Electronic Notes in Theoretical Computer Science 50 No. 3 (2001) – Proc. GT-VMT 2001
URL: http://www.elsevier.nl/locate/entcs/volume50.html 7 pages

Application of Attribute NCE Graph Grammars to Syntactic Editing of Tabular Forms

Tomokazu ARITA [a,1] Kiyonobu TOMIYAMA [a,2]
Kensei TSUCHIDA [b,3] Takeo YAKU [a,4]

[a] *Dept. Comput. Sci. and System Analysis, Nihon University*
3–25–40, Sakurajosui, Setagaya, Tokyo, 156-8550, Japan

[b] *Dept. Inf. and Comp. Sci., Toyo University*
2100, Kujirai, Kawagoe, Saitama, 350–8585, Japan

Abstract

In this paper, we deal with editing tabular forms for program specifications based on a particular graph grammar HNGG [2]. First, we formalize syntax-directed editing methods by extending of the notion of the Cornell Program Synthesizer [8] to attribute NCE graph grammars (cf. [1]). Next, we discuss the algorithms of the editing methods.

Key words: Graph Grammars, Visual Programming, Software Development, Syntax-Directed Editors

1 Introduction

Mechanical editing of tabular forms is one of the important issues in software engineering methodology. The Cornell Program Synthesizer is well-known and is often referred to as a structured and text-based editor which uses an attribute grammar successfully [8]. Tabular forms are represented by several different models (e.g., Pane [6]). We assigned each item in the tabular form to an attributed node. This assignment naturally represents the order of items and location of items in the tabular form. Since the number of items in the form is generally unbound and the order of items has some valid meaning, tabular forms are denoted by a graph grammar [2]. Accordingly, the mechanical

[1] Email: arita@cssa.chs.nihon-u.ac.jp
[2] Email: tomiyama@cssa.chs.nihon-u.ac.jp
[3] Email: kensei@eng.toyo.ac.jp
[4] Email: yaku@cssa.chs.nihon-u.ac.jp

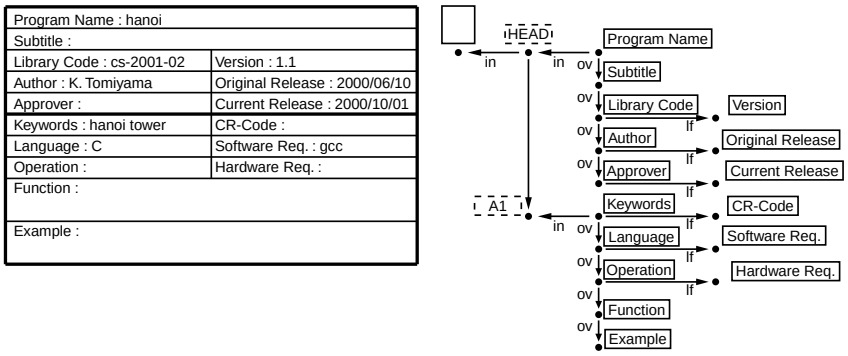

Fig. 1. Tabular form in a Hiform document and its corresponding graph.

editing of tabular forms is supposed to be executed by some syntactic editing methods.

In this paper, we consider a programming documentation, *Hiform*, as an example of tabular forms. Hiform document is a collection of 17 types of tabular forms and includes all items defined in the ISO6592 guideline [9],[2]. It should be noted that certain ISO6592 tabular forms are regarded as tabular forms, since they have modular structures. Such tabular forms are represented by graphs. Fig. 1 illustrates a Hiform form and its corresponding graph. This graph is constructed as follows: (1) A node label of the graph shows the type of an item of a tabular form. (2) An edge label shows relations between items. 'lf' denotes the meaning of 'left of', 'ov' denotes the meaning of 'over', and 'in' denotes the meaning of 'within'.

It is supposed that a mechanical processing of tabular forms can be realized effectively by syntactic manipulation of graphs. In [2], the inner structure of each form in Hiform is defined by an attribute NCE graph grammar. Hiform forms are specified by graphs that are generated its grammar.

The purpose of this paper is to extend Cornell Program Synthesizer mechanisms to graphs using the results reported in [1] and [2] and to formalize a syntactic editing mechanism for graphs. Insertion in HNGG [3] is defined so that this manipulation is validly executed by the confluence [7] of HNGG.

