Available online at www.sciencedirect.com
SCIENCE DIRECT®
Information and Software Technology 47 (2005) 805–817
INFORMATION AND SOFTWARE TECHNOLOGY
www.elsevier.com/locate/infsof

Computing dynamic slices of concurrent object-oriented programs

Durga Prasad Mohapatra, Rajib Mall, Rajeev Kumar*

Department of Computer Science and Engineering, Indian Institute of Technology Kharagpur, Kharagpur, WB 721 302, India

Received 8 November 2004; revised 9 February 2005; accepted 14 February 2005
Available online 1 June 2005

Abstract

We propose a novel dynamic program slicing technique for concurrent object-oriented programs. Our technique uses a *Concurrent System Dependence Graph* (CSDG) as the intermediate program representation. We mark and unmark the edges in the CSDG appropriately as and when the dependencies arise and cease during run-time. We mark an edge when its associated dependence exists and unmark an edge when the dependence ceases to exist. Our approach eliminates the use of trace files. Another advantage of our approach is that when a request for a slice is made, it is already available. This appreciably reduces the response time of slicing commands.

Keywords: Program slicing; Static slicing; Dynamic slicing; Program dependence graph; Debugging; Concurrent object-oriented programs; Threads

1. Introduction

Program slicing is a well known decomposition technique for extracting the statements of a program related to a particular computation [1]. A slice of a program P can be constructed with respect to a *slicing criterion*. A slicing criterion is a tuple $<s,V>$ where s is a program point of interest and V is a subset of the program's variables used or defined at s. The slice can be obtained by deleting the statements from the program P which have no effect on any of the variables in V as execution reaches statement s. There are two types of slices depending on the input to the program: *static slice* and *dynamic slice*. A *static slice* of a program P with respect to a slicing criterion $<s,V>$ is the set of all the statements of program P that might affect the slicing criterion *for every possible inputs* to the program [1]. A static slice may contain some statements that might not be executed during an actual run of a program. In contrast, a *dynamic slice* contains only those statements of program P that actually affect the slicing criterion for a *particular set of inputs* to the program [2]. So, dynamic slices are usually smaller than static slices. Program slicing has been found to be useful in various software engineering activities like debugging, program understanding, testing and maintenance, measuring cohesion, etc. [3–9]. Comprehensive surveys on the existing slicing techniques and their applications can be found in [10–12].

Object-oriented concepts such as encapsulation, inheritance, message passing and polymorphism, etc. make the traditional slicing techniques, inadequate for use with object-oriented programs. Researchers have extended existing slicing techniques to handle some of these features. Larson and Harrold [13] were the first to consider object-orientation aspects in their work. They introduced the *class dependence graph* which can represent a class hierarchy, data members, inheritance and polymorphism. They have constructed the system dependence graph (SDG) using the *class dependence graphs* to satisfactorily represent object-oriented programs. After the SDG is constructed, the two phase algorithm of Horwitz et al. [14] is used with minor modifications for computing slices. Larson and Harrold [13] have reported only a static slicing technique for sequential object-oriented programs, and did not address the *concurrency* and *dynamic slicing* aspects. Zhao [15], Song and Huynh [16], Wang et al. [17] and Xu and Chen [18] have addressed the issues of dynamic slicing of object-oriented programs, but they have not addressed the *concurrency* issues in object-oriented programs.

The size and complexity of object-oriented programs are increasing rapidly. This poses a formidable difficulty to

* Corresponding author. Tel.: +91 3222 283 464; fax: +91 3222 278 985.
E-mail addresses: durga@cse.iitkgp.ernet.in (D.P. Mohapatra), rajib@cse.iitkgp.ernet.in (R. Mall), rkumar@cse.iitkgp.ernet.in (R. Kumar).

0950-5849/$ - see front matter © 2005 Elsevier B.V. All rights reserved.
doi:10.1016/j.infsof.2005.02.002

a programmer to either understand the working of a program or debug an existing error. Efficient slicing techniques would help in debugging and understanding these programs. Many of the real life object-oriented programs are concurrent which run on different machines connected to a network. It is usually accepted that understanding and debugging of concurrent object-oriented programs are much harder compared to those of sequential programs. The non-deterministic nature of concurrent programs, the lack of global states, unsynchronized interactions among processes, multiple threads of control and a dynamically varying number of processes are some reasons for this difficulty. An increasing amount of resources are being spent in debugging, testing and maintaining these products. Slicing techniques promise to come in handy at this point. However, research results in slicing object-oriented programs have scarcely been reported in the literature [13,19–24]. Further, to our knowledge no research results addressing the problem of dynamic slicing of concurrent object-oriented programs have been published. It is the objective of this paper to present our work concerning development of a dynamic slicing algorithm for concurrent object-oriented programs.

