An Attribute Precedence Graph Grammar and Tabular Forms

Tomokazu Arita
Kiyonobu Tomiyama

Abstract This paper characterizes graph grammars which provide formal definition of program documentation tabular forms with respect to syntactic manipulation and mechanical drawing. We propose an attribute context-free graph grammar with 280 rewriting rules and 1248 attribute rules for ISO 6592 based nested program forms with 137 items. The grammar is shown to have precedence property [1] by 5376 relations over the marks. Furthermore, we consider context-sensitive graph grammars for tessellation tabular forms.

Keywords graph grammars, program documents, form layout.

1 Introduction

Graph grammars have been studied and utilized, by several authors, for their possible association with program diagrams, generation of general diagrams and computer aided design for the industrial objects. (see e.g. [1],[2])

This paper deals with tabular forms for program specification and its syntactic definition with respect to the mechanical drawing. Items in program specification documents were generally listed in [3]. The program specification documents are usually represented by tabular forms [5].

We came to notice that tabular forms can generally be represented by graphs. Thus, in this paper we regard the tabular forms as nested diagrams and represent nested diagrams by marked graphs.

In [1], Franck employed marked graphs for nested diagrams, introduced a precedence graph grammar for the marked graphs and formalized parsing of nested diagrams. Nishino [4] introduced an attribute graph grammar with respect to a drawing problem of tree-like diagrams and formalized transformation of tree-like diagrams. In [4], the drawing problems were specified by semantic rules of attributes. We have also studied syntactic and algorithmic manipulation of diagrams [6], [7] [8]. The purpose of this paper is to characterize graph grammars which provide formal definition of program specification forms with respect to syntactic manipulation and mechanical drawing.

2 Preliminaries

2.1 Program Documentation Language Hiform96

We introduce here a program documentation system called Hiform96 [6] based on ISO6592 [3].

The International Organization for Standardization issued a guideline in ISO6592
and described all items in program documentation in Annex A, B and C. Hiform96 includes all items defined in these Annexes. Hiform96 is defined by 17 types of forms. The Fig. 2.1 shows a Hiform96 program documentation form.

| program name | hanoi_main |
| library code | cs : 2000 - 01 |
| version | 1.0 |
| author | Tomohiko Arita |
| original release | 1999/12/22 |
| current release | 2000/01/28 |
| key words | Hanoi Tower |
| CR-code | |
| scope | Fundamental |
| variant | |
| language | Java |
| software req | JDK 1.2 |
| hardware req | |
| function | 1. list and explanation of input data or parameter, 2. list and explanation of output data or return value. |
| example | |

Fig. 2.1 A program documentation of Hiform96.

The order among tabular forms is defined by a context-free string grammar [5].

2.2 Nested Diagrams for Tabular Forms

We use a nested diagram as a formalization of a tabular form document. The following Fig. 2.2 illustrates the nested diagram that represents a tabular form.

2.3 Marked Graphs for Nested Diagrams

We introduce a marked graph for a nested diagram as an example. An edge label shows relations between items. ’lf’ is the meaning of ’left of’, ’ov’ is the meaning of ’over’. ’in’ is the meaning of ’within’.

Definition 2.3.1 [1]. A marked graph is a
system \((K, R, k, r)\) where \(K\) is a finite set of nodes, \(K \neq \emptyset, R \subseteq K \times K, k : K \rightarrow V\) a mapping for marking the nodes, \(r : R \rightarrow M\) a mapping for labeling the edges.

2.4 Context-Free Graph Grammar

We survey here context-free graph grammars [1] and precedence grammars [1]

Definition 2.4.1 [1]. A (context-free) production is a 4-tuple \(p = (A, H, p^x, p^s)\), where \(A\) is a single node graph (the left-hand side of \(p\)), \(H = (K_h, R_h, k_h, r_h)\) is a nonempty graph (the right-hand side of \(p\)), and \(p^x, p^s : M \rightarrow K_h\) are partial functions where \(M\) is the set of all labels for edges.