In Section 2, preliminary definitions are given. In Section 3, a formal definition for editing mechanisms is given by using instance sequences [1]. And we also show the validity of our definition by using the confluence of HNGG. Section 4 is devoted to our concluding remarks.

2 Preliminaries

2.1 edNCE Graph Grammars [7]

Let Σ be an alphabet of *node labels* and Γ be an alphabet of *edge labels*. A *graph* over *alphabets* Σ and Γ is a 3-tuple $H = (V, E, \lambda)$, where V is a finite nonempty *set of nodes*, $E \subseteq \{(v, \gamma, w) \mid v, w \in V, v \neq w, \gamma \in \Gamma\}$ is a *set of edges*, and $\lambda : V \to \Sigma$ is a node *labeling function*.

Definition 2.1 An *edNCE graph grammar* is a 6-tuple $G = (\Sigma, \Delta, \Gamma, \Omega, P, S)$, where Σ is the alphabet of node labels, $\Delta \subseteq \Sigma$ is 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*. A production is denoted by the form $p : X \to (D, C)$, where $X \in \Sigma - \Delta$, D is a graph over Σ and Γ, and $C \subseteq \Sigma \times \Gamma \times \Gamma \times V_D \times \{in, out\}$ is the *connection relation*. □

2.2 Composition of Production Copies [1]

The composite representation of the production copies of an edNCE graph grammar is a theoretical and practical method for representing the graph-rewriting rules for embedding subgraphs of desired structures into a graph.

Definition 2.2 [1] Let $G = (\Sigma, \Delta, \Gamma, \Omega, P, S)$ be an edNCE graph grammar. Let $p_1 : X_1 \to (D_1, C_1)$ $(D_1 = (V_{D_1}, E_{D_1}, \lambda_{D_1}))$ and $p_2 : X_2 \to (D_2, C_2)$ $(D_2 = (V_{D_2}, E_{D_2}, \lambda_{D_2}))$ be production copies of G. If $u \in V_{D_1}$ and $X_2 = \lambda_{D_1}(u)$, and D_1 and D_2 are disjoint, then a *composite production copy* (with a connection relation) $p : X_1 \to (D, C)$ is defined as follows: D is a graph as $V_D = \{V_{D_1}\text{-}\{u\}\} \cup V_{D_2}$ about nodes. $C = \{ (\sigma, \beta / \gamma, \omega, d) \in C_1 \mid \omega \in V_{D_1} \text{-} \{u\} \} \cup \{ (\sigma, \beta / \delta, y, d) \mid \exists \gamma \in \Gamma, (\sigma, \beta / \gamma, u, d) \in C_1, (\sigma, \gamma / \delta, y, d) \in C_2 \}$ The composite production copy p composed by p_1 and p_2, and denoted by $p_1 \circ p_2$. □

2.3 Confluence Property [7]

The confluence property guarantees that the result of a derivation shall not depend on the order of the applications [7] of the production. Confluence is a very important property because it guarantees the validity of the application of the composite production copies. The confluence is also important when developing efficient parsing algorithms.

Definition 2.3 [7] An edNCE graph grammar $G = (\Sigma, \Delta, \Gamma, \Omega, P, S)$ is *dynamically confluent* if the following holds for every intermediate graph H generated by G: if $H \Rightarrow_{u_1,p_1} H_1 \Rightarrow_{u_2,p_2} H_{12}$ and $H \Rightarrow_{u_2,p_2} H_2 \Rightarrow_{u_1,p_1} H_{21}$ $(p_1, p_2 \in P)$ are derivations of G with $u_1, u_2 \in V_H$ and $u_1 \neq u_2$, then $H_{12} = H_{21}$. □

2.4 Attribute NCE Graph Grammars [2]

We review an attribute graph grammar for the mechanical drawing of tabular forms. An *attribute NCE graph grammar* is given as follows.

