Syntactic Processing of Diagrams by Graph Grammars

Tomokazu Arita Kiyonobu Tomiyama Takeo Yaku
Dept. App. Math., Nihon Univ.
Setagaya 156-8550,JAPAN
{arita,tomiyama,yaku}@am.chs.nihon-u.ac.jp

Youzou Miyadera
Dept. Math. & Inf. Sci.,
Tokyo Gakugei Univ.
Koganei 184-8501,JAPAN
miyadera@u-gakugei.ac.jp

Kimio Sugita
Dept. Math.,
Tokai Univ.
Hiratsuka 259-1292,JAPAN
sugita@math.sm.u-tokai.ac.jp

Kensei Tsuchida
Dept. Inf. & Comp. Eng.,
Toyo Univ.
Kawagoe 350-8585,JAPAN
kensei@eng.toyo.ac.jp

Abstract We deal with syntactic definitions and processing of program diagrams based on graph grammars with respect to the mechanical drawing. We propose an attribute NCE graph grammar of hierarchical diagrams such as structured program diagrams. We also propose attribute context-free and context-sensitive NCE graph grammars for nested and tessellation diagrams, respectively. Attribute rules are used for the mechanical drawing. Furthermore, we introduce an integrated diagram processing method based on NCE graph grammars. The results could be applied to general diagram processing.

Keywords Software visualization, software engineering tools and environment, graph grammars, tabular forms.

1. Introduction

We deal with syntactic definitions and processings of program diagrams based on graph grammars with respect to the mechanical drawing.

Several models and properties of graph grammars have been investigated by Franck [1], Della Vigna [2], Rozenberg [12] and others [5]. Recently, NCE graph grammars (see.*e.g.* [12]) have been considered as reasonable models of design and analysis especially for artificial objects.

On the other hand, graph manipulating systems such as graph editors and graph drawing systems using combinatorial and constraint algorithms were developed.

In accordance with development of the graph grammar theory, syntactic graph manipulating systems were also developed such as DIAGEN. Among them, several large projects such as APPLIGRAPH have been also developed. In [12], Nagl et al introduced IPSEN systems.

In 1978, Yaku and Futatsugi introduced the graphical notation of program flowcharts Hichart. Several symbols and global structure of Hichart diagrams have been employed by other program diagram languages. Our project for the graph processing was named KEYAKI (see e.g.[10,14,16]). In 1987, we introduced a non-syntactic character based flowchart-processing systems (Yaku-Futatsugi-Adachi-Moriya [4]). In 1996, we formalized Hichart diagram editing commands by the attribute graph grammars based on Della Vigna's context-free graph grammar [2,10]. Yaku and Yamazaki also introduced several results in graph language theory [6]. A CAI system using our graph grammatical results was introduced in [11], and a system for rule based program visualization was introduced in 1998 [15].

In this paper, we consider three typical types of diagrams in program specifications, that is, (1) hierarchical diagrams such as structured program diagrams, (2) nested tabular diagrams such as a header of program specification forms, and (3) tessellation tabular diagrams such as symbol tables. The purpose of this paper is to characterize types of graph grammars that generate those three types of diagrams, and to propose processing methods of diagrams using the graph grammars.

We propose an attribute NCE graph grammar of hierarchical diagrams such as structured program diagrams.

We also propose attribute context-free and context-sensitive NCE graph grammars for nested and tessellation diagrams, respectively.

Furthermore, we introduce an integrated diagram processing method based on NCE graph grammars. The results could be applied to general diagram processing.

In Section 2, we review program flowcharts Hichart and program specification forms. In Section 3, we introduce graph grammars of program diagrams Hichart. In Section 4, we introduce attribute graph grammars for nested and tessellation diagrams. In Section 5, we introduce a uniform system which supports graph grammar based diagram processing methods. Section 6 provides concluding remarks.

2. Program Flowcharts and Program Specification Forms

In this Section, we review diagrams appeared in the software visualization. We consider two types of diagrams. One is *hierarchical diagrams* for program flowchart and The other is *tabular diagrams* for program specification forms.

