Towards Attributed Graphs in Groove

Work in Progress

Harmen Kastenberg [1]

Department of Computer Science, University of Twente
P.O. Box 217, 7500 AE, Enschede, The Netherlands
h.kastenberg@cs.utwente.nl

Abstract

Graphs are a very expressive formalism for system modeling, especially when attributes are allowed. Our research is mainly focused on the use of graphs for system verification.

Up to now, there are two main different approaches of modeling (typed) attributed graphs and specifying their transformation. Here we report preliminary results of our investigation on a third approach. In our approach we couple a graph to a data signature that consists of unary operations only. Therefore, we transform arbitrary signatures into a structure comparable to what is called a graph structure signature in the literature, and arbitrary algebras into the corresponding algebra graph.

Key words: attributed graphs, graph transformation, algebra graph, signature structure

1 Introduction

Representing (parts of) software systems (or their states) as graphs, has proven to be a very powerful approach for specifying program structure and verifying its behavior. In the Groove project [10] we aim at the use of graphs for verifying object oriented systems. Since the state of such systems is determined on basis of the occurring objects and the values of their attributes, it is necessary to extend the Groove Tool to support the use of attributed graphs.

Although we want to stay as close as possible to currently available theory about modeling attributed graphs and specifying their transformation [7,1,6], we believe there is a simpler, more intuitive way of specifying attributed graph transformations in the context of our tool. In our investigation we focus on

[1] The author is employed in the GROOVE project funded by the Dutch NWO (project number 500.19205).

This is a preliminary version. The final version will be published in Electronic Notes in Theoretical Computer Science
URL: www.elsevier.nl/locate/entcs

minimizing tool implementation efforts and keeping transformation specification as straightforward as possible. This means that instead of both changing the graphical representation of graphs in our tool (e.g. to more UML-like structures as used in [11]) and extending the underlying tool engine to support attribution, we combine the latter with introducing some notational conventions.

In order to support graph attribution in our tool we need to introduce data type signatures and couple those to ordinary graphs, as currently available. This coupling can be established by using special edges connecting nodes from the graph-part to nodes from the data-part representing attribute values. The difference with other approaches is that in our approach the data-part is based on data type signatures consisting of unary operations only. Therefore, we transform arbitrary signatures into a structure that is comparable to what is called a graph structure signature in the literature [5] and arbitrary algebras into the corresponding algebra graph. The algebra graph contains all necessary information about the data types that are supported and provides the semantics of the operations of each of the data types.

In this report we focus on how to model and transform attributed graphs in our tool, instead of focusing on the transformation of data type signatures, although some details of this transformation will be mentioned. In the future we will work on a more precise functor specification of this transformation in order to specify the exact relation to other approaches.

This paper is structured as follows. First we give some definitions of concepts used in the rest of this work and give some insight in how we transform arbitrary signatures into the structure we prefer. Then we show how we model attributed graphs and how we specify their transformation by means of a simple example. Thereafter, we list the advantages of our approach one by one. We conclude with a short note on related work and some comments on the restrictions of our approach.

2 Signature Structure and Attributed Graphs

In this section we define the notion of *signatures* and *attributed graphs* as they are generally accepted. We also show how we transform arbitrary signatures into signatures with unary operations only and how this is done for a small example.

Traditionally (e.g. [4]), signatures are defined as follows.

Definition 2.1 (signature) A *signature* $SIG = \langle s_1, \ldots, s_n; op_1, \ldots, op_m \rangle$ consists of sorts s_i $(1 \leq i \leq n)$ and constant and operation symbols op_j $(1 \leq j \leq m)$. □

We transform arbitrary signatures of the above form into signatures with unary operations only (this structure is comparable to what is called *graph structure signature* in [5]):

$SIG' = \langle s_1, \ldots, s_n, op'_1, \ldots, op'_m; proj_{1,1}, \ldots, proj_{1,a_1+1}, \ldots, proj_{m,1}, \ldots, proj_{m,a_m+1} \rangle$, such that op'_j is the sort-counterpart of the original SIG-operation op_j. When $op_j(x_1, \ldots, x_{a_j}) = r$, $proj_{j,i}$ projects op'_j on its i^{th} component and $proj_{j,a_j+1}$ projects op'_j on its last component, being the result of op_j (a_j is the arity of op_j and $1 \leq i \leq a_j$).

Example 2.2 (integer algebra) In order to specify an integer algebra with only the add-operation we start with the following signature:

$SIGINT = \langle \texttt{int}; + \rangle$,

where `int` is the set of integer values and $+ : \texttt{int} \times \texttt{int} \rightarrow \texttt{int}$.