Definition 2.4 [2] An attribute NCE Graph Grammar is a 3-tuple $AGG =< G, Att, F >$ where $G = (\Sigma, \Delta, \Gamma, \Omega, P, S)$ is a context-free edNCE graph grammar, called an *underlying graph grammar* of AGG. Here Att is the *set of attributes* of AGG, and F is *the set of semantic rules of AGG*. □

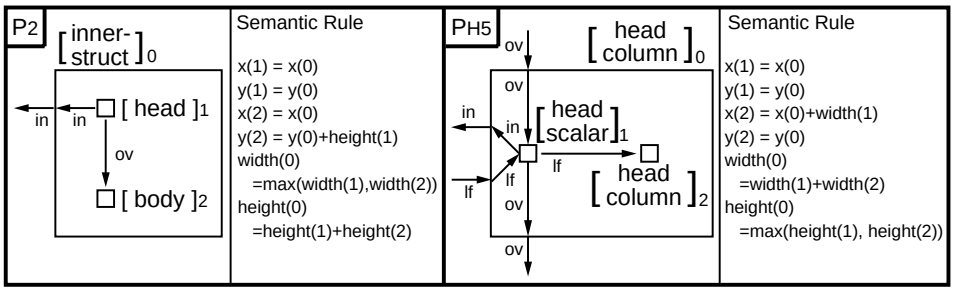

Fig. 2. Part of the productions of HNGG.

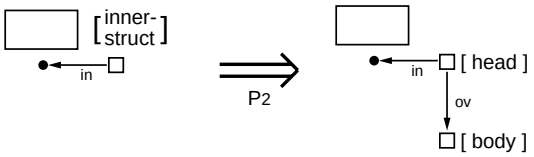

Fig. 3. An example of applying a production P_2.

2.5 HNGG [2] [3] [4]

We review an attribute NCE graph grammar for tabular forms. The grammar is called a *Hiform Nested tabular form Graph Grammar (HNGG). HNGG* $= < G_N, A_N, F_N >$ that generates modular tabular forms called *Hiform form* where $G_N = (\Sigma_N, \Delta_N, \Gamma_N, \Omega_N, P_N, S_N)$ is the underling edNCE graph grammar. Each production has a semantic rule for drawing information. The HNGG includes 280 productions and 1248 attribute rules. Fig. 2 illustrates a part of the productions with attribute rules of HNGG. We write productions like style of the edNCE graph grammar. Fig. 3 is an example of applying a production P_2. By applying a production P_2, a node labeled "inner–struct" is replaced to a graph of right hand side of P_2.

3 Editing of Modular Tabular Forms

In this section, we present a formal definition for editing manipulation by using production instances of HNGG, and we also show the validity of our definition by using confluence of HNGG.

3.1 Production Instance

The editing manipulations are defined by production instance as follows.

Definition 3.1 A *production instance* ("instance" for short) is a 3-tuple $(\omega, p_i, \overline{H_{p_i}})$, where (1) $\omega \in V_{D_{i-1}}$ is a node removed during the derivation $D_{i-1} \Rightarrow_{p_i} D_i$, (2) $p_i : X_{p_i} \rightarrow (H_{p_i}, C_{p_i}) \in P$ is a production, and (3) $\overline{H_{p_i}}$ is an embedded graph isomorphic to H_{p_i} during $D_{i-1} \Rightarrow_{p_i} D_i$.

We denote $D_{i-1} \overset{\omega \overline{H_{p_i}}}{\Rightarrow_{p_i}} D_i$ if D_i is directly derived from D_{i-1} by applying the instance $(\omega, p_i, \overline{H_{p_i}})$. □

