属性edNCEグラフ文法による表の構文的編集

冨山聖宣 † 有田友和 † 土田賢省 ‡ 夜久竹夫 †

† 日本大学文理学部応用数学科
156-8550 東京都世田谷区桜上水 3-25-40
‡ 東洋大学工学部情報工学科
350-8585 埼玉県川越市鯨井 2100
Email: † {tomiyama,arita,yaku}@am.chs.nihon-u.ac.jp
‡kensei@krc.toyo.ac.jp

図や表は、ソフトウェア視覚化などのマルチメディアソフトウェア工学について重要な役割を果たしている。我々は、[4] でグラフ文法に基づく木構造図に対する構文指向編集コマンドを導入し、[10] で表の構文を導入した。我々は、この論文で階層型表の構文的編集を扱う。ただし、表は、マーク付き木とみなされていることに注意する。ここでは、機械的描画を考慮して、属性 edNCE グラフ文法に基づく構文指向的な編集操作を定式化する。さらに、機械的描画について構文編集コマンドの有効性を示す。

視覚的プログラミング，ソフトウェア開発，属性グラフ文法，構文指向エディタ

Syntactic Editing of Tabular Forms by Attribute edNCE Graph Grammars

Kiyonobu Tomiyama† Tomokazu Arita† Kensei Tsuchida‡ Takeo Yaku†

†Department of Applied Mathematics,College of Humanities and Sciences, Nihon University.
3-25-40, Sakurajosui, Setagaya, Tokyo, 156-8550
‡Department of Information and Computer Sciences, Toyo University.
2100, Kujirai, Kawagoe, Saitama, 350-8585
Email: † {tomiyama,arita,yaku}@am.chs.nihon-u.ac.jp
‡kensei@krc.toyo.ac.jp

Tabular forms such as program specification forms [10] are naturally formalized by the attribute graphs [10], in which the attribute denotes locations of items and while the edge labels denotes relations between items. Documents of the tabular forms are represented by graph grammars (e.g., see [10]). Accordingly, a syntactic formalization of document editing provides the foundation for mechanical documentation. In this paper, first, we formalize syntax directed editing methods by extention of the notion of Cornell Program Synthesizer[2] to attribute edNCE graph grammars (cf.[4]). Next, we show the validity of our definition for editing a process under HNGG [10] using the confluence of HNGG. Furthermore, we provide an examples.

Visual programming, software development, graph grammars, syntax directed editors

1 Introduction

Mechanical editing of tabular forms is one of the important issues in the software engineering methodology. The Cornell Program Synthesizer (CPS) is well-known and is often referred to as a structured and text-based editor which uses an attribute grammar successfully [2]. Tabular forms are represented by several different models (e.g., Pane represents them by [9]). 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 infinite and the order of items has some valid meaning, tabular forms are denoted by graph grammars [10]. Accordingly, the mechanical editing of tabular forms supposed to be executed by some syntactic editing methods.

Program name : hanoi	
Subtitle :	
Library code : cs-2000-02	Version : 1.1
Author : K. Tomiyama	Original release :2000/6/10
Approver :	Current release :2000/10/1
Key words : hanoi tower	CR-code :
Scope : fundamental	
Variant :	
Language : C	Software req. : gcc
Operation :	Hardware req. :
References :	
Function : 1. List and Explanation of Input Data. ...	
Example : 1.Example of Operation ...	