Definition 2.4.2 [1]. A context-free graph grammar is a system \(GG = (V, T, M, P, S)\), where \(V\) is a finite set of alphabet, i.e. a set of symbols for labeling the nodes, \(T \subset V\) is a set of the terminal symbols, \(M\) is a finite set of labels for the edges, \(P\) is a finite set of productions of the form \(p = (A, H, p^x, p^s)\) explained above, \(S \in V - T\) is the start symbol, i.e. the start graph for \(GG\).

Notation 2.4.3 [1]. For \(m \in M\) let

\[\begin{align*}
\vdash_m & \overset{\text{def}}{=} \{(A, B) \mid A, B \in V \text{ and there exists a rule } p \text{ with an edge } (x, y) \text{ on the right-hand side where } x \text{ is labeled by } A, \text{ and } (x, y) \text{ has label } m.\} \\
\Rightarrow_m & \overset{\text{def}}{=} \{(A, B) \mid A, B \in V \text{ and there is a rule } p = (A, H, p^x, p^s) \text{ and } B \text{ is the label of the node } p^s(m) \text{ in } H\}
\end{align*}\]

Notation 2.4.4 [1]. For \(m \in M\) let

\[\begin{align*}
<_{m} & \overset{\text{def}}{=} \vdash_m \cdot \vdash_m \\
>_m & \overset{\text{def}}{=} \vdash_m \cdot \vdash_m \\
<_{m} & \overset{\text{def}}{=} \vdash_m \cdot \vdash_m
\end{align*}\]

where + denotes transitive closure.

Precedence relations are conflictless if and only if for every \(m \in M\) the relations \(<_{m}, \vdash_m, >_m\) and \(<_{m}\) are pairwise disjoint [1].

Definition 2.4.5 [1]. A context-free graph grammar is called a precedence grammar if and only if (i) the precedence relations are conflictless. (ii) all rules are uniquely invertible. (iii) there is no reflexive nonterminal symbol in the grammar.

3 Attribute Precedence Graph Grammars for Hi-form

3.1 Definitions for Attribute Graph Grammar

We introduce an another type of graph grammars for formalization of tabular forms based on [1] and [4].

Definition 3.1.1 (cf. [1], [4]) An attribute graph grammar is a 3-tuple \(AGG = (GG, Att, F)\), where
1. \( GG = (V,T,M,P,S) \) is called an underlying context-free graph grammar of AGG. Each production \( p \) in \( P \) is denoted by \( p = (A,H,p^e,p^r) \). \( \text{Lab}(H) \) denotes the set of all occurrences of the node symbols labeling the nodes in the graph \( H \).

2. Each node symbol \( X \in V \) of \( GG \) has two disjoint finite sets \( \text{Inh}(X) \) and \( \text{Syn}(X) \) of inherited and synthesized attributes, respectively. We denote the set of all attributes of nonterminal node symbols \( X \) by \( \text{Att}(X) = \text{Inh}(X) \cup \text{Syn}(X) \). \( \text{Att} = \bigcup_{X \in V} \text{Att}(X) \) is called the set of attributes of \( AGG \). We assume that \( \text{Inh}(S) = \emptyset \). An attribute \( a \) of \( X \) is denoted by \( a(X) \), and set of possible values of \( a \) is denoted by \( V(a) \).

3. Associated with each production \( p = (X_0,H,p^e,p^r) \in P \) is a set \( F_p \) of semantic rules which define all the attributes in \( \text{Syn}(X_0) \cup \bigcup_{X \in \text{Lab}(H)} \text{Inh}(X) \). A semantic rule defining an attribute \( a_0(X_{i_0}) \) has the form \( a_0(X_{i_0}) := f(a_1(X_{i_1}), \ldots, a_m(X_{i_m})) \), \( 0 \leq i_j \leq |\text{Lab}(H)| \), \( X_{i_j} \in \text{Lab}(H), 0 \leq j \leq m \). Here \( |\text{Lab}(H)| \) denotes the cardinality of the set \( \text{Lab}(H) \), and \( f \) is a mapping from \( V(a_1(X_{i_1}) \times \ldots \times a_m(X_{i_m})) \) into \( V(a_0(X_{i_0})) \).

