A Brief Survey Of Program Slicing

Baowen Xu Ju Qian Xiaofang Zhang Zhongqiang Wu Lin Chen

Department of Computer Science and Engineering, Southeast University, Nanjing 210096, China
Jiangsu Institute of Software Quality, Nanjing 210096, China

Abstract

Program slicing is a technique to extract program parts with respect to some special computation. Since Weiser first proposed the notion of slicing in 1979, hundreds of papers have been presented in this area. Tens of variants of slicing have been studied, as well as algorithms to compute them. Different notions of slicing have different properties and different applications. These notions vary from Weiser's syntax-preserving static slicing to amorphous slicing which is not syntax-preserving, and the algorithms can be based on dataflow equations, information-flow relations or dependence graphs.

Slicing was first developed to facilitate debugging, but it is then found helpful in many aspects of the software development life cycle, including program debugging, software testing, software measurement, program comprehension, software maintenance, program parallelization and so on.

Over the last two decades, several surveys on program slicing have been presented. However, most of them only reviewed parts of researches on program slicing or have now been out of date. People who are interested in program slicing need more information about the up to date researches. Our survey fills this gap. In this paper, we briefly review most of existing slicing techniques including static slicing, dynamic slicing and the latest slicing techniques. We also discuss the contribution of each work and compare the major difference between them. Researches on slicing are classified by the research hot spots such that people can be kept informed of the overall program slicing researches.

Keywords: program slicing, program analysis, dependence analysis, pointer analysis, debugging

1 Introduction

Program slicing, which is a reverse engineering technique, has been widely studied since it was first proposed in 1979. Although several survey [Binkley 1996; De Lucia 2001; Harman 1996a, 1998a, 2001a; Kamkar 1995b; Russell 2001; Tip 1995a] have been presented, many of them only reviewed parts of researches on program slicing or have been out of date now. People who are interested in program slicing need more information about the up to date researches. We presented a brief survey that reviews most of existing slicing techniques including static slicing, dynamic slicing and the latest slicing techniques. In this survey, we discuss the contribution of each work and compare the major difference between them. Researches on slicing are classified by the research hot spots such that people can be kept informed of the overall program slicing researches.

Contents

1.1 The Concept of Program Slicing

Program slicing, which was first introduced by Weiser in 1979 [Weiser 1979], is a decomposition technique that extracts from program statements relevant to a particular computation. A program slice consists of the parts of a program that potentially affect the values computed at some point of interest referred to as a slic-

* Correspondence to: Baowen Xu, Department. of Computer Sci. and Eng., Southeast University, Nanjing 210096, China. Email: bwxu@seu.edu.cn

ing criterion. Typically, a slicing criterion consists of a pair <p, V>, where p is a program point and V is a subset of program variables. The parts of a program that have a direct or indirect effect on the values computed at a slicing criterion C are called the program slice with respect to criterion C. The task of computing program slices is called program slicing.

Program slicing defined by Weiser is in fact a kind of executable backward static slicing. "Executable" means that the slice is not only a closure of statements, but also can be compiled and run. Non-executable slices are often smaller and thus more helpful in program comprehension. A backward slice consists all statements that the computation at the slicing criteria may depend on, while a forward slice includes all statements depending on the slicing criterion. Forward slicing is a kind of ripple effect analysis. "Static" means that only statically available information is used for computing slices, i.e. all possible executions of the program are taken into account.

Since 1979, several variants of slicing, which are not static, have been proposed.

Korel and Laski introduced the concept of dynamic slicing [Korel 1988b]. Different from static slices, a dynamic slice is constructed with respect to only one execution of the program. It does not include the statements that have no relevance with the slicing criteria on some particular input. Due to the run-time handling of arrays and pointer variables, dynamic slicing algorithms can be easier compared to static ones and result in more precise slices. As dynamic slice is execution specific and relatively smaller, it is helpful in debugging.

Agrawal, Horgan and et al. [Agrawal 1993b] extended dynamic slicing to relevant slicing. A relevant slice with respect to a variable contains not only the statements that have an influence on the variable but also those executed statements that did not affect the output, but could have affected it had they evaluated differently. Relevant slicing can facilitate incremental regression testing.

Venkatesh then introduced quasi-static slicing [Venkatesh 1991], which is a slicing method between static slicing and dynamic slicing. A quasi-static slice is constructed with respect to an initial prefix of the input sequence to the program. It is used to analyze the behavior of the program when some input variables are fixed while others vary. In the case all variables are unconstrained, the quasi-static slice coincides with a static slice, while when the values of all input variables are fixed, the slice is a dynamic slice. Field then proposed constrained slicing [Field 1995]. It is very similar to quasi-static slice. The major difference is that the input state of a constrained slice is characterized by a predicate in the programming language not an initial prefix of the input sequence. Later, Canfora et al. proposed conditioned slicing [Canfora 1998], which is a more general form of quasi-static slicing and constrained slicing with the input states characterized by a universally quantified, first order predicate logic formula. Actually, conditioned slicing is a framework of statement deleting based methods, i.e., the conditioned slicing criterion can be specified to obtain any form of slice. Conditioned slicing allows a better decomposition of the program giving human readers the possibility to analyze code fragments with respect to different perspectives.

Hall introduced simultaneous dynamic slicing [Hall 1995] to compute slices with respect to a set of program executions. The final slice is construct using dynamic slices corresponding to each execution in the program execution set. Simultaneous dynamic slicing can be used to locate functionality in code.

Different from Venkatesh, Field, Canfora et al. and Hall's approach, which slice programs only for a set of program executions, there is another kind of slicing called hybrid slicing which incorporate both static and dynamic information. In hybrid slicing static information is used to facilitate dynamic slicing or dynamic information is used to help static slicing.

Traditional slicing methods are all based on statement deletion. Harman introduced amorphous slicing which removes the limitation to statement deletion as the only means of simplification [Harman 1997c]. The syntactic requirement is dropped while the semantic requirement is retained. The slice preserves the selected behavior of interest from the original program. As a broader range of transformation rules, including statement deletion, can be applied, this leads to slices that are often considerably smaller than their syntax-preserving counterparts. Amorphous slicing is particularly useful in program comprehension.

Besides the original program slicing which to find program parts affect or be affected by computation at some point. Two variations on the slicing theme, which are closely related to slicing, have been presented. They are dicing [Lyle 1986] and chopping [Jackson 1994b]. A program dice is defined as the set difference between the static slices of an incorrect variable and that of a correct variable, i.e., the set of statements that affect the computation of incorrect variable while do not affect the computation of the correct one. It is a fault location technique for further reducing the number of statements that need to be examined when debugging. Chopping solves the problem of how one variable affects the other. Given two variable sets, source and sink, it tries to identify the statements that cause the definitions of source to affect the uses of sink.

1.2 Program Slicing Methods

There are three major kinds of approaches in program slicing. Weiser's original slicing approach is a kind of approach based on iteration of dataflow equations. In this approach, slices are computed in an iterative process, by computing consecutive sets of relevant variables for each node in the CFG. The algorithm first computes directly relevant statements for each node in the CFG, and then indirectly relevant statements are gradually added to the slice. The process stops when no more relevant statements is found.

Information-flow relations for programs [Bergeretti 1985] can also be used to compute slices. In this kind of approach, several types of relations are defined and computed in a syntax-directed, bottom-up manner. With these information-flow relations, slices can be easily obtained by relational calculus.

The most popular kind of slicing approach is slicing via graph reachability. In these approaches, slicing can be divided into two steps. In the first step, dependence graphs of the program are constructed, and then the algorithm produce slices by doing graph reachablity analysis on them. A dependence graph is a directed graph using vertexes to represent program statements and edges to represent dependences. The graph reachablity analysis can be done by traversing edges on the dependence graph from a node representing the slicing criteria. A dependence graph can represent not only dependences but also other relations such as process communications and so on. Different slices can be obtained by constructing different dependence graphs.

In program slicing, great efforts have been made to handle different features in programming languages.

Jump statement is such a feature of programming language that can cause the program to be unstructured and thus difficult to slice. Two kinds of approaches have been proposed to solve this problem. One tries to restructure program flow-graphs for the purpose of restructuring a program without jump statements and then slicing methods for jump free programs can be reused. The other directly determines which predicates and relevant jump statements to be included in the slice using control flow and dependence information.

Often access to arrays and pointers can only be determined dynamically, this also cause program difficult to slice precisely. In the presence of arrays, a simple approach is to treat each array as a whole. More precise approaches that distinguish the elements of array need complex dependence analysis. In the presence of pointers, a safe static slicing algorithm must safely handle alias. Besides collecting the alias information, new notions of data dependence should be defined to cover the potentially data dependence in the case of potential aliases.

To precisely handle procedural calls, the calling context problem must be solved, i.e., we must identify the realizable paths of the program. Several approaches solve this problem by traversing the extended dependence graph in stepwise way, while some others use call stacks to avoid entering a procedure and return to another.

To handle programming language features such as concurrent and object-oriented, the most important thing is building a proper representation to represent new kinds of dependence. When slicing concurrent programs, we need to correctly represent the interdependence between tasks. While when slicing object-oriented or aspect-oriented program, the main problem is to represent polymorphism, dynamic binding and the class inheritance hierarchy appropriately. During the last ten years many kind of graphs have been proposed, these representations have their benefit in different aspects.

1.3 Applications of Program Slicing

As a slice is an independent program guaranteed to faithfully represent the original program within the domain of the specified subset of behavior, it is helpful in program debugging, software maintenance, software testing, software measurement, program comprehension, program parallelization, and so on. These applications of program slicing are caused by two properties of themselves. When modifying a program, we only need to comprehend a section of the program rather than the whole program. This property leads to the applications of debugging, maintenance, testing, and so on. When comprehending the whole program, we can first comprehend several sections, then comprehend the relation between the sections, rather than comprehend the whole program directly. This property leads to the applications of measurement, program comprehension, program parallelization, and so on.

1.4 Paper Organization

The remainder of this paper is organized as follows. In section 2, we review the basic slicing approaches. In section 3, methods for slicing object-oriented programs are discussed. In section 4, we review the methods of handling composite datatypes and pointers when slicing. In section 5, methods for slicing concurrent programs are discussed. Section 6 is a brief survey on non-static slicing methods. Section 7 contains an overview of semantic related slicing and new slicing researches. In section 8, we review the applications of program slicing.

2 Basic Slicing Approaches

2.1 Basic Slicing Algorithms

The original concept of program slicing [Weiser 1984] was first proposed as the solution to a dataflow problem specified using the program's control-flow graph (CFG). Many researchers have investigated program slicing based on Weiser's definition, primarily looking at the problem of generating the smallest possible slice for a given criterion. In general this problem cannot be solved, but various approaches result in good approximations. Some techniques are based on data flow equations [Korel 1988b; Leung 1987; Weiser 1984] while others use graph representations of the program [Agrawal 1990, 1994; Ball 1993a; Binkley 1993a; Choi 1994; Horwitz 1990; Jackson 1994a].

It is Karl Ottenstein and Linda Ottenstein [Ottenstein 1984] that first used the program dependence graph (PDG) [Ferrante 1987; Kuck 1981] to compute program slices. They proposed a slicing algorithm based on graph reachability in the program dependence graph. However, they only considered the intraprocedural case as a PDG only describes the dependencies in a single subprogram. To model the dependencies among subprograms, Horwitz, Reps and Binkley [Horwitz 1990] introduced system dependence graphs (SDG), which is an extension of PDG. They also proposed a stepwise traversing algorithm on the SDG to find interprocedural program slices. Value dependence graph (VDG) [Ernst 1994; Weise 1994] is another kind of graph that can be used to perform slicing. A VDG is a data flow-like representation that evolved from an attempt to eliminate the control flow graph as the basis of the analysis phase and using demand dependences instead. It is composed of nodes which represent computation and arcs which carry value between computations. This representation is independent of the names of values, the locations of the values, and when the values' are computed.

An executable slice of a program with respect to program point p and variable x consists of a subset of the program that computes the same sequence of values for x at p [Weiser 1984]. In [Lyle 1984], Lyle presented a modified version of Weiser's algorithm for computing slices. Apart from some minor changes in terminology, this algorithm is essentially the same as that in [Weiser 1984]. Ottenstein and Ottenstein [Ottenstein 1984] defined a program slice to be simply the set of statements that influence the value of the variable; they suggested the Program Dependence Graph (PDG) [Ferrante 1987] as the natural basis for computing this notion of a non-executable slice. For many applications, such as optimization and program understanding, only this weaker notion of a slice is needed. Horwitz [Horwitz 1988a, 1988b, 1989a, 1990b, 1991] proposed that a slice of a program with respect to program point p and variable x consists of a set of statements of the program that might affect the value of x at p. Note that an executable slice by Ottenstein and Ottenstein is also a slice by Horwitz.

Bergeretti and Carre [Bergeretti 1985] proposed another approach that defines slices in terms of information-flow relations derived from a program in a syntax-directed fashion. This relation is a formalization of the results of Denning and Denning [Denning 1977] in secure information flow.

Danicic et al. [Danicic 1995] introduced a parallel algorithm for computing backward, static slices. This is accomplished by converting the CFG into a network of concurrent processes. Each process sends and receives messages that name the relevant sets of variables.

Chung, Lee and et al [Chung 2001; Lee 2001] proposed some methods using specification to facilitate program slicing. With the specification more precise slices can be obtained by removing statements that are not relevant to the specification for the slice. Their technique is based on the pre/post conditions. This specification-based slicing is helpful in program reconstruction, reusable component extracting and so on.

Besides slicing of program written in imperative languages, several approaches have been proposed in slicing other kinds of programming languages. [Gyimóthy 1998; Schoenig 1996; Szilagyi 2002; Vasconcelos 1994, 1995, 1998a, 1998b, 1999;] address the problem of slicing logic programs. Most of these work concentrate on slicing of Prolog programs. Slicing of logic programs is more sophisticated since it must account for this diversity of meanings for the same program. Furthermore, in purely declarative logic languages there are no explicit references to the flow of execution and iteration can only be achieved via recursion. Often adaptation of ideas used in slicing imperative languages and methods to address the aspect of logic programs are used together to perform such slicing. [Ahn 1999; Gandle 1993] addressed the problem of slicing programs written in functional languages. Both of these approaches make use of abstract interpretation (or techniques like abstract interpretation). Clarke et al. [Clarke 1999] presented an approach to adapt the notion of program slicing to hardware description languages. In their approach the hardware description languages is first transformed to other languages preserving the execution semantics and then slicing methods for traditional languages can be of use. [Chang 1994; Oda 1993; Woodward 1998; Wu 2004a; Zhao 1998c] addressed the problem of slicing specifications. The idea of slicing specifications was introduced by Oda and Araki [Oda 1993]. They defined a static slicing technique for Z. Chang et al. followed Oda and Araki's work and introduced the concept of dynamic specification slicing. Recently F. Wu and T. Yi [Wu 2004a] put forward the work of slicing Z. They introduced a new dependence in Z and proposed a slicing algorithm based on dependence graphs. Specification slicing gives designers the benefits of program slicing at the specification level. It allows designers to track dependencies and perform testing and debugging activities.

Since more and more kinds of slicing methods have been proposed, evaluations of these slicing methods have also been made. These evaluations [Bent 2000; Binkley 2003, 2004a; Hoffner 1995a, 1995b; Kusumoto 2002; Lyle 1984, 1986] mainly concentrated on the properties of slicing such as efficiency, size of resulting slices, applications, the interplay between slicing and other source code analysis and so on. Empirical results show that although slicing can assist the applications such as debugging by simplifying the program under consideration, the slices constructed by static slicing tend to be rather large. This is particularly true for well-constructed programs, which are typically highly cohesive. This high level of cohesion results in programs where the computation of the value of each variable is highly dependent upon the values of many other variables.

2.2 Slicing Programs with Arbitrary Control Flow

In intraprocedural program slicing, the critical problem is to determine which predicates and relevant jump statements to be included in the slice when the program contains jump statements.

The original slicing algorithm proposed by Weiser was able to determine which predicates to be included in the slice even when the program contains jump statements. It did not, however, make any attempt to determine the relevant jump statements themselves to be included in the slice. Thus, Weiser's algorithm for static slicing may yields incorrect slices in the presence of unstructured control flow.

Lyle [Lyle 1984] proposed an extremely conservative algorithm to determine which jump statements to include in a slice. His algorithm produces slices including every goto statement that has a non-empty set of active variables associated with it.

Gallagher [Gallagher 1990, 1991b] proposed a modification of the Weiser's algorithm. In the algorithm, a goto statement is included in the slice if it jumps to a label of an included statement. The algorithm does not produce correct slices in all cases. Jiang, Zhou, and Robson [Jiang 1991] have also proposed a set of rules to determine which jump statements to include in a slice. Unfortunately Agrawal [Agrawal 1994] pointed out that their rules also fail to identify all relevant jump statements.

Ball and Horwitz [Ball 1993a, 1993b] and Choi and Ferrante [Choi 1994] discovered independently that conventional PDG-based slicing algorithms produce incorrect results in the presence of unstructured control flow, slices may compute values at the criterion that differ from what the original program does. These problems are due to the fact that the algorithms do not determine correctly when unconditional jumps such as break, goto, and continue statements are required in a slice. They proposed two similar algorithms to determine the relevant jump statements to include in a slice. Both of them require that the jumps be represented as pseudo-predicates and the control dependence graph of a program be constructed from an augmented flow-graph of the program and two formal proofs have been proposed to show their algorithms compute correct slices. Choi and Ferrante also proposed another algorithm to construct an executable slice in the presence of jump statements when a "slice" is not constrained to be a subprogram of the original program. The algorithm constructs new jump statements to add to the slice to ensure that other statements in it are executed in the correct order.

Hiralal Agrawal [Agrawal 1994] proposed an algorithm has the same precision as that of the above two algorithms. It is, however, appealing in that it leaves the flow-graph and the program dependence graph of the program intact and uses a separate graph to store the additional required information. More importantly, it lends itself to substantial simplification, when the program under consideration is a structured program. Also, the simplified algorithm directly leads to a conservative approximation algorithm that permits on-the-fly detection of the relevant jump statements while applying the conventional slicing algorithm.

Harman and Danicic [Harman 1998c] defined an extension to Agrawal's algorithm that produces smaller slices by using a refined criterion for adding jump statements (from the original program) to the slice computed using Ottensteins' algorithm for building and slicing the PDG [Ottenstein 1984].

Sumit Kumar and Susan Horwitz [Kumar 2001] extended previous work on program slicing by providing a new definition of "correct" slices, by introducing a representation for C-style switch statements, and by defining a new way to compute control dependences and to slice a program dependence graph so as to compute more precise slices of programs that include jumps and switches. As they claimed, the time complexity of their algorithm is linear in the size of the computed slice.

Sinha, Harrold, and Rothermel [Sinha 1999] discussed interprocedural slicing in the presence of arbitrary interprocedural control flow, e.g., statements (like halt, setjmp-longjmp) that prevent procedures from returning to their call sites. Their approach is based on an extension of the System Dependence Graph proposed in [Horwitz 1990].

There also have been several approaches to restructuring program flowgraphs. Peterson et al. [Peterson 1973] presented a proof that every flowgraph can be transformed into an equivalent well-formed flowchart. They presented a graph algorithm to do such a transformation using a technique of node-splitting and they proved the transformation was correct. William and Ossher [Williams 1977, 1978] also used node-splitting, but they presented the problem as recognizing five basic unstructured sub-graphs, and showed how to replace these sub-graphs with equivalent structured forms. Ashcroft and Manna [Ashcroft 1971] tackled the problems of restructuring by presenting two algorithms for converting program schemas into while schemas. Rather than using node-splitting they used extra logical variables to achieve these transformations.

All of the previous methods were intended to restructure all flow charts. However, there have also been approaches suggested that are used to restructure programs in order to expose the natural structure of the program, leaving some gotos unstructured. The first such method was given by Baker [Baker 1977] as a method for restructuring Fortran programs in order to make them more understandable. Cifuentes [Cifuentes 1993] presented an algorithm for restructuring in the context of decompilation. This work is similar in spirit to Baker's problem in that she only structures the parts of the program that correspond naturally to structured control constructs. Allen et al [Allen 1983] presented the IF conversion method that converts control dependences into data dependences by eliminating goto statements and introducing logical variables to control the execution of the statements. The method presented by Ammarguellat, which she calls control-flow normalization, is the closest work in terms of the goals of restructuring. While Ammarguellat [Ammarguellat 1992] restructures a lisp-like intermediate representation and she requires that all loops have a single exit. Ana M. Erosa and that all loops have a single exit. Ana M. Erosa and Laurie J. Hendren [Erosa 1994] proposed a straightforward algorithm to structure C programs by eliminating all goto statements. The method proceeds by eliminating each goto by first applying a sequence of goto-movement transformations followed by the appropriate goto-elimination transformation.

2.3 Static Interprocedural Slicing Methods

Weiser introduced an interprocedural program slicing algorithm in [Weiser 1984]. The algorithm is a simple extension of his intraprocedural one with the idea that interprocedural slices can be obtained by extending the original slicing criteria set with relevant slicing criteria in other procedures. This approach can be easily implemented as it directly makes use of the intraprocedural slicing algorithm. However, it does not address the calling-context problem, i.e. the transitive-closure operation fails to account for the calling context of a called procedure. Thus, the algorithm cannot produce precise slices.

Hwang, Du and Chou [Hwang 1988a] proposed an iterative solution for interprocedural static slicing based on replacing recursive calls by instances of the procedure body. This approach does not suffer from the calling context problem because expansion of recursive calls does not lead to considering infeasible execution paths. However, Reps [Reps 1994c, 1996a] showed that for a certain family of recursive programs, this algorithm takes time that exponential in the length of the program.

Horwitz, Reps and Binkley [Horwitz 1990] proposed an interprocedural slicing algorithm that operates on a program representation called the System Dependence Graph (SDG). The notion of SDG is an extension of Procedural Dependence Graph proposed in [Ottenstein 1984]. Their algorithm involves two steps: first, complete the SDG with summary edges, which represent transitive dependences due to procedure calls; second, slices are computed by doing a two phase traversing on the SDG. The slicing algorithm is efficient and can handle the calling context problem. What' more, the SDG can be widely used in other applications.

Several extensions of Horwitz-Reps-Binkley (HRB) algorithm have been presented. Lakhotia [Lakhotia 1992] adapted the idea of lattice theory to interprocedural slicing and presented a slicing algorithm based on the augmented SDG in which a tag is contained for each vertex of SDG. Different from HRB algorithm, this algorithm only need one traverse on the SDG. Binkley extended HRB algorithm to produce executable interprocedural program slices in [Binkley 1993a]. He also presented an interprocedural slicing algorithm in the presence of parameter aliases in [Binkley 1993b]. The algorithm is parameterized by a set of aliasing patterns for each procedure. Flow dependence edges are created on different "levels" to separately represent different possible flow dependences for different possible aliasing patterns. Slices can then be obtained by traversing the multiple-level flow dependence edges in the SDG.

Clarke et al. [Clarke 1999] extended HRB slicing algorithm to VHDL, using an approach based on capturing the operational semantics of VHDL in traditional constructs. Their algorithm first maps the VHDL constructs onto traditional programming language constructs and then slices using a language-independent approach.

Orso et al. [Orso 2001b] proposed a SDG-based incremental slicing technique, in which slices are computed based on types of data dependences. They classified the data dependences into different types. The scope of a slice can be increased in steps, by incorporating additional types of data dependences at each step. As the initial slice is small and each increment to the slice can be significantly smaller than the complete slice, the approach can be useful in software maintenance.

Reps [Reps 1994a] derived a demand version of interprocedural program slicing method by applying the magic-sets transformation that is developed form logic-programming and deduced database communities. Other demand program analysis technique can be found in [Horwitz 95; Reps 1994b].

There were some other approaches that concentrate on improving the efficiency of computing summary edges. Livadas et al. [Livadas 1993b, 1994a, 1999] presented several methods to compute transitive dependences between parameters in the presence of recursion. Their methods require neither the calculation of the GMOD and GREF sets nor the construction of a linkage grammar and the corresponding subordinate characteristic graphs of the linkage grammar's nonterminals. Reps [Reps 1994c] proposed another approach to improve the computation of summary edges. This approach can significantly improve the time complexity of HRB algorithm. A demand version of the algorithm, which incrementally determines the summary edges of a SDG, is presented in [Reps 1994d]. Forgacsy and Gyimóthy [Forgacsy 1997a] introduced the idea of divide and conquer to the computation of summary edges. Dividing the call graph and analyzing procedures in each group in topological order can significantly improve the efficiency of interprocedural slicing method.

To the problem of SDG constructing, Forgacsy and Gyimóthy [Forgacsy 1997a] presented a method to reduce the SDG. Livadas and Croll [Livadas 1994b] extended the SDG and proposed a method to construct SDG directly from parser trees. Their algorithm is conceptually much simpler, but it cannot handle recursion. Kiss [Kiss 2003] presented an approach to construct SDG from the binary executable programs and proposed an algorithm to slice on them. Sinha, Harrold and Rothermel [Sinha 1999] extended the SDG to represent interprocedural control dependences. Their extension is based on Augmented Control Flow Graph (ACFG), a CFG augmented with edges to represent interprocedural control dependences. Hisley et al. [Hisley 2002] extended the SDG to threaded System Dependence Graph (tSDG) in order to represent non-sequential programs.