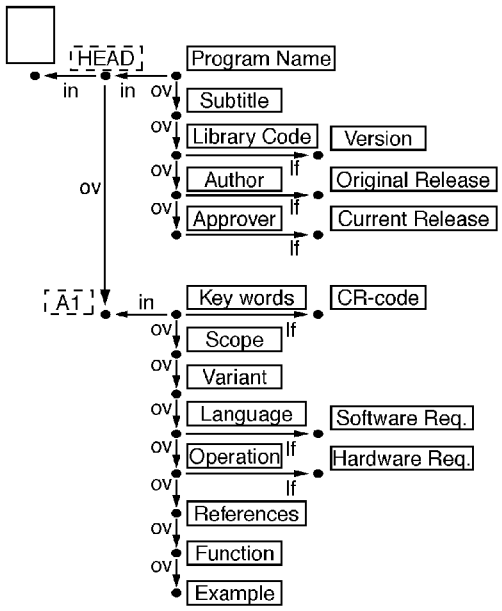


Figure 1: Tabular form in Hiform document and its corresponding graph

In this paper, we consider a programming documentation *Hiform* as an example of the tabular forms. Hiform document is a collection of 17 types of the tabular forms and includes all items defined in the guideline in ISO6592 [3],[5]. It is to be noted that certain ISO6592 tabular forms are regarded as tabular forms, since they have nested structures. Those tabular forms are represented by graphs. Fig.1 illustrates a Hiform document and its corresponding graph. This graph is decided as follows: (1) A node label of graph shows an item of a tabular form. (2) An edge label shows relations between items.

A mechanical processing of tabular forms supposed to be realized effectively by syntactic manipulation of graphs. In [10], the inner structure of each form in Hiform is defined by an attribute edNCE graph grammar.

The purpose of this paper is to extend CPS mechanism to graph using results in [4][10] and to formalize a syntactic editing mechanism for graphs, in this connection some examples are given in consideration with software documentation tabular forms. Definition is made as to insertion and deletion in HNGG [10] so that two manipulations are validly executed by the confluence[6] of HNGG.

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

2 Preliminaries

We review context-free edNCE graph grammars [6] and an attribute edNCE graph grammar [10]. It is to be noted that the edNCE graph grammar allows for new edges to be established only between neighboring nodes and embedded nodes as specified by connection instructions in the embedding process.

2.1 EdNCE Graph Grammars [6]

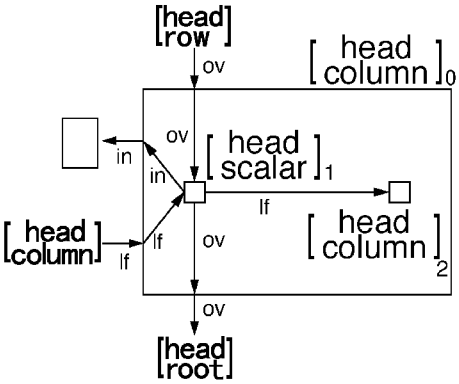
A *graph* over *alphabets* Σ and Γ is a 3-tuple $H = (V, E, \lambda)$, where V is a finite nonempty *set of nodes*, $E \subseteq \{(v, \gamma, w) \mid v, w \in V, v \neq w, \gamma \in \Gamma\}$ is a *set of edges* and $\lambda : V \to \Sigma$ is a node *labeling function*. Let Σ be an alphabet of node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

The set of all *concrete* graphs over Σ and Γ is denoted by $GR_{\Sigma,\Gamma}$, and the set of all *abstract* graph is denoted by $[GR_{\Sigma,\Gamma}]$. A subset of $[GR_{\Sigma,\Gamma}]$ is called a *graph language*. The set of all *graphs with embedding* over Σ and Γ is denoted by $GRE_{\Sigma,\Gamma}$.

Definition An *edNCE graph grammar* is a 6-tuple $G = (\Sigma, \Delta, \Gamma, \Omega, P, S)$, where Σ is the alphabet of *node labels*, $\Delta \subseteq \Sigma$ 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 $X \in \Sigma - \Delta$, $(D, C) \in GRE_{\Sigma,\Gamma}$, $D \in GR_{\Sigma,\Gamma}$ and $C \subseteq \Sigma \times \Gamma \times \Gamma \times V_D \times \{in, out\}$: *connection relation*. Each element $(\delta, \beta, \gamma, x, d)$ of C (with $\delta \in \Sigma$, $\beta, \gamma \in \Gamma$, $x \in V_D$, $d \in \{in, out\}$) is a *connection instruction* of (D, C). To improve readability, a connection instruction $(\delta, \beta, \gamma, x, d)$ will always be written as $(\delta, \beta/\gamma, x, d)$. The following Fig.2 shows a production of edNCE graph grammar.



