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

亀山裕亮[†] 飯塚和久[†]
志築文太郎[††] 田中二郎[††]

従来，空間解析器生成系に与える図形文法の定義にはテキストによる記述が用いられてきた．そのため，文法を定義する者にとって文法を定義することや，表している意味を理解することが困難であった．そこで本論文で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する．この手法では，図式表現を直接操作することによってルールを定義することが可能であり，さらに図式表現されたルールからその意味を容易に把握することが可能である．我々は，提案手法に基づき，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 を実装した．GIGA では，図形文法の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って構成要素の定義を行う．入力された構成要素間の関係から GIGA が制約を推論し図示する．制約の推論を行う際には，関連する図形の属性を重ね合わせるという操作によって，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している．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可能である．

GIGA: Graphical Definition of Visual Grammar in a Spatial Parser Generator

Hiroaki Kameyama,[†] Kazuhisa Iizuka,[†] Buntarou Shizuki[††] and Jiro Tanaka[††]

Conventionally, the definition of the grammar has been described with the textual representation. Therefore, it is difficult to define the grammar or to understand the meaning. In this paper, we propose an approach which graphicaly defines the grammar. In our approach, the direct manipulation is used to define rules and it helps users to understand the meaning. We have implemented the GIGA system. The GIGA sysytem makes users enable to define the components of the rule by drawing the corresponding figures. GIGA infers and shows the constraints from the positional relationships among the figures. GIGA omits inferring of the unnecessary results by overlapping the attributes of the components.

1. はじめに

通常のプログラミング言語の処理系では，パーサが入力されたテキストを文法に基づいて解析する．また，そのようなパーサをプログラミング言語の文法記述から自動的に作成する生成系として Yacc [11] や Rie [10] などが存在する．これに対し，ダイヤグラムエディタのように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（Spatial Parser Generator）と呼ぶ．空間解析器生成系としては，我々が開発を行ってきた恵比寿[9),15)~17)]や，Chok らの Penguins [1),2)] などがある．

従来，図形文法を定義するには，テキストによる記述が用いられてきた．しかし，図形文法では図形の位置関係のように 2 次元的な情報を表現する規則を扱うことが多いため，テキストによる文法記述では文法が表す意味を直感的に理解しつつ定義を行うことが困難であった．

そこで我々は図形文法をグラフィカルに編集するた

† 筑波大学大学院工学研究科
Doctoral Program in Engineering, University of Tsukuba
†† 筑波大学電子・情報工学系
Institute of Information Sciences and Electronics, University of Tsukuba

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 を実装した．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可能である．

本論文の構成は以下のとおりである．2 章では対象とする図形文法である拡張 CMG と，拡張 CMG を用いて作成されたビジュアルシステムの例を説明する．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説明し，4 章ではグラフィカルな文法編集システム GIGA について，図形文法の定義方法を例を交えて説明する．

2. 図形文法とビジュアルシステム

空間解析器生成系 Penguins は図形文法として CMG [12] を用いている．CMG は，図形が任意個の属性を持つことができる，記述することのできるクラスが広いなどの特徴を持つ．CMG を用いて記述されたビジュアルシステムの例として，Chok らはフローチャート，n 分木，数式などのエディタを報告している [2]．しかし，実際のビジュアルシステムにおいては図形間の解析を行うだけでは十分ではなく，描いた図形に対して解析結果に応じたフィードバックを行う必要がある．

そこで我々は，解析時に，図形の追加や削除，図形の属性値の変更などを行えるように，CMG にアクションを導入した拡張 CMG（Extended CMG）を提案した [16]．

以下に拡張 CMG のルールの構文を示す．

T ::= $T_1, \cdots, T_n$ where (
  Constraints
) {
  AttributeAssignments
} {
  Actions
}

