Graph Transformation with Variables

Berthold Hoffmann

Technologiezentrum Informatik, Universität Bremen,
Bremen, Germany
hof@informatik.uni-bremen.de

Abstract. Variables make rule-based systems more abstract and expressive, as witnessed by term rewriting systems and two-level grammars. In this paper we show that variables can be used to define advanced ways of graph transformation as well. Taking the gluing approach to graph transformation [7, 3] as a basis, we consider extensions of rules with attribute variables, clone variables, and graph variables, respectively. In each case, the variables in a rule are instantiated in order to obtain a set of rule instances that in turn defines the transformation relation. By combining different kinds of variables, we define very expressive rules, and reduce them to plain rules by instantiation. Since gluing graph transformation has a well developed theory, this opens the door to lift results of that theory from instances to rules with variables.

1 Introduction

Rules are frequently used in computer science, for specifying the behavior of systems in an axiomatic way. Rules do often contain variables. *Term rewriting systems*, for instance, specify the evaluation of functions by rewrite rules such as

$$\mathsf{fib}(\mathsf{s}(\mathsf{s}(\mathsf{N}))) \to \mathsf{fib}(\mathsf{s}(\mathsf{N})) + \mathsf{fib}(\mathsf{N})$$

wherein the substitution of variables like N by terms yields ground rules like

$$\mathsf{fib}(\mathsf{s}(\mathsf{s}(\mathsf{0})))) \to \mathsf{fib}(\mathsf{s}(\mathsf{s}(\mathsf{0})))) + \mathsf{fib}(\mathsf{s}(\mathsf{0}))$$

that define the term rewrite relation [19]. *Two-level grammars*, another example, derive languages of words by rules such as

$$\langle \mathsf{T}_0\ \mathsf{expression} \rangle \ ::= \ \langle \mathsf{T}_1\ \mathsf{to}\ \mathsf{T}_0\ \mathsf{operator} \rangle\ \langle \mathsf{T}_1\ \mathsf{expression} \rangle$$

wherein variables like T_0 and T_1 are substituted by words of a context-free meta grammar in order to obtain context-free production rules like

$$\langle \mathsf{bool\ expression} \rangle \ ::= \ \langle \mathsf{int\ to\ bool\ operator} \rangle\ \langle \mathsf{int\ expression} \rangle$$

which in turn define the derivation relation of the grammar [2]. In both cases, rewriting is used twice: On the *meta level*, rule are instantiated by substituting variables, producing rule instances that generate the rewrite relation on the *object level*. Variables make rules more abstract and more expressive: term rewriting

H.-J. Kreowski et al. (Eds.): Formal Methods (Ehrig Festschrift), LNCS 3393, pp. 101–115, 2005.

systems define functions on infinite sets in a finite way, and two-level grammars derive recursively enumerable languages on the basis of context-free derivation.

This paper is about the use of variables in the area of graph transformation. Surprisingly, this concept has hardly been used in the major approaches of graph transformation that are documented in the handbook [27]. Early attempts by H. Göttler [11] and W. Hesse [17] to extend two-level word grammars to graphs were not successful. This work was inspired by three papers: D. Plump and A. Habel have devised variable hyperedges as placeholders for hypergraphs [25]; N. van Eetvelde and D. Janssens have introduced variable nodes as placeholders for graphs in [32]; and, D. Plump and S. Steinert have proposed variable labels as placeholders for attribute values [26]. These proposals follow the two-level model outlined above, but are based on different approaches to graph transformation. We catch on their ideas, but reformulate them so that they coherently use a single way of graph transformation on the object level. As a common basis, we choose the well-known gluing approach to graph transformation [7, 3] (also known as the algebraic, or double-pushout approach). In addition, we define clones, which are nodes that stand for sets of similar nodes within the same framework. The unified notions of variables allow to model several advanced concepts of graph transformation, such as the connection instructions of [10], and object set nodes and path expressions devised for programmed graph transformation [30]. The two-level model is modular so that different kinds of variables can also be combined easily. In this way, even the very advanced rules of [32] can be reduced to sets of simple gluing rules. Since variable instantiation is simply defined and easily understood, the resulting definitions are easily understood as well. And, since gluing transformation, the common basis of the instantiations, has a rich theory, results of this theory can help to prove properties of the rules as well.

The paper may raise a fundamental objection: *Need rules be that sophisticated?* Indeed, generative power is no issue, as gluing rules without variables already derive the recursively enumerated languages. However, in complex applications like software refactoring [22, 32], operations can be developed and verified more easily if they are expressed as a single rule – even a complex one – rather than as programs that control the application of simple rules in order to achieve the same effect.

The paper is structured as follows. We first recall graph transformation by gluing rules with relabeling in Section 2. This is the basis for discussing the use of variables in graph transformation in Section 3. Three kinds of variables are considered in Sections 4 to 6: attribute variables, clones, and graph variables. Section 7 discusses how instantiations can be combined, and how they may help to lift properties and results known from rule instances to rules. In Section 8 we conclude with pointers to related work and a discussion of further research.

2 Basic Graph Transformation

Rules in the gluing approach to graph transformation [7, 3] shall be used as rule instances on the object level. We have chosen this approach as it is not committed to a particular notion of graph (not even to graphs!), is widely used and has a