2.1 Hierarchical Diagrams for Program Flowchart

We introduce a program flowchart description language Hichart (Hierarchical flow CHART description language) [4]. Hichart is of a tree structured program flowchart type. Figure 1 shows an example of Hichart flowcharts for the "Hanoi Tower" problem.

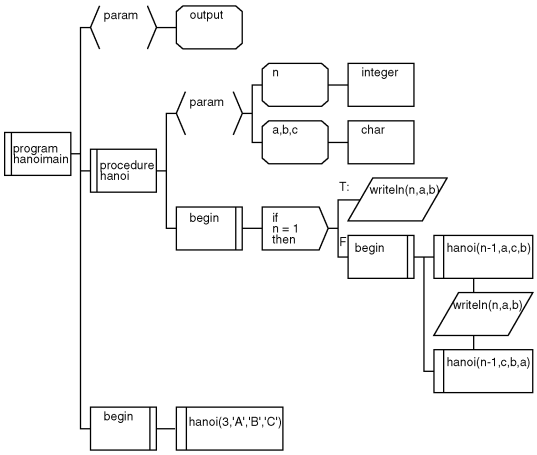


Figure 1. A Hichart program flowchart (Tower of Hanoi).

Hichart flowcharts have the following characteristics:

- Each process is described within a corresponding cell.
- Cells are arranged in a hierarchical manner.
- Cells are connected to each other by lines.
- Some lines run vertically and some horizontally, and these do not cross.
- Diagrams are displayed on a plane.

Formal definition of the hierarchical diagrams will be discussed in Section 3.

2.2 Tabular Diagrams for Program Specification Forms

We introduce here a program specification language called Hiform [14] based on ISO6592 [3].

The International Organization for Standardization issued a guideline in ISO6592 and described all items in program documentation in Annexes A, B and C. We considered the ISO6592 items and introduced Hiform96, which includes all items defined in these Annexes. Hiform[14] is defined by 17 types of forms.

Hiform was originally developed for the purpose of facilitating the software development at schools. Hiform Specification is a collection of tabular diagrams. Using tabular diagrams, one can understand at a glance what information should be obtained, what information is lacking, how a project is proceeding, and how to develop and maintain the software. Besides these characteristics, the tabular forms can include various description styles such as letters and diagrams.

The following Figure 2 shows a Hiform program specification form.

The order among tabular forms is defined by a context-free string grammar [11]. The order and graphical structure of cells inside tabular diagrams will be formally discussed in Section 4.

Project Code: hanoi_main	A 5
Program Name: procedure hanoi	Program Specification-1 p
Library Code: cs-2000-01	Version: 2.1
Author: Tomokazu Arita	Original Release: 1999/12/25
Approver:	Current Release: 2000/1/31
Problem Description: How to discribe the structure of this program.	
Problem Supplementary Information (Theoretical Principles, Methods and References): Theoretical Principles: Hichart Diagram	
Problem Solution: 1.Conventions and Terminology 2.Principles and Algorithms 1. Convention : Hichart Diagram	

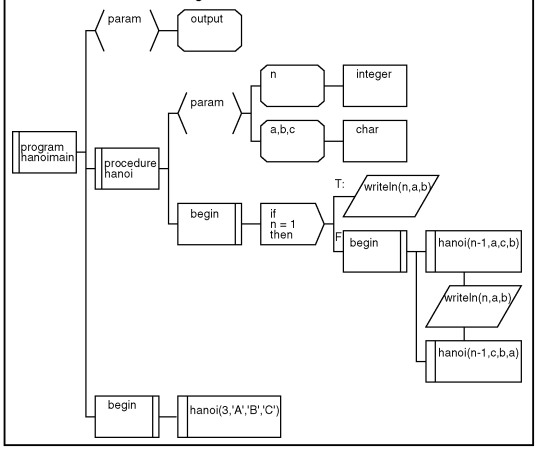


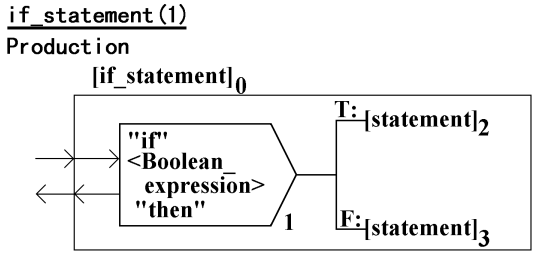
Figure 2. Hiform program specification form (Program Specification-1).

