ソフトウェア工学 90－7
（1993. 2. 4）

# 時間付き属性超グラフ文法に基づく図形編集環境の構成法

敷田 幹文　徳田 雄洋

東京工業大学 情報工学科

これまで図形の意味を扱うことができる個別的エディタは作成に多くの労力を必要としていた。我々は既に、属性文法の基底文脈自由文法を超グラフ文法に拡張し、時間的に変化する意味を持った図形を簡潔に表現する時間付き属性超グラフ文法を提案している。本論文では、この時間付き属性超グラフ文法を用いて対話型の図形編集環境を実現するための方法について述べる。最初に、この文法で扱う図形の対話的な定義方法について述べ、次に文法を意識せずに自由に描いた図形から時間付き属性超グラフ文法の構文木を得るために必要な構文解析法について述べる。最後にシミュレーションのアニメーションなど図形の意味計算の可能性について検討する。

# A Method for Constructing Semantic / Graphical Editing Environments Based on Timed Attribute Hypergraph Grammars

Mikifumi SHIKIDA　Takehiro TOKUDA

Department of Computer Science, Tokyo Institute of Technology

We proposed timed attribute hypergraph grammars. This class of grammars is a natural extension of attribute grammars in such a way that grammar symbols of the underlying context-free grammar can have parameters and that the concept of time is introduced in semantic functions. Using them, we can easily represent formal descriptions of applications dealing with animations based on graph structure. In this paper, we describe methods for defining figures interactively and for analyzing syntactic structures of figures in timed attribute hypergraph grammars consisting of terminal objects. Finally, we examine applications dealing with semantic computations on figures.

## 1 はじめに

表示装置などの発達に伴い、近年図形を扱うアプリケーションが増えてきた。しかし、これまでの多くの汎用図形エディタは、直線や円などの基本的な要素の集まりで図形を表しているに過ぎず、全体としてどのような意味を持っているかは扱っていない。また、回路図エディタや化学構造式エディタなどの図形の意味計算が可能なものもあるが、これらのアプリケーションを個々に作成するには多くの労力を必要とする。

しかし、既にテキスト情報の分野では、Gandalf システム [8] の ALOE 生成系や Knuth の属性文法 [4, 5] に基づく Cornell Synthesizer Generator[7] など、構造エディタ生成系の研究が盛んである。一方、図形のパターン認識やその構造解析の分野では図形の構造を形式的に定義することのできるグラフ文法 [12] が研究されている。

我々は、属性文法の基底文脈自由文法 [1, 13] を超グラフ文法 [10] に拡張し、その上で属性計算が可能となるようにした。また、アニメーションなどのアプリケーションのように時間毎に意味が変化する振る舞いを簡潔に記述する必要性から属性計算に時間の概念も導入し、**「時間付き属性超グラフ文法 (Timed Attribuute Hypergraph Grammar: TAHG)」** [3, 9, 11] を提案した。

本稿では、この TAHG を用いて対話型の図形編集環境を実現するための方法について述べる。最初に TAHG について簡単に述べ、次にこのような環境で扱う図形の表現方法について検討し、最後に文法を意識せずに描いた図形から TAHG の構文木を得るために必要な TAHG の構文解析の方法について述べる。

## 2 時間付き属性超グラフ文法

ここでは我々の提案した時間付き属性超グラフ文法について簡単に説明する。

### 2.1 定義

TAHG では、図形を 0 個以上の任意個の点を結ぶ超辺の集合、すなわち超グラフとして表現する。ここで、超辺は図形を構成する部品を表すことになる。

さらに、超辺は接続の情報を点または点のリストとしてパラメータに持つ記号として表現される。

例えば、「2 点 a, b を直線で結んだ図形」というのは、"点", "直線" をそれぞれ "point", "line" という終端超辺名にすると次のようになる。

**point**(a) **line**(a, b) **point**(b)

また、超グラフは超辺の集合であり、基本的には順序がないので

**point**(a) **point**(b) **line**(a, b)
**line**(a, b) **point**(b) **point**(a)

なども同じ図形を表す。

超グラフを使った超グラフ文法にもいろいろなものがあるが、我々が TAHG の原型として採用したものは、1 つの非終端超辺から 0 個以上の超辺、すなわち部分超グラフへの書換えを生成規則に持つ種類のものである。1 つの生成規則で同じ記号が超辺のパラメータに現れた場合は、これらは同じ点または点リストを指す。