In this situation, we say that \( a_0(X_{i_0}) \) depends on \( a_j(X_{i_j}) \) for \( j, 1 \leq j \leq m \) in \( p \). The set \( F = \bigcup_{p \in P} F_p \) is called the set of semantic rules of \( AGG \).

The characterized forms are called Hiform2000.

The grammar which formalizes Hiform2000 is called Hiform Attribute Graph Grammar(HFAGG). We show productions of HFAGG in Table 3.1. HFAGG consists of 280 productions. The mark of the start graph is "[struct]".

We also construct 1248 semantic rules of HFAGG as shown in Table 3.1.

**Proposition 1.** The grammar HFAGG above is an attribute precedence graph grammar.

**Proof.** We can construct 5376 relations over the marks in HFAGG as shown in Table 3.2. The relation are shown to be pairwise disjoint. \( \square \)

**Remark.** We can implement a linear time parser[1] for the underlying graph grammar of HFAGG. \( \square \)

### 3.3 The Layout Problem of Hiform

Layout problems of nested diagrams are solved by attribute evaluation [4]. We use attributes which are \( x, y, \text{width} \) and \( \text{height} \). Symbols \( x \) and \( y \) are used to calculate \( x \) coordinate and \( y \) coordinate, respectively. And \( \text{width} \) and \( \text{height} \) are also used to calculate width and height, respectively.

**Proposition 2.** Attributes in HFAGG are evaluated in linear time. \( \square \)

### 3.2 An Attribute Precedence Graph Grammar for Hiform

We propose an attribute graph grammar which characterize the Hiform documents.

4 Tessellation Forms

We consider here tessellation forms that represent tables such as symbol tables. We
note that ISO6592 does not issue about any symbol tables. We introduce an attribute NCE context-sensitive graph grammar that generates tessellation forms.

Fig. 4.1 A tessellation form and its corresponding marked graph.

4.1 NCE Graph Grammars

\( \Sigma \) is the alphabet of node labels, \( \Gamma \) is the alphabet of edge labels. The set of all (concrete) graphs over \( \Sigma \) and \( \Gamma \) is denoted \( GR_{\Sigma,\Gamma} \).

A graph with (neighborhood controlled) embedding over \( \Sigma \) and \( \Gamma \) is a pair \((H, C)\) with \( H \in GR_{\Sigma,\Gamma} \) and \( C \subseteq \Sigma \times \Gamma \times \Gamma \times V_H \times \{in, out\} \). \( C \) is the connection relation of \((H, C)\), and each element \((\sigma, \beta, \gamma, x, d)\) of \( C \) (with \( \sigma \in \Sigma, \beta, \gamma \in \Gamma, x \in V_H \), and \( d \in \{in, out\} \) is a connection instruction of \((H, C)\).

The set of all graphs with embedding over \( \Sigma \) and \( \Gamma \) is denoted \( GRE_{\Sigma,\Gamma} \).

Definition 4.1.1 [9]. An edNCE grammar is a tuple \( G = (\Sigma, \Delta, \Gamma, \Omega, P, S) \), where \( \Sigma \) is the alphabet of node labels, \( \Delta \subseteq \Sigma \) is the alphabet of terminal node labels, \( \Gamma \) 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 of the form \( X \rightarrow (D, C) \) with \( X \in \Sigma - \Delta \) and \( (D, C) \in GRE_{\Sigma,\Gamma} \).
4.2 A Context-sensitive Attribute Graph Grammar for Tessellation Forms

We consider here an edNCE context-sensitive graph grammar for tessellation forms. We extend edNCE graph grammars and introduce a context-sensitive attribute graph grammar called an attribute NCE graph grammar. We show productions of an attribute NCE graph grammar TFAGG in Fig. 4.2 for tessellation forms.

![Diagram](image)

**Fig. 4.2 Productions of TFAGG.**

5 Conclusion

We suggested a graph grammar that characterizes ISO6592 based program documentation forms with respect to both the logical and visual structures. We are now developing a software documentation system utilizing our suggested approach in this paper.

Acknowledgment We thank Professor K.

Sugita of Tokai University, Professor K. Tsuchida of Toyo University, and Professor T. Yaku of Nihon University for valuable suggestions. We also thank Mr. S. Kanai’s advice in the course of preparing the manuscript.

References