データベースシステム 115－14
情 報 学 基 礎 49－14
（1998. 5. 15）

多次元言語パーザ自動生成システムを用いたグラフィカル問い合わせ言語インタプリタの実現

畠山竜輔　宝珍輝尚　都司達夫
福井大学工学部情報工学科
〒910-8507 福井県福井市文京3-9-1
{rhata, hochin, tsuji}@pear.fuis.fukui-u.ac.jp

従来、グラフィカル問い合わせ言語DUOのインタプリタをプログラミング言語を用いて直接製作してきた。しかし、限られた文法しかサポートしておらず、様々な要素間の空間配置の考慮は困難であった。そこで、本論文では、多次元言語パーザ自動生成システムNAEを用いたDUOインタプリタの実現について報告する。NAEは多次元言語の定義から自動的に多次元言語のパーザを生成するシステムである。NAEを用いることにより、複雑な空間規則を規則として記述でき、開発が容易になることを示す。

Implementation of a Graphical Query Language Interpreter Using a Multi-Dimensional Language Parser Generator

Ryusuke HATAKEYAMA　Teruhisa HOCHIN　Tatsuo TSUJI
Department of Information Science, Faculty of Engineering, Fukui University
3-9-1 Bunkyo, Fukui-Shi Fukui 910-8507 JAPAN
{rhata, hochin, tsuji}@pear.fuis.fukui-u.ac.jp

The interpreter for the graphical query language named DUO has been implemented in a programming language. This interpreter supports the only limited set of the DUO grammers. Considering various spatial relationships among various elements is very difficult. This paper describes the experience of implementing the DUO interpreter by using a paser generator for multi-dimensional language. This parser generator named NAE receives the definitions for a multi-dimensional language, and generates its parser. This paper clarifies that the implementation becomes easier because complex spatial relationships can be described with rules.

1 はじめに

現在までにソフトウェア構築のために様々なプログラミング言語が開発され、ソフトウェアの発展に貢献してきた。従来のプログラミング言語は、様々な文字列を記述することによりプログラミングを行う。このような言語を文字列言語とよぶことにする。文字列言語は我々が普段用いる自然言語とは異なり、プログラミング特有の独特な文字列 記号の列挙であり、使用される文字列も英語やその短縮形であることが多い。また、プログラミング言語特有の独特の構文やアルゴリズムを修得しなければならない。そのためプログラミング初心者などにとっては、修得が困難であるという問題がある。また、文字列を線型に記述しているため、複雑な構造 (3次元配列など次元数の高い配列、線型リストや木) を持つデータの扱いが難しく、読解も非常に困難になっている。

これらの問題点を解決するものとしてヴィジュアル言語が提案されている [1]。ヴィジュアル言語とは、2次元平面上に様々な図形や文字列を配置することによってプログラミングを行う言語である。ヴィジュアル言語を用いることにより、視覚的にプログラミングを行うことができる。また、修得、読解が比較的容易であり、複雑なデータを容易に記述することもできる。

このようなヴィジュアル言語のひとつとして、グラフィカル問い合わせ言語 DUO が提案されており [3]、そのインタプリタを一昨年より開発してきている [4]。ヴィジュアル言語処理系の実装には、グラフィカルなユーザインタフェースの実現や、グラフィカルな記号の空間関係を考慮した構文解析を行う必要があるという文字列言語処理系にはない問題点が存在する。DUO インタプリタも、この点が問題になっており、解釈できる文法が限定されている。

一方、著者らは多次元言語パーザ自動生成システム NAE を提案、開発してきている [2]。NAE は、多次元言語の定義から自動的に多次元言語のパーザを生成するシステムである。NAE では、記号の型定義、記号間の関係の定義、並びに、生成規則の定義を与えると、これらに基づく多次元言語パーザの C++ソースプログラムを自動的に生成する。NAE を用いることにより DUO の文法を完全にサポートできる可能性がある。