3. An Attribute Graph Grammar for Hierarchical Diagrams

In this Section, we introduce an attribute graph grammar HiChart Graph Grammar (HCGG) for the hierarchical diagrams in Hichart. HCGG provides graph grammatical definitions and attribute rewriting rules. Graph grammatical part is defined by a *context-free NCE graph grammars* [14]. On the other hand, *attribute rules* compute attributes for diagram layout such as coordinates and size of each cell. The major characterization of HCGG is as follows.

Grammar 3.1 HCGG is an attribute context-free NCE graph grammar for the hierarchical diagrams in Hichart.

Typical productions of HCGG are shown in Figures 3A and 3B.



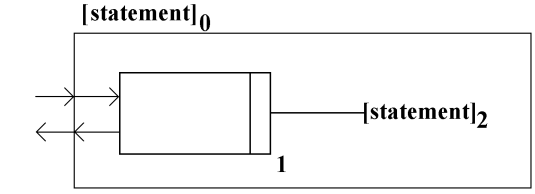
Semantic Rules

top(2)=top(0)	cl(2)="T:"
top(3)=bottom(2)+GapY	cl(3)="F:"
x(1)=x(0)	id(1)=id(0)
x(2)=x(0)+w(1)+GapX	id(2)=id(1)+1
x(3)=x(0)+w(1)+GapX	id(3)=id(2)+nc(2)
y(0)=(y(2)+y(3))/2	nc(0)=1+nc(2)+nc(3)

bottom(0)=max(bottom(1),bottom(3))

w(1)=MinW
h(1)=get_height(["if",<Boolean_expression>,"then"])
cell(1)="exclusive_selection"
string(1)=get_str(["if",<Boolean_expression>,"then"])
lines(1)=get_line(1,[2,3])

Figure 3A. Production in HCGG.



Semantic Rules

top(2)=top(0)	id(1)=id(0)
x(1)=x(0)	id(2)=id(1)+1
x(2)=x(0)+w(1)+GapX	nc(0)=1+nc(2)
y(0)=y(1)	
y(1)=y(2)	

bottom(0)=max(bottom(1),bottom(2))
bottom(1)=y(1)+h(1)

w(1)=MinW
h(1)=MinH
cell(1)="continuous_iteration"
string(1)=get_str([" "])
lines(1)=get_line_begin(1,[2])

Figure 3B. Production in HCGG.

Property 3.2 Attributes in HCGG are evaluated in linear time.

A derivation process of a Hichart diagram by HCGG is shown in the following Figure 4.