There is another kind of interprocedural slicing methods that uses a call stack to solve the calling-context problem. Harrold and Ci [Harrold 1998b] proposed an interprocedural program slicing algorithm (HC98) that is also based on the ACFG. The algorithm is different from the one presented in [Sinha 1999]. In this approach, call stacks are also used to handle the calling-context problem and caches are used to reuse the slice information. Compared to the SDG-based approach presented in [Sinha 1999], this one need less work in slicing preparation but has more complexity when slicing. Liang and Harrold [Liang 1999a] found that the HC98 algorithm could be inefficient in the presence of pointers and the precision need to improve for programs with recursion. They proposed an improved algorithm to solve these problems by handling programs with pointer variables uses equivalence analysis [Liang 1999b] and handling programs with recursion by computing a minimal fixed point for the procedures involved in the recursion. Liang and Harrold [Liang 2000] also presented the light-weight context recovery, a technique that can efficiently determine whether a memory location is accessed by a procedure under a specific call-site, for programs using pointer variables. They applied the light-weight context recovery in HC98 algorithm and found that it can improve the slicing algorithm both in efficiency and precision.

Atkinson and Griswold [Atkinson 1996] proposed an empirical study of an interprocedural slicing method using call stacks. In their approach, the context-depth, which is the maximum call stack depth, is flexible to facilitate demand-driven analysis. The result showed that there is a trade-off in the context-depth between the precision and the efficiency. Agrawal and Guo evaluated the explicitly context-sensitive program slicing techniques in [Agrawal 2001]. They studied two questions, namely, what is the level of precision lost if context-sensitivity is not maintained and what are the additional costs for achieving context-sensitivity. Their results show that the context-sensitive technique, in spite of its worst case exponential complexity, can be very efficient in practice. On the average, it cost twice more execution time than the context-insensitive technique, but produce slices only half in the size of the one produced by context-insensitive technique. Similar approaches were presented in [Binkley 2003; Krinke 2002, 2004b].

Ouarbya et al. [Ouarbya 2002] proposed a denotational interprocedural slicing method for programs in the presence of side-effects. Using the denotational approach, slices can be defined in terms of the abstract syntax of the object language without the need of a control flow graph or similar intermediate structure. Their algorithm mainly concentrates on the WSL language. It is capable of correctly handling the interplay between function and procedure calls, side-effects, and short-circuit expression evaluation but still can not handle programs with local scope, multiple return statements and recursive functions.

Ernst [Ernst 1994] proposed an interprocedural slicing algorithm based on Value Dependence Graphs (VDG) incorporating the idea of HRB algorithm. This approach is expression-oriented. It can identify different sub-expressions in a statement and can identify the temporal variables implicitly constructed by complier and therefore can produce fine-gained slices.

Harman et al. [Harman 2002b] presented an interprocedural amorphous slicing algorithm and illustrated the way in which interprocedural amorphous slicing improves upon interprocedural syntax-preserving slicing.

Besides the above researches, there are also many researches concentrate on the problems that are relevant to interprocedural program slicing. Callahan et al. studied the problem of constructing the call graph [Antoniol 1999; Callahan 1990; Grove 2001; Hall 1992; Lakhotia 1993a; Milanova 2004; Tip 2000]. Harrold et al. [Ezick 2001; Harrold 1998a; Sinha 2001] presented several approaches to compute interprocedural control dependence. Reps [Reps 1996a] studied the complexity of interprocedural analysis. Lakhotia et al. [Lakhotia 1999] proposed several flow analysis models for graph-based slicing algorithms.

3 OO and AOP Slicing

When slicing object-oriented programs, how to represent the programs is an important problem. [Krishnaswamy 1994] designed an OPDG based on Dependency Graph. Concepts, including polymorphism, dynamic binding and the class inheritance hierarchy, can be represented.

[Larsen 1996] described the construction of system dependence graphs for object-oriented software on which efficient slicing algorithms can be applied. With these system dependence graphs, slices for individual classes, groups of interacting classes and complete programs can be computed via graph reachability. The system dependence graphs can be constructed incrementally. And slices can be computed for incomplete object-oriented programs such as classes or class libraries.

Unlike the statement slice, [Tip 1996] was concerned with eliminating unnecessary components from the declarative parts of C++ programs. Given a C++ class hierarchy and a program P that uses the hierarchy, the algorithm presented by the paper eliminates from the hierarchy those data members, member functions, classes, and inheritance relations that are unnecessary for ensuring that the semantics of P is maintained.

[Zhao 1996] concerned the problem of slicing concurrent object-oriented programs. To solve the problem, the system dependence net (SDN), which extends previous program dependence representations to represent concurrent object-oriented programs, was proposed. An SDN of a concurrent object-oriented program consists of a collection of procedure dependence nets each representing a main procedure, a free standing procedure, or a method in a class of the program, and some additional arcs to represent direct dependences between a call and the called procedure/method and transitive interprocedural data dependences. Once a concurrent object-oriented program is represented by its SDN, the slices of the program can be computed based on the SDN as a simple vertex reachability problem in the net.

[Chen 1997a] presented an Object-Oriented Dependency Graph (ODG) to represent the structure of OO programs. The ODG is defined based on a property multi-digraph that is extended from a directed graph by augmenting multiple edge types, vertex properties, and property relations. With the extension, the ODG can avoid the specious dependencies due to object encapsulation. Based on the ODG, a program slicing method for OO software was developed in the paper.

[Chen 1997b] defined two types of program slices, state and behavior slices, by taking the dependencies of OO features into consideration. A state slices for an object is a set of messages and control statements that might affect the state of the object, while a behavior slice is a set of attributes and methods defined in related classes that might affect the behavior of the object.

[Zhao 1998d] addressed the problem of dynamic slicing of object-oriented programs. To solve the problem, the dynamic object-oriented dependence graph (DODG) which is an arc-classified digraph to explicitly represent various dynamic dependences between statement instances for a particular execution of an object-oriented program, was presented. Based on the DODG, a two-phase algorithm for computing a dynamic slice of an object-oriented program was described.

[Liang 1998] presented an SDG for object-oriented software that is an extension of [Larsen 1996; Tonella 1997]. This SDG more fully represents the features of object-oriented programs, including objects that are present in languages such as C++ and Java. The paper also introduced the concept of object slicing, which identifies statements in methods of an object that might affect the slicing criterion. The main benefits of this SDG are that it distinguishes data members for different objects, represents objects that are used as parameters or data members in other objects, and represents the effects of polymorphism on parameters and parameter binding. Another benefit is that the SDG construction algorithm is more efficient than previous approaches, which may enable more efficient slicing on larger programs. A final benefit is that the technique for object slicing lets a user inspect statements in a slice, object by object.

Unlike [Larsen 1996; Zhao 1998d], [Ohata 2001] thought static slicing cannot compute practical (or precise) analysis results, and dynamic slicing requires too much computation time and memory space. So it adopted an intermediate slicing method between static slicing and dynamic slicing named Dependence-Cache (DC) slicing to object-oriented programs. DC slicing methods uses dynamic data dependence analysis and static control dependence analysis, which computes more precise analysis results than static slicing and needs less analysis costs than dynamic slicing.

[Chen 2001a] presented a method to represent OO Java Software that is different from [Larsen 1996; Liang 1998; Tonella 1997]. Methods can interact with each other by the interfaces that are dependences among parameters, and the dependences among inner data and statements are invisible outside. The program dependence graph is a set of PDGs of methods and classes. The PDG of a class consists of a set of PDGS of its methods. Each PDG is an independent graph, and does not connect to any other PDGs. This paper also introduced three concepts: object slicing, class slicing and partial slicing. Object slicing identifies data members and statements in methods of the class that might affect the slicing criterion. Partial slicing provides a way for the users to emphasize the parts in which they are interested.

[Xu 2002] proposed a method to dynamically slice object-oriented (OO) programs based on dependence analysis. It uses the object program dependence graph and other static information to reduce the information to be traced during program execution. It deals with OO features such as inheritance, polymorphism and dynamic bindings. Based on this model, the paper presented methods to slice methods, objects and classes. The slicing criterion is also modified to fit for debugging.

[Zhao 2002a] presented a dependence-based representation called aspect-oriented system dependence graph (ASDG), which extends previous dependence graphs, to represent aspect-oriented software. The ASDG of an aspect-oriented program consists of three parts: a system dependence graph for non-aspect code, a group of dependence graphs for aspect code, and some additional dependence arcs used to connect the system dependence graph to the dependence graphs for aspect code. The paper also showed how to compute a static slice of an aspect-oriented program based on the ASDG.

[Ishio 2003] evaluated the usefulness of AOP in the area of program analysis. At first, the application of AOP to collecting dynamic information from program execution and calculating program slice was examined. Then, a program slicing system using AspectJ was developed, and benefits, usability, cost effectiveness of the module of dynamic analysis based on AOP was also described.

[Zhao 2003a] extended previous dependence-based representations called system dependence graphs (SDGs) to represent aspect-oriented programs and presented an SDG construction algorithm. After the construction, the result is the complete SDG. The SDGs capture the additional structure present in many aspect-oriented features such as join points, advice, introduction, aspects, and aspect inheritance, and various types of interactions between aspects and classes. They also correctly reflect the semantics of aspect-oriented concepts such as advice precedence, introduction scope, and aspect weaving. SDGs therefore provide a solid foundation for the further analysis of aspect-oriented programs.

4 Slicing in the Presence of Composite Datatypes and Pointers

4.1 Slicing Arrays

When slicing, there are two approaches to handle arrays. A simple approach for arrays is to treat each array as a whole [Lyle 1984]. However, this approach leads to unnecessary large slices. To be more precise requires distinguishing the elements of array. And this needs dependence analysis.

In 1988, U. Banerjee presented the Extended GCD Test [Banerjee 1988]. It can be applied to analyze the general objects (multi-dimensional arrays and nested trapezoidal loops). The test is derived from number theory. The single equation $a_1x_1 + a_2x_2 + \ldots + a_nx_n = b$ has an integer solution iff $\gcd(a_i)$ divides b. This gives us an exact test for single-dimensional arrays ignoring bounds. And it can be extended to multi-dimensional arrays. If the equation has an integer solution, it can also get some constraints for next analysis.

Also in [Banerjee 1988], U. Banerjee proposed an approach to get the constraints from the Extended GCD Test. We called it Single Variable Per Constraint Test. If the solution to the generalized GCD test has at most 1 free variable then one can solve the problem. Else, we can only implement the single variable constraint. In this situation, it may get the result. At least, it can reduce the constraints. In fact, this test is a superset of the well known single loop, single dimension exact test [Banerjee 1979].

In 1991, Maydan, Hennessy and Lam [Maydan 1991] developed the Acyclic Test for cases where at least on constrain has more than on variable. The test creates a directed graph based on the constraints. If there is no cycle in the graph, it can get the result. Else, it can implement the constraints which are not in the cycle. This simplifies the system for the next stages.

Pratt [Pratt 1977] proposed an approach (Simple Loop Residue Test) to handle the graph which has at least a cycle. The approach works only when all constraints are of the form $t_i \leqslant t_j + c$. Shostak [Shostak 1981] extends the approach first to deal with inequalities of the form $a\, t_i \leqslant b\, t_j + c$ and then to handle cases with more than two variables. However, these extensions make the approach inexact.

Dantzig and Eaves [Dantzig 1973] proposed a Fourier-motzkin algorithm. It can solve the general non-integer linear programming case exactly. Theoretically it can be exponential. Experimentally, Triolet [Triolet 1985] has implemented this approach and seems to be satisfied with its efficiency. But Li, Yew and Zhu [Li 1990] don't think so.

Maydan, Hennessy and Lam [Maydan 1991] presented one approach to use a series of special case exact tests. It cascades the exact tests including Extended GCD Test, Single Variable Per Constraint Test, Acyclic Test, Simple Loop Residue Test and Fourier-motzkin. If the input is not of the appropriate form for a test, it will try the next. Because the approach can use the results of previous tests to help the following analyses, it is more precise and has acceptable efficiency.

4.2 Slicing Pointers

In the presence of pointers, situations may occur where two or more variables refer to the same memory location. This phenomenon is commonly called aliasing. In this phenomenon, we need pointer analysis to get the information of data dependence.

[Hind 2000, 2001a] are two survey of the pointer analysis. [Hind 2000] described an empirical comparison of the effectiveness of five pointer analysis algorithms on C programs. The effectiveness of the analyses is quantified in terms of compile-time precision and efficiency. And [Hind 2001a] presented issues related to pointer analysis and remaining open problems.

The target of pointer analysis is to get the approximate storage figure, i.e. the aliasing information. There are two methods to represent aliasing information. A pointer alias analysis [Landi 1992c] attempts to determine when two pointer expressions refer to the same storage location. A point-to analysis [Andersen 1994], or similarly, an analysis based on a "compact representation" [Burke 1994; Choi 1993; Hind 1999], attempts to determine what storage locations a pointer can point to. Tradeoffs between the two representations are discussed in [Marlowe 1993].

Precise pointer analysis is undecidable [Landi 1992a]. So the analysis has to do a trade-offs between cost and precision. There are several dimensions that affect the trade-offs. How a pointer analysis addresses each of these dimensions helps to categorize the analysis.

Are objects named by allocation site, or is a more sophisticated shape analysis performed? This is a rule to distinguish the algorithms. Because it is possible to construct unbounded data structures, we must necessarily represent them in some finite way. In other words, we should do some compression. Most work, e.g. [Horwitz 1989c; Jones 1981], does this compression by bounding acyclic path length in the modeled data structures; this is known as k-bounded approximation. They limit the length of paths to k by truncating long paths with summary nodes containing all paths occurring in the original. Besides k-bounded approximation, [Chase 1990] proposed a different way of summarizing linked data structures that allows a particularly efficient implementation for the sparse case, preserves information about unbounded data structures, and takes advantage of structure present in the original program.

Is control-flow information of a procedure used during the analysis? This is also a rule to distinguish the algorithms. If it uses the information, we call it flow-sensitivity [Choi 1993; Hind 1999]; else we call it flow-insensitivity [Andersen 1994; Shapiro 1997a; Steensgard 1996]. The aim of pointer analysis is to get the storage figure. Each statement may change the figure. So each statement may have different figures. Giving a figure for each statement or using one figure to represent several statements is a choice. If we give a figure for each statement, it really means flow-sensitivity. Otherwise, it means flow-insensitivity. Method in [Choi 1993] utilizes kill information and can provide improved precision over a flow-insensitivity analysis that does not utilize kill information during analysis. Method in [Steensgard 1996] is equality-based, which uses a union-find data structure. It computes one solution set for the entire program, and its time complexity is almost linear. [Andersen 1994] also presented an approach that computes one solution set for the entire program. But it performs an iterative implementation rather than the merging, so it is more precise.

Is calling context considered when analyzing a function? This is another rule to distinguish the algorithms. If it considers the calling context, we call it context-sensitivity [Emami 1994; Wilson 1995, 1997], else we call it context-insensitivity [Choi 1993]. [Emami 1994] presented a context-sensitive approach that generates a graph representing all invocation paths (in the absence of recursion). [Wilson 1995, 1997] presented an approach that summarizes the effect of a procedure call for an input subset and reuses this summary at other call sites that invoke the procedure with the same inputs, eliminating the cost of reanalyzing the procedure at such call sites. Method in [Choi 1993] merges the calling context and performs at most a single analysis of each procedure during a traversal over the PCG.

Obviously, the flow-sensitivity and context-sensitivity algorithms are more precise. But they are less efficient. So when we choose the pointer analysis, we must consider the client problem's needs [Hind 2000]. And we shall also analyze the real effect of different analyses. [Hind 2000, 2001b] suggested that for context-insensitive analyses, a flow-sensitive analysis does not offer much precision improvement over a subset-based flow-insensitive analysis. [Ruf 1995] pointed out that context-sensitivity did not improve precision for a common flow-sensitive analysis.

Slicing in the presence of aliasing requires a generalization of the notion of data dependence to take potential aliases into account. This can be done roughly as follows: a statement B is potentially data dependent on a statement A if: (1) A defines a variable x, (2) B uses a variable y, (3) x and y are potential aliases, and (4) there exists a path from A to B in the CFG where x is not necessarily defined. Such paths may contain definitions to potential aliases of x. This altered notion of data dependence can in principle be used in any static slicing algorithm. [Horwitz 1989c] proposed one approach about this. They defined the dependence in terms of potential definitions and uses of abstract memory locations. [Agrawal 1991] presented a method implementing a similar idea, and it also presented a method for dynamic slicing in the presence of pointers.

Besides the data dependence, in the presence of pointers, the reaching definitions also need to be changed, and the l-valued expressions have to be taken into account. Based on the definitions, we can implement some different methods.

[Jiang 1991] presented an algorithm for slicing C programs with pointers and arrays. [Lyle 1993] use a variation on symbolic execution to assign addresses to pointer variables and to build lists of possible addresses contained by each pointer at each statement. [Ernst 1995] reports its slicer. The slicer, which explicitly represents the store as an aggregate value, supports arbitrary pointer manipulations and is faster than more limited techniques. The multiple levels of indirection for both assignment and reference of pointers create difficult problems. [Lyle 1995] proposes a pointer state graph (PSS) to solve the problem. [Ross 1998] presents a simple reduction of the program dependence problem to the may-alias problem. The reduction has the property of always computing conservative program dependences when used with a conservative may-alias algorithm.

5 Slicing Concurrent Programs

Most of the methods for slicing concurrent programs we know are based on graph reachability. One of the earliest approaches to static slicing of threaded programs was presented by Cheng in [Cheng 1993]. He extended the notion of slicing for concurrent programs and presented a graph-theoretical approach to slicing concurrent programs. Some dependences, named selection dependence, synchronization dependence, and communication dependence are introduced. The selection dependence is a special kind of control dependence. The synchronization dependence is a mixture of control and data dependence. The communication dependence is an interprocess data dependence. These dependences are modeled by the Process Dependence Net (PDN), and slices are defined based on graph reachability. The slice is not precise because they didn't take into account that the interprocess data dependence is not transitive. A slicing algorithm for concurrent java programs based on MDG was proposed by Zhao [Zhao 1998a, Zhao 1999a]. He also did not consider the intransitive dependence.

An intransitive dependence analysis for concurrent programs was done by Chen to show why some of the previous slicing methods are not precise and a new technique is proposed to get a more precise result [Chen 2001c, 2002a].

A threaded control flow graph (tCFG) and a threaded program dependence graph (tPDG), which extend the well known CFG and PDG for threaded programs with interference, are introduced by Krinke [Krinke 1998b]. The interference dependence is added to the tPDG. Based on these graphs they computed more precise static slices than previous work. But the algorithm works only intraprocedural and the concurrent model does not include synchronization, so this method cannot be applied widely.

Nanda improved Krinke's approach to include loop-carried data dependence. Some optimizations are proposed to slice more efficiently [Nanda 2000]. The paper said that it could get near linear behavior for many practical concurrent programs.

In the latest work [Krinke 2003a], Krinke proposed a context-sensitive method to slice concurrent recursive programs accurately and this method does not require serialization or inlining.

Approaches for computing dynamic slices of concurrent programs also have been considered. Duesterwald, Gupta, and Soffa presented a parallel algorithm for computing dynamic slices of distributed programs which is based on the Distributed Dependence Graph (DDG) [Duesterwald 1992b]. M. Kamkar introduced a distributed dynamic dependence graph (DDDG) which represents control, data and communication dependences to compute accurate dynamic slices [Kamkar 1995a], and a distributed dynamic slicer for ANSI-C programming language has been implemented. Korel and Ferguson extended the dynamic slicing method to distributed programs with Ada-type rendezvous communication [Korel 1992].

Recently, Rilling, Hon F. Li and Goswami presented two novel predicate-based dynamic slicing algorithms for message passing programs [Rilling 2002; Li 2004]. They defined a slicing criterion which focuses on those parts of the program that influence the predicate. The dynamic predicate slices capture some global requirements or suspected error properties of a distributed program and compute all statements that are relevant and a proof of correctness of these algorithms is provided.

Ramalingam proved optimal slicing for concurrent programs with procedures and synchronization is undecidable [Ramalingam 2000]. Moreover, Müller-Olm et al. proved that optimal slicing is undecidable even if synchronization is ignored [Müller-Olm 2001]. It is proved by a reduction from the halting problem for two-counter machines. The theory that intraprocedural slicing stays PSPACE-hard is proved by a reduction from the REGULAR EXPRESSION INTERSECTION program. They also show us that slicing becomes NP-hard for loop-free programs. Garg et al. gave the necessary and sufficient condition for a computation slice to exist. An algorithm with O(N2|E|) complexity was shown by them [Garg 2001].

6 Non-Static Program Slicing

6.1 Dynamic Slicing

It is Korel and Laski [Korel 1988b, 1990] who first proposed the notion of dynamic slice. A dynamic slice is a part of a program that affects the concerned variable in a particular program execution. As only one execution is taken into account, dynamic program slicing may significantly reduce the size of the slice as compared to static slicing. It is not only useful in software debugging but also in software maintenance, program comprehension, and software testing. Lots of approaches have been presented on dynamic slicing. Survey of dynamic program slicing can be found in [Korel 1998a; Tip 1995a].

6.1.1 Basic dynamic slicing methods

Korel et al. [Korel 1988b, 1990] proposed a method to computes dynamic slices by solving the associated data-flow equations. They formalized the execution history of a program as a trajectory consisting of a sequence of occurrences of statements and control predicates. Labels serve to distinguish between different occurrences of a statement in the execution history. Several dynamic flow concepts are then introduced to formalize the dependences between occurrences of statements in a trajectory. Their method needs lots of space to store execution history. As executable slice is desired, the algorithm may produce inaccurate results.

Gopal [Gopal 1991] constructed non-executable dynamic slices based on dynamic dependence relations. His method is an extension of Bergeretti et al.'s information-flow relation [Bergeretti 1985]. Often the result slices of Gopal's method are similar as that of method presented in [Korel 1990] or even smaller. However, this method may produce non-terminating slices for terminating programs.

Miller and Choi [Miller 1988] first proposed the notion of dynamic dependence graph. However, their method mainly concentrates on parallel program debugging and flowback analysis.

Agrawal and Horgan [Agrawal 1989, 1990] developed an approach for using dependence graphs to compute non-executable dynamic slices. Their first two algorithms construct slices based on annotated program dependence graphs. The slicing result is not as precise as desired. Another algorithm is then proposed based on the notion of dynamic dependence graph (DDG). As DDG construct a node for each statement in the execution trace, it is too big. A Reduced Dynamic Dependence Graph (RDDG) was then proposed to reduce the size of DDG without losing the precision of slice. Tip pointed out that RDDG still might be exponential in the size of statements [Tip 1995a].

Goswami and Mall [Goswami 2002] presented a dynamic slicing algorithm based on the notion of compact dynamic dependence graph (CDDG). The control dependence edges of the CDDG are constructed statically while the data-flow dependence edges are constructed dynamically. This approach is space and time efficient compared to RDDG-based approaches.

Mund and Mall et al. [Mund 2002] used PDG as an intermediate program representation, and modified it by introducing the concepts of stable and unstable edges. Their algorithm is based on marking and unmarking the unstable edges of the Modified Program Dependence Graph (MPDG).

Mund et al. [Mund 2003] found that CDDG-based approaches [Goswami 2002] may not produce correct result in some cases. They proposed three intraprocedural dynamic slicing methods, two based on marked PDG and another based on their notion of Unstructured Program Dependence Graph (UPDG) which can be used for unstructured programs. Their first method also based on the runtime marking and unmarking of edges, while the other two based on the runtime marking and unmarking of nodes. It is claimed that all the three algorithms are precise and more space and time efficient than former algorithms.

Zhang et al. [Zhang 2003a] presented three precise dynamic slicing algorithms that differ in the degree of preprocessing they carry out prior to computing any dynamic slices. (Preprocessing is a process of building dependence graphs before do slicing based on them.) Of these algorithms, the limited preprocessing (LP) algorithm is practical. It performs some preprocessing to first augment the execution trace with summary information that allows faster traversal of the trace and then during slicing it uses demand driven analysis to recover the dynamic dependences from the compacted execution trace.

Zhang and Gupta [Zhang 2004a] found that different dynamic dependence could be expressed by one edge in the dependence graph. They presented a practical dynamic slicing algorithm which is based upon a novel representation of the dynamic dependence graph that is highly compact and rapidly traversable. Research indicated that this transformed dynamic dependence graphs could save significant amount of space compare to the full dynamic dependence graph and the corresponding algorithm is more efficient than that of [Zhang 2003a] both in time and space.

Forward computation of dynamic slices creates dynamic slices during program execution without recording of the execution trace and therefore save a lot of space. A forward algorithm starts from the first statement in the program, proceeds “forward” with program execution and at the same time performs the computation of the dynamic slices for program variables along with the program execution. Two main forward dynamic slicing have been proposed in the last decades.

Korel and Yalamanchili [Korel 1994] proposed a method of forward dynamic slicing based on the notion of removable blocks. The idea of their algorithm is that during program execution on each regular exit from a block the algorithm determines whether the executed block should be included in a dynamic slice or not. Beszédes et al. [Beszédes 2001a, 2001b] proposed another forward global dynamic slicing method for the C programming language. The method records the defined variable chain and used variable chain of each statement without maintaining a dependence graph. It computes slices by tracing recent variable definitions and control predicates in the execution history in parallel to the program execution.

