A Syntax Directed Environment for Tabular Form Designing

Shun-ichi Nakagawa, Tomokazu Arita, Kiyonobu Tomiyama, Takeo Yaku
Dept. App. Math., Nihon Univ.
3-25-40, Sakurajosui,
Setagaya, Tokyo, 156-8550, Japan
+81 3 3329 1151
{nakagawa, arita, yaku}@am.chs.nihon-u.ac.jp

Youzou Miyadera[†], Kensei Tsuchida[‡]
[†]Tokyo Gakugei Univ., [‡]Toyo Univ.
[†]Koganei, Japan, [‡]Kawagoe, Japan
[†]+81 423 29 7475, [‡]+81 492 39 1434
[†]miyadera@u-gakugei.ac.jp
[‡]kensei@eng.toyo.ac.jp

ABSTRACT
We deal with mechanical documentation in software development tools. First, we review tabular forms for program specification and their formal syntax by an attribute NCE graph grammar. Next we introduce parser based on the syntactic definitions and attribute rules. Furthermore, we introduce syntactic editor of tabular forms. Finally, we introduce system structure of whole processing system for mechanical documentation. These results are applied to mechanical manipulation for general tabular forms.

Keywords
software documentation, visualization, graph grammars, tabular forms

1 INTRODUCTION

Mechanical documentation such as automatic drawing and editing of program specification forms is one of the important problems in software development tools. Software documentation includes tabular forms such as program specification forms and tabular forms such as program flowcharts. This paper deals with tabular forms and their mechanical manipulating problems.

In mechanical manipulation of tabular forms, it is necessary to explicitly define both the syntax and drawing conditions (cf. [2]). Attribute graph grammars formulate syntactic structure. They also formulated visual structure among items in forms, universally by syntax with attribute rewriting rules. Franck [1] introduced precedence graph grammars and applied them to nested program tabular forms called PLAN2D. We formulated the hierarchical structured program tabular form [4, 7].

In 2000, we introduced partly a syntactic definition of program specification forms based on ISO6592 standard [6]. It is to be noted that those forms are represented by attribute marked graphs. For this purpose, we employed graph grammars. In this paper, we propose a universal processing system for tabular forms. Accordingly, we employ as the common models attribute NCE graph grammars.

In Section 2, we review tabular forms for program specification and introduce a formal syntax of those forms by attribute NCE graph grammars. In Section 3, we introduce a parser for tabular forms, which provides mechanical verifier and mechanical drawer. In Section 4, we introduce a syntax editor of tabular forms based on syntactic definition of editing manipulations. In Section 5, we propose a syntactic processing system using above methods. Section 6 provides conclusions.

2 TABULAR FORMS AND THEIR SYNTAX

In this Section, we review tabular forms for program specification.

Tabular Forms for Program Specification

We consider here a program specification language called Hiform [6] based on ISO6592 [3].

Project Code:	A 5
Program Name:	**Program Specification-1** p
Library Code:	Version:
Author:	Original Release:
Approver:	Current Release:
Problem Description:	
Problem Supplementary Information (Theoretical Principles, Methods and References):	
Problem Solution: 1.Conventions and Terminology 2.Principles and Algorithms	

Figure 1. Hiform program specification form

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 Hiform, which includes all items defined in these Annexes. Hiform [6] is defined by 17 types of forms. Figure 1 shows a Hiform program specification form.

The order among tabular forms was already defined by a context-free string grammar [6]. The order and graphical structure of cells inside tabular forms will be discussed later.

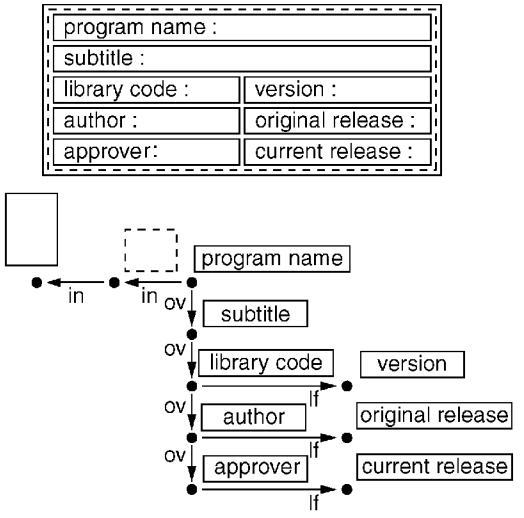

Figure 2. Nested tabular form and its corresponding marked graph

Hiform is characterized by graph grammars for graph syntax and attribute rules for drawing conditions. In the graph grammar of Hiform, a program form is represented by a marked graph with locations. We illustrate examples in Figure 2.

