を指定したら、定義範囲の左側の枠の中に構成要素として、中心の座標を揃えた円とテキストを 2 つ描く。1 つ目の生成規則がすでに定義されているので、空間解析器によりこれらは 2 つのノードとして認識される。更に、円の中心に向かう直線及び中心の座標を揃えた円とテキストを描く。この円とテキストもノードとして認識されそうであるが、1 つ目の生成規則において not exist 制約として直線が定義されている。そのためこの円とテキストはノードとして認識されることはない。次にアクションとして、生成規則の右側の枠の中に複製された構成要

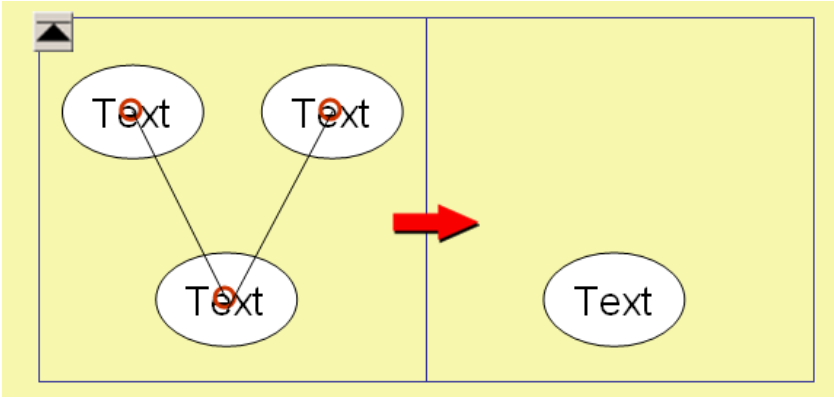

図 4.14: 計算の木のノード–2

素の中から 2 つのノードと 2 つの直線を削除し、2 つ目の生成規則の定義を終える (図 4.14)。

最後に、この 2 つの生成規則は同じ図形単語であるノードに還元される生成規則であることを定義するために、2 つの生成規則を縮小しアイコン表示する。2 つのアイコンを重ねることで、これらの生成規則が適用されると同じ図形単語 (Viola システムにより自動的に名前が付けられる) に還元されることを定義することができる。

4.5.2　計算の木の実行

計算の木の 2 つの生成規則の定義を行うと、Viola により計算の木を認識する空間解析器が自動的に生成され、実際に図形エディタ上で計算の木の解析を実行することが可能になる。定義を行った計算の木を Viola 上で実行している様子を図 4.15 に示す。

4.6　まとめ

本章では、従来の空間解析器生成系における、定義した図形文法を確認しながらビジュアルシステムを実行できないという問題を解決するために、図形文法定義システム GIGA に対して、空間解析器生成系と制約解消系を統合し、グラ