In a $SIGINT$-algebra A the $+$-operation could then be partially specified as $+ = \{\langle (1,2), 3 \rangle, \langle (2,3), 5 \rangle\}$.

The transformed signature $SIGINT'$ then has the following structure:

$SIGINT' = \langle \texttt{int}, +'; \texttt{arg0}, \texttt{arg1}, \texttt{result} \rangle$,

where $+' : \texttt{int} \times \texttt{int} \times \texttt{int}$ is the sort representing the original $+$-operation and the operations `arg0`, `arg1`, and `result` (all of type $+' \rightarrow \texttt{int}$) are projections that correspond to the $+'$-sort.

In the $SIGINT'$-algebra A' the $+'$-sort would contain the tuples $\langle 1,2,3 \rangle$ and $\langle 2,3,5 \rangle$ and the projections would look as follows:

$$\texttt{arg0}(\langle 1,2,3 \rangle) = 1 \quad \texttt{arg1}(\langle 1,2,3 \rangle) = 2 \quad \texttt{result}(\langle 1,2,3 \rangle) = 3$$

$$\texttt{arg0}(\langle 2,3,5 \rangle) = 2 \quad \texttt{arg1}(\langle 2,3,5 \rangle) = 3 \quad \texttt{result}(\langle 2,3,5 \rangle) = 5$$

□

Attributed graphs generally consist of two parts: a graph-part and a data-part.

Definition 2.3 (attributed graph) Consider a data signature $DSIG = \langle S, OP \rangle$ with attribute value sorts S and a graph $G = \langle V, E \rangle$. An *attributed graph* $AG = \langle G, D \rangle$ consists of a graph G and a $DSIG$-algebra D such that $\uplus_{s \in S} D_s \subseteq G_V$. □

When applying the described transformation to the data signature $DSIG$ from Definition 2.3, there exists a straightforward way of visually modeling the transformation of (the data-part of) attributed graphs in which the constants and operations are part of the *algebra graph*. The algebra operations to be applied are represented by nodes, labeled with the operation-symbols, being connected to the operands on which they will be applied and the resulting value.

The algebra graph can be looked upon as being a *bipartite graph* in which the nodes representing the instances of the algebra operations (with arity > 0) form one set and the nodes representing the constant data values form the other disjoint set. Moreover, the edges of the algebra graph all have the same direction, namely from the set of algebra operations to the set of constant data values. Fig. 2.1 shows part of the algebra graph of the $INTSIG'$-algebra A' from Example 2.2 as a bipartite graph. The right subset contains the

instances of the algebra operations; the left subset contains the constant data values. This bipartitioning property of the algebra graph will later on play an important role when discussing the finiteness of attributed graphs in our approach.

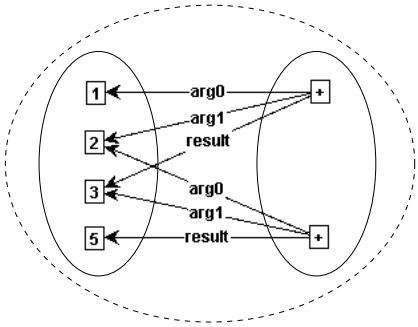

Figure 2.1. Bipartitioning of the algebra graph.

3 Transformation Specification

We have just explained how we model attributed graphs theoretically. In this section we will explain how we model and transform attributed graphs in a visual way by means of an example, focusing on how to change attribute values. The example, inspired by [5], is a graphical representation of method signatures in which a method is identified by its name and its ordered parameters.

3.1 Attributed Graphs

In Groove the attribute values are each represented by a single node and the names of the attributes are represented by the labels on the edges connecting them to the graph-part. An example method signature can then look as shown in Fig. 3.1.

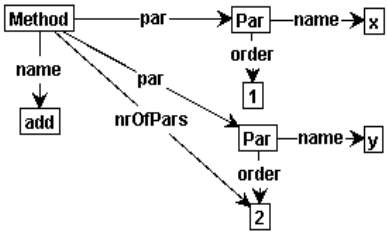

Figure 3.1. Graphical representation of the method signature `add(x,y)`.

3.2 Changing Attribute Values

Specifying the transformation of attributed graphs basically consists of two parts: specifying (1) graph-structure changes and (2) attribute value changes.

The first part is performed by *graph rewriting*, while the second part involves *term graph rewriting* [8]. Here we focus on the second part. Fig. 3.2 shows a transformation rule, using the single pushout approach [3], which adds a parameter to a method signature.