非終端超辺もパラメータに点または点リストをもつことができ、この超グラフ文法は高い表現能力を持っている。

この文法では超グラフ構造が表現できるが、文脈自由文法に似た生成規則であるため文脈自由文法と同様に木構造の構文木ができる。したがって、各終端超辺、非終端超辺のそれぞれに属性を付けて属性文法と同様の意味計算が可能である。

TAHG による記述例を次に示す。図 1 のような無向グラフ全体は図 2 に示す TAHG の記述で表すことができる。この文法では意味として「グラフ全体の辺の数」を属性 count を用いて計算している。また、この文法で図 1 の無向グラフを生成すると図 3 に示す構文木ができる。

また、TAHG にはシミュレーションなどの応用を考えて属性計算に時間の概念を導入してある。

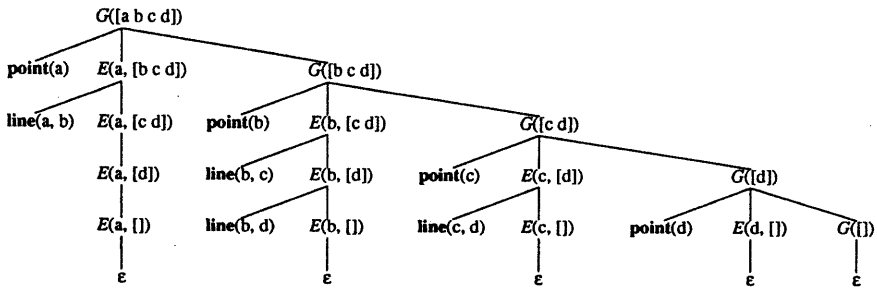
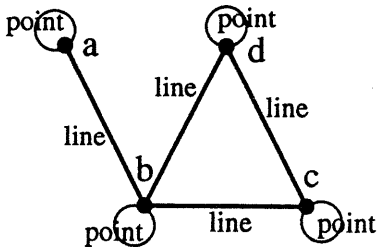



図 3: "無向グラフ" の構文木




point(a) point(b) point(c) point(d)
line(a, b) line(b, c) line(b, d) line(c, d)

図 1: 無向グラフの超グラフによる表現例

このため属性関数は次のような 2 つの形式に別れている。

- 時刻によって変化しない属性では文脈自由文法の属性文法と同様な記述である。
- 時刻によって変化する属性の場合は、初期値とある時刻の次の時刻の値を与える式の 2 つを記述する。この場合にはそれぞれの属性関数の左辺に **Init** または **Next** を付加する。

前述の無向グラフの例では意味が辺の数であるため時刻によって変化する意味は扱っていない。

次に、時刻に依存する属性計算の例として図 4 に "Markov 過程" を示す。各状態の初期確率を与えると各時刻における値が図 5 のように表示される。このとき確率は属性 poststatus で保持されている。

1. $G([x]D) \rightarrow$ **point**(x) $E(x, D)$ $G(D)$
   $G_0$.count := E.count + $G_1$.count
2. $G([]) \rightarrow \varepsilon$
   $G_0$.count := 0
3. $E(x, [y]D) \rightarrow$ **line**(x, y) $E(x, D)$
   $E_0$.count := $E_1$.count + 1
4. $E(x, [y]D) \rightarrow E(x, D)$
   $E_0$.count := $E_1$.count
5. $E(x, []) \rightarrow \varepsilon$
   $E_0$.count := 0

図 2: "無向グラフ" の TAHG による記述

## 2.2 構文木の作成方法

属性計算を行うためには構文木を構築する必要があるが、対話型の図形編集環境を考えた場合に以下のような 2 つの方法が考えられる。

### 2.2.1 構文解析法

ユーザは汎用の作図ツールで描くように自由に各終端超辺に対応する図形を配置し接続する。この方法では構文木を作るために描き終わった図形を構文解析する必要が生じる。しかし、TAHG は文脈自由文法ではないので、文脈自由文法と全く同じ構文解析はできない。