T はルールが適用されたときに新しく作成される**図形単語**である．$T_1, \cdots, T_n$ はルールの**構成要素**である．構成要素 $T_1, \cdots, T_n$ は図形単語か基本図形である．図形単語は基本図形や図形単語自身の再帰的な組合せである（以降では基本図形と図形単語を合わせて，図形と呼ぶこととする）．基本図形には，円（circle）や長方形（rectangle），テキスト（text），直線（line）などがある．また，各基本図形は中心の座標や色など，あらかじめ定義された属性を持つ．*Constraints* は**制約**である．制約は，構成要素の属性の間の等式関係や不等式関係を組み合わせた条件式である．ルールが適用されるためには，構成要素が存在し，かつ制約が満たされる必要がある．*AttributeAssignments* は図形単語 T の**属性**を定める．*Actions* はこのルールが適用されるときに実行される**アクション**である．拡張 CMG で追加したアクションとは図形の生成（create），図形の削除（delete），図形の属性値の変更（alter）である．拡張 CMG ではルールを必要な数だけ定義することにより 1 つのビジュアルシステムを定義することができる．

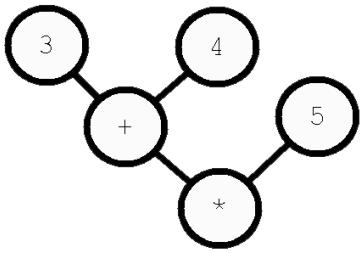


図 1 計算の木
Fig. 1 Calculation tree.

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 4) や 16) 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．

- **図 1** に示すような計算の木を編集し，実際に計算を行うビジュアルシステム（計算の木）
- 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（スタック）
- パイプの機能を持ったシェルを視覚化したビジュアルシェル（VSH）
- 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（GUI）
- ファイルやコマンドを図的に表現したアイコンを 2 つ以上重ね合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 [8] のサブセット（HI-VISUAL）
- 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 [7] のサブセット（VISPATCH）

これらのビジュアルシステムの拡張 CMG 記述に含まれるルールの数や制約の数などを**表 1** にあげる．

拡張 CMG の記述例として，計算の木の文法記述を**図 2** に示す．計算の木は 2 つのルールから成る．ルール 1 は，ノード（node）の構成要素は円とテキスト

表 1 拡張 CMG で記述されたビジュアルシステムの例
Table 1 Examples of visual systems specified with Extended CMG.

	ルール数	図形に関する制約数/総制約数	座標値を持つ属性数/総属性数	書き換えアクション数/総アクション数
計算の木	2	7/7 (100%)	6/8 (75%)	5/5 (100%)
スタック	4	5/5 (100%)	12/18 (67%)	2/6 (34%)
VSH	11	35/55 (64%)	10/29 (34%)	0/1 (0%)
GUI	14	72/85 (85%)	58/100 (58%)	1/1 (100%)
HI-VISUAL	15	23/43 (65%)	32/50 (64%)	4/14 (29%)
VISPATCH	24	112/134 (84%)	65/100 (65%)	3/5 (60%)

```
// ルール 1
Node ::= c:Circle, t:Text where(
  c.mid == t.mid
){
  right := c.rl_x
  left  := c.lu_x
  mid   := c.mid
  value := t.text
} {
}

// ルール 2
Node ::= n1:Node, n2:Node,
 l1:Line, l2:Line, c:Circle, t:Text
where(
  l1.start == c.mid &&
  l1.end   == n1.mid &&
  l2.start == c.mid &&
  l2_end   == n2.mid &&
  c.mid    == t.mid &&
  n1.right < n2.left
){
  left  := c.left
  right := c.right
  mid   := c.mid
  value := {script.integer
       {@n1.value@ @t.text@ @n2.value@}}
} {
  delete {@n1@ @n2@ @l1@ @l2@}
  alter @t@ text @value@
}
```

図 2 計算の木の拡張 CMG 記述
Fig. 2 Extended CMG specification of calculation tree.

であることを定義している（構成要素であるノード，円，テキストにはそれらを識別するために c や t のように識別子を付ける必要があり，この名前を使って構成要素の属性値を利用することができる）．また，制約として，円の中心とテキストの中心の座標が一致することを定義している．ノードには left，right，mid，value の 4 つの属性があり，各属性の値は円とテキストの属性を用いて決定する．

ルール 2 では，2 つのノード，2 つの直線，1 つの円，1 つのテキストが構成要素であり，これらの構成要素が 6 つの制約を満たしたときに，構成要素がノードに還元されることを定義している．解析時には left，right，mid，value の 4 つの属性の値を決定し，アクションを実行する．このルールではアクションとして，2 つのノードと 2 つの直線を削除するアクションと，テキストの属性値を 2 つのノードの値を用いた計算結果に変更するアクションを与えている．

