例示入力図を用いた Spatial Parser Generator

Spaital Parser Generator using Example Figures

藤山健一郎† 田中二郎††
Kenichiro FUJIYAMA Jiro TANAKA

†筑波大学 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 Science and Electronics, University of Tsukuba

概 要

我々は、例示入力図を用いたCMG入力インタフェイスを持つ Spatal Parser Generator VICを作成した。VICでは拡張CMGを用いて図形言語の定義を行なうが、テキストによる記述を行なう事なく、例示入力図に対する操作によって図形言語を定義する。したがってユーザは拡張CMGに対する十分な知識がなくても直感的にSpatial Parserを記述、生成できる。また有効性を実証するために、実際にビジュアルシステム「計算の木」を例に上げ、その生成規則を恵比寿とVICで定義し、作業時間を比較した。その結果、作業効率が約２倍になることが確認できた。

1 はじめに

ビジュアルプログラミングシステムの研究 [1] では対象となるものが図形を用いた言語（図形言語）であるため、実装する際に図形言語の解析をおこなうSpatial Parserが必要となる。しかし個々のシステム毎に Spatial Parser を実装するのは困難で時間のかかる仕事である。そこで図形言語の文法を定義することにより、この Spatial Parser を自動生成し、様々な図形言語を解析し、さらに実行することができるビジュアルシステム「恵比寿」 [2] が開発されている。

図形言語を定義することは図形間の関係を記述することであり、２次元上の図形的な情報を扱う。しかしながら恵比寿等の既存のシステムでは１次元的なテキストで入力するため直感的に理解しにくいという欠点があった。またテキストで記述するため、メタ文法である拡張CMGを十分知っていなくては図形文法を定義できない。そこで我々は図形言語の定義をテキストではなく、図形を用いて視覚的に、かつ拡張CMGを意識することなく定義することにより、これらの欠点を解消できると考えた。

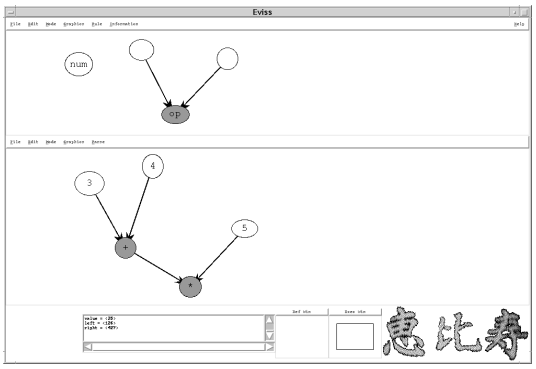

図１ 恵比寿の実行画面

2 恵比寿における文法

2.1 拡張されたCMG

ビジュアルシステム「恵比寿」（図1）は、図形言語の文法を定義することによりSpatial Parser を生成し、これを用いて、描画された図形を解析し、処理を行うことができる。

「恵比寿」では図形言語の定義をするために拡張CMG [2] を用いる。拡張CMGとは、従来のCMG

[3] をベースに、生成規則が適用されたときに図形の書き換えなどのactionを記述できるようにしたものである。拡張されたCMGの生成規則は以下のようになる。

$$
\begin{aligned}
T(\vec{x}) \quad ::= \quad & T_1(\vec{x_1}), \cdots, T_n(\vec{x_n})\ where \\
& exists\ T'_1(\vec{x'_1}), \cdots, T'_m(\vec{x'_m}) \\
& not\ exists\ T''_1(\vec{x''_1}), \cdots, T''_l(\vec{x''_l}) \\
& all\ T'''_1(\vec{x'''_1}), \cdots, T'''_k(\vec{x'''_k}) \\
& where\ C\ and\ \vec{x} = F\ and\ A
\end{aligned}
$$

