Sydney Area Programming Languages INterest Group
The correspondence between Attribute Grammars and Term Rewriting
(Towards strategy free rewriting with Attribute Grammars)

Shirley Goldrei, Macquarie University

14th May, 2007

The research literature in the area of generating compilers and other language based tools divides the high level description of language semantics into two main approaches. The first of these is Attribute Grammars, introduced by Knuth in 1969 [3]. Ordered Attribute Grammars [2] are a class of Attribute Grammars that permit the computation of node visit sequences at the time the attribute evaluator is generated. Higher-order attribute grammars [9] are an extension to attribute grammars that allow subtrees to be computed dynamically and "grafted" onto (or "beside") node of existing substrees. Evaluation of attributes on Higher-Order Attribute Grammars are defined in the literature in terms of Ordered Attribute Grammars and their visit sequences.

The second main approach is term rewriting systems which have similarly been studied over many years. Most notably the work of Visser et. al. [4, 8, 7, 5, 1, 6] show the advantages of allowing the user to define rewrite rules separately from the strategies used to apply them to a tree. This approach is implemented in a system known as Stratego [5].

It is widely acknowledged that each of the two general approaches has strengths and weaknesses. In particular Attribute Grammars are most powerful and easy to use for describing analysis over a tree, usually the Abstract Syntax Tree. Where they are weak is in the description of the transformations of the tree from the source or input language to the target or output language. Conversely term rewriting systems are ideal for describing transformations on trees, but are very awkward to use for most analysis tasks.

This work aims to provide a formal description as well as a practical implementation of a system which combines Higher Order Attribute Grammars with term rewriting. As much as possible we would aim to leverage the dependency analysis of Attributes Grammars to free the user from having to explicitly provide strategies for applying rewrite rules. We also aim to leverage the static analysis of Ordered Attribute Grammars to improve the efficiency in both time and space requirements for transformations.

This talk will present some preliminary results showing the informal correspondence of simple rewrite rules and their equivalent translations in a Higher Order Attribute Grammar.

In strategic untyped rewriting systems, such as Stratego, the formalism is broken down into a number of basic operations and combinators for constructing complex operations from the simpler ones. The basic operations are: term matching, term binding, term construction, term replacement, one-step application of strategies to descendants. The combinators are: sequential composition, left choice, non-deterministic choice, if-then-else. Complex operations may also be defined recursively using a *recursive closure* operator. We consider each of the basic operations in turn and provide an informal semantic mapping to Attribute Grammars. We map combinators into attribute dependencies as will also be discussed.

In mapping rewriting systems to Attribute Grammars we map operations on terms to attributes and attribute equations within context-free production rules.

In a rewriting system, such as Stratego, terms are of the form $T(t_1, ...t_n)$ where T is the root of the subtree and $t_1..t_n$ are the immediate descendant terms. In Attribute Grammars, productions are of the form $X_0 \rightarrow X_1...X_n$ where X_0 is a non-terminal in the grammar and X_i for $1 \leq i \leq n$ are either terminal or non-terminal symbols.

In this discussion we will refer to a term being of the same *type* as a production if the non-terminal on the left hand side of the production, X_0, has the same name as the root of the term, T, and each of the symbols on the right hand side of the production, $X_1..X_n$, have the same name same as the root of each of the terms t_1 through t_n.

As an example we describe here the mapping of term matching in a rewriting system to an equivalent Attribute Grammar specification. Term matching involves a notion of success and failure. We statically determine which of the productions in the grammar describes a term of the same type as the term being matched. We model each match operation with a boolean attribute of the node on the left hand side of the production rule. Within such a "matched production" we model each variable mentioned within the matched term by a higher-order attribute with the same name as the variable. The value of this attribute is defined as the subtree rooted at the descendent (i.e. right hand symbol) that is found in the same positions as the variable is in the term ie. t_p where $1 \leq p \leq n$.

The presentation will cover other such mappings and will also discuss some open questions and some further desirable outcomes of the work.

References

[1] M. Bravenboer, A. van Dam, K. Olmos, and E. Visser. Program transformation with scoped dynamic rewrite rules. Technical Report UU-CS-2005-005, Institute of Information and Computing Sciences, Utrecht University, 2005.

[2] Uwe Kastens. Ordered attributed grammars. *Acta Informatica*, 13(3):229–256, 1980.

[3] Donald E. Knuth. Semantics of context-free languages. *Theory of Computing Systems*, 2(2):127–145, June 1968.

[4] Bas Luttik and Eelco Visser. Specification of rewriting strategies. In M. P. A. Sellink, editor, *2nd International Workshop on the Theory and Practice of Algebraic Specifications (ASF+SDF'97)*, Electronic Workshops in Computing, Berlin, November 1997. Springer-Verlag.

[5] Eelco Visser. Program transformation with stratego/xt. Technical Report UU-CS-2004-011, Institute of Information and Computing Science Utrecht University, 2004.

[6] Eelco Visser. A survey of strategies in rule-based program transformation systems. *Journal of Symbolic Computation*, 40(1):831–873, 2005. Special issue on Reduction Strategies in Rewriting and Programming.

[7] Eelco Visser and Zine-el-Abidine Benaissa. A core language for rewriting. In C. Kirchner and H. Kirchner, editors, *Second International Workshop on Rewriting Logic and its Applications (WRLA'98)*, volume 15 of *Electronic Notes in Theoretical Computer Science*, Pont-à-Mousson, France, September 1998. Elsevier Science Publishers.

[8] Eelco Visser, Zine-el-Abidine Benaissa, and Andrew Tolmach. Building program optimizers with rewriting strategies. In *Proceedings of the third ACM SIGPLAN International Conference on Functional Programming (ICFP'98)*, pages 13–26. ACM Press, September 1998.

[9] H. H. Vogt, S. D. Swierstra, and M. F. Kuiper. Higher order attribute grammars. In *PLDI '89: Proceedings of the ACM SIGPLAN 1989 Conference on Programming language design and implementation*, pages 131–145, New York, NY, USA, 1989. ACM Press.