また、作図中には文法に従っているかどうかの検査が行われないため、その文法からは生成できな

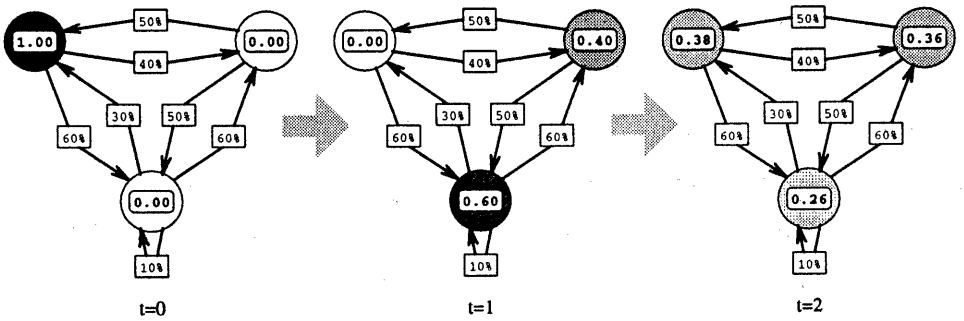



図 5: "Markov 過程" の実行例

い図形も自由に描けてしまうという問題がある。

この方法は文脈自由文法の場合の、汎用テキストエディタでプログラムを作成し、コンパイラで構文解析する方法に対応する。

#### 2.2.2 生成規則展開法

ユーザは開始超辺から順に生成規則に従った展開を行う。この方法で完成した図形は必ず構文的に正しいものであり、構文木も図形と同時にできる。

これは文脈自由文法の場合の構造エディタを利用する場合に対応する。

TAHG で扱う図形は超グラフであるが、この構造の接続状態は超辺のパラメータによって表現されており、構文木の木構造はこれとは直接関係がない。つまり、ユーザは図形の外見とは違った木構造に沿って展開を行って図形を描かなければいけないことになる。また、TAHG では一般的に非終端超辺のパラメータの数が多い傾向にあり、図形を描いている途中の状態はパラメータの接続を示す膨大な量の線が表示される。従って、対話型の編集環境ではこれらを回避する手段が必要であろう。

## 3 作図ツール

ここでは前節で述べた TAHG を利用して対話的に図形を編集し、その図形の意味計算を行う環境について述べる。なお、構文木の構築には構文解析を用いる。

この方法を用いたシステムは一般的に図 6 に示すような構成になる。すなわち、与えられた文法とそれぞれのオブジェクトの表示に関する情報から作図ツールを自動生成する。また、このツールで描いた図形は、同じ文法から自動生成された構文解析器で解析することにより構文木に変換され、その上で図形の意味計算を行う。

### 3.1 図形の定義

TAHG の文法には構文規則と意味規則は記述されているが、対話型の環境の上で各超辺をどのように表示するかは定義されていない。

超グラフの接続関係は各超辺のパラメータで表現されているので、ユーザに対しては各パラメータの値を線による接続で示さなければいけない。しかし、一般に TAHG のパラメータには順序があり、どの線がどのパラメータなのかがユーザに理解できなければいけない。また、図形によっては線が繋がる場所が決まっている場合がある。例えば、論理回路図エディタの "OR ゲート" は線の繋がる場所で入出力を区別している。

このため、パラメータ列を線の繋がる位置によって次の 3 種類に分けて考える。

- 繋がる位置がどこでもよい場合
  パラメータの順序を換えても意味が同じ図形がこれに相当する。
- 位置を作図時に対話的に決定する場合
  どの線がどのパラメータかを作図ツールで表

1. *MARKOV*(S) → *STAT*(S, S)

STAT.poststatus
:= *multiply*(STAT.probmatrix, STAT.prestatus)

2. *STAT*([x]S, T) →
**stat**(x) *TRANS*(x, T) *STAT*(S, T)

$STAT_1$.poststatus := *cdr*($STAT_0$.poststatus)
$STAT_0$.prestatus := *cons*(stat.status, $STAT_1$.prestatus)
$STAT_0$.probmatrix
:= *cons*(*normalize*(TRANS.problist), $STAT_1$.probmatrix)
**Init** stat.status := *input*()
**Next** stat.status := *car*($STAT_0$.poststatus)

3. *STAT*([ ], T) → $\varepsilon$

STAT.probmatrix := ( )
STAT.prestatus := ( )

4. *TRANS*(x, [y]T) → **trans**(x, y) *TRANS*(x, T)

$TRANS_0$.problist := *cons*(trans.prob, $TRANS_1$.problist)

5. *TRANS*(x, [y]T) → *TRANS*(x, T)

$TRANS_0$.problist := *cons*(0, $TRANS_1$.problist)

