Acta Informatica 31, 601–627 (1994)

Modularity and reusability in attribute grammars

U. Kastens[1] **and W. M. Waite**[2]

[1] Universität-GH Paderborn, Mathematik-Informatik (FB 17), D-33098 Paderborn, Germany
[2] Department of Electrical and Computer Engineering, University of Colorado, Boulder, CO 80309-0425, USA

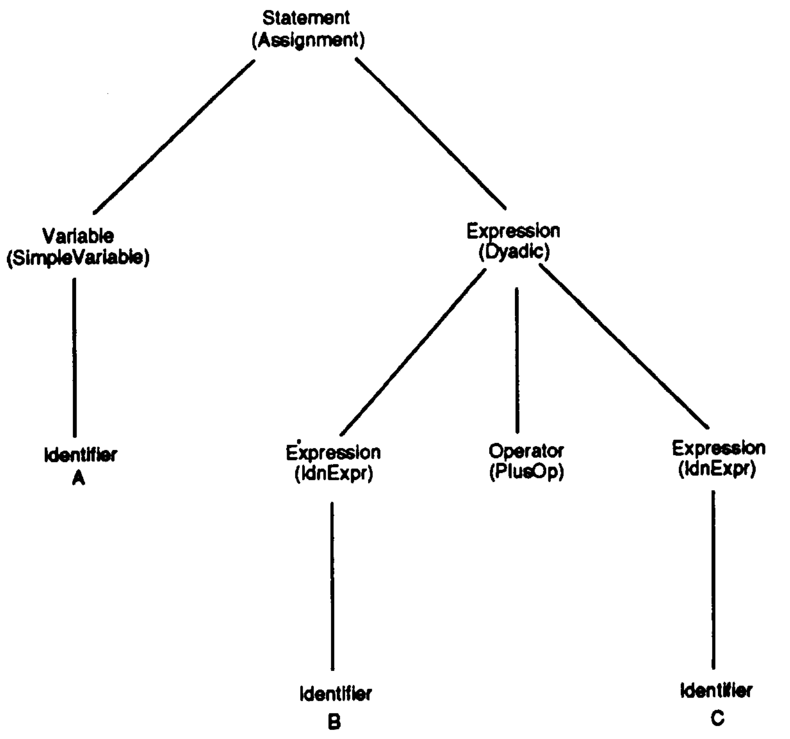
Received June 16, 1992 / January 14, 1994

Abstract. An attribute grammar is a declarative specification of dependence among computations carried out at the nodes of a tree. Attribute grammars have proven remarkably difficult to decompose into logical fragments. As a result, they have not yet been accepted as a viable specification technique. By combining the ideas of remote attribute access and inheritance, we have been able to define "attribution modules" that can be reused in a variety of applications. As an example, we show how to define reusable modules for name analysis that embody different scope rules.

1. Introduction

A fairly standard decomposition of the compiler construction task has evolved over the past twenty years, and most compilers have very similar structures. Many of the subproblems defined by the standard decomposition can be described as computations over trees, in which information is attached to individual tree nodes and used to control various decisions [27]. For example, consider the subproblem of determining the meaning of an operator that appears in an expression. The Pascal statement "$A := B + C$;" might be represented internally by the tree fragment shown in Fig. 1a. Here each node is classified as a $Statement$, $Variable$, $Expression$, $Operator$ or $Identifier$. Because there are different ways to construct nodes of each of these classes, the specific rule used is given in parentheses below the node class. (The class can be deduced from the rule, so only an indication of the rule is physically stored in the tree.)

According to the definition of Pascal, the meaning of the the dyadic expression's operator might be integer addition, real addition or set union. In order to determine the meaning, the compiler might *decorate* the tree by attaching additional information to the nodes as shown in Fig. 1b. The decoration `AddReal` would then be used in the decision to emit (say) a floating-point add instruction as the translation of the dyadic expression node.



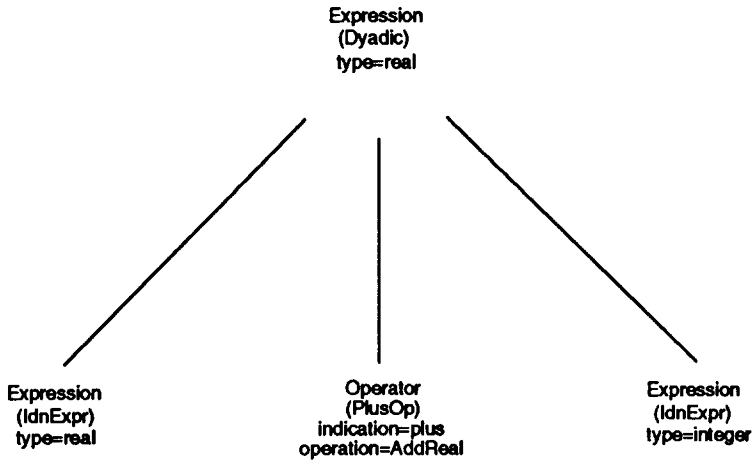


Fig. 1. a,b. A Tree Decoration Problem. **a** A Pascal tree fragment. **b** The dyadic expression decorated with type information