The Refinement Relation of Graph-Based Generic Programs
Extended Abstract

Karl Lieberherr and Boaz Patt-Shamir

1 College of Computer Science
Northeastern University
Boston, MA 02115, USA
lieber@ccs.neu.edu
http://www.ccs.neu.edu/home/lieber

2 Dept. of Electrical Engineering–Systems
Tel Aviv University
Tel Aviv 69978, Israel
boaz@eng.tau.ac.il

Abstract. This paper studies a particular variant of Generic Programming, called Adaptive Programming (AP). We explain the approach taken by Adaptive Programming to attain the goals set for Generic Programming. Within the formalism of AP, we explore the important problem of refinement: given two generic programs, does one express a subset of the programs expressed by the other? We show that two natural definitions of refinement coincide, but the corresponding decision problem is computationally intractable (co-NP-complete). We proceed to define a more restricted notion of refinement, which arises frequently in the practice of AP, and give an efficient algorithm for deciding it.

1 Introduction

What is Generic Programming (GP) [MS94]? According to the organizers of this Dagstuhl workshop, GP has the following important characteristics:

− Expressing algorithms with minimal assumptions about data abstractions, and vice versa, thus making them as interoperable as possible.
− Lifting of a concrete algorithm to as a general level as possible without losing efficiency, i.e., the most abstract form such that when specialized back to the concrete case the result is just as efficient as the original algorithm.

GP is about parametric polymorphism and we think that non-traditional kinds of parametric polymorphism lead to particularly useful forms of Generic Programming. By non-traditional kinds of parametric polymorphism we mean that parameterization is over larger entities than classes. In this paper we focus on parameterization with entire class graphs and we outline how Adaptive Programming is a form of Generic Programming which attempts to satisfy the
two characteristics mentioned above. We show the role of traversal strategies in Adaptive Programming by an analogy to Generic Programming and present new results about traversal strategies. We focus on the concept of graph refinement which is important when traversals are specialized. We show that the obvious definition of refinement leads to a co-NP-complete decision problem and we propose a refinement definition, called strong refinement, which is computationally tractable and useful for practical applications. The results are summarized in Table 1.

Table 1. Graph relationships for software evolution. \( N \) is a mapping of nodes of \( G_2 \) to nodes of \( G_1 \). \( G_1 \leq_N G_2 \) if and only if \( G_1 \leq_N G_2 \), \( G_1 \sqsubseteq_N G_2 \) implies \( G_1 \leq_N G_2 \).

<table>
<thead>
<tr>
<th>Relationship</th>
<th>Complexity</th>
<th>Symbol</th>
</tr>
</thead>
<tbody>
<tr>
<td>path-set-refinement</td>
<td>co-NP-complete</td>
<td>( G_1 \leq_N G_2 )</td>
</tr>
<tr>
<td>expansion</td>
<td>co-NP-complete</td>
<td>( G_1 \leq_N G_2 )</td>
</tr>
<tr>
<td>strong refinement</td>
<td>polynomial</td>
<td>( G_1 \sqsubseteq_N G_2 )</td>
</tr>
</tbody>
</table>

A generic program \( P \) defines a family of programs \( P(G) \), where \( G \) ranges over a set of permissible actual parameters. In this paper we let \( G \) range over directed graphs restricted by the program \( P \). Those graphs are abstractions of the data structures on which the program operates. Given two generic programs \( P_1 \) and \( P_2 \), an important question is whether the programs defined by \( P_1 \) are a subset of the programs defined by \( P_2 \). We say that \( P_1 \) is a refinement of \( P_2 \). For example, the generic program \( P_1 \) “Find all B-objects contained in X-objects contained in an A-object” defines a subset of the programs determined by the generic program \( P_2 \) “Find all B-objects contained in an A-object.” \( P_1 \) and \( P_2 \) are generic programs since they are parameterized by a class graph (e.g., a UML class diagram). Furthermore, the computations done by \( P_1 \) are a refinement of the computations done by \( P_2 \).

Formalizing the notion of refinement between generic programs leads to graph theoretic problems which have several applications. Refinement can be used to define “subroutines” in adaptive programs as well as to define common evolution relationships between class graphs.

1.1 Adaptive Programming (AP)

Adaptive Programming \cite{Lie92, Lie96} is programming with traversal strategies. The programs use graphs which are referred to by traversal strategies. A traversal strategy defines traversals of graphs without referring to the details of the traversed graphs. AP is a special case of Aspect-Oriented Programming \cite{Kic96, KLM+97}.

AP adds flexibility and simultaneously simplifies designs and programs. We make a connection between GP (as practiced in the STL community) and AP (see Table 2). In GP, algorithms are parameterized by iterators so that they can be used with several different data structures. In AP, algorithms are parame-