3. 図形文法の定義における問題とアプローチ

図形文法の記述には，表 1 が示すように，図形の形状や位置関係などの 2 次元的な情報を与えるものが多い．そのような 2 次元的な情報をテキストのみを用いて記述する場合，以下にあげる問題が生じていた．

(1) 制約の把握・定義が困難である ルールが適用される条件の 1 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．計算の木という簡単な例でも，ルール 2 の適用条件の把握には，6 つの式から成る制約を読み，各構成要素間の関係を理解しなければならない．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．たとえば計算の木のルール 1 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 1 つ 1 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．

我々は，先行研究として恵比寿[9),15)~17)]やVIC[5)]を開発し，例示入力を用いて上記の問題のうち(1)と(3)について部分的に解決している．

(1)に対しては，恵比寿やVICでは基本図形を直接描画して構成要素を定義するため，構成要素を画面上で識別することが可能である．また，構成要素の基本図形を制約を満たすように配置して（すなわち，例示を行い），ルール生成ボタンを押すとシステムは基本図形間の位置関係から制約を推論し，ルールを自動生成する．ただし，既存の実装では構成要素の配置を変更するごとにルール生成ボタンを押す必要があり，インタラクティブ性がない．

(3)に対してはVICでは例示入力した基本図形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ことを提案している．ただし，具体的な実装までは行っていない．

しかしながら，例示を用いて制約の推論を行うと以下のような問題が生じる．

(4) 不必要な制約が生成される 恵比寿やVICでは制約の推論はすべての図形の組合せに対して行っているため，不必要な制約が多数生成され，ユーザはそれを取捨選択する必要がある．

そこで本論文では，(1)〜(4)の問題の解決を行うため，以下のアプローチを試みる．

- (1)に関しては，例示により文法の編集を行っている間，成立している制約を画面上に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．
- (2)に関しては，図形単語の属性の定義を，その構成要素の属性を直接指定することによって行うようにする．
- (3)に関しては，VICで試みた方式を踏襲し，例示入力した図形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方式を実際に実装する．
- (4)に関しては，関係する図形を明示的にユーザが関連づけすることによって，不必要な制約の推論を抑えるようにする．

4. グラフィカルな文法編集システムGIGA

図式表現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，Java言語（J2SE v1.4.1）を用いて実装を行った．Java言語を用いることによりマルチプラットフォームで実行することが可能である．システムのコードの量はおよそ5,000行である．システムの構成としては，図形の編集を行う部分，入力された図形からルールの推論を行う部分，推論したルールを整形しファイルへと出力する部分に分かれている．

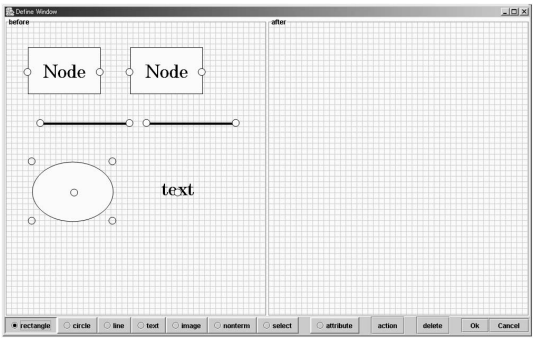


図3 GIGAの図形文法定義インタフェース
Fig. 3 Definition window of GIGA.

GIGAにおける文法定義のための定義インタフェースを**図3**に示す．定義インタェースの左側が拡張CMGの構成要素と制約を定義するためのものであり，右側が属性とアクションを定義するためのものである．定義インタフェースの下にあるツールパネルには各種のボタンが配置されている．

以降では，計算の木の例を用いて，拡張CMGの各要素をGIGAを用いてどのように定義するかを解説する．

4.1 構成要素の編集

図形単語の定義は構成要素となる基本図形を直接定義インタフェースに描くことにより行う．構成要素となる基本図形はツールパネルに用意されたボタンの中から，必要なものを選択することにより入力する．一般の図形エディタと同様に，構成要素となる基本図形に対し，移動，サイズの変更，削除といった編集操作を行うことができる．このとき，各図形には一意な識別子が自動的に与えられる．