そこで本研究では、DUO 文法を完全にサポート可能なインタプリタの実現を目的として、DUO インタプリタを多次元言語パーザ自動生成システム NAE を用いて製作する。DUO 用の記号の型定義、記号の関係の定義、並びに、生成規則の定義を行い、これにより生成されたパーザを DUO インタプリタに組み込む。また、従来の DUO インタプリタのインタフェースに不足していた機能を追加し、使い勝手の向上も図る。NAE を利用することにより、行数的には従来と同程度の記述量で従来以上の DUO 文法をサポートできることを示す。これにより DUO 文法の完全サポートが容易に可能なインタプリタを作成する。

以下、2 ではグラフィカル問い合わせ言語 DUO と従来の DUO インタプリタについて概説する。3 では多次元言語パーザ自動生成システム NAE について述べる。4 では実際に NAE を用いた DUO インタプリタの実現について述べる。5 では製作した DUO インタプリタの評価を行う。

2 DUO と DUO インタプリタ

2.1 グラフィカル問い合わせ言語 DUO

問い合わせ言語 DUO の問い合わせ対象とするデータはラベル付き有向グラフで表されるデータである。データの例を図 1 に示す。図 1 では、都市を表す点 city とある都市の下りにある隣接した都市の関係を表す有向枝 down によりデータが表現されている。

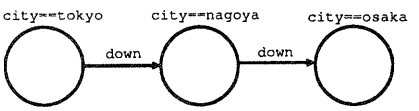

図 1: DUO の問い合わせ対象の例

DUO の問い合わせの基本は問い合わせグラフである。問い合わせグラフは、要素 (点と枝) と括弧で表現される。例えば、“都市の連続データから ‘nagoya’ の下り方面に存在する都市” を求める場合、図 2 のように記述する。

図 2 では、nagoya という city から 1 以上の arc によって連結された city を求めている。図中の “+” は括弧 (()) で 囲まれたパスが 1 以上繰り返されることを示している。また、返答を希望するものは図中の右の点のように太線で、そうでないものは他の要素のようにグレイの線で描かれる。

また、括弧内に複数のパスを並記することによって、パスの和を表現する (図 3)。図 3 では、dish の Left もしくは、On の関係にある Obj を求めている。