Zhang et al. [Zhang 2004c] studied the statistical characteristics of dynamic slices by experiments. Based on the forward slicing methods, they introduced a way of using reduced ordered binary decision diagrams (roBDDs) to represent a set of dynamic slices. Within this technique, the space and time requirements of maintaining dynamic slices are greatly reduced. Thus, the efficiency of dynamic slicing can be improved. A comparison of backward computation and forward computation algorithms is given out in [Zhang 2004c] together with an introduction of their best using scenarios.

6.1.2 Interprocedural dynamic slicing

Several approaches have been presented concerning on interprocedural dynamic slicing.

Agrawal, DeMillo and Spafford [Agrawal 1991] considered dynamic slicing of procedures with various parameter-passing mechanisms. Dynamic data dependences based on definitions and uses of memory locations are used and therefore the use of global variables inside procedures does not pose any problems and no alias analysis is needed.

Kamkar et al. [Kamkar 1992, 1993b] further discussed the problem of interprocedural dynamic slicing. They proposed a method that primarily concerned with procedure level slices. This method is based on the dynamic dependence summary graphs constructed during program execution. An extended algorithm for compute dynamic slices at statement-level was then presented in [Kamkar 1993a]. Slices produced by Kamkar’s method are non-executable.

Song and Huynh [Song 1998] also proposed an interprocedural dynamic slicing method. Their algorithm is an extension of algorithm that presented in [Korel 1994].

6.l.3 Dynamic slicing in the presence of composite Datatypes and pointers

Korel and Laski [Korel 1990] considered slicing in the presence of composite variables by regarding each element of an array or field of a record as a distinct variable. Dynamic data structures are treated as two distinct entities, namely the pointer itself and the object being pointed to. For dynamically allocated objects, they proposed a solution where a unique name is assigned to each object. Agrawal et al. [Agrawal 1991] proposed a uniform approach to handle pointers and composite datatypes in dynamic slicing. The basic idea of this approach is that treat each variable as a memory cell, and dependences are analyzed by checking intersection between these memory cells. Faragó and Gergely [Faragó 2002] also discussed the problem of handling pointers, arrays and structures of C programs when doing forward dynamic slicing. Abstract memory locations are used in this method and program instrumentation is used to extract these locations.

6.1.4 Dynamic slicing in the presence of jump statements

Korel et al. [Korel 1995, 1997a] used a removable block based approach to handle jump statements in dynamic program slicing. This approach can produce correct slices in the presence of unstructured programs. Huynh and Song [Huynh 1997] then extended the forward dynamic slicing method presented in [Korel 1994] to handle jump statements. However, their algorithm can handle unstructured programs having only structured jumps. Mund, Mall and Sarkar [Mund 2003] proposed a notion of jump dependence. Based on this notion, they build the Unstructured Program Dependence Graph as the intermediate representation of a program. Their slicing algorithm based on UPDG can produce precise slices. Faragó and Gergely [Faragó 2001, 2002] handled jump statements for the forward dynamic slicing by building a transformed D/U structure for all relevant statements. This method can be applied to handle goto, break, continue and switch statements of C programs.

6.1.5 Dynamic slicing of concurrent and distributed programs

Korel and Ferguson [Korel 1992] extended the dynamic slicing method presented in [Korel 1988b, 1990] such that it can be used to slicing concurrent Ada programs with rendezvous communication. They formalized the execution history as a distributed program path and introduced the notion of communication influence to capture the interdependence between tasks. This slicing method is space costly.

Duesterwald et al. [Duesterwald 1992b] proposed a method for dynamic slicing distributed programs based on dependence graphs. They introduced Distributed Dependence Graph (DDG) to represent distributed programs. A DDG contains a single vertex for each statement and control predicate in the program. In this graph, control dependence edges are determined statically and data and communication dependence edges are added at run-time. Both the construction of the DDG and the computation of slices are performed in a distributed manner. This algorithm may produce inaccurate slices in the presence of loops.

Cheng [Cheng 1993] proposed an approach of dynamic slicing distributed and concurrent programs based on Process Dependence Net (PDN). PDN, in addition to the usual control and data dependence, represents selection, synchronization and communication dependences. Cheng's method is a generalization of the approach presented in [Agrawal 1990] and therefore may also compute inaccurate slices.

Kamkar et al. [Kamkar 1995a, 1996] then proposed a method based on Distributed Dynamic Dependence Graph (DDDG). A DDDG is a run-time constructed graph that and can represent the control, data and communication dependences.

Of these four methods, Korel-Ferguson [Korel 1992] and Duesterwald et al. [Duesterwald 1992b]'s methods produce executable dynamic slices, while the others produce non-executable ones. Only Kamkar et al. [Kamkar 1995a, 1996]'s method can do interprocedural dynamic slicing.

Besides these four methods, Goswami and Mall [Goswami 2000a] introduced a notion of a Dynamic Program Dependence Graph (DPDG) to represent various intra- and inter- process dependences of concurrent programs and presented a framework for computing dynamic slices of concurrent programs using such dependence graphs as intermediate representations. Two kinds of interprocess communication, shared memory and message passing mechanisms, have been considered in their work. Rilling et al. [Rilling 2002] presented a novel predicate-based dynamic slicing algorithm for message passing programs which analyzes and captures control, data and synchronization behavior of such programs with respect to a global predicate. Their slicing criterions are somewhat different with the traditional ones in order to fit the analyzing of distributed programs and the algorithm is an extension of the forward dynamic slicing method presented in [Korel 1994].

J. Cheng [Cheng 2001] proposed a methodological review for dynamic slicing of concurrent programs from the viewpoint of wholeness, uncertainty and self-measurement principles of concurrent systems. He reviewed the first four methods and gave out a discussion about the future of the dynamic slices of concurrent programs.

6.1.6 Dynamic slicing for logic language

Vasconcelos and Aragao [Vasconcelos 1998b] compared the existing program slicing techniques for procedural languages and logical languages and proposed some technique for adapting dynamic slicing of procedural language to logic programming. They presented a flexible framework for slicing logic programs [Vasconcelos 1998b, 1999]. It is not language-specific and can do both static slicing and dynamic slicing. As many practical aspects of logic programs are addressed in the framework, it can handle realistic programs. Szilagyi [Szilagyi 2002] studied the problem of dynamic slicing constraint logic programs. Biswas [Bisas1997] and Abadi [Abadi 1999] also discussed how to do dynamic slicing in higher-order programming languages.

6.1.7 Language independent dynamic slicing

Tip [Tip 1994] studied how to use two language-independent techniques, original tracking and dynamic dependence tracking, to derive powerful language-specific dynamic program slicing tools from algebraic specifications of interpreter. Field and Tip [Field 1998] defined a general notion of slice that can apply to any unconditional term rewriting system (TRS) and a dynamic dependence relation witch can be used to compute dynamic slice. Gouranton and Métayer [Gouranton 1998, 1999] proposed a notion of a dynamic slicing analysis format. The analysis format is a generic, language-independent, slicing analysis. They focused on dynamic analysis and introduced a generic analyzer which can be instantiated to derive a slicing analyzer for any programming languages conforming the semantics format they proposed.

6.1.8 Dynamic slicing of Object-Oriented programs

Zhao [Zhao 1998d] proposed a dynamic object-oriented dependence graph (DODG) which is an arc-classified digraph to represent various dynamic dependences between statement instances for a particular execution of an object-oriented program. A two-phase algorithm is given out for computing dynamic slices of OO program based on DODG.

Song and Huynh [Song 1999b] analyzed the message passing and the parameter passing of object-oriented programs and presented a notion of dynamic object relation diagrams (DORD).

New slice criterion is also defined such that slice can be computed for constructors and member functions as well as traditional statements. An extension of the forward dynamic interprocedural slicing method presented in [Song 1998] is then used to compute dynamic slice on DORD. Additional information of this method can be found in [Song 2001].

Xu et al. [Xu 2002] proposed a dynamic slicing for object-oriented (OO) programs based on dependence analysis. This approach used the object program dependence graph (OPDG) and other static information to reduce the information to be traced during program execution. It is a combination of both forward analysis and backward one. The slicing criterion is also modified to fit for debugging.

6.2 Relevant Slicing

Agrawal et al. [Agrawal 1993b] introduced an extension of dynamic slice: relevant slice. A relevant slice with respect to a variable contains not only the statements that have an influence on the variable but also those executed statements that did not affect the output, but could have affected it had they evaluated differently. Relevant slices are helpful in incremental regression testing. Based on the relevant slicing algorithms proposed in [Agrawal 1993b], Gyimóthy et al. [Gyimóthy 1999] presented a forward global method for computing relevant slices. They first compute the dynamic slice using a forward algorithm and then augment this algorithm with the computation of the potential dependences. Taking advantage of the forward slicing methods, this method is more space efficient than that of [Agrawal 1993b].

6.3 Conditioned Slicing

Canfora et al. introduced the notion of conditioned slice in [Canfora 1998]. A conditioned slice consists of a subset of program statements which preserves the behavior of the original program with respect to a slicing criterion for a given set of execution paths. The set of initial states of the program that characterize these paths is specified in the form of a first order logic formula on the input variables. Given a program and the set of initial states, the conditioned slicing algorithm first use a symbolic executor to reduce the program by discarding infeasible paths according to these initial states. Then slicing will be performed on the reduced program. As infeasible paths a discarded, the slicing result is more precise then that of traditional slicing methods.

Later, Harman et al. [Harman 2001b] presented and formalized the pre/post conditioned slicing method, which combines forward and backward conditioning to provide a unified framework for conditioned program slicing. The pre/post conditioned slicing can be used to improve the analysis of programs in terms of pre- and post- conditions.

Fox et al. introduced backward conditioning and illustrated its usage in [Fox 2001]. Like forward conditioning (used in conditioned slicing), backward conditioning consists of specializing a program with respect to a condition inserted into the program. However, it addresses questions of the form 'what parts of the program could potentially lead to the program arriving in a state satisfying condition c?' which is different from forward conditioning.

Several works about the conditional slicing tools have been presented. Harman et al. [Harman 2000a] introduced ConSIT, the first fully automated conditional slicing system which is based upon conventional static slicing, symbolic execution and theorem proving. The ConSIT system operates on a subset of C, for which a tokeniser and symbolic executor were written in Prolog. Other information about ConSIT can be found in [Fox 2003]. Daoudi et al. introduced ConSUS, a conditioner for the Wide Spectrum Language (WSL). The symbolic executor of ConSUS prunes the symbolic execution paths, and its predicate reasoning system uses the FermaT simplify transformation in place of a more conventional theorem prover. Results show that this combination of pruning and simplification-as-reasoner leads to a more scalable approach to conditioning. More information about ConSUS can be found in [Daninic 2004a].

Constrained slice [Field 1995] and quasi-static slice [Venkatesh 1991] are two kinds of slices that have many common with conditioned slice. A constrained slice is valid for all instantiation of the inputs that satisfy a given set of constraints and a quasi-static slice is constructed with respect to an initial prefix of the input sequence to the program.

6.4 Union Slicing

Hall [Hall 1995] introduced the notion of simultaneous dynamic program slicing to extract executable program subsets. The basic approach is to apply any kind of dynamic slicing algorithm that meets certain criteria (one of which is to be able to produce executable slices) and incrementally builds the simultaneous slice using an iterative algorithm for all test cases.

Simultaneous dynamic program slicing can facilitate program subsetting and redesign.

Beszedes [Beszedes 2002a, 2002b] introduced the concept of union slices in and their computing algorithm. A union slice is the union of dynamic slices for a (finite) set of test cases, which is very similar as simultaneous dynamic program slicing. A union slice is an approximation of a static slice and is much smaller than the static one. Combined with static slices, they can help to reduce the size of program parts that need to be investigated by concentrating on the most important parts first. Thus, union slices are useful in software maintenance.

Daninic et al. [Daninic 2004b] presented an algorithm for computing executable union slices, using conditioned slicing. The work showed that the executable union slice is not only applicable for program comprehension, but also for applicable component reuse guided by software testing.

De Lucia et al. [De Lucia 2003] studied the properties of union of slices and found that the union of two static slices is not necessarily a valid slice, based on Weiser's definition of a (static) slice. Reasons of this property are considered and some new questions are presented in [De Lucia 2003].

6.5 Hybrid Slicing

As static slicing suffer from the problem of imprecision and dynamic slicing is specific to only one execution, Gupta et al. [Cupta 1995] presented a general approach for improving the precision and quality of static slices by incorporating dynamic information in static slicing. They proposed a concept of hybrid slicing, an slicing technique that exploits information readily available during debugging when computing slices statically. More information about this hybrid slicing can be found in [Gupta 1997a]. With in the idea of hybrid slicing, Schoenig and Ducassé [Schoenig 1995] proposed a hybrid backward slicing algorithm for Prolog which can produce executable slices. Their algorithm is the first one proposed for Prolog.

Dependence-cache slicing is also a kind of slicing that incorporate both static and dynamic information. Information about this slicing method can be found in [Takada 2002; Umemori 2003]. Nishimatsu et al. [Nishimatsu 1999] proposed the concept of call-mark slicing, a slicing method combines static analysis of a program's structure with lightweight dynamic analysis. In their approach, the data dependences and control dependences among the program statements are statically analyzed beforehand, and procedure/function invocations are recorded during execution. From this information, the dynamic dependences of the variables are explored. This method produces more precise slices than static slicing methods and is more efficient than dynamic slicing when execution needs lots of time.

Other similar approaches that use both static and dynamic information have been proposed in [Choi 1991; Duesterwald 1992b; Kamkar 1993b; Netzer 1994]. These approaches use static information to improve the execution time performance of dynamic slicing while maintaining the precision of dynamic slicing. Rilling presented a hybrid slicing framework and introduced two general hybrid slicing algorithms in [Rilling 2001].

7 Beyond Traditional Program Slicing

Besides traditional program slicing, several studies investigating the semantic basis of program slicing have also been conducted [Cartwright 1989; Harman 1994a; Hwang 1988b; Sloane 1996]. Harman et al.'s amorphous slicing [Harman 1997c] allows for any simplifying transformations that preserve this semantic projection, thereby improving upon the simplification power of traditional slicing and consequently its applicability to program comprehension. However, this method is not really based on formal semantics of a program. Harman et al. [Harman 1997c] also proposed an amorphous slicing algorithm based on system dependence graph. In [Harman 2002b], an interprocedural amorphous slicing algorithm for Ward's Wide Spectrum Language is proposed. A new amorphous slicing algorithm that slices directly from abstract syntax tree is given out in [Harman 2004a]. P. A. Hauser's denotational slicing [Hausler 1989] employed the functional semantics of a program language in the denotational (and static) program slicer. L.Ouarbya [Ouarbya 2002] extended it to fit for the static slicing of interprocedural subsequently. G.A.Venkatesh [Venkatesh 1991] also took account of denotational slicing with formal slicing algorithms including dynamic and static. This approach is indeed based on the standard denotational semantics of a program language. The language Venkatesh considered is a very simply one without procedures. T. Reps and W. Yang [Reps 1988a, 1989a, 1989b] presented the operational semantics of static program slicing based on the semantics of the program dependence. J. Field et al.'s parametric program slicing [Field 1995] shows how to mechanically extract a family of practical algorithms for computing slices directly from semantic specifications. These algorithms are based on combining the notion of dynamic dependence tracking in term rewriting systems with a program representation whose behavior is defined via an equational logic.

Two variations on the slicing theme, which are closely related to slicing, are chopping [Jackson 1994b] and dicing [Chen 1993; Lyle 1986]. In chopping dependence chains from a source (of the dependence) to a sink (of the dependence) are identified. Chopping tries to find the program part that causes one variable s affect variable t. However, Jackson and Rollins defined only a limited form of chopping: Among other restrictions, they imposed the restriction that s and t be in the same procedure. Reps and Rosay [Reps 1995b] then solved the unrestricted interprocedural chopping problem, as well as a variety of other useful variants of interprocedural chopping. Krinke [Krinke 2002] then presented an evaluation of context-sensitive chopping with the context-insensitive chopping. He proposed an approximative chopping method and found that this method is much faster and can produce results almost as precise as that of Reps'. In [Krinke 2003a, 2004a], he also presented an approach that use barriers to further reduce chopping for program understanding. Program dicing which was first proposed by Lyle [Lyle 1987] tries to find the difference between static slices of two variables. Dicing is often used to automatically identify a set of statements likely to contain the bug when computation on some variables fails while computation on other variables successes. T.Y Chen and Y.Y. Cheung [Chen 1993, 1997] introduced dynamic program dicing to facilitate debugging. The dynamic dices are often smaller thus more suitable for debugging. After T. Y. Chen et al., Samadzadeh and Wichaipanitch [Samadzadeh 1993; Wichaipanitch 2003] then studied the implementation of C/C++ debugging tool incorporate dynamic dicing.

8 Applications of Program Slicing

8.1 Software Quality Assurance

The original motivation for program slicing was to aid the location of faults during debugging activities. The idea was that the slice would contain the fault, but would not contain lines of code that could not have caused the failure observed. This is achieved by setting the slicing criterion to be the variable for which an incorrect value is observed. Weiser [Lyle 1986; Weiser 1982, 1986] investigated the ways in which programmers mentally slice a program when attempting to understand and debug it, and this formed an original motivation for the consideration of techniques for automated slicing.

Clearly slicing cannot be used to identify bugs such as 'missing initialization of variable'. If the original program does not contain a line of code then the slice will not contain it either. Although slicing cannot identify errors of omission, it has been argued that slicing can be used to aid the detection of such errors [Harman 1997c].

In debugging, one is often interested in a specific execution of a program that exhibits anomalous behavior. Dynamic slicing is one variation of program slicing introduced to assist in debugging [Korel 1988a]. Dynamic slices are particularly useful here [Agrawal 1993a; Korel 1997b], because they only reflect the actual dependences of that execution, resulting in smaller slices than static ones. Agrawal's thesis [Agrawal 1992] contains a detailed discussion how static and dynamic slicing can be utilized for semi-automated debugging of programs. He proposed an approach where the user gradually 'zooms out' from the location where the bug manifested itself by repeatedly considering larger data and control slices.

Fritzson et al. used interprocedural static [Fritzson 1992] and dynamic [Kamkar 1993a] slicing for algorithmic debugging [Shahmehri 1991; Shapiro 1982]. An algorithmic debugger partially automates the task of localizing a bug by comparing the intended program behavior with the actual program behavior. The intended behavior is obtained by asking the user whether or not a program unit (e.g., a procedure) behaves correctly. Using the answers given by the user, the location of the bug can be determined at the unit level. Debugging is also the motivation for program dicing and latter program chopping [Jackson 1994b; Lyle 1987; Reps 1995b].

Testing is an important part of software engineering as it consumes at least half of the labor expended to produce a working program [Beizer 1990].

A program satisfies a 'conventional' dataflow testing criterion if all def-use pairs occur in a successful test-case. Duesterwald, Gupta, and Soffa proposed a more rigorous testing criterion, based on program slicing in [Duesterwald 1992a]. The advantage of this approach is that it combines reasonable efficiency with reasonable precision. In [Kamkar 1993c], Kamkar, Shahmehri, and Fritzson extended the work of Duesterwald, Gupta, and Soffa to multi-procedure programs.

Initial work in this area has shown that amorphous static slicing forms a good support to detection of equivalent mutants in mutation testing [Hierons 1999], that conditioned slicing can complement partition-base testing [Hierons 2000] and that slicing and related dependence analyses can be use to support mutation testing [Harman 2000b].

Regression testing is part of the larger problem of program testing. Appling program slicing to the problem of reducing the cost of regression testing has been studied for a long time. Researches on this area can be divided into three groups. The first group uses dynamic slicing [Agrawal 1990; Korel 1988a]. The second group represents programs using program dependence graphs [Ferrante 1987; Horwitz 1989a] and the third group is based on Weiser's data flow definition of slicing [Weiser 1984].

Agrawal et al. [Agrawal 1993b] presented three algorithms for reducing the cost of regression testing. One drawback of the relevant slice strategy is that it requires static data dependence information. This static information can be expensive to compute and is necessarily inaccurate due to the use of pointers, arrays and dynamic memory allocation. Agrawal et al. suggested an approximation that is simpler, but more conservative.

In [Bates 1993], Bates and Horwitz used a variation of the PDG notion of [Horwitz 1989a] for incremental program testing. Bates and Horwitz presented test case selection algorithms for the all vertices and all flow-edges test data adequacy criterion. They proved that statements in the same class are exercised by the same test cases. The work of Bates and Horwitz considered single procedure programs. Binkley [Binkley 1997] presented two complementary algorithms for reducing the cost of regression testing that operate on programs with procedures and procedure calls. There is another dependence graphs technique makes limited use of program slicing. Rothermel and Harrold [Rothermel 1994] presented algorithms that select tests for affected du-pairs, and that determine those du-pairs that may produce different behavior. Both intraprocedural and interprocedural versions are presented. This is in contrast to the test-case selection algorithms presented by Bates and Horwitz and by Binkley, which are not safe in the Rothermel and Harrold sense because of their treatment of deleted components [Rothermel 1996].

The final pair of techniques is based of Weiser's original dataflow model for computing program slices. Gupta et al., presented an algorithm for reducing the cost of regression testing that uses slicing to determine components affected transitively by an edit at point p [Gupta 1992b]. However, the algorithms are designed to determine the information necessary for regression testing only. The approach is self-described as "slicing-based" because it does not directly use slicing operators, but rather is based on the slicing algorithms introduced by Weiser. Gallagher and Lyle introduced the notion of "decomposition slicing" and its use in "a new software maintenance process model which eliminates the need for regression testing" [Gallagher 1991a, 1991b]. This model decomposes the program into two executable parts: a decomposition slice and its complement.

In addition, program differencing [Binkley 1992, 2002; Horwitz 1989b] can be used to further reduce the cost of regression by reducing the size of the program that the tests must be run on. Calling context is more accurately accounted for by replacing equivalent execution patterns with the weaker notions of common execution patterns [Binkley 1995b].

8.2 Software Maintenance and Re-engineering

Slicing technologies can be used to software maintenance. [Gallagher 1991b] introduced the concept of decomposition slicing and discussed its application to software maintenance. A decomposition slice is defined with respect to a variable v, independently of any program point. It captures all the computation on a given variable.

The need to integrate several versions of a program into a common one arises frequently. In some approaches of program integration [Berzins 1995; Horwitz 1989a; Reps 1989a], slicing techniques play an important role. They are used to facilitate the study of program behavior.

Program slicing can be used to do software architecture analysis. Zhao [Zhao 1998c] introduced architectural slicing to aid architectural understanding and reuse. Given a software architecture, an architecture slice contains a special set of components, connectors and configuration of the architecture with respect to some architectural slicing criteria. Besides Zhao, Kim et al. also [Kim 1999a, 1999b, 2000] proposed some approaches to using dynamic slicing to analysis the properties of software architecture.

Slicing can also be used to identify reusable functions [Canfora 1994a, 1994b; Cimitile 1995a, 1995b; Lanubile 1997]. Canfora et al. presented a method to identify functional abstraction in existing code in [Canfora 1994a]. In this approach, program slicing is used to isolate the external functions of a system and these are then decomposed into more elementary components by intersection slices. They also found that conditioned slicing could be used to extract procedures from program functionality [Canfora 1994b]. Cimitile et al. [Cimitile 1995a, 1995b] presented a case study of using specification driven slicing to identify and isolate functions in C programs. In this approach, the specification of the function to be isolated is used together with symbolic execution and theorem proving techniques to identify the slicing criterion. Lanubile and Visaggio [Lanubile 1997] proposed a method to extract reusable functions from illstructured programs using transform slicing which is a variant of the conventional program slice. A transform slice only includes statements which contribute directly or indirectly to transform a set of input variables into a set of output variables, i.e., it do not include either the statements necessary to get input data or the statements which test the binding conditions of the function.

Changing the internal structure of a program without changing its behavior is called restructuring. [Canfora 2000; Kang 1998; Kim 1994; Lakhotia 1998] studied the problem of restructuring. Kim et al. [Kim 1994] proposed a reconstructing method based on program metrics. They used a notion of module strength (cohesion) and defined processing blocks which are similar to the data slices of Bieman and Ott [Bieman 1994]. Modules with low module strength are then split while other modules are decomposed and the resulting components are grouped. Lakhotia et al. [Lakhotia 1998] introduced a transformation called tuck for restructuring programs by decomposing large functions into small functions. Tuck consists of three steps: Wedge, Split, and Fold. A wedge, a subset of statements in a slice, contains computations that are related and that may create a meaningful function. The statements in a wedge are split from the rest of the code and folded into a new function. That tuck does not alter the behavior of the original function follows from the semantic preserving properties of a slice. Canfora et al. [Canfora 2000] proposed an approach to program decomposition as a preliminary step for the migration of legacy systems. In the approach, program slicing techniques is used for identifying the set of program statements that contribute to implement database and user interface components.

Besides above application in software maintenance, program slicing can also be used in reverse engineering [Beck 1993; Jackson 1994a]. Beck and Eichmann [Beck 1993] applied program slicing techniques to reverse engineering by using it to assist in the comprehension of large software systems, through traditional slicing techniques at the statement level, and through a new technique, interface slicing, at the module level. A dependence model for reverse engineering should treat procedures in a modular fashion and should be fine-grained, distinguishing dependences that are due to different variables. Jackson and Rollins [Jackson 1994a] proposed an improved program dependence graph (PDG) that satisfies both, while retaining the advantages of the PDG. They proposed an algorithm to compute chopping form their dependence graph which can produces more accurate results than algorithms based directly on the PDG.