構成要素として図形単語を選択することもできる．ツールパネルにある`nonterm`と書かれたボタンを押すと，すでに定義を行った図形単語の一覧が表示されるので，その中から必要なものを選択する．

基本図形には多数の属性がある．図形文法の定義を行う際には，それらの属性から使用したいものを選択する必要がある．基本図形が持つ属性の中で，長方形の中心や4つの頂点の座標のように値が座標値である属性は，基本図形上の対応する位置に半透明の**小さな**

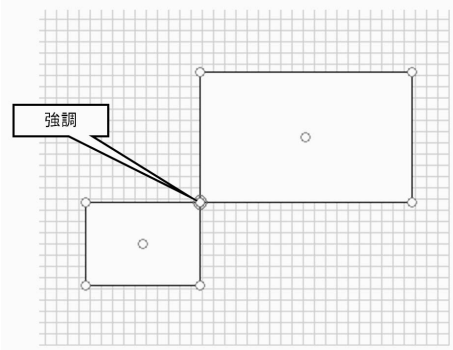


図 4 eq 制約の定義 1
Fig. 4 Definition of eq constraint 1.

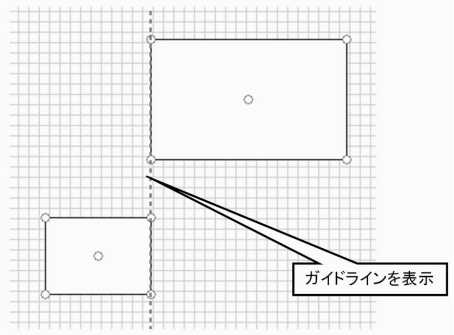


図 5 eq 制約の定義 2
Fig. 5 Definition of eq constraint 2.

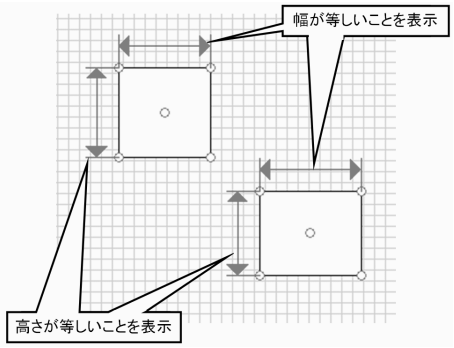


図 6 eq 制約の定義 3
Fig. 6 Definition of eq constraint 3.

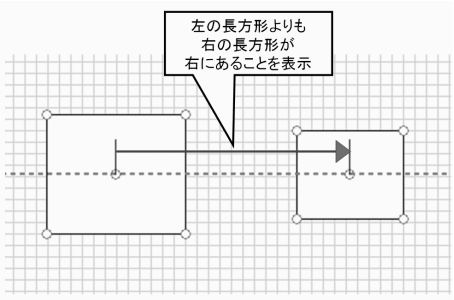


図 7 gt 制約の定義
Fig. 7 Definition of gt constraint.

丸で表示される．その中から選択したい属性をマウスでクリックすることにより，選択された小さな丸が半透明ではなくなり，属性の選択を行うことができる．色などのそれ以外の属性については，基本図形の属性をデフォルトの値から変更した場合にのみ，属性が選択される．

たとえば，計算の木のルール 2 の場合，ノードは円とテキスト，2 つの直線，2 つのノードで構成されているので，このルールの定義を行うためには図 3 に示すように定義インタフェースの左側に対応する基本図形を描く．次に，直線の両端や円の中心の属性を制約の定義に使用するために，対応する位置にある小さな丸をクリックし属性を選択する．なお，この段階では構成要素についてのみ定義をしており，制約についてはまだ定義をしていないため，各図形の位置に意味はない．

4.2 制約の編集

構成要素として定義インタフェースの左側に描画した図形を操作し，定義したいルールを例示することにより制約の定義を行う．このとき，定義している制約を直感的に理解することができるよう，以下のように制約を図示する．

(1) 属性の中で座標値を持つ属性については，選択された属性の位置が他の図形の属性の位置と**完全に一致**する場合には**図 4** のように，一致している属性の位置にある小さな丸を強調表示する．

(2) 選択された属性の位置が，他の図形の属性の位置と **X 座標（もしくは Y 座標）のみ一致**している場合には**図 5** に示すように，ガイドライン（太い点線）により図示する．