さらに、矩形を用いることによりスコープの範囲を指定することができ、これを利用して否定を表現したり、矩形領域の重なりによって集合演算をグラフィカルに表

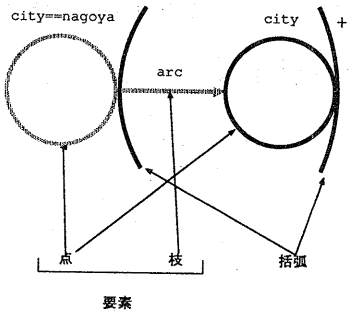

図 2: 問い合わせグラフの例

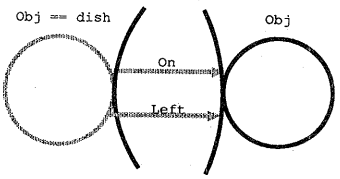

図 3: パスの並記の例

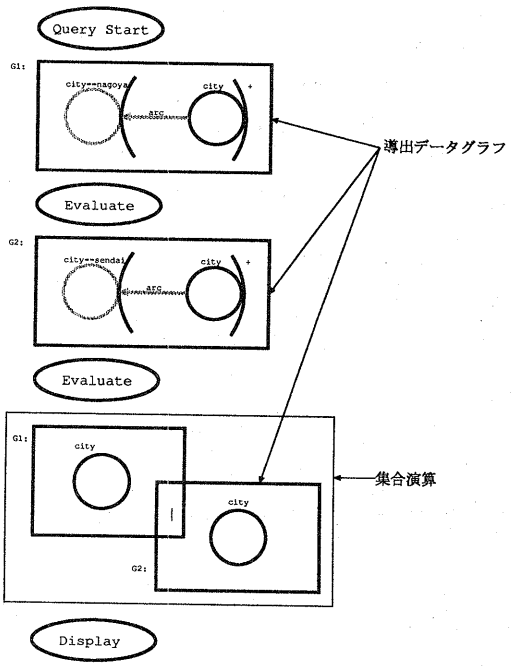

図 4: 問い合わせ表現の例

現することができる。

DUO は 2 段階の問い合わせの表現方法を持っている。簡単な問い合わせを宣言的に行う問い合わせグラフと複雑な問い合わせを手続き的に行う問い合わせ表現である。例として、都市のつながりを表すデータから “nagoya の上り方面または sendai の上り方面の city を求める” という問い合わせ表現を図 4 に示す。ここでは、nagoya の上り方面を求め、次に sendai の上り方面を求め、最後にそれらの city の和を求めている。

問い合わせ表現は複数の導出データグラフと 3 種類のラベルから構成される。3 つのラベル “Query Start”、“Evaluate”、“Display” はそれぞれ、” 問い合わせを開始する”、“問い合わせグラフを評価する”、“結果を表示する” ことを示す。

2.2 従来の DUO インタプリタ

従来のインタプリタは、DUO 言語の要素 (点と枝) と 1 つの枝のみを含む括弧を解釈することができる。

従来の DUO インタプリタのシステム構成を図 5 に示す。

インタフェースは Java で記述されている。また、インタフェースに入力された情報はすべて文字列に変換され、パイプで本体に渡されている。本体は C 言語で記述されている。本体に渡された文字列は、統合部で各部品ごとのグラフ情報としてデータ構造体に格納される。グラフ表現は、それぞれ同じ部品ごとのリストである。また、問い合わせを処理する際に、DUO 表現という中間言語を使用している。DUO 表現は、DUO のセマンティックスに沿って定義されている。グラフ→ DUO 表現変換部では、グラフ表現から DUO 表現への変換が行われる。DUO → Quixote 変換部では、DUO 表現の情報を用いて演繹オブジェクト指向 DBMS、Quixote [5, 6] に対する問い合わせ文を生成し、Quxiote に問い合わせを行う。

グラフ表現から DUO 表現への変換にあたり、要素の空間的な配置関係等をプログラムで検査している。記述量が莫大となるため、構文に制約を設けている。具体的には点、枝、並びに、1 つの枝を含む括弧のみをサポートし、水平方向のみの枝をサポートしている。

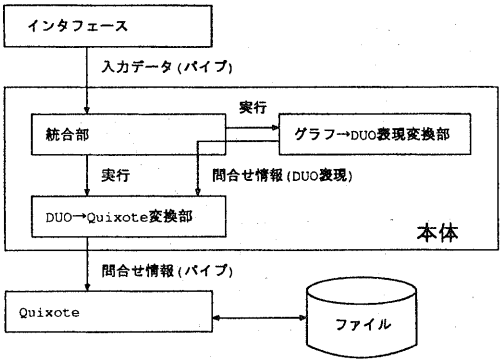

図 5: システム構成

3 多次元言語パーザ自動生成システム NAE

3.1 概要

NAE とは、制約 ・関係マルチセット文法を用いた多次元言語の定義から自動的に多次元言語のパーザを生成するシステムである。制約 ・関係マルチセット文法とは、Constraint Multiset Grammar に記号間の関係を導入した文法である。これにより、制約を簡潔に表現することができ、生成規則も簡潔なものとなる。NAE では、2 次元平面上に様々な図形や文字列等を配置することによってプログラミングを行う言語であるヴィジュアル言語だけでなく、3 次元空間での物体の配置や時間軸を考慮したプログラミング言語などの実現も考慮している。

NAE は多次元言語の定義を与えることによって、多次元言語パーザの C++ソースプログラムを自動的に生成する。(図 9) 多次元言語の定義として、記号の型定義、記号間の関係定義、並びに、生成規則定義を与えることになる。これらをもとに生成されたファイルと共通パーザをまとめてコンパイルすると所望の多次元言語パーザが得られる。