C={([$head - row$], ov/ov, 1, in), ($\square$, in/in, 1, out),
([$head - column$], lf/lf, 1, in),
([$head - root$], ov/ov, 1, out)}

Figure 2: Production of edNCE graph grammar

The process of rewriting on an edNCE graph grammar is defined through the notion of substitution, in a standard theoretic language manner: Let $sn(S.z)$ denote the graph with a single S-labeled node z, no edges, and empty connection relation. The *graph language generated by* G is

$L(G) \stackrel{def}{=} \{[H] | H \in GR_{\Delta,\Omega}$ and $sn(S, z) \Rightarrow^* H$ for some $z\}$.

2.2 Compositions of Production Copies [4]

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 sub-graphs of desired structures into a graph.

Definition[4] 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 $p_2 : X_2 \to (D_2, C_2)$ $(X_2 = \lambda_{D_1}(u), D_2 = (V_{D_2}, E_{D_2}, \lambda_{D_2}))$ be production copies of G. If $u \subseteq V_{D_1}$, then a *composite production copy* (with a connection relation) $p : X_1 \to (D, C)$ is defined as follows:
D is a graph as $V_D = \{V_{D_1} - \{u\}\} \cup V_{D_2}$ about nodes.
$C = \{(\sigma, \beta/\gamma, \omega, d) \in C_1 | \omega \in V_{D_1} - \{u\}\}$
$\cup$ $\{(\sigma, \beta/\delta, y, d) | \exists \gamma \in \Gamma, (\sigma, \beta/\gamma, u, d) \in C_1, (\sigma, \gamma/\delta, y, d) \in C_2\}$
The composite production copy p composed by p_1 and p_2, and denoted by $p_1 \circ p_2$.

□

The composite production copy $p_1 \circ p_2$ is also called *definable* in this case.

Proposition [4] 1: Each production is a composite production copy. 2: p_1 and p_2 are production copies and $p_1 \circ p_2$ is definable. Then, $p_1 \circ p_2$ is composite production copy.

□

2.3 Confluence Property [6]

In general, the resulting graph of derivation based on an edNCE graph grammar depends on the order by which the production copies were applied.

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

Definition [6] An edNCE graph grammar $G = (\Sigma, \Delta, \Gamma, \Omega, P, S)$ is *dynamically confluent* if the following holds for every sentential form H of 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 (creative) derivation of G with $u_1, u_2 \in V_H$ and $u_1 \neq u_2$, then$H_{12} = H_{21}$.

□

2.4 Attribute edNCE Graph Grammars [10]

We review an attribute graph grammar for the mechanical drawing. An *attribute edNCE graph grammar* is as follows.

Definition [10] An attribute edNCE Graph Grammar is a 3-tuple $AGG =< G, Att, F >$ where
(1)$G = (\Sigma, \Delta, \Gamma, \Omega, P, S)$ is context-free edNCE graph grammar, called an *underlying graph grammar* of AGG. Each production p in P is denoted by $p : X \to (D, C)$. $Lab(D)$ denotes the set of all occurrences of the node symbols labeling the nodes in the graph D.

(2) Each node symbol $Y \in \Sigma$ of G has two disjoint finite sets $Inh(Y)$ and $Syn(Y)$ of *inherited* and *synthesized attributes*, respectively. We denote the set of all attributes of nonterminal node symbols Y by $Att(Y) = Inh(Y) \cup Syn(Y)$. $Att = \bigcup_{Y \in V} Att(Y)$ is called the *set of attributes* of AGG. We assume that

$Inh(S) = \phi$. An attribute a of Y is denoted by $a(Y)$, and a set of possible values for a is denoted by $V(a)$.