(3) 図形の大きさを表す属性値が，他の図形の大きさの属性値と完全に一致している場合には，**図 6** のように大きさの一致している部分を矢印を用いて強調表示する．

(4) ある図形が他の図形よりもつねに右側の位置に存在するといった図形間の相対的な位置に関する制約については，**図 7** に示すように矢印を用いて図形がどの方向にあるかを表示する．

制約の推論はインタラクティブに行われており，制約が成立しなくなった時点で，その制約の図示が行われなくなる．これにより，編集している最中に，どのような制約が定義されているかが把握できるようになっている．

なお，(1) でたとえば図形の位置が一致するという

制約を作成するためには，それぞれの図形をまったく同じ位置に移動しなければならないため，操作が困難になってしまう．そこで GIGA ではあるしきい値よりも近い位置にあるなど，成立する可能性のある制約が存在する場合には，図形どうしをスナップし座標を自動的に一致させる．

また，(2)，(3)，(4) の制約を推論する際に，すべての図形の関係から推論を行うと不必要な制約が多数生成され，図示されるという問題が生じる．これに対し GIGA では，ユーザにより明示的に図形の属性の**関連付け**を行わせ，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，図形を移動し別の図形と属性どうしを重ね合わせることにより行うことができる．一度関連付けを行うと，図形を移動し属性どうしが離れても関連付けは保持される．

たとえば，計算の木のルール 2 の制約は，GIGA を用いて次のように定義することができる．直線の始点の座標と円の中心の座標が同じであるという制約を定義するために，直線の始点を円の中心に移動する．移動は直線の始点と円の中心の座標が一致し強調表示されるまで行う．次に，同じように直線の終点をノードの中心に，一致した点が図示されるまで移動する．もう 1 つの直線についても同様に始点と終点を移動する．円とテキストの中心の座標が同じであるという制約を定義するために，テキストの中心の座標と円の中心の座標が同じ位置になるように，テキストを移動する．最後に，ノード 1 の属性 right を表す小さな丸とノード 2 の属性 left を表す小さな丸を重ね合わせることにより 2 つの属性を関連付けした後，ノード 2 をノード 1 よりも右側に移動することによって，ノード 2 はノード 1 よりも右側にあるという制約を定義する．すべての制約の定義が終わった後の図を**図 8** に示す．

4.3 属性の編集

構成要素である図形の座標値を図形単語の属性として定義する場合には，定義インタフェースの右側に属性を表す円を入力し，それを図形の座標の上に置くことにより属性の定義を行う．また，座標値以外の値を持つ属性を定義する場合には，属性を表す円にテキストで属性の計算式を入力することにより定義を行うことができる．

計算の木のルール 1 では right，left，mid，value という 4 つの属性を定義している．

属性 right の値は構成要素である円の右上の頂点の X 座標として定義する．まず属性 right を表す図形を入力する．次に，その図形を移動し円の右上の頂点を表す小さな丸と X 座標を揃えることにより，属性 right を定義することができる．属性 left についても同様に定義を行う．属性 mid の値は構成要素である円の中心の座標として定義する．属性 value の値は構成要素であるテキストの文字列として定義する．まず属性を表す図形を入力する．次に，その図形に代入式をテキストで記述することにより，属性 value を定義することができる．具体的には，テキストの識別子が `t` でありテキストの文字列を表す属性名が `text` であるので，`value := t.text` と記述する．ルール 1 のすべての属性の定義を終えた状態を**図 9** に示す．

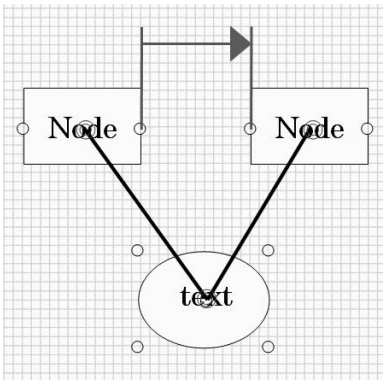


図 8 計算の木における制約の定義
Fig. 8 Definition of calculation tree constraints.