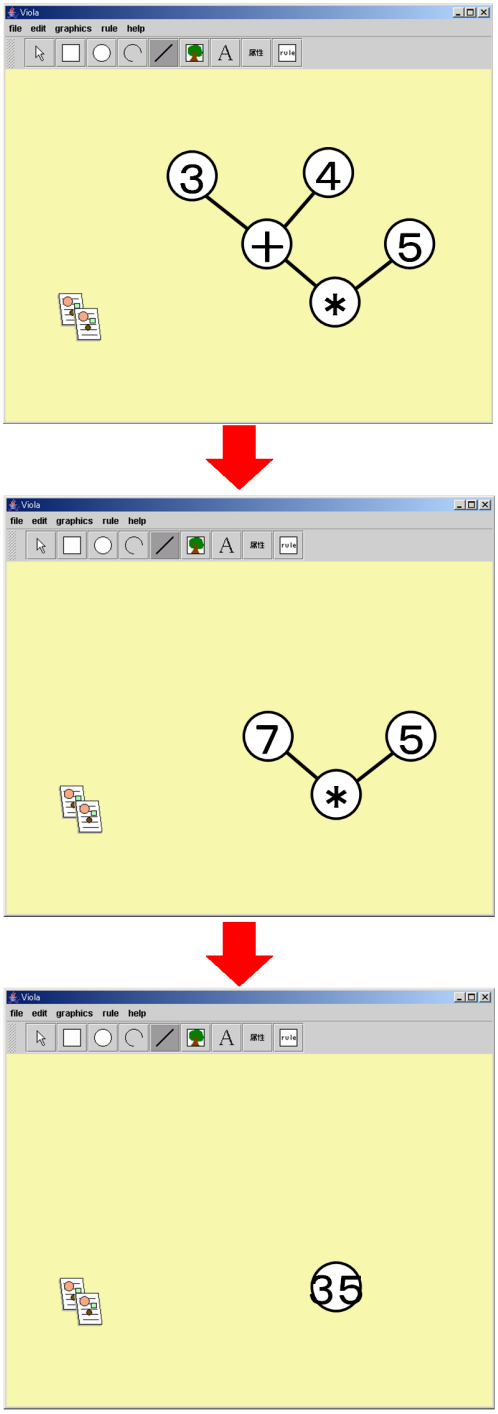

図 4.15: ビジュアルシステムの実行

フィカルな定義インタフェースを持つ空間解析器生成系 Viola を実現した。Viola では図形文法の定義とビジュアルシステムの実行の作業を一貫した一つの図形エディタ上で行うため、定義を行っている文法を確認しながらビジュアルシステムの実行を行うことができる。さらに、図形文法の定義に用いている図形に対しても文法に基づいた解析を行うことにより、再帰的な生成規則の定義を容易に行うことが可能である。

第5章　結論

本論文ではビジュアルシステムの開発時において、図形文法の定義を図式を編集することで視覚的に行う手法及び、定義と実行の作業を一貫した1つの画面上で行う手法について述べた。

2章では、拡張CMGをグラフィカルに編集するために作成したシステムGIGAについて述べた。GIGAは、生成規則の各構成要素を視覚的に表現し、各種の手法を用いてそれらの要素に対する直接操作を行う。これによりユーザは生成規則の定義を行うことができ、さらに図式表現された図形文法の生成規則からその意味を容易に把握することができる。具体的には、生成規則を構成する構成要素、制約、属性、およびアクションのそれぞれについて、以下のように定義・把握することができる。構成要素については、構成要素に対応する図形を定義インタフェースに描画することにより定義を行うため、構成要素を定義インタフェース上で識別することができる。制約については、その図形を直接操作してシステムに制約の推論をさせる。推論の結果が即座に定義インタフェースに提示されるため、制約が意図した通りかどうかをインタラクティブに確認することができる。この際、図形の属性を重ね合わせるという操作によって図形の関連付けを行うことにより、推論対象の図形を絞り込み、不要な推論がシステムによって行われないようにする指定を行なっている。属性については、座標値の場合には構成要素の対応部分を定義インタフェースで指定することにより定義することができる。定義した属性は定義インタフェースを見るだけで把握することができる。アクションについては、アクションが実行される前の図と実行された後の図を作成することによって定義することができる。定義を行なったアクションは定義インタフェースの左側と右側を見比べることによって一目で把握することが可能である。GIGAを用いることによって、恵比寿におけるテキストと図形を用いた入力では困難であった、制約の把握・定義、属性

の記述の煩雑さ、アクションの結果の把握しづらさ、といった問題について改善することができた。

3 章では、GIGA のグラフィカルな文法編集システムに、空間解析器と制約解消系を加え、図形文法の定義部と実行部を統合した空間解析器生成系 Viola について述べた。図形文法の定義とビジュアルシステムの実行の作業を一貫した一つの図形エディタ上で行うため、定義を行っている文法を確認しながらビジュアルシステムの実行を行うことができる。また、編集している図形に対しても空間解析を行うことにより、再帰的な生成規則の定義を行うことが可能である。

本論文で述べたこれらの技法を用いることで、従来の空間解析器生成系において問題となっていた、制約の把握・定義、属性の記述の煩雑さ、アクションの結果の把握しづらさ、などの定義の際の問題を解決できる。さらに、グラフィカルに図形文法を定義し、同時にビジュアルシステムの実行を行うことにより、文法の正しさを検査するという作業が容易になる。これらにより、空間解析器生成系を用いたビジュアルシステムの開発作業を効率よく行うことが可能になった。

謝辞

本研究を進めるにあたって、筑波大学電子・情報工学系田中二郎教授には研究の始めから、指導教官として多くの御指導・御助言をいただきました。ここに深く感謝し、心から御礼を申し上げます。

筑波大学電子・情報工学系の大保信夫教授、西川博昭教授、福井幸男教授、櫻井鉄也助教授、志築文太郎講師には、本論文の審査にあたって数多くの貴重な助言・討論をいただきました。ここに深く感謝いたします。