8.3 Measurement

Cohesion and coupling are two important metrics in software measurement.

Cohesion is an attribute of a software unit that purports to measure the “relatedness" of the unit. It has been qualitatively characterized as coincidental, logical, procedural, communicational, sequential and functional, with coincidental being the weakest and functional being the strongest.

Several approaches of using program slicing to measure cohesion have been presented. It is Longworth [Longworth 1985] that first studied the use of program slicing as indicator of cohesion.

Ott and Thuss [Ott 1989] then noted the visual relationship that existed between the slices of a module and its cohesion as depicted in a slice profile. In [Ott 1992a, 1992b, 1993; Thuss 1988] certain inconsistencies noted by Longworth are eliminated through the use of metric slices. A metric slice takes into account both uses and used by data relationships; that is, they are the union of Horwitz et al.’s backward and forward slices.

Bieman and Ott [Bieman 1994] examined the functional cohesion of procedures using a data slice abstraction. A data slice is a backward and forward static slice that uses data tokens rather than statements as the unit of decomposition. Their approach identifies the data tokens that lie on more than one slice as the "glue" that binds separate components together. Cohesion is measured in terms of the relative number of glue tokens, tokens that lie on more than one data slice, and super-glue tokens, tokens that lie on all data slices in a procedure, and the adhesiveness of the tokens. Harman et al. [Harman 1995d] improved the cohesion metrics introduced by Ott and her colleagues by replacing the use of 'Line of Code' metric with their expression metrics proposed to evaluate the complexity of an expression. This work can give a more accurate measure of cohesion. Later, Ott and Bieman [Ott 1998] found a way of using syntax preserving static program slicing to measure program cohesion.

Cohesion may also be measured using only procedure interface information. Bieman and Kang derived design-level cohesion measures in [Kang 1996; Bieman 1998]. They defined design-level functional cohesion measures only in terms of the dependencies between module input and output components. Leminen applied the concept of slicing to study the cohesion of Z formal specification schemas [16]. Ott et al. [Ott 1995] and Gupta [Gupta 1997b] extended the slice-based approaches for measuring object-oriented cohesion.

A review of the slice-based cohesion measures for both the procedural paradigm and other paradigms has been presented in [Ott 1998]. Empirical results of slice-based cohesion measurement can be found in [Karstu 1994].

Coupling is the measure of how one module depends upon or affects the behavior of another. Harman et al. [Harman 1997b] proposed a method of using program slicing to measure coupling. It is claimed that this method produce more precise measurement than information flow based metrics. Li [Li 2001a] presented a framework to measure coupling of object-oriented program based on hierarchical slice model [Li 2000].

Similar to the work that uses program slicing to facilitate software measurement, several approaches have been proposed using the dependence analysis upon which slicing is based to measure cohesion. Lakhotia used the dependence analysis to formalize the seven informally defined levels of cohesion [Lakhotia 1993b]. Recently, new approaches have been proposed in [Chen 2002c, 2003a; Zhou 2002].

8.4 Comprehension

Slicing can help with the comprehension phase of maintenance. Several kinds of slice have been used in program comprehension. De Lucia, Fasolino and Munro [De Lucia 1996] used conditioned slicing to facilitate program comprehension. Constrained slicing [Field 1995] and quasi-static slicing [Venkatesh 1991] can also be used in program comprehension. These slicing techniques share the property that a slice is constructed with respect to a condition in addition to the traditional static and thus can give the maintainer the possibility to analyze code fragments with respect to different perspectives. Harman et al. [Harman 1995a] found that the restriction to statement deletion in slice construction is an unnecessary hindrance when it is applied to program comprehension. Thus, amorphous program slicing [Binkley 2000; Harman 1997c], which relaxes the syntactic constraint of traditional slicing, is also of much use in program comprehension. In addition to these slicing techniques, decomposition slicing [Gallagher 1991b, 2001] which captures all computation on a variable also has been found helpful in program comprehension. Beside these statement level slices, Korel and Rilling [Korel 1998b] presented several program slicing concepts on the module level like call graph slicing. Static and dynamic program slicing using these concepts can better facilitate the program understanding process of large programs.

Slicing can be used in many aspect of program comprehension. Harman, Sivagurunathan and Danicic [Harman 1998b; Sivagurunathan 2002] used program slicing in understanding dynamic memory access properties. Komondoor and Horwitz presented an approach that use program dependence graph and slicing to find duplicated code fragments in C programs [Komondoor 2001]. Henrard et al. [Henrard 1998] made use of program slicing in database understanding. Korel and Rilling used dynamic slicing to help understand the program execution [Korel 1997c].

Slice visualization takes an important part when using slicing to assist program comprehension. Several approaches for slice visualization have been presented [Agrawal 1993a; Anderson 2001; Ball 1994a; Deng 2001; Ernst 1994; Gallagher 1996, 1997; Jackson 1994c, 1994b; Krinke 2004c; Kuhn 1995; Reps 1993; Steindl 1998c, 1999b]. Most of slice interfaces introduced in these papers support interaction and allowed multi-level slicing. Of these approaches, [Deng 2001; Gallagher 1996, 1997; Kuhn 1995] introduced tools which can represent the program in graphic model, while others just represent slices in textual model. Together with the visualization of slices [Anderson 2001; Antoniol 1997; Balmas 2001; Krinke 2004c; Richardson 1992] presented some methods of visualizing program dependence graphs.

8.5 Other Applications

Besides the above applications, program slicing can also be used in dead code elimination [Bodik 1997], model construction

[Dwyer 1999b; Hatcliff 2000], model checking [Millett 1998, 2000], garbage collection [Plakal 2000] and so on.

9 Acknowledgments

We thank Bixin Li and Jianjun Zhao for their support of this work. They reviewed this paper and provided many fruitful discussions and comments on its earlier drafts. We thank David Binkley, Andrea De Lucia, Mark Harman, Mariam Kamkar, Jeff Russell, and Frank Tip for their earlier reviews on program slicing. Besides, we also thank the anonymous referees who have given remarks in their papers that are cited in our paper.

10 References

[Abadi 1999] Martín Abadi, Anindya Banerjee, Nevin Heintze and Jon G. Riecke, Dynamic slicing in High-Order programming languages, Proceedings of the 26th ACM SIGPLAN-SIGACT symposium on Principles of programming languages, pp. 147-160, 1999.

[Agrawal 1988] Hiralal Agrawal, Richard DeMillo and Eugene Spafford, A process state model to relate testing and debugging, Technical Report SERC-TR-27-P, Purdue University, 1988.

[Agrawal 1989] Hiralal Agrawal and Joseph R. Horgan, Dynamic program slicing, Technical Report SERC-TR-56-P, Purdue University, 1989.

[Agrawal 1990] Hiralal Agrawal and Joseph R. Horgan, Dynamic program slicing, Proceedings of the ACM SIGPLAN'90 Conference on Programming Language Design and Implementation, pp. 246-256, Jun. 1990. (SIGPLAN Notices 25(6), Jun. 1990)

[Agrawal 1991] Hiralal Agrawal, Richard A. DeMillo and Eugene H. Spafford. Dynamic slicing in the presence of unconstrained pointers, Proceeding of ACM Fourth Symposium on Testing, Analysis, and Verification (TAV4), pp. 60-73, 1991.

[Agrawal 1992] Hiralal Agrawal, Towards automatic debugging of Computer Programs, PhD thesis, Purdue University, 1992.

[Agrawal 1993a] Hiralal Agrawal, Richard DeMillo and Eugene Spafford, Debugging with dynamic slicing and backtracking, Software— Practice and Experience, 23(6), pp. 589-616, Jun. 1993.

[Agrawal 1993b] Hiralal Agrawal, Joseph R. Horgan, Edward W. Krauser and Saul A. London, Incremental regression testing, Proceedings of the Conference on Software Maintenance-1993, pp. 1-10, 1993.

[Agrawal 1994] Hiralal Agrawal, On slicing program with jump statements, Proceedings of the ACM SIGPLAN'94 Conference on Programming Language Design and Implementation (Orlando, Florida, 1994).

[Agrawal 2001] Gagan Agrawal and Liang Guo, Evaluating explicitly context-sensitive program slicing, PASTE 2001, pp. 6-12.

[Ahn 1999] Joonseon Ahn and Taisook Han, Static slicing of a first-order functional language based on operational semantics, Korea Advanced Institute of Science & Technology (KAIST) Technical Report CS/TR-99-144, Dec. 1999.

[Allen 1972] Frances E. Allen and John Cocke, Graph theoretic constructs for program control flow analysis, Technical Report RC 3923 (No. 17789), IBM, Thomas J. Watson Research Center, New York, July 1972.

[Allen 1983] J. R. Allen, Ken Kennedy, Carrie Porterfield, and Joe Warren, Conversion of control dependence to data dependence, Conf. Rec. of the POPL-10, Austin, Texas, pp. 177-189, Jan. 1983.

[Allen 2003] Matthew Allen and Susan Horwitz, Slicing java programs that throw and catch exceptions, ACM SIGPLAN Notices, 2003.

[Ammarguellat 1992] Zahira Ammarguellat, A control-flow normalization algorithm and its complexity, IEEE Trans. on Software Eng., 18(3), pp. 237-250, 1992.

[Amme 1998] Wolfram Amme and Eberhard Zehendner, Data dependence analysis in programs with pointers, Parallel Computing, 24(3-4), pp. 505 - 525, May 1998.

[Andersen 1994] Lars Ole. Andersen, Program analysis and specialization for the C programming language, PhD thesis, DIKU, University of Copenhagen, May 1994.

[Anderson 2001] Paul Anderson and Tim Teitelbaum, Software inspection using Codesurfer, In Workshop on Inspection in Software Engineering (CAV 2001), 2001.

[Antoniol 1997] Giuliano Antoniol, Roberto Fiutem, G. Lutteri, Paolo Tonella, S. Zanfei and Ettore Merlo, Program understanding and maintenance with the CANTO environment, Proceedings of International Conference on Software Maintenance, pp. 72-81, 1997.

[Antoniol 1999] Giuliano Antoniol, F. Calzolari, Paolo Tonella, Impact of function pointers on the call graph, Proceedings of the Third European Conference on Software Maintenance and Reengineering, p. 51, 1999.

[Ashcroft 1971] E. Ashcroft and Zohar Manna, The translation of verb -goto- programs into verb -while- programs, IFIP Congress, Amsterdam, 1971.

[Atkinson 1996] Darren C. Atkinson, William G. Griswold, The design of whole-program analysis tools, Proceedings of the 18th International Conference on Software Engineering, Berlin, IEEE, pp. 16-27, Mar. 1996.

[Atkinson 1998] Darren C. Atkinson and William G. Griswold, Effective whole-program analysis in the presence of pointers, Proceedings of the ACM SIGSOFT Symposium on the Foundations of Software Engineering, pp. 131-142, Nov. 1998.

[Atkinson 2001] Darren C. Atkinson and William G. Griswold, Implementation techniques for efficient data-flow analysis of large programs, Proceedings of International Conference on Software Maintenance, pp. 52-61, 2001.

[Atkinson 2002] Darren C. Atkinson, Markus Mock, Craig Chambers and Susan J. Eggers, Program slicing using dynamic points-to data, ACM SIGSOFT 10th Symposium on the Foundations of Software Engineering (FSE), Nov. 2002.

[Ball 1989] Thomas Ball, Susan Horwitz and Thomas Reps, Correctness of an algorithm for reconstituting a program from a dependence graph, Computer Sciences Dept., Univ. of Wisconsin, Madison, Technical Report in preparation, Spring 1989.

[Ball 1993a] Thomas Ball and Susan Horwitz, Slicing program with arbitrary control-flow, Proceedings of the First International Workshop on Automated and Algorithmic Debugging (1993), P. Fritzson, Ed., vol. 749 of Lecture Notes in Computer Science, Springer-Verlag, pp. 206-222.

[Ball 1993b] Thomas Ball, The use of control-flow and control dependence in software tools, PhD thesis, University of Wisconsin-Madison, 1993.

[Ball 1994a] Thomas Ball and Stephen G. Eick, Visualizing program slices, Proceedings of IEEE Symposium on Visual Languages, pp. 288-295, 1994.

[Ball 1994b] Thomas Ball and James R. Larus, Optimally profiling and tracing programs, ACM Transactions on Programming Languages and Systems, 16(4), pp. 1319-1360, July 1994.

[Balance 1992] Robert A. Balance and Arthur B. Maccabe, Program dependence graphs for the rest of us, Technical Report, The University of New Mexico, Oct. 1992.

[Balmas 2001] Francoise Balmas, Displaying dependence graphs: a hierarchical approach, Proceedings of 8th Working Conference on Reverse Engineering, pp. 261-270, 2001.

[Baker 1977] Brenda S. Baker, An algorithm for structuring flowgraphs, Journal of the ACM, 24(1), pp. 98-120,1977.

[Banerjee 1979] Utpal Banerjee, Speedup of ordinary programs, PhD thesis, University of Illinois at Urbana-champaign, Oct. 1979.

[Banerjee 1988] Utpal Banerjee, Dependence analysis for supercomputing, Kluwer Academic, 1988.

[Bates 1993] Samuel Bates and Susan Horwitz, Incremental program testing using program dependence graphs, Conference Record of the Twentieth ACM Symposium on Principles of Programming Languages, pp. 384-396, 1993.

[Beck 1993] John Beck and David Eichmann, Program and interface slicing for reverse engineering, Proceedings of 15th International Conference on Software Engineering, pp. 509-518, 1993.

[Beizer 1990] Boris Beizer, Software Testing Techniques, van Nostrand Reinhold, second edition, 1990.

[Bent 2000] Leeann Bent, Darren C. Atkinson and William G. Griswold, A qualitative study of two whole-program slicers for C, Technical Report CS2000-0643, Univ. of California at San Diego, (a preliminary version appeared at FSE 2000) (2000).

[Bergeretti 1985] Jean Francois Bergeretti and Bernard A. Carré, Information-flow and data-flow analysis of while-programs, ACM Transactions on Programming Languages and Systems, 7(1), pp. 37-61, Jan. 1985.

[Berzins 1995] Valdis Berzins, Software merging and slicing, IEEE Computer Society Press, Los Alamitos, 1995.

[Beszédes 2001a] Árpád Beszédes, Tamás Gergely, Zsolt Mihály Szabó, Csaba Faragó and Tibor Gyimóthy, Forward computation of dynamic slices of C programs. Technical Report TR-2000-001, RDAI, 2000.

[Beszédes 2001b] Árpád Beszédes, Tamás Gergely, Zsolt Mihály Szabó, János Csirik and Tibor Gyimóthy, Dynamic slicing method for maintenance of large C programs. Proceedings of the 5th European Conference on Software Maintenance and Reengineering (CSMR 2001), Lisbon, Portugal, Mar. 2001.

[Beszedes 2002a] Arpad Beszedes, Csaba Farago, Zsolt Mihaly Szabo and Tibor Gyimothy, Union slices for program maintenance, International Conference on Software Maintenance, Montreal, Oct. 2002.

[Beszedes 2002b] Arpad Beszedes and Tibor Gyimothy, Union slices for the approximation of the precise slice, IEEE International Conference on Software Maintenance (ICSM'02), pp. 12-20, Montreal, Canada, Oct. 2002.

[Bieman 1994] James M. Bieman and Linda M. Ott, Measuring functional cohesion, IEEE Transactions on Software Engineering, 20(8), pp. 644-657, 1994.

[Bieman 1998] James M. Bieman and Byung-Kyoo Kang, Measuring design-level cohesion, IEEE Transaction on Software Engineering, 24(2), pp. 111-124, Feb. 1998.

[Binkley 1992] David Binkley, Using semantic differencing to reduce the cost of regression testing, Proc. IEEE Conference on Software Maintenance, Washington, D. C., pp. 41-50, Nov. 1992.

[Binkley 1993a] David Binkley, Precise executable interprocedural slices, ACM Letters on Programming Languages and Systems, 2(1-4), pp. 31-45, 1993.

[Binkley 1993b] David Binkley, Slicing in the presence of parameter aliasing. Proceedings of the 1993 Software Engineering Research Forum, (Orlando, FL), pp. 261-268, Nov. 1993.

[Binkley 1995a] David Binkley, Susan Horwitz, and Thomas Reps, Program integration for languages with procedure calls, ACM Transactions on Software Engineering and Methodology, 4(1), pp. 3-35, Jan. 1995.

[Binkley 1995b] David Binkley, Reducing the cost of regression testing by semantics guided test case selection, IEEE International Conference on Software Maintenance, 1995.

[Binkley 1996] David Binkley and Keith Brian Gallagher, Program slicing, Advances in Computers, 43, pp. 1-50, 1996.

[Binkley 1997] David Binkley, Semantics guided regression test cost reduction, IEEE Transactions on Software Engineering, 23(8), pp. 489-516, Aug. 1997.

[Binkley 1998a] David W. Binkley and James R. Lyle, Application of the pointer state subgraph to static program slicing, The Journal of Systems and Software, pp. 17-27, 1998.

[Binkley 1998b] David Binkley, The application of program slicing to regression testing, Information and Software Technology, 40(11-12), pp. 583-594, 1998.

[Binkley 1999] David Binkley, Computing amorphous program slices using dependence graphs and a data-flow model, ACM Symposium on Applied Computing, pp. 519-525, 1999.

[Binkley 2000] David Binkley, L. Ross Raszewski, Christopher Smith and Mark Harman, An empirical study of amorphous slicing as a program comprehension support tool, Proceedings of the

IEEE Eighth International Workshop on Program Comprehension, Limerick, Ireland, pp. 161-170, Jun. 2000.

[Binkley 2002] David Binkley, An empirical study of the effect of semantic differences on programmer comprehension, Proceedings of the IEEE Eighth International Workshop on Program Comprehension, Paris, France, pp. 97-106, Jun. 2002.

[Binkley 2003] David Binkley, Mark Harman, A large-scale empirical study of forward and backward static slice size and context sensitivity, Proceeding of IEEE International Conference on Software Maintenance (ICSM 2003), Amsterdam, Netherlands, p. 44, 2003.

[Binkley 2004a] David Binkley and Mark Harman, A survey of empirical results on program slicing, Advances in Computers, 62, pp. 105-178, 2004.

[Binkley 2004b] David Binkley, Sebastian Danicic, Tibor Gyimóthy, Mark Harman, Ákos Kiss and Lahcen Ouarbya, Formalizing executable dynamic and forward slicing, Proceedings of the 4th IEEE Workshop on Source Code Analysis and Manipulation, Chicago, USA, pp. 15-16, Sep. 2004.

[Bisas 1997] Sandip K. Biswas, Dynamic slicing in higher-order programming languages, PhD thesis, University OF Pennsylvania, 147 pages, 1997.

[Bodik 1997] Rastislav Bodik and Rajiv Gupta, Partial dead code elimination using slicing transformations, Proceedings of Conference on Programming Language Design and Implementation, pp. 159-170, 1997.

[Burke 1994] Michael Burke, Paul Carini, Jong-Deok Choi and Michael Hind, Flow-insensitive interprocedural alias analysis in the presence of pointers, In Proceedings of the Seventh International Workshop on Languages and Compilers for Parallel Computing, Aug. 1994.

[Callahan 1988] David Callahan, The program summary graph and flow-sensitive interprocedural data flow analysis, Proceedings of the ACM SIGPLAN 88 Conference on Programming Language Design and Implementation, (Atlanta, GA, June 1988), ACM SIGPLAN Notices 23(7), pp. 47-56, July 1988.

[Callahan 1990] David Callahan, Alan Carle, Mary Wolcott Hall and Ken Kennedy, Constructing the procedure call multigraph, IEEE Transactions on Software Engineering, 16(4), pp. 483-487, 1990.

[Canfora 1994a] Gerardo Canfora, Andrea De Lucia, Giuseppe A. Di Lucca and A. R. Fasolino, Slicing large programs to isolate reusable functions, Proceedings of EUROMICRO Conference, Liverpool, U.K, IEEE CS Press, pp. 140-147, 1994.

[Canfora 1994b] Gerardo Canfora, Andrea De Cimitile, A. De Lucia, Giuseppe A. Di Lucca, Software salvaging based on conditions, Proceedings of International Conference on Software Maintenance, pp. 424-433, 1994.

[Canfora 1998] Gerardo Canfora, Aniello Cimitile, and Andrea De Lucia. Conditioned program slicing. In Mark Harman and Keith Gallagher, editors, Information and Software Technology Special Issue on Program Slicing, volume 40, pages 595–607.Elsevier Science B.V., 1998.

[Canfora 2000] Gerardo Canfora, Aniello Cimitile, Andrea De Lucia and Giuseppe A. Di Lucca, Decomposing legacy programs: a first step towards migrating to client-server platforms, The Journal of Systems and Software, 54, pp. 99-110, 2000.

[Cartwright 1989] Robert Cartwright and Matthias Felleisen, The semantics of program dependence, ACM SIGPLAN Conference on Programming Language Design and Implementation, pp. 13-27, 1989.

[Chang 1994] Juei Chang and Debra J. Richardson, Static and dynamic specification slicing, Proceedings of the Fourth Irvine Software Symposium, Irvine, CA, Apr. 1994.

[Chase 1990] David R. Chase, Mark Wegman and F. Kenneth Zadek, Analysis of pointers and structures, Proceedings of the SIGPLAN '90 Conference on Program Language Design and Implementation, White Plains, NY, pp. 296-310, Jun. 1990.

[Chen 1997a] Jiun-Liang Chen, Feng-Jian Wang and Yung-Lin. Chen, An object-oriented dependency graph for program slicing, Technology of Object-Oriented Languages and Systems Tools, Beijing, China, 24, Sep. 1997.

[Chen 1997b] Jiun-Liang Chen, Feng-Jian Wang and Yung-Lin Chen, Slicing object-oriented programs, 4th Asia-Pacific Software Engineering and International Computer Science Conference (APSEC '97 / ICSC '97), Clear Water Bay, HONG KONG, Dec. 1997.

[Chen 2000] Guilin Chen, Bo Huang, Binyu Zang, and Chuanqi Zhu, Transformation of programs to remove nonlocal control flow, The Fourth International Conference on High-Performance Computing in the Asia-Pacific Region, Beijing, China, 1, pp.14-17, May 2000.

[Chen 1993] T. Y. Chen and Y. Y. Cheung, Dynamic program dicing, Proceedings of IEEE Conference on Software Maintenance (ICSM 1993), pp. 378-385, 1993.

[Chen 1997] T. Y. Chen and Y. Y. Cheung, On program dicing, Software Maintenance: Research and Practice, 9, pp. 33-46, 1997.

[Chen 1999a] Zhenqiang Chen, Baowen Xu, Dependence Analysis and Static Slices of Concurrent Programs, In 5th International Conference for Young Computer Scientist (ICYCS'99), 1999.

[Chen 1999b] Zhenqiang Chen and Baowen Xu, Dependency analysis based dynamic slicing for debugging, International Software Engineering Symposium 2001 (ISES'01), Wuhan, China, pp. 398-404, 2001. Engineering and Technology (ASSET '99), IEEE CS Press, pp. 230-237, 1999.

[Chen 2000] Zhenqiang Chen, Baowen Xu, Hongji Yang, Kecheng Liu and Jianping Zhang, An approach to analyzing dependency of concurrent programs, Proceedings of APAQS'2000, IEEE CS Press, Hongkong, pp. 34-39, 2000.

[Chen 2001a] Zhenqiang Chen and Baowen Xu, Slicing object-oriented Java programs, ACM SIGPLAN Notices, 36, pp. 33-40, Apr. 2001.

[Chen 2001b] Zhenqiang Chen, Baowen Xu and Hongji Yang, Slicing Tagged Objects in Ada, Ada-Europe 2001, pp. 100-112, 2001.

[Chen 2001c] Zhenqiang Chen, Baowen Xu, Slicing Concurrent Java Programs, ACM SIGPLAN Notices, 36(4), pp. 41-47, 2001.

[Chen 2001d] Zhenqiang Chen and Baowen Xu, Dependency analysis based dynamic slicing for debugging, International Software Engineering Symposium 2001 (ISES'01), Wuhan, China, pp. 398-404, 2001.

[Chen 2002a] Zhenqiang Chen, Baowen Xu, An overview of methods for dependence analysis of concurrent programs, ACM SIGPLAN Notices, 37(8), pp. 45-52, 2002.

[Chen 2002b] Zhenqiang Chen, Baowen Xu, Hongji Yang, Jianjun Zhao, Static dependency analysis for concurrent Ada 95 programs, In 7th International Conference on Reliable Software Technology (Ada-Europe'2002), LNCS 2361, pp. 219-230, 2002.

[Chen 2002c] Zhenqiang Chen, Yuming Zhou, Baowen Xu, Jianjun Zhao, and Hongji Yang, A novel approach to measuring class cohesion based on dependence analysis, Proceedings of IEEE International Conference on Software Maintenance (ICSM'02), Montreal, Canada, pp. 377-384, Oct. 2002.

[Chen 2002d] Zhenqiang Chen, Baowen Xu, Hongji Yang and Jianjun Zhao, Concurrent Ada dead statements detection, Information and Software Technology, 44(13), pp. 733-741, Oct. 2002.