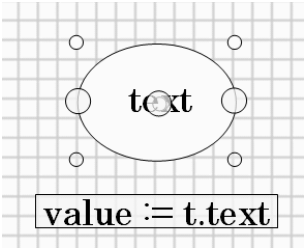


図 9 属性の定義
Fig. 9 Definitifon of attributes.

4.4 アクションの定義

ルールから作成されたビジュアルシステムでは，ルールを満たす図形が入力されると，それらの図形が 1 つの図形単語として認識され，同時にルールで定義されたアクションが実行される．

GIGA では定義インタフェースの左側で，構成要素，制約について定義を行うと，作成した図形がそのまま定義インタフェースの右側に複製される．この図を編集し，アクションが実行された後の図にすることによって，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．

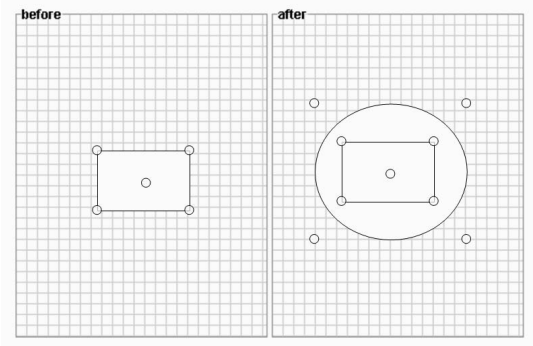


図 10 create アクションの定義
Fig. 10 Definition of create action.

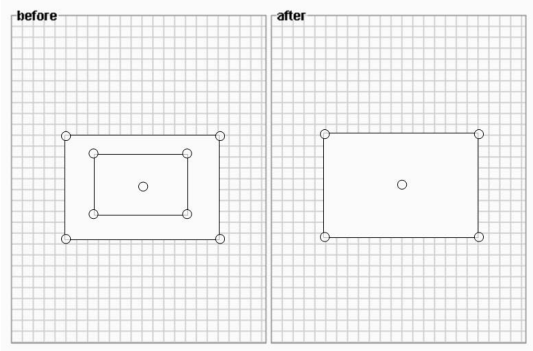


図 11 delete アクションの定義
Fig. 11 Definition of delete action.

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．たとえば，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 10**）．
- 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．たとえば，2 つの長方形が重なったとき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 11**）．
- 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．たとえば，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 12**）．

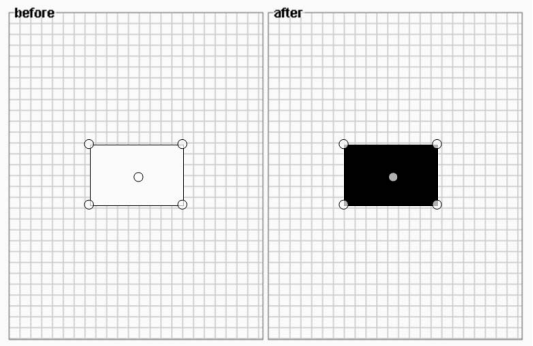


図 12 alter アクションの定義
Fig. 12 Definition of alter action.

たとえば，計算の木のルール 2 では 2 つのアクションが定義されている．

最初のアクションは計算の木から 2 つのノードと 2 つの直線を削除するアクションであり，以下のように記述されている．

```
delete {@n1@ @n2@ @l1@ @l2@}
```

delete アクションの引数となる構成要素の指定には，各構成要素に付けた識別子の前後に`@`を付けたものを用いる（n1 と n2 が 2 つのノード，l1 と l2 が 2 つの直線を表している）．このアクションを GIGA を用いて定義するには，定義インタフェースの右側でそれらの図形を削除すればよい．

もう 1 つのアクションはテキストの属性を計算により求めた値に変更する alter アクションであり，以下のように記述されている．

```
alter @t@ text @value@
```

alter アクションの第 1 引数では構成要素の識別子，第 2 引数では基本図形の属性名，第 3 引数では変更後の値をそれぞれ指定する．計算の木のルール 2 では，alter アクションにより第 1 引数に指定された基本図形 `t`（テキスト）の属性 `text`（テキストの文字列）の値が，変数 `value` に変更される．このアクションを GIGA を用いて定義するには，まず定義インタフェースの右側で構成要素であるテキストを選択し，その値を`@value@`に変更する．

