Abstract

This thesis presents our work concerning dynamic slicing of object-oriented programs. We first develop a suitable intermediate representation for simple object-oriented programs with inheritance and polymorphism. This intermediate representation is known as *extended system dependence graph* (ESDG). Then, we present an efficient dynamic slicing algorithm, called *edge marking dynamic slicing* (EMDS) algorithm for simple object-oriented programs using ESDG. Our EMDS algorithm is based on marking and unmarking the edges of the ESDG as and when dependencies arise and cease during run-time. Our algorithm marks an edge of the ESDG when its associated dependence exists and unmarks an edge when the dependence ceases to exist. Then, we present another dynamic slicing algorithm, called *node marking dynamic slicing* (NMDS) algorithm for simple object-oriented programs using ESDG as the intermediate representation. Our NMDS algorithm is based on marking and unmarking the nodes of the ESDG appropriately during run-time. We have shown that each of our proposed algorithm is more efficient than the related algorithms. Also, we have shown that the *NMDS* algorithm is efficient than the *EMDS* algorithm. The space complexity of both our algorithms is $O(n^3)$, where n is the number of statements in the program. The run-time complexity of both our algorithms is $O(n^2S)$, where S is the length of execution of the program.

Next, we extend our intermediate representation (ESDG) to be able to represent *concurrent object-oriented programs.* We have named this intermediate representation *concurrent system dependence graph* (CSDG). Our CSDG correctly represents the *concurrency* aspects such as *synchronization dependencies* and *communication dependencies* in object-oriented programs. Then, we extend our edge marking dynamic slicing (EMDS) algorithm to handle these *concurrency* issues in object-oriented programs. We have named our algorithm *marking based dynamic slicing* (MBDS) algorithm for object-oriented programs. Our MBDS algorithm successfully handles the *concurrency* aspects such as *synchronization dependencies* and *communication dependencies* while computing dynamic slices of object-oriented programs. The space complexity of the MBDS algorithm is $O(n^3)$, where n is the number of statements in the program. The run-time complexity of the MBDS algorithm is $O(n^2S)$, where S is the length of execution of the *concurrent* program.

Next, we extend our intermediate representation (CSDG) to represent object-

oriented programs distributed on several nodes connected to a network. We add some additional nodes and edges to the CSDG to represent *communication issues* in distributed object-oriented programs. We have named this modified intermediate representation *distributed program dependence graph* (DPDG). We construct the DPDG separately for each of the component program distributed on different nodes. Then, we modify our MBDS algorithm to take care of *communication issues* in distributed object-oriented programs. We have named our modified algorithm *distributed dynamic slicing* (DDS) algorithm for object-oriented programs. We apply the DDS algorithm on the DPDG to compute dynamic slices of distributed object-oriented programs. To achieve fast response time, our DDS algorithm can run in a fully distributed manner on several nodes connected through a network, rather than running on a centralized node. We use local slicers at each node in a network. A local slicer is responsible for slicing the part of the program executions occurring on the local machine. The space complexity of the DDS algorithm is $O(N^3)$, where N is the *total* number of statements in the distributed program. The run-time complexity of the DDS algorithm is $O(N^2S)$, where S is the total length of execution of the *distributed* program.

We show that our dynamic slicing algorithms compute *correct* dynamic slices, and are more efficient than the existing algorithms. Our algorithms neither use trace files nor create any additional nodes during run-time. This saves the expensive file handling and I/O steps. Another advantage of our algorithms is that when a request for a slice is made, it is already available. Our intermediate representations at any time during program execution are annotated with the uptodate dynamic slices. Thus computation of dynamic slice involves only examining all the marked dependencies existing at a node for the instance of program in execution. As a result, our algorithms extract dynamic slices in constant time. Besides the theoretical analysis and studies that we have conducted, we have also implemented our dynamic slicing algorithms to experimentally verify their *correctness*. Our experimentation involving a large number of test programs convinces us about the *correctness* of our proposed algorithms.