佐賀大学理工学部知能情報システム学科の山下義行教授、法政大学情報科学部コンピュータ科学科の中田育男教授、筑波大学図書館情報学系の中井央助教授、静岡大学情報学部情報科学科助手の秡川友宏氏、立命館大学理工学部情報学科助手の糸賀裕弥氏には筑波大学プログラミング研究室に所属している際に、多くの御指導・御助言をいただき、大変お世話になりました。ここに深く感謝いたします。

筑波大学電子・情報工学系助手の三浦元喜氏には、研究に関して多くの貴重な助言をいただきました。飯塚和久氏、酒寄保隆氏をはじめとする筑波大学インタラクティブプログラミング研究室の皆様には研究を進める上での議論に参加・協力していただき、多くの貴重な助言をいただきました。ここに深く感謝いたします。

筑波大学工学システム学類の同期の飯島賢吾氏、井上剛毅氏、遠藤浩通氏、小川高星氏、奥村穂高氏、小倉知樹氏、片下敏宏氏、國生政義氏、古口晴敏氏、小柳光生氏、清水智彦氏、羽田靖史氏、松井俊輔氏、柳谷正章氏、山下淳氏、吉田智章氏には苦しい時にいろいろと励まして頂きました。この場を借りて御礼を申し上げます。

最後に、今日まで私を育ててくれた父と母、そして祖父と祖母には、あらためて深く感謝いたします。

参考文献

[Bab98a] 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, No. 5, pp. 1385–1394, May 1998.

[Bab98b] Akihiro Baba and Jiro Tanaka. Eviss: A visual system having a spatial parser generator. In *Proceedings of Asia Pacific Computer Human Interaction*, pp. 158–164, July 1998.

[Bab98c] 馬場昭宏. Spatial parser generator を持ったビジュアルシステム. 筑波大学大学. Master's thesis, 筑波大学大学院工学研究科, January 1998.

[Bab99] 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, No. 2, pp. 497–506, 1999.

[Bor97] Alan Borning, Kim Marriott, Peter J. Stuckey, and Yi Xiao. Solving linear arithmetic constraints for user interface applications. In *Proceedings of the ACM Symposium on User Interface Software and Technology*, pp. 87–96, October 1997.

[Cho95a] Sitt Sen Chok and Kim Marriott. Automatic construction of user interfaces from constraint multiset grammars. In *Proceedings of IEEE Symposium on Visual Language*, pp. 242–249, 1995.

[Cho95b] Sitt Sen Chok and Kim Marriott. Parsing visual languages. In *Proceedings of 18th Australasian Computer Science Conference*, pp. 90–98, 1995.

[Cho98] Sitt Sen Chok and Kim Marriott. Automatic construction of intelligent diagram editors. In *Proceedings of ACM Symposium on User Interface Software and Technology*, pp. 185–194, 1998.

[Cho03] Sitt Sen Chok and Kim Marriott. Automatic generation of intelligent diagram editors. Vol. 10, No. 3, pp. 244–276, Sepember 2003.

[Cor91] R. Corbett and R. Stallman. Bison: Gnu parser generator. texinfo documentation, free software foundation, cambridge, mass, 1991.

[Cos93] Gennaro Costagliola, Sergio Orefice, Genoveffa Tortora, and Maurizio Tucci. Automatic parser generation for pictorial languages. In *Proceedings of the IEEE Symposium on Visual Languages*, pp. 306–313, 1993.

[Cos95] Gennaro Costagliola, Genoveffa Tortora, Sergio Orefice, and Andrea de Lucia. Automatic generation of visual programming environments. *IEEE Computer*, Vol. 28, No. 3, pp. 56–66, March 1995.

[Cos98] Gennaro Costagliola, Andrea de Lucia, Sergio Orefice, and Genoveffa Tortora. Positional grammars: A formalism for lr-like parsing of visual languages. In *Visual language Theory*, pp. 171–191. Springer, 1998.

[Cos99] Gennaro Costagliola, Filomena Ferrucci, Giuseppe Polese, and Giuliana Vitiello. Supporting hybrid and hierarchical visual language definition. In *Proceedings of IEEE Symposium on Visual Languages*, pp. 236–245, September 1999.

[Cyp95] Allen Cypher and David Canfield Smith. Kidsim: End user programming of simulations. In *Proceedings of the ACM Conference on Human Factors in Computing Systems (CHi' 95)*, pp. 27–34, May 1995.