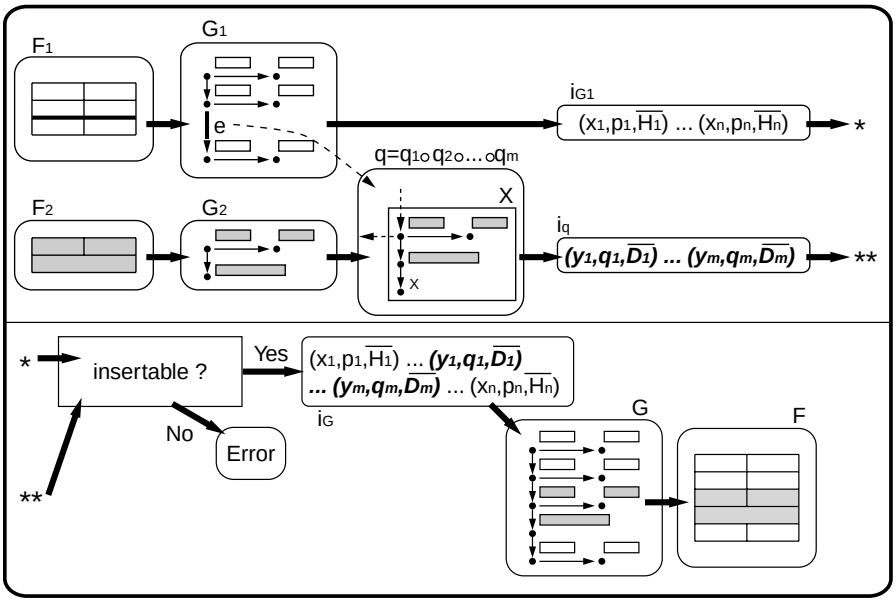

Fig. 4. Flow of an insertion process.

If there is a production sequence $p = (p_1, \cdots, p_n)$ and instance $(\omega_i, p_i, \overline{H_{p_i}})$ for each production p_i $(1 \leq i \leq n)$, an *instance sequence* is a sequence of $((\omega_1, p_1, \overline{H_{p_1}}), \cdots, (\omega_n, p_n, \overline{H_{p_n}}))$.

A derivation of an edNCE graph grammar is represented by a derivation tree. However, a derivation sequence is convenient as a representation of a derivation for a prosessing model. Therefore, we represent a derivation with a production sequence.

3.2 Syntactic Insertion

In this section, we define the syntactic insertion. This manipulation is based on HNGG. Syntax-directed editing is defined by using instance sequences.

Definition 3.2 For a derivation sequence $D_0 \overset{\omega_1 \overline{H_{p_1}}}{\Rightarrow_{p_1}} \cdots \overset{\omega_{i-1} \overline{H_{p_{i-1}}}}{\Rightarrow_{p_{i-1}}} D_{i-1} \overset{\omega \overline{H_{p_i}}}{\Rightarrow_{p_i}} D_i \overset{\omega_{i+1} \overline{H_{p_{i+1}}}}{\Rightarrow_{p_{i+1}}} \cdots \overset{\omega_n \overline{H_{pn}}}{\Rightarrow_{p_n}} D_n$ with instance $(p_j : X_{p_j} \rightarrow (H_{p_j}, C_{p_j}), 1 \leq j \leq n)$, we assume that q is *insertable* (for p_i) if there is an instance $(\omega, q, \overline{H_q})$ $(q : X_q \rightarrow (H_q, C_q) \in P_N)$ such that $D_{i-1} \overset{\omega \overline{H_q}}{\Rightarrow_q} Q$ and if there is a derivation sequence $D_{i-1} \overset{\omega \overline{H_q}}{\Rightarrow_q} Q \overset{\omega' \overline{H_{p_i}}}{\Rightarrow_{p_i}} D'_i \overset{\omega_{i+1} \overline{H_{p_{i+1}}}}{\Rightarrow_{p_{i+1}}} \cdots \overset{\omega_n \overline{H_{pn}}}{\Rightarrow_{p_n}} D'_n$ where ω' is a node in Q, a node label of ω' is left hand side of p_i. □

Definition 3.3 For a production $q : X_q \rightarrow (H_q, C_q) \in P_N$, which is insertable for $p_i : X_{p_i} \rightarrow (H_{p_i}, C_{p_i})$ and $\bigcup_{i=1}^{n} \overline{H_{p_i}} \cap \overline{H_q} = \phi$, an instance sequence S is obtained by *insertion* of an instance $(\omega, q, \overline{H_q})$ into an instance sequence $((\omega_1, p_1, \overline{H_{p_1}}), \cdots, (\omega_n, p_n, \overline{H_{p_n}})) \overset{def}{\Leftrightarrow} S = ((\omega_1, p_1, \overline{H_{p_1}}), \cdots, (\omega_{i-1}, p_{i-1}, \overline{H_{p_{i-1}}}), (\omega, q, \overline{H_q}), (\omega, p_i, \overline{H_{p_i}}), \cdots, (\omega_n, p_n, \overline{H_{p_n}}))$. The instance sequence S is given

as follows. (1) Trace the derivation sequence with instance D_n back to D_{i-1}. (2) Apply the instance $(\omega, q, \overline{H_q})$ to D_{i-1}, and obtain the resultant graph Q. (3) Apply the instance sequence ($(\omega', p_i, \overline{H_{p_i}})$, $(\omega_{i+1}, p_{i+1}, \overline{H_{p_{i+1}}})$, $\cdots$, $(\omega_n, p_n, \overline{H_{p_n}})$) to Q, and get the resultant graph D'_n. □

Inserting some instances into an instance sequence brings a new item into existence. That is, they correspond to a manipulation to insert a new item into a permissible place in a Hiform document.