(3) Associated with each production $p : X_0 \rightarrow (D, C) \in P$, there exists a set F_p of *semantic rules* which define all the attributes in $Syn(X_0) \cup \bigcup_{Y \in Lab(D)} Inh(Y)$. A semantic rule defining an attribute $a_0(X_{i0})$ has the form $a_0(X_{i0}) := f(a_1(X_{i1}), \cdots, a_m(X_{im}))$, $0 \leq i_j \leq |Lab(D)|$, $X_{ij} \in Lab(D)$, $0 \leq j \leq m$. Here $|Lab(D)|$ denotes the cardinality of the set $Lab(D)$, and f is a mapping from $V(a_1(X_{i1}) \times \cdots \times a_m(X_{im}))$ into $V(a_0(X_{i0}))$. In this situation, we say that $a_0(X_{i0})$ depends on $a_j(X_{ij})$ for j, $0 \leq j \leq m$ in p. The set $F = \bigcup_{p \in P} F_p$ is called *the set of semantic rules of AGG*.

□

2.5 HNGG [10]

In this section, we consider an attribute edNCE graph grammar. The grammar is called *Hiform Nested Graph Grammar* (HNGG). **HNGG** $= < G_N, A_N, F_N >$ that generates nested tabular forms called Hiform form. *Underlying* graph grammar $G_N = (\Sigma_N, \Delta_N, \Gamma_N, \Omega_N, P_N, S_N)$ is a context-free edNCE graph grammar. The following Fig.3 illustrates productions of HNGG.

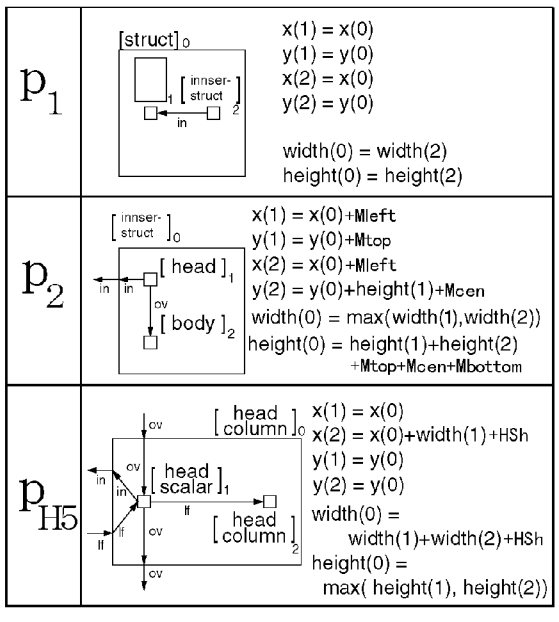


Figure 3: Productions with semantic rules of HNGG

Proposition 2.2 [10] HNGG is precedence edNCE graph grammar.

□

Proposition 2.3 [10] HNGG has confluence property.

□

3 Editing of Nested Tabular Form

In this section, we deal with a formal definition for editing manipulation using production instance of HNGG, and we also show the validity of our definition using confluency of HNGG.

3.1 Production Instance

We introduce editing manipulations in latter part. The editing manipulations are exactly defined by production instance as follows. In this part, we introduce 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. $p_i : X_{p_i} \rightarrow (H_{p_i}, C_{p_i}) \in P$ 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} is directly derived D_i by applying the instance (ω, p_i, H'_{p_i}).
If there is a production sequence $p = (p_1, \cdots, p_n)$ and instance $(\omega_i, p_i, H'_{p_i})$ for each production p_i, an *instance sequence* is a sequence of $((\omega_1, p_1, H'_{p_1}), \cdots, (\omega_n, p_n, H'_{p_n}))$.

3.2 Syntactic Insertion

In this part, we define the syntactic insertion, and denote the flow of insertion manipulation. This manipulation is based on HNGG. Syntax directed editing is executed by sequences of production instances.