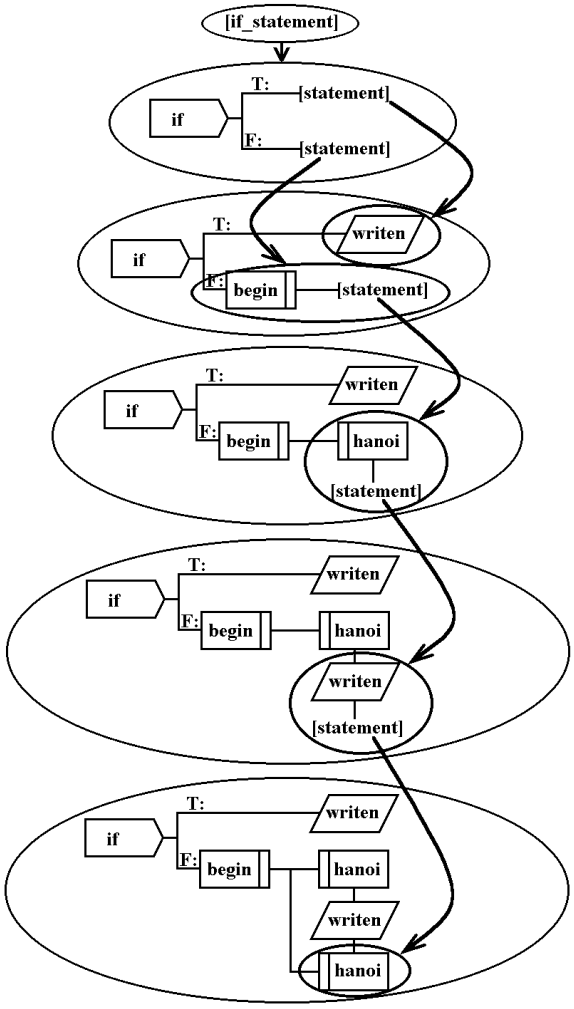


Figure 4. Derivation Processes of Hichart Diagram based on HCGG.

4. Attribute Graph Grammars for Tabular Diagrams

In this Section, we deal with the syntactic formalization of tabular diagram Hiform. The formal definition of Hiform documentation consists of two stages of grammars. One provides the order among tabular forms, and the other provides arrangement of items inside each form. The former is a context-free string grammar [11], and the latter is an attribute graph grammar.

Hiform has graph grammars for graph syntax and attribute rules for drawing conditions. In the graph grammar of Hiform, a program form will be represented by completed graphs with locations.

We illustrate examples which are a nested graph and a marked graph for a program documentation in Figure 5.

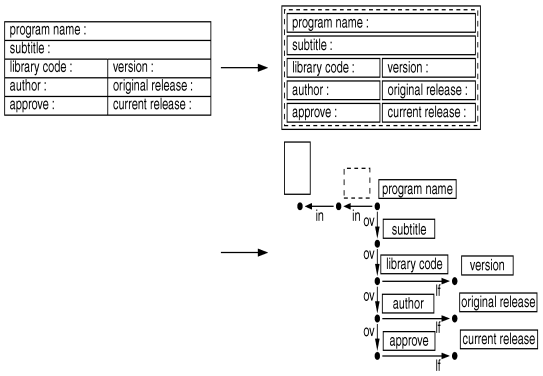


Figure 5. A tabular diagram, a corresponding nested diagram and a corresponding marked graph.

In Figure 5, a label of an edge in the marked graph denotes the relation between two nodes that are start node and end node of the edge. Label "in" denotes *within*, label "ov" denotes *over* and label "lf" denotes *left of*.

Grammar 4.1 Hiform Nested Graph Grammar (HNGG) is an attribute NCE graph grammar for the nested diagrams in Hiform header part.

We show typical productions of HNGG in Figure 6. Each production has attribute rules for drawing conditions.

The HNGG includes 280 productions and 1204 attribute rules for the definition of nested diagram part such as the program form headers.

HNGG syntax has the precedence property.

Property 4.2 Grammar HNGG is a precedence graph grammar.

Thus, the parsing algorithm of Hiform is partly given by the Franck's linear time parsing algorithm [1].

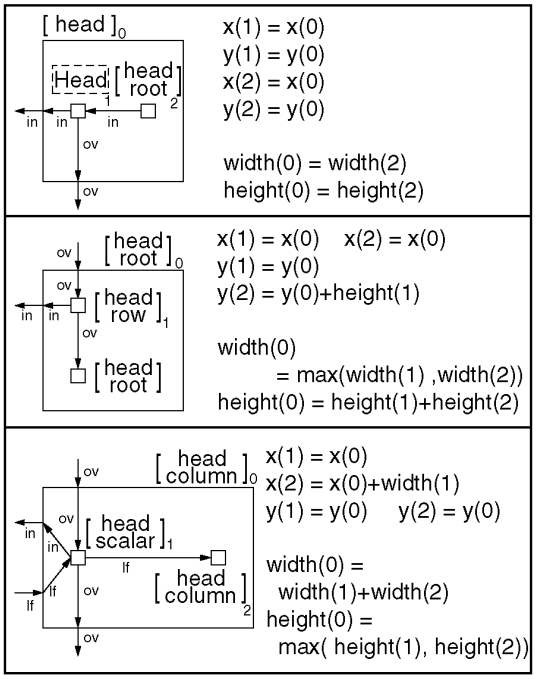


Figure 6. Productions and semantic rules of HNGG.