3.2 使用例

NAEにおける多次元言語の定義方法について説明する。

型名とその型の属性を列挙することで記号の型を定義する。この定義は最終的に C++のクラスに変換されるため、列挙する属性の型は C++で利用可能なものでなければならない。標準で C++に用意されていない型を利用する場合は、ユーザ定義型もしくはクラスを用意する必要がある。

ここで図 6 を定義する例を考える。

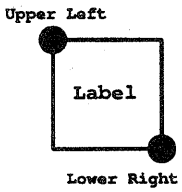

図 6: Rectangle 型

```
terminal-type Rectangle {
        int  upleft_x;
        int  upleft_y;
        int  lowright_x;
        int  lowright_y;
        string  label;
}
```

関係を表す記号の属性を用いて記号間の関係を記述することで関係を定義する。

すでに定義した Rectangle 型を用いて図 7 のような関係 HOR を定義する例を考える。

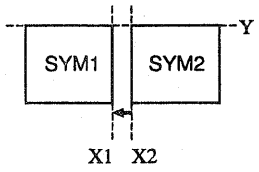

図 7: 関係 HOR(horizontal)

```
relation HOR(Rectangle SYM1, Rectangle SYM2)
{
    SYM1.lowright_x <= SYM2.upleft_x;
    SYM1.upleft_y == SYM2.upleft_y;
}
```

関係 HOR は Rectoangle 型の記号が水平に並んでいる関係を表している。制約として、左上の y 座標が等しい

かどうかと、左の記号の右下のx座標が右の記号の左上のx座標より小さいかどうかを定義する。この制約の場合、2つの記号がどれだけ離れていても制約さえ満たせば水平であると関係付けられる。また、y座標方向に少しでもズレがあれば制約を満たすことはできない。これらの問題を解決するためには、記号の属性に対する制約を詳細に記述する必要がある。

図8のように、すでに定義したRectangle型と関係HORを用いて、2つのRectangle型の記号から新しいRectangle型の記号を生成する生成規則を定義する例を考える。

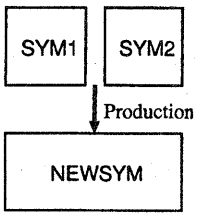

図8: 生成規則

```
SYMBOL:Rectangle ::= SYM1:Rectangle
                     SYM2:Rectangle
{
    where
    HOR(SYM1,SYM2);
    and
    SYMBOL.upleft_x = SYM1.upleft_x;
    SYMBOL.upleft_y = SYM1.upleft_y;
    SYMBOL.lowright_x = SYM2.lowright_x;
    SYMBOL.lowright_y = SYM2.lowright_y;
}
```

NEWSYM、SYM1、SYM2は全てRectangle型の記号である。where以降に制約を、and以降に代入形式の制約を列挙する。この生成規則の場合、記号の属性に対する制約はなく、記号に対する関係のみである。

