Generic Directed Acyclic Graphs (Generic DAGs)

Tze Meng Low
Supervisor: Prof. Gordon Novak Jr.
University of Texas at Austin

May 16, 2003

1 Abstract

Generic algorithms are algorithms using abstract data types. Multiple generic algorithms can be linked together in a directed acyclic graph (generic DAG) so that more complex generic algorithms can be formed. Views describe relationships between different data types. By compilation, specialized generic algorithms can be created from the generic DAGs. Views allow the specialized algorithm to manipulate actual data types used by the current applications instead of the abstract data types. This specialization process allows the generic DAG algorithms to work on different object descriptions of the same abstract data type, thus achieving greater software reuse.

Graphical user interfaces (GUIs) allow views to be specified quickly and easily. GUIs also enable independent generic algorithms to be linked together in a directed acyclic graph (DAG) to form more complex generic algorithms. Views can be created to represent an abstract data type as another abstract data type. This allows generic algorithms using different abstract data types to be connected together.

2 Introduction

Every piece of software is a huge investment in human cost, since programmers produce only a few lines of code per day. Even with the possibility of software reuse, such reuse is inhibited due to the different ways in which equivalent data could be represented and the time needed to adapt existing software components so as to fit the needs of the current application.

Generic algorithms and views can reduce the human cost by increasing the possibility of software reuse. Generic algorithms work on an abstract data type; views describe the way in which the actual application data types correspond to

the abstract data types. Using both generic algorithms and views, specialized code can be produced regardless of the actual application data types.

Apart from the human cost associated with reusing existing software components, another part of the human cost incurred by software engineering is the time spent in organizing different components in order to create the desired application behavior. Difficulties in visualizing the relationship between components and having to convert from one type to another between components are some of the causes for the time spent in organizing components.

This paper describes a method by which complex generic algorithms can be built, via a graphical interface, from existing generic components through the use of directed acyclic graphs(DAGs). DAGs can be represented visually and allow programmers to visualize the relationship between components easily. In addition, specialization of the generic components allows the complex generic algorithms to efficiently manipulate the actual application data.

The relationship between the abstract data type of one generic component and the abstract data type of another generic component can be captured in a view, similar to the views between abstract and actual data types. The view between abstract data types can then be used to generate functions for converting between two abstract data types. Therefore, the human cost involving converting between types is reduced.

Section 3 describes generic algorithms and the advantages of using generic algorithms. Section 4 describes views, how views can be created and how views allow generic algorithms to be specialized into efficient code. Section 5 describes how complex generic algorithms can be built from simple generic components via a graphical user interface and describes the algorithm used to generate a program from a directed acyclic graph. Section 6 presents related works, and Section 7 presents possible areas for further research. The final section summarizes this paper.

3 Generic Algorithms

Generic algorithms are algorithms that have been written based on abstract data representations instead of actual data representation. [4] On compilation, these generic algorithms are specialized based on object descriptions provided as inputs to create specialized codes based on actual data types.

An advantage of using generic algorithms is that a single generic algorithm can be specialized for use by different object representations. This allows greater code reuse. For example, two common representations of graphs (shown in Figure 1) are a matrix of weights and a list of edge-weight pairs. These two representations are conceptually similar *(both represent graphs)* but have different actual data types *(one is a 2-dimensional array while the other is a linked-list)*.

A second advantage of using generic algorithms is that the specialized codes execute faster at runtime. Current object-centered programming system are driven by interpreting messages to function calls at runtime. These runtime interpretations add a significant amount of time to the execution time. In

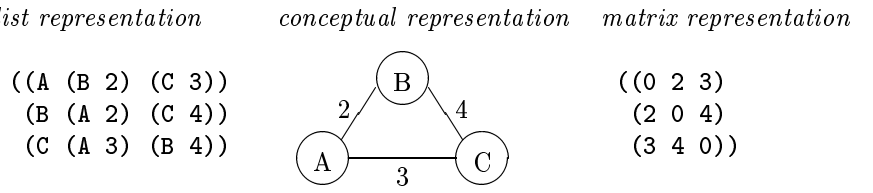

Figure 1: A graph and its 2 common representations.

addition, as interpretation of the messages occurs at runtime, error checking, if done, must also occur at runtime. This increases the execution time of a method call. However, specializing from generic algorithms allows the compiler to partially evaluate messages to known objects at compile time. This reduces the amount of overhead incurred from message interpretation and error checking at runtime.

4 Specialization through Views

Views [5][7] describe how actual data types correspond to the abstract data types of an object. A view encapsulates an actual data type and presents an interface that consists of the basic variables in the abstract data type.

Using the relationships between the actual data types and the abstract data types described by views, the GLISP compiler generates specialized code that uses the actual data types instead of the abstract data types. At compile time, the relationships in a view are compiled into the specialized code, allowing the specialized algorithm to manipulate the actual data types directly. Furthermore, optimization through partial evaluation at compile time improves the performance of the specialized code. Therefore, the use of views in the process of creating specialized code has little impact on the execution time of the specialized code itself.

A graphical user interface allows views to be generated easily. The program, `mkv`, written by Dr. Gordon Novak, takes a target type and a source type as inputs and allows users to create a view associating the source type with the basic variables of the target type.

5 Directed Acyclic Graphs

As given in Rosen [8], a popular undergraduate text on discrete mathematics, a directed graph, G, is a set of vertices V and a set of edges E that are ordered pairs of elements of V. A graph is said to be acyclic when there is no sequence of connected edges such that a vertex is visited twice. Two properties of an acyclic graph are that there exists at least one vertex (root vertex) such that it has no edges leading to it and there exists at least one vertex (leaf vertex) that does not have an edge leading out of it. Using mathematical notations a DAG