Next, we consider tessellation tabular form such as symbol tables in specification form in Hiform.

Following Figure 7 shows an example of tessellation diagrams.

a	b	c	d
e	f	g	h
i	j	k	l
m	n	o	p

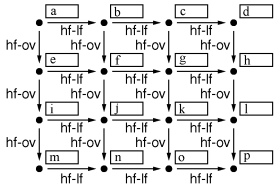


Figure 7. Tessellation diagram and its corresponding marked graph.

Now we introduce a *context-sensitive* attribute NCE graph grammar for tessellation diagrams.

Grammar 4.3 Hiform Tessellation Graph Grammar (HTGG) is a context-sensitive attribute NCE graph grammar for the tessellation diagrams in Hiform.

The grammar HTGG includes 56 syntax rules and 256 attribute rules. Following Figure 8 shows typical productions and attribute rules in HTGG.

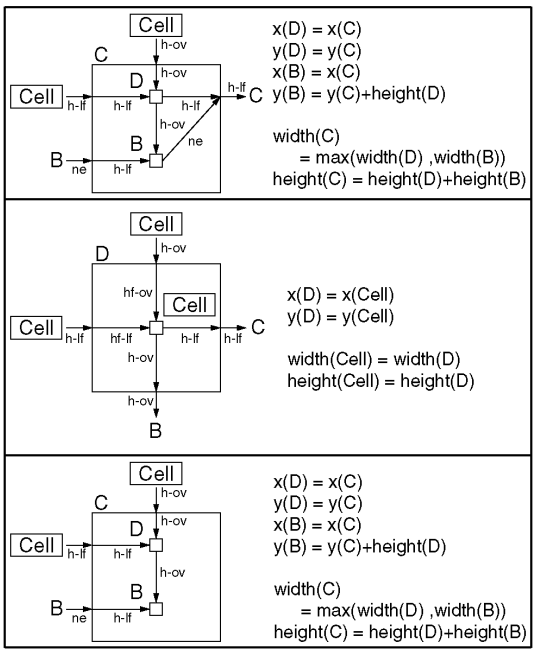


Figure 8. Part of the productions and semantic rules in HTGG.

Tessellation diagrams generated by HTGG is classified into row-oriented tessellation diagrams and column-oriented diagrams.

5. Diagram Processing System

This section describes a diagram processing system which is called KEYAKI-CASE2000. The KEYAKI-CASE2000 consists of the following components: (1) Hichart program diagram editing component (HichartED), (2) Hichart program diagram filtering component (HiTS), (3) Program variable analyzing component (LIVE), and (4) Hiform diagram component (HiformED).

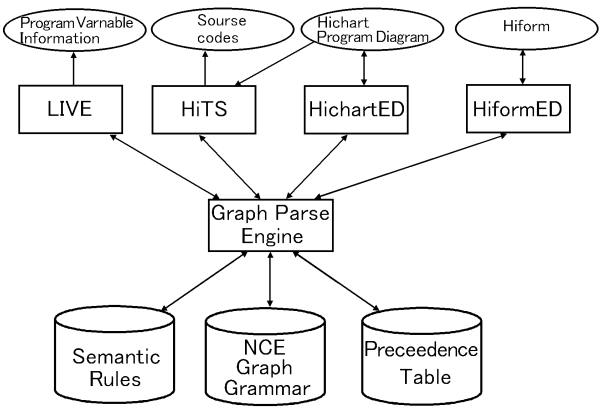


Figure 9. System Structure of KEYAKI-CASE2000.

These components are based on the theoretical research described in Sections 3 and 4. Figure 9 shows the system structure of KEYAKI-CASE2000.