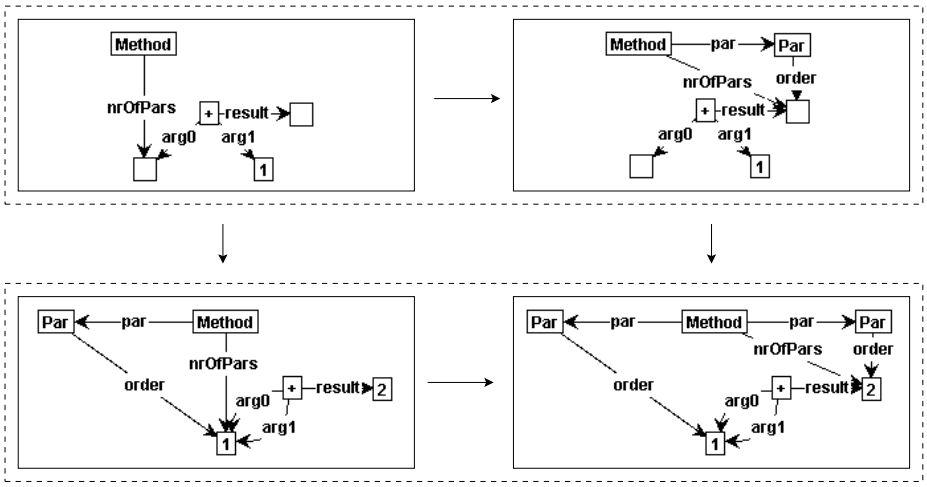

Figure 3.2. Rule application for adding a parameter to a method signature.

In this rule we specify how to add a new parameter to the method signature. [2] This involves the calculation of the new value of the nrOfPars-attribute and the creation of a new parameter which gets this new value as its order-attribute. The part of this rule that specifies the attribute value change in this rule consists of four nodes and three edges connecting them. One node represents the operation, two nodes represent the operands on which the operation must be applied and one node represents the result. Note that two nodes in the transformation rule (upper row in the figure) representing constant data values are left unlabeled. The value of the unlabeled operand can be determined after matching the rule's left-hand-side in the source graph. The result of applying the algebra operation on the operands is then determined by the algebra graph.

Note that we assume that the algebra elements (operations and constants) are *always* present. Of course, this is practically not possible because this would imply infinitely large graphs. In a tool implementation this could be resolved by only including those attribute-values of the algebra graph that are directly reachable from the graph-part. Combining the facts that the graph-part only refers to nodes from the algebra graph representing constant data values and that those nodes do not have outgoing edges (remember the bipartitioning property of the algebra graph discussed in Sect. 2) then implies the inclusion of a finite subgraph of the algebra graph in any attributed graph. Another point of attention is the fact that constant data values are represented

[2] In the given rule specification we have left out the parts involving the creation of the name-attribute because of implementation issues.

by *unique* nodes. This becomes clear from the rule application part of Fig. 3.2 where the node representing the integer-value 1 is both the first and second operand of the +-operation. The uniqueness of algebra operations is determined by their operands, i.e. algebra operations having distinct operands are represented by distinct nodes labeled with the operation symbol being connected to the nodes representing the operands and the corresponding unique result [8].

4 Advantages

We have shown how we transform arbitrary signatures in a signature structure with unary operations only and how we use the latter to specify the transformation of attributed graphs. Here we discuss a number of advantages of this approach, some of which are mainly related to tool implementation issues.

4.1 Variables

Normally, specifying the transformation of attributed graph requires the introduction of a set of variables when changing data values. In that case the transformation process involves activities such as assigning the actual value of the attributes to those variables, calculating the new value by applying the algebraic operation and assigning that new value to the original attribute. In our approach we do not introduce such a set of variables. We, instead, re-use ordinary rule nodes to stand for data values. When these nodes are matched in the source-graph, we can easily obtain the data value it stands for. This reduces the efforts needed for implementing attribution support in our tool.

4.2 Graphical Representation

The way we specify the transformation of attributed graphs graphically is closely related to the underlying theory. The part of the transformation rule shown in Fig. 3.2 that specifies the attribute value change, could also be viewed as being a hyperedge, having the nodes representing the constant data values as its endpoints and the operation symbol as its label. The list of endpoints then is an element of the sort corresponding to that operation. The labels of each tentacle represent the corresponding projection functions. Since our tool does not (yet) support hypergraphs, the algebra operation to be applied is represented by a distinct node having one outgoing edge for each tentacle of the corresponding hyperedge.