6. *TRANS*(x, [ ]) → $\varepsilon$

TRANS.problist := ( )

図 4: TAHG の例: Markov 過程

示しなければいけない。

- パラメータによって位置が決っている場合
図形定義の段階で各パラメータの位置を決めなければいけない。

また、パラメータが点リストの場合にも同様な問題が起こる。この場合には位置が既に決っているので、その点リストの要素の順序に意味があるかどうかだけを区別すればよい。

パラメータによって位置が決っている論理回路図エディタの“OR ゲート”で図形定義エディタの表示例を図 7 に示す。

なお、構文解析法による作図ツールでは終端超辺のみを描画するため非終端超辺の図形定義は必要ない。

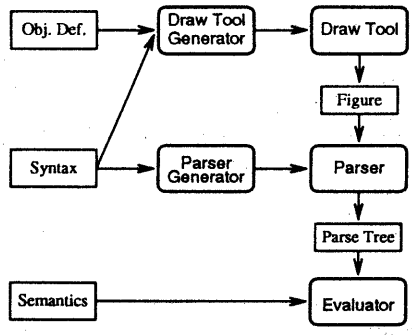



図 6: TAHG を利用した環境の構成

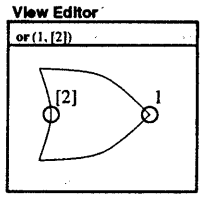



図 7: 図形定義の例

### 3.2 編集環境の生成

文法と各終端超辺の図形の定義を与えるとその文法専用のエディタが自動生成できる。生成されたエディタは次のような機能を持つ。

- 各終端超辺および連結のための点を自由に配置できる。
- 各終端超辺のパラメータの位置に点を指定することにより、パラメータの位置と点の間に連結線が描かれる。ここで、前述のパラメータ列の種類が作図時に位置を決めるものにはパラメータの区別のためのラベルを付加する。点リストに順序がある場合にも同様にラベルを描く。

例えば、論理回路図のエディタの例を示すと図 8 のようになる。この例で描かれた図形は次のように出力される。

**input**(a) **input**(b) **input**(c) **input**(d)
**and**(e, [c d]) **or**(f, [a b e]) **output**(f)

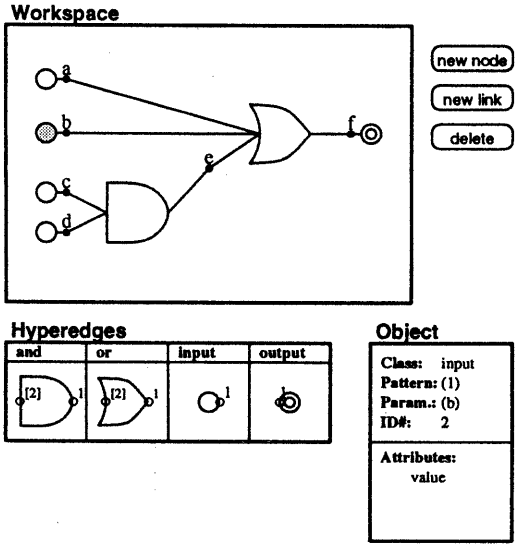



図 8: 生成された編集環境の例

## 4 構文解析

この節では、超グラフ文法の構文解析の方法について述べる。

TAHG がもとにした超グラフ文法は Prolog で直接処理できる形になっており、かつ、Prolog 以外の方法では困難になっていた。我々は、この超グラフ文法が文脈自由文法に似た性質を持っていることに注目し、既存の文脈自由文法に近い処理が可能となるように TAHG を設計した。その結果、TAHG ではより効率のよい構文解析が可能となった。

ここではボトムアップとトップダウンの 2 つの方法について述べる。

### 4.1 構文解析時の問題点

超グラフ文法の構文解析を行うには次の 2 つの問題が生じる。

1. 順序

   文脈自由文法の文はアルファベットの並びである。しかし、TAHG の文は基本的には終端超辺の集合であり、順序がないので文脈自由文法のように左から順に走査する方法が使えない。

   例えば、次の 2 つの図形は文脈自由文法で考えると異なるものに見えるが、TAHG では全く同じ図形を表しているので構文解析の結果は同じにならなければいけない。

   1. **point**(a) **point**(b) **line**(a, b)
   2. **point**(a) **line**(a, b) **point**(b)