[Chen 2003a] Zhenqiang Chen, Baowen Xu, An approach to measurement of class cohesion based on dependence analysis, Journal of Software, 14(11), pp. 1849-1856, 2003.

[Chen 2003b] Zhenqiang Chen, Baowen Xu and Guan Jie, Test coverage analysis based on program slicing, Journal of Electronics, 20(3), pp. 232-236, 2003.

[Chen 2003c] Zhenqiang Chen, Baowen Xu, Hongji Yang, Test coverage analysis based on program slicing, Proceedings of The 2003 IEEE International Conference on Information Reuse and Integration (IRI'2003), October 27-29, 2003, Las Vegas, USA.

[Chen 2003d] Zhenqiang Chen, Baowen Xu, William Chu, Hongji Yang and Jianjun Zhao, Slicing larger programs partially, Proceedings of The 2003 International Conference on Software Engineering Research and Practice (SERP'03: June 23-26, 2003, Las Vegas, Nevada, USA)

[Chen 2004] Zhenqiang Chen, Baowen Xu and Yuming Zhou, Measuring Class Cohesion Based on Dependence Analysis, Journal of Computer Science and Technology, 2004,19(5): 574-581.

[Cheng 2000] Ben-Chung Cheng and Wen-Mei W. Hwu, Modular interprocedural pointer analysis using access paths: design, implementation, and evaluation, Proceedings of the ACM SIGPLAN 2000 conference on Programming language design and implementation, Aug. 2000.

[Cheng 1991] Jingde Cheng, Process dependence net: A concurrent program representation, Proceedings of JSSST 8th Conference, pp. 513-516, 1991.

[Cheng 1993] Jingde Cheng, Slicing concurrent programs--A graph-theoretical approach, Proceedings of the First International Workshop on Automated and Algorithmic Debugging (1993), P. Fritzson, Ed., vol. 749 of Lecture Notes in Computer Science, Springer-Verlag, pp. 223-240.

[Cheng 1997a] Jingde Cheng, Task dependence nets for concurrent systems with Ada 95 and its applications, ACM TRI-Ada International Conference, pp. 67-78, 1997.

[Cheng 1997b] Jingde Cheng, Dependence analysis of parallel and distributed programs and its applications, Proceedings of International Conference on Advances in Parallel and Distributed Computing, p. 370, 1997.

[Cheng 2001] Jingde Cheng, Dynamic slicing of concurrent programs: Where are we and where is the right way?, Proceedings of 2001 International Conference on Parallel and Distributed Processing Techniques and Applications, Las Vegas, USA, pp. 1905-1911, Jun. 2001.

[Choi 1991] Jong-Deok Choi, Barton P. Miller and Robert H.B. Netzer, Techniques for debugging parallel programs with flowback analysis, ACM Transactions on Programming Languages and Systems, 13(4), pp. 491-530, Oct. 1991.

[Choi 1993] Jong-Deok Choi, Michael Burke and Paul Carini, Efficient flow-sensitive interprocedural computation of pointer-induced aliases and side-effects, In Conference Record of the Twentieth Annual ACM Symposium on Principles of Programming Languages, ACM Press, pp. 232-245, Jan. 1993.

[Choi 1994] Jong-Deok Choi and Jeanne Ferrante, Static slicing in the presence of goto statements, ACM Transactions on Programming Languages and Systems, 16(4), pp. 1097-1113, July 1994.

[Choi 2002] Jong-Deok Choi, Keunwoo Lee, Alexey Loginov, Robert O'Callahan, Vivek Sarkar and Manu Sridharan, Efficient and precise datarace detection for multithreaded object-oriented programs, Proceedings of ACM SIGPLAN 2002 Conference on Programming Language Design and Implementation (PLDI'02), Jun. 2002.

[Chung 2001] I. S. Chung, W. K. Lee and G. S. Yoon, Y. R. Kwon, Program slicing based on specification, Proceedings of the 2001 ACM symposium on Applied computing, pp. 605-609, 2001.

[Cifuentes 1993] Cristina Cifuentes, A structuring algorithm for decompilation, XIX Conference Latinoamericana de Informatica, Buenos Aires, Argentina, 2(6), pp. 267-276, Aug. 1993.

[Cimitile 1995a] Aniello Cimitile, Andrea De Lucia and Malcolm Munro, Identifying reusable functions using specification driven program slicing: a case study, Proceedings of International Conference on Software Maintenance, Opio (Nice), France, IEEE CS Press, pp. 124-133, 1995.

[Cimitile 1995b] Aniello Cimitile, Andrea De Lucia, and Malcolm Munro, A specification driven slicing process for identifying reusable functions, Technical Report no. 3/95, Dep. of Computer Science, University of Durham, U.K., 1995.

[Clarke 1999] Edmund M. Clarke and Masahiro Fujita and Sreeranga P. Rajan and Thomas W. Reps and Subash Shankar and Tim Teitelbaum, Program slicing of hardware description languages, Technical Report CMU-CS99 -103, Carnegie Mellon University, 1999.

[Comuzzi 1996] Joseph J. Comuzzi and Johnson M. Hart, Program slicing using weakest preconditions, FME '96: Industrial Benefit and Advances in Formal Methods, Oxford, England, 1051, pp. 557-575, March 1996.

[Coutant 1986] Deborah S. Coutant, Retargetable high-level alias analysis, Proceedings of the Thirteenth Annual ACM Symposium on Principles of Programming Languages, ACM Press, pp. 110-118, 1986.

[Cutillo 1993] F. Cutillo, Piernicola Fiore and Giuseppe Visaggio, Identification and extraction of "domain independent" components in large programs, Proceedings of the 1st Proceedings of the Working Conference on Reverse Engineering, pp. 83-92, 1993.

[Cytron 1990] Ron Cytron, Jeanne Ferrante, and V. Sarker, Compact representations for control dependence, Proceedings of the ACM SIGPLAN'90 Conference on Programming Language Design and Implementation, SIGPLAN Notices 25(6), pp. 337-351, 1990.

[Cytron 1991] Ron Cytron, Jeanne Ferrante, Barry K. Rosen, Mark N. Wegman, and F. Kenneth Zadeck, Efficiently computing static single assignment form and the control dependence graph, A CM Transactions on Programming Languages and Systems, pp. 451-490, Oct. 1991.

[Danicic 1995] Sebastian Danicic, Mark Harman, and Yogasundary Sivagurunathan, A parallel algorithm for static program slicing. Information Processing Letters, 56(6), pp. 307−313, December 1995.

[Danicic 1999] Sebastian Danicic, Dataflow Minimal Slicing, PhD thesis, University of North London, UK, School of Informatics, Apr. 1999.

[Danicic 2000] Sebastian Danicic and Mark Harman, Espresso: A slicer generator, In ACM Symposium on Applied Computing, (SAC'00), pp. 831-839, 2000.

[Daninic 2004a] Sebastian Danicic, Dave Daoudi, Chris Fox, Mark Harman, Rob Hierons, John Howroyd, Lahcen Ouarbya and Martin Ward, ConSUS: A light-weight program conditioner, Journal of Systems and Software, 2004.

[Daninic 2004b] Sebastian Daninic, Andrea De Lucia and Mark Harman, Building executable union slices using conditioned slicing, Proceedings of the 12th IEEE International Workshop on Program Comprehension (IWPC'04), p. 89, 2004.

[Danicic] Sebastian Danicic, Chris Fox, Mark Harman, Robert Mark Hierons, John Howroyd, and Mike Laurence, Slicing algorithms are minimal for programs which can be expressed as linear, free, liberal schemas, ACM Transactions on Programming Languages and Systems, In revision.

[Dantzig 1973] G. Dantzig and B. C. Eaves, Fourier-motzkin elimination and its dual, Journal of Combinatorial Theory, A(14), pp. 288-297, 1973.

[Daoudi 2002] Dave Daoudi, Lahcen Ouarbya, John Howroyd, Sebastian Danicic, Mark Harman, Chris Fox, Martin P. Ward, ConSUS: A Scalable Approach to Conditioned Slicing, Proceedings of the Ninth Working Conference on Reverse Engineering (WCRE'02), Richmond, Virginia, p. 109, 2002.

[Das 2000] Manuvir Das, Unification-based pointer analysis with directional assignments, Proceedings of the SIGPLAN Conference on Programming Language Design and Implementation, pp. 35-46, Jun. 2000.

[DeMillo 1996] Richard A. DeMillo, Hsin Pan and Eugene H. Spafford, Critical slicing for software fault localization, ACM SIGSOFT Software Engineering Notes, 21(3), pp. 121-134, May 1996.

[Denning 1977] Dorothy E. Denning and Peter J. Denning, Certification of programs for secure information flow, Communications of the Association for Computing Machinery, 20(7), pp. 504-513, July 1977.

[Deng 2001] Yunbo Deng, Suraj Kothari and Yogy Namara, Program slice browser, Proceedings of Ninth International Workshop on Program Comprehension (IWPC'01), pp. 50-59, 2001.

[De Lucia 1996] Andrea De Lucia, Anna Rita Fasolino and Malcolm Munro, Understanding function behaviors through program slicing, Proceedings of 4th IEEE Workshop on Program Comprehension, Berlin, Germany, pp. 9-18, Mar. 1996.

[De Lucia 2001] Andrea De Lucia, Program slicing: Methods and applications, In IEEE workshop on Source Code Analysis and Manipulation (SCAM 2001), 2001.

[De Lucia 2003] Andrea De Lucia, Mark Harman, Rob Hierons and Jens Krinke, Unions of slices are not slices. Proceedings of European Conference on Software Maintenance and Reengineering (CSMR '03), Benevento, Italy, Mar. 2003.

[Deutsch 1994] Alain Deutsch, Interprocedural may-alias analysis for pointers: beyond k-limiting, Proceedings of the ACM SIGPLAN'94 Conference on Programming Language Design and Implementation, pp. 230-241, Jun. 1994.

[Dhamdhere 2000] Dhananjay M. Dhamdhere, Effective execution histories for debugging and dynamic slicing, Technical Report, CSE Department, IIT Bombay, 2000.

[Dhamdhere 2003] Dhananjay M. Dhamdhere, K. Gururaja and Prajakta G. Ganu, A compact execution history for dynamic slicing, Information Processing Letters, 85(3), pp. 145-152, Feb. 2003.

[Diwan 1998] Amer Diwan, Kathryn S. McKinley, and J. Eliot B. Moss, Type-based alias analysis, Proceedings of ACM SIGPLAN Conference on Programming Language Design and Implementation, Montreal, Canada, 1998.

[Duesterwald 1992a] Evelyn Duesterwald, Rajiv Gupta and Mary Lou Soffa, Rigorous data flow testing through output influences, Proceedings of the Second Irvine Software Symposium ISS'92 California, pp. 131-145, 1992.

[Duesterwald 1992b] Evelyn Duesterwald, Rajiv Gupta, Mary Lou Soffa, Distributed slicing and partial re-execution for distributed programs, Proceedings of 5th Workshop on Languages and Compilers for Parallel Computing, pp. 497-511, 1992.

[Duesterwald 1993] Evelyn Duesterwald, Rajiv Gupta and Mary Lou Soffa, Demand driven program analysis, Technical Report TR-93-15, Department of Computer Science, University of Pittsburgh, Pittsburgh, PA, Oct. 1993.

[Duesterwald 1995] Evelyn Duesterwald, Rajiv Gupta, and Mary Lou Soffa, Demand driven computation of interprocedural data flow, in Conference Record of the Twenty-Second A CM Symposium on Principles of Programming Languages, San Francisco, CA, Jan. 1995.

[Duesterwald 1997] Evelyn Duesterwald, Rajiv Gupta, and Mary Lou Soffa, A practical framework for demand-driven interprocedural data flow analysis, ACM Transaction on Programming Language and System, 19(6), pp. 992-1030, Nov. 1997.

[Dwyer 1995] Matthew B. Dwyer, Lori A. Clarke and Kari A. Nies, A compact Petri net representation for concurrent programs, ICSE'95, IEEE CS Press, pp. 147-157, 1995.

[Dwyer 1999a] Matthew B. Dwyer, James C. Corbett, John Hatcliff, Stefan Sokolowski and Hongjun Zheng, Slicing multi-threaded java programs: A case study, Technical Report KSU CIS TR 99-7, Department of Computing and Information Sciences, Kansas State University, 1999.

[Dwyer 1999b] Matthew Dwyer and John Hatcliff, Slicing software for model construction, Proceedings of the 1999 ACM Workshop on Partial Evaluation and Program Manipulation (PEPM'99), pp. 105-118, Jan. 1999.

[Emami 1994] Maryam Emami, Rakesh Ghiya and Laurie J. Hendren, Context-sensitive interprocedural point-to analysis in the presence of function pointers, Proceedings of the ACM SIGPLAN 1994 conference on Programming language design and implementation, pp. 242-256, Jun. 1994.

[Ernst 1994] Michael D. Ernst, Practical fine-grained static slicing of optimized code, Technical Report MSR-TR-94-14, Microsoft Research, Redmond, WA, July 1994.

[Ernst 1995] Michael D. Ernst, Slicing pointers and procedures, Microsoft Research technical report, Redmond, WA, MSR-TR-95-23, Jan. 1995.

[Erosa 1994] Ana M. Erosa and Laurie J. Hendren, Taming control flow: a structured approach to eliminating goto statements, International Conference on Computer Languages, pp. 229-240, May 1994.

[Ezick 2001] James Ezick, Gianfranco Bilardi and Keshav Pingali, Efficient computation of interprocedural control dependence, Cornell University, Computer Science/Arts & Eng. Dept. Upson Hall Ithaca, NY, Sept. 2001

[Faragó 2001] Csaba Faragó and Tamás Gergely, Handling the unstructured statements in the forward dynamic slice algorithm. Proceedings of the 7th Symposium on Programming Languages and Software Tools (SPLST 2001), pp. 16-27, Jun. 2001.

[Faragó 2002]Csaba Faragó, Tamás Gergely, Handling pointers and unstructured statements in the forward computed dynamic slice algorithm. Acta Cybernetica, 15(4), pp. 489-508, 2002.

[Ferrante 1987] Jeanne Ferrante, Karl J. Ottenstein and Joe D. Warren, The program dependence graph and its use in optimization, ACM Transactions on Programming Languages and Systems, 9(3), pp. 319-349, 1987.

[Filed 1995] John Field, G. Ramalingam and Frank Tip, Parametric program slicing, Proceedings of the ACM Symposium on Principles of Programming Languages, New York, pp. 379-392, 1995.

[Field 1998] John Field and Frank Tip, Dynamic dependence in term of rewriting systems and its application to program slicing, Information and Software Technology, 40(11-12), pp. 609-636, 1998.

[Flater 1993] David W. Flater, Yelina Yesha, and E. K. Park, Extensions to the C programming language for enhanced fault detection, Software -- Practice and Experience, 23(6), pp. 617-628, June 1993.

[Foster 2000] Jeffrey S. Foster, Manuel Fähndrich and Alexander Aiken, Polymorphic versus monomorphic flow-insensitive point-to analysis for C, In Seventh International Static Analysis Symposium, Jun. 2000.

[Forgacsy 1997a] Istvan Forgacsy and Tibor Gyimóthy, An efficient interprocedural slicing method for large programs, Proceedings of SEKE'97, pp. 279-287, Madrid, Spain, 1997.

[Forgacsy 1997b] Istvan Forgacsy and Antonia Bertolino, Feasible test path selection by principal slicing, Proceedings of the Sixth European Software Engineering Conference (ESEC/FSE 97), LNCS 1301, Springer-Verlag, Sep. 1997.

[Fox 2001] Chris Fox, Mark Harman, Rob Hierons and Sebastian Danicic, Backward conditioning: a new program specialization technique and its application to program comprehension, In 9th IEEE International Workshop on Program Comprehension (IWPC'01), pp. 89-97, May 2001.

[Fox 2003] Chris Fox, Sebastian Danicic, Mark Harman and Robert Mark Hierons, ConSIT: a fully automated conditioned program slicer, Software--Practice and Experience, 34, pp. 15-46, 2004.

[Francel 1999] Margaret Ann Francel and Spencer Rugaber, The relationship of slicing and debugging to program understanding, Proceeding of 7th International Workshop on Program Comprehension, p. 106, May 1999.

[Francel 2001] Margaret Ann Francel and Spencer Rugaber, The value of slicing while debugging, Proceedings of the 7th International Workshop on Program Comprehension, pp. 151-169, 2001.

[Fritzson 1992] Peter Fritzson, Nahid Shahmehri, Mariam Kamkar and Tibor Gyimothy, Generalized algorithmic debugging and testing, ACM Letters on Programming Languages and Systems, 1(4), pp. 303-322, 1992.

[Gallagher 1990] Keith Brian Gallagher, Using program slicing for program maintenance, PhD thesis, University of Maryland, College Park, Maryland, 1990.

[Gallagher 1991a] Keith Brian Gallagher, Using program slicing to eliminate the need for regression testing, Proceedings of 8th International Conference on Testing Computer Software, May 1991.

[Gallagher 1991b] Keith Brian Gallagher and James R. Lyle, Using program slicing in software maintenance, IEEE Transactions on Software Engineering, 17(8), pp. 751-761, 1991.

[Gallagher 1993] Keith Brian Gallagher and J. R. Lyle, Program slicing and software safety, Proceedings of the Eighth Annual Conference on Computer Assurance, pp. 71-80, Jun. 1993.

[Gallagher 1996] Keith Brian Gallagher, Visual impact analysis, Proceedings of the Conference on Software Maintenance, p. 52, 1996.

[Gallagher 1997] Keith Brian Gallagher and Liam O'Brien, Reducing visualization complexity using decomposition slices, Proceedings of Software Visualization Workshop, pp. 113-118, 1997.

[Gallagher 2001] Keith Brian Gallagher and Liam O'Brien, Analyzing programs via decomposition slicing: initial data and observation, Proceeding of 7th Workshop on Empirical Studies of Software Maintenance, Florence, Italy, Nov. 2001.

[Gallagher 2003] Keith Brian Gallagher and David Binkley, An empirical study of computation equivalence as determined by decomposition slice equivalence, Proceedings of the IEEE Tenth Working conference on Reverse Engineering, pp. 13-16, Nov. 2003.

[Gandle 1993] M. Gandle, A. Santal, and G. Venkatesh, Slicing functional programs using collecting abstract interpretation, First Symposium on Algorithmic and Automated Debugging, Linkoping, Sweeden, 1993.

[Garg 2001] Vijay K. Garg and Neeraj Mittal, On slicing a distributed computation, Proceedings of the 21st IEEE International Conference on Distributed Computing Systems (ICDCS), Phoenix, Arizona, USA, pp. 322-329, Apr. 2001.

[Gerber 1997] Richard Gerber and Seongsoo Hong, Slicing real-time programs for enhanced schedulability, ACM Transactions on Programming Languages and Systems, 13(3), pp. 525-555, 1997.

[Ghiya 1992] Rakesh Ghiya, Interprocedural analysis in the presence of function pointers, ACAPS Technical Memo 62, School of Computer Science, McGill University, Dec. 1992.

[Ghiya 1996a] Rakesh Ghiya and Laurie J. Hendren, Connection analysis: a practical interprocedural heap analysis for C, International Journal of Parallel Programming, 24(6), pp. 547-578, 1996.

[Ghiya 1996b] Rakesh Ghiya and Laurie J. Hendren, Is it a tree, a DAG, or a cyclic graph? A shape analysis for heap-directed pointers in C, Proceedings of the 23rd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, pp. 1-15, Jan. 1996.

[Ghiya 1998] Rakesh Ghiya and Laurie J. Hendren, Putting pointer analysis to work, Proceedings of 25th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, Jan. 1998.

[Ghiya 2001] Rakesh Ghiya, Daniel Lavery and David Sehr, On the importance of points-to analysis and other memory disambiguation methods for C programs, Proceedings of the ACM SIGPLAN'01 conference on Programming language design and implementation, ACM Press, pp. 47-58, 2001.

[Gopal 1991] Rajiv Gopal, Dynamic program slicing based on dependence relations, Proceedings of Conference on Software Maintenance, Sorrento, Italy, IEEE CS Press, pp. 191-200, 1991.

[Goswami 2000a] Diganta Goswami and Rajib Mall, Dynamic slicing of concurrent programs, Proceedings of the 7th International Conference on High Performance Computing, pp. 15-26, Dec. 2000.

[Goswami 2000b] Diganta Goswami, Rajib Mall, Prosenjit Chatterjee, Static slicing in Unix process environment, Software--Practice and Experience, 30(1), pp. 17-36, 2000.

[Goswami 2002] Diganta Goswami and Rajib Mall, An efficient method for computing dynamic program slices, Information Processing Letters, 81(2), pp. 111-117, Jan. 2002.

[Gouranton 1998] Valérie Gouranton and Daniel Le Métayer, Dynamic slicing: A generic analysis based on a natural semantics format, Technical Report 3375, INRIA, Domaine de Voluceau, Rocquencourt, BP 105, 78153 Le chesnay Cedex FRANCE, Mar. 1998.

[Gouranton 1999] Valérie Gouranton and Daniel Le Métayer, Dynamic slicing: A generic analysis based on a natural semantics format, Journal of Logic and Computation, 9(6), pp. 835-871, 1999.

[Grove 2001] David Grove, Craig Chambers, A framework for call graph construction algorithms, ACM Transactions on Programming Languages and Systems (TOPLAS), 23(6), pp. 685-746, Nov. 2001.

[Gupta 1992a] Rajiv Gupta and Mary Lou Soffa, A framework for generalized slicing, Technical Report TR-92-07, University of Pittsburgh, 1992.

[Gupta 1992b] Rajiv Gupta, Mary Jean Harrold, and Mary Lou Soffa, An approach to regression testing using slicing, Proceedings of the Conference on Software Maintenance, Orlando, FL, U.S.A., IEEE CS Press, pp. 299-308, 1992.

[Gupta 1995] Rajiv Gupta and Mary Lou Soffa, Hybrid slicing: An approach for refining static slices using dynamic information, ACM SIGSOFT Third Symposium on the Foundations of Software Engineering, pp. 29-40, Oct. 1995.

[Gupta 1997a] Rajiv Gupta, Mary Lou Soffa and John Howard, Hybrid slicing: integrating dynamic information with static analysis, ACM Transactions on Software Engineering and Methodology (TOSEM), 6(4), pp. 370-397, Oct. 1997.

[Gupta 1997b] Bindu S. Gupta, A critique of cohesion measures in the object-oriented paradigm, Master's thesis, Department of Computer Science, Michigan Technological University, 1997.

[Gyimóthy 1998] Tibor Gyimóthy and Jukka Paakki, Static slicing of logic programs, Proceedings of 2nd International Workshop on Automated and Algorithmic Debugging, IRISA, France, Mar. 1998.

[Gyimóthy 1999] Tibor Gyimóthy, Árpád Beszédes and István Forgács, An efficient relevant slicing method for debugging, Proceedings of 7th European Software Engineering Conference (ESEC), Toulouse, France, LNCS 1687, pp. 303-321, Sept. 1999.

[Hall 1992] Mary W. Hall, Ken Kennedy, Efficient call graph analysis, ACM Letters on Programming Languages and Systems (LOPLAS), 1(3), pp. 227-242, Sep. 1992.

[Hall 1995] Robert J. Hall, Automatic extraction of executable program subsets by simultaneous dynamic program slicing, Automated Software Engineering, 2(1), pp. 33-53, Mar. 1995.

[Harman 1994a] Mark Harman and Sebastian Danicic, A new approach to program slicing, 7th International Software Quality Week, San Francisco, May 1994.

[Harman 1994b] Mark Harman and Sebastian Danicic, A framework for defining equivalence preserving program simplification and its application to program slicing, Technical Report, University of North London, Project Project, Mar. 1994.

[Harman 1995a] Mark Harman, Sebastian Danicic and Yoga Sivagurunathan, Program comprehension assisted by slicing and transformation, 1st UK workshop on program comprehension, University of Durham, July 1995.

[Harman 1995b] Mark Harman and Sebastian Danicic, A simultaneous slicing theory and derived program slicer, 4th RIMS Workshop in Computing, Kyoto University, Kyoto, Japan, July 1996.

[Harman 1995c] Mark Harman, Sebastian Danicic, Barry Jones, Balasubramaniam Sivagurunathan, and Yogasundary Sivagurunathan, Pseudo variable in slicing criteria-- capturing implicit computation, 8th International Quality Week, San Francisco, May 29th - June 2nd. 1995.

[Harman 1995d] Mark Harman, Sebastian Danicic, Yoga Sivagurunathan, Bala Sivagurunathan and Barry Jones, Cohesion metrics, 8th International Software Quality Week, San Francisco CA, paper 4-T-4, May 1995.

[Harman 1995e] Mark Harman and Sebastian Danicic, Slicing programs in the presence of errors, Technical Report, University of North London, Project Project, Feb. 1995.

[Harman 1995f] Mark Harman and Sebastian Danicic, Using program slicing to simplify testing, Journal of Software Testing, Verification and Reliability, 5(3), pp. 143-162, 1995.

[Harman 1996a] Mark Harman, Sebastian Danicic, Yogasundary Sivagurunathan, and Dan Simpson, The next 700 slicing criteria, Malcolm Munro, editor, 2nd UK workshop on program comprehension, Durham University, UK, July 1996.