Definition For an derivation sequence $D_0 \overset{\omega_1 H'_{p_1}}{\Rightarrow}_{p_1} \cdots \overset{\omega_{i-1} H'_{p_{i-1}}}{\Rightarrow}_{p_{i-1}} D_{i-1} \overset{\omega H'_{p_i}}{\Rightarrow}_{p_i} D_i \overset{\omega_{i+1} H'_{p_{i+1}}}{\Rightarrow}_{p+1} \cdots \overset{\omega_n H'_{p_n}}{\Rightarrow}_{p_n} D_n$ ($p_j : X_{p_j} \rightarrow (H_{p_j}, C_{p_j}), 1 \leq j \leq n$), we say that q is *insertable* (for p_i) if there is an instance (ω, q, H'_q) ($q : X_q \rightarrow (H_q, C_q) \in P_N$) such that $D_{i-1} \overset{\omega H'_q}{\Rightarrow}_q Q$ and if there is a derivation sequence such that $D_{i-1} \overset{\omega H'_q}{\Rightarrow}_q Q \overset{\omega' H'_{p_i}}{\Rightarrow}_{p_i} D'_i \overset{\omega_{i+1} H'_{p_{i+1}}}{\Rightarrow}_{p_{i+1}} \cdots \overset{\omega_n H'_{p_n}}{\Rightarrow}_{p_n} D'_n$. Furthermore, if a production $q : X_q \rightarrow (H_q, C_q)$ is insertable for p_i, and any instance sequence applicable to D_i can be applied to D'_i, then q is *strictly insertable* (for p_i).

□

If a production $q : X_q \rightarrow (H_q, C_q) \in P_N$ is strictly insertable for $p_i : X_{p_i} \rightarrow (H_{p_i}, C_{p_i})$ and $\bigcup_{i=1}^{n} H'_{p_i} \cap H'_q = \phi$, then *insertion of an instance* (ω, q, H'_q) *into an instance sequence* $((\omega_1, p_1, H'_{p_1}), \cdots, (\omega_n, p_n, H'_{p_n}))$ makes an instance sequence

$((\omega_1, p_1, H'_{p_1})$, $\cdots$,$(\omega_{i-1}, p_{i-1}, H'_{p_{i-1}})$, (ω, q, H'_q) , $(\omega'_i, p_i, H'_{p_i})$, $\cdots$, $(\omega'_n, p_n, H'_{p_n}))$ which derives from a graph D'_n in the following step, where $H'_q = (V_{H'_q}, E_{H'_q}, \lambda_{H'_q})$, $\omega' \in H'_q, X_{p_i} = \lambda_{H'_q}(\omega')$.

1. Trace the derivation sequence D_n back to D_{i-1}.
2. Apply the instance (ω, q, H'_q) to D_{i-1} , and obtain the resultant graph Q .
3. Apply the instance sequence ($(\omega'_i, p_i, H'_{p_i})$, $(\omega'_{i+1}, p_{i+1}, H'_{p_{i+1}})$, $\cdots$, $(\omega'_n, p_n, H'_{p_n})$) to Q, and get the resultant graph D'_n.

Inserting some instances into an instance sequence bring 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.

Definition In the same manner as the editing by the instance for a production, we can further define *insertable by composite production copies.*

□

Definition To *insert a source graph A at edge x in a target graph H* is defined as follows.
1. A composite production copy q for the graph A exists.
2. The composite production copy q can be inserted at the edge x in the graph H.
3. $H \rightarrow \overset{q}{\cdots} \rightarrow H'$: H' is the *inserted graph* which inserts the graph A at the edge x in the graph H.

□

The following Fig.4 illustrates insertion process of the editor.

3.3 Syntactic Deletion of Item

We define here the deletion of inner most item in the form, that is a leaf node of the marked tree.

Definition For a derivation sequence $D_0 \overset{\omega_1 H'_{p_1}}{\Rightarrow_{p_1}} \cdots \overset{\omega_k H'_{p_k}}{\Rightarrow_{p_k}} F \overset{\omega H'_p}{\Rightarrow_p} D_p \overset{\omega_l H'_{p_l}}{\Rightarrow_{p_l}} \cdots \overset{\omega_n H'_{p_n}}{\Rightarrow_{p_n}} D_n$, let a graph D_p be the first graph of the instance sequence in which a node $u \in V_{D_p}$ is brought into existence and the node u is not applied to any production after that.
A production $p : X_p \rightarrow (H_p, C_p) \in P_N$ such that $H_p = (V_{H_p}, E_{H_p}, \lambda_{H_p})$ is *deletable* if one of the following Assumptions $1-3$ is met.