4.3 Changing Semantics

A third advantage of our approach is the separation of the use of algebra operations in transformation rules and their semantics. Since the semantics of the algebra operations are enclosed in the algebra graph, operation semantics

can be changed by changing the algebra graph, or better stated, by changing the algebra from which the algebra graph is derived. Fortunately, this has no effect on the transformation rules themselves, because they do not refer to the algebraic semantics of the used operations: the nodes representing the result of applying algebra operations are left unlabeled. Actually, other approaches may be using the same idea, but to our best knowledge this has never been stated explicitly.

5 Conclusion

A number of approaches to transform attributed graphs have been developed [7,6,5]. They all distinguish between the graph-part and the data-part of attributed graphs, but describe different ways of connecting these two parts together. In this report we focussed on how to specify attribute-value changes. In the literature, two main different approaches appear for this: relabeling attribute-nodes (see e.g. [7,9]) and reconnecting graph-nodes to the new attribute-nodes (see e.g. [6]). Our work is based on the second approach and differs from [6] in the way of specifying the actual attribute value change, since we store the semantics of the algebra operations in, what we call, the algebra graph by means of hyperedge-like structures. This way of modeling operation application is comparable to what is called a graph structure signature in [5]. This graphical structure binds every operation to the corresponding projection functions. In contrast to [5], our approach neither allows edge attribution nor typing.

Further work on this subject consists firstly of specifying the exact relation between our way of modeling attributed graphs and the other approaches (functor specification) and secondly of finishing the implementation of the tool concerning attribution support.

Acknowledgements

We would like to thank the anonymous referees for their detailed comments and constructive suggestions.

References

[1] Berthold, M. R., I. Fischer and M. Koch, *Attributed Graph Transformation with Partial Attribution*, in: H. Ehrig and G. Taentzer, editors, *Proceedings Joint APPLIGRAPH/GETGRATS Workshop on Graph Transformation Systems* (2000), pp. 171–178.

[2] Ehrig, H., G. Engels, F. Parisi-Presicce and G. Rozenberg, editors, "Proceedings of the 2^{nd} International Conference on Graph Transformation," Lecture Notes in Computer Science **3256**, Springer, 2004.

[3] Ehrig, H., R. Heckel, M. Korff, M. Löwe, L. Ribeiro, A. Wagner and A. Corradini, *Algebraic Approaches to Graph Transformation, Part II: Single Pushout Approach and Comparison with Double Pushout Approach*, in: G. Rozenberg, editor, *Handbook of Graph Grammars and Computing by Graph Transformation. Vol. 1: Foundations*, World Scientific, 1997 pp. 247–312.

[4] Ehrig, H. and B. Mahr, "Fundamentals of Algebraic Specification 1: Equations and Initial Semantics," Monographs on Theoretical Computer Science **6**, Springer-Verlag, 1985.

[5] Ehrig, H., U. Prange and G. Taentzer, *Fundamentel Theory for Typed Attributed Graph Transformation*, in: Ehrig et al. [2] pp. 161–177.

[6] Heckel, R., J. M. Küster and G. Taentzer, *Confluence of Typed Attributed Graph Transformation Systems*, in: A. Corradini, H. Ehrig, H. J. Kreowski and G. Rozenberg, editors, *Proceedings of the* 1^{st} *International Conference on Graph Transformations*, Lecture Notes in Computer Science **2505**, Springer, 2002 pp. 161–176.

[7] Löwe, M., M. Korff and A. Wagner, *An Algebraic Framework for the Transformation of Attributed Graphs*, in: *Term Graph Rewriting: Theory and Practice*, John Wiley and Sons Ltd., 1993 pp. 185–199.

[8] Plump, D., *Term Graph Rewriting*, in: H. Ehrig, G. Engels, H. J. Kreowski and G. Rozenberg, editors, *Handbook of Graph Grammars and Computing by Graph Transformation. Vol. 2: Applications, Languages and Tools*, World Scientific, 1999 pp. 3–61.

[9] Plump, D. and S. Steinert, *Towards Graph Programs for Graph Algorithms*, in: Ehrig et al. [2], pp. 128–143.

[10] Rensink, A., *The GROOVE Simulator: A Tool for State Space Generation*, in: J. L. Pfalz, M. Nagl and B. Böhlen, editors, *Applications of Graph Transformations with Industrial Relevance (AGTIVE)*, Lecture Notes in Computer Science **3062** (2004), pp. 479–485.

[11] Taentzer, G., *AGG: The Attributed Graph Grammar System* (2005), http://tfs.cs.tu-berlin.de/agg/.