[Fer94] F. Ferrucci, G. Tortora, M. Tucci, and G. Vitiello. A predictive parser for visual languages specified by relation grammars. In *Proceedings of the IEEE Symposium on Visual Languages*, pp. 245–252, 1994.

[Fuj99] Kenichiro Fujiyama, Kazuhisa Iizuka, and Jiro Tanaka. Vic:cmg input system using example figures. In *Proceedings of International Symposium on Future Software Technology (ISFST' 99), Nanjing, China*, pp. 67–72, October 1999.

[Fuj01] 藤山健一郎. 例示入力図を用いた spatial parser generator. Master's thesis, 筑波大学大学院工学研究科, 2001.

[Gle94] Michael Gleicher and Andrew Witkin. Drawing with Constraints. *The Visual Computer*, Vol. 11, No. 1, pp. 39–51, 1994.

[Gol91] Eric J. Golin. Parsing visual languages with picture layout grammars. *Journal of Visual Languages and Computing*, Vol. 2, No. 4, pp. 371–394, December 1991.

[Gol93] Eric J. Golin and Tom Magliery. A compiler generator for visual languages. In *Proceedings of IEEE Symposium on Visual Languages*, pp. 316–321, August 1993.

[Har97] Yasunori Harada, Kenji Miyamoto, and Rikio Onai. Vispatch:graphical rule-based language controlled by user event. In *Proceedings of the 1997 IEEE Symposium on Visual Languages*, 1997.

[Hir91] Masahito Hirakawa, Yukihiro Nishimura, Motoji Kado, and Tadao Ichikawa. Interpretation of icon overlapping in iconic programming. In *Proceedings of the 1991 IEEE Workshop on Visual Languages*, pp. 254–259, 1991.

[Hon00] Jason I. Hong and James A. Landay. Satin: a toolkit for informal ink-based applications. In *Proceedings of the ACM Symposium on User Interface Software and Technology*, pp. 63–72, 2000.

[Hos01] Hiroshi Hosobe. A modular geometric constraint solver for user interface applications. In *Proceedings of the 14th Annual ACM Symposium on User Interface Software and Technology (UIST'01)*, Vol. 3, pp. 91–100, 2001.

[Iga98] 五十嵐健夫, 松岡聡, 河内谷幸子, 田中英彦. 対話的整形による幾何学的図形の高速描画. 情報処理学会論文誌, Vol. 39, No. 5, pp. 1373–1384, May 1998.

[Joh79] Steven C. Johnson. Yacc: Yet another compiler compiler. In *UNIX Programmer's Manual*, Vol. 2B, pp. 353–387. Holt, Rinehart, and Winston, 1979.

[Jou00a] Joung Sucktae and Jiro Tanaka. Generating a visual system with soft layout constraints. In *Proceedings of the International Conference on Information*, pp. 138–145, 2000.

[Jou00b] Joung Sucktae and Jiro Tanaka. Rainbow: ビジュアルシステム生成系におけるレイアウト制約の実現. 情報処理学会論文誌, Vol. 41, No. 5, pp. 1246–1256, 2000.

[jou01] 丁錫泰, 田中二郎. 空間パーサにおける木構造レイアウト制約の実現とその評価. 電子情報通信学会論文誌, Vol. J84–D–I, No. 9, pp. 1350–1361, 2001.

[Kam02] Hiroaki Kameyama, Kazuhisa Iizuka, Buntarou Shizuki, and Jiro Tanaka. Giga:graphical definiton of production rules in a spatial parser generator. In *Proceedings of the International Symposium on Future Software Technology (ISFST2001),Wuhan and Xian, China*, 2002. (6 pages).

[Kam03] 亀山裕亮, 飯塚和久, 志築文太郎, 田中二郎. GIGA:空間解析器生成系におけるグラフィカルな文法編集システム. 情報処理学会論文誌, Vol. 44, No. 11, pp. 2565–2574, November 2003.

[Mar94] Kim Marriott. Constraint multiset grammars. In *Proceedings of the IEEE Symposium on Visual Languages*, pp. 118–125, 1994.

[Mar96] Kim Marriott and Bernd Mayer. Towards a hierarchy of visual languages. In *Proceedings of the 1996 IEEE Symposium on Visual Languages*, pp. 196–203, 1996.