ここで $T(\vec{x})$ は生成される図形単語である。また $T_1(\vec{x_1}), ..., T_n(\vec{x_n})$ は n 個の図形単語、もしくは図形文字であり、normalの構成要素と呼ぶ。これは実際の非終端記号を構成する部品となる。$T'_1(\vec{x'_1}), ..., T'_m(\vec{x'_m})$ も m 個の図形単語、もしくは図形文字であり、existの構成要素と呼ぶ。このトークンが存在しなければ生成規則は適用されない。$T''_1(\vec{x''_1}), ..., T''_l(\vec{x''_l})$ もまた l 個の図形単語、もしくは図形文字であり、not_existの構成要素と呼ぶ。このトークンのどれかが1つでも存在するとき、生成規則は適用されない。$T'''_1(\vec{x'''_1}), ..., T'''_k(\vec{x'''_k})$ は k 個の図形単語、もしくは図形文字であり、allの構成要素と呼ぶ。これは図形文の中にある指定された同一種類の複数のトークンを指定する際に使用する。C は属性 $\vec{x_1}, \cdots, \vec{x_n}$、$\vec{x'_1}, \cdots, \vec{x'_m}$、$\vec{x'''_1}, \cdots, \vec{x'''_l}$ に関する制約である。恵比寿における制約とは、2つの属性間、もしくは1つの属性と定数の間になんらかの条件を課すことである。Fは属性 $\vec{x_1}, \cdots, \vec{x_n}$、$\vec{x'_1}, \cdots, \vec{x'_m}$、$\vec{x'''_1}, \cdots, \vec{x'''_l}$ を引数とする関数であり、生成される非終端記号$T(\vec{x})$ の属性$\vec{x}$ を定義している。A はアクションを記述したものである。アクションとは「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」と定義される。

2.2 従来の文法の定義手法

これまでの「恵比寿」で生成規則を定義するには、まず図形を用いて大まかな文法と構成要素を与える。以降この図形を例示入力図と呼ぶ。すると恵比寿がその例示入力図から制約をある程度自動生成してCMG入力部と呼ばれるウィンドウにテキストで出力する。そしてこれを編集するというものであった（図2）。

しかしこの手法では、最初の入力で図形を利用するものの、その後の編集ではこの図は利用されず、

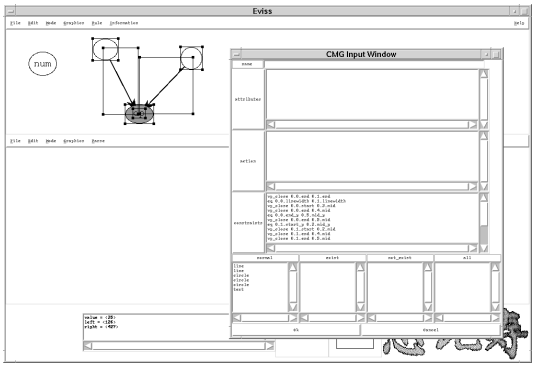

図2　恵比寿のCMG入力法

実際にはほとんどテキストによる編集となっていた。また制約を自動生成するが、実際にはユーザの意図に沿わないことが多い。その上、制約の対象となる構成要素の示すIDが、構成要素の種類と順番に基づいたものであるため、構成要素の種類を1つでも変更しただけで、CMGの大部分が無意味となってしまう。結局白紙からテキストで書き起こすのとほとんど変わらない努力を要するという欠点があった。

3　システム VIC

我々は例示入力図を利用したCMG入力インタフェイスを恵比寿上に実装したシステムVIC [4] を作成した。VICの実行画面は図3のようになる。

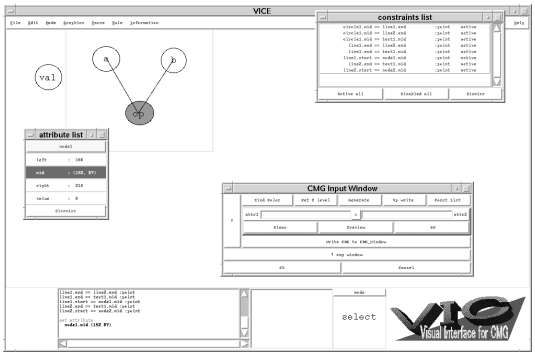

図3　実行画面

VICでは、生成規則の概要を与えるに過ぎなかった例示入力図をより活用し、テキスト編集をすること無く、例示入力図に対する操作で文法を定義できる。そのため拡張CMGにおける文法をほとんど意識する必要はない。したがって拡張CMGに対する十分な知識がないユーザでも、直感的に図形言語を定義できる。また、識別IDを管理する必要もなくなる。