EdNCE Graph Grammars [8]
The following provides a formal model of tabular forms.

Definition An *edNCE graph grammar* is a 6-tuple $G = (\Sigma, \Delta, \Gamma, \Omega, P, S)$, where Σ is the *alphabet of node labels*, $\Delta \subseteq \Sigma$ denote the alphabet of *terminal* node labels, Γ is the alphabet of *edge* labels, $\Omega \subseteq \Gamma$ is the alphabet of *final* edge labels, P is the finite set of *productions* and $S \in \Sigma - \Delta$ is the *initial nonterminal*. □

Tabular forms in software documentation are classified into (1)*nested* tabular forms (see Figure 2.) and (2)*tessellation* tabular forms (see Figure 8.) such as *symbol* tables. We characterize the former (1) by the following *Hiform Nested Graph Grammar* (HNGG) and the latter (2) by the following *Hiform Tessellation Graph Grammar* (HTGG).

HNGG [8, 9]
In this part, we consider an *attribute edNCE* graph grammar **HNGG** $= < G_N, A_N, F_N >$ that generates nested tabular forms in Hiform *form*. *Underlying* graph grammar $G_N = (\Sigma_N, \Delta_N, \Gamma_N, \Omega_N, P_N, S_N)$ is an edNCE *context-free* graph grammar.

The following Figure 3 illustrates productions of HNGG. Each production has *attribute rules* for drawing information. The HNGG includes 280 productions and 1248 attribute rules for the definition of nested graph part such as the program form.

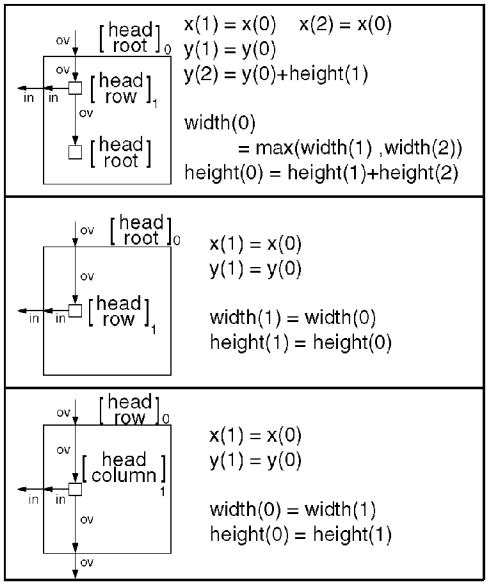

Figure 3. Productions of HNGG

Property Grammar HNGG is a precedence graph grammar [1]. Thus, the parsing algorithm of Hiform is given by the Franck's linear time parsing algorithm [8].

HTGG [8, 11]
We consider *tessellation* tabular form such as symbol tables in specification form in Hiform. HTGG is a *context-sensitive* attribute NCE graph grammar for the tessellation tabular forms in Hiform. The grammar HTGG includes 69 syntax rules and 308 attribute rules. Following Figure 4 shows productions of HTGG. Tessellation tabular form and its corresponding marked graph is shown in Figure 8 in Appendix.

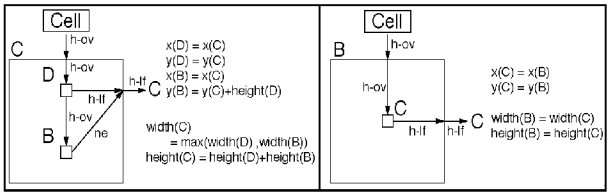

Figure 4. Productions of HTGG

3 TABULAR FORM PARSER

First, by a parsing for marked graph, derivation tree is generated. Next, by an attribute evaluation for derivation tree, attribute marked graph is generated. Attribute marked graph has layout information. Tabular form is generated with this flow. We show the *parsing process* in the following Figure 5.