A major goal of any dynamic slicing technique is efficiency since the results may be used during interactive applications such as program debugging. Efficiency is especially an important concern for slicing concurrent object-oriented programs, since their sizes are typically very large. Large programs result in very large intermediate graphs and can result in response times of several hundreds of seconds.

With this motivation, in this paper we propose a new dynamic slicing algorithm for computing slices of concurrent Java programs. Only the concurrency issues are addressed here, traditional object-oriented features are not discussed in this paper. Handling standard object-oriented features can be found in [13]. So, these representations of object-oriented features can easily be incorporated into our algorithm. Our algorithm uses a modified form of the program dependence graph (PDG) as the intermediate representation. We have named this graph *concurrent system dependence graph* (CSDG). We first statically construct the CSDG [25]. Then, we apply our algorithm to the CSDG to compute dynamic slices of concurrent object-oriented programs. Our algorithm is based on marking and unmarking the edges of the CSDG appropriately as and when dependencies arise and cease at run-time. So, we have named our algorithm *marking-based dynamic slicing* (MBDS) algorithm for concurrent object-oriented programs. Such an approach is more time- and space-efficient and also allows to completely to eliminate the use of a trace file at run-time to record the execution history. In dynamic slicing, it is desirable to eliminate the slow file *I/O* operations that occur while accessing a trace file as these make the response times unacceptably large in interactive sessions. Another advantage of our approach is that when a request for a slice is made, it is already available. This appreciably reduces the response time of slicing commands.

The rest of the paper is organized as follows. In Section 2, we present some basic concepts and definitions that will be used in our algorithm. In Section 3, we discuss the intermediate program representations: *concurrent control flow graph* (CCFG) and *concurrent system dependence graph* (CSDG). In Section 4, we present our *marking-based dynamic slicing* (MBDS) algorithm for concurrent object-oriented programs. In Section 5, we present a brief description of a slicing tool we have developed to implement our proposed dynamic slicing algorithm for concurrent object-oriented programs. In Section 6, we compare our work with related work. Section 7 concludes the paper.

2. Basic concepts and definitions

Before presenting our dynamic slicing algorithm, we introduce a few definitions that would be used in our algorithm. In the following definitions and throughout the rest of the paper, we use the terms statement, node and vertex interchangeably. We explain the definitions by using Figs. 1 and 3. Fig. 1 represents an example concurrent Java program and Fig. 3 represents the CSDG of the example program of Fig. 1.

Definition 1. *Precise Dynamic Slice*. A dynamic slice is said to be *precise* if it includes only those statements that actually affect the value of a variable at a program point for the given execution. However, this is only an informal definition of a precise slice since the question as to whether a statement should actually be included in a precise slice is undecidable [1].

Definition 2. *def*(*obj*), *defset*(*obj*). Let *obj* be an object in a class in the program *P*. A node *x* is said to be a *def*(*obj*) node if *x* represents a definition (assignment) statement that defines the object *obj*. The set defset(obj) denotes the set of all *def*(*obj*) nodes.

In Fig. 3, nodes 2, 9 and 17 are the Def(a2) nodes and Defset(a2) = {2,9,17}.

Definition 3. *use*(*obj*) *node*. Let *obj* be an object in a class in the program *P*. A node *x* is said to be a *use*(*obj*) node iff it uses the object *obj*.

In Fig. 3, the node 4 is a use(a3) node and nodes 2, 6 and 12 are use(a2) nodes.

Definition 4. *recentdef*(*obj*). For each object *obj*, *recentdef*(*obj*) represents the node (the label number of the statement) corresponding to the most recent definition of the object *obj*.

Definition 5. *Concurrent Control Flow Graph* (*CCFG*). A *concurrent control flow graph* (CCFG) *G* of a program *P* is a directed graph (*N*, *E*, *Start*, *Stop*), where each node $n \in N$