4 VICにおけるCMGの入力方法

CMGの生成規則を定義するには

- 生成されるトークンの名前 （name）
- 属性 （attributes）
- アクション （action）
- 制約 （constraints）
- 構成要素 （normal, exist, not_exist, all）

を定義する必要がある。

ここではビジュアルシステムの例として「計算の木」をとりあげ、その生成規則２を定義する過程を通して、VICの機能、特に制約と構成要素の入力について説明をする。

4.1 ビジュアルシステムの例

図1は数式を視覚的に表現したビジュアルシステム「計算の木」の定義している場面である。「計算の木」とは２つのノードの値を引数とし、それらのノードが結線された円の演算子を作用させた結果を返すビジュアルシステムである。「計算の木」は以下の２つの生成規則により再帰的に定義される。

生成規則１ node は circle の中心に textが描いてあるもの。

生成規則２ node は１つの circle にnode1、node2 がそれぞれline1、line2 によって繋がれているもの。ただしnode1はnode2より左にある。

4.2 構成要素の入力

一つの生成規則を定義する際、その構成要素となる部品についての情報がまず必要となる。どのようなタイプの構成要素が必要となるかは例示的に入力、すなわちキャンバスに実際に描画することによって指定する。ただしここで問題となるのが、構成要素が終端記号だけでなく、他の生成規則によって定義された非終端記号—この場合node—が含まれる場合である。

例えば計算の木の生成規則１が定義してある状態で生成規則２を定義する際、非終端記号であるnodeを例示的に入力するのは、キャンバス上にcircleとtextを描く。しかし恵比寿ではこれを単純に終端記号のcircleとtextと認識してしまい、後で改めてnodeと手動で書き直さなくてはならない。これでは構成要素中に非終端記号が占める割合が増加するにつれ、その修正の手間も比例して増えるので、大規模なビジュアルシステムを構築することは困難となる。そこでVICでは他の生成規則を適用して非終端記号を直接認識する。この例の場合では生成規則１を適用してcircleとtextを１つのnodeという非終端記号として認識できる。描かれた図形を１つの意味のある非終端記号として認識するためには、他の生成規則を利用してその図形が一度解析される、すなわちSpatial Parsingが行われる必要がある。

4.3 構成要素の種類の入力

構成要素は入力された時点での種類はすべてnormalとなっている。これを必要に応じ変更するが、恵比寿ではテキストを書換えることによって行っていた。一方VICでは例示入力図より変更すべき構成要素を選択し、種類を変更する。この手法は、例示入力図に対して操作を行うことにより、その構成要素の図形間の位置関係を確認したまま変更作業が行えるという利点がある。また構成要素名を入力する必要がないので、より容易に変更が可能である。さらに色を用いて構成要素の種類を表示してあるので、その種類を変更するとその結果として描かれた構成要素の色が変わるというフィードバックが得られるので、よりインタラクティブに作業ができる。

4.4 制約の入力

VICでは、まず例示入力図から、制約となる構成要素間の関係を合う程度推測して制約を自動的に生成する。ユーザは生成された制約のうち不必要な制約を削除したり、足りない制約を追加したりして生成規則を完成させる。恵比寿のように制約をテキストで記述する場合、拡張CMGについて正確な知識を持ち合わせていないと記述できず、また知っていても構文エラーを犯す場合がある。そこでVICではテキストを用いずに制約の編集を行う。

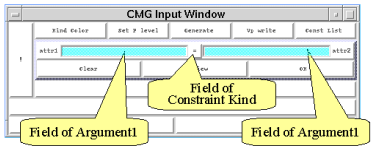

図４　制約生成ウィンドウ

拡張CMGで記述する制約とは、２つの属性間、

もしくは１つの属性と定数の間になんらかの条件を課すことである。VICでは図4のような制約生成ウィンドウを設け、その各フィールドに制約の種類、属性名、定数を入力することによって制約を生成する。