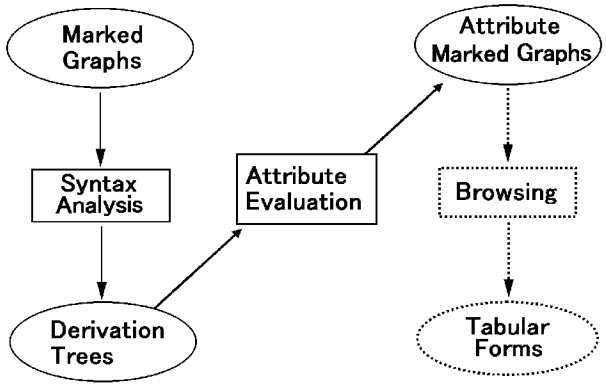

Figure 5. Parsing Process

4 TABULAR FORM EDITOR

Syntactic Editing for HNGG [10]

Formalization of syntax directed editing of tabular forms is based on the following notation of production instance.

A *production instance* ("instance" for short) is a 3-tuple (ω, p_i, H'_{p_i}), where

1. $\omega \in V_{D_{i-1}}$ is a node removed during the derivation $D_{i-1} \Rightarrow_{p_i} D_i$.
2. $p_i \in P$ is a production.
3. H'_{p_i} is an embedded graph isomorphic to H_{p_i} during $D_{i-1} \Rightarrow_{p_i} D_i$.

We denote $D_{i-1} \overset{\omega H'_{p_i}}{\Rightarrow_{p_i}} D_i$ if D_{i-1} directly derives D_i by applying instance (ω, p_i, H'_{p_i}).

Syntactic Insertion

Insertion is defined as substitution of the source sequence of production instance to a production instances in the target sequences of productions instances. Other manipulations are similarly defined by HNGG. Those manipulations generate only the grammatically correct forms.

Claim If a graph H' is obtained from a target graph H by insertion, H' is grammatically correct with respect to HNGG.

Editor

Our syntax directed editor executes editing manipulations on sequences of production instances. The following Figure 6 illustrates insertion process of the editor.

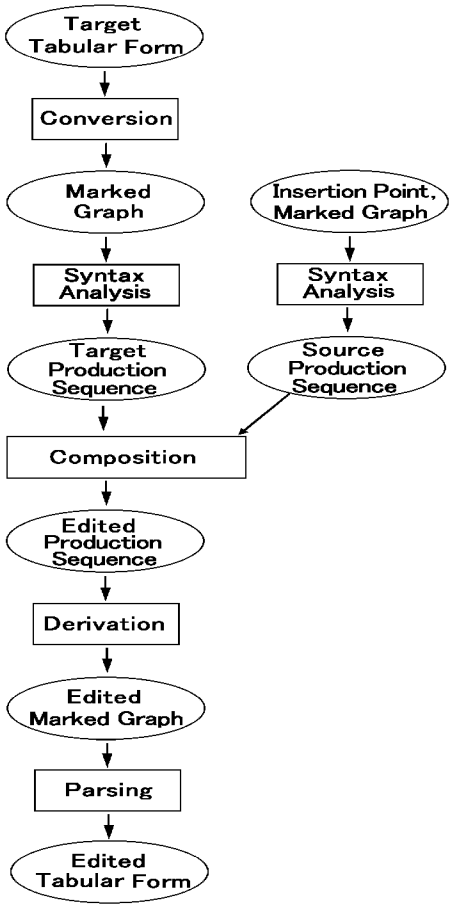

Figure 6. Insertion Process

5 SYSTEM OVERVIEW

This system is constructed on a graph parsing engine for Hiform. This engine is constructed on three parts, which are productions for tabular form syntax, attribute rules for calculating values of tabular form's layout information and precedence table for tabular form parsing.