[Rum91] James Rumbaugh, Michael Blaha, William Premerlani, Frederick Eddy, and Wiliam Lorensen. *Object–Oriented Modeling and Design.* Prentice Hall, 1991.

[Rum92] ジェームズ・ランボー, マイケル・ブラハ, ウィリアム・プレメラニ, フレデリック・エディ, ウィリアム・ローレンセン. オブジェクト指向方法論 OMT モデル化と設計. トッパン, 1992.

[San94a] Michael Sannella. Constraint satisfaction and debugging for interactive user interfaces. Technical report, University of Wasington, 1994.

[San94b] Michael Sannella. Skyblue: A multi-way local propagation constraint solver for user interface construction. In *Proceedings of the ACM Symposium on User Interface Software and Technology*, 1994.

[Sas93] 佐々政孝, 石塚治志, 中田育男. 1 パス型属性文法に基づくコンパイラ生成系 rie. コンピュータソフトウェア, Vol. 10, No. 3, pp. 20–36, May 1993.

[Shi03] Buntarou Shizuki, Hideto Yamada, Kazuhisa Iizuka, and Jiro Tanaka. A unified approach for interpreting handwritten strokes using constraint multiset grammars. In *2003 IEEE Symposium on Human-Centric Computing Languages and Environments*, pp. 180–182, 2003.

[Sil86] Abraham Silberschatz, Henry F. Korth, and S. Sudarshan. *Database System Concepts.* McGraw-Hill, Inc, 1986.

[Wit91] K. Wittenburg, L. Weitzman, and J. Talley. Unification-based grammars and tabular parsing for graphical languages. *Journal of Visual Languages and Computing*, Vol. 2, No. 4, pp. 347–370, 1991.

[Yam96] Kakuya Yamamoto. Visulan: A visual programming language for self-changing bitmap. In *Proceedings of International Conference on Visual Information Systems, Victoria University of Tech. cooperation with IEEE (Melbourne, Australia)*, pp. 88–96, 1996.

[Yam03] 山田英仁. ビジュアルシステム生成系における手書き入力と図形文法との統合. Master's thesis, 筑波大学大学院工学研究科, February 2003.

著者論文リスト

- 亀山裕亮, 志築文太郎, 田中二郎. 直接操作を用いたグラフィカルな図形文法編集システム. 日本ソフトウェア科学会第 20 回大会, 2003 年 9 月.
- 亀山裕亮, 飯塚和久, 志築文太郎, 田中二郎. GIGA:空間解析器生成系におけるグラフィカルな文法編集システム. 情報処理学会論文誌, Vol. 44, No. 11, pp. 2565–2574, Nov 2003.
- 飯塚和久, 亀山裕亮, 志築文太郎, 田中二郎. インクリメンタルな解析による空間解析器の高速化. 情報処理学会論文誌プログラミング, Vol. 44, No. SIG 13(PRO 18), pp. 100–109, 2003.
- 亀山裕亮, 飯塚和久, 志築文太郎, 田中二郎. GIGA: 空間解析器生成系におけるグラフィカルな文法編集システム. 第 10 回 インタラクティブシステムとソフトウェアに関するワークショップ (WISS2002) デモ発表, 2002 年 12 月.
- Hiroaki Kameyama, Kazuhisa Iizuka, Buntarou Shizuki, and Jiro Tanaka. GIGA: Graphical definiton of production rules in a spatial parser generator. In *Proceedings of the International Symposium on Future Software Technology (ISFST2001),Wuhan and Xian, China*, October 2002. (6 pages).
- 亀山裕亮, 中井央, 山下義行, 田中二郎. コンパイラのための意味解析器生成系. 日本ソフトウェア科学会第 18 回大会, 2001 年 9 月.
- Nakai, H., Sassa, M., Kameyama, H. and Nakata, I. Incremental Attribute Evaluation of LR-attributed Grammars Using Space-Efficient Data

Structure. In *Parigot, D. and Mernik, M. (ed.) Proc. WAGA '2000 - 3rd Workshop on Attribute Grammars and their Applications, Ponte de Lima, Portugal, INRIA*, pp. 99–116, July 2000.

- 亀山 裕亮, 中井 央, 山下 義行, 中田育男. 属性文法に基づいたインクリメンタルな Pascal-S コンパイラ. 情報処理学会第 55 回 (平成 9 年後期) 全国大会論文集 (分冊 1), 1997 年 9 月.