3.3 システム構成

NAEは4つの要素から構成される。それぞれ、記号の型変換部、関係変換部、生成規則変換部、共通パーザである。NAEのシステム構成を図9に示す。

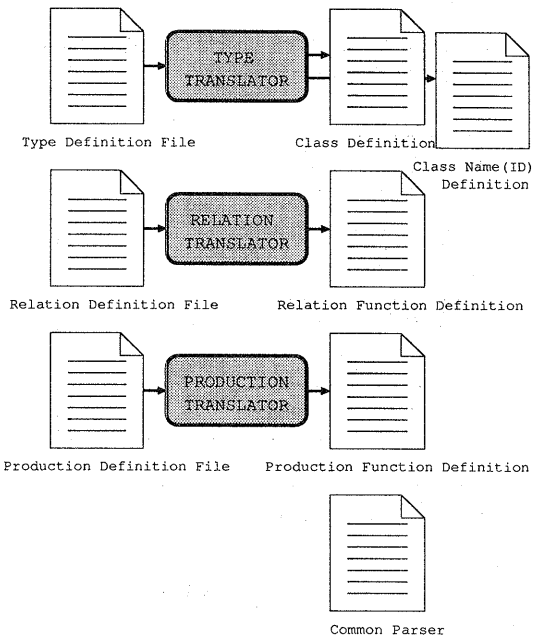

図9: 多次元言語パーザ自動生成システムNAE

記号の型変換部では、記号の型の定義をC++のクラスへと変換する。この記号の型の定義には、終端記号、非終端記号の型と属性が記述される。また、このとき必要なメソッドを自動的に追加する。実際に構文解析で用いられる記号は、このクラスのインスタンスとなる。また、変換したクラスの識別番号であるクラスIDを記述するための別ファイルも同時に作成する。

関係変換部では、記号間の関係の定義をC++の関数へと変換する。この関係の定義には、関係を持つ記号の属性に対する制約が記述される。生成される関数は、記号を引数とし制約を満たすかどうか(真か偽か)を返す。

生成規則変換部では、生成規則の定義を、記号を引数とする制約を満たすかどうかを判断し新しい記号を生成するC++の関数へと変換する。

共通パーザは、どのような多次元言語であっても構文解析を行うときに用いられるC++で記述された構文解析関数である。各変換部で生成されたクラス、関係関数、生成規則関数を用いて構文解析を行う。

4 実現

NAEにより生成したパーザを用いたDUOインタプリタのシステム構成を図10に示す。

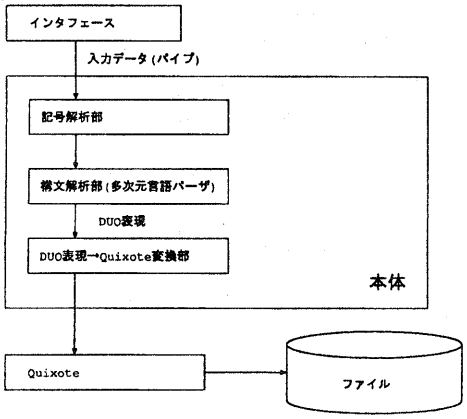

図 10: システム構成

DUOインタプリタは大きく分けてインタフェースと本体の2つに分けられ、本体はさらに記号解析部、構文解析部、DUO表現→ Quixote変換部の3つに分けられる。本体とインタフェースは、パイプを用いて情報のやりとりを行う。インタフェースは入力情報を文字列として本体に送る。記号解析部では、送られた文字列を記号の集合という形に変換し構文解析部へ送る。構文解析部は、記号解析部から送られた記号の集合に対して構文解析を行い、DUOのセマンティックスに沿った表現であるDUO表現へと変換を行う。この構文解析部の作成を、多次元言語パーザ自動生成システムNAEを用いて行う。DUO表現→ Quixote変換部では、構文解析部から送られたDUO表現をQuixoteの問い合わせ表現へと変換し、パイプを用いてQuixoteへ実際に問い合わせを行う。

従来のシステムとは次のような相違点がある。グラフ表現という表現方法をとりやめ、多次元言語パーザであるNaesetクラスによって記号の集合を扱う。このために、統合部とグラフ→ DUO表現変換部を、記号解析部と多次元言語パーザにそれぞれ置き換えている。

構文解析部は、NAEによって生成された多次元言語パーザを用いる。以下、記号の型の定義、関係の定義、並びに、生成規則の定義の概要を示す。

DUO文法の記号として、次の記号の型を定義する。

- Node
  点の情報を示す。
- Edge
  線の情報を示す。
- Blacket
  かっこの情報を示す。
- Rect
  矩形の情報を示す。
- Elm
  NodeまたはEdgeを指し、接続している要素のポインタを持つ。
- Par
  Blacketを指し、また中に含んでいる要素列へのポインタを持つ。
- Scope
  Rectを指し、また中に含んでいる要素列へのポインタを持つ。
- FullElms
  完全な要素列を示す。完全な要素列とは、始終点ともに点の要素列のことを指す。
- DestinationElms
  終点がない要素列を示す。
- SourceElms
  始点がない要素列を示す。
- ArcElms
  始点、終点ともにない要素列を示す。
- QueryGraph
  入力されたグラフ全体を示す。
- QueryElms
  問い合わせを行う記号の集合を示す。