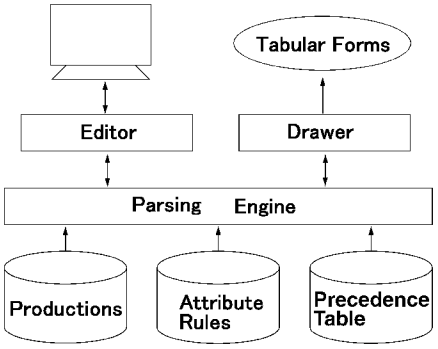

Figure 7. System Overview

Figure 7 illustrates the System Overview, and the following Table 1 shows the features. An execution screen is shown in Figure 9 in the Appendix.

Editor		under development
Drawer	3k	lines
Parsing Engine	2k	lines
Production	349	rules
Attribute Tree	1556	rules
Precedence Table	5376	relations

Table 1. Feature of Environment

6 RELATED WORKS [5]

Graph manipulating systems such as graph editors and graph drawing systems using combinatorial and constraint algorithms have been 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. Nagl et al. introduced IPSEN systems.

7 CONCLUSIONS

We proposed syntactic tabular form designing environment (cf. [8]), based on attribute NCE graph grammars. Similarly, We are investigating environment for tessellation forms based on HTGG.

ACKNOWLEDGEMENTS

We thank Mr. S. Kanai's advise in the course of preparing the manuscript. We also thanks to Mr. K. Sugita for valuable discussions.

REFERENCES

[1] Reinhold Franck, A Class of Linearly Parsable Graph Grammars, Acta Infomatica 10, 175-201 (1978)

[2] Tim Teitelbaum and Thomas Reps, The Cornell Program Synthesizer: A Syntax-Directed Programming Environment, *Comm. ACM*, Vol.24, 563-573, (1981).

[3] ISO6592-1985, Guidelines for the Documentation of Computer-Based Application Systems, (1985).

[4] Y. Adachi, K. Anzai et al., Hierarchical Program Diagram Editor Based on Attribute Graph Grammar, *Proc. COMPSAC*96, 205-213(1996).

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

Advanced Software Mechanisms for Computer-Aided Instruction Information Literacy, *APEC-CIL*'97, (1997).

[6] 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, 74-79 (1998).

[7] A. Adachi, T. Tsuchida and T. Yaku, Program Visualization Using Attribute Graph Grammars, CD-ROM Book, *IFIP World Computer Congress* 98, (1998).

[8] T. Arita, K. Tomiyama, T. Yaku, Y. Miyadera, K. Sugita, K. Tsuchida, Syntactic Processing of Diagrams by Graph Grammars, *Proc. IFIP WCC ICS 2000*,145-151 (2000).

[9] T. Arita, A Precedence Attribute NCE Graph Grammar for Hiform, http://www.hichart.org/keyaki/archive/HC00-001

[10] K. Tomiyama et al., Syntactic Editing for Nested Tabular Forms, http://www.hichart.org/keyaki/archive/HC00-002

[11] T. Arita et al., A Context-Sensitive Attribute NCE Graph Grammar for Tabular Form, http://www.hichart.org/keyaki/archive/HC00-003

APPENDIX

Name	Type	Size
X	int	1
Y	float	2

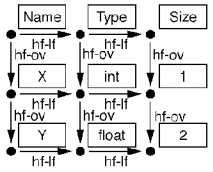

Figure 8. Tessellation tabular form and its corresponding marked graph

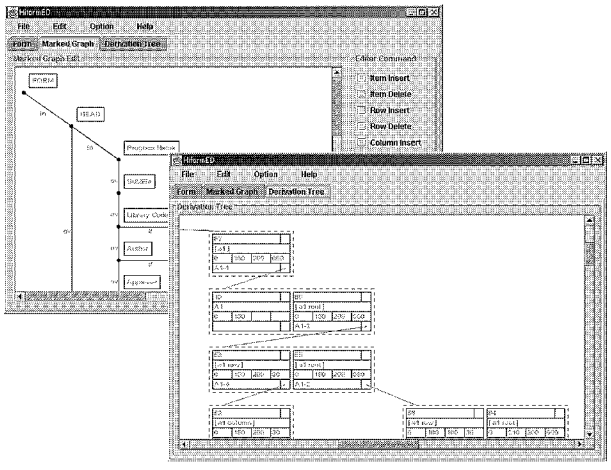

Figure 9. An Execution Screen of the Editor: A Derivation Tree and Its Derived Attribute Marked Graph