[Harman 1996b] Mark Harman, Dan Simpson and Sebastian Danicic, Slicing programs in the presence of errors, Formal Aspects of Computing, 8(4), 1996.

[Harman 1997a] Mark Harman, Cleaving together - program cohesion with slices, EXE, 11(8), pp. 35-42, Jan. 1997.

[Harman 1997b] Mark Harman, Margaret Okunlawon, Bala Sivagurunathan and Sebastian Danicic, Slice-based measurement of coupling, In Rachel Harrison, editor, 19th ICSE, Workshop on Process Modeling and Empirical Studies of Software Evolution, Boston, Massachusetts, USA, May 1997.

[Harman 1997c] Mark Harman and Sebastian Danicic, Amorphous program slicing, Proceedings of IEEE International Workshop on Program Comprehension (IWPC'97), Dearborn, Michigan, pp. 70-79, May 1997.

[Harman 1998a] Mark Harman and Keith Brian Gallagher, Program slicing, Information and Software Technology, 40(11-12), pp. 577-581, 1998.

[Harman 1998b] Mark Harman, Yoga Sivagurunathan and Sebastian Danicic, Analysis of dynamic memory access using amorphous slicing, Proceedings of IEEE International Conference on Software Maintenance, Washington, DC, pp. 336-345, Nov. 1998.

[Harman 1998c] Mark Harman and Sebastian Danicic, A new algorithm for slicing unstructured programs, Journal of Software Maintenance: Research and Practice, 10(6), Nov. 1998.

[Harman 1998d] Mark Harman and Keith Brian Gallagher, editors. Special Issue on Program Slicing, Elsevier, 40, Nov. 1998.

[Harman 2000a] Chris Fox, Mark Harman, Sebastian Danicic and Robert Hierons, ConSIT: A conditioned program slicer, IEEE International Conference on Software Maintenance (ICSM'00) (San Jose, California, USA), IEEE Computer Society Press, Los Alamitos, California, USA, pp. 216-226, Oct. 2000.

[Harman 2000b] Mark Harman, Robert M. Hierons and Sebastian Danicic, The relationship between program dependence and mutation analysis, Mutation 2000 Workshop, San Jose, California, USA, pp. 15-23, Oct. 2000.

[Harman 2001a] Mark Harman and Rob Hierons, An overview of program slicing, Software Focus, 2(3), pp. 85-92, 2001.

[Harman 2001b] Mark Harman, Rob Hierons, Chris Fox, Sebastian Danicic and John Howroyd, Pre/Post conditioned slicing, Proceeding of IEEE International Conference on Software Maintenance (ICSM'01), Florence, Italy, pp. 138-147, Nov. 2001.

[Harman 2001c] Mark Harman, Robert M. Hierons, Sebastian Danicic, John Howroyd, Mike Laurence, and Chris Fox, Node coarsening calculi for program slicing, Working Conference on Reverse Engineering, 2001.

[Harman 2002a] Mark Harman, Nicolas Gold, Rob Hierons and David Binkley, Code extraction algorithms which unify slicing and concept assignment, Proceedings of 9th IEEE Working Conference on Reverse Engineering (WCRE'02), pp. 11-21, 2002.

[Harman 2002b] Mark Harman, Lin Hu, Malcolm Munro, Xingyuan Zhang, Sebastian Danicic, Mohammed Daoudi and Lahcen Ouarbya, An interprocedural amorphous slicer for WSL, Proceedings of the Second IEEE International Workshop on Source Code Analysis and Manipulation (SCAM'02), pp. 105-114, Oct. 2002.

[Harman 2002c] Mark Harman, Lin Hu, Rob Hierons, Chris Fox, Sebastian Danicic, Andre Baresel, Harmen Sthamer and Joachim Wegener, Evolutionary testing supported by slicing and transformation, Proceedings of 18th IEEE International Conference on Software Maintenance (ICSM 2002), p. 285, 2002.

[Harman 2004a] Mark Harman, Lin Hu, Malcolm Munro, Xingyuan Zhang, David Binkley, Sebastian Danicic, Lahcen Ouarbya and Dave (Mohammed) Daoudi, Syntax-Directed amorphous slicing, Journal of Automated Software Engineering, 11(1), pp. 27-61, 2004.

[Harman 2004b] Mark Harman, Lin Hu, Rob Hierons, Joachim Wegener, Harmen Sthamer, Baresel Baresel, Marc Roper, Testability transformation, IEEE Transaction on Software Engineering, 30(1), pp. 3-16, Jan. 2004.

[Harrold 1991] Mary Jean Harrold and Mary Lou Soffa, Selecting dataflow integration testing, IEEE Software, 8(2), pp. 58-65, 1991.

[Harrold 1993] Mary Jean Harrold, Brian Malloy and Gregg Rothermel, Efficient construction of program dependence graphs, International Symposium on Software Testing and Analysis (ISSTA '93), 1993.

[Harrold 1998a] Mary Jean Harrold, Gregg Rothermel, and Saurabh Sinha, Computation of interprocedural control dependence, Proceedings of the ACM International Symposium on Software Testing and Analysis, Clearwater, FL, pp. 11-21, Mar. 1998.

[Harrold 1998b] Mary Jean Harrold and Ning Ci, Reuse-driven interprocedural slicing, Proceedings of the 20th International Conference on Software Engineering (ICSE'98), Kyoto, Japan, pp. 74-83, April 1998.

[Hasti 1998] Rebecca Hasti and Susan Horwitz, Using static single assignment form to improve flow-insensitive pointer analysis, Proceedings of the ACM SIGPLAN'98 Conference on Programming Language Design and Implementation, pp. 97-105, 17-19, Jun., SIGPLAN Notices 33(5), May 1998.

[Hatcliff 1999] John Hatcliff, James C. Corbett, Matthew B. Dwyer, Stefan Sokolowski, and Hongjun Zheng, A formal study of slicing for multi-threaded programs with JVM concurrency primitives, In Static Analysis Symposium, pp. 1-18, 1999.

[Hatcliff 2000] John Hatcliff, Matthew B. Dwyer, and Hongjun Zheng, Slicing software for model construction, Higher-Order and Symbolic Computation, 13(4), pp. 315-353, 2000.

[Hausler 1989] Prove Hausler, Denotational program slicing, Proceedings of the 22nd Hawaii International Conference on System Sciences, Hawaii, pp. 486-494, 1989.

[Hecht 1977] Matthew S. Hecht, Flow analysis of computer programs, Elsevier Science Inc, New York, USA, 1977.

[Heimdahl 1997] Mats P. E. Heimdahl and Michael W. Whalen, Reduction and slicing of hierarchical state machines, Proceedings of the 6th European Software Engineering Conference (ESEC/FSE 97), pp. 450-467, 1997.

[Heintze 2001a] Nevin Heintze and Olivier Tardieu, Demand-driven pointer analysis, Proceedings of the ACM SIGPLAN 2001 Conference on Programming Language Design and Implementation, pp. 20-22, Jun. 2001.

[Heintze 2001b] Nevin Heintze and Olivier Tardieu, Ultra-fast aliasing analysis using CLA: a million lines of C code in a second, Proceedings of the SIGPLAN Conference on Programming Language Design and Implementation, pp. 254-263, Jun. 2001.

[Henrard 1998] Jean Henrard, Vincent Englebert, Jean-Marc Hick, Didier Roland, Jean-Luc Hainaut, Program understanding in databases reverse engineering, Proceedings of the 9th International Conference on Databases and Expert Systems Applications (DEXA'98), Vienna, Springer-Verlag, pp. 70-79, Jun. 1998.

[Hendren 1992] Laurie J. Hendren, Joseph Hummell and Alexandru Nicolau, Abstractions for recursive pointer data structures: improving the analysis and transformation of imperative programs, Proceedings of the ACM SIGPLAN 1992 conference on Programming language design and implementation, July 1992.

[Hierons 1999] Robert M. Hierons, Mark Harman and Sebastian Danicic, Using program slicing to assist in the detection of equivalent mutants, Journal of Software Testing, Verification and Reliability, 9(4), pp. 233-262, Dec. 1999.

[Hierons 2000] Robert M. Hierons and Mark Harman, Program analysis and test hypotheses complement, IEEE ICSE International Workshop on Automated Program Analysis, Testing and Verification, Limerick, Ireland, Jun. 2000.

[Hierons 2002] Robert Mark Hierons, Mark Harman, Chris Fox, Lahcen Ouarbya and Mohammed Daoudi, Conditioned slicing supports partition testing, Software Testing, Verification and Reliability, 12(1), pp. 23-28, Mar. 2002.

[Hind 1999] Michael Hind, Michael Burke, Paul Carini, and Jong-Deok Choi, Interprocedural pointer alias analysis, ACM Transaction on Programming Languages and Systems, 21(4), pp. 848-894, May 1999.

[Hind 2000] Michael Hind and Anthony Pioli, Which pointer analysis should I use?, Proceedings of the ACM SIGSOFT International Symposium on Software Testing and Analysis (ISSTA), 2000.

[Hind 2001a] Michael Hind, Pointer analysis: haven't we solved this problem yet?, In Program Analysis for Software Tools and Engineering (PASTE'01), ACM, Snowbird, Utah, USA, Jun. 2001.

[Hind 2001b] Michael Hind and Anthony Pioli, Evaluating the effectiveness of pointer alias analyses, Science of Computer Programming, 39(1), pp. 31-55, Jan. 2001.

[Hisley 2002] Dixie Hisley, Matt Bridges, and Lori Pollock, Static Interprocedural slicing of shared memory parallel programs, International Conference on Parallel and Distributed Processing Techniques and Applications, (PDPTA'02), pp. 658-664, Jun. 2002.

[Hoffner 1995a] Tommy Hoffner, Mariam Kamkar and Peter Fritzson, Evaluation of program slicing tools, In 2nd International Workshop on Automated and Algorithmic Debugging, pp. 51-69, 1995.

[Hoffner 1995b] Tommy Hoffner, Evaluation and Comparison of Program Slicing Tool, Technical Report, LiTH-IDA-R-95-01, Department of Computer and Information Science, Linkping University, Sweden, 1995.

[Horwitz 1988a] Susan Horwitz, Jan Prins, and Thomas Reps, On the adequacy of program dependence graphs for representing programs, Conference Record of the Fifteenth Annual ACM Symposium on Principles of Programming Language, ACM, pp. 146-157, 1988.

[Horwitz 1988b] Susan Horwitz, Tomas Reps, and David Binkley, Interprocedural slicing using dependence graphs, Proceedings of the SIGPLAN 88 Conference on Programming Language Design and Implementation, (Atlanta, GA), ACM SIGPLAN Notices 23(7), pp. 35-46, July 1988.

[Horwitz 1988c] Susan Horwitz, Jan Prins, and Thomas Reps, Support for integrating program variants in an environment for programming in the large, Proceedings of the International Workshop on Software Version and Conjuration Control 88, Grassau, Germany, Jan. 1988.

[Horwitz 1989a] Susan Horwitz, Jan Prins, and Thomas Reps, Integrating non-interfering versions of programs, ACM Transactions on Programming Languages and Systems, 11(3), pp. 345-387, July 1989.

[Horwitz 1989b] Susan Horwitz, Identifying the semantic and textual differences between two versions of a program, Proceedings of the ACM SIGPLAN 90 Conference on Programming Language Design and Implementation, (White Plains, NY, Jun. 1990), ACM SIGPLAN Notices 25(6) pp. 234-245, Jun. 1989.

[Horwitz 1989c] Susan Horwitz, Phil Pfeiffer and Thomas Reps, Dependence analysis for pointer variables, Proceedings of the SIGPLAN Conference on Programming Language Design and Implementation, pp. 28-40, Jun. 1989.

[Horwitz 1990] Susan Horwitz, Thomas Reps, and David Binkley, Interprocedural slicing using dependency graphs, ACM Transaction on Programming Languages and Systems, 22(1), pp. 26-60, Jan. 1990.

[Horwitz 1991] Susan Horwitz and Thomas Reps, Efficient comparison of program slices, Acta Informatica 28(9), pp. 713-732, 1991.

[Horwitz 1995] Susan Horwitz, Tomas Reps and Mooly Sagiv, Demand interprocedural dataflow analysis, Proceeding of the 3rd ACM SIGSOFT Symposium on the Foundations of Software Engineering, pp. 104-115, Oct. 1995.

[Hwang 1988a] J. C. Hwang, M. W. Du and C. R. Chou, Finding program slices for recursive procedures, Proceedings of the 12th Annual International Computer Software and Applications Conference (Chicago, 1988).

[Hwang 1988b] J. C. Hwang, M. W. Du and C. R. Chou, The influence of language semantics on program slices, Proceedings of the 1988 International Conference on Computer Languages, Miami Beach, 1988.

[Hu 2004] Lin Hu, Mark Harman, Robert M. Hieron and David Binkley, Loop Squashing Transformations for Amorphous Slicing, Proceedings of the IEEE Eleventh Working conference on Reverse Engineering, Nov. 2004.

[Huang 1996] H. Huang, Wei-Tek Tsai and Satish Subramanian, Generalized Program Slicing for Software Maintenance, Proceedings of Software Engineering and Knowledge Engineering, pp. 261-268, 1996.

[Huynh 1997] D.T. Huynh, Y. Song, Forward dynamic slicing in the presence of structured jump statements, Proceedings of ISACC'97, pp. 73-81, 1997.

[Ishio 2003] Takashi Ishio, Shinji Kusumoto and Katsuro Inoue, Application of aspect-oriented programming to calculation of program slice, Technical Report, ICSE2003,

[Iwaihara 1996] Mizuho Iwaihara, Masaya Nomura, Shigeru Ichinose and Hiroto Yasuura, Program slicing on vhdl descriptions and its applications, Proceedings of Asian Pacific Conference on Hardware Description Languages (APCHDL), pp. 132-139, 1996.

[Jackson 1994a] Daniel Jackson and Eugene Rollins, A new model of program dependence for reverse engineering, David Wile, editor, Proceedings of the Second ACM SIGSOFT Symposium no Foundations of Software Engineering, volume 19 of ACM SIGSOFT Software Engineering Notes, pp. 2-10, Dec. 1994.

[Jackson 1994b] Daniel Jackson and Eugene J. Rollins, Chopping: a generalization of slicing, Technical Report, Carnegie Mellon University, School of Computer Science, Number CS-94-169, p. 21, July 1994.

[Jackson 1994c] Daniel Jackson and Eugene J. Rollins, Abstraction mechanisms for pictorial slicing, Proceedings of the IEEE Workshop on Program Comprehension, pp. 82-88, 1994.

[Jiang 1991] J. Jiang, X. Zhou, and D.J. Robson, Program slicing for C – the problems in implementation, Proceedings of the Conference on Software Maintenance, pp. 182-190, 1991.

[Joiner 1993] J. K. Joiner and W. T. Tsai, Ripple effect analysis, program slicing, and dependence analysis, Technical Report, TR number: TR 93-84, Department of Computer Science University of Minnesota Minneapolis.

[Jones 1981] N. D. Jones and S. S. Muchnick, Flow analysis and optimization of LISP-like structures, In S. S. Muchnick and N. D. Jones, editors, Program Flow Analysis, chapter 4, pp. 102-131, Prentice-Hall, 1981.

[Kamkar 1991] Mariam Kamkar, An overview and comparative classification of static and dynamic program slicing, Technical Report LiTH-IDA-R-91-19, Linkoping University, Linkoping, 1991, to appear in Journal of System and Software.

[Kamkar 1992] Mariam Kamkar, Nahid Shahmehri and Peter Fritzson, Interprocedural dynamic slicing, Proceedings of the 4th International Symposium on Programming Language Implementation and Logic Programming, LNCS 631, Springer-Verlag, Aug. 1992.

[Kamkar 1993a] Mariam Kamkar, Interprocedural dynamic slicing with applications to debugging and testing, PhD thesis, Linkoping University, Sweden, 1993.

[Kamkar 1993b] Mariam Kamkar, Peter Fritzson and Nahid Shahmehri, Three approaches to interprocedural dynamic slicing, Microprocessing and Microprogramming, 38, pp. 625–636, 1993.

[Kamkar 1993c] Mariam Kamkar, Peter Fritzson and Nahid Shahmehri, Interprocedural dynamic slicing applied to interprocedural data flow testing, Proceedings of the Conference on Software Maintenance, Montreal, Canada, pp. 386-395, 1993.

[Kamkar 1995a] Mariam Kamkar and Patrik Krajina, Dynamic slicing of distributed programs, Proceedings of the International Conference on Software Maintenance, pp. 222-231, Oct. 1995.

[Kamkar 1995b] Mariam Kamkar. An overview and comparative classification of program slicing techniques, Journal Systems Software, 31, pp. 197- 214, 1995.

[Karmkar 1996] Mariam Karmkar, Patrik Krajina and Peter Frizson, Dynamic slicing of parallel message passing programs, 4th Euromicro Workshop on Parallel and Distributed Processing (PDP '96), PORTUGAL, pp. 170-179, Jan. 1996.

[Kamkar 1998] Mariam Kamkar, Application of program slicing in algorithmic debugging, Information and Software Technology, 40(11-12), pp. 635-646, Nov. 1998.

[Kang 1996] Byung-Kyoo Kang and James M. Bieman, Design-level cohesion measures: Derivation, comparison, and applications, Colorado State University, Technical Report CS-96-104, Jan. 1996.

[Kang 1998] Byung-Kyoo Kang and James M. Bieman, Using design abstractions to visualize, quantify, and restructure software, Journal of Systems and Software, 2(2), pp. 175-187, 1998.

[Karstu 1994] Sakari Karstu, An examination of the behavior of the slice based cohesion measures, Master's thesis, Department of Computer Science, Michigan Technological University, 1994.

[Kennedy 1981] Ken Kennedy, A survey of data flow analysis techniques, Steven S. Muchnick and Neil D. Jones, editors, Program Flow Analysis: Theory and Applications. Prentice-Hall, Englewood Cliffs, New Jersey, 1981.

[Kim 1994] Hyeon-Soo Kim and Yong-Rae Kwon, Restructuring programs through program slicing, International Journal of Software Engineering and Knowledge Engineering, 4(3), pp. 349-368, 1994.

[Kim 1999a] Taeho H. Kim, Yeong-Tae Song, Lawrence Chung and Dung T. Huynh, Software architecture analysis using dynamic slicing, Proceedings of AoM/IAoM CS'99, 17(2), pp. 242-247, Aug. 1999.

[Kim 1999b] Taeho H. Kim, Yeong-Tae Song, Lawrence Chung and Dung T. Huynh, Dynamic software architecture slicing, Proceedings of the 23rd IEEE Annual International Computer Software and Applications Conference, Oct. 1999.

[Kim 2000] Taeho Kim, Yeong-Tae Song, Lawrence Chung, Dung T. Huynh, Software architecture analysis: a dynamic slicing approach. ACIS International Journal of Computer & Information Science, 1(2), pp. 91-103, Mar. 2000.

[Kiss 2003] Ákos Kiss, Judit Jász, Gábor Lehotai and Tibor Gyimóthy, Interprocedural static slicing of binary executables, Proceedings of the 3rd IEEE International Workshop on Source Code Analysis and Manipulation (SCAM 2003), Amsterdam, The Netherlands, pp. 118-127, Sep. 2003.

[Knuth 1974] Donald Knuth, Structured programming with goto statements, Computing Surveys, pp. 261-302, Dec. 1974.

[Komondoor 2001] Raghavan Komondoor and Susan Horwitz, Using slicing to identify duplication in source code, Lecture Notes in Computer Science, 2126, p. 40, 2001.

[Korel 1988a] Bogdan Korel and Janusz Laski, STAD - a system for testing and debugging: user perspective, Proceedings of the Second Workshop on Software Testing, Verification and Analysis, pp. 13-20, July 1988.

[Korel 1988b] Bogdan Korel and Janusz Laski, Dynamic program slicing, Information Processing Letters, 29(3), pp. 155-163, 1988.

[Korel 1990] Bogdan Korel and Janusz Laski, Dynamic slicing of computer programs, The Journal of Systems and Software, 13(3), pp. 187-195, 1990.

[Korel 1992] Bogdan Korel and Roger Ferguson, Dynamic slicing of distributed programs, Applied Mathematics and Computer Science Journal, 2(2), pp. 199-215, 1992.

[Korel 1994] Bogdan Korel and Satish Yalamanchili, Forward computation of dynamic Program slices, Proceedings of the 1994 International Symposium on Software Testing and Analysis, pp 66-79, 1994.

[Korel 1995] Bogdan Korel, Computation of dynamic slices for programs with arbitrary control-flow, Second International Workshop on Automated and Algorithmic Debugging, St. Malo, France, 1995.

[Korel 1997a] Bogdan Korel, Computation of dynamic slices for unstructured programs, IEEE Transactions on Software Engineering, 23(1), pp. 17-34, 1997.

[Korel 1997b] Bogdan Korel, Application of dynamic slicing in program debugging, Third International Workshop on Automated Debugging, pp. 59-74, 1997.

[Korel 1997c] Bogdan Korel and Juergen Rilling, Dynamic program slicing in understanding of program execution, Proceedings of 5th International Workshop on Program Comprehension (WPC '97), pp 80-90, 1997.

[Korel 1998a] Bogdan Korel and Jurgen Rilling, Dynamic program slicing methods, Information and Software Technology, 40(11-12), pp. 647-659, 1998.

[Korel 1998b] Bogdan Korel and Juergen Rilling, Program slicing in understanding of large program, Proceedings of 6th IEEE International Workshop on Program Comprehension (IWPC'98), pp. 145-152, 1998.

[Kovács 1996] Gyula Kovács, Ferenc Magyar, and Tibor Gyimóthy, Static slicing of java programs, Technical Report TR-96-108, József Attila University, Hungary, 1996.

[Krinke 1998a] Jens Krinke and Gregor Snelting, Program slicing, Information and Software Technology, 40(11-12), pp. 661-676, Nov. 1998.

[Krinke 1998b] Jens Krinke, Static slicing of threaded programs, ACM SIGPLAN Notices, 33(7), pp. 35-42, 1998.

[Krinke 1998c] Jens Krinke and Gregor Snelting, Validation of measurement software as an application of slicing and constraint solving, Information and Software Technology, 40(11-12), pp. 661-675, Dec. 1998.

[Krinke 2002] Jens Krinke, Evaluating context-sensitive slicing and chopping, Proceedings of the International Conference on Software Maintenance (ICSM'02), pp. 22-31, Oct. 2002.

[Krinke 2003a] Jens Krinke, Context sensitive slicing of concurrent programs, ACM SIGSOFT Software Engineering Notes, 2003.

[Krinke 2003b] Jens Krinke, Barrier slicing and chopping, Proceedings of 3rd IEEE International Workshop on Source Code Analysis and Manipulation (SCAM 2003), Amsterdam, Netherlands, pp. 81-87, Sep. 2003.

[Krinke 2004a] Jens Krinke, Slicing, Chopping and path conditions with barriers, Software Quality Journal 12(4), pp. 339-360, December 2004.

[Krinke 2004b] Jens Krinke. Context-sensitivity matters, but context does not, Fourth IEEE International Workshop on Source Code Analysis and Manipulation (SCAM 2004), Chicago, Sep. 2004.

[Krinke 2004c] Jens Krinke, Visualization of program dependence and slices, Proceedings of International Conference on Software Maintenance (ICSM 2004), Chicago, Sep. 2004.

[Krishnaswamy 1994] Anand Krishnaswamy, Program slicing: an application of object-oriented program dependency graphs, Technical Report, TR94-108, Department of Computer Science, Clemson University, 1994.

[Kuck 1981] David J. Kuck, R. H. Kuhn, David A. Padua, B. Leasure, and Michael Wolfe, Dependence graphs and compiler optimizations, Conference Record of the Eighth ACM Symposium on Principles of Programming Languages, pp. 207-218, 1981.

[Kumar 2001] Sumit Kumar and Susan Horwitz, Better slicing of programs with jumps and switches. Technical Report TR-1429, Computer Sciences, University of Wisconsin-Madison, 2001.

[Kumar 2002] Sumit Kumar and Susan Horwitz, Better slicing of programs with jumps and switches, University of Wisconsin and GrammaTech, Inc. FASE, pp. 96-112, 2002.

[Kuhn 1995] Bradley M. Kuhn, Dennis J. Smith and Keith B. Gallagher, The decomposition slice display system, Proceedings of the 1995 Conference on Software Engineering and Knowledge Engineering SEKE'95, Jun. 1995.

[Kusumoto 2002] Shinji Kusumoto, Akira Nishimatsu, Keisuke Nishie and Katsuro Inoue, Experimental evaluation of program slicing for fault location, Empirical Software Engineering, 7, pp. 49-76, 2002.

[Lakhotia 1991] Arun Lakhotia, Graph theoretic foundations of program slicing and integration, Report CACS TR-91-5-5, University of Southwestern Louisiana, 1991.

[Lakhotia 1992] Arun Lakhotia, Improved interprocedural slicing algorithm. Technical Report CACS TR-92-5-8, The Center for Advanced Computer Studies, University of Southwestern Louisian, Lafayette, LA 70504, 1992.

[Lakhotia 1993a] Arun Lakhotia, Constructing call multigraphs using dependence graphs, Conference Record of the Twentieth ACM Symposium on Principles of Programming Languages (Charleston, SC, 1993), pp. 273-284, 1993.