DUO文法の記号間の関係として、次の関係を定義する。

- connect
  2つの要素列がつながっているという関係を示す。
- parallel
  2つの要素列が並行であるという関係を示す。
- contain
  かっこ、矩形が要素列を中に含むという関係を示す。

DUO文法の生成規則のとして、次の規則を定義する。

- QueryGraph ::= FullElms
  FullElms より QueryGraph を生成する。
- FullElms ::= FullElms DestinationElms
  FullElms と DestinationElms が、connect 関係にあれば FullElms を生成する。
- FullElms ::= SourceElms FullElms
  SourceElms と FullElms が、connect 関係にあれば FullElms を生成する。
- FullElms ::= FullElms Blacket
  FullElms と Blacket が、contain 関係にあれば FullElms を生成する。
- FullElms ::= FullElms FullElms
  FullElms と FullElms が、parallel 関係にあれば FullElms を生成する。
- FullElms ::= Node
  Node から FullElms を生成する。
- DestinationElms ::= ArcElms FullElms
  ArcElms と FullElms が、connect 関係にあれば DestinationElms を生成する。
- DestinationElms ::= DestinationElms Blacket
  DestinationElms と Blacket が、contain 関係にあれば DestinationElms を生成する。
- SourceElms ::= FullElms ArcElms
  FullElms と ArcElms が、connect 関係にあれば SourceElms を生成する。
- SourceElms ::= SourceElms Blacket
  SourceElms と Blacket が、contain 関係にあれば SourceElms を生成する。
- ArcElms ::= DestinationElms ArcElms
  DestinationElms と ArcElms が、connect 関係にあれば ArcElms を生成する。
- ArcElms ::= ArcElms SourceElms
  ArcElms と SourceElms が、connect 関係にあれば ArcElms を生成する。
- ArcElms ::= ArcElms Blacket
  ArcElms と Blacket が、contain 関係にあれば ArcElms を生成する。
- ArcElms ::= ArcElms ArcElms
  ArcElms と ArcElms が、parallel 関係にあれば ArcElms を生成する。
- ArcElms ::= Edge
  Edge から ArcElms を生成する。

さらに、DUO インタプリタの構文解析部に NAE を用いることにより、NAE に不足している機能、並びに、サポートした方が良い機能を明確化した。NAE に追加した機能を以下に示す。

- 各要素クラスにメンバ関数を持たせる機能
  各要素クラスにメンバ関数を持たせることにより、記号の関係をより簡潔に表記できる。
- すべての要素クラスに共通のメンバ関数を持たせる機能
  すべての要素クラスに共通のメンバ関数を持たせることにより、再帰的表記が可能となる。有向グラフの生成規則を表記するときなど、ループを用いなければならない場合に再帰的処理により簡潔なプログラミングが可能となる。
- 出力コードの一部を定義に直接コーディングする機能
  前述のメンバ関数を持たせる機能を追加したことにより、その関数の内容を直にコーディング可能とする機能が必要となる。

また、DUO インタプリタのインタフェース部の改良も行った。インタフェース部は Java により記述されており、本体とは独立したプロセスで動作している。ここでは、要素の移動、詳細データの変更、ならびに、アンドゥといった機能を追加した。また、要素の数が多いとき、再描画時に、ちらつきが目立つという問題が発生した。この問題を解決するためにダブルバッファリングという方法を採用した。ダブルバッファリングとは、直接、表示部に描画するのではなく、まず計算結果をダミーの表示部に描き、そのデータを表示部にコピーするという手法である。これにより、描画時間自体は増えるが、表示部の再描画にかかる時間を減らすとができ、ちらつきをおさえることができる。