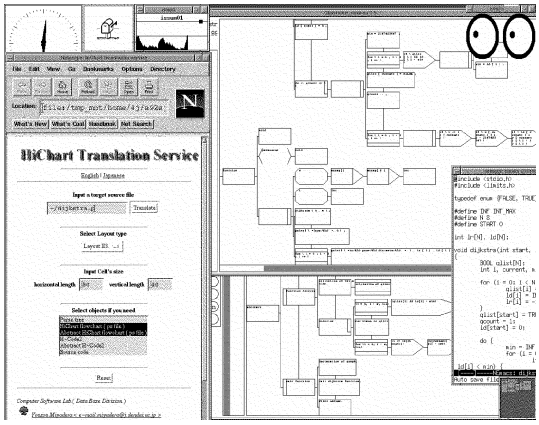


Figure 10. Execution Screen of HiTS.

We describe each component in detail as follows.

5.1 HichartED

The Hichart program diagram editing component is called HichartED. HichartED is a syntax-directed diagram editor using an attribute graph grammar [12]. The Hichart editor-commands are defined by productions of the attribute graph grammar which formally defines the Hichart program diagram. This guarantees that any grammatically correct diagram can be generated and that there will be no syntax errors in generation of the program and editing processes with the editor.

5.2 HiTS

The Hichart program diagram filtering component is called HiTS (Hichart Translation service) [7,9]. The theoretical research concerning the HiTS was described in Section 3. HiTS is a program flowchart processing system that automatically generates tree-structured program flowcharts (Hichart)[4] incorporating the results of the theoretical research which is a tidy drawing of trees. Specifically, the system automatically generates a program flowchart and assists the user in visualizing data structure and control flow. Moreover, the flowcharts used in HiTS can be formed from program specifications as well as from program source code (C, Pascal). The descriptions in the window used for explanatory text can be used as nodes to generate a flowchart that abstracts the program.

Figure 10 shows a program diagram (upper right window) and an abstracted program diagram (lower right window) for a C program that determines the shortest path using a Dijkstra algorithm. And, Figure 11 is also a program diagram (right most windows) for a Pascal program which is generated by HiTS system. The Pascal version of this tool was implemented in about 26,500 steps, While the C version consists of approximately 30,000 steps.

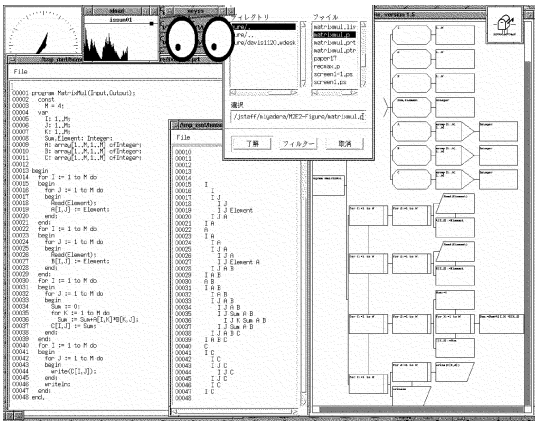


Figure 11. Execution Screen of LIVE.

5.3 LIVE

We also provide a program variable analyzing component called LIVE base on attribute grammars. LIVE was developed to support the program modularization[9]. LIVE is developed by the theoretical research which is the data-flow analysis theory. HiTS can also be used to judge whether or not appropriate program modules have been formed by simultaneously referring to the information in the LIVE. Based on the results of this evaluation, the user can modify the data structure and modules.

The result of applying the LIVE operation is shown in Figure 11. The window on the left side of the screen displays the program source code implementing matrix multiplication, while that in the middle shows the active variable derived by LIVE, and the Hichart flowchart is on the right. It is apparent from the LIVE results that the code can be broken into modules at the 30th line and at the 40th line.

This tool is written in Pascal and consists of approximately 15,000 steps.

5.4 HiformED

We are also developing a Hiform diagram processing component called HiformED. An execution screen model is shown in the following Figure 12.