計算の木の生成規則２の「node1がnode2より左にある」という制約を定義する場合を例としてあげる。

まず制約の種類は制約種類入力フィールドのメニューから選ぶ。この場合「より左にある」（小さい）という制約を選択する。次に構成要素の属性名の入力方法であるが、順番としては、まず構成要素を選んでから、その構成要素中にある属性名を選択するというのが自然であり、直感的に理解しやすい。まず構成要素は例示入力図より選択する。構成要素を指定してダブルクリックすると、属性名のリストが現れるので、属性はその中から選ぶ。そのリストから選択した属性名を制約生成ウィンドウのArgフィールド入力する。図5は node2 の座標を示す属性を入力している。

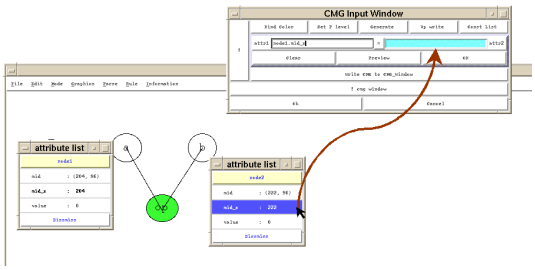

図５　制約生成ウィンドウ（属性名の入力）

5　評価

VICの評価をするために、恵比寿と比較実験を行った。実験の手法は、被験者に恵比寿、VICのそれぞれで実際に「計算の木」の生成規則２の構成要素、制約の定義をしてもらい、その作業時間を比較する。作業時間が短い程、より使いやすいシステムであると考えられる。被験者はA～Cが恵比寿は使い慣れているがVICは初めてという者、D～Fが恵比寿もVICも初めて扱う者、Gは恵比寿もVICも使い慣れた者である。被験者にはそれぞれ生成規則２の概要を示したメモを手渡し、既に生成規則１が定義された状態で定義を行ってもらった。その作業時間を図6に示す

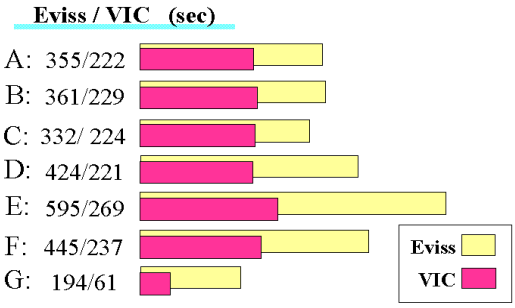

図６　結果

被験者A～Cの結果をみると、どの被験者も慣れている恵比寿より、初めて扱うVICで定義した方が、半分程度の時間で定義できている。被験者D～Fの結果でも、やはりVICのほうが作業時間が短くなっている。被験者GのようにVICの扱いに慣れた者は、作業時間を $\frac{1}{3}$ 以下にまで短縮している。

いずれにせよ、VICのほうが作業効率が２倍ほど高くなっていることが分かる。

6　結論

本論文では例示入力図を用いたCMG入力編集インタフェイスをもつ Spatial Parser Generator VIC を作成した。VICではテキストによる記述を行わず、例示入力図に対する操作によって図形言語を定義することができる。その結果、拡張CMGに対する十分な知識が無くても、直感的に制約の記述を行うことを可能とした。そして、その有効性を実証するために、実際にビジュアルシステム「計算の木」の生成規則を恵比寿とVICで定義し、その作業時間を比較した。その結果、VICを使用した方が作業効率が２倍近く高まっており、VICの有効性を示すことが出来た。

参考文献

[1] Shu, N. C. : Visual Programming. *Van Nostrand Reinhold,* 1988.

[2] 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, 1998.

[3] Kim Marriot : Constraint Multiset Grammars. In *Proceedings of the 1994 IEEE Symposium on Visual Languages,* pages 118-125, 1994.

[4] Kenichiro Fujiyama, Kazuhisa Iizuka and Jiro Tanaka : VIC:CMG Input System using Example Figures. In *Proceedings of ISFST'99,* pages 67-72, 1999.