問い合わせグラフの表示例を図 11 に示す。

5 評価

これまでのインタプリタと比較し、NAE を導入したことによる構文解析部の評価を行う。

従来のインタプリタでは構文解析を行うために、グラフ→DUO 表現変換部を製作している。このグラフ→DUO

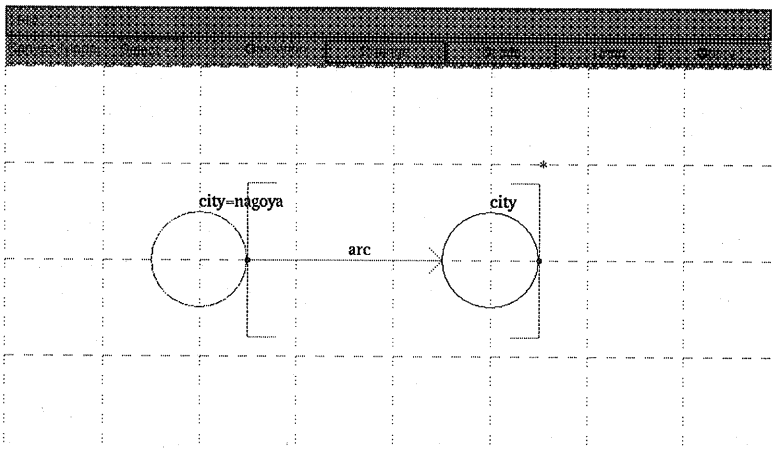

図 11: インタフェース部

表現変換部は、C 言語で約 1200 行のプログラムによって製作されている。

それに対し、本研究で製作されたインタプリタは構文解析に NAE によって生成された多次元言語パーザを用いているため、より容易に作成することができた。また、機能面では、従来できなかった並列の枝の処理が行えるようになった。

これらをまとめて表 1 に示す。

表 1: 構文解析のための記述量、並びに解釈できる文法の比較

項目	従来	改良
プログラム記述量 (行)	1200	700
定義記述量 (行)	—	400
解釈できる文法	点と枝と括弧による文	左に加え並列の枝の文

6 おわりに

グラフィカル問い合わせ言語 DUO のインタプリタの実装を行った。DUO インタプリタの構文解析部に多次元言語パーザ自動生成システム NAE により生成されたパーザを用いることによって、より容易にその製作を行うことができた。また、NAE を用いることで、記号の型の定義、関係の定義、並びに、生成規則の定義の記述によりパーザが生成できるので、DUO 文法の完全サポートや文法の拡張に容易に対応できるようになった。

今後の課題として、DUO インタプリタには DUO 文法の完全サポート、問い合わせ結果のグラフィカル表示、ならびに、多次元言語パーザ自動生成システム NAE には生成するパーザの高速化があげられる。

参考文献

[1] 西川博昭、寺田浩詔. 視覚的プログラミング環境, 情報処理 *Vol.30, No.4, pp.354-362 (1989)*

[2] 石井義知、宝珍輝尚、都司達夫. 多次元言語の構文解析に関する一手法, 信学技法 *SS96-64, pp.25-32 (1997)*

[3] 宝珍輝尚. グラフィカル問い合わせ言語 DUO の問い合わせ能力, 情報処理 *Vol.36, No.4, pp.959-970 (1995)*

[4] 神田儀道、宝珍輝尚、都司達夫. グラフィカル問合わせ言語 DUO インタプリタの設計・製作, データベースシステム *108-3, pp.17-24 (1996)*

[5] 財団法人 新世代コンピュータ技術開発機構, big-Quixote システム利用マニュアル *(1995/2)*

[6] 財団法人 新世代コンピュータ技術開発機構, Quixote 言語マニュアル *(1995/1)*

[7] Ira Pohl. C++エッセンス ANSI/ISO リファレンスとスタイルガイド, 青雲社 *(1997)*

[8] David R.Musser, Atul Saini. STL 標準テンプレートライブラリによる C++プログラミング, 青雲社 *(1997)*