[Lakhotia 1993b] Arun Lakhotia, Rule-based approach to computing module cohesion, Proceedings of the 15th Conference on Software Engineering (ICSE-15), pp. 34-44, 1993.

[Lakhotia 1998] Arun Lakhotia and Jean-Christophe Deprez, Restructuring programs by tucking statements into functions, Information and Software Technology, 40(11/12), pp. 677-691, 1998.

[Lakhotia 1999] Arun Lakhotia, Jean-Christophe Deprez and Shreyash S. Kame, Flow analysis models for interprocedural program slicing algorithms, Technical Report TR-99-5-1, University of Southwestern Louisiana, July 1999.

[Lanubile 1997] Filippo Lanubile and Giuseppe Visaggio, Extracting reusable functions by flow graph-based program slicing, IEEE Transactions on Software Engineering, 23(4), pp. 246-259, 1997.

[Landi 1990] William Landi and Barbara G. Ryder, Pointer-induced aliasing: a problem classification, Proceedings of the Seventeenth Annual ACM Symposium on Principles of Programming Languages, pp. 93-103, Jan. 1990.

[Landi 1992a] William Landi, Undecidability of static analysis, ACM Letters on Programming Languages and Systems, 1(4), pp. 323-337, Dec. 1992.

[Landi 1992b] William Landi, Interprocedural aliasing in the presence of pointers, PhD thesis, Rutgers University, 1991.

[Landi 1992c] William Landi and Barbara G. Ryder, A safe approximate algorithm for interprocedural aliasing, ACM SIGPLAN Notices, July 1992.

[Landi 1993] William Landi, Barbara G. Ryder, and Sean Zhang, Interprocedural modification side effect analysis with pointer aliasing, Proceedings of the SIGPLAN '93 Conference on Programming Language Design and Implementation, pp. 56-67, Jun. 1993.

[Lange 2001] Carola Lange, Program slicing and slicing book technology, ARTI 8800, Spring 2001.

[Larsen 1996] Loren Larsen and Mary Jean Harrold, Slicing Object-Oriented software, International Conference on Software Engineering Proceedings of the 18th international conference on Software engineering, Berlin, Germany, pp. 495-505, 1996.

[Larus 1993] James R. Larus and Satish Chandra, Using tracing and dynamic slicing to tune compilers, Technical Report CS-TR-93-1174, University of Wisconsin-Madison, Aug. 1993.

[Laski 1989] Janusz Laski, Data flow testing in stad, The Journal of Systems and Software, 1989.

[Laski 1991a] Janusz Laski, A data flow based algorithm for the derivation of program slices, Technical Report TR-CSE-10-1, Oakland University, 1991.

[Laski 1991b] Janusz Laski and Wojciech Szermer, Reachability slicing in software reliability studies, Technical Report TR-CSE-91-12-1, Oakland University, 1991.

[Law 1994] R. C. H. Law, Object-Oriented program slicing, PhD Thesis, University of Regina, Regina, Canada, 1994.

[Lee 2001] Wan Kwon Lee, In Sang Chung, Gwang Sik Yoon and Yong Rae Kwon, Specification-based program slicing and its applications, Journal of Systems Architecture: the EUROMICRO Journal, 47(5), pp. 427-443, 2001.

[Leminen 1994] Janne A. Leminen, Slicing and slice based measures for the assessment of functional cohesion of z operation schemas, Master's thesis, Department of computer Science, Michigan Technological University, 1994.

[Lengauer 1979] Thomas Lengauer and Robert Endre Tarjan, A fast algorithm for finding dominators in a flowgraph, In ACM Transactions on Programming Languages and Systems, 1(1), July 1979.

[Leung 1987] Hareton K.N. Leung and Hassan K. Reghbati, Comments on program slicing, IEEE Transactions on Software Engineering SE-13, 12, pp. 1370-1371, 1987.

[Li 1990] Z. Li, P. Yew and C. Zhu, An efficient data dependence analysis for parallelizing compilers, IEEE Transactions on Parallel and Distributed Systems, 1(1), pp. 26-34, Jan. 1990.

[Li 2000] Bixin Li, Program slicing techniques and its application in object-oriented software metrics and software test, PhD thesis, Nanjing University, P. R. China. Dec. 2000.

[Li 2001a] Bixin Li, A hierarchical slice-based framework for object-oriented coupling measurement, Turku Centre for Computer Science, TUCS Technical Reports, No.415, Turku, Finland, July 2001.

[Li 2001b] Bixin Li and Xiaocong Fan, JATO: Slicing Java programs hierarchically, TUCS Technical Report, No 416, Turku, Finland, 2001.

[Li 2002] Bixin Li, Analyzing information-flow in Java program based on program slicing technique, Software Engineering Notes, 27(5), pp. 98-103, 2002.

[Li 2003a] Bixin Li, Managing dependencies in component-based systems based on matrix model, Proceedings Of Net.Object.Days 2003, pp. 22-25, Sept. 2003.

[Li 2003b] Bixin Li, A technique to analyze information-flow in object-oriented programs, Information and Software Technology, Elsevier science, 45(6), pp. 305-314, 2003.

[Li 2004a] Bixin Li, SSA: A core algorithm to slice programs, to appear in International Journal of Information and Computational Science (JOICS), Binary Information Press, USA, 2004.

[Li 2004b] Bixin Li, Xiaocong Fan, Jun Pang and Jianjun Zhao, A model for slicing Java programs hierarchically, Journal of Computer Science & Technology, 19(6), 2004.

[Li 2004c] Bixin Li and Ying Zhou, Measuring and validating a kind of object-oriented software coupling, to appear in Proceedings of Info2004, Tokyo, Japan.

[Li 2004d] Hon F. Li, Juergen Rilling and Dhrubajyoti Goswami, Granularity-driven dynamic predicate slicing algorithms for message passing systems, Automated Software Engineering Journal, Kluwer Academic Publishers, 11(1), pp. 63-89, Jan. 2004.

[Liang 1998] Donglin Liang and Mary Jean Harrold, Slicing objects using system dependence graphs, Proceedings of the International Conference on Software Maintenance (ICSM'98), Bethesda, MD, pp. 358-367, Nov. 1998.

[Liang 1999a] Donglin Liang and Mary Jean Harrold, Reuse-Driven interprocedural slicing in the presence of pointers and recursion. Proceedings of the International Conference on Software Maintenance (ICSM'99), pp. 421-432, Aug. 1999.

[Liang 1999b] Donglin Liang and Mary Jean Harrold, Equivalence analysis: A general technique to improve the efficiency of data-flow analyses in the presence of pointers, Proceedings of the ACM SIGPLAN-SIGSOFT workshop on Program Analysis for Software Tools and Engineering '99, Sept. 1999.

[Liang 1999c] Donglin Liang and Mary Jean Harrold, Efficient points-to analysis for whole-program analysis, Proceedings of the 7th European engineering conference held jointly with the 7th ACM SIGSOFT international symposium on Foundations of software engineering, Oct. 1999.

[Liang 2000] Donglin Liang and Mary Jean Harrold, Towards efficient and accurate program analysis using light-weight context recovery, Proceedings of 22nd International Conference on Software Engineering (ICSE'00), pp. 366-375, Jun. 2000.

[Liang 2001a] Donglin Liang and Mary Jean Harrold, Efficient computation of parameterized pointer information for interprocedural analyses, Proceedings of 7th International Static Analysis Symposium, July 2001.

[Liang 2001b] Donglin Liang, Maikel Pennings and Mary Jean Harrold, Extending and evaluating flow-insensitive and context-insensitive points-to analyses for Java, Proceedings of the 2001 ACM SIGPLAN-SIGSOFT workshop on Program analysis for software tools and engineering, Jun. 2001.

[Liang 2002] Donglin Liang and Mary Jean Harrold, Equivalence Analysis and Its Application in Improving the Efficiency of Program Slicing, ACM Transactions on Software Engineering and Methodology, 11(3), pp. 347-383, July 2002.

[Livadas 1992a] Panos E. Livadas and Stephen Croll, Program slicing, Technical Report SERC-TR-61-F, Computer Science and Information Services Department, University of Florida, Gainesville, FL, Oct. 1992.

[Livadas 1992b] Panos E. Livadas and Prabal K. Roy, Program dependence analysis, Proceedings of the International Conference on Software Maintenance, IEEE Computer Society Press, pp. 356-365, 1992.

[Livadas 1993a] Panos E. Livadas and Scott D. Alden, A toolset for program understanding, Proceedings of the IEEE Second Workshop on Program Comprehension, 1993.

[Livadas 1993b] Panos E. Livadas and Stephen Croll, System dependence graph construction for recursive programs, Proceedings of the Seventeenth International Computer Software and Applications Conference, pp. 414-420, Nov. 1993.

[Livadas 1994a] Panos E. Livadas and Stephen Croll, A new algorithm for the calculation of transitive dependences, Journal of Software Maintenance, 6, pp. 100-127, 1994.

[Livadas 1994b] Panos E. Livadas and Stephen Croll. System dependence graphs based on parse trees and their use in software maintenance, Journal of Information Sciences, 76(3-4), pp. 197-232, Feb. 1994.

[Livadas 1994c] Panos E. Livadas and Adam Rosenstein, Slicing in the presence of pointer variables, Technical Report SERC-TR-74-F, Computer Science and Information Services Department, University of Florida, Gainesville, FL, June 1994.

[Livadas 1995] Panos E. Livadas and Theodore Johnson, An optimal algorithm for the construction of the system dependence graph, Technical report, Computer and Information Sciences Department, University of Florida, 1995.

[Livadas 1999] Panos E. Livadas and Theodore Johnson, An optimal algorithm for the construction of the system dependence graph, Information Sciences, 125(1-4), pp. 99-131, 2000.

[Longworth 1985] Herbert D. Longworth, Slice-based program metrics, Master's thesis, Michigan Technological University, 1985.

[Longworth 1986] Herbert D. Longworth, Linda M. Ott, and M. R. Smith, The relationship between program complexity and slice complexity during debugging tasks, COMPSAC 86, 1986.

[Lu 1988] Lu Qi, Zhang Fubo, and Qian Jiahua, Program slicing, Journal of Computer Science and Technology, 3(1), pp. 29-39, 1988.

[Lyle 1984] James R. Lyle, Evaluating variations of program slicing for debugging, PhD thesis, University of Maryland, College Park, Maryland, Dec. 1984.

[Lyle 1986] James R. Lyle and Mark Weiser, Experiments on slicing-based debugging tools, Empirical studies of programming, Elliot Soloway (editor), Ablex Publishing, Norwood, New Jersey, Jun. 1986.

[Lyle 1987] James R. Lyle and Mark Weiser, Automatic program bug location by program slicing, Proceedings of 2nd International Conference on Computers and Applications, Peking, China, pp. 877-882, 1987.

[Lyle 1993] James R. Lyle and David Binkley, Program slicing in the presence of pointers, Proceedings of the Third Annual Software Engineering Research Forum, Orlando, FL, Nov. 1993.

[Lyle 1995] James R. Lyle, Dolores R. Wallace, James R. Graham, Keith B. Gallagher, Joseph P. Poole, and David W. Binkley, A CASE tool to evaluate functional diversity in high integrity software, IR-5691. U.S. Department of Commerce, Technology Administration, National Institute of Standards and Technology, Computer Systems Laboratory, Gaithersburg, MD, 1995.

[Malloy 1994] Brian A. Malloy, John D. McGregor, Anand Krishnaswamy and Murali Medikonda, An extensible program representation for Object-Oriented software, ACM SIGPLAN Notices, 29(12), pp. 38-47, Dec. 1994.

[Marlowe 1993] Thomas Marlowe, William Landi, Barbara Ryder, Jong-Deok Choi, Michael Burke and Paul Carini, Pointer-induced aliasing: a clarification, In SIGPLAN Notices, 28(9), pp. 67-70, Sep. 1993.

[Maydan 1991] Dror E. Maydan, John L. Hennessy and Monica S. Lam, Efficient and exact data dependence analysis, Proceedings of the ACM SIGPLAN'91 Conference on Programming Language Design and Implementation, SIGPLAN Notices 26(6), pp. 1-14, 1991.

[Meyers 2004a] Timothy M. Meyers, David Binkley, A Longitudinal and Comparative Study of Slice-Based Metrics, Proceedings of the 10th International Software Metrics Symposium, Chicago, USA, pp. 14-16, Sep. 2004.

[Meyers 2004b] Timothy M. Meyers, David Binkley, Slice-Based Cohesion Metrics and Software Intervention, Proceedings of the IEEE Eleventh Working conference on Reverse Engineering, Delft University, Netherlands, Nov. 2004.

[Milanova 2004] Ana Milanova, Atanas Rountev and Barbara G. Ryder, Precise call graphs for C programs with function pointers, Automated Software Engineering, 11(1), pp. 7-26, Jan. 2004.

[Miller 1988] Barton P. Miller, Jong-Deok Choi, A mechanism for efficient debugging of parallel programs, Proceedings of the ACM SIGPLAN 1988 conference on Programming Language design and Implementation, pp. 135-144, 1988, ACM SIGPLAN Notices 23(7).

[Millett 1998] Lynette I. Millett and Tim Teitelbaum, Slicing promela and its applications to model checking, Proceedings of the 4th International SPIN Workshop, 1998

[Millett 2000] Lynette I. Millett and Tim Teitelbaum, Issues in slicing promela and its applications to model checking, protocol understanding, and simulation, International Journal on Software Tools for Technology Transfer, 2(4), pp. 343-349, 2000.

[Mittal 2001] Neeraj Mittal and Vijay K. Garg, Computation slicing: techniques and theory, Proceedings of the 15th International Symposium on Distributed Computing (DISC), Lisbon, Portugal, pp. 78-92, Oct. 2001.

[Mittal 2003] Neeraj Mittal, Vijay K. Garg, Software fault tolerance of distributed programs using computation slicing, Proceedings of the 23rd International Conference on Distributed Computing Systems, p. 105, May 2003.

[Mock 2001] Markus Mock, Manuvir Das, Craig Chambers and Susan J. Eggers, Dynamic points-to sets: a comparison with static analyses and potential applications in program understanding and optimization, Proceedings of the 2001 ACM SIGPLAN-SIGSOFT Workshop on Program Analysis for Software Tools and Engineering, Snowbird, UT, USA, pp. 66-72, Jun. 2001.

[Mock 2002] Markus Mock, Darren C. Atkinson, Craig Chambers and Susan J. Eggers, Improving program slicing with dynamic points-to data, ACM SIGSOFT Software Engineering Notes, 27(6), pp. 71-80, Nov. 2002.

[Moretti 2001] Eric Moretti, Gills Chanteperdrix, and Angel Osorio, New algorithms for control-flow graph structuring, Fifth European Conference on Software Maintenance and Reengineering, Lisbon, Portugal, pp. 184, Mar. 2001.

[Müller-Olm 2001] Markus Müller-Olm and Helmut Seidl, On optimal slicing of parallel programs, STOC 2001 (33rd ACM Symposium on Theory of Computing), pp. 647-656, 2001.

[Müller-Olm 2004] Markus Müller-Olm, Precise interprocedural dependence analysis of parallel programs, Theoretical Computer Science, 311(1-3), pp. 325-388, Jan. 2004.

[Mund 2002] G. B. Mund, Rajib Mall and Sudeshna Sarkar, An efficient dynamic program slicing technique, Information and Software Technology, 44(2), pp. 123-132, Feb. 2002.

[Mund 2003] G. B. Mund, Rajib Mall and Sudeshna Sarkar, Computation of intraprocedural dynamic program slices, Information and Software Technology, 45(8), pp. 499-512, Jun. 2003.

[Nanda 2000] Mangala Gowri Nanda and S. Ramesh, Slicing concurrent programs, Software Engineering Notes, 25(5), pp. 180-190, 2000.

[Naumovich 1998] Gleb Naumovich and George S. Avrunin, A conservative dataflow algorithm for detecting all pairs of statements that may happen in parallel, Proceedings of the 6th International Symposium on the Foundations of Software Engineering, pp. 24-34, 1998.

[Naumovich 1999] Gleb Naumovich, George S. Avrunin and Lori A. Clarke, An efficient algorithm for computing MHP information for concurrent Java programs, ESEC/FSE '99, pp. 338-354, 1999.

[Netzer 1994] Robert H. B. Netzer and Mark H. Weaver, Optimal tracing and incremental re-execution for debugging long-running programs, Proceedings of the ACM SIGPLAN Conference on Programming Language Design and Implementation, New York, pp. 313-325, Jun. 1994.

[Nishimatsu 1999] Akira Nishimatsu, Minoru Jihira, Shinji Kusumoto, and Katsuro Inoue, Call-mark slicing: an efficient and economical way of reducing slice, International Conference of Software Engineering, pp. 422-431, 1999.

[Ochoa 2004] Claudio Ochoa, Josep Silva and Germán Vidal, Dynamic slicing based on redex trails, PEPM 2004, pp. 123-134, 2004.

[Oda 1993] Tomohiro Oda and Keijiro Araki, Specification slicing in formal methods of software development, Proceedings of the 17th Annual International Computer Software and Applications Conference, pp. 313-319, Nov. 1993.

[Ohata 2001] Fumiaki Ohata, Kouya Hirose, Masato Fujii and Katsuro Inoue, A slicing method for object-oriented programs using lightweight dynamic information, Eighth Asia-Pacific Software Engineering Conference (APSEC'01), Macao, China, Dec. 2001.

[Orso 2001a] Alessandro Orso, Saurabh Sinha and Mary Jean Harrold, Effects of pointers on data dependences, Proceedings of 9th International Workshop on Program Comprehension (IWPC'01), pp. 39-49, May 2001.

[Orso 2001b] Alessandro Orso, Saurabh Sinha and Mary Jean Harrold, Incremental slicing based on data-dependences types. Proceedings of IEEE International Conference on Software Maintenance (ICSM'01), Florence, Italy, pp. 158-167, Nov. 2001.

[Orso 2003] Alessandro Orso, Saurabh Sinha and Mary Jean Harrold, Understanding data dependences in the presence of pointers, Technical Report GIT-CERCS-03-10, College of Computing, Georgia Institute of Technology, Atlanta, Ga., May 2003.

[Orso 2004] Alessandro Orso, Saurabh Sinha and Mary Jean Harrold, Classifying data dependences in the presence of pointers for program comprehension, testing, and debugging, ACM Transactions on Software Engineering and Methodology, 13(2), pp. 199-239, 2004.

[Ottenstein 1984] Karl J. Ottenstein and Linda M. Ottenstein, The program dependence graph in a software development environment, Proceedings of the ACM SIGSOFT/SIGPLAN Software Engineering Symposium on Practical Software Development Environments, pp. 177-184, Apr. 1984.

[Ott 1989] Linda M. Ott and Jeffrey J. Thuss, The relationship between slices and module cohesion, Proceedings of the 11th ACM conference on Software Engineering, pp. 198-204, May 1989.

[Ott 1992a] Linda M. Ott and James M. Bieman, Effects of software changes on module cohesion, Proceedings of the Conference on Software Maintenance-1992, pp. 345-353, Nov. 1992.

[Ott 1992b] Linda M. Ott, Using slice profiles and metrics during software maintenance, Proceedings of the 10th Annual Software Reliability Symposium, pp. 16-23, 1992.

[Ott 1993] Linda M. Ott and Jeffrey J. Thuss, Slice based metrics for estimating cohesion, Proceedings of the IEEE-CS International Metrics Symposium, pp. 78-81, 1993.

[Ott 1995] Linda Ott, James M. Bieman, Byung-Kyoo Kang and Bindu Mehra, Developing measures of class cohesion for object-oriented software, Proceeding of 7th Annual Oregon Workshop on Software Metrics (AOWSM'95), Jun. 1995.

[Ott 1998] Linda M. Ott and James M. Bieman, Program slices as an abstraction for cohesion measurement, Information and Software Technology, 40(11-12), pp. 691-700, Nov. 1998.

[Ouarbya 2002] Lahcen Ouarbya, Sebastian Danicic, Dave (Mohammed) Daoudi, Mark Harman and Chris Fox, A denotational interprocedural program slicer, Proceedings of IEEE Working Conference on Reverse Engineering (WCRE 2002), Richmond, Virginia, USA, pp. 181-189, 2002.

[Peterson 1973] W. W. Peterson, T. Kasami, and N. Tokura, On the capabilities of while, repeat and exit statements, Comm. of the ACM, 16(8), pp. 503-512, 1973.

[Petersen 1992] Paul M. Petersen and David A. Padua, Dynamic dependence analysis: A novel method for data dependence evaluation, Conference Record of the 5th Workshop on Languages and Compilers for Parallel Computing, Yale University, Department of Computer Science, YALEU/DCS/RR915, Aug. 1992

[Plakal 2000] Manoj Plakal and Charles N. Fischer, Concurrent garbage collection using program slices on multithreaded processors, Proceedings of the International Symposium on Memory Management (ISMM), 36(1), pp. 94-100, Oct. 2000.

[Pratt 1977] V.R. Pratt, Two easy theories whose combination is hard, Technical Report, Mass Institute of Technology, Sep. 1977.

[Pugh 1992] William Pugh and David Wonnacott, Eliminating false data dependences using the omega test, Proceedings of the ACM SIGPLAN 92 Conference on Programming Language Design and Implementation, pp. 140-151, 1992.

[Qi 2004] Xiaofang Qi and Baowen Xu, Dependence Analysis of Concurrent Programs Based on Reachability Graph and Its Applications, International Conference on Computational Science, pp. 405-408, 2004.

[Ramalingam 2000] G. Ramalingam, Context-sensitive synchronization-sensitive analysis is undecidable, ACM TOPLAS, 22(2), pp. 416-430, 2000.

[Ranganath 2004] Venkatesh Prasad Ranganath and John Hatcliff, Pruning interference and ready dependence for slicing concurrent Java programs, Proceedings of 13th International Conference on Compiler Construction (CC 2004), pp. 9-56, 2004.

[Reps 1988a] Thomas Reps and Wuu Yang, The semantics of program slicing, Technical Report 777, University of Wisconsin - Madison, Jun. 1988.

[Reps 1988b] Thomas Reps and Susan Horwitz, Semantics-based program integration, In Proceedings of the Second European Symposium on Programming (ESOP '88), Nancy, France, pp. 133-145, Mar. 1988.

[Reps 1989a] Thomas Reps and Wuu Yang, The semantics of program slicing and program integration, In Proceedings of the Colloquium on Current Issues in Programming Languages, 352 of Lecture Notes in Computer Science, pp. 360-374, Springer 1989.

[Reps1989b] Thomas Reps and Thomas Bricker, Semantics-based program integration: illustrating interference in interfering versions of programs, Proceedings of the Second International Workshop on Software Configuration Management, Princeton, New Jersey, pp. 46-55, Oct. 1989.

[Reps 1993] Tomas Reps, The Wisconsin program-integration system reference manual: Release 2.0, Technical Report Unpublished report University of Wisconsin, Madison, WI, July 1993.

[Reps 1994a] Thomas Reps. Demand interprocedural program analysis using logic databases, Kluwer Academic Publishers, pp. 163-196, 1994.

[Reps 1994b] Thomas Reps, Solving demand versions of interprocedural analysis problems, Proceedings of the 5th International Conference on Compiler Construction, Lecture Notes in Computer Science, 786, ed. P. Fritzson, Springer-Verlag, pp. 389-403, 1994.

[Reps 1994c] Tomas Reps, Susan Horwitz, Mooly Sagiv and Genevieve Rosay, Speeding up slicing, Proceedings of the 2nd ACM SIGSOFT symposium on Foundations of software engineering, pp. 11-20, Dec. 1994.

[Reps 1994d] Tomas Reps, Mooly Sagiv and Susan Horwitz, Interprocedural dataflow analysis via graph reachability, Report DIKU TR 94-14, University of Copenhagen, Copenhagen, 1994

[Reps 1995a] Tomas Reps, Susan Horwitz and Mooly Sagiv, Precise interprocedural dataflow analysis via graph reachability, Proceedings of ACM Symposium on Principles of Programming Languages, pp. 49-61, 1995.

[Reps 1995b] Thomas Reps and Genevieve Rosay, Precise interprocedural chopping, Proceedings of the 3rd ACM Symposium on the Foundations of Software Engineering (Washington, DC), Oct. 1995.

[Reps 1996a] Thomas Reps, On the sequential nature of interprocedural program-analysis problems, Acta Informatica 33(8), pp. 739-757, 1996.

[Reps 1996b] Thomas Reps and Todd Turnidge, Program specialization via program slicing, Proceedings of the Dagstuhl Seminar on Partial Evaluation, Lecture Notes in Computer Science, 1110, pp. 409-429, 1996.

[Reps1998] Thomas Reps, Program analysis via graph reachability, Information and Software Technology, 40(11-12), pp. 701-726, Nov. 1998.

[Ricca 2001] Filippo Ricca and Paolo Tonella, Web application slicing, Proceedings of the IEEE International Conference on Software Maintenance (ICSM'01), pp. 148-157, 2001.

[Ricca 2002] Filippo Ricca and Paolo Tonella, Construction of the system dependence graph for web application slicing, Proceedings of 2nd IEEE International Workshop on Source Code Analysis and Manipulation (SCAM'02), pp. 123-132, Oct. 2002.

[Richardson 1992] Debra J. Richardson, T. Owen O'Malley, Cynthia Tittle Moore and Stephanie Leif Aha, Developing and integrating PRODAG into the arcadia environment, Proceedings of the Fifth Symposium on Software Development Environments, pp. 109-119, 1992.