Remark 3.4 In the same manner as the editing by using the instance of a production, we can further define *insertable by composite production copy.* □

Definition 3.5 A graph H' is *obtained by syntactic insertion* of a graph A at an edge x in a graph H, if the following conditions hold: (1) A composite production copy q for graph A and edge x exists. (2) There exists an instance sequence i_q for q and an instance sequence i_H for H. An instance sequence S is obtained by insertion of i_q into an instance sequence i_H. (3) The graph H' is derived from instance sequence S. □

Proposition 3.6 *Let H be the graph obtained from G by the insertion of graph a and graph b at edge x and edge y respectively in this order in HNGG. Let H' be the graph obtained from G by the insertion of b and a at y and x respectively in this order in HNGG. Then, $H = H'$.*

Proof. HNGG has a confluence property. Thus, the proposition is verified.□

Proposition 3.7 *Insertion in HNGG is executed in linear time.*

Proof. An insert point of a production instance is found in linear time for the length of an embedded production sequence. Let n be the number of nodes in a target graph. In a derivation of our HNGG, any node in a target graph is changing to a terminal node by at most five application. Therefore, the maximum length of an instance sequence for the target graph is $5n$. Since HNGG is a precedence edNCE graph grammar [2], syntax analysis is executed in linear time [5]. Attribute evaluation is also executed in linear time [2]. □

An example of an insertion is given in Fig. 4. Here, we insert a form F_2 into a form F_1. Let G_1 be a graph for F_1, and let G_2 be a graph for F_2. Then, a syntactic insertion of G_2 at edge e in G_1 is done as follows: (1) A composite production copy q for G_2 is obtained from e and G_2. (2) An instance sequence i_q for q and an instance sequence i_{G_1} existed. If q is insertable for p in i_{G_1}, an instance sequence i_G is obtained by insertion of i_q into an instance sequence i_{G_1}. (3) A graph G is generated from this instance sequence i_G. G is a new form which is obtained by inserting F_2 into F_1.

4 Conclusion

Spread sheets and software documents are used for software visualization widely. Our results are a proposal of a theoretical model such visualization.

We proposed an editing method, based on attribute NCE graph grammar of tabular forms with a homogenous cell size. This method includes attribute rules for mechanical drawing. It can exactly edit valid tabular forms defined by edNCE graph grammar. A linear time editing algorithm with attribute rules for primitive drawing exists.

These syntactic editing methods could be applied to syntactic manipulation of spreadsheet languages. We are reconstructing attribute rules for more sophisticated drawing. We are investigating other edit manipulations such as a division manipulation, a combination manipulation and so on. Furthermore we are now developing a tabular-form editor system.

Acknowledgement

We thanks Prof. K. Sugita for his valuable suggestions. We thank Mr. S. Kanai for his advice in the course of preparing the manuscript. We also thank Mr. S. Nakagawa and Mr. K. Ruise for their valuable discussions.

References

[1] Adachi, Y., K.Anzai, et al. Hierarchical Program Diagram Editor Based on Attribute Graph Grammar, *Proc. COMPSAC96* (1996), 205-213.

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

[3] Arita, T., K. Sugita, K. Tsuchida, T. Yaku, Syntactic Tabular Form Processing by Precedence Attribute Graph Grammars, *Proc. IASTED Applied Informatics 2001* (2001), 637-642.

[4] Arita, T., A Precedence Attribute NCE Graph Grammar for Hiform, (2000), URL: `http://www.hichart.org/` or `http://www.cssa.chs.nihon-u.ac.jp/~yaku/keyaki/archive/HC00-001`

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

[6] Pane, F. John, Brad A. Myers, Tabular and Textual Methods for Selecting Objects from a Group, *Proc. 2000 IEEE Symp. on Visual Language* (2000), 157-164.

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

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

[9] ISO 6592–1985, Guidelines for the Documentation of Computer–Based Application Systems, (1985).