2. パラメータ

   TAHG の超辺はパラメータを持っている。

   例えば、"*G*([a b])" から "**point**(a) **line**(a, b) **point**(b)" は導出されるが、"**point**(a) **line**(a, c) **point**(c)" は導出できない。

   すなわち、構文解析器のパターンマッチングにはパラメータも含まれなければいけない。

### 4.2 ボトムアップ

文脈自由文法では LALR(1) 解析器を使ってボトムアップの構文解析がよく行われる。ここでは一般の LALR(1) 解析器を TAHG に利用した場合に前述の問題点を解決する方法について述べる。

#### 4.2.1 超辺の順序

TAHG では基本的に終端超辺の間に順序はないが、順序の問題を回避するために超辺の間に順序を導入する。

これは次のように生成規則の右辺の並べ替えによって行う。

- 終端超辺 — 左端に辞書順
- 左辺と同じ非終端超辺 — 右端に辞書順
- パラメータ — 左辺での出現順

このような方法で並べ替えを行うことによって、導出される終端超辺はパラメータの辞書順で規則的に並ぶ。したがって、作図ツールでもこれと同

じ順序で図形を出力することにより、左から右へ順に走査する構文解析が可能になる。

#### 4.2.2 パラメータの除去

次に、パラメータの問題であるが、ここでは既存のLALR(1)のツールを用いるためにパラメータを取り除く方法について述べる。

生成規則の変換の手順は図9の例のように、まずパラメータを単純に取り去り、次にパラメータを変化させるだけの"$E \rightarrow E$"のような規則を取り除く。ただし、このままではパラメータが求まらないので、還元の際のアクションの手続きの中でパラメータのチェックを行い、構文木に埋めていく。

$$\begin{array}{l} E(x, [y]D) \rightarrow \textbf{line}(x, y)\ E(x, D) \\ E(x, [y]D) \rightarrow E(x, D) \\ E(x, []) \rightarrow \varepsilon \end{array} \Rightarrow \begin{array}{l} E \rightarrow \textbf{line}\ E \\ E \rightarrow \varepsilon \end{array}$$

図9: パラメータの除去例

#### 4.2.3 パラメータの修正

しかし、この方法では必ずしもパラメータが正しく求まるとは限らない。例えば、

$$E(a, [b\ c]) \overset{*}{\Rightarrow} \textbf{line}(a, b)$$

という導出が行われて図形が作られていた場合に、"line(a, b)"から還元を行ったのでは"$E$(a, [b])"が仮定されてしまう。また、還元の際にはまだ決定できない場合も考えられる。しかし、このような場合にもパラメータを仮定しておき、後で別の部分木のパラメータとの間で矛盾が起こった時点で、その部分木のパラメータを修正することにより正しい構文木が得られる。

### 4.3 トップダウン

次に、トップダウンの再帰降下型の構文解析を行う場合について述べる。

順序の問題はボトムアップの場合と同様に順序を決めて右辺の超辺を並べ替える必要がある。しかし、パラメータに関しては、走査した超辺と生成規則の左辺を比較するときにパラメータを含めたパターンマッチングを行うことによって生成規則を決定でき、解析と同時にパラメータも決定する。このため、ボトムアップのときのようにパラメータを後で修正する必要はない。

しかし、生成規則にしたがって置き換えた後の各超辺のパラメータを決定するためには右辺に出現するパラメータ変数名が必ず左辺にもなければいけない。したがって、例えば"$P$(x, y) → $Q$(x, z) $R$(z, y)"という生成規則では構文解析できないことになる。ただし、これまでのいろいろな実例の記述ではこのような生成規則が必要となることはあまりなく、また出現しない文法に書き換えることが可能であった。

さらに、文脈自由文法での再帰降下型構文解析では**"左再帰性の除去"**という問題が生じるが、TAHGの場合には右辺の各超辺に順序がないため容易に除去できる。

## 5 実現

前節までで述べた、TAHGに基づく対話型図形編集環境生成系のプロトタイプを現在実現中である。これには、オブジェクト指向プログラミングが可能なSchemeインタプリタ[2]であるElk[6]を利用している。

現在の実現ではまだ対話型の環境になっていないので、各超辺の図形定義とTAHGの記述をS式で与える。各超辺はElkのオブジェクト指向環境の中でそれぞれ独立したクラスになるように生成される。生成された各クラスと作図ツールの雛形を合成することによりその文法専用の作図ツールが構築できる。