構成要素，制約，属性，アクションの定義を行い，計算の木の 2 つの目のルールについて定義が終わった後の図を**図 13** に示す．

GIGA では定義を終了した文法記述を拡張 CMG の形式でファイルに出力することができる．出力された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により，定義を行ったビジュアルシステムの実行を

行うことができる．

計算の木を恵比寿上で実行した様子を**図 14** に示す．入力された計算の木 $(3+4)*5$ に対し，まず初めに $(3+4)$ 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．

5. 議　　論

グラフィカルな文法編集システム 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にくさ，といった問題について改善することができた．表 1 に示すように，多くのビジュアルシステムのルールでは，図形の位置を用いた式が制約や属性に多用される．アクションについても図形の書き換えに関する定義が多い．したがって，多くの場合において GIGA のグラフィカルな定義インタフェースは図形文法の定義において効果的である．

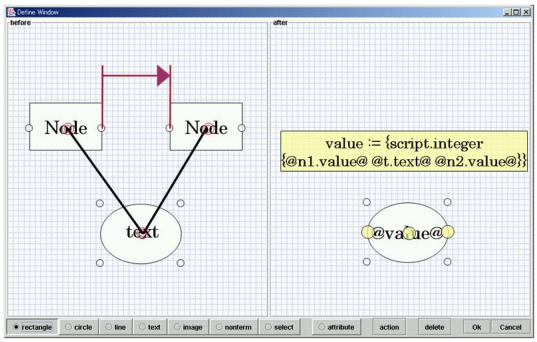


図 13　計算の木の定義
Fig. 13　Definition of calculation tree.

制約に関する関連研究として Briar [6] と Pegasus [14] 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したりすることにより，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たりす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，図形文法のルールを入力・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どうしに関連付けを行い，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 KIDSIM [3] や，Visulan [13]，VISPATCH [7] 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 つのルールと見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

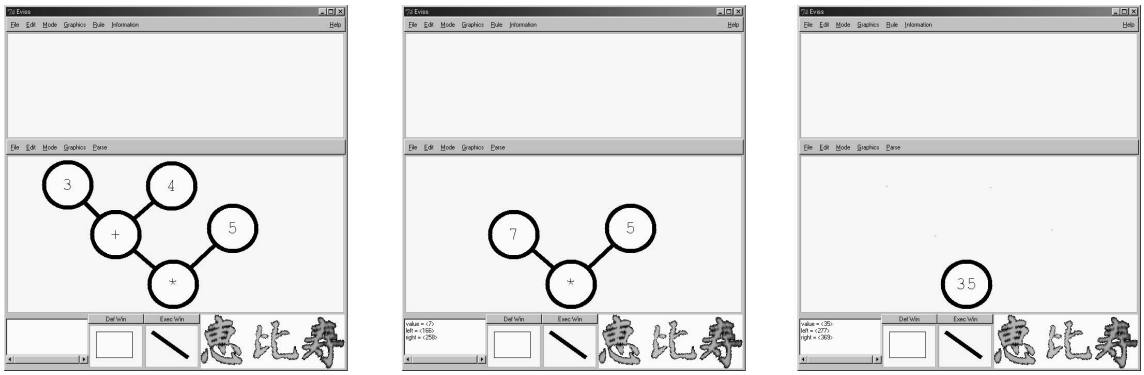


図 14　計算の木の実行
Fig. 14　Execution of calculation tree.

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 ではルール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ルールから構成される．この編集作業において，編集対象となるルール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ルールの意味を大つかみに把握し，探し出すルールに見えるものが見つかったときに初めて細かな部分を読めばよいからである．

6. ま と め

本論文では，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．

提案手法では，ルール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，ルール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ルール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，ルールを構成する**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，制約が意図したとお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．