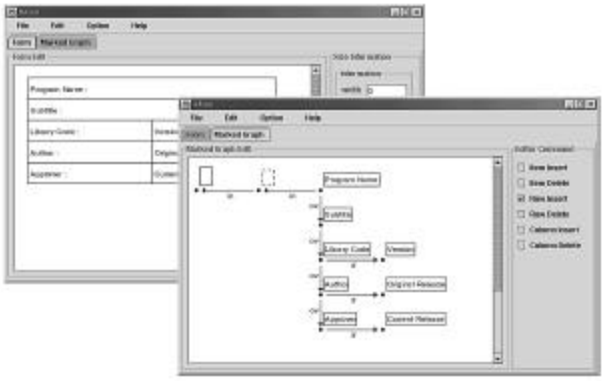


Figure 12. Execution Screen of HiformED.

6. Conclusions

We proposed an attribute context-sensitive NCE grammars as a universal model of visual processing of diagrams. We proposed an attribute NCE graph grammar with 67 productions and 723 attribute rules for Hichart program diagrams. We also proposed attribute NCE context-free graph grammar with 280 productions and 1204 attribute rules for ISO 6592 documentation with 137 items. Furthermore, we proposed an attribute context-sensitive NCE graph grammar for tessellation diagrams, which provide first concrete edge-sensitive NCE graph grammars. Processing methods were also considered.

References

[1] Reinhold Franck, A class of linearly parsable graph grammars, *Acta Infomatica* 10 (1978), 175-201.

[2] Pierluigi Della Vigna and Carlo Ghezzi, Context free graph grammars, *Inform. Contr*. 37 (1978), 207-233.

[3] ISO6592-1985, Guidelines for the documentation of computer-based application systems, (1985)

[4] T. Yaku, K. Futatsugi, A. Adachi and E. Moriya, HICHART-A hierarchical flowchart description language-, *Proc. IEEE COMPSAC* 11 (1987), 157-163.

[5] T. Nishino, Attribute graph grammars with applications to hichart program chart editors., *Advances in Software Science and Technology* 1 (1989), 426-433.

[6] K. Yamazaki and T. Yaku, A pumping lemma and structure of derivations in the boundary NLC graph languages, *Imform. Sci.*75 (1993), 81-97.

[7] Y. Miyadera, K. Tsuchida, and T. Yaku, A tidy drawing problem on the minimum area for tree-structured diagrams and Its application to program diagrams, *IFIP Transac*. A-51 (1994), 282-287.

[8] K. Tsuchida, The complexity of drawing tree-structured diagrams, *IEICE Trans. Inf. & Syst*. E78-D (1995), pp.901-908.

[9] Y. Miyadera, A. Tsuchiya, T. Yaku, and H. Konya, Network-based programming language education environment based on a modular program diagram, *Proc. IEEE International Conference on Multi Media in Education*, (1996), 425-434.

[10] Y. Adachi, K. Anzai, K. Tsuchida and T. Yaku, Hierarchical program diagram editor based on attribute graph grammar, *Proc. IEEE COMPSAC* 21 (1996), 205-213

[11] K. Sugita, Y. Adachi, Y. Miyadera, K. Tsuchida and T. Yaku, Advanced software mechanisms for computer-aided instruction in information literacy, *Proc. APEC Conf. Inform. Literacy. (APEC-CIL'97)*, (1997)

[12] Grsegorz Rozenberg (Ed.), Handbook of Graph Grammar and Computing by Graph Transformation, World Scientific Publishing (1997).

[13] Y. Miyadera, K. Anzai, H. Unno and T. Yaku, Depth-first layout algorithm for trees, *Information Processing Letters* 66 (1998), 187-194.

[14] K. Sugita, A. Adachi, Y. Miyadera, K. Tsuchida and T. Yaku, A visual programming environment based on graph grammars and tidy graph drawing, *Proc. Internat. Conf. Software Engin.* (ICSE '98) 20-II (1998), 74-79.

[15] A. Adachi, T. Tsuchida and T. Yaku, Program visualization using attribute graph grammars, *CD-ROM Proc. IFIP World Computer Congress* 98 (1998).

[16] Kimio Sugita and et al, Integrated visualization environment for computer science education, *Proc. ICEUT2000 (IFIP WCC2000)*, to appear.