<Assumption 1>: For $p \in P_N$, there exists a production $p' : X_{p'} \rightarrow (H_{p'}, C_{p'}) \in P_N$ such that $H_{p'} = (V_{H_{p'}}, E_{H_{p'}}, \lambda_{H_{p'}})$ which satisfies the followings.

1. $X_{p'} = X_p$
2. $V_{H_{p'}} \equiv V_{H'_p} - \{u\}$
3. If f and g are isomorphic mappings such that
   $f : V_{H'_p} \rightarrow V_{H_p}$
   $g : V_{H_p - \{f(u)\}} \rightarrow V_{H_{p'}}$,
   then $(\sigma, \beta/\gamma, y, d) = (\sigma, \beta/\gamma, g(y), d)$

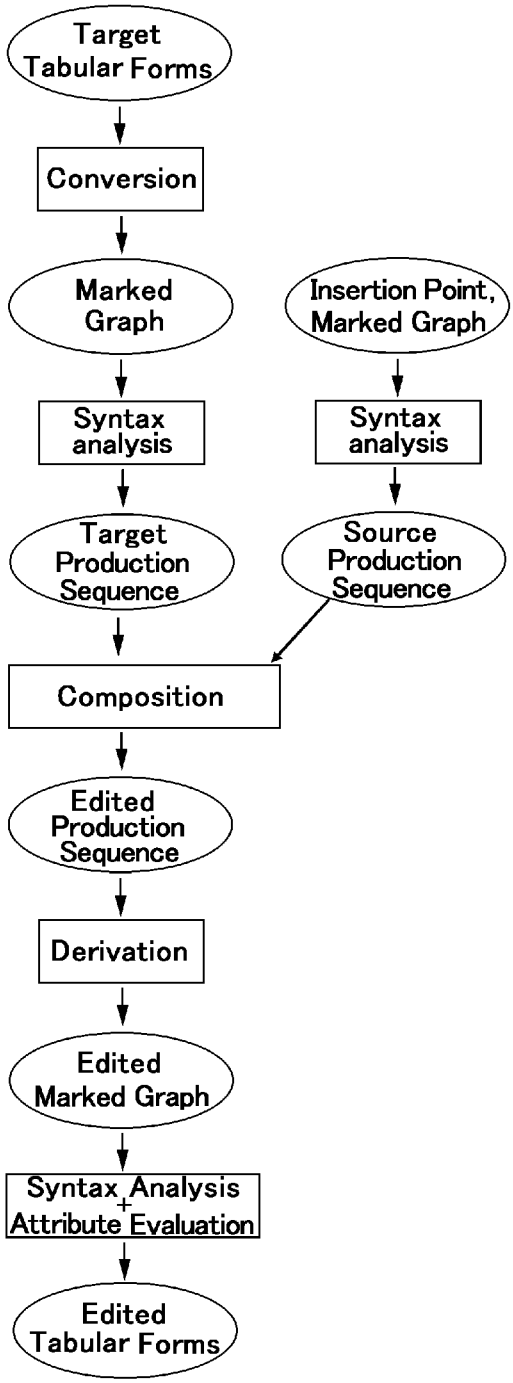


Figure 4: A process flow for an insertion of Hiform editing system

<Assumption 2>: $V_{H'_p} = \{u, v\}$, $X_p = \lambda_{H'_p}(v)$

<Assumption 3>: $\omega_j \notin H'_p, l \leq j \leq n$

□

If a production $p \in P_N$ is deletable, *deletion of an instance* (ω, p, H'_p) *from an instance sequence* $\mathcal{L}$=$((\omega_1, p_1, H'_{p_1}), \cdots, (\omega_k, p_k, H'_{p_k}), (\omega, p, H'_p)$, $(\omega_l, p_l, H'_{p_l})$, $\cdots, (\omega_n, p_n, H'_{p_n}))$ makes an instance sequence $((\omega_1, p_1, H'_{p_1}), \cdots, (\omega_k, p_k, H'_{p_k}),$ $(\omega_l, p_l, H'_{p_l}), \cdots, (\omega_n, p_n, H'_{p_n}))$ or $((\omega_1, p_1, H'_{p_1}),$ $\cdots, (\omega_k, p_k, H'_{p_k}), (\omega, p', H'_{p'}), (\omega_l, p_l, H'_{p_l})$ $, \cdots, (\omega_n, p_n, H'_{p_n}))$ such that $H'_{p'} = (V_{H'_p} - \{u\}, E_{H'_{p'}}, \lambda_{H'_{p'}})$, which derives at a graph D'_n as follows.