また、S式のTAHGの記述から再帰降下型構文解析器も自動生成される。前述の作図ツールが出力する図形はS式で表現されており、この構文解析器によって構文木が作成できる。

さらに、属性評価器もElkで実現されており、この構文木とTAHGの意味規則の記述を与えるこ

とにより、各時刻の属性値を計算することができる。作図ツールの生成時に作られた各クラスの図形定義はなどは評価器でも使用されることになる。

なお、Elk はさらに X Window System[1] とのインターフェースも持っており、X Toolkit や Xlib による細かい制御も可能なためグラフィカルユーザインターフェースも実装される。

例えば、TAHG を利用して自動生成された論理回路図エディタは図 8 に示すようなものになる。

## 6 まとめ

時間的に変化する意味を持った図形を簡潔に表現する時間付き属性超グラフ文法について述べ、この文法に基づく図形を編集する対話型の環境を自動生成するための、図形の定義方法と構文解析の方法について述べた。

現在、TAHG の例もいろいろと記述されており、"Markov 過程","ペトリネットシミュレータ" なども非グラフィカルな環境で意味計算が可能となっている。

この方法により、複雑な結合関係を持った図形の意味を扱う環境をこれまでより容易に実現することが可能になった。しかし、TAHG では図形の構造と直接的な関係のない木構造の上で意味計算を行うため、意味の記述は必ずしも容易ではないという欠点もある。

今後の課題としては以下のことが考えられる。

現在は実現の作業が完了していないため動作する例があまり多くない。このため現在構文解析可能な範囲内で充分実用的かどうか、プロトタイプの実現後にさらに多くの例を記述する必要がある。

扱える意味の範囲を広げるために、TAHG に動的に構文木を変化させる機能を導入することも有効と思われる。また、TAHG を用いずにオブジェクト指向のメッセージなどによる意味計算との比較も考えられる。

[1] X Window System is a trademark of the Massachusetts Institute of Technology.

## 参考文献


[1] A. V. Aho, R. Sethi, and J. D. Ullman. *Compilers: Principles, Techniques, and Tools*. Addison-Wesley, Reading, MA, 1986.

[2] William Clinger and Jonathan Rees (Editors). *Revised[4] Report on the Algorithmic Language Scheme*, November 1991.

[3] Y. Kimura and T. Tokuda. Timed attribute hypergraph grammars. 日本ソフトウェア科学会 第 9 会大会論文集, pp. 405–408, 1992.

[4] D.E. Knuth. Semantics of context-free languages. *Mathematical Systems Theory*, Vol. 2, No. 2, pp. 127–145, 1968.

[5] D.E. Knuth. Semantics of context-free languages: Correction. *Mathematical Systems Theory*, Vol. 5, No. 1, pp. 95–96, 1971.

[6] Oliver Laumann. *Reference Manual for the Elk Extension Language Interpreter*. Communications and Operating Systems Research Group, Technical University of Berlin, Germany, November 1990.

[7] T. Reps and T. Teitelbaum. The Synthesizer Generator. In *Proceedings of the ACM SIGSOFT/SIGPLAN Software Engineering Symposium on Practical Software Development Environments*, pp. 42–48. ACM SIGPLAN Notice 19, 5, May 1984. Pittsburgh, PA, Apr. 23–25, 1984.

[8] J. Staudt, Barbara, W. Krueger, Charles, A. N. Habermann, and Vincenzo Ambriola. *The GANDALF System Reference Manuals*. Dept. of Computer Science, Carnegie-Mellon University, May 1986.

[9] Y. Watanabe, M. Shikida, and T. Tokuda. Applications of timed attribute hypergraph grammars. 日本ソフトウェア科学会 第 9 会大会論文集, pp. 409–412, 1992.

[10] 塩谷 勇, 中村 克彦. 論理超グラフ文法と文脈自由超グラフ文法. 電子情報通信学会論文誌 D, Vol. J73-D-I, No. 4, pp. 385–394, 1990.

[11] 木村喜昌. 属性超グラフ文法に基づく図形情報アプリケーション生成系の研究. Master's thesis, 東京工業大学 工学部 情報工学科, 1992.

[12] 西野哲朗. 属性グラフ文法とその hichart 型プログラム図式に対するエディタへの応用. コンピュータソフトウェア, Vol. 5, No. 2, pp. 81–92, 1988.

[13] 徳田雄洋. 構文解析. 昭晃堂, 1989.