参 考 文 献

1) Chok, S.S. and Marriott, K.: Automatic Construction of User Interfaces from Constraint Multiset Grammars, *Proc. IEEE Symposium on Visual Language*, pp.242–249 (1995).
2) Chok, S.S. and Marriott, K.: Automatic Construction of Intelligent Diagram Editors, *Proc. ACM Symposium on User Interface Software and Technology*, pp.185–194 (1998).
3) Cypher, A. and Smith, D.C.: KIDSIM: End User Programing of Simulations, *Proc. ACM Conference on Human Factors in Computing Systems* (*CHI'95*), pp.27–34 (1995).
4) 藤山健一郎：例示入力図を用いた Spatial Parser Generator，修士論文，筑波大学大学院工学研究科 (2001).
5) Fujiyama, K., Iizuka, K. and Tanaka, J.: VIC: CMG Input System Using Example Figures, Nanjing, China, *Proc. International Symposium on Future Software Technology* (*ISFST '99*), pp.67–72 (1999).
6) Gleicher, M. and Witkin, A.: DRAWING WITH CONSTRAINTS, *The Visual Computer*, Vol.11, No.1, pp.39–51 (1994).
7) Harada, Y., Miyamoto, K. and Onai, R.: VISPATCH: Graphical rule-based language controlled by user event, *IEEE Symposium on Visual Language* (1997).
8) Hirakawa, M., Nishimura, Y., Kado, M. and Ichikawa, T.: Interpretation of Icon Overlapping in Iconic Programming, *Proc. 1991 IEEE Workshop on Visual Languages*, pp.254–259 (1991).
9) Iizuka, K., Tanaka, J. and Shizuki, B.: Describing a Drawing Editor by Using Constraint Multiset Grammars, Zhen Zhou, China, *Proc. International Symposium on Future Software Technology* (*ISFST2001*), pp.119–124 (2001).

10) 石塚治志，佐々政孝，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 (1993).
11) Johnson, S.C.: Yacc: Yet Another Compiler-Compiler, *UNIX Programmer's Manual, 7th Edition*, Vol.2, pp.353–387 (1979).
12) Marriott, K.: Constraint Multiset Grammars, *Proc. IEEE Symposium on Visual Languages*, pp.118–125 (1994).
13) Yamamoto, K.: Visulan: A Visual Programming Language for Self-Changing Bitmap, *Proc. International Conference on Visual Information Systems*, Melbourne, Australia, pp.88–96, Victoria University of Tech. (in cooperation with IEEE) (1996).
14) 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 (1998).
15) 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 (1998).
16) 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).
17) 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（掲載予定）(2003).

(平成 15 年 4 月 21 日受付)
(平成 15 年 9 月 5 日採録)

亀山　裕亮（学生会員）

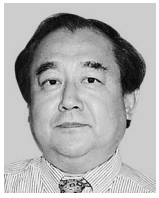
1975 年生．1998 年筑波大学第三学群工学システム学類卒業．同年同大学院工学研究科博士課程入学．プログラミング言語や図形言語の処理系に関する興味を持つ．日本ソフトウェア科学会会員．

飯塚　和久（学生会員）

1999 年筑波大学大学院工学研究科修士課程理修了．現在，同大学院工学研究科博士課程在学中．図形言語の処理系，空間解析器とその応用に興味を持つ．日本ソフトウェア科学会，ACM 各会員．

志築文太郎（正会員）

1971 年生．1994 年東京工業大学理学部情報科学科卒業．2000 年同大学院情報理工学研究科数理・計算科学専攻博士課程単位取得退学．博士（理学）．現在，筑波大学電子・情報工学系講師．ヒューマンインタフェースに関する研究に興味を持つ．日本ソフトウェア科学会，ACM，IEEE Computer Society，電子情報通信学会，ヒューマンインタフェース学会各会員．



田中　二郎（正会員）

1975 年東京大学理学部卒業．1977 年同大学院理学系研究科修士課程修了．1984 年米国ユタ大学計算機科学科博士課程修了，Ph.D. in Computer Science．1984 年から（財）新世代コンピュータ技術開発機構で第五世代コンピュータ核言語の研究開発に従事．1993 年より筑波大学に勤務．現在，電子・情報工学系教授．主としてヒューマンインタフェースおよびユビキタスコンピューティング関連の研究に従事．2001 年 4 月から筑波大学学際領域研究センター（TARA）マルチメディア情報研究アスペクトで「実世界指向インタラクションの研究」の研究代表者をつとめている．2002 年 4 月から筑波大学第三学群情報学類長．現在，ACM 日本支部で CACM 日本語版編集長．ヒューマンインタフェース学会評議員．日本ソフトウェア科学会学会誌編集委員．IEEE Computer Society，電子情報通信学会，計測自動制御学会，人工知能学会各会員．