The case satisfying the Assumption 1:

1. Trace the derivation sequence D_n back to F.
2. Apply the instance $(\omega, p', H'_{p'})$ to F, so as to obtain the resultant graph $D'_{p'}$.
3. Apply the instance sequence $((\omega_l, p_l, H'_{p_l}), \cdots, (\omega_n, p_n, H'_{p_n}))$ to $D'_{p'}$, so as to obtain the resultant graph D'_n.

The case satisfying the Assumption 2:

1. Trace the derivation sequence D_n back to F.
2. Rename the node $\omega \in V_F$ as v, so as to obtain the resultant graph F'.
3. Apply the instance sequence $((\omega_l, p_l, H'_{p_l}), \cdots, (\omega_n, p_n, H'_{p_n}))$ to F', so as to obtain the resultant graph D'_n.

The case satisfying the Assumption 3:

1. Trace the derivation sequence D_n back to F.
2. Apply the instance sequence $((\omega_l, p_l, H'_{p_l}), \cdots, (\omega_n, p_n, H'_{p_n}))$ to F, so as to get the resultant graph D'_n.

The deletion of an instance (ω, p, H'_p) from an instance sequence $((\omega_1, p_1, H'_{p_1}), \cdots, (\omega_n, p_n, H'_{p_n}))$ matches pruning or replacement of a derivation tree.

Definition To *delete a node A from a graph H* is defined as follows.
1. A production q having a node A on the right hand side exists.
2. The production q is deletable in the instance sequence for graph H.
3. $H \rightarrow \overset{q}{\cdots} \rightarrow H'$: H' is the *deleted graph* which deletes the node A in the graph H.

□

3.4 Deletion of Blocked Items

We extend here deletion of a node (an item) defined above to deletion of a subgraph (items) from the given marked graph.

The *deletion of* the subgraph is defined by using the removal of instance defined in the 3.3. Let $D = (V_D, E_D, \lambda_D)$ be a graph. Let $T \subseteq D$ be a subgraph. Then, the deletion of the production about the derivation of T has been performed as follows.

1. $D' = D$
2. Let $T \subseteq D'$ be a subgraph
3. In derivation sequence $D_0 \Rightarrow \cdots \Rightarrow D'$, q can be removed from the production in T
   (a) If q exists, the graph which removed q from the sequence, and then renew D' and return to 2
   (b) It is finished if q does not exist

3.5 Property of Editing Method

We can edit Hiform documents by simply using composite production copies.
Next, we discuss properties of the editing method utilized in the nested diagrams. We can show the following.

Theorem 3.1 Deletion (insertion, block delition) in HNGG is executed in linear time.

The following Theorem guarantees the effectiveness of our definitions on editing.

Theorem 3.2 Let H be the graph obtained from G by the deletion of nodes a and b in this ouder, in HNGG. Let H' be the graph obtained from G by the deletion of nodes b and a in this order, in HNGG. Then, $H = H'$.

We have same proposition about insertion and block deletion.

4 Conclusion

We are investigating detailed algorithm. We are investigating other edit manipulation that are a division manipulation, a combination manipulation and so on. Furthermore we are now developing a tabular form editor system by using this approach.

Acknowledgment
We thank Mr. S. Kanai's advise in the course of preparing the manuscript.
We also thanks to Mr. S. Nakagawa and Mr. K. Ruise 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] K. Sugita, Y. Adachi, Y. Miyadera, K. Tsuchida and T. Yaku, Advanced Software Mechanisms for Computer-Aided Instruction in Information Literacy, *APEC-CIL*'97, (1997).

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

[7] 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).

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

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

[10] 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).

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