[Rilling 1998] Juergen Rilling, Investigation of dynamic slicing and its application in program comprehension, PhD Thesis, Illinois Institute of Technology, July 1998.

[Rilling 2001] Juergen Rilling and Bhaskar Karanth, A hybrid program slicing framework, IEEE International Workshop on Source Code Analysis and Manipulation (SCAM'01), Florence, Italy, Nov. 2001.

[Rilling 2002] Juergen Rilling, Hon F. Li, Dhrubajyoti Goswami. Predicate-based dynamic slicing of message passing programs, Proceeding of the IEEE International Conference on Software Maintenance (ICSM 2002), Montreal, Canada, pp. 133-142, Oct. 2002.

[Rilling 2003] Juergen Rilling and Tuomas Klemola, Identifying comprehension bottlenecks using program slicing and cognitive complexity metrics, Proceeding of 11th IEEE International Workshop on Program Comprehension (IWPC'03), p.115, May 2003.

[Robschink 2002] Torsten Robschink, Gregor Snelting, Efficient path conditions in dependence graphs, Proceedings of the 24th international conference on Software engineering, Orlando, Florida, pp. 478-488, 2002.

[Ross 1998] John L. Ross and Mooly Sagiv, Building a bridge between pointer aliases and program dependences, Nordic Journal of Computing, 8, pp. 361-386,1998.

[Rothermel 1994] Gregg Rothermel and Mary Jean Harrold, Selecting tests and identifying test coverage requirements for modified software, Proceedings of the ACM SIGSOFT International Symposium on Software Testing and Analysis, pages 169-184, Aug. 1994.

[Rothermel 1996] Gregg Rothermel and Mary Jean Harrold, Analyzing regression test selection techniques, IEEE Transactions on Software Engineering, 22(8), pp. 529-551, 1996.

[Rountey 2001a] Atanas Rountev, Ana Milanova, and Barbara G. Ryder, Points-to analysis for Java based on annotated constraints, Proceedings of Conference on Object-Oriented Programming Systems, Languages, and Applications, pp. 43-55, Oct. 2001.

[Rountey 2001b] Atanas Rountev and Barbara G. Ryder, Points-to and side-effect analyses for programs built with precompiled libraries, Proceedings of the International Conference on Compiler Construction, Apr. 2001.

[Ruf 1995] Eric Ruf, Context-insensitive alias analysis reconsidered, Proceedings of the ACM SIGPLAN 1995 conference on Programming language design and implementation, pp. 13-22, Jun. 1995.

[Rugina 1999] Radu Rugina and Martin C. Rinard, Pointer analysis for multithreaded programs, Proceedings of the ACM SIGPLAN'99 Conference on Programming Language Design and Implementation, pp. 77-90, May, SIGPLAN Notices, 34(5), May 1999.

[Russell 2001] Jeff Russell, Program slicing literature survey, Seminar and Draft Materials, University of Texas at Austin, 2001, http://www.ece.utexas.edu/~jrussell/seminar/slicing_survey.pdf.

[Russell 2002] Jeffry T. Russell, Co-design architecture and synthesis: Program slicing for codesign, Proceedings of the tenth international symposium on Hardware/software codesign, 2002.

[Russel 2002] Jeffry Russel, Program slicing for codesign, Proceedings of Tenth International Symposium on Hardware/Software Codesign, 2002.

[Samadzadeh 1993] Mansur H. Samadzadeh and Winai Wichaipanitch, An interactive debugging tool for C based on dynamic slicing and dicing, Proceedings of the 21st Annual ACM Conference on Computer Science, pp. 30-37, 1993.

[Schoenig 1995] Stéphane Schoenig and Mireille Ducassé, A hybrid backward slicing algorithm producing executable slices for Prolog, Proceedings of the 7th Workshop on Logic Programming Environments, pp. 41-48, Dec. 1995.

[Schoenig 1996] Stephane Schoenig and Mireille Ducasse, A backward slicing algorithm for prolog, In Static Analysis Symposium, pp. 317-331, 1996.

[Shahmehri 1991] Nahid Shahmehri, Generalized algorithmic debugging, PhD thesis, Linkoping University, 1991.

[Shapiro 1982] Ehud Y. Shapiro, Algorithmic program debugging, MIT Press, 1982.

[Shapiro 1997a] Marc Shapiro and Susan Horwitz, Fast and accurate flow-insensitive points-to analysis, Proceedings of the 24th ACM SIGPLANSIGACT Symposium on Principles of Programming Languages, pp. 1-14, Jan. 1997.

[Shapiro 1997b] Marc Shapiro and Susan Horwitz., The effects of the precision of pointer analysis, In P. V. Hentenryck editor, Lecture Notes in Computer Science, 1302, pp. 16-34, Springers-Verlag, 1997. Proceedings from the 4th International Static Analysis Symposium.

[Shatz 1987] Sol M. Shatz and W. K. Cheng, A Petri Net framework for automatic static analysis of Ada tasking behavior, Journal of Systems Software, 8, pp. 343-359, 1987.

[Shimomura 1992] Takao Shimomura, The program slicing technique and its application to testing, debugging and maintenance, Journal of IPS of Japan, 9(9), pp. 1078-1086, Sep. 1992.

[Shostak 1981] Robert Shostak, Deciding linear inequalities by computing loop residues, ACM Journal, 28(4), pp. 769-779, Oct. 1981.

[Sinha 1999] Saurabh Sinha, Mary Jean Harrold, and Gregg Rothermel, System-dependence-graph-based slicing of programs with arbitrary interprocedural control flow, Proceedings of the 21st IEEE International Conference on Software Engineering (ICSE'99), pp. 432-441, May 1999.

[Sinha 2001] Saurabh Sinha, Mary Jean Harrold and Gregg Rothermel, Interprocedural control dependence, ACM Transactions on Software Engineering and Methodology, 10(2), pp. 209-254, 2001.

[Sivagurunathan 1997] Yoga Sivagurunathan, Mark Harman, and Sebastian Danicic, Slicing, I/O and the implicit state, Mariam Kamkar, editor, 3rd International Workshop on Automated Debugging (AADEBUG'97), Sweden, May 1997.

[Sivagurunathan 2002] Yoga Sivagurunathan, Mark Harman and Bala Sivagurunathan, Slice-based dynamic memory modeling -- a case study, Proceedings of the 26th International Computer Software and Applications Conference on Prolonging Software Life: Development and Redevelopment, pp. 351-356, 2002.

[Sloane 1996] Anthony M. Sloane and Jason Holdsworth, Beyond traditional program slicing, Proceedings of the 1996 international symposium on Software testing and analysis, San Diego, California, United States, pp. 180-186, 1996.

[Snelting 1996] Gregor Snelting, Combining slicing and constraint solving for validation of measurement software, Static Analysis Symposium, Lecture Notes in Computer Science, 1145, Springer, pp. 332-348, 1996.

[Snelting 2003] Gregor Snelting, Torsten Robschink, and Jens Krinke, Efficient path conditions in dependence graphs for software safety analysis, Submitted for publication, 2003.

[Song 1998] Yeong-Tae Song and Dung T. Huynh, Forward dynamic interprocedural program slicing, Proceedings of the 1998 conference on CS & I '98, Research Triangle Park, NC, Oct. 1998.

[Song 1999a] Yeong-Tae Song and Dung T. Huynh, Forward dynamic slicing software systems, PhD thesis, 103 pages, Jan. 1999.

[Song 1999b] Yeong-Tae Song, Dung T. Huynh, Forward dynamic object-oriented program slicing, Application-Specific Systems and Software Engineering and Technology (ASSET '99), IEEE CS Press, pp. 230-237, 1999.

[Song 2001] Yeong-Tae Song and Dung T. Huynh, Dynamic slicing object-oriented programs using dynamic object relationship diagrams, ACIS International Journal of Computer & Information Science, 2(1), pp. 35-48, Mar. 2001.

[Stafford 2000] Judith Alyce Stafford, A formal, language-independent, and compositional approach to interprocedural control dependence analysis, PhD thesis, 153 pages, 2000.

[Steensgaard 1996] Bjarne Steensgaard, Points-to analysis in almost linear time, Proceedings of 23rd Annual ACM SIGACT-SIGPLAN Symposium on the Principles of Programming Languages, pp. 32-41, Jan. 1996.

[Steindl 1998a] Christoph Steindl, Program slicing (1), data structures and computation of control flow information, Technical Report 11, Institut für Praktische Informatik, University Linz, 1998.

[Steindl 1998b] Christoph Steindl, Program slicing (2), computation of data flow information. Technical Report 12, Institut für Praktische Informatik, University Linz, 1998.

[Steindl 1998c] Christoph Steindl, Intermodular slicing of object-oriented programs. Proceedings of International Conference on Compiler Construction, vol. 1383 of LNCS, Springer, pp. 264-278, 1998.

[Steindl 1999a] Christoph Steindl, Static analysis of object-oriented programs, 9th ECOOP Workshop for PhD Students in Object-Oriented Programming, Lisbon, Portugal, pp. 14-15, Jun. 1999.

[Steindl 1999b] Christoph Steindl, Program slicing for object-oriented programming languages, PhD thesis, Johannes Kepler University Linz, 1999.

[Steindl 1999c] Christoph Steindl, Benefits of a dataflow-aware programming environment, Workshop on Program Analysis for Software Tools and Engineering (PASTE'99), 1999.

[Sward 2003] Ricky E. Sward and A. T. Chamillard, AdaSlicer: an Ada program slicer, Proceedings of the ACM SIGAda Annual International Conference (SIGAda'03), San Diego, California, pp. 10-16, Dec. 2003.

[Szilagyi 2002] Gyongyi Szilagyi, Tibor Gyimothy and Jan Maluszynski, Static and dynamic slicing of constraint logic programs, Automated Software Engineering, 9(1), pp. 41-65, Jan. 2002.

[Takada 2002] Tomonori Takada, Fumiaki Ohata and Katsuro Inoue, Dependence-cache slicing: A program slicing method using light-weight dynamic information, Proceedings of the 10th International Workshop on Program Comprehension, p. 169, 2002.

[Thuss 1988] Jeffrey J. Thuss, An investigation into slice-based cohesion metrics, Master's thesis, Michigan Technological University, 1988.

[Tibor 1999] Tibor Gyimóthy, Árpád Beszédes and Istán Forgács, An efficient relevant slicing method for debugging, ACM SIGSOFT Software Engineering Notes, Nov. 1999.

[Tip 1994] Frank Tip, Generic techniques for source-level debugging and dynamic program slicing, Report CS-R9453, Centrum voor Wiskunde en Informatica (CWI), Amsterdam, 1994.

[Tip 1995a] Frank Tip, A survey of program slicing techniques, Journal of Programming Languages, 3(3), pp. 121-189, Sep. 1995.

[Tip 1995b] Frank Tip, Generation of Program Analysis Tools, PhD thesis, Centrum voor Wiskunde en Informatica, Amsterdam, 1995.

[Tip 1996] Frank Tip, Jong-Deok Choi, John Field, and G. Ramalingham, Slicing class hierarchies in c++, Proceedings of the 11th conference on Object-Oriented Programming, Systems, Languages and Applications (OOPSLA'96), San Jose, pp. 179-197, Oct. 1996.

[Tip 2000] Frank Tip and Jens Palsberg, Scalable propagation-based call graph construction algorithms, ACM SIGPLAN Notices, 35(10), pp. 281-293, Oct. 2000.

[Tonella 1997] Paolo Tonella, Giuliano Antoniol, Roberto Fiutem, and Ettore Merlo, Flow insensitive C++ pointers and polymorphism analysis and its application to slicing, In International Conference on Software Engineering, pp. 433-443, 1997.

[Tonella 2003] Paolo Tonella, Using a concept lattice of decomposition slices for program understanding and impact analysis, IEEE Transactions on Software Engineering, 29(6), pp. 495-509, Jun. 2003.

[Triolet 1985] R. Triolet, Interprocedural analysis for program restructuring with parafrase, Technical Report, CSRD Rep, No.538, University of Illinois Urbana-Champaign, Dec. 1985.

[Umemori 2003] Fumiaki Umemori, Kenji Konda, Reishi Yokomori and Katsuro Inoue, Design and implementation of bytecode-based Java slicing system, Third IEEE International Workshop on Source Code Analysis and Manipulation, Amsterdam, The Netherlands, p. 108, Sep. 2003.

[Vasconcelos 1994] Wamberto Weber Vasconcelos, A method of extracting Prolog programming techniques, Technical Paper 27, DAI, Edinburgh Univ., 1994.

[Vasconcelos 1995] Wamberto Weber Vasconcelos, Extracting, organizing, designing and reusing Prolog programming techniques, PhD thesis, DAI, Edinburgh Univ., Aug. 1995.

[Vasconcelos 1998a] Wamberto Weber Vasconcelos and M. A. T. Aragão, Slicing logic programs, Technical Report, 1998.

[Vasconcelos 1998b] Wamberto Weber Vasconcelos and Marcelo A. T. Aragão, An adaptation of dynamic slicing techniques for logic programming, Proceedings of the 14th Brazilian Symposium on Artificial Intelligence, SBIA'1998, pp. 151-160, Nov. 1998. Lecture Notes in Artificial Intelligence, 1515, Springer-Verlag.

[Vasconcelos 1999] Wamberto Weber Vasconcelos, A flexible framework for dynamic and static slicing of logic programs, Proceedings of PADL'99, Lecture Notes in Computer Science, 1551, pp. 259-274, 1999.

[Venkatesh 1991] G. A. Venkatesh, The semantic approach to program slicing, Proceedings of the ACM Transactions on Programming Languages and Systems, 13(2), pp. 181-210, 1991.

[Venkatesh 1995] G. A. Venkatesh, Experimental results from dynamic slicing of C programs, Transactions on Programming Languages and Systems, 17(2), pp. 197-216, 1995.

[Vidal 2002] Germán Vidal, Forward Slicing of Multi-paradigm Declarative Programs Based on Partial Evaluation, LOPSTR 2002, pp. 219-237, 2002.

[Ward 2002] M. P. Ward, Program slicing via FermaT transformations, Proceedings of the 26th International Computer Software and Applications Conference on Prolonging Software Life: Development and Redevelopment, pp. 357-362, Aug. 2002.

[Weihl 1980] William E. Weihl, Interprocedural data flow analysis in the presence of pointers, procedure variables, and label variables, In Symposium on Principles of Programming Languages, pp. 83-94, Association for Computing Machinery, Jan. 1980.

[Weise 1994] Daniel Weise, Roger F. Crew, Michael Ernst, Bjarne Steensgaard, Value dependence graphs: representation without taxation, Proceedings of the ACM SIGPLAN-SIGACT Twenty-first Symposium on Principles of Programming Languages, pp. 297-310, Jan. 1994.

[Weiser 1979] Mark Weiser, Program slices: formal, psychological, and practical investigations of an automatic program abstraction method, PhD thesis, University of Michigan, Ann Arbor, MI, 1979.

[Weiser 1981] Mark Weiser, Program slicing, Proceedings of 5th International Conference on Software Engineering, pp. 439-449, San Diego, CA, Mar. 1981.

[Weiser 1982] Mark Weiser, Programmers use slicing when debugging, Communications of the ACM, 25(7), pp. 446-452, July 1982.

[Weiser 1983] Mark Weiser, Reconstructing sequential behavior from parallel behavior projections, Information Processing Letters 17(3), pp. 129-135, 1983.

[Weiser 1984] Mark Weiser, Program slicing, IEEE Transactions on Software Engineering, 10(4), pp. 352-357, 1984.

[Weiser 1986] Mark Weiser and James R. Lyle, Experiments on slicing-based debugging aids, Empirical Studies of Programmers, Ablex Publishing Corporation, pp. 187-197, 1986.

[Weiser 1998] Mark Weiser, Foreword to special issue on program slicing, Information and Software Technology, 40(11-12), pp. 575-576, Nov. 1998.

[Wichaipanitch 2003] Winai Wichaipanitch, An interactive debugging tool for c++ based on dynamic slicing and dicing, PhD thesis, 165 pages, Jan. 2003.

[Williams 1977] M. Howard Williams, Generating structured flow diagrams: the nature of unstructuredness, The Computer Journal, 20(1), pp. 45-50,1977.

[Williams 1978] M. H. Williams and H. L. Ossher, Conversion of unstructured flow diagrams to structured, The Computer Journal, 21(2), pp. 161-167, 1978.

[Wilson 1995] Robert P. Wilson and Monica S. Lam, Efficient context-sensitive pointer analysis for C programs, Proceedings of the ACM SIGPLAN'95 Conference on Programming Language Design and Implementation, La Jolla, CA, pp. 1-12, Jun. 18-21, 1995.

[Wilson 1997] Robert P. Wilson, Efficient context-sensitive pointer analysis for C programs, PhD thesis, Stanford University, 1997.

[Woodward 1998] M. R. Woodward and S. P. Allen, Slicing algebraic specifications, Information and Software Technology, ISSN: 0950−5849, 40(2), pp. 105-118, May 1998.

[Wotawa 2002] Franz Wotawa, On the relationship between model-based debugging and program slicing, Artificial Intelligence, 135(1-2), pp. 125-143, Feb. 2002.

[Wu 2004a] Fangjun Wu and Tong Yi, Slicing Z specifications, ACM SIGPLAN Notices, 39(8), pp. 39-48, 2004.

[Wu 2004b] Junhua Wu, Baowen Xu and Jixiang Jiang, Slicing web application based on hyper graph, Proceedings of First International Workshop on Web Computing in Cyberworlds (WCCW2004), Nov. 2004, Tokyo, Japan.

[Xu 1993] Baowen Xu, Reverse program flow dependency analysis and applications, Chinese Journal of Computers, 16(5), pp. 385-392, 1993.

[Xu 2001a] Baowen Xu, Zhenqiang Chen and Xiaoyu Zhou, Slicing object-oriented Ada95 programs based on dependence analysis, Journal of Software, 12, pp. 208-213, 2001 (in Chinese with English abstract).

[Xu 2001b] Baowen Xu and Zhenqiang Chen, Dependence analysis for recursive Java programs, SIGPLAN Notices 36(12), pp. 70-76, 2001.

[Xu 2002] Baowen Xu, Zhenqiang Chen and Hongji Yang, Dynamic slicing object-oriented programs for debugging, Second IEEE International Workshop on Source Code Analysis and Manipulation (SCAM'02), Montreal, Canada, p. 115, Oct. 2002.

[Xu 2003a] Baowen Xu, Zhenqiang Chen and Jianjun Zhao, Measuring cohesion of packages in Ada95, Proceedings of the 2003 annual international conference on Ada, pp. 62-67, 2003.

[Xu 2003b] Lei Xu, Baowen Xu, Zhenqiang Chen, Jixiang Jiang and Huowang Chen, Regression testing for web applications based on slicing, Proceedings of the 27th Annual International Conference on Computer Software and Applications（IEEE COMPSAC2003）, November, 2003, Texas, USA.

[Yang 1997] Hong Yang and Baowen Xu, Design and implementation of a PSS/ADA program slicing system, Chinese Journal of Computer Research and Development, 34(3), pp. 217-222, 1997.

[Yong 1999] Suan Hsi Yong, Susan Horwitz and Thomas Reps, Pointer analysis for programs with structures and casting, Proceedings of the ACM SIGPLAN 1999 conference on Programming language design and implementation, May 1999.

[Yur 1999] Jyh-shiarn Yur, Barbara G. Ryder and William A. Landi, An incremental flow- and context-sensitive pointer aliasing analysis, Proceedings of the 21st international conference on Software engineering, May 1999.

[Zhang 1999] Lujun Zhang, Baowen Xu and Xiaoyu Zhou, Analysis and slicing of Ada95, In 5th International Conference for Young Computer Scientist (ICYCS'99), pp. 189-193, 1999.

[Zhang 1998] Xiang-Xiang Sean Zhang, Practical pointer aliasing analyses for C, PhD thesis, Rutgers University, Aug. 1998.

[Zhang 2003a] Xiangyu Zhang, Rajiv Gupta and Youtao Zhang, Precise dynamic slicing algorithms, 25th International Conference on Software Engineering, pp. 319-329, May 2003.

[Zhang 2003b] Xiangyu Zhang, Rajiv Gupta, Hiding program slices for software security, Proceedings of the International Symposium on Code Generation and Optimization, p. 325, Mar. 2003.

[Zhang 2004a] Xiangyu Zhang and Rajiv Gupta, Cost effective dynamic program slicing, ACM SIGPLAN Conference on Programming Language Design and Implementation, Jun. 2004.

[Zhang 2004b] Xiangyu Zhang and Rajiv Gupta, Recovering dynamic dependence graphs, Technical Report TR04-01, University of Arizona, Department of Computer Science, Tucson, AZ, 2004.

[Zhang 2004c] Xiangyu Zhang, Rajiv Gupta and Youtao Zhang. Efficient forward computation of dynamic slices using reduced ordered binary decision diagrams, IEEE/ACM International Conference on Software Engineering, Edinburgh, UK, May 2004.

[Zhang 2004d] Yingzhou Zhang, Baowen Xu, Liang Shi, Bixin Li, Hongji Yang, Modular monadic program slicing. The 28th Annual International Computer Software and Applications Conference (COMPSAC 2004), pp. 66-71.

[Zhang 2004e] Yingzhou Zhang, Baowen Xu, Ju Qian, Modular monadic slicing of concurrent programs, The 8th IASTED International Conference on Software Engineering and Applications (SEA 2004), Nov. 2004.

[Zhao 1996] Jianjun. Zhao, Jingde Cheng and Kazuo Ushijima, Static slicing of concurrent object-oriented programs, Proceedings of the 20th IEEE Annual International Computer Software and Applications Conference, pp. 312-320, 1996.

[Zhao 1998a] Jianjun Zhao, Applying program dependence analysis to Java software, Proceedings of Workshop on Software Engineering and Database Systems, pp. 162-169, 1998.

[Zhao 1998b] Jianjun Zhao, Jingde Cheng, and Kazuo Ushijima, A dependence-based representation for concurrent object-oriented software maintenance, Proceedings of the 2nd Euromicro Conference on Software Maintenance and Reengineering, pp. 60-66, 1998.

[Zhao 1998c] Jianjun Zhao, Applying slicing technique to software architectures, Proceedings of the Fourth IEEE International Conference on Engineering of Complex Computer Systems, pp. 87-98, Aug. 1998.

[Zhao 1998d] Jianjun Zhao, Dynamic slicing of object-oriented programs, Technical Report SE-98-119, Information Processing Society of Japan (IPSJ), pp. 17-23, May 1998.

[Zhao 1999a] Jianjun Zhao, Slicing concurrent Java programs, Proceedings of 7th IEEE International Workshop on Program Comprehension, IEEE CS Press, pp. 126-133, 1999.

[Zhao 1999b] Jianjun Zhao, Multithreaded dependence graphs for concurrent java programs, Proceedings of 1999 International Symposium on Software Engineering for Parallel and Distributed Systems, pp. 13-23, 1999.

[Zhao 2001] Jianjun Zhao, Jingde Cheng and Kazuo Ushijima, Computing executable slices for concurrent logic programs, Proceedings of the Second Asia-Pacific Conference on Quality Software, pp. 13-22, 2001.

[Zhao 2002a] Jianjun Zhao, Slicing aspect-oriented software, Proceedings of the 10th IEEE International Workshop on Programming Comprehension, pp. 251-260, Jun. 2002.

[Zhao 2002b] Jianjun Zhao, Hongji Yang, Liming Xiang and Baowen Xu, Change impact analysis to support architectural evolution, Journal of Software Maintenance and Evolution: Research and Practice, 14(5), pp. 317-333, 2002.

[Zhao 2003a] Jianjun Zhao and Martin Rinard, System dependence graph construction for aspect-oriented programs, MIT-LCS-TR-891, Laboratory for Computer Science, MIT, Mar. 2003.

[Zhao 2003b] Jianjun Zhao, Cunwei Lu and Baowen Xu, A toolkit for Java bytecode analysis, Proceedings of the Seventh IASTED International Conference on Software Engineering and Applications, Nov. 2003, Marina del Rey, CA, USA, v 7, 2003, pp. 482-487.

[Zhao 2004a] Jianjun Zhao and Baowen Xu, Measuring aspect cohesion，Proceedings of The 2004 European Joint Conferences on Theory and Practice of Software (ETAPS2004), March 27 - April 4, 2004, Barcelona, Spain，Lecture Notes in Computer Science 2984 Springer 2004, pp. 54-68.

[Zhao 2004b] Jianjun Zhao and Bixin Li, Dependence-based representation for concurrent Java programs and its application to slicing, to appear in Proceedings of ISFST2004, Xian, China.

[Zhou 2002] Yuming Zhou, Baowen Xu, Jianjun Zhao and Hongji Yang, ICBMC: An improved cohesion measure for classes, Proceedings of IEEE International Conference on Software Maintenance (ICSM'02), Montreal, Canada, pp. 44-53, Oct. 2002.

[Zhou 2003] Yuming Zhou, Lijie Wen, Jianmin Wang, Yujian Chen, Hongmin Lu, and Baowen Xu, DRC: A dependence relationships based cohesion measure for classes, 10th Asia-Pacific Software Engineering Conference (APSEC 2003), Chiangmai, Thailand, Dec. 2003.